Splunk® Enterprise
Managing Indexers and Clusters of Indexers 9.2.1

Generated: 6/24/2024 9:00 am

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Indexing overview

Indexes, indexers, and indexer clusters

This manual discusses Splunk Enterprise data repositories and the Splunk Enterprise components that create and manage them.

The **index** is the repository for Splunk Enterprise data. Splunk Enterprise transforms incoming data into **events**, which it stores in indexes.

An **indexer** is a Splunk Enterprise instance that indexes data. For small deployments, a single instance might perform other Splunk Enterprise functions as well, such as data input and search management. In a larger, distributed **deployment**, however, the functions of data input and search management are allocated to other Splunk Enterprise components. This manual focuses exclusively on the indexing function, in the context of either a single-instance or a distributed deployment.

An **indexer cluster** is a group of indexers configured to replicate each others' data, so that the system keeps multiple copies of all data. This process is known as **index replication**, or indexer clustering. By maintaining multiple, identical copies of data, clusters prevent data loss while promoting data availability for searching.

Indexes

As Splunk Enterprise processes incoming data, it adds the data to indexes. Splunk Enterprise ships with several indexes, and you can create additional indexes as needed.

A Splunk Enterprise index contains a variety of files. These files fall into two main categories:

- The raw data in compressed form (**rawdata**)
- Indexes that point to the raw data (**index files**, also referred to as **tsidx files**), plus some metadata files

The files reside in directories organized by age. These directories are called **buckets**. See How Splunk Enterprise stores indexes.

Splunk Enterprise manages its indexes to facilitate flexible searching and fast data retrieval, eventually archiving them according to a user-configurable schedule. Splunk Enterprise handles everything with flat files; it doesn't require any third-party database software running in the background.

To start indexing, you simply specify the data inputs that you want Splunk Enterprise to index. You can add more inputs at any time, and Splunk Enterprise will begin indexing them as well. See What Splunk Enterprise can index in *Getting Data In* to learn how to add data inputs.

Splunk Enterprise, by default, puts all user data into a single, preconfigured index. It also employs several other indexes for internal purposes. You can add new indexes and manage existing ones to meet your data requirements. See Manage indexes.

Event processing

During indexing, Splunk Enterprise performs **event processing**. It processes incoming data to enable fast search and analysis, storing the results in the index as events. While indexing, Splunk Enterprise enhances the data in various ways,

including by:

- Separating the datastream into individual, searchable events.
- Creating or identifying timestamps.
- Extracting fields such as host, source, and sourcetype.
- Performing user-defined actions on the incoming data, such as identifying custom fields, masking sensitive data, writing new or modified keys, applying breaking rules for multi-line events, filtering unwanted events, and routing events to specified indexes or servers.

Getting Data In describes how to configure event processing to meet the needs of your data. See Overview of event processing.

Index types

Splunk Enterprise supports two types of indexes:

- Events indexes. Events indexes impose minimal structure and can accommodate any type of data, including metrics data. Events indexes are the default index type.
- Metrics indexes. Metrics indexes use a highly structured format to handle the higher volume and lower latency demands associated with metrics data. Putting metrics data into metrics indexes results in faster performance and less use of index storage, compared to putting the same data into events indexes. For information on the metrics format, see the *Metrics* manual.

There are minimal differences in how indexers process and manage the two index types. Despite its name, event processing occurs in the same sequence for both events and metrics indexes. Metrics data is really just a highly structured kind of event data.

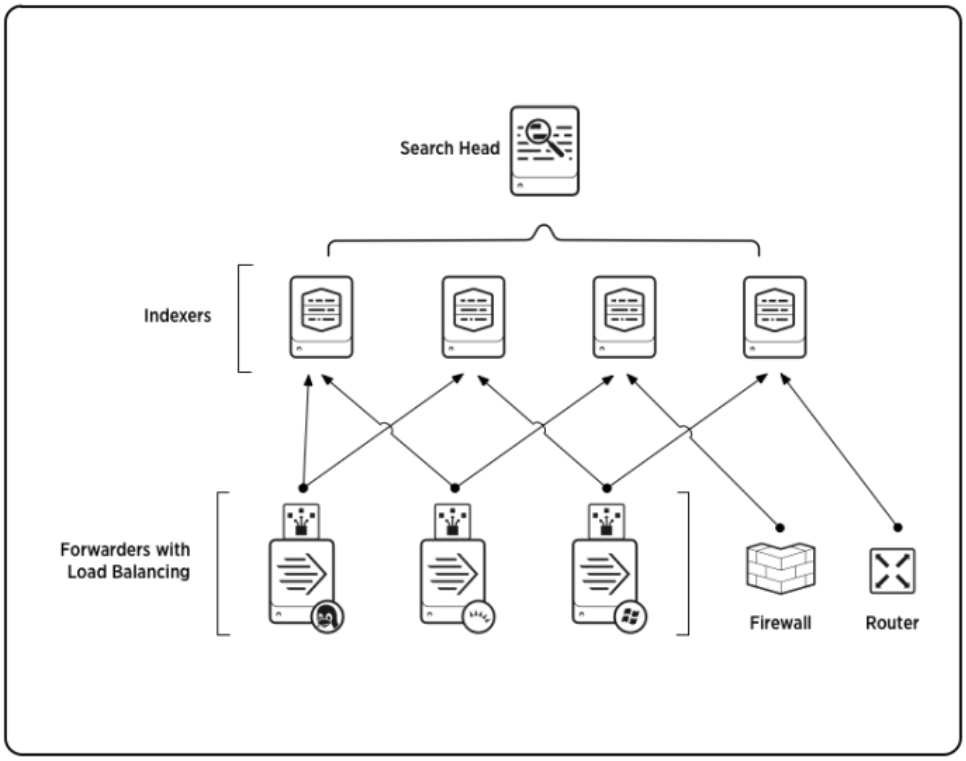
Indexers

The indexer is the Splunk Enterprise component that creates and manages indexes. The primary functions of an indexer are:

- Indexing incoming data.
- Searching the indexed data.

In single-machine deployments consisting of just one Splunk Enterprise instance, the indexer also handles the **data input** and **search management** functions. This type of small deployment might handle the needs of a single department in an organization.

For larger-scale needs, indexing is split out from the data input function and sometimes from the search management function as well. In these larger, distributed deployments, the Splunk Enterprise indexer might reside on its own machine and handle only indexing, along with searching of its indexed data. In those cases, other Splunk Enterprise components take over the non-indexing roles. **Forwarders** consume the data, indexers index and search the data, and **search heads** coordinate searches across the set of indexers. Here's an example of a scaled-out deployment:



For more information on using indexers in a distributed deployment, see Indexers in a distributed deployment.

A Splunk indexer is simply a Splunk Enterprise instance. To learn how to install a Splunk Enterprise instance, read the *Installation Manual.*

Indexer clusters

An indexer cluster is a group of Splunk Enterprise nodes that, working in concert, provide a redundant indexing and searching capability. There are three types of nodes in a cluster:

- A single **manager node** to manage the cluster. The manager node is a specialized type of indexer.
- Several **peer nodes** that handle the indexing function for the cluster, indexing and maintaining multiple copies of the data and running searches across the data.
- One or more search heads to coordinate searches across all the peer nodes.

Indexer clusters feature automatic failover from one peer node to the next. This means that, if one or more peers fail, incoming data continues to get indexed and indexed data continues to be searchable.

The first part of this manual contains configuration and management information relevant for all indexers, independent of whether they are part of a cluster. The second part of this manual, starting with the topic About indexer clusters and index replication, is relevant only for clusters.

How indexing works

Splunk Enterprise can **index** any type of time-series data (data with **timestamps**). When Splunk Enterprise indexes data, it breaks it into **events**, based on the timestamps.

The indexing process follows the same sequence of steps for both events indexes and metrics indexes.

Event processing and the data pipeline

Data enters the indexer and proceeds through a pipeline where **event processing** occurs. Finally, the processed data it is written to disk. This pipeline consists of several shorter pipelines that are strung together. A single instance of this end-to-end data pipeline is called a **pipeline set**.

Event processing occurs in two main stages, parsing and indexing. All data that comes into Splunk Enterprise enters through the **parsing pipeline** as large (10,000 bytes) chunks. During parsing, Splunk Enterprise breaks these chunks into events which it hands off to the **indexing pipeline**, where final processing occurs.

While parsing, Splunk Enterprise performs a number of actions, including:

- Extracting a set of default fields for each event, including `host`, `source`, and `sourcetype`.
- Configuring character set encoding.
- Identifying line termination using linebreaking rules. While many events are short and only take up a line or two, others can be long.
- Identifying timestamps or creating them if they don't exist. At the same time that it processes timestamps, Splunk identifies event boundaries.
- Splunk can be set up to mask sensitive event data (such as credit card or social security numbers) at this stage. It can also be configured to apply custom metadata to incoming events.

In the indexing pipeline, Splunk Enterprise performs additional processing, including:

- Breaking all events into segments that can then be searched upon. You can determine the level of segmentation, which affects indexing and searching speed, search capability, and efficiency of disk compression.
- Building the index data structures.
- Writing the raw data and index files to disk, where post-indexing compression occurs.

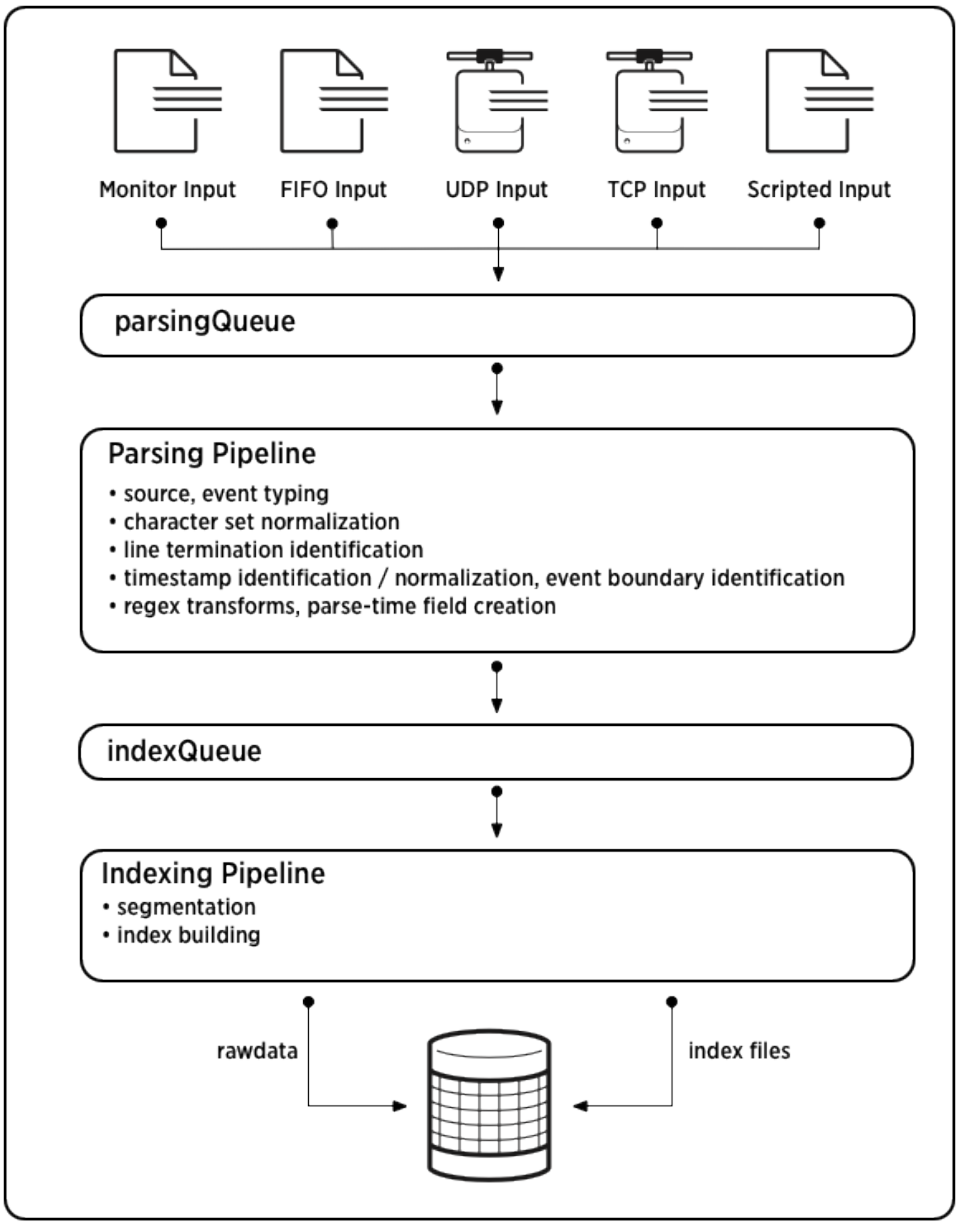
Note: The term "indexing" is also used in a more general sense to refer to the entirety of event processing, encompassing both the parsing pipeline and the indexing pipeline. The differentation between the parsing and indexing pipelines is of relevance mainly when deploying **heavy forwarders**.

Heavy forwarders can run raw data through the parsing pipeline and then forward the parsed data on to indexers for final indexing. **Universal forwarders** do not parse data in this way. Instead, universal forwarders forward the raw data to the indexer, which then processes it through both pipelines.

Note that both types of forwarders do perform a type of parsing on certain structured data. See Extract data from files with headers in *Getting Data In*.

For more information about events and how the indexer transforms data into events, see the chapter Configure event processing in *Getting Data In*.

This diagram shows the main processes inherent in indexing:


Monitor Input
FIFO Input
UDP Input
TCP Input
Scripted Input
parsingQueue
Parsing Pipeline
• source, event typing
• character set normalization
• line termination identification
• timestamp identification / normalization, event boundary identification
• regex transforms, parse-time field creation
indexQueue
Indexing Pipeline
• segmentation
• index building
rawdata
index files

Note: This diagram represents a simplified view of the indexing architecture. It provides a functional view of the architecture and does not fully describe Splunk Enterprise internals. In particular, the parsing pipeline actually consists of three pipelines: **parsing**, **merging**, and **typing**, which together handle the parsing function. The distinction can matter during troubleshooting, but does not generally affect how you configure or deploy Splunk Enterprise.

For a more detailed discussion of the data pipeline and how it affects deployment decisions, see How data moves through Splunk Enterprise: the data pipeline in *Distributed Deployment*.

What's in an index?

Splunk Enterprise stores the data it processes in indexes. An index consists of a collection of subdirectories, called **buckets**. Buckets consist mainly of two types of files: **rawdata files** and **index files**. See How Splunk Enterprise stores indexes.

Immutability of indexed data

Once data has been added to an index, you cannot edit or otherwise change the data. You can delete all data from an index or you can delete, and optionally archive, individual index buckets based on policy, but you cannot selectively delete individual events from storage.

See Remove indexes and indexed data.

Default set of indexes

Splunk Enterprise comes with a number of preconfigured indexes, including:

- **main**: This is the default Splunk Enterprise index. All processed data is stored here unless otherwise specified.
- **_internal**: Stores Splunk Enterprise internal logs and processing metrics.
- **_audit**: Contains events related to the file system change monitor, auditing, and all user search history.

A Splunk Enterprise administrator can create new indexes, edit index properties, remove unwanted indexes, and relocate existing indexes. Splunk Enterprise administrators manage indexes through Splunk Web, the CLI, and configuration files such as `indexes.conf`. See Managing indexes.

Index time versus search time

Splunk Enterprise documentation contains references to the terms **"index time"** and **"search time"**. These terms distinguish between the types of processing that occur during indexing, and the types that occur when a search is run.

It is important to consider this distinction when administering Splunk Enterprise. For example, say that you want to use custom **source types** and **hosts**. You should define those custom source types and hosts before you start indexing, so that the indexing process can tag events with them. After indexing, you cannot change the host or source type assignments.

If you neglect to create the custom source types and hosts until after you have begun to index data, your choice is either to re-index the data, in order to apply the custom source types and hosts to the existing data, as well as to new data, or, alternatively, to manage the issue at search time by tagging the events with alternate values.

Conversely, as a general rule, it is better to perform most knowledge-building activities, such as **field extraction**, at search time. Index-time custom field extraction can degrade performance at both index time and search time. When you

add to the number of fields extracted during indexing, the indexing process slows. Later, searches on the index are also slower, because the index has been enlarged by the additional fields, and a search on a larger index takes longer.

You can avoid such performance issues by instead relying on search-time field extraction. For details on search-time field extraction, see About fields and When Splunk Enterprise extracts fields in the *Knowledge Manager Manual*.

At index time

Index-time processes take place between the point when the data is consumed and the point when it is written to disk.

The following processes occur during index time:

- Default field extraction (such as `host`, `source`, `sourcetype`, and `timestamp`)
- Static or dynamic host assignment for specific inputs
- Default host assignment overrides
- Source type customization
- Custom index-time field extraction
- Structured data field extraction
- Event timestamping
- Event linebreaking
- Event segmentation (also happens at search time)

At search time

Search-time processes take place while a search is run, as events are collected by the search. The following processes occur at search time:

- Event segmentation (also happens at index time)
- Event type matching
- Search-time field extraction (automatic and custom field extractions, including multivalue fields and calculated fields)
- Field aliasing
- Addition of fields from lookups
- Source type renaming
- Tagging

The data pipeline

The **data pipeline** provides a more detailed way to think about the progression of data through the system. The data pipeline is particularly useful for understanding how to assign configurations and work across a distributed deployment. See How data moves through Splunk: the data pipeline in *Distributed Deployment*.

Install an indexer

All full Splunk Enterprise instances serve as indexers by default. To learn how to install a Splunk Enterprise instance, read the Installation Manual. Then return to this manual to learn how to configure the indexer.

If you plan to deploy an indexer in a distributed deployment, next read the topic, Indexers in a distributed environment.

Indexers in a distributed deployment

Important: To better understand this topic, you should be familiar with Splunk Enterprise distributed environments, covered in *Distributed Deployment.*

The indexer is the Splunk Enterprise component that creates and manages indexes. The primary functions of an indexer are:

- Indexing incoming data.
- Searching the indexed data.

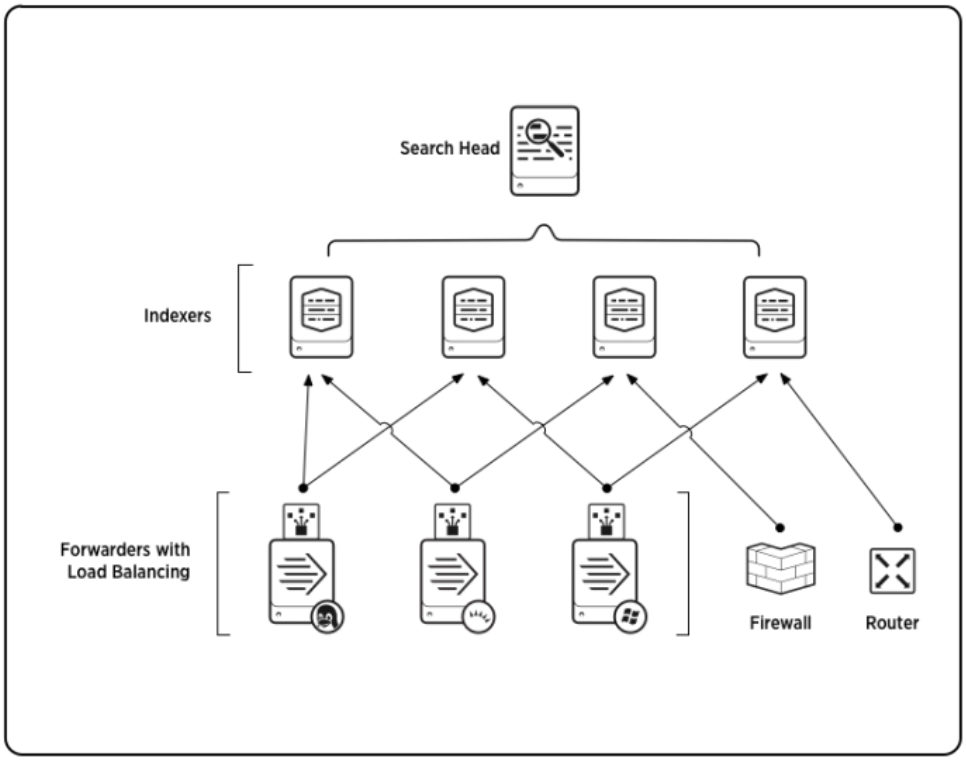
In single-machine deployments consisting of just one Splunk Enterprise instance, the indexer also handles the **data input** and **search management** functions.

For larger-scale needs, indexing is split out from the data input function and sometimes from the search management function as well. In these larger, distributed deployments, the indexer might reside on its own machine and handle only indexing, along with searching of its indexed data. In those cases, other Splunk Enterprise components take over the non-indexing roles.

For instance, you might have a set of Windows and Linux machines generating events, which need to go to a central indexer for consolidation. Usually the best way to do this is to install a lightweight instance of Splunk Enterprise, known as a **forwarder**, on each of the event-generating machines. These forwarders handle data input and send the data across the network to the indexer residing on its own machine.

Similarly, in cases where you have a large amount of indexed data and numerous concurrent users searching on it, it can make sense to split off the search management function from indexing. In this type of scenario, known as **distributed search**, one or more **search heads** distribute search requests across multiple indexers. The indexers still perform the actual searching of their own indexes, but the search heads manage the overall search process across all the indexers and present the consolidated search results to the user.

Here is an example of a scaled-out deployment:



While the fundamental issues of indexing and event processing remain the same for distributed deployments, it is important to take into account deployment needs when planning your indexing strategy.

Forward data to an indexer

To forward remote data to an indexer, you use forwarders, which are Splunk Enterprise instances that receive data inputs and then consolidate and send the data to a Splunk Enterprise indexer. Forwarders come in two flavors:

- **Universal forwarders**. These maintain a small footprint on their host machine. They perform minimal processing on the incoming data streams before forwarding them on to an indexer, also known as the **receiver**.
- **Heavy forwarders**. These retain most of the functionality of a full Splunk Enterprise instance. They can parse data before forwarding it to the receiving indexer. (See How indexing works for the distinction between parsing and indexing.) They can store indexed data locally and also forward the parsed data to a receiver for final indexing on that machine as well.

Both types of forwarders tag data with metadata such as host, source, and source type, before forwarding it on to the indexer.

Forwarders allow you to use resources efficiently when processing large quantities or disparate types of data coming from remote sources. They also enable a number of interesting deployment topologies, by offering capabilities for **load balancing**, data **filtering**, and **routing**.

For an extended discussion of forwarders, including configuration and detailed use cases, read *Forwarding Data.*

Search across multiple indexers

In distributed search, search heads send search requests to indexers and then merge the results back to the user. This is useful for a number of purposes, including horizontal scaling, access control, and managing geo-dispersed data.

For an extended discussion of distributed search and search heads, including configuration and detailed use cases, see *Distributed Search.*

Indexer clusters also use search heads to coordinate searches across the cluster's peer nodes. See About indexer clusters and index replication.

Deploy indexers in a distributed environment

To implement a distributed environment similar to the diagram earlier in this topic, you need to install and configure three types of components:

- Indexers
- Forwarders (typically, universal forwarders)
- Search head(s)

Install and configure the indexers

By default, all full Splunk Enterprise instances serve as indexers. For horizontal scaling, you can install multiple indexers on separate machines.

To learn how to install a Splunk Enterprise instance, read the *Installation Manual.*

Then return to this manual for information on configuring each individual indexer to meet the needs of your specific deployment. Start with the chapter Manage indexers and continue with the chapters that follow.

Install and configure the forwarders

A typical distributed deployment has a large number of forwarders feeding data to a few indexers. For most forwarding purposes, the universal forwarder is the best choice. The universal forwarder is a separate downloadable from the full Splunk Enterprise instance.

To learn how to install and configure forwarders, read *Forwarding Data.*

Install and configure the search head(s)

You can install one or more search heads to handle your distributed search needs. Search heads are just full Splunk Enterprise instances that have been specially configured.

To learn how to configure a search head, read *Distributed Search.*

Other deployment tasks

You need to configure Splunk Enterprise licensing by designating a **license manager**. See the chapter Configure Splunk Enterprise licenses in the *Admin Manual* for more information.

You can use the Splunk Enterprise **deployment server** to simplify the job of updating the deployment components. For details on how to configure a deployment server, see *Updating Splunk Enterprise Instances*.

Install a cluster of indexers

If data availability, data fidelity, and data recovery are key issues for your deployment, then you should consider deploying an indexer cluster, rather than a series of individual indexers. For further information, see About indexer clusters and index replication.

Manage indexes

About managing indexes

When you add data, the indexer processes it and stores it in an **index**. By default, data you feed to an indexer is stored in the **main** index, but you can create and specify other indexes for different data inputs.

An index is a collection of directories and files. These are located under `$SPLUNK_HOME/var/lib/splunk`. Index directories are also called **buckets** and are organized by age. For information on index storage, see How Splunk Enterprise stores indexes.

In addition to the **main** index, Splunk Enterprise comes preconfigured with a number of internal indexes. Internal indexes begin with an underscore (_); for example, **_audit** and **_internal**.

To see a full list of internal indexes, go to Splunk Web, select the **Settings** link in the upper portion of the screen, and then select **Indexes**.

A number of topics in this manual describe ways you can manage your indexes. In particular, the following topics are helpful in index management:

- Create custom indexes
- Remove indexes and data from Splunk
- Configure index storage
- Move the index database
- Use multiple partitions for index data
- Configure index size
- Set limits on disk usage
- Back up indexed data
- Set a retirement and archiving policy

Index types

Splunk Enterprise supports two types of indexes:

- Events indexes. Events indexes impose minimal structure and can accommodate any type of data, including metrics data. Events indexes are the default index type.
- Metrics indexes. Metrics indexes use a highly structured format to handle the higher volume and lower latency demands associated with metrics data. Putting metrics data into metrics indexes results in faster performance and less use of index storage, compared to putting the same data into events indexes. For information on the metrics format, see the *Metrics* manual.

There are minimal differences in how indexers handle the two index types. Any differences that do exist are described in the relevant topic.

For more information about the indexing process

To learn more about indexes, see:

- The topic How indexing works in this manual.

- The topic How Splunk stores indexes in this manual.
- The chapter About clusters and index replication in this manual.
- The chapter Configure event processing in *Getting Data In.*
- The chapter Set up and use summary indexes in the *Knowledge Manager Manual,* for information about working with extremely large datasets.

Create custom indexes

You can create two types of indexes:

- Events indexes
- Metrics indexes

Events indexes are the default index type. To create events indexes, see Create events indexes.

To create metrics indexes, see Create metrics indexes. For general information on metrics indexes, see the *Metrics* manual, starting with Overview of metrics.

For information on creating SmartStore indexes, see Add a SmartStore index.

Create events indexes

The `main` index, by default, holds all your events. It also serves as the default index for any inputs or search commands that don't specify an index, although you can change the default.

With a Splunk Enterprise license, you can add an unlimited number of additional indexes. You can add indexes using Splunk Web, the CLI, or indexes.conf.

This topic covers:

- The reasons why you might want multiple indexes.
- How to create new indexes.
- How to send events to specific indexes.
- How to search specific indexes.

Why have multiple indexes?

There are several key reasons for having multiple indexes:

- To control user access.
- To accommodate varying retention policies.
- To speed searches in certain situations.

The main reason you'd set up multiple indexes is to control user access to the data that's in them. When you assign users to **roles**, you can limit user searches to specific indexes based on the role they're in.

In addition, if you have different policies for retention for different sets of data, you might want to send the data to different indexes and then set a different archive or retention policy for each index.

Another reason to set up multiple indexes has to do with the way search works. If you have both a high-volume/high-noise data source and a low-volume data source feeding into the same index, and you search mostly for events from the low-volume data source, the search speed will be slower than necessary, because the indexer also has to search through all the data from the high-volume source. To mitigate this, you can create dedicated indexes for each data source and send data from each source to its dedicated index. Then, you can specify which index to search on. You'll probably notice an increase in search speed.

Create events indexes

You can create events indexes with Splunk Web, the CLI, or by editing `indexes.conf` directly.

Note: To add a new index to an indexer cluster, you must directly edit `indexes.conf`. You cannot add an index via Splunk Web or the CLI. For information on how to configure `indexes.conf` for clusters, see Configure the peer indexes in an indexer cluster. That topic includes an example of creating a new cluster index.

Use Splunk Web

> You cannot use Splunk Web to add a SmartStore index. You also cannot use Splunk Web to add a non-SmartStore index, if the indexer has any SmartStore indexes.

1. In Splunk Web, navigate to **Settings > Indexes** and click **New**.
2. To create a new index, enter:
   - A name for the index. User-defined index names must consist of only numbers, lowercase letters, underscores, and hyphens. They cannot begin with an underscore or hyphen, or contain the word "kvstore".
   - The index data type. For event data, click **Events.** This is the default data type.
   - The path locations for index data storage:
     - Home path. Leave blank for default `$SPLUNK_DB/<index_name>/db`
     - Cold path. Leave blank for default `$SPLUNK_DB/<index_name>/colddb`
     - Thawed path. Leave blank for default `$SPLUNK_DB/<index_name>/thaweddb`
   - Enable/disable data integrity check.
   - The maximum size of the entire index. Defaults to 500000MB.
   - The maximum size of each index bucket. When setting the maximum size, use `auto_high_volume` for high volume indexes (such as the main index); otherwise, use `auto`.
   - The frozen archive path. Set this field if you want to archive frozen buckets. For information on bucket archiving, see Archive indexed data.
   - The app in which the index resides.
   - The tsidx retention policy. See Reduce tsidx usage.

   For more information on index settings, see Configure index storage.
3. Click **Save**.

You can edit an index by clicking on the index name in the **Indexes** section of the **Settings** menu in Splunk Web. Properties that you cannot change in Splunk Web are grayed out. To change those properties, edit `indexes.conf`, then restart the indexer.

Note: Some index properties are configurable only by editing the `indexes.conf` file. Check the `indexes.conf` topic for a complete list of properties.

Use the CLI

Navigate to the `$SPLUNK_HOME/bin/` directory and use the `add index` command. You do not need to stop the indexer first.

To add a new index called "fflanda", enter the following command:

```
splunk add index fflanda
```

Note: User-defined index names must consist of only numbers, lowercase letters, underscores, and hyphens. They cannot begin with an underscore or hyphen, or contain the word "kvstore".

If you do not want to use the default path for your new index, you can use parameters to specify a new location:

```
splunk add index foo -homePath /your/path/foo/db -coldPath /your/path/foo/colddb
    -thawedPath /your/path/foo/thawedDb
```

You can also edit an index's properties from the CLI. For example, to edit an index called "fflanda" using the CLI, type:

```
splunk edit index fflanda -<parameter> <value>
```

For detailed information on index settings, see Configure index storage.

Edit indexes.conf

To add a new index, add a stanza to `indexes.conf` in `$SPLUNK_HOME/etc/system/local`, identified by the name of the new index. For example:

```
[newindex]
homePath=<path for hot and warm buckets>
coldPath=<path for cold buckets>
thawedPath=<path for thawed buckets>
...
```

For information on index settings, see Configure index storage and the indexes.conf spec file.

Note: User-defined index names must consist of only numbers, lowercase letters, underscores, and hyphens. They cannot begin with an underscore or hyphen, or contain the word "kvstore".

You must restart the indexer after editing `indexes.conf`.

For information on adding or editing index configurations on cluster nodes, see Configure the peer indexes in an indexer cluster.

Send events to specific indexes

By default, all external events go to the index called **main**. However, you might want to send some events to other indexes. For example, you might want to route all data from a particular input to its own index. Or you might want to segment data or send event data from a noisy source to an index that is dedicated to receiving it.

Important: To send events to a specific index, the index must already exist on the indexer. If you route any events to an index that doesn't exist, the indexer will drop those events.

Send all events from a data input to a specific index

To send all events from a particular data input to a specific index, add the following line to the input's stanza in inputs.conf on the Splunk Enterprise component where the data is entering the system: either the indexer itself or a forwarder sending data to the indexer:

```
index = <index_name>
```

The following example `inputs.conf` stanza sends all data from `/var/log` to an index named `fflanda`:

```
[monitor:///var/log]
disabled = false
index = fflanda
```

Route specific events to a different index

Just as you can route events to specific queues, you can also route specific events to specific indexes. You configure this on the indexer itself, *not* on the forwarder sending data to the indexer, if any.

To route certain events to a specific index, edit props.conf and transforms.conf on the indexer:

1. Identify a common attribute for the events that can be used to differentiate them.

2. In `props.conf`, create a stanza for the source, source type, or host. This stanza specifies a `transforms_name` that corresponds to a regex-containing stanza you will create in `transforms.conf`.

3. In `transforms.conf`, create an stanza named with the `transforms_name` you specified in step 2. This stanza:

- Specifies a regular expression that matches the identified attribute from step 1.
- Specifies the alternate index that events matching the attribute should be routed to.

The sections below fill out the details for steps 2 and 3.

Edit props.conf

Add the following stanza to `$SPLUNK_HOME/etc/system/local/props.conf`:

```
[<spec>]
TRANSFORMS-<class_name> = <transforms_name>
```

Note the following:

- `<spec>` is one of the following:
  - `<sourcetype>`, the sourcetype of an event
  - `host::<host>`, where `<host>` is the host for an event
  - `source::<source>`, where `<source>` is the source for an event
- `<class_name>` is any unique identifier.
- `<transforms_name>` is whatever unique identifier you want to give to your transform in `transforms.conf`.

Edit transforms.conf

Add the following stanza to `$SPLUNK_HOME/etc/system/local/transforms.conf`:

```
[<transforms_name>]
REGEX = <your_custom_regex>
DEST_KEY = _MetaData:Index
FORMAT = <alternate_index_name>
```

Note the following:

- `<transforms_name>` must match the `<transforms_name>` identifier you specified in `props.conf`.
- `<your_custom_regex>` must provide a match for the attribute you identified earlier, in step 1.
- `DEST_KEY` must be set to the index attribute `_MetaData:Index`.
- `<alternate_index_name>` specifies the alternate index that the events will route to.

Example

This examples routes events of `windows_snare_log` source type to the appropriate index based on their log types. "Application" logs will go to an alternate index, while all other log types, such as "Security", will go to the default index.

To make this determination, it uses `props.conf` to direct events of `windows_snare_log` source type through the `transforms.conf` stanza named "AppRedirect", where a regex then looks for the log type, "Application". Any event with a match on "Application" in the appropriate location is routed to the alternate index, "applogindex". All other events go to the default index.

1. Identify an attribute

The events in this example look like this:

```
web1.example.com        MSWinEventLog        1        Application        721         Wed Sep 06 17:05:31
2006
4156        MSDTC        Unknown User        N
/A        Information        WEB1        Printers                String
message: Session idle timeout over, tearing down the session.        179

web1.example.com        MSWinEventLog        1        Security        722  ¡     Wed Sep 06 17:59:08 2006
576        Security        SYSTEM        User        Success Audit        WEB1        Privilege Use
Special privileges assigned to new logon:     User Name:      Domain:      Logon
ID: (0x0,0x4F3C5880)     Assigned: SeBackupPrivilege   SeRestorePrivilege
SeDebugPrivilege   SeChangeNotifyPrivilege   SeAssignPrimaryTokenPrivilege 525
```

Some events contain the value "Application", while others contain the value "Security" in the same location.

2. Edit props.conf

Add this stanza to `$SPLUNK_HOME/etc/system/local/props.conf`:

```
[windows_snare_syslog]
TRANSFORMS-index = AppRedirect
```

This directs events of `windows_snare_syslog` sourcetype to the `AppRedirect` stanza in `transforms.conf`.

3. Edit transforms.conf

Add this stanza to `$SPLUNK_HOME/etc/system/local/transforms.conf`:

```
[AppRedirect]
REGEX = MSWinEventLog\s+\d+\s+Application
DEST_KEY = _MetaData:Index
```

```
FORMAT = applogindex
```
This stanza processes the events directed here by `props.conf`. Events that match the regex (because they contain the string "Application" in the specified location) get routed to the alternate index, "applogindex". All other events route as usual to the default index.

Search a specific index

When the indexer searches, it targets the default index (by default, **main**), unless the search explicitly specifies an index. For example, this search command searches in the `hatch` index:

```
index=hatch userid=henry.gale
```

You can also specify an alternate default index for a given role to search when you create or edit that role.

Create metrics indexes

You can create metrics indexes with Splunk Web, the CLI, the REST API, or by editing `indexes.conf` directly. For more about metrics, see Overview of metrics in the *Metrics* manual.

Use Splunk Web

1. In Splunk Web, navigate to **Settings > Indexes** and click **New**.
2. For **Index Name**, type a name for the index. User-defined index names must consist of only numbers, lowercase letters, underscores, and hyphens. Index names cannot begin with an underscore or hyphen, or contain the word "kvstore".
3. For **Index Data Type**, click **Metrics**.
4. (Optional) Set **Timestamp Resolution** to **Milliseconds** if you want the metrics index to store metric data points at that increased level of granularity. Metrics indexes with millisecond timestamp resolution have decreased search performance. See Metrics indexes with millisecond timestamps.
5. Enter the remaining properties of the index as needed. For details, see Create events indexes.
6. Click **Save**.

Use the command line interface (CLI)

1. Open a command prompt.
2. Navigate to the **$SPLUNK_HOME/bin/** directory.
3. Use the `add index` command to create an index. User-defined index names must consist of only numbers, lowercase letters, underscores, and hyphens. Index names cannot begin with an underscore or hyphen, or contain the word "kvstore".

For example, to create an index called `mymetricsindex`, enter the following command:

```
splunk add index mymetricsindex -datatype metric
```
To list all metrics indexes, enter the following command:

```
splunk list index -datatype metric
```
To list all indexes, including events indexes, enter the following command:

```
splunk list index -datatype all
```

Use the REST API

Create an index using the /**data**/**indexes** endpoint with the "`datatype=metric`" parameter. For details, see /**data**/**indexes** in the *REST API Reference Manual*.

For example, to create a metrics index called `mymetricsindex`, enter the following command:

```
curl -k -u admin:pass https://localhost:8089/services/data/indexes  \
    -d name=mymetricsindex                                          \
    -d datatype=metric
```

To list all metrics indexes using the REST API, enter the following command:

```
curl -k -u admin:pass https://localhost:8089/services/data/indexes?datatype=metric
```

To list all indexes, including events indexes, enter the following command:

```
curl -k -u admin:pass https://localhost:8089/services/data/indexes?datatype=all
```

Edit indexes.conf

To create a new metrics index, add a stanza to `indexes.conf` in `$SPLUNK_HOME/etc/system/local`, identified by the name of the new index. Change the `datatype` parameter to `datatype = metric`.

For example, to create a metrics index called "mymetricsindex", add the following stanza:

```
[mymetricsindex]
homePath=<path for hot and warm buckets>
coldPath=<path for cold buckets>
thawedPath=<path for thawed buckets>
datatype = metric
metric.timestampResolution = <s | ms>
...
```

For information on index settings, see Configure index storage and the indexes.conf spec file.

> User-defined index names must consist of only numbers, lowercase letters, underscores, and hyphens. They cannot begin with an underscore or hyphen, or contain the word "kvstore".

You must restart the indexer after editing `indexes.conf`.

For information on adding or editing index configurations on cluster nodes, see Configure the peer indexes in an indexer cluster.

Metrics indexes with millisecond timestamps

By default, metrics indexes are only searchable at a second-by-second precision. This is unlike events indexes, which can be searched with subsecond precision by default.

If you are dealing with a high volume source of metric data, such as a utility grid that has the potential to generate millions of metric data points per second, this means that the metric index is populated with sample metric data points or metric data points that are aggregated views of the raw metric data, taken at regular intervals.

If you are concerned about high index volume, this can be a good thing. Having second precision metrics indexes keeps your indexes lean and saves you from having to search through huge numbers of events over relatively short time ranges. But this also means that you cannot run time-based metrics searches that have subsecond precision. Similarly, you cannot set up `mstats` searches that group by subsecond `span` values.

If you need the capability to perform metric searches with subsecond precision, give your new metric index a **Timestamp Resolution** of **Milliseconds**. Metrics indexes with millisecond timestamp resolution can have decreased search performance in comparison to metrics indexes that have the default second timestamp precision.

> Metrics indexes set to millisecond precision might incur more license usage than similar metrics indexes set to second precision. The license cost per metric data point remains the same, but millisecond-precision indexes can index more data points than second-precision indexes ingesting data from the same source.

About changing timestamp resolutions of metrics indexes

You can change the timestamp resolution of a metrics index after you create it. However, if you change the timestamp resolution of a metrics index from millisecond to second, it may look like data loss to people who regularly run searches against that metrics index. This is because the index won't ingest data at millisecond resolution after the change.

When your index is at millisecond timestamp resolution, your indexed metric data points might have timestamps like this.

_timestamp (seconds)
1.000
1.001
1.002
2.000
2.435
3.123
3.651
4.000

After four seconds, if you change the timestamp resolution from millisecond timestamp resolution to second timestamp resolution, your index is restricted to indexing one metric data point per second:

_timestamp (seconds)
5.000
6.000
7.000
8.000
9.000

Some users may perceive this as a data loss when in fact they are just seeing their data with a less granular timestamp resolution.

Similarly, users of a metrics index that is switched from a second timestamp resolution to a millisecond timestamp resolution may be surprised to see their indexes ingesting more events than they did before the switch.

As an administrator of a Splunk Enterprise deployment it is up to you to communicate this change and its implications to your users.

Remove indexes and indexed data

You can remove indexed data or even entire indexes from the indexer. These are the main options:

- Delete events from subsequent searches.
- Remove all data from one or more indexes.
- Remove or disable an entire index.
- Remove older data, based on a retirement policy.

Removing data is irreversible. If you want to get your data back once you've removed data using any of the techniques described in this topic, you must re-index the applicable data sources.

Delete events from subsequent searches

The Splunk search language provides the `delete` command to delete event data from subsequent searches.

The `delete` command is available only with events indexes. You cannot use it with metrics indexes

You cannot run the `delete` command for a real-time search. If you try to use `delete` during a real-time search, Splunk Enterprise will display an error.

The delete command only deletes the events from subsequent searches. The data itself remains in the index.

Who can delete?

The `delete` command can only be run by a user with the "delete_by_keyword" **capability**. By default, Splunk Enterprise ships with a special **role**, "can_delete" that has this capability (and no others). The admin role does not have this capability by default. It's recommended that you create a special user that you log into when you intend to delete index data.

For more information, refer to Add and edit roles in *Securing Splunk Enterprise.*

How to delete

First run a search that returns the events you want deleted. Make sure that this search returns *only* the events you want to delete, and no other events. Once you're certain of that, you can pipe the results of the search to the `delete` command.

For example, if you want to remove the events you've indexed from a source called `/fflanda/incoming/cheese.log` so that they no longer appear in searches, do the following:

1. Disable or remove that source so that it no longer gets indexed.

2. Search for events from that source in your index:

```
source="/fflanda/incoming/cheese.log"
```

3. Look at the results to confirm that this is the data you want to delete.

4. Once you've confirmed that this is the data you want to delete, pipe the search to `delete`:

```
source="/fflanda/incoming/cheese.log" | delete
```

See the page about the delete command in the Search Reference Manual for more examples.

Note: When running Splunk on Windows, substitute the forward slashes (/) in the examples with backslashes (\).

Piping a search to the `delete` command marks all the events returned by that search so that subsequent searches do not return them. No user (even with admin permissions) will be able to see this data when searching.

Note: Piping to `delete` does not reclaim disk space. The data is not actually removed from the index; it is just invisible to searches.

The `delete` command does not update the metadata of the events, so any metadata searches will still include the events although they are not searchable. The main **All indexed data** dashboard will still show event counts for the deleted sources, hosts, or sourcetypes.

The delete operation and indexer clusters

In the normal course of index replication, the effects of a `delete` operation get quickly propagated across all bucket copies in the cluster, typically within a few seconds or minutes, depending on the cluster load and amount of data and buckets affected by the `delete` operation. During this propagation interval, a search can return results that have already been deleted.

Also, if a peer that had primary bucket copies at the time of the `delete` operation goes down before all the results have been propagated, some of the deletes will be lost. In that case, you must rerun the operation after the primary copies from the downed peer have been reassigned.

Remove all data from one or all indexes

To delete indexed data permanently from your disk, use the CLI `clean` command. This command completely deletes the data in one or all indexes, depending on whether you provide an `<index_name>` argument. Typically, you run `clean` before re-indexing all your data.

Note: The `clean` command does not work on clustered indexes.

How to use the clean command

Here are the main ways to use the `clean` command:

- To access the help page for `clean`, type:

```
splunk help clean
```

- To permanently remove data from **all indexes**, type:

```
splunk clean eventdata
```

- To permanently remove data from **a single index**, type:

```
splunk clean eventdata -index <index_name>
```
where `<index_name>` is the name of the targeted index.

- Add the `-f` parameter to force `clean` to skip its confirmation prompts.

Important: You must stop the indexer before you run the `clean` command.

Note: In pre-5.0 versions of Splunk Enterprise, running the `clean` command caused the indexer to reset the next bucket ID value for the index to 0. Starting with version 5.0, this is no longer the case. So, if the latest bucket ID was 3, after you run `clean`, the next bucket ID will be 4, not 0. For more information on bucket naming conventions and the bucket ID, see What the index directories look like.

Examples

This example removes data from all indexes:

```
splunk stop
splunk clean eventdata
```
This example removes data from the `_internal` index and forces Splunk to skip the confirmation prompt:

```
splunk stop
splunk clean eventdata -index _internal -f
```
Remove an index entirely

To remove an index entirely (and not just the data contained in it) from a non-clustered indexer, you can use Splunk Web or the CLI. You can also edit `indexes.conf` directly

Before removing an index, look through all `inputs.conf` files on your indexer and on any forwarders sending data to the indexer and make sure that none of the stanzas are directing data to the index you plan to delete. For example, if you want to delete an index called "nogood", make sure the following attribute/value pair does not appear in any of your input stanzas: `index=nogood`. Once the index has been deleted, the indexer will discard any data still being sent to that index.

To remove an index in Splunk Web, navigate to **Settings > Indexes** and click **Delete** to the right of the index you want to remove. This action deletes the index's data directories and removes the index's stanza from `indexes.conf`.

To remove an index through the CLI, run the `splunk remove index` command:

```
splunk remove index <index_name>
```
This command deletes the index's data directories and removes the index's stanza from `indexes.conf`.

You can run `splunk remove index` while the indexer is running. You do not need to restart the indexer after the command completes.

The index deletion process is ordinarily fast, but the duration depends on several factors:

- The amount of data being deleted.
- Whether you are currently performing heavy writes to other indexes on the same disk.

- Whether you have a large number of small `.tsidx` files in the index you're deleting.

You can also remove an index by editing `indexes.conf` directly and deleting the index's stanza. Restart the indexer and then remove the index's directories.

To remove an index from an indexer cluster, you must edit `indexes.conf` and delete the index's stanza. You cannot use Splunk Web or the CLI. As with all such changes on an indexer cluster, you first edit the file on the manager node and then apply the changes to the peer nodes. See Configure the peer indexes in an indexer cluster Once you've applied the `indexes.conf` changes and the peer nodes have restarted, remove the index's directories from each peer node.

Disable an index without removing it

Once an index is disabled, the indexer no longer accepts data targeted at it. However, disabling an index does not delete index data, and the operation is reversible.

You can disable an index in Splunk Web. To do this, navigate to **Settings > Indexes** and click **Disable** to the right of the index you want to disable. To re-enable the index, click **Enable** to the right of the index.

You can also disable an index with the CLI command `splunk disable index`:

```
splunk disable index <index_name>
```

To re-enable the index, use the `splunk enable index` command.

To disable an index for an indexer cluster, you must edit `indexes.conf` and set `disabled=true` in the index's stanza. You cannot use Splunk Web or the CLI. As with all such changes on an indexer cluster, you first edit the file on the manager node and then apply the changes to the peer nodes. See Configure the peer indexes in an indexer cluster

Remove older data based on retirement policy

When a bucket in an index reaches a specified age or when the index grows to a specified size, the bucket rolls to the "frozen" state, at which point the indexer removes it from the index. Just before removing the bucket, the indexer can save it to an archive, depending on how you configure your retirement policy.

For more information, see Set a retirement and archiving policy.

Manage pipeline sets for index parallelization

Index parallelization is a feature that allows an indexer to maintain multiple **pipeline sets**. A pipeline set handles the processing of data from ingestion of raw data, through event processing, to writing the events to disk. A pipeline set is one instance of the processing pipeline described in How indexing works. It is called a "pipeline set" because it comprises the individual pipelines, such as the parsing pipeline and the indexing pipeline, that together constitute the overall processing pipeline

By default, an indexer runs just a single pipeline set. However, if the underlying machine is under-utilized, in terms of available cores and I/O both, you can configure the indexer to run two pipeline sets. By running two pipeline sets, you potentially double the indexer's indexing throughput capacity.

Note: The actual amount of increased throughput on your indexer depends on the nature of your data inputs and other factors.

In addition, if the indexer is having difficulty handling bursts of data, index parallelization can help it to accommodate the bursts, assuming again that the machine has the available capacity.

To summarize, these are some typical use cases for index parallelization, dependent on available machine resources:

- Scale indexer throughput.
- Handle bursts of data.

For a better understanding of the use cases and to determine whether your deployment can benefit from multiple pipeline sets, see Parallelization settings in the *Capacity Planning Manual*.

Note: You cannot use index parallelization with multiple pipeline sets for data that is streaming from a single data source, such as UDP or TCP. Instead, use HTTP Event Collector for streaming data input. This restriction affects both metrics and event data.

Configure the number of pipeline sets

Caution: Before you increase the number of pipeline sets from the default of one, be sure that your indexer can support multiple pipeline sets. Read Parallelization settings in the *Capacity Planning Manual*. In addition, consult with Professional Services, particularly if you want to increase the number of pipeline sets beyond two.

To set the number of pipeline sets to two, change the `parallelIngestionPipelines` attribute in the `[general]` stanza of `server.conf`:

```
parallelIngestionPipelines = 2
```

You must restart the indexer for the change to take effect.

> Unless Professional Services advises otherwise, limit the number of pipeline sets to a maximum of 2.

How the indexer handles multiple pipeline sets

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 disk. The pipeline sets operate independently of each other, with no knowledge of each other's activities. The effect is essentially the same as if each pipeline set were running on its own, separate indexer.

When an indexer receives a new input, it assigns that input to a pipeline set, which then processes the input's data through the entire pipeline process, to the point of writing the input's data to an index on disk.

For example,Â if the indexer is directly ingesting a file, a single pipeline set processes the entire file. The pipeline sets do not share the file's data. Similarly, when a forwarder begins sending data to an indexer, the indexer assigns the entire input from that forwarder to a single pipeline set. The pipeline set continues to process all data from that forwarder until the forwarder switches to another indexer, assuming that the forwarder is load balancing across multiple indexers.

Each pipeline set can handle multiple inputs simultaneously.

Each pipeline set writes to its own set of hot buckets.

How the indexer allocates inputs to pipeline sets

When an indexer receives a new input, for example, from a forwarder that has just connected with it, the indexer allocates the input to one of its pipeline sets. Depending on the configuration, the indexer uses one of these allocation methods:

- round-robin selection
- weighted-random selection

In most cases, weighted-random selection is the preferred method. The default method, and the only method available for indexers running a pre-7.3 release, is round-robin selection.

Round-robin selection

By default, the indexer uses round-robin selection to allocate new inputs across its pipeline sets. With round-robin selection, the indexer simply allocates new inputs to each pipeline set in turn.

This method has the downside that it does not take into account the current loads on the pipeline sets. Pipeline sets can have widely varying loads, due to variation in size and complexity of different inputs. With round-robin selection, the indexer might, for example, allocate the next input to a heavily-loaded pipeline set while another pipeline set stands idly by.

Weighted-random selection

The weighted-random selection method considers the pipeline sets' relative loads. The indexer monitors the weighted loads of its pipeline sets over time and uses that information to choose the next pipeline set for data ingestion, in an attempt to balance the relative loads.

With the weighted-random method, each pipeline set reports metrics on its current processing load to the indexer. The indexer looks at the relative loads over a configurable time period and allocates the next new input to the pipeline set with the least overall load during that time period.

Configure the selection method

Specify the selection method with the `pipelineSetSelectionPolicy` setting in `server.conf`:

```
pipelineSetSelectionPolicy = round_robin | weighted_random
```

The default selection method is `round_robin`.

You can modify the behavior of the weighted-random selection policy through these settings in `server.conf`:

- `pipelineSetWeightsUpdatePeriod`
- `pipelineSetNumTrackingPeriods`

See the server.conf spec file for details.

View pipeline set activity

You can use the monitoring console to monitor most aspects of your deployment. This section discusses the console dashboard that provides insight into pipeline set performance.

The primary documentation for the monitoring console is located in *Monitoring Splunk Enterprise*.

For information on pipeline set performance, select the **Indexing Performance: Advanced** submenu under the **Indexing** menu.

The Indexing Performance: Advanced dashboard provides data on pipeline set performance on a per-indexer basis. You can use the dashboard to gain insight into the activity of the pipeline sets and their component pipelines.

This dashboard is mainly of use when troubleshooting performance issues in consultation with Splunk Support. Without expert-level knowledge of the underlying processes, it can be difficult to interpret the information sufficiently to determine performance issue remediation.

The effect of multiple pipeline sets on indexing settings

Some indexing settings are scoped to pipeline sets. These include any settings that are related to a pipeline, processor or queue. Examples of these include `max_fd` and `maxKBps` in `limits.conf` and `maxHotBuckets` in `indexes.conf`.

If you have multiple pipeline sets, these limits apply to each pipeline set individually, not to the indexer as a whole. For example, each pipeline set is separately subject to the `maxHotBuckets` limit. If you set `maxHotBuckets` to 4, each pipeline set is allowed a maximum of four hot buckets at a time, for a total of eight on an indexer with two pipeline sets.

Forwarders and multiple pipeline sets

You can also configure forwarders to run multiple pipeline sets. Multiple pipeline sets increase forwarder throughput and allow the forwarder to process multiple inputs simultaneously.

This can be of particular value, for example, when a forwarder needs to process a large file that would occupy the pipeline for a long period of time. With just a single pipeline, no other files can be processed until the forwarder finishes the large file. With two pipeline sets, the second pipeline can ingest and forward smaller files quickly, while the first pipeline continues to process the large file.

Assuming that the forwarder has sufficient resources and depending on the nature of the incoming data, a forwarder with two pipelines can potentially forward twice as much data as a forwarder with one pipeline.

How forwarders use multiple pipeline sets

When you enable multiple pipeline sets on a forwarder, each pipeline handles both data input and output. In the case of a heavy forwarder, each pipeline also handles parsing.

The forwarder forwards the output streams independently of each other. If the forwarder is configured for load balancing, it load balances each output stream separately. The receiving indexer handles each stream coming from the forwarder separately, as if each stream were coming from a different forwarder.

The pipeline sets on forwarders and indexers are entirely independent of each other. For example, a forwarder with multiple pipeline sets can forward to any indexer, no matter whether the indexer has one pipeline set or two. The forwarder does not know the pipeline configuration on the indexer, and it does not need to know it. Similarly, an indexer with multiple pipeline sets can receive data from any forwarder, no matter how many pipeline sets the forwarder has.

Configure pipeline sets on a forwarder

You configure the number of pipeline sets for forwarders in the same way as for indexers, with the `parallelIngestionPipelines` attribute in the `[general]` stanza of `server.conf`.

For heavy forwarders, the indexer guidelines apply: The underlying machine must be significantly under-utilized. You should generally limit the number of pipeline sets to two and consult with Professional Services. See Parallelization settings in the *Capacity Planning Manual*.

For universal forwarders, a single pipeline set uses, on average, around 0.5 of a core, but utilization can reach a maximum of 1.5 cores. Therefore, two pipeline sets will use between 1.0 and 3.0 cores. If you want to configure more than two pipeline sets on a universal forwarder, consult with Professional Services first.

As with indexers, you can specify the selection method with the `pipelineSetSelectionPolicy` setting in `server.conf`.

> In the case of monitored inputs, the selection method setting has no practical effect. Monitored input sources stick to the initially assigned pipeline until the instance is restarted.

Optimize indexes

While the indexer is indexing data, one or more instances of the `splunk-optimize` process will run intermittently, merging index files together to optimize performance when searching the data. The `splunk-optimize` process can use a significant amount of cpu but only briefly.

If `splunk-optimize` does not run frequently enough, searching will be less efficient.

`splunk-optimize` runs only on hot **buckets**. You can run it on warm buckets manually, if you find one with a larger number of index (`.tsidx`) files; typically, more than 25. To run `splunk-optimize`, go to `$SPLUNKHOME/bin` and type:

```
splunk-optimize -d|--directory <bucket_directory>
```

`splunk-optimize` accepts a number of optional parameters. To see a list of available parameters, type:

```
splunk-optimize
```

To enable verbose logging from `splunk-optimize` to `splunkd.log`, you can set `category.SplunkOptimize` in `log.cfg` to INFO or DEBUG. The recommended way to do this is through the CLI:

```
splunk set log-level SplunkOptimize -level DEBUG -auth admin:passwd
```

For more information on buckets, see How Splunk stores indexes.

Use the monitoring console to view indexing performance

You can use the monitoring console to monitor most aspects of your deployment. This topic discusses the console dashboards that provide insight into indexing performance.

The primary documentation for the monitoring console is located in *Monitoring Splunk Enterprise*.

There are two performance dashboards under the **Indexing** menu:

- Indexing Performance: Instance
- Indexing Performance: Deployment

As their names suggest, they provide similar information, scoped either to a single instance or to the entire deployment.

The performance dashboards provide a wide variety of information on indexing processes, such as:

- Indexing rate
- Queue fill ratios
- CPU information for various parts of the data pipeline

View the dashboards themselves for more information. In addition, see Indexing performance dashboards in *Monitoring Splunk Enterprise*.

Manage index storage

How the indexer stores indexes

As the indexer indexes your data, it creates a number of files:

- The raw data in compressed form (**the rawdata journal**)
- Indexes that point to the raw data (**tsidx files**)
- Some other metadata files

Together, these files constitute the Splunk Enterprise **index**. The files reside in sets of directories, or **buckets**, organized by age. Each bucket contains a rawdata journal, along with associated tsidx and metadata files. The data in each bucket is bounded by a limited time range.

An index typically consists of many buckets, and the number of buckets grows as the index grows. As data continues to enter the system, the indexer creates new buckets to accommodate the increase in data.

Some buckets on the indexer contain newly indexed data; others contain previously indexed data. The number of buckets in an index can grow quite large, depending on how much data you're indexing and how long you retain the data.

Why the details might, or might not, matter to you

The indexer handles indexed data by default in a way that gracefully ages the data through several states. After a long period of time, typically several years, the indexer removes old data from your system. You might well be fine with the default scheme it uses.

However, if you are indexing large amounts of data, have specific data retention requirements, or otherwise need to carefully plan your aging policy, you need to read this topic. Also, to back up your data, it helps to know where to find it. So, read on....

How data ages

A bucket moves through several states as it ages:

- hot
- warm
- cold
- frozen
- thawed

As buckets age, they "roll" from one state to the next. When data is first indexed, it gets written to a hot bucket. Hot buckets are buckets that are actively being written to. An index can have several hot buckets open at a time. Hot buckets are also searchable.

When certain conditions are met (for example, the hot bucket reaches a certain size or the indexer gets restarted), the hot bucket becomes a warm bucket ("rolls to warm"), and a new hot bucket is created in its place. The warm bucket is renamed but it remains in the same location as when it was a hot bucket. Warm buckets are searchable, but they are not actively written to. There can be a large number of warm buckets.

Once further conditions are met (for example, the index reaches some maximum number of warm buckets), the indexer begins to roll the warm buckets to cold, based on their age. It always selects the oldest warm bucket to roll to cold. Buckets continue to roll to cold as they age in this manner. Cold buckets reside in a different location from hot and warm buckets. You can configure the location so that cold buckets reside on cheaper storage.

Finally, after certain other time-based or size-based conditions are met, cold buckets roll to the frozen state, at which point they are deleted from the index, after being optionally archived.

If the frozen data has been archived, it can later be thawed. Data in thawed buckets is available for searches.

Settings in indexes.conf determine when a bucket moves from one state to the next.

Here are the states that buckets age through:

Bucket state	Description	Searchable?
Hot	New data is written to hot buckets. Each index has one or more hot buckets.	Yes
Warm	Buckets rolled from hot. New data is not written to warm buckets. An index has many warm buckets.	Yes
Cold	Buckets rolled from warm and moved to a different location. An index has many cold buckets.	Yes
Frozen	Buckets rolled from cold. The indexer deletes frozen buckets, but you can choose to archive them first. Archived buckets can later be thawed.	No
Thawed	Buckets restored from an archive. If you archive frozen buckets, you can later return them to the index by thawing them.	Yes

Note: For indexes enabled with the **SmartStore** feature, which places data on a remote store such as S3, the cold state does not ordinarily exist. See Bucket states and SmartStore.

What the index directories look like

Each index occupies its own directory under `$SPLUNK_HOME/var/lib/splunk`. The name of the directory is the same as the index name. Under the index directory are a series of subdirectories that categorize the buckets by state (hot/warm, cold, or thawed).

Each bucket is a subdirectory within those directories. The bucket names indicate the age of the data they contain.

Here is the directory structure for the default index (`defaultdb`):

Bucket state	Default location	Notes
Hot	`$SPLUNK_HOME/var/lib/splunk/defaultdb/db/*`	Each hot bucket occupies its own subdirectory.
Warm	`$SPLUNK_HOME/var/lib/splunk/defaultdb/db/*`	Each warm bucket occupies its own subdirectory.
Cold	`$SPLUNK_HOME/var/lib/splunk/defaultdb/colddb/*`	Each cold bucket occupies its own subdirectory. When warm buckets roll to cold, they get moved to this directory.

Bucket state	Default location	Notes
Frozen	When buckets freeze, they get deleted or archived into a location that you specify.	Deletion is the default. See Archive indexed data for information on how to archive the data instead.
Thawed	`$SPLUNK_HOME/var/lib/splunk/defaultdb/thaweddb/*`	Buckets that are archived and later thawed reside in this directory. See Restore archived data for information on restoring archived data to a thawed state.

The paths for the hot/warm, cold, and thawed directories are configurable. See Configure index storage and Use multiple partitions for index data.

All index locations must be writable.

Note: In pre-6.0 versions of Splunk Enterprise, replicated copies of indexer cluster buckets always resided in the `colddb` directory, even if they were hot or warm buckets. Starting with 6.0, hot and warm replicated copies reside in the `db` directory, the same as for non-replicated copies.

Bucket naming conventions

Bucket names depend on:

- The state of the bucket: hot or warm/cold/thawed
- The type of bucket directory: non-clustered, clustered originating, or clustered replicated

Important: Bucket naming conventions are subject to change.

Non-clustered buckets

A standalone indexer creates non-clustered buckets. These use one type of naming convention.

Clustered buckets

An indexer that is part of an **indexer cluster** creates clustered buckets. A clustered bucket has multiple exact copies. The naming convention for clustered buckets distinguishes the types of copies, originating or replicated.

Briefly, a bucket in an indexer cluster has multiple copies, according to its **replication factor**. When the data enters the cluster, the receiving indexer writes the data to a hot bucket. This receiving indexer is known as the source cluster **peer**, and the bucket where the data gets written is called the originating copy of the bucket.

As data is written to the hot copy, the source peer streams copies of the hot data, in blocks, to other indexers in the cluster. These indexers are referred to as the target peers for the bucket. The copies of the streamed data on the target peers are known as replicated copies of the bucket.

When the source peer rolls its originating hot bucket to warm, the target peers roll their replicated copies of that bucket. The warm copies are exact replicas of each other.

For an introduction to indexer cluster architecture and replicated data streaming, read Basic indexer cluster architecture.

Bucket names

These are the naming conventions:

Bucket type	Hot bucket	Warm/cold/thawed bucket
Non-clustered	`hot_v1_<localid>`	`db_<newest_time>_<oldest_time>_<localid>`
Clustered originating	`hot_v1_<localid>`	`db_<newest_time>_<oldest_time>_<localid>_<guid>`
Clustered replicated	`<localid>_<guid>`	`rb_<newest_time>_<oldest_time>_<localid>_<guid>`

Note:

- `<newest_time>` and `<oldest_time>` are timestamps indicating the age of the data in the bucket. The timestamps are expressed in UTC epoch time (in seconds). For example: `db_1223658000_1223654401_2835` is a warm, non-clustered bucket containing data from October 10, 2008, covering the period of 4pm - 5pm.
- `<localid>` is an ID for the bucket. For a clustered bucket, the originating and replicated copies of the bucket have the same `<localid>`.
- `<guid>` is the `guid` of the source peer node. The `guid` is located in the peer's `$SPLUNK_HOME/etc/instance.cfg` file.

In an indexer cluster, the originating warm bucket and its replicated copies have identical names, except for the prefix (`db` for the originating bucket; `rb` for the replicated copies).

Note: In an indexer cluster, when data is streamed from the source peer to a target peer, the data first goes into a temporary directory on the target peer, identified by the hot bucket convention of `<localid>_<guid>`. This is true for any replicated bucket copy, whether or not the streaming bucket is a hot bucket. For example, during bucket fix-up activities, a peer might stream a warm bucket to other peers. When the replication of that bucket has completed, the `<localid>_<guid>` directory is rolled into a warm bucket directory, identified by the `rb_` prefix.

Buckets and Splunk Enterprise administration

When you are administering Splunk Enterprise, it helps to understand how the indexer stores indexes across buckets. In particular, several admin activities require a good understanding of buckets:

- For information on setting a retirement and archiving policy, see Set a retirement and archiving policy. You can base the retirement policy on either the size or the age of data.
- For information on how to archive your indexed data, see Archive indexed data. To learn how to restore data from archive, read Restore archived data.
- To learn how to back up your data, read Back up indexed data. That topic also discusses how to manually roll hot buckets to warm, so that you can then back them up.
- For information on setting limits on disk usage, see Set limits on disk usage.
- For a list of configurable bucket settings, see Configure index storage.
- For information on configuring index size, see Configure index size.
- For information on partitioning index data, see Use multiple partitions for index data.
- For information on how buckets function in indexer clusters, see Buckets and indexer clusters.

- For information on buckets and SmartStore, see Bucket states and SmartStore.

In addition, see indexes.conf in the *Admin Manual*.

Configure index storage

You configure indexes in `indexes.conf`. How you edit `indexes.conf` depends on whether you're using **index replication**, also known as indexer clustering:

- **For non-clustered indexes,** edit the version of `indexes.conf` in `$SPLUNK_HOME/etc/system/local/`, or create one if it does not already exist there. Do not edit the copy in `$SPLUNK_HOME/etc/system/default`. For information on configuration files and directory locations, see About configuration files.
- **For clustered indexes,** create or edit a version of `indexes.conf` on the cluster manager node and then distribute it to all the peer nodes, as described in Configure the peer indexes in an indexer cluster.

For non-clustered indexes only, you can optionally use Splunk Web to configure the path to your indexes. Go to **Settings > Server settings > General settings**. Under the section **Index settings**, set the field **Path to indexes**. After doing this, you must restart the indexer from the CLI, not from within Splunk Web. Most other settings, however, require direct editing of `indexes.conf`.

Attributes that affect index buckets

This table lists the key `indexes.conf` attributes affecting **buckets** and what they configure. It also provides links to other topics that show how to use these attributes. For the most detailed information on these attributes, as well as others, always refer to the indexes.conf spec file.

Be careful when adjusting settings, such as maxWarmDBCount, so as to cause the reduction in the number of existing warm buckets. The reduction process, if involving a significant number of buckets, has the potential to overwhelm your network. You can minimize the effect of warm bucket reduction by reducing the setting's value gradually.

Note: This list is specific to non-SmartStore indexes. The set of attributes that control **SmartStore** indexes is significantly different. See Configure SmartStore.

Attribute	What it configures	Default	For more information, see ...
homePath	The path that contains the hot and warm buckets. (**Required.**) This location must be writable.	`$SPLUNK_HOME/var/lib/splunk/defaultdb/db/` (for the default index only)	Configure index path attributes
coldPath	The path that contains the cold buckets. (**Required.**) This location must be writable.	`$SPLUNK_HOME/var/lib/splunk/defaultdb/colddb/` (for the default index only)	Configure index path attributes
thawedPath	The path that contains any thawed buckets. (**Required.**) This location must be writable.	`$SPLUNK_HOME/var/lib/splunk/defaultdb/thaweddb/` (for the default index only)	Configure index path attributes

Attribute	What it configures	Default	For more information, see ...
repFactor	Determines whether the index gets replicated to other cluster peers. (**Required for indexes on cluster peer nodes.**)	0 (which means that the index will not get replicated to other peers; the correct behavior for non-clustered indexes). For clustered indexes, you *must* set `repFactor` to `auto`, which causes the index to get replicated.	Configure the peer indexes in an indexer cluster
maxHotBuckets	The maximum number of concurrent hot buckets. This value should be at least 2, to deal with any archival data. The **main** default index, for example, has this value set to 10.	3, for new, custom indexes.	How data ages
maxDataSize	**Determines rolling behavior, hot to warm.** The maximum size for a hot bucket. When a hot bucket reaches this size, it rolls to warm. This attribute also determines the approximate size for all buckets.	Special value "auto", which sets the size to 750MB.	How data ages
maxWarmDBCount	**Determines rolling behavior, warm to cold.** The maximum number of warm buckets. When the maximum is reached, warm buckets begin rolling to cold.	300	Use multiple partitions for index data
maxTotalDataSizeMB	**Determines rolling behavior, cold to frozen.** The maximum size of an index. When this limit is reached, cold buckets begin rolling to frozen.	500000 (MB)	Set a retirement and archiving policy
frozenTimePeriodInSecs	**Determines rolling behavior, cold to frozen.** Maximum age for a bucket, after which it rolls to frozen.	188697600 (in seconds; approx. 6 years)	Set a retirement and archiving policy
coldToFrozenDir	Location for archived data. Determines behavior when a bucket rolls from cold to frozen. If set, the indexer will archive frozen buckets into this directory just before deleting them from the index.	If you don't set either this attribute or `coldToFrozenScript`, the indexer will just log the bucket's directory name and then delete it once it rolls to frozen.	Archive indexed data
coldToFrozenScript	Script to run just before a cold bucket rolls to frozen. If you set both this attribute and `coldToFrozenDir`, the indexer will use `coldToFrozenDir` and ignore this attribute.	If you don't set either this attribute or `coldToFrozenDir`, the indexer will just log the bucket's directory name and then delete it once it rolls to frozen.	Archive indexed data
homePath.maxDataSizeMB **coldPath.maxDataSizeMB**	Maximum size for `homePath` (hot/warm bucket storage) or `coldPath` (cold bucket storage). If either attribute is missing or set to 0, its path is not individually constrained in size.	None	Configure index size according to bucket type
maxVolumeDataSizeMB	Maximum size for a volume. If the attribute is missing, the individual volume is not constrained in size.	None	Configure index size with volumes

Configure index path attributes

When creating a new index, you configure several index path attributes, for example, `homePath` and `coldPath`. When you configure path attributes, follow these restrictions and recommendations:

- The path must be writable. In the case of `homePath`, the parent path must also be writable.
- Do not use environment variables in index paths. The only exception to this is `SPLUNK_DB`.
- The path cannot be a root directory, such as `homePath=/myindex` or `homePath=C:\myindex`.
- It is recommended that you specify the path using `$_index_name` as placeholder for the index name. For example:

```
homePath = $SPLUNK_DB/$_index_name/db
```

  At run time, the indexer expands `$_index_name` to the name of the index. For example, if the index name is "newindex", `homePath` becomes `$SPLUNK_DB/newindex/db`.

The set of index path attributes includes:

- `homePath`
- `coldPath`
- `thawedPath`
- `bloomHomePath`
- `summaryHomePath`
- `tstatsHomePath`

For more information on path attributes, see the indexes.conf spec file.

For information on using multiple partitions to hold your index data, see Use multiple partitions for index data .

Index size and indexer clusters

Note: This section pertains to non-SmartStore indexes only. Clusters handle sizing of SmartStore indexes differently. See Configure data retention for SmartStore indexes.

The attributes that control the size of a non-SmartStore index and its number of buckets operate on each peer node individually. They do not operate across the cluster.

For example, consider the `maxTotalDataSizeMB` attribute. This attribute specifies the maximum size of the index. Its value is applied on a per-peer basis to limit the size of the index on each peer. When an index reaches its maximum size on a particular peer node, the peer freezes the oldest bucket in its copy of the index.

This means that the size of an index on a peer node is determined by the total size of all bucket copies for that index on that peer node. It doesn't matter whether the copies are primary copies, searchable copies, non-searchable copies, or excess copies. They all count toward the index size on that peer.

Because a cluster usually does not distribute bucket copies perfectly evenly across the set of peer nodes, an index typically has a different size on each peer node. This means that the index might reach its maximum size on one peer while still having room to grow on the other peers.

To handle this situation, each peer tells the manager when it freezes a copy of a bucket. At that point, the manager no longer initiates fix-up activities for the frozen bucket. The manager does not, however, instruct the other peers to freeze their copies of that bucket. Each peer will subsequently freeze its copy of the bucket, if any, when its copy of the index

reaches the maximum size limit. See How the cluster handles frozen buckets.

Note: Although these attributes operate separately on each peer, you should set them to the same values across all peers in the cluster. See Configure the peer indexes in an indexer cluster.

For help in sizing your cluster disk space needs, see Storage considerations.

Rawdata journal compression

When the indexer indexes data, it writes the data to the **rawdata journal**. The indexer compresses the data during this process. By default, the compression uses the `zstd` algorithm, but you can change the compression algorithm through the `journalCompression` setting in indexes.conf. Available compression algorithms are:

- `gzip`
- `lz4`
- `zstd`

If you change the compression algorithm, journals in new buckets will be compressed using the changed method, but journals in existing buckets will continue to be stored through the compression method with which they were originally indexed.

Move the index database

You can move the index database from one location to another. You do this by changing the path definition of `SPLUNK_DB` through the command-line interface of your operating system.

The procedures in this topic assume that the index database is in the default location, created during installation.

If you move individual indexes or parts of an index to separate locations, the procedures in this topic are not valid. For information on the structure of Splunk Enterprise indexes, see How the indexer stores indexes. For information on how to change the location for a single index, see Configure index storage.

Note: Although you can use Splunk Web to change the locations of individual indexes or index volumes, you cannot use it to change the default storage location of indexes, `SPLUNK_DB`.

For *nix users

Prerequisties

Make sure the target file system has at least 1.2 times the size of the total amount of raw data that you plan to index.

Steps

1. Create the target directory with write permissions for the user that Splunk Enterprise runs as. For example, if Splunk Enterprise runs as user "splunk", give it ownership of the directory:

```
mkdir /foo/bar
chown splunk /foo/bar/
```

For information on setting the user that Splunk Enterprise runs as, see Run Splunk Enterprise as a different or non-root user in the *Installation Manual.*

2. Stop the indexer:

```
splunk stop
```

3. Copy the index file system to the target directory:

```
cp -rp $SPLUNK_DB/* /foo/bar/
```

4. Unset the `SPLUNK_DB` environment variable:

```
unset SPLUNK_DB
```

5. Change the `SPLUNK_DB` attribute in `$SPLUNK_HOME/etc/splunk-launch.conf` to specify the new index directory:

```
SPLUNK_DB=/foo/bar
```

6. Start the indexer:

```
splunk start
```

The indexer picks up where it left off, reading from, and writing to, the new copy of the index.

7. You can delete the old index database after verifying that the indexer can read and write to the new location.

For Windows users

Prerequisties

Make sure the target drive or directory has at least 1.2 times the size of the total amount of raw data that you plan to index.

Caution: Do not use mapped network drives for index stores.

Steps

1. From a command prompt, make sure that the target directory has permissions that allow the `splunkd` process to write to that directory:

```
C:\Program Files\Splunk> D:
D:\> mkdir \new\path\for\index
D:\> cacls D:\new\path\for\index /T /E /G <the user Splunk Enterprise runs as>:F
```

For more information about determining the user Splunk Enterprise runs as, see Install on Windows in the *Installation Manual.*

2. Stop the indexer:

```
splunk stop
```

You can also use the Services control panel to stop the `splunkd` and `splunkweb` services.

3. Copy the existing index file system to the target directory:

```
xcopy "C:\Program Files\Splunk\var\lib\splunk\*.*" D:\new\path\for\index /s /e /v /o /k
```
4. Unset the `SPLUNK_DB` environment variable:

```
set SPLUNK_DB=
```
5. Edit the `SPLUNK_DB` attribute in `%SPLUNK_HOME%\etc\splunk-launch.conf` to specify the new index directory:

```
SPLUNK_DB=D:\new\path\for\index
```
If the line in the configuration file that contains the `SPLUNK_DB` attribute has a pound sign (#) as its first character, remove the #.

6. Start the indexer:

```
splunk start
```
The indexer picks up where it left off, reading from, and writing to, the new copy of the index.

7. You can delete the old index database after verifying that the indexer can read and write to the new location.

Use multiple partitions for index data

Note: This topic is not relevant to **SmartStore** indexes. See About SmartStore.

The indexer can use multiple partitions for its index data. It's possible to configure the indexer to use many disks/partitions/filesystems on the basis of multiple indexes and **bucket** types, so long as you mount them correctly and point to them properly from `indexes.conf`. However, for most purposes, the best practice is to use a single high performance file system to hold the index data.

If you do use multiple partitions, the most common way to arrange the index data is to keep the hot/warm buckets on the local machine and to put the cold buckets on a separate array of disks suitable for longer term storage. You'll want to store your hot/warm buckets on a machine with with fast read/write partitions, because most searching will happen there.

Configure multiple partitions

To configure multiple partitions:

1. Set up partitions just as you'd normally set them up in any operating system.

2. Mount the disks/partitions.

3. Edit indexes.conf to point to the correct paths for the partitions. You set paths on a per-index basis, so you can set separate partitions for different indexes. Each index has its own `[<index>]` stanza, where `<index>` is the name of the index. These are the main path settings:

- `homePath` is the path that contains the hot and warm buckets for the index.
- `coldPath` is the path that contains the cold buckets for the index.
- `thawedPath` is the path that contains any thawed buckets for the index.

See Configure index path attributes for guidelines on defining index paths.

Configure maximum index size

Note: This topic is not relevant to **SmartStore** indexes. See Configure data retention for SmartStore indexes.

You can configure maximum index size in a number of ways:

- On a per-index basis
- For hot/warm and cold **buckets** separately
- Across indexes, using volumes

To configure index storage size, you set attributes in indexes.conf. For more information on the attributes mentioned in this topic, read "Configure index storage".

Caution: While processing indexes, the indexer might occasionally exceed the configured maximums for short periods of time. When setting limits, be sure to factor in some buffer space. Also, note that certain systems, such as most Unix systems, maintain a configurable reserve space on their partitions. You must take that reserve space, if any, into account when determining how large your indexes can grow.

Configure index size for each index

To set the maximum index size on a per-index basis, use the `maxTotalDataSizeMB` attribute. When this limit is reached, buckets begin rolling to frozen.

Configure index size according to bucket type

To set the maximum size for `homePath` (hot/warm bucket storage) or `coldPath` (cold bucket storage), use the `maxDataSizeMB` settings:

```
# set hot/warm storage to 10,000MB
homePath.maxDataSizeMB = 10000
# set cold storage to 5,000MB
coldPath.maxDataSizeMB = 5000
```

The `maxDataSizeMB` attributes can be set globally or for each index. An index-level setting will override a global setting. To control bucket storage across groups of indexes, use the `maxVolumeDataSizeMB` attribute, described below.

When the size of the `homePath` directory exceeds `homePath.maxDataSizeMB`, the indexer rolls the oldest warm bucket to cold, moving the bucket to the `coldPath` directory.

When the size of the `coldPath` directory exceeds `coldPath.maxDataSizeMB`, the indexer rolls the oldest cold bucket to frozen.

Configure index size with volumes

You can manage disk usage across multiple indexes by creating volumes and specifying maximum data size for them. A volume represents a directory on the file system where indexed data resides.

Volumes can store data from multiple indexes. You would typically use separate volumes for hot/warm and cold buckets.

For instance, you can set up one volume to contain the hot/warm buckets for all your indexes, and another volume to contain the cold buckets.

You can use volumes to define `homePath` and `coldPath`. You cannot use them to define `thawedPath`.

In addition, you must use volumes if you explicitly define `bloomHomePath`.

Configure a volume

To set up a volume, use this syntax:

```
[volume:<volume_name>]
path = <pathname_for_volume>
```

You can also optionally include a `maxVolumeDataSizeMB` attribute, which specfies the maximum size for the volume.

For example:

```
[volume:hot1]
path = /mnt/fast_disk
maxVolumeDataSizeMB = 100000
```

The example defines a volume called "hot1", located at `/mnt/fast_disk`, with a maximum size of 100,000MB.

Similarly, this stanza defines a volume called "cold1" that uses a maximum of 150,000MB:

```
[volume:cold1]
path = /mnt/big_disk
maxVolumeDataSizeMB = 150000
```

Use a volume

Once you configure volumes, you can use them to define an index's `homePath` and `coldPath`. For example, using the volumes configured above, you can define two indexes:

```
[idx1]
homePath = volume:hot1/idx1
coldPath = volume:cold1/idx1

[idx2]
homePath = volume:hot1/idx2
coldPath = volume:cold1/idx2
```

You can use volumes to manage index storage space in any way that makes sense to you. Usually, however, volumes correlate to hot/warm and cold buckets, because of the different storage requirements typical when dealing with different bucket types. So, you will probably use some volumes exclusively for designating `homePath` (hot/warm buckets) and others for `coldPath` (cold buckets).

When a volume containing warm buckets reaches its `maxVolumeDataSizeMB`, it starts rolling buckets to cold. When a volume containing cold buckets reaches its `maxVolumeDataSizeMB`, it starts rolling buckets to frozen. If a volume contains both warm and cold buckets (which will happen if an index's `homePath` and `coldPath` are both set to the same volume), the oldest bucket will be rolled to frozen.

Put it all together

This example shows how to use the per-index `homePath.maxDataSizeMB` and `coldPath.maxDataSizeMB` attributes in combination with volumes to maintain fine-grained control over index storage. In particular, it shows how to use those attributes to prevent bursts of data into one index from triggering massive bucket moves from other indexes. You can use these per-index settings to ensure that no index will ever occupy more than a specified size, thereby alleviating that concern.

```
# global settings

# Inheritable by all indexes: No hot/warm directory (homePath) can exceed 1 TB.
# Individual indexes can override this setting.
homePath.maxDataSizeMB = 1000000

# volumes

[volume:caliente]
path = /mnt/fast_disk
maxVolumeDataSizeMB = 100000

[volume:frio]
path = /mnt/big_disk
maxVolumeDataSizeMB = 1000000

# indexes

[i1]
homePath = volume:caliente/i1
# homePath.maxDataSizeMB is inherited from the global setting
coldPath = volume:frio/i1
# coldPath.maxDataSizeMB not specified anywhere:
# This results in no size limit - old-style behavior

[i2]
homePath = volume:caliente/i2
homePath.maxDataSizeMB = 1000
# overrides the global default
coldPath = volume:frio/i2
coldPath.maxDataSizeMB = 10000
# limits the size of cold buckets

[i3]
homePath = /old/style/path
homePath.maxDataSizeMB = 1000
coldPath = volume:frio/i3
coldPath.maxDataSizeMB = 10000
```

Set limits on disk usage

Note: This topic is not relevant to **SmartStore** indexes. See Initiate eviction based on occupancy of the cache's disk partition for information on how SmartStore controls local disk usage.

Splunk Enterprise uses several methods to control disk space. Indexes consume most of the disk space. If you run out of disk space, the indexer stops indexing. You can set a minimum free space limit to control how low free disk space falls before indexing stops. Indexing resumes once space exceeds the minimum.

Note: To determine how much space you need for your indexes, see "Estimate your storage requirements" in the *Capacity Planning Manual*.

Set minimum free disk space

You can set a minimum amount of free disk space for the disk where indexed data is stored. If the limit is reached, the indexer stops operating. Both indexing and searching are affected:

- Periodically, the indexer checks space on all partitions that contain indexes. If the free disk space limit has been reached on any of those partitions, the indexer stops indexing data until more space is available. A UI banner and `splunkd` warning are posted to indicate the need to clear more disk space.
- Before attempting to launch a search, the indexer requires that the specified amount of free space be available on the file system where the dispatch directory is stored, `$SPLUNK_HOME/var/run/splunk/dispatch`

The default minimum free disk space is 5000MB.

Note:

- The indexer does not clear any of its disk space with this method. It simply pauses until more space becomes available.
- Incoming data can be lost while indexing is suspended.

You can set minimium free disk space through Splunk Web, the CLI, or the `server.conf` configuration file.

In Splunk Web

To specify minimum disk usage in Splunk Web:

1. Click **Settings** in the upper right portion of Splunk Web.

2. Click **Server settings**.

3. Click **General settings**.

4. Under the **Index settings** section, set the field **Pause indexing if free disk space (in MB) falls below** to the desired minimum free disk space in megabytes.

5. Click **Save**.

6. Restart the indexer for your changes to take effect.

From the command line interface (CLI)

You can set the minimum free disk space with the CLI. This example sets the minimum free disk space to 20,000MB (20GB):

```
splunk set minfreemb 20000
splunk restart
```

For information on using the CLI, see "About the CLI" in the Admin manual.

In server.conf

You can also set the minimum free disk space in the server.conf file. The relevant stanza/attribute is this:

```
[diskUsage]
minFreeSpace = <num>
```
Note that `<num>` represents megabytes. The default is 5000.

Control index storage

The indexes.conf file contains index configuration settings. You can control disk storage usage by specifying maximum index size or maximum age of data. When one of these limits is reached, the oldest indexed data will be deleted (the default) or archived. You can archive the data by using a predefined archive script or creating your own.

For detailed instructions on how to use `indexes.conf` to set maximum index size or age, see "Set a retirement and archiving policy".

For detailed information on index storage, see "How the indexer stores indexes".

Reduce tsidx disk usage

There are 2 options available to minimize the disk space used by **tsidx files**. You can configure additional compression and optimizations through the use of tsidxWritingLevel, and schedule the removal of the tsidx files using a tsidx retention policy.

The tsidx retention policy

The tsidx retention policy determines how long the indexer retains the tsidx files that it uses to search efficiently and quickly across its data. By default, the indexer retains the tsidx files for all its indexed data for as long as it retains the data itself. By adjusting the policy to remove tsidx files associated with older data, you can set the optimal trade-off between storage costs and search performance.

The indexer stores tsidx files in **buckets** alongside the **rawdata files**. The tsidx files are vital for efficient searching across large amounts of data. They also occupy substantial amounts of storage.

For data that you are regularly running searches across, you absolutely need the tsidx files. However, if you have data that requires only infrequent searching as it ages, you can adjust the tsidx retention policy to reduce the tsidx files once they reach a specified age. This allows you to reduce the disk space that your indexed data occupies.

The **tsidx reduction** process eliminates the full-size tsidx files and replaces them with mini versions of those files that contain essential metadata. The rawdata files and some other metadata files remain untouched. You can continue to search across the aged data, if necessary,Â but such searches will exhibit significantly worse performance. Rare term searches, in particular, will run slowly.

To summarize, the main use case for tsidx reduction is for environments where most searches run against recent data. In that case, fast access to older data might not be worth the cost of storing the tsidx files. By reducing tsidx files for older data, you incur little performance hit for most searches while gaining large savings in disk usage.

Estimate the storage savings

Tsidx reduction replaces a bucket's full-size tsidx files with smaller versions of those files, known as mini-tsidx files. It also eliminates the bucket's `merged_lexicon.lex` file.

The full-size tsidx files usually constitute a large portion of the overall bucket size. The exact amount depends on the type of data. Data with many unique terms requires larger tsidx files. As a general guideline, the tsidx reduction process decreases bucket size by approximately one-third to two-thirds. For example, a 1GB bucket decreases in size to somewhere between 350MB and 700MB.

To make a rough estimate of a bucket's reduction potential, look at the size of its `merged_lexicon.lex` file. The `merged_lexicon.lex` file is an indicator of the number of unique terms in a bucket's data. Buckets with larger `merged_lexicon.lex` files have tsidx files that reduce to a greater degree, because of the greater number of unique terms.

The size of a mini-tsidx file is generally about 5% to 10% of the size of the corresponding original, full-size file. As mentioned earlier, however, the overall reduction in bucket size is less than that - typically, one-third to two-thirds. This is because, in addition to the mini-tsidx files, the **reduced bucket** retains the rawdata file and a number of metadata files.

How tsidx reduction works

When you enable tsidx reduction, you specify a reduction age, on a per-index basis. When buckets in that index reach the specified age, the indexer reduces their tsidx files.

The reduction process

The tsidx reduction process runs, by default, every ten minutes. It checks each bucket in the index and reduces the tsidx files in any bucket whose most recent event is at least the specified reduction age.

The reduction process runs on only a single bucket at a time. If multiple buckets are ready for reduction, the process handles them sequentially.

The reduction process is fast. For example, when running on a 1GB bucket, it typically completes in just a few seconds.

Once a tsidx file is reduced, it stays reduced. If you disable the tsidx reduction setting or increase the reduction age, the change affects only buckets that are not already reduced. If necessary, however, there is a way to convert reduced buckets back into buckets with full tsidx files. See Restore reduced buckets to their original state.

Effect of reduction on bucket files

The tsidx reduction process eliminates the full-size tsidx files from each targeted bucket after replacing them with mini versions that contain only essential metadata. The mini-tsidx file consists of the header of the original tsidx file, which contains metadata about each event. In addition, tsidx reduction eliminates the bucket's `merged_lexicon.lex` file.

The bucket retains its rawdata file, along with the mini-tsidx files and certain other metadata files, including the `bloomfilter` file.

Full size tsidx files have a `.tsidx` filename extension. Mini-tsidx files use the `.mini.tsidx` extension.

> The full-size version of the tsidx file gets deleted only after the mini version has been created. This means that the bucket will briefly contain both versions of the file, with the commensurate increase in disk usage.

Effect of reduction on in-progress searches

If a search is in progress on a particular bucket that qualifies for tsidx reduction, the reduction for that bucket will be delayed until the search on the bucket completes. The mini-tsidx files will be created but deletion of the full-size files will await the search completion.

If the indexer is performing a search that ranges across multiple buckets, including one that is ready for reduction, reduction of the bucket might complete before the search reaches it. As expected, when the search does reach the reduced bucket, it will run slowly on that bucket.

Searches across reduced buckets

Once a bucket has undergone tsidx reduction, you can run searches across the bucket, but they will take much longer to complete. Since the indexer searches the most recent buckets first, it will return results from all non-reduced buckets before it reaches the reduced buckets.

When the search hits the reduced buckets, a message appears in Splunk Web to warn users of a potential delay in search completion: "Search on most recent data has completed. Expect slower search speeds as we search the reduced buckets."

The following search commands do not work with reduced buckets: `typeahead`, `tstats`, and `walklex`. A warning is added to `search.log` if search using these commands touches a reduced bucket: "The full buckets will return results and the reduced buckets will return 0 results." In addition, for the `tstats`command only, the following message appears in Splunk Web: "Reduced buckets were found in index={index}. Tstats searches are not supported on reduced buckets. Search results will be incorrect."

Tsidx reduction does not touch tsidx files for accelerated data models, which are maintained in their own directories, separate from the index buckets. Therefore, tstats commands that are restricted to an accelerated data model will continue to function normally and are not affected by this feature.

Configure the tsidx retention policy

By default, the indexer retains all tsidx files for the life of the buckets. To change the policy, you must enable tsidx reduction.

You can also change the tsidx retention period from its default of seven days. A bucket gets reduced only when all events in the bucket exceed the retention period.

Configure through Splunk Web

To enable tsidx reduction on an index, edit the index:

1. Navigate to **Settings > Indexes**.

2. Click the name of the index that you want to edit.

3. Go to the Storage Optimization section of the Edit screen.

4. In the Tsidx Retention Policy field, click **Enable Reduction.**

5. To modify the default retention period, edit the "Reduce tsidx files older than" field.

6. Click **Save**.

Configure in indexes.conf

You can enable tsidx reduction by directly editing `indexes.conf`. You can enable reduction for one or more indexes individually or for all indexes globally.

To enable tsidx reduction for a single index, place the relevant attributes under the index's stanza in `indexes.conf`. For example, to enable reduction for the "newone" index and to set the retention period to 10 days:

```
[newone]
enableTsidxReduction = true
timePeriodInSecBeforeTsidxReduction = 864000
```

To enable tsidx reduction for all indexes, place the settings under the `[default]` stanza.

You must restart the indexer for the settings to take effect.

Configure through the CLI

To enable tsidx reduction, with a 10 day retention period, on an index called "newone":

```
splunk edit index newone -enableTsidxReduction true -timePeriodInSecBeforeTsidxReduction 864000
```

You do not need to restart the indexer after running this command.

Performance impact when you first enable tsidx reduction

Once you enable tsidx reduction, the indexer begins to look for buckets to reduce. It reduces all buckets that exceed the specified retention period. The indexer reduces only one bucket at a time, so performance impact should be minimal.

Determine whether a bucket is reduced

Run the `dbinspect` search command:

```
| dbinspect index=_internal
```

The `tsidxState` field in the results specifies "full" or "mini" for each bucket.

Tsidx reduction and indexer clusters

An indexer cluster runs tsidx reduction in the same way, and according to the same rules and settings, as a standalone indexer. However, since only searchable bucket copies have tsidx files to begin with, reduction only occurs on searchable copies. With tsidx reduction enabled, a searchable bucket copy can contain either a full-size or a mini tsidx file, depending on the age of the bucket.

You must push changes to the tsidx reduction settings by means of the **configuration bundle** method. This ensures that all peer nodes use the same settings. Tsidx reduction then occurs at approximately the same time for all searchable copies of a reduction-ready bucket, no matter what peers they reside in.

If, post-reduction, the cluster must convert a non-searchable copy of a reduced bucket to searchable to meet the search factor, there are two ways that the conversion can proceed:

- If another searchable copy of the bucket exists in the cluster, the cluster will stream that copy's mini-tsidx files to the non-searchable copy. When streaming is complete, the copy is considered searchable.

- If no other searchable copy of the bucket exists, the cluster has no mini-tsidx files available for streaming to the non-searchable copy. In that case, the cluster must first build full-size tsidx files from the non-searchable copy's rawdata file and then reduce the full-size files. There is no way to create mini-tsidx files directly from a rawdata file.

For more information on how an indexer cluster makes non-searchable copies of a bucket searchable, see Bucket-fixing scenarios.

Restore reduced buckets to their original state

You cannot restore reduced buckets to their original state merely by increasing the age setting for tsidx reduction. That setting does not affect buckets that have already been reduced.

Instead, to revert a bucket with mini-tsidx files to full-size tsidx files:

1. Stop the indexer.

2. In `indexes.conf`, either disable tsidx reduction or increase the age setting for tsidx reduction beyond the age of the buckets that you want to restore. Otherwise, the bucket will be reduced for a second time soon after you revert it.

3. Run the `splunk rebuild` command on the bucket:

```
splunk rebuild <bucket directory> <index name>
```
See "Rebuild a single bucket."

4. Restart the indexer.

Restrictions on tsidx reduction

SmartStore

You cannot reduce tsidx files for buckets in **SmartStore** indexes. However, you can still search buckets that were tsidx-reduced before migration to SmartStore. See About SmartStore.

Indexed real-time searches

Tsidx reduction is not compatible with indexed real-time searches. Also, tsidx reduction should not be enabled on indexes for apps that rely on indexed real-time searches; for example, the index `itsi_tracked_alerts` in ITSI.

The tsidx writing level

The Splunk platform uses time-series index files that maintain a list of keywords for use in accelerating the selection of index buckets while searching for data. By default, the indexer retains the tsidx files for as long as it retains the event data. While the tsidx files are important for efficiently searching across large amounts of data, they also occupy a substantial amount of storage space. By adjusting the tsidx writing level, you can reduce the aggregate storage impact of maintaining the tsidx files.

There are multiple optimizations available to improve the tsidx compression. These optimizations are encapsulated in levels, with new levels added in higher releases of Splunk Enterprise. Changing the default `tsidxWritingLevel` setting in indexes.conf changes the optimizations used by both the index tsidx files and data model accelerations. See indexes.conf in the *Admin Manual*.

Compatibility for tsidxWritingLevel

As higher releases of Splunk Enterprise add new levels, there are improved optimizations available. Changing the `tsidxWritingLevel` setting to the highest level limits backward compatibility for searching data.

Splunk Enterprise Version	The tsidxWritingLevel supported	The supported tsidx level for searching	Default value
7.2.x	1, 2	1, 2	1
7.3.x	1, 2, 3	1, 2, 3	1
8.0.x	1, 2, 3	1, 2, 3	1
8.1.x	1, 2, 3, 4	1, 2, 3, 4	1
8.2.x	1, 2, 3, 4	1, 2, 3, 4	2
9.x.x	1, 2, 3, 4	1, 2, 3, 4	3

Example: An index bucket written by a Splunk Enterprise 8.1.x indexer configured with `tsidxWritingLevel=3` can be searched on Splunk Enterprise 7.3.x. An index bucket written by a Splunk Enterprise 8.3.x indexer configured with `tsidxWritingLevel=4` cannot be searched on versions of Splunk Enterprise lower than 8.1.x.

You do not need to restart the instance for the change to take effect.

If your infrastructure planning requires backwards compatibility with a Splunk Enterprise release lower than 8.1.x, choose a lower tsidxWritingLevel.

Expected behavior after changing the setting

Reviewing and adjusting the `tsidxWritingLevel` to a higher level is recommended after the indexers or cluster peers have been upgraded to a higher release of Splunk Enterprise.

- A change to the `tsidxWritingLevel` is applied to new index bucket tsidx files. There is no change to the existing tsidx files.
- A change to the `tsidxWritingLevel` is applied to newly accelerated data models, or after a rebuild of the existing data models is initiated. All existing data model accelerations will not be affected.

To upgrade multisite indexer clusters without search interruption, defer changing tsidxWritingLevel until after the multisite indexer cluster upgrade is complete. Configure all peer nodes to switch to the new level simultaneously.

Combining tsidx settings

The `tsidxWritingLevel` setting can be used in conjunction with a tsidx retention policy to significantly lower the storage used by high-value, immediately searchable data, and the reduce the long-term storage impact of that data.

Determine which indexes.conf changes require restart

Some changes to `indexes.conf` require that you restart the indexer for the changes to take effect:

- Changing any of these attributes: `rawChunkSizeBytes`, `minRawFileSyncSecs`, `syncMeta`, `maxConcurrentOptimizes`, `coldToFrozenDir`, `coldtoFrozenScript`, `memPoolMB`, `maxRunningProcessGroups`, `maxVolumeDataSizeMB`
- Changing any of these attributes for existing indexes: `repFactor`, `homePath`, `coldPath`, `thawedPath`, `bloomHomePath`, `summaryHomePath`, `tstatsHomePath`, `remotePath`, `coldPath.maxDataSizeMB`, `datatype`
- Adding or removing a volume
- Enabling or disabling an index that contains data

You do not need to restart the indexer if you only make these changes:

- Adding new index stanzas.
- Removing an index
- Changing any attributes not listed as requiring restart
- Enabling or disabling an index that contains no data

The configuration changes that cause the peer nodes in an indexer cluster to undergo a rolling restart are a superset of those listed here. See Restart or reload after configuration bundle changes?

Note: For information on other configuration changes, outside of `indexes.conf`, that require a restart, see When to restart Splunk Enterprise after a configuration file change in the *Admin Manual.*

Use the monitoring console to view index and volume status

You can use the monitoring console to monitor most aspects of your deployment. This topic discusses the console dashboards that provide insight into indexing performance.

The primary documentation for the monitoring console is located in *Monitoring Splunk Enterprise*.

There are several dashboards that monitor the status of indexes and volumes. The dashboards are scoped either to a single instance or to the entire deployment. They are located under the **Indexing** menu:

- Indexes and Volumes: Instance
- Indexes and Volumes: Deployment
- Index Detail: Instance
- Index Detail: Deployment
- Volume Detail: Instance
- Volume Detail: Deployment

These dashboards provide a wealth of information about your indexes and volumes, such as:

- Disk usage by index
- Volume usage
- Index and volume size over time
- Data age
- Statistics for bucket types
- Bucket settings

View the dashboards themselves for more information. In addition, see "Indexing: Indexes and volumes" in *Monitoring Splunk Enterprise.*

Back up and archive your indexes

Back up indexed data

Note: Much of this topic is not relevant to **SmartStore** indexes. See About SmartStore.

To decide how to back up indexed data, it helps to understand first how the indexer stores data and how the data ages once it has been indexed. Then you can decide on a backup strategy.

Before you read this topic, you should look at "How the indexer stores indexes" to get familiar with the structure of indexes and the options for configuring them. But if you want to jump right in, the next section below attempts to summarize the key points from that topic.

How data ages

Indexed data resides in database directories consisting of subdirectories called **buckets**. Each index has its own set of databases.

As data ages, it moves through several types of buckets. You determine how the data ages by by configuring attributes in indexes.conf. Read "Configure index storage" for a description of the settings in `indexes.conf` that control how data ages.

Briefly, here is a somewhat simplified version of how data ages in an index:

1. When the indexer first indexes data, it goes into a "hot" bucket. Depending on your configuration, there can be several hot buckets open at one time. Hot buckets cannot be backed up because the indexer is actively writing to them, but you can take a snapshot of them.

2. The data remains in the hot bucket until the policy conditions are met for it to be reclassified as "warm" data. This is called "rolling" the data into the warm bucket. This happens when a hot bucket reaches a specified size or age, or whenever `splunkd` gets restarted. When a hot bucket is rolled, its directory is renamed, and it becomes a warm bucket. (You can also manually roll a bucket from hot to warm, as described below.) It is safe to back up the warm buckets.

3. When the index reaches one of several possible configurable limits, usually a specified number of warm buckets, the oldest bucket becomes a "cold" bucket. The indexer moves the bucket to the `colddb` directory. The default number of warm buckets is 300.

4. Finally, at a time based on your defined policy requirements, the bucket rolls from cold to "frozen". The indexer deletes frozen buckets. However, if you need to preserve the data, you can tell the indexer to archive the data before deleting the bucket. See "Archive indexed data" for more information.

You can set retirement and archiving policy by controlling several different parameters, such as the size of indexes or buckets or the age of the data.

To summarize:

- **hot buckets** - Currently being written to; do not back these up.
- **warm buckets** - Rolled from hot; can be safely backed up.
- **cold buckets** - Rolled from warm; buckets are moved to another location.
- **frozen buckets** - The indexer deletes these, but you can archive their contents first.

You set the locations of index databases in `indexes.conf`. (See below for detailed information on the database locations for the default index.) You also specify numerous other attributes there, such as the maximum size and age of hot buckets.

Locations of the index database directories

Here's the directory structure for the default index (`defaultdb`):

Bucket type	Default location	Notes
Hot	`$SPLUNK_HOME/var/lib/splunk/defaultdb/db/*`	There can be multiple hot subdirectories. Each hot bucket occupies its own subdirectory, which uses this naming convention: `hot_v1_<ID>`
Warm	`$SPLUNK_HOME/var/lib/splunk/defaultdb/db/*`	There are separate subdirectories for each warm bucket. These are named as described below in "Warm/cold bucket naming convention".
Cold	`$SPLUNK_HOME/var/lib/splunk/defaultdb/colddb/*`	There are multiple cold subdirectories. When warm buckets roll to cold, they get moved into this directory, but are not renamed.
Frozen	N/A: Frozen data gets deleted or archived into a directory location you specify.	Deletion is the default; see "Archive indexed data" for information on how to archive the data instead.
Thawed	`$SPLUNK_HOME/var/lib/splunk/defaultdb/thaweddb/*`	Location for data that has been archived and later thawed. See "Restore archived data" for information on restoring archived data to a thawed state.

The paths for hot/warm and cold directories are configurable, so you can store cold buckets in a separate location from hot/warm buckets. See "Configure index storage" and "Use multiple partitions for index data".

Important: All index locations must be writable.

Choose your backup strategy

There are two basic backup scenarios to consider:

- Ongoing, incremental backups of warm data
- Backup of all data - for example, before upgrading the indexer

How you actually perform the backup will, of course, depend entirely on the tools and procedures in place at your organzation, but this section should help provide you the guidelines you need to proceed.

Incremental backups

The general recommendation is to schedule backups of any new warm buckets regularly, using the incremental backup utility of your choice. If you're rolling buckets frequently, you should also include the cold database directory in your backups, to ensure that you don't miss any buckets that have rolled to cold before they've been backed up. Since bucket directory names don't change when they roll from warm to cold, you can just filter by name.

To back up hot buckets as well, you need to take a snapshot of the files, using a tool like VSS (on Windows/NTFS), ZFS snapshots (on ZFS), or a snapshot facility provided by the storage subsystem. If you do not have a snapshot tool

available, you can manually roll a hot bucket to warm and then back it up, as described below. However, this is not generally recommended, for reasons also discussed below.

Back up all data

It is recommended that you back up all your data before upgrading the indexer. This means the hot, warm, and cold buckets.

There are obviously a number of ways to do this, depending on the size of your data and how much downtime you can afford. Here are some basic guidelines:

- For smaller amounts of data, shut down the indexer and just make a copy of your database directories before performing the upgrade.
- For larger amounts of data, you will probably instead want to snapshot your hot buckets prior to upgrade.

In any case, if you have been doing incremental backups of your warm buckets as they've rolled from hot, you should really need to backup only your hot buckets at this time.

Rolling buckets manually from hot to warm

To roll the buckets of an index manually from hot to warm, use the following CLI command, replacing `<index_name>` with the name of the index you want to roll:

```
splunk _internal call /data/indexes/<index_name>/roll-hot-buckets â  auth <admin_username>:<admin_password>
```

Important: It is ordinarily not advisable to roll hot buckets manually, as each forced roll permanently decreases search performance over the data. As a general rule, larger buckets are more efficient to search. By prematurely rolling buckets, you're producing smaller, less efficient buckets. In cases where hot data needs to be backed up, a snapshot backup is the preferred method.

Note: It is not recommended to roll hot buckets manually in an environment that leverages accelerated data-model summaries. If a hot bucket is rolled while other index-management jobs are in progress, the integrity of the data may be compromised.

Recommendations for recovery

If you experience a non-catastrophic disk failure (for example you still have some of your data, but the indexer won't run), Splunk recommends that you move the index directory aside and restore from a backup rather than restoring on top of a partially corrupted datastore. The indexer will automatically create hot directories on startup as necessary and resume indexing. Monitored files and directories will pick up where they were at the time of the backup.

Clustered data backups

Even though an **indexer cluster** already contains redundant copies of data, you might also want to back up the cluster data to another location; for example, to keep a copy of the data offsite as part of an overall disaster recovery plan.

The simplest way to do this is to back up the data on each individual peer node on your cluster, in the same way that you back up data on individual, non-clustered indexers, as described earlier in this topic. However, this approach will result in backups of duplicate data. For example, if you have a cluster with a **replication factor** of 3, the cluster is storing three copies of all the data across its set of peer nodes. If you then back up the data residing on each individual node, you end up with backups containing, in total, three copies of the data. You cannot solve this problem by backing up just the data

on a single node, since there's no certainty that a single node contains all the data in the cluster.

The solution to this would be to identify exactly one copy of each bucket on the cluster and then back up just those copies. However, in practice, it is quite a complex matter to do that. One approach is to create a script that goes through each peer's index storage and uses the bucket ID value contained in the bucket name to identify exactly one copy of each bucket. The bucket ID is the same for all copies of a bucket. For information on the bucket ID, read "Warm/cold bucket naming convention". Another thing to consider when designing a cluster backup script is whether you want to back up just the bucket's rawdata or both its rawdata and index files. If the latter, the script must also identify a searchable copy of each bucket.

Because of the complications of cluster backup, it is recommended that you contact Splunk Professional Services for guidance in backing up single copies of clustered data. They can help design a solution customized to the needs of your environment.

Set a retirement and archiving policy

Note: Most of this topic is not relevant to **SmartStore** indexes. See Configure data retention for SmartStore indexes.

Configure data retirement and **archiving** policy by controlling the size of indexes or the age of data in indexes.

The indexer stores indexed data in directories called **buckets**. Buckets go through four stages of retirement. When indexed data reaches the final, frozen state, the indexer removes it from the index. You can configure the indexer to archive the data when it freezes, instead of deleting it entirely. See "Archive indexed data" for details.

Bucket stage	Description	Searchable?
Hot	Contains newly indexed data. Open for writing. One or more hot buckets for each index.	Yes
Warm	Data rolled from hot. There are many warm buckets.	Yes
Cold	Data rolled from warm. There are many cold buckets.	Yes
Frozen	Data rolled from cold. The indexer deletes frozen data by default, but you can also archive it. Archived data can later be thawed.	No

You configure the sizes, locations, and ages of indexes and their buckets by editing indexes.conf, as described in "Configure index storage".

Caution: When you change your data retirement and archiving policy settings, the indexer can delete old data without prompting you.

Set attributes for cold to frozen rolling behavior

The `maxTotalDataSizeMB` and `frozenTimePeriodInSecs` attributes in `indexes.conf` help determine when buckets roll from cold to frozen. These attributes are described in detail below.

Freeze data when an index grows too large

You can use the size of an index to determine when data gets frozen and removed from the index. If an index grows larger than its maximum specified size, the oldest data is rolled to the frozen state.

The default maximum size for an index is 500,000MB. To change the maximum size, edit the `maxTotalDataSizeMB` attribute in `indexes.conf`. For example, to specify the maximum size as 250,000MB:

```
[main]
maxTotalDataSizeMB = 250000
```
Specify the size in megabytes.

Restart the indexer for the new setting to take effect. Depending on how much data there is to process, it can take some time for the indexer to begin to move buckets out of the index to conform to the new policy. You might see high CPU usage during this time.

This setting works with `frozenTimePeriodInSecs` to determine when data gets frozen. Data rolls to frozen when either setting is reached.

If maxTotalDataSizeMB is reached before frozenTimePeriodInSecs, data will be rolled to frozen before the configured time period has elapsed. If archiving policy has not been properly configured, unintended data loss can occur.

Freeze data when it grows too old

You can use the age of data to determine when a bucket gets rolled to frozen. When the most recent data in a particular bucket reaches the configured age, the entire bucket is rolled.

To specify the age at which data freezes, edit the `frozenTimePeriodInSecs` attribute in `indexes.conf`. This attribute specifies the number of seconds to elapse before data gets frozen. The default value is 188697600 seconds, or approximately 6 years. This example configures the indexer to cull old events from its index when they become more than 180 days (15552000 seconds) old:

```
[main]
frozenTimePeriodInSecs = 15552000
```
Specify the time in seconds.

Depending on how much data there is to process, it can take some time for the indexer to begin to move buckets out of the index to conform to the new policy. You might see high CPU usage during this time.

Archive data

If you want to archive frozen data instead of deleting it entirely, you must tell the indexer to do so, as described in "Archive indexed data". You can create your own archiving script or you can just let the indexer handle the archiving for you. You can later restore ("thaw") the archived data, as described in "Restore archived data".

Other ways that buckets age

There are a number of other conditions that can cause buckets to roll from one stage to another, some of which can also trigger deletion or archiving. These are all configurable, as described in "Configure index storage". For a full understanding of all your options for controlling retirement policy, read that topic and look at the indexes.conf spec file.

For example, the indexer rolls buckets when they reach their maximum size. You can reduce bucket size by setting a smaller `maxDataSize` in `indexes.conf` so they roll faster. But note that it takes longer to search more small buckets than fewer large buckets. To get the results you are after, you will have to experiment a bit to determine the right size for your

buckets.

Troubleshoot the archive policy

I ran out of disk space so I changed the archive policy, but it's still not working

If you changed your archive policy to be more restrictive because you've run out of disk space, you may notice that events haven't started being archived according to your new policy. This is most likely because you must first free up some space so the process has room to run. Stop the indexer, clear out ~5GB of disk space, and then start the indexer again. After a while (exactly how long depends on how much data there is to process) you should see INFO entries about `BucketMover` in `splunkd.log` showing that buckets are being archived.

Archive indexed data

Note: Although **SmartStore** indexes do not usually contain cold buckets, you still use the attributes described here (`coldToFrozenDir` and `coldToFrozenScript`) to archive SmartStore buckets as they roll directly from warm to frozen. See Configure data retention for SmartStore indexes.

You can configure the indexer to archive your data automatically as it ages; specifically, at the point when it rolls to "frozen". To do this, you configure indexes.conf.

Caution: By default, the indexer deletes all frozen data. It removes the data from the index at the moment it becomes frozen. If you need to keep the data around, you **must** configure the indexer to archive the data before removing it. You do this by either setting the `coldToFrozenDir` attribute or specifying a valid `coldToFrozenScript` in `indexes.conf`.

For detailed information on data storage, see How the indexer stores indexes. For information on editing `indexes.conf`, see Configure index storage.

How the indexer archives data

The indexer rotates old data out of the index based on your data retirement policy, as described in Set a retirement and archiving policy. Data moves through several stages, which correspond to file directory locations. Data starts out in the **hot** database, located as subdirectories ("**buckets**") under `$SPLUNK_HOME/var/lib/splunk/defaultdb/db/`. It then moves to the **warm** database, also located as subdirectories under `$SPLUNK_HOME/var/lib/splunk/defaultdb/db`. Eventually, data is aged into the **cold** database `$SPLUNK_HOME/var/lib/splunk/defaultdb/colddb`.

Finally, data reaches the **frozen** state. This can happen for a number of reasons, as described in Set a retirement and archiving policy. At this point, the indexer erases the data from the index. If you want the indexer to archive the frozen data before erasing it from the index, you must specify that behavior. You can choose two ways of handling the archiving:

- Let the indexer perform the archiving automatically.
- Specify an archiving script for the indexer to run.

The archiving behavior depends on which of these `indexes.conf` attributes you set:

- `coldToFrozenDir`. This attribute specifes a location where the indexer will automatically archive frozen data.
- `coldToFrozenScript`. This attribute specifes a user-supplied script that the indexer will run when the data is frozen. Typically, this will be a script that archives the frozen data. The script can also serve some other purpose altogether. While the indexer ships with one example archiving script that you can edit and use (`$SPLUNK_HOME/bin/coldToFrozenExample.py`), you can actually specify any script you want the indexer to run.

Note: You can only set one or the other of these attributes. The `coldToFrozenDir` attribute takes precedence over `coldToFrozenScript`, if both are set.

If you don't specify either of these attributes, the indexer runs a default script that simply writes the name of the bucket being erased to the log file `$SPLUNK_HOME/var/log/splunk/splunkd_stdout.log`. It then erases the bucket.

Let the indexer archive the data for you

If you set the `coldToFrozenDir` attribute in `indexes.conf`, the indexer will automatically copy frozen buckets to the specified location before erasing the data from the index.

Add this stanza to `$SPLUNK_HOME/etc/system/local/indexes.conf`:

```
[<index>]
coldToFrozenDir = <path to frozen archive>
```

Note the following:

- `<index>` specifies which index contains the data to archive.
- `<path to frozen archive>` specifies the directory where the indexer will put the archived buckets.

Note: When you use Splunk Web to create a new index, you can also specify a frozen archive path for that index. See Create custom indexes for details.

How the indexer archives the frozen data depends on whether the data was originally indexed in a pre-4.2 release:

- For buckets created from version 4.2 and on, the indexer will remove all files except for the **rawdata file**.
- For pre-4.2 buckets, the script simply gzip's all the `.tsidx` and `.data` files in the bucket.

This difference is due to a change in the format of rawdata. Starting with 4.2, the rawdata file contains all the information the indexer needs to reconstitute an index bucket.

For information on thawing these buckets, see Restore archived indexed data.

Specify an archiving script

If you set the `coldToFrozenScript` attribute in `indexes.conf`, the script you specify will run just before the indexer erases the frozen data from the index.

You'll need to supply the actual script. Typically, the script will archive the data, but you can provide a script that performs any action you want.

Add this stanza to `$SPLUNK_HOME/etc/system/local/indexes.conf`:

```
[<index>]
coldToFrozenScript = ["<path to program that runs script>"] "<path to script>"
```

Note the following:

- `<index>` specifies which index contains the data to archive.
- `<path to script>` specifies the path to the archiving script. The script must be in `$SPLUNK_HOME/bin` or one of its subdirectories.

- `<path to program that runs script>` is optional. You must set it if your script requires a program, such as python, to run it.
- If your script is located in `$SPLUNK_HOME/bin` and is named `myColdToFrozen.py`, set the attribute like this:

```
coldToFrozenScript = "$SPLUNK_HOME/bin/python" "$SPLUNK_HOME/bin/myColdToFrozen.py"
```

- For detailed information on the archiving script, see the indexes.conf spec file.

The indexer ships with an example archiving script that you can edit, `$SPLUNK_HOME/bin/coldToFrozenExample.py`.

Note: If using the example script, edit it to specify the archive location for your installation. Also, rename the script or move it to another location to avoid having changes overwritten when you upgrade the indexer. **This is an example script** and should not be applied to a production instance without editing to suit your environment and testing extensively.

The example script archives the frozen data differently, depending on whether the data was originally indexed in a pre-4.2 release:

- For buckets created from version 4.2 and on, it will remove all files except for the rawdata file.
- For pre-4.2 buckets, the script simply gzip's all the `.tsidx` and `.data` files.

This difference is due to a change in the format of rawdata. Starting with 4.2, the rawdata file contains all the information the indexer needs to reconstitute an index bucket.

For information on thawing these buckets, see Restore archived indexed data.

As a best practice, make sure the script you create completes as quickly as possible, so that the indexer doesn't end up waiting for the return indicator. For example, if you want to archive to a slow volume, set the script to copy the buckets to a temporary location on the same (fast) volume as the index. Then use a separate script, outside the indexer, to move the buckets from the temporary location to their destination on the slow volume.

Data archiving and indexer clusters

In an **Indexer cluster**, each individual peer node rolls its buckets to frozen, in the same way that a non-clustered indexer does; that is, based on its own set of configurations. Because all peers in a cluster should be configured identically, all copies of a bucket should roll to frozen at approximately the same time.

However, there can be some variance in the timing, because the same index can grow at different rates on different peers. The cluster performs processing to ensure that buckets freeze smoothly across all peers in the cluster. Specifically, it performs processing so that, if a bucket is frozen on one peer but not on another, the cluster does not initiate fix-up activities for that bucket. See How the cluster handles frozen buckets.

The problem of archiving multiple copies

Because indexer clusters contain multiple copies of each bucket. If you archive the data using the techniques described earlier in this topic, you archive multiple copies of the data.

For example, if you have a cluster with a **replication factor** of 3, the cluster stores three copies of all its data across its set of peer nodes. If you set up each peer node to archive its own data when it rolls to frozen, you end up with three archived copies of the data. You cannot solve this problem by archiving just the data on a single node, since there's no certainty that a single node contains all the data in the cluster.

The solution to this would be to archive just one copy of each bucket on the cluster and discard the rest. However, in practice, it is not feasible to create a process to reliably do so.

Specifying the archive destination

If you choose to archive multiple copies of the clustered data, you must guard against name collisions. You cannot route the data from all peer nodes into a single archive directory, because multiple, identically named copies of the bucket will exist across the cluster (for deployments where replication factor >= 2), and the contents of a directory must be named uniquely. Instead, you need to ensure that the buckets from each of your peer nodes go to a separate archive directory. This, of course, will be somewhat difficult to manage if you specify a destination directory in shared storage by means of the `coldToFrozenDir` attribute in `indexes.conf`, because the `indexes.conf` file must be the same across all peer nodes, as discussed in Configure the peer indexes in an indexer cluster. One alternative approach would be to create a script that directs each peer's archived buckets to a separate location on the shared storage, and then use the `coldToFrozenScript` attribute to specify that script.

Restore archived indexed data

You restore archived data by first moving an archived **bucket** into the thawed directory which you previously configured for its index in `indexes.conf`. By default, the thawed directory is `$SPLUNK_DB/<your_index_name>/thaweddb`. For example, `$SPLUNK_DB/important_data/thaweddb`.

You then process the bucket, as described later in this topic. Data in the thawed directory is not subject to the server's index aging scheme (hot > warm> cold > frozen). You can put archived data in the thawed directory for as long as you need it. When the data is no longer needed, simply delete it or move it out of thawed.

Important: You restore archived data differently depending on whether it was originally indexed in Splunk Enterprise version 4.2 or later. This is because Splunk Enterprise changed its rawdata format in 4.2.

See "Archive indexed data" for information on how to archive data in the first place. You can also use that page as guidance if you want to re-archive data after you've thawed it.

Restored data does not count against your license.

Restrictions when restoring an archive to a different instance of the indexer

For the most part, you can restore an archive to any instance of the indexer, not just the one that originally indexed it. This, however, depends on a couple of factors:

- **Splunk Enterprise version.** You cannot restore a bucket created by Splunk Enterprise 4.2 or later to a pre-4.2 indexer. The bucket data format changed between 4.1 and 4.2, and pre-4.2 indexers do not understand the new format. This means:
  - **4.2+ buckets:** You can restore a 4.2+ bucket to any 4.2+ instance.
  - **Pre-4.2 buckets:** You can restore a pre-4.2 bucket to any indexer, pre-4.2 or post-4.2, aside from a few OS-related issues, described in the next bullet.

- **OS version.** You can usually restore buckets to an indexer running on a different OS. Specifically:
  - **4.2+ buckets:** You can restore a 4.2+ bucket to an indexer running any operating system.
  - **Pre-4.2 buckets:** You can restore a pre-4.2 bucket to an indexer running any operating system, with the restriction that you cannot restore pre-4.2 data to a system of different endian-ness. For example, data generated on 64-bit systems is not likely to work well on 32-bit systems, and data cannot be moved from

PowerPC or Sparc systems to x86 or x86-64 systems, and vice versa.

In addition, make sure that you do not introduce bucket ID conflicts to your index when restoring the archived bucket. This issue is discussed later.

How to tell whether your archive bucket contains 4.2+ data

Before thawing the archive bucket, you need to identify whether the archive bucket is pre- or post-4.2. Here's how to tell the difference, assuming you archived the buckets using `coldToFrozenDir` or the provided example script:

- **4.2+ bucket:** The bucket directory contains only the rawdata directory, which contains `journal.gz`.
- **Pre-4.2 bucket:** The bucket directory contains gzipped versions of `.tsidx` and `.data` files, along with a rawdata directory containing files named `<int>.gz`.

Important: If you archived the data through some script of your own, the resulting bucket could contain just about anything.

If you archived the buckets using `coldToFrozenDir` or the provided example script, you can use the following procedures to thaw them.

Thaw a 4.2+ archive

nix users

Here is an example of safely restoring a 4.2+ archive bucket to thawed:

1. Copy your archive bucket into the thawed directory:

```
cp -r db_1181756465_1162600547_1001 $SPLUNK_HOME/var/lib/splunk/defaultdb/thaweddb
```

Note: The bucket id cannot conflict with any other bucket in the index. This example assumes that the bucket id '1001' is unique for the index. If it isn't, choose some other, non-conflicting bucket ID.

2. Execute the `splunk rebuild` command on the archive bucket to rebuild the indexes and associated files:

```
splunk rebuild $SPLUNK_HOME/var/lib/splunk/defaultdb/thaweddb/db_1181756465_1162600547_1001
```

3. Restart the indexer:

```
splunk restart
```

Windows users

Here is an example of safely restoring a 4.2+ archive bucket to thawed:

1. Copy your archive bucket into the thawed directory:

```
xcopy
D:\MyArchive\db_1181756465_1162600547_1001 %SPLUNK_HOME%\var\lib\splunk\defaultdb\thaweddb\db_1181756465
_1162600547_1001 /s /e /v
```

Note: The bucket id cannot conflict with any other bucket in the index. This example assumes that the bucket id '1001' is unique for the index. If it isn't, choose some other, non-conflicting bucket ID.

2. Execute the `splunk rebuild` command on the archive bucket to rebuild the indexes and associated files:

```
splunk rebuild %SPLUNK_HOME%\var\lib\splunk\defaultdb\thaweddb\db_1181756465_1162600547_1001
```

3. Restart the indexer:

```
splunk restart
```

Thaw a pre-4.2 archive

nix users

Here is an example of safely restoring a pre-4.2 archive bucket to thawed:

1. Copy your archive bucket to a temporary location in the thawed directory:

```
# cp -r db_1181756465_1162600547_0
 $SPLUNK_HOME/var/lib/splunk/defaultdb/thaweddb/temp_db_1181756465_1162600547_0
```

2. If the bucket was compressed when originally archived, uncompress the contents in the thawed directory.

3. Rename the temporary bucket to something that the indexer will recognize:

```
# cd $SPLUNK_HOME/var/lib/splunk/defaultdb/thaweddb/
# mv temp_db_1181756465_1162600547_0 db_1181756465_1162600547_1001
```

Note: You must choose a bucket id that does not conflict with any other bucket in the index. This example assumes that the bucket id '1001' is unique for the index. If it isn't, choose some other, non-conflicting bucket ID.

4. Refresh the manifests:

```
# cd $SPLUNK_HOME/bin
# ./splunk login
# ./splunk _internal call /data/indexes/main/rebuild-metadata-and-manifests
```

After a few moments, the contents of your newly thawed bucket should be searchable again.

Windows users

Here is an example of safely restoring a pre-4.2 archive bucket to thawed:

1. Copy your archive bucket to the thawed directory:

```
> xcopy
D:\MyArchive\db_1181756465_1162600547_0 %SPLUNK_HOME%\var\lib\splunk\defaultdb\thaweddb\temp_db_1181756465
_1162600547_0 /s /e /v
```

2. If the bucket was compressed when originally archived, uncompress the contents in the thawed directory.

3. Rename the temporary bucket to something that the indexer will recognize:

```
> cd %SPLUNK_HOME%\var\lib\splunk\defaultdb\thaweddb
> move temp_db_1181756465_1162600547_0 db_1181756465_1162600547_1001
```

Note: You must choose a bucket id that does not conflict with any other bucket in the index. This example assumes that the bucket id '1001' is unique for the index. If it isn't, choose some other, non-conflicting bucket ID.

4. Refresh the manifests:

```
> cd %SPLUNK_HOME%\bin
> splunk login
> splunk _internal call /data/indexes/main/rebuild-metadata-and-manifests
```

After a few moments, the contents of your newly thawed bucket should be searchable again.

Clustered data thawing

You can thaw archived clustered data onto individual peer nodes the same way that you thaw data onto any individual indexer. However, as described in "Archive indexed data", it is difficult to archive just a single copy of clustered data in the first place. If, instead, you archive data across all peer nodes in a cluster, you can later thaw the data, placing the data into the thawed directories of the peer nodes from which it was originally archived. You will end up with replication factor copies of the thawed data on your cluster, since you are thawing all of the original data, including the copies.

Note: Data does not get replicated from the thawed directory. So, if you thaw just a single copy of some bucket, instead of all the copies, only that single copy will reside in the cluster, in the thawed directory of the peer node where you placed it.

SmartStore indexes and bucket thawing

For information on thawing data archived from a SmartStore index, see Thawing data and SmartStore.

Overview of indexer clusters and index replication

About indexer clusters and index replication

Indexer clusters are groups of Splunk Enterprise indexers configured to replicate each others' data, so that the system keeps multiple copies of all data. This process is known as **index replication**. By maintaining multiple, identical copies of Splunk Enterprise data, clusters prevent data loss while promoting data availability for searching.

Indexer clusters feature automatic failover from one indexer to the next. This means that, if one or more indexers fail, incoming data continues to get indexed and indexed data continues to be searchable.

The key benefits of index replication are:

- **Data availability.** An indexer is always available to handle incoming data, and the indexed data is available for searching.
- **Data fidelity.** You never lose any data. You have assurance that the data sent to the cluster is exactly the same data that gets stored in the cluster and that a search can later access.
- **Data recovery.** Your system can tolerate downed indexers without losing data or losing access to data.
- **Disaster recovery.** With multisite clustering, your system can tolerate the failure of an entire data center.
- **Search affinity.** With multisite clustering, search heads can access the entire set of data through their local sites, greatly reducing long-distance network traffic.

The key trade-off in index replication is between the benefits of data availability/recovery and the costs of storage (and, to a minor degree, increased processing load). The degree of data recovery that the cluster possesses is directly proportional to the number of copies of data it maintains. But maintaining more copies of data means higher storage requirements. To manage this trade-off to match the needs of your enterprise, you can configure the number of copies of data that you want the cluster to maintain. This is known as the **replication factor**.

Note: Clusters with **SmartStore** indexes rely on the inherent capabilities of a remote object store to ensure high availability, data fidelity, data recovery, and disaster recovery for most of the index data. See About SmartStore.

You can also use clusters to scale indexing capacity, even in situations where index replication is not a requirement. See "Use indexer clusters to scale indexing".

Note: Search head clusters provide high availability and scalabilty for groups of search heads. They are a separate feature from indexer clusters, but you can combine them with indexer clusters to build a high availability, scalable solution across your entire Splunk Enterprise deployment. See "About search head clustering" in the *Distributed Search* manual.

Parts of an indexer cluster

An indexer cluster is a group of Splunk Enterprise instances, or **nodes**, that, working in concert, provide a redundant indexing and searching capability. Each cluster has three types of nodes:

- A single **manager node** to manage the cluster.
- Several to many **peer nodes** to index and maintain multiple copies of the data and to search the data.
- One or more **search heads** to coordinate searches across the set of peer nodes.

The **manager node** manages the cluster. It coordinates the replicating activities of the peer nodes and tells the search head where to find data. It also helps manage the configuration of peer nodes and orchestrates remedial activities if a

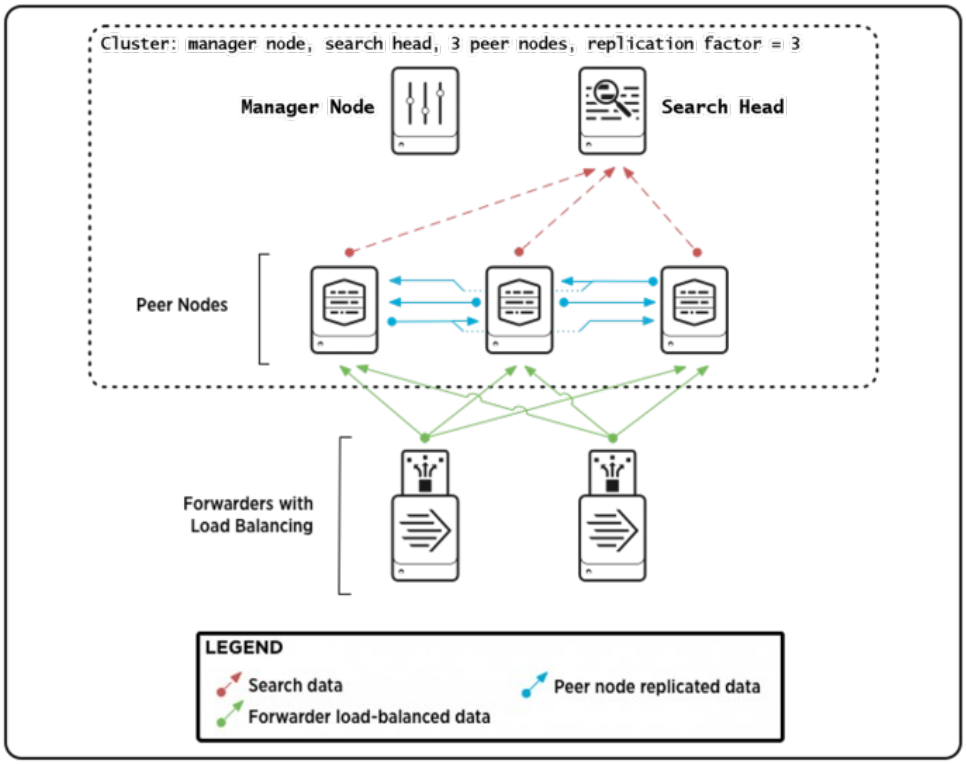
peer goes down.

The **peer nodes** receive and index incoming data, just like non-clustered, stand-alone indexers. Unlike stand-alone indexers, however, peer nodes also replicate data from other nodes in the cluster. A peer node can index its own incoming data while simultaneously storing copies of data from other nodes. You must have at least as many peer nodes as the replication factor. That is, to support a replication factor of 3, you need a minimum of three peer nodes.

The **search head** runs searches across the set of peer nodes. You must use a search head to manage searches across indexer clusters.

For most purposes, it is recommended that you use **forwarders** to get data into the cluster.

Here is a diagram of a basic, single-site indexer cluster, containing three peer nodes and supporting a replication factor of 3:



This diagram shows a simple deployment, similar to a small-scale non-clustered deployment, with some forwarders sending load-balanced data to a group of indexers (peer nodes), and the indexers sending search results to a search head. There are two additions that you don't find in a non-clustered deployment:

- The indexers are streaming copies of their data to other indexers.
- The manager node, while it doesn't participate in any data streaming, coordinates a range of activities involving the search peers and the search head.

Multisite indexer clusters

Multisite indexer clusters allow you to maintain complete copies of your indexed data in multiple locations. This offers the advantages of enhanced disaster recovery and **search affinity**. You can specify the number of copies of data on each site. Multisite clusters are similar in most respects to basic, single-site clusters, with some differences in configuration and behavior. See "Multisite indexer clusters".

How to set up a cluster

Clusters are easy to set up. The process is similar to setting up a group of stand-alone indexers. Basically, you install the indexers and perform a bit of configuration.

The main difference is that you also need to identify and enable the cluster nodes. You designate one Splunk Enterprise instance as the manager node and other instances as peer nodes. You need at least as many peer nodes as the size of your replication factor. To increase indexing capacity for horizontal scaling, you just add more peer nodes.

You also need to set up one or more search heads to manage searches across the peers and to consolidate the results for the user.

You enable cluster nodes in the same way that you configure any settings in Splunk Enterprise: through Splunk Web or the CLI, or directly, by editing configuration files.

See the chapter "Deploy the indexer cluster".

How to search a cluster

You search a cluster the same way you search any non-clustered group of indexers. You submit your searches through a search head.

What happens behind the scenes is a bit different, though. Once you have submitted your search, the search head consults the manager node to determine the current set of peer nodes. The search head then distributes the search tasks directly to those peers. The peers do their part and send their results back to the search head, which then consolidates the results and returns them to Splunk Web. From the user's standpoint, it is no different than searching any standalone indexer or non-clustered group of indexers. See "How search works in an indexer cluster."

With a multisite cluster, you can also implement search affinity. In search affinity, a search head gets search results only from indexers local to its site, when possible. At the same time, the search still has access to the full set of data. See "Implement search affinity in a multisite indexer cluster."

Before you go any further

Clusters are easy to set up and use, but you need to have a good grounding in the basics of Splunk Enterprise indexing and deployment first. Before you continue, make sure you know this stuff:

- **How to configure indexers.** See "How the indexer stores indexes", along with the other topics in the current manual that describe managing indexes.
- **What a search head does.** For an introduction to distributed search and search heads, see "About distributed search" in the *Distributed Search* manual.
- **How to use a forwarder to get data into an indexer.** See "Use forwarders" in the *Getting Data In* manual.

Migrating from a non-clustered Splunk Enterprise deployment?

Clustered indexers have several different requirements from non-clustered indexers. It is important that you be aware of these issues before you migrate your indexers. For details, see "Key differences between clustered and non-clustered Splunk Enterprise deployments of indexers". After you read that material, go to "Migrate non-clustered indexers to a clustered environment" for details on the actual migration process.

Multisite indexer clusters

Indexer clusters have built-in site-awareness, meaning that you can explicitly configure a **multisite indexer cluster** on a site-by-site basis. This simplifies and extends the ability to implement a cluster that spans multiple physical sites, such as data centers.

Use cases

Multisite clusters offer two key benefits over single-site clusters:

- **Improved disaster recovery.** By storing copies of your data at multiple locations, you maintain access to the data if a disaster strikes at one location. Multisite clusters provide site failover capability. If a site goes down, indexing and searching can continue on the remaining sites, without interruption or loss of data.

- **Search affinity.** If you configure each site so that it has both a search head and a full set of searchable data, the search head on each site will limit its searches to local peer nodes. This eliminates any need, under normal conditions, for search heads to access data on other sites, greatly reducing network traffic between sites.

Multisite configuration

You configure multisite clusters somewhat differently from basic, single-site clusters. These are the key differences for multisite clusters:

- You assign a site to each node.
- You can specify the replication and search factors on a site-by-site basis. That is, you can specify the number of copies and searchable copies that you want to maintain on each site, along with the number that you want to maintain on the cluster overall.

There are a few other configuration differences as well. See "Multisite deployment overview".

Multisite architecture

The architecture of single-site and multisite clusters is similar. These are the main differences for multisite clusters:

- Each node belongs to an assigned site.
- Replication of bucket copies occurs in a site-aware manner.

- Search heads distribute their searches across local peers only, whenever possible.

For more information on multisite cluster architecture, read "Multisite indexer cluster architecture".

For more information

These chapters and topics describe multisite clusters in detail:

- "Deploy and configure a multisite indexer cluster". This chapter contains topics that describe multisite configuration, including search affinity configuration and multisite replication and search factors.
- "Manage a multisite indexer cluster". This chapter covers issues such as handling manager site failure and converting a multisite cluster to single-site.
- "Migrate an indexer cluster from single-site to multisite". This topic describes how to convert a single-site cluster to multisite.

Other topics in this manual differentiate multisite and single-site clusters as needed.

The basics of indexer cluster architecture

This topic introduces **indexer cluster** architecture. It describes the nodes of a single-site cluster and how they work together. It also covers some essential concepts and describes briefly how clusters handle indexing and searching.

Multisite cluster architecture is similar to single-site cluster architecture. There are, however, a few areas of significant difference. For information on multisite cluster architecture and how it differs from single-site cluster architecture, read the topic Multisite indexer cluster architecture.

For a deeper dive into cluster architecture, read the chapter How indexer clusters work.

For information on how cluster architecture differs for **SmartStore** indexes, see SmartStore architecture overview and Indexer cluster operations and SmartStore.

Cluster nodes

A cluster includes three types of nodes:

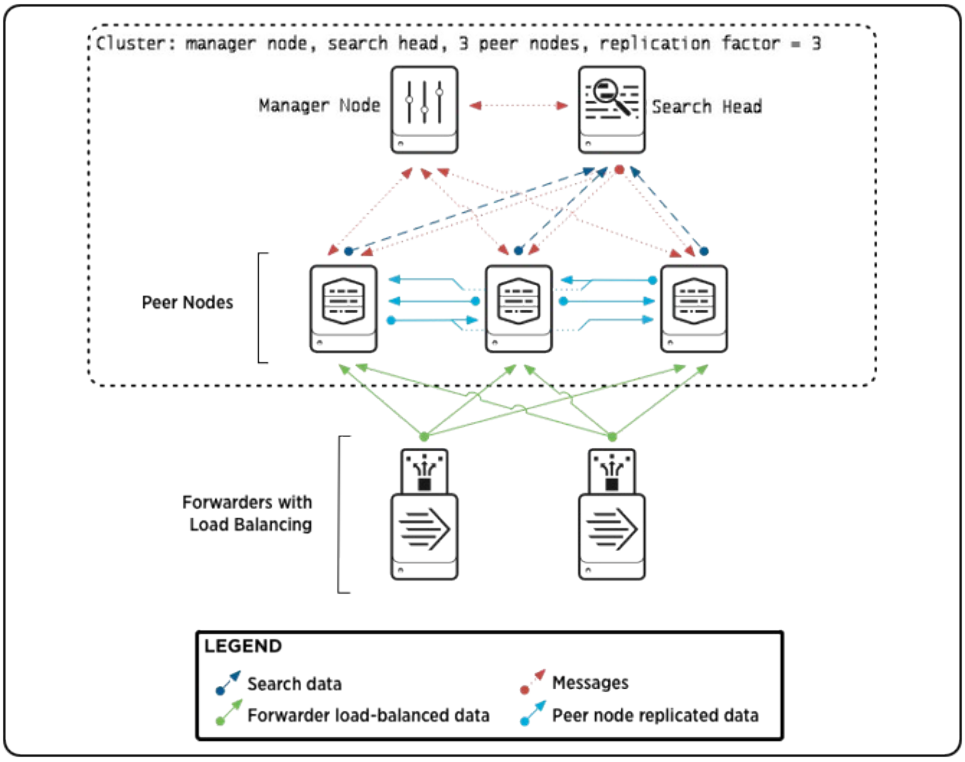
- A single **manager node** to manage the cluster.
- Multiple **peer nodes** to index and replicate data and to search the data.
- One or more **search heads** to coordinate searches across all the peer nodes.

In addition, a cluster deployment usually employs **forwarders** to ingest and forward data to the peers.

Manager nodes, peer nodes, and search heads are all specialized Splunk Enterprise instances. All nodes must reside on separate instances and separate machines. For example, the manager node cannot reside on the same instance or machine as a peer node or a search head.

The manager node and all peer nodes must be specific to a single cluster. A manager node cannot manage multiple clusters. A peer node cannot connect to multiple manager nodes. Search heads, however, can search across multiple clusters.

Here is a diagram of a simple single-site cluster, with a few peers and some forwarders sending data to them:



Some of what is happening in this diagram might not make sense yet; read on.

Manager node

The **manager node** manages the cluster. It coordinates the replicating activities of the peer nodes and tells the search head where to find data. It also helps manage the configuration of peer nodes and orchestrates remedial activities if a peer goes offline.

Unlike the peer nodes, the manager node does not index **external data**. A cluster has exactly one manager node.

Peer node

Peer nodes perform the indexing function for the cluster. They receive and index incoming data. They also send replicated data to other peer nodes in the cluster and receive replicated data from other peers. A peer node can index its own external data while simultaneously receiving and sending replicated data. Like all indexers, peers also search across their indexed data in response to search requests from the search head.

The number of peer nodes you deploy is dependent on two factors: the cluster **replication factor** and the indexing load. For example, if you have a replication factor of 3 (which means you intend to store three copies of your data), you need at least three peers. If you have more indexing load than three indexers can handle, you can add more peers to increase capacity.

Search head

The **search head** manages searches across the set of peer nodes. It distributes search queries to the peers and consolidates the results. You initiate all searches from the search head. A cluster must have at least one search head.

Forwarder

Forwarders function the same as in any Splunk Enterprise deployment. They consume data from external sources and then forward that data to indexers, which, in clusters, are the peer nodes. You are not required to use forwarders to get data into a cluster, but, for most purposes, you will want to. This is because only with forwarders can you enable **indexer acknowledgment**, which ensures that incoming data gets reliably indexed. In addition, to deal with potential peer node failures, it is advisable to use **load-balancing** forwarders. That way, if one peer goes down, the forwarder can switch its forwarding to other peers in the load-balanced group. For more information on forwarders in a clustered environment, read Use forwarders to get data into the indexer cluster in this manual.

Important concepts

To understand how a cluster functions, you need to be familiar with a few concepts:

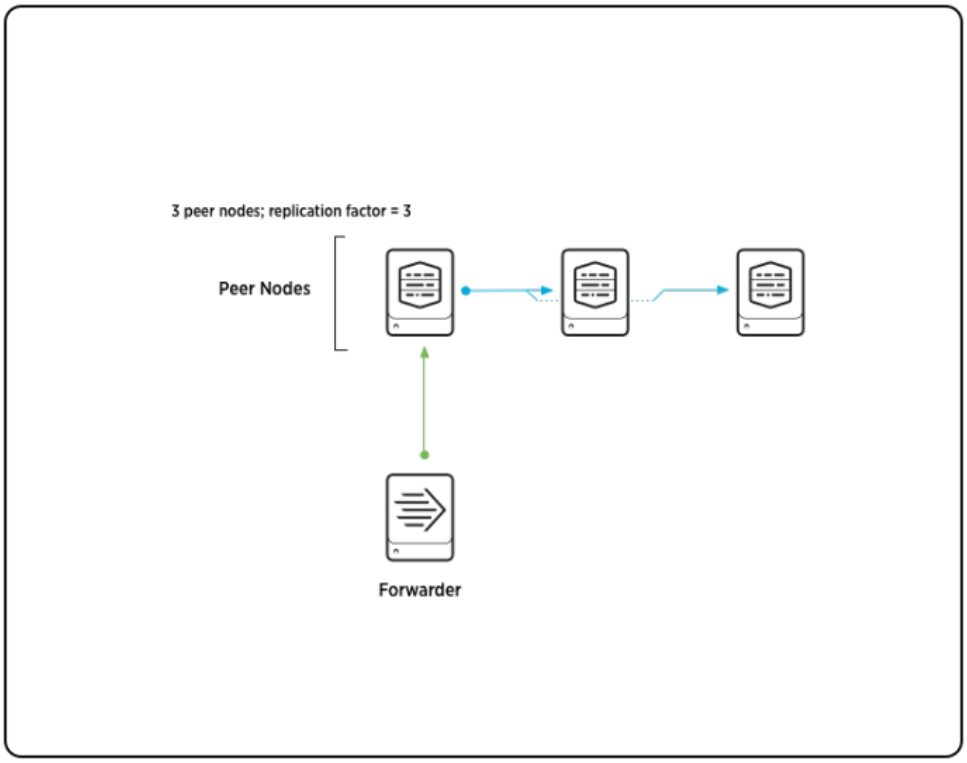
- **Replication factor**. This determines the number of copies of data the cluster maintains and therefore, the cluster's fundamental level of failure tolerance.
- **Search factor**. This determines the number of searchable copies of data the cluster maintains, and therefore how quickly the cluster can recover its searching capability after a peer node goes down.
- **Buckets**. Buckets are the basic units of index storage. A cluster maintains replication factor number of copies of each bucket.

This section provides a brief introduction to these concepts.

Replication factor

As part of configuring the manager node, you specify the number of copies of data that you want the cluster to maintain. The number of copies is called the cluster's **replication factor**. The replication factor is a key concept in index replication, because it determines the cluster's failure tolerance: a cluster can tolerate a failure of (replication factor - 1) peer nodes. For example, if you want to ensure that your system can handle the failure of two peer nodes, you must configure a replication factor of 3, which means that the cluster stores three identical copies of your data on separate nodes. If two peers go down, the data is still available on a third peer. The default value for the replication factor is 3.

Here is a high-level representation of a cluster with three peers and a replication factor of 3:



In this diagram, one peer is receiving data from a forwarder, which it processes and then streams to two other peers. The cluster will contain three complete copies of the peer's data. This diagram represents a very simplified version of peer replication, where all data is coming into the system through a single peer. In most three-peer clusters, all three peers would be receiving external data from a forwarder, as well as replicated data from other peers.

For a detailed discussion of the replication factor and the trade-offs involved in adjusting its value, see the topic Replication factor.

Important: Multisite clusters use a significantly different version of the replication factor. See Multisite replication and search factors.

Search factor

When you configure the manager node, you also designate a **search factor**. The search factor determines the number of immediately **searchable** copies of data the cluster maintains.

Searchable copies of data require more storage space than **non-searchable** copies, so it is best to limit the size of your search factor to fit your exact needs. For most purposes, use the default value of 2. This allows the cluster to continue searches with little interruption if a single peer node goes down.

The difference between a searchable and a non-searchable copy of some data is this: The searchable copy contains both the data itself and some extensive index files that the cluster uses to search the data. The non-searchable copy contains just the data. Even the data stored in the non-searchable copy, however, has undergone initial processing and is stored in a form that makes it possible to recreate the index files later, if necessary.

For a detailed discussion of the search factor and the trade-offs involved in adjusting its value, see the topic Search factor.

Important: Multisite clusters use a significantly different version of the search factor. See Multisite replication and search factors.

Buckets

Splunk Enterprise stores indexed data in **buckets**, which are directories containing files of data. An index typically consists of many buckets.

A **complete** cluster maintains replication factor number of copies of each bucket, with each copy residing on a separate peer node. The bucket copies are either searchable or non-searchable. A complete cluster also has search factor number of searchable copies of each bucket.

Buckets contain two types of files: a **rawdata file**, which contains the data along with some metadata, and - for searchable copies of buckets - **index files** into the data.

The cluster replicates data on a bucket-by-bucket basis. The original bucket and its copies on other peer nodes have identical sets of rawdata. Searchable copies also contain the index files.

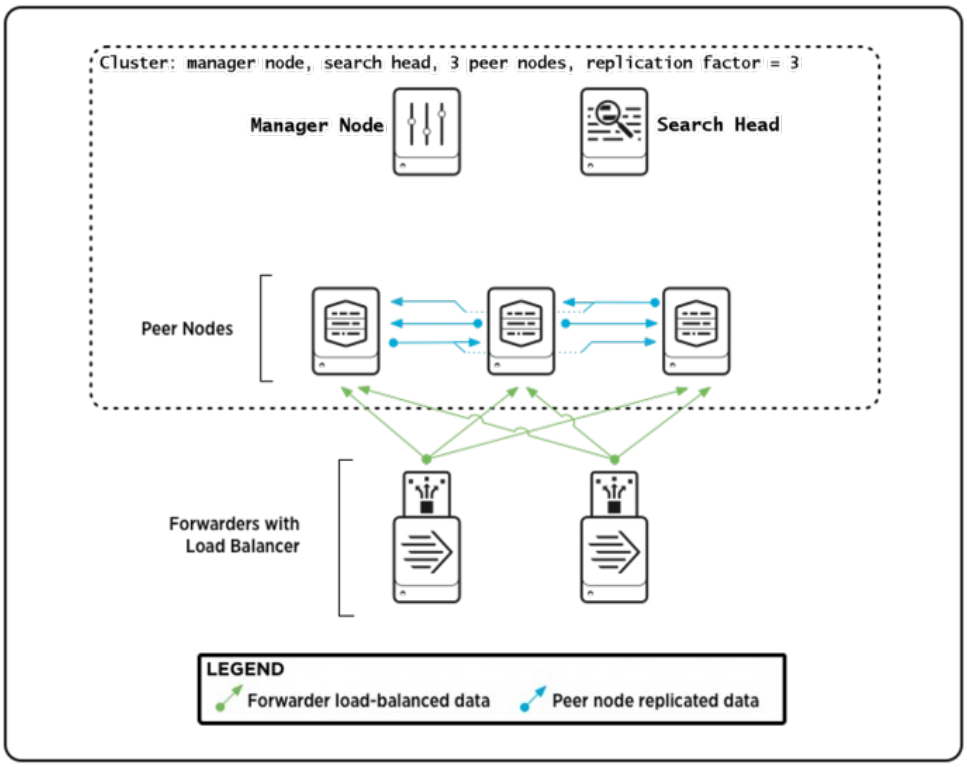
Each time a peer creates a new bucket, it communicates with the manager node to get a list of peers to stream the bucket's data to. If you have a cluster in which the number of peer nodes exceeds the replication factor, a peer might stream data to a different set of peers each time it creates a new bucket. Eventually, the copies of the peer's original buckets are likely to be spread across a large number of peers, even if the replication factor is only 3.

You need a good grasp of buckets to understand cluster architecture. For an overview of buckets in general, read How the indexer stores indexes. Then read the topic Buckets and indexer clusters. It provides detailed information on bucket concepts of particular importance for a clustered deployment.

How indexing works

Clustered indexing functions like non-clustered indexing, except that the cluster stores multiple copies of the data. Each peer node receives, processes, and indexes external data - the same as any non-clustered indexer. The key difference is that the peer node also streams, or "replicates", copies of the processed data to other peers in the cluster, which then store those copies in their own buckets. Some of the peers receiving the processed data might also index it. The replication factor determines the number of peers that receive the copies of data. The search factor determines the number of peers that index the data.

A peer node can be indexing external data while simultaneously storing, and potentially indexing, copies of replicated data sent to it from other peers. For example, if you have a three-peer cluster configured with a replication factor of 3, each peer can be ingesting and indexing external data while also storing copies of replicated data streamed to it by the other peers. If the cluster's search factor is 2, one of the peers receiving a copy of streamed data will also index it. (In addition, the peer that originally ingests the data always indexes its own copy.) This diagram shows the movement of data into peers, both from forwarders and from other peers:



You can set up your cluster so that all the peer nodes ingest external data. This is the most common scenario. You do this simply by configuring inputs on each peer node. However, you can also set up the cluster so that only a subset of the peer nodes ingest data. No matter how you disperse your inputs across the cluster, all the peer nodes can, and likely will, also store replicated data. The manager node determines, on a bucket-by-bucket basis, which peer nodes will get replicated data. You cannot configure this, except in the case of multisite clustering, where you can specify the number of copies of data that each site's set of peers receives.

In addition to replicating indexes of external data, the peers also replicate their internal indexes, such as `_audit`, `_internal`, etc.

The manager node manages the peer-to-peer interactions. Most importantly, it tells each peer what peers to stream its data to. Once the manager node has communicated this, the peers then exchange data with each other, without the manager node's involvement, unless a peer node goes down. The manager node also keeps track of which peers have searchable data and ensures that there are always search factor number of copies of searchable data available. When a peer goes down, the manager node coordinates remedial activities.

For detailed information, read the topic How clustered indexing works.

For information on how indexing works in a multisite cluster, read Multisite indexing.

For information on how indexing works with SmartStore indexes, see How indexing works in SmartStore.

How search works

In an indexer cluster, a search head coordinates all searches. The process is similar to how **distributed searches** work in a non-clustered environment. The main difference is that the search head relies on the manager node to tell it who its

search peers are. Also, there are various processes in place to ensure that a search occurs over one and only one copy of each bucket.

To ensure that exactly one copy of each bucket participates in a search, one searchable copy of each bucket in the cluster is designated as **primary**. Searches occur only across the set of primary copies.

The set of primary copies can change over time, for example, in response to a peer node going down. If some of the bucket copies on the downed node were primary, other searchable copies of those buckets will be made primary to replace them. If there are no other searchable copies (because the cluster has a search factor of 1), non-searchable copies will first have to be made searchable before they can be designated as primary.

The manager node rebalances primaries across the set of peers whenever a peer joins or rejoins the cluster, in an attempt to improve distribution of the search load. See Rebalance the indexer cluster primary buckets.

The manager node keeps track of all bucket copies on all peer nodes, and the peer nodes themselves know the status of their bucket copies. That way, in response to a search request, a peer knows which of its bucket copies to search.

Periodically, the search head gets a list of active search peers from the manager node. To handle searches, it then communicates directly with those peers, as it would for any distributed search, sending search requests and knowledge bundles to the peers and consolidating search results returned from the peers.

For example, assume a cluster of three peers is maintaining 20 buckets that need to be searched to fulfill a particular search request coming from the search head. Primary copies of those 20 buckets could be spread across all three peers, with 10 primaries on the first peer, six on the second, and four on the third. Each peer gets the search request and then determines for itself whether its particular copy of a bucket is primary and therefore needs to participate in the search.

For detailed information, read the topic How search works in an indexer cluster.

Important: There are key differences in how searching works in a multisite cluster. For example, each site in the cluster typically has a complete set of primary buckets, so that a search head can perform its searches entirely on data local to its site. For more information, read Multisite searching.

For information on how search works with SmartStore indexes, see How search works in SmartStore.

How clusters deal with peer node failure

If a peer node goes down, the manager node coordinates attempts to reproduce the peer's buckets on other peers. For example, if a downed node was storing 20 copies of buckets, of which 10 were searchable (including three primary bucket copies), the manager node will direct efforts to create copies of those 20 buckets on other nodes. It will likewise attempt to replace the 10 searchable copies with searchable copies of the same buckets on other nodes. And it will replace the primary copies by changing the status of corresponding searchable copies on other peers from non-primary to primary.

Replication factor and node failure

If there are less peer nodes remaining than the number specified by the replication factor, the cluster will not be able to replace the 20 missing copies. For example, if you have a three-node cluster with a replication factor of 3, the cluster cannot replace the missing copies when a node goes down, because there is no other node where replacement copies can go.

Except in extreme cases, however, the cluster should be able to replace the missing primary bucket copies by designating searchable copies of those buckets on other peers as primary, so that all the data continues to be accessible to the

search head.

The only case in which the cluster cannot maintain a full set of primary copies is if a replication factor number of nodes goes down. For example, if you have a cluster of five peer nodes, with a replication factor of 3, the cluster will still be able to maintain a full set of primary copies if one or two peers go down but not if a third peer goes down.

Search factor and node failure

The search factor determines whether full search capability can quickly resume after a node goes down. To ensure rapid recovery from one downed node, the search factor must be set to at least 2. That allows the manager node to immediately replace primaries on the downed node with existing searchable copies on other nodes.

If instead the search factor is set to 1, that means the cluster is maintaining just a single set of searchable bucket copies. If a peer with some primary copies goes down, the cluster must first convert a corresponding set of non-searchable copies on the remaining peers to searchable before it can designate them as primary to replace the missing primaries. While this time-intensive process is occurring, the cluster has an incomplete set of primary buckets. Searches can continue, but only across the available primary buckets. Eventually, the cluster will replace all the missing primary copies. Searches can then occur across the full set of data.

If, on the other hand, the search factor is at least 2, the cluster can immediately assign primary status to searchable copies on the remaining nodes. The activity to replace the searchable copies from the downed node will still occur, but in the meantime searches can continue uninterrupted across all the cluster's data.

For detailed information on peer failure, read the topic What happens when a peer node goes down.

For information on how a multisite cluster handles peer node failure, read How multisite indexer clusters deal with peer node failure.

How clusters deal with manager node failure

If a manager node goes down, peer nodes can continue to index and replicate data, and the search head can continue to search across the data, for some period of time. Problems eventually will arise, however, particularly if one of the peers goes down. There is no way to recover from peer loss without the manager node, and the search head will then be searching across an incomplete set of data. Generally speaking, the cluster continues as best it can without the manager node, but the system is in an inconsistent state and results cannot be guaranteed.

For detailed information on manager node failure, read the topic What happens when a manager node goes down.

Multisite indexer cluster architecture

This topic describes the architecture of **multisite indexer clusters**. It focuses primarily on how multisite clusters differ from single-site clusters. For an overview of cluster architecture, focusing on single-site clusters, read The basics of indexer cluster architecture.

How multisite and single-site architecture differ

Multisite clusters differ from single-site clusters in these key respects:

- Each node (manager/peer/search head) has an assigned site.
- Replication of bucket copies occurs with site-awareness.

- Search heads distribute their searches across local peers only, when possible.
- Bucket-fixing activities respect site boundaries when applicable.

Multisite cluster nodes

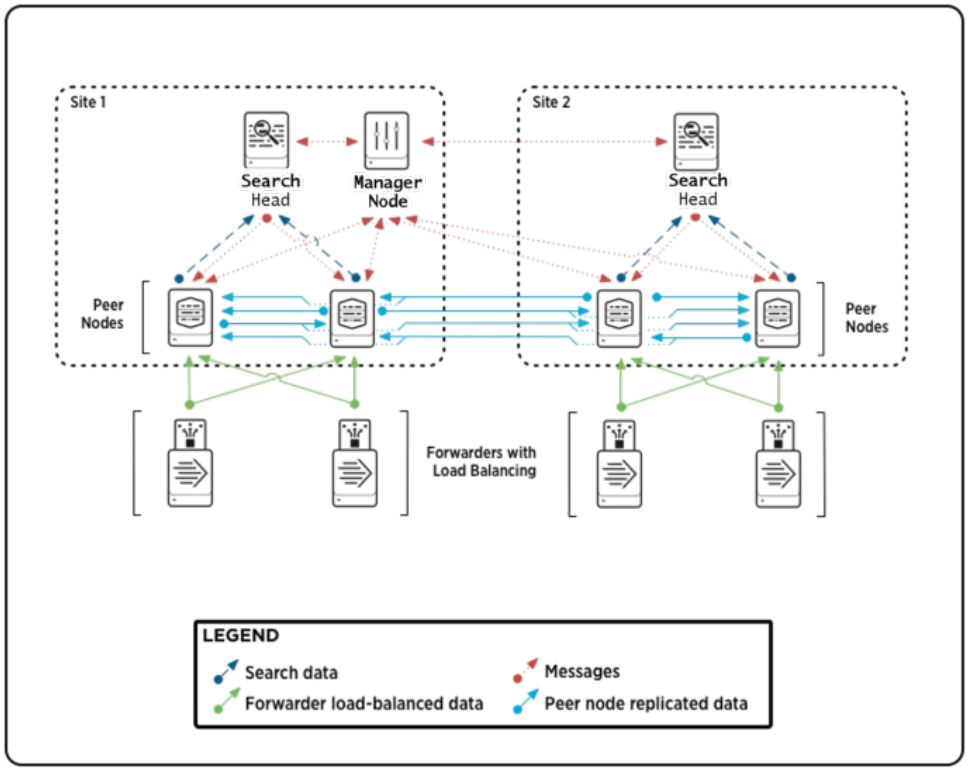
Multisite and single-site nodes share these characteristics:

- Clusters have three types of nodes: manager, peers, and search heads.
- Each cluster has exactly one manager node.
- The cluster can have multiple peer nodes and search heads.

Multisite nodes differ in these ways:

- Every node belongs to a specific site. Physical location typically determines a site. That is, if you want your cluster to span servers in Boston and Philadelphia, you assign all nodes in Boston to site1 and all nodes in Philadelphia to site2.
- A typical multisite cluster has search heads on each site. This is necessary for **search affinity**, which increases search efficiency by allowing a search head to access all data locally.

Here is an example of a two-site cluster.



Note the following:

- The manager node controls the entire cluster. Although the manager node resides physically on a site, the manager is not actually a member of any site. However, each manager node has a built-in search head, and that search head requires that you specify a site for the manager node as a whole. Note that the manager's search head is for testing purposes only. Do not use it in a production environment.
- This is an example of a cluster that has been configured for search affinity. Each site has its own search head, which searches the set of peer nodes on its site. Depending on circumstances, however, a search head might also search peers outside its own site. See Multisite searching and search affinity.
- The peers replicate data across site boundaries. This behavior is fundamental for both disaster recovery and search affinity.

Multisite replication and search factors

As with their single-site counterparts, multisite replication and search factors determine the number of bucket copies and searchable bucket copies, respectively, in the cluster. The difference is that the multisite replication and search factors also determine the number of copies on each site. A multisite replication factor for a three-site cluster might look like this:

```
site_replication_factor = origin:2, site1:1, site2:1, site3:1, total:4
```

This replication factor specifies that each site will get one copy of each bucket, except when the site is the originating site for the data, in which case it will get two copies. It also specifies that the total number of copies across the cluster is four.

In this particular example, the replication factor explicitly specifies all sites, but that is not a requirement. An **explicit site** is a site that the replication factor explicitly specifies. A **non-explicit site** is a site that the replication factor does not explicitly specify.

Here is another example of a multisite replication factor for a three-site cluster. This replication factor specifies only two of the sites explicitly:

```
site_replication_factor = origin:2, site1:1, site2:2, total:5
```

In this example, the number of copies that the non-explicit site3 gets varies. If site1 is the origin, site1 gets two copies, site2 gets two copies, and site3 gets the remainder of one copy. If site2 is the origin, site1 gets one copy, site2 gets two copies, and site3 gets two copies. If site3 is the origin, site1 gets one copy, site2 gets two copies, and site3 gets two copies.

Note: In this example, the `total` value cannot be 4. It must be at least 5. This is because, when the replication factor has non-explicit sites, the total must be at least the sum of all explicit site and origin values.

For details on replication factor syntax and behavior, read "Configure the site replication factor".

The site search factor works the same way. For details, read "Configure the site search factor".

Multisite indexing

Multisite indexing is similar to single-site indexing, described in "The basics of indexer cluster architecture". A single manager node coordinates replication across all the peers on all the sites.

This section briefly describes the main multisite differences, using the example of a three-site cluster with this replication factor:

```
site_replication_factor = origin:2, site1:1, site2:1, site3:1, total:4
```
These are the main multisite issues to be aware of:

- Data replication occurs across site boundaries, based on the replication factor. If, in the example, a peer in site1 ingests the data, it will stream one copy of the data to another peer in site1 (to fulfill the `origin` setting of 2), one copy to a peer in site2, and one copy to a peer in site3.
- Multisite replication has the concept of the origin site, which allows the cluster to handle data differently for the site that it originates on. The example illustrates this. If site1 originates the data, it gets two copies. If another site originates the data, site1 gets only one copy.
- As with single-site replication, you cannot specify the exact peers that will receive the replicated data. However, you can specify the sites whose peers will receive the data.

For information on how the cluster handles migrated single-site buckets, see "Migrate an indexer cluster from single-site to multisite".

Multisite searching and search affinity

Multisite searching is similar in most ways to single-site searching, described in "The basics of indexer cluster architecture". Each search occurs across a set of primary bucket copies. There is one key difference, however.

Multisite clusters provide **search affinity**, which allows searches to occur on site-local data. You must configure the cluster to take advantage of search affinity. Specifically, you must ensure that both the searchable data and the search heads are available locally.

To accomplish this, you configure the search factor so that each site has at least one full set of searchable data. The manager node then ensures that each site has a complete set of primary bucket copies, as long as the sites are functioning properly. This is known as the **valid** state.

With search affinity, the search heads still distribute their search requests to all peers in the cluster, but only the peers on the same site as the search head respond to the request by searching their primary bucket copies and returning results to the search head.

If the loss of some of a site's peers means that it no longer has a full set of primaries (and thus is no longer in the valid state), bucket fix-up activities will attempt to return the site to the valid state. During the fix-up period, peers on remote sites will participate in searches, as necessary, to ensure that the search head still gets a full set of results. After the site regains its valid state, the search head once again uses only local peers to fulfill its searches.

Note: You can disable search affinity if desired. When disabled for a particular search head, that search head can get its data from peers on any sites.

For more information on search affinity and how to configure the search factor to support it, see "Implement search affinity in a multisite indexer cluster". For more information on the internals of search processing, including search affinity, see "How search works in an indexer cluster".

Multisite clusters and node failure

The way that multisite clusters deal with node failure has some significant differences from single-site clusters.

Before reading this section, you must understand the concept of a "reserve" bucket copy. A reserve bucket copy is a virtual copy, awaiting eventual assignment to a peer. Such a copy does not actually exist yet in storage while it is in the reserve state. Once the manager node assigns it to an available peer, the bucket gets streamed to that peer in the usual manner. As a consequence of multisite peer node failure, some bucket copies might not be immediately assigned to a peer. For example, in a cluster with a total replication factor of 5, the manager might tell the originating peer to stream buckets to just three other peers. This results in four copies (the original plus the three streamed copies), with the fifth copy awaiting assignment to a peer once certain conditions are met. The fifth, unassigned copy is known as a reserve copy. This section describes how the cluster must sometimes reserve copies when peer nodes fail.

How multisite clusters deal with peer node failure

When a peer goes down, bucket fix-up happens within the same site, if possible. The cluster tries to replace any missing bucket copies by adding copies to peers remaining on that site. (In all cases, each peer can have at most one copy of any particular bucket.) If it is not possible to fix up all buckets by adding copies to peers within the site, then, depending on the replication and search factors, the cluster might make copies on peers on other sites.

The fix-up behavior under these circumstances depends partially on whether the failed peer was on an explicit or non-explicit site.

If enough peers go down on an explicit site such that the site can no longer meet its site-specific replication factor, the cluster does not attempt to compensate by making copies on peers on other sites. Rather, it assumes that the requisite number of peers will eventually return to the site. For new buckets also, it holds copies in reserve for the return of peers to the site. In other words, it does not assign those copies to peers on a different site, but instead waits until the first site has peers available and then assigns the copies to those peers.

For example, given a three-site cluster (site1, site2, site3) with this replication factor:

```
site_replication_factor = origin:2, site1:1, site2:2, total:5
```

the cluster ordinarily maintains two copies on site2. But if enough peers go down on site2, so that only one remains and the site can no longer meet its replication factor of 2, that remaining peer gets one copy of all buckets in the cluster, and the cluster reserves another set of copies for the site. When a second peer rejoins site2, the cluster streams the reserved copies to that peer.

When a non-explicit site loses enough peers such that it can no longer maintain the number of bucket copies that it already has, the cluster compensates by adding copies to other sites to make up the difference. For example, assume that the non-explicit site3 in the example above has two copies of some bucket, and that it then loses all but one of its peers, so that it can only hold one copy of each bucket. The cluster compensates by streaming a copy of that bucket to a peer on one of the other sites, assuming there is at least one peer that does not yet have a copy of that bucket.

For information on how a cluster handles the case where all the peer nodes on a site fail, see See "How the cluster handles site failure".

How the cluster handles site failure

Site failure is just a special case of peer node failure. Cluster fix-up occurs following the rules described earlier for peer node failure. Of particular note, the cluster might hold copies in reserve against the eventual return of the site.

For any copies of existing buckets that are held in reserve, the cluster does not add copies to other sites during its fix-up activities. Similarly, for any new buckets added after the site goes down, the cluster reserves some number of copies until the site returns to the cluster.

Here is how the cluster determines the number of copies to reserve:

- For explicit sites, the cluster reserves the number of copies and searchable copies specified in the site's search and replication factors.
- For non-explicit sites, the cluster reserves one searchable copy if the `total` components of the site's search and replication factors are sufficiently large, after handling any explicit sites, to accommodate the copy. (If the search factor isn't sufficiently large but the replication factor is, the cluster reserves one non-searchable copy.)

For example, say you have a three-site cluster with two explicit sites (site1 and site2) and one non-explicit site (site3), with this configuration:

```
site_replication_factor = origin:2, site1:1, site2:2, total:5
site_search_factor = origin:1, site1:1, site2:1, total:2
```

In the case of a site going down, the cluster reserves bucket copies like this:

- If site1 goes down, the cluster reserves one searchable copy.
- If site2 goes down, the cluster reserves two copies, including one that is searchable.
- If site3 goes down, the cluster reserves one non-searchable copy.

Once the reserved copies are accounted for, the cluster replicates any remaining copies to other available sites, both during fix-up of existing buckets and when adding new buckets.

When the site returns to the cluster, bucket fix-up occurs for that site to the degree necessary to ensure that the site has, at a minimum, its allocation of reserved bucket copies, both for new buckets and for buckets that were on the site when it went down.

If the site that fails is the site on which the manager node resides, you can bring up a stand-by manager node on one of the remaining sites. See "Handle indexer cluster manager site failure".

How multisite clusters deal with manager node failure

A multisite cluster handles manager node failure the same as a single-site cluster. The cluster continues to function as best it can under the circumstances. See "What happens when a manager node goes down".

Deploy the indexer cluster

Indexer cluster deployment overview

This topic describes the main steps to deploying **indexer clusters**. Subsequent topics describe these steps in detail.

Before you attempt to deploy a cluster, you must be familiar with several areas of Splunk Enterprise administration:

- **How to configure indexers.** In particular, see "How the indexer stores indexes", along with the other topics in this manual that describe managing indexes.
- **What a search head does.** For an introduction to distributed search, see "About distributed search" in the *Distributed Search* manual. Note, however, that you configure search heads for indexer clusters somewhat differently from other search heads. For information on the differences see "Search head configuration overview" in this manual.
- **How to use a forwarder to get data into an indexer.** See "Use forwarders" in the *Getting Data In* manual.

Important: This chapter assumes that you are deploying independent search heads in the indexer cluster. For information on how to incorporate search heads that are members of a **search head cluster**, see "Integrate the search head cluster with an indexer cluster" in the *Distributed Search* manual.

Migrating from a non-clustered Splunk Enterprise deployment?

Clustered indexers (peers) have several different requirements from non-clustered indexers. It is important that you be aware of these issues before you migrate your indexers. See "Key differences between clustered and non-clustered deployments of indexers". After you read that material, go to "Migrate non-clustered indexers to a clustered environment" for the actual migration process.

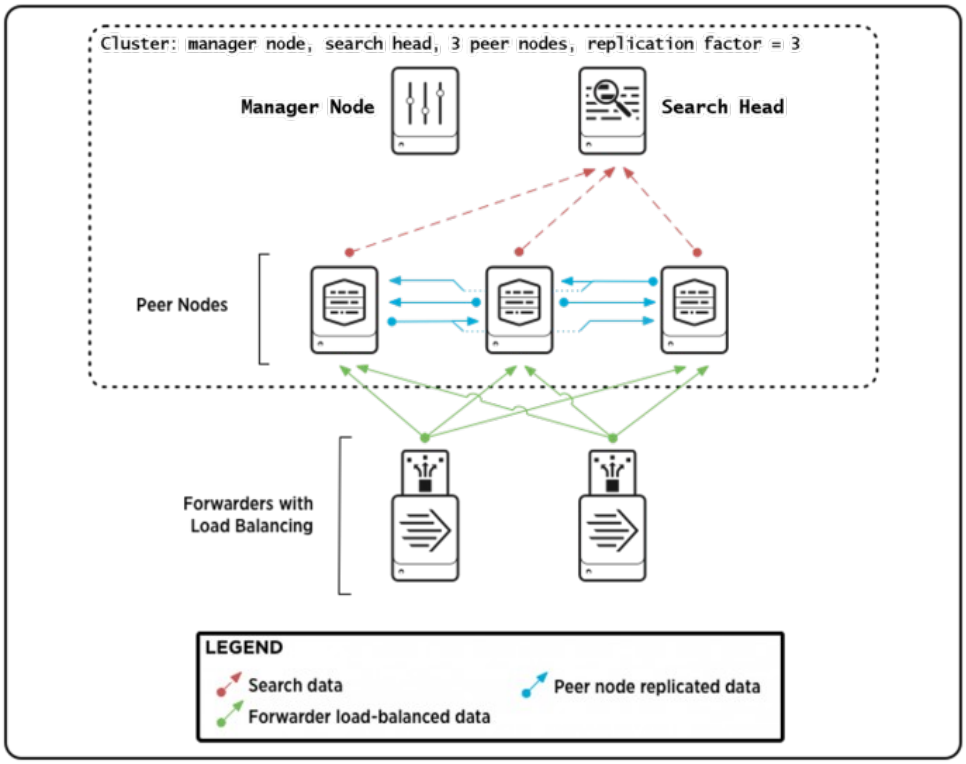
Important: Before migrating an indexer from non-clustered to clustered, be certain of your needs. The process goes in one direction only. There is no supported procedure for converting an indexer from clustered to non-clustered.

Deploying a multisite cluster?

Multisite indexer clusters are considerably more complex than single-site clusters. Deploying them requires that you consider some additional issues and perform an entirely different set of configurations. If you are deploying a multisite cluster, read this topic first and then read "Multisite indexer cluster deployment overview".

Deploy a cluster

When you deploy a cluster, you enable and configure the cluster manager node and the peer nodes that perform the indexing. You also enable a search head to search data in the cluster. In addition, you usually set up forwarders to send data to the cluster. Here is a diagram of a small cluster, showing the various nodes that you deploy:



These are the key steps in deploying clusters:

1. Identify your requirements:

a. Understand your data availability and failover needs. See "About indexer clusters".

b. Determine whether you will be deploying a basic, single-site cluster or a multisite cluster. Multisite clusters offer strong disaster recovery capabilities because they allow you to distribute copies of your data across multiple locations. They also enable search affinity, which reduces network traffic by limiting searches to local data. For more information, read "Multisite indexer clusters".

c. Decide what **replication factor** you want to implement. The replication factor is the number of copies of raw data that the cluster maintains. Your optimal replication factor depends on factors specific to your environment, but essentially involves a trade-off between failure tolerance and storage capacity. A higher replication factor means that more copies of the data will reside on more peer nodes, so your cluster can tolerate more node failures without loss of data availability. But it also means that you will need more nodes and more storage to handle the additional data. For multisite clusters, you also need to decide how many copies to put on each site. For more information, see "Replication factor".

Warning: Make sure you start by choosing the right replication factor for your needs. It is inadvisable to increase the replication factor after the cluster contains a significant amount of data. The cluster would need to perform a large amount of bucket copying to match the increased replication factor, slowing significantly the overall performance of your cluster while the copying is occurring.

d. Decide what **search factor** you want to implement. The search factor tells the cluster how many searchable copies of indexed data to maintain. This helps determine the speed with which a cluster can recover from a downed node. A higher search factor allows the cluster to recover more quickly, but it also requires more storage space and processing power.

For most single-site deployments, the default search factor value of 2 represents the right trade-off, allowing searches usually to continue with little interruption when a node goes down. For multisite clusters, you also need to decide how many searchable copies to put on each site. For more information, see "Search factor".

Warning: Make sure you start by choosing the right search factor for your needs. It is inadvisable to increase the search factor after the cluster contains a significant amount of data. The cluster would need to perform a large amount of processing (transforming non-searchable bucket copies into searchable copies) to match the increased search factor, and this will have an adverse effect on the overall performance of your cluster while the processing is occurring.

e. Identify other factors that also determine the size of your cluster; for example, the quantity of data you will be indexing. It usually makes sense to keep all your indexers in a single cluster, so for horizontal scaling, you will need to add peer nodes beyond those required by the replication factor. Similarly, depending on the anticipated search load, you might need to add more than one search head.

f. Study the topic "System requirements and other deployment considerations for indexer clusters" for information on other key issues.

2. Install the Splunk Enterprise cluster instances on your network. At a minimum, you will need (replication factor + 2) instances:

- You need at least the replication factor number of **peer nodes**, but you might want to add more peers to boost indexing capacity, as mentioned in step 1e.
- You also need two more instances, one for the **manager node** and the other for the **search head**.

For multisite clusters, you must also take into account the search head and peer node requirements of each site, as determined by your search affinity and disaster recovery needs. See "Multisite indexer cluster deployment overview".

For information on how to install Splunk Enterprise, read the *Installation Manual*.

3. Enable clustering on the instances:

a. Enable the manager node. See "Enable the manager node".

b. Enable the peer nodes. See "Enable the peer nodes".

c. Enable the search head. See "Enable the search head".

Important: For multisite clusters, the process of enabling cluster nodes is different. See "Multisite indexer cluster deployment overview".

4. Complete the peer node configuration:

a. Configure the peers' index settings. This step is necessary only if you need to augment the set of default indexes and apps. In general, all the peers must use the same set of indexes, so if you add indexes (or apps that define indexes) to one peer, you must add them to all peers, using a cluster-specific distribution method. There might also be other configurations that you need to coordinate across the set of peers. See "Prepare the peers for index replication" for information on how to do this.

b. Configure the peers' data inputs. For most purposes, it is best to use forwarders to send data to the peers, as discussed in "Ways to get data into an indexer cluster". As that topic states, you will usually want to deploy load-balancing forwarders with indexer acknowledgment enabled.

After you enable the nodes and set up data inputs for the peers, the cluster automatically begins indexing and replicating the data.

5. Configure the manager node to forward its data to the peer nodes. This best practice provides several advantages. See "Best practice: Forward manager node data to the indexer layer".

Other deployment scenarios

This manual also provides guidance on a few other cluster deployment scenarios:

- Add indexers with existing data to a cluster. See "Migrate non-clustered indexers to a clustered environment".
- Deploy **SmartStore** indexes on a new indexer cluster. See "Deploy SmartStore on a new indexer cluster".
- Migrate data currently on an indexer cluster to SmartStore. See "Migrate existing data on an indexer cluster to SmartStore".
- Bootstrap SmartStore indexes onto an indexer cluster. See "Bootstrap SmartStore indexes".
- Migrate a single-site cluster to multisite. See "Migrate an indexer cluster from single-site to multisite".
- Employ clusters purely for index scalability, where index replication is not a requirement. See "Use indexer clusters to scale indexing".

Key differences between clustered and non-clustered deployments of indexers

This topic describes key differences between clustered and non-clustered indexers. In particular, it discusses issues regarding system requirements and deployment.

Read this topic carefully if you plan to migrate your current set of indexers to a cluster.

Do not use deployment server or third-party deployment tools with cluster peers

Neither the deployment server nor any third party deployment tool (such as Puppet or CFEngine, among others) is supported as a means to distribute configurations or apps to cluster peers (indexers).

To distribute configurations across the set of cluster peers, instead use the **configuration bundle** method outlined in the topic "Update common peer configurations". As that topic explains, the configuration bundle method involves first placing peer apps on the manager node, which then distributes those apps to the peer nodes in a coordinated fashion.

For information on how to migrate app distribution from the deployment server to the configuration bundle method, see "Migrate apps to a cluster".

Note: You can use deployment server to distribute updates to search heads in indexer clusters, as long as they are standalone search heads. You cannot use the deployment server to distribute updates to members of a **search head cluster**.

Differences in system requirements

Peer nodes have some different system requirements compared to non-clustered indexers. Before migrating your indexer, read the topic "System requirements and other deployment considerations for indexer clusters". In particular, be aware of the following differences:

- When you convert an indexer to a cluster peer, disk usage will go up significantly. Make sure that you have sufficient disk space available, relative to daily indexing volume, search factor, and replication factor. For detailed

information on peer disk usage, read "Storage considerations".
- Cluster nodes cannot share Splunk Enterprise instances. The manager node, peer nodes, and search head must each run on its own instance.

Other considerations and differences from a non-cluster deployment

In addition, note the following:

- For most types of cluster deployments, you should enable **indexer acknowledgment** for the forwarders sending data to the peer. See "How indexer acknowledgment works."
- You can simplify the process of connecting forwarders to peer nodes by using the **indexer discovery** feature. See "Advantages of the indexer discovery method."
- There will be some overall reduction in performance due to a few factors. The primary one is indexer acknowledgment. In addition, the need to store, and potentially index, replicated data coming from other peer nodes has some effect on performance.
- When restarting cluster peers, you should use Splunk Web or one of the cluster-aware CLI commands, such as `splunk offline` or `splunk rolling-restart`. Do not use `splunk restart`. For details, see "Restart the entire indexer cluster or a single cluster node."

Migrate a non-clustered indexer

To learn how to migrate an existing indexer to a cluster and the ramifications of doing so, read the topic "Migrate non-clustered indexers to a clustered environment".

System requirements and other deployment considerations for indexer clusters

Indexer clusters are groups of Splunk Enterprise indexers, so, for the most part, you just need to adhere to the system requirements for indexers. For detailed software and hardware requirements for indexers, read "System requirements" in the *Installation Manual*. The current topic notes additional requirements for clusters.

Note: System requirements vary somewhat for clusters with **SmartStore** indexes. See SmartStore system requirements.

Summary of key requirements

These are the main issues to note:

- Each cluster node (manager, peer, or search head) must reside on a separate Splunk Enterprise instance.
- Each node instance must run on a separate machine or virtual machine.
- Each machine must be running the same operating system, including version.
- All nodes must be connected over a network.
- There are strict version compatibility requirements between cluster nodes.

For example, to deploy a cluster consisting of three peer nodes, one manager node, and one search head, you need five Splunk Enterprise instances running on five machines connected over a network. And all machines must be running the same operating system and version.

These are some additional issues to be aware of:

- Compared to a non-clustered deployment, clusters require more storage, to accommodate the multiple copies of data.
- Index replication, in and of itself, does not increase your licensing needs.
- You cannot use a deployment server to distribute updates to peers.

See the remainder of this topic for details.

Required Splunk Enterprise instances

Each cluster node must reside on its own Splunk Enterprise instance. Therefore, the cluster must consist of at least (replication factor + 2) instances: a minimum of **replication factor** number of peer nodes, plus one manager node and one or more search heads. For example, if you want to deploy a cluster with a replication factor of 3, you must set up at least five instances: three peers, one manager node, and one search head. To learn more about the replication factor, see "Replication factor" in this manual.

The size of your cluster depends on other factors besides the replication factor, such as the amount of data you need to index. See "Indexer cluster deployment overview".

Although the manager node has search capabilities, you should only use those capabilities for debugging purposes. The resources of the manager node must be dedicated to fulfilling its critical role of coordinating cluster activities. Under no circumstances should the manager node be employed as a production search head. See "Additional roles for the manager node".

Splunk Enterprise version compatibility

Interoperability between the various types of cluster nodes is subject to strict compatibility requirements. In brief:

- The manager node must run the same or a higher version from the peer nodes and search heads.
- The search heads must run the same or a higher version from the peer nodes.
- The peer nodes must all run exactly the same version, down to the maintenance level.

Compatibility between the manager and the peer nodes and search heads

Peer nodes and search heads can run different versions from the manager, subject to these restrictions:

- The manager node must run the same or a higher version from the peer nodes and search heads.
- The manager node can run at most three minor version levels higher from the peer nodes.
- All nodes must run a supported version of Splunk Enterprise.

Compatibility between peer nodes

All peer nodes must run the same version of Splunk Enterprise, down to the maintenance level. You must update all peer nodes to a new release at the same time.

Compatibility between peer nodes and search heads

The peer nodes and search heads can run different versions from each other. The search heads must run the same or a higher version from the peer nodes.

Search head clusters participating in an indexer cluster have the same compatibility requirements as individual search heads. For information on other search head cluster version requirements, see "System requirements and other

deployment considerations for search head clusters" in the *Distributed Search* manual.

Machine requirements

Each node of the cluster (manager node, peer nodes, and search heads) must run on its own, separate machine or virtual machine. Other than that, the hardware requirements, aside from storage, are basically the same as for any Splunk Enterprise instance. See "Reference hardware" in the *Capacity Planning Manual*.

Note the following:

- Peer nodes have specific storage requirements, discussed elsewhere in this topic.
- The storage needs of the manager node are significantly lower than those of peer nodes, since the manager node does not index external data.
- For peer nodes, the best practice is to use homogeneous machines with identical hardware specifications, to ensure full utilization of processing capacity across the indexing tier.

Operating system requirements

Indexer clustering is available on all operating systems supported for Splunk Enterprise. For a list of supported operating systems, see System requirements in the *Installation Manual.*

All indexer cluster nodes (manager node, peer nodes, and search heads) must run on the same operating system and version.

If the indexer cluster is integrated with a search head cluster, then the search head cluster instances, including the deployer, must run on the same operating system and version as the indexer cluster nodes.

Synchronization of system clocks across the cluster

It is important that you synchronize the system clocks on all machines, virtual or physical, that are running Splunk Enterprise instances participating in the cluster. Specifically, this means your manager node, peer nodes, and search heads. Otherwise, various issues can arise, such as timing problems between the manager and peer nodes, search failures, or premature expiration of search artifacts.

The synchronization method you use depends on your specific set of machines. Consult the system documentation for the particular machines and operating systems on which you are running Splunk Enterprise. For most environments, Network Time Protocol (NTP) is the best approach.

Storage considerations

When determining storage requirements for your clustered indexes, you need to consider the increased capacity, across the set of peer nodes, necessary to handle the multiple copies of data.

It is strongly recommended that you provision all peer nodes to use the same amount of disk storage.

Clusters use the usual settings for managing index storage, as described in "Configure index storage".

Determine your storage requirements

It is important to ensure you have enough disk space to accommodate the volume of data your peer nodes will be processing. For a general discussion of Splunk Enterprise data volume and how to estimate your storage needs, refer to

"Estimating your storage requirements" in the *Capacity Planning Manual*. That topic provides information on how to estimate storage for non-clustered indexers, so you need to supplement its guidelines to account for the extra copies of data that a cluster stores.

With a cluster, in addition to considering the volume of incoming data, you must consider the replication factor and **search factor** to arrive at your total storage requirements across the set of peer nodes. With a replication factor of 3, you are storing three copies of your data. You will need extra storage space to accommodate these copies, but you will not need three times as much storage. Replicated copies of non-searchable data are smaller than copies of searchable data, because they include only the data and not the associated index files. So, for example, if your replication factor is 3 and your search factor is 2, you will need more than two, but less than three, times the storage capacity compared to storing the same data on non-clustered indexers.

Exactly how much less storage your non-searchable copies require takes some investigation on your part. The index files excluded by non-searchable copies can vary greatly in size, depending on factors described in "Estimating your storage requirements" in the *Capacity Planning Manual*.

Important: A manager node is not aware of the amount of storage on individual peer nodes, and therefore it does not take available storage into account when it makes decisions about which peer node should receive a particular set of replicated data. It also makes arbitrary decisions about which peer should make some set of replicated data searchable (in cases where the search factor is 2 or greater). Therefore, you must ensure that each peer node has sufficient storage not only for the data originating on that peer, but also for any replicated copies of data that might get streamed to it from other peers. You should continue to monitor storage usage throughout the life of the cluster.

Storage requirement examples

As a ballpark figure, incoming syslog data, after it has been compressed and indexed, occupies approximately 50% of its original size:

- 15% for the rawdata file.
- 35% for associated index files.

In practice, this estimate can vary substantially, based on the factors described in "Estimating your storage requirements" in the *Capacity Planning Manual*.

Assume you have 100GB of syslog data coming into Splunk Enterprise. In the case of a non-clustered indexer, that data would occupy approximately 50GB (50% of 100GB) of storage on the indexer. However, in the case of clusters, storage calculations must factor in the replication factor and search factor to arrive at total storage requirements across all the cluster peers. (As mentioned earlier, you cannot easily predict exactly how much storage will be required on any specific peer.)

Here are two examples of estimating cluster storage requirements, both assuming 100GB of incoming syslog data, resulting in 15GB for each set of rawdata and 35GB for each set of index files:

- **3 peer nodes, with replication factor = 3; search factor = 2:** This requires a total of 115GB across all peer nodes (averaging 38GB/peer), calculated as follows:
  - Total rawdata = (15GB * 3) = 45GB.
  - Total index files = (35GB * 2) = 70GB.
- **5 peer nodes, with replication factor = 5; search factor = 3:** This requires a total of 180GB across all peer nodes (averaging 36GB/peer), calculated as follows:
  - Total rawdata = (15GB * 5) = 75GB.
  - Total index files = (35GB * 3) = 105GB.

Storage hardware

In pre-6.0 versions of Splunk Enterprise, replicated copies of cluster buckets always resided in the `colddb` directory, even if they were hot or warm buckets. Starting with 6.0, hot and warm replicated copies reside in the `db` directory, the same as for non-replicated copies. This eliminates any need to consider faster storage for `colddb` for clustered indexes, compared to non-clustered indexes.

Licensing information

As with any Splunk Enterprise deployment, your licensing requirements are driven by the volume of data your indexers process. Contact your Splunk sales representative to purchase additional license volume. Refer to "How licensing works" in the *Admin Manual* for more information about Splunk Enterprise licensing.

There are just a few license issues that are specific to index replication:

- All cluster nodes, including manager nodes, peer nodes, and search heads, need to be in an Enterprise **license pool**, even if they're not expected to index any data.
- Cluster nodes must share the same licensing configuration.
- Only incoming data counts against the license; replicated data does not.
- You cannot use index replication with a free license.

Ports that the cluster nodes use

These ports must be available to cluster nodes:

- **On the manager node:**
  - The management port (by default, 8089) must be available to all other cluster nodes.
- **On each peer node:**
  - The management port must be available to all other cluster nodes.
  - The replication port must be available to all other peer nodes.
  - The receiving port must be available to all forwarders sending data to that peer.
- **On each search head:**
  - The management port must be available to all other nodes.
  - The http port (by default, 8000) must be available to any browsers accessing data from the search head.

Deployment server and clusters

Do not use deployment server with cluster peers.

The deployment server is not supported as a means to distribute configurations or apps to cluster peers. To distribute configurations across the set of cluster peers, instead use the **configuration bundle** method outlined in the topic "Update common peer configurations".

For information on how to migrate app distribution from deployment server to the configuration bundle method, see "Migrate apps to a cluster".

Additional roles for the manager node

As a general rule, you should dedicate the Splunk Enterprise instance running the manager node to that single purpose. Constrain use of the manager's built-in search head to debugging only.

Under limited circumstances, however, you might be able to colocate one or more of these lightweight functions on the manager instance:

- **License manager**
- **Monitoring console**
- **Search head cluster deployer**

To use the manager instance for any of these additional roles, the manager's cluster must remain below the following limits:

- 30 indexers
- 100,000 buckets
- 10 indexes
- 10 search heads

Do not colocate a deployment server on the manager node under any circumstances. A manager node and a deployment server both consume significant system resources while performing their tasks. The manager node needs reliable and continuous access to resources to perform the ongoing management of the cluster, and the deployment server can easily overwhelm those resources while deploying updates to its deployment clients.

For a general discussion of management component colocation, see Components that help to manage your deployment in the *Distributed Deployment Manual*.

Enable the indexer cluster manager node

Before reading this topic, read Indexer cluster deployment overview.

A cluster has one, and only one, **manager node**. The manager node coordinates the activities of the peer nodes. It does not itself store or replicate data (aside from its own internal data).

Important: A manager node cannot do double duty as a peer node or a search node. The Splunk Enterprise instance that you enable as manager node must perform only that single indexer cluster role. In addition, the manager cannot share a machine with a peer. Under certain limited circumstances, however, the manager instance can handle a few other lightweight functions. See "Additional roles for the manager node".

You must enable the manager node as the first step in deploying a cluster, before setting up the peer nodes.

The procedure in this topic explains how to use Splunk Web to enable a manager node. You can also enable a manager node in two other ways:

- Directly edit the manager's `server.conf` file. See "Configure the manager with server.conf" for details. Some advanced settings can only be configured by editing this file.
- Use the CLI `edit cluster-config` command. See "Configure the manager with the CLI" for details.

Important: This topic explains how to enable a manager node for a single-site cluster only. If you plan to deploy a multisite cluster, see "Configure multisite indexer clusters with server.conf".

Enable the manager node

To enable an indexer as the manager node:

1. Click **Settings** in the upper right corner of Splunk Web.

2. In the **Distributed environment** group, click **Indexer clustering**.

3. Select **Enable indexer clustering**.

4. Select **Manager node** and click **Next**.

5. There are a few fields to fill out:

- **Replication Factor.**The **replication factor** determines how many copies of data the cluster maintains. The default is 3. For more information on the replication factor, see Replication factor. Be sure to choose the right replication factor now. It is inadvisable to increase the replication factor later, after the cluster contains significant amounts of data.
- **Search Factor.** The **search factor** determines how many immediately searchable copies of data the cluster maintains. The default is 2. For more information on the search factor, see Search factor. Be sure to choose the right search factor now. It is inadvisable to increase the search factor later, once the cluster has significant amounts of data.
- **Security Key**. This is the key that authenticates communication between the manager node and the peers and search heads. The key must be the same across all cluster nodes. The value that you set here must be the same that you subsequently set on the peers and search heads as well.
- **Cluster Label**. You can label the cluster here. The label is useful for identifying the cluster in the monitoring console. See Set cluster labels in *Monitoring Splunk Enterprise*.

6. Click **Enable manager node**.

The message appears, "You must restart Splunk for the manager node to become active. You can restart Splunk from Server Controls."

7. Click **Go to Server Controls**. This takes you to the Settings page where you can initiate the restart.

Important: When the manager node starts up for the first time, it will block indexing on the peers until you enable and restart the full replication factor number of peers. Do not restart the manager node while it is waiting for the peers to join the cluster. If you do, you will need to restart the peers a second time.

View the manager node dashboard

After the restart, log back into the manager node and return to the Clustering page in Splunk Web. This time, you see the manager node clustering dashboard. For information on the dashboard, see "View the manager node dashboard".

Perform additional configuration

For information on post-deployment manager node configuration, see "Manager node configuration overview".

Enable the peer nodes

Before reading this topic, read "Indexer cluster deployment overview".

You ordinarily need to enable multiple **peer nodes** to deploy a cluster. At a minimum, you must enable at least **replication factor** number of peers - and potentially more to accommodate horizontal scaling.

Before enabling the set of peers, you must enable and restart the **manager node**, as described in "Enable the manager node". When the manager node starts up for the first time, it will block indexing on the peers until you have enabled and restarted the replication factor number of peers.

The procedure in this topic explains how to use Splunk Web to enable a peer node. You can also enable a peer in two other ways:

- Directly edit the peer's `server.conf` file. See "Configure peer nodes with server.conf" for details.
- Use the CLI `edit cluster-config` command. See "Configure peer nodes with the CLI" for details.

Important: This topic explains how to enable a peer for a single-site cluster only. If you plan to deploy a multisite cluster, see "Configure multisite indexer clusters with server.conf".

Enable the peer

To enable an indexer as a peer node:

1. Click **Settings** in the upper right corner of Splunk Web.

2. In the **Distributed environment** group, click **Indexer clustering**.

3. Select **Enable indexer clustering**.

4. Select **Peer node** and click **Next**.

5. There are a few fields to fill out:

- **Manager URI**. Enter the manager node's URI, including its management port. For example: `https://10.152.31.202:8089`.
- **Peer replication port**. This is the port on which the peer receives replicated data streamed from the other peers. You can specify any available, unused port for this purpose. This port must be different from the management or receiving ports.
- **Security key**. This is the key that authenticates communication between the manager node and the peers and search heads. The key must be the same across all cluster nodes. Set the same value here that you previously set on the manager node.

6. Click **Enable peer node**.

The message appears, "You must restart Splunk for the peer node to become active."

7. Click **Go to Server Controls**. This takes you to the Settings page where you can initiate the restart.

8. Repeat this process for all the cluster's peer nodes.

When you have enabled the replication factor number of peers, the cluster can start indexing and replicating data, as described in "Enable the manager node".

View the peer dashboard

After the restart, log back into the peer node and return to the Clustering page in Splunk Web. This time, you see the peer's clustering dashboard. For information on the dashboard, see "View the peer dashboard".

Configure the peers

After enabling the peers, you need to perform additional configuration before you start indexing data. For details, read these topics:

- "Configure the peer indexes in an indexer cluster"
- "Get data into the indexer cluster"

You might also need to configure some other settings on the peers. See "Peer node configuration overview".

Enable the search head

Before reading this topic, read "Indexer cluster deployment overview".

To search the cluster, you need to enable at least one **search head** in the indexer cluster.

Before enabling the search head, you must enable and restart the manager node, as described in "Enable the manager node".

The procedure in this topic explains how to use Splunk Web to enable a search head. You can also enable a search head in two other ways:

- Directly edit the search head's `server.conf` file. See "Configure the search head with server.conf" for details. Some advanced settings, including multi-cluster search, can only be configured by editing this file.
- Use the CLI `edit cluster-config` command. See "Configure the search head with the CLI" for details.

Important: This topic explains how to enable an individual search head for a single-site cluster only:

- If you plan to deploy a multisite cluster, see "Configure multisite indexer clusters with server.conf".
- If you plan to incorporate search heads that are members of a **search head cluster**, see "Integrate the search head cluster with an indexer cluster" in the *Distributed Search* manual.

Enable the search head

To enable a Splunk instance as a search head in an indexer cluster:

1. Click **Settings** in the upper right corner of Splunk Web.

2. In the **Distributed environment** group, click **Indexer clustering**.

3. Select **Enable clustering**.

4. Select **Search head node** and click **Next**.

5. There are a few fields to fill out:

- **Manager URI**. Enter the manager node's URI, including its management port. For example: `https://10.152.31.202:8089`.
- **Security key**. This is the key that authenticates communication between the manager node and the peers and search heads. The key must be the same across all cluster nodes. Set the same value here that you previously set on the manager node.

6. Click **Enable search head node**.

The message appears, "You must restart Splunk for the search node to become active. You can restart Splunk from Server Controls."

7. Click **Go to Server Controls**. This takes you to the Settings page where you can initiate the restart.

Next steps

Now that you have enabled the search head, you can:

- View the search head dashboard
- Allow the search head to search other clusters
- Add search heads to the cluster
- Perform additional configuration on the search head

View the search head dashboard

After the restart, log back into the search head and return to the Clustering page in Splunk Web. This time, you see the search head's clustering dashboard. See "View the search head dashboard" for more information.

Allow the search head to search multiple clusters

From the dashboard, you can add additional clusters for the search head to search. For details, see "Search across multiple indexer clusters".

Add search heads to an indexer cluster

You can set up multiple search heads to accommodate more simultaneous searches. For information on how to determine your search head needs, see "Hardware capacity planning" in the *Capacity Planning* manual.

If you want to set up more search heads for an indexer cluster, just repeat the enablement procedure for additional instances. If you want to deploy a search head cluster, so that the search heads share configurations and jobs, see the additional configuration instructions in the topic "Integrate the search head cluster with an indexer cluster" in the *Distributed Search* manual.

Perform additional configuration

For more information on configuration of search heads in an indexer cluster, see "Search head configuration overview".

Best practice: Forward manager node data to the indexer layer

It is considered a best practice to forward all manager node internal data to the indexer (peer node) layer. This has several advantages:

- It accumulates all data in one place. This simplifies the process of managing your data: You only need to manage your indexes and data at one level, the indexer level.
- It enables diagnostics for the manager node if it goes down. The data leading up to the failure is accumulated on the indexers, where a search head can later access it.

The preferred approach is to forward the data directly to the indexers, without indexing separately on the manager node. You do this by configuring the manager node as a forwarder. These are the main steps:

1. Make sure that all necessary indexes exist on the indexers. This is normally the case, unless you have created custom indexes on the manager node. Since `_audit` and `_internal` exist on indexers as well as the manager, you do not need to create separate versions of those indexes to hold the corresponding manager data.

2. Configure the manager node as a forwarder. Create an `outputs.conf` file on the manager node that configures it for load-balanced forwarding across the set of peer nodes. You must also turn off indexing on the manager node, so that the manager does not both retain the data locally as well as forward it to the peers.

Here is an example `outputs.conf` file:

```
# Turn off indexing on the manager node
[indexAndForward]
index = false

[tcpout]
defaultGroup = my_peers_nodes
forwardedindex.filter.disable = true
indexAndForward = false

[tcpout:my_peers_nodes]
server=10.10.10.1:9997,10.10.10.2:9997,10.10.10.3:9997
```

This example assumes that each peer node's receiving port is set to 9997.

For details on configuring `outputs.conf`, read "Configure forwarders with outputs.conf" in the Forwarding Data manual.

Prepare the peers for index replication

After you enable the peers, you might need to perform further configuration to prepare them for index replication.

If you use only the default set of indexes and default configurations, you can start replicating data right away. However, if you need to install apps or change configurations on the peers, you usually must apply the set of changes to all peers. In particular, if you need to add indexes (including indexes defined by an app), you must do so in a way that ensures that the peers use a common set of indexes.

For information on how to configure indexes across cluster peers, read "Configure the peer indexes in an indexer cluster".

For information on how to configure apps across peers, as well as other peer configuration issues, read "Peer node configuration overview".

Use indexer clusters to scale indexing

The main purpose of indexer clusters is to enable **index replication**. However, clusters can also be generally useful in scale-out deployment topologies as a way to manage multiple indexers, even when index replication isn't a requirement.

For example, say you want to create a deployment of three indexers and one search head, so that you can index larger quantities of data than a single indexer is capable of. The non-clustered way of doing this is to set up each of the indexers independently, add in a search head, and then use a tool like **deployment server** to coordinate the indexer configurations.

With clustering, you can instead configure this deployment scenario as a cluster, with three peer nodes replacing the three independent indexers. Even if you don't need index replication and its key advantages like data availability and disaster tolerance, there are several reasons why it might be beneficial to use a cluster to coordinate multiple indexer instances:

- Simplified management and coordination of indexer configuration (in place of using deployment server or performing manual updates). See "Update common peer configurations" for details.
- Simplified set up and control of distributed search. See "Enable the search head".
- Better insight into the state of your indexers through the clustering dashboards. See "View the manager node dashboard".
- Ability to take advantage of additional cluster management capabilities as they're developed.

The main downsides of employing a cluster for scaling indexing capacity are these:

- You must install an additional Splunk Enterprise instance to function as the cluster manager node.
- The cluster does not support heterogeneous indexers. All cluster nodes must be at the same version level. In addition, all peer nodes in a cluster must use the same `indexes.conf` configuration. For further details, see the next section, "Cluster peer management compared to deployment server".
- You cannot use the deployment server to distribute configurations or apps across the cluster peers. For further details, see the next section, "Cluster peer management compared to deployment server".

Cluster peer management compared to deployment server

One useful cluster feature is the ability to manage and update the configuration for all indexers (peer nodes) from a central location, the manager node. In that respect, it's similar in function to the **deployment server**. Unlike the deployment server, however, cluster peer management does not have any concept of server classes. Because of this, and because of the way clusters coordinate their activities, you cannot specify different app or `indexes.conf` configurations for different groups of indexers. (All peer nodes in a cluster must use the same `indexes.conf` configurations, as well as some other configurations, as described in "Peer node configuration overview".) If you need to maintain a heterogeneous set of indexers, you cannot employ clusters for scaling purposes.

On the other hand, the configuration bundle method used to download updates to peer nodes has certain advantages over the deployment server. Specifically, it not only distributes updates, it also validates them on the peers, and then (when necessary) initiates a rolling restart of the peers. See "Update common peer configurations" for details.

Important: Do not use deployment server or third-party distributed configuration management software, such as Puppet or Chef, to deploy updates directly to peer nodes. You can use such tools to deploy updates to the manager node, which then deploys those updates to the peers. See "Use deployment server to distribute the apps to the manager node".

Configure a cluster for scale-out deployment

To set up a cluster for scale-out deployment, without index replication, just set both the **replication factor** and **search factor** to 1. This causes the cluster to function purely as a coordinated set of Splunk Enterprise instances, without data replication. The cluster will not make any duplicate copies of the data, so you can keep storage size and processing overhead to a minimum.

Migrate non-clustered indexers to a clustered environment

Read the topic "Key differences between clustered and non-clustered deployments of indexers" carefully if you plan to migrate your current set of indexers to an indexer cluster. That topic describes issues that you must understand before initiating the migration process.

You can add a non-clustered indexer to a cluster (as a peer node) at any time. To do so, just enable the indexer as a peer, as described in "Enable the peer nodes".

Once the indexer has been made a peer, it participates in the cluster the same as any other peer. Any new data coming into the peer gets replicated according to the cluster's replication factor, and the peer is also a candidate for receiving replicated data from other peers. Data already on the indexer does not get automatically replicated, but it does participate in searches, as described below.

Important: Before migrating an indexer from non-clustered to clustered, be certain of your needs. The process goes in one direction only. There is no supported procedure for converting an indexer from clustered to non-clustered.

Manage legacy data

The term "legacy data" means data stored on an indexer prior to its conversion to a cluster peer.

How the cluster handles legacy indexed data

When you add an existing indexer to the cluster, the cluster does not replicate any buckets that are already on the indexer.

Buckets already on the indexer prior to its being added to the cluster are called "standalone" buckets. Searches will still occur across those buckets and will be combined with search results from the cluster's replicated buckets.

Is there any way to migrate my legacy data?

Because of the high processing cost of converting standalone buckets to replicated buckets (due to the need to make multiple searchable and non-searchable copies of those buckets to fulfill the cluster's replication and search factors), it is generally a bad idea to attempt to do so, particularly in the case of indexers with large numbers of standalone buckets. There is no supported procedure for this conversion. If, however, you have a strong need for such a conversion, contact Splunk Professional Services to discuss the trade-offs and requirements for this operation.

Migrate apps to a cluster

In your current, non-clustered environment, you might be using **deployment server** to distribute apps to your set of indexers. You will no longer be able to do so once you convert the indexers to cluster peer nodes. See "Key differences between clustered and non-clustered deployments of indexers" for details.

To distribute apps to peer nodes, you must instead use the **configuration bundle** method described in "Update common peer configurations and apps". You place the apps in a special location on the manager node, and the manager pushes the apps to each indexer when you first enable it as a peer. If you need to add or change apps later on, you make the changes and additions on the manager node and then tell the manager to push the updated configuration bundle to the entire set of peers.

For information on how the configuration bundle method compares with the deployment server, see "Cluster peer management compared to deployment server".

Important: You *must* use the configuration bundle method to distribute apps to peer nodes; the deployment server is unsupported for that purpose.

How to migrate apps

It's recommended that you migrate apps at the time that you enable the cluster. The procedure described below describes how to do that.

Important: You must be familiar with the preceding topics in the current chapter before attempting this procedure. Start with "Indexer cluster deployment overview" and read onwards until you return to this topic.

This procedure assumes that you are starting with a distributed search environment with a set of indexers configured as deployment clients of a deployment server. The deployment server is being used to push apps to the indexers. It is these indexers that you will be converting to cluster peer nodes. Once they've become peer nodes, you can no longer use the deployment server to push apps to them; instead, you must use the configuration bundle method.

To migrate your apps at the time of cluster enablement, follow these steps:

1. Enable the manager node, as described in "Enable the manager node".

2. On the deployment server, locate the set of apps that it pushes to the indexers that you're planning to migrate. These should be under the deployment server's `$SPLUNK_HOME/etc/deployment-apps` directory.

3. Copy these apps from the deployment server to the cluster manager's `$SPLUNK_HOME/etc/manager-apps` directory. See "Structure of the configuration bundle" for information on the `manager-apps` directory.

4. Examine each app in `manager-apps` for any `indexes.conf` file(s). In those files, locate any stanzas that define new indexes. In each of those stanzas, add this attribute/value pair:

```
repFactor=auto
```

This enables replication for the index. See "The indexes.conf repFactor attribute" for more information.

Note: If you are the creator and maintainer of the app, you can also make this change in `$SPLUNK_HOME/etc/manager-apps/<appname>/default/indexes.conf`.

5. Convert each indexer to a cluster peer, one at a time:

a. Reconfigure the indexer so that it's no longer a deployment client.

b. Delete the deployed apps from the indexer's `$SPLUNK_HOME/etc/apps` directory. Leave the default apps, such as Search, in place.

c. Enable the indexer as a cluster peer, as described in "Enable the peer nodes".

d. Restart the peer to complete the enablement process.

e. Verify that the manager node has pushed the expected set of apps to the peer's `$SPLUNK_HOME/etc/peer-apps` directory.

6. Once all peers have been enabled, go to the manager node dashboard and verify that the expected set of indexes is being replicated. This dashboard is described in "View the manager node dashboard".

For more information on configuring cluster peers, see these topics:

- "Configure the peer indexes in an indexer cluster"
- "Peer node configuration overview"
- "Update common peer configurations and apps"

Upgrade an indexer cluster

The upgrade process differs considerably depending on the nature of the upgrade. This topic covers these version-based scenarios:

- Upgrading from 7.1 or higher
- Upgrading from 6.x to 7.1
- Upgrading to a new maintenance release (for example, from 8.1.1 to 8.1.2)

In addition, the topic describes:

- How to upgrade an indexer cluster that integrates with a search head cluster.
- How to perform a site-by-site upgrade of a multisite indexer cluster.

Migrating from single-site to multisite?

To convert a single-site indexer cluster to multisite, perform the upgrade first and then read Migrate an indexer cluster from single-site to multisite.

Upgrading an indexer cluster that integrates with a search head cluster?

Upgrade the search head cluster as part of the procedure for performing an indexer cluster rolling upgrade. See Perform a rolling upgrade of a Splunk Enterprise indexer cluster.

If you don't want to perform an indexer cluster rolling upgrade, you can instead follow the steps for tiered upgrades. See Upgrade each tier separately.

Upgrading an indexer cluster that does not have a custom security key?

The security key, also known as the `pass4SymmKey` setting, authenticates communication between the manager node and the peers and search heads.

Starting in 6.6, a non-default security key is required. If the cluster's security key was never explicitly set to a custom value, a warning message appears on the manager node:

```
pass4SymmKey setting in the clustering or general stanza of server.conf is set to empty or the default
value. You must change it to a different value.
```

To remediate this situation, you must set the security key on all the cluster nodes (manager node, peer nodes, search heads) while the cluster is down. The key must be the same across all cluster nodes.

You set the security key with the `pass4SymmKey` attribute in `server.conf`. See Configure the security key.

To set the key during cluster upgrade, you must upgrade all cluster tiers at once, following the procedure in Upgrade all tiers at once. Set the security key while all nodes are down, so that they all have the same security key when they start up.

Use a rolling upgrade for a 7.1 or higher indexer cluster

When upgrading a 7.1 or higher indexer cluster, the preferred method is to perform a rolling upgrade. The rolling upgrade method lets you upgrade peer nodes to the new version with minimal interruption of ongoing searches. See Perform a rolling upgrade of a Splunk Enterprise indexer cluster.

Upgrade all peer nodes in a single operation

Using this method, you take down all peer nodes at the same time during the upgrade operation. You can upgrade all tiers of the cluster (manager node, search heads, peer nodes) at once or you can upgrade each tier separately.

This method is necessary when upgrading a 6.x cluster. However, it can also be used for the upgrade of a cluster of any version. It has the disadvantage of requiring downtime across the entire peer node tier, affecting indexing and searching during that time,

Note: If you have a multisite cluster, you can, in most cases, instead upgrade one site at a time. See Site-by-site upgrade for multisite indexer clusters.

The approach of upgrading each tier separately is particularly useful if the search head tier consists of a search head cluster, because it eliminates the need to upgrade the search head cluster simultaneously with the indexer cluster peer nodes.

Caution: Even when upgrading each tier separately, it is strongly recommended that you complete the entire upgrade process quickly, to avoid any possibility of incompatibilities between node types running different versions.

To upgrade all cluster tiers at once, see Upgrade all tiers at once.

To upgrade each tier separately, see Upgrade each tier separately.

Upgrade all tiers at once

Perform the following steps:

1. Stop the manager node.
2. Stop all the peers and search heads.
   When bringing down the peers, use the `splunk stop` command, not `splunk offline`.
3. If the cluster does not use a non-default (custom) security key, you must set one now. Starting in 6.6, indexer clusters require a non-default security key. This key must be the same across all nodes in the cluster. See Upgrading an indexer cluster that does not have a custom security key?. On each node (manager, peers, and

search heads), set the key using the procedure in Configure the security key.
4. Upgrade the manager node, following the normal procedure for any Splunk Enterprise upgrade, as described in How to upgrade Splunk Enterprise in the *Installation Manual*. **Do not upgrade the peers yet.**
5. Start the manager node, accepting all prompts, if it is not already running.
6. Run `splunk enable maintenance-mode` on the manager node. To confirm that the manager node is in maintenance mode, run `splunk show maintenance-mode`. This step prevents unnecessary bucket fix-ups. See Use maintenance mode.
7. Upgrade the search heads, followed by the peer nodes. Use the normal procedure for any Splunk Enterprise upgrade, as described in How to upgrade Splunk Enterprise in the *Installation Manual*.
8. Start the peer nodes and search heads, if they are not already running.
9. Run `splunk disable maintenance-mode` on the manager node. To confirm that the manager node is not in maintenance mode, run `splunk show maintenance-mode`.

You can view the manager node dashboard to verify that all cluster nodes are up and running.

Upgrade each tier separately

When upgrading tiers separately:

- You must upgrade the tiers in the prescribed order.
- Within each tier, you must upgrade all nodes as a single operation.

Functionality introduced in the new release will not be available until all tiers complete the upgrade.

Caution: Even when upgrading each tier separately, it is strongly recommended that you complete the entire upgrade process quickly, to avoid any possibility of incompatibilities between node types running different versions.

You must follow this order of upgrade when upgrading the tiers in discrete operations:

1. Upgrade the manager node.
2. Upgrade the search head tier.
3. Upgrade the peer node tier.

1. Upgrade the manager node

1. Stop the manager node.
2. Upgrade the manager node, following the normal procedure for any Splunk Enterprise upgrade, as described in How to upgrade Splunk Enterprise in the *Installation Manual*.
3. Start the manager node, accepting all prompts, if it is not already running.

You can view the manager node dashboard to verify that all cluster nodes are up and running.

2. Upgrade the search head tier

The method that you use to upgrade the search head tier depends on whether or not the tier consists of a search head cluster:

- If the search head tier consists of a search head cluster, follow the procedure in Upgrade a search head cluster. If desired, you can perform a rolling upgrade of the search head cluster, as described in that topic.

- If the search head tier consists of independent search heads, follow this procedure:

1. Stop all the search heads.
2. Upgrade the search heads, following the normal procedure for any Splunk Enterprise upgrade, as described in How to upgrade Splunk Enterprise in the *Installation Manual*.
3. Start the search heads, if they are not already running.

You can view the manager node dashboard to verify that all cluster nodes are up and running.

3. Upgrade the peer node tier

1. Run `splunk enable maintenance-mode` on the manager node.
   To confirm that the manager node is in maintenance mode, run `splunk show maintenance-mode` on the manager node.
   This step prevents unnecessary bucket fix-ups. See Use maintenance mode.
2. Stop all the peer nodes.
   When bringing down the peers, use the `splunk stop` command, not `splunk offline`.
3. Upgrade the peer nodes, following the normal procedure for any Splunk Enterprise upgrade, as described in How to upgrade Splunk Enterprise in the *Installation Manual*.
4. Start the peer nodes, if they are not already running.
5. Run `splunk disable maintenance-mode` on the manager node.
   To confirm that the manager node is not in maintenance mode, run `splunk show maintenance-mode` on the manager node.

You can view the manager node dashboard to verify that all cluster nodes are up and running.

Site-by-site upgrade for multisite indexer clusters

Caution: You cannot use this method to upgrade a cluster with metrics data from 7.3.x or earlier to 8.0.x or later, due to a change introduced in 8.0 in how metrics data is stored. Instead, you must either use the rolling upgrade method described in Perform a rolling upgrade of a Splunk Enterprise indexer cluster or take down and upgrade all peer nodes as a single operation, as described in the methods covered by Upgrade all peer nodes in a single operation. As you will note when studying each of these methods, they all put the cluster into maintenance mode for the duration of the peer upgrades. This step is essential, because it ensures that no replication of metrics data takes place between a 7.3.x peer and an 8.0.x peer.

If you have a multisite cluster, you can upgrade one site at a time, as long as you are upgrading across no more than one sequential *n.n* version (for example, from 6.5 to 6.6, or 6.6 to 7.0, but not 6.5 to 7.0). Because each site has a full set of primary copies, this method allows searches to continue uninterrupted during the upgrade.

You cannot perform a site-by-site upgrade if you are upgrading across more than one sequential *n.n* version (for example, from 6.4 to 6.6 or 6.5 to 7.0). To upgrade across multiple sequential *n.n* versions, you must either use the rolling upgrade method described in Perform a rolling upgrade of a Splunk Enterprise indexer cluster or take down and upgrade all peer nodes as a single operation, as described in the methods covered by Upgrade all peer nodes in a single operation.

Alternatively, to upgrade across multiple sequential *n.n* versions, you can upgrade via interim releases of not more than a single *n.n* version. For example, if you are upgrading from 6.4 to 6.6, you can first upgrade to a 6.5 interim release using the site-by-site method. You can then upgrade to 6.6.

Functionality introduced in the new release will not be available until all nodes complete the upgrade.

For a two-site cluster, the upgrade procedure has three distinct phases:

1. Upgrade of the manager node.

2. Upgrade of the site1 peers and search heads.

3. Upgrade of the site2 peers and search heads.

Here are the steps in detail:

1. Stop the manager node.

2. Upgrade the manager node, following the normal procedure for any Splunk Enterprise upgrade, as described in How to upgrade Splunk Enterprise in the *Installation Manual*.

3. Start the manager node, accepting all prompts, if it is not already running.

4. Run `splunk enable maintenance-mode` on the manager node. To confirm that the manager is in maintenance mode, run `splunk show maintenance-mode`. This step prevents unnecessary bucket fix-ups. See Use maintenance mode.

5. Stop all the peers and search heads on site1 with the `splunk stop` command.

6. Upgrade the site1 peer nodes and search heads.

7. Start the site1 peer nodes and search heads, if they are not already running.

8. Run `splunk disable maintenance-mode` on the manager node. To confirm that the manager is not in maintenance mode, run `splunk show maintenance-mode`.

9. Wait until the manager node dashboard shows that both the search factor and replication factor are met.

10. Run `splunk enable maintenance-mode` on the manager node. To confirm that the manager is in maintenance mode, run `splunk show maintenance-mode`.

11. Stop all the peers and search heads on site2 with the `splunk stop` command.

12. Upgrade the site2 peer nodes and search heads.

13. Start the site2 peer nodes and search heads, if they are not already running.

14. Run `splunk disable maintenance-mode` on the manager node. To confirm that the manager is not in maintenance mode, run `splunk show maintenance-mode`.

You can view the manager node dashboard to verify that all cluster nodes are up and running.

Upgrade to a maintenance release

To upgrade a cluster to a maintenance release (for example, from 8.1.1 to 8.1.2), you do not need to bring down the entire cluster at once. Instead, you can perform a rolling, online upgrade, in which you upgrade the nodes one at a time.

Caution: Even with a rolling upgrade, you should upgrade all nodes quickly, for several reasons:

- Proper functioning of the cluster depends on all peer nodes running the same version of Splunk Enterprise, as

stated in System requirements and other deployment considerations for indexer clusters.
- Other version compatibility requirements must also be met, as described in System requirements and other deployment considerations for indexer clusters.
- If you upgrade the manager node but not the peers, the cluster might generate errors and warnings. This is generally okay for a short duration, but you should complete the upgrade of all nodes as quickly as possible.

To upgrade a cluster node, follow the normal procedure for any Splunk Enterprise upgrade, with the few exceptions described below. For general information on upgrading Splunk Enterprise instances, see How to upgrade Splunk Enterprise.

To perform a rolling maintenance upgrade, follow these steps:

1. Upgrade the manager node

Upgrade the manager node first.

For information on what happens when the manager node goes down, as well as what happens when it comes back up, see What happens when the manager node goes down.

2. Upgrade the search heads

The only impact to the cluster when you upgrade the search heads is disruption to searches during that time.

3. Put the manager node into maintenance mode

Run `splunk enable maintenance-mode` on the manager node. To confirm that the manager is in maintenance mode, run `splunk show maintenance-mode`. This step prevents unnecessary bucket fix-ups. See Use maintenance mode.

4. Upgrade the peer nodes

When upgrading peer nodes, note the following:

- Peer upgrades can disrupt ongoing searches.
- To minimize downtime and to limit any disruption to indexing and searching, upgrade the peer nodes one at a time.
- To bring a peer down prior to upgrade, use the `splunk offline` command, as described in Take a peer offline.
- During the interim between when you upgrade the manager node and when you finish upgrading the peers, the cluster might generate various warnings and errors.
- For multisite clusters, the site order of peer upgrades does not matter.

5. Take the manager node out of maintenance mode

Run `splunk disable maintenance-mode` on the manager node. To confirm that the manager is not in maintenance mode, run `splunk show maintenance-mode`.

Upgrade a cluster that uses cluster manager redundancy

When upgrading a cluster with cluster manager redundancy, the best practice to ensure continued high availability of a manager throughout the process is to upgrade the standby managers first, followed by the active manager:

1. Put all cluster managers into maintenance mode by running `splunk enable maintenance-mode` on each manager. To confirm that each manager is in maintenance mode, run `splunk show maintenance-mode` on each manager.
2. For each standby manager:
    1. Stop the manager.
    2. Upgrade the manager, following the normal procedure for any Splunk Enterprise upgrade, as described in How to upgrade Splunk Enterprise in the *Installation Manual*.
    3. Start the manager.
    4. Check that the manager enters standby mode and resyncs with the active manager by running `splunk cluster-manager-redundancy -show-status`.
3. Upgrade the active manager:
    1. Switch one of the standby managers to active mode by running `splunk cluster-manager-redundancy -switch-mode active` on the designated new active manager. Check that the switch was successful by running `splunk cluster-manager-redundancy -show-status`.
    2. Stop the previously active manager.
    3. Upgrade the manager.
    4. Start the manager.
    5. Check that the manager enters standby mode and resyncs with the new active manager.
4. Upgrade the search heads and peer nodes in the usual fashion.
5. (Optional) Switch the original active manager back to active mode.
6. Take all managers out of maintenance mode by running `splunk disable maintenance-mode` on each manager.

Perform a rolling upgrade of an indexer cluster

Splunk Enterprise version 7.1.0 and higher supports rolling upgrade for indexer clusters. Rolling upgrade lets you perform a phased upgrade of indexer peers with minimal interruption to your ongoing searches. You can use rolling upgrade to minimize search disruption when upgrading peer nodes to a new version of Splunk Enterprise.

Requirements and considerations

Review the following requirements and considerations before you initiate a rolling upgrade:

- Rolling upgrade only applies to upgrades from version 7.1.x to a higher version of Splunk Enterprise.
- The manager node and all peer nodes must be running version 7.1.0 or higher. For upgrade instructions, see Upgrade an indexer cluster.
- All search heads and search head clusters must be running version 7.1.0 or higher.
- Do not attempt any clustering maintenance operations, such as rolling restart, bundle pushes, or node additions, during upgrade.

Hardware or network failures that prevent node shutdown or restart might require manual intervention.

How a rolling upgrade works

When you initiate a rolling upgrade, you select a peer and take it offline. During the offline process, the manager node reassigns bucket primaries to other peers to retain the searchable state, and the peer completes any in-progress

searches within a configurable timeout. See The fast offline process.

After the manager node shuts down the peer, you perform the software upgrade and bring the peer back online, at which point the peer rejoins the cluster. You repeat this process for each peer node until the rolling upgrade is complete.

A rolling upgrade behaves in the following ways:

- Peer upgrades occur one at a time based on the default search factor of 2. With a search factor of 3 or greater, you can upgrade (search factor - 1) number of peers at a time. For example, with a search factor of 3 you can upgrade 2 peers at a time. The number of peers you can upgrade simultaneously is the same for both single-site and multisite clusters, as the guidance for multisite clusters is to upgrade one site at a time. So for a multisite cluster, with a search factor of 3, you can upgrade 2 peers at a time in the same site.
- The peer waits for any in-progress searches to complete. It will wait up to a maximum time period determined by the `decommission_search_jobs_wait_secs` attribute in `server.conf`. The default of 180 seconds is enough time for the majority of searches to complete in most cases.
- Rolling upgrades apply to both historical searches and real-time searches.

In-progress searches that take longer than the default 180 seconds might generate incomplete results and a corresponding error message. If you have a scheduled search that must complete, either increase the decommission_search_jobs_wait_secs value or do not perform a rolling upgrade within the search's timeframe.

Before you perform a rolling upgrade, make sure the search_retry attribute in the [search] stanza of limits.conf is set to false (the default). Setting this to true might cause searches that take longer than the decommission_search_jobs_wait_secs value to generate duplicate or partial results without an error message.

Scheduled search behavior during rolling upgrade

As of version 8.2.x, by default, during rolling upgrade, continuous scheduled searches continue to run and are not deferred, based on the `defer_scheduled_searchable_idxc = <boolean>` setting in `savedsearches.conf`, which now defaults to a value of "false".

In some cases, continuous scheduled searches can return partial results during rolling upgrade. If necessary, you can defer continuous scheduled searches until after rolling upgrade completes, as follows:

1. On the search head or search head cluster, edit `SPLUNK_HOME/etc/system/local/savedsearches.conf`.
2. Set `defer_scheduled_searchable_idxc = true`.
3. Restart Splunk.

Also as of version 8.2.x, by default, during rolling upgrade, real-time scheduled searches continue to run and are not deferred, based on the newly added `skip_scheduled_realtime_idxc = <boolean>` setting in `savedsearches.conf`, which defaults to a value of "false".

For more information on `defer_scheduled_searchable_idxc` and `skip_scheduled_realtime_idxc` settings, see savedsearches.conf in the *Admin Manual.*

For information on real-time and continuous scheduled search modes, see Real-time scheduling and continuous scheduling.

Data model acceleration searches are skipped during rolling upgrade.

Perform a rolling upgrade

To upgrade an indexer cluster with minimal search interruption, perform the following steps:

1. Run preliminary health checks

On the manager node, run the `splunk show cluster-status` command with the `verbose` option to confirm the cluster is in a searchable state:

```
splunk show cluster-status --verbose
```

This command shows information about the cluster state. Review the command output to confirm that the search factor is met and all data is searchable before you initiate the rolling upgrade.

> The cluster must have two searchable copies of each bucket to be in a searchable state for a rolling upgrade.

Here is an example of the output from the `splunk show cluster-status --verbose` command:

```
splunk@manager1:~/bin$ ./splunk show cluster-status --verbose

Pre-flight check successful .................. YES
 â  â  â  â  â  â  â   Replication factor met ............... YES
 â  â  â  â  â  â  â   Search factor met ..................... YES
 â  â  â  â  â  â  â   All data is searchable ............... YES
 â  â  â  â  â  â  â   All peers are up ..................... YES
 â  â  â  â  â  â  â   CM version is compatible ............. YES
 â  â  â  â  â  â  â   No fixup tasks in progress ........... YES
 â  â  â  â  â  â  â   Splunk version peer count { 7.1.0: 3 }

 Indexing Ready YES

 idx1          0026D1C6-4DDB-429E-8EC6-772C5B4F1DB5         default
         Searchable YES
         Status  Up
         Bucket Count=14
         Splunk Version=7.1.0

 idx3          31E6BE71-20E1-4F1C-8693-BEF482375A3F         default
         Searchable YES
         Status  Up
         Bucket Count=14
         Splunk Version=7.1.0

 idx2          81E52D67-6AC6-4C5B-A528-4CD5FEF08009         default
         Searchable YES
         Status  Up
         Bucket Count=14
         Splunk Version=7.1.0
```

The output shows that the health check is successful, which indicates the cluster is in a searchable state to perform a rolling upgrade.

For information on health check criteria, see Health check output details.

Health checks do not cover all potential cluster health issues. The checks apply only to the criteria listed.

Or, send a GET request to the following endpoint to monitor cluster health:

```
cluster/manager/health
```

If the endpoint output shows `pre_flight_check: 1`, then the health is successful.

For endpoint details, see cluster/manager/health in the *REST API Reference Manual*.

2. Upgrade the manager node

1. Stop the manager node.
2. Upgrade the manager node, following Splunk Enterprise upgrade procedure. See How to upgrade Splunk Enterprise in the *Installation Manual*.
3. Start the manager node and accept all prompts, if it is not already running.

You can use the manager node dashboard to verify that all cluster nodes are up and running. See View the manager node dashboard.

3. Upgrade the search head tier

If the search head tier consists of independent search heads, follow this procedure:

1. Stop all the search heads.
2. Upgrade the search heads, following the normal procedure for any Splunk Enterprise upgrade, as described in How to upgrade Splunk Enterprise in the *Installation Manual*.
3. Start the search heads, if they are not already running.

If the search head tier consists of a search head cluster, follow the procedure in Upgrade a search head cluster.

4. Initialize rolling upgrade

Run the following CLI command on the manager node:

```
splunk upgrade-init cluster-peers
```

Or, send a POST request to the following endpoint:

```
cluster/manager/control/control/rolling_upgrade_init
```

This initializes the rolling upgrade and puts the cluster in maintenance mode.

For endpoint details, see cluster/manager/control/control/rolling_upgrade_init in the *REST API Reference Manual*.

5. Take the peer offline

Taking multiple peers offline simultaneously can impact searches.

Run the following CLI command on the peer node:

```
splunk offline
```
Or, send a POST request to the following endpoint.

```
cluster/peer/control/control/decommission
```
The manager node reassigns bucket primaries, completes any ongoing searches, and then shuts down the peer.

For endpoint details, see cluster/peer/control/control/decommission in the *REST API Reference Manual*.

(Optional) Monitor peer status

To monitor the status of the offline process, send a GET request to the following enpoint:

```
cluster/manager/peers/<peer-GUID>
```

If the response shows "ReassigningPrimaries", the peer is not yet shut down.

For endpoint details, see cluster/manager/peers/{name} in the *REST API Reference Manual*.

6. Upgrade the peer node

Upgrade the peer node, following standard Splunk Enterprise upgrade procedure. See How to upgrade Splunk Enterprise in the *Installation Manual*.

7. Bring the peer online

Run the following command on the peer node.

```
splunk start
```
The peer node starts and automatically rejoins the cluster.

8. Validate version upgrade

Validate the version upgrade using the following endpoint:

```
cluster/manager/peers/<peer-GUID>
```
For endpoint details, see cluster/manager/peers/{name} in the *REST API Reference Manual*.

9. Repeat steps 5-8

Repeat steps 5-8 until upgrade of all peer nodes is complete.

10. Finalize rolling upgrade

Run the following CLI command on the manager node:

```
splunk upgrade-finalize cluster-peers
```
Or, send a POST request to the following endpoint:

`cluster/manager/control/control/rolling_upgrade_finalize`
This completes the upgrade process and takes the cluster out of maintenance mode.

For endpoint details, see cluster/manager/control/control/rolling_upgrade_finalize in the *REST API Reference Manual*.

Conflicting operations

You cannot run certain operations simultaneously:

- Data rebalance
- Excess bucket removal
- Rolling restart
- Rolling upgrade

If you trigger one of these operations while another one is already running, `splunkd.log`, the CLI, and Splunk Web all surface an error that shows a conflicting operation is in progress.

Get data into the indexer cluster

Ways to get data into an indexer cluster

Cluster peer nodes can get their data directly from any of the same sources as a non-clustered indexer. However, if data fidelity matters to you, you will use **forwarders** to initially consume the data before forwarding it to the peer nodes, rather than ingesting the data directly into the nodes.

Advantages of forwarders for data input

There are several key reasons for using forwarders to send data to your cluster:

- **To ensure that all incoming data gets indexed.** By activating the forwarder's optional **indexer acknowledgment** feature, you can ensure that all incoming data gets indexed and stored on the cluster. With indexer acknowledgement, when a source peer receives a block of data from a forwarder, it sends the forwarder an acknowledgment after it indexes the data and successfully replicates it to the target peers. If the forwarder does not receive an acknowledgment from the source peer, the forwarder resends the data. The forwarder continues to resend the data until it gets the acknowledgment. Indexer acknowledgment is the only way to ensure end-to-end data fidelity. See "How indexer acknowledgment works."

- **To handle potential node failure.** With **load-balanced** forwarders, if one peer in the group goes down, the forwarder continues to send its data to the remaining peers in the group. If, instead, you use direct inputs to the peers, the data source cannot continue to send data when its receiving peer goes down. See "How load balancing works."

- **To simplify the process of connecting data sources and peer nodes.** By enabling **indexer discovery** on your forwarders, the forwarders automatically load balance across all available peer nodes, including any that are later added to the cluster. See "Advantages of the indexer discovery method."

Configure inputs directly on the peers

If you decide not to use forwarders to handle your data inputs, you can set up inputs on each peer in the usual way; for example, by editing `inputs.conf` on the peers. For information on configuring inputs, read "Configure your inputs" in the *Getting Data In* Manual.

Use forwarders to get data into the indexer cluster

The main reasons to use forwarders with indexer clusters are:

- **To ensure that all incoming data gets indexed.** By activating the forwarder's optional **indexer acknowledgment** feature, you can ensure that all incoming data gets indexed and stored on the cluster. See "How indexer acknowledgment works."

- **To handle potential node failure.** With **load-balanced** forwarders, if one peer in the group goes down, the forwarder continues to send its data to the remaining peers in the group. See "How load balancing works."

- **To simplify the process of connecting data sources and peer nodes.** By enabling indexer discovery on your forwarders, the forwarders automatically load balance across all available peer nodes, including any that are later

added to the cluster. See "Advantages of the indexer discovery method."

To use forwarders to get data into clusters, you must perform two types of configuration:

- Connect forwarders to peer nodes.
- Configure the forwarders' data inputs.

Before continuing, you must be familiar with forwarders and how to use them to get data into Splunk Enterprise. For an introduction to forwarders, read "About forwarding and receiving" in the *Forwarding Data* manual. Subsequent topics in that manual describe all aspects of deploying and configuring forwarders.

Connect forwarders to peer nodes

There are two ways to connect forwarders to peer nodes:

- **Use the indexer discovery feature.** With **indexer discovery**, each forwarder queries the manager node for a list of all peer nodes in the cluster. It then uses load balancing to forward data to the set of peer nodes. In the case of a multisite cluster, a forwarder can optionally query the manager for a list of all peers on a single site. See "Use indexer discovery to connect forwarders to peer nodes."

- **Connect forwarders directly to peer nodes.** This is the traditional method for establishing forwarder/indexer connectivity. You specify the peer nodes directly on the forwarders as **receivers.** See "Connect forwarders directly to peer nodes."

Advantages of the indexer discovery method

Indexer discovery has advantages over the traditional method:

- When new peer nodes join the cluster, you do not need to reconfigure and restart your forwarders to connect to the new peers. The forwarder automatically gets the updated list of peers from the manager node. It uses load balancing to forward to all peers in the list.

- You can add new forwarders without needing to determine the current set of cluster peers. You just configure indexer discovery on the new forwarders.

- You can use **weighted load balancing** when forwarding data across the set of peers. With indexer discovery, the manager node can track the amount of total disk space on each peer and communicate that information to the forwarders. The forwarders then adjust the amount of data they send to each peer, based on the disk capacity.

Configure the data inputs to each forwarder

After you specify the connection between the forwarders and the receiving peers using the method you prefer, you must specify the data inputs to each forwarder, so that the forwarder has data to send to the cluster. You usually do this by editing the forwarder's `inputs.conf` file.

Read the *Getting Data In* manual, starting with "What Splunk can index" for detailed information on configuring data inputs. The topic in that manual entitled "Use forwarders" provides an introduction to specifying data inputs on forwarders.

How indexer acknowledgment works

To ensure end-to-end data fidelity, you must explicitly enable indexer acknowledgment on each forwarder sending data to the cluster.

In brief, indexer acknowledgment works like this: The forwarder sends data continuously to a peer node, in blocks of approximately 64kB. The receiving peer, also known as the "source peer", then streams each data block to its target peers. The forwarder maintains a copy of the block in memory until it gets an acknowledgment from the source peer. While waiting, it continues to send more data blocks.

When indexer acknowledgment is enabled, its default behavior ensures that the replication factor number of peers receive each block of data. To handle ingestion delays due to slow indexers, you can change the behavior to limit the number of peer nodes that must receive the data before acknowledgment occurs. In doing so, you are trading guarantee of data high availability in exchange for better cluster performance.

The main scenarios are:

- Full acknowledgment (default)
- Limited acknowledgment

Full acknowledgment behavior

By default, the cluster waits until the number of peers equal to the replication factor receive the data before returning acknowledgment to the forwarder. For example, in a replication factor 3 cluster, the source peer and its two target peers must all receive the data before the source peer returns acknowledgment to the forwarder.

If all goes well, the source peer:

1. receives the block of data, parses and indexes it, and writes the data (raw data and index data) to its file system.

2. streams copies of the raw data to each of its target peers to fulfill the replication factor.

3. receives notification from each target peer of either a successful or unsuccessful write.

4. sends an acknowledgment back to the forwarder.

At the conclusion of this process, the source peer plus its target peers have all received copies of the data block.

The acknowledgment assures the forwarder that the data was successfully written to the cluster. Upon receiving the acknowledgment, the forwarder releases the block from memory.

If the forwarder does not receive the acknowledgment, that means there was a failure along the way. Either the source peer went down or that peer was unable to contact its set of target peers. The forwarder then automatically resends the block of data. If the forwarder is using load-balancing, it sends the block to another peer in the load-balanced group. If the forwarder is not set up for load-balancing, it attempts to resend data to the same peer as before.

Limited acknowledgment behavior

To ensure that data continues to flow into the cluster at the expected rate, even in clusters with a slow or malfunctioning peer node, you can configure the acknowledgment behavior so that the process does not require that the full replication factor number of peers receive the data before acknowledgment. Instead the source peer can return acknowledgment to

the forwarder once some smaller number of peer nodes have received the data, without waiting for the entire replication factor number of peers to receive it.

For example, you can configure the behavior so that only one peer (that is, the source peer) must receive the data before returning the acknowledgment. Or, in a replication factor 3 cluster, you can specify that only two peers (the source peer and one of the target peers) must receive the data before the source peer returns an acknowledgment to the forwarder.

By doing so, you can avoid situations where the forwarder might hang unnecessarily while awaiting acknowledgment, due to a peer node performing poorly.

Limited acknowledgment only determines the timing for when the source peer returns an acknowledgment to the forwarder. The source peer continues to stream the data to its target peers, even after returning the acknowledgment, thus ensuring that the replication factor number of copies of the data resides on peers in the cluster.

Configure indexer acknowledgment behavior

The `ack_factor` setting in each peer node's `server.conf` determines the number of peers that must receive data before the source peer returns acknowledgment to the forwarder.

By default, `ack_factor` is set to 0, which signifies the replication number of peers. So, if `ack_factor=0` and the cluster has a replication factor of 3, then a total of 3 peer nodes must receive each data block before acknowledgment is returned to the forwarder.

To limit the number of peer nodes that must receive the data, you can set `ack_factor` to an integer between 1 and the replication factor. For example, to limit the number of peer nodes that must receive the data to one, set `ack_factor=1`. To limit the number of peer nodes to two, set `ack_factor=2`, and so on.

Note:

- The setting is valid only if `useACK=true` and `mode=peer`.
- All peer nodes in the cluster must set `ack_factor` to the same value.
- This setting requires a restart to take effect.

For more information on how indexer acknowledgment works, read "Protect against loss of in-flight data" in the *Forwarding Data* manual.

How load balancing works

In load balancing, the forwarder distributes incoming data across several receiving peer nodes. Each node gets a portion of the total data, and together the receiving nodes get all the data.

Splunk forwarders perform "automatic load balancing". The forwarder routes data to different nodes based on a specified time interval. For example, assume you have a load-balanced group consisting of three peer nodes: A, B, and C. At the interval specified by the `autoLBFrequency` attribute in `outputs.conf` (30 seconds by default), the forwarder switches the data stream to another node in the group, selected at random. So, every 30 seconds, the forwarder might switch from node B to node A to node C, and so on. If one node is down, the forwarder immediately switches to another.

Note: To expand on this a bit, each of the forwarder's inputs has its own data stream. At the specified interval, the forwarder switches the data stream to the newly selected node, if it is safe to do so. If it cannot safely switch the data stream to the new node, it keeps the connection to the previous node open and continues to send the data stream to that

node until it has been safely sent.

Load balancing, in conjunction with indexer acknowledgment, is of key importance in a clustered deployment because it helps ensure that you don't lose any data in case of node failure. If a forwarder does not receive indexer acknowledgment from the node it is sending data to, it resends the data to the next available node in the load-balanced group.

Forwarders using the indexer discovery feature always use load balancing to send data to the set of peer nodes. You can enable weighted load balancing, which means that the forwarder distributes data based on each peer's disk capacity. For example, a peer with a 400GB disk receives twice the data of a peer with a 200GB disk. See "Use weighted load balancing."

For further information on:

- load balancing with indexer discovery, see "Use indexer discovery to connect forwarders to peer nodes."
- load balancing without indexer discovery, see "Set up load balancing" in the *Forwarding Data* manual.
- how load balancing works with indexer acknowledgment, read "Protect against loss of in-flight data" in the *Forwarding Data* manual.

Use indexer discovery to connect forwarders to peer nodes

Indexer discovery streamlines the process of connecting forwarders to peer nodes in indexer clusters. It simplifies the set-up and maintenance of an indexer cluster. See Advantages of the indexer discovery method. Indexer discovery is available only for forwarding to indexer clusters.

Each forwarder queries the manager node for a list of all peer nodes in the cluster. It then uses load balancing to forward data to the set of peer nodes. In the case of a multisite cluster, a forwarder can query the manager for a list of all peers on a single site.

How indexer discovery works

Briefly, the process works like this:

1. The peer nodes provide the manager node with information on their receiving ports.

2. The forwarders poll the manager at regular intervals for the list of available peer nodes. You can adjust this interval. See Adjust the frequency of polling.

3. The manager transmits the peer nodes' URIs and receiving ports to the forwarders.

4. The forwarders send data to the set of nodes provided by the manager.

In this way, the forwarders stay current with the state of the cluster, learning of any peers that have joined or left the cluster and updating their set of receiving peers accordingly.

In the case of a multisite cluster, each forwarder can identify itself as a member of a site. In that case, the manager node transmits a list of all peer nodes for that site only, and the forwarder limits itself to load balancing across that site. See Use indexer discovery in a multisite cluster.

In addition, the peer nodes can use **weighted load balancing** to adjust the amount of data they send to each peer based on that peer's relative disk capacity. See Use weighted load balancing.

Note: If the manager node goes down, the forwarders will rely on their most recent list of available peer nodes. However, the list does not persist through a forwarder restart. Therefore, if a forwarder restarts while the manager is down, it will not have a list of peer nodes and will not be able to forward data, resulting in potential data loss. Similarly, if a forwarder starts up for the first time, it must wait for the manager to return before it can get a list of peers.

Configure indexer discovery

These are the main steps for setting up connections between forwarders and peer nodes, using indexer discovery:

1. Configure the peer nodes to receive data from forwarders.

2. Configure the manager node to enable indexer discovery.

3. Configure the forwarders.

After you set up the connection, you must configure the data inputs on the forwarders. See Configure the data inputs to each forwarder.

1. Configure the peer nodes to receive data from forwarders

In order for a peer to receive data from forwarders, you must configure the peer's **receiving port**. One way to specify the receiving port is to edit the peer's inputs.conf file. For example, this setting in `inputs.conf` sets the receiving port to 9997:

```
[splunktcp://9997]
disabled = 0
```

Restart the peer node after making the change.

See Enable a receiver in the *Forwarding Data* manual.

Caution: When using indexer discovery, each peer node can have only a single configured receiving port. The port can be configured for either `splunktcp` or `splunktcp-ssl`, but not for both. You must use the same method for all peer nodes in the cluster: `splunktcp` or `splunktcp-ssl`.

You can simplify peer input configuration by deploying a single, identical `inputs.conf` file across all the peers. The receiving port that you specify in the common copy of `inputs.conf` will supersede any ports you enable on each individual peer. For details on how to create and deploy a common `inputs.conf` across all peers, see Update common peer configurations and apps.

When forwarding to a multisite cluster, you can configure the forwarder to send data only to peers in a specified site. See Use indexer discovery in a multisite cluster.

2. Configure the manager node to enable indexer discovery

In `server.conf` on the manager node, add this stanza:

```
[indexer_discovery]
pass4SymmKey = <string>
polling_rate = <integer>
indexerWeightByDiskCapacity = <bool>
```

Note the following:

- The `pass4SymmKey` attribute specifies the security key used with communication between the manager node and the forwarders. Its value must be the same for all forwarders and the manager node. The `pass4SymmKey` attribute used for indexer_discovery should have a different value from the `pass4SymmKey` attribute used for communication between the manager and the cluster nodes, which is set in the `[clustering]` stanza, as described in Configure the security key.
- The `polling_rate` attribute (optional) provides a means to adjust the rate at which the forwarders poll the manager for the latest list of peer nodes. Its value must be an integer between 1 and 10. The default is 10. See Adjust the frequency of polling.
- The `indexerWeightByDiskCapacity` attribute (optional) determines whether indexer discovery uses weighted load balancing. The default is false. See Use weighted load balancing.

3. Configure the forwarders

a. Configure the forwarders to use indexer discovery

On each forwarder, add these settings to the `outputs.conf` file:

```
[indexer_discovery:<name>]
pass4SymmKey = <string>
manager_uri = <uri>

[tcpout:<target_group>]
indexerDiscovery = <name>

[tcpout]
defaultGroup = <target_group>
```

Note the following:

- In the `[indexer_discovery:<name>]` stanza, the `<name>` references the `<name>` set in the `indexerDiscovery` attribute in the `[tcpout:<target_group>]` stanza.
- The `pass4SymmKey` attribute specifies the security key used with communication between the manager and the forwarders. Its value must be the same for all forwarders and the manager node. You must explicitly set this value for each forwarder.
- The `<manager_uri>` is the URI and management port for the manager node. For example: "https://10.152.31.202:8089".
- In the `[tcpout:<target_group>]` stanza, set the `indexerDiscovery` attribute, instead of the `server` attribute that you would use to specify the receiving peer nodes if you were not enabling indexer discovery. With indexer discovery, the forwarders get their list of receiving peer nodes from the manager, not from the `server` attribute. If both attributes are set, `indexerDiscovery` takes precedence.

b. Enable indexer acknowledgment for each forwarder

Note: This step is required to ensure end-to-end data fidelity. If that is not a requirement for your deployment, you can skip this step.

To ensure that the cluster receives and indexes all incoming data, you must turn on indexer acknowledgment for each forwarder.

To configure indexer acknowledgment, set the `useACK` attribute in each forwarder's `outputs.conf`, in the same stanza where you set the `indexerDiscovery` attribute:

```
[tcpout:<target_group>]
```

```
indexerDiscovery = <name>
useACK=true
```
For detailed information on configuring indexer acknowledgment, read Protect against loss of in-flight data in the *Forwarding Data* manual.

Example

In this example:

- The manager node enables indexer discovery.
- The manager and forwarders share a security key.
- Forwarders will send data to peer nodes weighted by the total disk capacity of the peer nodes' disks.
- The forwarders use indexer acknowledgment to ensure end-to-end fidelity of data.

In the manager node's: `server.conf`:

```
[indexer_discovery]
pass4SymmKey = my_secret
indexerWeightByDiskCapacity = true
```
In each forwarder's `outputs.conf`:

```
[indexer_discovery:manager1]
pass4SymmKey = my_secret
manager_uri = https://10.152.31.202:8089

[tcpout:group1]
autoLBFrequency = 30
forceTimebasedAutoLB = true
indexerDiscovery = manager1
useACK=true

[tcpout]
defaultGroup = group1
```
Use indexer discovery in a multisite cluster

In multisite clustering, the cluster is partitioned into sites, typically based on the location of the cluster nodes. See Multisite indexer clusters. When using indexer discovery with multisite clustering, you can configure each forwarder to be site-aware, so that it forwards data to peer nodes only on a single specified site.

When you use indexer discovery with multisite clustering, you must assign a `site-id` to all forwarders, whether or not you want the forwarders to be site-aware.:

- If you want a forwarder to be site-aware, you assign it a `site-id` for a site in the cluster, such as "site1," "site2," and so on.
- If you do not want a forwarder to be site-aware, you assign it the special `site-id` of "site0". When a forwarder is assigned "site0", it will forward to peers across all sites in the cluster.

Assign a site-id to each forwarder

To assign a site-id, add this stanza to the forwarder's `server.conf` file:

```
[general]
site = <site-id>
```
Note the following:

- You must assign a `<site-id>` to each forwarder sending data to a multisite cluster. This must either be a valid site in the cluster or the special value "site0".
- If you want the forwarder to send data only to peers at a specific site, assign the id for that site, such as "site1."
- If you want the forwarder to send data to all peers, across all sites, assign a value of "site0".
- If you do not assign any id, the forwarder will not send data to any peer nodes.
- See also Site values.

Configure the forwarder site failover capability

If you assign a forwarder to a specific site and that site goes down, the forwarder, by default, will not fail over to another site. Instead, it will stop forwarding data if there are no peers available on its assigned site. To avoid this issue, you must configure the forwarder site failover capability.

To configure the forwarder site failover capability, set the `forwarder_site_failover` attribute in the manager node's `server.conf` file.

For example:

```
[clustering]
forwarder_site_failover = site1:site2, site2:site3
```
This example configures failover sites for site1 and site2. If site1 fails, all forwarders configured to send data to peers on site1 will instead send data to peers on site2. Similarly, if site2 fails, all forwarders explicitly configured to send data to peers on site2 will instead send data to peers on site3.

Note: The failover capability does not relay from site to site. In other words, in the previous example, if a forwarder is set to site1 and site1 goes down, the forwarder will then start forwarding to peers on site2. However, if site2 subsequently goes down, the site1 forwarder will not then failover to site3. Only forwarders explicitly set to site2 will failover to site3. Each forwarder can have only a single failover site.

The forwarders revert to their assigned site, as soon as any peer on that site returns to the cluster. For example, assume that the manager node includes this configuration:

```
[clustering]
forwarder_site_failover = site1:site2
```
When site1 goes down, such that there are no peers running on site1, the forwarders assigned to site1 start sending data to peers on site2 instead. This failover condition continues until a site1 peer returns to the cluster. At that point, the forwarders assigned to site1 start forwarding to that peer. They no longer forward to peers on site2.

Use weighted load balancing

When you enable indexer discovery, the forwarders always stream the incoming data across the set of peer nodes, using load balancing to switch the data stream from node to node. This operates in a similar way to how forwarders without indexer discovery use load balancing, but with some key differences. In particular, you can enable weighted load balancing.

In weighted load balancing, the forwarders take each peer's disk capacity into account when they load balance the data. For example, a peer with a 400GB disk receives approximately twice the data of a peer with a 200GB disk.

Important: The disk capacity refers to the total amount of local disk space on the peer, not the amount of free space.

How weighted load balancing works

Weighted load balancing behaves similarly to normal forwarder load balancing. The `autoLBFrequency` attribute in the forwarder's `outputs.conf` file still determines how often the data stream switches to a different indexer. However, when the forwarder selects the next indexer, it does so based on the relative disk capacities. The selection itself is random but weighted towards indexers with larger disk capacities.

In other words, the forwarder uses weighted picking. So, if the forwarder has an `autoLBFrequency` set to 60, then every sixty seconds, the forwarder switches the data stream to a new indexer. If the load balancing is taking place across two indexers, one with a 500GB disk and the other with a 100GB disk, the indexer with the larger disk is five times as likely to be picked at each switching point.

The overall traffic sent to each indexer is based this ratio:

```
indexer_disk_capacity/total_disk_capacity_of_indexers_combined
```

For a general discussion of load balancing in indexer clusters, see How load balancing works.

Enable weighted load balancing

The `indexerWeightByDiskCapacity` attribute in the manager node's `server.conf` file controls weighted load balancing:

```
[indexer_discovery]
indexerWeightByDiskCapacity = <bool>
```

Note the following:

- The `indexerWeightByDiskCapacity` attribute is set to false by default. To enable weighted load balancing, you must set it to true.

Change the advertised disk capacity for an indexer

In some cases, you might want weighted load balancing to treat an indexer as though it has a lower disk capacity than it actually has. You can use the `advertised_disk_capacity` attribute to accomplish this. For example, if you set that attribute to 50 (signifiying 50%) on an indexer with a 500GB disk, weighted load balancing will proceed as though the actual disk capacity was 250GB.

You set the `advertised_disk_capacity` attribute in the indexer's `server.conf` file:

```
[clustering]
advertised_disk_capacity = <integer>
```

Note the following:

- The `advertised_disk_capacity` attribute indicates the percentage that will be applied to the indexer's actual disk capacity before it sends the capacity to the manager node. For example, if set to 50 on an indexer with a 500GB disk, the indexer tells the manager that the disk capacity is 250GB.

- The value can vary from 10 to 100.
- The default is 100.

Adjust the frequency of polling

Forwarders poll the manager node at regular intervals to receive the most recent list of peers. In this way, they become aware of any changes to the set of available peers and can modify their forwarding accordingly. You can adjust the rate of polling.

The frequency of polling is based on the number of forwarders and the value of the `polling_rate` attribute, configured in the manager's `server.conf` file. The polling interval for each forwarder follows this formula:

```
(number_of_forwarders/polling_rate + 30 seconds) * 1000 = polling interval, in milliseconds
```

Here are some examples:

```
# 100 forwarders, with the default polling_rate of 10
(100/10 + 30) * 1000 = 40,000 ms., or 40 seconds

# 10,000 forwarders, with the default polling_rate of 10
(10000/10 + 30) * 1000 = 1,030,000 ms., or 1030 seconds, or about 17 minutes

# 10,000 forwarders, with the minimum polling_rate of 1
(10000/1 + 30) * 1000 = 10,030,000 ms., or 10,030 seconds, or a bit under three hours
```

To configure `polling_rate`, add the attribute to the `[indexer_discovery]` stanza in `server.conf` on the manager node:

```
[indexer_discovery]
polling_rate = <integer>
```

Note the following:

- The `polling_rate` attribute must be an integer between 1 and 10.
- The default is 10.

Configure indexer discovery with SSL

You can configure indexer discovery with SSL. The process is nearly the same as configuring without SSL, with just a few additions and changes:

1. Configure the peer nodes to receive data from forwarders over SSL.

2. Configure the manager node to enable indexer discovery.

3. Configure the forwarders for SSL.

The steps below provide basic configuration information only, focusing on the differences when configuring for SSL. For full details on indexer discovery configuration, see Configure indexer discovery.

1. Configure the peer nodes to receive data from forwarders over SSL

Edit each peer's inputs.conf file to specify the receiving port and to configure the necessary SSL settings:

```
[splunktcp-ssl://9997]
disabled = 0

[SSL]
serverCert = <path to server certificate>
sslPassword = <certificate password>
```

Note: When using indexer discovery, each peer node can have only a single receiving port. For SSL, you must configure a port for `splunktcp-ssl` only. Do not configure a `splunktcp` stanza.

In addition, confirm that `sslRootCAPath` is set in each peer's `server.conf` file.

2. Configure the manager node to enable indexer discovery

In `server.conf` on the manager node, add this stanza:

```
[indexer_discovery]
pass4SymmKey = <string>
polling_rate = <integer>
indexerWeightByDiskCapacity = <bool>
```

This is the same as for configuring a non-SSL set-up.

3. Configure the forwarders for SSL

On each forwarder, add these settings to the `outputs.conf` file:

```
[indexer_discovery:<name>]
pass4SymmKey = <string>
manager_uri = <uri>

[tcpout:<target_group>]
indexerDiscovery = <name>
useACK = true
clientCert = <path to  client certificate>
sslPassword = <CAcert password>

[tcpout]
defaultGroup = <target_group>
```

In addition, confirm that `sslRootCAPath` is set in each forwarder's `server.conf` file.

Connect forwarders directly to peer nodes

These are the main steps for setting up connections between forwarders and peer nodes, using the traditional method of connecting each forwarder directly to each peer node:

1. Configure the peer nodes to receive data from forwarders.

2. Configure the forwarders to send data to the peer nodes.

3. Enable indexer acknowledgment for each forwarder. This step is required to ensure end-to-end data fidelity. If that is not a requirement for your deployment, you can skip this step.

Once you finish setting up the connection, you must configure the data inputs on the forwarders. See "Configure the forwarder's data inputs".

1. Configure the peer nodes to receive data from forwarders

In order for a peer to receive data from forwarders, you must configure the peer's **receiving port**. For information on how to configure the receiving port, read "Enable a receiver" in the *Forwarding Data* manual.

One way to specify the receiving port is to edit the peer's inputs.conf file. You can simplify peer input configuration by deploying a single, identical `inputs.conf` file across all the peers. The receiving port that you specify in the common copy of `inputs.conf` will supersede any ports you enable on each individual peer. For details on how to create and deploy a common `inputs.conf` across all peers, read "Update common peer configurations".

2. Configure the forwarders to send data to the peer nodes

When you set up a forwarder, you specify its receiving peer by providing the peer's IP address and receving port number. For example: `10.10.10.1:9997`. You do this in the forwarder's outputs.conf file, as described in "Configure forwarders with outputs.conf" in the *Forwarding Data* manual. To specify the receiving peer, set the `server` attribute, like this:

```
server=10.10.10.1:9997
```

The receiving port that you specify here is the port that you configured on the peer in step 1.

To set up the forwarder to use load-balancing, so that the data goes to multiple peer nodes in sequence, you configure a load-balanced group of receiving peers. For example, this attribute/value pair in `outputs.conf` specifies a load-balanced group of three peers:

```
server=10.10.10.1:9997,10.10.10.2:9997,10.10.10.3:9997
```

To learn more about configuring load balancing, read "Set up load balancing" in the *Forwarding Data* manual.

Note: There are several other ways that you can specify a forwarder's receiving peer(s). For example:

- You can specify the receiving peer during universal forwarder deployment (for Windows universal forwarders only), as described in Install a Windows universal forwarder in the *Universal Forwarder* manual.
- You can specify the receiver with the CLI command `add forward-server`, as described in Enable a receiver in the *Forwarding Data* manual.

Both of these methods work by modifying the underlying `outputs.conf` file. No matter what method you use to specify the receiving peers, you still need to directly edit the underlying `outputs.conf` file if you want to turn on indexer acknowledgment, as described in the next step.

3. Enable indexer acknowledgment for each forwarder

This step is required to ensure end-to-end data fidelity. If that is not a requirement for your deployment, you can skip this step.

To ensure that the cluster receives and indexes all incoming data, you must turn on indexer acknowledgment for each forwarder.

Caution: Indexer acknowledgement can, under some circumstances, result in duplicate events. To learn about this issue and how to work around it, see Protect against loss of in-flight data in the *Forwarding Data* manual.

To configure indexer acknowledgment, set the `useACK` attribute in each forwarder's `outputs.conf`:

```
[tcpout:<peer_target_group>]
useACK=true
```

For detailed information on configuring indexer acknowledgment, read Protect against loss of in-flight data in the *Forwarding Data* manual.

Caution: For indexer acknowledgment to work properly, the forwarders' wait queues must be configured to the optimal size. For forwarders at version 5.0.4 or above, the system handles this automatically. For earlier version forwarders, follow the instructions in the version of the Protect against loss of in-flight data topic *for that forwarder version.* Specifically, read the subtopic on adjusting the `maxQueueSize` setting.

Example: A load-balancing forwarder with indexer acknowledgment

Here is a sample `outputs.conf` configuration for a forwarder that is using load balancing to send data in sequence to three peers in a cluster. It assumes that each of the peers has previously been configured to use 9997 for its receiving port:

```
[tcpout]
defaultGroup=my_LB_peers

[tcpout:my_LB_peers]
autoLBFrequency=40
server=10.10.10.1:9997,10.10.10.2:9997,10.10.10.3:9997
useACK=true
```

The forwarder starts by sending data to one of the peers listed for the `server` attribute. After 40 seconds, it switches to another peer, and so on. If, at any time, it doesn't receive acknowledgment from the current receiving node, it resends the data, this time to the next available node.

Configure the indexer cluster

Indexer cluster configuration overview

To configure the indexer cluster, you configure the individual nodes. You perform two types of configuration:

- Configuration of the behavior of the cluster itself.
- Configuration of the cluster's indexing and search behavior.

The current chapter provides an overview of the ways to configure cluster behavior specifically.

Configuration of each node type

The settings for each node type handle different aspects of the cluster:

- **Manager node.** Configuration of overall cluster behavior.
- **Peer node.** Configuration of individual peer node and cluster indexing behavior.
- **Search head.** Configuration of individual search head and search behavior in an indexer cluster.

See the chapters on specific node types for information on configuring each node type. The chapter "Configure the peers," for example, includes some topics on configuring the peer cluster node settings and other topics that describe how to configure the indexes that a peer uses.

Methods for configuring cluster behavior

Initial configuration of each node occurs during deployment. If you need to change the configuration of a cluster node post-deployment, you have these choices:

- You can edit the configuration from the node's dashboard in Splunk Web, as described in "Configure the indexer cluster with the dashboards".
- You can directly edit the `[clustering]` stanza in the node's `server.conf` file. See "Configure the indexer cluster with server.conf" for details. To configure some advanced settings, you must edit this file.
- You can use the CLI. See "Configure and manage the indexer cluster with the CLI" for details.

Configure the indexer cluster with the dashboards

To configure an indexer cluster node through its dashboard:

1. Click **Settings** on the upper right side of Splunk Web on the node.

2. In the **Distributed Environment** group, click **Indexer clustering**.

3. Select the **Edit** button on the upper right side of the dashboard.

For information on the settings for each node type, see:

- "Configure the manager node with the dashboard"
- "Configure peer nodes with the dashboard"
- "Configure the search head with the dashboard"

Configure the indexer cluster with server.conf

Before reading this topic, see "About configuration files" and the topics that follow it in the *Admin Manual.* Those topics explain how Splunk Enterprise uses configuration files.

Indexer cluster settings reside in the server.conf file, located in `$SPLUNK_HOME/etc/system/local/`. When you deploy a cluster node through Splunk Web or the CLI, the node saves the settings to that file. You can also edit `server.conf` file directly, either to deploy initially or to change settings later.

The main `server.conf` stanza that controls indexer clustering is `[clustering]`. Besides the basic attributes that correspond to settings in Splunk Web, `server.conf` provides a number of advanced settings that control communication between cluster nodes. Unless advised by Splunk Support, do not change those settings.

This topic discusses some issues that are common to all node types.

Configure the various node types

For specific instructions for each node type, see:

- "Configure the manager node with server.conf".
- "Configure peer nodes with server.conf".
- "Configure the search head with server.conf".

For details on all the clustering attributes, including the advanced ones, read the server.conf specification.

For multisite cluster configurations, also read "Configure multisite indexer clusters with server.conf".

Configure the security key

Set the `pass4SymmKey` attribute to configure a security key that authenticates communication between the manager node, peers, and search heads. You must use the same key value for all cluster nodes.

You set `pass4SymmKey` when you deploy the cluster. For details on how to set the key on the manager node, see Enable the indexer cluster manager node. You also set it when when enabling the peer nodes and search heads.

If you set the key directly in `server.conf`, you must set it inside the `[clustering]` stanza for indexer clustering.

Important: Save a copy of the key in a safe place. Once an instance starts running, the security key changes from clear text to encrypted form, and it is no longer recoverable from `server.conf`. If you later want to add a new node, you will need to use the clear text version to set the key.

For information on setting the security key for a combined search head cluster and indexer cluster, see Integrate the search head cluster with an indexer cluster in *Distributed Search*.

Restart after modifying server.conf?

After you configure an instance as a cluster node for the first time, you need to restart it for the change to take effect.

If you make a configuration change later on, you might not need to restart the instance, depending on the type of change. Avoid restarting peers when possible. Restarting the set of peers can result in prolonged amounts of bucket-fixing.

Initial configuration

After initially configuring instances as cluster nodes, you need to restart all of them (manager node, peers, and search head) for the changes to take effect. You can do this by invoking the CLI `restart` command on each node:

```
$SPLUNK_HOME/bin/splunk restart
```

When the manager node starts up for the first time, it blocks indexing on the peers until you enable and restart the replication factor number of peers. Do not restart the manager while it is waiting for the peers to join the cluster. If you do, you will need to restart the peers a second time.

Important: Although you can use the CLI `restart` command when you initially enable an instance as a cluster peer node, do not use it for subsequent restarts. The `restart` command is not compatible with index replication once replication has begun. For more information, including a discussion of safe restart methods, read "Restart a single peer".

Subsequent configuration changes

If you change any of the following attributes in the `server.conf` file, you do not need to restart the node.

On a peer node:

- `manager_uri`
- `notify_scan_period`

On a search head:

- `manager_uri`

On a manager node:

- `quiet_period`
- `heartbeat_timeout`
- `restart_timeout`
- `max_peer_build_load`
- `max_peer_rep_load`
- `cluster_label`
- `access_logging_for_heartbeats`
- `use_batch_mask_changes`
- `percent_peers_to_restart`
- `summary_replication`

All other cluster-related configuration changes require a restart.

Configure and manage the indexer cluster with the CLI

You can use the CLI to perform a wide set of indexer cluster activities, including:

- Configuring cluster nodes
- Viewing cluster information
- Managing the cluster

Some clustering commands are available only for a specific node type,Â such as the manager node.

This topic discusses issues that are common to all node types.

Configure cluster nodes

You can use the CLI to enable any of the cluster node types or to change their configurations later:

- To enable or edit a manager node, see "Configure the manager node with the CLI".
- To enable or edit a peer node, see "Configure peer nodes with the CLI".
- To enable or edit a search head, see "Configure the search head with the CLI".

For details on specific command-line options, read "Configure the indexer cluster with server.conf".

For multisite cluster configurations, also read "Configure multisite indexer clusters with the CLI".

Specify a security key

You specify a security key for the cluster by appending the `-secret` flag when you enable each cluster node. For example, you specify it when configuring a peer node:

```
splunk edit cluster-config -mode peer -manager_uri https://10.160.31.200:8089 -replication_port 9887 -secret your_key
```

The security key authenticates communication between the manager node and the peers and search heads. The key is required and must be the same across all cluster nodes.

The -secret flag modifies the pass4SymmKey setting in the [clustering] stanza of server.conf.

View cluster information

There are a number of `splunk list` commands that return different types of cluster information. For example, to get detailed information on each peer in the cluster, run this command on the manager node:

```
splunk list cluster-peers
```

To get information on the cluster configuration, run this command from any node:

```
splunk list cluster-config
```

See the CLI clustering help for the full set of `splunk list` commands.

Manage the cluster

You can also use the CLI to perform a number of different actions on the cluster. Those actions are described in their own topics:

- Use the `splunk offline` command to take a peer offline.
- Use the `splunk apply cluster-bundle` command to update common peer configurations.
- Use the `splunk rolling-restart cluster-peers` command to restart all the cluster peers.
- Use the `splunk enable maintenance-mode` command to enable maintenance mode.
- Use the `splunk remove excess-buckets` command to remove excess bucket copies.
- Configure multi-cluster search.

Get help on the CLI commands

The CLI provides online help for its commands. For general help on the full set of clustering commands, go to `$SPLUNKHOME/bin` and type:

```
splunk help cluster
```

For help on specific commands, specify the command name. For example:

```
splunk help list cluster-config
```

For general information on the CLI, read the "Administer Splunk Enterprise with the command line interface (CLI)" chapter in the *Admin Manual,* or type:

```
splunk help
```

Configure the manager node

Manager node configuration overview

You initially configure the manager node when you enable it, as described in "Enable the managernode". This is usually all the configuration the manager node needs.

Change the configuration

If you need to edit the configuration, you have these choices:

- You can edit the configuration from the manager node dashboard in Splunk Web. See "Configure the manager node with the dashboard".
- You can directly edit the `[clustering]` stanza in the manager's `server.conf` file. To configure some advanced settings, you must edit this file. See "Configure the manager node with server.conf".
- You can use the CLI. See "Configure the manager node with the CLI".

After you change the manager node configuration, you must restart the manager for the changes to take effect.

Important: The manager node has the sole function of managing the other cluster nodes. Do not use it to index external data or to search the cluster.

Changes that require caution

Be careful when making changes to these settings:

- **Replication factor and search factor.** It is inadvisable to increase either of these settings after your indexer cluster contains significant amounts of data. This will kick off a great deal of bucket activity, affecting the cluster's performance adversely while bucket copies are being created or made searchable.
- **Heartbeat timeout.** Do not change the `heartbeat_timeout` attribute from its default value of 60 (seconds) unless instructed to do so by Splunk Support. In particular, do not decrease it. This can overload the peers.

Configure a stand-by manager node

To prepare for manager node failure, configure a stand-by manager node that can take over if the current manager goes down. See "Replace the manager node on the indexer cluster".

Configure a multisite manager node

A multisite manager node has a number of configuration differences and additions, compared to a basic, single-site cluster. See "Configure multisite indexer clusters with server.conf".

Configure the manager node with the dashboard

You can edit the configuration of an existing manager node through its dashboard:

1. In Splunk Web, click **Settings > Indexer clustering**.
   The manager node dashboard appears.
2. Select the **Edit** button on the upper right side of the dashboard.
   The **Edit** button presents several options:

- **Node Type**. Change the instance's node type. **Caution:** It is extremely unlikely that you will want to change the node type for nodes in an active cluster. Consider the consequences carefully before doing so.
- **Manager Node Configuration**. Change these manager node settings:
  - **Replication Factor**. Change the cluster's replication factor. **Caution:** It is inadvisable to increase the replication factor after your cluster contains significant amounts of data. Doing so will kick off a great deal of bucket activity, which will have an adverse effect on the cluster's performance while bucket copies are being created.
  - **Search Factor**. Change the cluster's search factor. **Caution:** It is inadvisable to increase the search factor after your cluster contains significant amounts of data. Doing so will kick off a great deal of bucket activity, which will have an adverse effect on the cluster's performance while bucket copies are being made searchable.
  - **Security Key**. Change the security key. Only change the security key if you are also changing it for all other nodes in the cluster. The key must be the same across all instances in a cluster.
  - **Cluster Label**. Label the cluster. The label is useful for identifying the cluster in the monitoring console. See Set cluster labels in *Securing Splunk Enterprise.*

  > The Manager Node Configuration option is disabled for multisite clusters.
- **Configuration Bundle Actions.** Click **Push** to distribute the configuration bundle from the manager node to peer nodes. Optionally, validate the bundle and check restart without applying the bundle, or rollback to the previous bundle. See Update common peer configurations and apps.
- **Data Rebalance.** Rebalance the buckets so that each peer has approximately the same number of bucket copies. See Rebalance the indexer cluster.
- **Disable Indexer Clustering**. Remove this node from the cluster. **Caution:** If you remove the manager node from the cluster, the entire cluster will eventually fail.

For information on using this dashboard to enable a manager node initially, see Enable the indexer cluster manager node.

For information on using this dashboard to view cluster status, see View the manager node dashboard.

Configure the manager node with server.conf

Prerequisites

Before reading this topic, see Configure the indexer cluster with server.conf. It discusses configuration issues that are common to all cluster node types.

Enable the manager node

The following example shows the basic settings that you configure when enabling a manager node. Unless otherwise

noted, the settings are required. The configuration attributes correspond to fields on the **Enable clustering** page of Splunk Web.

```
[clustering]
mode = manager
replication_factor = 4
search_factor = 3
pass4SymmKey = whatever
cluster_label = cluster1
```

This example specifies that:

- the instance is a cluster manager node.
- the cluster's replication factor is 4.
- the cluster's search factor is 3.
- the security key is "whatever". All nodes in the cluster use the same security key. See Configure the security key.
- the cluster label is "cluster1." The optional cluster label is useful for identifying the cluster in the monitoring console. See Set cluster labels in *Monitoring Splunk Enterprise.* You set this attribute on the manager node only.

When the manager node starts up for the first time, it will block indexing on the peers until you enable and restart the full replication factor number of peers. Do not restart the manager while it is waiting for the peers to join the cluster. If you do, you will need to restart the peers a second time.

When you enable the manager node in Splunk Web, the resulting server.conf stanza includes attributes only for non-default values. For example, if you accept the default replication factor of 3 and do not enter a new value for it, the resulting stanza does not include the replication_factor attribute.

Edit the manager node settings

You can change these settings later, if necessary. For example, to change the cluster's security key, you edit the `pass4SymmKey` value on each node.

For details on all cluster attributes, including some advanced ones that rarely require editing, read the server.conf specification.

Configure the manager node with the CLI

Read this first

Before reading this topic, see:

- "Configure and manage the indexer cluster with the CLI". This topic explains the basics of indexer cluster configuration with the CLI. It provides details on issues that are common to all cluster node types.

Enable the manager node

The following example shows the basic settings that you typically configure when enabling a manager node. The configuration attributes correspond to fields on the **Enable clustering** page of Splunk Web.

```
splunk edit cluster-config -mode manager -replication_factor 4 -search_factor 3 -secret your_key
-cluster_label cluster1

splunk restart
```

The -secret flag modifies the pass4SymmKey setting in the [clustering] stanza of server.conf.

When the manager node starts up for the first time, it will block indexing on the peers until you enable and restart the full replication factor number of peers. Do not restart the manager while it is waiting for the peers to join the cluster. If you do, you will need to restart the peers a second time.

Edit the manager node settings

You can also use the CLI to edit the configuration later. Use the `splunk edit cluster-config` command, the same command used to enable the manager initially.

Refer to the CLI clustering help, along with the server.conf specification file, for the list of configurable settings.

Warning: Do not increase the replication factor or search factor on the manager node

Although it is possible to change the settings for the replication factor and search factor, it is inadvisable to increase either of them after your cluster contains significant amounts of data. Doing so will kick off a great deal of bucket activity, which will have an adverse effect on the cluster's performance while bucket copies are being created or made searchable.

Replace the manager node on the indexer cluster

You might need to replace the manager node for either of these reasons:

- The node fails.
- You must move the manager to a different machine or site.

The best practice for anticipating failover needs is through the cluster manager redundancy feature, described in Implement cluster manager redundancy.

You can also use the more manual method described in this topic, which involves configuring a standby manager that you can immediately bring up if the active manager goes down. You can use the same method to replace the manager intentionally.

This topic describes the key steps in replacing the manager:

1. Back up the files that the replacement manager needs.

   This is a preparatory step. You must do this before the manager fails or otherwise leaves the system.
2. Ensure that the peer and search head nodes can find the new manager.
3. Replace the manager.

In the case of a multisite cluster, you must also prepare for the possible failure of the site that houses the manager. See Handle manager site failure.

Back up the files that the replacement manager needs

There are several files and directories that you must backup so that you can later copy them to the replacement manager:

- The manager's `server.conf` file, which is where the manager cluster settings are stored. You must back up this file whenever you change the manager's cluster configuration.
- The manager's `$SPLUNK_HOME/etc/manager-apps` directory, which is where common peer configurations are stored, as described in Update cluster peer configurations. You must back up this directory whenever you update the set of content that you push to the peer nodes.
- The manager's `$SPLUNK_HOME/var/run/splunk/cluster/remote-bundle/` directory, which contains the actual configuration bundles pushed to the peer nodes. You must back up this directory whenever you push new content to the peer nodes.

If the $SPLUNK_HOME/var/run/splunk/cluster/remote-bundle/ directory contains a large number of old bundles, you can optionally back up only the files associated with the active and previously active bundles. Look for the two files ending with .bundle_active and .bundle_previousActive. Each of those files has has an associated directory and a file that are each identified by the bundle id. You must back up all six files/directories in total. For example, If the directory contains the file 42af6d880c6a1d43e935e8d8a0062089-1571637961.bundle_active, it will also contain the file 42af6d880c6a1d43e935e8d8a0062089-1571637961.bundle and the directory 42af6d880c6a1d43e935e8d8a0062089-1571637961. To back up the active bundle, you must back up the two files and the directory. Similarly, to back up the previously active bundle, you must back up the file that ends with .bundle_previousActive, as well as the directory and other file with the same id.

In addition to the above files and directories, back up any other configuration files that you have customized on the manager, such as `inputs.conf`, `web.conf`, and so on.

In preparing a replacement manager, you must copy over only these files and directories. You do not copy or otherwise deal with the dynamic state of the cluster. The cluster peers as a group hold all information about the dynamic state of a cluster, such as the status of all bucket copies. They communicate this information to the manager node as necessary, for example, when a downed manager returns to the cluster or when a standby manager replaces a downed manager. The manager then uses that information to rebuild its map of the cluster's dynamic state.

Ensure that the peer and search head nodes can find the new manager

You can choose between two approaches for ensuring that the peer nodes and search head can locate the replacement instance and recognize it as the manager:

- **The replacement uses the same IP address and management port as the primary manager.** To ensure that the replacement uses the same IP address, you must employ DNS-based failover, a load balancer, or some other technique. The management port is set during installation, but you can change it by editing `web.conf`.
- **The replacement does not use the same IP address or management port as the primary manager.** In this case, after you bring up the new manager, you must update the `manager_uri` setting on all the peers and search heads to point to the new manager's IP address and management port.

Replace the manager

Prerequisite

You must have up-to-date backups of the set of files and directories described in Back up the files that the replacement manager needs.

Steps

If you want to skip steps 3 and 5, you can replace the [general] and [clustering] stanzas on the replacement manager in step 4, instead of copying the entire server.conf file.

1. Stop the old manager, if this is a planned replacement. If the replacement is due to a failed manager, then this step has already been accomplished for you.
2. Install, start, and stop a new Splunk Enterprise instance. Alternatively, you can reuse an existing instance that is not needed for another purpose. This will be the replacement manager.
3. Copy the `sslPassword` setting from the replacement manager's `server.conf` file to a temporary location.

   In release 6.5, the sslKeysfilePassword attribute was deprecated and replaced by the sslPassword attribute. If the server.conf file is using sslKeysfilePassword, then copy that setting instead.
4. Copy the backup of the old manager's `server.conf` file to the replacement manager.
5. Delete the `sslPassword` setting in the copied `server.conf`, and replace it with the version of the setting that you saved in step 3.
6. Delete the encrypted value for pass4symmkey in the copied `server.conf`, and replace it with the plain text value. See Configure the security key.
7. Copy the backup of the old manager's `$SPLUNK_HOME/etc/manager-apps` directory to the replacement manager.
8. Copy the backup of the old manager's `$SPLUNK_HOME/var/run/splunk/cluster/remote-bundle/` directory to the new manager.
9. Start the replacement manager.
10. Make sure that the peer and search head nodes are pointing to the new manager through one of the methods described in Ensure that the peer and search head nodes can find the new manager.

For information on the consequences of a manager failing, see What happens when the manager node goes down.

Implement cluster manager redundancy

Implementation of an active/standby cluster manager topology requires navigating a sometimes complex and difficult set of procedures customized to achieve your goals in your environment. Involve Splunk Professional Services to ensure that your deployment is successful.

To achieve cluster manager high availability, you can deploy two or more cluster managers in an active/standby configuration. You can configure the managers to support either automatic or manual failover.

The active and standby cluster managers continuously sync the state of the cluster among themselves. This activity ensures that all cluster managers, whether active or standby, have the same configuration bundle, search generation, and peer list, thus ensuring a smooth transition to a new cluster manager when the need arises.

Configuration changes made to the cluster managers themselves (for example, in the cluster managers' copies of server.conf) are not synced automatically between managers. You must make such configuration changes directly on each cluster manager, or by employing some third-party tool that can push the changes to the set of managers.

During automatic failover, the cluster nodes that connect to the manager - peer nodes, search heads, and forwarders (when configured for indexer discovery) - must switch to the new active cluster manager. To support automatic switchover to a new active cluster manager, you can deploy a third-party load balancer between the cluster managers and the peer nodes, search heads, and forwarders.

Similarly, during manual failover, the cluster nodes must switch to the new active manager. To support switchover for a manual failover deployment, you can use DNS mapping.

You can also use DNS mapping with automatic failover. See Use DNS mapping to support cluster manager redundancy.

Deploying a cluster for active/standby high availability thus requires configuration in three areas:

- Multiple cluster managers
- Peer nodes, search heads, and forwarders (if indexer discovery is enabled)
- Third-party load balancer or DNS mapping

System requirements

You must deploy at least two cluster managers. Each manager must reside on its own machine or virtual machine.

Only the cluster managers need to be running a version of Splunk Enterprise that supports cluster manager redundancy. The other cluster nodes can run earlier versions, subject to the general indexer cluster version compatibility requirements, described here: Splunk Enterprise version compatibility.

The set of requirements for third-party load balancers is described separately.

Use a load balancer to support cluster manager redundancy

The third-party load balancer sits in front of the cluster managers and directs traffic from the nodes that talk to the cluster manager -- that is, the peer nodes, search heads, and forwarders, if enabled through indexer discovery. The load balancer ensures that traffic goes only to the currently active cluster manager. If the active manager goes offline or switches to standby, the load balancer redirects traffic to the newly active manager.

The load balancer solution is typically employed with managers configured to use the automatic switchover mode.

Load balancer requirements

A number of third-party load balancers can be used for this purpose, assuming they meet the following requirements:

- They must support the REST-based health check API, described in the *REST API Reference Manual*: cluster/manager/ha_active_status.
- They can forward traffic only to the cluster manager that responds 200 to the health check API.
- They must provide an "always up" service.

How the load balancer directs traffic

You configure the cluster nodes to connect through the load balancer IP address or hostname, rather than through the IP address or hostname of the cluster manager itself.

The load balancer handles only traffic between the active cluster manager and the cluster nodes. The active and standby cluster managers sync among themselves, without the involvement of the load balancer. The standby managers directly monitor the availability of the active cluster manager and perform an automatic switchover if the active manager fails.

To determine the active cluster manager, the load balancer periodically sends out a health check probe. The active manager responds with 200; the standby managers respond with 503. Based on the health check results, the load balancer determines which manager should receive traffic from the nodes.

Load balancers in a multisite indexer cluster

Multisite indexer clusters use a single active cluster manager that services all sites. For an overview of multisite clusters, see Multisite indexer cluster architecture.

With cluster manager redundancy, the cluster managers are deployed across at least two sites. Only one cluster manager is active at any time; the other managers are in standby mode.

Each site has its own load balancer. The load balancers can be deployed in a variety of topologies to support your environment and goals. These are some examples:

- Each load balancer is on a separate subnet and has its own IP address. You can optionally use DNS to unify the `manager_uri` configuration across all peer nodes on all sites. The disadvantage of this method is that peer nodes on one site can lose access to the cluster manager if that site's load balancer goes down.

- The sites employ an extended L2 network. The load balancers have IP addresses in the same subnet by means of the extended L2 network. The load balancers use a next hop redundancy protocol to negotiate for the primary load balancer. All peer nodes across all sites send traffic through the primary load balancer. The disadvantage of this method is that performance can be affected by reliance on a single load balancer.

- The load balancers are deployed in a high availability configuration. Each site has its own load balancer, with its own subnet and IP address. The load balancers must sync status for the active cluster manager among themselves, ensuring that all load balancers send their traffic to the active manager. A DNS record on each site includes IP addresses for all of the load balancers, with preference for the local load balancer. This is the most robust solution.

Use DNS mapping to support cluster manager redundancy

With the DNS-based solution, a DNS record manages the connection between the active cluster manager and the nodes. Nodes connect to the cluster manager indirectly, through the DNS record.

The DNS-based solution is typically employed with managers configured to use the manual switchover mode. You can also use DNS with auto switchover mode, but you still must manually update the DNS record.

You need to have external monitoring in place to detect the loss of the active cluster manager:

- If the managers are configured for manual switchover, when a loss is detected, you must manually switch one of the standby managers to active status. You must also update the DNS record to point to the new active cluster

manager.
- If the managers are configured for automatic switchover, when a loss is detected, the system automatically chooses a new active manager, but you must still update the DNS record to point to the new active cluster manager.

You can create a script that detects the loss of the active manager, switches to a new active manager (if using manual switchover mode), and updates the DNS record.

Configure cluster manager redundancy

You must configure the cluster managers, as well as the other cluster nodes (peer nodes, search heads, and forwarders, if indexer discovery is enabled).

Configure the cluster managers

Configure these `server.conf` settings identically on all cluster managers:

- `manager_switchover_mode`. This setting must be set to either "auto", for automatic failover, or "manual", for manual failover:
  - If set to "auto", the managers automatically adjust modes when the need arises. For example, if the active manager goes down, one of the standby managers switches automatically to active mode.
  - If set to "manual", you must manually switch one of the standby managers to active if the current active manager goes down.
  - The default value is "disabled", which means no cluster manager redundancy.
- `manager_uri`. In this context, `manager_uri` is a prioritized list of all active and standby cluster managers. When the switchover mode is set to auto, the configured priority determines which manager becomes active if the current active manager goes down. In both auto and manual modes, the priority also determines which manager gets set to active when multiple cluster managers are starting at the same time.
- `[clustermanager:<cm-nameX>]`. Multiple instances of this stanza identify each cluster manager's URI. Each cluster manager's `server.conf` file must include the set of stanzas for all cluster managers, including itself.
- `pass4SymmKey`. This setting is required for any indexer cluster node. When implementing redundant cluster managers, ensure that the setting is identical across all cluster managers.

In this example, the cluster has three cluster managers. Each manager includes the following group of settings in its `server.conf` file. The settings must be identical on each manager.

```
[clustering]
mode = manager
manager_switchover_mode = auto
manager_uri = clustermanager:cm1,clustermanager:cm2,clustermanager:cm3
pass4SymmKey = changeme

[clustermanager:cm1]
manager_uri = https://10.16.88.3:8089

[clustermanager:cm2]
manager_uri = https://10.16.88.4:8089

[clustermanager:cm3]
manager_uri = https://10.16.88.5:8089
```

The order specified by `manager_uri` indicates the priority of the cluster manager. In this example, cm1, if available upon cluster startup, will be the active cluster manager. If cm1 fails, cm2 takes over as active, with cm3 switching to active only if both cm1 and cm2 are unavailable.

Note the following points regarding `manager_uri` prioritization when a cluster manager starts:

- If there is already an active cluster manager present (irrespective of its priority), the starting cluster manager will take the standby role.
- If other cluster managers are starting simultaneously and are of higher priority, the lower priority starting manager will follow the priority list and take the standby role.
- If other cluster managers are starting simultaneously and are of lower priority, the higher priority starting cluster manager will take the active role.

Place the active cluster manager into maintenance mode before changing its settings.

You must restart the cluster manager for the settings to take effect.

> Configuration changes to the cluster managers are not synced automatically between managers. You must make such configuration changes directly on each cluster manager, or by employing some third-party tool that can push the changes to the set of managers.

Configure peer nodes, search heads, and forwarders

Peer nodes and search heads both use the setting `manager_uri` in the `[clustering]` stanza of their `server.conf` files to identify the cluster manager. In the case of redundant cluster managers, the nodes must reference either the load balancer or DNS record, rather than the cluster manager itself. The load balancer or DNS record then redirects the node traffic to the active manager.

For example, on each peer node or search head, update the `manager_uri` setting in `server.conf`, like this:

```
[clustering]
manager_uri = https://<LB-IP-OR-DNS-HOSTNAME>
```

To avoid a restart of the peer nodes, implement the change on each peer via REST, rather than through the configuration bundle method. For example:

```
curl -k -v -u admin:changeme https://<IDX IP>:8089/services/cluster/config/clusterconfig -d
'manager_uri=https://<LB-IP-OR-DNS-HOSTNAME>:8089'
```

If a search head is searching across multiple indexer clusters, make the appropriate changes within all of their `[clustermanager]` stanzas that reference a cluster employing cluster manager redundancy. See Search across multiple indexer clusters for general information on configuring search heads to search across multiple clusters.

Forwarders enabled for indexer discovery use `manager_uri` in the `[indexer_discovery:<name>]` stanza of their `output.conf` files to identify the cluster manager. In the case of redundant cluster managers, the forwarders must reference either the load balancer or DNS record:

```
[indexer_discovery:<name>]
manager_uri = https://LB-OR-DNS:8089
```

These changes all require a restart of the peer nodes, search heads, and forwarders.

Deployment considerations

When deploying or updating your cluster managers, here are some guidelines:

- Update the nodes in your deployment in this order:

1. Cluster managers: current active manager, followed by new active manager, followed by any remaining standby managers
2. Load balancer or DNS record
3. Peer nodes
4. Search heads and forwarders

- For an existing deployment, put the current active manager in maintenance mode and switch it to standby before performing other updates to its configuration.

Manage cluster manager redundancy

Use the CLI

Run the following command from any cluster manager to view the cluster manager redundancy status for all managers in the cluster:

```
splunk cluster-manager-redundancy -show-status -auth <user:passwd>
```

To change the active cluster manager, run the following command on the manager that you want to switch to active mode:

```
splunk cluster-manager-redundancy -switch-mode active
```

The formerly active manager will automatically restart in standby mode. The newly active manager does not require a restart.

Use the REST API

To view cluster manager redundancy status for all managers in the cluster, initiate a GET for this endpoint from any of the managers:

```
services/cluster/manager/redundancy/
```

To change the active cluster manager, initiate a POST for this endpoint on the manager that you want to switch to active mode:

```
services/cluster/manager/redundancy/
/services/cluster/manager/redundancy/ -d "_action=switch_mode" -d "ha_mode=Active"
```

The formerly active manager will automatically restart in standby mode. The newly active manager does not require a restart.

Use the manager node dashboard

You can use the manager node dashboard to manage cluster manager redundancy. The dashboard is available on active and standby managers through **Settings > Indexer Clustering**, as described in Configure the manager node with the dashboard.

If the cluster manager is in active mode, you will see the usual set of tabs for peers, indexes, and search heads, along with a new tab, "Cluster Manager". Click on it, and you'll see a table with a row for each cluster manager, indicating its HA mode (active/standby), as well as some other basic information.

The "Edit" button is available with the usual set of capabilities, along with a "Switch HA Mode" button to switch the manager to standby. When you switch a manager from active to standby, it restarts automatically. If the managers' switchover mode is set to active, the standby manager with the highest priority will then automatically switch to active.

If the cluster manager is in standby mode, you will see a statement at the top of the dashboard, "This cluster is in standby mode". Instead of the full set of tabs that you see for an active manager, the standby manager's dashboard contains only a table with rows for each of the cluster managers. The "Edit" button is disabled, but the "Switch HA Mode" button is available to switch the manager to active. When a standby manager becomes the active manager, the formerly active manager will then restart in standby mode.

Update the peer nodes' configuration bundle

The process of updating peer nodes works the same as with non-redundant clusters, aside from a few minor issues. For a general discussion of updating peer nodes, see Manage common configurations across all peers.

To add or update the **configuration bundle** for the peer nodes, place the changes in the active cluster manager's manager-apps directory. To distribute the bundle to the peer nodes, apply the bundle as usual from the active manager.

You must put the configuration bundle updates in the manager-apps directory on the active cluster manager. The updates will not be synchronized to the standby cluster managers until they have been applied and distributed to the peer nodes via the active manager, and the new bundle becomes the active bundle for the cluster.

> Do not switch the active cluster manager until any pending configuration bundle changes have been successfully applied to the peer nodes.

Configure the peers

Peer node configuration overview

Configuration of the peer nodes falls under two categories:

- Configuration of the basic indexer cluster settings, such as the manager URI and the replication port.
- Configuration of input, indexing, and related settings. This includes the deployment of apps to the peer nodes.

Initial configuration

Most peer cluster configuration happens during initial deployment:

1. When you enable the peer, you specify its cluster settings, such as its manager node and the port on which it receives replicated data. See Enable the peer nodes.

2. After you enable the set of peers, you configure their indexes, if necessary. See Configure the peer indexes in an indexer cluster.

3. Finally, you configure their inputs, usually by means of forwarders. See Use forwarders to get data into the indexer cluster.

These are the key steps in configuring a peer. You might also need to update the configurations later, as with any indexer.

Change the cluster configuration

There are two main reasons to change the cluster node configuration:

- **Redirect the peer to another manager.** This can be useful in the case where the manager node fails but you have a stand-by manager ready to take over. For information on stand-by managers, see Replace the manager node on the indexer cluster.
- **Change the peer's security key for the cluster.** Only change the key if you are also changing it for all other nodes in the cluster. The key must be the same across all instances in a cluster.

To edit the cluster configuration, change each peer node individually, using one of these methods:

- Edit the configuration from the peer node dashboard in Splunk Web. See Configure peer nodes with the dashboard.
- Edit the peer's `server.conf` file. See Configure peer nodes with server.conf for details.
- Use the CLI. See Configure peer nodes with the CLI for details.

For additions and differences when configuring multisite peer nodes, see Configure multisite indexer clusters with server.conf.

Configure indexing and related behavior

The set of index stanzas in `indexes.conf` must be identical across all peers, aside from very limited exceptions described in Manage configurations on a peer-by-peer basis. It is also important that index-time processing be the same across the peers. For the cluster to properly replicate data and handle node failover, peers must share the same indexing functionality, and they cannot do this if certain key files vary from peer to peer.

As a best practice, you should treat your peers as interchangeable, and therefore you should maintain identical versions of configuration files and apps across all peers. At the least, the following files should be identical:

- `indexes.conf`
- `props.conf`
- `transforms.conf`

To ensure that the peers share a common set of configuration files and apps, place the files and apps on the manager node and then use the **configuration bundle** method to distribute them, in a single operation, to the set of peers.

These topics describe how to maintain identical configurations across the set of peers:

- Manage common configurations across all peers
- Manage app deployment across all peers
- Configure the peer indexes in an indexer cluster
- Update common peer configurations and apps

Manage single-peer configurations

You might occasionally need to handle some configurations on a peer-by-peer basis, for testing or other purposes. As a general rule, however, try to use the same configurations across all peers, so that the peers are interchangeable.

For information on single-peer configuration, see Manage configurations on a peer-by-peer basis.

Configure peer nodes with the dashboard

You can edit the configuration of an existing peer node through its dashboard. To access the dashboard:

1. Click **Settings** on the upper right side of Splunk Web.

2. In the **Distributed Environment** group, click **Indexer clustering**.

3. Select the **Edit** button on the upper right side of the dashboard.

The **Edit** button provides a number of options that affect the configuration.

Note: The **Edit** button is disabled for multisite clusters.

For information on using this dashboard to enable a peer node initially, see Enable the peer nodes.

For information on using this dashboard to view cluster status, see View the peer dashboard.

Change cluster configuration

To change the configuration for this peer node, select the **Configuration** option:

- To change the manager node, edit the **Manager URI** field.
- To change the replication port, edit the **Peer replication port** field.
- To change the security key, edit the **Security key** field.

Remove the peer from a cluster

To remove the peer from the cluster, select the **Disable Indexer Clustering** option.

Other edits

The **Edit** button presents one other option: **Node Type**.

It is extremely unlikely that you will want to change the node type for nodes in an active cluster. Consider the consequences carefully before doing so.

Configure peer nodes with server.conf

Prerequisites

Before reading this topic, see Configure the indexer cluster with server.conf. It discusses configuration issues that are common to all cluster node types.

Enable a peer node

The following example shows the basic settings that you must configure when enabling a peer node. The configuration attributes shown in these examples correspond to fields on the **Enable clustering** page of Splunk Web.

```
[replication_port://9887]

[clustering]
manager_uri = https://10.152.31.202:8089
mode = peer
pass4SymmKey = whatever
```

This example specifies that:

- the peer will use port 9887 to listen for replicated data streamed from the other peers. You can specify any available, unused port as the replication port. Do not re-use the management or receiving ports.
- the peer's manager node resides at `10.152.31.202:8089`.
- the instance is a peer node.
- the security key is "whatever".

You must restart the instance for the settings to take effect.

Edit the peer settings

You can change these settings later, if necessary. For example, to change the cluster's security key, you change the `pass4SymmKey` value on each node.

Configure peer nodes with the CLI

Read this first

Before reading this topic, see:

- Configure and manage the indexer cluster with the CLI. This topic explains the basics of cluster configuration with the CLI. It provides details on issues that are common to all indexer cluster node types.

Enable a peer node

The following example shows the basic settings that you typically configure when enabling a peer node. The configuration attributes correspond to fields on the **Enable clustering** page of Splunk Web.

To enable an instance as a peer node, set `mode` to "peer". You also need to specify `manager_uri`, which specifies the cluster's manager node, and `replication_port`. In addition, you must specify the cluster-wide security key (`secret`):

```
splunk edit cluster-config -mode peer -manager_uri https://10.160.31.200:8089 -replication_port 9887 -secret your_key

splunk restart
```

The -secret flag modifies the pass4SymmKey setting in the [clustering] stanza of server.conf.

Edit the peer settings

You can also use the CLI to edit the configuration later. Use the `splunk edit cluster-config` command, the same command that you use to enable the peer initially.

Refer to the CLI clustering help, along with the server.conf specification file, for the list of configurable settings.

Manage common configurations across all peers

You should attempt to maintain a common set of configuration files, including apps, across all peers in an indexer cluster. This enhances high availability by making the peers essentially interchangeable. In addition, certain configurations must be identical so that all the peers index the data in the same way.

The manager node distributes files and apps to all peers as a single action, through the **configuration bundle** method. You must use this method to manage common configurations. See Update common peer configurations and apps.

Files that need to be identical across all peers

It is highly recommended that you distribute the same versions of these files across all peers:

- `indexes.conf`. It is critical that all peers share the same set of clustered indexes.
- `props.conf` and `transforms.conf`. All peers must use the same set of rules when indexing data.

Beyond these three key files, you can greatly simplify cluster management by maintaining identical versions of other configuration files across all peers. For example, if your peers are able to share a single set of inputs, you can maintain a single `inputs.conf` file across all peers.

Because apps often contain versions of those configuration files, you should ordinarily distribute the same set of apps to all peers, rather than installing them individually on single peers. See Manage app deployment across all peers.

Note: Under limited circumstances (for example, to perform local testing or monitoring), you might want to add an index to one peer but not the others. You can do this by creating a single-peer `indexes.conf`, as long as you are careful about how you configure the index and are clear about the ramifications. The data in such an index will not get replicated. The single-peer `indexes.conf` supplements, but does not replace, the common version of the file that all peers get. You can similarly maintain single-peer apps, if necessary. See Add an index to a single peer.

Distribute configuration files to all peers

To distribute configurations across the peer nodes:

1. If distributing any `indexes.conf` files, configure them so that they support index replication. See Configure the peer indexes in an indexer cluster.

2. Place the files in the `$SPLUNK_HOME/etc/manager-apps` directory on the manager node. The set of subdirectories in this location constitute the configuration bundle.

3. Use Splunk Web or the CLI to distribute the configuration bundle to the peer nodes.

For details on these steps, see Update common peer configurations and apps.

Configuration management for peers compared to standalone indexers

The configuration bundle method is the only supported method for managing common configurations and app deployment across the set of peers. It ensures that all peers use the same versions of these files.

Note these critical differences in how you manage peer configuration files compared to configurations for standalone indexers:

- Do not make configuration changes on individual peers that will modify configurations you need to maintain on a cluster-wide basis. For example, do not use Splunk Web or the CLI to configure index settings.
- Do not edit cluster-wide configuration files, like `indexes.conf`, directly on the peers. Instead, edit the files on the manager node and distribute them through the configuration bundle method.
- Do not use deployment server or any third party deployment tool, such as Puppet or CFEngine, to manage common configuration files across peer nodes. Instead, use the configuration bundle method.

When you distribute updates through the configuration bundle, the manager node orchestrates the distribution to ensure that all peers use the same set of configurations, including the same set of clustered indexes.

If, despite all recommendations, you choose to use another distribution method instead of the configuration bundle method, you must make sure, at a minimum, that settings for any new clustered indexes are successfully distributed to all peers, and that all the peers have been reloaded, before you start sending data to the new indexes.

Note: Although you cannot use deployment server to directly distribute apps to the peers, you can use it to distribute apps to the manager node's configuration bundle location. Once the apps are in that location, the manager node can then distribute them to the peer nodes via the configuration bundle method. See Use deployment server to distribute the apps to the manager.

Manage app deployment across all peers

Before reading this topic, see Manage common configurations across all peers. App deployment is just a special case of the configuration file deployment described in that topic.

> You must use the manager node to deploy apps to the peer nodes. Do not use deployment server or any third party deployment tool, such as Puppet or CFEngine.

Distribute an app to the peer nodes

To distribute an app across the peer nodes:

1. Inspect the app for `indexes.conf` files. For each index defined in an app-specific `indexes.conf` file, set `repFactor=auto`, so that the index gets replicated across all peers. See The indexes.conf repFactor attribute.
2. Place the app in the `$SPLUNK_HOME/etc/manager-apps` directory on the manager node. The set of subdirectories in this location constitute the **configuration bundle**.
3. Use Splunk Web or the CLI to distribute the configuration bundle to the peer nodes.

   For detailed information on each of these steps, see Update common peer configurations and apps.

Once an app has been distributed to the set of peers, you launch it on each peer in the usual manner, with Splunk Web. See the chapter Meet Splunk apps in the *Admin Manual*.

> When it comes time to access an app, you do so from the search head, not from an individual peer. Therefore, you must also install the app on the search head. On the search head, put the app in the conventional location for apps, that is, under the $SPLUNK_HOME/etc/apps directory. Install the app in the usual fashion, according to the app-specific instructions.

Delete an app from the peer nodes

To delete an app that you previously distributed to the peers, remove its directory from the configuration bundle. When you next push the bundle, the app will be deleted from each peer.

Configure the peer indexes in an indexer cluster

You configure indexes by editing the indexes.conf file. This file determines an indexer's set of indexes, as well as the size and attributes of its **buckets**. Since all peers in a cluster must use the same set of indexes (except for limited purposes, described later), the `indexes.conf` file should ordinarily be the same across all peers.

The cluster peers deploy with a peer-specific default `indexes.conf` file that handles basic cluster needs. If you want to add indexes or change bucket behavior, you edit a new `indexes.conf` file in a special location on the manager node and then distribute the file simultaneously to all the peers.

Important: You cannot use Splunk Web or the CLI to configure index settings on peer nodes. You must edit `indexes.conf` directly.

All peers must use the same set of indexes.conf files

The set of `indexes.conf` files should ordinarily be identical across all peers in a cluster. In particular, all peers must use the same set of clustered indexes. This is essential for index replication to work properly. (The manager node, on the other hand, has its own, separate `indexes.conf` file, because it indexes only its own internal data.) There is a limited exception to this restriction, which is described a bit later.

When you first create the cluster, the manager node distributes a special default `indexes.conf` file to each of the peers. This version supplements the standard default `indexes.conf` that all indexers get. The peer-specific default `indexes.conf` turns on replication for the `main` index, as well as the internal indexes, such as `_audit` and `_internal`.

Depending on your system, you might also need to edit and distribute a modified `indexes.conf` to the peers, to accommodate additional indexes or changes to bucket attributes. To ensure that all peers use the same `indexes.conf`, you must use the manager node to distribute the file to all the peers as a single process.This process, known as the **configuration bundle** method, is described in Update common peer configurations and apps.

You must also use the configuration bundle method to distribute apps across all the peers. These apps might contain their own `indexes.conf` files, which will layer appropriately with any non-app version of the file that you might also distribute to the peers. For information on app distribution, read Manage app deployment across all peers.

Note: Under limited circumstances (for example, to perform local testing or monitoring), you can create an `indexes.conf` for a single peer only. Such an index will not get replicated. The single-peer `indexes.conf` supplements, but does not replace, the common version of the file that all peers get. See Add an index to a single peer for details.

Configure a set of indexes for the peers

There are two steps to configuring indexes across the set of peers:

1. Edit a common `indexes.conf` file on the manager node.

2. Use the manager node to distribute the file across the set of peers.

These two steps are described below.

1. Edit indexes.conf

For details on configuring `indexes.conf`, read the topics in the chapters Manage indexes and Manage index storage in this manual. For a list of all `indexes.conf` attributes, see the indexes.conf specification file in the *Admin Manual*.

For the most part, you edit the cluster peer `indexes.conf` in the same way as for any indexer. However, there are a few differences to be aware of.

The indexes.conf repFactor attribute

When you add a new index stanza, you must set the `repFactor` attribute to "auto". This causes the index's data to be replicated to other peers in the cluster. For example:

```
[idx1]
repFactor = auto
homePath = $SPLUNK_DB/$_index_name/db
coldPath = $SPLUNK_DB/$_index_name/colddb
thawedPath = <path for thawed buckets>
...
```

Note: By default, `repFactor` is set to 0, which means that the index will not be replicated. For clustered indexes, you must set it to "auto".

The only valid values for `repFactor` are 0 and "auto".

Resetting `repFactor` from "auto" to 0 will stop further replication, but it will not automatically remove copies of already replicated buckets. In addition, searches across buckets with multiple copies will return duplicate events. To free up associated disk space and eliminate the possibility of duplicate events, you must remove the excess copies manually.

Specify the index path attributes with forward-slash directory separators

In heterogeneous environments, it is possible that the manager node's operating system could use a different convention for specifying directory paths from the peer nodes' operating system. This presents a problem because you edit the `indexes.conf` file on the manager node but then you distribute it to the peers.

For example, if you have a Windows manager node and a set of Linux peers, the normal way to specify the `homePath` on the Windows manager node, where the file gets edited, would be to use the Windows backward-slash convention as a directory separator, while the Linux peers, where the file gets distributed, require forward slashes.

To deal with this possibility, the best practice is to always use forward slashes when specifying directory paths in in the index path attributes, no matter which operating systems your manager and peers use. For example:

```
homePath = $SPLUNK_DB/$_index_name/db
```

Splunk Enterprise always accepts the forward slash as a directory separator.

2. Distribute the new indexes.conf file to the peers

After you edit `indexes.conf`, you need to distribute it to the cluster's set of peer nodes. To learn how to distribute configuration files, including `indexes.conf`, across all the peers, read Update common peer configurations and apps.

For information about other types of peer configuration, including app distribution, read Peer node configuration overview.

View the indexes

To see the set of indexes on your peer nodes, click the Indexes tab on the manager node dashboard. See View the manager node dashboard.

Note: A new index appears under the tab only after it contains some data. In other words, if you configure a new index on the peer nodes, a row for that index appears only after you send data to that index.

Update common peer configurations and apps

The peer update process described in this topic ensures that all peer nodes share a common set of key configuration files. You must manually invoke this process to distribute and update common files, including apps, to the peer nodes. The process also runs automatically when a peer joins the cluster.

For information on peer configuration files, see Manage common configurations across all peers. That topic details exactly which files must be identical across all peers. In brief, the configuration files that must be identical in most circumstances are `indexes.conf`, `props.conf`, and `transforms.conf`. Other configuration files can also be identical, depending on the needs of your system. Since apps usually include versions of those key files, you should also maintain a common set of apps across all peers.

The set of configuration files and apps common to all peers, which is managed from the manager node and distributed to the peers in a single operation, is called the **configuration bundle**. The process used to distribute the configuration bundle is known as the configuration bundle method.

To distribute new or edited configuration files or apps across all the peers, you add the files to the configuration bundle on the manager node and tell the manager to distribute the files to the peers.

Structure of the configuration bundle

The configuration bundle consists of the set of files and apps common to all peer nodes.

On the manager node

On the manager, the configuration bundle resides under the `$SPLUNK_HOME/etc/manager-apps` directory. The set of files under that directory constitute the configuration bundle. They are always distributed as a group to all the peers. The directory has this structure:

```
$SPLUNK_HOME/etc/manager-apps/
     _cluster/
          default/
          local/
     <app-name>/
     <app-name>/
     ...
```

Prior to Splunk Enterprise version 9.0, the cluster manager used the master-apps directory as the configuration bundle repository. Starting with 9.0, a new directory called manager-apps was added to the cluster manager as a replacement for master-apps. Although master-apps is deprecated, you can continue to use it, rather than switching to manager-apps. If you choose to do so, substitute master-apps for all references to manager-apps in this manual. This issue is discussed in detail elsewhere in this topic.

Note the following:

- The `/_cluster` directory is a special location for configuration files that need to be distributed across all peers:
  - The `/_cluster/default` subdirectory contains a default version of `indexes.conf`. Do not add any files to

this directory and do not change any files in it. This peer-specific default `indexes.conf` has a higher precedence than the standard default `indexes.conf`, located under `$SPLUNK_HOME/etc/system/default`.
    - The `/_cluster/local` subdirectory is where you can put new or edited configuration files that you want to distribute to the peers.
- The `/<app-name>` subdirectories are optional. They provide a way to distribute any app to the peer nodes. Create and populate them as needed. For example, to distribute "appBestEver" to the peer nodes, place a copy of that app in its own subdirectory: `$SPLUNK_HOME/etc/manager-apps/appBestEver`.
- To delete an app that you previously distributed to the peers, remove its directory from the configuration bundle. When you next push the bundle, the app will be deleted from each peer.
- The manager node only pushes the contents of subdirectories under `manager-apps`. It will not push any standalone files directly under `manager-apps`. For example, it will not push the standalone file `/manager-apps/file1`. Therefore, be sure to place any standalone configuration files in the `/_cluster/local` subdirectory.

You explicitly tell the manager node when you want it to distribute the latest configuration bundle to the peers. In addition, when a peer registers with the manager (for example, when the peer joins the cluster), the manager distributes the current configuration bundle to it.

When the manager distributes the bundle to the peers, it distributes the entire bundle, overwriting the entire contents of any configuration bundle previously distributed to the peers.

The `manager-apps` location is only for peer node files. The manager does not use the files in that directory for its own configuration needs.

Which directory to use: manager-apps or master-apps?

Prior to Splunk Enterprise version 9.0, the cluster manager used the `master-apps` directory as the sole repository for configuration-bundles. In version 9.0, a new repository directory with the name `manager-apps` was added to the cluster manager to replace `master-apps`.

So as not to break any customer-generated scripts or other tooling, the `master-apps` directory continues to exist on upgraded installations and continues to hold any configuration bundle files already existing within that directory. You can use either directory as the repository, but you cannot use both. The cluster manager pushes the bundle from only a single location.

The best practice is to use manager-apps as the configuration bundle repository, because master-apps has been deprecated and will be eliminated in some future release.

Here is how the manager decides which directory to use for the bundle push:

- If you have user-created content in `manager-apps`, the manager pushes the bundle from there. (If there is also content in `master-apps`, the manager ignores that content except to log a warning in `splunkd.log`.)
- If you do not have content in `manager-apps`, the manager pushes the bundle from `master-apps`.

When deciding which directory to use, the manager ignores any files residing under the /_cluster/default subdirectory.

Therefore, when you upgrade your cluster manager from pre-9.0 to 9.0 or later, you must decide which location to use for your configuration bundle:

- If you choose to continue using `master-apps`.as the repository, do not place any files in `manager-apps`.
- If you choose to switch to `manager-apps`, move any existing files from `master-apps` into `manager-apps` and delete those files from `master-apps`.

You can choose to change the bundle repository location from `master-apps` to `manager-apps` either immediately following the upgrade or at some later time. When you change the repository location, be sure to upgrade any scripts, and so on, that reference the previous location.

On the peers

On the peers, the distributed configuration bundle resides under `$SPLUNK_HOME/etc/peer-apps`. This directory is created soon after a peer is enabled, when the peer initially gets the latest bundle from the manager.

Prior to Splunk Enterprise version 9.0, the peer nodes used the slave-apps directory as the configuration bundle repository. Starting with 9.0, peer-apps replaces slave-apps. If your peer node was upgraded from a pre-9.0 version, the slave-apps directory was renamed to peer-apps during the upgrade process.

When you upgrade to 9.0, the slave-apps directory is automatically renamed to peer-apps. As part of the upgrade, you must manually change any external hardcoded references to slave-apps, such as external scripts, SSL certificates, and so on, to peer-apps.

Except for the different name for the top-level directory, the structure and contents of the configuration bundle are the same as on the cluster manager:

```
$SPLUNK_HOME/etc/peer-apps/
     _cluster/
          default/
          local/
     <app-name>/
     <app-name>/
     ...
```

Leave the downloaded files in this location and do not edit them. If you later distribute an updated version of a configuration file or app to the peers, it will overwrite any earlier version in $SPLUNK_HOME/etc/peer-apps. You want this to occur, because all peers in the cluster must be using the same versions of the files in that directory. For the same reason, do not add any files or subdirectories directly to $SPLUNK_HOME/etc/peer-apps. The directory gets overwritten each time the manager redistributes the configuration bundle.

When Splunk software evaluates configuration files, the files in the `$SPLUNK_HOME/etc/peer-apps/[_cluster|<app-name>]/local` subdirectories have the highest precedence. For information on configuration file precedence, see Configuration file precedence in the *Admin Manual*.

Settings that you should not distribute through the configuration bundle

The `$SPLUNK_HOME/etc/peer-apps` directory on the peers is read-only. This is necessary and beneficial behavior, because each time you distribute a new bundle, the directory gets overwritten in its entirety. You would thus otherwise lose any changes made to settings in that directory. Also, the cluster relies on the settings in that directory being identical across all peers.

Therefore, if you distribute a setting through the configuration bundle method that the peer needs to update automatically in some way, the peer will do so by creating a new version of the app under `$SPLUNK_HOME/etc/apps`. Since you cannot have two apps with the same name, this generates "unexpected duplicate app" errors in `splunkd.log`.

A common cause of this behavior is distributing SSL passwords through the configuration bundle. Splunk Enterprise overwrites the password with an encrypted version upon restart. But if you distribute the setting through the configuration bundle, the peers cannot overwrite the unencrypted password in its bundle location under `$SPLUNK_HOME/etc/peer-apps`. Therefore, upon restart after bundle push, they instead write the encrypted version to `$SPLUNK_HOME/etc/apps`, in an app directory with the same name as its name under `$SPLUNK_HOME/etc/peer-apps`.

For example, do not push the following setting in `inputs.conf`:

```
[SSL]
password = <your_password>
```

If the setting is in an app directory called "newapp" in the configuration bundle, upon restart the peer will create a "newapp" directory under `$SPLUNK_HOME/etc/apps` and put the setting there. This results in duplicate "newapp" apps.

Best practice for distributing app.conf in the configuration bundle

If you add a new `reload.<conf_file_name> = simple` parameter to `app.conf` in the configuration bundle, you must first push `app.conf` to the peer nodes, before you push updates to the configuration file referenced in the new parameter. After you push the `app.conf` file, subsequent changes to the referenced configuration file in the app context will not require a peer restart.

For example, if you add `reload.inputs = simple` to `app.conf`, then push `app.conf` to peer nodes, when you next push an update to `inputs.conf` in the specific app context, peer nodes will reload and not require a restart.

To distribute `app.conf` in the configuration bundle:

1. For each app that you want to distribute, determine if `app.conf` contains a new `reload.<conf_file_name> = simple` parameter.
2. Remove any configuration files referenced in `reload.<conf_file_name> = simple` from the configuration bundle.
3. Push the modified configuration bundle containing `app.conf` to the set of peers. See Apply the bundle to the peers.
4. Return the referenced configuration file to the appropriate app directory in the configuration bundle.
5. Make any changes to the configuration file that you want to distribute to peer nodes.
6. Push the configuration bundle to the peers.

For more information on `reload.<conf_file_name> = simple`, see app.conf in the *Admin Manual*.

Distribute the configuration bundle

To distribute new or changed files and apps across all peers, follow these steps:

1. Prepare the files and apps and test them.
2. Move the files and apps into the configuration bundle on the manager node.
3. (Optional) Validate the bundle and check restart.
4. Apply the bundle to the peers.
   The manager pushes the entire bundle to the peers. This overwrites the contents of the peers' current bundle.

1. Prepare the files and apps for the configuration bundle

Make the necessary edits to the files you want to distribute to the peers. It is advisable that you then test the files, along with any apps, on a standalone test indexer to confirm that they are working correctly, before distributing them to the set of peers. Try to combine all updates in a single bundle, to reduce the impact on the work of the peer nodes.

For more information on how to configure the files, see Manage common configurations across all peers and Configure the peer indexes in an indexer cluster.

If the configuration bundle subdirectories contain any indexes.conf files that define new indexes, you must explicitly set each index's repFactor attribute to auto. This is necessary for indexes.conf files that reside in app subdirectories, as well as any indexes.conf file in the _cluster subdirectory.

For more information, see The indexes.conf repFactor attribute.

2. Move the files and apps into the configuration bundle on the manager node

When you are ready to distribute the files and apps, copy them to `$SPLUNK_HOME/etc/manager-apps/` on the manager node:

- Put apps directly under the `manager-apps` directory. For example, `$SPLUNK_HOME/etc/manager-apps/<app-name>`.
- Put standalone files in the `$SPLUNK_HOME/etc/manager-apps/_cluster/local` subdirectory.

3. (Optional) Validate the bundle and check restart

You can validate the bundle and check whether applying the bundle will require a restart of the peer nodes, without applying the bundle. Once you confirm that the bundle is valid across all peer nodes, you can then apply it as a separate step.

Validation is useful for ensuring that the bundle will apply across all peer nodes without problems. The validation process also provides information that is useful for debugging invalid bundles.

The ability to check restart is useful for admins who want to defer bundle distribution until a period of low activity or a maintenance window, and avoid possible restart-related indexing or search interruption. See Restart or reload after configuration bundle changes?

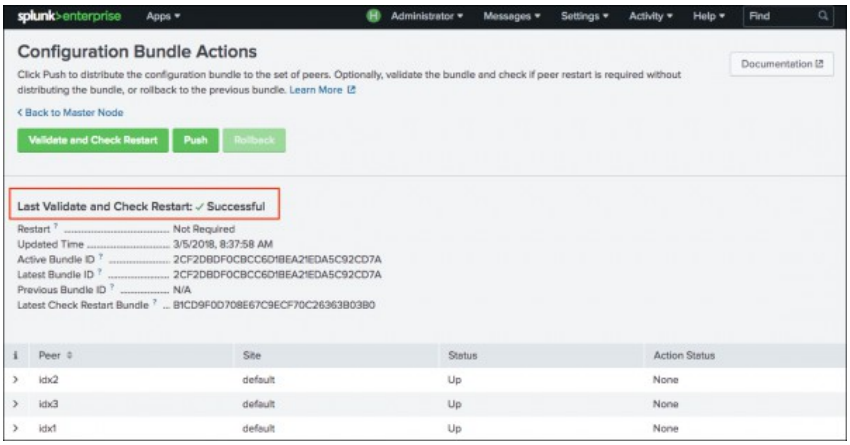
You can validate the bundle and check restart using Splunk Web or the CLI.

Use Splunk Web to validate the bundle and check restart

Use the Validate and Check Restart button to validate the bundle and check if peer nodes will require a restart. This button performs the same function as the CLI command: `splunk validate cluster-bundle --check-restart`. See Use the CLI to validate the bundle and check restart.

To validate the bundle and check restart:

1. On the manager node, in Splunk Web, click **Settings > Indexer Clustering**.
   The Manager Node dashboard opens.
2. Click **Edit > Configuration Bundle Actions**.
3. Click **Validate and Check Restart > Validate and Check Restart**.
   A message appears that indicates bundle validation and check restart success or failure.



If bundle validation and check restart succeeds, then the bundle is acceptable for distribution to the peer nodes. Information about the validated bundle appears in the UI, including whether a restart of peer nodes is required. The UI also displays bundle checksums, which you can use to identify and track the active bundle, the previous bundle, and the latest check restart bundle.

You can distribute the bundle from the manager to the peer nodes using either Splunk Web or the CLI. See 4. Apply the bundle.

If validation and check restart fails, then the bundle is not acceptable for distribution to the peers. In this case, review the bundle details for information that might help you troubleshoot the issue. Make sure that the configuration bundle structure is correct for distribution to peer nodes. See Structure of the configuration bundle.

Use the CLI to validate the bundle and check restart

To validate the bundle only, run `splunk validate cluster-bundle`:

```
splunk validate cluster-bundle
```

This command returns a message confirming that bundle validation has started. In certain failure conditions, it also indicates the cause of failure.

To validate the bundle and check whether a restart is necessary, include the `--check-restart` parameter:

```
splunk validate cluster-bundle --check-restart
```

This version of the command first validates the bundle. If validation succeeds, it then checks whether a peer restart is necessary.

To view the status of bundle validation, run the `splunk show cluster-bundle-status` command:

```
splunk show cluster-bundle-status
```

This command indicates validation success. In the case of validation failure, it provides insight into the cause of failure. It also indicates if peer restart is required.

Here is an example of the output from the `splunk show cluster-bundle-status` command after a successful validation:

```
manager
         cluster_status=None
```

```
         active_bundle
                checksum=576F6BBB187EA6BC99CE0615B1DC151F
                timestamp=1495569737 (in localtime=Tue May 23 13:02:17 2017)
         latest_bundle
                checksum=576F6BBB187EA6BC99CE0615B1DC151F
                timestamp=1495569737 (in localtime=Tue May 23 13:02:17 2017)
         last_validated_bundle
                checksum=1E0C4F0A7363611774E1E65C8B3932CF
                last_validation_succeeded=1
                timestamp=1495574646 (in localtime=Tue May 23 14:24:06 2017)
         last_check_restart_bundle
                 checksum=1E0C4F0A7363611774E1E65C8B3932CF
                 last_check_restart_result=restart required
                 timestamp=1495574646 (in localtime=Tue May 23 14:24:06 2017)

Peer 1         1D00A8C2-026B-4CAF-90D6-5D5D39445569         default
         active_bundle=576F6BBB187EA6BC99CE0615B1DC151F
         latest_bundle=576F6BBB187EA6BC99CE0615B1DC151F
         last_validated_bundle=1E0C4F0A7363611774E1E65C8B3932CF
         last_bundle_validation_status=success
         last_bundle_checked_for_restart=1E0C4F0A7363611774E1E65C8B3932CF
         last_check_restart_result=restart required
         restart_required_apply_bundle=0
         status=Up
...
```

The `last_validated_bundle` identifies the newly validated bundle. Notice that it differs from the `active_bundle`, which identifies the bundle that was most recently applied and is currently active across the peer nodes.

The `last_validation_succeeded=1` field indicates that validation succeeded.

On the manager node, the `last_check_restart_result=restart required` field indicates that a restart is required on at least one of the cluster peers.

On the peers, the `last_check_restart_result=restart required` field indicates that a restart of that individual peer is required.

If you validate the bundle without applying it, the contents of the $SPLUNK_HOME/etc/manager-apps directory on the manager node will differ from the contents of the $SPLUNK_HOME/etc/peer-apps directory on the peer nodes until you do apply the bundle. This has no effect on the operation of the cluster, but it is important to be aware that the difference exists.

4. Apply the bundle to the peers

To apply the configuration bundle to the peers, you can use Splunk Web or the CLI.

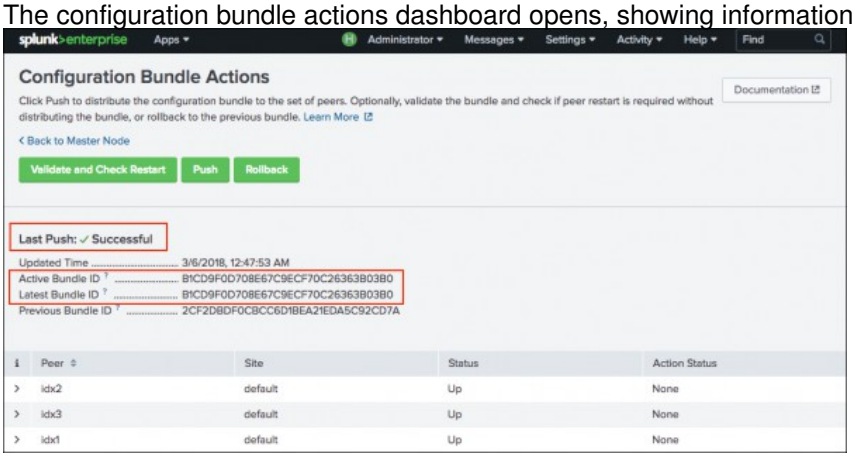
You cannot initiate a configuration bundle push if a bundle push is currently in progress.

Use Splunk Web to apply the bundle

To apply the configuration bundle to the peers:

1. On the manager node, in Splunk Web, click **Settings > Indexer clustering**.
   The Manager Node dashboard appears.

2. Click **Edit > Configuration Bundle Actions**.
   The configuration bundle actions dashboard opens, showing information on the last successful bundle push.



3. Click **Push**.
   A pop-up window warns you that the distribution might, under certain circumstances, initiate a restart of all peer nodes. For information on which configuration changes cause a peer restart, see Restart or reload after configuration bundle changes?.
4. Click **Push Changes**.
   The screen provides information on the distribution progress. Once the distribution completes or aborts, the screen indicates the result.
   - In the case of a successful distribution, after each peer successfully validates the bundle, the manager coordinates a rolling restart of all the peer nodes, if necessary.
   - In the case of an aborted distribution, it indicates which peers could not receive the distribution. Each peer must successfully receive and apply the distribution. If any peer is unsuccessful, none of the peers will apply the bundle.

   When the push is successful, the peers use the new set of configurations, now located in their local `$SPLUNK_HOME/etc/peer-apps`.

> Leave the files in $SPLUNK_HOME/etc/peer-apps.

For more information on the distribution process, see the following section on applying the bundle through the CLI.

Use the CLI to apply the bundle

1. To apply the configuration bundle to the peers, run this CLI command on the manager node:

   ```
   splunk apply cluster-bundle
   ```
   It responds with this warning message:

   ```
   Caution: Under some circumstances, this command will initiate a rolling restart
   of all peers. This depends on the contents of the configuration bundle. For
   details, refer to the documentation. Do you wish to continue? [y/n]:
   ```
   For information on which configuration changes cause a rolling restart, see Restart or reload after configuration bundle changes?.
2. To proceed, respond to the message with `y`. You can avoid this message by appending the flag `--answer-yes` to the command:

   ```
   splunk apply cluster-bundle --answer-yes
   ```

The `splunk apply cluster-bundle` command causes the manager node to distribute the new configuration bundle to the peers, which then individually validate the bundle. During this process, each peer validates the settings for all `indexes.conf` files in the bundle. After all peers successfully validate the bundle, the manager coordinates a rolling restart of all the peer nodes, if necessary.

The download and validation process usually takes just a few seconds to complete. If any peer is unable to validate the bundle, it sends a message to the manager, and the manager displays the error on its dashboard in Splunk Web. The process will not continue to the next phase - reloading or restarting the peers - unless all peers successfully validate the bundle.

If validation is not successful, you must fix any problems noted by the manager and rerun `splunk apply cluster-bundle`.

Once validation is complete, the manager node tells the peers to reload or, if necessary, it initiates a rolling restart of all the peers. For details on how rolling restart works, see Perform a rolling restart of an indexer cluster. To set searchable rolling restart as the default mode for rolling restarts triggered by a bundle push, see Use searchable rolling restart with configuration bundle push.

When the process is complete, the peers use the new set of configurations, located in their local `$SPLUNK_HOME/etc/peer-apps`.

Leave the files in $SPLUNK_HOME/etc/peer-apps.

Once an app has been distributed to the set of peers, you launch and manage it on each peer in the usual manner, with Splunk Web. See Managing app configurations and properties in the *Admin Manual.*

The apply cluster-bundle command takes an optional flag, --skip-validation, for use in cases where a problem exists in the validation process. You should only use this flag under the direction of Splunk Support and after ascertaining that the bundle is valid. Do not use this flag to circumvent the validation process unless you know what you are doing.

You can also validate the bundle without applying it. This is useful for debugging some validation issues. See 3. (Optional) Validate the bundle.

Use the CLI to view the status of the bundle push

To see how the cluster bundle push is proceeding, run this command from the manager node:

```
splunk show cluster-bundle-status
```

This command tells you whether bundle validation succeeded or failed. It also indicates the restart status of each peer.

Restart or reload after configuration bundle changes?

Some changes to files in the configuration bundle require that the peers restart. In other cases, the peers can reload the configuration files, avoiding any interruption to indexing or searching. The bundle reload phase on the peers determines whether a restart is required and directs the manager to initiate a rolling restart of the peers only if necessary.

Determine when restart or reload occurs

The following summarizes reload and restart behavior when pushing configuration changes to indexer cluster peers:

Reload occurs when:

- You make changes to any configuration file listed under the `[triggers]` stanza in `app.conf`, with the exception of certain changes to `indexes.conf` that still require a restart. For a list of changes to `indexes.conf` that require a restart, see Determine which indexes.conf changes require restart.
- You make any of these changes to `indexes.conf` (these changes do not require a restart and trigger a reload only):
  - Adding new index stanzas
  - Enabling or disabling an index with no data
  - Changing any attributes not explicitly listed as requiring restart in Determine which indexes.conf changes require restart.
- A `server.conf` file contains only the `conf_replication_include <conf_file_name> = true` attribute under the `shclustering` stanza and the file contains no other attributes.

Restart occurs when:

- You make changes to any configuration file not listed in the `[triggers]` stanza in `app.conf`.
- You make any of the changes to `indexes.conf` listed in Determine which indexes.conf changes require restart.
- You make any changes to a `server.conf` file that contains attributes other than `conf_replication_include <conf_file_name> = true`.
- You delete an existing app from the configuration bundle.

For more information, see When to restart Splunk Enterprise after a configuration file change in the *Admin Manual*.

Configuration file reload triggers in app.conf

Splunk apps can contain a combination of Splunk Enterprise core configuration files and custom configuration files, such as those created by app developers for both private apps and public apps on Splunkbase. Whether these configuration files reload when you push configuration changes to cluster peers depends on reload trigger settings in `app.conf`.

Many Splunk Enterprise core configuration files reload by default, with some exceptions outlined in the previous section. These files have pre-defined reload triggers specified under the `[triggers]` stanza in `$SPLUNK_HOME/etc/system/default/app.conf`, which causes them to reload automatically.

A custom configuration file is by definition any configuration file that does not have a corresponding `.spec` file in `$SPLUNK_HOME/etc/system/README`. This includes custom configuration files found in third party apps, such as `aws_settings.conf`, `service_now.conf`, `eventgen.conf`, and so on.

All custom configuration files reload by default, unless the file has a custom reload trigger in `app.conf`. For example, the Splunk Security Essentials app, `app.conf` contains the following custom reload trigger: `reload.ssenav = http_get /SSEResetLocalNav`. When you push configuration changes for a custom configuration file that has a custom reload trigger in `app.conf`, Splunk software tries to honor the custom reload trigger setting. If that custom reload trigger fails, then a rolling restart occurs.

If a custom configuration file does not have a reload trigger specified in app.conf, the default behavior is to restart for unknown configs. If a restart is not required, you can set the conf level trigger in `app.conf` to `reload.<conf_file_name> = simple`.

For detailed information on reload trigger settings for custom configuration files, see app.conf in the *Admin Manual*.

For a listing of the most frequently used reloadable apps and .conf files, see Restart vs. reload behavior of common apps and .conf files in the Splunk Cloud Platform documentation.

Stanza-level reload triggers for inputs.conf

Stanza-level reload triggers in `app.conf` enable the reload of only those specific configuration file stanzas that change when you perform a configuration bundle push. This lets you perform more efficient configuration updates based on which stanzas in the configuration file will change.

Stanza-level reload currently applies to a subset of stanzas in `inputs.conf` only. Stanzas in `inputs.conf` that have a `reload.<conf_file_name>.<conf_stanza_prefix>` entry under the `[triggers]` stanza in `app.conf` will reload when changes are made to the specified stanza.

Changes made to any `inputs.conf` stanzas that are not specified in a stanza-level reload entry will trigger a rolling restart.

> Stanza-level reload for inputs.conf applies only when pushing changes to the configuration bundle in the indexer clustering context.

The following stanzas are reloadable in `inputs.conf`:

.conf file name	stanza prefix	Reload or restart
inputs.conf	http	reload
inputs.conf	script	reload
inputs.conf	monitor	reload
inputs.conf	<modular_input>	reload
inputs.conf	batch	reload

For detailed information on stanza-level reload triggers, see app.conf. in the *Admin Manual*.

Disable reload triggers in app.conf

You can disable both .conf-level reload triggers and stanza-level reload triggers by specifying the value `never` for any reload trigger entry in `app.conf`. Any reload trigger entry with a value of `never` will trigger a rolling restart when configuration changes occur. This can be useful if for any reason you want a specific configuration change to trigger a rolling restart.

For more information on configuring reload triggers, see app.conf. in the *Admin Manual*.

Use searchable rolling restart with configuration bundle push

Searchable rolling restart lets you perform a rolling restart of peer nodes with minimal interruption of in-progress searches. You can set searchable rolling restart in `server.conf` as the default mode for all rolling restarts triggered by a configuration bundle push. For instructions, see Set searchable rolling restart as default mode for bundle push.

For more information, see Perform a rolling restart of an indexer cluster.

Rollback the configuration bundle

You can rollback the configuration bundle to the previous version. This action allows you to recover from a misconfigured bundle.

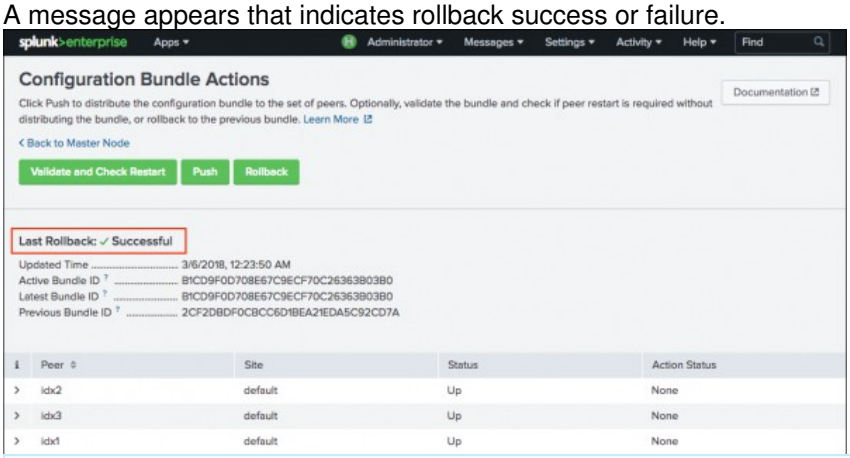
The rollback action toggles the most recently applied configuration bundle on the peers with the previously applied bundle. You cannot rollback beyond the previous bundle.

For example, say that the peers have an active configuration bundle "A" and you apply a configuration bundle "B", which then becomes the new active bundle. If you discover problems with B, you can rollback to bundle A, and the peers will then use A as their active bundle. If you rollback a second time, the peers will return again to bundle B . If you rollback a third time, the bundles will return again to A, and so on. The rollback action always toggles the two most recent bundles.

You can rollback the configuration bundle using Splunk Web or the CLI.

Rollback the configuration bundle using Splunk Web

1. On the manager node, in Splunk Web, click **Settings > Indexer Clustering**.
   The Manager Node dashboard opens.
2. Click **Edit > Configuration Bundle Actions**.
3. Click **Rollback**.
   A message appears that indicates rollback success or failure.



If no previous configuration bundles exists, the Rollback button will be disabled.

Rollback the configuration bundle using the CLI

To rollback the configuration bundle, run this command from the cluster manager:

```
splunk rollback cluster-bundle
```

As with `splunk apply cluster-bundle`, this command initiates a rolling restart of the peer nodes, when necessary.

You can use the `splunk show cluster-bundle-status` command to determine the current active bundle. You can use the `cluster/manager/info` endpoint to get information about the current active and previous active bundles.

If the `manager-apps` folder gets corrupted, resulting in rollback failure, a message specifying the failure and the workaround appears on the manager node dashboard, as well as in `splunkd.log`. To remediate, follow the instructions in the message. This includes removing the `$SPLUNK_HOME/etc/manager-apps.dirty` marker file, which indicates failure, and

manually copying over the active bundle, as specified in the message.

On Windows, the rollback operation fails if there are open file handles to $SPLUNK_HOME/etc/manager-apps and its contents.

Distribution of the bundle when a peer starts up

After you initially configure a Splunk instance as a peer node, you must restart it manually in order for it to join the cluster, as described in Enable the peer nodes. During this restart, the peer connects with the manager node, downloads the current configuration bundle, validates the bundle locally, and then restarts again. The peer only joins the cluster if bundle validation succeeds. This same process also occurs when an offline peer comes back online.

If validation fails, the user must fix the errors and run `splunk apply cluster-bundle` from the manager.

Use deployment server to distribute the apps to the manager node

Although you cannot use deployment server to directly distribute apps to the peers, you can use it to distribute apps to the manager node's configuration bundle location. Once the apps are in that location, the manager can distribute them to the peer nodes, using the configuration bundle method described in this topic.

In addition to the deployment server, you can also use third party distributed configuration management software, such as Puppet or Chef, to distribute apps to the manager.

To use the deployment server to distribute files to the configuration bundle on the manager node:

1. Configure the manager as a client of the deployment server, as described in Configure deployment clients in *Updating Splunk Enterprise Instances*.
2. On the manager, edit deploymentclient.conf and set the `repositoryLocation` attribute to the `manager-apps` location:

   ```
   [deployment-client]
   serverRepositoryLocationPolicy = rejectAlways
   repositoryLocation = $SPLUNK_HOME/etc/manager-apps
   ```
3. On the deployment server, create and populate one or more deployment apps for download to the manager's configuration bundle. Make sure that the apps follow the structural requirements for the configuration bundle, as outlined earlier in this topic. For information on creating deployment apps, see Create deployment apps in *Updating Splunk Enterprise Instances* .
4. Create one or more server classes that map the manager node to the deployment apps. For information on creating server classes, see Create server classes in *Updating Splunk Enterprise Instances*.
5. Each server class must include the `stateOnClient = noop` setting:

   ```
   [serverClass:<serverClassName>]
   stateOnClient = noop
   ```

   Do not override this setting at the app stanza level.
6. Download the apps to the manager node. Once the manager receives the new or updated deployment apps in the configuration bundle, you can distribute the bundle to the peers, using the method described in the current topic.

   Take steps to ensure that the manager does not restart automatically after receiving the deployment apps. Specifically, when defining deployment app behavior, do not change the value of the restartSplunkd setting from its default of "false" in serverclass.conf. If you are using forwarder management to define your server classes, make sure that the "Restart splunkd" field on the Edit App screen is not checked.

For detailed information on the deployment server and how to perform necessary operations, read the *Updating Splunk Enterprise Instances* manual.

Manage configurations on a peer-by-peer basis

Most configurations need to be the same across all peers. See Manage common configurations across all peers. For limited purposes, such as testing, you can handle some configurations on a peer-by-peer basis.

Configure data inputs

Forwarders are the recommended way to handle data inputs to peers. For information on configuring this process, read Use forwarders to get your data into the indexer cluster.

If you want to input data directly to a peer, without a forwarder, you can configure your inputs on the peer in the same way as for any indexer. For more information, read Configure your inputs in the *Getting Data In* manual.

Important: Although you can configure inputs on a peer-by-peer basis, consider whether your needs allow you to use a single set of inputs across all peers. This should be possible if all data is channeled through forwarders, and the receiving ports on all peers are the same. If that is the case, you can use the manager node to manage a common `inputs.conf` file, as described in Update common peer configurations and apps.

Add an index to a single peer

If you need to add an index to a single peer, you can do so by creating a separate `indexes.conf` file on the peer. However, the data in the new index will remain only on that peer and will not get replicated. The main use case for this is to perform some sort of local testing or monitoring, possibly involving an app that you download to only that one peer. The peer-specific `indexes.conf` supplements, but does not replace, the common versions of the file that all peers get.

If you create a version of `indexes.conf` for a single peer, you can put it in any of the acceptable locations for an indexer, as discussed in About configuration files and Configuration file directories in the *Admin Manual*. The one place where you cannot put the file is under `$SPLUNK_HOME/etc/peer-apps`, which is where the configuration bundle resides on the peer. If you put it there, it will get overwritten the next time the peer downloads a configuration bundle.

Important: If you add a local index, leave its `repFactor` attribute set to the default value of 0. Do not set it to `auto`. If you do set it to `auto`, the peer will attempt to replicate the index's data to other peers in the cluster. Since the other peers will not be configured for the new index, there will be nowhere on those peers to store the replicated data, resulting in various, potentially serious, problems. In addition, when the manager node next attempts to push a configuration bundle to the peers, the peer with the incorrectly configured index will return a bundle validation error to the manager, preventing the manager from successfully applying the bundle to the set of peers.

Make other configuration changes

If you need to make some other configuration changes specific to an individual peer, you can configure the peer in the usual way for any Splunk Enterprise instance, clustered or not. You can use Splunk Web or the CLI, or you can directly edit configuration files.

Restart the peer

As with any indexer, you sometimes need to restart a peer after you change its configuration. Unlike non-clustered indexers, however, do not use the CLI `splunk restart` command to restart the peer. Instead, use the restart capability in Splunk Web. For detailed information on how to restart a cluster peer, read Restart a single peer.

For information on configuration changes that require a restart, see Restart after modifying server.conf? and Restart or reload after configuration bundle changes?.

Configure the search head

Search head configuration overview

Configuration of the search head in an indexer cluster falls into these categories:

- Cluster node configuration. The basic configuration of the search head node occurs during initial deployment of the indexer cluster. You can edit the configuration later.
- Advanced features and topologies. These features, such as mounted bundles, are available to all search heads, whether or not they are participating in an indexer cluster.
- Combined searches. You can combine searches across multiple clusters or across clustered and non-clustered search peers.

Important: This chapter discusses independent search heads that function as nodes in an indexer cluster. For information on how to incorporate search heads that are members of a **search head cluster** into an indexer cluster, see "Integrate the search head cluster with an indexer cluster" in the *Distributed Search* manual. In addition, see the "Configure search head clustering" chapter in the *Distributed Search* manual.

Cluster node configuration

Basic configuration of a Splunk Enterprise instance as a search head for an indexer cluster occurs when you initially deploy the indexer cluster. You can edit the configuration later.

Perform the initial configuration

You configure and enable the search head at the same time that you enable the other cluster nodes, as described in "Enable the search head". The cluster's set of peer nodes become **search peers** of the search head. For basic functionality, you do not need to set any other configurations.

Edit the configuration

There are two main reasons for editing the basic search head configuration for a particular cluster:

- **Redirect the search head to another manager node for the same cluster.** This can be useful in the case where a manager node fails but you have a stand-by manager for that cluster which you can redirect the search head to. For information on stand-by manager nodes, see "Replace the manager node on the indexer cluster".
- **Change the search head's security key for the cluster.** Only change the key if you are also changing it for all other nodes in the cluster. The key must be the same across all instances in a cluster.

To edit the search head's cluster node configuration, use one of these methods:

- Edit the configuration from the search head node dashboard in Splunk Web. See "Configure the search head with the dashboard".
- Edit the search head's `server.conf` file. See "Configure the search head with server.conf".

- Use the CLI. See "Configure the search head with the CLI".

Configure multisite search heads

For additions and differences when configuring multisite search heads, see "Implement search affinity in a multisite indexer cluster" and "Configure multisite indexer clusters with server.conf".

Advanced features and topologies

To implement some advanced features of distributed search, such as mounted bundles, you must edit `distsearch.conf` on the search head.

For instructions on how to perform advanced configuration, read the *Distributed Search* manual. That book focuses on environments with non-clustered indexers, but you configure most advanced search head features in the same way when working with indexer clusters, except as described here.

Search heads running on an indexer cluster compared to search heads running against non-clustered indexers

Most settings and capabilities are the same for search heads running on an indexer cluster and those running against non-clustered indexers.

The main difference is that, for indexer clusters, search heads and search peers are automatically connected to each other as part of the cluster enablement process. You do not perform any configuration in `distsearch.conf` to enable automatic discovery.

A few attributes in `distsearch.conf` are not valid for search heads in indexer clusters. A search head in an indexer cluster ignores these attributes:

```
servers
disabled_servers
heartbeatMcastAddr
heartbeatPort
heartbeatFrequency
ttl
checkTimedOutServersFrequency
autoAddServers
```

As when running against non-clustered indexers, search head access to search peers is controlled through public key authentication. However, you do not need to distribute the keys manually. The search head in an indexer cluster automatically pushes its public key to the search peers.

Mounted bundles and search peer configurations

Most `distsearch.conf` settings are valid only for search heads. However, to implement mounted bundles, you need to distribute a small `distsearch.conf` file to the search peers. For indexer clusters, you should use the manager node to distribute this file to the peers. For information on how to use the manager node to manage peer configurations, read "Update common peer configurations and apps" in this manual. For information on how to configure mounted bundles, read the "Mounted knowledge bundle replication" in the *Distributed Search* manual.

How the Distributed Search page works with indexer clusters

Do not use the Distributed Search page on Splunk Web to configure a search head in an indexer cluster or to add peers to the cluster. You can, however, use that page to view the list of search peers.

Combined searches

To search across multiple indexer clusters, see "Search across multiple indexer clusters".

To search across both clustered and non-clustered search peers, see see "Search across both clustered and non-clustered search peers".

Configure the search head with the dashboard

You can edit the configuration of an existing search head node through its dashboard. To access the dashboard:

1. Click **Settings** on the upper right side of Splunk Web.

2. In the **Distributed Environment** group, click **Indexer clustering**.

The dashboard includes a number of menu items and actions that affect the configuration.

For information on using this dashboard to enable a search head initially, see Enable the search head.

For information on using this dashboard to view indexer cluster status, see View the search head dashboard.

Change configuration for a particular cluster

To change the search head's configuration for a particular indexer cluster, select the **Edit Configuration** action on the cluster's row:

- To change the manager node, edit the **Manager URI** field.
- To change the security key, edit the **Security key** field.

Join another indexer cluster

To connect the search head to another cluster, see Search across multiple indexer clusters.

Remove the search head from a cluster

To remove the search head from an indexer cluster, select the **Remove Cluster** action on the row for that cluster. This disassociates the search head from that cluster, but leaves it connected to all other clusters, if any.

To remove the search head from all clusters, select **Disable Clustering** from the **Edit** menu on the upper right corner of the dashboard.

Other edits

If you need to change this instance to some other node type, like a peer node, you can do so through the **Edit** button on the upper right side of the dashboard.

It is extremely unlikely that you will want to change the node type for nodes in an active cluster. Consider the consequences carefully before doing so.

The **Edit** button is disabled for multisite clusters.

Configure the search head with server.conf

Prerequisites

Before reading this topic, see Configure the indexer cluster with server.conf. It discusses configuration issues that are common to all cluster node types.

Enable a search head

The following example shows the basic settings that you must configure when enabling a search head node. The configuration attributes shown here correspond to fields on the **Enable clustering** page of Splunk Web.

```
[clustering]
manager_uri = https://10.152.31.202:8089
mode = searchhead
pass4SymmKey = whatever
```

This example specifies that:

- the search head's cluster manager node resides at `10.152.31.202:8089`.
- the instance is a cluster search head.
- the security key is "whatever".

Edit the search head settings

You can change these settings later, if necessary. For example, to change the cluster's security key, you change the `pass4SymmKey` value on each node.

You can also configure the search head to search across multiple indexer clusters or across clustered and non-clustered search peers. See:

- Search across multiple indexer clusters
- Search across both clustered and non-clustered search peers.

Configure the search head with the CLI

Read this first

Before reading this topic, see:

- "Configure and manage the indexer cluster with the CLI". This topic explains the basics of indexer cluster configuration with the CLI. It provides details on issues that are common to all cluster node types.

Enable a search head

The following example shows the basic settings that you typically configure when enabling a search head. The configuration attributes correspond to fields on the **Enable clustering** page of Splunk Web.

To enable an instance as a search head, set `mode` to "searchhead". You also need to specify the `manager_uri` and the cluster-wide security key (`secret`):

```
splunk edit cluster-config -mode searchhead -manager_uri https://10.160.31.200:8089 -secret your_key

splunk restart
```

The -secret flag modifies the pass4SymmKey setting in the [clustering] stanza of server.conf.

Edit the search head settings

You can also use the CLI to edit the configuration later.

Important: When you first enable a search head, you use the `splunk edit cluster-config` command. To change the search head configuration, you must instead use the `splunk edit cluster-manager` command.

For example, to change the security key (`secret`), use this command:

```
splunk edit cluster-manager https://10.160.31.200:8089  -secret newsecret123
```

Important: The `splunk edit cluster-manager` command always takes the current manager node URI:port value as its initial parameter. For example, this command connects the search head to a different manager node by setting a new value for the `-manager_uri` parameter, but it provides the value for the old manager node as its initial parameter:

```
splunk edit cluster-manager https://10.160.31.200:8089  -manager_uri https://10.160.31.55:8089
```

Refer to the CLI clustering help, along with the server.conf specification file, for the list of configurable settings.

Search across multiple indexer clusters

You can configure a search head to search across multiple indexer clusters. The method you use depends on whether the clusters are single-site or multisite.

Configure multi-cluster search for single-site indexer clusters

To configure multi-cluster search:

1. Configure the search head for one of the clusters in the usual way, as described in "Enable the search head".

2. Point the search head at the manager node for the new cluster. You can do this with Splunk Web, through the CLI, or by editing the search head's `server.conf` file.

In Splunk Web

In Splunk Web, configure multi-cluster search from the search head dashboard:

1. Select the **Add cluster to be searched** button on the uppper right corner of the dashboard.

2. Fill out the fields in the pop-up window:

- **Manager URI**. Enter the manager node's URI, including its management port. For example: `https://10.152.31.202:8089`.
- **Security Key**. This is the key that authenticates communication between a cluster's manager node, peers, and search heads. The key must be the same across all nodes in a cluster. Enter the security key for the new cluster here. The key might be different for each of the search head's clusters.

To remove the search head from a cluster, see "Remove the search head from a cluster".

Through the CLI

In the CLI, you can configure multi-cluster search with these commands:

```
splunk add cluster-manager <manager_uri:port>
splunk edit cluster-manager <manager_uri:port>
splunk remove cluster-manager <manager_uri:port>
splunk list cluster-manager
```

You do not need to restart the search head after running these commands.

For example, to add the search head to a cluster whose manager node is located at https://10.160.31.200:8089, run this command:

```
splunk add cluster-manager https://10.160.31.200:8089 -secret your_key
```

For more information on any command, see its CLI help.

By editing server.conf

You can configure multi-cluster search in the search head's `server.conf` file by specifying a comma-separated list of manager node references in the `manager_uri` attribute, followed by individual stanzas for each manager. For example:

```
[clustering]
mode = searchhead
manager_uri = clustermanager:east, clustermanager:west

[clustermanager:east]
manager_uri=https://SplunkManager01.example.com:8089
pass4SymmKey=someSecret

[clustermanager:west]
manager_uri=https://SplunkManager02.example.com:8089
pass4SymmKey=anotherSecret
```

In this example, the search head will use the `pass4SymmKey` "someSecret" when communicating with SplunkManager01 and `pass4SymmKey` "anotherSecret" when communicating with SplunkManager02.

After you edit `server.conf`, you must restart the search head for the changes to take effect.

For details on configuring multi-cluster search, see the server.conf specification file.

Configure multi-cluster search for multisite indexer clusters

A search head can search across multiple multisite clusters or a combination of single-site and multisite clusters. To configure this, you need to specify the search head's `site` attribute when connecting it to a multisite cluster.

Through the CLI

In the CLI, you configure multi-cluster search with the `splunk add cluster-manager` command. When adding a multisite cluster, include the search head's `site` value:

```
splunk add cluster-manager <manager_uri:port> -site site<n>
```

You do not need to restart the search head after running this command.

By editing server.conf

To configure multi-cluster search for a multisite cluster, you need to set two multisite-specific attributes: `site` and `multisite`. The locations of these attributes vary, depending on a few factors.

If the search head will be searching across only multisite clusters, and the search head is on the same site in each cluster, put the `site` attribute under the `[general]` stanza and the `multisite` attribute under each `[clustermanager]` stanza:

```
[general]
site=site1

[clustering]
mode = searchhead
manager_uri = clustermanager:multieast, clustermanager:multiwest

[clustermanager:multieast]
multisite=true
manager_uri=https://SplunkManager01.example.com:8089
pass4SymmKey=someSecret

[clustermanager:multiwest]
multisite=true
manager_uri=https://SplunkManager02.example.com:8089
pass4SymmKey=anotherSecret
```

If the search head will be searching across only multisite clusters, and the search head is on a different site in each cluster, put both the `site` and the `multisite` attributes under the `[clustermanager]` stanzas:

```
[clustering]
mode = searchhead
manager_uri = clustermanager:multieast, clustermanager:multiwest

[clustermanager:multieast]
multisite=true
manager_uri=https://SplunkManager01.example.com:8089
pass4SymmKey=someSecret
site=site1

[clustermanager:multiwest]
```

```
multisite=true
manager_uri=https://SplunkManager02.example.com:8089
pass4SymmKey=anotherSecret
site=site2
```

If the search head will be searching across a combination of single-site and multisite clusters, put both the `site` and the `multisite` attributes under the `[clustermanager]` stanza for any multisite clusters. In this example, the search head searches across two clusters, only one of which is multisite:

```
[clustering]
mode = searchhead
manager_uri = clustermanager:multi, clustermanager:single

[clustermanager:multi]
multisite=true
manager_uri=https://SplunkManager01.example.com:8089
pass4SymmKey=someSecret
site=site1

[clustermanager:single]
manager_uri=https://SplunkManager02.example.com:8089
pass4SymmKey=anotherSecret
```

After you edit `server.conf`, you must restart the search head for the changes to take effect.

For more information on multisite cluster configuration, see "Configure multisite indexer clusters with server.conf".

Search across both clustered and non-clustered search peers

You can search across both clustered and non-clustered search peers:

1. Configure an indexer cluster search head in the standard fashion, as described in "Enable the search head".

2. Use Splunk Web or the CLI to add one or more non-clustered search peers, as described in "Add search peers to the search head" in *Distributed Search*.

Note the following:

- The procedure assumes that you are starting with a new Splunk Enterprise instance that you want to enable as a search head for both an indexer cluster and one or more non-clustered indexers. If the instance is already functioning as a search head in one of those two roles, you only need to enable it for the other role. For example, if the search head is already part of an indexer cluster, then you only perform step 2.
- You must specify the non-clustered search peers through either Splunk Web or the CLI. Due to authentication issues, you cannot specify the search peers by directly editing `distsearch.conf`. When you add a search peer with Splunk Web or the CLI, the search head prompts you for public key credentials. It has no way of obtaining those credentials when you add a search peer by directly editing `distsearch.conf`. For more information on public keys and distributed search, read "Add search peers to the search head" in *Distributed Search*.

- An indexer can be either a cluster peer, in which case, it is automatically a search peer for that cluster's search head, or a non-clustered search peer, with an entry in the search head's `distsearch.conf` file. It cannot be both. If you mistakenly configure an indexer as both a cluster peer and a non-clustered search peer, the search head's Distributed Search page in Splunk Web will contain two entries for the peer, and the status for one entry will read, "Peer member of cluster and distsearch.conf". To remediate, disable or delete the entry for that peer in `distsearch.conf`.

- This type of search is not compatible with search head pooling.

Deploy and configure a multisite indexer cluster

Multisite indexer cluster deployment overview

Before reading this topic, see:

- "Indexer cluster deployment overview". That topic provides a general overview of deployment for both single-site and **multisite** indexer clusters. The topic you are reading now describes only the multisite differences.

Important: This chapter assumes that you are deploying independent search heads in the multisite indexer cluster. For information on how to incorporate search heads that are members of a **search head cluster**, see "Integrate the search head cluster with an indexer cluster" in the *Distributed Search* manual.

Migrating from a single-site cluster?

To migrate from a single-site to a multisite indexer cluster, read "Migrate an indexer cluster from single-site to multisite".

Deploy a multisite indexer cluster

To deploy a multisite cluster, you configure the set of nodes for each site:

- A single manager node resides on one of the sites and controls the entire multisite cluster.
- A set of peer nodes resides on each site.
- A search head resides on each site that searches cluster data. If you want all searches to be local,Â you must install a search head on each site. This is known as **search affinity.**

For example, to set up a two-site cluster with three peers and one search head on each site, you install and configure these instances:

- One manager node on one of the sites, either site 1 or site 2
- Three peer nodes on site 1
- Three peer nodes on site 2
- One search head on site 1
- One search head on site 2

Note: The manager node itself is not actually a member of any site, aside from its physical location. However, each manager node has a built-in search head, and that search head requires that you set a site attribute in the manager's configuration. You must specify a site for the manager, even if you never use its built-in search head. Note that the search head is for testing only. Do not use it for production purposes.

Configure multisite nodes

To deploy and configure multisite cluster nodes, you must directly edit `server.conf` or use the CLI. You cannot use Splunk Web.

Multisite-specific configuration settings

When you deploy a multisite cluster, you configure the same settings as for single-site, along with some additional settings to specify the set of sites and the location of replicated and searchable copies across the sites.

On the manager node, you:

- Enable the cluster for multisite.
- Enumerate the set of sites for the cluster.
- Set a multisite replication factor.
- Set a multisite search factor.
- Adjust the single-site replication and search factors as necessary. See "Multisite cluster does not meet its replication or search factors."

On each cluster node, you:

- Identify the site that the node resides on.

Configure with server.conf

To configure a multisite manager node with `server.conf`, see "Configure multisite indexer clusters with server.conf".

Configure with the CLI

To configure a multisite manager node with the CLI, see "Configure multisite indexer clusters with the CLI"

Use indexer discovery with a multisite cluster

If you are using indexer discovery to connect forwarders to the peer nodes, you must assign a site to each forwarder. See "Use indexer discovery in a multisite cluster."

Implement search affinity in a multisite indexer cluster

One of the key benefits of multisite indexer clustering is that it allows you to configure a cluster so that search heads get search results only from data stored on their local sites. This reduces network traffic while still providing access to the entire set of data, because each site contains a full copy of the data. This benefit is known as **search affinity**.

For example, say you have two data centers in California, one in San Francisco and the other in Los Angeles. You set up a two-site cluster, with each site corresponding to a data center. Search affinity allows you to reduce long-distance network traffic. Search heads at the San Francisco data center get results only from the peers in San Francisco, while search heads in Los Angeles get results only from their local peers.

How search affinity works

For those sites that you want to support search affinity, you must configure multisite clustering so that the site has a full set of searchable data and a local search head. The search head on any particular site then gets data only from its local site, as long as that site is **valid**.

When search affinity is functioning, each search head sends its searches to all peers, across all sites, but only the local peers search their data (that, is, their primary bucket copies) and return results to the search head.

If a local peer holding some of the primary searchable data goes down and the site temporarily loses its valid state, the search head will, if necessary, get data from peers on remote sites while the local site is undergoing bucket fixing. During this time, the search head will still get as much of the data as possible from the local site.

Once the site regains its valid state, new searches again occur across only the local site.

For more details on how the cluster handles search affinity, see "Multisite indexer cluster architecture" and "Search locally in a multisite cluster".

Implement search affinity

With multisite clusters, search affinity is enabled by default. However, you must perform a few steps to take advantage of it. Specifically, you must ensure that both the searchable data and the search heads are available locally.

To implement search affinity:

1. Configure the site search factor so that you have at least one searchable copy on each site where you require search affinity.

One way to do this is to explicitly specify a search factor for each site that requires search affinity. For example, a four-site cluster with `site_search_factor = origin:1, site1:1, site2:1, total:3` ensures that both site1 and site2 have searchable copies of every bucket. The third set of searchable copies will be spread across the two non-explicit sites, with no guarantee that either site will have a full set of searchable copies. Thus, search affinity is enabled for only site1 and site2. Site1 and site2 will each hold primary copies of all buckets.

There are also ways to configure the site search factor to ensure that all sites have searchable copies, even without explicitly specifying some or all of them. For example, a three-site cluster with `site_search_factor = origin:1, total:3` guarantees one searchable copy per site, and thus enables search affinity for each site. Each site will have primary copies of all buckets.

For more information on how replication and search factors distribute copies across sites, see "Configure the site replication factor" and "Configure the site search factor".

2. Deploy a search head on each site where you require search affinity.

Disable search affinity

You can disable search affinity for any search head. When search affinity is disabled, the search head does not attempt to obtain search results from a single site only. Rather, it can obtain results from multiple sites. This can be useful, for example, if you have two data centers in close proximity with low latency, and you want to improve overall performance by spreading the processing across indexers on both sites.

What happens when search affinity is disabled

When search affinity is disabled on a search head, search results can come from indexers on any or all sites. If the site search factor stipulates searchable bucket copies on multiple sites, the search head uses undefined criteria to choose which of the searchable copies to search. It is likely to choose some bucket copies from one site and other bucket copies from other sites, so the results will come from multiple sites.

Search heads always select from primary bucket copies. For example, say you have a two site cluster with this search factor:

```
site_search_factor = origin:2, total:3
```

The origin site will store two searchable copies and the second site will store one searchable copy of each bucket. So, for some buckets (those originating on site1), site1 will have two searchable copies and for other buckets (those originating

on site2), site2 will have two searchable copies. Each site, however, has only a single primary copy.

A search head with search affinity enabled limits its searches to the primary copies on its own site, when possible.

In contrast, a search head with search affinity disabled distributes its search across primary copies on both sites. For a given bucket, you cannot know whether it will select the primary on site1 or the primary on site2. It does tend to use the same primaries from one search to the next.

How to disable search affinity

To disable search affinity for a search head, set the search head's site value to "site0" in `server.conf`:

```
[general]
site = site0

[clustering]
multisite = true
...
```

By setting `site=site0`, you cause searches to behave like they would on a single-site cluster, with no preference for any particular site.

For more information on configuring multisite search heads, see "Configure the search heads."

Limit primaries to sites with search heads

By default, the indexer cluster limits primary bucket copies to sites with search heads. This behavior significantly reduces primary fixup and rebalance activity, particularly in cases where all search heads are assigned to only one or a few sites and search affinity is disabled.

This behavior can be changed in the manager node's `server.conf` file, with the setting `assign_primaries_to_all_sites`. By default, the setting is "false", which causes the cluster to limit primaries to sites with search heads:

```
[clustering]
assign_primaries_to_all_sites=false
```

When `assign_primaries_to_all_sites=false`, the manager assigns primaries only to sites currently with an online search head. For example, if no online search head is assigned to site1, no primaries will be assigned to site1. Similarly, if no search head is assigned to site0, no primaries will be assigned to site0.

When a search head joins a site that previously did not have an assigned search head, the manager gradually assigns primary status to one searchable copy of each bucket on that site, for all buckets with searchable copies on that site.

If a site loses all its search heads, all existing primaries on that site remain as such but the manager does not assign primary status to any new bucket copies on the site.

You can also use the CLI on the manager node to change this setting:

```
splunk edit cluster-config -assign_primaries_to_all_sites false|true
```

Configure multisite indexer clusters with server.conf

Read this first

Before reading this topic, see:

- "Multisite indexer cluster deployment overview". This topic provides important background information about configuring a multisite cluster.
- "Configure the indexer cluster with server.conf". This topic explains the basics of cluster configuration. It focuses on single-site clusters, but most of the information is relevant to multisite clusters as well.

How multisite configuration differs from single-site configuration

You configure multisite indexer clusters in a similar way to how you configure clusters for a single site, with the exception of a few new attributes:

On all multisite cluster nodes (manager/peers/search heads):

- The `site` attribute specifies the site that a node resides on.

On the manager node and search head:

- The `multisite` attribute indicates that the manager or search head is participating in a multisite cluster.

On the manager node only:

- The `available_sites` attribute names the sites that the manager is managing.
- The `site_replication_factor` replaces the standard `replication_factor` used with single-site clusters. For details, see "Configure the site replication factor".
- The `site_search_factor` replaces the standard `search_factor` used with single-site clusters. For details, see "Configure the site search factor".

Important: Although the `site_replication_factor` effectively replaces the single-site `replication_factor`, and the `site_search_factor` replaces single-site `search_factor`, those single-site attributes continue to exist in the manager's configuration, with their default values of 3 and 2, respectively. This can cause problems on start-up if any site has fewer peer nodes than either of those values; that is, if any site has only one or two peer nodes. The symptom will be a message that the multisite cluster does not meet its replication or search factor. For example, if one of your sites has only two peers, the default single-site replication factor of 3 will cause the error to occur. To avoid or fix this problem, you must set the single-site replication and search factors to values that are less than or equal to the smallest number of peers on any site. In the case where one site has only two peers, you must therefore explicitly set the `replication_factor` attribute to a value of 2. See "Multisite cluster does not meet its replication or search factors."

If you are migrating a cluster from single-site to multisite, you must keep the existing `replication_factor` and `search_factor` attributes for the existing single-site buckets, while also adding the new multisite `site_replication_factor` and `site_search_factor` attributes for the new multisite buckets. See "Migrate an indexer cluster from single-site to multisite".

Configure multisite cluster nodes

To configure a multisite cluster, you configure the nodes for each site, editing each node's `server.conf` file. For details on the clustering attributes, read the server.conf specification.

Site values

Site values identify the site on which a node resides. You assign a site value to each node in a multisite cluster. To do this, you set the `site` attribute in the node's `[general]` stanza.

Site values have the syntax:

```
site<n>
```

where <n> is an integer in the range of 0 to 63: site1, site2, site3, ...

For example:

```
site=site1
```

The special value "site0" can be set only on search heads, or on forwarders that are participating in indexer discovery:

- For a search head, "site0" disables **search affinity** for the search head.â See Disable search affinity.
- For a forwarder participating in indexer discovery, "site0" causes the forwarder to send data to all peer nodes across all sites. See Use indexer discovery in a multisite cluster.

Configure the manager node

You configure the key attributes for the entire cluster on the manager node. Here is an example of a multisite configuration for a manager node:

```
[general]
site = site1

[clustering]
mode = manager
multisite = true
available_sites = site1,site2
site_replication_factor = origin:2,total:3
site_search_factor = origin:1,total:2
pass4SymmKey = whatever
cluster_label = cluster1
```

This example specifies that:

- the instance is located on site1.
- the instance is a cluster manager node.
- the cluster is multisite.
- the cluster consists of two sites: site1 and site2.
- the cluster's replication factor is the default "origin:2,total:3".
- the cluster's search factor is "origin:1,total:2".
- the cluster's security key is "whatever".

- the cluster label is "cluster1."

Note the following:

- You specify the `site` attribute in the `[general]` stanza. You specify all other multisite attributes in the `[clustering]` stanza.
- You can locate the manager node on any site in the cluster, but each cluster has only one manager node.
- You must set `multisite=true`.
- You must list all cluster sites in the `available_sites` attribute.
- You must set a `site_replication_factor` and a `site_search_factor`. For details, see Configure the site replication factor and Configure the site search factor.
- The `pass4SymmKey` attribute, which sets the security key, must be the same across all cluster nodes. See Configure the indexer cluster with server.conf for details.
- The cluster label is optional. It is useful for identifying the cluster in the monitoring console. See Set cluster labels in *Monitoring Splunk Enterprise.*

Important: When the manager starts up for the first time, it blocks indexing on the peers until you enable and restart the full replication factor number of peers. For example, given a three-site cluster with "site_replication_factor = origin:2, site1:1, site2:2, site3:3, total:8", the manager blocks indexing until there are at least eight peers in total across all sites, including at least one in site1, two in site2, and three in site3.

Do not restart the manager while it is waiting for the peers to join the cluster. If you do, you will need to restart the peers a second time.

Configure the peer nodes

To configure a peer node in a multisite cluster, you set a `site` attribute in the `[general]` stanza. All other configuration settings are identical to a peer in a single-site cluster.

Here is an example configuration for a multisite peer node:

```
[general]
site = site1

[replication_port://9887]

[clustering]
manager_uri = https://10.152.31.202:8089
mode = peer
pass4SymmKey = whatever
```

This example specifies that:

- the instance is located in site1. A peer can belong to only a single site.
- the peer will use port 9887 to listen for replicated data streamed from the other peers. You can specify any available, unused port as the replication port. Do not re-use the management or receiving ports.
- the peer's cluster manager node is located at `10.152.31.202:8089`.
- the instance is a cluster peer node.
- the security key is "whatever".

Configure the search heads

Multisite search heads can provide search affinity. For information, see "Implement search affinity in a multisite indexer cluster".

To configure a search head in a multisite cluster, you set a `site` attribute in the `[general]` stanza and a `multisite` attribute in the `[clustering]` stanza. All other configuration settings are identical to a search head in a single-site cluster. Here is an example configuration for a multisite search head node:

```
[general]
site = site1

[clustering]
multisite = true
manager_uri = https://10.152.31.202:8089
mode = searchhead
pass4SymmKey = whatever
```

This example specifies that:

- the instance is located in site1. A search head can belong to only a single site per cluster.
- the search head is a member of a multisite cluster.
- the search head's cluster manager is located at `10.152.31.202:8089`.
- the instance is a cluster search head.
- the security key is "whatever".

Note: You can also use `server.conf` to enable multi-cluster search, in which the search head searches across multiple clusters, multisite or single-site. For searching across multiple multisite clusters, you can configure the search head to be a member of a different site on each cluster. For details, see "Configure multi-cluster search for multisite clusters".

When reconfiguring a search head that is up-and-running, Splunk recommends that you use the CLI command described in "Configure multisite indexer clusters with the CLI", rather than editing `server.conf` directly. If you use the CLI, you do not need to restart the search head.

Restart the cluster nodes

After initial configuration

After configuring instances as multisite cluster nodes, you need to restart all of them (manager, peers, and search head) for the changes to take effect. You can do this by invoking the CLI `restart` command on each node:

```
$SPLUNK_HOME/bin/splunk restart
```

Important: When the manager starts up for the first time, it blocks indexing on the peers until you enable and restart the full replication factor number of peers. For example, given a three-site cluster with "site_replication_factor = origin:2, site1:1, site2:2, site3:3, total:8", the manager blocks indexing until there are at least eight peers in total across all sites, including at least one in site1, two in site2, and three in site3.

Do not restart the manager while it is waiting for the peers to join the cluster. If you do, you will need to restart the peers a second time.

After changing the configuration

On the manager

You must restart the manager after you change any of the following attributes:

- `multisite`
- `available_sites`
- `site_replication_factor`
- `site_search_factor`

After you restart the manager, you must also initiate a rolling-restart of the cluster peers. If you don't, the cluster will be in an indeterminate state. For information on the `splunk rolling-restart` command, see "Use rolling restart".

You do not need to restart if you change the `site` value on a manager.

On the peers

If you change the `site` value on a peer, you must restart it for the change to take effect.

Important: Although you can use the CLI `restart` command when you initially enable an instance as a cluster peer node, you should not use it for subsequent restarts. The `restart` command is not compatible with index replication once replication has begun. For more information, including a discussion of safe restart methods, see "Restart a single peer".

On the search head

You do not need to restart if you change the `site` value on a search head.

Configure multisite indexer clusters with the CLI

Read this first

Before reading this topic, see:

- "Multisite indexer cluster deployment overview". This topic provides important background information about configuring a multisite cluster.
- "Configure the indexer cluster with the CLI". This topic explains the basics of using the CLI to configure a cluster. It focuses on single-site clusters, but most of its information is relevant to multisite clusters as well.
- "Configure multisite indexer clusters with server.conf". This topic provides useful information on configuring a multisite cluster, including details on the attributes corresponding to the command-line options described in the current topic.

Configure multisite cluster nodes

You configure instances as multisite cluster nodes with the `splunk edit cluster-config` command. After enabling an instance, you must restart it.

Site values

Site values identify the site on which a node resides. You assign a site value to each node in a multisite cluster.

Site values have the syntax:

```
>site<n>
```

where <n> is an integer in the range of 1 to 63: site1, site2, site3, ...

Note: In the case of a search head only, you can also set the site value to "site0". This setting disables **search affinity** for the search head.

Configure the manager node

Here is an example of a multisite configuration for a manager mode:

```
splunk edit cluster-config -mode manager -multisite true -available_sites site1,site2 -site site1
-site_replication_factor origin:2,total:3 -site_search_factor origin:1,total:2 -secret your_key

splunk restart
```

This example specifies that:

- the instance is a cluster manager node.
- the cluster is multisite.
- the cluster consists of two sites: site1 and site2.
- the manager is located on site1.
- the cluster's replication factor is the default "origin:2,total:3".
- the cluster's search factor is "origin:1,total:2".
- the manager, along with the other nodes in the cluster, uses "your_key" as its security key. The `-secret` flag modifies the `pass4SymmKey` setting in the `[clustering]` stanza of `server.conf`.

Note the following:

- Each cluster has only one manager.
- You must set `multisite` to `true` for multisite cluster managers.
- You must list all cluster sites with the `available_sites` attribute.
- You must set a `site_replication_factor` and a `site_search_factor`. For details, see "Configure the site replication factor" and "Configure the site search factor".

You might also need to adjust the single-site replication and search factors. See "How multisite configuration differs from single-site configuration." When the manager starts up for the first time, it blocks indexing on the peers until you enable and restart the full replication factor number of peers. For example, given a three-site cluster with "site_replication_factor = origin:2, site1:1, site2:2, site3:3, total:8", the manager blocks indexing until there are at least eight peers in total across all sites, including at least one in site1, two in site2, and three in site3.

Do not restart the manager while it is waiting for the peers to join the cluster. If you do, you will need to restart the peers a second time.

You do not need to restart the manager if you later change its `site` value.

Configure the peer nodes

To configure a peer node in a multisite cluster, you set a `site` attribute. All other configuration settings are identical to a peer in a single-site cluster.

Here is an example configuration for a multisite peer node:

```
splunk edit cluster-config -mode peer -site site1 -manager_uri https://10.160.31.200:8089 -replication_port
9887 -secret your_key

splunk restart
```

This example specifies that:

- the instance is a cluster peer node.
- the instance is located in site1. A peer can belong to only a single site.
- the peer's cluster manager node is located at `10.160.31.200:8089`.
- the peer will use port 9887 to listen for replicated data streamed from the other peers. You can specify any available, unused port as the replication port. Do not re-use the management or receiving ports.

You do not need to restart the peer if you later change its `site` value.

Configure the search heads

To configure a search head for a multisite cluster, set the `site` parameter. All other settings are the same as for a search head in a single-site cluster.

You use different commands to configure a search head initially and to change its configuration later.

To initially configure a search head:

Use the `splunk edit cluster-config` command. Here is an example configuration for a multisite search head:

```
splunk edit cluster-config -mode searchhead -site site1 -manager_uri https://10.160.31.200:8089 -secret
your_key

splunk restart
```

This example specifies that:

- the instance is a cluster search head.
- the search head is located in site1. A search head can belong to only one site in each cluster.
- the search head's indexer cluster manager node is located at `10.160.31.200:8089`.

To disable search affinity for a search head, so that it gets its data randomly from all sites in the cluster, set the `site` attribute to "site0".

Note: When you specify the `site` parameter, the command automatically sets `multisite=true` in the search head's `server.conf` file. You do not need to explicitly pass a `multisite` parameter.

To edit the search head configuration later:

Use the `splunk edit cluster-manager` command, not the `splunk edit cluster-config` command.

For example, assume that you initially configured a single-site search head using the `splunk edit cluster-config` command:

```
splunk edit cluster-config -mode searchhead -manager_uri https://10.160.31.200:8089

splunk restart
```

To later reconfigure the search head for a multisite cluster, use the `splunk edit cluster-manager` command:

```
splunk edit cluster-manager https://10.160.31.200:8089  -site site1
```

Important: The `splunk edit cluster-manager` command always takes the current manager node's URI:port value as its initial parameter. For more examples, see "Configure the indexer cluster search head with the CLI".

For information on configuring a multisite search head for multi-cluster search, see "Configure multi-cluster search for multisite clusters".

Note: You do not need to restart the search head if you later change its `site` value.

Configure the site replication factor

Read this first

Before attempting to configure the site replication factor, you must be familiar with:

- The basic, single-site replication factor. See "The basics of indexer cluster architecture" and "Replication factor".
- Multisite cluster configurations. See "Configure multisite indexer clusters with server.conf".

What is a site replication factor?

To implement multisite indexer clustering, you must configure a site replication factor. This replaces the standard replication factor, which is specific to single-site deployments. You specify the site replication factor on the manager node, as part of the basic configuration of the cluster.

The site replication factor provides site-level control over the location of bucket copies, in addition to providing control over the total number of copies across the entire cluster. For example, you can specify that a two-site cluster maintain a total of three copies of all buckets, with one site maintaining two copies and the second site maintaining one copy.

You can also specify a replication policy based on which site originates the bucket. That is, you can configure the replication factor so that a site receiving external data maintains a greater number of copies of buckets for that data than for data that it does not originate. For example, you can specify that each site maintains two copies of all data that it originates but only one copy of data originating on another site.

Syntax

You configure the site replication factor with the `site_replication_factor` attribute in the manager's server.conf file. The attribute resides in the `[clustering]` stanza, in place of the single-site `replication_factor` attribute. For example:

```
[clustering]
mode = manager
multisite=true
available_sites=site1,site2
site_replication_factor = origin:2,total:3
site_search_factor = origin:1,total:2
```
You can also use the CLI to configure the site replication factor. See "Configure multisite indexer clusters with the CLI".

You must configure the `site_replication_factor` attribute correctly. Otherwise, the manager will not start.

For disaster recovery purposes, be sure to configure the site replication factor so that the cluster maintains copies of each bucket on multiple sites.

Here is the formal syntax:

```
site_replication_factor = origin:<n>, [site1:<n>,] [site2:<n>,] ..., total:<n>
```
where:

- `<n>` is a positive integer indicating the number of copies of a bucket.
- `origin:<n>` specifies the minimum number of copies of a bucket that will be held on the site originating the data in that bucket (that is, the site where the data first entered the cluster). When a site is originating the data, it is known as the "origin" site.
- `site1:<n>, site2:<n>, ...,` indicates the minimum number of copies that will be held at each specified site. The identifiers "site1", "site2", and so on, are the same as the `site` attribute values specified on the peer nodes.
- `total:<n>` specifies the total number of copies of each bucket, across all sites in the cluster.

Note the following:

- This attribute specifies the per-site replication policy. It is specified globally and applies to all buckets in all indexes.
- This attribute is valid only if `mode=manager` and `multisite=true`. Under those conditions, it supersedes any `replication_factor` attribute.
- The `origin` and `total` values are required.
- Site values (`site1:<n>, site2:<n>, ...`) are optional. A site that is specified here is known as an "explicit" site. A site that is not specified is known as a "non-explicit" site.
- Here is how the cluster determines the minimum number of copies a site gets:
  - When a site is functioning as origin, the minimum number of copies the site gets is the greater of either its site value, if any, or `origin`.
  - When a site is not functioning as origin, the site value, if any, determines the minimum number of copies the site gets.
  - A non-explicit site is not guaranteed any copies except when it is functioning as the origin site.

    For example, in a four-site cluster with "site_replication_factor = origin:2, site1:1, site2:2, site3:3, total:8", site1 gets two copies of any data that it originates and one copy of data that any other site originates. Site2 gets two copies of data, whether or not it originates it. Site3 gets three copies of data, whether or not it originates it. The non-explicit site4 gets two copies of data that it

originates, two copies of data that site2 or site3 originates, and one copy of data that site1 originates. (Site4 gets the number of copies necessary to ensure that the number of copies of each bucket meets the `total` value of 8.) Here is how you calculate site4's number of copies, according to where the data originates:

```
originate at site1 -> 2 copies in site1, 2 copies in site2, 3 copies in site3, 1 copy in site4 => total=8
originate at site2 -> 1 copy in site1, 2 copies in site2, 3 copies in site3, 2 copies in site4 => total=8
originate at site3 -> 1 copy in site1, 2 copies in site2, 3 copies in site3, 2 copies in site4 => total=8
originate at site4 -> 1 copy in site1, 2 copies in site2, 3 copies in site3, 2 copies in site4 => total=8
```

- When specifying the `site_replication_factor`, here is how you determine the minimum required value for `total`, based on the site and `origin` values:
  - If there are some non-explicit sites, then the `total` value must be at least the sum of all explicit site and `origin` values.

    For example, with a three-site cluster and "site_replication_factor = origin:2, site1:1, site2:2", the `total` must be at least 5: 2+1+2=5. For another three-site cluster with "site_replication_factor = origin:2, site1:1, site3:3", the `total` must be at least 6: 2+1+3=6.

  - If all sites are explicit, then the `total` must be at least the minimum value necessary to fulfill the dictates of the site and `origin` values.

    For example, with a three-site cluster and "site_replication_factor = origin:1, site1:1, site2:1, site3:1", the `total` must be at least 3, because that configuration never requires more than three copies. For a three-site cluster and "site_replication_factor = origin:2, site1:1, site2:1, site3:1", the `total` must be at least 4, because one of the sites will always be the origin and thus require two copies, while the other sites will each require only one. For a three-site cluster and "site_replication_factor = origin:3, site1:1, site2:2", site3:3", the `total` must be at least 8, to cover the case where site1 is the origin.

    The easiest way to determine this is to substitute the origin value for the smallest site value and then sum the site values (including the substituted origin value). So, in the last example ("site_replication_factor = origin:3, site1:1, site2:2", site3:3"), site1 has the smallest value, which is 1. You substitute the origin's 3 for that 1 and then add the site2 and site3 values: 3+2+3=8.

- Because the `total` value can be greater than the total set of explicit values, the cluster needs a strategy to handle any "remainder" bucket copies. Here is the strategy:
  - If copies remain to be assigned after all site and origin values have been satisfied, those remainder copies are distributed across all sites, with preference given to sites with less or no copies, so that the distribution is as even as possible. Assuming that there are enough remainder copies available, each site will have at least one copy of the bucket.

    For example, given a four-site cluster with "site_replication_factor = origin:1, site1:1, site4:2, total:4", if site1 is the origin, there will be one remainder copy. This copy gets distributed randomly to either site2 or site3. However, if site2 is the origin, it gets one copy, leaving no remainder copies to distribute to site3.

    In another example, given a four-site cluster where "site_replication_factor = origin:2, site1:2, site4:2, total:7", if site1 is the origin, there are three remainder copies to distribute. Because site2 and site3 have no explicitly assigned copies, the three copies are distributed between them, with each site getting at least one copy. If site2 is the origin, however, it gets two copies and site3 gets the one remainder copy.

    This entire process depends on the availability of a sufficient number of peers on each site. If a site does not have enough peers available to accept additional copies, the copies go to sites with available peers.

In any case, at least one copy will be distributed or reserved for each site, assuming enough copies are available.

Here are a few more examples:

- A three-site cluster with "origin:1, total:3": The distribution guarantees one copy per site. **Note**: If you prefer, you can instead explicitly specify a copy for each site.
- A three-site cluster with "origin:1, total:4": The distribution guarantees one copy per site, with one additional copy going to a site that has at least two peers.
- A three-site cluster with "origin:1, total:9", where site1 has only one peer and site2 and site3 have 10 peers each: The distribution guarantees one copy per site, with the six remaining copies distributed evenly between site2 and site3.

- If all peers on one non-explicit site are down and there are still remainder copies after all other non-explicit sites have received a copy, the cluster will reserve one of the remainder copies for that site, pending the return of its peers. During that time, the `site_replication_factor` cannot be met, because the total number of copies distributed will be one less than the specified `total` value, due to the copy that is being held in reserve for the site with the downed peers.

  For example, given a four-site cluster with "site_replication_factor = origin:1, site1:1, site4:2, total:5", if site1 is the origin, there will be two remainder copies to distribute between site2 and site3. If all the peers on site2 are down, one remainder copy goes to site3 and the other copy will be held in reserve until one of the site2 peers rejoins the cluster. During that time, the `site_replication_factor` is not met. However, given a four-site cluster with "site_replication_factor = origin:1, site1:1, site4:2, total:4" (the only difference from the last example being that the `total` value is 4 instead of 5), if site1 is the origin, there will be only one remainder copy, which will go to either site2 or site3. If all the peers on site2 are down, the copy will go to site3 and no copy will be held in reserve for site2. The `site_replication_factor` is met in this example, because no copies are being held in reserve for site2.

- Each site must deploy a set of peers at least as large as the greater of the origin value or its site value.

  For example, given a three-site cluster with "site_replication_factor = origin:2, site1:1, site2:2, site3:3, total:7", the sites must have at least the following number of peers: site1: 2 peers; site2: 2 peers; site3: 3 peers.

- The total number of peers deployed across all sites must be greater than or equal to the `total` value.

- The `total` value must be at least as large as the `replication_factor` attribute, which has a default value of 3. Therefore, if the `total` value is 2, you should explicitly set the value of `replication_factor` to 2.

- If you are migrating from a single-site cluster, the `total` value must be at least as large as the `replication_factor` for the single-site cluster. See "Migrate an indexer cluster from single-site to multisite".

- The attribute defaults to: "origin:2, total:3."

Examples

- **A cluster of two sites (site1, site2), with the default "site_replication_factor = origin:2, total:3":** For any given bucket, the site originating the data stores two copies. The remaining site stores one copy.

- **A cluster of three sites (site1, site2, site3), with the default "site_replication_factor = origin:2, total:3":** For any given bucket, the site originating the data stores two copies. One of the two non-originating sites, selected at random, stores one copy, and the other doesn't store any copies.

- **A cluster of three sites (site1, site2, site3), with "site_replication_factor = origin:1, site1:1, site2:1, site3:2, total:5":** For all buckets, site1 and site2 store a minimum of one copy each and site3 stores two copies. The fifth copy gets distributed to either site1 or site2, because those sites have fewer assigned copies than site3.

- **A cluster of three sites (site1, site2, site3), with "site_replication_factor = origin:2, site1:1, site2:1, total:4":** Site1 stores two copies of any bucket it is originating and one copy of any other bucket. Site2 follows the same pattern. Site3, whose site value is not explicitly defined, follows the same pattern.

- **A cluster of three sites (site1, site2, site3), with "site_replication_factor = origin:2, site1:1, site2:2, total:5":** Site1 stores two copies of any bucket it originates, one or two copies of any bucket site2 originates, and one copy of any bucket that site3 originates. Site2 stores two copies of any bucket, whether or not it originates it. Site3, whose site value is not explicitly defined, stores two copies of any bucket it originates, one copy of any bucket site1 originates, and one or two copies of any bucket site2 originates. (When site2 originates a bucket, one copy remains after initial assignments. The manager assigns this randomly to site1 or site3.)

- **A cluster of three sites with "site_replication_factor = origin:1, total:4":** Four copies of each bucket are distributed randomly across all sites, with each site getting at least one copy.

Handle the persistence of the single-site replication_factor

Important: Although the `site_replication_factor` effectively replaces the single-site `replication_factor`, the single-site attribute continues to exist in the manager's configuration, with its default value of 3. This can cause problems on start-up if any site has fewer than three peer nodes. The symptom will be a message that the multisite cluster does not meet its replication factor. For example, if one of your sites has only two peers, the default single-site replication factor of 3 will cause the error to occur. To avoid or fix this problem, you must set the single-site replication factor to a value that is less than or equal to the smallest number of peers on any site. In the case where one site has only two peers, you must therefore explicitly set the `replication_factor` attribute to a value of 2. See "Multisite cluster does not meet its replication or search factors."

Configure the site search factor

Read this first

Before attempting to configure the site search factor, you must be familiar with:

- The basic, single-site search factor. See "The basics of cluster architecture" and "Search factor".
- The site replication factor. See "Configure the site replication factor".
- Multisite cluster configurations. See "Configure multisite indexer clusters with server.conf".

What is a site search factor?

To implement multisite indexer clustering, you must configure a site search factor. This replaces the standard search factor, which is specific to single-site deployments. You specify the site search factor on the manager node, as part of the basic configuration of the cluster.

The site search factor provides site-level control over the location of searchable bucket copies, in addition to providing control over the total number of searchable copies across the entire cluster. For example, you can specify that a two-site cluster maintain a total of three searchable copies of all buckets, with one site maintaining two copies and the second site maintaining one copy.

You can also specify a search policy based on which site originates the bucket. That is, you can configure the search factor so that a site receiving external data maintains a greater number of searchable copies of buckets for that data than for data that it does not originate. For example, you can specify that each site maintains two searchable copies of all data that it originates but only one copy of data originating on another site.

The site search factor helps determine whether the cluster has search affinity. See "Implement search affinity in a multisite indexer cluster".

Syntax

The syntax for `site_search_factor` and `site_replication_factor` are identical, except for the additional requirement that the values and explicit sites in `site_search_factor` be a subset of those in `site_replication_factor`. This section describes the syntax in full detail.

You configure the site search factor with the `site_search_factor` attribute in the manager's server.conf file. The attribute resides in the `[clustering]` stanza, in place of the single-site `search_factor` attribute. For example:

```
[clustering]
mode = manager
multisite=true
available_sites=site1,site2
site_replication_factor = origin:2,total:3
site_search_factor = origin:1,total:2
```

You can also use the CLI to configure the site search factor. See "Configure multisite indexer clusters with the CLI".

Warning: You must configure the `site_search_factor` attribute correctly. Otherwise, the manager will not start.

Here is the formal syntax:

```
site_search_factor = origin:<n>, [site1:<n>,] [site2:<n>,] ..., total:<n>
```

where:

- `<n>` is a positive integer indicating the number of searchable copies of a bucket.
- `origin:<n>` specifies the minimum number of searchable copies of a bucket that will be held on the site originating the data in that bucket (that is, the site where the data first entered the cluster). When a site is originating the data, it is known as the "origin" site.
- `site1:<n>, site2:<n>, ...,` indicates the minimum number of searchable copies that will be held at each specified site. The identifiers "site1", "site2", and so on, are the same as the `site` attribute values specified on the peer nodes.
- `total:<n>` specifies the total number of searchable copies of each bucket, across all sites in the cluster.

Note the following:

- This attribute specifies the per-site searchable copy policy. It is specified globally and applies to all buckets in all indexes.
- This attribute is valid only if `mode=manager` and `multisite=true`. Under those conditions, it supersedes any `search_factor` attribute.
- The `origin` and `total` values are required.

- Site values (`site1:<n>, site2:<n>, ...`) are optional. A site that is specified here is known as an "explicit" site. A site that is not specified is known as a "non-explicit" site.

- To determine the minimum number of searchable copies a site gets, use the same rules as for determining the minimum number of replicated copies a site gets through `site_replication_factor`. See "Configure the site replication factor".

- To determine the minimum required value for `total`, use the same rules as for determining the minimum `total` value for the `site_replication_factor`. See "Configure the site replication factor".

- Because the `total` value can be greater than the total set of explicit values, the cluster needs a strategy to handle any "remainder" searchable bucket copies. The strategy follows the strategy for remainder replicated copies, described in "Configure the site replication factor".

- All values must be less than or equal to their corresponding values in the `site_replication_factor`.

  For example, if you have a three-site cluster with "site_replication_factor = origin:2, site1:1, site2:2, total:5", then, in `site_search_factor`, the `origin` value cannot exceed 2, the `site1` value cannot exceed 1, the `site2` value cannot exceed 2, and the `total` value cannot exceed 5.

- If a site value is explicit in `site_search_factor`, it must also be explicit in `site_replication_factor`. However, an explicit site value in `site_replication_factor` does not need be explicit in `site_search_factor`.

  For example, if you have a three-site cluster with "site_replication_factor = origin:2, site1:1, site2:2, total:5" (with a non-explicit site3), you can specify "site_search_factor = origin:1, site2:2, total:4" (removing the explicit site1), but you cannot specify "site_search_factor = origin:1, site1:1, site2:2, site3:1, total:4" (making the non-explicit site3 explicit).

- For search affinity, you must configure the `site_search_factor` so that you have at least one searchable copy on each site where you require search affinity. Only explicit sites adhere to search affinity.

- If you are migrating from a single-site cluster, the `total` value must be at least as large as the `search_factor` for the single-site cluster. See "Migrate an indexer cluster from single-site to multisite".

- The attribute defaults to: "origin:1, total:2."

Examples

For examples of site search factor syntax, refer to the examples in "Configure the site replication factor". The syntax for specifying origin/site/total values in `site_search_factor` is identical to `site_replication_factor`.

Handle the persistence of the single-site search_factor

Important: Although the `site_search_factor` effectively replaces the single-site `search_factor`, the single-site attribute continues to exist in the manager's configuration, with its default value of 2. This can cause problems on start-up if any site has only one peer node. The symptom will be a message that the multisite cluster does not meet its search factor. To avoid or fix this problem, you must set the single-site search factor to a value that is less than or equal to the smallest number of peers on any site. In the case where one site has only one peer, you must therefore explicitly set the `search_factor` attribute to a value of 1. See "Multisite cluster does not meet its replication or search factors."

Migrate an indexer cluster from single-site to multisite

You can migrate an indexer cluster from single-site to multisite. During this process, you incorporate your existing single-site cluster into a new multisite cluster.

Post-migration bucket behavior

After migration, all buckets created before migration continue to adhere to their single-site replication and search factor policies by default. You can change this behavior so that legacy buckets adhere instead to the multisite policies.

Buckets created after migration always adhere to the multisite policies.

Maintain legacy buckets as single-site

By default, after migration, the cluster holds both single-site and multisite buckets. It maintains them separately, following these rules:

- Single-site legacy buckets (those existing at the time of migration) continue to respect the single-site `replication_factor` and `search_factor` settings.
- Multisite buckets (those created after migration) follow the multisite `site_replication_factor` and `site_search_factor` policies.

Convert legacy buckets to multisite

You can configure the manager node to convert legacy buckets to multisite. This process causes buckets that were following the single-site replication and search policies, pre-migration, to adhere instead to the multisite replication and search policies.

When deciding whether to convert legacy buckets, you must weigh the value of maintaining those buckets across multiple sites against the possibly considerable time sink required for the bucket fixup activity needed to convert them to muitisite.

You can make the necessary configuration change either before migration or at any point after migration.

If you change the configuration before migration, legacy buckets will follow the `site_replication_factor` and `site_search_factor` policies immediately post-migration.

If you change the configuration post-migration, any pre-migration buckets that have been following the single-site policies will then follow the multisite policies.

To see how many buckets will require conversion to multisite, use `services/cluster/manager/buckets?filter=multisite_bucket=false&filter=standalone=false` before changing the manager node configuration.

Configure the manager to convert legacy buckets to multisite

To cause legacy single-site buckets to adhere to the multisite replication and search factor policies, change the `constrain_singlesite_buckets` setting in the manager's `server.conf` file to "false":

```
[clustering]
mode = manager
```

```
constrain_singlesite_buckets = false
```
You must restart the manager node for the change to take effect.

Perform the multisite migration

Prerequisites

- The manager node must be running Splunk Enterprise version 7.2 or later.
- All nodes in the post-migration cluster must adhere to the version compatibility rules described in Splunk Enterprise version compatibility. Therefore, before migrating to multisite, you might need to upgrade your single-site cluster. Follow the appropriate procedure in Upgrade an indexer cluster.
- If you want existing buckets to adhere to the multisite replication and search policies post-migration, you must change a configuration on the manager node. See Configure the manager to convert existing buckets to multisite. Alternatively, you can perform this step at any time post-migration.

Steps

To migrate a single-site cluster to multisite, configure each node for multisite:

1. Configure the manager node for multisite and restart it, following the instructions in Configure multisite indexer clusters with the CLI. For example:

```
splunk edit cluster-config -mode manager -multisite true -available_sites site1,site2 -site site1
-site_replication_factor origin:2,total:3 -site_search_factor origin:1,total:2

splunk restart
```
Note the following:

- Do not remove the existing single-site attributes for replication factor and search factor, `replication_factor` and `search_factor`. The manager needs them to handle the migrated buckets.
- The `total` values for `site_replication_factor` and `site_search_factor` must be at least as large as `replication_factor` and `search_factor`, respectively.
- If the number of peers on any site is less than the single-site `replication_factor` or `search_factor`, you must reduce the values of those attributes to match the least number of peers on any site. For example, if `replication_factor` is 3 and `search_factor` is 2, and one of the sites has only 2 peers, you must change `replication_factor` to 2. Otherwise, the migrated buckets might not meet the replication and search factors, due to the way the cluster replicates migrated buckets. See Multisite cluster does not meet its replication or search factors.

2. Set maintenance mode on the manager:

```
splunk enable maintenance-mode
```
This step prevents unnecessary bucket fix-ups. See Use maintenance mode.

To confirm that the manager has entered maintenance mode, run `splunk show maintenance-mode`.

3. Configure the existing peer nodes for multisite. For each peer, specify its manager node and site. For example:

```
splunk edit cluster-config -site site1
```
You will be prompted to restart the peer.

Do this for each peer, specifying the site for that peer.

4. If you want to add new peers to the cluster, follow the instructions in Configure multisite indexer clusters with the CLI. For example:

```
splunk edit cluster-config -mode peer -site site1 -manager_uri https://10.160.31.200:8089 -replication_port
9887

splunk restart
```
Do this for each new peer that you want to add to the cluster.

5. Configure the search heads for multisite. For each search head, specify its manager node and site. For example:

```
splunk edit cluster-manager https://10.160.31.200:8089 -site site1
```
Do this for each search head, specifying the site for that search head.

Note: The configuration is essentially the same if the search heads are members of a search head cluster. See Integrate with a multisite indexer cluster in *Distributed Search.*

6. If you want to add new search heads to the cluster, follow the instructions in Configure multisite indexer clusters with the CLI. For example:

```
splunk edit cluster-config -mode searchhead -site site1 -manager_uri https://10.160.31.200:8089

splunk restart
```
Do this for each new search head that you want to add to the cluster.

7. Disable maintenance mode on the manager:

```
splunk disable maintenance-mode
```
To confirm that the manager has left maintenance mode, run `splunk show maintenance-mode`.

You can view the manager node dashboard to verify that all cluster nodes are up and running.

During the migration, the cluster tags each single-site bucket with a site value.

Note: You can also configure a multisite cluster by directly editing `server.conf`. See Configure multisite indexer clusters with server.conf

8. If you are using indexer discovery to connect forwarders to the peer nodes, you must assign a site to each forwarder. See Use indexer discovery in a multisite cluster.

If you configured the manager to convert existing single-site buckets to the multisite replication and search factor policies, bucket fixup will likely continue for some time after the cluster migration process itself completes. If you have a large number of existing buckets, the bucket fixup can continue for a long time.

How the cluster migrates and maintains existing buckets

Buckets in multisite clusters include a property that identifies the origin site. Buckets in single-site clusters do not include that property. So, when a cluster migrates from single-site to multisite, it must tag each single-site bucket with an origin site value. Since the bucket name includes the GUID of the originating peer, the cluster always knows the originating peer. With that information, it infers an origin site for the bucket:

- If the originating peer still exists in the cluster, the cluster assumes that the bucket originated on the site that the originating peer has been assigned to. It sets the bucket's origin to that site.
- If the originating peer is no longer in the cluster, the cluster assumes that the site with the most copies of the bucket is the origin site. It sets the bucket's origin to that site.

If the cluster is configured to maintain existing buckets as single-site

Here is how the cluster uses the inferred origin site to maintain the single-site bucket going forward, to handle any necessary fix-up so that the bucket continues to meet the single-site replication and search factors:

- If the cluster needs to replicate additional copies of the bucket to fulfill the replication factor, it only replicates within the bucket's inferred origin site.
- If the cluster needs to make a non-searchable copy of the bucket searchable to fulfill the search factor, it might do so on a non-origin site, if a non-searchable copy of that bucket already exists on some other site.

The cluster will never create a new copy of the bucket on a non-origin site.

Because, when pre-migration buckets are maintained as single-site, the cluster creates new copies of those buckets only on the origin site, it is possible to experience a situation where the replication factor cannot be met. For example, if replication_factor is set to 3, but there are only two peer nodes on the origin site, the cluster will not create a third copy on a non-origin site, even if there had previouly been a third copy of the bucket on a non-origin site. To remediate this situation, you can either convert pre-migration buckets to multisite or you can reduce the single-site replication factor so that it does not exceed the number of peer nodes on any site.

If the cluster is configured to convert existing buckets to multisite

The cluster uses the described methodology to infer an origin site for each bucket.

The cluster's process of converting buckets from adherence to the single-site policies to adherence to the multisite policies is the same as any bucket-fixup process, involving cross-site streaming and such. If you have a large number of existing buckets, the process can take a long time to complete. This fixup process has the same priority as any other concurrent fixup processes.

While fixup is continuing, the manager node dashboard indicates that the replication factor and search factors are not met. Once the fixup process is finished, the cluster returns to a complete state, as indicated by the manager node dashboard.

Handle blocked indexing on the new site

If you configure the manager node for multisite clustering, but the new site is not yet fully operational, the manager blocks indexing while it waits for enough peers to become available to fulfill the multisite replication factor. To unblock indexing, you can run the `splunk set indexing-ready` command on the manager. See Restart indexing in multisite cluster after manager restart or site failure.

View indexer cluster status

View the manager node dashboard

This dashboard provides detailed information on the status of the entire indexer cluster. You can also get information on each of the manager's peer nodes from here.

For information on the other clustering dashboards, read:

- "View the peer dashboard"
- "View the search head dashboard"

Access the manager node dashboard

1. Click **Settings** on the upper right side of Splunk Web.
2. In the **Distributed Environment** group, click **Indexer clustering**.
   The Manager Node dashboard appears.

You can only view this dashboard on an instance that has been enabled as a manager.

View the manager node dashboard

The manager node dashboard contains these sections:

- Cluster overview
- Peers tab
- Indexes tab
- Search Heads tab

Cluster overview

The cluster overview summarizes the health of your cluster. It tells you:

- whether the cluster's data is fully searchable; that is, whether all buckets in the cluster have a **primary** copy.
- whether the search and replication factors have been met.
- how many peers are searchable.
- how many indexes are searchable.

Depending on the health of your cluster, it might also provide warning messages such as:

- Some data is not searchable.
- Replication factor not met.
- Search factor not met.

For details on the information presented in the cluster overview, browse the tabs underneath.

On the upper right side of the dashboard, there are three buttons:

- **Edit.** For information on this button, see Configure the manager node with the dashboard..

- **More Info.** This button provides details on the manager node configuration:
  - **Name**. The manager's `serverName`, as specified in the manager's `$SPLUNK_HOME/etc/system/local/server.conf` file.
  - **Replication Factor.** The cluster's **replication factor**.
  - **Search Factor.** The cluster's **search factor**.
  - **Generation ID.** The cluster's current **generation ID**.
- **Documentation.**

Peers tab

For each peer, the manager node dashboard lists:

- **Peer Name.** The peer's `serverName`, as specified in the peer's `$SPLUNK_HOME/etc/system/local/server.conf` file.
- **Fully searchable.** This column indicates whether the peer currently has a complete set of primaries and is fully searchable.
- **Site.** (For multisite only.) This column displays the site value for each peer.
- **Status.** The peer's status. For more information about the processes discussed here, see Take a peer offline. Possible values include:
  - `Up`
  - `Pending`. This occurs when a replication fails. It transitions back to Up on the next successful heartbeat from the peer to the manager.
  - `AutomaticDetention`. A peer goes into this state when it runs low on disk space. While in this state, a peer does not perform its normal functions. For details, see Put a peer in detention.
  - `ManualDetention`. A peer goes into this state through manual intervention. While in this state, a peer does not perform most of its normal functions. For details, see Put a peer in detention.
  - `ManualDetention-PortsEnabled`. A peer goes into this state through manual intervention. While in this state, a peer continues to consume and index external data, but it does not serve as a replication target. It continues to participate in searches. See Put a peer in detention.
  - `Restarting`. When you run the `splunk offline` command without the `enforce-counts` flag, the peer enters this state temporarily after it leaves the `ReassigningPrimaries` state. It remains in this state for the `restart_timeout` period (60 seconds by default). If you do not restart the peer within this time, it then moves to the `Down` state. The peer also enters this state during rolling restarts or if restarted via Splunk Web.
  - `ShuttingDown`. The manager detects that the peer is shutting down.
  - `ReassigningPrimaries`. A peer enters this state temporarily when you run the `splunk offline` command without the `enforce-counts` flag.
  - `Decommissioning`. When you run the `splunk offline` command with the `enforce-counts` flag, the peer enters this state and remains there until all bucket fixing is complete and the peer can shut down.
  - `GracefulShutdown`. When you run the `splunk offline` command with the `enforce-counts` flag, the peer enters this state when it finally shuts down at the end of a successful decommissioning. It remains in this state for as long as it is offline.
  - `Stopped`. The peer enters this state when you stop it with the `splunk stop` command.
  - `Down`. The peer enters this state when it goes offline for any reason other than those resulting in a status of `GracefulShutdown` or `Stopped`: either you ran the version of the `splunk offline` command without the `enforce-counts` flag and the peer shut down for longer than the `restart_timeout` period (60 seconds by default), or the peer went offline for some other reason (for instance, it crashed).
- **Buckets.** The number of buckets for which the peer has copies.

To get more information for any peer, click on the arrow to the left of the peer name. These fields appear:

- **Location.** The peer's IP address and port number.
- **Last Heartbeat.** The time of the last heartbeat the manager received from the peer.
- **Replication port.** The port on which the peer receives replicated data from other peers.
- **Base generation ID.** The peer's base generation ID, which is equivalent to the cluster's generation ID at the moment that the peer last joined the cluster. This ID will be less or equal to the cluster's current generation ID. So, if a peer joined the cluster at generation 1 and has stayed in the cluster ever since, its base generation ID remains 1, even though the cluster might have incremented its current generation ID to, say, 5.
- **GUID.** The peer's GUID.

Note: After a peer goes down, it continues to appear on the list of peers, although its status changes to "Down" or "GracefulShutdown." To remove the peer from the manager's list, see Remove a peer from the manager node's list.

Indexes tab

For each index, the manager node dashboard lists:

- **Index Name.** The name of the index. Internal indexes are preceded by an underscore (_).
- **Fully searchable.** Is the index fully searchable? In other words, does it have at least one searchable copy of each bucket? If even one bucket in the index does not have a searchable copy, this field will report the index as non-searchable.
- **Searchable Data Copies.** The number of complete searchable copies of the index that the cluster has.
- **Replicated Data Copies.** The number of copies of the index that the cluster has. Each copy must be complete, with no buckets missing.
- **Buckets.** The number of buckets in the index. This number does not include replicated bucket copies.
- **Cumulative Raw Data Size.** The size of the index's raw data, excluding hot buckets. This number does not included replicated copies of the raw data.

The list of indexes include the internal indexes, **_audit** and **_internal**. As you would expect in a cluster, these internal indexes contain the combined data generated by *all* peers in the cluster. If you need to search for the data generated by a single peer, you can search on the peer's host name.

This tab also reveals a button with the label, **Bucket Status.** If you click on it, you go to the Bucket Status dashboard. See View the bucket status dashboard..

Note: A new index appears here only after it contains some data. In other words, if you configure a new index on the peer nodes, a row for that index appears only after you send data to that index.

Search heads tab

For each search head accessing this cluster, the manager node dashboard lists:

- **Search head name.** The search head's `serverName`, as specified in its `$SPLUNK_HOME/etc/system/local/server.conf` file.
- **Site.** (For multisite only.) This column displays the site value for each search head.
- **Status.** Is the search head up or down? The manager decides that a search head is down if the search head does not poll the manager for generation information within a period twice the length of the `generation_poll_interval`. That attribute is configurable in `server.conf`.

Note: The list includes the manager node as one of the search heads. Although the manager has search head capabilities, you should only use those capabilities for debugging purposes. The resources of the manager must be dedicated to fulfilling its critical role of coordinating cluster activities. Under no circumstances should the manager be employed as a production search head. Also, unlike a dedicated search head, the search head on the manager cannot be

configured for multi-cluster search; it can only search its own cluster.

To get more information for any search head, click on the arrow to the left of the search head name. These fields appear:

- **Location.** The search head's server name and port number.
- **GUID.** The search head's GUID.

View the bucket status dashboard

The Bucket Status dashboard provides status for the buckets in the cluster. It contains three tabs:

- Fixup Tasks - In Progress
- Fixup Tasks - Pending
- Indexes with Excess Buckets

Fixup Tasks - In Progress

This tab provides a list of buckets that are currently being fixed. For example, if a bucket has too few copies, fixup activities must occur to return the cluster to a **valid** and **complete** state. While those activities are occurring, the bucket appears on this list.

Fixup Tasks - Pending

This tab provides a list of buckets that are waiting to be fixed. You can filter the fixup tasks by search factor, replication factor, and generation.

For more information on bucket fixup activities, see What happens when a peer goes down.

This tab also includes an Action button that allows you to fix issues with individual buckets. For details, see Handle issues with individual buckets.

Indexes with Excess Buckets

This tab provides a list of indexes with excess bucket copies. It enumerates both buckets with excess copies and buckets with excess searchable copies. It also enumerates the total excess copies in each category. For example, if your index "new" has one bucket with three excess copies, one of which is searchable, and a second bucket with one excess copy, which is non-searchable, the row for "new" will report:

- 2 buckets with excess copies
- 1 bucket with excess searchable copies
- 4 total excess copies
- 1 total excess searchable copies

If you want to remove the excess copies for a single index, click the **Remove** button on the right side of the row for that index.

If you want to remove the excess copies for all indexes, click the **Remove All Excess Buckets** button.

For more information on excess bucket copies, see Remove excess bucket copies from the indexer cluster.

Use the monitoring console to view status

You can use the monitoring console to monitor most aspects of your deployment, including the status of your indexer cluster. The information available through the console duplicates much of the information available on the manager node dashboard.

For more information, see Use the monitoring console to view indexer cluster status.

View the peer dashboard

The indexer cluster peer dashboard provides detailed information on the status of a single peer node.

For a single view with information on all the peers in a cluster, use the manager node dashboard instead, as described in "View the manager node dashboard".

Access the peer dashboard

To view the peer dashboard:

1. Click **Settings** on the upper right side of Splunk Web.

2. In the **Distributed Environment** group, click **Indexer clustering**.

You can only view this dashboard on an instance that has been enabled as a peer.

View the dashboard

The dashboard provides information on the peer's status:

- **Name**. The peer's `serverName`, as specified in its `$SPLUNK_HOME/etc/system/local/server.conf` file.
- **Replication port.** The port on which the peer receives replicated data from other peers.
- **Manager location.** The manager node's IP address and port number.
- **Base Generation ID.** The peer's base generation ID, which is equivalent to the cluster's **generation ID** at the moment that the peer last joined the cluster. This ID will be less or equal to the cluster's current generation ID. So, if a peer joined the cluster at generation 1 and has stayed in the cluster ever since, its base generation ID remains 1, even though the cluster might have incremented its current generation ID to, say, 5.

Configure the peer

The **Edit** button at the top right of the peer dashboard offers several options for changing the configuration of the peer. See "Configure peer nodes with the dashboard."

Note: The **Edit** button is disabled for multisite clusters.

View the search head dashboard

This dashboard provides detailed information on the status of the search head in an indexer cluster.

Access the dashboard

To access the dashboard:

1. Click **Settings** on the upper right side of Splunk Web.

2. In the **Distributed Environment** group, click **Indexer clustering**.

You can only view this dashboard on an instance that has been enabled as a cluster search head.

View the dashboard

The dashboard lists the manager nodes for all clusters the search head belongs to, along with some information on the status of each cluster.

For more information on the manager node and its cluster, click the arrow at the far left of each row.

You can get information on the search head itself by selecting the **More Info** button on the upper right corner of the dashboard:

- **Name**. The search head's `serverName`, as specified in its `$SPLUNK_HOME/etc/system/local/server.conf` file.

Configure the search head

The dashboard offers several options for acting on the search head or otherwise changing its configuration. See "Configure the search head with the dashboard."

View information on search peers

You can also view information on the search head's **search peers** (identical, in clustering, to the set of cluster peer nodes) from the search head's Distributed Search page in Splunk Web:

1. On the search head, click **Settings** in the upper right corner of Splunk Web.

2. In the **Distributed environment** section, click **Distributed search**.

3. Click **Search peers** to view the set of search peers.

Caution: Do not use the Distributed Search page in Splunk Web to change your search head configuration or to add peers. For information on how to configure an indexer cluster search head correctly, see "Search head configuration overview".

Use the monitoring console to view indexer cluster status

You can use the monitoring console to monitor most aspects of your deployment. This topic discusses the console dashboards that provide insight into indexing performance.

The primary documentation for the monitoring console is located in *Monitoring Splunk Enterprise*.

There are two indexer clustering dashboards under the **Indexing** menu:

- Indexer Clustering: Status
- Indexer Clustering: Service Activity

The Indexer Clustering: Status dashboard provides information on the state of your cluster. To a large extent, it duplicates the information in the manager node dashboard, described in "View the manager node dashboard".

The Indexer Clustering: Service Activity dashboard provides information on matters such as bucket-fixing activities and warnings and errors.

View the dashboards themselves for more information. In addition, see "Indexer Clustering: Status" and "Indexer Clustering: Service Activity" in the *Distributed Management Console Manual*.

Manage the indexer cluster

Add a peer to the cluster

You can add peer nodes to the indexer cluster at any time. To do so, just enable a Splunk Enterprise instance as a peer node.

Before you enable a peer node, familiarize yourself with the relevant system requirements, detailed in System requirements and other deployment considerations for indexer clusters. Ensure that the Splunk Enterprise instance meets the version compatibility and all other documented requirements.

For single-site clusters, follow one of these procedures:

- To enable a peer node with the Splunk Web dashboard, see Enable the peer nodes.
- To enable a peer node with the CLI , see Configure peer nodes with the CLI.
- To enable a peer node by editing `server.conf`, see Configure peer nodes with server.conf.

To enable a peer node for a multisite cluster, see Configure multisite indexer clusters with server.conf.

When the peer starts up, the manager node distributes the latest configuration bundle to the peer. This process ensures that the new peer has the same set of configurations, including the set of indexes and other index settings, as the other peer nodes. See Distribution of the bundle when a peer starts up.

After you enable the peer node, you might need to take steps to ensure that the peer is receiving data from forwarders. There are a variety of ways to configure the relationship between forwarders and peer nodes. Depending on how your cluster connects to forwarders, you might need to configure either the new peer node or the forwarders, or both. For details, read the topics in the chapter Get data into the indexer cluster.

As a final step, consider rebalancing the cluster's data, so that existing data gets moved onto the new peer node. This allows the new peer to share the search burden. See Rebalance indexer cluster data.

Take a peer offline

Use the `splunk offline` command to take a peer offline.

Caution: Do not use `splunk stop` to take a peer offline. Instead, use `splunk offline`. It stops the peer in a way that minimizes disruption to your searches.

Depending on your needs, you can take a peer offline permanently or temporarily. In both cases, the cluster performs actions to regain its **valid** and **complete** states:

- A **valid** cluster has **primary** copies of all its buckets and therefore is able to handle search requests across the entire set of data. In the case of a multisite cluster, a valid cluster also has primary copies for every site with search affinity.
- A **complete** cluster has **replication factor** number of copies of all its buckets, with **search factor** number of searchable copies. It therefore meets the designated requirements for failure tolerance. A complete cluster is also a valid cluster.

Offline use cases

The `splunk offline` command has two versions, one that takes the peer offline temporarily and another that takes it offline permanently.

When you take a peer offline temporarily

When you take a peer offline temporarily, it is usually to perform a short-term maintenance task, such as a machine or operating system upgrade. You want the cluster to continue to process data and handle searches without interruption, but it is acceptable if the cluster does not meet its replication factor or search factor during the time that the peer is offline.

When a peer goes offline temporarily, the manager node kicks off processes to return the cluster to a valid state, but it does not ordinarily attempt to return the cluster to a complete state, because the cluster regains its complete state as soon as the peer comes back online.

When you take a peer offline permanently

When you take an indexer cluster peer offline permanently, you want to ensure that the cluster continues to process data and handle searches without interruption. You also want the cluster to replace any copies of buckets that are lost by the peer going offline. For example, if the offline peer was maintaining copies of 10 buckets (three searchable and seven non-searchable), the cluster must recreate those copies on other peers in the cluster, to fulfill its replication factor and search factor.

When a peer goes offline permanently, the manager node kicks off various bucket-fixing processes, so that the cluster returns to a valid and complete state.

The splunk offline command

The `splunk offline` command handles both types of peer shutdown: temporary and permanent. It takes the peer down gracefully, attempting to allow in-progress searches to complete, while also returning the cluster quickly to the valid state. In this way, it tries to eliminate disruption to existing or future searches.

The `splunk offline` command also initiates remedial bucket-fixing activities to return the cluster to a complete state. Depending on the version of the command that you run, it will start this process either immediately or after waiting a specified period of time, to give the peer time to come back on line and avoid the need for bucket-fixing.

There are two versions of the `splunk offline` command that correspond to the typical use cases:

- `splunk offline`. Used to take a peer down temporarily for maintenance operations. Also known as the "fast offline" command.
- `splunk offline --enforce-counts`. Used to remove a peer permanently from the cluster. Also known as the "enforce-counts offline" command.

Take a peer down temporarily: the fast offline command

The fast version of the `splunk offline` command has the simple syntax: `splunk offline`.

The cluster attempts to regain its valid state before the peer goes down. It does not attempt to regain its complete state. You can use this version to bring the peer down briefly without kicking off any bucket-fixing activities.

You can also use this version in cases where you want the peer to go down permanently but quickly, with the bucket-fixing occurring after it goes down.

The fast offline process

The peer goes down after the cluster attempts, within certain constraints, to meet two conditions:

- Reallocation of primary copies on the peer, so that the cluster regains its valid state
- Completion of any searches that the peer is currently participating in

The peer goes down after its primary bucket copies have been reallocated to searchable copies on other peers, so that the cluster regains its valid state. The maximum time period allotted for the primary allocation activity is determined by the value of the `decommission_node_force_timeout` setting in the peer node's `server.conf` file, which is five minutes by default.

You can also change the primary allocation timeout period for just a single execution of the `splunk offline` command. See Syntax for the fast offline command.

Note: If the cluster has a search factor of 1, the cluster does not attempt to reallocate primary copies before allowing the peer to go down. With a search factor of 1, the cluster cannot fix the primaries without first creating new searchable copies, which takes significant time and thus would defeat the goal of a fast shutdown.

The peer also waits for any ongoing searches to complete, up to a maximum time period, as determined by the `decommission_search_jobs_wait_secs` attribute in `server.conf`. The default for this attribute is three minutes.

Once these conditions have been met, or the maximum durations for the activities have been exceeded, the peer goes down.

Syntax for the fast offline command

Here is the syntax for the fast version of the `splunk offline` command:

```
splunk offline
```
You run this command directly on the peer.

When you run this command, the peer shuts down after the cluster returns to a valid state and the peer completes any ongoing searches, as described in The fast offline process.

To change the time period allocated for the cluster to return to the valid state, run this form of the command:

```
splunk offline --decommission_node_force_timeout <seconds>
```
This changes the primary allocation timeout period for the current offline operation only. For example, to change the timeout period to 10 minutes for the current operation:

```
splunk offline --decommission_node_force_timeout 600
```
Any future offline operations for this peer will use the value for the setting saved in `server.conf`, which is five minutes by default.

After the peer shuts down, you have 60 seconds (by default) to complete any maintenance work and bring the peer back online by invoking the `splunk restart` command. If the peer does not return to the cluster within this time, the manager

node initiates bucket-fixing activities to return the cluster to a complete state. If you need more time, you can extend the time that the manager waits for the peer to come back online by configuring the `restart_timeout` attribute, as described in Extend the restart period.

Important: To minimize any bucket-fixing activities, you should ordinarily take down only one peer at a time. If you are performing an operation that involves taking many peers offline temporarily, consider invoking maintenance mode during the operation. See Use maintenance mode.

For detailed information on the processes that occur when a peer goes offline, read What happens when a peer node goes down.

Extend the restart period

If you need to perform maintenance on a peer and you expect the time required to exceed the manager's `restart_timeout` period (set to 60 seconds by default), you can change the value of that setting. Run this CLI command on the manager node:

```
splunk edit cluster-config -restart_timeout <seconds>
```

For example, this command resets the timeout period to 900 seconds (15 minutes):

```
splunk edit cluster-config -restart_timeout 900
```

You can run this command on the fly. You do not need to restart the manager after it runs.

You can also change this value in `server.conf` on the manager node.

Take a peer down permanently: the enforce-counts offline command

The enforce-counts version of the offline command is intended for use when you want to take a peer offline permanently, but only after the cluster has returned to its complete state.

In this version of the command, the cluster performs the necessary bucket-fixing activities to regain its valid and complete state before allowing the peer to go down.

Caution: The enforce-counts version of the offline command ensures that the offlined peer's replicated data is available on other peer nodes in the cluster. However, it does not consider, or in any way handle, any standalone buckets that might exist on the peer node, due to a possible earlier migration of the node from a non-clustered indexer into the cluster, as described in Migrate non-clustered indexers to a clustered environment. Therefore, if you need to retain data residing in standalone buckets, you must take other steps separate from the offline process.

The enforce-counts offline process

The peer goes down after the cluster meets two conditions:

- Completion of all bucket-fixing activities that are necessary for the cluster to regain its complete state
- Completion of ongoing searches that the peer is participating in, constrained by a time limit

The peer goes down only after its searchable and non-searchable bucket copies have been reallocated to other peers, causing the cluster to regain its complete state.

Because this version of `splunk offline` requires that the cluster return to a complete state before the peer can go down, certain preconditions are necessary before you can run this command:

- The cluster must have (replication factor + 1) number of peers, so that it can reallocate bucket copies to other peers as necessary and can continue to meet its replication factor after the peer goes down.
- In a multisite cluster, the peer's site must have enough peers so that the site continues to fulfill the requirements of its site replication factor, in terms of the number of origin or explicit peers.
- The cluster cannot be in maintenance mode, because bucket fixup does not occur during maintenance mode.

The peer also waits for any ongoing searches to complete, up to a maximum time, as determined by the `decommission_search_jobs_wait_secs` attribute in `server.conf`. The default for this attribute is three minutes.

Syntax for the enforce-counts offline command

Here is the syntax for the enforce-counts version of the `splunk offline` command:

```
splunk offline --enforce-counts
```

You run this command directly on the peer.

This version of the command initiates an operation called **decommissioning**, during which the manager coordinates a wide range of remedial processes. The peer does not shut down until those processes finish and the cluster returns to a complete state. This can take quite a while if the peer is maintaining a large set of bucket copies.

The actual time required to return to the complete state depends on the amount and type of data the peer was maintaining. See Estimate the cluster recovery time when a peer gets decommissioned for details.

If the cluster is unable to return to the complete state, the peer will not shut down. This is due to issues described in The enforce-counts offline process. If you need to take a peer offline despite such issues, run the fast version of the `splunk offline` command instead.

For detailed information on the processes that occur when a peer gets decommissioned, read What happens when a peer node goes down.

After a peer goes down, it continues to appear on the list of peers on the manager node dashboard, although its status changes to "GracefulShutdown." To remove the peer from the manager's list, see Remove a peer from the manager node's list.

Estimate the cluster recovery time when a peer gets decommissioned

When you decommission a peer, the manager node coordinates activities among the remaining peers to fix the buckets and return the cluster to a complete state. For example, if the peer going offline is storing copies of 10 buckets and five of those copies are searchable, the manager instructs peers to:

- Stream copies of those 10 buckets to other peers, so that the cluster regains a full complement of bucket copies (to match the replication factor).
- Make five non-searchable bucket copies searchable, so that the cluster regains a full complement of searchable bucket copies (to match the search factor).

This activity can take some time to complete. Exactly how long depends on many factors, such as:

- **System considerations**, such as CPU specifications, storage type, interconnect type.

- **Amount of other indexing currently being performed** by the peers that are tasked with making buckets searchable.
- **The size and number of buckets** stored on the offline peer.
- **The size of the index files** on the searchable copies stored on the offline peer. (These index files can vary greatly in size relative to rawdata size, depending on factors such as amount of segmentation.) For information on the relative sizes of rawdata and index files, read Storage considerations.
- **The search factor.** This determines how quickly the cluster can convert non-searchable copies to searchable. If the search factor is at least 2, the cluster can convert non-searchable copies to searchable by copying index files to the non-searchable copies from the remaining set of searchable copies. If the search factor is 1, however, the cluster must convert non-searchable copies by rebuilding the index files, a much slower process. (For information on the types of files in a bucket, see Data files.)

Despite these variable factors, you can make a rough determination of how long the process will take. Assuming you are using Splunk Enterprise reference hardware, here are some basic estimates of how long the two main activities take:

- To stream 10GB (rawdata and/or index files) from one peer to another across a LAN takes about 5-10 minutes.
- The time required to rebuild the index files on a non-searchable bucket copy containing 4GB of rawdata depends on a number of factors such as the size of the resulting index files, but 30 minutes is a reasonable approximation to start with. Rebuilding index files is necessary if the search factor is 1, meaning that there are no copies of the index files available to stream. A non-searchable bucket copy consisting of 4GB rawdata can grow to a size approximating 10GB once the index files have been added. As described earlier, the actual size depends on numerous factors.

Use maintenance mode

Maintenance mode halts most bucket fixup activity and prevents frequent rolling of hot buckets. It is useful when performing peer upgrades and other maintenance activities on an indexer cluster. Because it halts critical bucket fixup activity, use maintenance mode only when necessary.

Why use maintenance mode

Certain conditions can generate errors during hot bucket replication and cause the source peer to roll the bucket. While this behavior is generally beneficial to the health of the indexer cluster, it can result in many small buckets across the cluster, if errors occur frequently. Situations that can generate an unacceptable number of small buckets include persistent network problems or repeated offlining of peers.

To stop this behavior, you can temporarily put the cluster into maintenance mode. This can be useful for system maintenance work that generates repeated network errors, such as network reconfiguration. Similarly, if you need to upgrade your peers or otherwise temporarily offline several peers, you can invoke maintenance mode to forestall bucket rolling during that time.

Note: The CLI commands `splunk apply cluster-bundle` and `splunk rolling-restart` incorporate maintenance mode functionality into their behavior by default, so you do not need to invoke maintenance mode explicitly when you run those commands. A message stating that maintenance mode is running appears on the manager node dashboard.

The effect of maintenance mode on cluster operation

To prevent buckets from rolling unnecessarily, maintenance mode halts most bucket fix-up activity. The only bucket fix-up that occurs during maintenance mode is primary fixup. The manager node will attempt, when necessary, to reassign primaries to available searchable bucket copies.

In particular, the cluster does not perform fixup that entails replicating buckets or converting buckets from non-searchable to searchable. This means that the manager node does not enforce replication factor or search factor policy during maintenance mode. Therefore, if the cluster loses a peer node during maintenance mode, it can be operating under a valid but incomplete state. See **Indexer cluster states** to understand the implications of this.

Similarly, if the cluster loses peer nodes in numbers equal to or greater than the replication factor, it also loses its valid state for the duration of maintenance mode.

In addition, if the cluster loses even a single peer node while in maintenance mode, it can potentially return incomplete results for searches running during the subsequent period of primary fixup. This period is usually short, often just a few seconds, but even a short period of primary fixup can affect in-progress searches.

Maintenance mode works the same for single-site and multisite clusters. It has no notion of sites.

Enable maintenance mode

Put the cluster into maintenance mode before starting maintenance activity. Once you have finished with maintenance, you should disable maintenance mode.

To invoke maintenance mode, run this CLI command on the manager node:

```
splunk enable maintenance-mode
```

When you run the `enable` command, a message warning of the effects of maintenance mode appears and requires confirmation that you want to continue.

Effective with version 6.6, maintenance mode persists across manager node restarts.

Disable maintenance mode

To return to the standard bucket-rolling behavior, run:

```
splunk disable maintenance-mode
```

Determine maintenance mode status

To determine whether maintenance mode is on, run:

```
splunk show maintenance-mode
```

A returned value of `1` indicates that maintenance mode is on. `0` indicates that maintenance mode is off.

Restart the entire indexer cluster or a single peer node

This topic describes how to restart the entire indexer cluster (unusual) or a single peer node.

When you restart a manager or peer node, the manager rebalances the primary bucket copies across the set of peers, as described in Rebalance the indexer cluster primary buckets.

For information on configuration changes that require a restart, see Restart after modifying server.conf? and Restart or reload after configuration bundle changes?.

Restart the entire cluster

You ordinarily do not need to restart the entire cluster. If you change a manager's configuration, you restart just the manager. If you update a set of common peer configurations, the manager restarts just the set of peers, and only when necessary, as described in Update common peer configurations.

If, for any reason, you do need to restart both the manager and the peer nodes:

1. Restart the manager node, as you would any instance. For example, run this CLI command on the manager:

```
splunk restart
```

2. Once the manager restarts, wait until all the peers re-register with the manager, and the manager node dashboard indicates that all peers and indexes are searchable. See View the manager node dashboard.

3. Restart the peers as a group, by running this CLI command on the manager:

```
splunk rolling-restart cluster-peers
```

See Perform a rolling restart of an indexer cluster.

If you need to restart the search head, you can do so at any time, as long as the rest of the cluster is running.

Restart a single peer

You might occasionally have need to restart a single peer; for example, if you change certain configurations on only that peer.

Do not use the CLI `splunk restart` command to restart the peer, for the reasons described later in this section. Instead, there are two ways that you can safely restart a single peer:

- Use Splunk Web (**Settings>Server Controls**).
- Run the command `splunk offline`, followed by `splunk start`.

When you use Splunk Web or the `splunk offline`/`splunk start` commands to restart a peer, the manager waits 60 seconds (by default) before assuming that the peer has gone down for good. This allows sufficient time for the peer to come back on-line and prevents the cluster from performing unnecessary remedial activities.

Note: The actual time that the manager waits is determined by the value of the manager's `restart_timeout` attribute in server.conf. The default for this attribute is 60 seconds. If you need the manager to wait for a longer period, you can change the `restart_timeout` value, as described in Extend the restart period.

The `splunk offline`/`splunk start` restart method has an advantage over the Splunk Web method in that it waits for in-progress searches to complete before stopping the peer. In addition, since it involves a two-step process, you can use it if you need the peer to remain down briefly while you perform some maintenance.

For information on the `splunk offline` command, read Take a peer offline.

Caution: Do not use the `splunk restart` command to restart the peer. If you use the `splunk restart` command, the manager will not be aware that the peer is restarting. Instead, after waiting a default 60 seconds for the peer to send a heartbeat, the manager will initiate the usual remedial actions that occur when a peer goes down, such as adding its bucket copies to other peers. The actual time the manager waits is determined by the manager's `heartbeat_timeout`

attribute. It is inadvisable to change its default value of 60 seconds without consultation.

Perform a rolling restart of an indexer cluster

A rolling restart performs a phased restart of all peer nodes, so that the indexer cluster as a whole can continue to perform its function during the restart process. A rolling restart also helps ensure that load-balanced forwarders sending data to the cluster always have a peer available to receive the data.

A rolling restart occurs under these circumstances:

- You initiate a rolling restart in Splunk Web.
- You run the `splunk rolling-restart` CLI command.
- The manager node automatically initiates a rolling restart, when necessary, after distributing a configuration bundle to the peer nodes. For details on this process, see Distribute the configuration bundle.

Rolling restart modes

There are two rolling restart modes for indexer clusters:

- **Rolling restart:** Restarts peer nodes in successive groups (based on a pre-defined percentage) with no guarantee that the cluster is searchable. See How a rolling restart works.
- **Searchable rolling restart:** Restarts peer nodes one at a time with minimal interruption of ongoing searches. See Perform a searchable rolling restart.

To set the default rolling restart mode for the `splunk rolling-restart cluster-peers` command, see Set rolling restart behavior in server.conf.

How a rolling restart works

During a rolling restart, approximately 10% (by default) of the peer nodes simultaneously undergo restart, until all peers in the cluster complete restart. If there are less than 10 peers in the cluster, one peer at a time undergoes restart. The manager node orchestrates the restart process, sending a message to each peer when it is its turn to restart.

The restart percentage tells the manager how many restart slots to keep open during the rolling-restart process. For example, if the cluster has 30 peers and the restart percentage is set to the default of 10%, the manager keeps three slots open for peers to restart. When the rolling-restart process begins, the manager issues a restart message to three peers. As soon as each peer completes its restart and contacts the manager, the manager issues a restart message to another peer, and so on, until all peers have restarted. Under normal circumstances, in this example, there will always be three peers undergoing restart, until the end of the process.

If the peers are restarting slowly due to inadequately provisioned machines or other reasons, the number of peers simultaneously undergoing restart can exceed the restart percentage. See Handle slow restarts.

At the end of the rolling restart period, the manager rebalances the cluster primary buckets. See Rebalance the indexer cluster primary buckets to learn more about this process.

Here are a few things to note about the behavior of a rolling restart:

- The manager restarts the peers in random order.

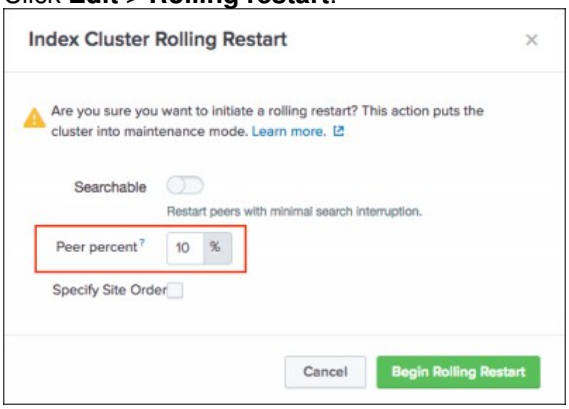
- The cluster enters maintenance mode for the duration of the rolling restart period. This prevents unnecessary **bucket fixup** while a peer undergoes restart.
- During a rolling restart, there is no guarantee that the cluster will be fully searchable.

Initiate a rolling restart

You can initiate a rolling restart from Splunk Web or from the command line.

Initiate a rolling restart from Splunk Web

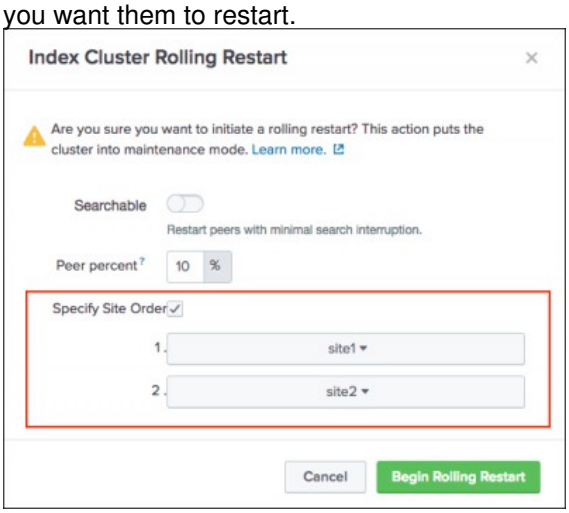
1. Log in to the manager node instance.
2. Click **Settings** > **Indexer clustering**.
3. Click **Edit** > **Rolling restart**.



4. (Optional) In the "Percent peers to restart" field, enter a number to change the percentage of peers you want the manager to restart simultaneously. The default percentage is 10.

   If you make changes to the percentage, the manager overrides the default value for percent_peers_to_restart in server.conf and the new value becomes the default.

5. (Optional) If the cluster is a multisite cluster, you can change the order in which sites in the cluster restart. Click the **Specify site order** checkbox, then click the drop-down boxes to arrange the available sites in the order that you want them to restart.



The site order drop down boxes only appear if the cluster is a multisite cluster.

6. Click **Begin rolling restart**.

Initiate a rolling restart from the command line

You can invoke the `splunk rolling-restart` command from the manager:

```
splunk rolling-restart cluster-peers
```

Specify the percentage of peers to restart at a time

By default, 10% of the peers restart at a time. The restart percentage is configurable through the `percent_peers_to_restart` attribute in the `[clustering]` stanza of `server.conf`. For convenience, you can configure this setting with the CLI `splunk edit cluster-config` command.

For example, to cause 20% of the peers to restart simultaneously, run this command:

```
splunk edit cluster-config -percent_peers_to_restart 20
```

To cause all peers to restart immediately, run the command with a value of 100:

```
splunk edit cluster-config -percent_peers_to_restart 100
```

An immediate restart of all peers can be useful under certain circumstances, such as when no users are actively searching and no forwarders are actively sending data to the cluster. It minimizes the time required to complete the restart.

After you change the `percent_peers_to_restart` value, you must run the `splunk rolling-restart` command to initiate the actual restart.

Perform a searchable rolling restart

Splunk Enterprise 7.1.0 and later provides a searchable option for rolling restarts. The searchable option lets you perform a rolling restart of peer nodes with minimal interruption of ongoing searches. You can use searchable rolling restart to minimize search disruption, when a rolling restart is required due to regular maintenance or a configuration bundle push.

How searchable rolling restart works

When you initiate a searchable rolling restart, the manager performs a restart of all peer nodes one at a time. During the restart process for each peer, the manager reassigns bucket primaries to other peers to retain the searchable state, and all in-progress searches complete within a configurable timeout period. The manager then restarts the peer and the peer rejoins the cluster. This process repeats for each peer until the rolling restart is complete.

Things to note about the behavior of searchable rolling restart:

- The manager restarts peers one at a time.
- The manager runs health checks to confirm that the cluster is in a searchable state before it initiates the searchable rolling restart.
- The peer waits for in-progress searches to complete, up to a maximum time period, as determined by the `decommission_search_jobs_wait_secs` attribute in `server.conf`. The default for this attribute is 180 seconds. This

covers the majority of searches in most cases.
- Searchable rolling restart applies to both historical searches and real-time searches.
- As of version 8.2.x, during a searchable rolling restart, continuous scheduled searches and real-time scheduled searches continue to run and are not deferred by default.
- Data model acceleration searches are skipped during a searchable rolling restart.

Best practices for searchable rolling restart

Before you initiate a searchable rolling restart, to help ensure a successful outcome, consider these best practices:

- To ensure that indexers do not get stuck in the reassigning primaries state and are able to restart during a searchable rolling restart, set the following settings in the `[clustering]` stanza in `server.conf` on the manager node:

  ```
  [clustering]
  restart_timeout = 600
  rolling_restart = searchable_force
  decommission_force_timeout = 180
  ```

  For more information on the above settings, see server.conf in the *Admin Manual*.

  > If you specify the searchable or searchable_force option when you initiate a rolling restart from the CLI or Splunk Web, the option you specify takes precedence over the existing rolling_restart setting in server.conf.

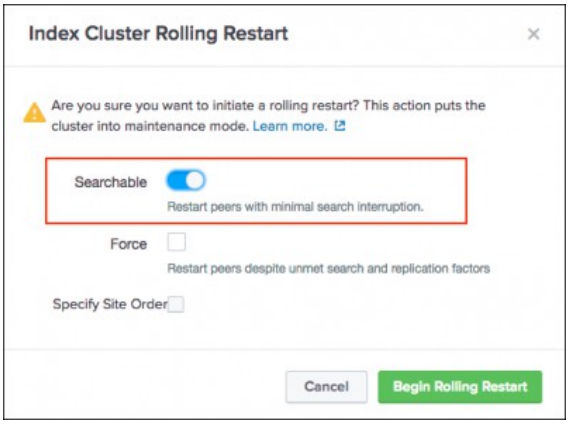
- In-progress searches that take longer than the `decommission_search_jobs_wait_secs` (default=180s), might generate incomplete results and a corresponding error message. If you have a scheduled search that must complete, either increase the value of the `decommission_search_jobs_wait_secs` setting in `server.conf`, or do not run a searchable rolling restart within the search's timeframe.
- Make sure the `search_retry` setting in the `[search]` stanza of `limits.conf` is set to `false` (the default). Setting this to `true` might cause searches that take longer than the `decommission_search_jobs_wait_secs` to generate duplicate or partial results with no error message.

Initiate a searchable rolling restart

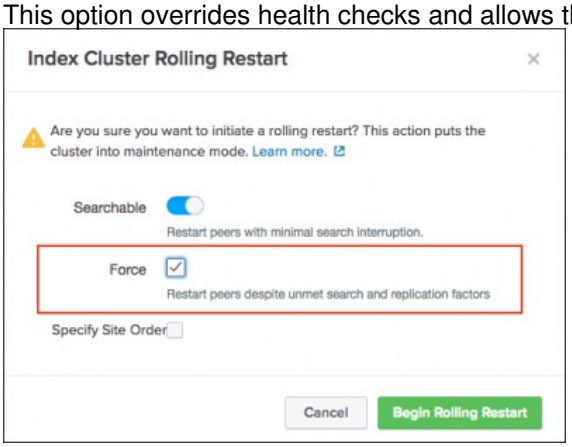
You can initiate a searchable rolling restart from Splunk Web or from the command line.

Inititate a searchable rolling restart from Splunk Web

1. On the manager, click **Settings > Indexer Cluster**.
2. Click **Edit > Rolling Restart**.
3. In the **Index Cluster Rolling Restart** modal, select **Searchable**.



4. Click **Begin Rolling Restart**.
   This initiates the searchable rolling restart.
5. (Optional) To proceed with the searchable rolling restart despite health check failures, select the **Force** option.
   This option overrides health checks and allows the searchable rolling restart to proceed.



> Use the Force option with caution. This option can impact searches.

6. Click **Begin Rolling Restart**.
   This initiates the searchable rolling restart. You can monitor the progress of the searchable rolling restart with the manager node dashboard.

Initiate a searchable rolling restart from the command line

To perform a searchable rolling restart from the command line:

1. (Optional) Run preliminary health checks to determine if the cluster is in a searchable state (search factor is met and all data is searchable).
2. Initiate a searchable rolling restart (includes health checks).
   Optionally, initiate a searchable rolling restart using the force option (overrides health checks).

1. (Optional) Run preliminary health checks

To check the current health of the cluster, run the following command on the manager:

```
splunk show cluster-status --verbose
```

This command shows information about the cluster state. Review the command output to confirm that the cluster is in a searchable state (search factor is met, all data is searchable) before you initiate the searchable rolling restart.

> The cluster must have two searchable copies of each bucket to be in a searchable state for a searchable rolling restart.

Here is an example of the output from the `splunk show cluster-status --verbose` command:

```
splunk@manager1:~/bin$ ./splunk show cluster-status --verbose

Pre-flight check successful .................. YES
 â  â  â  â  â  â  â   Replication factor met ............... YES
 â  â  â  â  â  â  â   Search factor met ..................... YES
 â  â  â  â  â  â  â   All data is searchable ................ YES
 â  â  â  â  â  â  â   All peers are up ..................... YES
 â  â  â  â  â  â  â   CM version is compatible ............. YES
 â  â  â  â  â  â  â   No fixup tasks in progress ........... YES
 â  â  â  â  â  â  â   Splunk version peer count { 7.1.0: 3 }

 Indexing Ready YES

 idx1          0026D1C6-4DDB-429E-8EC6-772C5B4F1DB5         default
         Searchable YES
         Status  Up
         Bucket Count=14
         Splunk Version=7.1.0

 idx3          31E6BE71-20E1-4F1C-8693-BEF482375A3F         default
         Searchable YES
         Status  Up
         Bucket Count=14
         Splunk Version=7.1.0

 idx2          81E52D67-6AC6-4C5B-A528-4CD5FEF08009         default
         Searchable YES
         Status  Up
         Bucket Count=14
         Splunk Version=7.1.0
```

The output shows that the health check is successful, which indicates the cluster is in a searchable state for a searchable rolling restart.

Health check output details

The table shows output values for the criteria used to determine the health of the indexer cluster.

Health Check	Output Value	Description
Replication factor met	YES	The cluster has the specified number of copies of raw data.
Search factor met	YES	The cluster has the specified number of searchable copies of data.
All data is searchable	YES	The cluster has a searchable copy of all data.
CM version is compatible	YES	The manager is running a compatible version of Splunk Enterprise.
No fixup tasks in progress	YES	No cluster remedial activities (such as bucket replication or indexing non-searchable bucket copies) is underway.

Health Check	Output Value	Description
All peers are up	YES	All indexer cluster peers are running.
Splunk version peer count	7.1 or later : # of peers	Number of peers running the Splunk Enterprise version.

Health checks are not all inclusive. Checks apply only to the criteria listed.

2. Initiate a searchable rolling restart

To initiate a searchable rolling restart:

On the manager, invoke the `splunk rolling-restart cluster-peers` command using the `searchable` option.

```
splunk rolling-restart cluster-peers -searchable true
```

This command automatically runs health checks against the cluster. If these health checks fail, the command returns the following message that indicates the cluster is not in a searchable state.

```
"Request rejected. Wait until search factor is met and all data is searchable."
```

If you want to proceed with the searchable rolling restart despite the health check failure, use the `force` option to override the health check and initiate the searchable rolling restart, as follows:

On the manager, invoke the `splunk rolling-restart cluster-peers` command using the `force` option.

```
splunk rolling-restart cluster-peers -searchable true \
-force true \
-restart_inactivity_timeout <secs> \
-decommission_force_timeout <secs>
```

When using the `force` option you can specify custom values for the following additional parameters. If you do not specify these parameters their default values are used.

- `decommission_force_timeout`: The amount of time, in seconds, after which the manager forces the peer to restart. Default: 180.
- `restart_inactivity_timeout`: The amount of time, in seconds, after which the manager considers the peer restart a failure and proceeds to restart other peers. Default: 600.

Set rolling restart default behavior in server.conf

You can set the default behavior for rolling restart using the `rolling_restart` attribute in the `[clustering]` stanza of `server.conf`. This attribute lets you define the type of rolling restart the manager performs on the peers. It also provides a convenient way to automatically load options for the `splunk rolling-restart cluster-peers` command as an alternative to passing them from the command line.

The `rolling_restart` attribute supports these settings:

- `restart`: Initiates a rolling restart.
- `shutdown`: Initiates a staged rolling restart. Shuts down a single peer, then waits for a manual restart. The process repeats until all peers are restarted.
- `searchable`: Initiates a rolling restart with minimum search interruption.

- `searchable_force`: Overrides cluster health checks and initiates a searchable rolling restart.

Specifying the searchable option in the CLI or UI overrides the rolling_restart = shutdown setting in server.conf.

To set the `rolling_restart` attribute:

1. On the manager, edit `$SPLUNK_HOME/etc/system/local/server.conf`.
2. In the `[clustering]` stanza, specify the `rolling_restart` attribute value. For example:

```
[clustering]
mode = manager
replication_factor = 3
search_factor = 2
pass4SymmKey = whatever
rolling_restart = searchable
```

3. Restart the manager.

When you set `rolling_restart = searchable_force`, you can specify custom values for the following additional attributes in the `[clustering]` stanza. If you do not specify these attributes their default values are used.

- `decommission_force_timeout`: The amount of time, in seconds, after which the manager forces the peer to restart. Default: 180.
- `restart_inactivity_timeout` : The amount of time, in seconds, after which the manager considers the peer restart a failure and proceeds to restart other peers. Default: 600.

For more information, see server.conf.spec in the *Admin Manual.*

Set searchable rolling restart as default mode for bundle push

Configuration bundle pushes that require a restart use the `rolling_restart` value in `server.conf`. You can set the `rolling_restart` value to `searchable` to make searchable rolling restart the default mode for all rolling restarts triggered by a bundle push.

To set searchable rolling restart as the default mode for configuration bundle push, specify one of the following attributes in the `[clustering]` stanza of `server.conf`:

`rolling_restart = searchable` or `rolling_restart = searchable_force`

For more information on configuration bundle push for indexer clusters, see Apply the bundle to the peers.

Default scheduled search behavior during searchable rolling restart

As of version 8.2.x, during a searchable rolling restart, continuous scheduled searches continue to run and are no longer deferred by default, as the `defer_scheduled_searchable_idxc` setting in `savedsearches.conf` has been changed to a default value of "false".

In some cases, continuous scheduled searches can return partial results during a searchable rolling restart. If necessary, you can defer continuous scheduled searches until after the rolling restart completes, as follows:

1. On the search head or search head cluster, edit `SPLUNK_HOME/etc/system/local/savedsearches.conf`.
2. Set `defer_scheduled_searchable_idxc = true`.
3. Restart Splunk.

Also as of version 8.2.x, during searchable rolling restart, real-time scheduled searches continue to run and are no longer deferred by default, based on the newly added `skip_scheduled_realtime_idxc = <boolean>` setting in `savedsearches.conf`, which is set by default to a value of "false".

For more information on `defer_scheduled_searchable_idxc` and `skip_scheduled_realtime_idxc` setting, see savedsearches.conf in the *Admin Manual.*

For information on real-time and continuous scheduled search modes, see Real-time scheduling and continuous scheduling.

Rolling restart on a multisite cluster

With a multisite cluster, by default, the rolling restart proceeds with site awareness. That is, the manager restarts all peers on one site before proceeding to restart the peers on the next site, and so on. This ensures that the cluster is always fully searchable, assuming that each site has a full set of primaries.

Invoke rolling restart on a multisite cluster

When you invoke the `splunk rolling-restart` command on a multisite cluster, the manager completes a rolling restart of all peers on one site before proceeding to the peers on the next site.

You can specify the site restart order, through the `-site-order` parameter.

Here is the multisite version of the command:

```
splunk rolling-restart cluster-peers [-site-order site<n>,site<n>, ...]
```

Note the following points regarding the `-site-order` parameter:

- This parameter specifies the site restart order.
- You can specify a subset of sites. Only the specified sites are restarted, in the order given.
- The default, if this parameter is not specified, is to select sites at random.

For example, if you have a three-site cluster, you can specify rolling restart with this command:

```
splunk rolling-restart cluster-peers -site-order site1,site3,site2
```

The manager initiates the restarts in this order: site1, site3, site2. So, the manager first initiates a rolling restart for the peers on site1 and waits until the site1 peers complete their restarts. Then the manager initiates a rolling restart on site3 and waits until it completes. Finally, it initiates a rolling restart on site2.

If you do not want the peer nodes to restart on a site-by-site basis, but instead prefer the manager to select the next restart peer randomly, from across all sites, you can use the parameter -site-by-site=false.

How the manager determines the number of multisite peers to restart in each round

The number of peers in each site of a multisite cluster that can restart simultaneously is determined by the `server.conf` settings:

- `searchable_rolling_site_down_policy`. The default is half.
- `percent_peers_to_restart`. The default is 10 percent.

Before the `searchable_rolling_site_down_policy` setting is applied, there must be one searchable copy in each site of the multisite cluster. In addition, the cluster must be healthy, and no peers can have site-constrained buckets. If all conditions are met, a site uses the `searchable_rolling_site_down_policy` setting to determine how many peers are restarted.

If any of the conditions are not met, the process uses the `percent_peers_to_restart` to calculate the number of peers allowed to restart simultaneously.

The restart proceeds like this:

1. The restart policy is set, representing the number of peers in a site that can be restarted simultaneously.
2. The manager selects a site to restart first. The site order is configurable.
3. The manager begins restarting peers from the first site. It won't restart peers across multiple sites.
4. The selected peers move to a status of `ReassigningPrimaries`, and then through `Restarting`, `BatchAdding`, and on to the final status of `Up`.
5. The manager continues to restart peers from the first site until all of the site peers are restarted.
    1. If a peer experiences an issue such as a timeout while going through the rolling restart process, the policy and peer counts do not change. The manager will continue until every peer in the site has been restarted.
6. The manager moves to restarting peers on the next site.
7. The manager continues to restart peers from the next site, and once that site is complete it will begin restarting peers on another site, until all peers in all sites have been restarted.

Handle slow restarts

If the peer instances restart slowly, the peers in one group might still be undergoing restart when the manager tells the next group to initiate restart. This can occur, for example, due to inadequate machine resources. To remedy this issue, you can increase the value of `restart_timeout` in the manager's `server.conf` file. Its default value is 60 seconds.

Abort a rolling restart of an indexer cluster

You can abort an ongoing user-initiated rolling restart of an indexer cluster. Aborting a rolling restart can be useful if a rolling restart becomes stuck or unacceptably slow due to an issue with the an indexer or the system.

You can abort a user-initiated rolling restart using the CLI or REST API.

To abort a rolling restart your role must have the `edit_indexer_cluster` capability.

> Abort rolling restart functionality applies to user-initiated rolling restarts only. You cannot abort a rolling restart triggered automatically by the cluster manager due to configuration changes, such as a bundle push, as this can leave the cluster in an inconsistent state, with groups of indexers having different sets of configurations.

Abort a rolling restart using the CLI

To abort an ongoing user-initiated rolling restart of an indexer cluster using the CLI:

On the cluster manager, run the `abort-rolling-restart cluster-peers` command. For example:

```
splunk abort-rolling-restart cluster-peers -auth admin:password
```

The command returns a list of peers that have not been restarted. For example:

```
Aborting the rolling restart initiated successfully. List of peers skipped restarting:
E30CA8C0-23E5-4A6B-9F28-D2EC991CCD75,9E3FED8B-59A0-4B95-8116-F8F8A67A7686,32790C7F-82CB-4E39-8689
-3600F72D4D01,2B6C57ED-9FFC-44F0-9E58-CD8BE3519F3F,5A65CEB6-79A6-40D7-914C-4859DEACF79B,8C2DC775-EB8E-44D7-AFF8
-38482B3A9990,033085C7-F31B-467D-9577-B8A5E5131810
```

Abort a rolling restart using the REST API

To abort an ongoing user-initiated rolling restart of an indexer cluster using the REST API:

Send an HTTP POST request to the `/services/cluster/manager/control/default/abort_restart` endpoint. For example:

```
curl -k -u admin:password -X POST
"https://chieftain:15511/services/cluster/manager/control/default/abort_restart?output_mode=json"
```

For endpoint details, see cluster/manager/control/default/abort_restart in the *REST API Reference Manual*.

Conflicting operations

You cannot run certain operations simultaneously:

- Data rebalance
- Excess bucket removal
- Rolling restart
- Rolling upgrade

If you trigger one of these operations while another one is already running, `splunkd.log`, the CLI, and Splunk Web all surface an error to the effect that a conflicting operation is in progress.

Rebalance the indexer cluster

By rebalancing the indexer cluster, you balance the distribution of bucket copies across the set of peer nodes. A balanced set of bucket copies optimizes each peer's search load and, in the case of data rebalancing, each peer's disk storage.

Types of indexer cluster rebalancing

There are two types of indexer cluster rebalancing:

- Primary rebalancing
- Data rebalancing

Primary rebalancing

The goal of primary rebalancing is to balance the search load across the peer nodes.

Primary rebalancing redistributes the primary bucket copies across the set of peer nodes. It attempts, to the degree possible, to ensure that each peer has approximately the same number of primary copies.

Primary rebalancing simply reassigns primary markers across the set of existing searchable copies. It does not move searchable copies to different peer nodes. Because of this limitation, primary rebalancing is unlikely to achieve a perfect balance of primaries.

Because primary rebalancing only reassigns markers and does not cause any bucket copies to move between peers, it completes quickly.

See Rebalance indexer cluster primary bucket copies.

Data rebalancing

The goal of data rebalancing is to balance the storage distribution across the peer nodes.

Data rebalancing redistributes bucket copies so that each peer has approximately the same number of copies. It balances searchable, non-searchable, and primary copies.

During data rebalancing, the cluster moves bucket copies from peers with more copies to peers with fewer copies. Since this type of rebalancing includes searchable copies, it overcomes the limitation inherent in primary rebalancing and achieves a significantly better balance of primaries.

Because data rebalancing involves significant fixup activity, such as moving bucket copies between peers, it can be a slow and lengthy process.

See Rebalance indexer cluster data.

Rebalance indexer cluster primary bucket copies

When you start or restart a manager or peer node, the manager rebalances the set of **primary** bucket copies across the peers in an attempt to spread the primary copies as equitably as possible. Ideally, if you have four peers and 300 buckets, each peer would hold 75 primary copies. The purpose of primary rebalancing is to even the search load across the set of peers.

How primary rebalancing works

To achieve primary rebalancing, the manager reassigns the primary state from existing bucket copies to **searchable** copies of the same buckets on other peers, as necessary. This rebalancing is a best-effort attempt; there is no guarantee that full, perfect rebalance will result.

Primary rebalancing occurs automatically whenever a peer or manager node joins or rejoins the cluster. In the case of a rolling restart, rebalancing occurs once, at the end of the process.

> Even though primary rebalancing occurs when a new peer joins the cluster, that peer won't participate in the rebalancing, because it does not yet have any bucket copies. The rebalancing takes place among any existing peers that have searchable bucket copies.

When performing primary rebalancing on a bucket, the manager simply reassigns the primary state from one searchable copy to another searchable copy of the same bucket, if there is one, and if, by doing so, the balance of primaries across peers will improve. It does not cause peers to stream bucket copies, and it does not cause peers to make unsearchable copies searchable. If an existing peer does not have any searchable copies, it will not gain any primaries during rebalancing.

Initiate primary rebalancing manually

If you want to initiate the primary rebalancing process manually, you can either restart a peer or hit the `/services/cluster/manager/control/control/rebalance_primaries` REST endpoint on the manager node. For example, run this command on the manager node:

```
curl -k -u admin:pass --request POST \
  https://localhost:8089/services/cluster/manager/control/control/rebalance_primaries
```

For more information, see cluster/manager/control/control/rebalance_primaries in the *Rest API Reference Manual*.

Rebalance primaries on a multisite cluster

There are a few differences in how primary rebalancing works in a multisite cluster. In a multisite cluster, multiple sites typically have full sets of primary copies. When you rebalance the cluster, the rebalancing occurs independently for each site. For example, in a two-site cluster, the cluster separately rebalances the primaries in site1 and site2. It does not shift primaries between the two sites.

The start or restart of a peer on any site triggers primary rebalancing on all sites. For example, if you restart a peer on site1 in a two-site cluster, rebalancing occurs on both site1 and site2.

View the number of primaries on a peer

To gain insight into the primary load for any peer, you can use the `cluster/manager/peers` endpoint to view the number of primaries that the peer currently holds. The `primary_count` shows the number of primaries that the peer holds for its local site. The `primary_count_remote` shows the number of primaries that the peer holds for non-local sites, including site0.

By using this endpoint on all your peers, you can determine whether the cluster could benefit from primary rebalancing.

See cluster/manager/peers in the *Rest API Reference Manual*.

Summary of indexer cluster primary rebalancing

Primary rebalancing is the rebalancing of the primary assignments across existing searchable copies in the cluster.

Primary rebalancing occurs under these circumstances:

- A peer joins or rejoins a cluster
- At the end of a rolling restart
- A manager rejoins the cluster
- You manually invoke the `rebalance_primaries` REST endpoint on the manager node

Rebalance indexer cluster data

To rebalance indexer cluster data, you rebalance the set of bucket copies so that each peer holds approximately the same number of copies. This helps ensure that each peer has a similar storage distribution.

The problem of imbalanced data

Imbalanced data increases the likelihood that one or more peers will run out of disk space and thus transition to the detention state. In the detention state, the peer no longer indexes new data, and so forwarders can no longer send data to that peer. When that happens, a load-balanced forwarder shifts its incoming data to other peers, increasing the indexing

burden on those peers. In the worst-case scenario, if the forwarder is not configured for load-balancing, its data gets lost.

In addition, as its existing data ages, the peer in detention will contain relatively older data compared to other peers. Since most searches focus on newer data, this means that the peer's data will typically get searched less frequently, shifting the burden of the search load onto the peers that are not in detention.

Aside from detention considerations, imbalanced data can affect peer utilization during searches. If some peer nodes hold more bucket copies for a particular index compared to other peer nodes, they will have a greater part of the search workload for searches on that index. For this reason, data rebalancing is index-aware. When rebalancing completes, each peer will have approximately the same number of bucket copies for each index.

Conditions that cause imbalanced data

A number of factors can cause an imbalance of bucket copies. These include the following:

- **Newly added peer nodes.** When you add a new peer, it initially has no bucket copies. Through data rebalancing, you can move copies onto that peer from other peers.
- **Uneven forwarding of data.** If the forwarders are sending more data to some peer nodes, it is likely that those peers will hold more bucket copies. Rebalancing provides a way to move copies from those peers to peers with fewer copies.

What data rebalancing accomplishes

Data rebalancing attempts to achieve an equitable distribution of bucket copies across the set of peer nodes. It factors in several bucket characteristics:

- Data rebalancing balances both non-searchable and searchable bucket copies.
- Data rebalancing balances for each index, so that, in addition to each peer holding approximately the same number of bucket copies in total, each peer will hold the same number of bucket copies for each index. See Data rebalancing and indexes.
- Data rebalancing operates on warm and cold buckets only. It does not rebalance hot buckets.
- Data rebalancing operates only on buckets that meet their replication and search factors.

Data rebalancing attempts to achieve a best-effort balance, not a perfect balance. See Configure the data rebalancing threshold.

Data rebalancing and storage utilization

Data rebalancing has limitations regarding its effect on storage utilization.

Data rebalancing balances the number of bucket copies, not the actual data storage. In addition, data rebalancing attempts to achieve a practical balance, not a perfect balance. In most cases, the process achieves an optimal approximation of balanced storage, as determined by several criteria:

- The process assumes that all peers have the same amount of disk storage available for indexes. It is a best practice to use homogeneous instances for your peer nodes.
- The process assumes that all buckets are the same size, although bucket size does sometimes vary.
- The process stops when the number of copies on each peer is within a small range of a perfect balance. It does not ordinarily attempt to put precisely the same number of copies on each peer. See Configure the data rebalancing threshold.

Make data rebalance search-safe

You can run data rebalance in searchable mode to avoid search downtime. In searchable mode, the rebalance operation is search-safe, depending on limits controllable through timeout settings.

Do not use searchable data rebalance with SmartStore indexes. Searchable mode is not optimized for SmartStore and can cause data rebalance to proceed slowly. Use non-searchable data rebalance instead. In any case, non-searchable data rebalance of SmartStore indexes usually causes only minimal search disruption. The data rebalance process runs quickly on SmartStore indexes, because it moves only bucket metadata, not the bucket data itself.

The default mode for data rebalance is non-searchable, which means that the operation is not search-safe. Because, as part of its operation, data rebalancing removes bucket copies from their old peers after streaming copies to new peers, the data rebalancing operation might remove bucket copies needed by a search while the search is in progress. Therefore, search results are not guaranteed to be complete during non-searchable data rebalancing.

With searchable data rebalance, the operation waits until in-progress searches are complete, up to a configurable time limit, before removing the old bucket copies that pertain to the search's generation. New searches with a newer generation can simultaneously perform searches against new bucket copies.

Data rebalance can take longer to complete in searchable mode. Searchable data rebalance also requires more storage space because excess buckets are removed in batches, rather than immediately, as is the case with non-searchable data rebalance.

As a best practice, perform an excess bucket removal as a separate operation before running a searchable data rebalance. See Remove excess bucket copies from the indexer cluster. By performing the excess bucket removal separately, the removal process runs as a non-searchable operation, which is significantly faster than the searchable removal operation that otherwise occurs at the start of a searchable data rebalance. The performance improvement is particularly noticeable for larger numbers of excess buckets, in the thousand bucket range and beyond.

You can specify the searchable mode on a per-operation basis when you initiate a data rebalance operation, either through the CLI, as described in Initiate data rebalancing, or through the manager node dashboard, as described in Use the manager dashboard to initiate and configure rebalancing.

You can also specify the searchable mode for all data rebalancing operations with the `searchable_rebalance` setting in `server.conf` on the manager node. If you set `searchable_rebalance = true`, you do not need to specify the searchable mode each time that you initiate a data rebalance.

If you find that some of your searches are not completing during data rebalance, you can change the `rebalance_search_completion_timeout` setting in `server.conf` to increase the maximum time that the data rebalance operation waits for in-progress searches to complete. Its default value is 180 seconds.

Searches that take longer to complete than the rebalance_search_completion_timeout value are not retried.

Other settings in `server.conf` allow you to control additional aspects of the searchable data rebalance operation when necessary. Modify those settings only in consultation with Splunk Support.

Searchable data rebalance requires that the manager node and all peer nodes run Splunk Enterprise version 7.3.0 or higher.

How data rebalancing works

The manager node controls the data rebalancing process. To achieve the goal of equitable distribution of bucket copies across all peer nodes, the manager moves bucket copies from peers with an above-average number of copies to peers with a below-average number of copies. It continues this process until the cluster is balanced, with each peer holding approximately the same number of bucket copies.

Note: When you initiate a data rebalance operation, the cluster first runs an excess buckets removal operation to remove any bucket copies that exceed the cluster's replication or search factors. See Remove excess bucket copies from the indexer cluster. As described below, the cluster also removes excess buckets during the data rebalance operation, to clean up the extra buckets created during the operation itself.

To achieve rebalancing, the cluster uses the fundamental processes of bucket fixup. It streams bucket copies from one peer to another. The process of "moving" a bucket copy to another peer thus involves streaming the copy from a peer with an above-average number of bucket copies to a peer with a below-average number of bucket copies.

The cluster processes buckets sequentially, one after another, until rebalancing is complete. It does not wait for one bucket to complete before starting rebalancing on the next, so there will typically be a number of buckets simultaneously undergoing rebalancing. See Control rebalancing load.

In the case of non-searchable data rebalance, once the streaming of a bucket to another peer is complete, the cluster handles the excess bucket copy that now exists by immediately removing a copy of the bucket from a peer with an above-average number of bucket copies overall. The cluster performs primary rebalancing at the end of the data rebalancing process.

In the case of searchable data rebalance, the cluster instead removes excess copies in batches, after waiting for any active searches to complete. By batching excess copy removal and waiting until active searches are complete, the cluster is able to minimize the possibility that a bucket for an in-progress search is removed while the search is still running. In addition, to accommodate both in-progress searches and new searches, the cluster also swaps primaries before removing the batch of excess copies. This action results in a new generation. In-progress searches continue to use the old generation, while new searches use the new generation. In this way, the rebalance operation is search-safe throughout the process no matter when a search begins.

The rebalancing process can terminate prematurely, either due to manual intervention or because it times out. Termination conditions are discussed elsewhere in this topic.

Data rebalancing and indexes

The rebalancing process balances the bucket copies by index. When rebalancing completes, each peer holds approximately the same number of bucket copies in total, as well as the same number of copies split by index.

For example, say you have a cluster with four peers and two indexes, index1 and index2. Index1 has 100 bucket copies distributed across all peers. Index2 has 300 copies distributed across all peers, for a total of 400 copies of all buckets across all peers.

Before rebalancing, the bucket distribution across the set of peers might look like this:

- Peer1: 110 total
    - Index1: 10
    - Index2: 100
- Peer2: 100 total
    - Index1: 50

    - Index2: 50
- Peer3: 50 total
    - Index1: 20
    - Index2: 30
- Peer4: 140 total
    - Index1: 20
    - Index2: 120

After rebalancing, the bucket distribution will look approximately like this:

- Peer1: 100 total
    - Index1: 25
    - Index2: 75
- Peer2: 100 total
    - Index1: 25
    - Index2: 75
- Peer3: 100 total
    - Index1: 25
    - Index2: 75
- Peer4: 100 total
    - Index1: 25
    - Index2: 75

Data rebalancing in multisite indexer clusters

Multisite data rebalancing operates in essentially the same way as single-site rebalancing. However, in multisite clusters, the manager first balances each bucket across sites to the degree that the site configuration allows. It then balances the bucket within each site.

The degree to which you can balance a multisite cluster across sites depends on the site replication and search factors. For example, if you have a site replication factor of `origin:2,total:3`, the cluster maintains two-thirds of the copies on their origin site. If one site originates more buckets than another, rebalancing cannot address the resulting imbalance without violating the site replication factor, and so the imbalance will remain. Similarly, the cluster will not rebalance in a way that violates explicit site requirements. Intersite balancing does, however, balance non-explicit copies across sites.

Initiate data rebalancing

You can rebalance the data for all indexes or for just a single index. In addition, you can set a timeout for the rebalancing.

To rebalance the data, run this CLI command on the manager node:

```
splunk rebalance cluster-data -action start [-searchable true] [-index index_name] [-max_runtime
interval_in_minutes]
```

Note the following:

- Set the optional `-searchable` parameter to `true` to enable search-safe data rebalance. Data rebalance can take longer to complete in searchable mode. The default for this parameter is `false` (non-searchable data rebalance).
- Use the optional `-index` parameter to rebalance just a single index. Otherwise, the command rebalances all indexes.
- Use the optional `-max_runtime` parameter to limit the rebalance activity to a specified number of minutes. The rebalancing stops automatically if the timeout limit is reached, even if there are still more buckets to process. For

details on what happens when data rebalancing stops prematurely, see Stop data rebalancing.

You can also initiate rebalancing from the manager node dashboard. See Use the manager dashboard to initiate and configure rebalancing.

It is a best practice to perform data rebalancing during a maintenance window for these reasons:

- Data rebalancing can cause primary bucket copies to move to new peers, so search results are not guaranteed to be complete while data rebalancing continues.
- The fixup activities associated with data rebalancing have a low priority compared to other bucket fixup activities, such as maintaining the replication and search factors, so rebalancing will wait while other fixup activity completes.

Stop data rebalancing

To stop data rebalancing prematurely, run this CLI command on the manager node:

```
splunk rebalance cluster-data -action stop
```

For any bucket in the midst of rebalancing when this command occurs, the cluster finishes the current process. It does not initiate any additional processing on the bucket, however. For example, if the cluster is in the midst of copying a non-searchable copy of a bucket, it finishes making the copy. It does not check whether the bucket balance can improve by also processing a searchable copy. It also does not remove any excess copies of the bucket.

View status of data rebalancing

To see whether data rebalancing is running, run this CLI command on the manager node:

```
splunk rebalance cluster-data -action status
```

You can also use the manager dashboard to view rebalancing status. See Use the manager dashboard to initiate and configure rebalancing.

Configure the data rebalancing threshold

The manager attempts to achieve a reasonable, but not perfect, balance, in which the resulting number of copies on each peer lies within a narrow range to either side of the average number of copies for all peers.

This balance is configurable through the `rebalance_threshold` attribute in the manager's `server.conf`. You can adjust the setting either directly in `server.conf` or by means of the CLI. For example:

```
splunk edit cluster-config -mode manager -rebalance_threshold 0.95 -auth admin:your_password
```

You can also configure the rebalancing threshold through the manager node dashboard. See Use the manager dashboard to initiate and configure rebalancing.

A `rebalance_threshold` value of 1.00 means that rebalancing will continue until the cluster is fully balanced, with each peer having the same number of copies. The default value is 0.90, which means that rebalancing will continue until each peer is within 90% of a perfect balance.

With the default setting of 0.90, rebalancing continues until the number of copies on each peer lies within a range of .90 to 1.10 of the average. For example, if you have three peers holding between them a total of 300 copies, meaning that there is an average of 100 copies per peer, the rebalancing process stops when every peer holds between 90 and 110 copies.

If you decide instead that a 95% balance is preferable, you can set `rebalance_threshold` to 0.95. The manager will then perform any necessary rebalancing until the number of copies on each peer lies within a range of .95 to 1.05 of the average.

The cluster considers each index individually against the threshold. That is, the goal of the rebalancing is to ensure that each index is balanced to the tolerance set by the `rebalance_threshold` attribute.

Use the manager dashboard to initiate and configure rebalancing

You can initiate and configure rebalancing through the manager node dashboard. See Configure the manager node with the dashboard.

1. Click the **Edit** button on the upper right side of the dashboard.
2. Select the **Data Rebalance** option.
   A pop-up window appears with several fields.
3. Fill out the fields as necessary:
   - **Threshold.** Change the rebalancing threshold.
   - **Max Runtime.** Stop the rebalancing process after a set period of time. If you leave this field empty, rebalancing continues until all peers are within the threshold limit.
   - **Index.** Run rebalancing on a single index or across all indexes.
   - **Searchable.** Enable search-safe data rebalance. Data rebalance can take longer to complete in searchable mode.
4. To start rebalancing, click the **Start** button.

The window also provides rebalance status information.

Control rebalancing load

You can configure the rebalancing load to minimize the effect that the consequent fixup activity has on peer indexing and search performance.

The maximum number of buckets that can be rebalancing at a time is subject to the same attributes that determine peer load for any fixup activity: `max_peer_rep_load` and `max_peer_build_load` in the `[clustering]` stanza of `server.conf` on the manager node.

Data rebalancing uses the value of these attributes reduced by 1, if the attribute value is greater than 1. For example, if you set `max_peer_rep_load` to 4, then the peer can participate in a maximum of three (not four) concurrent rebalancing replications as a target.

Conflicting operations

You cannot run certain operations simultaneously:

- Data rebalance
- Excess bucket removal
- Rolling restart
- Rolling upgrade

If you trigger one of these operations while another one is already running, `splunkd.log`, the CLI, and Splunk Web each surface an error that indicates that a conflicting operation is in progress.

Remove excess bucket copies from the indexer cluster

Excess bucket copies are copies that exceed the cluster's replication factor or search factor. For example, if the cluster has a replication factor of 3, each bucket should optimally have exactly three copies residing across the set of peer nodes. If one bucket has four copies, that bucket has one excess copy.

Excess copies do not interfere with the operation of the cluster, but they are unnecessary and require extra disk space.

You can view and remove excess bucket copies from the manager node dashboard or from the CLI.

Caution: Before removing excess buckets, ensure that the cluster is in a **complete** state. There should be few, if any, pending bucket fixup jobs. In the case of a large number of excess buckets, in the range of several million, the best practice is to put the cluster in maintenance mode and remove excess buckets one index at a time.

How excess copies originate

Excess copies can result from peers leaving the cluster and then returning to it. When a peer goes down, the cluster initiates **bucket fixing** activities to compensate for any copies on that peer, because those copies are no longer available to the cluster. The goal of bucket fixing is return the cluster to the **complete** state, where each bucket has a replication factor number of copies and a search factor number of searchable copies.

If the peer later returns to the indexer cluster, any bucket copies that the peer retained while down are once again available to the cluster. This can result in the cluster maintaining excess copies of some buckets, as described in the topic What happens when a peer node comes back up.

In effect, a returning peer can cause the cluster to store more copies of some buckets than are needed to fulfill the replication factor and, possibly, the search factor as well. It can sometimes be useful to keep the extra copies around, as that topic explains, but you can save disk space by instead removing them.

Use the manager node dashboard

To view or remove excess bucket copies:

1. On the manager node, click **Settings** on the upper right side of Splunk Web.

2. In the **Distributed Environment** group, click **Indexer clustering**.

This takes you to the manager node dashboard.

3. Select the Indexes tab.

4. Click the **Bucket Status** button.

This takes you to the Bucket Status dashboard.

5. Select the Indexes with Excess Buckets tab.

This tab provides a list of indexes with excess bucket copies. It enumerates both buckets with excess copies and buckets with excess searchable copies. It also enumerates the total excess copies in each category. For example, if your index "new" has one bucket with three excess copies, one of which is searchable, and a second bucket with one excess copy, which is non-searchable, the row for "new" will report:

- 2 buckets with excess copies
- 1 bucket with excess searchable copies
- 4 total excess copies
- 1 total excess searchable copies

If you want to remove the excess copies for a single index, click the **Remove** button on the right side of the row for that index.

If you want to remove the excess copies for all indexes, click the **Remove All Excess Buckets** button.

Use the CLI

The Splunk CLI has two commands that help manage and remove excess bucket copies. You can run these commands either across the entire set of indexes or on just a single index.

Determine whether the cluster has extra copies

To find out how many buckets have extra copies, including extra searchable copies, run this command from the manager:

```
splunk list excess-buckets [index-name]
```

The output from `splunk list excess-buckets` looks like this:

```
index=_audit
       Total number of buckets=4
       Number of buckets with excess replication copies=0
       Number of buckets with excess searchable copies=0
       Total number of excess replication copies across all buckets=0
       Total number of excess searchable copies across all buckets=0
index=_internal
       Total number of buckets=4
       Number of buckets with excess replication copies=0
       Number of buckets with excess searchable copies=0
       Total number of excess replication copies across all buckets=0
       Total number of excess searchable copies across all buckets=0
index=main
       Total number of buckets=5
       Number of buckets with excess replication copies=5
       Number of buckets with excess searchable copies=5
       Total number of excess replication copies across all buckets=10
       Total number of excess searchable copies across all buckets=5
```

Remove extra bucket copies

To remove all extra bucket copies from the cluster (or from one index on the cluster), run this command from the manager:

```
splunk remove excess-buckets [index-name]
```

The manager determines which peers to remove the extra copies from. This is not configurable, and the extras will not necessarily be removed from the peer that has most recently returned to the cluster.

Conflicting operations

You cannot run certain operations simultaneously:

- Data rebalance
- Excess bucket removal
- Rolling restart
- Rolling upgrade

If you trigger one of these operations while another one is already running, `splunkd.log`, the CLI, and Splunk Web all surface an error to the effect that a conflicting operation is in progress.

Put a peer into detention

When a peer is in the state of detention, its functionality is reduced. It stops replicating data from other peer nodes and, depending on the type of detention, stops indexing most or all data. It continues to participate in searches.

A peer can enter detention either automatically, in response to a low level of free disk space, or manually.

Automatic detention

When a peer enters the detention state automatically, it

- stops indexing all data, internal and external.
- stops replicating data from other peer nodes.
- stops participating in searches.

The peer node enters the detention state automatically when it runs low on disk space. The setting that controls automatic detention is `minFreeSpace` in `server.conf`. The default value is 5000, or 5GB, meaning that the peer enters detention when it has less than 5GB of free disk space.

The peer automatically leaves the detention state when its free disk space grows to exceed `minFreeSpace`.

Manual detention

When a peer enters the detention state manually, it

- stops replicating data from other peer nodes.
- optionally stops accepting data from the ports that consume external data, causing the peer to no longer index most types of external data.
- continues to index internal data and stream the data to target peer nodes.
- continues to participate in searches.

When you manually put a peer into the detention state, it remains in detention until you remove it from detention. Manual detention persists through peer restart.

The effect of not accepting data from external data ports

When setting a peer to manual detention, you can optionally direct the peer to stop accepting data from the ports that consume incoming data.

This brings a halt to the indexing of most external data, including

- TCP inputs
- UDP inputs
- HEC inputs
- data sent from a forwarder to the peer through its **receiving port**

Attempts to send HEC data to the peer generate the return error "HTTPSTATUS_NOT_FOUND, 404".

However, external data can continue to enter the peer through these methods:

- scripted inputs
- file and folder monitoring
- the `receivers/stream` endpoint

In addition, the indexer can continue to route incoming data to another Splunk Enterprise instance or to a third-party system.

Use cases

Here are some of the key use cases for manual detention:

- To bring a near halt to the growth of disk usage on the peer, for example, if the peer is close to running out of space.
- To partially decommission an old peer, making it available only for searches on existing data.
- To stop a troublesome peer from handling external or replicated data, while keeping the peer available for diagnostics.
- To force new data to go to the new peers, when you add new peers to a cluster.

  Note: You can also use data rebalancing to move data to new peers. See Rebalance the cluster.

- To slow the growth of disk usage on a peer that belongs to a pre-approved firewall exception list and needs to continue receiving incoming data. For this use case, you can configure the peer to stop replication activity but continue to consume external data.

Put a peer into manual detention

To put a peer into detention, run the CLI command `splunk edit cluster-config` with the `-manual_detention` parameter.

You can set the `-manual_detention` parameter to one of several values:

- `on`. The peer enters detention and stops accepting data from the ports that consume incoming data. These ports are the receiving TCP, UDP, and HTTP event collector ports.This action has the effect of halting indexing of most external data. The peer continues to index internal data. The peer stops replicating data from other peer nodes.
- `on_ports_enabled`. The peer enters detention and the ports stay open to accept incoming data. The peer continues to index both external and internal data. The peer stops replicating data from other peer nodes.
- `off`. The peer is not in detention. This is the default.

You can run this command from the peer itself or from the manager node.

Caution: The peer must be in the `Up` state, or "status," before you put it in detention. For information on how to determine the status of a peer, see View the manager node dashboard.

To run the command from the peer:

```
splunk edit cluster-config -auth <username>:<password> -manual_detention [off|on|on_ports_enabled]
```

To run the command from the manager node:

```
splunk edit cluster-config -auth <username>:<password> -peers <peer_guid1>,<peer_guid2>,...
-manual_detention [off|on|on_ports_enabled]
```

Note the following:

- `-peers` specifies the set of peers that you want to put in detention. Identify each peer by its GUID. When you run the command from the manager node, you must include this parameter.

Take a peer out of manual detention

To take a peer out of detention:

```
splunk edit cluster-config -auth <username>:<password> -manual_detention off
```

Use a REST endpoint to put the peer into manual detention

You can use the REST endpoint `cluster/peer/control/control/set_manual_detention` to put a peer into manual detention.

Note: A previous endpoint, `cluster/peer/control/control/set_detention_override`, has been deprecated. Use `cluster/peer/control/control/set_manual_detention` in its place.

See the REST API documentation for cluster/peer/control/control/set_manual_detention.

View the detention state

You can view the states, detention-related or otherwise, of all peers from the manager node dashboard. See View the manager node dashboard.

These are the possible detention states:

- **AutomaticDetention.** Peer entered detention automatically.
- **ManualDetention.** Peer entered detention manually and no longer consumes external data.
- **ManualDetention-PortsEnabled.** Peer entered detention manually and continues to consume external data.

You can also use the DMC to view the state of the peers.

In addition, some CLI commands also provide peer state information:

- To view the state of all peers, run this command on the manager:

```
splunk list cluster-peers
```

- To view the state of a single peer, run this command on the peer:

```
splunk list cluster-config
```

Remove a peer from the manager node's list

After a peer goes down, it remains on the manager's list of peer nodes. The main effect is that it continues to appear on the manager node dashboard, although its status changes to "Down", "GracefulShutdown", or Stopped", depending on how it went down.

You can use the `splunk remove cluster-peers` command to remove peers from the list:

```
splunk remove cluster-peers -peers <guid>,<guid>,<guid>,...
```

Note the following:

- All peers removed must be in the "Down", "GracefulShutdown", or "Stopped" state.
- You specify the peers with a comma-separated list of GUIDs, one per peer.
- The GUIDs can be specified with or without hyphens. For example: 4EB4D230-CB8B-4DEB-AD68-CF9209A6868A and 4EB4D230CB8B4DEBAD68CF9209A6868A are both valid.
- If any GUID on the list is invalid, because one of the GUIDs does not correlate to a downed peer, the manager aborts the entire operation.
- You can obtain peer GUIDs by running the command `splunk list cluster-peers` on the manager.

You can also remove the peer from the manager's list by restarting the manager.

For information on the manager node dashboard's list of peers, see View the manager node dashboard.

Manage a multisite indexer cluster

Handle manager site failure

If the site holding the manager node fails, you lose the manager's functionality. You must immediately start a new manager on one of the remaining sites.

Until the new manager starts up, the cluster continues to function as best it can. The peers continue to stream data to other peers based on the list of target peers that they were using at the time the manager went down. If some of their target peers go down (as would likely be the case in a site failure), they remove them from their lists of streaming targets and continue to stream data to any peers remaining on their lists.

To deal with manager site failure, do the following:

1. Configure a stand-by manager on at least one of the sites not hosting the current manager. See Replace the manager node on the indexer cluster. **This is a preparatory step. You must do this before the need arises.**

2. When the manager site goes down, bring up a stand-by manager on one of the remaining sites. See Replace the manager node on the indexer cluster.

3. Restart indexing on the cluster, following the instructions in Restart indexing in multisite cluster after manager restart or site failure.

The new manager now fully replaces the old manager.

Note: If the failed site later comes back up, you need to point the peers on that site to the new manager. See Ensure that the peer and search head nodes can find the new manager.

Restart indexing in multisite cluster after manager restart or site failure

When a manager restarts, it blocks indexing until enough peers exist across the indexer cluster to fulfill the replication factor. In a basic, single-site cluster, this is usually desired behavior. However, in the case of a multisite cluster, you might want to restart indexing even though you do not have enough available peers to fulfill all aspects of the site replication factor (for example, in the case of site failure).

The two cases where this need typically arises are:

- A site goes down and you later need to restart the manager for any reason.
- The site with the manager goes down and you bring up a stand-by manager on another site.

If a site goes down but the manager, running on another site, remains up, indexing continues as usual, because the manager only runs the check at start-up.

Run the `splunk set indexing-ready` command on the manager to unblock indexing when replication factor number of peers are not available:

```
splunk set indexing-ready -auth admin:your_password
```

For example, assume you have a three-site cluster configured with "site_replication_factor = origin:1, site1:2, site2:2, site3:2, total:7", with the manager located on site1. If site2 goes down and you subsequently restart the manager, the manager blocks indexing after it restarts, because it is waiting to hear from a minimum of two peers on site2 ("site2:2"). In this situation, you can use the command to restart indexing on the remaining sites.

Similarly, if site1, which has the manager, goes down and you bring up a stand-by manager on site2, the new manager initially blocks indexing because site1 is not available. You can then use the command to tell the new manager to restart indexing.

Important: You must run the `splunk set indexing-ready` command every time you restart the manager under the listed circumstances. The command unblocks indexing only for the current restart.

Note: Although this command is designed with site failure in mind, you can also use it to restart indexing on a single-site cluster prior to the replication factor number of peers being available. In that circumstance, however, it is usually better just to wait until the replication number of peers rejoin the cluster.

Convert a multisite indexer cluster to single-site

You can convert a multisite indexer cluster to a basic, single-site cluster. When you do so, all peers and search heads become part of the implicit single site.

1. Stop all cluster nodes (manager/peers/search heads).

2. On the manager node, edit `server.conf`:

a. Set `multisite` to `false`.

b. Set the single-site `replication_factor` and `search_factor` attributes to implement the desired replication behavior.

c. Remove the `site` attribute.

3. On each search head, edit `server.conf`:

a. Set `multisite` to `false`.

b. Remove the `site` attribute.

4. On each peer, edit `server.conf`:

a. Remove the `site` attribute.

5. Start the manager node.

6. Start the peer nodes and search heads.

Note the following:

- The manager ignores any multisite attributes remaining in `server.conf` (`site_replication_factor` and so on).

- Upon restart, the manager picks a primary copy for each bucket.
- After the conversion, any bucket copies that exceed the single-site replication factor remain in place. For information on removing these extra copies, see Remove excess bucket copies.
- Future bucket replication and bucket fixing will follow the values set for the single-site replication and search factors.

For information on how to convert a single-site cluster to multisite, see Migrate an indexer cluster from single-site to multisite.

Move a peer to a new site

Use this procedure to relocate a peer to another site. This can be useful if a peer was shipped to the wrong site and the mistake was discovered only after the peer was deployed on the cluster.

1. Take the peer offline with the `offline` command, as described in Take a peer offline. The manager will reassign the bucket copies handled by this peer to other peers in the same site.

2. Ship the peer's server to the new site.

3. Delete the entire Splunk Enterprise installation from the server, including its index database with all bucket copies.

4. Reinstall Splunk Enterprise on the server, re-enable clustering, and set the peer's site value to the new site location.

The peer rejoins the cluster as a new peer.

Decommission a site in a multisite indexer cluster

To decommission a site, you reconfigure several site-specific attributes.

Decommission a site

Caution: Before proceeding, be aware of these issues:

- If a peer on the decommissioned site contains any buckets that were created before the peer was part of a cluster, such buckets exist only on that peer and therefore will be lost when the site is decommissioned.
- Similarly, if the decommissioned site started out as a single-site cluster before it became part of a multisite cluster, any buckets that were created when it was a single-site cluster exist only on that site and will be lost when the site is decommissioned.
- Once the manager restarts at the end of the decommissioning process, it will commence bucket fix-up activities to return the cluster to a **complete** state. This can take a considerable amount of time, particularly if the decommissioned site held a large number of origin buckets.

Prerequsites

The cluster must meet these conditions before you decommission one of its sites:

- The cluster must be in a complete state.

- The manager node cannot be on the site that you are planning to decommission. If it is, move the manager to to a new site, following the guidelines in Handle manager node site failure.
- The `site_replication_factor` attribute must be configured so that at least one copy of each bucket resides on a site not due for decommissioning. For example, in a two-site cluster, a valid configuration would be `site_replication_factor = origin:1,total:2`.
- The `site_search_factor` attribute must be configured so that at least one searchable copy of each bucket resides on a site not due for decommissioning. For example, in a two-site cluster, a valid configuration would be `site_search_factor = origin:1,total:2`.
- If you need to reconfigure `site_replication_factor` or `site_search_factor` so that all buckets have copies on other sites, you must then wait until the manager completes fix-up activities and returns the cluster to a complete state before proceeding with the decommissioning.

Steps

1. For each search head on the site, either disable the search head or change the search head's `site` attribute to specify a remaining site. For example, to change the search head's site to site2:

   ```
   splunk edit cluster-manager https://10.160.31.200:8089  -site site2
   ```

   See Configure the search heads.

2. For each forwarder that uses indexer discovery and specifies the decommissioned site, change its `site` attribute to specify a remaining site. For example, to change the forwarder's site to site2:

   ```
   [general]
   site = site2
   ```
   You must restart the forwarder for the configuration change to take effect. See Use indexer discovery in a multisite cluster.
3. Run `splunk enable maintenance-mode` on the manager. This step prevents unnecessary bucket fix-ups. See Use maintenance mode.
4. To confirm that the manager has entered maintenance mode, run `splunk show maintenance-mode`.
5. Update these attributes on the manager:
   - `available_sites`
   - `site_replication_factor`
   - `site_search_factor`
   - `site_mappings`

   For details on the necessary updates, see Reconfigure attributes.
6. Restart the manager. This step causes the attribute changes to take effect.
   Note: When the manager restarts, the peers from the decommissioned site will unsuccessfully try to rejoin the cluster. Ignore the resulting messages.
7. Run `splunk disable maintenance-mode` on the manager. This step starts fix-up activities for peers on the remaining sites.
8. To confirm that the manager has left maintenance mode, run `splunk show maintenance-mode`.
9. Run `splunk stop` on each peer on the decommissioned site. You can now remove these peers from the site.
10. To verify that that the decommissioning was successful, look at the top of the manager node dashboard. It should state that both the search factor and the replication factor are met. When both are met, the cluster is in a complete state, indicating that the decommissioning was successful. See View the manager node dashboard.
    Note: Because site decommissioning typically involves a large amount of bucket fix-up activity, it can take a considerable amount of time for the cluster to return to its complete state.

Reconfigure attributes

When you decommission a site, you must make changes to several site-specific attributes in `server.conf` on the manager node:

- `available_sites`: Remove the decommissioned site from the site list for this attribute.
- `site_replication_factor` and `site_search_factor`: If the decommissioned site is an explicit site in either of these attributes, remove it and otherwise reconfigure the attribute as necessary.
- `site_mappings`: Add a mapping for the decommissioned site to this attribute. See Map the decommissioned site.

You must restart the manager node after you change any of these attributes.

Map the decommissioned site

When a site gets decommissioned, the origin bucket copies for that site remain bound to the decommissioned site until you map the site to a remaining, active site. This can make the cluster unable to meet its replication or search factors.

To deal with this issue, you can map decommissioned sites to active sites. The bucket copies for which a decommissioned site is the origin site will then be replicated to the active site specified by the mapping, allowing the cluster to again meet its replication and search factors.

Note: The `site_replication_factor` and `site_search_factor` attributes determine the cluster's number of origin bucket copies.

Syntax

Map the site before you decommission it. See Decommission a site.

To map a site that you are planning to decommission to a remaining site, use the `site_mappings` attribute in `server.conf`. Set this attribute on the manager node only, using this syntax:

```
site_mappings = <comma-separated string>
```

Note the following:

- The `<comma-separated string>` contains mappings from decommissioned sites to the remaining, active sites. These mappings can be of two types:
  - `<decommissioned_site_id>:<active_site_id>`. For example, site2:site3, where site2 is a decommissioned site and site3 is an active site. This is called an explicit mapping. There can be multiple explicit mappings.
  - `default_mapping:<active_site_id>`. For example, default_mapping:site4, where site4 is an active site. There can be at most one default mapping. It is recommended that you always include a default mapping, as fallback for an incorrect or missing explicit mapping.
- In the case of `<decommissioned_site_id>:<active_site_id>`, the origin bucket copies in `<decommissioned_site_id>` will get replicated from remaining sites to peers on the `<active_site_id>`. This allows the cluster to meet the requirements of its replication and search factors.
- In the case of `default_mapping:<active_site_id>`, the origin bucket copies for any decommissioned site without an explicit mapping will get replicated to `<active_site_id>`.
- If an active site in a mapping is itself later decommissioned, its previous mappings must be remapped to a currently active site. For example, in the case of site2:site3, if site3 is itself decommissioned, you must replace the previous mapping, site2:site3, with a new set of mappings that map both site2 and site3 to an active site, such as site4, using the string site2:site4,site3:site4.

Restart the manager node after changing this attribute.

Examples

These examples assume a cluster that originally had five sites, site1 through site5.

- **"site_mappings = site2:site3"** This configuration maps a decommissioned site2 to an active site3. The mapping causes origin bucket copies for site2 to replicate to site3. There is no default site mapping.
- **"site_mappings = site1:site3,default_mapping:site4"** This configuration maps a decommissioned site1 to site3 and maps all other decommissioned sites to site4. The mapping causes origin bucket copies for the decommissioned sites to replicate to their respective mapped sites.
- **"site_mappings = default_mapping:site5"** This configuration maps all decommissioned sites to site5. The mapping causes origin bucket copies for all decommissioned sites to replicate to site5.

How indexer clusters work

Basic indexer cluster concepts for advanced users

To understand how a cluster functions, you need to be familiar with a few concepts:

- **Replication factor.** This specifies how many copies of the data the cluster maintains. It influences the cluster's resiliency, its ability to withstand multiple node failures.
- **Search factor.** This specifies how many copies of the data are **searchable**. It influences how quickly a cluster can recover from a downed node.
- **Buckets.** These are the basic storage containers for indexes. They correspond to subdirectories in the indexer's database.
- **Cluster states.** These states describe the health of the cluster.

You can find an overview to these concepts in the introductory topic on cluster architecture, Basic indexer cluster architecture. Topics in the chapter you are now reading provide more detail.

Replication factor

As part of setting up an indexer cluster, you specify the number of copies of data that you want the cluster to maintain. Peer nodes store incoming data in **buckets**, and the cluster maintains multiple copies of each bucket. The cluster stores each bucket copy on a separate peer node. The number of copies of each bucket that the cluster maintains is the **replication factor**.

Replication factor and cluster resiliency

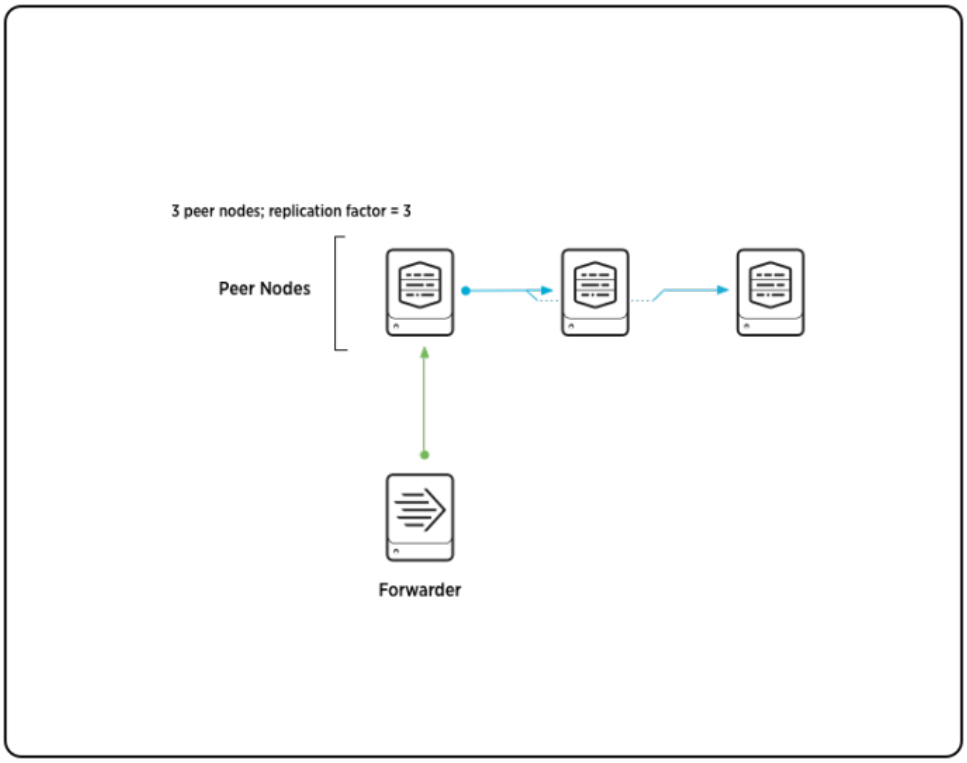
The cluster can tolerate a failure of (replication factor - 1) peer nodes. For example, to ensure that your system can tolerate a failure of two peers, you must configure a replication factor of 3, which means that the cluster stores three identical copies of each bucket on separate nodes. With a replication factor of 3, you can be certain that all your data will be available if no more than two peer nodes in the cluster fail. With two nodes down, you still have one complete copy of data available on the remaining peers.

By increasing the replication factor, you can tolerate more peer node failures. With a replication factor of 2, you can tolerate just one node failure; with a replication factor of 3, you can tolerate two concurrent failures; and so on.

The trade-off is that you need to store and process all those copies of data. Although the replicating activity doesn't consume much processing power, still, as the replication factor increases, you need to run more indexers and provision more storage for the indexed data. On the other hand, since data replication itself requires little processing power, you can take advantage of the multiple indexers in a cluster to ingest and index more data. Each indexer in the cluster can function as both originating indexer ("source peer") and replication target ("target peer"). It can index incoming data and also store copies of data from other indexers in the cluster.

Example: Replication factor in action

In the following diagram, one peer is receiving data from a forwarder, which it processes and then streams to two other peers.The cluster will contain three complete copies of the peer's data, one copy on each peer.



Note: This diagram represents a highly simplified version of peer replication, where all data is entering the system through a single peer. There are a few issues that add complexity to a real-life scenario:

- In most clusters, each of the peer nodes would be functioning as both source and target peer, receiving external data from a forwarder, as well as replicated data from other peers.
- To accommodate horizontal scaling, a cluster with a replication factor of 3 could consist of many more peers than three. At any given time, each source peer would be streaming copies of its data to two target peers, but each time it started a new hot bucket, its set of target peers could potentially change.

Later topics in this chapter describe in detail how clusters process data.

Replication factor in multisite clusters

A multisite cluster uses a special version of the replication factor, the site replication factor. This determines not only the number of copies that the entire cluster maintains but also the number of copies that each site maintains. For information on the site replication factor, see Configure the site replication factor.

Search factor

When you configure the manager node, you designate a **search factor**. The search factor determines the number of **searchable** copies of data the indexer cluster maintains. In other words, the search factor determines the number of searchable copies of each **bucket**. The default value for the search factor is 2, meaning that the cluster maintains two searchable copies of all data. The search factor must be less than or equal to the **replication factor**.

Searchable and non-searchable bucket copies

The difference between a searchable and a **non-searchable** copy of a bucket is this: The searchable copy contains both the data itself and some very extensive index files that the peer node uses to search the data. The non-searchable copy contains just the data. Even the data stored in the non-searchable copy, however, has undergone initial processing and is stored in a form that makes it possible to create the index files later, if necessary. For more information on the files that constitute Splunk Enterprise indexes, read the subtopic Data files.

Search recovery from peer node failure

With a search factor of at least 2, the cluster is able to continue searching with little interruption if a peer node goes down. For example, say you specify a replication factor of 3 and a search factor of 2. The cluster will maintain three copies of all buckets on separate peers across the cluster, and two copies of each bucket will be searchable. Then, if a peer goes down and it contains a bucket copy that has been participating in searches, a searchable copy of that bucket on another peer can immediately step in and start participating in searches.

On the other hand, if the cluster's search factor is only 1 and a peer goes down, there will be a significant lag before searching can resume across the full set of cluster data. Although non-searchable copies of the buckets can be made searchable, doing so takes time, because the index files must first be built from the raw data file. The processing time can be significant if the peer that went down was storing a large quantity of searchable data. For help estimating the time needed to make non-searchable copies searchable, look here.

The reason you might want to limit the number of searchable copies on your cluster is because searchable data occupies a lot more storage than non-searchable data. The trade-off, therefore, is between quick access to all your data in case of peer node failure versus increased storage requirements. For help estimating the relative storage sizes of searchable and non-searchable data, read Storage considerations. For most needs, the default search factor of 2 represents the right trade-off.

Search factor in multisite clusters

A multisite cluster uses a special version of the search factor, the site search factor. This determines not only the number of searchable copies that the entire cluster maintains but also the number of searchable copies that each site maintains. For information on the site search factor, see Configure the site search factor.

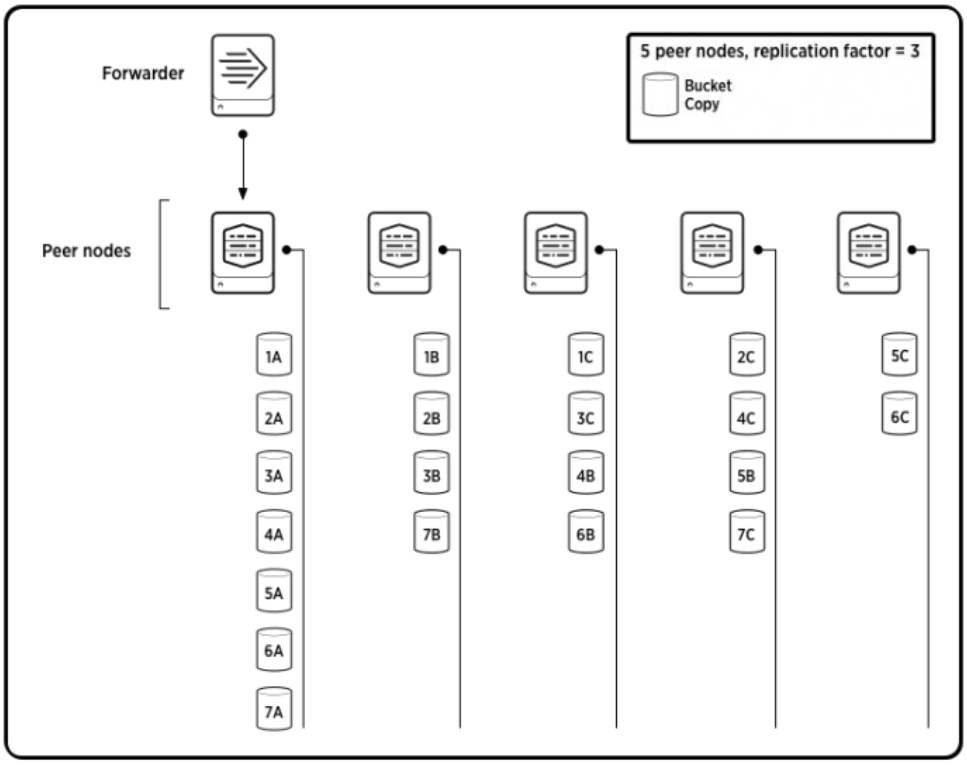
Buckets and indexer clusters

Splunk Enterprise stores indexed data in **buckets**, which are directories containing both the data and index files into the data. An index typically consists of many buckets, organized by age of the data.

The indexer cluster replicates data on a bucket-by-bucket basis. The original bucket copy and its replicated copies on other peer nodes contain identical sets of data, although only **searchable** copies also contain the index files.

In a cluster, copies of buckets originating from a single source peer can be spread across many target peers. For example, if you have five peers in your cluster and a replication factor of 3 (a typical scenario for horizontal scaling), the cluster will maintain three copies of each bucket (the original copy on the source peer and replicated copies on two target peers). Each time the source peer starts a new hot bucket, the manager node gives the peer a new set of target peers to replicate data to. Therefore, while the original copies will all be on the source peer, the replicated copies of those buckets will be randomly spread across the other peers.This behavior is not configurable. The one certainty is that you will never have two copies of the same bucket on the same peer. In the case of a multisite cluster, you can also configure the site location of the replicated copies, but you still cannot specify the actual peer location.

The following diagram shows the scenario just described - five peers, a replication factor of 3, and seven original source buckets, with their copies spread across all the peers. To reduce complexity, the diagram only shows the buckets for data originating from one peer. In a real-life scenario, most, if not all, of the other peers would also be originating data and replicating it to other peers on the cluster.



In this diagram, 1A is a source bucket. 1B and 1C are copies of that bucket. The diagram uses the same convention with 2A/B/C, 3A/B/C, and so on.

You need a good grasp of buckets to understand cluster architecture. The rest of this section describes some bucket concepts of particular importance for a clustered deployment. For a thorough introduction to buckets, read "How the indexer stores indexes".

Data files

There are two key types of files in a bucket:

- The processed external data in compressed form (**rawdata**)
- Indexes that point to the rawdata (**index files**, also referred to as **tsidx files**)

Buckets contain a few other types of files as well, but these are the ones that are most important to understand.

Rawdata is not actually "raw" data, as the term might be defined by a dictionary. Rather, it consists of the external data after it has been processed into events. The processed data is stored in a compressed rawdata journal file. As a journal file, the rawdata file, in addition to containing the **event data**, contains all information necessary to generate the associated index files, if they are missing.

All bucket copies, both **searchable** and **non-searchable**, contain rawdata files. Searchable copies also contain index files.

When a peer node receives a block of data from a forwarder, it processes the data and adds it to the rawdata file in its local hot bucket. It also indexes it, creating the associated index files. In addition, it streams copies of just the processed rawdata to each of its target peers, which then adds it to the rawdata file in its own copy of the bucket. The rawdata in both the original and the replicated bucket copies are identical.

If the cluster has a search factor of 1, the target peers store only the rawdata in the bucket copies. They do not generate index files for the data. By not storing the index files on the target peers, you limit storage requirements. Because the rawdata is stored as a journal file, if the peer maintaining the original, fully indexed data goes down, one of the target peers can step in and generate the indexes from its copy of the rawdata.

If the cluster has a search factor greater than 1, some or all of the target peers also create index files for the data. For example, say you have a replication factor of 3 and a search factor of 2. In that case, the source peer streams its rawdata to two target peers. One of those peers then uses the rawdata to create index files, which it stores in its copy of the bucket. That way, there will be two searchable copies of the data (the original copy and the replicated copy with the index files). As described in "Search factor", this allows the cluster to recover more quickly in case of peer node failure. For more information on searchable bucket copies, see "Bucket searchability" later in this topic.

See these topics for more information on bucket files:

- For information on how bucket files get regenerated when a peer goes down, read "What happens when a peer node goes down".
- For information on the relative sizes of rawdata and index files, read "Storage considerations".

Bucket stages

As a bucket ages, it rolls through several stages:

- hot
- warm
- cold
- frozen

For detailed information about these stages, read "How the indexer stores indexes".

For the immediate discussion of cluster architecture, you just need a basic understanding of these bucket stages. A hot bucket is a bucket that's still being written to. When an indexer finishes writing to a hot bucket (for example, because the bucket reaches a maximum size), it rolls the bucket to warm and begins writing to a new hot bucket. Warm buckets are readable (for example, for searching) but the indexer does not write new data to them. Eventually, a bucket rolls to cold and then to frozen, at which point it gets archived or deleted.

There are a couple other details that are important to keep in mind:

- Hot/warm and cold buckets are stored in separately configurable locations.
- The filename of a warm or cold bucket includes the time range of the data in the bucket. For detailed information on bucket naming conventions, read "What the index directories look like".
- Searches occur across hot, warm, and cold buckets.
- The conditions that cause buckets to roll are configurable, as described in "Configure index storage".
- For storage hardware information, such as help on estimating storage requirements, read "Storage considerations".

Bucket searchability and primacy states

A copy of a bucket is either **searchable** or **non-searchable**. Since a cluster can maintain multiple searchable copies of a bucket, the cluster needs a way to identify which copy participates in a search. To handle this, clusters use the concept of **primacy**. A searchable bucket copy is either **primary** or non-primary.

A bucket copy is searchable if it contains index files in addition to the rawdata file. The peer receiving the external data indexes the rawdata and also sends copies of the rawdata to its peers. If the search factor is greater than 1, some or all of those peers will also generate index files for the buckets they're replicating. So, for example, if you have a replication factor of 3 and a search factor of 2 and the cluster is **complete**, the cluster will contain three copies of each bucket. All three copies will contain the rawdata file, and two of the copies (the copy on the source peer and one of the copies on the target peers) will also contain index files and therefore be searchable. The third copy will be non-searchable, but it can be made searchable if necessary. The main reason that a non-searchable copy gets made searchable is because a peer holding a searchable copy of the bucket goes down.

A primary copy of a bucket is the searchable copy that participates in a search. A single-site **valid** cluster has exactly one primary copy of each bucket. That way, one and only one copy of each bucket gets searched. If a node with primary copies goes down, searchable but non-primary copies on other nodes can immediately be designated as primary, thus allowing searches to continue without any need to first wait for new index files to be generated.

Note: In the case of a multisite cluster, a valid cluster is a cluster that has a set of primary copies for each site that supports search affinity. In search affinity, search heads perform searches across the peers on their local site. This requires that each site have its own set of primary buckets.

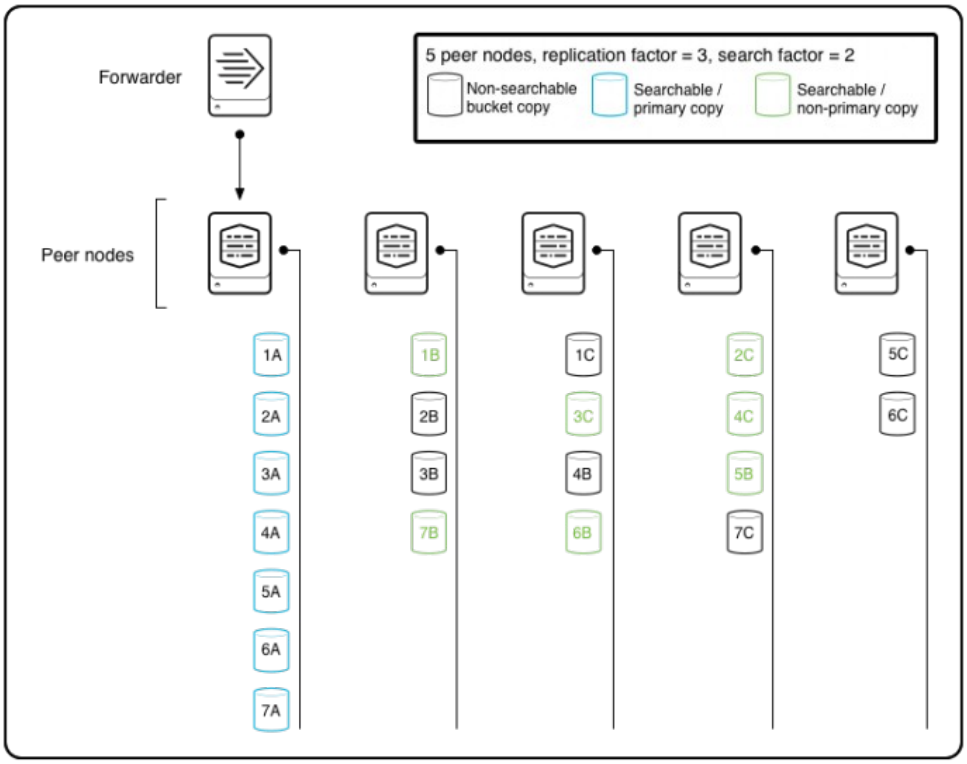
Initially, the copy of the bucket on the peer originating the data is the primary copy, but this can change over time. For example, if the peer goes down, the manager node reassigns primacy from any primary copies on the downed peer to corresponding searchable copies on remaining peers. For more information on this process, read "What happens when a peer node goes down".

Primacy reassignment also occurs when the manager rebalances the cluster, in an attempt to achieve a more even distribution of primary copies across the set of peers. Rebalancing occurs under these circumstances:

- A peer joins or rejoins a cluster.
- A manager node rejoins the cluster.
- You manually invoke the `rebalance_primaries` REST endpoint on the manager node.

See "Rebalance the indexer cluster primary buckets" for details.

The following diagram shows buckets spread across all the peers, as in the previous diagram. The cluster has a replication factor of 3 and a search factor of 2, which means that the cluster maintains two searchable copies of each bucket. Here, the copies of the buckets on the source peer are all primary (and therefore also searchable). The buckets' second searchable (but non-primary) copies are spread among most of the remaining peers in the cluster.



The set of primary bucket copies define a cluster's generation, as described in the next section.

Generations

A **generation** identifies which copies of a cluster's buckets are primary and therefore will participate in a search.

Note: The actual set of buckets that get searched also depends on other factors such as the search time range. This is true for any indexer, clustered or not.

The generation changes over time, as peers leave and join the cluster. When a peer goes down, its primary bucket copies get reassigned to other peers. The manager also reassigns primaries under certain other circumstances, in a process known as "cluster rebalancing".

Here is another way of defining a generation: A generation is a snapshot of a **valid** state of the cluster; "valid" in the sense that every bucket on the cluster has exactly one primary copy.

All peers that are currently registered with the manager participate in the current generation. When a peer joins or leaves the cluster, the manager creates a new generation.

Note: Since the process of reassigning primary to new bucket copies is not instantaneous, the cluster might quickly go through a number of generations while reassigning primacy due to an event such as a downed peer, particularly in the case where numerous primaries were residing on the downed peer.

The generation is a cluster-wide attribute. Its value is the same across all sites in a multisite cluster.

How cluster nodes use the generation

Here is how the various cluster nodes use generation information:

- The manager creates each new generation, and assigns a **generation ID** to it. When necessary, it communicates the current generation ID to the peers and the search head. It also keeps track of the primary bucket copies for each generation and on which peers they are located.
- The peers keep track of which of their bucket copies are primary for each generation. The peers retain primary information across multiple generations.
- For each search, the search head uses the generation ID that it gets from the manager to determine which peers to search across.

When the generation changes

The generation changes under these circumstances:

- The manager comes online.
- A peer joins the cluster.
- A peer goes down, either intentionally (through the CLI `offline` command) or unintentionally (by crashing). When a peer goes down, the manager reassigns primacy from bucket copies on the downed node to searchable copies of the same buckets on the remaining nodes and creates a new generation.
- Whenever rebalancing of the primary copies occurs, such as when you manually hit the `rebalance_primaries` REST endpoint on the manager. For information on rebalancing, see "Rebalance the indexer cluster primary buckets".

- When the manager resolves certain bucket anomalies.

The manager does not create a new generation merely when a bucket rolls from hot to warm, thus causing a new hot bucket to get created (unless the bucket rolled for one of the reasons listed above). In that situation, the set of peers doesn't change. The search head only needs to know which peers are part of the generation; that is, which peers are currently participating in the cluster. It does not need to know which bucket copies on a particular peer are primary; the peer itself keeps track of that information.

How the generation is used in searches

The search heads poll the manager for the latest generation information at regular intervals. When the generation changes, the manager gives the search heads the new generation ID and a list of the peers that belong to that generation. Each search head, in turn, gives the peers the ID whenever it initiates a search. The peers use the ID to identify which of their buckets are primary for that search.

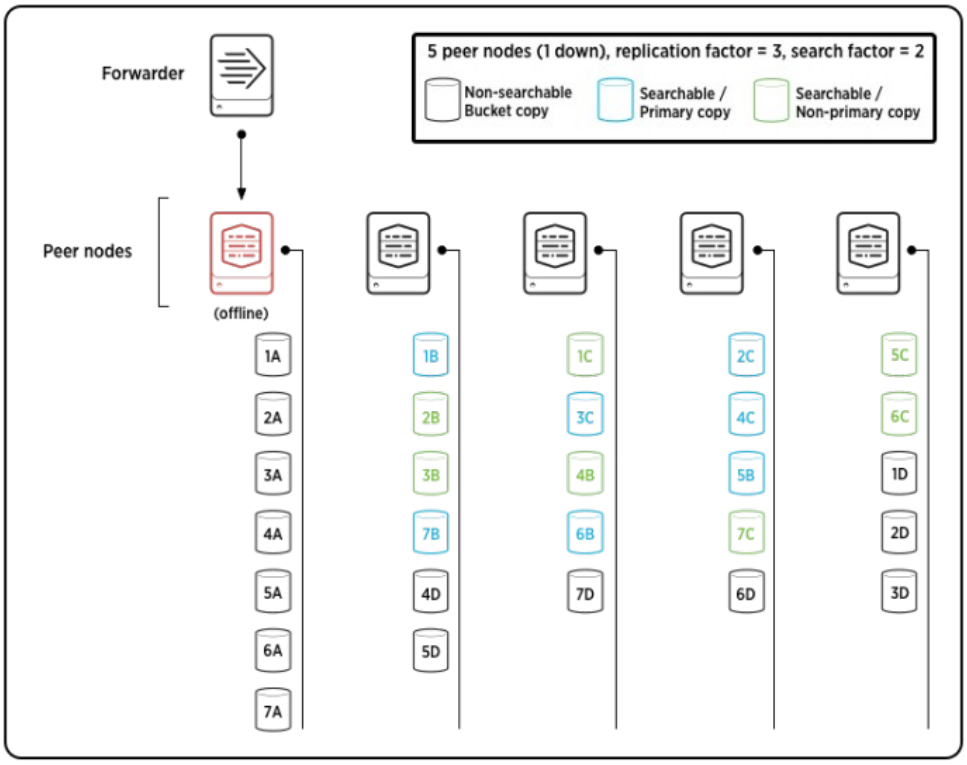
Usually, a search occurs over the most recent generation of primary bucket copies. In the case of long-running searches, however, it is possible that a search could be running across an earlier generation. This situation typically occurs because a peer went down in the middle of the search. This allows the long-running search to complete, even though some data might be missing due to the downed peer node. The alternative would be to start the search over again, which you can always do manually if necessary.

Why a downed peer causes the generation to change

The reason that a downed peer causes the manager to create a new generation is because, when a peer goes down, the manager reassigns the downed peer's primary copies to copies on other peers. A copy that was not primary for a previous generation becomes primary in the new generation. By knowing the generation ID associated with a search, a peer is able to determine which of its buckets are primary for that search.

For example, the diagram that follows shows the same simplified version of a cluster as earlier, after the source node holding all the primary copies has gone down and the manager has directed the remaining peers in fixing the buckets. First, the manager reassigned primacy to the remaining searchable copy of each bucket. Next, it directed the peers to make their non-searchable copies searchable, to make up for the missing set of searchable copies. Finally, it directed the replication of a new set of non-searchable copies (1D, 2D, etc.), spread among the remaining peers.

Even though the source node went down, the cluster was able to fully recover both its **complete** and **valid** states, with replication factor number (3) of total bucket copies, search factor number (2) of searchable bucket copies, and exactly one primary copy of each bucket. This represents a different generation from the previous diagram, because primary copies have moved to different peers.



Note: This diagram only shows the buckets originating from one of the peers. A more complete version of this diagram would show buckets originating from several peers as they have migrated around the cluster.

How the cluster handles frozen buckets

In the case of a standalone indexer, when a bucket rolls to frozen, the indexer deletes it from its `colddb` directory. Depending on its retirement policy, the indexer might copy it to an archive directory before deleting it. See "Archive indexed data."

In the case of an indexer cluster, when a peer freezes a copy of a bucket, it notifies the manager. The manager then stops doing fix-ups on that bucket. It operates under the assumption that the other peers will eventually freeze their copies of that bucket as well. If the freezing behavior is determined by the `maxTotalDataSizeMB` attribute, which limits the maximum size of an index, it can take some time for all copies of the bucket to freeze, as an index will typically be a different size on each peer. Therefore, the index can reach its maximumÂ size on one peer, causing the oldest bucket to freeze, even though the index is still under the limit on the other peers.

Note: In 6.3, a change was made in how the cluster responds to frozen primary bucket copies, in order to prolong the time that a bucket remains available for searching:

- In a pre-6.3 cluster, when a primary copy freezes, the cluster does not attempt to reassign the primary to any other remaining searchable copy. Searching on a bucket ceases once the primary is frozen.
- In 6.3 and later, when a primary copy freezes, the cluster reassigns the primary to another searchable copy, if one exists. Searching then continues on that bucket with the new primary copy. When that primary also freezes, the cluster attempts to reassign the primary yet again to another searchable copy. Once all searchable copies of the bucket have been frozen, searching ceases on that bucket.

In both pre-6.3 and post-6.3, when a copy freezes, the cluster does not perform fix-up on the bucket; that is, it does not attempt to create a new copy,or to convert a non-searchable copy to searchable, to meet replication and search factors for the bucket.

Indexer cluster states

An indexer cluster in good working order is both **valid** and **complete**:

- A **valid** cluster has exactly one **primary** copy of each bucket. In the case of a multisite cluster, a valid cluster has one full set of primary copies for each site that supports search affinity.
- A **complete** cluster has **replication factor** number of copies of each bucket and **search factor** number of **searchable** copies of each bucket. In the case of a multisite cluster, the number of bucket copies must also fulfill the site-specific requirements for the replication and search factors.

Note these points:

- A valid cluster is able to handle search requests across the entire set of data. A valid multisite cluster also meets any inherent search affinity goals.
- A complete cluster meets the designated requirements for failure tolerance.
- A complete cluster is also a valid cluster, but a valid cluster is not necessarily complete.

In addition, to ensure robust data availability, a cluster must not only be complete, but its search factor must be set to at least 2. This guarantees that a search head can continue to search across the cluster without interruption, if a peer goes down.

When a peer node goes down, the manager directs the cluster in activities designed to recover both its valid and complete states. In some cases, the cluster might be able to return to a valid state but not to a complete state. (For example, consider a cluster containing three peers, with a replication factor of 3. If one peer goes down, the cluster cannot recover its complete state as long as the peer remains down, but it should be able to recover its valid state.) See What happens when a peer node goes down for details on how the cluster recovers from a downed node.

How clustered indexing works

When discussing how data and messages flow between nodes during indexing, it is useful to distinguish between the two roles that a peer node plays:

- **Source node.** The source node ingests data from forwarders or other external sources.
- **Target node.** The target node receive streams of replicated data from the source nodes.

In practice, a single peer functions as both a source and a target node, often simultaneously.

Important: In a typical indexer cluster deployment, all the peer nodes are source nodes; that is, each node has its own set of external inputs. This is not a requirement, but it is generally the best practice. There is no reason to reserve some peers for use just as target nodes. The processing cost of storing replicated data is minimal, and, in any case, you cannot currently specify which nodes will receive replicated data. The manager node determines that on a bucket-by-bucket basis, and the behavior is not configurable. You must assume that all the peer nodes will serve as targets.

Note: In addition to replicating external data, each peer replicates its internal indexes to other peers in the same way. To keep things simple, this discussion focuses on external data only.

How the target peers are chosen

Whenever the source peer starts a hot bucket, the manager node gives it a list of target peers to stream its replicated data to. The list is bucket-specific. If a source peer is writing to several hot buckets, it could be streaming the contents of each bucket to a different set of target peers.

The manager chooses the list of target peers randomly. In the case of multisite clustering, it respects site boundaries, as dictated by the replication factor, but chooses the target peers randomly within those constraints.

When a peer node starts

These events occur when a peer node starts up:

1. The peer node registers with the manager and receives the latest **configuration bundle** from the manager.

2. The manager rebalances the **primary** bucket copies across the cluster and starts a new generation.

3. The peer starts ingesting external data, in the same way as any indexer. It processes the data into events and then appends the data to a rawdata file. It also creates associated index files. It stores these files (both the rawdata and the index files) locally in a hot bucket. This is the primary copy of the bucket.

4. The manager gives the peer a list of target peers for its replicated data. For example, if the replication factor is 3, the manager gives the peer a list of two target peers.

5. If the search factor is greater than 1, the manager also tells the peer which of its target peers should make its copy of the data **searchable**. For example, if the search factor is 2, the manager picks one specific target peer that should make its copy searchable and communicates that information to the source peer.

6. The peer begins streaming the processed rawdata to the target peers specified by the manager. It does not wait until its rawdata file is complete to start streaming its contents; rather, it streams the rawdata in blocks, as it processes the incoming data. It also tells any target peers if they need to make their copies searchable, as communicated to it by the manager in step 5.

7. The target peers receive the rawdata from the source peer and store it in local copies of the bucket.

8. Any targets with designated searchable copies start creating the necessary index files.

9. The peer continues to stream data to the targets until it rolls its hot bucket.

Note: The source and target peers rarely communicate with each other through their management ports. Usually, they just send and receive data to each other over their replication ports. The manager node manages the overall process.

This is just the breakdown for data flowing from a single peer. In a cluster, multiple peers will be both originating and receiving data at any time.

When a peer node rolls a hot bucket

When a source peer rolls a hot bucket to warm (for example, because the bucket has reached its maximum size), the following sequence of events occurs:

1. The source peer tells the manager and its target peers that it has rolled a bucket.

2. The target peers roll their copies of the bucket.

3. The source peer continues ingesting external data as this process is occurring. It indexes the data locally into a new hot bucket and streams the rawdata to a new set of target peers that it gets from the manager.

4. The new set of target peers receive the rawdata for the new hot bucket from the source peer and store it in local copies of the bucket. The targets with designated searchable copies also start creating the necessary index files.

5. The source peer continues to stream data to the targets until it rolls its next hot bucket. And so on.

How a peer node interacts with a forwarder

When a peer node gets its data from a forwarder, it processes it in the same way as any indexer getting data from a forwarder. However, in a clustering environment, you should ordinarily enable **indexer acknowledgment** for each forwarder sending data to a peer. This protects against loss of data between forwarder and peer and is the only way to ensure end-to-end data fidelity. If the forwarder does not get an acknowledgment for a block of data it has sent to a peer, it resends the block.

For details on how to set up forwarders to send data to peers, read "Use forwarders to get your data into the indexer cluster". To understand how peers and forwarders process indexer acknowledgment, read the section "How indexer acknowledgment works" in that topic.

How search works in an indexer cluster

In a single-site indexer cluster, the search head performs searches across the entire set of peers.

With a multisite indexer cluster, you can implement **search affinity.** With search affinity, searches occur across peers on the same site as the search head. This improves network efficiency without reducing access to the full set of cluster data.

Under rare circumstances, described later, you might want to initiate a search on a single peer.

Search across a single-site cluster

Searching across an indexer cluster works in a way similar to how **distributed search** works with non-clustered indexers. The main difference is that the **search head** gets its list of **search peers** from the manager node. It also gets a generation ID from the manager. After that, it communicates directly with the peers.

Note: In an indexer cluster search, the search peers are the set of cluster peers that are currently registered with the manager (in other words, the peers that are up-and-running and participating in the cluster).

When the search head initiates a search:

1. The search head contacts the manager node.

2. The manager node gives the search head the current generation ID and a list of the peers in that generation (that is, the peers that are currently registered with the manager).

3. The search head communicates with the search peers in the same way as in a distributed search not involving an indexer cluster. It provides the peers with exactly the same information (search request and knowledge bundle), except that it also gives the search peers the generation ID.

4. The search peers use the generation ID to identify which of their bucket copies, if any, are **primary** for the generation and thus need to participate in the search. As in any other search, the peers also use the search's time range to determine whether to search a particular bucket.

5. The search peers search their primary copies of buckets and send the results back to the search head, which consolidates the results.

You can integrate the indexer cluster with a search head cluster, for search head scaling and high availability. See "Integrate the search head cluster with an indexer cluster" in the *Distributed Search* manual.

For details on these and other available features of distributed search, read the *Distributed Search* manual, starting with "About distributed search". Also, read "Configure the search head" in this manual to learn about a few configuration differences when dealing with a search head in an indexer cluster.

Search a multisite cluster

The way that searches function in a multisite cluster depends on whether a search head is configured for search affinity.

Search locally in a multisite cluster

In a multisite cluster, you typically put search heads on each site. This allows you to take advantage of search affinity. In search affinity, searches normally return results only from peers on the same site as the requesting search head.

Search affinity is enabled by default for search heads on multisite clusters. However, you must perform a few steps to take advantage of it. Specifically, you must ensure that both the searchable data and the search heads are available locally. For information on how to set up search affinity, see "Implement search affinity in a multisite indexer cluster".

Once a site has been configured for search affinity, the actual search process works the same as for single-site clusters. The search head distributes the current generation ID, along with the search and knowledge bundle, to all peers across the entire cluster. The local peers, however, are the only ones to respond, if the cluster is in a **valid** state. They search their primary buckets and return results to the search head, using the generation ID to determine which of their bucket copies are primary.

If the cluster is not in a valid state and the local site does not have a full complement of primaries (typically, because some peers on the site are down), remote peers also participate in the search, providing results from any primaries missing from peers local to the site. In that case, the search does not adhere to search affinity, in order to maintain access to the full set of data. Once the site returns to a valid state, subsequent searches again adhere to search affinity.

Note: Hot bucket data is replicated in blocks, as described in "How clustered indexing works". If a local search involves a replicated hot bucket copy, where the origin copy is on a different site, there might be a time lag while the local peer waits to get the latest block of hot data from the originating peer. During this time, the search does not return the latest data.

Search globally in a multisite cluster

If search affinity is disabled for a search head (by setting its site to "site0"), the search head uses the site0 set of primaries, which ordinarily consists of primaries from all sites across the cluster. The site0 set of primaries is chosen randomly from searchable copies on each site, such that the site0 primary for bucketA might be on site1, the site0 primary for bucketB on site2, and so on.

While the selection result for site0 primaries is functionally random, the site0 primary for any bucket starts out as the bucket's primary copy on its source site. Over time, the site0 primary can change to a primary copy on a different (target) site through actions such as primary rebalance and cluster restarts.

For information on disabling search affiniity, see "Disable search affinity".

Search a single peer

For debugging purposes, you might occasionally need to search a single peer node. You do this by initiating the search directly on the peer, in the usual manner. The search accesses any **searchable** data on that peer. It does not have access to unsearchable copies of data on the peer or to searchable copies of data on other peers.

Note: Keep in mind that there is no way to configure exactly what data will be searchable on any individual peer. However, at a minimum, all data that has entered the cluster through the peer should be searchable on that peer.

How indexer clusters handle report and data model acceleration summaries

By default, indexer clusters do not replicate **report acceleration** and **data model acceleration** summaries. This means that only primary bucket copies will have associated summaries.

You can configure the manager so that the cluster does replicate summaries. All searchable bucket copies will then have associated summaries. **This is the recommended behavior.**

For details on report acceleration and data model acceleration, read the chapter Use data summaries to accelerate searches in the *Knowledge Manager Manual*.

For information on summaries and SmartStore, see How SmartStore handles report and data model acceleration summaries.

Where summaries reside

The summaries reside on the peer nodes in their own directories. You specify the directory locations in `indexes.conf`, with the `summaryHomePath` and `tstatsHomePath` attributes for the report acceleration and data model acceleration summaries, respectively. See the indexes.conf specification file for details.

A summary correlates with one or more buckets, depending on the summary's time span.

Replicated summaries

If you want the cluster to replicate summaries, you must set this attribute in the manager node's `server.conf` file:

```
[clustering]
```

```
summary_replication = true
```
You must restart the manager.

You can also use the CLI on the manager node to set the attribute:

```
splunk edit cluster-config -summary_replication true
```
This command does not require a restart.

When the cluster is configured to replicate summaries, the cluster takes steps to ensure that each searchable bucket copy has an associated summary copy:

- **For hot buckets.** The cluster creates a summary for each searchable copy of a hot bucket.
- **For warm/cold buckets.** The cluster replicates summaries for searchable copies of warm or cold buckets, when necessary. The cluster will use replication to fill in any missing summaries for searchable copies of warm or cold buckets.

When you turn on summary replication for the first time, the cluster might need to replicate a large number of summaries. This can have an impact on network bandwidth. To limit the number of summary replications occurring simultaneously, you can change the value of the `max_peer_sum_rep_load` attribute in the manager node's `server.conf` file. Its default value is 5.

Non-replicated summaries

If you keep the default behavior, the cluster will not replicate summaries. This section describes how the cluster handles non-replicated summaries.

A summary correlates with one or more buckets, depending on the summary's time span. When a summary is generated, it resides on the peer that holds the primary copy of the bucket for that time span. If the summary spans multiple buckets, and the primary copies of those buckets reside on multiple peers, then each of those peers will hold the corresponding part of the summary.

If **primacy** gets reassigned from one copy of a bucket to another (for example, because the peer holding the primary copy fails), the summary does not move to the peer with the new primary copy. Therefore, it becomes unavailable. It will not be available again until the next time Splunk Enterprise attempts to update the summary, finds that it is missing, and then regenerates it.

In multisite clusters, like single-site clusters, the summaries reside with the primary bucket copy. Because a multisite cluster has multiple primaries, one for each site that supports search affinity, the summaries reside with the particular primary that the generating search head accessed when running the search. Due to search affinity, that usually means that the summaries reside on primaries on the same site as the generating search head.

Summary replication and contention for resources

A search head with acceleration enabled runs special searches on the peers. These searches build the summaries. See, for example, the description of building report acceleration summaries in "Manage report acceleration" in the *Knowledge Manager Manual*.

In the case of replicated summaries on an indexer cluster, summaries are built on each searchable copy of a hot bucket. A peer node can be building summaries simultaneously both for copies of buckets originating on that peer and also for copies of buckets originating on other peers. This means that, with summary replication enabled, summary-generating

searches use more resources across the cluster, and the searches can take longer to complete.

How indexer cluster nodes start up

This topic describes what happens when:

- the manager node starts
- a peer node joins a new cluster
- a peer node joins an existing cluster

When the manager node starts

When a manager node comes online (either the first time or subsequently), it begins listening for cluster peers. Each online peer registers with the manager, and the manager adds it to the cluster. The manager waits until the replication factor number of peers register, and then it starts performing its functions.

When you first deploy the cluster, you must enable the manager before enabling the peer nodes, as described in "Indexer cluster deployment overview". The manager blocks indexing on the peers until you enable and restart the full replication factor number of peers.

If you subsequently restart the manager, it waits for a quiet period to allow all peers the opportunity to register with it. Once the quiet period ends and the replication factor number of peers register with it, the manager can start performing its coordinating functions, such as rebalancing the **primary** bucket copies and telling peers where to stream copies of incoming data. Therefore, you must make sure that there are at least replication factor number of peers running when you restart the manager.

After the quiet period is over, you can view the manager node dashboard for information on the status of the cluster.

For more information on what occurs when a manager goes down and then restarts, see "What happens when the manager node goes down".

When a peer joins a new cluster

When you initially deploy a cluster, you must first enable the manager and then enable the peer nodes, as described in "Indexer cluster deployment overview". The manager blocks indexing on the peers until you enable and restart the full replication factor number of peers.

Each peer registers with the manager when it comes online, and the manager automatically distributes the latest **configuration bundle** to it. The peer then validates the configuration bundle locally. The peer will only join the cluster if bundle validation succeeds.

The peer starts indexing data after the replication factor number of peers join the cluster.

When a peer joins an existing cluster

A peer can also come online at some later time, when the cluster is already up and running with a manager and the replication factor number of peers. The peer registers with the manager when it comes online, and the manager distributes the latest configuration bundle to it. The peer validates the configuration bundle locally. The peer joins the cluster only if bundle validation succeeds.

Note: Adding a new peer to an existing cluster causes a rebalancing of primary bucket copies across the set of existing peer nodes, as described in "Rebalance the indexer cluster primary buckets". However, the new peer won't get any of those primary copies, because the manager performs rebalancing only across the set of searchable copies, and a new peer has no searchable copies when it starts up. The peer can participate in future bucket replication, but the manager does not automatically shift bucket copies or reassign primaries from existing peers to the new peer.

What happens when a peer node goes down

A peer node can go down either intentionally (by invoking the CLI `offline` command, as described in Take a peer offline) or unintentionally (for example, by a server crashing).

No matter how a peer goes down, the manager coordinates remedial activities to recreate a full complement of bucket copies. This process is called **bucket fixing**. The manager keeps track of which bucket copies are on each node and what their states are (**primacy**, **searchability**). When a peer goes down, the manager can therefore instruct the remaining peers to fix the cluster's set of buckets, with the aim of returning to the state where the cluster has:

- Exactly one **primary** copy of each bucket (the **valid** state). In a multisite cluster, the valid state means that there is a primary copy on each site that supports search affinity, based on the `site_search_factor`.
- A full set of **searchable** copies for each bucket, matching the search factor. In the case of a multisite cluster, the number of searchable bucket copies must also fulfill the site-specific requirements for the search factor.
- A full set of copies (searchable and non-searchable combined) for each bucket, matching the replication factor (the **complete** state). In the case of a multisite cluster, the number of bucket copies must also fulfill the site-specific requirements for the replication factor.

Barring catastrophic failure of multiple nodes, the manager can usually recreate a valid cluster. If the cluster (or the site in a multisite cluster) has a search factor of at least 2, it can do so almost immediately. Whether or not a manager can also recreate a complete cluster depends on the number of peers still standing compared to the replication factor. At least replication factor number of peers must remain standing for a cluster to be made complete. In the case of a multisite cluster, each site must have available at least the number of peers specified for it in the site replication factor.

The time that the cluster takes to return to a complete state can be significant, because it must first stream buckets from one peer to another and make non-searchable bucket copies searchable. See Estimate the cluster recovery time when a peer gets decommissioned for more information.

Besides the remedial steps to fix the buckets, a few other key events happen when a node goes down:

- The manager rolls the **generation** and creates a new **generation ID**, which it communicates to both peers and search heads when needed.
- Any peers with copies of the downed node's hot buckets roll those copies to warm.

When a peer node gets taken offline intentionally

The `splunk offline` command removes a peer from the cluster and then stops the peer. It takes the peer down gracefully, allowing any in-progress searches to complete while quickly returning the cluster to a fully searchable state.

There are two versions of the `splunk offline` command:

- `splunk offline`: This is the fast version version of the `splunk offline` command. The peer goes down quickly, after a maximum of five minutes, even if searches or remedial activities are still in progress.

- `splunk offline --enforce-counts`: This is the enforce-counts version of the command, which is designed to validate that the cluster has returned to the complete state. If you invoke the `enforce-counts` flag, the peer does not go down until all remedial activities are complete.

For detailed information on running the `splunk offline` command, see Take a peer offline.

The fast offline command

The fast version of the `splunk offline` command has this syntax:

`splunk offline`
This version of the command causes the following sequence of actions to occur:

1. Partial shutdown. The peer immediately undergoes a partial shutdown. The peer stops accepting both external inputs and replicated data. It continues to participate in searches for the time being.

2. Primacy reassignment. The manager reassigns primacy from any primary bucket copies on the peer to available searchable copies of those buckets on other peers (on the same site, if a multisite cluster). At the end of this step, which should take just a few moments if the cluster (or the cluster site) has a search factor of at least 2, the cluster returns to the valid state.

During this step, the peer's status is `ReassigningPrimaries`.

3. Generation ID rolling. The manager rolls the cluster's generation ID. At the end of this step, the peer no longer joins in new searches, but it continues to participate in any in-progress searches.

4. Full shutdown. The peer shuts down completely after a maximum of five minutes, or when in-progress searches and primacy re-assignment activities complete - whichever comes first. It no longer sends heartbeats to the manager. For details on the conditions that precede full shutdown, see The fast offline process.

5. Restart count. After the peer shuts down, the manager waits the length of the `restart_timeout` attribute (60 seconds by default), set in `server.conf`. If the peer comes back online within this period, the manager rebalances the cluster's set of primary bucket copies but no further remedial activities occur.

During this step, the peer's status is `Restarting`. If the peer does not come back up within the timeout period, its status changes to `Down`.

6. Remedial activities. If the peer does not restart within the `restart_timeout` period, the manager initiates actions to fix the cluster buckets and return the cluster to a complete state. It tells the remaining peers to replicate copies of the buckets on the offline peer to other peers. It also compensates for any searchable copies of buckets on the offline peer by instructing other peers to make non-searchable copies of those buckets searchable. At the end of this step, the cluster returns to the complete state.

If taking the peer offline results in less than replication factor number of remaining peers, the cluster cannot finish this step and cannot return to the complete state. For details on how bucket-fixing works, see Bucket-fixing scenarios.

In the case of a multisite cluster, remedial activities take place within the same site as the offline peer, if possible. For details, see Bucket-fixing in multisite clusters.

The enforce-counts offline command

The enforce-counts version of the `splunk offline` command has this syntax:

```
splunk offline --enforce-counts
```

This version of the `splunk offline` command runs only if certain conditions are met. In particular it will not run if taking the peer offline would result in less than replication factor number of peers remaining in the cluster. See The enforce-counts offline process for the set of conditions necessary to run this command.

This version of the command initiates a process called **decommissioning**, during which the following sequence of actions occurs:

1. Partial shutdown. The peer immediately undergoes a partial shutdown. The peer stops accepting both external inputs and replicated data. It continues to participate in searches for the time being.

2. Primacy reassignment. The manager reassigns primacy from any primary bucket copies on the peer to available searchable copies of those buckets on other peers (on the same site, if a multisite cluster). At the end of this step, which should take just a few moments if the cluster (or the cluster site) has a search factor of at least 2, the cluster returns to the valid state.

During this step, the peer's status is `ReassigningPrimaries`.

3. Generation ID rolling. The manager rolls the cluster's generation ID. At the end of this step, the peer no longer joins in new searches, but it continues to participate in any in-progress searches.

4. Remedial activities. The manager initiates actions to fix the cluster buckets so that the cluster returns to a complete state. It tells the remaining peers to replicate copies of the buckets on the offline peer to other peers. It also compensates for any searchable copies of buckets on the offline peer by instructing other peers to make non-searchable copies of those buckets searchable. At the end of this step, the cluster returns to the complete state.

During this step, the peer's status is `Decommissioning`. For details on how bucket-fixing works, see Bucket-fixing scenarios.

5. Full shutdown. The peer shuts down when the cluster returns to the complete state. Once it shuts down, the peer no longer sends heartbeats to the manager. At this point, the peer's status changes to `GracefulShutdown`. For details on the conditions that precede full shutdown, see The enforce-counts offline process.

When a peer node goes down unintentionally

When a peer node goes down for any reason besides the `offline` command, it stops sending the periodic heartbeat to the manager. This causes the manager to detect the loss and initiate remedial action. The manager coordinates essentially the same actions as when the peer gets taken offline intentionally, except for the following:

- The downed peer does not continue to participate in ongoing searches.
- The manager waits only for the length of the heartbeat timeout (by default, 60 seconds) before reassigning primacy and initiating bucket-fixing actions.

Searches can continue across the cluster after a node goes down; however, searches will provide only partial results until the cluster regains its valid state.

In the case of a multisite cluster, when a peer goes down on one site, the site loses its search affinity, if any, until it regains its valid state. During that time, searches continue to provide full results through involvement of remote peers.

Bucket-fixing scenarios

To replace bucket copies on a downed peer, the manager coordinates bucket-fixing activities among the peers. Besides replacing all the bucket copies, the cluster must ensure that it has a full complement of primary and searchable copies.

Note: The bucket-fixing process varies somewhat for clusters with **SmartStore** indexes. See Indexer cluster operations and SmartStore.

Bucket fixing involves three activities:

- **Compensating for any primary copies** on the downed peer by assigning primary status to searchable copies of those buckets on other peers.
- **Compensating for any searchable copies** by converting non-searchable copies of those buckets on other peers to searchable.
- **Replacing all bucket copies** (searchable and non-searchable) by streaming a copy of each bucket to a peer that doesn't already have a copy of that bucket.

For example, assume that the downed peer had 10 bucket copies, and five of those were searchable, and two of the searchable copies were primary. The cluster must:

- Reassign primary status to two searchable copies on other peers.
- Convert five non-searchable bucket copies on other peers to searchable.
- Stream 10 bucket copies from one standing peer to another.

The first activity - converting a searchable copy of a bucket from non-primary to primary - happens very quickly, because searchable bucket copies already have index files and so there's virtually no processing involved. (This assumes that there is a spare searchable copy available, which requires the search factor to be at least 2. If not, the cluster has to first make a non-searchable copy searchable before it can assign primary status to the copy.)

The second activity - converting a non-searchable copy of a bucket to searchable - takes some time, because the peer must copy the bucket's index files from a searchable copy on another peer (or, if there's no other searchable copy of that bucket, then the peer must rebuild the bucket's index files from the rawdata file). For help estimating the time needed to make non-searchable copies searchable, read Estimate the cluster recovery time when a peer gets decommissioned.

The third activity - streaming copies from one peer to another - can also take a significant amount of time (depending on the amount of data to be streamed), as described in Estimate the cluster recovery time when a peer gets decommissioned.

The two examples below illustrate how a manager 1) recreates a valid and complete cluster and 2) creates a valid but incomplete cluster when insufficient nodes remain standing. The process operates the same whether the peer goes down intentionally or unintentionally.

Remember these points:

- A cluster is valid when there is one primary searchable copy of every bucket. Any search across a valid cluster provides a full set of search results.

- A cluster is complete when the cluster has replication factor number of copies of buckets with search factor

number of searchable copies.

- A cluster can be valid but incomplete if there are searchable copies of all buckets, but there are less than replication factor number of copies of buckets. So, if a cluster with a replication factor of 3 has exactly three peer nodes and one of those peers goes down, the cluster can be made valid but it cannot be made complete, since, with just two standing nodes, it is not possible to fulfill the replication factor by maintaining three sets of buckets.

Example: Fixing buckets to create a valid and complete cluster

Assume:

- The peer went down unintentionally (that is, not in response to an `offline` command).

- The downed peer is part of a cluster with these characteristics:
  - 5 peers, including the downed peer
  - replication factor = 3
  - search factor = 2
- The downed peer holds these bucket copies:
  - 3 primary copies of buckets
  - 10 searchable copies (including the primary copies)
  - 20 total bucket copies (searchable and non-searchable combined)

When the peer goes down, the manager sends messages to various of the remaining peers as follows:

1. For each of the three primary bucket copies on the downed peer, the manager identifies a peer with another searchable copy of that bucket and tells that peer to mark the copy as primary.

When this step finishes, the cluster regains its valid state, and any subsequent searches provide a full set of results.

2. For each of the 10 searchable bucket copies on the downed peer, the manager identifies 1) a peer with a searchable copy of that bucket and 2) a peer with a non-searchable copy of the same bucket. It then tells the peer with the searchable copy to stream the bucket's index files to the second peer. When the index files have been copied over, the non-searchable copy becomes searchable.

3. For each of the 20 total bucket copies on the downed peer, the manager identifies 1) a peer with a copy of that bucket and 2) a peer that does not have a copy of the bucket. It then tells the peer with the copy to stream the bucket's rawdata to the second peer, resulting in a new, non-searchable copy of that bucket.

When this last step finishes, the cluster regains its complete state.

Example: Fixing buckets to create a valid but incomplete cluster

Assume:

- The peer went down unintentionally (that is, *not* in response to an `offline` command).

- The downed peer is part of a cluster with these characteristics:
  - 3 peers, including the downed peer
  - replication factor = 3
  - search factor = 1
- The downed peer holds these bucket copies:
  - 5 primary copies of buckets

- 5 searchable copies (the same as the number of primary copies; because the search factor = 1, all searchable copies must also be primary.)
- 20 total bucket copies (searchable and non-searchable combined)

Since the cluster has just three peers and a replication factor of 3, the downed peer means that the cluster can no longer fulfill the replication factor and therefore cannot be made complete.

When the peer goes down, the manager sends messages to various of the remaining peers as follows:

1. For each of the five searchable, primary bucket copies on the downed node, the manager first identifies a peer with a non-searchable copy and tells the peer to make the copy searchable. The peer then begins building index files for that copy. (Because the search factor is 1, there are no other searchable copies of those buckets on the remaining nodes. Therefore, it is not possible for the remaining peers to make non-searchable bucket copies searchable by streaming index files from another searchable copy. Rather, they must employ the slower process of creating index files from the rawdata file of a non-searchable copy.)

2. The manager then tells the peers from step 1 to mark the five, newly searchable copies as primary. Unlike the previous example, the step of designating other bucket copies as primary cannot occur until non-searchable copies have been made searchable. Because the cluster's search factor = 1, there are no standby searchable copies.

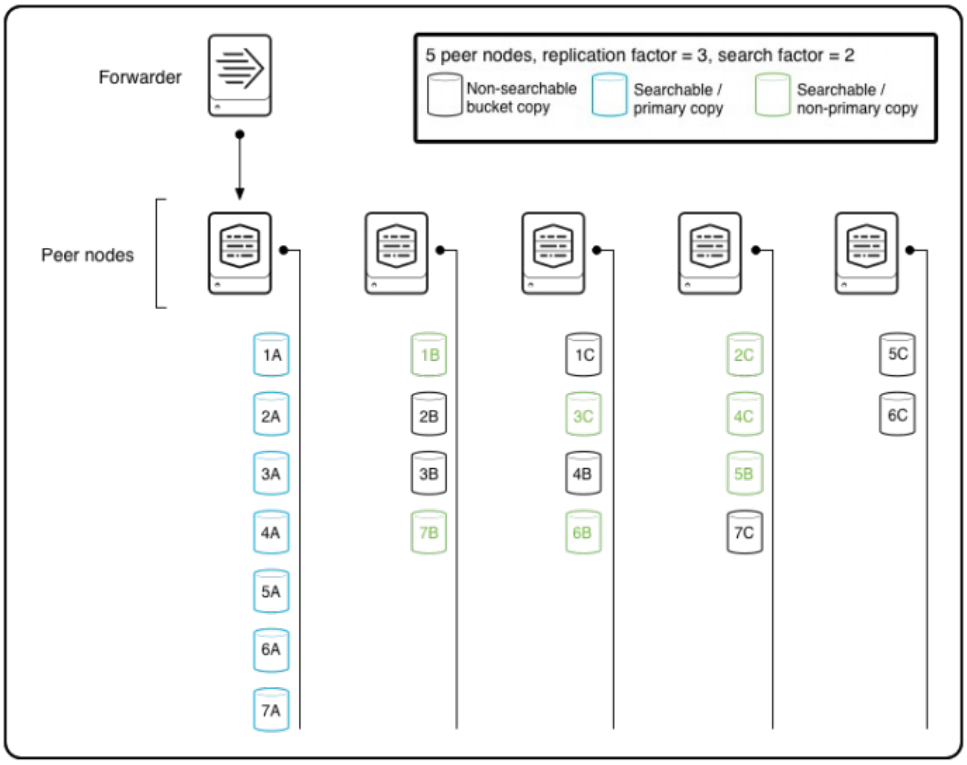
Once step 2 completes, the cluster regains its valid state. Any subsequent searches provide a full set of results.

3. For the 20 bucket copies on the downed node, the manager cannot initiate any action to make replacement copies (so that the cluster will again have the three copies of each bucket, as specified by the replication factor), because there are too few peers remaining to hold the copies.

Since the cluster cannot recreate a full set of replication factor number of copies of buckets, the cluster remains in an incomplete state.

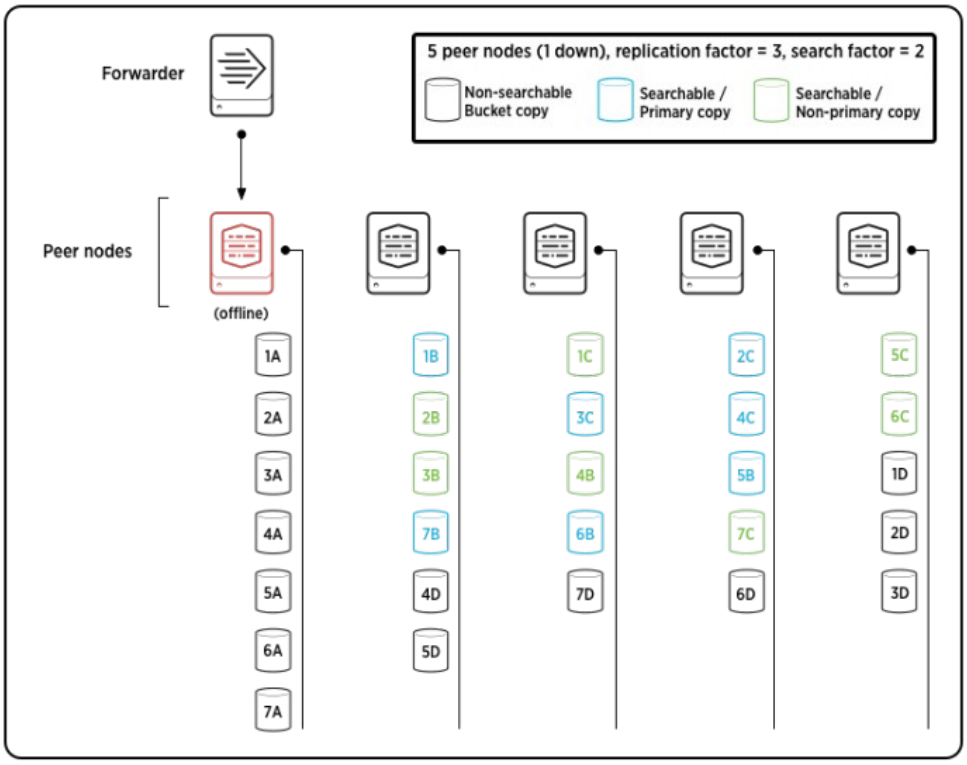
Pictures, too

The following diagram shows a cluster of five peers, with a replication factor of 3 and a search factor of 2. The primary bucket copies reside on the source peer receiving data from a forwarder, with searchable and non-searchable copies of that data spread across the other peers.



Note: This is a highly simplified diagram. To reduce complexity, it shows only the buckets for data originating on one peer. In a real-life scenario, most, if not all, of the other peers would also be originating data and replicating it to other peers on the cluster.

The next diagram shows the same simplified version of the cluster, after the source node holding all the primary copies has gone down and the manager has finished directing the remaining peers in fixing the buckets:



The manager directed the cluster in a number of activities to recover from the downed peer:

1. The manager reassigned primacy from bucket copies on the downed peer to searchable copies on the remaining peers. When this step finished, it rolled the generation ID.

2. It directed the peers in making a set of non-searchable copies searchable, to make up for the missing set of searchable copies.

3. It directed the replication of a new set of non-searchable copies (1D, 2D, etc.), spread among the remaining peers.

Even though the source node went down, the cluster was able to fully recover both its complete and valid states, with replication factor number of total buckets, search factor number of searchable bucket copies, and exactly one primary copy of each bucket. This represents a different generation from the previous diagram, because primary copies have moved to different peers.

Bucket-fixing in multisite clusters

The processes that multisite clusters use to handle node failure have some significant differences from single-site clusters. See Multisite clusters and node failure.

View bucket-fixing status

You can view the status of bucket fixing from the manager node dashboard. See View the manager node dashboard.

What happens when a peer node comes back up

A peer node can go down either intentionally (through the CLI `offline` command) or unintentionally (for example, by a server crashing). When the peer goes down, the cluster undertakes remedial activities, also known as **bucket fixing**, as described in the topic, What happens when a peer node goes down. The topic you're now reading describes what happens when and if the peer later returns to the cluster.

When a peer comes back up, it starts sending heartbeats to the manager. The manager recognizes it and adds it back into the cluster. If the peer still has intact bucket copies from its earlier activities in the cluster, the manager adds those copies to the counts it maintains of buckets. The manager also rebalances the cluster, which can result in searchable bucket copies on the peer, if any, being assigned **primary** status. For information on rebalancing, see Rebalance the indexer cluster primary buckets.

Note: When the peer connects with the manager, it checks to see whether it already has the current version of the **configuration bundle**. If the bundle has changed since it went down, the peer downloads the latest configuration bundle, validates it locally, and restarts. The peer rejoins the cluster only if bundle validation succeeds.

How the manager counts buckets

To understand what happens when a peer returns to the cluster, you must first understand how the manager tracks bucket copies.

The manager maintains counts for each bucket in the cluster. For each bucket, it knows:

- how many copies of the bucket exist on the cluster.
- how many **searchable** copies of the bucket exist on the cluster.

The manager also ensures that there's always exactly one **primary** copy of a given bucket.

With multisite clusters, the manager keeps track of copies and searchable copies for each site, as well as for the cluster as a whole. It also ensures that each site with an explicit search factor has exactly one primary copy of each bucket.

These counts allow the manager to determine whether the cluster is **valid** and **complete**. For a single-site cluster, this means that the cluster has:

- Exactly one primary copy of each bucket.
- A full set of searchable copies for each bucket, matching the search factor.
- A full set of copies (searchable and non-searchable) for each bucket, matching the replication factor.

For a multisite cluster, a valid and complete cluster has:

- Exactly one primary copy of each bucket for each site with an explicit search factor.
- A full set of searchable copies for each bucket, matching the search factor for each site as well as for the cluster as a whole.
- A full set of copies (searchable and non-searchable) for each bucket, matching the replication factor for each site as well as for the cluster as a whole.

Bucket-fixing and the copies on the peer

When a peer goes down, the manager directs the remaining peers in bucket-fixing activities. Eventually, if the bucket fixing is successful, the cluster returns to a complete state.

If the peer later returns to the cluster, the manager adds its bucket copies to its counts (assuming that the copies were not destroyed by whatever problem caused the peer to go down in the first place). The consequences vary somewhat depending on whether bucket-fixing activity has completed by the time the peer comes back up.

If bucket-fixing is finished

If bucket-fixing has already completed and the cluster is in a complete state, the copies from the returned peer are just extras. For example, assume the replication factor is 3 and the cluster has fixed all the buckets so that there are again three copies of each bucket in the cluster, including the ones that the downed peer was maintaining before it went down. When the downed peer then comes back up with its copies intact, the manager just adds those copies to the count, so that instead of three copies, there will be four copies of some buckets. Similarly, there could be an excess of searchable bucket copies if the returned peer was maintaining some searchable bucket copies. These excess copies might come in handy later, if another peer maintaining copies of some of those buckets goes down.

If bucket-fixing is still underway

If the cluster is still replacing the copies that were lost when the peer went down, the return of the peer can curtail the bucket-fixing. Once the manager has added the copies on the returned peer to its counts, it knows that the cluster is complete and valid, and so it will no longer direct the other peers to make copies of those buckets. However, any peers that are currently in the middle of some bucket-fixing activity, such as copying buckets or making copies searchable, will complete their work on those copies. Since bucket-fixing is time-intensive, it is worthwhile to bring a downed peer back online as soon as possible, particularly if the peer was maintaining a large number of bucket copies.

Remove excess bucket copies

If the returning peer results in extra copies of some buckets, you can save disk space by removing the extra copies. See Remove excess bucket copies from the indexer cluster.

What happens when the manager node goes down

The manager is essential to the proper running of an indexer cluster, with its role as coordinator for much of the cluster activity. However, if the manager goes down, the peers and search head have default behaviors that allow them to function fairly normally, at least for a while. Nevertheless, you should treat a downed manager as a serious failure.

To deal with the possibility of a downed manager, you can configure a stand-by manager that can take over if needed. For details, see "Replace the manager node on the indexer cluster".

When a manager goes down

If a manager goes down, the cluster can continue to run as usual, as long as there are no other failures. Peers can continue to ingest data, stream copies to other peers, replicate buckets, and respond to search requests from the search head.

When a peer rolls a hot bucket, it normally contacts the manager to get a list of target peers to stream its next hot bucket to. However, if a peer rolls a hot bucket while the manager is down, it will just start streaming its next hot bucket to the same set of peers that it used as targets for the previous hot bucket.

Eventually, problems will begin to arise. For example, if a peer goes down and the manager is still down, there will be no way to coordinate the necessary remedial **bucket-fixing** activity. Or if, for some reason, a peer is unable to connect with one of its target peers, it has no way of getting another target.

The search head can also continue to function without a manager, although eventually the searches will be accessing incomplete sets of data. (For example, if a peer with primary bucket copies goes down, there's no way to transfer primacy to copies on other peers, so those buckets will no longer get searched.) The search head will use the last generation ID that it got before the manager went down. It will display a warning if one or more peers in the last generation are down.

When the manager comes back up

Peers continue to send heartbeats indefinitely, so that, when the manager comes back up, they will be able to detect it and reconnect.

When the manager comes back up, it waits for a quiet period, so that all peers have an opportunity to re-register with it. Once the quiet period ends, the manager has a complete view of the state of the cluster, including the state of peer nodes and buckets. Assuming that at least the replication factor number of peers have registered with it, the manager initiates any necessary bucket-fixing activities to ensure that the cluster is **valid** and **complete**. In addition, it rebalances the cluster and updates the generation ID as needed.

Bucket fixing can take some time to complete, because it involves copying buckets and making non-searchable copies searchable. For help estimating the time needed to complete the bucket-fixing activity, look here.

After the quiet period is over, you can view the manager dashboard for accurate information on the status of the cluster.

Note: You must make sure that at least replication factor number of peers are running when you restart the manager.

Implement SmartStore to reduce local storage requirements

About SmartStore

SmartStore is an indexer capability that provides a way to use remote object stores, such as Amazon S3, Google GCS, or Microsoft Azure Blob storage, to store indexed data.

As a deployment's data volume increases, demand for storage typically outpaces demand for compute resources. SmartStore allows you to manage your indexer storage and compute resources in a cost-effective manner by scaling those resources separately.

SmartStore introduces a remote storage tier and a **cache manager**. These features allow data to reside either locally on indexers or on the remote storage tier. Data movement between the indexer and the remote storage tier is managed by the cache manager, which resides on the indexer.

With SmartStore, you can reduce the indexer storage footprint to a minimum and choose I/O optimized compute resources. Most data resides on remote storage, while the indexer maintains a local cache that contains a minimal amount of data: hot buckets, copies of warm buckets participating in active or recent searches, and bucket metadata.

You can enable SmartStore for all indexes or for a subset of indexes.

SmartStore advantages

SmartStore offers several advantages to the deployment's indexing tier:

- Reduced storage cost. Your deployment can take advantage of the economy of remote object stores, instead of relying on costly local storage.
- Access to high availability and data resiliency features available through remote object stores.
- The ability to scale compute and storage resources separately, thus ensuring that you use resources efficiently.
- Simple and flexible configuration with per-index settings.
- A bootstrapping capability that allows a new cluster or standalone indexer to inherit data from an old cluster or standalone indexer.

SmartStore offers additional advantages specific to deployments of indexer clusters:

- Fast recovery from peer failure and fast data rebalancing, requiring only metadata fixups for warm data.
- Lower overall storage requirements, as the system maintains only a single permanent copy of each warm bucket.
- Full recovery of warm buckets even when the number of peer nodes that goes down is greater than or equal to the replication factor.
- Global size-based data retention.
- Simplified upgrades.

An intelligent cache manager ensures that, for most search use cases, SmartStore provides similar performance to local storage configurations.

Choosing SmartStore

While SmartStore-enabled indexes can significantly decrease storage and management costs under the right circumstances, there are also times when you might find it preferable to continue to rely on local storage.

When to consider moving to SmartStore

SmartStore can help you to achieve significant costs savings for medium to large scale deployments. In particular, consider enabling SmartStore under these circumstances:

- As the amount of data in local storage continues to grow. While local storage costs might not be a significant issue for a small deployment, you should reconsider your use of local storage as your deployment scales over time.
- If you are using indexer clusters to take advantage of features such as data recovery and disaster recovery. Through SmartStore, you can achieve these aims through the native capabilities of the remote store, without the need to store large amounts of redundant data on local storage.
- If you are using indexer clusters and you find that considerable amounts of your time and your compute resources are devoted to managing the cluster. Through SmartStore, you can eliminate much of the cluster management overhead. In particular, you can greatly reduce the scale of time-consuming activities such as offlining peer nodes, data rebalancing, and bucket fixup, because most of the data no longer resides on the peer nodes.
- When most searches are over recent data.

When not to move to SmartStore

There are a few situations where local storage might be a better choice:

- If you have a small deployment, with limited amounts of stored data, the advantages of SmartStore might not compensate for the costs of setting up and maintaining a remote store.
- If you have frequent need to run rare searches, SmartStore might not be appropriate for your purposes, as rare searches can require the indexer to copy large amounts of data from remote to local storage, causing a performance impact. This is particularly the case with searches that cover long timespans. If, however, the searches are across recent data and thus the necessary buckets are already in the cache, then there is no performance impact.
- If you run frequent long lookback searches, you might need to increase your cache size or continue to rely on local storage.

Features not supported by SmartStore

The following capabilities are not available for SmartStore-enabled indexes. Their corresponding settings must use their default values.

- Tsidx reduction. Do not set `enableTsidxReduction` to "true". Tsidx reduction modifies bucket contents and is not supported by SmartStore. **Note:** You can still search any existing buckets that were tsidx-reduced before migration to SmartStore. As with non-SmartStore deployments such searches will likely run slowly. See Reduce tsidx disk usage.
- Data integrity control feature. SmartStore-enabled indexes are not compatible with the data integrity control feature, described in Manage data integrity in the *Securing Splunk Enterprise* manual.
- Disabling Bloom filters. Do not set `createBloomfilter` to "false". Bloom filters play an important role in SmartStore by helping to reduce downloads of tsidx files from remote storage.
- Changing the location of Bloom filters. Do not change `bloomHomePath`. Bloom filters must remain in their default locations inside their bucket directories.
- Summary replication. Summary replication is unnecessary with SmartStore, because summaries are uploaded to remote storage after creation and are accessible to all peers in the cluster.
- Hadoop data roll.
- Certain other `indexes.conf` settings are incompatible with SmartStore. See Settings in indexes.conf that are incompatible with SmartStore or otherwise restricted

Current restrictions on SmartStore use

At this time, SmartStore support requires that your indexing tier conform to certain restrictions:

- With indexer clusters, replication factor and search factor must be equal (for example, 3/3 or 2/2).
- The home path and cold path of each index must point to the same partition.
- Certain other `indexes.conf` settings are restricted with SmartStore. See Settings in indexes.conf that are incompatible with SmartStore or otherwise restricted
- A SmartStore-enabled index cannot be converted to non-SmartStore.
- For multisite clusters, if any SmartStore indexes use report acceleration or data model acceleration, you must disable search affinity by setting all search heads to site0. See Implement search affinity in a multisite indexer cluster.

SmartStore and Splunk Enterprise Security

SmartStore is compatible with Splunk Enterprise Security versions 5.3.0 and later.

For SmartStore use with Splunk Enterprise Security, confirm that you have enough local storage available to accommodate 90 days of indexed data, instead of the 30 days otherwise recommended. See Local storage requirements.

SmartStore architecture overview

The architectural goal of the SmartStore feature is to minimize the amount of data on local storage, while maintaining the fast indexing and search capabilities characteristic of Splunk Enterprise deployments. Except in a few uncommon scenarios, indexers return search results for SmartStore-enabled indexes with speeds similar to those for non-SmartStore indexes.

SmartStore in a nutshell

A SmartStore-enabled index minimizes its use of local storage, with the bulk of its data residing on remote object stores such as S3, GCS, or Azure Blob storage.

Indexing and searching of data occurs on the indexer, just as in a traditional deployment that stores all data locally.

The key difference with SmartStore is that the remote object store becomes the location for master copies of warm buckets, while the indexer's local storage is used to cache copies of warm buckets currently participating in a search or that have a high likelihood of participating in a future search.

A cache manager on the indexer fetches copies of warm buckets from the remote store and places them in the indexer's local cache when the buckets are needed for a search. The cache manager also evicts warm bucket copies from the cache once their likelihood of being searched again diminishes.

Buckets and SmartStore

With SmartStore indexes, as with non-SmartStore indexes, hot buckets are built in the indexer's local storage cache. However, with SmartStore indexes, when a bucket rolls from hot to warm, a copy of the bucket is uploaded to remote storage. The remote copy then becomes the master copy of the bucket.

Eventually, the cache manager evicts the local bucket copy from the cache. When the indexer needs to search a warm bucket for which it doesn't have a local copy, the cache manager fetches a copy from remote storage and places it in the

local cache.

The remote storage has a copy of every warm bucket.

Each indexer's local cache contains several types of data:

- Hot buckets. Hot buckets are created in local storage. They continue to reside solely on the indexer until they roll to warm.
- Copies of warm buckets that are currently participating in searches.
- Copies of recently created or recently searched warm buckets. The indexer maintains a cache of warm buckets, to minimize the need to fetch the same buckets from remote storage repeatedly.
- Metadata for remote buckets. The indexer maintains a small amount of information about each bucket in remote storage.

The buckets of SmartStore indexes ordinarily have just two active states: hot and warm. The cold state, which is used with non-SmartStore indexes to distinguish older data eligible for moving to cheap storage, is not necessary with SmartStore because warm buckets already reside on inexpensive remote storage. Buckets roll directly from warm to frozen.

Cold buckets can, in fact, exist in a SmartStore-enabled index, but only under limited circumstances. Specifically, if you migrate an index from non-SmartStore to SmartStore, any migrated cold buckets use the existing cold path as their cache location, post-migration. In all respects, cold buckets are functionally equivalent to warm buckets. The cache manager manages the migrated cold buckets in the same way that it manages warm buckets. The only difference is that the cold buckets will be fetched into the cold path location, rather than the home path location.

The cache manager

The indexer's cache manager manages the local cache. It fetches copies of warm buckets from remote storage when the buckets are needed for a search. It also evicts buckets from the cache, based on factors such as the bucket's search frequency, its data recency, and various other, configurable criteria.

Certain characteristics that are common to the great majority of Splunk platform searches drive the cache manager's strategy for optimizing bucket location. Specifically, most searches have these characteristics:

- They occur over near-term data. 97% of searches look back 24 hours or less.
- They have spatial and temporal locality. If a search finds an event at a specific time or in a specific log, it's likely that other searches will look for events within a closely similar time range or in that log.

The cache manager therefore favors recently created buckets and recently accessed buckets, ensuring that most of the data that you are likely to search is available in local cache. Those buckets that are likely to participate in searches only infrequently still exist on remote storage and can be fetched to local storage as needed.

Indexer clusters

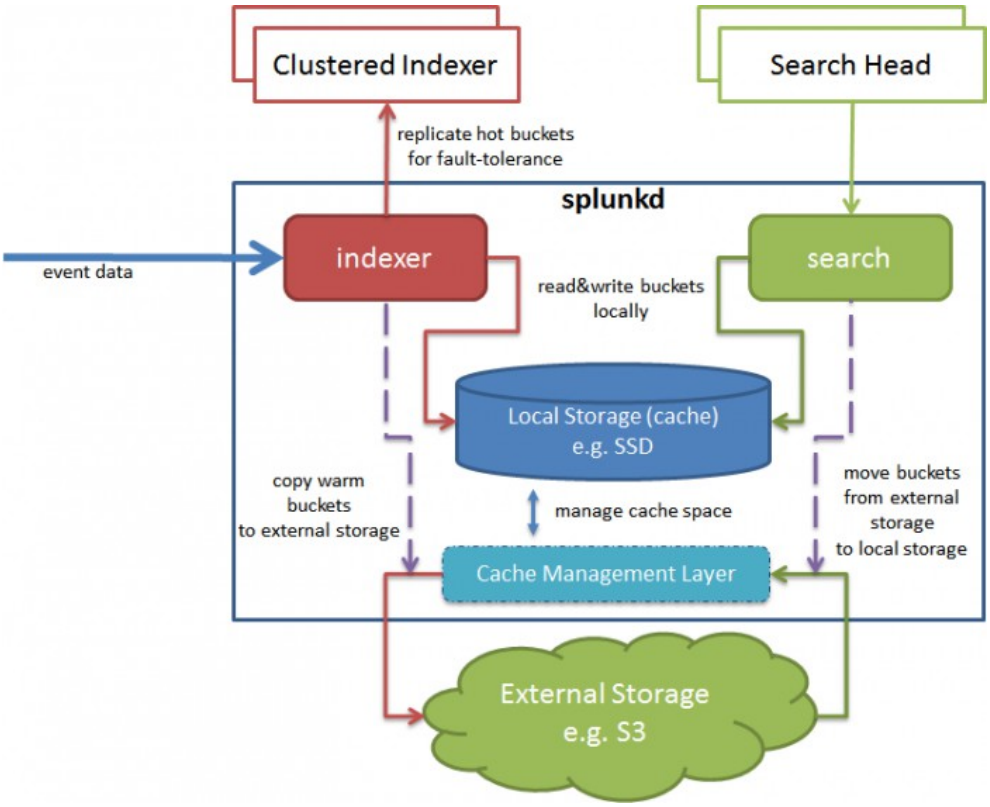
With SmartStore indexes, indexer clusters maintain replication and search factor copies of hot buckets only. The remote storage is responsible for ensuring the high availability, data fidelity, and disaster recovery of the warm buckets.

Because the remote storage handles warm bucket high availability, peer nodes replicate only warm bucket metadata, not the buckets themselves. This means that any necessary bucket fixup for SmartStore indexes proceeds much more quickly than it does for non-SmartStore indexes.

If a group of peer nodes equaling or exceeding the replication factor goes down, the cluster does not lose any of its SmartStore warm data because copies of all warm buckets reside on the remote store.

SmartStore data flow

This diagram encapsulates the flow of data for a SmartStore-enabled index in an indexer cluster. Although it contains specific references to indexer clusters, the essential architecture is the same for non-clustered indexers.



Data streams to a source indexer, which indexes it and saves it locally in a hot bucket. The indexer also replicates the hot bucket data to target indexers. So far, the data flow is identical to the data flow for non-SmartStore indexes.

When the hot bucket rolls to warm, the data flow diverges, however. The source indexer copies the warm bucket to the remote object store, while leaving the existing copy in its cache, since searches tend to run across recently indexed data. The target indexers, however, delete their copies, because the remote store ensures high availability without the need to maintain multiple local copies.The master copy of the bucket now resides on the remote store.

The cache manager on the indexer is central to the SmartStore data flow. It fetches copies of buckets from the remote store, as necessary, to handle search requests. It also evicts older or less searched copies of buckets from the cache, as the likelihood of their participating in searches decreases over time. The job of the cache manager is to optimize use of the available cache, while ensuring that searches have immediate access to the buckets they need.

For further information

The chapter How SmartStore works offers a deeper understanding of the SmartStore architecture. It includes these topics:

- The SmartStore cache manager
- How indexing works in SmartStore
- How search works in SmartStore
- Indexer cluster operations and SmartStore

Deploy SmartStore

SmartStore system requirements

The requirements for SmartStore-enabled indexers are basically the same as for non-SmartStore-enabled indexers. The main differences are:

- Local storage requirements
- The need to connect to remote storage

Indexer requirements

Indexer hardware requirements

Hardware requirements, with the exception of local storage (see below), are the same as for any Splunk Enterprise indexer. See:

- System requirements and other deployment considerations for indexer clusters in this manual
- Reference hardware in *Capacity Planning Manual*

SmartStore operations generally have no significant impact on CPU performance.

Indexer hosting options

The type of object storage used for SmartStore determines where the indexers must be hosted:

- Indexers running SmartStore with an AWS S3 remote store must be hosted on AWS.
- Indexers running SmartStore with a GCP GCS remote store must be hosted on GCP.
- Indexers running SmartStore with an Azure Blob remote store must be hosted on Azure.
- Indexers running SmartStore with S3 API-compliant on-premises object storage must be hosted in an on-premises data center.

Local storage requirements

Based on your indexer deployment type, provision local storage as follows::

- If you are running Splunk Enterprise indexers on local Linux machines, the preferred local storage type is SSD.
- If you are running Splunk Enterprise indexers on AWS, use AWS with NVMe SSD Instance Storage (for example, AWS i3ens or i3s).
- If you are running Splunk Enterprise indexers on GCP, use N1-high memory machine type (n1-highmem-64 or n1-highmem-32 preferred) with zonal SSD persistent disks.
- If you are running Splunk Enterprise indexers on Azure, use Azure with high memory instance + SSD (E series, like Edv4 and Edsv4-series).

The amount of local storage available on each indexer for cached data must be in proportion to the expected working set. For best results, provision enough local storage to accommodate the equivalent of 30 days' worth of indexed data. For

example, if the indexer is adding approximately 100GB/day of indexed data, the recommended size reserved for cached data is 3000GB. At a minimum, provision enough storage to keep at least 7-10 days of data in cache, as searches typically occur on data indexed within the last 7 - 10 days.

For use with Splunk Enterprise Security, provision enough local storage to accommodate 90 days' worth of indexed data, rather than the otherwise recommended 30 days.

Note: The size of data indexed is typically around 50% of the size of data ingested, due to compression of data during indexing. However, there are a number of other factors that also enter into determining the size ratio of indexed data to ingested data. For a general discussion of Splunk Enterprise data volume and how to estimate your storage needs, refer to "Estimating your storage requirements" in the *Installation Manual*.

For indexer clusters, other factors that enter into determining storage requirements include:

- The distribution of ingested data across the indexer cluster. Strive for balanced data ingestion across the indexers, so that all the indexers are indexing, and storing, approximately the same amount of data.
- The number of hot buckets on the indexers. Hot buckets follow replication and search factor policies, so they have, on a per bucket basis, larger storage requirements than warm buckets of the same size.

For purposes of configuring size, the cache is identical with the partition that the cache resides on. Typically, that partition also includes many other files, such as splunk binaries, the operating system, search artifacts, non-SmartStore indexes (if any), and so on. The index data itself resides in $SPLUNK_DB. Therfore, you must consider such files when provisioning storage and configuring cache size. For more information on cache sizing, see "Initiate eviction based on occupancy of the cache's disk partition".

Operating system requirements for on-premise deployments

If you are running Splunk Enterprise in your data center, each on-premise machine must run the same Linux 64-bit operating system.

No other operating systems are supported for on-premise deployments of SmartStore.

Splunk Enterprise version requirements

Standalone indexers must be running Splunk Enterprise version 7.2 or later.

In the case of indexer clusters, all nodes (manager, peer, and search head) in the cluster must be running Splunk Enterprise version 7.2 or later. Additional version compatibility requirements apply to nodes in any indexer cluster, as described in Splunk Enterprise version compatibility.

Other indexer cluster requirements

All indexer cluster requirements apply to SmartStore indexes. For example:

- All index-related settings, including SmartStore-settings, must be configured identically across the peer nodes.
- You must enable SmartStore for the same set of indexes on all peer nodes.
- When you add a new index stanza, you must set `repFactor` to "auto".

For indexer cluster requirements in general, see System requirements and other deployment considerations for indexer clusters.

Network requirements

The indexers are clients of the remote store and use the standard `https` port to communicate with it.

For optimal performance, use 10Gbps network connectivity from each indexer to the remote store.

Remote store requirements

SmartStore can run on either an AWS S3 remote store (including S3 API-compliant on-premise object stores), a GCP GCS remote store, or an Azure Blob remote store. Depending on which remote store type you plan to deploy on, see Configure the S3 remote store for SmartStore, Configure the GCS remote store for SmartStore, or Configure the Azure Blob remote store for SmartStore

Multiple indexer clusters or standalone indexers cannot access the same remote volume. In other words, the value of any path setting in indexes.conf must be unique to a single running indexer or cluster. Do not share path settings among multiple indexers or clusters.

Configure the S3 remote store for SmartStore

Before you configure SmartStore settings on the indexers, you must ensure that your remote store is properly set up, so that it is available to the indexers.

Later, when you configure remote volumes for SmartStore, you configure settings specific to the remote store in `indexes.conf`. The indexer uses those settings to communicate with the remote store.

Supported remote storage services

Supported remote storage services include S3, Google GCS, and Microsoft Azure Blob storage. For information on GCS, see Configure the GCS remote store for SmartStore. For information on Azure Blob storage, see Configure the Azure Blob remote store for SmartStore

For S3, SmartStore can use:

- AWS S3
- S3-API-compliant object stores

To determine whether your object store is S3-compliant, use the S3 compatibility checking tool, located here: https://github.com/splunk/s3-tests. To use the tool, follow the instructions in the repository's README file.

Configure an S3 remote store

When configuring S3 buckets:

- The buckets must have read, write, and delete permissions.
- If the indexers are running on EC2, provision the buckets for the same region as the EC2 instances that use it.
- The S3 buckets must be used only by SmartStore. Do not share S3 buckets with other tools such as ingest actions and edge processors.

See the Amazon S3 documentation for information on on how to create and configure buckets.

For S3-specific settings available through Splunk Enterprise, search for settings in the indexes.conf spec file that start with `remote.s3`.

For information on security-related settings, such as settings for S3 authentication and encryption, see SmartStore on S3 security strategies.

Remote store versioning support

Remote store versioning support is optional and turned on for SmartStore by default. Some third party remote stores do not support versioning and so the SmartStore capability must be turned off. Check with your third party vendor and turn off versioning if necessary.

To turn off versioning support, use the `remote.s3.supports_versioning` setting in `indexes.conf`.

Accommodate the remote store addressing model

Amazon S3 currently supports two addressing models, or request URI styles: path style, or V1, and virtual-hosted style, or V2. In V1, the bucket name is in the URI path; for example, `//s3.amazonaws.com/<bucketname>/key`. In V2, the bucket name is part of the domain name; for example, `//<bucketname>.s3.amazonaws.com/key`.

Amazon is deprecating support for V1 and, going forward, will require requests for new S3 buckets to use the V2 model.

Splunk Enterprise remote store addressing for native S3

Splunk Enterprise accommodates both V1 and V2 models automatically for Amazon S3 buckets.You can use either model, but Splunk Enterprise will convert V1 URIs to V2 when communicating with S3.

Using the V1 model, you specify the URI like this:

```
[volume:s3volume]
storageType = remote
path = s3://<bucketname>/rest/of/path
```

Using the V2 model, you specify the URI like this:

```
[volume:s3volume]
storageType = remote
path = s3://rest/of/path
remote.s3.bucket_name = <bucketname>
```

Similarly, if you specify an endpoint that ends in `amazonaws.com`, Splunk Enterprise determines the URI version from the endpoint, since the structure is fixed. For example:

```
[volume:s3volume]
storageType = remote
path = s3://<bucketname>/rest/of/path
remote.s3.endpoint = https://s3.us-west-1.amazonaws.com
```

or

```
[volume:s3volume]
```

```
storageType = remote
path = s3://rest/of/path
remote.s3.endpoint = https://<bucketname>.s3.us-west-1.amazonaws.com
```
Both of these specify the same bucket, and Splunk Enterprise will correctly resolve either one.

Splunk Enterprise remote store addressing for S3-compatible remote stores

If you use an S3-compatible remote store, rather than native S3, you might need to specify the addressing model that the S3-compatible store supports. You need to specify the model only if the S3-compatible remote store does not support V1.

Splunk Enterprise provides the `remote.s3.url_version` setting to specify the model that you are using. Its default value is v1. To change the addressing model to V2, change the setting to v2. For example:

```
[volume:s3volume]
storageType = remote
path = s3://rest/of/path
remote.s3.url_version = v2
remote.s3.endpoint = https://bucketname.whatever.customer.com
```

Configure the GCS remote store for SmartStore

Before you configure SmartStore settings on the indexers, you must ensure that your remote store is properly set up, so that it is available to the indexers.

Later, when you configure remote volumes for SmartStore, you configure settings specific to the remote store in `indexes.conf`. The indexer uses those settings to communicate with the remote store.

Supported remote storage services

Supported remote storage services include GCS, AWS S3, and Microsoft Azure Blob storage. For information on S3, see Configure the S3 remote store for SmartStore. For information on Azure Blob storage, see Configure the Azure Blob remote store for SmartStore

Configure a GCS remote store

When configuring GCS buckets:

- The indexers' Compute Engine service account must have read, write and delete permissions for the GCS buckets that the indexers are associated with..
- Provision the buckets to run in the same GCP region as the indexer Compute Engine instances.

See the Google GCS documentation for information on on how to create and configure buckets.

For GCS-specific settings available through Splunk Enterprise, search for settings in the indexes.conf spec file that start with `remote.gs`.

For information on security-related settings, such as settings for GCS authentication and encryption, see SmartStore on GCS security strategies.

Configure the Azure Blob remote store for SmartStore

Before you configure SmartStore settings on the indexers, you must ensure that your remote store is properly set up, so that it is available to the indexers.

Later, when you configure remote volumes for SmartStore, you configure settings specific to the remote store in `indexes.conf`. The indexer uses those settings to communicate with the remote store.

Supported remote storage services

Supported remote storage services include Azure Blob storage, Google GCS and AWS S3,. For information on S3, see Configure the S3 remote store for SmartStore. For information on GCS, see Configure the GCS remote store for SmartStore.

Configure an Azure Blob remote store

When configuring Azure Blob storage containers:

- The indexers' Azure service account must have read, write and delete permissions for the container that the indexers are associated with.
- The container must run in the same Azure region as the indexer instances.

See the Microsoft Azure documentation for information on on how to create and configure containers.

For Azure-specific settings available through Splunk Enterprise, search for settings in the indexes.conf spec file that start with `remote.azure`.

For information on security-related settings, such as settings for Azure Blob storage authentication and encryption, see SmartStore on Azure Blob security strategies.

Choose the storage location for each index

You can configure SmartStore globally, using the same settings for all indexes, or you can configure SmartStore on a per-index basis. If you configure SmartStore on a per-index basis, you can have a mix of SmartStore and non-SmartStore indexes on the same indexer or indexer cluster. You can also specify different remote volumes for different SmartStore indexes.

The remote volume, defined in indexes.conf, points to the remote store where SmartStore stores warm buckets, such as a location on an S3 bucket, a GCS bucket, or an Azure Blob storage container.

To summarize, you can choose from these storage options:

- All indexes stored remotely, on a single volume.
- All indexes stored remotely, on multiple volumes.
- Some indexes stored locally, with others stored remotely on one or more remote volumes.

Even with a SmartStore index, some index data is temporarily stored locally, in the cache. However, except for hot

buckets, the index's master copies of buckets are stored remotely. For the sake of simplicity, the list of storage options assumes the index's master copies when discussing storage volumes and does not consider data stored locally in the SmartStore cache.

Keep these limitations in mind:

- Each remote volume is limited to a single indexer cluster or standalone indexer. That is, each remote store holds buckets for only a single cluster or standalone indexer. From the configuration standpoint, the `path` setting within each remote volume stanza in `indexes.conf` must be unique to the cluster or indexer. For example, if indexes on one cluster use a particular remote volume, no index on any other cluster or standalone indexer can uses the same remote volume.
- Each SmartStore index is limited to a single remote volume. All warm buckets for that index must reside in the same remote store.
- Each indexer or indexer cluster is limited to a single remote storage type across all indexes. The available remote storage types are S3, GCS, or Azure Blob storage.
- All peer nodes on an indexer cluster must use the same SmartStore settings.

SmartStore on S3 security strategies

SmartStore security strategies vary depending on the type of remote storage service. This topic covers security when using the Amazon Simple Storage Service (S3) as the remote storage service.

Authenticate with the remote storage service

How you authenticate into the remote storage service depends on the cloud infrastructure that your indexer or indexer cluster uses.

- If the indexer or indexer cluster runs on Amazon Elastic Compute Cloud (EC2), you can use the access and secret keys from its Identity and Access Management (IAM) role to authenticate.
- If the indexer or indexer cluster does not run on EC2, use hardcoded keys in `indexes.conf`. These are the relevant settings for hardcoding keys for S3:
  - `remote.s3.access_key`: The access key to use when authenticating with the remote storage system.
  - `remote.s3.secret_key`: The secret key to use when authenticating with the remote storage system.
  - `remote.s3.endpoint`: The URL of the remote storage system. This setting tells the indexer where to go for S3 authentication. Use the value for the S3 bucket region. For example, `https://s3.us-west-2.amazonaws.com`.

For more information on these settings, see the indexes.conf spec file.

The credentials that you use, whether from the IAM role or from `indexes.conf`, need permission to perform S3 operations. They also need permission to perform Amazon Key Management Service (KMS) operations if you are encrypting data-at-rest on the remote store.

IAM permissions required for SmartStore operations

The following permissions are needed for basic S3 operations:

- `s3:ListBucket`
- `s3:PutObject`
- `s3:GetObject`

- s3:DeleteObject

In addition, if using remote.s3.supports_versioning=true, the following permissions are required for bucket-freezing operations, in order to delete all versions of the files:

- s3:ListBucketVersions
- s3:DeleteObjectVersion

It is usually a better practice to set remote.s3.supports_versioning=false and instead use S3 lifecycle rules.

By default, s3:AbortMultipartUpload permission is already given to the S3 bucket owner and the initiator of the multipart upload, but it can be explicitly added if needed.

Use AssumeRoleWithWebIdentity to authenticate

SmartStore supports AWS AssumeRoleWithWebIdentity used with an identity provider compatible with OpenID Connect.Â This method provides temporary security credentials without including long-term AWS credentials in the application. The temporary security credentials returned by this API consist of an access key ID, a secret access key, and a security token. SmartStore will use these temporary security credentials to sign calls to AWS service API operations.

Once OIDC is setup, you must set the following environment variables:

```
AWS_DEFAULT_REGION -  default region
AWS_WEB_IDENTITY_TOKEN_FILE - path to JWT token file
AWS_REGIONÂ  - AWS region configured
AWS_ROLE_ARN - role that this web token can assume
AWS_STS_REGIONAL_ENDPOINTS - regional
```

For example:

```
AWS_DEFAULT_REGION=us-west-2
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
AWS_REGION=us-west-2
AWS_ROLE_ARN=arn:aws:iam::667741767953:role/oidc-test-role
AWS_STS_REGIONAL_ENDPOINTS=regional
```

Manage SSL certifications for the remote store

The SSL certification settings vary according to the remote storage service type. This section provides information for managing SSL for an S3 remote store, using the settings provided in indexes.conf. For more details on any of these settings, as well as for information on additional S3-related SSL settings, see the indexes.conf spec file.

When you configure SSL settings for a remote volume, you must do so on a per-volume basis. This means you must specify SSL settings for each individual remote volume that you define in the indexes.conf file.

The S3 SSL settings are overlaid on the sslConfig stanza in the server.conf configuration file, with the following exceptions:

- sslVerifyServerCert
- sslAltNameToCheck
- sslCommonNameToCheck

If you run into problems with configuring SSL for S3 storage, consult the server.conf SSL settings in addition to the

remote-storage-specific settings.

The following table includes common settings and their recommended values.

SSL setting	Description	Recommended value
`remote.s3.sslVerifyServerCert`	Specifies whether to check the server certificate provided by the S3 endpoint.	true
`remote.s3.sslVersions`	The SSL version to use.	tls1.2
`remote.s3.sslAltNameToCheck`	List of alternative names in the certificate presented by the server to match against. For example, `s3.<region>.amazonaws.com`.	N/A
`remote.s3.sslRootCAPath`	Absolute path to the Privacy Enhanced Mail (PEM) format file that contains the list of root certificates.	N/A

`remote.s3.cipherSuite`	Ciphers to use to connect with S3.	Check with your security experts. Here is an example of the type of value to enter for this setting:
		`ECDHE-ECDSA-AES128-SHA256:`
		`ECDHE-ECDSA-AES256-SHA384:`
		`ECDHE-ECDSA-AES128-GCM-SHA256:`
		`ECDHE-ECDSA-AES256-GCM-SHA384:`
		`ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA:`
		`ECDHE-ECDSA-AES128-SHA:`
		`ECDHE-ECDSA-AES256-SHA:`
		`AES128-SHA256:AES256-SHA256:`
		`AES128-SHA:AES256-SHA:`
		`DHE-RSA-AES128-SHA:`
		`DHE-RSA-AES256-SHA:`
		`DHE-RSA-AES128-SHA256:`
		`DHE-RSA-AES256-SHA256`

`remote.s3.ecdhCurves`	Elliptic Curve-Diffie Hellman (ECDH) curves to send.	Check with your security experts. Here is an example of the type of value to enter for this attribute: prime256v1, secp384r1, secp521r1

Encrypt the data on the remote store

SmartStore supports client- and server-side encryption of data at rest, or data which is neither in use nor in transit, on S3. SmartStore supports four encryption schemes through the `remote.s3.encryption` setting in the `indexes.conf` configuration file. You must use configuration files to encrypt data on S3 volumes, as there is no other method to perform this kind of configuration..

When you enable encryption, depending on the encryption method you use, the Splunk platform generates an encryption key and encrypts data that you upload to the target volume with this key. After a certain interval, the platform generates a new key.

With most encryption methods, the Splunk platform instance that does the encryption must maintain a connection to AWS or KMS. Failure to maintain this connection can cause problems with generating new keys for encryption.

The high-level procedure to encrypt data on a SmartStore volume on a single Splunk platform instance follows. If you want to encrypt data on a SmartStore volume on an indexer cluster, do not follow this procedure. Instead, see Update common peer configurations and apps. When you perform that procedure, supply the settings that appear in this

procedure.

1. Choose the encryption method that you want to use on a SmartStore volume.

   Choosing the encryption method is a one-time decision. You cannot change the encryption method later.
2. On the Splunk platform instance where you want to encrypt data on a SmartStore volume, open the `$SPLUNK_HOME/etc/system/local/indexes.conf` for editing.
3. Specify the type of encryption method you want to apply to each SmartStore volume by using the `remote.s3.encryption` setting under the `[volume:<volume_name>]` stanza for that volume:

   ```
   [volume:myvolume]
   remote.s3.encryption = sse-s3 | sse-kms | sse-c | cse | none
   ```
4. Depending on the type of encryption you use, specify additional settings that are required to interact with AWS or KMS to do the encryption. See the encryption examples later in this topic for information on the settings to use.
5. Save the `indexes.conf` file and close it,
6. Restart the Splunk platform.

Encryption occurs at the time of data upload to the volumes that you specify, and remains in effect until you change the encryption scheme again. When you configure encryption for the remote volume, you do not cause data that is already on the volume to be encrypted.

If you disable encryption, you do not cause existing encrypted data to be decrypted. Any encrypted data becomes unusable, because the Splunk platform cannot decrypt it.

If you do not already know which encryption scheme you want to use, the best choice for server-side encryption is `sse-c` (server-side encryption with customer keys). This method avoids running into potential throttling issues from KMS. For client-side encryption, `cse` is the best and only choice.

- For detailed information on the settings to use for encryption, see the indexes.conf spec file.
- For information on configuring server-side encryption in AWS, see the Amazon documentation.

Server-side encryption with customer-provided encryption keys (sse-c)

Here is an example of setting server-side encryption with customer keys. All of these settings go into the `indexes.conf` configuration file.

```
[volume:example_volume]
remote.s3.encryption = sse-c
remote.s3.encryption.sse-c.key_type = kms
remote.s3.encryption.sse-c.key_refresh_interval = 86400
# 86400 equals 24 hours. This is the default and recommended value.
# The minimum value is 3600.
# Setting a very low value can degrade performance.
remote.s3.kms.auth_region = <aws_region>
remote.s3.kms.key_id = <kms_keyid>
# The kms_keyid must be a unique key ID, the Amazon Resource Name (ARN)
# of the CMK, or the name or ARN of an alias that points to the CMK.

# SSL settings for KMS communication
remote.s3.kms.sslVerifyServerCert = true
remote.s3.kms.sslVersions = tls1.2
remote.s3.kms.sslAltNameToCheck = kms.<aws_region>.amazonaws.com
remote.s3.kms.sslRootCAPath = $SPLUNK_HOME/etc/auth/kms_rootcert.pem
```

```
remote.s3.kms.cipherSuite =
ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM
-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA:AES128
-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES2
remote.s3.kms.ecdhCurves = prime256v1, secp384r1, secp521r1
```

Server-side encryption with Amazon S3-managed encryption keys (sse-s3)

Here is an example of setting server-side encryption with AES256. All of these settings go into the `indexes.conf` configuration file.

```
[volume:example_volume]
remote.s3.encryption = sse-s3
```

Server-side encryption with customer master keys stored in AWS KMS (sse-kms)

Here is an example of setting server-side encryption with KMS-managed keys. All of these settings go into the `indexes.conf` configuration file.

```
[volume:example_volume]
remote.s3.encryption = sse-kms
remote.s3.kms.key_id = <kms_keyid>
```

Client-side encryption (cse)

Client side encryption of data ensures that the cloud service provider (CSP) cannot read the encrypted data in any way.

SmartStore contacts the AWS KMS service on an interval that you specify. It uses KMS to generate data encryption keys (DEKs) based on the Customer Master Key (CMK) that KMS stores. When you upload data after you enable CSE, SmartStore stores the data encryption keys with the uploaded data buckets.

There is no support for key revocation of any kind in the software. You are responsible for managing the customer master key and any DEKs. Also, you cannot encrypt data that has already been encrypted with a new DEK. You must decrypt the data first, then have the platform generate a new DEK before uploading the data again.

You must use AWS KMS to take advantage of this feature. It does not work with any other type of key management service.

There are some caveats to enabling client-side encryption on an index in SmartStore:

- Performance can degrade up to 20% due to the data encryption.
- There are limitations to some of the remote file system CLI commands that you can run on a SmartStore volume for maintenance or troubleshooting purposes:
  - The `splunk cmd splunkd rfs getF` command only lets you upload the `receipt.json` file, and CSE does not encrypt this file on upload.
  - The `splunk cmd splunkd rfs putF` command only lets you download the `receipt.json` file, and assumes the file has not been encrypted.
  - The `splunk cmd splunkd rfs get` command must include `receipt.json` as an argument, otherwise it fails.

Here is an example of setting client-side encryption. All of these settings go into the `indexes.conf` configuration file.

```
# The volume stanza and path specify the name of the remote volume.
# This is the volume that the index that will store the encrypted data uses.
```

```
# The Splunk platform expects the path to be in the remote storage format.
[volume:<VOLUME_NAME>]
path = <S3_BUCKET_PATH>
storageType = remote

# The following settings facilitate interaction with AWS KMS. You need
# KMS to generate the client-side encryption key. The settings
# set the the KMS endpoint and authentication region,
#
# You must configure one of the following two settings
# If you configure neither setting, the Splunk platform attempts
# to use the 'remote.s3.endpoint' and 'remote.s3.auth_region' settings
# before it fails to start.
remote.s3.kms.endpoint = <KMS_ENDPOINT>
remote.s3.kms.auth_region = <KMS_AUTH_REGION>

# The unique ID or Amazon Resource Name (ARN) of the
# customer master key to use, or the alias name or ARN
# of the alias that refers to this key.
remote.s3.kms.key_id = <KEY_ID>

# The signature version to use when authenticating the remote storage
# system supporting the S3 API. Since CSE uses KMS to manage encryption
# keys, you must set this to "v4".
remote.s3.signature_version = v4

# Enable client-side encryption on the remote volume.
remote.s3.encryption = cse

# The bucket encryption algorithm to use for CSE.
# Currently, "aes-256-gcm" is the only acceptable value.
remote.s3.encryption.cse.algorithm = aes-256-gcm

# How long to wait, in seconds, between generation of
# keys to encrypt data that is uploaded to S3 when CSE
# is active.
remote.s3.encryption.cse.key_refresh_interval = 86400

# The key mechanism to use for CSE. Currently, "kms"
# is the only acceptable value. You must configure this
# setting for the Splunk platform to start.
remote.s3.encryption.cse.key_type = kms


[INDEX_NAME]
# The index you specify for CSE of data. When the Splunk
# platform adds data to this index, it encrypts the data with
# the key that you provide. The index references
# the remote S3 storage volume.
homePath = $SPLUNK_DB/<INDEX_NAME>/db
coldPath = $SPLUNK_DB/<INDEX_NAME>/colddb
thawedPath = $SPLUNK_DB/<INDEX_NAME>/thaweddb
remotePath = volume:<VOLUME_NAME>/$_index_name
```

SmartStore on GCS security strategies

SmartStore security strategies vary according to the type of remote storage service. This topic covers security when using Google Cloud Storage (GCS) as the remote storage service.

Authenticate with the remote storage service

The indexer uses a credential file to authenticate with GCP.

How the indexer obtains its credential

The indexer can obtain its credential in several ways. In descending order of precedence, the indexer uses:

1. The credential file specified by the `remote.gs.credential_file` setting in `indexes.conf`, if set.
2. The credential for the account associated with the `remote.gs.service_account_email` setting in `indexes.conf`, if set,
3. The credential for the Compute Engine's default service account.

Specify a custom credential on the indexer

To specify a custom credential on the indexer, use the `remote.gs.credential_file` setting in `indexes.conf`:

```
remote.gs.credential_file = <credentials.json>
```

The file specified by this setting must located on the indexer as follows:

- For standalone indexers, the file must be located under `$SPLUNK_HOME/etc/auth`.
- For peer nodes in a cluster, you can distribute the file through the configuration bundle method, putting the file under the `$SPLUNK_HOME/etc/manager-apps/_cluster/local` directory on the manager node and distributing the bundle to each of the peer nodes, causing the file to reside under their `$SPLUNK_HOME/etc/peer-apps/_cluster/local` directories. See "Structure of the configuration bundle" for information on the `manager-apps` directory. You can also put the file in `$SPLUNK_HOME/etc/auth` on each peer node. The distributed bundle location has precedence.

The credential file is encrypted on startup.

Credential permissions

The credentials that you use need permission to perform GCS operations. They also need permission to perform GCP Key Management Service (KMS) operations if you are using the gcp-sse-c or gcp-sse-kms encryption schemes.

Manage SSL certifications for the remote store

The SSL certification settings vary according to the remote storage service type. This section provides information for managing SSL for a GCS remote store, using the settings provided in `indexes.conf`. For more details on any of these settings, as well as for information on additional GCS-related SSL settings, see the indexes.conf spec file.

The GCS SSL settings are overlaid on the `sslConfig` stanza in `server.conf` for `caCertFile` (`sslRootCAPath`) and `cipherSuite`.Therefore, if you run into issues, consult the `server.conf` SSL settings, in addition to the remote-storage-specific settings.

Specify SSL settings on a per-remote-volume basis.

The following table includes common attributes and their recommended values.

SSL setting	Description	
`remote.gs.sslVerifyServerCert`	Specifies whether to check the server certificate provided by the GCS endpoint.	true
`remote.gs.sslVerifyServerName`	Specifies whether to verify that either the Common Name or Subject Alternate Name in the server certificate matches the hostname in the URL it connects to.	true
`remote.gs.sslVersionsForClient`	Specifies the minimum SSL/TLS version to use for outgoing connections.	tls1.2
`remote.gs.sslRootCAPath`	Absolute path to the PEM format file containing list of root certificates.	N/A
`remote.gs.cipherSuite`	Ciphers to use to connect with GCS.	Check with your security experts. Here is an example of the type of value to enter for this attribu ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-

Encrypt the data on the remote store

SmartStore supports server-side encryption of data-at-rest on GCS. SmartStore supports three encryption schemes through the `remote.gs.encryption` attribute in `indexes.conf`:

```
remote.gs.encryption = gcp-sse-c | gcp-sse-kms | gcp-sse-gcp
```

The default is "gcp-sse-gcp".

Configure this attribute on a per-volume basis.

The recommended method for encryption on the remote store is "gcp-sse-c" (server-side encryption with customer keys). This method avoids running into potential throttling issues from KMS.

Choosing the encryption method is a one-time decision. You cannot change the encryption method later.

For details on encryption modes, see:

- The indexes.conf spec file for the `remote.gs` encyrption-related settings.
- The Google documentation on configuring Cloud Storage encryption keys.

Encryption with gcp-sse-c

Here is an example of setting server-side encryption with customer-supplied encryption keys:

```
[volume:my_gcs_vol]
storageType = remote
path = gs://your_gcp_test_1
remote.gs.project_id = your-app-test
remote.gs.encryption = gcp-sse-c
remote.gs.encryption.gcp-sse-c.key_refresh_interval = 86400
remote.gs.encryption.gcp-sse-c.key_type = gcp_kms
remote.gs.gcp_kms.locations = us-central1
remote.gs.gcp_kms.key_ring = your_ring_1
remote.gs.gcp_kms.key = test_key_1
```

Encryption with gcp-sse-gcp

Here is an example of setting server-side encryption with Google-managed encryption keys:

```
[volume:my_gcs_vol_3]
storageType = remote
remote.gs.encryption = gcp-sse-gcp
path = gs://your_gcp_test_3
```

Encryption with gcp-sse-kms

When configuring the GCS bucket, you can either specify the key or, by default, use the key associated with the GCS bucket.

Here is an example of setting server-side encryption with customer-managed encryption keys:

```
[volume:my_gcs_vol_2]
storageType = remote
path = gs://your_gcp_test_2
remote.gs.project_id = your-app-test
remote.gs.encryption = gcp-sse-kms
remote.gs.gcp_kms.locations = us-central1
remote.gs.gcp_kms.key_ring = your_ring_us
remote.gs.gcp_kms.key = test_key_3
```

If you want to use the key associated with the GCS bucket service account, leave the key settings empty:

```
[volume:my_gcs_vol_2]
storageType = remote
path = gs://your_gcp_test_2
remote.gs.project_id = your-app-test
remote.gs.encryption = gcp-sse-kms
```

SmartStore on Azure Blob security strategies

SmartStore security strategies vary according to the type of remote storage service. This topic covers security when using Azure Blob storage as the remote storage service.

Authenticate with the remote storage service

The best practice is to authenticate through Azure Active Directory using managed identities. Authorizing access with Azure Active Directory provides superior security and ease of use over other authorization options.

SmartStore on Azure Blob supports the following managed-identity-based authentication methods:

- Authentication through system-assigned managed identity assigned to the Azure Virtual Machine that the indexer instance is running on.
- Authentication through user-assigned managed identity using the registered application's client secret.

The client secret method is configured in each indexer's `indexes.conf` file:

- `remote.azure.tenant_id`: The ID of the tenant. The tenant is an instance of an Azure AD directory. Check your Azure subscription for details.
- `remote.azure.client_id`: The ID of the client. This ID is also known as the application ID, the unique identifier that Azure AD issues to an application registration. The ID identifies a specific application and its associated configurations. You can obtain the client ID for an application from the Azure Portal in the Overview section for the registered application.
- `remote.azure.client_secret`: The secret key to use when authenticating through the client secret method. You generate the secret key through the Azure Portal.

For more information on these settings, see the indexes.conf spec file.

Appropriate access policies and permissions need to be assigned to the security principal to allow access to the Azure container.

Caution: You can also use access/secret keys authentication. This method has security implications, but it might be appropriate for smaller or test deployments, depending on your needs. For more information on this method, see the entries for `remote.azure.access_key`, `remote.azure.secret_key`, and `remote.azure.endpoint` in the indexes.conf spec file.

Manage SSL certifications for the remote store

The SSL certification settings vary according to the remote storage service type. This section provides information for managing SSL for an Azure Blob remote store, using the settings provided in `indexes.conf`. For more details on any of these settings, as well as for information on additional Azure-related SSL settings, see the indexes.conf spec file.

The Azure SSL settings are overlaid on the `sslConfig` stanza in `server.conf` for `caCertFile` (`sslRootCAPath`) and `cipherSuite`.Therefore, if you run into issues, consult the `server.conf` SSL settings, in addition to the remote-storage-specific settings.

Specify SSL settings on a per-remote-volume basis.

The following table includes common attributes and their recommended values.

SSL setting	Description	Recommended value
`remote.azure.sslVerifyServerCert`	Specifies whether to check the server certificate provided by the Azure endpoint.	true
`remote.azure.sslVerifyServerName`	Specifies whether to verify that either the Common Name or Subject Alternative Name in the server certificate matches the hostname in the URL it connects to.	true
`remote.azure.sslVersions`	Specifies the minimum SSL/TLS version to use for outgoing connections.	tls1.2
`remote.azure.sslRootCAPath`	Absolute path to the PEM format file containing list of root certificates.	N/A
`remote.azure.cipherSuite`	Ciphers to use to connect with Azure Blob storage.	Check with your security experts. Here is an example of the type of value to enter for this attrib ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE

Encrypt the data on the remote store

SmartStore supports server-side encryption of data-at-rest on Azure Blob storage. SmartStore supports two encryption schemes through the `remote.azure.encryption` attribute in `indexes.conf`:

```
remote.azure.encryption = azure-sse-kv | azure-sse-ms
```

Configure this attribute on a per-volume basis.

Choosing the encryption method is a one-time decision. You cannot change the encryption method later.

These encryption schemes let you choose between Microsoft-managed keys or customer-managed keys fully configured on the Azure Blob storage account side (transparent from Splunk Enterprise point-of-view)

The default is "azure-sse-ms". (Microsoft-managed keys).

For details on encryption modes, see:

- The indexes.conf spec file for the `remote.azure` encyrption-related settings.
- The Azure documentation on configuring Azure Blob storage encryption keys.

Encryption with azure-sse-ms

Here is an example of setting server-side encryption with Microsoft-managed encryption keys:

```
[volume:azure_storage_ms_keys]
storageType = remote
path = azure://my/msencrypted/storage
remote.azure.encryption = azure-sse-ms
```

Encryption with azure-sse-kv

Customer-managed keys require an encryption scope. To create an encryption scope, use either keys kept in a key vault or the Microsoft-managed keys.

> Encryption scope must be already configured in the storage account before you can enable azure-sse-kv.

Here is an example of setting server-side encryption with customer-supplied encryption keys:

```
[volume:azure_storage_kv_keys]
storageType = remote
path = azure://my/cmencrypted/storage
remote.azure.encryption = azure-sse-kv
remote.azure.azure-sse-kv.encryptionScope = customer_key_scope_1
```

Deploy SmartStore on a new indexer cluster

During this procedure, you:

1. Install a new indexer cluster.
2. Configure the cluster peer nodes to access the remote store.
3. Test the deployment.

Note: This procedure configures SmartStore for a new indexer cluster only. It does not describe how to load existing data onto an indexer cluster. To migrate existing local indexes to SmartStore, see Migrate existing data on an indexer cluster to SmartStore. To bootstrap existing SmartStore data onto a new indexer cluster, see Bootstrap SmartStore indexes.

To deploy SmartStore on a standalone indexer, see Deploy SmartStore on a new standalone indexer.

Prerequisites

Read:

- Indexer cluster deployment overview
- Update common peer configurations and apps
- SmartStore system requirements
- Choose the storage location for each index
- Documentation provided by the vendor of the remote storage service that you are using

Cautions

Be aware of these configuration issues:

- The value of the `path` setting for each remote volume stanza must be unique to the indexer cluster. You can share remote volumes only among indexes within a single cluster. In other words, if indexes on one cluster use a particular remote volume, no index on any other cluster or standalone indexer can use the same remote volume.
- You must set all SmartStore indexes in an indexer cluster to use `repFactor = auto`.
- Leave `maxDataSize` at its default value of "auto" (750MB) for each SmartStore index.
- The `coldPath` setting for each SmartStore index requires a value, even though the setting is ignored except in the case of migrated indexes.
- The `thawedPath` setting for each SmartStore index requires a value, even though the setting has no practical purpose because you cannot thaw data to a SmartStore index. See Thawing data and SmartStore.

Deploy SmartStore

The following procedure assumes that you are deploying SmartStore on a new indexer cluster. It also assumes that you are deploying SmartStore for all indexes on the cluster, using a single remote location. If you want to deploy SmartStore for some indexes only, or if you want to use multiple remote locations for the SmartStore indexes, you can modify the procedure to fit your need.

1. Ensure that you have met the prerequisites. In particular, read:
    - SmartStore system requirements
    - The appropriate topic for configuring your remote storage type:
        - Configure the S3 remote store for SmartStore
        - Configure the GCS remote store for SmartStore
        - Configure the Azure Blob remote store for SmartStore
2. Understand SmartStore security strategies and prepare to implement them as necessary during the deployment process. See the topic on security strategies for your remote storage type:
    - SmartStore on S3 security strategies
    - SmartStore on GCS security strategies
    - SmartStore on Azure Blob security strategies
3. Install a new Splunk Enterprise instance and enable it as the manager node for a new indexer cluster. See Enable the indexer cluster manager node.
4. Set the indexer cluster's replication factor and search factor to equal values, for example, 3/3.
5. On the manager node, create or edit `$SPLUNK_HOME/etc/manager-apps/_cluster/local/indexes.conf` and specify the SmartStore settings, as shown in the examples below. When the peer nodes later start up, the manager automatically distributes these settings, along with the rest of the configuration bundle, to the peer nodes. See "Structure of the configuration bundle" for information on the `manager-apps` directory.

    Using an S3 remote object store:
    This example configures SmartStore indexes, using an S3 remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

repFactor = auto

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = s3://mybucket/some/path

# The following S3 settings are required only if you're using the access and secret
# keys. They are not needed if you are using AWS IAM roles.

remote.s3.access_key = <S3 access key>
remote.s3.secret_key = <S3 secret key>
remote.s3.endpoint = https:|http://<S3 host>

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

Using a GCS remote object store:

This example configures SmartStore indexes, using a GCS remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

repFactor = auto

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = gs://mybucket/some/path
```

```
# There are several ways to specify credentials. For details, see the topic,
# "SmartStore on GCS security strategies." One way to specify credentials
# is to point to a file, as shown here.
remote.gs.credential_file = credential.json

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

Using an Azure Blob remote object store:

This example configures SmartStore indexes, using an Azure Blob remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

repFactor = auto

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
# There are multiple ways to fully specify the location. Here, for example, the
# Azure container is specified in its own setting, but it can also be specified as
# part of the "path" setting. See the indexes.conf.spec file for more information.
remote.azure.endpoint = https://account-name.blob.core.windows.net
remote.azure.container_name = your-container
path = azure://example/20_39/TID_01

# To authenticate with the remote storage service, you must use either hardcoded access/secret
# keys or Azure Active Directory with configured Managed Identity. See the topic, "SmartStore on
# Azure Blob security strategies."

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

6. On the manager node, run:

```
splunk apply cluster-bundle --answer-yes
```

7. Install and enable the peer nodes and search head, as for any new indexer cluster. See Enable the peer nodes and Enable the search head. Wait briefly for the peer nodes to download the configuration bundle with the SmartStore settings. To view the status of the configuration bundle process, you can run the `splunk show cluster-bundle-status` command, described in Update common peer configurations and apps.
8. Test the deployment.
    1. You can monitor the status of the cluster start-up process from the manager node with this command:

       ```
       splunk show cluster-status -auth <admin>:<password>
       ```
    2. To confirm remote storage access across the indexer cluster:
        1. Place a sample text file in the remote store.
        2. From one of the peer nodes, run this command, which recursively lists any files that are present in the remote store:

           ```
           splunk cmd splunkd rfs -- ls --starts-with volume:remote_store
           ```

       If you see the sample file when you run the command, you have access to the remote store.
    3. Validate data transfer to the remote store:
        1. Send some data to the indexers.
        2. Wait for buckets to roll. If you don't want to wait for buckets to roll naturally, you can manually roll some buckets from a peer node:

           ```
           splunk _internal call /data/indexes/<index_name>/roll-hot-buckets -auth
           <admin>:<password>
           ```
        3. Look for warm buckets being uploaded to remote storage.
    4. Validate data transfer from the remote store:

       Note: At this point, you should be able to run normal searches against this data. In the majority of cases, you will not be transferring any data from the remote storage, because the data will already be in a local cache. Therefore, to validate data transfer from the remote store, it is recommended that you first evict a bucket from the local cache of one of the peer nodes.
        1. On one of the peer nodes, evict a bucket from the cache, with a POST to this REST endpoint:

           ```
           services/admin/cacheman/<cid>/evict
           ```
           where `<cid>` is `bid|<bucketId>|`. For example:
           "bid|cs_index~0~7D76564B-AA17-488A-BAF2-5353EA0E9CE5|"

           Note: To get the `bucketId` for a bucket, go to a search head node and run a search on your test index. For example:

           ```
           splunk search "|rest /services/admin/cacheman | search title=*cs_index*  | fields
           splunk_server, title" -auth <admin>:<password>
           ```
           The results list the set of buckets (by `bucketId`) in the specified test index, along with their associated peer nodes. You can use this information to evict one of the buckets from the cache of one of the peer nodes.
        2. Run a search locally on the peer node. The search must be one that requires data from the evicted bucket.

       The peer node must now transfer the bucket from remote storage to run the search. After running the search, you can check that the bucket has reappeared in the cache.

Follow-on steps

Once your cluster is running with SmartStore, there are a number of configuration matters that warrant your immediate attention. In particular:

- On the manager node, edit the `$SPLUNK_HOME/etc/manager-apps/_cluster/local/indexes.conf` file and configure the data retention settings to ensure that the cluster follows your desired freezing behavior. See Configure data retention for SmartStore indexes.

This step is extremely important, to avoid unwanted bucket freezing and possible data loss. SmartStore bucket-freezing behavior and settings are different from the non-SmartStore behavior and settings.

- On the manager node, edit `$SPLUNK_HOME/etc/manager-apps/_cluster/local/server.conf` to make any necessary changes to the SmartStore-related `server.conf` settings on the peer nodes. In particular, configure the cache size to fit the needs of your deployment. See Configure the SmartStore cache manager.

After you make these changes to the configuration bundle on the manager node, apply the bundle to distribute the settings to the peer nodes:

```
splunk apply cluster-bundle --answer-yes
```

For details on other SmartStore settings, see Configure SmartStore.

Deploy multisite indexer clusters with SmartStore

You can deploy multisite indexer clusters with SmartStore to meet disaster recovery requirements. The clusters and their object stores can be hosted with a public cloud provider or in on-premises data centers.

The following SmartStore multisite deployment types are available:

- Active-active multisite hosted in a public cloud provider (AWS, GCP, or Azure), within a single region across multiple zones
- Active-active multisite hosted across data centers, using S3 API-compliant on-premise object stores

The multisite deployment must be hosted within either a single public cloud provider or a set of on-prem data centers. Sites cannot span multiple cloud providers, nor can they span a cloud provider and a data center.

Requirements for multisite indexer clusters with SmartStore

There are several layers of requirements to consider when deploying multisite indexer clusters with SmartStore:

- Standard requirements for all multisite clusters and for all SmartStore-enabled clusters. See Standard requirements for multisite clusters and for SmartStore-enabled clusters.
- Requirements for all multisite clusters with SmartStore, independent of deployment type. See Requirements for all multisite clusters with SmartStore.
- Specific requirements for each deployment type.
  - For public cloud provider hosted, see Public cloud provider hosted, within a single region.
  - For on-premises hosted, see On premises hosted, across data centers.

Standard requirements for multisite clusters and for SmartStore-enabled clusters

The standard requirements and deployment methods for multisite clusters and for SmartStore-enabled clusters also apply to the combination of multisite clusters with SmartStore. See the relevant topics in this manual, including, in particular:

- Multisite indexer cluster deployment overview
- SmartStore system requirements
- Deploy SmartStore on a new indexer cluster

Do not proceed with the deployment until you thoroughly understand the requirements and deployment methods for multisite indexer clusters and SmartStore-enabled indexer clusters.

Requirements for all multisite clusters with SmartStore

Multisite clusters with SmartStore have several additional requirements on top of those standard requirements.The requirements listed in this section apply to all deployments of multisite clusters with SmartStore.

Deployment requirements

These requirements apply to all deployments of multisite clusters with SmartStore:

- Designate one site as primary. This site hosts the active cluster manager node.
- Each cluster site must have a manager node. Exactly one manager node in the cluster must be active at any time. The other manager nodes must be maintained in standby mode.
- Search heads are typically deployed on each site, but this is not a strict requirement.
- Site locations host two object stores in an active-active replicated relationship. Depending on the deployment type, the set of cluster peer nodes can be sending data to one or both object stores.

Configuration requirements

These configurations are essential for all multisite clusters with SmartStore:

- On each search head, in server.conf, set `site=site0` to disable search affinity.
- On the manager node, in server.conf, configure `site_replication_factor` and the `site_search_factor` to ensure that each site holds at least one searchable copy of each bucket.
- To ensure that the cluster continues to ingest all incoming data if one or more peer nodes fail, configure the forwarders to load balance their data across all peer nodes on all sites. See Use forwarders to get data into the indexer cluster.

Prepare the standby manager nodes

Each site location must host a manager node. A properly functioning cluster has exactly one active manager node. The other manager nodes must be maintained in standby mode, so that they are available if the site with the active manager fails.

See Handle manager site failure for details on how to prepare for and implement manager node failover. These are the main preparatory steps described in that topic:

1. Set up a manager node on each site.
2. Set the manager node on the primary site to be the active manager.

3. Copy over the active manager's configurations to each standby manager.
4. On an ongoing basis, copy any new or changed configurations from the active to the standby managers.
5. Develop a plan for explicit failover to a standby manager upon loss of the active manager.

Public cloud provider hosted, within a single region

This deployment type is available with AWS, GCP, or Microsoft Azure as cloud provider. AWS-hosted deployments use S3 as the remote store. GCP-hosted deployments use GCS as the remote store. Azure-hosted deployments use Azure Blob storage as the remote store.

In this deployment type, each indexer cluster site is hosted in a separate zone within a single region. Indexers in each site/zone are configured with the same remote store endpoint. The public cloud provider handles object store failover between zones within a region, in case of any zone failures, including the loss of an entire zone.

In addition, each zone must host a cluster manager node. Only one manager is active at a time. The other managers are prepared as standby failovers.

Deployment topology

These are the essential elements of the deployment topology:

- Each indexer cluster site resides on a separate zone within a single region.
- Network latency between zones is less than 15ms.
- A typical deployment has three sites (zones) in total.
- Each zone hosts either an active or a standby cluster manager node.
- All peer nodes across all sites point to a single object store URI endpoint.
- The public cloud provider handles synchronous replication between object stores across zones in the same region and performs failover when necessary.

Site failure and recovery

If a site fails, note the following:

- If the failed site was hosting the active cluster manager, you must immediately switch to a standby manager node on one of the remaining sites. See Handle manager site failure.
- The public cloud provider automatically handles object store failures within a zone.
- Forwarders configured for load balancing across the cluster automatically redirect to peers on the remaining sites.
- Any searches in progress when a site fails might return incomplete results if the search heads were accessing peers on the failed site. Post-failure, new searches are automatically redirected to peers on the remaining sites.

When the failed site returns to the cluster, you must handle these cluster manager-related issues:

1. Ensure that the peer nodes on the recovered site point to the new active manager.
2. Because the recovered site's manager now serves as a standby manager, include it in future configuration updates, as described in Prepare the standby manager nodes.

On premises hosted, across data centers

This deployment type is available only for S3-API-compliant object stores.

This deployment type is limited to two sites, with each site hosted in an on-premises data center. Both sites host active replicated object stores. One site hosts an active cluster manager and the other site hosts a standby cluster manager.

Deployment topology

These are the essential elements of the deployment topology:

- Each indexer cluster site resides in an on-premises data center.
- Network latency between sites is a maximum of 300ms. For best performance, the recommended maximum latency is 100ms.
- This topology is limited to two sites.
- Each site location hosts an active object store.
- Each peer node points to the same remote object store URI through a third-party VIP or GSLB,
- The VIP or GSLB routes traffic from each peer node to the object store hosted on its site's location. In the case of object store failure, the VIP or GSLB reroutes traffic as necessary to the remaining active object store.
- One site location hosts an active cluster manager, and the other site location hosts a standby cluster manager node.

Object store requirements

For the basic requirements for S3 remote object stores for SmartStore, see Configure the S3 remote store for SmartStore.

Requirements for third-party object stores for on-premises multisite deployments include the following:

- Must be compliant with the S3 API.
- Supports bi-directional replication between physical object stores.
- Retries replication of uploaded objects to the target store to ensure data is synchronized in the case of object store failure and recovery.
- Supports object versioning similar to AWS S3. Versioning of objects must be based on object creation or modification timestamps, not replication timestamp.
- Supports delete marker replication.

In addition, the object store vendor must provide assurance that maximum replication lag time between remote stores does not exceed your Recovery Point Objective (RPO) requirements.

Details of normal operation

Peer nodes in SmartStore-enabled multisite clusters upload and download data from the remote store no differently from peer nodes in SmartStore-enabled single-site clusters. The `path` setting on all peer nodes across the cluster must share an identical URI to identify the remote store. The difference is that the URI points to a VIP or GSLB that routes traffic to the active remote store on the peer's site location (data center). Thus, in normal operation, the peers always upload data to, and download data from, their local remote store.

Data uploaded to one remote store is replicated to the other remote store, using the capability provided by the remote store vendor. Replication solutions usually rely on asynchronous replication, which introduces a lag that can result in temporary unavailability of the most recent data replicated to the non-local store. The amount of replication lag varies according to the upload traffic, available network bandwidth between locations, and vendor-specific replication latency.

Replication lag is an inherent characteristic of asynchronous replication. Splunk Enterprise provides no additional protections for data that has not yet been replicated to a second object store due to replication lag.

Because search affinity is disabled, search heads can access peer nodes on either site to fulfill search requests. This can result in WAN traffic across data centers.

Search requests that involve data not resident on local peer node caches can return incomplete data if the requested data is not yet available on the local object store but is in the process of replication to the local object store. This condition affects recent datasets within the replication lag time frame.

Site failure and recovery

If a site location fails, note the following:

- If the failed site was hosting the active cluster manager, you must immediately switch to the standby manager node on the remaining site. See Handle manager site failure.
- Forwarders configured for load balancing across the cluster automatically redirect to peers on the remaining site.
- Any searches in progress when a site fails might return incomplete results if the search heads were accessing peers on the failed site. Post-failure, new searches are automatically redirected to peers on the remaining site.
- Data uploaded to the remaining object store is not replicated to the object store on the failed location.
- Due to replication lag, the remaining object store might be missing data recently uploaded to the failed object store. This situation is temporary and will be rectified when the failed object store returns to service.

In the event of a catastrophic failure that results in permanent failure of an object store, data recently uploaded to that object store which was not replicated to the remaining object store might be permanently lost. In such a case, the cluster emits incomplete search results warnings and bucket fixup errors in response to attempts to access the lost buckets. To eliminate these warnings and errors, contact Splunk Customer Support for assistance in removing the corresponding bucket metadata.

When the failed site returns to the cluster, you must handle these cluster manager-related issues:

1. Ensure that the peer nodes on the recovered site point to the new active manager.
2. Because the recovered site's manager now serves as the standby manager, include it in future configuration updates, as described in Prepare the standby manager nodes.

When the failed object store comes back online, object store replication restarts, with the object stores also replicating any data that was unsent at the time of the failure, as well as data uploaded to the remaining object store while the other object store was down. The amount of time required to resync the object stores depends on factors such as the length of time the failure lasted (and thus the amount of accumulated unreplicated data) and replication throughput.

While the object stores are resyncing, search requests might produce incomplete results. Since the recovering object store can be missing significant amounts of recent data, it takes some time for it to catch up through the resyncing process. Partial search results can occur from attempts to access data on the recovering object store that has not yet been synced to that store.

Object-store-only failure and recovery

In cases where an object store fails but its corresponding cluster site remains active, the VIP or GSLB redirects traffic from the peer nodes on the site to the remaining object store. This results in increased WAN traffic across data centers.

Due to replication lag, the remaining object store might be missing data recently uploaded to the failed object store, possibly resulting in incomplete search results.This situation is temporary and will be rectified when the failed object store returns to service.

In the event of a catastrophic failure that results in permanent failure of an object store, data recently uploaded to that object store which was not replicated to the remaining object store might be permanently lost. In such a case, the cluster emits incomplete search results warnings and bucket fixup errors, in response to attempts to access the lost buckets. To eliminate these warnings and errors, contact Splunk Customer Support for assistance in removing the corresponding bucket metadata.

When the failed object store comes back online, object store replication restarts, with the object stores also replicating any data that was unsent at the time of the failure, as well as data uploaded to the remaining object store while the other object store was down. The amount of time required to resync the object stores depends on factors such as the length of time the failure lasted (and thus the amount of accumulated unreplicated data) and replication throughput.

While the object stores are resyncing, search requests might produce incomplete results. Since the recovering object store can be missing significant amounts of recent data, it takes some time for it to catch up through the resyncing process. Partial search results can occur from attempts to access data on the recovering object store that has not yet been synced to that store.

Deploy SmartStore on a new standalone indexer

During this procedure, you:

1. Install a new indexer .
2. Configure the indexer to access the remote store.
3. Test the deployment.

Note: This procedure configures SmartStore for a new standalone indexer only. It does not describe how to load existing data onto an indexer. To migrate existing local indexes to SmartStore, see Migrate existing data on a standalone indexer to SmartStore. To bootstrap existing SmartStore data onto a new standalone indexer, see Bootstrap SmartStore indexes.

To deploy SmartStore on a cluster, see Deploy SmartStore on a new indexer cluster.

Prerequisites

Read:

- SmartStore system requirements
- Choose the storage location for each index
- Documentation provided by the vendor of the remote storage service that you are using

Cautions

Be aware of these configuration issues:

- The value of the `path` setting for each remote volume stanza must be unique to the indexer. You can share remote volumes only among indexes within a single standalone indexer. In other words, if indexes on one indexer use a particular remote volume, no index on any other standalone indexer or indexer cluster can use the same

remote volume.
- Leave `maxDataSize` at its default value of "auto" (750MB) for each SmartStore index.
- The `coldPath` setting for each SmartStore index requires a value, even though the setting is ignored except in the case of migrated indexes.
- The `thawedPath` setting for each SmartStore index requires a value, even though the setting has no practical purpose because you cannot thaw data to a SmartStore index. See Thawing data and SmartStore.

Deploy SmartStore

The following procedure assumes that you are deploying SmartStore on a new indexer. It also assumes that you are deploying SmartStore for all indexes on the indexer, using a single remote location. If you want to deploy SmartStore for some indexes only, or if you want to use multiple remote locations for the SmartStore indexes, you can modify the procedure to fit your need.

1. Ensure that you have met the prerequisites. In particular, read:
    - SmartStore system requirements
    - The appropriate topic for configuring your remote storage type:
        - Configure the S3 remote store for SmartStore
        - Configure the GCS remote store for SmartStore
        - Configure the Azure Blob remote store for SmartStore
2. Understand SmartStore security strategies and prepare to implement them as necessary during the deployment process. See the topic on security strategies for your remote storage type:
    - SmartStore on S3 security strategies
    - SmartStore on GCS security strategies
    - SmartStore on Azure Blob security strategies
3. Install a new Splunk Enterprise instance to serve as the standalone indexer. For information on how to install Splunk Enterprise, read the *Installation Manual*.
4. Create or edit `$SPLUNK_HOME/etc/system/local/indexes.conf` and specify the SmartStore settings, as shown in the examples below.

    Using an S3 remote object store:
    This example configures SmartStore indexes, using an S3 remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = s3://mybucket/some/path

# The following S3 settings are required only if you're using the access and secret
# keys. They are not needed if you are using AWS IAM roles.
```

```
remote.s3.access_key = <S3 access key>
remote.s3.secret_key = <S3 secret key>
remote.s3.endpoint = https:|http://<S3 host>

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

Using a GCS remote object store:

This example configures SmartStore indexes, using a GCS remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = gs://mybucket/some/path

# There are several ways to specify credentials. For details, see the topic,
# "SmartStore on GCS security strategies." One way to specify credentials
# is to point to a file, as shown here.
remote.gs.credential_file = credential.json

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

Using an Azure Blob remote object store:

This example configures SmartStore indexes, using an Azure Blob remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
```

```
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
# There are multiple ways to fully specify the location. Here, for example, the
# Azure container is specified in its own setting, but it can also be specified as
# part of the "path" setting. See the indexes.conf.spec file for more information.
remote.azure.endpoint = https://account-name.blob.core.windows.net
remote.azure.container_name = your-container
path = azure://example/20_39/TID_01

# To authenticate with the remote storage service, you must use either hardcoded access/secret
# keys or Azure Active Directory with configured Managed Identity. See the topic, "SmartStore on
# Azure Blob security strategies."

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

5. Restart the indexer.
6. Test the deployment.
    1. To confirm remote storage access:
        1. Place a sample text file in the remote store.
        2. On the indexer, run this command, which recursively lists any files that are present in the remote store:

            ```
            splunk cmd splunkd rfs -- ls --starts-with volume:remote_store
            ```

        If you see the sample file when you run the command, you have access to the remote store.
    2. Validate data transfer to the remote store:
        1. Send some data to the indexer.
        2. Wait for buckets to roll. If you don't want to wait for buckets to roll naturally, you can manually roll some buckets:

            ```
            splunk _internal call /data/indexes/<index_name>/roll-hot-buckets -auth
            <admin>:<password>
            ```

        3. Look for warm buckets being uploaded to remote storage.
    3. Validate data transfer from the remote store:

        Note: At this point, you should be able to run normal searches against this data. In the majority of cases, you will not be transferring any data from the remote storage, because the data will already be in the local cache. Therefore, to validate data transfer from the remote store, it is recommended that you first evict a bucket from the local cache.
        1. Evict a bucket from the cache, with a POST to this REST endpoint:

            ```
            services/admin/cacheman/<cid>/evict
            ```

where `<cid>` is `bid|<bucketId>|`. For example: "bid|cs_index~0~7D76564B-AA17-488A-BAF2-5353EA0E9CE5|"

Note: To get the `bucketId` for a bucket, run a search on your test index. For example:

```
splunk search "|rest /services/admin/cacheman | search title=*cs_index*  | fields
title" -auth <admin>:<password>
```

2. Run a search that requires data from the evicted bucket.

The indexer must now transfer the bucket from remote storage to run the search. After running the search, you can check that the bucket has reappeared in the cache.

7. Perform all other configurations necessary for any deployment of a standalone indexer. For example, configure the connections between indexer and forwarders and between indexer and search head.

Follow-on steps

Once your indexer is running with SmartStore, there are a number of configuration matters that warrant your immediate attention. In particular:

- Edit `$SPLUNK_HOME/etc/system/local/indexes.conf` and configure the data retention settings to ensure that the indexer follows your desired freezing behavior. See Configure data retention for SmartStore indexes.

This step is extremely important, to avoid unwanted bucket freezing and possible data loss. SmartStore bucket-freezing behavior and settings are different from the non-SmartStore behavior and settings.

- Edit `$SPLUNK_HOME/etc/system/local/server.conf` to make any necessary changes to the SmartStore-related `server.conf` settings. In particular, configure the cache size to fit the needs of your deployment. See Configure the SmartStore cache manager.

After you make these changes, restart the indexer.

For details on other SmartStore settings, see Configure SmartStore.

Migrate existing data on an indexer cluster to SmartStore

You can migrate the existing data on your indexer cluster from local storage to the remote store.

This procedure describes how to migrate all the indexes on the indexer cluster to SmartStore. You can modify the procedure if you only want to migrate some of the indexes. Indexers support a mixed environment of SmartStore and non-SmartStore indexes.

Because this process requires the cluster to upload large amounts of data, it can take a long time to complete and can have a significant impact on concurrent indexing and searching.

You cannot revert an index to non-SmartStore after you migrate it to SmartStore.

Migrate data

Perform the migration operation in two phases:

1. Test the SmartStore configurations and remote connectivity on a standalone test instance.
2. Run the migration by applying the configurations to your production indexer cluster.

Prerequisites

- Read:
  - SmartStore system requirements
  - Configure SmartStore
  - Update common peer configurations and apps
  - Choose the storage location for each index
  - Documentation provided by the vendor of the remote storage service that you are using
- If the cluster was once migrated from single-site to multisite, you must convert any pre-existing, single-site buckets to follow the multisite replication and search policies. To do so, change the `constrain_singlesite_buckets` setting in the manager node's `server.conf` file to "false" and restart the manager node. See Configure the manager to convert existing buckets to multisite.
- The cluster should be on the smaller side, with a maximum of 20 indexers. If you want to migrate a larger cluster, consult Splunk Professional Services.
- Be aware of these configuration issues:
  - The value of the `path` setting for each remote volume stanza must be unique to the indexer cluster. You can share remote volumes only among indexes within a single cluster. In other words, if indexes on one cluster use a particular remote volume, no index on any other cluster or standalone indexer can use the same remote volume.
  - You must set all SmartStore indexes in an indexer cluster to use `repFactor = auto`.
  - Leave `maxDataSize` at its default value of "auto" (750MB) for each SmartStore index.
  - The `coldPath` setting for each SmartStore index requires a value, even though the setting is ignored except in the case of migrated indexes.
- The `thawedPath` setting for each SmartStore index requires a value, even though the setting has no practical purpose because you cannot thaw data to a SmartStore index. See Thawing data and SmartStore.
- Reconfigure the cluster as necessary to conform with the lists of unsupported features, current restrictions, and incompatible settings:
  - Features not supported by SmartStore
  - Current restrictions on SmartStore use. Regarding the requirement that replication factor and search factor be equal, you can make this change post-migration.
  - Settings in indexes.conf that are incompatible with SmartStore or otherwise restricted

1. Test the configuration on a standalone instance

The purpose of performing the test on a standalone instance is to:

- test remote store connectivity.
- validate the configuration.

Steps

1. Ensure that you have met all prerequisites relevant to this test setup. In particular, read:
   - SmartStore system requirements
   - The appropriate topic for configuring your remote storage type:

    - ◊ Configure the S3 remote store for SmartStore
    - ◊ Configure the GCS remote store for SmartStore
    - ◊ Configure the Azure Blob remote store for SmartStore
2. Understand SmartStore security strategies and prepare to implement them as necessary during the deployment process. See the topic on security strategies for your remote storage type:
    - ♦ SmartStore on S3 security strategies
    - ♦ SmartStore on GCS security strategies
    - ♦ SmartStore on Azure Blob security strategies
3. Install a new Splunk Enterprise instance. For information on how to install Splunk Enterprise, read the *Installation Manual*.
4. Edit `indexes.conf` in `$SPLUNK_HOME/etc/system/local` to specify the SmartStore settings for your indexes. These should be the same group of settings that you intend to use later on your production deployment.

Using an S3 remote object store:
This example configures SmartStore indexes, using an S3 remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = s3://mybucket/some/path

# The following S3 settings are required only if you're using the access and secret
# keys. They are not needed if you are using AWS IAM roles.

remote.s3.access_key = <S3 access key>
remote.s3.secret_key = <S3 secret key>
remote.s3.endpoint = https:|http://<S3 host>

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

Using a GCS remote object store:
This example configures SmartStore indexes, using a GCS remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = gs://mybucket/some/path

# There are several ways to specify credentials. For details, see the topic,
# "SmartStore on GCS security strategies." One way to specify credentials
# is to point to a file, as shown here.
remote.gs.credential_file = credential.json

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

Using an Azure Blob remote object store:

This example configures SmartStore indexes, using an Azure Blob remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
# There are multiple ways to fully specify the location. Here, for example, the
# Azure container is specified in its own setting, but it can also be specified as
# part of the "path" setting. See the indexes.conf.spec file for more information.
remote.azure.endpoint = https://account-name.blob.core.windows.net
remote.azure.container_name = your-container
path = azure://example/20_39/TID_01

# To authenticate with the remote storage service, you must use either hardcoded access/secret
```

```
# keys or Azure Active Directory with configured Managed Identity. See the topic, "SmartStore on
# Azure Blob security strategies."

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

5. Restart the instance.
6. Test the deployment:
    1. To confirm remote storage access:
        1. Place a sample text file in the remote store.
        2. On the Splunk Enterprise instance, run this command, which recursively lists any files that are present in the remote store:

           ```
           splunk cmd splunkd rfs -- ls --starts-with volume:remote_store
           ```

        If you see the sample file when you run the command, you have access to the remote store.
    2. Validate data transfer to the remote store:
        1. Send some data to the indexer.
        2. Wait for buckets to roll. If you don't want to wait for buckets to roll naturally, you can manually roll some buckets:

           ```
           splunk _internal call /data/indexes/<index_name>/roll-hot-buckets -auth
           <admin>:<password>
           ```
        3. Look for warm buckets being uploaded to remote storage.
    3. Validate data transfer from the remote store:

       Note: At this point, you should be able to run normal searches against this data. In the majority of cases, you will not be transferring any data from the remote storage, because the data will already be in the local cache. Therefore, to validate data transfer from the remote store, it is recommended that you first evict a bucket from the local cache.
        1. Evict a bucket from the cache, with a POST to this REST endpoint:

           ```
           services/admin/cacheman/<cid>/evict
           ```
           where `<cid>` is `bid|<bucketId>|`. For example: "bid|cs_index~0~7D76564B-AA17-488A-BAF2-5353EA0E9CE5|"

           Note: To get the `bucketId` for a bucket, run a search on your test index. For example:

           ```
           splunk search "|rest /services/admin/cacheman | search title=*cs_index*  | fields
           title" -auth <admin>:<password>
           ```
        2. Run a search that requires data from the evicted bucket.

       The instance must now transfer the bucket from remote storage to run the search. After running the search, you can check that the bucket has reappeared in the cache.

2. Run the migration on the indexer cluster

In this procedure, you configure your cluster for SmartStore. The goal of the procedure is to migrate all existing warm and cold buckets on all indexes to SmartStore. Going forward, all new warm buckets will also reside in SmartStore.

The migration process takes a while to complete. If you have a large amount of data, it can take a long while. Expect some degradation of indexing and search performance during the migration. For that reason, it is best to schedule the migration for a time when your indexers will be relatively idle.

Steps

1. Ensure that you have met the prerequisites. In particular, read:
    - SmartStore system requirements
    - The appropriate topic for configuring your remote storage type:
        - Configure the S3 remote store for SmartStore
        - Configure the GCS remote store for SmartStore
        - Configure the Azure Blob remote store for SmartStore
    - Features not supported by SmartStore
    - Current restrictions on SmartStore use
2. Understand SmartStore security strategies and prepare to implement them as necessary during the deployment process. See the topic on security strategies for your remote storage type:
    - SmartStore on S3 security strategies
    - SmartStore on GCS security strategies
    - SmartStore on Azure Blob security strategies
3. Upgrade all cluster nodes (manager node, peer nodes, search heads) to the latest version of Splunk Enterprise. See Upgrade an indexer cluster.
4. Confirm that the cluster is in a **valid** and **complete** state, with both replication and search factors met. Go to the Manager Node Dashboard to confirm.
5. Confirm that there are no bucket fixup tasks in progress or pending. Go to the Manager Node Dashboard, click on the Indexes tab, and then click on the Bucket Status button to confirm.
6. Run `splunk enable maintenance-mode` on the manager node. To confirm that the manager is in maintenance mode, run `splunk show maintenance-mode`.
7. Stop all the peer nodes. When bringing down the peers, use the `splunk stop` command, not `splunk offline`.
8. On the manager node, edit the existing `$SPLUNK_HOME/etc/manager-apps/_cluster/local/indexes.conf` file to make the following additions. See "Structure of the configuration bundle" for information on the `manager-apps` directory.

    Do not replace the existing indexes.conf file. You need to retain its current settings, such as its index definition settings. Instead, merge these additional settings into the existing file. Be sure to remove any other copies of these settings from the file.

    1. Specify the SmartStore index global and volume settings. Assuming that you have already tested these settings on your standalone instance, you can simply copy the settings over from your standalone instance, remembering to add a line for `repFactor=auto`. For example:

        Using an S3 remote object store:

        ```
        [default]
        # Configure all indexes to use the SmartStore remote volume called
        # "remote_store".
        # Note: If you want only some of your indexes to use SmartStore,
        # place this setting under the individual stanzas for each of the
        # SmartStore indexes, rather than here.
        remotePath = volume:remote_store/$_index_name

        repFactor = auto

        # Configure the remote volume.
        ```

```
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = s3://mybucket/some/path

# The following S3 settings are required only if you're using the access and secret
# keys. They are not needed if you are using AWS IAM roles.

remote.s3.access_key = <S3 access key>
remote.s3.secret_key = <S3 secret key>
remote.s3.endpoint = https:|http://<S3 host>

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify
them here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

Using a GCS remote object store:

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

repFactor = auto

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = gs://mybucket/some/path

# There are several ways to specify credentials. For details, see the topic,
# "SmartStore on GCS security strategies." One way to specify credentials
# is to point to a file, as shown here.
remote.gs.credential_file = credential.json

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify
them here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

Using an Azure Blob remote object store:

```
[default]
# Configure all indexes to use the SmartStore remote volume called
```

```
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

repFactor = auto

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
# There are multiple ways to fully specify the location. Here, for example, the
# Azure container is specified in its own setting, but it can also be specified as
# part of the "path" setting. See the indexes.conf.spec file for more information.
remote.azure.endpoint = https://account-name.blob.core.windows.net
remote.azure.container_name = your-container
path = azure://example/20_39/TID_01

# To authenticate with the remote storage service, you must use either hardcoded
access/secret
# keys or Azure Active Directory with configured Managed Identity. See the topic, "SmartStore
on
# Azure Blob security strategies."

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify
them here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

2. Configure the data retention settings, as necessary, to ensure that the cluster will follow your desired freezing behavior, post-migration. See Configure data retention for SmartStore indexes.

   This step is extremely important, to avoid unwanted bucket freezing and possible data loss. SmartStore bucket-freezing behavior and settings are different from the non-SmartStore behavior and settings.

9. On the manager node, edit `$SPLUNK_HOME/etc/manager-apps/_cluster/local/server.conf` to make any necessary changes to the SmartStore-related `server.conf` settings on the peer nodes. In particular, configure the cache size to fit the needs of your deployment. See Configure the SmartStore cache manager.
10. If the setting `rolling_restart` in `$SPLUNK_HOME/etc/system/local/server.conf` on the manager node has some value other than "restart", such as "searchable_force", you must change the value to "restart" before applying the bundle. On the manager node, run:

    ```
    splunk edit cluster-config -rolling_restart restart
    ```
11. On the manager node, run:

    ```
    splunk apply cluster-bundle --answer-yes
    ```
12. If you changed the value of `rolling_restart` prior to applying the bundle, revert the setting to its original value. On the manager node, run:

    ```
    splunk edit cluster-config -rolling_restart <original_restart_type>
    ```
13. Start all the peer nodes. Wait briefly for the peer nodes to download the configuration bundle with the SmartStore settings. To view the status of the configuration bundle process, you can run the `splunk show cluster-bundle-status` command, described in Update common peer configurations and apps.

14. Run `splunk disable maintenance-mode` on the manager. To confirm that the manager is not in maintenance mode, run `splunk show maintenance-mode`.
15. Wait briefly for the peer nodes to begin uploading their warm and cold buckets to the remote store.

    Cold buckets use the cold path as their cache location, post-migration. In all respects, cold buckets are functionally equivalent to warm buckets. The cache manager manages the migrated cold buckets in the same way that it manages warm buckets. The only difference is that the cold buckets will be fetched into the cold path location, rather than the home path location.
16. To confirm remote storage access across the indexer cluster, run this command from one of the peer nodes:

    ```
    splunk cmd splunkd rfs -- ls --starts-with volume:remote_store
    ```
    This command recursively lists any files that are present in the remote store. It should show that the cluster is starting to upload warm buckets to the remote store. If necessary, wait a little while for the first uploads to occur.
17. On the manager node, make any necessary changes to ensure that the indexer cluster's replication factor and search factor use the same values, for example, 3/3.
18. To determine that migration is complete:
    - Confirm that the cluster is in a **valid** and **complete** state, with both replication and search factors met.
    - Confirm that the search `|rest splunk_server=idx1 ... /services/admin/cacheman |search cm:bucket.stable=0 |stats count` returns 0.
19. Test SmartStore functionality. At this point, you should be able to run normal searches against this data. In the majority of cases, you will not be transferring any data from the remote storage, because the data will already be in the local cache. To validate data fetching from remote storage, do the following:
    1. On one of the peer nodes, look for a fully populated bucket, containing both tsidx files and the rawdata file.
    2. Evict the bucket from the cache, using this REST endpoint:

       ```
       services/admin/cacheman/<cid>/evict
       ```
       where `<cid>` is `bid|<bucketId>|`. For example:
       "bid|cs_index~0~7D76564B-AA17-488A-BAF2-5353EA0E9CE5|"

       Note: To get the `bucketId` for a bucket, go to a search head node and run a search on your test index. For example:

       ```
       splunk search "|rest /services/admin/cacheman | search title=*cs_index*  | fields splunk_server, title" -auth <admin>:<password>
       ```
       The results list the set of buckets (by `bucketId`) in the specified test index, along with their associated peer nodes. You can use this information to evict one of the buckets from the cache of one of the peer nodes.
    3. Run a search locally on the peer node. The search must be one that requires data from the evicted bucket.

    The peer must now transfer the bucket from remote storage to run the search. After running the search, you can check that the bucket has reappeared in the cache.

If you need to restart the cluster during migration, upon restart, migration will continue from where it left off.

Refrain from rebalancing data or removing excess buckets until you have run the SmartStore-enabled cluster successfully for a while. In particular, run these operations only after you have set the replication factor and search factor to use equal values and the cluster has performed any related bucket fixup.

Monitor the migration process

You can use the monitoring console to monitor migration progress. See Troubleshoot with the monitoring console.

You can also run an endpoint from the manager node to determine the status of the migration:

```
$ splunk search "|rest /services/admin/cacheman/_metrics |fields splunk_server migration.*" -auth
<admin>:<password>
```

The endpoint returns data on the migration, which you can use to determine how far along in the process each of the peers is. In this example, peer1 is on its 8th job, out of a total of 35, so the peer's migration is about 20-25% complete. The start_epoch field tells you when the migration began, allowing you to extrapolate an approximate completion time:

```
splunk_server      migration.current_job migration.start_epoch migration.status migration.total_jobs
---------------  ----------------------- --------------------- ---------------- ---------------------
cluster1-manager                                                  not_started
peer1.ajax.com               8                  1484942186         running                35
peer2.ajax.com               7                  1484942190         running                37
peer2.ajax.com               5                  1484942194         running                36
```

Once migration.status reaches "finished" on all peers, the migration is finished, and current_job will match total_jobs.

If any peer restarts during migration, its migration information is lost, and this endpoint cannot be used to check status of that peer, although the migration will, in fact, resume. The peer's reported status will remain "not_started" even after migration resumes. Instead, you can run the following endpoint on the restarted peer: "|rest /services/admin/cacheman |search cm:bucket.stable=0 |stats count" The count equals the number of upload jobs remaining, where an upload job represents a single bucket to be uploaded, or, in other words, (total_jobs - current_jobs) from the earlier endpoint. The count decrements to zero as migration continues.

Migrate existing data on a standalone indexer to SmartStore

You can migrate the existing data on your standalone indexer from local storage to the remote store.

This procedure describes how to migrate all the indexes on the indexer to SmartStore. You can modify the procedure if you only want to migrate some of the indexes. Indexers support a mixed environment of SmartStore and non-SmartStore indexes.

Because this process requires the indexer to upload large amounts of data, it can take a long time to complete and can have a significant impact on concurrent indexing and searching.

You cannot revert an index to non-SmartStore after you migrate it to SmartStore.

Migrate data

Perform the migration operation in two phases:

1. Test the SmartStore configurations and remote connectivity on a test indexer.
2. Run the migration by applying the configurations to your production indexer.

Prerequisites

- Read:
  - SmartStore system requirements
  - Configure SmartStore
  - Choose the storage location for each index
  - Documentation provided by the vendor of the remote storage service that you are using
- Be aware of these configuration issues:
  - The value of the `path` setting for each remote volume stanza must be unique to the indexer. You can share remote volumes only among indexes within a single standalone indexer. In other words, if indexes on one indexer use a particular remote volume, no index on any other standalone indexer or indexer cluster can use the same remote volume.
  - Leave `maxDataSize` at its default value of "auto" (750MB) for each SmartStore index.
  - The `coldPath` setting for each SmartStore index requires a value, even though the setting is ignored except in the case of migrated indexes.
- The `thawedPath` setting for each SmartStore index requires a value, even though the setting has no practical purpose because you cannot thaw data to a SmartStore index. See Thawing data and SmartStore.
- Reconfigure the indexer as necessary to conform with the lists of unsupported features, current restrictions, and incompatible settings:
  - Features not supported by SmartStore
  - Current restrictions on SmartStore use.
  - Settings in indexes.conf that are incompatible with SmartStore or otherwise restricted

1. Test the configuration on a test indexer

The purpose of testing the configuration is to:

- test remote store connectivity.
- validate the configuration.

Steps

1. Ensure that you have met all prerequisites relevant to this test setup. In particular, read:
   - SmartStore system requirements
   - The appropriate topic for configuring your remote storage type:
     - Configure the S3 remote store for SmartStore
     - Configure the GCS remote store for SmartStore
     - Configure the Azure Blob remote store for SmartStore
2. Understand SmartStore security strategies and prepare to implement them as necessary during the deployment process. See the topic on security strategies for your remote storage type:
   - SmartStore on S3 security strategies
   - SmartStore on GCS security strategies
   - SmartStore on Azure Blob security strategies
3. Install a new Splunk Enterprise instance. For information on how to install Splunk Enterprise, read the *Installation Manual*.
4. Edit `indexes.conf` in `$SPLUNK_HOME/etc/system/local` to specify the SmartStore settings for your indexes. These should be the same group of settings that you intend to use later on your production deployment.

   Using an S3 remote object store:
   This example configures SmartStore indexes, using an S3 remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = s3://mybucket/some/path

# The following S3 settings are required only if you're using the access and secret
# keys. They are not needed if you are using AWS IAM roles.

remote.s3.access_key = <S3 access key>
remote.s3.secret_key = <S3 secret key>
remote.s3.endpoint = https:|http://<S3 host>

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

Using a GCS remote object store:

This example configures SmartStore indexes, using a GCS remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = gs://mybucket/some/path

# There are several ways to specify credentials. For details, see the topic,
# "SmartStore on GCS security strategies." One way to specify credentials
# is to point to a file, as shown here.
remote.gs.credential_file = credential.json
```

```
# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

Using an Azure Blob remote object store:

This example configures SmartStore indexes, using an Azure Blob remote object store. The SmartStore-related settings are configured at the global level, which means that all indexes are SmartStore-enabled, and they all use a single remote storage volume, named "remote_store". The example also creates one new index, "cs_index".

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
# There are multiple ways to fully specify the location. Here, for example, the
# Azure container is specified in its own setting, but it can also be specified as
# part of the "path" setting. See the indexes.conf.spec file for more information.
remote.azure.endpoint = https://account-name.blob.core.windows.net
remote.azure.container_name = your-container
path = azure://example/20_39/TID_01

# To authenticate with the remote storage service, you must use either hardcoded access/secret
# keys or Azure Active Directory with configured Managed Identity. See the topic, "SmartStore on
# Azure Blob security strategies."

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify them
here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

For details on these settings, see Configure SmartStore. Also see indexes.conf.spec in the *Admin Manual.*

5. Restart the indexer.
6. Test the deployment:
    1. To confirm remote storage access:
        1. Place a sample text file in the remote store.
        2. On the indexer, run this command, which recursively lists any files that are present in the remote store:

```
splunk cmd splunkd rfs -- ls --starts-with volume:remote_store
```

    If you see the sample file when you run the command, you have access to the remote store.

2. Validate data transfer to the remote store:
    1. Send some data to the indexer.
    2. Wait for buckets to roll. If you don't want to wait for buckets to roll naturally, you can manually roll some buckets:

       ```
       splunk _internal call /data/indexes/<index_name>/roll-hot-buckets -auth
       <admin>:<password>
       ```
    3. Look for warm buckets being uploaded to remote storage.
3. Validate data transfer from the remote store:

   Note: At this point, you should be able to run normal searches against this data. In the majority of cases, you will not be transferring any data from the remote storage, because the data will already be in the local cache. Therefore, to validate data transfer from the remote store, it is recommended that you first evict a bucket from the local cache.
    1. Evict a bucket from the cache, with a POST to this REST endpoint:

       `services/admin/cacheman/<cid>/evict`
       where `<cid>` is `bid|<bucketId>|`. For example:
       "bid|cs_index~0~7D76564B-AA17-488A-BAF2-5353EA0E9CE5|"

       Note: To get the `bucketId` for a bucket, run a search on your test index. For example:

       ```
       splunk search "|rest /services/admin/cacheman | search title=*cs_index*  | fields
       title" -auth <admin>:<password>
       ```
    2. Run a search that requires data from the evicted bucket.

   The indexer must now transfer the bucket from remote storage to run the search. After running the search, you can check that the bucket has reappeared in the cache.

2. Run the migration on the production indexer

In this procedure, you configure your production indexer for SmartStore. The goal of the procedure is to migrate all existing warm and cold buckets on all indexes to SmartStore. Going forward, all new warm buckets will also reside in SmartStore.

The migration process takes a while to complete. If you have a large amount of data, it can take a long while. Expect some degradation of indexing and search performance during the migration. For that reason, it is best to schedule the migration for a time when your indexer will be relatively idle.

Steps

1. Ensure that you have met the prerequisites. In particular, read:
    - SmartStore system requirements
    - The appropriate topic for configuring your remote storage type:
        - Configure the S3 remote store for SmartStore
        - Configure the GCS remote store for SmartStore
        - Configure the Azure Blob remote store for SmartStore
    - Features not supported by SmartStore
    - Current restrictions on SmartStore use
2. Understand SmartStore security strategies and prepare to implement them as necessary during the deployment process. See the topic on security strategies for your remote storage type:
    - SmartStore on S3 security strategies

- SmartStore on GCS security strategies
- SmartStore on Azure Blob security strategies

3. Upgrade the indexer to the latest version of Splunk Enterprise.
4. Stop the indexer.
5. Edit the existing `$SPLUNK_HOME/etc/system/local/indexes.conf` file to make the following additions.

> Do not replace the existing indexes.conf file. You need to retain its current settings, such as its index definition settings. Instead, merge these additional settings into the existing file. Be sure to remove any other copies of these settings from the file.

1. Specify the SmartStore index global and volume settings. Assuming that you have already tested these settings on your test instance, you can simply copy the settings over from the test instance. For example:

Using an S3 remote object store:

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = s3://mybucket/some/path

# The following S3 settings are required only if you're using the access and secret
# keys. They are not needed if you are using AWS IAM roles.

remote.s3.access_key = <S3 access key>
remote.s3.secret_key = <S3 secret key>
remote.s3.endpoint = https:|http://<S3 host>

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify
them here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

Using a GCS remote object store:

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
```

```
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
path = gs://mybucket/some/path

# There are several ways to specify credentials. For details, see the topic,
# "SmartStore on GCS security strategies." One way to specify credentials
# is to point to a file, as shown here.
remote.gs.credential_file = credential.json

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify
them here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

Using an Azure Blob remote object store:

```
[default]
# Configure all indexes to use the SmartStore remote volume called
# "remote_store".
# Note: If you want only some of your indexes to use SmartStore,
# place this setting under the individual stanzas for each of the
# SmartStore indexes, rather than here.
remotePath = volume:remote_store/$_index_name

# Configure the remote volume.
[volume:remote_store]
storageType = remote

# The volume's 'path' setting points to the remote storage location where
# indexes reside. Each SmartStore index resides directly below the location
# specified by the 'path' setting.
# There are multiple ways to fully specify the location. Here, for example, the
# Azure container is specified in its own setting, but it can also be specified as
# part of the "path" setting. See the indexes.conf.spec file for more information.
remote.azure.endpoint = https://account-name.blob.core.windows.net
remote.azure.container_name = your-container
path = azure://example/20_39/TID_01

# To authenticate with the remote storage service, you must use either hardcoded
access/secret
# keys or Azure Active Directory with configured Managed Identity. See the topic, "SmartStore
on
# Azure Blob security strategies."

# This example stanza configures a custom index, "cs_index".
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
# SmartStore-enabled indexes do not use thawedPath or coldPath, but you must still specify
them here.
coldPath = $SPLUNK_DB/cs_index/colddb
thawedPath = $SPLUNK_DB/cs_index/thaweddb
```

2. Configure the data retention settings, as necessary, to ensure that the indexer will follow your desired freezing behavior, post-migration. See Configure data retention for SmartStore indexes.

This step is extremely important, to avoid unwanted bucket freezing and possible data loss. SmartStore bucket-freezing behavior and settings are different from the non-SmartStore behavior and settings.

6. Edit `$SPLUNK_HOME/etc/system/local/server.conf` to make any necessary changes to the SmartStore-related `server.conf` settings. In particular, configure the cache size to fit the needs of your deployment. See Configure the SmartStore cache manager.
7. Start the indexer.
8. Wait briefly for the indexer to begin uploading its warm and cold buckets to the remote store.

   Cold buckets use the cold path as their cache location, post-migration. In all respects, cold buckets are functionally equivalent to warm buckets. The cache manager manages the migrated cold buckets in the same way that it manages warm buckets. The only difference is that the cold buckets will be fetched into the cold path location, rather than the home path location.
9. To confirm remote storage access, run this command:

   ```
   splunk cmd splunkd rfs -- ls --starts-with volume:remote_store
   ```
   This command recursively lists any files that are present in the remote store. It should show that the indexer is starting to upload warm buckets to the remote store. If necessary, wait a little while for the first uploads to occur.
10. To determine that migration is complete, see Monitor the migration process.
11. Test SmartStore functionality. At this point, you should be able to run normal searches against this data. In the majority of cases, you will not be transferring any data from the remote storage, because the data will already be in the local cache. To validate data fetching from remote storage, do the following:
    1. On the indexer, look for a fully populated bucket, containing both tsidx files and the rawdata file.
    2. Evict the bucket from the cache, with a POST to this REST endpoint:

       ```
       services/admin/cacheman/<cid>/evict
       ```
       where `<cid>` is `bid|<bucketId>|`. For example:
       "bid|cs_index~0~7D76564B-AA17-488A-BAF2-5353EA0E9CE5|"

       Note: To get the `bucketId` for a bucket, run a search on your test index. For example:

       ```
       splunk search "|rest /services/admin/cacheman | search title=*cs_index*  | fields title"
       -auth <admin>:<password>
       ```
    3. Run a search locally on the indexer. The search must be one that requires data from the evicted bucket.

    The indexer must now transfer the bucket from remote storage to run the search. After running the search, you can check that the bucket has reappeared in the cache.

If you need to restart the indexer during migration, upon restart, migration will continue from where it left off.

Monitor the migration process

You can use the monitoring console to monitor migration progress. See Troubleshoot with the monitoring console.

You can also run an endpoint from the indexer to determine the status of the migration:

```
$ splunk search "|rest /services/admin/cacheman/_metrics |fields splunk_server migration.*" -auth
<admin>:<password>
```
The endpoint returns data on the migration, which you can use to determine how far along in the process the indexer is.

If the indexer restarts during migration, its migration information is lost, and this endpoint cannot be used to check status, although the migration will, in fact, resume. The indexer's reported status will remain "not_started" even after migration resumes. Instead, you can run the following endpoint on the indexer: "|rest /services/admin/cacheman |search cm:bucket.stable=0 |stats count" The count equals the number of upload jobs remaining, where an upload job represents a single bucket to be uploaded. The count decrements to zero as migration continues.

Bootstrap SmartStore indexes

Bootstrapping is the process of transferring SmartStore indexes to a new indexer cluster or standalone indexer. You first bring down an old indexer cluster or indexer that has SmartStore indexes. You then point a new indexer cluster or standalone indexer to the remote storage location previously used by the SmartStore indexes on the decommissioned cluster or indexer. The new cluster or indexer inherits the SmartStore buckets from the old cluster or indexer.

For example, if you want to replace an indexer cluster, you can take down the old cluster and bootstrap a new cluster. The new cluster inherits all the buckets from the old cluster. This can be useful if you need to upgrade the machines running your indexer cluster.

Similarly, you can use bootstrapping for scalability and high availability purposes, to move SmartStore indexes from a standalone indexer to an indexer cluster.

Multiple indexer clusters or standalone indexers cannot access the same remote volume. In other words, the value of any path setting in indexes.conf must be unique to a single running indexer or cluster. Do not share path settings among multiple indexers or clusters. Therefore, you must shut down the cluster or standalone indexer currently accessing buckets from a particular remote volume before you bootstrap a new cluster or indexer to access buckets from the same remote volume.

Types of bootstrapping

You can bootstrap SmartStore indexes from one indexer cluster or standalone indexer to another indexer cluster or standalone indexer. In other words, these types of bootstrapping are supported:

- Indexer cluster to indexer cluster
- Standalone indexer to standalone indexer
- Standalone indexer to indexer cluster
- Indexer cluster to standalone indexer

Bootstrap indexes onto an indexer cluster

You can boostrap indexes onto an indexer cluster from an old cluster or standalone indexer.

How to bootstrap indexes at initial start-up

You can bootstrap all indexes, or just a subset of indexes. See Deploy SmartStore on a new indexer cluster for information on how to use `indexes.conf` to configure SmartStore either globally or on a per-index basis.

You can bootstrap SmartStore indexes when the cluster initially starts up or later in its life cycle. This section describes how to bootstrap indexes at initial start-up.

To bootstrap SmartStore indexes onto a new indexer cluster:

1. Bring down the old cluster or standalone indexer after ensuring that all local data, including data from any hot buckets, is available on the remote store.
2. Bring up the new cluster and configure its indexes for SmartStore. Follow the procedure in Deploy SmartStore on an indexer cluster.
   Note the following:
   - When configuring `indexes.conf`, add stanzas for all SmartStore indexes that you want to bootstrap.
   - Confirm that you have set `repFactor=auto` globally, so that it applies to all indexes in the cluster, including the ones that you intend to bootstrap.
   - Configure the `path` attribute to point to the same remote storage location as on the old cluster or standalone indexer.
   - Maintain the remote store encryption and access settings used by the old cluster or standalone indexer.
   - All indexes that you bootstrap must be SmartStore-enabled on the new cluster. You cannot use bootstrapping to revert an index to non-SmartStore.

When bootstrapping occurs

The manager node initiates bootstrapping under either of these conditions:

- When the replication factor number of peer nodes have registered with the manager after indexer cluster start-up.
- After the manager pushes a new bootstrapping configuration to the peer nodes through the configuration bundle push method. This allows you to bootstrap SmartStore indexes after the cluster is already up and running, or to bootstrap SmartStore indexes selectively at varying points in the life of the cluster.

The bootstrap process

During bootstrapping, the manager node coordinates a discovery process:

1. The manager obtains a list of all SmartStore indexes from the peer nodes.
2. Bootstrapping proceeds on a per-index basis:
   1. The manager assigns one peer to obtain a list of all buckets for the index from the remote store.
   2. The peer returns the bucket list to the manager in batches.

      Batch size is controlled by the recreate_index_fetch_bucket_batch_size attribute in the manager node's server.conf file.
   3. The manager uses the list to assign buckets, randomly but evenly, across all available peer nodes.
   4. The manager provides each peer node with a set of target peer nodes, of a number equal to the replication factor.
   5. Each peer node downloads, from the remote store, the metadata for its assigned buckets.
   6. The peer node uses the downloaded metadata to update the index's `.bucketManifest` file and to create empty directories for its assigned buckets. The primary marker for each bucket resides on the peer that downloaded that bucket's metadata from the remote store.
   7. The peer node replicates copies of its primary buckets' metadata to target peer nodes to meet the replication factor.

The bootstrapping process proceeds quickly because only the bucket metadata is downloaded and replicated. The bucket contents themselves are not downloaded during bootstrapping, but only later, if needed for searching.

You can use the monitoring console to monitor bootstrapping progress. See Troubleshoot with the monitoring console.

Bootstrap indexes onto a standalone indexer

You can bootstrap indexes onto a standalone indexer from an old standalone indexer or cluster.

You cannot run multiple standalone indexers against the same remote store. Only a single standalone indexer can access a particular remote store.

How to bootstrap indexes at initial start-up

You can bootstrap all indexes, or just a subset of indexes, from the remote store. See Deploy SmartStore on a new standalone indexer for information on how to use `indexes.conf` to configure SmartStore either globally or on a per-index basis.

You can bootstrap SmartStore indexes when the indexer initially starts up or later in its life cycle. This section describes how to bootstrap indexes at initial start-up. The process for bootstrapping indexes later is a simple variant of adding an index to an existing indexer.

To bootstrap SmartStore indexes onto a new indexer:

1. Bring down the old standalone indexer or cluster after ensuring that all local data, including data from any hot buckets, is available on the remote store.
2. Bring up the new indexer and configure its indexes for SmartStore. Follow the procedure in Deploy SmartStore on a new standalone indexer.
   Note the following:
   - When configuring `indexes.conf`, add stanzas for all SmartStore indexes that you want to bootstrap.
   - Configure the `path` attribute to point to the same remote storage location as on the old cluster or standalone indexer.
   - Maintain the remote store encryption and access settings used by the old cluster or standalone indexer.
   - All indexes that you bootstrap must be SmartStore-enabled on the new indexer. You cannot use bootstrapping to revert an index to non-SmartStore.
3. Restart the indexer.

The indexer initiates bootstrapping after it restarts.

The bootstrap process

During bootstrapping, the indexer performs a discovery process. Bootstrapping proceeds on a per-index basis:

1. The indexer obtains a list of all buckets for the index from the remote store.
2. The indexer downloads, from the remote store, the metadata for the index's buckets.
3. The indexer uses the downloaded metadata to update the index's `.bucketManifest` file and to create empty directories for the buckets.

The bootstrapping process proceeds quickly because only the bucket metadata is downloaded. The bucket contents themselves are not downloaded during bootstrapping, but only later, if needed for searching.

You can use the monitoring console to monitor bootstrapping progress. See Troubleshoot with the monitoring console.

Manage SmartStore

Configure SmartStore

SmartStore configuration settings reside in three files:

- indexes.conf
- server.conf
- limits.conf

SmartStore settings in indexes.conf

The SmartStore settings in `indexes.conf` enable and control SmartStore indexes.

You can enable SmartStore for all of an indexer's indexes, or you can enable SmartStore on an index-by-index basis, allowing a mix of SmartStore and non-SmartStore indexes on the same indexer.

When you configure these settings on an indexer cluster's peer nodes, you must deploy the settings through the configuration bundle method. As with all settings in indexes.conf, SmartStore settings must be the same across all peer nodes.

The table lists the main SmartStore-related settings in `indexes.conf` .

SmartStore indexes.conf setting	Stanza level	Description
`remotePath = <root path for remote volume>`	index or global	Enables SmartStore and sets the remote path for the index's volume. See Deploy SmartStore on a new indexer cluster.
`storageType = remote`	volume	Sets the volume's storage type to remote. See Deploy SmartStore on a new indexer cluster.
`path = <scheme>://<remote-location-specifier>`	volume	Sets the remote storage location where indexes reside. Each SmartStore index resides directly below the location specified by the path setting. The <scheme> identifies a supported remote storage system type, such as s3 (S3), gs (GCS), or azure (Azure). The <remote-location-specifier> is a string specific to the remote storage system that specifies the location of the indexes inside the remote system. See Deploy SmartStore on a new indexer cluster. **Caution:** The value of the `path` setting for each remote volume stanza must be unique to the indexer cluster or standalone indexer. You can share remote volumes only among indexes within a single cluster or standalone indexer. For example, if indexes on one cluster use a particular remote volume, no index on any other cluster or standalone indexer can use the same remote volume.

SmartStore indexes.conf setting	Stanza level	Description
`maxGlobalDataSizeMB = <integer>`	index or global	Determines bucket freezing behavior. Sets the maximum amount of space that the warm and cold buckets of a SmartStore index can occupy. When this maximum is exceeded, the oldest bucket gets frozen. See Configure data retention for SmartStore indexes.
`maxGlobalRawDataSizeMB= <integer>`	index or global	Determines bucket freezing behavior. Sets the maximum amount of cumulative raw data allowed in the warm and cold buckets of a SmartStore index. When this maximum is exceeded, the oldest bucket gets frozen. See Configure data retention for SmartStore indexes.
`hotlist_recency_secs = <integer>`	index	Specifies the time period, based on the bucket's age, that the cache manager attempts to protect a recent bucket from eviction. This setting operates on a per-index level, while the version of the setting in `server.conf` operates across all indexes. See Configure the SmartStore cache manager.
`hotlist_bloom_filter_recency_hours = <integer>`	index	Specifies the time period, based on the bucket's age, that the cache manager attempts to protect the bucket's non-journal and non-tsidx files, such as the `bloomfilter` file, from eviction. This setting operates on a per-index level, while the version of the setting in `server.conf` operates across all indexes. See Configure the SmartStore cache manager.

In `indexes.conf`, you can also configure various settings specific to your remote storage type. For example, there are a number of settings that begin with `remote.s3`, such as `remote.s3.access_key` and `remote.s3.secret_key`. These settings are specific to S3. Similarly, there are a number of settings that begin with `remote.gs` and `remote.azure`. These settings are specific to GCS and Azure, respectively. You configure these settings in the stanza where you configure the remote volume. For details on these and other `indexes.conf` settings, see indexes.conf.spec.

When specifying a setting that begins with remote.s3, you must use a lowercase "s" in "s3". Uppercase "S", such as remote.S3... is not a valid substitute and causes any attempted update to the setting's value to not take effect.

Compress tsidx files upon upload to S3

To reduce S3 usage and improve network performance, SmartStore can compress tsidx files before uploading them to the remote store. This capability uses zstd compression. When the files are subsequently downloaded to indexers, SmartStore will automatically decompress the files before placing them in the cache.

To enable tsidx file compression, set the `remote.s3.tsidx_compression` setting in `indexes.conf` to true. The setting default is false.

This capability is available for AWS S3 and S3-compliant remote stores only.

This feature is not backward compatible. If you downgrade your indexers to a pre-9.0 version after enabling tsidx compression, the indexers will be unable to use the compressed tsidx files.

If enabling this setting on an indexer cluster, you must make the change on all peer nodes through the configuration bundle method, as for any `indexes.conf` setting on an indexer cluster. A restart is not required.

Non-SmartStore-specific settings in indexes.conf

You must specify these configurations for all SmartStore indexes:

- `repFactor = auto`, for indexes on indexer cluster peer nodes (not standalone indexers)
- `maxDataSize = auto`. This is the default value (750MB), so you usually do not need to set it explicitly.
- `homePath` requires a path value. This is the location on local storage where hot and cached warm buckets reside.
- `coldPath` requires a path value, even though the setting is ignored except in the case of migrated indexes.
- `thawedPath` requires a path value, even though the setting has no practical purpose because you cannot thaw data to a SmartStore index. See Thawing data and SmartStore.

Settings in indexes.conf that are incompatible with SmartStore or otherwise restricted

SmartStore indexers and indexer clusters support a subset of features available with non-SmartStore indexers and indexer clusters. Settings related to incompatible features require special consideration.

In addition, SmartStore indexers and indexer clusters handle certain needs, such as data retention, differently from non-SmartStore indexers and indexer clusters. The associated settings are also different.

For information on features that are unsupported or restricted when using SmartStore, see:

- Features not supported by SmartStore
- Current restrictions on SmartStore use

The following `indexes.conf` settings must retain their default values:

- `createBloomfilter`. Do not change from default of true.
- `enableOnlineBucketRepair`. Do not change from default of true.
- `isReadOnly`. Do not change from default of false.
- `enableTsidxReduction`. Do not change from default of false.
- `maxDataSize`. Do not change from default of auto (recommended).
- `bloomHomePath`. Do not change the path location.
- `summaryHomePath`. Do not change the path location.
- `tstatsHomePath` . Do not change the path location.

The following `indexes.conf` settings are ignored by SmartStore:

- `maxWarmDBCount`
- `maxTotalDataSizeMB`
- `warmToColdScript`
- `homePath.maxDataSizeMB`
- `coldPath.maxDataSizeMB`
- `maxVolumeDataSizeMB`

The `journalCompression` setting can be set to either zstd (the default) or gzip.

Path settings and SmartStore

SmartStore uses the settings `remotePath` and `path` to identify the location of index storage on the remote store.

All other path settings identify locations on the local cache, not the remote store. These settings include:

- homePath
- coldPath
- thawedPath
- bloomHomePath
- summaryHomePath
- tstatsHomePath

See other sections of this topic for configuration and usage notes for each of these settings.

SmartStore settings in server.conf

The SmartStore-related settings in `server.conf` control the behavior of indexers, including the functionality of the cache manager. The table lists the most frequently configured settings.

SmartStore server.conf setting	Stanza level	Description
`eviction_policy = <string>`	`[cachemanager]`	Sets the policy that determines which buckets the cache manager evicts next. Defaults to "lru". See Configure the SmartStore cache manager.
`max_cache_size = <integer>`	`[cachemanager]`	Specifies the maximum space, in megabytes, that the cache can occupy on a disk partition. See Configure the SmartStore cache manager.
`eviction_padding = <integer>`	`[cachemanager]`	Specifies the additional space, in megabytes, beyond `minFreeSpace` that the cache manager uses as the threshold to start evicting data. See Configure the SmartStore cache manager.
`hotlist_recency_secs = <integer>`	`[cachemanager]`	Specifies the time period, based on the bucket's age, that the cache manager attempts to protect a recent bucket from eviction. This setting operates on a global level, across all indexes, while the version of the setting in `indexes.conf` operates on a per-index level. See Configure the SmartStore cache manager.
`hotlist_bloom_filter_recency_hours = <integer>`	`[cachemanager]`	Specifies the time period, based on the bucket's age, that the cache manager attempts to protect the bucket's non-journal and non-tsidx files, such as the `bloomfilter` file, from eviction. This setting operates on a global level, across all indexes, while the version of the setting in `indexes.conf` operates on a per-index level. See Configure the SmartStore cache manager.
various settings	`[clustering]`	Includes various, mostly low-level, settings that control remote bucket operations in indexer clusters.
`cleanRemoteStorageByDefault = <bool>`	`[general]`	Causes the `splunk clean eventdata` command to clean the remote indexes. Defaults to "false".

For details on these and other `server.conf` settings, see server.conf.spec.

Most of the SmartStore server.conf settings, including all of the common ones, are configured on the peer nodes. A few rarely-changed settings are configured instead on the manager node. All settings configured on the peer nodes must be the same across all peer nodes.

SmartStore settings in limits.conf

The `limits.conf` file contains several low-level search-related settings; for example, `bucket_localize_max_timeout_sec`. These settings are primarily related to the process of localizing buckets. Do not change the settings without a clear understanding of the underlying processes.

For details on `limits.conf` settings, see limits.conf.spec.

Configure the SmartStore cache manager

The cache manager maximizes search efficiency through intelligent management of the local cache. It favors retaining in the cache copies of buckets and files that have a high likelihood of participating in future searches. When the cache fills up, the cache manager removes, or "evicts", copies of buckets that are least likely to participate in future searches.

Since the cache manager removes only the cached copies of buckets, the eviction process does not result in loss of data. The manager copies continue to reside in remote storage,

For details on how the cache manager operates, see The SmartStore cache manager.

Cache manager settings reside in the `[cachemanager]` stanza in server.conf. In the case of an indexer cluster, you configure the cache manager on each peer node.

The cache manager operates at the global level, across all indexes on an indexer. Aside from the recency settings, you cannot configure the cache manager on a per-index basis.

Set the cache eviction policy

The `eviction_policy` setting in `server.conf` determines the cache eviction policy.

Eviction policy	Description
lru (default)	Evict the least recently used bucket.
lruk	Evict the least recently used bucket, keeping track of the last K references to popular buckets, where K=3.
clock	Evict the bucket with the oldest events first, unless it has been accessed recently.
lrlt	Evict the bucket with the oldest events first.
random	Randomly evict a bucket.
noevict	Don't evict.

If you want to use an eviction_policy other than "lru" or "lruk", consult with Splunk Support first.

Initiate eviction based on occupancy of the cache's disk partition

These settings in `server.conf` initiate eviction based on occupancy of the cache's disk partition:

- The `max_cache_size` setting specifies the maximum occupied space, in megabytes, for the disk partition that contains the cache.
- The `minFreeSpace` setting specifies the minimum free space, in megabytes, for a partition.
- The `eviction_padding` setting controls the amount of additional space, in megabytes, that the cache manager protects, beyond the `minFreeSpace` value.

The minFreeSpace setting is not strictly a cache-specific setting, and therefore it does not reside in the [cachemanager] stanza, but it nevertheless helps determine cache size limits.

When the occupied space on the cache's partition exceeds `max_cache_size`, or the partition's free space falls below (`minFreeSpace` +`eviction_padding`), the cache manager begins to evict data.

Set cache retention periods based on data recency

You can protect recently indexed data from eviction. You can use this capability in two ways:

- On a global level (across all indexes), to favor recently indexed data over recently used data.
- On a per-index level, to favor data in critical indexes over data in non-critical indexes.

To set cache retention periods based on data recency, use the `hotlist_recency_secs` and `hotlist_bloom_filter_recency_hours` settings. These settings serve to override the eviction policy. You can scope these settings globally or on a per-index level.

The hotlist_recency_secs setting

The `hotlist_recency_secs` setting causes the cache manager to protect buckets that contain recent data over other buckets. The setting determines the cache retention period for warm buckets based on age. When eviction is necessary, the cache manager will not evict buckets until they reach the configured age, unless all other buckets have already been evicted.

The cache manager attempts to defer bucket eviction until all data in the bucket is older than the value of the setting. The setting defaults to 86400 seconds, or 24 hours.

To determine a bucket's age, or "recency", the age of the bucket is calculated by subtracting the time of the bucket's most recent event data from the current time. For example, if the current time (expressed in UTC epoch time) is **1567891234** (Sep 7 23:20:34 CEST 2019) and the bucket is named db_**1567809123_1557891234**_10_8A21BEE9-60D4-436B-AA6D-21B68F631A8B (between May 15 05:33:54 CEST 2019 and Sep 7 00:32:03 CEST 2019), thus indicating that the time of the most recent event in the bucket is **1567809123** (Sep 7 00:32:03 CEST 2019), then the bucket's age, in seconds, is 82111 (~23 hours).

Ensure that the cache is of sufficient size to handle the value of this setting. Otherwise, cache eviction cannot function optimally. In other words, do not configure this setting to a size that will cause the cache to retain a quantity of buckets that approach or exceed the size of the cache based on this setting alone. Also, consider the rate of data ingestion and the typical time spans of your searches to determine for how long your recent buckets should remain in cache. As a best practice, start with a fairly low value for this setting and adjust over time. For example, if the cache size is 100 GB and you typically add 10 GB of new buckets to the indexer in a 24 hour period, configuring this setting to 172800 (48 hours) means that the cache manager will try to keep 20 GB of recent buckets in the cache at all times.

The hotlist_bloom_filter_recency_hours setting

The `hotlist_bloom_filter_recency_hours` setting protects certain small metadata files, such as the `bloomfilter` file, from eviction. By inspecting such metadata files, the cache manager can sometimes eliminate the need to fetch larger bucket files, such as the rawdata journal and the `tsidx` files, from remote storage when handling search requests. See The SmartStore cache manager.

The `hotlist_bloom_filter_recency_hours` setting affects the cache retention period for small warm bucket files. The cache manager attempts to defer eviction of the non-journal and non-tsidx bucket files, such as the `bloomfilter` file, until the interval between the bucket's latest time and the current time exceeds this setting. This setting defaults to 360 hours, or 15 days.

The recency of a `bloomfilter` file is based on its bucket's recency and is calculated in the same manner described for `hotlist_recency_secs`.

This setting works in concert with `hotlist_recency_secs`, which is designed to be configured for a shorter age. If `hotlist_recency_secs` leads to the eviction of a bucket, the bucket's `bloomfilter` and associated files will continue to remain in the cache until they reach the age configured with `hotlist_bloom_filter_recency_hours`. Thus, the bucket will remain in cache, but without its journal and `tsidx` files.

Configure recency globally or for individual indexes

When you configure these settings globally, they override the eviction policy, which, by default, favors buckets that have been recently searched. For example, if `hotlist_recency_secs` is set globally to 604800 (7 days), the cache manager will attempt to retain buckets with data that is less than seven days old. It will instead evict older buckets, even if those older buckets were searched more recently. The cache manager will only evict buckets containing data less than seven days old if there are no older buckets to evict.

By configuring the recency settings on a per-index level, you can favor data in critical indexes over data in less critical indexes. Since all SmartStore indexes share the cache and otherwise follow the global cache eviction policy, the per-index recency settings provide the only means to retain data from critical indexes for a longer period than data from less critical indexes.

For example, if you have an index with critical data, such as the ES `threat_activity` index, and another index whose data is less critical, such as the default `_internal` index, you can set `hotlist_recency_secs` to 5184000 (60 days) for `threat_activity`, while keeping the default setting of 86400 (1 day) for `_internal`. By doing so, you cause the cache manager to favor `threat_activity` buckets over `_internal` buckets, thus reducing the likelihood that the cache will need to fetch data from the remote store to handle `threat_activity` searches.

Configure globally for all indexes

To configure the `hotlist_recency_secs` and `hotlist_bloom_filter_recency_hours` settings globally, for all SmartStore indexes, you must set them in the `[cachemanager]` stanza in `server.conf`.

You can override the global settings on a per-index basis.

Configure for individual indexes

To configure the `hotlist_recency_secs` and `hotlist_bloom_filter_recency_hours` settings on a per-index basis, you must set them in each index's stanza in `indexes.conf`.

If you do not configure the settings for a particular SmartStore index, that index inherits the global value from `server.conf`.

Set the maximum download and upload rates

The `max_concurrent_downloads` setting in `server.conf` specifies the maximum number of buckets that can be downloaded simultaneously from remote storage. Its default is 8.

The `max_concurrent_uploads` setting in `server.conf` specifies the maximum number of buckets that can be uploaded simultaneously to remote storage. Its default is 8.

Configure data retention for SmartStore indexes

Data retention policy for SmartStore indexes is configured with settings similar to those for non-SmartStore indexes.

On indexer clusters, data retention for SmartStore indexes is managed cluster-wide. Once a bucket in the index meets the criteria for freezing, the cluster removes the bucket entirely from the system, both from remote storage and from any local caches where copies of it exist.

Because SmartStore indexes do not support the cold bucket state, except for the case of migrated buckets, buckets roll from warm directly to frozen.

For general information on bucket freezing and archiving, see Set a retirement and archiving policy and Archive indexed data. Most of the material in those topics is relevant to all indexes, SmartStore or not. The differences are covered here.

Data retention policy

Use these `indexes.conf` settings to configure data retention policies for a SmartStore index:

- `maxGlobalDataSizeMB`
- `maxGlobalRawDataSizeMB`
- `frozenTimePeriodInSecs`

Bucket freezing occurs whenever any of these limits is reached. This can result in unexpected data loss if the settings are improperly configured. For example, if maxGlobalDataSizeMB is reached before frozenTimePeriodInSecs, buckets will be rolled to frozen before the configured time period has elapsed. If you require data to remain in your system for a specific amount of time, ensure that the other settings won't pre-empt your frozenTimePeriodInSecs setting. When configuring data retention limits, be sure to accommodate any critical retention criteria.

In the case of an indexer cluster, data retention settings, like all `indexes.conf` settings, must be the same for all peer nodes. Use the configuration bundle method to distribute the settings from the manager node to the peer nodes, as described in Update common peer configurations and apps.

These settings, available for non-SmartStore indexes only, have no effect on a SmartStore index:

- `maxTotalDataSizeMB`
- `maxWarmDBCount`

maxGlobalDataSizeMB

The `maxGlobalDataSizeMB` setting specifies the maximum size, in MB, for all warm and cold buckets in a SmartStore index. When the size of an index's set of warm and cold buckets exceeds this value, the system freezes the oldest buckets, until the size again falls below this value.

The total size of an index's warm and cold buckets approximates closely the size that the index occupies in remote storage. Note these aspects of the size calculation:

- It applies on a per-index basis.
- In the case of an indexer cluster, it applies across all peers in the cluster.
- In the case of a standalone indexer, it applies only to that indexer. Standalone indexers, by their nature, each manage their own data retention.

- It includes the sum of the size of all buckets that reside on remote storage, along with any buckets that have recently rolled from hot to warm and are awaiting upload to remote storage.
- It includes only one copy of each bucket. If a duplicate copy of a bucket exists on an indexer, the size calculation does not include it. For example, if the bucket exists on both remote storage and on an indexer's local cache, the calculation ignores the copy on local cache.
- It includes only the size of the buckets themselves. It does not include the size of any associated files, such as report acceleration or data model acceleration summaries.

If the total size of an index's warm and cold buckets exceeds `maxGlobalDataSizeMB`, the oldest bucket in the index is frozen. For example, assume that `maxGlobalDataSizeMB` is set to 5000 for an index, and the index's warm and cold buckets occupy 4800MB. If a 750MB hot bucket then rolls to warm, the index size now exceeds `maxGlobalDataSizeMB`, triggering bucket freezing. The cluster freezes the oldest buckets on the index, until the total warm and cold bucket size falls below `maxGlobalDataSizeMB`.

You set this value under the stanza for the index to which it applies. To specify the same value for all indexes, you can set it at the global stanza level. In that case, however, the value still applies individually to each index. That is, if you set `maxGlobalDataSizeMB` at the global stanza level to 5000MB, then indexA has its own maximum of 5000MB, indexB has its own maximum of 5000MB, and so on.

This setting defaults to 0, which means that it does not limit the amount of space that the warm and cold buckets on an index can occupy.

maxGlobalRawDataSizeMB

The `maxGlobalRawDataSizeMB` setting specifies the maximum size, in MB, of raw data residing in all warm buckets in a SmartStore index. When the size of an index's raw data in the set of warm buckets exceeds this value, the system freezes the oldest buckets, until the size again falls below this value.

Raw data size is the uncompressed size of data, as measured at the time that the indexers ingested it. It is not the size of the compressed rawdata journal files that reside in the buckets.

This setting is useful in cases where you want to retain data based on the amount of raw data originally ingested, rather than based on the age of the data or the size that the data occupies in storage. For example, you might have a requirement that an index retain the last 10TB of ingested raw data. Since the indexing process compresses and indexes ingested data, the size of the indexed data stored on disk could vary markedly from its raw size.

In the case of an indexer cluster, `maxGlobalRawDataSizeMB` is calculated as the total amount of raw data ingested for the index and currently residing in warm or cold buckets, across all peer nodes. Only the amount of ingested data counts in the calculation, so the amount of raw data is not increased by data replication. For example, in a three peer cluster with a replication factor of 3, if peer1 ingests 300MB of raw data, peer2 ingests 400MB, and peer3 ingests 500MB, the total amount of raw data residing on the cluster is 1200MB, not 3600MB.

Note these key aspects of the size calculation:

- It uses the raw data size, that is, the uncompressed size of the data at the time that the indexer ingested it.
- It applies on a per-index basis.
- It applies to warm and cold buckets only. Hot bucket data is ignored
- In the case of an indexer cluster, it applies across all peers in the cluster.

If the total raw ingested size of the data residing in an index's warm buckets exceeds `maxGlobalRawDataSizeMB`, the oldest bucket in the index is frozen. For example, assume that `maxGlobalRawDataSizeMB` is set to 5000 (MB) for an index, and the

index's warm buckets contain 4800MB of raw data. If a hot bucket containing 500MB of raw data then rolls to warm, the amount of raw data in the index now exceeds `maxGlobalRawDataSizeMB`, triggering bucket freezing. The system freezes the oldest buckets on the index, until the total amount of raw data residing in warm buckets falls below `maxGlobalRawDataSizeMB`.

You set this value under the stanza for the index to which it applies. To specify the same value for all indexes, you can set it at the global stanza level. In that case, however, the value still applies individually to each index. That is, if you set `maxGlobalRawDataSizeMB` at the global stanza level to 5000MB, then indexA has its own maximum of 5000MB, indexB has its own maximum of 5000MB, and so on.

This setting defaults to 0, which means that it does not limit the amount of raw data in an index.

The maxGlobalRawDataSizeMBsetting is available only for indexers running version 7.3.0 or later.

frozenTimePeriodInSecs

This setting is the same setting used with non-SmartStore indexes. It specifies that buckets freeze when they reach the configured age. The default value is 188697600 seconds, or approximately 6 years.

For details on this setting, see Freeze data when it grows too old.

The process of freezing buckets on indexer clusters

The process of freezing a SmartStore index's buckets on an indexer cluster proceeds in this fashion:

1. The manager node runs a search every 15 minutes, by default, on all peer nodes, to identify any buckets that need to be frozen.
   The search period is controlled by the remote_storage_retention_period setting in server.conf on the manager.
2. For each bucket to be frozen, the manager randomly assigns the job to one of the peers with a local copy of the bucket; that is, to one of the peers with metadata for the bucket in its index's `.bucketManifest` file.
3. For each bucket to be frozen:
   1. The designated peer checks whether the bucket is present on remote storage. It continues the freezing process for that bucket only if the bucket exists on remote storage. Otherwise, the peer skips the bucket.
      The most likely reason for a warm bucket not existing on remote storage is if the bucket recently rolled from hot to warm.
   2. The next steps taken by the bucket's designated peer vary, depending on whether the cluster peers are configured to archive frozen buckets before deleting them:
      - ◊ If the cluster peers are configured to archive buckets, the designated peer fetches the bucket from remote storage if it is not already in local cache. It then archives the bucket and removes its local copy.
      - ◊ If the peers are configured to remove frozen buckets without first archiving them, the designated peer does not fetch the bucket. It simply removes its local copy.

      For information on how to configure archiving, see Archive indexed data.
   3. The designated peer removes the bucket from the remote store.
   4. The designated peer notifies the manager of the remote store deletion.
   5. The manager tells each other peer with a local copy of the bucket to remove its copy.

In this context, he term "local bucket copy" means all information about the bucket on the peer, optionally including the actual bucket copy. When a peer removes its local bucket copy during the freezing process, it removes the bucket's metadata from the index's .bucketManifest file, as well as any copy of the bucket in its cache. If there is no copy in its cache, then it removes the empty directory for that bucket.

The process of freezing buckets on standalone indexers

The process of freezing a SmartStore index's buckets on a standalone indexer proceeds in this fashion:

1. The indexer checks every 60 seconds, by default, to identify any buckets that need to be frozen.
   The service period is controlled by the rotatePeriodInSecs setting in indexes.conf.
2. For each bucket to be frozen:
   1. The indexer checks whether the bucket is present on remote storage. It continues the freezing process for that bucket only if the bucket exists on remote storage. Otherwise, it skips the bucket.
      The most likely reason for a warm bucket not existing on remote storage is if the bucket recently rolled from hot to warm.
   2. The next steps taken by the indexer vary, depending on whether the indexer is configured to archive frozen buckets before deleting them:
      - If the indexer is configured to archive buckets, the indexer fetches the bucket from remote storage if it is not already in local cache. It then archives the bucket and removes its local copy.
      - If the indexer is configured to remove frozen buckets without first archiving them, the indexer does not fetch the bucket. It simply removes its local copy.

      For information on how to configure archiving, see Archive indexed data.
   3. The indexer removes the bucket from the remote store.

In this context, the term "local bucket copy" means all information about the bucket on the indexer, optionally including the actual bucket copy. When an indexer removes its local bucket copy during the freezing process, it removes the bucket's metadata from the index's .bucketManifest file, as well as any copy of the bucket in its cache. If there is no copy in its cache, then it removes the empty directory for that bucket.

Thawing data and SmartStore

You cannot thaw an archived bucket into a SmartStore index, even if the bucket, prior to freezing, was part of a SmartStore index.

Instead, you can thaw the bucket into the thawed directory of a non-SmartStore index. When thawing a bucket to a non-SmartStore index, you must make sure thatÂ its bucket ID is unique within that index.

If you plan to thaw buckets frequently, you might want to create a set of non-SmartStore indexes that parallel the SmartStore indexes in name. For example, "nonS2_main".

For information on bucket IDs, see Bucket names.

For information on thawing buckets, see Thaw a 4.2+ archive.

Add a SmartStore index

The process of adding a SmartStore index is similar to adding a non-SmartStore index. You create a new index stanza in `indexes.conf` and configure path information and other settings.

You cannot add, edit, or delete a SmartStore index through Splunk Web.

Prequisites

Read:

- Create custom indexes
- Configure SmartStore

Add a SmartStore index by directly editing indexes.conf

You can add a SmartStore index by directly editing `indexes.conf`. You can use this method with both indexer clusters and standalone indexers.

As with all indexes.conf settings, use the configuration bundle method when adding an index to the peer nodes on an indexer cluster.

Note the following:

- SmartStore-related settings in `indexes.conf` are usually configured at the global level, not at the individual index level. In that case, it is unlikely that you will need to specify those settings when adding a new index. However, if you are using multiple remote volumes or if you have a mix of SmartStore and non-SmartStore indexes, you must specify the SmartStore settings at the index level.
- If the SmartStore settings are configured globally, then a new index stanza usually needs only the `homePath`, `thawedPath`, and `coldPath` settings. You must specify values for `coldPath` and `thawedPath`, even though SmartStore does not use those settings.
- Use `maxDataSize = auto` (the default value, which is 750MB).
- In the case of indexer clusters, you must specify `repFactor = auto` .
- See SmartStore settings in indexes.conf for a list of relevant SmartStore settings.

For example, assume that you have already set all SmartStore-related configurations at the global level. Then, to configure a new index called "cs_index", just add the following stanza to `indexes.conf`:

```
[cs_index]
homePath = $SPLUNK_DB/cs_index/db
thawedPath = $SPLUNK_DB/cs_index/thaweddb
coldPath = $SPLUNK_DB/cs_index/colddb
```

Troubleshoot SmartStore

SmartStore, in common with other features of Splunk Enterprise, provides a number of tools that you can use to troubleshoot your deployment:

- Monitoring console

- Log files
- CLI commands
- REST endpoints

This topic discuss each of these tools in the SmartStore troubleshooting context. In addition, it covers some common SmartStore issues and their possible causes.

Troubleshoot with the monitoring console

You can use the monitoring console to monitor most aspects of your deployment. This section discusses the console dashboards that provide insight into SmartStore activity and performance.

The primary documentation for the monitoring console is located in *Monitoring Splunk Enterprise*.

Several dashboards monitor SmartStore status. The dashboards are scoped either to a single instance or to the entire deployment. Find the dashboards under the **Indexing** menu and the **SmartStore** submenu:

- SmartStore Activity: Instance
- SmartStore Activity: Deployment
- SmartStore Cache Performance: Instance
- SmartStore Cache Performance: Deployment

SmartStore Activity dashboards

The SmartStore Activity dashboards provide information on activity related to the remote storage, such as:

- Remote storage connectivity
- Bucket upload/download activity
- Bucket upload/download failure count

The SmartStore Activity dashboards also include check boxes that you can select to show progress if you are currently performing data migration or bootstrapping.

SmartStore Cache Performance dashboards

The SmartStore Cache Performance dashboards provide information on the local caches, such as:

- The values for the `server.conf` settings that affect cache eviction
- The bucket eviction rate
- Portion of search time spent downloading buckets from remote storage
- Cache hits and misses
- Repeat bucket downloads

View the dashboards themselves for more information. In addition, see Indexing:Indexes and volumes in *Monitoring Splunk Enterprise.*

Troubleshoot with log files

Several log files can provide insight into the state of SmartStore operations.

`splunkd.log`. Examine these log channels:

- `S3Client`. Communication with S3.
- `GCSClient`. Communication with GCS.
- `AzureStorageClient`. Communication with Azure Blob storage.
- `StorageInterface`. External storage activity (at a higher level than `S3Client`, `GCSClient`, or `AzureStorageClient`).
- `CacheManager`. Activity of the cache manager component.
- `CacheManagerHandler`. Cache manager REST endpoint activity (both server and client side).
- `KeyProviderManager`. Errors related to key provider setup and configuration. The key provider is used when the system has encrypted data on the remote store.

`search.log` . Examine these log channels:

- `CacheManagerHandler`. Bucket operations with cache manager REST endpoint activity.
- `S2BucketCache`. Search-time bucket management (open, close, and so on).
- `BatchSearch, CursoredSearch, IndexScopedSearch, ISearchOperator`. Search activity related to buckets.

`audit.log`

- Contains information on bucket operations, such as upload, download, evict, and so on.

`metrics.log`

- Contains metrics concerning operations on external storage.

`splunkd_access.log`

- Contains a trail of the search process activity against the cache manager REST endpoint.

Use dbinspect to obtain information about SmartStore buckets

You can search with the `dbinspect` command to obtain information about SmartStore buckets. It is important to understand the effect of the `cached` argument on SmartStore bucket searches.

If the `cached` argument is set to "t", `dbinspect` gets its statistics from the bucket's manifest. If set to "f", `dbinspect` examines the bucket itself.Â

The default for `cached` is "t" for SmartStore indexes and "f" for non-SmartStore indexes. Do not change the default.

Although the `cached` argument bears no direct relationship to the term "cache" used with SmartStore, the effect of the `cached` argument matters significantly for SmartStore buckets, because `dbinspect` with `cached=f` examines an indexer's local copy of the bucket while `dbinspect` with `cached=t` examines instead the bucket's manifest, which contains information about the canonical version of the bucket that resides in the remote store.

To understand the importance of this distinction, consider `sizeOnDiskMB`, which is one of the fields that `dbinspect` returns. If a copy of a SmartStore bucket is no longer in the indexer's local cache, the bucket directory is empty and thus has a size of 0.Â Therefore, if `cached=f`, `dbinspect` inspects the local bucket to determine the value of `sizeOnDiskMB`. Since the local version of the bucket consists only of the empty directory, `dbinspect` returns a size of 0 for the bucket, even though the full bucket exists in remote storage.Â Similarly, a cached copy of a bucket might contain only a subset of the files in the full bucket, and `cached=f` will return only the size of that subset, rather than the full size of the bucket.

However, the true size of the bucket (that is, the size of the canonical copy of the bucket in the remote store) is available through the bucket's manifest.Â So, if `cached=t`, the indexer pulls the bucket's manifest from the copy in the remote store and `dbinspect` then uses the statistics in that manifest, thus returning the true size of the bucket.

For more information on `dbinspect`, see dbinspect in the *Search Reference*.

Test connectivity with remote storage

One common problem is connectivity with the remote storage. Connectivity problems can result from network or permissions issues. Use the `splunkd cmd rfs` command to test connectivity with remote storage. This section demonstrates some uses for the command.

List the contents of the "foobar" index on remote storage:

```
splunk cmd splunkd rfs ls index:foobar
```

List the contents of a given bucket on remote storage:

```
splunk cmd splunkd rfs ls bucket:foo~737~B1CE2AB0-CE4A-4697-83F2-1C5DBFB6485A
```

Test getting a file from remote storage:

```
splunk cmd splunkd rfs getF
bucket:foo~737~B1CE2AB0-CE4A-4697-83F2-1C5DBFB6485A/guidSplunk-B1CE2AB0-CE4A-4697-83F2-1C5DBFB6485A/Hosts.data
 /tmp/foo/
```

Test getting an unencrypted file from remote storage that uses SSE-C:

```
splunk cmd splunkd rfs getR bucket:foo~737~B1CE2AB0-CE4A-4697-83F2-1C5DBFB6485A/receipt.json  /tmp/foo/
```

Troubleshoot with REST searches

Use the following search to get a list of buckets that are actively being searched:

```
| rest /services/admin/cacheman search=cm:bucket.ref_count>0
```

The ref_count value increments by 1 when the search opens the bucket and decrements by 1 when the search closes the bucket.

Use the following search to get a list of buckets that have not been uploaded to the remote store (that is, are not "stable" on the remote store):

```
| rest /services/admin/cacheman search=cm:bucket.stable=0
```

Common issues

Searches are running slowly or appear stuck

Slow or stuck searches are often due to these issues:

- Performance issues with remote storage.
- The cache manager is evicting buckets too aggressively.
- Cold cache issues. A cold cache occurs when an indexer cluster peer node participating in a search does not have a local copy of some needed buckets and therefore must download the buckets from remote storage. A cold cache can result from the manager reassigning primary bucket copies to different peers.

Searches erroring out

The search-related error message "Failed to localize fileSet='....' for bid='...'. Results will be incomplete." indicates an error condition while downloading the specified bucket,

For more details, examine `splunkd.log` on the indexer issuing the error.

Disk full issues

A disk full related message indicates that the cache manager is unable to evict sufficient buckets. These are some possible causes:

- Search load overwhelming local storage. For example, the entire cache might be consumed by buckets opened by at least one search process. When the search ends, this problem should go away.
- Cache manager issues. If the problem persists beyond a search, the cause could be related to the cache manager. Examine `splunkd.log` on the indexer issuing the error.

How SmartStore works

The SmartStore cache manager

Each indexer incorporates a cache manager that manages the SmartStore data in local storage. The cache manager attempts to maintain in local storage any data that is likely to participate in future searches. By caching the search working set, the cache manager minimizes the potential of search delays resulting from data being downloaded from the remote store.

Each indexer's cache manager operates independently from those on other indexers.

The cache manager performs these functions:

- It copies buckets from local to remote storage when the buckets roll from hot to warm.
- It fetches bucket files from remote storage into the local storage cache when a search needs the files.
- It evicts files from the local storage cache when they are no longer likely to be needed for future searches.

How the cache manager handles hot buckets rolling to warm

When a hot bucket rolls to warm, the cache manager uploads a copy of the bucket to the remote storage. Once uploaded to remote storage, the bucket is eligible for eviction from the local cache. This process is described in detail in How indexing works in SmartStore.

Once a bucket is uploaded to remote storage, the files in it do not change, but a few files can be added to it later. For example, the cache manager might upload report acceleration summaries or delete journals as they are created on the indexer.

How the cache manager fetches buckets

The cache manager fetches bucket files from remote storage to fulfill search requests.

> When the cache manager fetches a file from remote storage, it copies the file to the local cache. The master copy of the file remains on remote storage.

The cache manager does not always fetch entire buckets from remote storage. Instead, it can fetch individual bucket files if the search might not need the entire bucket. For example, by fetching the `bloomfilter` file first, the cache manager can, in some cases, avoid the need to fetch the rest of the bucket. Similarly, some searches, such as `metadata` and `tstats`, do not need the rawdata `journal.gz` file. Others, such as `dbinspect` and `eventcount`, do not need any bucket content.

The cache manager attempts to prefetch buckets during a search, so that each bucket is already local by the time the search is ready to access it. When prefetching buckets, the cache manager uses a heuristic that adjusts according to how the `bloomfilter` and `tsidx` files in other buckets eliminate results. Accordingly, the cache manager might prefetch either entire buckets or just individual files within buckets. For example, during a search, if the search on several previous buckets did not require the `tsidx` files, the cache manager does not prefetch `tsidx` files as it continues to process additional buckets.

When prefetching, the cache manager notes how long the search process is taking per bucket, and it prefetches buckets at a speed that allows the search to continue uninterrupted without the need to wait for a bucket fetch to complete. The

cache manager also adjusts its speed to avoid prefetching an excessive number of buckets.

How the cache manager evicts buckets

The cache manager attempts to minimize the use of local storage by retaining those buckets and files that have a high likelihood of participating in a future search and evicting the rest. When the cache manager evicts a bucket, it removes the copy of that bucket residing on the cache. Because cached buckets are local copies of buckets whose master copies reside in remote storage, the eviction process does not result in loss of data.

When a bucket is evicted from the cache, its directory remains in the cache, but the directory is now empty. The `.bucketManifest` file for the bucket's index also retains metadata for the bucket.

The cache manager does not necessarily evict all files in a bucket. It favors evicting large files, such as the rawdata journal and the `tsidx` files, while leaving small files, such as `bloomfilter` and `metadata`, in the cache. By doing so, the cache manager can reduce the need to fetch the large files from remote storage. For example, inspection of a bucket's `bloomfilter` file at the start of a search can sometimes eliminate the need to search the bucket's data, therefore avoiding the need to fetch the bucket's rawdata journal and `tsidx` files. This behavior is configurable through the cache manager recency settings. See Set cache retention periods based on recency.

The cache manager operates according to a configurable policy. The default policy is "lru", which tells the cache manager to evict the least recently used bucket. To learn about the set of eviction policies available, see Set the eviction policy.

Configure the cache manager

See Configure the SmartStore cache manager.

How indexing works in SmartStore

Indexers handle buckets in SmartStore indexes differently from buckets in non-SmartStore indexes.

Bucket states and SmartStore

Indexers maintain buckets for non-SmartStore indexes in these states:

- Hot buckets
- Warm buckets
- Cold buckets

Indexers maintain buckets for SmartStore indexes in these states only:

- Hot buckets
- Warm buckets

The hot buckets of SmartStore indexes reside on local storage, just as with non-SmartStore indexes. Warm buckets reside on remote storage, although copies of those buckets might also reside temporarily in local storage.

The concept of cold buckets goes away, because the need to distinguish between warm and cold buckets no longer exists. With non-SmartStore indexes, the cold bucket state exists as a way to identify older buckets that can be safely moved to some type of cheaper storage, because buckets are typically searched less frequently as they age. But with SmartStore indexes, warm buckets are already on inexpensive storage, so there is no reason to move them to another

type of storage as they age.

Buckets roll to frozen directly from warm.

Cold buckets can, in fact, exist in a SmartStore index, but only under limited circumstances. Specifically, if you migrate an index from non-SmartStore to SmartStore, any migrated cold buckets will use the existing cold path as their cache location, post-migration. In all respects, cold buckets in SmartStore indexes are functionally equivalent to warm buckets. The cache manager manages the migrated cold buckets in the same way that it manages warm buckets. The only difference is that the cold buckets, when needed, will be fetched into the cold path location, rather than the home path location.

The indexing process

The indexing process is the same with SmartStore and non-SmartStore indexes.

The indexer indexes the incoming data and writes the data to hot buckets in local storage. In the case of an indexer cluster, the source peer streams the hot bucket data to target peers to fulfill the replication factor.

When buckets roll to warm, however, the SmartStore process differs from non-SmartStore.

Warm bucket handling

Starting from the point that a bucket rolls from hot to warm, the indexer handles SmartStore indexes differently from non-SmartStore indexes.

When a bucket in a SmartStore index rolls to warm, the bucket is copied to remote storage.

The rolled bucket does not immediately get removed from the indexer's local storage. Rather, it remains cached locally until it is evicted in response to the cache manager's eviction policy. Because searches tend to occur most frequently across recent data, this process helps to minimize the number of buckets that need to be retrieved from remote storage to fulfill a search request.

After the cache manager finally does remove the bucket from the indexer's local storage, the indexer still retains metadata information for that bucket in the index's `.bucketManifest` file. In addition, the indexer retains an empty directory for the bucket.

Note: Under certain circumstances the cache manager retains the `bloomfilter` file, as well as some other small files, when it otherwise evicts a bucket. That is, it deletes the bucket's `rawdata` journal and `tsidx` files but leaves the small files temporarily in place. This behavior is configurable through the cache manager recency settings. See Set cache retention periods based on recency.

In the case of an indexer cluster, when a bucket rolls to warm, the source peer uploads the bucket to remote storage. The source peer continues to retain its bucket copy in local cache until, in due course, the cache manager evicts the copy.

After successfully uploading the bucket, the source peer sends messages to the bucket's target peers, notifying them that the bucket has been uploaded to remote storage. The target peers, like the source peer, continue to retain their bucket copies in local cache until, in due course, their cache managers evict the copies.

During the upload process, if the target peers do not hear from the source peer within five minutes, they query the remote storage to learn whether the bucket was uploaded. If it wasn't uploaded by the source peer, one of the target peers then uploads it.

When a bucket copy does get evicted from a peer's local cache, the peer retains metadata for the bucket, so that the cluster has enough copies of the bucket, in the form of its metadata, to match the replication factor.

In addition to retaining metadata information for the bucket, the source peer continues to retain the primary designation for the bucket. The peer with primary designation fetches the bucket from remote storage when the bucket is needed for a search.

How SmartStore handles report and data model acceleration summaries

SmartStore-enabled indexers handle summaries in a similar way to non-SmartStore-enabled indexers. In an indexer cluster, the summary is created on the peer node that is primary for the associated bucket or buckets. The peer then uploads the summary to remote storage. When a peer needs the summary, its cache manager fetches the summary from remote storage.

Summary replication is unnecessary, and is therefore unsupported, because the uploaded summary is available to all peer nodes.

When using SmartStore, the settings `summaryHomePath` and `tstatsHomePath` must remain unset. See Settings in indexes.conf that are incompatible with SmartStore or otherwise restricted.

For details on report and data model acceleration summaries in indexer clusters, see How indexer clusters handle report and data model acceleration summaries. For general information on report and data model acceleration, see Manage report acceleration and Accelerate data models, respectively, in the Knowledge Manager Manual.

Bucket freezing and SmartStore

See Configure data retention for SmartStore indexes.

How search works in SmartStore

SmartStore is optimized for certain characteristics that are common to the great majority of Splunk platform searches. Specifically, most searches have these characteristics:

- They occur over near-term data. 97% of searches look back 24 hours or less.
- They have spatial and temporal locality. If a search finds an event at a specific time or in a specific log, it's likely that other searches will look for events within a closely similar time range or in that log.

The cache manager favors recently created buckets and recently accessed buckets, attempting to ensure that most of the data that a search is likely to need is available in local cache. Similarly, the cache manager tends to evict buckets that are likely to participate in searches only infrequently.

Most fundamentals of search are the same for SmartStore and non-SmartStore indexes. An indexer searches buckets in response to a request from a search head. The process of searching buckets always occurs in local storage. As the first step in any search, the indexer compiles a list of buckets that it needs to search across.

To understand how search works on a SmartStore index and how it differs from search on a non-SmartStore index, it is necessary to differentiate between these bucket states:

- Hot buckets
- Warm buckets residing in the local cache
- Warm buckets not residing in the local cache

Hot buckets always reside in local storage, whether the bucket is part of a SmartStore index or a non-SmartStore index. Therefore, searches of hot buckets work the same in both environments.

The search process differs, however, when it comes to searches of warm buckets.

When an indexer needs to search a warm bucket for a SmartStore index, it first "opens" that bucket. Later, when it has finished searching the bucket, it "closes" it. The designation of "open" and "closed" is necessary for the proper functioning of the cache eviction process. When a bucket is open, the indexer's cache manager knows that the bucket is participating in a search and is thus not eligible for eviction. Once a bucket is closed, it becomes eligible for eviction.

Once the warm bucket is open, the indexer determines whether the bucket currently resides in the local cache. The next stage of processing depends on whether the bucket is currently in the cache:

- If the bucket resides in the local cache, the indexer searches it as usual. After the bucket has been searched and closed, it becomes eligible for eviction.

- If the bucket does not reside in the local cache, the cache manager must first fetch a copy of it from remote storage and place it in the local cache. The indexer then searches the bucket as usual. After the bucket has been searched and closed, it becomes eligible for eviction.

Note: In some cases, the cache manager only fetches certain bucket files, not the entire bucket. See How the cache manager fetches buckets.

In the case of indexer clusters, primary bucket copies function similarly as with non-SmartStore indexes. With a hot bucket, only the primary copy of the bucket participates in a search. With a warm bucket, only the peer with the primary designation for that bucket fetches the bucket or accesses a cached copy.

Because the primary designation for a bucket rarely switches between peers, warm buckets in cache usually reside on the peer with the primary designation for that bucket. It is unusual for copies of the same warm bucket to reside in caches on multiple peers, or for the bucket's primary designation to reside on a peer without a local copy while another peer already has a local copy of that bucket.

Indexer cluster operations and SmartStore

Indexer clusters treat SmartStore indexes differently from non-SmartStore indexes in some fundamental ways:

- The responsibility for high availability and disaster recovery of SmartStore warm buckets shifts from the cluster to the remote storage service. This shift offers the important advantage that warm bucket data is fully recoverable even if the cluster loses a set of peer nodes that equals or exceeds the replication factor in number.

- The effect of the replication factor on SmartStore warm buckets differs from its effect on non-SmartStore warm buckets. In particular, the cluster uses the replication factor to determine how many copies of SmartStore warm

bucket metadata it maintains. The cluster does not attempt to maintain multiple copies of the warm buckets themselves. In cases of warm bucket fixup, the cluster only needs to replicate the bucket metadata, not the entire contents of the bucket directories.

Replication factor and replicated bucket copies

A cluster treats hot buckets in SmartStore indexes the same way that it treats hot buckets in non-SmartStore indexes. It replicates the hot buckets in local storage across the replication factor number of peer nodes.

When a bucket in a SmartStore index rolls to warm and moves to remote storage, the remote storage service takes over responsibility for maintaining high availability of that bucket. The replication factor has no effect on how the remote storage service achieves that goal.

At the point that the bucket rolls to warm and gets uploaded to remote storage, the peer nodes no longer attempt to maintain replication factor number of local copies of the bucket. The peers do keep their copies of the bucket in local cache for some period of time, as determined by the cache eviction policy.

However, even when copies of a bucket are no longer stored locally, the replication factor still controls the metadata that the cluster stores for that bucket. Peer nodes equal in number to the replication factor maintain metadata information about the bucket in their `.bucketManifest` files. Those peer nodes also maintain empty directories for the bucket, if a copy of the bucket does not currently reside in their local cache.

For example, if the cluster has a replication factor of 3, three peer nodes continue to maintain metadata information, along with populated or empty directories, for each bucket.

By maintaining metadata for each bucket on the replication factor number of peer nodes, the cluster simplifies the process of fetching the bucket when it is needed, in the case of any interim peer node failure.

Unlike non-SmartStore indexes, a cluster can recover most of the data in SmartStore indexes if it loses peer nodes that equal or exceed the replication factor in number. In such a circumstance, the cluster can recover all of the SmartStore warm buckets, because those buckets are stored remotely and so are unaffected by peer node failure. The cluster will likely lose some of its SmartStore hot buckets, because those buckets are stored locally.

For example, in a cluster with a replication factor of 3, the index loses both hot and warm data from its non-SmartStore indexes if three or more peer nodes are simultaneously offline. However, the same cluster can lose any number of peer nodes, even all of its peer nodes temporarily, and still not lose any SmartStore warm data, because that data resides on remote storage.

Search factor, searchable copies, and primary copies

The search factor has the same effect on hot buckets in SmartStore indexes as it does on hot buckets in non-SmartStore indexes. That is, the search factor determines the number of copies of each replicated bucket that include the `tsidx` files and are thus searchable.

For SmartStore warm buckets, the search factor has no practical meaning. The remote storage holds the master copy of each bucket, and that copy always includes the set of `tsidx` files, so it is, by definition, searchable. When, in response to a search request, a peer node's cache manager fetches a copy of a bucket to the node's local cache, that copy is searchable to the degree that it needs to be for the specific search. As described in How the cache manager fetches buckets, the cache manager attempts to download only the bucket files needed for a particular search. Therefore, in some cases, the cache manager might not download the `tsidx` files.

Primary tags work the same with SmartStore indexes as with non-SmartStore indexes. Each bucket has exactly one peer node with a primary tag for that bucket. For each search, the peer node with the primary tag for a particular warm bucket is the one that searches that bucket, first fetching a copy of that bucket from remote storage when necessary

How an indexer cluster handles SmartStore bucket fixup stemming from peer node failure

When a peer node fails, indexer clusters handle bucket fixup for SmartStore indexes in basically the same way as for non-SmartStore indexes, with a few differences. For a general discussion of peer node failure and the bucket-fixing activities that ensue, see What happens when a peer node goes down.

This section covers the differences in bucket fixup that occur with SmartStore. One advantage of SmartStore is that loss of peer nodes equal to, or in excess of, the replication factor in number does not result in loss of warm bucket data. In addition, warm bucket fixup proceeds much faster for SmartStore indexes.

Bucket fixup with SmartStore

In the case of hot buckets, bucket fixup for SmartStore indexes proceeds the same way as for non-SmartStore indexes.

In the case of warm buckets, bucket fixup for SmartStore indexes proceeds much more quickly than it does for non-SmartStore indexes. This advantage occurs because SmartStore bucket fixup requires updates only to the `.bucketManifest` files on each peer node, without the need to stream the buckets themselves. In other words, bucket fixup replicates only bucket metadata.

Each peer node maintains a `.bucketManifest` file for each of its indexes. The file contains metadata for each bucket copy that the peer node maintains. When you replicate a bucket from a source peer to a target peer, the target peer adds metadata to its `.bucketManifest` file for that bucket copy.

The buckets themselves are not replicated during SmartStore fixup because the master copy of each bucket remains present on remote storage and is downloaded to the peer nodes' local storage only when needed for a search.

During SmartStore replication, the target peer nodes also create an empty directory for each warm bucket that has metadata in their .bucketManifest files.

As with non-SmartStore indexes, the SmartStore bucket fixup process also ensures that exactly one peer node has a primary tag for each bucket.

Fixup when the number of nodes that fail is less than the replication factor

For both SmartStore and non-SmartStore indexes, the cluster can fully recover its valid and complete states through the fixup process. The fixup process for SmartStore warm buckets requires only the replication of metadata internal to the cluster, so it proceeds much more quickly.

Fixup when the number of nodes that fail equals or exceeds the replication factor

In contrast to non-SmartStore indexes, a cluster can recover all the warm bucket data for its SmartStore indexes even when the number of failed nodes equals or exceeds the replication factor. As with non-SmartStore indexes, though, there will likely be some data loss associated with hot buckets.

The cluster needs to recover only missing warm bucket metadata because the warm buckets themselves remain present on remote storage.

To recover any lost warm bucket metadata, the manager node uses its comprehensive list of all bucket IDs. It compares those IDs with the IDs in the set of `.bucketManifest` files on the peer nodes, looking for IDs that are present in its list but are not present in any `.bucketManifest` file. For all such bucket IDs, the manager assigns peer nodes to query the remote storage for the information necessary to populate metadata for those buckets in their `.bucketManifest` files. Additional fixup occurs to meet the replication factor requirements for the metadata and to assign primary tags.

If the manager node goes down during this recovery process, the warm bucket metadata is still recoverable. When the cluster regains a manager node, the manager initiates bootstrapping to recover all warm bucket metadata. For information on bootstrapping, see Bootstrap SmartStore indexes.

Troubleshoot indexers and clusters of indexers

Non-clustered bucket issues

This section tells you how to deal with an assortment of bucket problems that can exist independent of clustering.

Rebuild all buckets

The indexer usually handles crash recovery without your intervention. If an indexer goes down unexpectedly, some recently received data might not be searchable. When you restart the indexer, it will automatically run the `fsck` command in the background. This command diagnoses the health of your buckets and rebuilds search data as necessary.

Caution: It is unlikely that you will need to run `fsck` manually. This is a good thing, because to run it manually, you must stop the indexer, and the command can take several hours to complete if your indexes are large. During that time your data will be inaccessible. However, if Splunk Support directs you to run it, the rest of this section tells you how to do so.

To run `fsck` manually, you must first stop the indexer. Then run `fsck` against the affected buckets. To run `fsck` against buckets in all indexes, use this command:

```
splunk fsck repair --all-buckets-all-indexes
```

This will rebuild all types of buckets (hot/warm/cold) in all indexes.

To rebuild all buckets in just a single index, use this version of the command:

```
splunk fsck repair --all-buckets-one-index --index-name=<your_index>
```

Note: The `fsck` command only rebuilds buckets created by version 4.2 or later of Splunk Enterprise.

The `fsck repair` command can take several hours to run, depending on the size of your indexes If you determine that you only need to rebuild a few buckets, you can run the `rebuild` command on just those buckets, as described in the next section, Rebuild a single bucket.

If you just want to diagnose the state of your indexes (without taking any immediate remedial action), run:

```
splunk fsck scan --all-buckets-all-indexes
```

To learn more about the `fsck` command, including a list of all options available, enter:

```
splunk fsck --help
```

Rebuild a single bucket

If the index and metadata files in a bucket (version 4.2 and later) somehow get corrupted, you can rebuild the bucket from the raw data file alone. Use this command:

```
splunk rebuild <bucket directory> <index-name>
```

The indexer automatically deletes the old index and metadata files and rebuilds them. You don't need to delete any files yourself.

Note:

- Rebuilding a bucket does not count against your license.
- The time required to rebuild a bucket can be significant. Depending on various system considerations, such as your hardware specifications, it can take anywhere from half an hour to a few hours to rebuild a 10 GB bucket.
- `splunk rebuild` is an alias of `splunk fsck repair --one-bucket`.

Recover invalid pre-4.2 hot buckets

A hot bucket becomes an invalid hot (`invalid_hot_<ID>`) bucket when the indexer detects that the metadata files (`Sources.data, Hosts.data, SourceTypes.data`) are corrupt or incorrect. Incorrect data usually signifies incorrect time ranges; it can also mean that event counts are incorrect.

The indexer ignores invalid hot buckets. Data does not get added to such buckets, and they cannot be searched. Invalid buckets also do not count when determining bucket limit values such as `maxTotalDataSizeMB`. This means that invalid buckets do not negatively affect the flow of data through the system, but it also means that they can result in disk storage that exceeds the configured maximum value.

To recover an invalid hot bucket, use the `recover-metadata` command:

1. Make backup copies of the metadata files, `Sources.data, Hosts.data, SourceTypes.data`.

2. Rebuild the metadata from the raw data information:

```
splunk cmd recover-metadata path_to_your_hot_buckets/invalid_hot_<ID>
```

3. If successful, rename the bucket as it would normally be named.

Rebuild index-level bucket manifests

The index-level bucket manifest file is `.bucketManifest`. It contains a list of all buckets in the index.

It is unusual to need to rebuild the manifest. One situation where you might need to do so is if you manually copy a bucket into an index.

> Only rebuild the manifest if Splunk Support directs you to. Do not rebuild it on your own.

This command rebuilds the `.bucketManifest` file for the main index only:

```
splunk _internal call /data/indexes/main/rebuild-bucket-manifest
```

To rebuild the manifests for all indexes, use the asterisk (*) wildcard:

```
splunk _internal call /data/indexes/*/rebuild-bucket-manifest
```

Bucket replication issues

Network issues impede bucket replication

If there are problems with the connection between peer nodes such that a source peer is unable to replicate a hot bucket to a target peer, the source peer rolls the hot bucket and start a new hot bucket. If it still has problems connecting with the

target peer, it rolls the new hot bucket, and so on.

To prevent a situation from arising where a prolonged failure causes the source peer to generate a large quantity of small hot buckets, the source peer, after a configurable number of replication errors to a single target peer, stops rolling hot buckets in response to the connection problem with that target peer. The default is three replication errors. The following banner message then appears one or more times in the manager node's dashboard, depending on the number of source peers encountering errors:

```
Search peer <search peer> has the following message: Too many streaming errors to target=<target
peer>. Not rolling hot buckets on further errors to this target. (This condition might exist with
other targets too. Please check the logs.)
```

While the network problem persists, there might not be replication factor number of copies available for the most recent hot buckets.

If a particular peer is consistently being reported by other peer nodes as being the cause of replication errors, you can temporarily put the peer in manual detention. Once the root problem has been resolved, remove the peer from manual detention. See Put a peer into detention.

Configure the allowable number of replication errors

To adjust the allowable number of replication errors, you can configure the `max_replication_errors` attribute in `server.conf` on the source peer. However, it is unlikely that you will need to change the attribute from its default of 3, because replication errors that can be attributed to a single network problem are bunched together and only count as one error. The "Too many streaming errors" banner message might still appear, but it can be ignored.

Note: The bunching of replication errors is a change introduced in release 6.0. With this change, the number of errors will be unlikely to exceed the default value of 3, except in unusual conditions.

Evidence of replication failure on the source peer

Evidence of replication failure appears in the source peer's `splunkd.log`, with a reference to the failed target peer(s). You can locate the relevant lines in the log by searching on "CMStreamingErrorJob". For example, this `grep` command finds that there have been 15 streaming errors to the peer with the GUID "B3D35EF4-4BC8-4D69-89F9-3FACEDC3F46E":

```
grep CMStreamingErrorJob ../var/log/splunk/splunkd.log* | cut -d' ' -f10 | sort |uniq -c | sort -nr
15 failingGuid=B3D35EF4-4BC8-4D69-89F9-3FACEDC3F46E
```

Unable to disable and re-enable a peer

When you disable an indexer as a peer, any hot buckets that were on the peer at the time it was disabled are rolled to warm and named using the standalone bucket convention. If you later re-enable the peer, a problem arises because the manager remembers those buckets as clustered and expects them to be named according to the clustered bucket convention, but instead they are named using the convention for standalone buckets. Because of this naming discrepancy, the peer cannot rejoin the cluster.

To work around this issue, you must clean the buckets or otherwise remove the standalone buckets on the peer before re-enabling it.

Multisite cluster does not meet its replication or search factors

The symptom is a message that the multisite cluster does not meet its replication or search factors. This message can appear, for example, on the manager node dashboard. This condition occurs immediately after bringing up a multisite cluster.

Compare the values for the single-site `replication_factor` and `search_factor` attributes to the number of peers that you have on each site. (If you did not explicitly set the single-site replication and search factors, then they default to 3 and 2, respectively.) These attribute values cannot exceed the number of peers on any site. If either exceeds the number of peers on the smallest site, change it to the number of peers on the smallest site. For example, if the number peers on the smallest site is 2, and you are using the default values of `replication_factor=3` and `search_factor=2`, you must explicitly change the `replication_factor` to 2.

This condition can occur after you convert a single-site cluster to multisite. If you configure the cluster as multisite from the very beginning, before you first start it up, the issue does not occur.

Anomalous bucket issues

Anomalous buckets are buckets that remain in the fixup state indefinitely, without making any progress. Such buckets can indicate or cause a larger problem with your system. An anomalous bucket, for example, can prevent the cluster from meeting its replication and search factors.

The Bucket Status dashboard lets you identify anomalous buckets. It also allows you to take actions on those buckets that can often fix them. Specifically, you can:

- Get details on a bucket.
- Roll the bucket from hot to warm.
- Resync the state of a bucket copy between a peer and the manager.
- Delete a copy of the bucket on a single peer, or delete all copies of the bucket across all peers.

Consult with Splunk Support before performing these actions on a bucket. Some of the actions, performed without full understanding, can lead to further problems with your system or even to irreversible data loss.

Identify anomalous buckets

To identify anomalous buckets and to take action on them, use the Bucket Status dashboard.

1. From the manager node dashboard, go to the Bucket Status dashboard. See View the bucket status dashboard.
2. Click the **Fixup Tasks - Pending** tab.

You can filter the list of pending buckets by fixup type and by the amount of time that they have been waiting for fixup. If a bucket has been waiting an unusual amount of time for fixup, it might be the cause of problems.

Take action on an anomalous bucket

For buckets that have been stuck in fixup for long periods of time, you can take remedial action.

1. Click **Action** for the bucket that you want to manage.

2. Select one of the available actions:
    - ♦ View bucket details
    - ♦ Roll
    - ♦ Resync
    - ♦ Delete Copy

   A pop-up window appears to guide you through the selected action.

Use the following sequence when performing actions on anomalous bucket.

1. View bucket details
2. Roll
3. Resync
4. Delete Copy

Only perform the next action if the previous one does not resolve the issue.

View bucket details

This pop-up window provides details on the bucket such as:

- The bucket size
- Whether it is frozen
- Whether it has been force-rolled
- Whether it is a standalone bucket
- The peers on which it resides

These details can help to narrow down the cause of the bucket problem and what action to take to remediate it.

Roll

This action rolls the bucket from the hot state to the warm state. It has an effect only on hot buckets.

Resync

The manager holds information about each copy of a bucket. However, in some cases, the manager can have incorrect information about the copy on a particular peer. This condition can occur when communication problems arise between the manager and a peer.

Here are some examples of bucket copy state information that can be out of sync between peer and manager:

- Whether the copy is searchable
- Whether the copy is hot or warm
- Whether the copy is primary
- Whether the copy exists on that peer

The peer knows the state of its bucket copies, so if the peer and the manager have different state information for a bucket copy, the information on the manager is incorrect.

To resolve this problem, resync the bucket copy's state on the manager. When you resync a bucket, you specify the peer with the copy that you need to resync. The resync process causes the peer to send the manager its current information about the bucket copy.

Delete Copy

You can delete either a single copy of a bucket on a specific peer, or all copies of a bucket across the entire cluster.

If deleting a single copy causes the cluster to lose its complete state, the cluster will engage in fixup activities so that the bucket again meets both the search factor and the replication factor. This situation might result in another copy of the bucket appearing on the same peer. If, however, the specified bucket is frozen, the cluster does not attempt any fixup activities.

Performing the delete action on all copies of a bucket across the cluster results in irreversible data loss.

Configuration bundle issues

This topic describes problems that can arise when pushing a configuration bundle from the manager to the peer nodes.

Bundle validation failure when pushing a very large bundle

If you attempt to push a very large configuration bundle (>200MB), bundle validation might fail due to various timeouts. To remediate, you can adjust the number of peers that the manager simultaneously pushes the bundle to.

By default, the manager pushes the bundle to a maximum of 5 peers simultaneously. The `max_peers_to_download_bundle` setting in `server.conf` provides a means to further limit the number of peers that receive the bundle simultaneously.

For example if you set `max_peers_to_download_bundle = 3`, the manager pushes the bundle to three peers at a time. When one peer finishes the download, the manager pushes the bundle to another peer, and so on, until all peers have received the bundle.

For details, see the server.conf specification.

Bundle validation remains in progress and does not complete

If the bundle validation process remains in progress on the manager, and no validation response appears, you must cancel and reset the the bundle push operation to escape the in-progress state.

To cancel and reset the bundle push operation, hit the following endpoint on the manager: cluster/manager/control/default/cancel_bundle_push.

Archive data with Hadoop Data Roll

About archiving indexes with Hadoop Data Roll

Hadoop Data Roll is included with your Splunk license. Hadoop Data Roll provides a user-friendly way for you to copy warm, cold, and frozen index data as archived data. Archive Splunk indexed data into HDFS or S3 so that you can:

- Search archived data that is no longer available in Splunk.
- Search across archived buckets and indexes.
- Perform batch processing analysis for archived data.
- Archive indexer data to meet your data retention policies without using valuable indexer space.

Hadoop Data Roll is not supported for Windows

Setting it up

To configure archiving, you tell Splunk Enterprise:

- Which indexes to archive.
- Where to put the archived data in HDFS or S3.
- At what age buckets should be copied to the archive in HDFS.

There are two ways to configure the above information:

- By editing `indexes.conf`
- In Splunk Web

System requirements

Make sure you have access to at least one Hadoop cluster (with data in it) and the ability to run MapReduce jobs on that data.

Make sure you have Java Development Kit (JDK) 1.6 and above. However, for best results, upgrade to a version higher than JDK 1.6. The following distributions and versions were certified using JDK 1.8.

Hadoop Data Roll is supported on the following Hadoop distributions and versions:

- Apache Hadoop 3.2.1
- Open Apache 3.1.2
- Cloudera Distribution including Apache Hadoop v6.3
- Hortonworks Data Platform (HDP) 3.1.4
- MapR 6.1

What you need on your Hadoop nodes

On Hadoop TaskTracker nodes you need a directory on the *nix file system running your Hadoop nodes that meets the following requirements:

- One gigabyte of free disk space for a copy of Splunk.
- 5-10GB of free disk space for temporary storage. This storage is used by the search processes.

What you need on your Hadoop file system

On your Hadoop file system (HDFS or otherwise) you will need:

- A subdirectory under `jobtracker.staging.root.dir` (usually /user/) with the name of the user account under which Splunk Analytics for Hadoop is running on the search head. For example, if Splunk Analytics for Hadoop is started by user "BigDataUser" and `jobtracker.staging.root.dir=/user/` you need a directory `/user/HadoopAnalytics` that is accessible by user "BigDataUser".

- A subdirectory under the above directory that can be used by this server for intermediate storage, such as `/user/hadoopanalytics/server01/`

Searching archived indexes

You can search archived buckets as you normally search, simply include the archive virtual index in your searches. See Search archived index data for information about search commands that work with indexes stored in Hadoop.

You can for example, create one search that searches Splunk for:

- Data in a Splunk Enterprise index.
- Archived data copied into HDFS or S3.

Search performance

When you search archives, Splunk Enterprise performs batch searches on archived data, which is usually much slower than searches of indexed data. Since Splunk deletes cold data based on your `indexes.conf` settings, archived could also still be present in Splunk Enterprise indexes. It is important to be familiar with your archive and Splunk indexer retention policies and settings so that if you are looking for specific data that is still in Splunk, you can run more efficient searches.

To improve search time when searching archives, you can use dates to limit the buckets that are searched. The storage path in Splunk Indexer data includes the earliest time and the latest time of the buckets. So when you search within a certain time, Splunk is able to use that information to narrow searches to relevant buckets, rather than searching through the entire archived index.

How Hadoop Data Roll works

Beginning with version 8.3.0, Hadoop Data Roll works with buckets with journalCompression set to zstd.

After you configure an index as an Archive, a number of processes work to move aged data into archived indexes:

1. A saved search `| archivebuckets` automatically runs once an hour on the search head. This is a custom command packaged with the `archiver` which is implemented as the Python script `archivebuckets.py`.

2. `archivebuckets` queries the local REST endpoints to discover which indexes should be archived, and where to archive the indexes.

3. `archivebuckets` copies the Hadoop Data Roll jars into its own app directory, then launches distributed searches for each provider for the indexes to be archived.

The search used in this step is `| copybuckets`, which is a custom command automatically implemented by `copybuckets.py`.

4. The information for the index and its provider is fed to the search.

5. For each indexer, Splunk Enterprise copies the knowledge bundle needed to run the search.

6. On the indexer, `copybuckets` launches a Java process, with the same entry point (the SplunkMR class) used for Splunk Analytics for Hadoop searches.

7. Splunk Enterprise passes the info about providers and indexes to the Java process using `stdin`.

8. When the Java process sends events back to Splunk Enterprise, it writes them to `stdout`, and the custom search command (`copybuckets.py`) writes them back to the search process using `stdout`.

9. The Java process logs these actions to the `splunk_archiver.log` file.

10. The Java process checks all buckets in the designated indexes. If the buckets are ready to be archived, the process determines whether the buckets already exist in the archive. It accesses the archive using the provider information.

11. If the bucket has not yet been archived, the bucket is copied to a temporary directory at the archive. Once the bucket it is completely copied, and a receipt file added, it is moved to the correct folder in the archive.

12. If the bucket is previously archived, any new data that has reached it's archived date is copied into that bucket.

13. Archived buckets are ready to be searched in Splunk Web.

About the Hadoop Data Roll processes

Two search commands are defined to correspond with python scripts:

```
archivebuckets -> archivebuckets.py
copybuckets -> copybuckets.py
```

The implementation of the Hadoop Data Roll process uses the following processes:

process action	process name	notes
search process on the Search Head/Search Scheduler	archivebuckets	This is the search activity, including scheduling, that occurs on the Search Head
Python process on the Search Head	archivebuckets.py	
Search process on an indexer	copybuckets <JSON describing indexes>	This is all of the Search Activity that occurs in the Indexer.
Python process on an indexer	copybuckets.py	
	Hunk Java code	This process ties the other processes together, and does the following:

process action	process name	notes
Java Virtual Machine process on an indexer		1. Writes files to HDFS 2. logs information to `$SPLUNK_HOME/var/log/splunk/splunk_archiver.log` 3. Writes events to `stdout`, which is piped back to the Splunk search process `| copybuckets <JSON describing indexes>` 4. Information written in the Splunk Search process becomes an event returned by the search, and you can see these events in Splunk Enterprise with the search command `| archivebuckets forcerun=1`.

How the processes work together

The Hadoop Data Roll search framework strings these processes together and pipes them as follows:

1. `stdout` of the python process on an indexer `copybuckets.py` to `stdin` for the search process on the indexer `archivebuckets`.

2. `stdout` of `| copybuckets <JSON describing indexes>` to `stdin` to python process on the Search Head `archivebuckets.py`.

3. `stdout` of archivebuckets.py into search scheduler for the search process on the Search Head `archivebuckets`.

4. The python process on the Search Head `copybuckets.py` script pipes `stdout` of Hadoop Data Roll Java code (Java Virtual Machine process on an indexer) to `stdout` to the search process on the indexer `copybuckets <JSON describing indexes>`.

At the end of this process, anything that the Hadoop Data Roll Java code (JVM process on an indexer) writes to `stdout` becomes events returned from the search scheduler `| archivebuckets`.

Finalizing or aborting archiving process

When Hadoop Data Roll pauses or finalizes a search, this information must be passed to downstream processes.

For example, if the search process on an indexer shuts down, the search could kill the child process, which then prevents the indexer Python process from shutting shut down gracefully. If the Python process is using a shared resource such as a database connection, or an output stream to HDFS, this could cause failure and possible loss of data.

To resolve this, the search process lets the child process decide what to do should the search process suspend or shut down. If the search process on an indexer is paused, it stops reading from its pipe to the Python process on an indexer. When this happens, the Python process on the indexer can no longer write to the pipe once the buffer fills up.

The Python process is able to determine that the search process on an indexer still exists, but is paused.

If the search process on an indexer is stopped/finalized, it shuts down and the pipe to the Python search process is broken. This is how the Splunk custom search commands know that the upstream search has stopped running. This occurs whether the search is shut down cleanly due to user action, or shut down violently due to an upstream crash.

If the archiving Java process in the indexer finds a broken pipe to the indexer search process, it logs that information, but continues to finish archiving until the buffer is full. if this is not desired, simply kill the Java process.

Add or edit an HDFS provider in Splunk Web

You can set up multiple providers with multiple indexes for one provider. Have the following information at hand:

- The host name and port for the NameNode of the Hadoop cluster.
- The host name and port for the JobTracker of the Hadoop cluster.
- Installation directories of Hadoop command line libraries and Java installation.
- Path to a writable directory on the DataNode/TaskTracker *nix filesystem, the one for which the Hadoop user account has read and write permission.
- Path to a writable directory in HDFS that can be used exclusively by Splunk on this search head.

You can also add HDFS providers by editing indexes.conf.

Add a provider

1. In the top menu, select **Settings > Virtual Indexes**.

2. Select the **Providers** tab and click **New Provider** or the name of the provider you want to edit.

3. The Add New/Edit Provider page, give your provider a **Name**.

4. Select the **Provider Family** in the drop down list (note that this field cannot be edited).

5. Provide the following **Environment Variables**:

   - **Java Home**: provide the path to your Java instance.

   - **Hadoop Home**: Provide the path to your Hadoop client directory.

6. Provide the following **Hadoop Cluster Information:**

   - Hadoop Version: Specify which version of Hadoop the cluster is running one of: Hadoop 1.0, Hadoop 2.0 with MRv1 or Hadoop 2.0 with Yarn.

   - **JobTracker:** Provide the path to the Job Tracker.

   - **File System:** Provide the path to the default file system.

7. Provide the following Settings:

   - **HDFS working directory:** This is a path in HDFS (or whatever the default file system is) that you want to use as a working directory.

   - **Job queue:** This is job queue where you want the MapReduce jobs for this provider to be submitted to.

8. Click **Add Secure Cluster** to configure security for the cluster and provide your Kerberos Server configuration.

9. The **Additional Settings** fields specify your provider configuration variables. Hadoop Data Roll populates these preset configuration variables for each provider you create. You can leave the preset variables in place or edit them as needed. If you want to learn more about these settings, see Provider Configuration Variables in the reference section of this manual.

Note: If you are configuring Splunk Analytics for Hadoop to work with YARN, you must add new settings. See "Required configuration variables for YARN" in this manual.

9. Click **Save**.

Configure Splunk index archiving to Hadoop using the configuration files

Before you begin, note the following:

- You must configure a Hadoop provider.
- Splunk must be installed using the same user for all indexers and Splunk Enterprise instances. This is the user which connects to HDFS for archiving and the user and user permissions must be consistent.
- The data in the referring Index must be in warm, cold, or frozen buckets only.
- The Hadoop client libraries must be in the same location on each indexer. Likewise, the Java Runtime Environment must be installed in the same location on each indexer. See System and software requirements for updated information about the required versions.
- The Splunk user associated with the Splunk indexer must have permission to write to the HDFS node.
- Splunk cannot currently archive buckets with raw data larger than 5GB to S3. You can configure your Splunk Enterprise bucket sizes in `indexes.conf`. See Archiving Splunk indexes to S3 in this manual for known issues when archiving to S3.

Set bundle deletion parameters

Use the following attribute to specify how many bundles may accrue before Splunk Enterprise deletes them:

```
vix.splunk.setup.bundle.reap.limit = 5
```

The default value is 5, which means that when there are more than five bundles, Splunk Enterprise will delete the oldest one.

Configure index archiving in the configuration file

In `indexes.conf`, configure the following stanza:

```
[splunk_index_archive]
vix.output.buckets.from.indexes
vix.output.buckets.older.than
vix.output.buckets.path
vix.provider
```

Where:

- `vix.output.buckets.from.indexes` is the exact name of the Splunk index you want to copy into an archive. For example: "splunk_index." You can list multiple Splunk indexes separated by commas.

- `vix.output.buckets.older.than` is the age at which bucket data in the Splunk Index should be archived. For example, if you specify 432000 seconds (5 days), data will be copied into the archive when it is five days old. Note that Splunk does delete data after a while, based on index settings, so make sure that this setting copies the Splunk data before it is deleted in the Splunk Enterprise indexer.

- `vix.output.buckets.path` is the directory in HDFS where the archive bucket should be stored. For example: "/user/root/archive/splunk_index_archive". If you are using S3, you should prefix this value with `s3n://<s3-bucket>/` and add the additional attributes from the code example below.

- `vix.provider` is the virtual index provider for the new archive.

For S3 directories you must prefix `vix.output.buckets.path` with `s3n://<s3-bucket>/` and then add the following additional attributes to the provider stanza:

```
vix.fs.s3n.awsAccessKeyId = <your aws access key ID>
vix.fs.s3n.awsSecretAccessKey = <your aws secret access key>
```

Limit the bandwidth used for archiving

You can set bandwidth throttling to limit the transfer rate of your archives.

You set throttling for a provider, that limit is then applied across all archives assigned to that provider. To configure throttling, add the following attribute under the virtual index provider stanza you want to throttle.

```
vix.output.buckets.max.network.bandwidth = <bandwidth in bits/second>
```

For more about configuring a provider in `indexes.conf` see Add or edit a provider in Splunk Web.

Archive Splunk indexes to Hadoop in Splunk Web

Before you begin, note the following:

- You must configure a Hadoop provider.
- Splunk must be installed using the same user for all indexers and Splunk Enterprise instances. This is the user which connects to HDFS for archiving and the user and user permissions must be consistent.
- The data in the referring Index must be in warm, cold, or frozen buckets only.
- The Hadoop client libraries must be in the same location on each indexer. Likewise, the Java Runtime Environment must be installed in the same location on each indexer. See System and software requirements for updated information about the required versions.
- The Splunk user associated with the Splunk indexer must have permission to write to the HDFS node.
- Splunk cannot currently archive buckets with raw data larger than 5GB to S3. You can configure your Splunk Enterprise bucket sizes in `indexes.conf`. See Archiving Splunk indexes to S3 in this manual for known issues when archiving to S3.

Configure index archiving with the user interface

1. Navigate to **Settings > Virtual Indexes** and select the **Archived Indexes** tab. You can edit any existing archived index by clicking the arrow to its left.

2. Click **New Archived Indexes** to archive another index.

3. Type the names of the indexes you want to archive. You can add multiple indexes. Indexes that are already archived are disabled in the drop down list.

4. Provide a suffix for the new archive indexes. For example, if you select the "_archive" suffix, the new archived index will be "indexname_archive".

5. Select the Hadoop **Provider** that the new archived indexes will be assigned to.

Note you can determine the bandwidth by provider that these archives can use. See "Set bandwidth limits for archiving" in this topic.

6. For **Destination path in HDFS**, provide the path to the working directory your provider should use for this data. For example: `/user/root/archive/splunk_index_archive`. If you are copying data to S3, prefix this path with: `s3n://<s3-bucket>/`

7. Determine the age of the data that is copied to the archived index. For example, if you select "5 Days," data is copied from the warm, cold, or frozen bucket in the indexer to the archive bucket when it is five days old. **Note:** Splunk deletes data after a period of time defined in your indexer settings, so make sure that this field is set to copy the buckets before they are deleted.

Set bandwidth limits for archiving

If you have concerns about the bandwidth required for consistent archiving, you can set bandwidth throttling. When you set throttling for a provider, the limit you set for your provider is then applied across all indexes assigned to that provider.

Note: We currently cannot guarantee bandwitdh limits for bucket archival to S3 file systems.

To set bandwidth throttling:

1. In the Archived Indexes tab, click on **Max bandwidth (Provider)** for the index you want to edit. This opens the Edit Provider page for that index.

2. Under "Archive Settings" check **Enable Archive Bandwidth Throttling**.

3. Enter the maximum bandwidth you want to allow for all archived indexes associated with the provider.

4. Click **Save**.

Archive Splunk indexes to Hadoop on S3

For best performance and to avoid bucket size limits, you should use the S3A filesystem that was introduced in Apache Hadoop 2.6.0. Configuration for different Hadoop distribution may differ.

Configure archiving to Amazon S3

To enable Splunk Enterprise to archive Splunk data to S3:

1. Add the following configuration to your providers:

```
vix.env.HADOOP_HOME: /absolute/path/to/apache/hadoop-2.6.0
vix.fs.s3a.access.key: <AWS access key>
vix.fs.s3a.secret.key: <AWS secret key>
vix.env.HADOOP_TOOLS: $HADOOP_HOME/share/hadoop/tools/lib
vix.splunk.jars:
$HADOOP_TOOLS/hadoop-aws-2.6.0.jar,$HADOOP_TOOLS/aws-java-sdk-1.7.4.jar,$HADOOP_TOOLS/jackson-databind
-2.2.3.jar,$HADOOP_TOOLS/jackson-core-2.2.3.jar,$HADOOP_TOOLS/jackson-annotations-2.2.3.jar
```

2. Using the above configuration, now create an archive index with path prefixed with `s3a://`:

s3a://bucket/path/to/archive
In this example,

- `s3a` is the implementation Hadoop will use to transfer and read files from the supplied path
- `bucket` is the name of your S3 bucket
- `/path/to/archive` are directories within the bucket

Further configuration for unique setups

You may need to further configure Splunk Enterprise to search S3 archives depending on the specifics of your configuration.

If you are using a search head exclusively

If you're just using a search head to search your archives, then set the provider's `vix.mode` attribute to `stream`:

```
vix.mode = stream
```

When `vix.mode` is set to `stream`, Splunk Enterprise streams all the data the search matches to the search head, and will not spawn MapReduce jobs on Hadoop.

If you have configured a search head with a Hadoop cluster

If the Hadoop version for search head archive indexes is compatible with your Hadoop cluster, no additional configuration is necessary to search your archive indexes. Just go to the Splunk Web search bar and enter:

```
index=<your-archive-index-name>
```

The search head will spawn Hadoop MapReduce jobs against your archive when it's appropriate to do so.

If your Hadoop cluster version is not compatible with your Hadoop Home version

You can still use Data Roll if your Hadoop cluster is not compatible with your Hadoop client libraries (that have the S3a filesystem). An example of this is if you are using Apache Hadoop 2.6.0 for archiving, but you are using Hadoop 1.2.0 for your Hadoop cluster. To do this, use the older S3n filesystem to search your archives.

To configure S3n and an older Hadoop cluster to search your archives:

1. Configure a Provider for your Hadoop cluster.

2. For every archive index, configure `indexes.conf` from your terminal and add a new virtual index with these properties:

```
[<virtual_index_name_of_your_choosing>]
vix.output.buckets.path = <archive_index_destination_path_with_s3n_instead_of_s3a>
vix.provider = <hadoop_cluster_provider>
```

3. Make sure the `vix.output.buckets.path` is S3n so that Splunk Enterprise can search using the older filesystem to search your archives..

For example. Given an archive index named "main_archive", destination path "s3a://my-bucket/archive_root/main_archive" and provider = "hadoop_cluster", you should configure a virtual index like this:

```
[main_archive_search] vix.output.buckets.path = s3n://my-bucket/archive_root/main_archive vix.provider = hadoop_cluster
```

Known Issues with S3

When using Hadoop's S3N filesystem, you're limited to uploading files that are less than 5GB. While it's rare, it's possible that Splunk buckets become larger than 5GB. These buckets would not get archived when using the S3N filesystem.

If you use the S3N filesystem, configure your indexes to roll buckets from hot to warm at a size less than 5 GB via the `maxDataSize` attribute in `indexes.conf`.

Data Roll archiving requires, at a minimum, read-after-write consistency. For the US Standard region, S3 only guarantees this when accessed via the Northern Virginia endpoint. See the Amazon AWS S3 FAQ for more details.

Bucket raw data limit

Because of how Hadoop interacts with the S3 file system, Splunk Enterprise cannot currently archive buckets with raw data sets larger than 5GB to S3.

We recommend that you use a S3FileSystem implementation that supports uploads larger than 5GB. To ensure that all your data is archived, configure your indexes to roll buckets from hot to warm at a size less than 5 GB via the `maxDataSize` attribute in `indexes.conf`.

Data copy process

When archiving to S3, data is copied twice. This is because S3 does not support file renaming and the FileSystem implements file rename as follows:

- Download the file
- Upload it renamed
- Delete the original file

This process does not create duplicate data in your archive.

Bandwidth throttling limitations

Splunk Enterprise cannot guarantee that bandwidth throttling will be respected when archiving to S3. Splunk will still attempt to throttle bandwidth where possible, if configured to do so.

Search indexed data archived to Hadoop

Once you properly install and configure your archive indexes, you can create reports and visualize data as you would against data in a traditional Splunk index. Using virtual indexes alongside traditional Splunk Enterprise indexes, you can gather data from the virtual index alone; or you can query both local and virtual indexes for a single report.

For the most part, you can create reports for virtual indexes much as you would for local indexes. For more information

about creating reports, see the *Splunk Enterprise Search Manual*.

Since events are not sorted, any search command which depends on implicit time order will not work exactly the way you'd expect. (For example: head, delta, or transaction.) This means that a few search commands operate differently when used on virtual indexes, mostly because of the way Hadoop reports timestamps.

You can still use these commands, and may particularly want to when creating a single report for local and virtual indexes, but you should be aware of how they operate and return data differently.

Search language

For the most part, you can use Splunk Enterprise search language to create your reports. However, because Hadoop does not support strict requirements on the order of events, there are a few differences.

The following commands are not supported when the search includes an archived index:

- `transactions`
- `localize`

The following commands work on archived indexes, but their results may differ from Splunk. This is because in Hadoop, descending time order of events is not guaranteed:

- `streamstats`
- `head`
- `delta`
- `tail`
- `reverse`
- `eventstats`
- `dedup` (Since the command cannot distinguish order within an HDFS directory to pick the item to remove, Splunk Analytics for Hadoop will choose the item to remove based on modified time, or file order.)

Distributable and non-distributable commands in archives

Distributable search commands are the most effective commands Hadoop Data Roll reports because they can be distributed to search peers and archive indexes. Generally, non-distributable commands only work on local indexes and are not as effective on archived indexes.

You can create searches across different index types that use both distributable and non-distributable commands as long as you keep in mind that these such a search returns all data from the local indexes but limited data from the virtual indexes.

Header extractions to avoid when working with virtual indexes

Archived indexes do not support configuration of index time fields. Therefore properties specific to index-time field extractions do not apply to archive indexes. This includes the following properties:

- INDEXED_EXTRACTIONS
- HEADER_FIELD_LINE_NUMBER
- PREAMBLE_REGEX
- FIELD_HEADER_REGEX
- FIELD_DELIMITER
- FIELD_QUOTE

- HEADER_FIELD_DELIMITER
- HEADER_FIELD_QUOTE
- TIMESTAMP_FIELDS = field1,field2,...,fieldn
- FIELD_NAMES
- MISSING_VALUE_REGEX

Archive cold buckets to frozen in Hadoop

Data is aged locally on every indexer. The way you configure your index determines the data size or age at which the data to moves to the next state (hot, warm, cold, frozen) and is ultimately deleted.

Once you configure an index to archive data, the archiving of indexes runs on a schedule that is determined globally on the Splunk search head.

When both processes occur, a disconnect can occur between the indexer's local processes and the archiving process. As a result, the indexers can delete a bucket before it's been archived.

To avoid buckets from being deleted you can use the the `splunk_archiver` app `coldToFrozen.sh` script on the local indexer process. This script shifts the responsibility for deleting buckets from the indexer to Hadoop Data Roll, so only use this script for indexes that are being archived.

Consider the `coldToFrozen.sh` script as a fallback and not your primary hook for archiving. This script buys you more time when either your system is receiving data faster than normal, or when the archiving storage layer is down, so that you'll have more time to archive a given bucket. To facilitate this further, for each archive index you can set your `vix.output.buckets.older.than = seconds` as low as possible, so that buckets are archived as quickly as possible.

Configure the cold bucket to roll to frozen

Note the following if you are using the `coldToFrozen.sh` script:

- The script must be installed on each stanza which configures an index that is being archived.
- All the search peers to the search head must have the script installed. You can do each peer manually or use the deployer for search head clusters.
- The script must be removed from any index for which you disable archiving. Otherwise, the script will continue to run and the data will overfill your existing disk space because there is no archive to receive that data (and thus it will not get deleted).
- Do not add this script to any indexers that are not configured to archive data.

For each Splunk index, use the provided script located in `$SPLUNK_HOME/etc/apps/splunk_archiver/bin/` and named `coldToFrozen.sh` to archive your cold data to frozen. This path may very depending upon your configuration path. For example:

```
[<index name>]
coldToFrozenScript = "$SPLUNK_HOME/etc/apps/splunk_archiver/bin/coldToFrozen.sh"
```

Troubleshoot Hadoop Data Roll

Issue: Splunk user permission issues disrupt the ability to copy data to HDFS from the Indexers

Data archiving is performed from the Indexers. However, all the archiving setup is done from the Search Head. Permissions issues between the search head and indexer will prevent archiving.

For example, if the indexer permissions are not identical to the search head you might see this exception:

```
java.io.IOException: Login failure for null from keytab /etc/security/keytabs/splun.keytab:
javax.security.auth.login.LoginException: Unable to obtain password from user
```

You can test persmissions by copying files from the Indexers as the Splunk user to HDFS: `hadoop fs -put somefile /user/splunk/archive`

Check the following to make sure the Splunk user is configured consistently with the HDFS permissions:

- Kerberos Keytab path
- Hadoop Client library path
- Java library path
- Splunk user exists on the indexers.
- Splunk user has permission to write to HDFS.

Issue: You need to collect archiving errors

Trap all archiving errors in a generic way and then raise an alert

To open the archiving dashboard for debugging, access: *Settings* > *Virtual Index* > *Archived Indexes* > *View Dashboards*

Splunk Web uses the following query:

```
index=_internal source=*splunk_archiver.log* earliest=-1d | rex max_match=1000 "\d{4}-\d{2}-\d{2}
\d{2}:\d{2}:\d{2}\.\d+ -\d{4} (?<severity>\w+) " | where severity="ERROR"
```