What is Splunk Index Clustering?

Index clustering is Splunk’s implementation of Data High Availability. As we have seen in the example of an indexer failure, data will not be available for searching if events were sent to only one indexer. For customers interested in Splunk Data High Availability, Splunk Index Clustering is the solution.

Index Cluster consists of multiple indexers managed by a single server called cluster manager. The cluster manager orchestrates the replication of data (buckets) to one or more indexers depending on desired replication factor.

A Cluster would consist of:

- **A single server acting as a manager node.**
- **Multiple indexers, referred to by peer nodes, that index and replicate data to other peer nodes, as well as search data and forward to Search Heads when requested.**
- **One or more search heads that coordinate searches across all peer nodes.**

Here are some details that helps understanding the above diagram and index clustering.

Manager Node

The manager node manages the cluster, by orchestrating the replication activities among the peer nodes (indexers) and informs the Search Head where to find the required data. It does not index any data.

Peer Node

Peer Nodes (indexers) perform indexing for the whole cluster. Receive and index incoming data. They also send replicated data to other nodes in the cluster and receive replicated data from other peers. The number of peer nodes is dependent on two factors, the cluster replication factor and the indexing load. For example, if you have a replication factor of 3 (3 copies of the data), you need at least three peers. If you have more indexing load than 3 indexers can handle, you need to add more peers to increase capacity.

Search Head

The search head manages searches across all peer nodes, it distributes search queries to all peers, and consolidates the results. A cluster must have at least one search head.

Forwarder

Forwarders act the same as in any Splunk Enterprise Deployment. They consume data from external sources and forward the data to indexers (peers). It can communicate with Manager node (indexer discovery) to get an updated list of available indexers.

Replication Factor

Replication Factor is the number of copies of data the cluster maintains, determines the level of failure tolerance.

Search Factor

Search Factor is the number of searchable copies of the data the cluster maintains, which determines how quickly the cluster can recover its searching capability after a peer node goes down.

Buckets

Buckets are the basic units of index storage used.

The Benefits of Splunk Index Clustering

The most important benefit of Splunk Index Clustering is the High Availability of Data, making it immune to indexer outages. It does simplify forwarder outputs as well.

Benefit #1 High Availability of Data

Index Clustering provides a higher availability of data used for searching when needed by making the cluster immune to indexer outages. Without index clustering a Splunk search might not provide the correct results if one of the indexers is down (not available).

Benefit #2 Simplify Forwarder outputs

Index Clustering simplifies the outputs.conf file required to forward data by forwarders to indexers by contacting cluster manager to provide the list of available indexers. If index clustering is not used, then a complete list of indexers should be included in the outputs.conf file on forwarders, as well as to maintain the list of new indexers are added/removed.

Types of Splunk Index Cluster

There are two major types of indexer clusters

1. **Single-Site index cluster where data is replicated to multiple indexers within a single geographical location. Data will still be available if enough indexers are still available to meet search factor requirement.**
2. **Multi-Site index cluster where data is replicated to multiple indexers in more than one geographical location for Disaster Recovery requirements. Data will still be available even if a whole site goes down.**

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.

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.

Important concepts

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

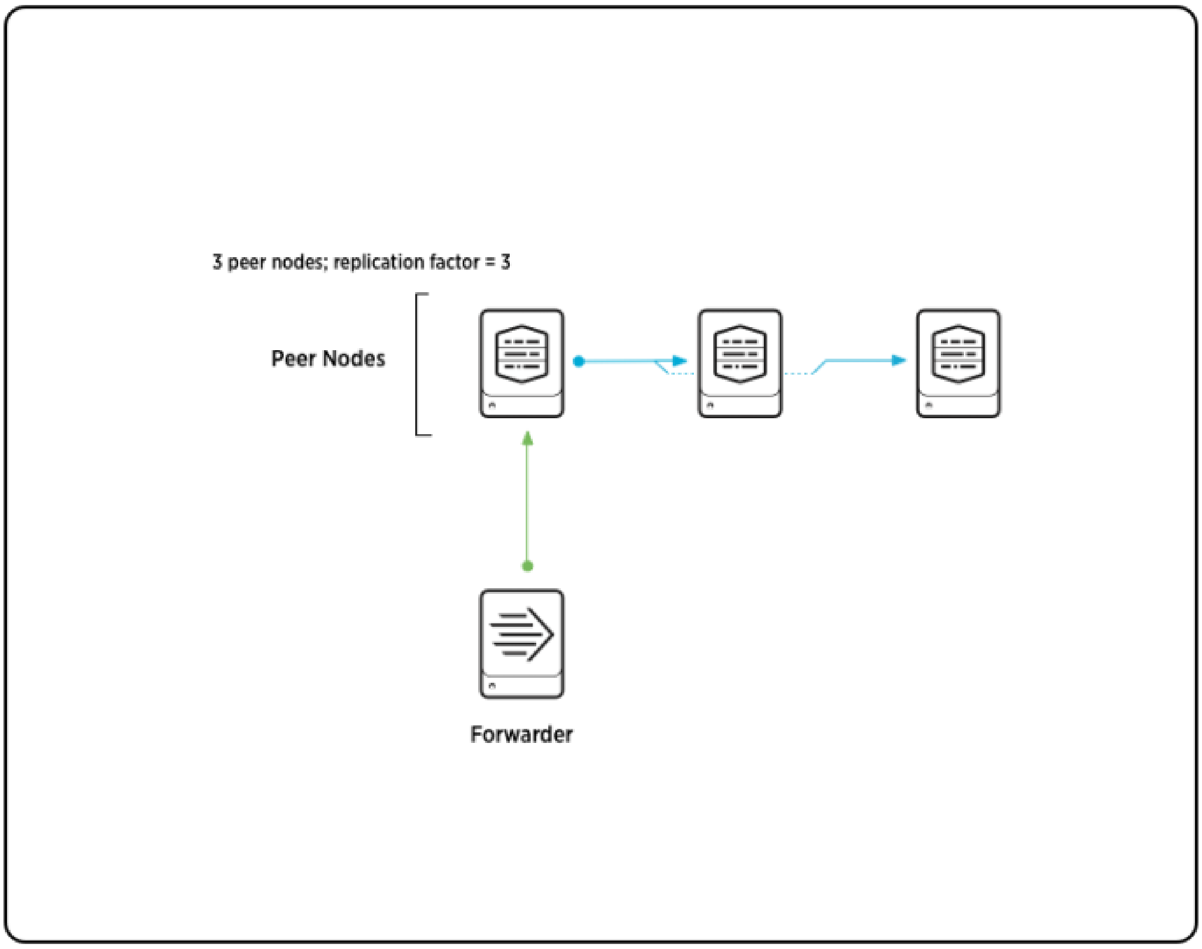
- **Replication factor**. This determines the number of copies of data the cluster maintains and therefore, the cluster's fundamental level of failure tolerance.
- **Search factor**. This determines the number of searchable copies of data the cluster maintains, and therefore how quickly the cluster can 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.

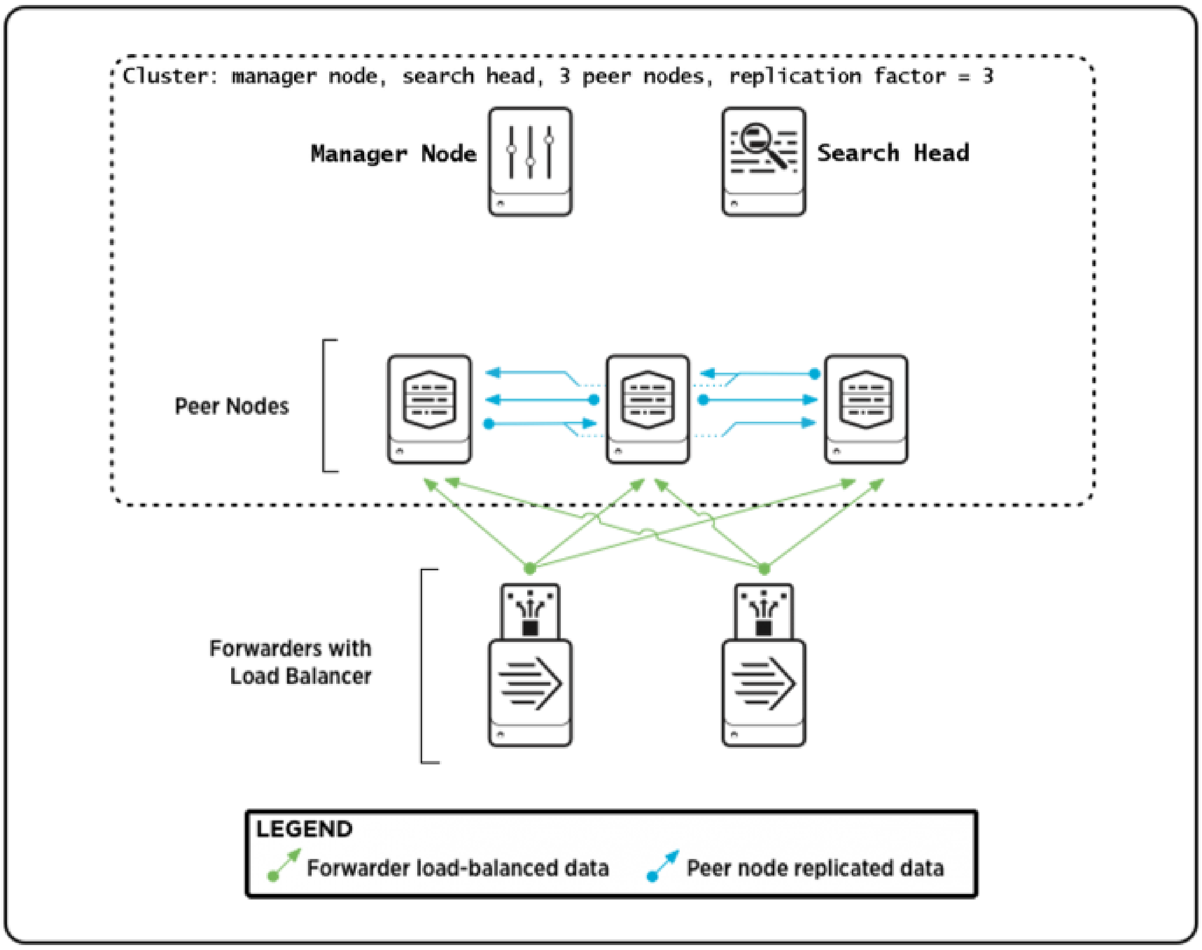
Each time a peer creates a new bucket, it communicates with the manager node to get a list of peers to stream the bucket's data to. If you have a cluster in which the number of peer nodes exceeds the replication factor, a peer might stream data to a different set of peers each time it creates a new bucket. 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:


Cluster: manager node, search head, 3 peer nodes, replication factor = 3
Manager Node
Search Head
Peer Nodes
Forwarders with Load Balancer
LEGEND
Forwarder load-balanced data
Peer node replicated data

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.

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.

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.

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.

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".

server.conf Settings for Various Splunk Functions

In the following sections the significant server.conf settings that determine which role a Splunk server plays and how it interacts with other Splunk nodes in a distributed deployment are explained and compared.

Note: The use of the term ‘master’ or ‘slave’ for some configuration settings has been replaced with ‘manager’ or ‘peer’ in more recent versions of Splunk. You may still see ‘master_uri’ or similar entries in older versions.

Standalone Splunk Enterprise or License Manager

The server.conf in the $SPLUNK_HOME/etc/system/local folder of a standalone instance of Splunk, or a Splunk server that is serving as a License Manager and has a non-free Splunk license installed will have the following entries:

```
[general]
```

```
serverName = ip-10-2-6-36.ec2.internal
pass4SymmKey =
$7$ZEqxybfpYs1H2DNH4eny19ih84dz4suy+RLmhMJeoS+YzGdt8YbNEsgBbjs1

[sslConfig]
sslPassword =
$7$p35gM9btomGgv2OUzJBUQ7wlcZflYy9SHDW3o0n6QYD2jhuw6MOwUQ==

[lmpool:auto_generated_pool_download-trial]
description = auto_generated_pool_download-trial
peers = *
quota = MAX
stack_id = download-trial

[lmpool:auto_generated_pool_forwarder]
description = auto_generated_pool_forwarder
peers = *
quota = MAX
stack_id = forwarder

[lmpool:auto_generated_pool_free]
description = auto_generated_pool_free
peers = *
quota = MAX
stack_id = free

[lmpool:auto_generated_pool_enterprise]
description = auto_generated_pool_enterprise
peers = *
quota = MAX
stack_id = enterprise

[license]
active_group = Enterprise
The significant stanzas and entries include:

[general]
```

serverName: the name that identifies this Splunk software instance for features such as   distributed search.

pass4SymmKey: this hashed password key is used to authenticate traffic between a license manager and its license peers, members of a cluster, or a deployment server and its deployment clients. The keys must match between these members.

[sslConfig] sslPassword: the server certificate password, if any, used for SSL/TLS communications.

[lmpool_auto_generated_pool_<function>]: auto-generated license pool configurations to support the use of download-trial, forwarder, free, or enterprise license types.

[license] active_group: this sets the license type.

With the exception of the license related entries, there are no function related entries in the local folder server.conf for a standalone or license manager node.

Cluster Manager

The Cluster Manager is responsible for pushing indexer configurations and managing bucket replication across the indexing cluster. In the following server.conf entries, the [general] and [sslConfig] stanzas and entries have been omitted (as they will be in further examples) as they are the same as for the previous example.

```
[license]
manager_uri = https://10.2.6.36:8089

[indexer_discovery]
pass4SymmKey =
$7$2N4Q7EspaU2oX8AaeleICi05sDJJg7GAT9gMwWxYc9rZf86bE0CHEF3ht0mb
polling_rate = 1

[clustering]
mode = manager
replication_factor = 2
search_factor = 2
pass4SymmKey =
$7$/gdzL5bjZEAv0v6Dy3yMKtdy/JqTGQblPKUhkHRbcTa+VmBmrKmzZhNtxw==
cluster_label = mdi_svac3_idxc
access_logging_for_heartbeats = 0
cm_heartbeat_period = 3
precompress_cluster_bundle = 1
rebalance_threshold = 0.9
```

The significant stanzas and entries for a Cluster Manager include:

[license] manager_uri: this is the URL of the license manager for licensing purposes.

[indexer_discovery]:
pass4SymmKey: hashed password for authenticating with Universal Forwarder nodes that use this feature.

polling_rate: set the indexer discovery polling rate.

[clustering]

mode: sets operational mode for this cluster node – manager = cluster manager.

replication_factor, search_factor: set the RF/SF for the indexing cluster.

pass4SymmKey: hashed password for authenticating with indexing cluster nodes.

cluster_label: specifies the label of the indexer cluster – only valid for mode = manager.

access_logging_for_heartbeats: enables/disables logging to the splunkd_access.log file

for peer heartbeats – only valid for mode = manager.

cm_heartbeat_period: frequency, in seconds, of cluster manager to cluster.

manager heartbeat for mode = manager and manager_switchover_mode = auto|manual

precompress_cluster_bundle: whether or not the manager compresses the configuration bundle files before it pushes them to peers.

rebalance_threshold: used as a percentage to determine when the cluster is balanced. 1 = 100% balanced.

The last four entries above are only valid for cluster manager nodes; cm_heartbeat_period is related to the redundant cluster manager capability introduced in Splunk v9.0.

Indexer

The server.conf entries for indexers are fairly simple – the [general] and [sslConfig] stanzas and entries are omitted below as they reflect examples already given.

[license] manager_uri = https://10.2.6.36:8089 [clustering] manager_uri = https://10.2.6.33:8089 mode = peer pass4SymmKey = 7/gdzL5bjZEAv0v6Dy3yMKtdy/JqTGQblPKUhkHRbcTa+VmBmrKmzZhNtxw== [replication_port://9887]

The significant stanzas and entries for an indexer include:
[license] manager_uri: this is the URL of the license manager for licensing purposes.
[clustering] manager_uri: URL of the cluster manager(s).

mode: sets operational mode for this cluster node – peer = an indexer

pass4SymmKey: hashed password for authenticating with indexing cluster nodes.

replication_port: TCP port for replicating bucket data to/from other cluster members.

Indexers communicate with the cluster manager via the master_uri to discover other indexers in the cluster, and replicate data buckets to those other indexers per the replication factors specified in the cluster manager server.conf.

Search Head in a Search Head Cluster

The server. conf entries for clustered search heads are a bit more involved than for other nodes because they communicate with the Deployer to fetch configuration bundles, communicate with a Captain and other search heads for knowledge object replication and scheduled search management, communiate with the Cluster Manager to discover indexer nodes, and directly with indexers for search operations. The [general], [sslConfig], and [license] stanzas and related entries are omitted below as they reflect examples already given.

```
[replication_port://9777]
[shclustering]
conf_deploy_fetch_url = https://10.2.6.39:8089
mgmt_uri = https://ip-10-2-4-40.ec2.internal:8089
replication_factor = 2
pass4SymmKey =
$7$cCuCd3Yv0HloWIvJwVZsw7vrI3LHuxNf21dJnhe6U5bX3TcIHoNoC+ugKg==
shcluster_label = mdi_svac3_shc
id = CE32AC4C-7715-4907-B1D2-C955D140E8C0

[clustering] manager_uri = https://10.2.6.33:8089
mode = searchhead
pass4SymmKey =
$7$/gdzL5bjZEAv0v6Dy3yMKtdy/JqTGQblPKUhkHRbcTa+VmBmrKmzZhNtxw==
```

The significant stanzas and entries for a clustered search head include:

replication_port: TCP port for replicating knowledge objects to/from other search head cluster members.

[shclustering] (communicating with the deployer and other search head cluster members)

conf_deploy_fetch_url: URL of a deployer from which members fetch configuration bundles.

mgmt_uri: used to identify the cluster member’s own address to itself.

replication_factor: how many copies of search artifacts are created in the cluster.

pass4SymmKey: hashed password for authenticating with the Deployer and other search head cluster nodes.

shcluster_label: the label of the search head cluster.

[clustering] (settings for communicating with the cluster manager and indexers)

manager_uri: URL to the cluster manager(s).

mode: sets operational mode for this cluster node

pass4SymmKey: hashed password for authenticating with the cluster manager and indexing cluster nodes.

Deployer

The Deployer distributes configuration bundles (apps and private user folders for search heads) to the search head cluster members. The [general], [sslConfig], and [license] stanzas and related entries are omitted below as they reflect examples already given.

```
[shclustering]

pass4SymmKey =
$7$cCuCd3Yv0HloWIvJwVZsw7vrI3LHuxNf21dJnhe6U5bX3TcIHoNoC+ugKg==
shcluster_label = mdi_svac3_shc
```

The significant stanzas and entries for a Deployer include:

[shclustering] (communicating with the search head cluster members)

pass4SymmKey: hashed password for authenticating with search head cluster nodes.

shcluster_label: the label of the search head cluster.

Deployment Server and Heavy Forward

The deployment server’s only role is to distribute applications to Splunk Universal Forwarders and any other Splunk nodes as specified by entries in the serverclass.conf that resides in $SPLUNK_HOME/etc/system/local; the presence of this file invokes this functionality.

Heavy forwarders are typically populated with apps containing other .conf files that configure inputs and any event data parsing before forwarding data to the indexing tier via outputs.conf settings.

Since these are basically standalone Splunk Enterprise servers the non-default entries in the /local server.conf file are pretty generic.

```
[general]
serverName = ip-10-2-6-5.ec2.internal
pass4SymmKey =
$7$ZEqxybfpYs1H2DNH4eny19ih84dz4suy+RLmhMJeoS+YzGdt8YbNEsgBbjs1

[sslConfig]
sslPassword =
$7$CKCCf2zYemSzb+IBpqGg6Nfxt78GMIbz1ZSREemWiP1tfxGmhPon/Q==
[license]
manager_uri = https://10.2.6.36:8089
```

This concludes an overview of typical server.conf entries for the various Splunk Enterprise nodes in a distributed, clustered environment.

Note that the hashed pass4SymmKey password in the [shcluster] section of server.conf should be the same across the Deployer and all clustered search heads. The hashed pass4SymmKey password for the [clustering] section should be the same across the Cluster Manager and all indexer nodes, as well as the [clustering] section of server.conf on search heads. Finally, the hashed pass4SymmKey password for Indexer Discovery must be the same between the Cluster Manager and the UF nodes. If Splunk configurations look correct but nodes aren't connecting and functioning correctly, checking that these passwords are properly matched is essential.

Conclusion

This tour of the typical non-default server.conf stanzas and entries should be helpful for understanding how these settings work together across a distributed, clustered Splunk deployment, and for verifying that the appropriate entries and reasonable settings are in place while troubleshooting a functional issue within a new or updated Splunk environment.