Table of Contents

Overview of distributed search..............................................................................................................1
  About distributed search.........................................................................................................................1

Knowledge bundle replication.................................................................................................................5
  What search heads send to search peers..................................................................................................5
  Modify the knowledge bundle..................................................................................................................6
  About mounted bundles..........................................................................................................................10
  Configure mounted bundles...................................................................................................................12
  Use mounted bundles with search head pooling..................................................................................15

Deploy distributed search.......................................................................................................................19
  Deploy a distributed search environment................................................................................................19
  System requirements and other deployment considerations for distributed search..........................21
  Add search peers to the search head.........................................................................................................24
  Best practice: Forward search head data to the indexer layer...............................................................28

Manage distributed search.....................................................................................................................30
  Manage distributed server names...........................................................................................................30
  Create distributed search groups...........................................................................................................30
  Remove a search peer.............................................................................................................................32

View distributed search status...............................................................................................................34
  View search peer status in Settings.........................................................................................................34
  Use the monitoring console to view distributed search status...............................................................35

Manage parallel reduce search processing............................................................................................36
  Overview of parallel reduce search processing....................................................................................36
  Configure parallel reduce search processing.........................................................................................39
  Apply parallel reduce processing to searches.......................................................................................43

Overview of search head clustering......................................................................................................45
  About search head clustering..................................................................................................................45
  Search head clustering architecture.......................................................................................................47

Deploy search head clustering...............................................................................................................62
  System requirements and other deployment considerations for search head clusters.......................62
  Deploy a search head cluster....................................................................................................................68
Table of Contents

Deploy search head clustering
  Integrate the search head cluster with an indexer cluster......................75
  Connect the search heads in clusters to search peers........................79
  Add users to the search head cluster................................................83
  Use a load balancer with search head clustering................................84
  Deploy a search head cluster in a multisite environment....................84
  Deploy a single-member search head cluster....................................87
  Migrate settings from a standalone search head to a search head cluster.................................89
  Migrate from a search head pool to a search head cluster..................93
  Upgrade a search head cluster.......................................................99
  Use rolling upgrade.......................................................................102

Configure search head clustering..........................................................109
  Configure the search head cluster....................................................109
  Choose the replication factor for the search head cluster..................112
  Set a security key for the search head cluster..................................113

Update search head cluster members.....................................................116
  How configuration changes propagate across the search head cluster.........................116
  Configuration updates that the cluster replicates..................................119
  Use the deployer to distribute apps and configuration updates..............127

Manage search head clustering..............................................................153
  Add a cluster member.....................................................................153
  Remove a cluster member..............................................................157
  Configure a cluster member to run ad hoc searches only.....................158
  Control captaincy..........................................................................160
  Handle failure of a search head cluster member...............................164
  Use static captain to recover from loss of majority............................166
  Put a search head cluster member into detention..............................171
  Restart the search head cluster.......................................................175
  Back up and restore search head cluster settings..............................184

Troubleshoot search head clustering....................................................188
  Use the search head clustering dashboard.......................................188
  Use the CLI to view information about a search head cluster..............188
  Use the monitoring console to view search head cluster status and
# Table of Contents

**Troubleshoot search head clustering**
- troubleshoot issues ................................................................. 191
- Deployment issues ................................................................. 195
- Runtime considerations ....................................................... 195
- Handle Raft issues .............................................................. 196

**Search head pooling** .............................................................. 198
- Overview of search head pooling ........................................... 198
- Create a search head pool .................................................... 201
- Use a load balancer with the search head pool ................... 204
- Other pooling operations .................................................... 205
- Manage configuration changes ............................................ 206
- Deployment server and search head pooling ....................... 207
- Select timing for configuration refresh ............................... 207
- Upgrade a search head pool ................................................ 208

**Distributed search in action** .................................................. 209
- How authorization works in distributed searches ................ 209
- How users can control distributed searches ....................... 212

**Troubleshoot distributed search** ........................................... 213
- Use the monitoring console to view distributed search status 213
- General troubleshooting issues .......................................... 213
- Handle slow search peers ................................................... 215
- Quarantine a search peer .................................................... 215
- Search head pooling configuration issues ......................... 217
- Distributed search error messages ..................................... 221
Overview of distributed search

About distributed search

Before reading this manual, see the Distributed Deployment Manual. That manual describes the fundamentals of Splunk Enterprise distributed deployment and shows how distributed search contributes to the overall deployment.

Distributed search provides a way to scale your deployment by separating the search management and presentation layer from the indexing and search retrieval layer.

Use cases

These are some of the key use cases for distributed search:

- **Horizontal scaling for enhanced performance.** Distributed search facilitates horizontal scaling by providing a way to distribute the indexing and searching loads across multiple Splunk Enterprise instances, making it possible to index and search large quantities of data.

- **Access control.** You can use distributed search to control access to indexed data. For example, some users, such as security personnel, might need access to data across the enterprise, while others need access to data only in their functional area.

- **Managing geo-dispersed data.** Distributed search allows local offices to access their own data, while maintaining centralized access at the corporate level. For example, users in Chicago and San Francisco can look just at their local data, while users at headquarters in New York can search the local data, as well as the data in Chicago and San Francisco.

Distributed search components

With distributed search, a Splunk Enterprise instance called a search head sends search requests to a group of indexers, or search peers, which perform the actual searches on their indexes. The search head then merges the results back to the user. Here is a basic distributed search scenario, with one search head managing searches across several indexers:
Types of distributed search

There are several basic options for deploying a distributed search environment:

- Use one or more independent search heads to search across the search peers.
- Deploy multiple search heads in a search head cluster. The search heads in the cluster share resources, configurations, and jobs. This offers a way to scale your deployment transparently to your users.
- Deploy search heads as part of an indexer cluster. Among other advantages, an indexer cluster promotes data availability and data recovery. The search heads in an indexer cluster can be either independent search heads or members of a search head cluster.

In each case, the search heads perform only the search management and presentation functions. They connect to search peers that index data and search across the indexed data.

**Independent search heads**

A small distributed search deployment has one independent search head; that is, a search head that is not part of a cluster.

To scale beyond a single search head, deploy a search head cluster.
**Search head clusters**

A search head cluster is a group of search heads that work together to provide scalability and high availability. It serves as a central resource for searching across a set of search peers.

The search heads in a cluster are, for most purposes, interchangeable. All search heads have access to the same set of search peers. They can also run or access the same searches, dashboards, knowledge objects, and so on.

A search head cluster is the recommended topology when you need to run multiple search heads across the same set of search peers. The cluster coordinates the activity of the search heads, allocates jobs based on the current loads, and ensures that all the search heads have access to the same set of knowledge objects.

See "About search head clustering."

**Indexer clusters and search heads**

Indexer clusters also use search heads to search across the set of indexers, or peer nodes. The search heads in an indexer cluster can be either independent search heads or members of a search head cluster.

You deploy and configure search heads very differently when they are part of an indexer cluster:

- For information on using independent search heads with indexer clusters, see "Configure the search head" in the Managing Indexers and Clusters of Indexers manual.

- For information on using search head clusters with indexer clusters, read "Integrate the search head cluster with an indexer cluster".

**Parallel reduce search processing**

If you struggle with extremely large high-cardinality searches, you might be able to apply parallel reduce processing to them to help them complete faster. You must have a distributed search environment to use parallel reduce search processing.

High-cardinality searches are searches that must match, filter, and aggregate fields with extremely large numbers of unique values. During a parallel reduce
search process, some or all of a high-cardinality search job is processed in parallel by indexers that have been configured to behave as intermediate reducers for the purposes of the search. This parallelization of reduction work that otherwise would be done entirely by the search head can result in faster completion times for high-cardinality searches.

If you want to take advantage of parallel reduce search processing, your indexers should be operating with a light to medium load on average. You can use parallel reduce search processing whether or not your indexers are clustered.

See Overview of parallel reduce search processing.
Knowledge bundle replication

What search heads send to search peers

When initiating a distributed search, the search head replicates and distributes its knowledge objects to its search peers, or indexers. Knowledge objects include saved searches, event types, and other entities used in searching across indexes. The search head needs to distribute this material to its search peers so that they can properly execute queries on its behalf. This set of knowledge objects is called the knowledge bundle.

What the knowledge bundle contains

The search peers use the search head's knowledge bundle to execute queries on its behalf. When executing a distributed search, the peers are ignorant of any local knowledge objects. They have access only to the objects in the search head's knowledge bundle.

Bundles typically contain a subset of files (configuration files and assets) from $SPLUNK_HOME/etc/system, $SPLUNK_HOME/etc/apps and $SPLUNK_HOME/etc/users.

The process of distributing knowledge bundles means that peers by default receive nearly the entire contents of the search head's apps. If an app contains large binaries that do not need to be shared with the peers, you can eliminate them from the bundle and thus reduce the bundle size. See "Modify the knowledge bundle".

Location of the knowledge bundle

On the search head, the knowledge bundles resides under the $SPLUNK_HOME/var/run directory. The bundles have the extension .bundle for full bundles or .delta for delta bundles. They are tar files, so you can run `tar tvf` against them to see the contents.

The knowledge bundle gets distributed to the $SPLUNK_HOME/var/run/searchpeers directory on each search peer. Because the knowledge bundle reside at a different location on the search peers than on the search head, search scripts should not hardcode paths to resources.
View replication status

After you add search peers to the search head, as described in "Add search peers to the search head," you can view the replication status of the knowledge bundle:

1. On the search head, click **Settings** at the top of the Splunk Web page.
2. Click **Distributed search** in the Distributed Environment area.
3. Click **Search peers**.

There is a row for each search peer. The column **Replication status** indicates whether the search head is successfully replicating the knowledge bundle to the search peer.

**Note:** In the case of a **search head cluster**, you must view replication status from the search head cluster captain. This is because only the captain replicates the knowledge bundle to the cluster’s search peers. The other cluster members do not participate in bundle replication. If you view the search peers’ status from a non-captain member, the **Replication status** column might read "Initial" instead of "Successful."

**User authorization**

All authorization for a distributed search originates from the search head. At the time it sends the search request to its search peers, the search head also distributes the authorization information. It tells the search peers the name of the user running the search, the user’s role, and the location of the distributed **authorize.conf** file containing the authorization information.

**Modify the knowledge bundle**

The **knowledge bundle** consists of a set of files that the search peers ordinarily need in order to perform their searches. You can, if necessary, modify this set of files. The main reasons for modifying the set of files are if:

- **As an app developer**, you want to customize the files for the needs of your app. This case usually involves manipulating the replication whitelist. You can also use a replication blacklist for this purpose.
• **As an admin**, you need to eliminate files from the knowledge bundle, in order to limit the bundle size. This case is somewhat unusual, because Splunk Enterprise uses delta-based replication to keep the bundle compact, with the search head usually only replicating the changed portion of the bundle to its search peers. This case requires that you identify unnecessary files and filter them out with a replication blacklist. It is also possible, although less common, to use a whitelist for this purpose.

See distsearch.conf in the *Admin Manual* for details on the settings discussed in this topic.

**Customize the bundle for an app**

The system looks at two stanzas in distsearch.conf to determine which *.conf files to include in the bundle, in this order:

1. [replicationWhitelist]
2. [replicationSettings:refineConf]

You typically only need to edit the [replicationSettings:refineConf] stanza to customize the bundle for your app, but, under rare circumstances, you might also need to modify the [replicationWhitelist] stanza.

Since the system starts by examining the [replicationWhitelist] stanza, this discussion does too.

**Edit the replicationWhitelist stanza**

The [replicationWhitelist] stanza in the system default version of distsearch.conf whitelists all the *.conf files that are specified in the [replicationSettings:refineConf] stanza. Therefore, to add or delete a *.conf file from the bundle, do not modify this stanza. Instead, change the set of files specified in the [replicationSettings:refineConf] stanza, as described in the next section, "Edit the replicationSettings:refineConf stanza."

The main reason for modifying the [replicationWhitelist] stanza is to include in the bundle some type of special file for use in a custom search command. This is an unusual circumstance.

If you do need to alter the whitelist, you can override the system default whitelist by creating a version of the [replicationWhitelist] stanza in

$SPLUNK_HOME/etc/apps/<appname>/default/distsearch.conf:
[replicationWhitelist]

The knowledge bundle will include all files that both satisfy the whitelist regex and are specified in [replicationSettings:refineConf]. If multiple regex’s are specified, the bundle will include the union of those files.

In this example, the knowledge bundle will include all files with extensions of either ".conf" or ".spec”:

[replicationWhitelist]
allConf = *.conf
allSpec = *.spec

The names, such as allConf and allSpec, are used only for layering. That is, if you have both a global and a local copy of distsearch.conf, the local copy can be configured so that it overrides only one of the regex’s. For instance, assume that the example shown above is the global copy and that you then specify a whitelist in your local copy like this:

[replicationWhitelist]
allConf = *.foo.conf

The two conf files will be layered, with the local copy taking precedence. Thus, the search head will distribute only files that satisfy these two regex’s:

allConf = *.foo.conf
allSpec = *.spec

For more information on attribute layering in configuration files, see Attribute precedence in the Admin manual.

Caution: Replication whitelists are applied globally across all conf data, and are not limited to any particular app, regardless of where they are defined. Be careful to pull in only your intended files.

Edit the replicationSettings:refineConf stanza

The [replicationSettings:refineConf] stanza in distsearch.conf specifies the ".conf" files and ".meta" stanzas that get included in the knowledge bundle. If you
want to modify the set of files in the bundle, add or delete them from this stanza.

The system default `distsearch.conf` file includes a version of this stanza that specifies the `*.conf` files that are normally included in the knowledge bundle:

```plaintext
[replicationSettings:refineConf]
# Replicate these specific *.conf files and their associated *.meta stanzas.
replicate.app = true
replicate.authorize = true
replicate.collections = true
replicate.commands = true
replicate.eventtypes = true
replicate.fields = true
replicate.segmenters = true
replicate.literals = true
replicate.lookups = true
replicate.multikv = true
replicate.props = true
replicate.tags = true
replicate.transforms = true
replicate.transactiontypes = true
```

If you want to replicate a `*.conf` file that is not in the system default version of the `[replicationSettings:refineConf]` stanza, create a version of the stanza in `$SPLUNK_HOME/etc/apps/<appname>/default/distsearch.conf` and specify the `*.conf` file there. Similarly, you can remove files from the bundle by setting them to "false" in this stanza.

### Eliminate files from the knowledge bundle

You can also create a replication blacklist, using the `[replicationBlacklist]` stanza. This is most useful for limiting the size of the knowledge bundle, particularly in the case of very large files that do not need to be replicated to the search peers. The blacklist takes precedence over any whitelist.

**Caution:** Replication blacklists are applied globally across all conf data, and are not limited to any particular app, regardless of where they are defined. If you are defining an app-specific blacklist, be careful to constrain it to match only files that your application will not need.
About mounted bundles

**Important:** For most deployments, Splunk recommends that you use normal bundle replication, not mounted bundles with shared storage. As a result of changes to bundle replication made in the 5.0 timeframe, such as the introduction of delta-based replication and improvements in streaming, the practical use case for mounted bundles is now extremely limited. In most cases, mounted bundles make little difference in the amount of network traffic or the speed at which bundle changes get distributed to the search peers. At the same time, they add significant management complexity, particularly when combined with shared storage. Because of delta-based replication, even if your configurations contain large files, normal bundle replication entails little ongoing replication cost, as long as those files rarely change.

The set of data that a search head distributes to its search peers is called the **knowledge bundle**. The bundle contents reside in the search head's `$SPLUNK_HOME/etc/{apps,users,system}` subdirectories. For information on the contents and purpose of this bundle, see "What search heads send to search peers".

By default, the search head replicates and distributes the knowledge bundle to each search peer. You can instead tell the search peers to mount the knowledge bundle's directory location, eliminating the need for bundle replication. When you mount a knowledge bundle on shared storage, it's referred to as a **mounted bundle**.

**Caution:** Most shared storage solutions don't work well across a WAN. Since mounted bundles require shared storage, you generally should not implement them across a WAN.

Mounted bundle architectures

Depending on your search head configuration, there are a number of ways to set up mounted bundles. These are some of the typical ones:

- **For a single search head.** Mount the knowledge bundle on shared storage. All the search peers then access the bundle to process search requests. This diagram illustrates a single search head with a mounted bundle on shared storage:
• **For multiple non-clustered search heads.** Maintain the knowledge bundle(s) on each search head's local storage. In this diagram, each search head maintains its own bundle, which each search peer mounts and accesses individually:
In each case, the search peers need access to each search head's
$SPLUNK_HOME/etc/{apps,users,system} subdirectories.

The search peers use the mounted directories only when fulfilling the search head's search requests. For indexing and other purposes not directly related to distributed search, the search peers will use their own, local apps, users, and system directories, the same as any other indexer.

Configure mounted bundles

To set up mounted bundles, you need to configure both the search head and its search peers. The procedures described here assume the bundles are on shared storage, but they do not need to be. They just need to be in some location that both the search head and its search peers can access.

Configure the search head

Here are the steps you take on the search head:

1. Mount the bundle subdirectories ($SPLUNK_HOME/etc/{apps,users,system}) on shared storage. The simplest way to do this is to mount the search head's entire $SPLUNK_HOME/etc directory:
   - On *nix platforms, set up an NFS mount.
   - On Windows, set up a CIFS (SMB) share.

   **Important:** The search head's Splunk user account needs read/write access to the shared storage location. The search peers must have only read access to the bundle subdirectories, to avoid file-lock issues. Search peers do not need to update any files in the shared storage location.

2. In the distsearch.conf file on the search head, set:

   shareBundles=false

   This stops the search head from replicating bundles to the search peers.

3. Restart the search head.
Configure the search peers

For each search peer, follow these steps to access the mounted bundle:

1. Mount the bundle directory on the search peer.

2. Create a `distsearch.conf` file in `$SPLUNK_HOME/etc/system/local/` on the search peer. For each search head that the peer is connected to, create a `[searchhead:<searchhead-splunk-server-name>]` stanza, with these attributes:

   ```
   [searchhead:<searchhead-splunk-server-name>]
   mounted_bundles=true
   bundles_location=<path_to_bundles>
   ```

   Note the following:

   - The search peer’s configuration file must contain only the `[searchhead:<searchhead-splunk-server-name>]` stanza(s). The other stanzas in `distsearch.conf` are for search heads only.

   - To identify the `<searchhead-splunk-server-name>`, run this command on the search head:

     ```
     splunk show servername
     ```

   - **Important:** If the search peer is running against a search head cluster, the `[searchhead:]` stanza on the peer must specify the cluster’s GUID, not the server name of any cluster members. For example:

     ```
     [searchhead:C7729EE6-D260-4268-A699-C1F95AAD07D5]
     ```

     To identify the GUID, run this command on a cluster member:

     ```
     splunk show shcluster-status
     ```

     The cluster GUID is the value of the `id` field, located in the captain section of the results.

   - The `<path_to_bundles>` needs to specify the mountpoint on the search peer, not on the search head. For example, say `$SPLUNK_HOME` on your...
search head is /opt/splunk, and you export /opt/splunk/etc via NFS. Then, on the search peer, you mount that NFS share at /mnt/splunk-head. The value of `<path_to_bundles>` should be /mnt/splunk-head, not /opt/splunk.

- If multiple non-clustered search heads will be distributing searches to this search peer, you must create a separate stanza on the search peer for each of them.

3. Restart the search peer.

**Note:** You can optionally set up symbolic links to the bundle subdirectories (apps, users, system) to ensure that the search peer has access only to the necessary subdirectories in the search head's /etc directory. See the following example for details on how to do this.

**Example configuration**

Here's an example of how to set up mounted bundles on shared storage:

**Search head**

On a search head whose Splunk Enterprise server name is "searcher01":

1. Mount the search head's `$SPLUNK_HOME/etc` directory to shared storage with read/write access.

2. In the `distsearch.conf` file on the search head, set:

```
[distributedSearch]
... shareBundles = false
```

3. Restart the search head.

**Search peers**

For each search peer:

1. Mount the search head's `$SPLUNK_HOME/etc` directory on the search peer to:

   /mnt/searcher01
2. (Optional.) Create a directory that consists of symbolic links to the bundle subdirectories:

```
/opt/shared_bundles/searcher01
/opt/shared_bundles/searcher01/system -> /mnt/searcher01/system
/opt/shared_bundles/searcher01/users -> /mnt/searcher01/users
/opt/shared_bundles/searcher01/apps -> /mnt/searcher01/apps
```

**Note:** This optional step is useful for ensuring that the peer has access only to the necessary subdirectories.

3. Create a `distsearch.conf` file in `$SPLUNK_HOME/etc/system/local/` on the search peer, with this stanza:

```
[searchhead:searcher01]
mounted_bundles = true
bundles_location = /opt/shared_bundles/searcher01
```

4. Restart the search peer.

5. Repeat the process for each search peer.

### Use mounted bundles with search head pooling

<table>
<thead>
<tr>
<th>This feature has been deprecated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search head pooling has been deprecated as of Splunk Enterprise version 6.2. This means that although it continues to function, it might be removed in a future version.</td>
</tr>
<tr>
<td>As an alternative, you can deploy search head clustering. See “About search head clustering”. For information on mounted bundles and search head clustering, see “Search head clustering and mounted bundles”.</td>
</tr>
<tr>
<td>For a list of all deprecated features, see the topic &quot;Deprecated features&quot; in the Release Notes.</td>
</tr>
</tbody>
</table>

The process for configuring mounted bundles is basically no different if you're using search head pooling to manage multiple search heads. A few things to keep in mind:
• Use the same shared storage location for both the search head pool and the mounted bundles. Search head pooling uses a subset of the directories required for mounted bundles.
• Search head pooling itself only requires that you mount the $SPLUNK_HOME/etc/{apps,users} directories. However, when using mounted bundles, you must also provide a mounted $SPLUNK_HOME/etc/system directory. This doesn't create any conflict among the search heads, as they will always use their own versions of the system directory and ignore the mounted version.
• The search peers must create separate stanzas in distsearch.conf for each search head in the pool. The bundles_location in each of those stanzas must be identical.

See "Configure search head pooling" for information on setting up a search head pool.

Example configuration: Search head pooling with mounted bundles

This example shows how to combine search head pooling and mounted bundles in one system. There are two main sections to the example:

1. Set up a search head pool consisting of two search heads. In this part, you also mount the bundles.

2. Set up the search peers so that they can access bundles from the search head pool.

The example assumes you're using an NFS mount for the shared storage location.

Part 1: Set up the search head pool

Before configuring the pool, perform these preliminary steps:

1. Enable two Splunk Enterprise instances as search heads. This example assumes that the instances are named "searcher01" and "searcher02".

2. Set up a shared storage location accessible to each search head. This example assumes that you set up an NFS mountpoint, specified on the search heads as /mnt/search-head-pooling.

For detailed information on these steps, see "Create a pool of search heads".
Now, configure the search head pool:

1. On each search head, stop splunkd:

    splunk stop splunkd

2. On each search head, enable search head pooling. In this example, you're using an NFS mount of /mnt/search-head-pooling as your shared storage location:

    splunk pooling enable /mnt/search-head-pooling [--debug]

   Among other things, this step creates empty /etc/apps and /etc/users directories under /mnt/search-head-pooling. Step 3 uses those directories.

3. Copy the contents of the $SPLUNK_HOME/etc/apps and $SPLUNK_HOME/etc/users directories on one of the search heads into the /etc/apps and /etc/users subdirectories under /mnt/search-head-pooling:

    cp -r $SPLUNK_HOME/etc/apps/* /mnt/search-head-pooling/etc/apps
    cp -r $SPLUNK_HOME/etc/users/* /mnt/search-head-pooling/etc/users

4. Copy one search head's $SPLUNK_HOME/etc/system directory to

    /mnt/search-head-pooling/etc/system.

    cp -r $SPLUNK_HOME/etc/system /mnt/search-head-pooling/etc/

5. Review the /mnt/search-head-pooling/etc/system/local/server.conf file for a [pooling] stanza. If it exists, remove any entries.

6. On each search head, edit the distsearch.conf file to set shareBundles = false:

    [distributedSearch]
    ...
    shareBundles = false

7. On each search head, start splunkd:

    splunk start splunkd
Your search head pool should now be up and running.

**Part 2: Mount bundles on the search peers**

Now, mount the bundles on the search peers.

**On each search peer, perform these steps:**

1. Mount the shared storage location (the same location that was earlier set to `/mnt/search-head-pooling` on the search heads) so that it appears as `/mnt/bundles` on the peer.

2. Create a directory that consists of symbolic links to the bundle subdirectories:

   ```
   /opt/shared_bundles/bundles/system -> /mnt/bundles/etc/system
   /opt/shared_bundles/bundles/users -> /mnt/bundles/etc/users
   /opt/shared_bundles/bundles/apps -> /mnt/bundles/etc/apps
   ```

3. Create a `distsearch.conf` file in `$SPLUNK_HOME/etc/system/local/` on the search peer, with stanzas for each of the two search heads:

   ```
   [searchhead:searcher01]
   mounted_bundles = true
   bundles_location = /opt/shared_bundles/bundles

   [searchhead:searcher02]
   mounted_bundles = true
   bundles_location = /opt/shared_bundles/bundles
   ```

4. Restart the search peer:

   ```
   splunk restart splunkd
   ```

Repeat the process for each search peer.
Deploy distributed search

Deploy a distributed search environment

Important: The topics in this chapter explain how to deploy a non-clustered distributed search topology. For information on deploying a search head cluster instead, read the chapter Deploy search head clustering.

The basic configuration to enable distributed search is simple. You designate one Splunk Enterprise instance as the search head and establish connections from the search head to one or more search peers, or indexers.

If you need to deploy more than a single search head, the best practice is to deploy the search heads in a search head cluster.

This is the type of topology that this topic specifically addresses:
The search head interfaces with the user and manages searches across the set of indexers. The indexers index incoming data and search the data, as directed by the search head.

**Deploy distributed search**

To set up a simple distributed search topology, consisting of a single dedicated search head and several search peers, perform these steps:

1. **Identify your requirements.** See System requirements and other deployment considerations for distributed search.

2. **Designate a Splunk Enterprise instance as the search head.** Since distributed search is enabled automatically on every full Splunk Enterprise instance, you do not actually perform any action in this step, aside from choosing the instance that you want to be your search head.

   Choose an existing instance that is not indexing external data or install a new instance. For installation information, see the topic in the *Installation Manual* specific to your operating system.

3. **Establish connections from the search head to all the search peers that you want it to search across.** This is the key step in the procedure. See Add search peers to the search head.

4. **Add data inputs to the search peers.** You add inputs in the same way as for any indexer, either directly on the search peer or through forwarders connecting to the search peer. See the *Getting Data In* manual for information on data inputs.

5. **Forward the search head's internal data to the search peers.** See Best practice: Forward search head data to the indexer layer.

6. **Log in to the search head and perform a search that runs across all the search peers, such as a search for *.** Examine the *splunk_server* field in the results. Verify that all the search peers are listed in that field.

7. **See the *Securing Splunk Enterprise* manual for information on setting up authentication.**

To increase indexing capacity, deploy additional search peers. To increase the search management capacity, deploy multiple search heads as members of a search head cluster.
Deploy multiple search heads

To deploy multiple search heads, the best practice is to deploy the search heads in a search head cluster. This provides numerous advantages, including simplified scaling and management. See the chapter Deploy search head clustering.

Deploy search heads in indexer clusters

Splunk indexer clusters use search heads to search across their set of indexers, or peer nodes. You deploy search heads very differently when they are part of an indexer cluster. To learn about deploying search heads in indexer clusters, read Enable the search head in the Managing Indexers and Clusters of Indexers manual.

System requirements and other deployment considerations for distributed search

This topic describes the key considerations when deploying a basic distributed search topology with search heads that function independently of each other. If instead you are deploying a search head cluster, see System requirements and other deployment considerations for search head clusters.

Hardware requirements for distributed search instances

For information on the hardware requirements for search heads and search peers (indexers), see Reference hardware in the Capacity Planning Manual.

Operating system compatibility

A non-clustered distributed search deployment can include a combination of search heads and indexers running on any supported operating system. For example, you can use a combination of indexers running on different supported Linux operating systems, such as RHEL 6.x and RHEL 7.x. See Supported operating systems in the Installation Manual.

For search head cluster and indexer cluster deployments, each cluster node must be running on the same operating system version. For more information on indexer cluster requirements, see System requirements and other deployment considerations for indexer clusters in Managing indexers and clusters of
Splunk Enterprise version compatibility

Upgrade search heads and search peers at the same time to take full advantage of the latest search capabilities. If you cannot do so, follow these version compatibility guidelines.

Compatibility between search heads and search peers

The following rules define compatibility requirements between search heads and search peers:

- 7.x search heads are compatible with 7.x and 6.x search peers.
- The search head must be at the same or a higher level than the search peers. See the note later in this section for a precise definition of "level" in this context.

Here is a non-exhaustive set of examples illustrating the sort of combinations that are compatible:

- A 6.4 search head is compatible with a 6.3 search peer.
- A 7.0 search head is compatible with a 6.4 search peer.
- A 7.0 search head is compatible with a 7.0 search peer.

In contrast, here are examples of some combinations that are not compatible:

- A 6.3 search head is not compatible with a 6.4 search peer.
- A 6.4 search head is not compatible with a 7.0 search peer.

Note the following:

- These guidelines are valid for standalone search heads and for search heads that are participating in a search head cluster.
- Search heads participating in indexer clusters have different compatibility restrictions. See Splunk Enterprise version compatibility in Managing Indexers and Clusters of Indexers.
- Compatibility is significant at the major/minor release level, but not at the maintenance level. For example, a 6.3 search head is not compatible with a 6.4 search peer, because the 6.3 search head is at a lower minor release level than the 6.4 search peer. However, a 6.3.1 search head is compatible with a 6.3.3 search peer, despite the lower maintenance release level of the search head.
Mixed-version distributed search compatibility

You can run a 6.x search head against 5.x search peers, but there are a few compatibility issues to be aware of. To take full advantage of the 6.x feature set, upgrade search heads and search peers at the same time.

This section describes the compatibility issues.

6.x features in a mixed-version deployment

When running a 6.x search head against 5.x search peers, note the following:

- You can use data models on the search head, but only without report acceleration.
- You can use Pivot on the search head.
- You can run predictive analytics (the `predict` command) on the search head.

Licenses for distributed search

Each instance in a distributed search deployment must have access to a license pool. This is true for both search heads and search peers. See Licenses and distributed deployments in Admin Manual.

Synchronize system clocks across the distributed search environment

Synchronize the system clocks on all machines, virtual or physical, that are running Splunk Enterprise distributed search instances. Specifically, this means your search heads and search peers. In the case of search head pooling or mounted bundles, this also includes the shared storage hardware. Otherwise, various issues can arise, such as bundle replication failures, search failures, or premature expiration of search artifacts.

The synchronization method that 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.
Add search peers to the search head

To activate distributed search, you add search peers, or indexers, to a Splunk Enterprise instance that you designate as a search head. You do this by specifying each search peer manually.

Important: A search head cannot perform a dual function as a search peer. The only exception to this rule is for the monitoring console, which functions as a "search head of search heads."

This topic describes how to connect a search head to a set of search peers.

If you need to connect multiple search heads to a set of search peers, you can repeat the process for each search head individually. However, if you require multiple search heads, the best practice is to deploy them in a search head cluster. A search head cluster can also replicate all search peers from one search head to all the other search heads in the cluster, so that you do not have to add the peers to each search head separately.

Important: Clusters establish connectivity between search heads and search peers differently from the procedures described in this topic:

• Indexer clusters automatically establish the connection between their search heads and indexers, or peer nodes. To learn how to configure search heads in indexer clusters, read Configure the search head in the Managing Indexers and Clusters of Indexers manual.
• Search head clusters have certain restrictions that you must consider when connecting search heads to search peers. See Connect the search heads in clusters to search peers.

Configuration overview

To set up the connection between a search head and its search peers, configure the search head through one of these methods:

• Splunk Web
• Splunk CLI
• The distsearch.conf configuration file

Splunk Web is the simplest method for most purposes.
The configuration occurs on the search head. For most deployments, no configuration is necessary on the search peers. Access to the peers is controlled through public key authentication.

**Prerequisites**

Before an indexer can function as a search peer, you must change its password from the default value. Otherwise, the search head will not be able to authenticate against it.

**Use Splunk Web**

*Specify the search peers*

To specify the search peers:

1. Log into Splunk Web on the search head and click Settings at the top of the page.

2. Click Distributed search in the Distributed Environment area.

3. Click Search peers.


5. Specify the search peer, along with any authentication settings.

**Note:** You must precede the search peer's host name or IP address with the URI scheme, either "http" or "https".

6. Click Save.

7. Repeat for each of the search head's search peers.

*Configure miscellaneous distributed search settings*

To configure other settings:

1. Log into Splunk Web on the search head and click Settings at the top of the page.

2. Click Distributed search in the Distributed Environment area.
3. Click Distributed search setup.

5. Change any settings as needed.

6. Click Save.

Use the CLI

To add a search peer, run this command from the search head:

```bash
splunk add search-server <scheme>://<host>:<port> -auth <user>:<password> -remoteUsername <user> -remotePassword <passremote>
```

Note the following:

- `<scheme>` is the URI scheme: "http" or "https".
- `<host>` is the host name or IP address of the search peer’s host machine.
- `<port>` is the management port of the search peer.
- Use the `-auth` flag to provide credentials for the search head.
- Use the `-remoteUsername` and `-remotePassword` flags for the credentials for the search peer. The remote credentials must be for an admin-level user on the search peer.

For example:

```bash
```

You must run this command for each search peer that you want to add.

Edit distsearch.conf

The settings available through Splunk Web provide sufficient options for most configurations. Some advanced configuration settings, however, are only available by directly editing distsearch.conf. This section discusses only the configuration settings necessary for connecting search heads to search peers. For information on the advanced configuration options, see the distsearch.conf spec file.

Add the search peers

To connect the search peers:
1. On the search head, create or edit a `distsearch.conf` file in `$_SPLUNK_HOME/etc/system/local`.

2. Add the search peers to the `servers` setting under the `[distributedSearch]` stanza. Specify the peers as a set of comma-separated values (host names or IP addresses with management ports). For example:

   ```
   [distributedSearch]
   ```

   **Note:** You must precede the host name or IP address with the URI scheme, either "http" or "https".

3. Restart the search head.

**Distribute the key files**

If you add search peers via Splunk Web or the CLI, Splunk Enterprise automatically configures authentication. However, if you add peers by editing `distsearch.conf`, you must distribute the key files manually. After adding the search peers and restarting the search head, as described above:

1. Copy the file `$_SPLUNK_HOME/etc/auth/distServerKeys/trusted.pem` from the search head to `$_SPLUNK_HOME/etc/auth/distServerKeys/<searchhead_name>/trusted.pem` on each search peer.

   The `<searchhead_name>` is the search head's `serverName`, specified in `server.conf`.

2. Restart each search peer.

**Authentication of multiple search heads from a single peer**

Multiple search heads can search across a single peer. The peer must store a copy of each search head's certificate.

The search peer stores the search head keys in directories with the specification `$_SPLUNK_HOME/etc/auth/distServerKeys/<searchhead_name>`. For example, if you have two search heads, named A and B, and they both need to search one particular search peer, do the following:
1. On the search peer, create the directories
   $SPLUNK_HOME/etc/auth/distServerKeys/A/ and $SPLUNK_HOME/etc/auth/distServerKeys/B/.

2. Copy A's trusted.pem file to $SPLUNK_HOME/etc/auth/distServerKeys/A/ and B's trusted.pem to $SPLUNK_HOME/etc/auth/distServerKeys/B/.

3. Restart the search peer.

Group the search peers

You can group search peers into distributed search groups. This allows you to target searches to subsets of search peers. See Create distributed search groups.

View search peer status

See View search peer status in Settings.

Best practice: Forward search head data to the indexer layer

It is considered a best practice to forward all search head internal data to the search peer (indexer) 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 search head if it goes down. The data leading up to the failure is accumulated on the indexers, where another search head can later access it.
- By forwarding the results of summary index searches to the indexer level, all search heads have access to them. Otherwise, they're only available to the search head that generates them.

Forward search head data

The preferred approach is to forward the data directly to the indexers, without indexing separately on the search head. You do this by configuring the search head as a forwarder. These are the main steps:
1. **Make sure that all necessary indexes exist on the indexers.** For example, the S.o.S app uses a scripted input that puts data into a custom index. If you install S.o.S on the search head, you need to also install the S.o.S Add-on on the indexers, to provide the indexers with the necessary index settings for the data the app generates. On the other hand, since _audit and _internal exist on indexers as well as search heads, you do not need to create separate versions of those indexes to hold the corresponding search head data.

2. **Configure the search head as a forwarder.** Create an outputs.conf file on the search head that configures the search head for load-balanced forwarding across the set of search peers (indexers). You must also turn off indexing on the search head, so that the search head does not both retain the data locally as well as forward it to the search peers.

Here is an example outputs.conf file:

```bash
# Turn off indexing on the search head
[indexAndForward]
index = false

[tcpout]
defaultGroup = my_search_peers
 forwardedindex.filter.disable = true
 indexAndForward = false

[tcpout:my_search_peers]
server=10.10.10.1:9997,10.10.10.2:9997,10.10.10.3:9997
```

This example assumes that each indexer's receiving port is set to 9997.

For details on configuring outputs.conf, read "Configure forwarders with outputs.conf" in the Forwarding Data manual.

**Forward data from search head cluster members**

You perform the same configuration steps to forward data from search head cluster members to their set of search peers. However, you must ensure that all members use the same outputs.conf file. To do so, do not edit the file on the individual search heads. Instead, use the deployer to propagate the file across the cluster. See "Use the deployer to distribute apps and configuration updates."
Manage distributed search

Manage distributed server names

The name of each search head and search peer is determined by its `servername` attribute, specified in server.conf. The `servername` attribute defaults to the server’s machine name.

In distributed search, all search heads and search peers in the group must have unique names. The `servername` has three specific uses in distributed search:

- **For authenticating search heads.** When search peers are authenticating a search head, they look for the search head's key file in `/etc/auth/distServerKeys/<searchhead_name>/trusted.pem`.
- **For identifying search peers in search queries.** `servername` is the value of the `splunk_server` field that you specify when you want to query a specific node. See Search across one or more distributed search peers in the Search manual.
- **For identifying search peers in search results.** `servername` gets reported back in the `splunk_server` field.

**Note:** `servername` is *not* used when adding search peers to a search head. In that case, you identify the search peers through their domain names or IP addresses.

The only reason to change `servername` is if you have multiple instances of Splunk Enterprise residing on a single machine, and they're participating in the same distributed search group. In that case, you'll need to change `servername` to distinguish them.

Create distributed search groups

You can group your search peers to facilitate searching on a subset of them. Groups of search peers are known as "distributed search groups." You specify distributed search groups in the `distsearch.conf` file.

For example, say you have a set of search peers in New York and another set in San Francisco, and you want to perform searches across peers in just a single location. You can do this by creating two search groups, NYC and SF. You can
then specify the search groups in searches.

Distributed search groups are particularly useful when configuring the monitoring console. See *Monitoring Splunk Enterprise*.

**Configure distributed search groups**

You define distributed search groups in `distsearch.conf`.

For example, to create the two search groups NYC and SF, create stanzas like these:

```plaintext
[distributedSearch]
# This stanza lists the full set of search peers.
servers = 192.168.1.1:8089, 192.168.1.2:8089, 175.143.1.1:8089,
175.143.1.2:8089, 175.143.1.3:8089

[distributedSearch:NYC]
# This stanza lists the set of search peers in New York.
default = false
servers = 192.168.1.1:8089, 192.168.1.2:8089

[distributedSearch:SF]
# This stanza lists the set of search peers in San Francisco.
default = false
servers = 175.143.1.1:8089, 175.143.1.2:8089, 175.143.1.3:8089
```

Note the following:

- The `servers` attribute lists groups of search peers by IP address and management port.
- The `servers` list for each search group must be a subset of the list in the general `[distributedSearch]` stanza.
- The group lists can overlap. For example, you can add a third group named "Primary_Indexers" that contains some peers from each location.
- If you set a group's `default` attribute to "true," the peers in that group will be the ones queried when the search does not specify a search group. Otherwise, if you set all groups to "false," the full set of search peers in the `[distributedSearch]` stanza will be queried when the search does not specify a search group.

**Use distributed search groups**

To use a search group in a search, specify the search group like this:
Distributed search groups and indexer clusters

This feature is not valid for indexer clustering, except for limited use cases in certain complex topologies.

In indexer clustering, the cluster replicates the data buckets arbitrarily across the set of search peers, or "cluster peer nodes". It then assigns one copy of each bucket to be the primary copy, which participates in searches. There is no guarantee that a specific peer or subset of peers will contain the primary bucket copies for a particular search. Therefore, if you put peers into distributed search groups and then run searches based on those groups, the searches might contain incomplete results.

For details of bucket replication in indexer clusters, see Buckets and indexer clusters in Managing Indexers and Clusters of Indexers.

These are some examples of indexer cluster deployments where distributed search groups might be of value:

- Multiple indexer clusters, where you need to identify the peer nodes for a specific cluster.
- Search heads that run searches across both an indexer cluster and standalone indexers. You might want to put the standalone indexers into their own group.

Remove a search peer

You can remove a search peer from a search head through Splunk Web or the CLI. As you might expect, doing so merely removes the search head's knowledge of that search peer; it does not affect the peer itself.

Remove a search peer via Splunk Web

You can remove a search peer from a search head through the Search peers page on the search head's Splunk Web. See View search peer status in Settings.
**Note:** This only removes the search peer entry from the search head; it does not remove the search head key from the search peer. In most cases, this is not a problem and no further action is needed.

**Remove a search peer via the CLI**

On the search head, run the `splunk remove search-server` command to remove a search peer from the search head:

```
splunk remove search-server -auth <user>:<password> <host>:<port>
```

Note the following:

- Use the `-auth` flag to provide credentials for the search head only.
- `<host>` is the host name or IP address of the search peer's host machine.
- `<port>` is the management port of the search peer.

For example:

```
splunk remove search-server -auth admin:password 10.10.10.10:8089
```

A message indicating success appears after the peer is removed.

In the case of a search head cluster, the peer removal action replicates to all other cluster members only if you have enabled **search peer replication**. Otherwise, you must remove the search peers from each member individually. For information on enabling search peer replication, see Replicate the search peers across the cluster.

**Disable the trust relationship**

As an additional step, you can disable the trust relationship between the search peer and the search head. To do this, delete the `trusted.pem` file from `$SPLUNK_HOME/etc/auth/distServerKeys/<searchhead_name>` on the search peer.

**Note:** The `<searchhead_name>` is the search head's `serverName`, as described in "Manage distributed server names".

This step is usually unnecessary.
View distributed search status

View search peer status in Settings

After you add search peers to the search head, you can view the search peers’ status in Settings:

1. On the search head, click Settings at the top of the Splunk Web page.

2. Click Distributed search in the Distributed Environment area.

3. Click Search peers.

There is a row for each search peer, with the following columns:

- **Peer URI**
- **Splunk instance name**
- **State.** Specifies whether the peer is up or down.
- **Replication status.** Indicates the status of knowledge bundle replication between the search head and the search peer:
  - **Initial.** Default state of the peer, before the peer has received its first knowledge bundle from this search head. The peer remains in this state for approximately \( \text{replication\_period\_sec} \) in \( \text{limits.conf} \), which is 60 seconds by default.
  - **In Progress.** A bundle replication is in progress.
  - **Successful.** The peer has received a bundle from this search head. The peer is ready to participate in distributed searches.
  - **Failed.** Something went wrong with bundle replication.
- **Cluster label.** This field contains a value if this peer is part of an indexer cluster and the indexer cluster has a label. See Set cluster labels in Monitoring Splunk Enterprise.
- **Health status.** When the search head sends a heartbeat to a peer (by default, every 60 seconds), it performs a series of health checks on that peer. The results determine the health status of the peer:
  - **Healthy.** The peer passes all health checks during 50% or more of the heartbeats over the past 10 minutes.
  - **Sick.** The peer fails a health check during more than 50% of the heartbeats over the past 10 minutes. See the Health check failures column for details.
  - **Quarantined.** A peer that does not currently participate in distributed searches. See Quarantine a search peer.
• **Health check failures.** This column provides details of any health check failures. It lists all failures over the last 10 minutes. Each heartbeat-timed set of health checks stops at the first health check failure, so the list includes only the first failure, if any, for each heartbeat.
• **Status.** Enabled or disabled.
• **Actions.** You can quarantine this peer or delete it from the search head. See [Quarantine a search peer](#) and [Remove a search peer](#).

You can also use the monitoring console to get information about the search peers. See [Use the monitoring console to view distributed search status](#).

### Use the monitoring console to view distributed search status

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

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

There are two distributed search dashboards under the **Search** menu:

- Distributed Search: Instance
- Distributed Search: Deployment

These dashboards provide detailed information on a variety of issues, such as:

- The health of the peer nodes
- The health of the search heads
- The knowledge bundle replication process
- The dispatch directories on the search heads

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

You can also use Settings to get information about the search peers. See [View search peers in Settings](#).
Manage parallel reduce search processing

Overview of parallel reduce search processing

High-cardinality searches are searches that must match, filter, and aggregate extremely large numbers of unique field values. User IDs, session IDs, and telephone numbers are examples of fields that tend to be high in cardinality. Searches that compute aggregates over high-cardinality fields can be slow to complete. If high-cardinality searches in your Splunk platform deployment are slow, you can use parallel reduce search processing to help them complete quicker.

In a typical distributed search process, there are two broad search processing phases: a map phase and a reduce phase. The map phase takes place across the indexers in your deployment. In the map phase, the indexers locate event data that matches the search query and sort it into field-value pairs. When the map phase is complete, indexers send the results to the search head for the reduce phase. During the reduce phase, the search heads process the results through the commands in your search and aggregate them to produce a final result set.

The following diagram illustrates the standard two-phase distributed search process.

The parallel reduce process inserts an intermediate reduce phase into the map-reduce paradigm, making it a three-phase map-reduce-reduce operation. In this intermediate reduce phase, a subset of your indexers serve as intermediate reducers. The intermediate reducers divide up the mapped results and perform reduce operations on those results for certain supported search commands. When the intermediate reducers complete their work, they send the results to the search head, where the final result reduction and aggregation operations take place. The parallel processing of reduction work that otherwise would be done
entirely by the search head can result in faster completion times for high-cardinality searches that aggregate large numbers of search results.

The following diagram illustrates the three-phase parallel reduce search process.

### Parallel reduce prerequisites

To enable parallel reduce search processing, you need the following prerequisites in place:

<table>
<thead>
<tr>
<th>Prerequisite</th>
<th>Details</th>
<th>For more information see</th>
</tr>
</thead>
<tbody>
<tr>
<td>A distributed search environment.</td>
<td>Parallel reduce search processing requires a distributed search deployment architecture.</td>
<td>About distributed search</td>
</tr>
<tr>
<td>An environment where the indexers are at a single site.</td>
<td>Parallel reduce search processing is not site-aware. Do not use it if your indexers are in a multisite indexer cluster, or if you have non-clustered indexers spread across several sites.</td>
<td></td>
</tr>
<tr>
<td>Splunk platform version 7.1.0 or later for all participating machines.</td>
<td>Upgrade all Splunk instances that participate in the parallel reduce process to version 7.1.0 or later. Participating instances include all indexers and search heads.</td>
<td>How to upgrade Splunk Enterprise in the Installation Manual</td>
</tr>
<tr>
<td>Internal search head data forwarded to the indexer layer.</td>
<td>The parallel reduce search process ignores all data on the search head. If you plan to run</td>
<td>Best Practice: Forward search head data to</td>
</tr>
</tbody>
</table>

![Three-Phase Parallel Reduce Search Process Diagram](image-url)
### Prerequisite Details

<table>
<thead>
<tr>
<th>Prerequisite</th>
<th>Details</th>
<th>For more information see</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel reduce searches, the best practice is to forward all search head data to the indexer layer.</td>
<td><strong>A low to medium average indexer load.</strong> Parallel reduce search processes add a significant amount of indexer load. If you attempt to run parallel reduce searches in an already overloaded indexer system, you might encounter slow performance. If you run an indexer cluster, you might see skipped heartbeats between peer nodes and the cluster master.</td>
<td>the indexer layer</td>
</tr>
<tr>
<td><strong>All indexers configured to allow secure communication with intermediate reducers.</strong> Admins must set an identical pass4SymmKey security key in the [parallelreduce] stanza of server.conf for all indexers. This security key enables communication between indexers and intermediate reducers.</td>
<td></td>
<td>See Use the monitoring console to view index and volume status, in <em>Managing Indexers and Clusters of Indexers</em></td>
</tr>
<tr>
<td><strong>Users with roles that include the run_multi_phased_searches capability.</strong> Users must have the run_multi_phased_searches capability to use the redistribute command. The redistribute command applies parallel reduce search processing to a search.</td>
<td></td>
<td>Configure your indexers to communicate with intermediate reducers</td>
</tr>
</tbody>
</table>

### Next steps

Learn how to configure your deployment for parallel reduce search processing. See [Configure parallel reduce search processing](https://www.example.com/parallel_reduce_searches).
Configure parallel reduce search processing

To enable parallel reduce search processing for your deployment, you need to configure your indexers to work as intermediate reducers and determine how your deployment should distribute the parallel reduction workload across your indexers.

If this is your first time reading about this feature, see Overview of parallel reduce search processing for an overview of parallel reduce search processing and a list of prerequisites.

Configure your indexers to work as intermediate reducers

To gain the benefits of parallel reduce search processing, you must configure all of your indexers so that they have the potential to work as intermediate reducers. You accomplish this configuration by giving each of your indexers an identical pass4SymmKey security key. This security key enables secure communication between indexers and intermediate reducers.

To update your indexer configurations, you must have access to the server.conf file for your Splunk deployment, located in $SPLUNK_HOME/etc/system/local/. See About configuration files and the topics that follow it in the Admin Manual for more information about making configuration file updates.

Parallel reduce search processing is not site-aware. Do not add this configuration to your indexers if they are in a multisite indexer cluster or if they are non-clustered and spread across several sites.

Set a security key for your intermediate reducers

Place a pass4SymmKey security key in a [parallelreduce] stanza for each indexer configuration in server.conf. The security key value must be identical for each indexer. It secures communication between the indexers and the intermediate reducers in your deployment.

Your indexer configurations might already have pass4SymmKey values under their [general] and [clustering] stanzas. Do not change those pass4SymmKey settings. Do not use the same security key values as those pass4SymmKey settings.

Save a copy of the key. After you set the key for an indexer and reboot the indexer, the security key changes from clear text to encrypted form, and it is no
longer recoverable from server.conf. If you add a new intermediate reducer later, you must use the clear text version of the key to set it.

Prerequisites

The following prerequisite topics are useful if you run an indexer cluster.

- Secure your clusters with pass4SymmKey, in Securing Splunk Enterprise. Learn how pass4SymmKey is also used to authenticate communications between members of indexer clusters and search head clusters.
- Configure the indexer cluster with server.conf and Configure peer nodes with server.conf, in Managing Indexers and Clusters of Indexers. Learn how to update configurations for individual indexers.

Steps

1. Open server.conf and locate the settings for an indexer. Indexers are identified with a [hostname:<port>] stanza.
2. Add the following stanza and security configuration to the settings for the indexer:

   [parallelreduce]
   pass4SymmKey=<password>

3. Save your server.conf changes.
4. Restart the indexer with the CLI restart command:

   $SPLUNK_HOME/bin/splunk restart

Repeat these steps for each indexer in your deployment. Use the same <password> for each indexer in your deployment.

Determine how your parallel reduction workload is distributed

Settings in the [parallelreduce] stanza of limits.conf determine the number of intermediate reducers that are selected from your indexers for a parallel reduce search process. They also determine how parallel reduce search processing work is distributed across your indexers.

For example, if you keep the default parallel reduce settings in limits.conf, the Splunk platform randomly selects a certain number of intermediate reducers each time you run a parallel reduce search. If all of your indexers are in a single-site indexer cluster, the random selection aids in distributing the parallel reduction workload across the cluster.
However, if your indexers are not clustered, and some of your indexers have large indexing loads on average while others do not, you can use the reducers setting to configure the low-load indexers to be dedicated intermediate reducers. Dedicated intermediate reducers are always used when you run a parallel reduce search process.

These two methods are mutually exclusive. When you set up dedicated intermediate reducers, the Splunk platform cannot randomly select intermediate reducers.

To configure parallel reduce search processing, you must have access to the limits.conf file for your Splunk deployment, located in $SPLUNK_HOME/etc/system/local/. See About configuration files and the topics that follow it in the Admin Manual for more information about making configuration file updates.

**Enable random selection of intermediate reducers**

Random selection of indexers for intermediate reduction service is ideal if you are running a single-site indexer cluster. If you run several parallel reduce searches concurrently, the random selection ensures that the intermediate reduction work is evenly distributed across the cluster.

The default parallel reduce search processing settings enable the Splunk platform to randomly select intermediate reducers from the larger set of indexers when you run parallel reduce searches. The default number of indexers that the Splunk platform repurposes as intermediate reducers during the intermediate reduce phase of the parallel reduce search process is 50% of the total number of indexers in your indexer pool, up to a maximum of 4 indexers.

Random intermediate reducer selection is determined by the maxReducersPerPhase and winningRate settings. They belong to the [parallelreduce] stanza of limits.conf.

<table>
<thead>
<tr>
<th>Setting name</th>
<th>Definition</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxReducersPerPhase</td>
<td>The maximum number of indexers that can be used as intermediate reducers in the intermediate reduce phase of a parallel reduce search.</td>
<td>4</td>
</tr>
<tr>
<td>winningRate</td>
<td>The percentage of indexers that can be selected from the total pool of indexers and</td>
<td>50</td>
</tr>
</tbody>
</table>
Enabling dedicated intermediate reducers

To configure a set of non-clustered indexers as dedicated intermediate reducers, add the `reducers` setting to the `[parallelreduce]` stanza in `limits.conf`.

The value of `reducers` is a comma-separated list of indexers that you have configured as search peers. Identify each indexer by specifying its host and port using the following format: `<host>:<port>`. For example:

```
reducers=docteam-unix-4:8089, docteam-unix-5:8089, docteam-unix-6:8089
```

Do not include clustered indexers on the `reducers` list.

All indexers in the `reducers` list are used as intermediate reducers when you run a parallel reduce search. If the number of indexers in the `reducers` list exceeds the value of the `maxReducersPerPhase` setting, the Splunk platform randomly selects the intermediate reducers from the `reducers` list. For example, if the `reducers` setting lists five reducers and `maxReducersPerPhase=4`, the Splunk platform randomly selects four intermediate reducers from the list.

If all of the indexers in the `reducers` list are down or are otherwise invalid, searches with the `redistribute` command run without parallel reduction. All reduce operations are processed on the search head.

When you configure the `reducers` setting for your deployment, the Splunk platform ceases to apply the `winningRate` setting.

**Override the number of reducers for a specific search**

When you run a parallel reduce search with the `redistribute` command, you can use the `num_ofReducers` argument to override the number of reducers determined by the parallel reduce search settings in the `limits.conf` file.

For example, say your `limits.conf` settings determine that seven intermediate reducers are used by default in all parallel reduce searches. You can design a

---

<table>
<thead>
<tr>
<th>Setting name</th>
<th>Definition</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>used as intermediate reducers in a parallel reduce search process. This setting applies only when the <code>reducers</code> setting is not configured in <code>limits.conf</code>. See <strong>Enable dedicated intermediate reducers</strong>.</td>
<td></td>
</tr>
</tbody>
</table>
parallel reduce search where \( \text{num\_of\_reducers} = 5 \). Every time that search runs, only five intermediate reducers are used in its intermediate reduce phase.

If you provide a value for the \( \text{num\_of\_reducers} \) setting that exceeds the limit set by the \( \text{maxReducersPerPhase} \) setting in the \text{limits.conf} file, the Splunk platform sets the number of reducers to the \( \text{maxReducersPerPhase} \) value.

**Next steps**

Use the \text{redistribute} command to apply parallel reduce search processing to your high-cardinality searches. See Apply parallel reduce processing to searches.

**Apply parallel reduce processing to searches**

If you have configured parallel reduce search processing for your deployment, you can use the \text{redistribute} command to apply it to your high-cardinality searches, so they can complete faster.

If this is your first time reading about this feature, see Overview of parallel reduce search processing for an overview of parallel reduce search processing and a list of prerequisites.

To configure your deployment to use this functionality, see Configure parallel reduce search processing.

**Use the \text{redistribute} command**

Use the \text{redistribute} command in a high-cardinality search to give that search the benefit of parallel reduce search processing. Only users with roles that have the \text{run\_multi\_phased\_searches} capability can use \text{redistribute}.

The \text{redistribute} command supports only \textbf{streaming commands} and the following nonstreaming commands: \texttt{stats}, \texttt{tstats}, \texttt{streamstats}, \texttt{eventstats}, \texttt{sichart}, \texttt{sitimechart}, and \texttt{transaction}.

See \text{redistribute} in the \textit{Search Reference}. 43
About the run_multi_phased_searches capability

The run_multi_phased_searches capability is not assigned to any role by default. As a best practice, we suggest that you create a specialized role for this capability and assign it only to users who can be trusted to run reasonable numbers of parallel reduce searches when overall indexer load is low.

See About defining roles with capabilities in Securing Splunk Enterprise.

Concurrent parallel reduce searches

By default, the number of concurrent parallel reduce searches that can run on an intermediate reducer is limited to the number of CPU cores in the reducer. This default is controlled by the maxPrdSearchesPerCpu setting in limits.conf.

If the number of concurrent parallel reduce search processes running on your intermediate reducers exceeds the number of cores in your reducers, you might lose the search performance gains that parallel reduce search processing is designed to deliver. If you cannot lower your average number of concurrent parallel reduce search processes, you can disable the useClientSSLCompression setting in server.conf on your search heads and intermediate reducers. This should restore the lost parallel reduce search performance.

Disabling useClientSSLCompression causes the bundle replication process to require additional network bandwidth. If you depend on efficient bundle replication do not disable this setting.

To disable or enable useClientSSLCompression, you must have access to the limits.conf file for your Splunk deployment, located in $SPLUNK_HOME/etc/system/local/. See About configuration files and the topics that follow it in the Admin Manual for more information about making configuration file updates.
Overview of search head clustering

About search head clustering

A search head cluster is a group of Splunk Enterprise search heads that serves as a central resource for searching. The members of a search head cluster are essentially interchangeable. You can run the same searches, view the same dashboards, and access the same search results from any member of the cluster.

To achieve this interchangeability, the search heads in the cluster must share configurations and apps, search artifacts, and job scheduling. Search head clusters automatically propagate most of these shared resources among the members.

Benefits of a search head cluster

Search head clusters provide these key benefits:

- **Horizontal scaling.** As the number of users and the search load increases, you can add new search heads to the cluster. By combining a search head cluster with a third-party load balancer placed between users and the cluster, the topology can be transparent to the users.
- **High availability.** If a search head goes down, you can run the same set of searches and access the same set of search results from any other search head in the cluster.
- **No single point of failure.** The search head cluster uses a dynamic captain to manage the cluster. If the captain goes down, another member automatically takes over management of the cluster.

Cluster architecture

A search head cluster consists of a group of networked search heads, called cluster members. One cluster member, the captain, coordinates all cluster-wide activities. If the member serving as captain goes down, another member takes its place.

The members share:

- **Job scheduling.** The cluster manages job scheduling centrally, allocating
each scheduled search to the optimal member, usually the member with
the least load.

- **Search artifacts.** The cluster replicates search artifacts and makes them
  available to all members.
- **Configurations.** The cluster requires that all members share the same set
  of configurations. For runtime updates to knowledge objects, such as
  updates to dashboards or reports, the cluster replicates configurations
  automatically to all members. For apps and some other configurations, the
  user must push configurations to the cluster members by means of the
  **deployer**, a Splunk Enterprise instance that resides outside the cluster.

See "Search head clustering architecture."

**How to set up the cluster**

You set up a cluster by configuring and deploying the cluster's search heads. The
process is similar to how you set up search heads in any distributed search
environment. The main difference is that you also need to configure the search
heads as cluster members.

See the chapter "Deploy search head clustering".

**How the user accesses the cluster**

Users access the cluster the same way that they access any search head. They
point their browser at any search head that is a member of the cluster. Because
cluster members share jobs, search artifacts, and configurations, it does not
matter which search head a user accesses. The user has access to the same set
of dashboards, searches, and so on.

To achieve the goals of high availability and load balancing, Splunk recommends
that you put a load balancer in front of the cluster. That way, the load balancer
can assign the user to any search head in the cluster and balance the user load
across the cluster members. If one search head goes down, the load balancer
can reassign the user to any remaining search head.

**Search head clusters and indexer clusters**

Search head clusters are different from **indexer clusters**. The primary purpose
of indexer clusters is to provide highly available data through coordinated groups
of indexers. Indexer clusters always include one or more associated search
heads to access the data on the indexers. These search heads might be, but are
not necessarily, members of a search head cluster.

For information on search heads in indexer clusters, see the chapter "Configure the search head" in the Managing Indexers and Clusters of Indexers manual.

For information on adding a search head cluster to an indexer cluster, see the topic "Integrate the search head cluster with an indexer cluster" in this manual.

**Search head clustering architecture**

A search head cluster is a group of Splunk Enterprise search heads that serves as a central resource for searching.

**Parts of a search head cluster**

A search head cluster consists of a group of search heads that share configurations, job scheduling, and search artifacts. The search heads are known as the cluster members.

One cluster member has the role of captain, which means that it coordinates job scheduling and replication activities among all the members. It also serves as a search head like any other member, running search jobs, serving results, and so on. Over time, the role of captain can shift among the cluster members.

In addition to the set of search head members that constitute the actual cluster, a functioning cluster requires several other components:

- **The deployer.** This is a Splunk Enterprise instance that distributes apps and other configurations to the cluster members. It stands outside the cluster and cannot run on the same instance as a cluster member. It can, however, under some circumstances, reside on the same instance as some other Splunk Enterprise components, such as a deployment server or an indexer cluster master node. See Use the deployer to distribute apps and configuration updates.
- **Search peers.** These are the indexers that cluster members run their searches across. The search peers can be either independent indexers or nodes in an indexer cluster. See Connect the search heads in clusters to search peers.
- **Load balancer.** This is third-party software or hardware optionally residing between the users and the cluster members. With a load balancer in place, users can access the set of search heads through a single
interface, without needing to specify a particular search head. See Use a load balancer with search head clustering.

Here is a diagram of a small search head cluster, consisting of three members:

![Diagram of a small search head cluster](image)

This diagram shows the key cluster-related components and interactions:

- One member serves as the captain, directing various activities within the cluster.
- The members communicate among themselves to schedule jobs, replicate artifacts, update configurations, and coordinate other activities within the cluster.
- The members communicate with search peers to fulfill search requests.
- Users can optionally access the search heads through a third-party load balancer.
- A deployer sits outside the cluster and distributes updates to the cluster members.

**Note:** This diagram is a highly simplified representation of a set of complex interactions between components. For example, each cluster member sends search requests directly to the set of search peers. On the other hand, only the captain sends the knowledge bundle to the search peers. Similarly, the diagram does not attempt to illustrate the messaging that occurs between cluster members. Read the text of this topic for the details of all these interactions.

**Search head cluster captain**

The captain is a cluster member with additional responsibilities, beyond the search activities common to all cluster members. It serves to coordinate the
activities of the cluster. Any member can perform the role of captain, but the cluster has just one captain at any time. Over time, if failures occur, the captain changes and a new member gets elected to the role.

The elected captain is known as a **dynamic captain**, because it can change over time. A cluster that is functioning normally uses a dynamic captain. You can deploy a **static captain** as a temporary workaround during disaster recovery, if the cluster is not able to elect a dynamic captain.

### Role of the captain

The captain is a cluster member and in that capacity it performs the search activities typical of any cluster member, servicing both ad hoc and scheduled searches. If necessary, you can limit the captain's search activities so that it performs only ad hoc searches and not scheduled searches. See Configure the captain to run ad hoc searches only.

The captain also coordinates activities among all cluster members. Its responsibilities include:

- Scheduling jobs. It assigns jobs to members, including itself, based on relative current loads.
- Coordinating alerts and alert suppressions across the cluster. The captain tracks each alert but the member running an initiating search fires it.
- Pushing the knowledge bundle to search peers.
- Coordinating artifact replication. The captain ensures that search artifacts get replicated as necessary to fulfill the replication factor. See Choose the replication factor for the search head cluster.
- Replicating configuration updates. The captain replicates any runtime changes to knowledge objects on one cluster member to all other members. This includes, for example, changes or additions to saved searches, lookup tables, and dashboards. See Configuration updates that the cluster replicates.

### Captain election

A search head cluster normally uses a dynamic captain. This means that the member serving as captain can change over the life of the cluster. Any member has the ability to function as captain. When necessary, the cluster holds an election, which can result in a new member taking over the role of captain.

Captain election occurs when:
The current captain fails or restarts.
A network partition occurs, causing one or more members to get cut from
the rest of the search head cluster. Subsequent healing of the network
partition triggers another, separate captain election.
The current captain steps down, because it does not detect that a majority
of members are participating in the cluster.

**Note:** The mere failure or restart of a non-captain cluster member, without an
associated network partition, does not trigger captain election.

To become captain, a member needs to win a majority vote of all members. For
example, in a seven-member cluster, election requires four votes. Similarly, a
six-member cluster also requires four votes.

The majority must be a majority of all members, not just of the members currently
running. So, if four members of a seven-member cluster fail, the cluster cannot
elect a new captain, because the remaining three members are fewer than the
required majority of four.

The election process involves timers set randomly on all the members. The
member whose timer runs out first stands for election and asks the other
members to vote for it. Usually, the other members comply and that member
becomes the new captain.

It typically takes one to two minutes after a triggering event occurs to elect a new
captain. During that time, there is no functioning captain, and the search heads
are aware only of their local environment. The election takes this amount of time
because each member waits for a minimum timeout period before trying to
become captain. These timeouts are configurable.

The cluster might re-elect the member that was the previous captain, if that
member is still running. There is no bias either for or against this occurring.

Once a member is elected as captain, it takes over the duties of captaincy.

**Important:** A majority of members must be running and participating in the
cluster at all times. If the captain does not detect a majority of members, it steps
down, relinquishing its authority. An election for a new captain will subsequently
occur, but without a majority of participating members, it will not succeed. If you
lose majority on a cluster, a temporary workaround is to deploy a static captain,
in place of the dynamic captain. Static captains are designated by the
administrator, not elected by the members. See Use static captain to recover
from loss of majority.
For details of your cluster's captain election process, view the Search Head Clustering: Status and Configuration dashboard in the monitoring console. See Use the monitoring console to view search head cluster status.

Control of captaincy

You have some control over which members become captain. In particular, you can:

- Set captaincy preference on a member-by-member basis. The cluster attempts to elect as captain a member designated as a preferred captain.
- Transfer captaincy from one member to another.
- Prevent an out-of-sync member from becoming captain. An out-of-sync member is a member that cannot sync its own set of replicated configurations with the common baseline set of replicated configurations maintained by the current or most recent captain. By default, the cluster attempts not to elect as captain an out-of-sync member.

For details on these captaincy control capabilities, see Control captaincy.

Consequences of a non-functioning cluster

If the cluster lacks a majority of members and therefore cannot elect a captain, the members will continue to function as independent search heads. However, they will only be able to service ad hoc searches. Scheduled reports and alerts will not run, because, in a cluster, the scheduling function is relegated to the captain. In addition, configurations and search artifacts will not be replicated during this time.

To remedy this situation, you can temporarily deploy a static captain. See Use static captain to recover from loss of majority.

Recovering from a non-functioning cluster

If you do not deploy a static captain during the time that the cluster lacks a majority, the cluster will not function again until a majority of members rejoin the cluster. When a majority is attained, the members elect a captain, and the cluster starts to function.

There are two key aspects to recovery:

- Runtime configurations
- Scheduled reports
Once the cluster starts functioning, it attempts to sync the runtime configurations of the members. Since the members were able to operate independently during the time that their cluster was not functioning, it is likely that each member developed its own unique set of configuration changes during that time. For example, a user might have created a new saved search or added a new panel to a dashboard. These changes must now be reconciled and replicated across the cluster. To accomplish this, each member reports its set of changes to the captain, which then coordinates the replication of all changes, including its own, to all members. At the end of this process, all members should have the same set of configurations.

Caution: This process can only proceed automatically if the captain and each member still share a common commit in their change history. Otherwise, it will be necessary to manually resync the non-captain member against the captain's current set of configurations, causing that member to lose all of its intervening changes. Configurable purge limits control the change history. For details of purge limits and the resync process, see Replication synchronization issues.

The recovered cluster also begins handling scheduled reports again. As for whether it attempts to run reports that were skipped while the cluster was down, that depends on the type of scheduled report. For the most part, it will just pick up the reports at their next scheduled run time. However, the scheduler will run reports employed by report acceleration and data model acceleration from the point when they were last run before the cluster stopped functioning. For detailed information on how the scheduler handles various types of reports, see Configure the priority of scheduled reports in the Reporting Manual.

Captain election process has deployment implications

The need of a majority vote for a successful election has these deployment implications:

- A cluster must consist of a minimum of three members. A two-member cluster cannot tolerate any node failure. Failure of either member will prevent the cluster from electing a captain and continuing to function. Captain election requires majority (51%) assent of all members, which, in the case of a two-member cluster, means that both nodes must be running. You therefore forfeit the high availability benefits of a search head cluster if you limit the cluster to one or two members.

Note: As an interim measure, when first deploying a search head cluster, you can bring up a single-member cluster. This approach allows you to start with a small distributed search deployment and later scale to a larger
cluster. However, a single-member cluster does not provide high availability search, which is the main benefit of a search head cluster. To fulfill that benefit, the cluster must comprise at least three members. See Deploy a single-member search head cluster.

- If you are deploying the cluster across two sites, your primary site must contain a majority of the nodes. If there is a network disruption between the sites, only the site with a majority can elect a new captain. See Important considerations when deploying a search head cluster across multiple sites.

**How the cluster handles search artifacts**

The cluster replicates most search artifacts, also known as search results, to multiple cluster members. If a member needs to access an artifact, it accesses a local copy, if possible. Otherwise, it uses proxying to access the artifact.

**Artifact replication**

The cluster maintains multiple copies of search artifacts resulting from scheduled saved searches. The replication factor determines the number of copies that the cluster maintains of each artifact. For example, if the replication factor is three, the cluster maintains three copies of each artifact: one on the member that originated the artifact, and two on other members.

The captain coordinates the replication of artifacts to cluster members. As with any search head, clustered or not, when a search is complete, its search artifact is placed in the dispatch directory of the member originating the search. The captain then directs the artifact’s replication process, in which copies stream between members until copies exist on the replication factor number of members, including the originating member.

The set of members receiving copies can change from artifact to artifact. That is, two artifacts from the same originating member might have their replicated copies on different members.

The captain maintains the artifact registry, with information on the locations of copies of each artifact. When the registry changes, the captain sends the delta to each member.

If a member goes down, thus causing the cluster to lose some artifact copies, the captain coordinates fix-up activities, with the goal of returning the cluster to a state where each artifact has the replication factor number of copies.
Search artifacts are contained in the dispatch directory, located under
$SPLUNK_HOME/var/run/splunk/dispatch. Each dispatch subdirectory contains
one search artifact. It is these subdirectories that the cluster replicates.

Replicated search artifacts can be identified by the prefix rsa_. The original
artifacts do not have this prefix.

For details of your cluster's artifact replication process, view the Search Head
Clustering: Artifact Replication dashboard in the monitoring console. See Use the
monitoring console to view search head cluster status.

**Artifact proxying**

The cluster only replicates search artifacts resulting from scheduled saved
searches. It does not replicate results from these other search types:

- Scheduled real-time searches
- Ad hoc searches of any kind (realtime or historical)

Instead, the cluster proxies these results, if they are requested by a
non-originating search head. They appear on the requesting member after a
short delay.

In addition, if a member needs an artifact from a scheduled saved search but
does not itself have a local copy of that artifact, it proxies the results from a
member that does have a copy. At the same time, the cluster replicates a copy of
that artifact to the requesting member, so that it has a local copy for any future
requests. Because of this process, some artifacts might have more than the
replication factor number of copies.

**Distribution of configuration changes**

With a few exceptions, all cluster members must use the same set of
configurations. For example, if a user edits a dashboard on one member, the
updates must somehow propagate to all the other members. Similarly, if you
distribute an app, you must distribute it to all members. Search head clustering
has methods to ensure that configurations stay in sync across the cluster.

There are two types of configuration changes, based on how they are distributed
to cluster members:

- **Replicated changes.** The cluster automatically replicates any runtime
  knowledge object changes on one member to all other members.
• **Deployed changes.** The cluster relies on an external instance, the deployer, to push apps and other non-runtime configuration changes to the set of members. You must initiate each push of changes from the deployer.

See [How configuration changes propagate across the search head cluster](#).

**Job scheduling**

The captain schedules saved search jobs, allocating them to the various cluster members according to load-based heuristics. Essentially, it attempts to assign each job to the member currently with the least search load.

The captain can allocate saved search jobs to itself. It does not, however, allocate scheduled real time searches to itself.

If a job fails on one member, the captain reassigns it to a different member. The captain reassigns the job only once, as multiple failures are unlikely to be resolvable without intervention on the part of the user. For example, a job with a bad search string will fail no matter how many times the cluster attempts to run it.

You can designate a member as "ad hoc only." In that case, the captain will not schedule jobs on it. You can also designate the captain functionality as "ad hoc only." The current captain then will never schedule jobs on itself. Since the role of captain can move among members, this setting ensures that captain functionality does not compete with scheduled searches. See [Configure a cluster member to run ad hoc searches only](#).

**Note:** The captain does not have insight into the actual CPU load on each member's machine. It assumes that all machines in the cluster are provisioned homogeneously, with the same number and type of cores, and so forth.

For details of your cluster's scheduler delegation process, view the Search Head Clustering: Scheduler Delegation dashboard in the monitoring console. See [Use the monitoring console to view search head cluster status](#).

**How the cluster handles concurrent search quotas**

The search head cluster, like non-clustered search heads, enforces several types of concurrent search limits:

• **Scheduler concurrency limit.** This limit is the maximum number of searches that the scheduler can run concurrently. In search head
clustering, a centralized scheduler on the captain handles scheduling for all cluster members. See the limits.conf spec file for details on how scheduler concurrency limits are determined.

- **User/role search quotas.** These quotas determine the maximum number of concurrent historical searches (combined scheduled and ad hoc) allowable for a specific user/role. These quotas are configured with srchJobsQuota and related settings in authorize.conf. See the authorize.conf spec file for details on all the settings that control these quotas.

- **Overall search quota.** This quota determines the maximum number of historical searches (combined scheduled and ad hoc) that the cluster can run concurrently. This quota is configured with max_searches_per_cpu and related settings in limits.conf. See the limits.conf spec file for details on all the settings that control these quotas.

The search head cluster enforces the scheduler concurrency limit on a cluster-wide basis. It enforces the user/role quotas and overall search quota on either a cluster-wide or a member-by-member basis.

**How the cluster enforces the scheduler concurrency limit**

The captain takes the base scheduler concurrency limit, as defined in limits.conf, and multiplies that limit by the number of members able to run scheduled searches. So, for example, given a seven-member cluster in which two members are configured as "ad hoc only," the captain multiplies the base limit by 5 to derive the maximum number of scheduled searches that it can run concurrently on the cluster. The captain only includes members in the "Up" state.

For information on determining the state of a member, see Show cluster status.

For information on "ad hoc only" members, see Configure a cluster member to run ad hoc searches only.

For details on how base scheduler concurrency limits are determined, see the limits.conf spec file.

**How the cluster enforces quotas**

Although each quota type (user/role or overall) has its own attribute for setting its enforcement behavior, the behavior itself works the same for each quota type.

**If you configure the cluster to enforce quotas on a member-by-member basis,** each individual member uses the base quota settings to determine
whether to allow a search to run. No cluster-wide enforcement of searches occurs.

If you configure the cluster to enforce quotas on a cluster-wide basis, the captain determines the search quota by multiplying the base concurrent search quota by the total number of cluster members in the "Up" state. This number includes all "Up" members that are capable of running searches, including those configured as "ad hoc only."

The captain uses the computed cluster-wide quota to determine whether to allow a scheduled search to run. No member-specific enforcement of searches occurs, except in the case of ad hoc searches, as described in Search quotas and ad hoc searches.

In the case of user/role quotas, the captain multiplies the base concurrent search quota allocated to a user/role by the number of "Up" cluster members to determine the cluster-wide quota for that user/role. For example, in a seven-member cluster, it multiplies the value of `srchJobsQuota` by 7 to determine the number of concurrent historical searches for the user/role.

Similarly, in the case of overall search quotas, the captain multiples the base overall search quota by the number of "Up" members to determine the cluster-wide quota for all searches.

When determining the number of cluster-wide concurrent searches, the captain includes both scheduled searches and ad hoc searches running on all members. The captain stops a scheduled search from running if it will cause the number of concurrent searches to exceed the cluster-wide search quota. It does not control the initiation of ad hoc searches, however. For more details on this process, see Search quotas and ad hoc searches.

For details of your cluster’s search concurrency status, view the Search Head Clustering: Status and Configuration dashboard in the monitoring console. See Use the monitoring console to view search head cluster status.

How the captain determines whether to allow a search to run

When determining whether to allow a historical scheduled search to run, the scheduler on the captain follows this order:

1. Does the search exceed the scheduler concurrency limit? If so, the search does not run.
2. In the case of cluster-wide enforcement only, does the search exceed the cluster-wide user/role search quota for the user/role running the search? If so, the search does not run.

3. In the case of cluster-wide enforcement only, does the search exceed the overall search quota? If so, the search does not run.

**Note:** The captain only controls the running of scheduled searches. It has no control over whether ad hoc searches run. Instead, each individual member decides for its own ad hoc searches, based on the individual member search limits. However, the members feed information on their ad hoc searches to the captain, which includes those searches when comparing concurrent searches against the quotas. see Search quotas and ad hoc searches.

**Cluster-wide search quotas and ad hoc searches**

Each search quota spans both scheduled searches and ad hoc searches. Because of the way that the captain learns about ad hoc searches, the number of cluster-wide concurrent searches can sometimes exceed the search quota. This is true for both types of search quotas, user/role quotas and overall quotas.

If, for example, you configure the cluster to enforce the overall search quota on a cluster-wide basis, the captain handles quota enforcement by comparing the total number of searches running across all members to the search quota.

So, to enforce quotas, the captain must know two values:

- The overall search quota
- The number of concurrent searches running across all members

The captain calculates the overall search quota by multiplying the base concurrent search quota by the number of "Up" cluster members, as described in How the cluster enforces quotas.

The captain calculates the number of concurrent searches running across all members by adding together the total number of scheduled and ad hoc searches in progress:

- For scheduled searches, it always knows the number of concurrent scheduled searches, because it controls the search scheduling operation.
- For ad hoc searches, it depends on reporting from the individual members. When a new ad hoc search starts, the member running the search informs the captain, and the captain adds that search to the total
When the number of all searches, both scheduled and ad hoc, reaches the quota, the captain ceases initiating new scheduled searches until the number of searches falls below the quota.

A user always initiates an ad hoc search directly on a member. The member uses its own set of search quotas, without consideration or knowledge of the cluster-wide search quota, to decide whether to allow the search. The member then reports the new ad hoc search to the captain. If the captain has already reached the cluster-wide quota, then a new ad hoc search causes the cluster to temporarily exceed the quota. This results in the captain reporting more searches than the number allowable by the search quota.

Configure quota enforcement behavior

You configure user/role-based quota enforcement behavior separately from overall search quota enforcement behavior.

Configure user/role-based quota enforcement behavior

Configure user/role-based quota enforcement behavior with the shc_role_quota_enforcement setting, under the [scheduler] stanza in limits.conf.

To enforce these quotas on a member-by-member basis, leave this attribute set to false, its default value.

To enforce these quotas on a cluster-wide basis instead, set the attribute to true:

```
shc_role_quota_enforcement=true
```

For details of this setting, see limits.conf.

Configure overall search quota enforcement behavior

Configure overall search quota enforcement behavior with the shc_syswide_quota_enforcement setting, under the [scheduler] stanza in limits.conf.

To enforce this quota on a member-by-member basis, leave this attribute set to false, its default value.
To enforce this quota on a cluster-wide basis instead, set the attribute to true:

```
shc_syswide_quota_enforcement=true
```

For details of this setting, see limits.conf.

**Change to the default behavior** With 6.5, there was a change in the default behavior for enforcing user/role-based concurrent search quotas.

<table>
<thead>
<tr>
<th>Version</th>
<th>Default enforcement</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3-6.4</td>
<td>cluster-wide</td>
</tr>
<tr>
<td>6.5+</td>
<td>member-by-member</td>
</tr>
</tbody>
</table>

**Deciding which scope of quota enforcement to use**

Each approach has its advantages.

**The case for cluster-wide enforcement**

The captain does not take into account the search user when it assigns a search to a member. Combined with member-enforced quotas, this could result in unwanted and unexpected behavior.

One consequence of the member-by-member behavior is this: If the captain happened to assign most of a particular user's searches to one cluster member, that member could quickly reach the quota for that user, even though other members had not yet reached their limit for the user. This could also occur in the case of role-based quotas.

For example, say you have a three-member cluster, and the search concurrency quota for role X is set to 4. At some point, two members are running four searches for X and one is running only two. The scheduler then dispatches a new search for X that lands on a member that is already running four searches. What happens next depends on whether the cluster is enforcing quotas on a member-by-member or cluster-wide basis:

- With member-by-member enforcement, the member sees that it has already reached the member-specific concurrency limit of 4 for role X. Therefore, it does not run the search. However, the consequences are usually minimal because, if one member cannot run a search, the captain retries the job on a different member. You can configure the number of retries with the `server.conf` attribute `remote_job_retry_attempts`.  

60
• With cluster-wide enforcement, the member sees that the cluster-wide concurrency limit for role X is 12 (4 * 3 members), but that, currently, there are only 10 (4 + 4 + 2) searches running for role X. Therefore, it runs the search.

The case for member-by-member enforcement

While cluster-wide enforcement has the advantage of allowing full utilization of the search concurrency quotas across the set of cluster members, it has the potential to cause miscalculations that result in oversubscribing or undersubscribing searches on the cluster.

When the captain enforces the cluster-wide search concurrency quotas, it includes both scheduled and ad hoc searches in its calculations.

This can lead to miscalculations due to network latency issues, because the captain must rely on each member to inform it of any ad hoc searches that it is running. If members are slow in responding to the captain, the captain might not be aware of some ad hoc searches, and thus oversubscribe the cluster.

Similarly, latency can cause members to be slow in informing the captain of completion of searches, scheduled or ad hoc, causing the captain to undersubscribe the cluster.

For these reasons, you might find that your needs are better met by using the member-by-member enforcement method.

Search head clustering and KV store

KV store can reside on a search head cluster. However, the search head cluster does not coordinate replication of KV store data or otherwise involve itself in the operation of KV store. For information on KV store, see About KV store in the Admin Manual.
Deploy search head clustering

System requirements and other deployment considerations for search head clusters

The members of a search head cluster have most of the same system requirements as any non-clustered search head. This topic details requirements specific to a search head cluster.

Summary of key requirements

These are the main issues to note regarding provisioning of cluster members:

- Each member must run on its own machine or virtual machine, and all machines must run the same operating system.
- All members must run on the same version of Splunk Enterprise.
- All members must be connected over a high-speed network.
- For high availability search, you must deploy at least three members.

In addition to the cluster members, you need a deployer to distribute updates to the members. The deployer must run on a non-member instance. In some cases, it can run on the same instance as a deployment server or an indexer cluster master node.

See the remainder of this topic for details on these and other issues.

Hardware and operating system requirements

Machine requirements for cluster members

Each member must run on its own, separate machine or virtual machine.

The hardware requirements for the machine are essentially the same as for any Splunk Enterprise search head. See Reference hardware in the Capacity Planning Manual. The main difference is the need for increased storage to accommodate a larger dispatch directory. See Storage considerations.

Splunk recommends that you use homogeneous machines with identical hardware specifications for all cluster members. The reason is that the cluster captain assigns scheduled jobs to members based on their current job loads.
When it does this, it does not have insight into the actual processing power of each member's machine. Instead, it assumes that each machine is provisioned equally.

**Operating system requirements for cluster members**

Search head 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 search head cluster members and the deployer must run on the same operating system.

If the search head cluster is connected to an indexer cluster, then the indexer cluster instances must run on the same operating system as the search head cluster members.

**Storage considerations**

When determining the storage requirements for your clustered search heads, you need to consider the increased capacity necessary to handle replicated copies of search artifacts.

For the purpose of developing storage estimates, you can observe the size over time of dispatch directories on the search heads in your non-clustered environment, if any, before you migrate to a cluster. Total up the size of dispatch directories across all the non-clustered search heads and then make adjustments to account for the cluster-specific factors.

The most important factor to take into consideration is the replication factor. For example, if you have a replication factor of 3, you will need approximately triple the amount of the total pre-cluster storage, distributed equally among the cluster members.

Other factors can further increase the cluster storage needs. One key factor is the need to plan for node failure. If a member goes down, causing its set of artifacts (original and replicated) to disappear from the cluster, fix-up activities take place to ensure that each artifact once again has its full complement of copies, matching the replication factor. During fix-up, the copies that were resident on the failed member get replicated among the remaining members, increasing the size of each remaining member's dispatch directory.
Other issues can also increase storage on a per-member basis. For example, the cluster does not guarantee an absolutely equal distribution of replicated copies across the members. In addition, the cluster can hold more than the replication factor number of some search artifacts. See How the cluster handles search artifacts.

As a best practice, equip each member machine with substantially more storage than the estimated need. This allows both for future growth and for temporarily increased need resulting from downed cluster members. The cluster will stop running searches if any of its members runs out of disk space.

**Splunk Enterprise instance requirements**

**Splunk Enterprise version compatibility**

You can implement search head clustering on any group of Splunk Enterprise instances, version 6.2 or above.

All cluster members must run the same version of Splunk Enterprise, down to the maintenance level. You must upgrade all members to a new release at the same time. You cannot, for example, run a search head cluster with some members at 6.3.2 and others at 6.3.1.

The deployer must run the same version as the cluster members, down to the minor level. In other words, if the members are running 6.3.2, the deployer must run some version of 6.3.x. It is strongly advised that you upgrade the deployer at the same time that you upgrade the cluster members. See Upgrade a search head cluster.

**Note:** During search head cluster upgrades, the cluster can temporarily include both members at the previous version and members at the new version. By the end of the upgrade process, all members must again run the same version. This is valid only when upgrading from version 6.4 or later. See Upgrade a search head cluster.

7.x search head clusters can run against 6.x or 7.x **search peers**. The search head cluster members must be at the same or a higher level than the search peers. For details on version compatibility between search heads and search peers, see Version compatibility.

**Important:** Search heads participating in indexer clusters have different compatibility restrictions. See Splunk Enterprise version compatibility in Managing Indexers and Clusters of Indexers.
**Licensing requirements**

Licensing needs are the same as for any search head. See Licenses and distributed deployments in the *Admin Manual*.

**Required number of instances**

The cluster must contain at a minimum the number of members needed to fulfill both of these requirements:

- Three members, so that the cluster can continue to function if one member goes down. See *Captain election process has deployment implications*.
- The replication factor number of instances. See *Choose the replication factor for the search head cluster*.

For example, if your replication factor is either 2 or 3, you need at least three instances. If your replication factor is 5, you need at least five instances.

You can optionally add more members to boost search and user capacity.

**Note:** As an interim measure, when first deploying a search head cluster, you can bring up a single-member cluster. This approach allows you to start with a small distributed search deployment and later scale to a larger cluster. However, a single-member cluster does not provide high availability search, which is the main benefit of a search head cluster. To fulfill that benefit, the cluster must comprise at least three members. See *Deploy a single-member search head cluster*.

**Maximum number of instances**

Search head clustering supports up to 100 members in a single cluster.

**Search head clusters running across multiple sites**

Although there is no formal notion of a multisite search head cluster, you can still deploy the cluster members across multiple sites.

When deploying the cluster across multiple sites, put a majority of the cluster members on the site that you consider primary. This ensures that the cluster can continue to elect a captain, and thus continue to function, as long as the primary site is running. See *Deploy a search head cluster in a multisite environment*.


**Cluster member cannot be a search peer**

A cluster member cannot be the search peer of another search head. For the recommended approach to accessing cluster member data, see Best practice: Forward search head data to the indexer layer.

**Network requirements**

**Network provisioning**

All members must reside on a high speed network where each member can access every other member.

The members do not necessarily need to be on the same subnet, or even in the same data center, if you have a fast connection between the data centers. You can adjust the various search head clustering timeout settings in server.conf. For help in configuring timeout settings, contact Splunk Professional Services.

**Ports that the cluster members use**

These ports must be available on each member:

- The management port (by default, 8089) must be available to all other members.
- The http port (by default, 8000) must be available to any browsers accessing data from the member.
- The KV store port (by default, 8191) must be available to all other members. You can use the CLI command `splunk show kvstore-port` to identify the port number.
- The replication port must be available to all other members.

These ports must be in your firewall's list of allowed ports.

**Caution:** Do not change the management port on any of the members while they are participating in the cluster. If you need to change the management port, you must first remove the member from the cluster.

**Synchronize system clocks across the distributed search environment**

It is important that you synchronize the system clocks on all machines, virtual or physical, that are running Splunk Enterprise instances participating in distributed search. Specifically, this means your cluster members and search peers. Otherwise, various issues can arise, such as search failures, premature
expiration of search artifacts, or problems with alerts.

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.

**Deployer requirements**

You need a Splunk Enterprise instance that functions as the **deployer**. The deployer updates member configurations. See Use the deployer to distribute apps and configuration updates.

Deployer functionality is only for use with search head clustering, but it is built into all Splunk Enterprise instances running version 6.2 or above. The processing requirements for a deployer are fairly light, so you can usually co-locate deployer functionality on an instance performing some other function. You have several options as to the instance on which you run the deployer:

- If you have a **deployment server** that is servicing only a small number of deployment clients (no more than 50), you can run the deployer on the same instance as the deployment server. The deployer and deployment server functionalities can interfere with each other at larger client counts. See Deployment server provisioning in Updating Splunk Enterprise Instances.

- If you are running an **indexer cluster**, you might be able to run the deployer on the same instance as the indexer cluster’s **master node**. Whether this option is available to you depends on the master's load. See Additional roles for the master node in Managing Indexers and Clusters of Indexers for information on cluster master load limits.

- If you have a **monitoring console**, you can run the deployer on the same instance as the console. See Which instance should host the console? in Monitoring Splunk Enterprise.

- You can run the deployer on the same instance as a **license master**. See Configure a license master in the Admin Manual.

- You can run the deployer on a dedicated Splunk Enterprise instance.

Do not locate deployer functionality on a search head cluster member. The deployer must run on a separate instance from any cluster member.
A deployer can service only a single search head cluster. If you have multiple clusters, you must use a separate deployer for each one. The deployers must run on separate instances.

For a general discussion of management component colocation, see Components that help to manage your deployment in the Distributed Deployment Manual.

Other considerations

**Deployment server and search head clusters**

Do not use deployment server to update cluster members.

The deployment server is not supported as a means to distribute configurations or apps to cluster members. To distribute configurations across the set of members, you must use the search head cluster deployer. See Use the deployer to distribute apps and configuration updates.

**Search head clustering and search head pooling**

You cannot enable search head clustering on an instance that is part of a search head pool. For information on migrating, see Migrate from a search head pool to a search head cluster.

Deploy a search head cluster

This topic covers the key steps needed to configure and start a search head cluster.

Parts of a search head cluster

A search head cluster consists of a group of search heads that share configurations, job scheduling, and search artifacts. The search heads are known as the cluster members.

One cluster member has the role of captain, which means that it coordinates job and replication activities among all the members. It also serves as a search head like any other member, running search jobs, serving results, and so on. Over time, the role of captain can shift among the cluster members.
In addition to the set of search head members that constitute the actual cluster, a functioning cluster requires several other components:

- **The deployer.** This is a Splunk Enterprise instance that distributes apps and other configurations to the cluster members. It stands outside the cluster and cannot run on the same instance as a cluster member. It can, however, under some circumstances, reside on the same instance as other Splunk Enterprise components, such as a deployment server or an indexer cluster master node.

- **Search peers.** These are the indexers that cluster members run their searches across. The search peers can be either independent indexers or nodes in an indexer cluster.

- **Load balancer.** This is third-party software or hardware optionally residing between the users and the cluster members. With a load balancer in place, users can access the set of search heads through a single interface, without needing to specify a particular one.

This diagram of a small search head cluster, consisting of three members, illustrates the various components and their relationships:
This topic focuses on setting up the cluster members and the deployer. Other topics in this chapter describe how to configure search peers, connect with an indexer cluster, and add a load balancer.

**Deploy the cluster**

These are the key steps in deploying clusters:

1. Identify your requirements.
2. Set up the deployer.
3. Install the Splunk Enterprise instances.
4. Initialize cluster members.
5. Bring up the cluster captain.
6. Perform post-deployment set-up.

**1. Identify your requirements**

a. Determine the cluster size, that is, the number of search heads that you want to include in it. It usually makes sense to put all your search heads in a single cluster. Factors that influence cluster size include the anticipated search load and number of concurrent users, and your availability and failover needs. See "About search head clustering".

b. Decide what replication factor you want to implement. The replication factor is the number of copies of search artifacts 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 search artifacts will reside on more cluster members, so your cluster can tolerate more member failures without needing to use a proxy to access the artifacts. But it also means that you will need more storage to handle the additional copies. See "Choose the replication factor for the search head cluster."

c. Determine whether the search head cluster will be running against a group of standalone indexers or an indexer cluster. For information on indexer clusters, see "About indexer clusters and index replication" in the Managing Indexers and Clusters of Indexers manual.
d. Study the topic "System requirements and other deployment considerations for search head clusters" for information on other key issues.

2. Set up the deployer

It is recommended that you select the deployer now, as part of cluster set-up, because you need a deployer in place before you can distribute apps and updated configurations to the cluster members.

a. Choose a Splunk Enterprise instance for the deployer functionality.

This instance cannot be a member of the search head cluster, but, under some circumstances, it can be a Splunk Enterprise instance in use for other purposes. If necessary, install a new Splunk Enterprise instance to serve as the deployer. See "Deployer requirements".

If you have multiple clusters, you must use a separate deployer for each cluster, unless you are deploying identical configurations across all the clusters. See "Deploy to multiple clusters."

Deployer functionality is automatically enabled on all Splunk Enterprise instances. The main configuration step is to specify the deployer's security key, as described in the next step. Later in the deployment process, you point the cluster members at this deployer instance, so that they have access to it.

For information on how to use the deployer to distribute apps to cluster members, see "Use the deployer to distribute apps and configuration updates."

b. Configure the deployer's security key.

See "Set a security key for the search head cluster."

The deployer uses the security key to authenticate communication with the cluster members. The cluster members also use it to authenticate with each other. You must set the key to the same value on all cluster members and the deployer. You set the key on the cluster members when you initialize them.

To set the key on the deployer, specify the pass4SymmKey attribute in the [shclustering] stanza of the deployer's server.conf file. For example:

```
[shclustering]
pass4SymmKey = yoursecuritykey
```
c. Set the search head cluster label on the deployer.

The search head cluster label is useful for identifying the cluster in the monitoring console. This parameter is optional, but if you configure it on one member, you must configure it with the same value on all members, as well as on the deployer.

To set the label, specify the `shcluster_label` attribute in the `[shclustering]` stanza of the deployer's `server.conf` file. For example:

```
[shclustering]
shcluster_label = shcluster1
```

See "Set cluster labels" in *Monitoring Splunk Enterprise*.

d. Restart the deployer to activate the configuration changes.

### 3. Install the Splunk Enterprise instances

Install the Splunk Enterprise instances that will serve as cluster members. For information on the minimum number of members necessary, see "Required number of instances."

**Caution:** Always use new instances. The process of adding an instance to a search head cluster overwrites any configurations or apps currently resident on the instance.

For information on how to install Splunk Enterprise, read the *Installation Manual*.

**Important:** You must change the admin password on each instance. The CLI commands that you use to configure the cluster will not operate on instances with the default password.

### 4. Initialize cluster members

For each instance that you want to include in the cluster, run the `splunk init shcluster-config` command and restart the instance:

```
splunk init shcluster-config -auth <username>:<password> -mgmt_uri <URI>:<management_port> -replication_port <replication_port> -replication_factor <n> -conf_deploy_fetch_url <URL>:<management_port> -secret <security_key> -shcluster_label <label>
splunk restart
```

Note the following:
• This command is only for cluster members. Do not run this command on the deployer.
• You can only execute this command on an instance that is up and running.
• The -auth parameter specifies your current login credentials for this instance. This parameter is required.
• The -mgmt_uri parameter specifies the URI and management port for this instance. You must use the fully qualified domain name. This parameter is required.
• The -replication_port parameter specifies the port that the instance uses to listen for search artifacts streamed from the other cluster members. You can specify any available, unused port as the replication port. Do not reuse the instance's management or receiving ports. This parameter is required.
• The -replication_factor parameter determines the number of copies of each search artifact that the cluster maintains. All cluster members must use the same replication factor. This parameter is optional. If not explicitly set, the replication factor defaults to 3.
• The -conf_deploy_fetch_url parameter specifies the URL and management port for the deployer instance. This parameter is optional during initialization, but you do need to set it before you can use the deployer functionality. See "Use the deployer to distribute apps and configuration updates."
• The -secret parameter specifies the security key that authenticates communication between the cluster members and between each member and the deployer. The key must be the same across all cluster members and the deployer. See "Set a security key for the search head cluster."

Important:

• The -shcluster_label parameter is useful for identifying the cluster in the monitoring console. This parameter is optional, but if you configure it on one member, you must configure it with the same value on all members, as well as on the deployer. See "Set cluster labels" in Monitoring Splunk Enterprise.

For example:

```
```
Caution: To add more members after you bootstrap the captain in step 5, you must follow the procedures in "Add a cluster member".

5. Bring up the cluster captain

a. Select one of the initialized instances to be the first cluster captain. It does not matter which instance you select for this role.

b. Run the `splunk bootstrap shcluster-captain` command on the selected instance:

```
splunk bootstrap shcluster-captain -servers_list
"<URI>:<management_port>,<URI>:<management_port>,..."  -auth
<username>:<password>
```

Note the following:

- This command designates the specified instance as the first cluster captain.
- Run this command on only a single instance.
- The `-servers_list` parameter contains a comma-separated list of the cluster members, including the member that you are running the command on. The members are identified by URI and management port. This parameter is required.
- **Important:** The URIs that you specify in `-servers_list` must be exactly the same as the ones that you specified earlier when you initialized each member, in the `-mgmt_uri` parameter. You cannot, for example, use `https://foo.example.com:8089` during initialization and `https://foo.subdomain.example.com:8089` here, even if they resolve to the same node.

Here is an example of the bootstrap command:

```
splunk bootstrap shcluster-captain -servers_list
-auth admin:changed
```

6. Perform post-deployment set-up

To complete set-up, perform these additional steps, as necessary:

a. Connect the search head cluster to search peers. This step is required. It varies according to whether the search peers reside in an indexer cluster:
• To connect the search head cluster to an indexer cluster, see "Integrate the search head cluster with an indexer cluster."

• To connect the search head cluster to non-clustered indexers, see "Connect the search heads in clusters to search peers".

b. **Add users.** This step is required. See "Add users to the search head cluster".

c. **Install a load balancer in front of the search heads.** This step is optional. See "Use a load balancer with search head clustering."

d. **Use the deployer to distribute apps and configuration updates to the search heads.** You must perform this step before you upgrade your set of configurations. See "Use the deployer to distribute apps and configuration updates."

### Check search head cluster status

To check the overall status of your search head cluster, run this command from any member:

```
splunk show shcluster-status -auth <username>:<password>
```

The command returns basic information on the captain and the cluster members. It indicates the status of each member, such as whether it is up or down.

You can also use the monitoring console to get more information about the status of the cluster. See Use the monitoring console to view search head cluster status and troubleshoot issues.

In addition to checking the status of the search head cluster itself, it is also advisable to check the status of the KV store running on the cluster. Run this command from any member:

```
splunk show kvstore-status -auth <username>:<password>
```

See KV store troubleshooting tools.

### Integrate the search head cluster with an indexer cluster
To integrate a search head cluster with an **indexer cluster**, configure each member of the search head cluster as a search head on the indexer cluster. Once you do that, the search heads get their list of **search peers** from the master node of the indexer cluster.

You can integrate search head clusters with either single-site or multisite indexer clusters.

In this diagram, a search head cluster performs searches across a single-site indexer cluster:
Integrate with a single-site indexer cluster

Configure each search head cluster member as a search head on the indexer cluster. Use the CLI `splunk edit cluster-config` command. For example:

```
splunk restart
```

You must run this CLI command on each member of the search head cluster.

This example specifies:

- The instance is a search head in an indexer cluster.
- The master node of the indexer cluster resides at `10.152.31.202:8089`.
- The secret key is "newsecret123".

The secret key that you set here is the indexer cluster secret key (which is stored in `pass4SymmKey` under the `[clustering]` stanza of `server.conf`), not the search head cluster secret key (which is stored in `pass4SymmKey` under the `[shclustering]` stanza of `server.conf`).

For a search head cluster to serve as the search tier of an indexer cluster, you must set both types of keys on each of the search head cluster members, because the members are serving both as nodes of the indexer cluster and as members of the search head cluster. Presumably, if you have already set up the search head cluster, you have set the search head cluster key before you get to this step.

Each key type must be identical on all nodes of its respective cluster. That is, the indexer cluster key must be identical on all nodes of the indexer cluster, while the search head cluster key must be identical on all search cluster members. It is recommended, however, that the indexer cluster key be different from the search head cluster key.

This is all you need for the basic configuration. The search heads now run their searches against the peer nodes in the indexer cluster.

Integrate with a multisite indexer cluster

In a multisite indexer cluster, each search head and indexer has an assigned site. Multisite indexer clustering promotes disaster recovery, because data is
allocated across multiple sites. For example, you might configure two sites, one in Boston and another in New York. If one site fails, the data remains accessible through the other site. See Multisite indexer clusters in *Managing Indexers and Clusters of Indexers*.

**Note:** Although a search head cluster can participate in a multisite indexer cluster, the search head cluster itself does not have site awareness. See Deploy a search head cluster in a multisite environment.

**Configure members**

To integrate search head cluster members with a multisite indexer cluster, configure each member as a search head on the indexer cluster, as in the single-site example. See Integrate with a single-site indexer.

The only difference from a single-site indexer cluster is that you must also specify the site for each member. This should ordinarily be "site0", so that all search heads in the cluster perform their searches across the same set of indexers. For example:

```
splunk restart
```

**Migrate members from a single-site indexer cluster to a multisite indexer cluster**

If the search head cluster members are already integrated into a single-site indexer cluster and you want to migrate that cluster to multisite, you must edit each search head's configuration to identify its site.

On each search head, specify its master node and its site. For example:

```
splunk edit cluster-master https://10.160.31.200:8089 -site site0
```

For complete details on migrating a single-site indexer cluster to multisite, see Migrate an indexer cluster from single-site to multisite in *Managing Indexers and Clusters of Indexers*.

**For more information**

For more information on configuration of search heads on indexer clusters, see the chapter Configure the search head in the *Managing Indexers and Clusters of Indexers*. 78
Indexers manual. That chapter also includes configuration for more complex scenarios, such as hybrid searching, where the search heads search across both indexer clusters and non-clustered indexers.

**Connect the search heads in clusters to search peers**

Before the search heads in the cluster can run searches, they need to know the identity of their indexers, or search peers. All members of a cluster must have access to the same set of search peers.

How the search heads find out about their search peers depends on whether the search head cluster is part of an indexer cluster. There are two scenarios to consider:

- The search head cluster will be running against an indexer cluster.
- The search head cluster will be running against individual, non-clustered indexers.

**Important:** Cluster members cannot distribute searches to other cluster members. In other words, a cluster member cannot be a search peer of the cluster.

**Search head cluster with indexer cluster**

If the search head cluster is connected to an indexer cluster, the master node on the indexer cluster provides the search heads with a list of peer nodes to search against.

Once you configure the search head cluster members so that they participate in the indexer cluster, you do not need to perform any further configuration for the search heads to know their search peers. See Integrate the search head cluster with an indexer cluster.

Even if you do not need the benefits of index replication, you can still take advantage of this simple approach to configuring the set of search peers. Just incorporate your set of indexers into an indexer cluster with a replication factor of 1. This topology also provides numerous other benefits from a management perspective. See Use indexer clusters to scale indexing in the Managing Indexers and Clusters of Indexers manual.
Search head cluster with non-clustered indexers

You can add non-clustered search peers in two ways:

- Add the search peers to each member individually.
- Add the search peers to one member and let the cluster replicate the peer configurations to all other cluster members. This is known as search peer replication.

Before Splunk Enterprise 6.4, only the first method was available. You had to add the search peers to each individual member. Starting with 6.4, you can add the search peers to just a single member and let the cluster replicate the peer configurations to the other members.

The replication method is usually preferable, for several reasons:

- It is simpler and faster.
- It ensures that all members have access to all peers.
- If you later add a new member to the cluster, it automatically gets the set of peers.

The main circumstance where you might prefer to add peers to individual members is if you already have a cluster and you have automated the process of adding search peers to each member.

You can switch to the replication method at any time. Peers already added individually will remain in the configuration. If you add a new member later, it will get the full set of peers, no matter how they were originally added to the cluster.

**Note:** The replication method does not use the configuration replication method described in Configuration updates that the cluster replicates. Instead, it uses a Raft state machine to replicate the changes to all active members. With this method, all active members receive the add request at the same time, ensuring that all members gain access to the same set of search peers.

**Replicate the search peers across the cluster**

1. Enable search peer replication on each member.

In each member’s server.conf file, configure the [raft_statemachinerstanza as follows:
[raft_statemachine]
disabled = false
replicate_search_peers = true

2. Restart each search head cluster member.

3. Use the CLI to add the search peers to one member. It does not matter which member you perform this on.

On one member, run the following command, one time for each search peer:

```
splunk add search-server <scheme>://<host>:<port> -auth <user>:<password> -remoteUsername <user> -remotePassword <passremote>
```

Note the following:

- `<scheme>` is the URI scheme for accessing the search peer: "http" or "https".
- `<host>` is the host name or IP address of the search peer's host machine.
- `<port>` is the management port of the search peer.
- `-auth` provides credentials for the member.
- `-remoteUsername` and `-remotePassword` provide credentials for the search peer. The remote credentials must be for an admin-level user on the search peer.

For example:

```
```

When you add a search peer to one cluster member, the cluster quickly replicates the operation to the other members. The members will then commit the change together.

**Important:** To add a peer through replication, you need a healthy cluster. Captaincy should remain with the same member until all active members have successfully committed the change. If you encounter a problem and the change does not get committed with the current captain, remediation is simple: Just rerun the `splunk add search-server` command.

4. Repeat the `splunk add search-server` command for each search peer.

**Note:** You can also use replication to remove search peers from the cluster members. See Remove a search peer via the CLI.
Add search peers to each member individually

To add the search peers individually to each search head, use the CLI. On each search head, invoke the `splunk add search-server` command for each search peer that you want to add:

```
splunk add search-server <scheme>://<host>:<port> -auth <user>:<password> -remoteUsername <user> -remotePassword <passremote>
```

You must repeat this procedure on each search head, for each search peer. For example, on a three member cluster, with five search peers, you must run this command a total of 15 times.

Caution: All search heads must use the same set of search peers.

Add search peers through Splunk Web

In addition to the CLI, you can add search peers through Splunk Web:

1. Unhide the hidden settings on the search head, as described in The Settings menu.

2. Follow the instructions in Use Splunk Web.

If you have enabled search peer replication, you add the search peers to only one of the cluster members. If you have not enabled search peer replication, you must add them to each cluster member.

Add search peers by directly editing distsearch.conf

If you are not using search peer replication, you can add search peers by directly editing `distsearch.conf` and distributing the configuration file via the deployer. This method requires that you also manually distribute the key file from each search head to each search peer. See Edit distsearch.conf.

Because of the need to manually distribute key files, this method is not compatible with search peer replication.

Forward search head data to the search peers

It is considered a best practice to forward all search head internal data to the search peer (indexer) layer. After you connect the search heads to the search peers, follow the instructions in Best practice: Forward search head data to the
indexer layer.

Add users to the search head cluster

In a search head cluster, all cluster members should maintain the same set of users, with the same set of roles.

To add users to the search head cluster, you can use any of the available authentication methods: Splunk Enterprise built-in authentication, LDAP, SAML, or scripted authentication. See the chapters on authentication in the Securing Splunk Enterprise manual for details.

The cluster automatically synchronizes user configurations across the set of members, in most cases. It uses configuration replication to do this. See "Configuration updates that the cluster replicates."

Use Splunk Enterprise built-in authentication

For Splunk Enterprise built-in authentication, you can use Splunk Web or the CLI to add users and map roles. Perform the operation on any one of the cluster members. The cluster then automatically distributes the changes to all members by replicating the $SPLUNK_HOME/etc/passwd file.

Authentication restrictions

Search head clustering does have a few restrictions regarding how you configure authentication:

- The cluster replicates the configuration changes automatically only if you configure authentication through Splunk Web, the Splunk CLI, or REST endpoints. If, instead, you edit a configuration file directly, you must use the deployer to distribute the file to the cluster members.

- Even when you configure authentication through Splunk Web, the CLI, or REST endpoints, the cluster only replicates the underlying configuration files, plus the $SPLUNK_HOME/etc/passwd file in the case of built-in authentication. If the authentication method that you are employing requires any other associated, non-configuration files, you must use the deployer to distribute them to the cluster members. For example:

  ◊ For SAML, you must use the deployer to push the certificates.
For scripted authentication, you must use the deployer to push the script. You must also use the deployer to push `authentication.conf`, because you can only configure scripted authentication by editing `authentication.conf` directly.

**How to use the deployer to push authentication files**

To push arbitrary groups of files, such as SAML certificates, from the deployer, you create an app directory specifically to contain those files.

For details on how to use the deployer to push files, see "Use the deployer to distribute apps and configuration updates."

**Use a load balancer with search head clustering**

Splunk recommends that you run a third-party hardware or software load balancer in front of your set of clustered search heads. That way, users can access the set of search heads through a single interface, without needing to specify a particular one.

There are a variety of third-party load balancers available that you can use for this purpose. Select a load balancer that employs layer-7 (application-level) processing.

Configure the load balancer so that user sessions are "sticky" or "persistent." This ensures that the user remains on a single search head throughout their session.

**Deploy a search head cluster in a multisite environment**

You can deploy search head cluster members across multiple physical sites. You can also integrate cluster members into a multisite indexer cluster. However, search head clusters do not have site awareness.

**Deploy a search head cluster across multiple physical sites**

There are no restrictions on where your cluster members can reside. In cases of high network latency between sites, however, you might notice some slowness in
UI responsiveness.

The amount of data that cluster members transfer to each other across the network is difficult to quantify, being dependent on a variety of factors, such as the number of users, the amount of user activity, the number and types of searches being run, and so on.

**Integrate a search head cluster with a multisite indexer cluster**

You can integrate the search head cluster members into a multisite indexer cluster. A multisite indexer cluster confers important advantages on your deployment. Most importantly, it enhances the high availability and disaster recoverability of your deployment. See "Multisite indexer clusters" in the *Managing Indexers and Clusters* manual.

To integrate a search head cluster with a multisite indexer cluster, configure each member as a search head in the multisite cluster. See "Integrate with a multisite indexer cluster."

It is recommended that you set each search head’s `site` attribute to "site0", to disable search affinity. When search affinity is disabled, the search head runs its searches across indexers spanning all sites. Barring any change in the set of available indexers, the search head will run its searches across the same set of primary bucket copies each time.

By setting all search heads to "site0", you ensure a seamless experience for end users, because the same set of primary bucket copies is used by all search heads. If, instead, you set different search heads to different sites, the end user might notice lag time in getting some results, depending on which search head happens to run a particular search.

If you have an overriding need for search affinity, you can assign the search heads to specific sites.

**Search head clusters do not have site awareness**

Unlike an indexer cluster, search head clusters lack site awareness:

- You cannot configure artifact replication on a site-by-site basis.
- The cluster does not guarantee that copies of each search artifact exist on each site.
Site awareness is less critical for a search head cluster than an indexer cluster. If a search head cluster member is missing a replicated copy of a search artifact, the cluster proxies it from another member, which could reside on the same site or on another site. See "How the cluster handles search artifacts." Even in the case of a site failure that results in the loss of all copies of some search artifacts, this is a manageable situation that you can recover from by rerunning searches and so on.

**Note:** There are ways that you can work around the lack of site awareness, if necessary. For example, if your search head cluster consists of four search heads divided evenly between two sites, you can set the replication factor to 3 and thus ensure that each site has at least one copy of each search artifact.

**Important considerations when deploying a search head cluster across multiple sites**

The choices you make when deploying a search head cluster across multiple sites can have significant implications for these failure scenarios:

- Site failure
- Network interruptions

In particular, in the case of a two-site cluster, you should put the majority of your members on the site that you consider primary.

**Why the majority of members should be on the primary site**

If you are deploying the cluster across two sites, put a majority of the cluster members on the site that you consider primary. This ensures that the cluster can continue to function as long as that site is running.

Under certain circumstances, such as when a member leaves or joins the cluster, the cluster holds an election in which it chooses a new captain. The success of this election process requires that a majority of all cluster members agree on the new captain. Therefore, the proper functioning of the cluster requires that a majority of members be running at all times. See "Captain election."

In the case of a cluster running across two sites, if one site fails, the remaining site can elect a new captain only if it holds a majority of members. Similarly, if there is a network disruption between the sites, only the site with a majority can elect a new captain. By assigning the majority of members to your primary site, you maximize its availability.
What happens when the site with the majority fails

If the site with a majority of members fails, the remaining members on the minority site cannot elect a new captain. Captain election requires the vote of a majority of members, but only a minority of members are running. The cluster does not function. See "Consequences of a non-functioning cluster."

To remediate this situation, you can temporarily deploy a static captain on the minority site. Once the majority site returns, you should revert the minority site to the dynamic captain. See "Use static captain to recover from loss of majority."

What happens when there is a network interruption between sites

If the network between sites fails, the members on each site will attempt to elect a captain. However, only a site that holds a majority of the total members will succeed. That site can continue to function as the cluster indefinitely.

During this time, the members on the other sites can continue to function as independent search heads. However, they will only be able to service ad hoc searches. Scheduled reports and alerts will not run, because, in a cluster, the scheduling function is relegated to the captain.

When the other sites reconnect to the majority site, their members will rejoin the cluster. For details on what happens when a member rejoins the cluster, see "When the member rejoins the cluster."

Clusters with more than two sites

If there are more than two sites, the cluster can function only if a majority of members across the sites are still able to communicate and elect a captain. For example, if you have site1 with five members, site2 with eight members, and site3 with four members, the cluster can survive the loss of any one site, because you will still have a majority of members (at least nine) among the remaining two sites. However, if you have site1 with six members, site2 with two members, and site3 with three members, the cluster can only function as long as site1 remains alive, because you need at least six members to constitute a majority.

Deploy a single-member search head cluster

For limited purposes, you can deploy a single-member search head cluster. This approach allows you to start with a small distributed search deployment and easily scale to a larger cluster later.
Why a single-member cluster?

The main benefit of a search head cluster is to provide high availability search. To fulfill that benefit, the cluster must comprise at least three members. See Captain election process has deployment implications.

If you do not require high availability and you need only the capacity provided by a single search head, you can deploy a non-clustered search head.

However, even if your current needs can be met by a single search head, you might want to deploy a single-member search head cluster rather than a non-clustered search head. This approach ensure a simple path to future expansion. When you are ready to scale your deployment's search capacity, you can expand the single-member cluster into a cluster of three or more members, thereby gaining access to high availability and the other benefits of a full-fledged cluster.

As an alternative to starting your distributed search deployment with a single-member cluster, you can start with a non-clustered, standalone search head. Although a non-clustered search head is simpler to deploy initially than a single-member search head cluster, scaling your search capacity in the future becomes a more complicated process, because it involves migrating the settings from the non-clustered search head to a search head cluster. See Migrate settings from a standalone search head to a search head cluster.

Deploy a single-member cluster

To deploy a single-member search head cluster, follow the same deployment procedure as you would for a multi-member cluster. See Deploy a search head cluster.

Note the following:

- The single member is also the cluster captain.
- You must choose a separate Splunk Enterprise instance to function as the deployer.
- You must set the replication factor on the member to 1.

The role of the deployer

In a search head cluster, the deployer distributes all apps and certain other configurations to the cluster members. See Use the deployer to distribute apps.
and configuration updates. You must use the deployer to distribute apps and other configurations.

Do not install apps and other similar configurations directly on the member. If you do, the configurations will not get added properly to new members when you later expand the cluster.

**Scale the cluster to three or more members**

When scaling the cluster, expand directly from the single member to a three-member cluster. A two-member cluster is not an inherently stable topology, due to the captain election process. Therefore, a two-member cluster is acceptable only for a brief period, during the process of scaling from one to three members.

To add new members to the cluster, follow the procedure described in Add a new member.

After you add the new members, update the replication factor on each member to the desired value for the expanded cluster, usually 3. All members must use the same value for the replication factor. See Choose the replication factor for the search head cluster.

3 is the default value for the replication factor. Therefore, if you choose to use 3 as the value, you only need to update the setting on the original member, which you previously set to 1 at the time of the initial deployment.

**Migrate settings from a standalone search head to a search head cluster**

You can migrate settings from an existing standalone search head to all members in a search head cluster.

You cannot migrate the search head instance itself, only its settings. You can only add clean, new Splunk Enterprise instances to a search head cluster.

**Types of objects to migrate**

There are two types of objects to migrate:
• Custom app configurations. These originate under `etc/apps` on the standalone search head.

• Private user configurations. These originate under `etc/users` on the standalone search head.

In both cases, you copy the relevant directories from the search head to the search head cluster's deployer. You then use the deployer to propagate these directories to the cluster.

The deployer pushes the configurations to the cluster, using a different method for each type. Post-migration, the app configurations obey different rules from the user configurations.

For information on where deployed settings reside on the cluster members, see "Where deployed configurations live on the cluster members."

**Custom app configurations**

When it migrates an app's custom settings, the deployer places them in the appropriate directories on the cluster members based on the `deployer_push_mode` setting in `server.conf`. This includes any runtime changes that were made while the apps were running on the standalone search head.

When migrating apps from a single search head to a search head cluster, set the `deployer_push_mode` to `full` before you push app configurations from the deployer to the cluster. This mode lets you retain the exact local and default directory configurations as they appear on the original search head. See Choose a deployer push mode.

Cluster users can override existing attributes by editing entities in place. Runtime changes get put in the local directories on the cluster members. Local directories override default directories, so the changes override the default settings.

**Private user configurations**

The deployer copies user configurations to the captain only. The captain then replicates the settings to all the cluster members through its normal method for replicating configurations, as described in "Configuration updates that the cluster replicates."

Unlike custom app configurations, the user configurations reside in the normal user locations on the cluster members and can later be deleted, moved, and so
on. They behave just like any runtime settings created by cluster users through Splunk Web.

When you migrate user configurations to an existing search head cluster, the deployer respects attributes that already exist on the cluster. It does not overwrite any existing attributes within existing stanzas.

For example, say the cluster members have an existing file

$SPLUNK_HOME/etc/users/admin/search/local/savedsearches.conf containing this stanza:

```
[my search]
school = index=_internal | head 1
```

and on the deployer, there's the file

$SPLUNK_HOME/etc/shcluster/users/admin/search/local/savedsearches.conf

with these stanzas:

```
[my search]
school = index=_internal | head 10
enableSched = 1
```

```
[my other search]
school = FOOBAR
```

This will result in a final merged configuration on the members:

```
[my search]
school = index=_internal | head 1
enableSched = 1
```

```
[my other search]
school = FOOBAR
```

The [my search] stanza, which already existed on the members, keeps the existing setting for its school attribute, but adds the migrated setting for the enableSched attribute, because that attribute did not already exist in the stanza. The [my other search] stanza, which did not already exist on the members, gets added to the file, along with its school attribute.

**Note:** Splunk does not support migration of per-user search history files.

**Do not migrate default apps**

When you migrate apps to the search head cluster, do not migrate any default apps, that is, apps that ship with Splunk Enterprise, such as the search app. If
you push default apps to cluster members, you overwrite the version of those apps residing on the members, and you do not want to do this.

You can, however, migrate custom settings from a default app:

- You can migrate any private objects associated with default apps. Private objects are located under the etc/users directory, not under etc/apps.

- You can migrate custom settings in the app itself by moving them to a new app and exporting them globally. The migration procedure in this topic includes a step for this.

Migrate settings to a search head cluster

This procedure assumes that you have already deployed the search head cluster. See Deploy a search head cluster.

To migrate settings:

1. Copy the $SPLUNK_HOME/etc/apps and $SPLUNK_HOME/etc/users directories on the standalone search head to a temporary directory on the deployer where you can edit them.

2. If you want to migrate custom settings from a default app, you can move them to a new app and export them globally. For example, to migrate settings from the search app:

   1. Copy the .../search/local directory in the temporary directory to a new app directory, such as search_migration_app, in the temporary directory. Do not name this new app "search."

   2. Export the settings globally to make them available to all apps, including the search app. To do this, create a ...

      /search_migration_app/metadata/local.meta file and populate it with the following content:

      []
      export=system

      See the default.meta specification file for details.

3. In the temporary directory, delete these subdirectories:

   - Any default apps, such as the search app. Do not push default apps to the cluster members. If you do, they will overwrite the versions of those apps already on the members.

   - Any apps already existing in the deployer's distribution directory. Otherwise, the versions from the standalone search head will overwrite the versions already on the members.
4. Copy all the remaining subdirectories from the temporary location to the
distribution directory on the deployer, located at
\$SPLUNK_HOME/etc/shcluster. Leave any subdirectories already in the
distribution directory unchanged.
For details on the distribution directory file structure, see Where to place
the configuration bundle on the deployer.
5. If you need to add new cluster members, you must deploy clean
instances. You cannot reuse the existing search head. For information on
adding cluster members, see Add a cluster member.
6. Set the deployer_push_mode to full. See Set the deployer push mode.
7. Run the splunk apply shcluster-bundle command on the deployer to
push the configuration bundle, including the migrated settings, to the
cluster. See Push the configuration bundle.

In full mode, the deployer pushes etc/apps/default directly to the
cluster members. It pushes both etc/apps/local and etc/users to the
captain, which asynchronously replicates the settings to the other cluster
members.

If you point the cluster members at the same set of search peers
previously used by the standalone search head, the cluster will need to
rebuild any report acceleration summaries or data model summaries
resident on the search peers. It does this automatically. It does not,
however, automatically remove the old set of summaries.

Migrate from a search head pool to a search head
cluster

You can migrate the settings from a search head pool to a search head cluster.
You cannot migrate the search head instances themselves, however. You must
use new instances when enabling search head cluster members.

The migration procedure varies somewhat depending on whether you are
migrating to a new cluster or to a cluster that is already running.

Types of objects to migrate

There are two types of objects to migrate:
• Custom app configurations. These originate under `etc/apps` on the search head pool shared storage.

• Private user configurations. These originate under `etc/users` on the search head pool shared storage.

In both cases, you copy the relevant directories from the search head pool shared storage to the search head cluster's deployer. You then use the deployer to propagate these directories to the cluster.

The deployer pushes the configurations to the cluster, using a different method for each type. Post-migration, the app configurations obey different rules from the user configurations.

For information on where deployed settings reside on the cluster members, see "Where deployed configurations live on the cluster members."

**Custom app configurations**

When it migrates an app's custom settings, the deployer places them in default directories on the cluster members. This includes any runtime changes that were made while the apps were running on the search head pool.

Because users cannot change settings in default directories, this means that users cannot perform certain runtime operations on these migrated entities:

- Delete. Users cannot delete any migrated entities.
- Move. Users cannot move these settings from one app to another.
- Change sharing level. Users cannot change sharing levels. For example, a user cannot change sharing from app-level to private.

Cluster users can override existing attributes by editing entities in place. Runtime changes get put in the local directories on the cluster members. Local directories override default directories, so the changes override the default settings.

**Private user configurations**

The deployer copies user configurations to the captain only. The captain then replicates the settings to all the cluster members through its normal method for replicating configurations, as described in "Configuration updates that the cluster replicates."
Unlike custom app configurations, the user configurations reside in the normal user locations on the cluster members and can later be deleted, moved, and so on. They behave just like any runtime settings created by cluster users through Splunk Web.

When you migrate user configurations to an existing search head cluster, the deployer respects attributes that already exist on the cluster. It does not overwrite any existing attributes within existing stanzas.

For example, say the cluster members have an existing file

```
$SPLUNK_HOME/etc/users/admin/search/local/savedsearches.conf
```

containing this stanza:

```
[my search]
search = index=_internal | head 1
```

and on the deployer, there’s the file

```
$SPLUNK_HOME/etc/shcluster/users/admin/search/local/savedsearches.conf
```

with these stanzas:

```
[my search]
search = index=_internal | head 10
enableSched = 1
```

```
[my other search]
search = FOOBAR
```

This will result in a final merged configuration on the members:

```
[my search]
search = index=_internal | head 1
enableSched = 1
```

```
[my other search]
search = FOOBAR
```

The [my search] stanza, which already existed on the members, keeps the existing setting for its search attribute, but adds the migrated setting for the enableSched attribute, because that attribute did not already exist in the stanza. The [my other search] stanza, which did not already exist on the members, gets added to the file, along with its search attribute.

**Note:** Splunk does not support migration of per-user search history files.
Do not migrate default apps

When you migrate apps to the search head cluster, do not migrate any default apps, that is, apps that ship with Splunk Enterprise, such as the search app. If you push default apps to cluster members, you overwrite the version of those apps residing on the members, and you do not want to do this.

You can, however, migrate custom settings from a default app by moving them to a new app and exporting them globally.

Each of the migration procedures in this topic includes a step for migrating default app custom settings.

Migrate to a new search head cluster

To migrate settings from a search head pool to a new search head cluster:

1. Follow the procedure for deploying any new search head cluster. Specify the deployer location at the time that you initialize the cluster members. See “Deploy a search head cluster.”

Caution: You must deploy new instances. You cannot reuse existing search heads.

2. Copy the etc/apps and etc/users directories on the shared storage location in the search head pool to the distribution directory on the deployer instance. The distribution directory is located at $SPLUNK_HOME/etc/shcluster.

For details on the distribution directory file structure, see "Where to place the configuration bundle on the deployer."

3. If you want to migrate custom settings from a default app, you can move them to a new app and export them globally. For example, to migrate settings from the search app :

   a. Copy the .../search/local directory in the distribution directory to a new app directory, such as search_migration_app, in the distribution directory. Do not name this new app "search."

   b. Export the settings globally to make them available to all apps, including the search app. To do this, create a .../search_migration_app/metadata/local.meta file and populate it with the following content:
export = system

See the default.meta specification file for details.

4. If $SPLUNK_HOME/etc/shcluster/apps contains any default apps, such as the search app, you must delete them now. Do not push them to the cluster members. If you do, they will overwrite the versions of those apps already on the members.

5. Run the `splunk apply shcluster-bundle` command on the deployer to push the configuration bundle to the cluster. See "Push the configuration bundle."

The deployer pushes `etc/apps` directly to the cluster members. It pushes `etc/users` to the captain, which asynchronously replicates the settings to the other cluster members.

**Note:** If you point the cluster members at the same set of search peers previously used by the search head pool, the cluster will need to rebuild any report acceleration summaries or data model summaries resident on the search peers. It does this automatically. It does not, however, automatically remove the old set of summaries.

**Migrate to an existing search head cluster**

To migrate settings from a search head pool to an existing search head cluster:

1. Copy the `etc/apps` and `etc/users` directories on the shared storage location in the search head pool to a temporary directory where you can edit them.

2. If you want to migrate custom settings from a default app, you can move them to a new app and export them globally. For example, to migrate settings from the search app:

   a. Copy the `.../search/local` directory in the temporary directory to a new app directory, such as `search_migration_app`, in the temporary directory. **Do not name this new app "search."**

   b. Export the settings globally to make them available to all apps, including the search app. To do this, create a `.../search_migration_app/metadata/local.meta` file and populate it with the following content:
3. In the temporary directory, delete these subdirectories:

- Any default apps, such as the search app. Do not push default apps to the cluster members. If you do, they will overwrite the versions of those apps already on the members.

- Any apps already existing in the deployer’s distribution directory. Otherwise, the versions from the search head pool will overwrite the versions already on the members.

4. Copy the remaining subdirectories from the temporary location to the distribution directory on the deployer, located at $SPLUNK_HOME/etc/shcluster. Leave any subdirectories already in the distribution directory unchanged.

For details on the distribution directory file structure, see "Where to place the configuration bundle on the deployer."

5. Run the `splunk apply shcluster-bundle` command on the deployer to push the configuration bundle, including the migrated settings, to the cluster. See "Push the configuration bundle."

The deployer pushes `etc/apps` directly to the cluster members. It pushes `etc/users` to the captain, which asynchronously replicates the settings to the other cluster members.

**Note:** If you point the cluster members at the same set of search peers previously used by the search head pool, the cluster will need to rebuild any report acceleration summaries or data model summaries resident on the search peers. It does this automatically. It does not, however, automatically remove the old set of summaries.

**Search head clustering and mounted bundles**

For most types of deployments, including search head clustering, Splunk recommends that you use normal bundle replication, rather than mounted bundles with shared storage.
As a result of changes to bundle replication made in the 5.0 timeframe, such as the introduction of delta-based replication and improvements in streaming, the practical use case for mounted bundles is now extremely limited. In most cases, mounted bundles make little difference in the amount of network traffic or the speed at which bundle changes get distributed to the search peers. At the same time, they add significant management complexity, particularly when combined with shared storage. Because of delta-based replication, even if your configurations contain large files, normal bundle replication entails little ongoing replication cost, as long as those files rarely change.

**Upgrade a search head cluster**

This topic describes how to upgrade a search head cluster. The process is the same for maintenance and major release upgrades.

Starting with version 6.5, you can perform a member-by-member upgrade. This lets you perform a phased upgrade of cluster members that allows the cluster to continue operating during the upgrade. To use the member-by-member upgrade process, you must be upgrading from version 6.4 or later.

Starting with version 7.1, you can perform a rolling upgrade. Rolling upgrade lets you perform a phased upgrade of cluster members with minimal interruption of ongoing searches. To use rolling upgrade, you must be upgrading from version 7.1 or later. For more information, see [Use rolling upgrade](#).

**Perform an offline upgrade**

In a regular offline upgrade, all cluster members are down for the duration of the upgrade process.

You must perform an offline upgrade when upgrading from version 6.3 or earlier.

Before performing the offline upgrade, note the following requirements:

- All cluster members must run the same version of Splunk Enterprise (down to the maintenance level).
- You can run search head cluster members against 5.x or later non-clustered search peers, so it is not necessary to upgrade standalone indexers at the same time. See [Splunk Enterprise version compatibility](#).

**Steps**
1. Stop all cluster members.
2. Upgrade all members.
3. Stop the deployer.
4. Upgrade the deployer.
5. Start the deployer.
6. Start the members.
7. Wait one to two minutes for captain election to complete. The cluster will then begin functioning.

**Perform a member-by-member upgrade**

When upgrading from version 6.4 or later, you can perform a member-by-member upgrade.

For a search head cluster that integrates with an indexer cluster, perform a member-by-member upgrade as part of the tiered upgrade procedure. See Upgrade each tier separately in *Managing Indexers and Clusters of Indexers*.

Before performing the upgrade, note the following requirements:

- Mixed-version clusters are not supported during the ongoing functioning of the cluster. Therefore, you must move quickly through the member-by-member upgrade process, first upgrading one member and then immediately upgrading the next, and so on, until you finish upgrading all members.
- Do not attempt any clustering maintenance operations, such as rolling restart, during upgrade.
- At the end of the upgrade, all members must be running the same version of Splunk Enterprise (down to the maintenance level).
- You can run search head cluster members against 5.x or later non-clustered search peers, so it is not necessary to upgrade standalone indexers at the same time. See [Splunk Enterprise version compatibility](https://docs.splunk.com/Documentation/Enterprise/7.1.2/Management/Upgrading/UpgradingEnterprise).

During member-by-member upgrade, KV store replication cannot be guaranteed. For this reason, there must be no KV store activity during the upgrade. To ensure there is no KV store activity during upgrade, perform an offline upgrade instead.

To perform a member-by-member upgrade:

1. Upgrade one member and make it captain:
   1. Stop the member.
   2. Upgrade the member.
   3. Start the member and wait while it joins the cluster.
4. Transfer captaincy to the upgraded member. See Transfer captaincy.
2. For each additional member, one-by-one:
   1. Stop the member.
   2. Upgrade the member.
   3. Start the member.
3. Upgrade the deployer:
   1. Stop the deployer.
   2. Upgrade the deployer.
   3. Start the deployer.

Perform a rolling upgrade

For detailed instructions on how to perform a rolling upgrade with minimal search disruption, see Use rolling upgrade.

Deployer initiates restart after post-6.2.6 upgrade

The deployer handles user configurations differently in versions higher than 6.2.6, compared to versions 6.2.6 and below. Because of this change, the first time that you use the deployer to distribute updates after upgrading your cluster to a version higher than 6.2.6, the deployer must initiate a rolling restart of all cluster members.

This restart takes place the first time, post-upgrade, that you run the `splunk apply shcluster-bundle` command. The restart only occurs if you had used the deployer to push user configurations in 6.2.6 or below.

This change in user configuration deployment means that such configurations no longer reside in default directories on the cluster members. This enables certain runtime operations on the configurations. Specifically, you can now delete or move the configurations or change their sharing levels. For more information on how the deployer handles user configurations post-6.2.6, see User configurations.

Changed behavior in 6.5 for user-based and role-based search quotas

The default behavior for handling user-based and role-based concurrent search quotas has changed with version 6.5.
In versions 6.3 and 6.4, the default is to enforce the quotas across the set of cluster members. Starting with 6.5, the default is to enforce the quotas on a member-by-member basis.

You can change quota enforcement behavior, if necessary. See Job scheduling.

**Use rolling upgrade**

Splunk Enterprise version 7.1.0 and later supports rolling upgrade for search head clusters. A rolling upgrade performs a phased upgrade of cluster members with minimal interruption to your ongoing searches. You can use a rolling upgrade to minimize search disruption when upgrading cluster members 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 later versions of Splunk Enterprise.
- All search head cluster members, cluster master, and peer nodes must be running version 7.1.0 or later.
- Do not attempt any clustering maintenance operations, such as rolling restart, bundle pushes, or node additions, during a rolling 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 cluster member and put that member into manual detention. While in manual detention, the member cannot accept new search jobs, and all in-progress searches try to complete within a configurable timeout. When all searches are complete, you perform the software upgrade and bring the member back online. You repeat this process for each cluster member until the rolling upgrade is complete.

A rolling upgrade behaves in the following ways:

- Cluster members are upgraded one at a time.
While in manual detention, the following applies to a cluster member:
- The cluster member cannot receive new searches, execute ad hoc searches, or receive new search artifacts from other members.
- The cluster member continues to participate in most cluster operations, such as captain election and conf replication.
- New scheduled searches are executed on other members.
- The cluster member waits for in-progress searches to complete, up to a maximum time set by the user. The default of 180 seconds is enough time for the majority of searches to complete in most cases.
- Rolling upgrades apply to both historical and real-time searches.

Perform a rolling upgrade

To upgrade a search head cluster with minimal search interruption, perform the following steps:

1. Run preliminary health checks

On any cluster member, run the `splunk show shcluster-status` command using the `--verbose` option to confirm that the cluster is in a healthy state before you begin the upgrade:

```
splunk show shcluster-status --verbose
```

Here is an example of the output from the command:

```
Captain:

decommission_search_jobs_wait_secs : 180
dynamic_captain : 1
elected_captain : Tue Mar  6 23:35:52

2018

id :

FEC6F789-8C30-4174-BF28-674CE4E4FAE2

initialized_flag : 1
label : sh3
max_failures_to_keep_majority : 1
mgmt_uri :
https://sroback180306192122accme_sh3_1:8089

min_peers_joined_flag : 1
rolling_restart : restart
rolling_restart_flag : 0
rolling_upgrade_flag : 0
service_ready_flag : 1
stable_captain : 1

Cluster Master(s):

103
The output shows a stable, dynamically elected captain, enough members to support the replication factor, no out-of-sync nodes, and all members running a compatible Splunk Enterprise version (7.1.0 or later). This indicates that the cluster is in a healthy 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. Checks apply only to the criteria listed.

Or, use this endpoint to monitor cluster health:

/services/shcluster/status?advanced=1
For endpoint details, see shcluster/status in the REST API Reference Manual.

Based on the health check results, either fix any issues impacting cluster health or proceed with caution and continue the upgrade.

2. Initialize rolling upgrade

Run the following CLI command on any cluster member:

splunk upgrade-init shcluster-members
Or, send a POST request to the following endpoint:

/services/shcluster/captain/control/control/upgrade-init
For endpoint details, see shcluster/captain/control/control/upgrade-init in the REST API Reference Manual.

3. Put a member into manual detention mode

Select a search head cluster member other than the captain and put that member into manual detention mode:

splunk edit shcluster-config -manual_detention on
Or, send a POST request to the following endpoint:

/servicesNS/admin/search/shcluster/member/control/control/set_manual_detention -d manual_detention=on
For endpoint details, see shcluster/member/control/control/set_manual_detention in the REST API Reference Manual.

For more information on manual detention mode, see Put a search head into detention.
4. **Confirm the member is ready for upgrade**

Run the following command to confirm that all searches are complete:

```
splunk list shcluster-member-info | grep "active"
```

The following output indicates that all historical and real-time searches are complete:

```
active_historical_search_count:0
active_realtime_search_count:0
```

Or send a GET request to the following endpoint:

```
/services/shcluster/member/info
```

For endpoint details, see `shcluster/member/info` in the *REST API Reference Manual*.

5. **Upgrade the member**

Upgrade the search head following the standard Splunk Enterprise upgrade procedure. See How to upgrade Splunk Enterprise in the *Installation Manual*.

6. **Bring the member back online**

1. Run following command on the cluster member:

   ```
splunk start
   ```

   On restart, the first member upgraded is automatically elected as cluster captain. This captaincy transfer occurs only once during a rolling upgrade.

2. Turn off manual detention mode:

   ```
splunk edit shcluster-config -manual_detention off
   ```

   Or, send a POST request to the following endpoint:

   ```
/servicesNS/admin/search/shcluster/member/control/control/set_manual_detention
   -d manual_detention=off
   ```

   For endpoint details, see `shcluster/member/control/control/set_manual_detention` in the *REST API Reference Manual*.
7. **Check cluster health status**

After you bring the member back online, check that the cluster is in a healthy state.

Run the following command on the cluster member:

```
splunk show shcluster-status --verbose
```

Or, use this endpoint to monitor cluster health:

```
/services/shcluster/status?advanced=1
```

For endpoint details, see shcluster/status in the *REST API Reference Manual*.

For information on what determines a healthy search head cluster, see *Health check output details*.

8. **Repeat steps 3-7 for all members**

Repeat steps 3-7 until you have upgraded all cluster members.

9. **Upgrade the deployer**

It is important to make sure that you upgrade the deployer at the same time that you upgrade the cluster members. The deployer must run the same version as the cluster members, down to the minor level. For example, if members are running 7.1.1, the deployer must run 7.1.x.

To upgrade the deployer, do the following:

1. Stop the deployer.
2. Upgrade the deployer, following standard Splunk Enterprise upgrade procedure. See How to upgrade Splunk Enterprise in the *Installation Manual*.
3. Start the deployer.

For more information on the deployer, see *Deployer requirements*.

10. **Finalize the rolling upgrade**

Run the following CLI command on any search head cluster member.
splunk upgrade-finalize shcluster-members

Or, send a POST request to the following endpoint:

/services/shcluster/captain/control/control/upgrade-finalize

For endpoint details, see shcluster/captain/control/control/upgrade-finalize in the REST API Reference Manual.

**Example upgrade automation script**

Version 7.1.0 and later includes an example automation script
(\texttt{shc\_upgrade\_template.py}) that you can use as the basis for automating the
search head cluster rolling upgrade process. Modify this template script based on your deployment.

\texttt{shc\_upgrade\_template.py} is located in SPLUNK\_HOME/bin and includes detailed
usage and workflow information.

\texttt{shc\_upgrade\_template.py} is an example script only. Do not apply the script to a
production instance without editing it to suit your environment and testing it extensively.
Configure search head clustering

Configure the search head cluster

This topic describes how to configure the behavior of the search head cluster itself. It does not describe how to configure the search-time environment of the cluster members, such as the set of saved searches, dashboards, and apps that the members have access to. For information on configuring the search-time environment, see the chapter "Update search head cluster members".

The members store their cluster configurations in their local server.conf files, located under $SPLUNK_HOME/etc/system/local/. See the server.conf specification file for details on all available configuration attributes.

Key information

Remember these key points while reading this topic:

- The essential configuration occurs when you initialize each member during the deployment process.
- Search head clustering has a large number of configuration settings available. With a few exceptions, you should not change these settings from their initial or default values without guidance from Splunk Support.
- You must maintain identical settings across all members, except as noted.
- When you do change a setting across all members, you must restart all the members at approximately the same time.

Initialization-time configurations

You can set all essential configurations during the deployment process, when you initialize each member. These are the key configuration attributes that you can or must set for each cluster member during initialization:

- The member’s URI. See "Deploy a search head cluster".
- The member’s replication port. See "Deploy a search head cluster".
- The cluster’s replication factor. See "Choose the replication factor for the search head cluster".
- The cluster’s security key. See "Set a security key for the search head cluster".
- The deployer location. See "Point the cluster members to the deployer".
The cluster’s label. See "Deploy a search head cluster".

**Caution:** It is strongly recommended that you set all these attributes during initialization and do not later change them. See "Deploy a search head cluster".

### Post-initialization configuration changes

The main configuration changes that you can safely perform on your own, post-initialization, are the ad hoc search settings. There are two of these: one for specifying whether a particular member should run ad hoc searches only, and another for specifying whether the member currently functioning as the captain should run ad hoc searches only. The captain will not assign scheduled searches to ad hoc members. See "Configure a cluster member to run ad hoc searches only".

You can also temporarily switch to a **static captain**, as a work around for disaster recovery. See "Use static captain to recover from loss of majority."

**Caution:** Do not edit the **id** attribute in the [shclustering] stanza. The system sets it automatically. This attribute must conform to the requirements for a valid GUID.

### Set the search head cluster label

You usually set the cluster label with the `splunk init` command when you deploy the cluster. If you did not set it during deployment, you can later set it for the cluster by running this command on any one member:

```
    splunk edit shcluster-config -shcluster_label <label>
```

You do not need to restart the member after setting the label.

**Note:** If you set the label on a cluster member, you must also set it on the deployer. See "Configure the deployer."

The `-shcluster_label` parameter is useful for identifying the cluster in the monitoring console. See "Set cluster labels" in *Monitoring Splunk Enterprise*.

### Maintain the same configuration settings across all members

The `server.conf` attributes for search head clustering must have the same values across all members, with these exceptions:
If any configuration values other than these ones vary from member to member, then the behavior of the cluster will change depending on which member is currently serving as captain. You do not want that to occur.

**Configuration methods**

Most of the configuration occurs during initial cluster deployment, through the CLI `splunk init` command. To perform further configuration later, you have two choices:

- **Use the CLI** `splunk edit shcluster-config` command.
- **Edit the** `[shclustering]` **stanza in** `server.conf` **directly.**

It is generally simpler to use the CLI.

**Caution:** You must make the same configuration changes on all members and then restart them all at approximately the same time. Because of the importance of maintaining identical settings across all members, do not use the `splunk rolling-restart` command to restart, except when changing the `captain_is_adhoc_searchhead` attribute, as described in "Configure a cluster member to run ad hoc searches only". Instead, run the `splunk restart` command on each member.

**Configure search head clustering with the CLI**

You can use the CLI `splunk edit shcluster-config` command to make edits to the `[shclustering]` stanza in `server.conf`. Specify each attribute and its configured value as a key value pair.

For example, to edit the `adhoc_searchhead` attribute:

```
splunk edit shcluster-config -adhoc_searchhead true -auth <username>::<password>
```

The CLI confirms that the operation was successful and instructs you to restart `splunkd`.

Note the following:
• You can use this command to edit any attribute in the [shclustering] stanza except the disabled attribute, which turns search head clustering on and off.
• You can only use this command on a member that has already been initialized. For initial configuration, use splunk init shcluster-config.

Configure search head clustering by editing server.conf

You can also change attributes by directly editing server.conf. The search head clustering attributes are located in the [shclustering] stanza, with one exception: To modify the replication port, use the [replication_port] stanza.

Choose the replication factor for the search head cluster

The replication factor determines the number of copies of each search artifact, or search result, that the cluster maintains. Replication occurs only for artifacts from scheduled saved searches. The cluster does not replicate results from ad hoc searches or realtime searches.

Effect of the replication factor

The cluster can tolerate a failure of (replication factor - 1) members without losing any search artifacts. For example, to ensure that your system can handle the failure of two members without losing search artifacts, configure a replication factor of 3. This configuration directs the cluster to store three copies of each search artifact, with each copy on a different member. If two members go down, the artifact is still available on a third member.

The default value for the replication factor is 3. This number is sufficient for most purposes.

Even with a large cluster of, for example, 50 search heads, you do not need a commensurately large replication factor. As long as you do not lose the replication factor number of members, at least one copy of each search artifact still exists somewhere on the cluster and is accessible to all cluster members. Any search head in the cluster can access any search artifact by proxying from a search head storing a copy of that artifact. The proxying operation is fast and unlikely to impede access to search results from any search head.
**Note**: The replication factor determines only the number of copies of search artifacts that the cluster maintains. It does not affect the replication of runtime configuration changes, such as new saved searches. Those changes get replicated to all cluster members by a different process. If you have 50 search heads, each of those 50 gets a copy of such configuration changes. See [Configuration updates that the cluster replicates](#).

### Replication factor configuration

All search head cluster members must use the same replication factor. The `server.conf` attribute that determines the replication factor is `replication_factor` in the `[shclustering]` stanza.

You specify the replication factor during deployment of the cluster, as part of member initialization. See [Initialize cluster members](#).

You can change the replication factor post-deployment, if necessary, but it is recommended that you consult Splunk Support before doing so. If you change the replication factor on one member, you must change it on all members. For information on modifying configuration values, see [Configure the search head cluster](#).

### For more information

For information on how the cluster replicates search artifacts, see [How the cluster handles search artifacts](#). That subtopic describes several key points about artifact replication, among them:

- In some cases, the cluster might replicate more than the replication factor number of a search artifact.
- Artifact proxying, along with additional replication, occurs if a member without a copy of the artifact needs access to it.
- If a member goes down, the cluster replaces the artifact copies that were being stored on that member.

See [List search artifacts](#) to learn how to view the set of artifacts in the cluster and on individual members.

### Set a security key for the search head cluster

113
The security key authenticates communication between all cluster members, as well as between members and the deployer instance.

For an overview of search head clustering configuration, see "Configure the search head cluster".

**Security key must be identical across all nodes**

You must set the key to the same value on all search head cluster members and the deployer.

**Set the security key during deployment**

It is recommended that you set the security key during initial cluster deployment. See "Deploy a search head cluster".

**Set the security key post-deployment**

If you neglected to set the key during deployment, you can set it post-deployment by configuring the `pass4SymmKey` attribute in `server.conf` on each cluster member and the deployer. Put the attribute under the `[shclustering]` stanza. For example:

```
[shclustering]
pass4SymmKey = yoursecuritykey
```

You must restart each instance for the key to take effect. For more information on post-deployment configuration, see "Configuration methods."

**Keep a copy of the security key**

You should 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 member, you will need to use the clear text version to set the key.

**Multiple search head clusters and the security key**

If your deployment includes multiple search head clusters, it is a best practice to use a different key for each cluster. By doing so, you avoid any possibility of mismatching clusters and their deployers, which could result in the content for one cluster being wrongly downloaded to a different one.
Set the security key for a combined search head cluster and indexer cluster

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.
Update search head cluster members

How configuration changes propagate across the search head cluster

Read this first

Before reading this topic, see:

- "Administer Splunk Enterprise with configuration files" in the Admin Manual. The topics in that chapter provide important background information on configuration files.

The importance of configuration files in a search head cluster

Settings in configuration files control the functionality of a search head, including the set of knowledge objects. For example, there are configuration files for saved searches, event types, and workflow actions. Other configuration files provide the settings for non-search functionality, such as data inputs and indexing. See "List of configuration files" in the Admin Manual.

Besides the configuration files, other files are important to search-time functionality. For example, static lookup tables, dashboards, and data models use various files as part of their definition.

For a search head cluster to function properly, its members must all use the same set of search-related configurations. For example, all search heads in the cluster need access to the same set of saved searches. They must therefore use the same savedsearches.conf settings.

Members should also use the same set of user-related settings. See "Add users to the search head cluster."

Apps must also be identical across all search heads in a cluster. An app is essentially just a set of configurations.

How configuration changes propagate in a search head cluster

A search head cluster uses two means to ensure that configurations are identical across its members: automatic replication and the deployer.
Replicated changes

The cluster automatically replicates any runtime knowledge object changes on one cluster member to all other members. This includes, for example, changes or additions to saved searches, lookup tables, and dashboards. For example, when a user in Splunk Web defines a field extraction, the cluster replicates that field extraction to all other search heads in the cluster.

In addition, the cluster replicates a few other runtime changes as well, such as changes to users and roles.

See "Configuration updates that the cluster replicates."

Deployed changes

The cluster does not replicate all configuration changes, but rather only certain changes, primarily to knowledge objects, made at runtime through Splunk Web, the CLI, or the REST API. For other configuration changes and additions, you must explicitly push the changes to all cluster members. You do this through a special Splunk Enterprise instance called the deployer.

Examples of changes that require use of the deployer include any configuration files that you edit directly. For example, if you make a change in limits.conf, you must push the change through the deployer. Similarly, if you directly edit a knowledge object configuration file, like savedsearches.conf, you must use the deployer to distribute it to cluster members. In addition, you must use the deployer to push new or upgraded apps to the cluster members.

You also use the deployer to migrate app and user settings from an existing search head pool or standalone search head to the search head cluster.

See "Use the deployer to distribute apps and configuration updates."

Add non-clustered search peers to a search head cluster

Adding non-clustered search peers (that is, indexers that are not part of an indexer cluster) to the search head cluster is an example of the type of configuration change that the cluster does not replicate automatically. At the same time, however, it might not be convenient to add search peers by using the deployer to push an updated distsearch.conf, because the deployer will then initiate a rolling restart of all cluster members.
To avoid a restart of cluster members, you can use the CLI `splunk add search-server` command to add peers to each cluster member individually. For details, see "Connect the search heads in clusters to search peers."

**Caution:** Complete this operation across all cluster members quickly, so that all members maintain the same set of search peers.

**The Settings menu**

The Settings menu in Splunk Web organizes settings into several groups, including one called Knowledge, which contains the knowledge object settings. Search head clustering hides most non-Knowledge groups in each member’s Settings menu by default. For example, it hides settings for data inputs and the distributed environment. You can unhide the hidden groups, if necessary.

The reason for hiding non-Knowledge settings is that the cluster only replicates certain setting changes, mainly those in the Knowledge category. If you make a change on one member to a setting in a non-Knowledge category, the cluster, with a few exceptions, does not automatically replicate that change to the other members. This can lead to the members being out of sync with each other.

If you need to access a hidden setting on a member, you can unhide those settings:

1. Click **Settings** in the upper right corner of Splunk Web. A list of settings, mainly limited to the Knowledge group, appears.

2. Click the **Show All Settings** button at the end of the list. A dialog box reminds you that hidden settings will not be replicated.

3. To continue, click **Show** in the dialog box. The full list of settings, dependent on your role permissions, appears.

The settings are now unhidden for all users with permission to view them; typically, all admin users. To rehide the settings, you must restart the instance.

**Important:** If you make a change to a hidden setting, the changed configuration will exist only on the cluster member where you made the change. If you want other members to get that change as well, you must use the deployer to push the underlying configuration file for that setting.
**CLI commands and cluster members**

Most general and search-related CLI commands are available for use on cluster members. If you run the command on one member, the cluster replicates the resulting configuration changes to the other members.

However, do not run the `splunk clean` command, in any of its variants, on an active cluster member. For example, the `splunk clean all` command should only be run after a member is removed from the cluster, as that command deletes the `_raft` folder, `/etc/passwd`, and so on. Similarly, if you run `splunk clean userdata` on one member, the user data will be cleaned on that member only. The change will not replicate to the other members, causing user/role information to differ between members.

For more information on replicated changes, see "Configuration updates that the cluster replicates."

**Configuration updates that the cluster replicates**

The cluster automatically replicates certain runtime configuration changes that a user makes on one cluster member to all the other members.

**Note:** The cluster replicates configuration changes to all cluster members. The cluster's replication factor applies only to search artifact replication. See Choose the replication factor for the search head cluster.

**The changes that the cluster replicates**

These are the main types of configuration changes that the cluster replicates:

- Runtime changes or additions to *knowledge objects*, such as *saved searches, lookup tables, and dashboards*. For example, when a user in Splunk Web defines a *field extraction*, the cluster replicates that field extraction to all search heads in the cluster.
- Runtime changes to users and roles. See Add users to the search head cluster.

Replication operates under these constraints:

- The cluster only replicates changes made at runtime, through specific configuration methods.
- A whitelist determines the specific types of changes that the cluster replicates.

**Configuration methods that trigger replication**

The cluster replicates changes made through these methods:

- Splunk Web
- The Splunk CLI
- The REST API

The cluster does not replicate any configuration changes that you make manually, such as direct edits to configuration files.

For example, if a user creates a saved search in Splunk Web on a cluster member, the cluster replicates that saved search to all cluster members. However, if you, as the administrator, add a saved search by directly editing the `savedsearches.conf` file on one cluster member, the cluster does not replicate that saved search to the other cluster members. You must use the `deployer` to push that saved search to all cluster members.

**The replication white list**

The cluster uses a whitelist to determine what changes to replicate. This whitelist is configured through the set of `conf_replication_include` attributes in the default version of `server.conf`, located in `$SPLUNK_HOME/etc/system/default`.

You can add or remove items from that list by editing the members’ `server.conf` files under `$SPLUNK_HOME/etc/system/local`. If you change the whitelist, you must make the same changes on all cluster members.

For a comprehensive list of items in the whitelist, consult the default version of `server.conf`. This is the approximate set of whitelisted items:

```plaintext
alert_actions
authentication
authorize
datamodels
event_renderers
eventtypes
fields
html
literals
lookups
```
The cluster replicates changes to all files underlying the whitelist items. In addition to configuration files themselves, this includes dashboard and nav XML, lookup table files, data model JSON files, and so on. The cluster also replicates permissions stored in *.meta files.

These are examples of the types of files replicated for various whitelist items:

```
# escape-hatch HTML views
conf_replication_include.html = true
# lookup table files
conf_replication_include.lookups = true
# manager XML
conf_replication_include.manager = true
# datamodel JSON files
conf_replication_include.models = true
# nav XML
conf_replication_include.nav = true
# view XML
conf_replication_include.views = true
```

**Note:** The cluster does not replicate user search history. This is reflected in the default server.conf file, which includes the line,

```
conf_replication_include.history = false.
```

Changing that value to "true" has no effect and does not cause the cluster to replicate search history.
The changes that the cluster ignores

The cluster ignores configuration changes for any items that are not on the whitelist. These include most system configuration files, such as `indexes.conf`, `server.conf`, and so on. For a complete list of such files, see Global configuration files in the Admin Manual. Exceptions from that list include certain settings in `authorize.conf` and `authentication.conf`.

You cannot work around this situation by simply adding system configuration files to the whitelist. Most settings in system configuration files require a restart to take effect, and there’s no mechanism to initiate an automatic restart of cluster members following replication of such configurations.

In addition, the cluster only replicates changes that are made through Splunk Web, the Splunk CLI, or the REST API. If you directly edit a configuration file, the cluster does not replicate it. Instead, you must use the deployer to distribute the file to all cluster members.

The cluster also does not replicate newly installed or upgraded apps.

For information on how to distribute such configuration changes through the deployer, see Use the deployer to distribute apps and configuration updates.

How replication works

When a user makes a configuration change to a cluster member search head, the member saves the change to a file, or set of files, locally and also sends the change to the captain. Approximately every five seconds, each cluster member contacts the captain and pulls any changes that have arrived since the last time it pulled changes. Each cluster member then applies the changes locally.

For example, assume a user on one cluster member uses Splunk Web to create a new field extraction. Splunk Web saves the field extraction in local files on that member. The member then sends the file changes to the captain. When each cluster member next contacts the captain, it pulls the changes, along with any other recent changes, and applies them locally. Within a few seconds, all cluster members have the new field extraction.

**Note:** Files replicated and updated this way are semantically and functionally equivalent across the set of cluster members. The files might not be identical on all members, however. For example, depending on circumstances such as the order in which changes reach the captain, it is possible that an updated setting in `props.conf` could appear in different locations within the file on different
members.

For details on the specifics of your cluster's configuration replication process, view the Search Head Clustering: Configuration Replication dashboard in the monitoring console. See Use the monitoring console to view search head cluster status and troubleshoot issues.

**When replication happens**

The purpose of replication is to keep search-related configurations in sync across all cluster members. To ensure this happens, replication occurs at various times, depending on the state of the member:

- **Each active cluster member** contacts the captain every five seconds and pulls any changes that have arrived since the last time it pulled changes.

- **When a new member joins the cluster**, it contacts the captain and downloads a tarball containing the current set of replicated configurations, including all changes that have been made over the life of the cluster. It applies the tarball locally.

- **When a member rejoins the cluster**. First, follow the procedure outlined in Add a member that was previously removed from the cluster, cleaning the instance before you re-add it to the cluster. The member then contacts the captain and downloads the tarball, the same way that a new member does.

- **During cluster recovery**. See How a recovering member resyncs with the cluster.

**Replication of deployer configurations**

The deployer distributes non-runtime configurations to the cluster. For some configuration types, it distributes the configurations directly to the cluster members. For other configuration types, it distributes the configurations to the captain, which then replicates the configurations to the members through the same method that it uses to distribute runtime configurations.

The deployer distributes these types of configurations to the captain:

- User configurations
- App local configurations
When the captain receives such configurations from the deployer, it replicates them to the members.

See:

- What exactly does the deployer send to the cluster?
- Where deployed configurations live on the cluster members

**View replication status**

The monitoring console contains a wealth of information about the status of configuration replication. See Use the monitoring console to view search head cluster status and troubleshoot issues.

To see when the members last pulled a set of configuration changes from the captain, run the `splunk show shcluster-status` command from any member:

```
splunk show shcluster-status
```

The output from this command includes, for each member, the field `last_conf_replication`. It indicates the last time that the member successfully pulled an updated set of configurations from the captain.

For general information on the command, see Show cluster status.

**Replication synchronization issues**

Under normal circumstances, the cluster continually replicates changes across all cluster members. Each member sends any changes to the captain, and the captain quickly replicates those changes to the other members. This process ensures that the members share a common baseline of configurations.

Certain conditions can cause a member’s baseline to get out-of-sync with the captain’s baseline, and thus with the other members’s baseline. In particular, a member can be out-of-sync when recovering from a loss of connectivity with the cluster. To remediate this situation, the member must resync with the cluster.

**How a recovering member resyncs with the cluster**

When a member rejoins the cluster, it must resync its baseline with the captain’s baseline. Until the process is complete, the member is considered to be out-of-sync with the cluster.
To resync its baseline, the member contacts the captain to request the set of intervening replicated changes. What happens next depends on whether the member and the captain still share a common commit in their replication change histories:

- If the captain and the member share a common commit, the member automatically downloads the intervening changes from the captain and applies them to its pre-offline configuration. The member also pushes its intervening changes, if any, to the captain, which replicates them to the other members. In this way, the member resyncs its baseline with the captain's baseline.

- If the captain and the member do not share a common commit, they cannot properly sync without manual intervention. To update the member's configuration, you must instruct the member to download the entire configuration tarball from the captain, as described in Perform a manual resync. The tarball overwrites the member's existing set of configurations, causing it to lose any local changes that occurred during the time that it was disconnected from the cluster.

Why a recovering member might need to resync manually

If the captain and the member do not share a common commit in their set of configuration changes, they cannot sync without manual intervention.

The members, including the captain, periodically purge older configuration changes from their change history. See Set replication history purging behavior.

If the recovering member has been disconnected from the cluster for so long that the cluster has purged some intervening change history, the recovering member will not share a common commit with the captain and therefore cannot apply the full set of intervening changes. Instead, the member must undergo a manual resync.

At the end of the manual resync process, the member once again shares a common baseline with the other members. In the process, the member loses any local changes made during the time that it was disconnected from the cluster. For this reason, a manual resync is also known as a "destructive resync."

See Handle failure of a search head cluster member.

A similar situation can occur if the entire cluster stops functioning for a while, and the members operate during that time as independent search heads. See
Recovery from a non-functioning cluster.

Perform a manual resync

Upon rejoining the cluster, the member attempts to apply the set of intervening replicated changes from the captain. If the set exceeds the purge limits and the member and captain no longer share a common commit, a banner message appears on the member's UI, with text similar to the following:

Error pulling configurations from the search head cluster captain; consider performing a destructive configuration resync on this search head cluster member.
The message also appears in the member's `splunkd.log` file.

If this message appears, it means that the member is unable to update its configuration through the configuration change delta and must apply the entire configuration tarball. It does not do this automatically. Instead, it waits for your intervention.

You must then initiate the process of downloading and applying the tarball by running this CLI command on the member:

```
splunk resync shcluster-replicated-config
```

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

Caution: This command causes an overwrite of the member's entire set of search-related configurations, resulting in the loss of any local changes.

Set replication history purging behavior

The purging of the configuration change history is determined by these attributes in `server.conf`:

- `conf_replication_purge.eligibile_count`. Its default is 20,000 changes.
- `conf_replication_purge.eligibile_age`. Its default is one day.

When both limits have been exceeded on a member, the member begins to purge the change history, starting with the oldest changes.

For more information on purge limit attributes, see the `server.conf` specification file.
Captain election and out-of-sync members

During captain election, it is important to ensure that out-of-sync members do not become captain. By default, the cluster attempts to prevent this situation from occurring.

An out-of-sync member lacks an up-to-date baseline configuration. If it becomes captain, it cannot manage the baseline for the cluster.

See Prevent out-of-sync members from becoming captain.

Troubleshoot the baseline configuration

The monitoring console provides information on the state of the baseline configuration across all cluster members. See Troubleshoot baseline consistency.

Use the deployer to distribute apps and configuration updates

The deployer is a Splunk Enterprise instance that you use to distribute apps and certain other configuration updates to search head cluster members. The set of updates that the deployer distributes is called the configuration bundle.

The deployer distributes the configuration bundle in response to your command, according to the deployer push mode that you select. The deployer also distributes the bundle when a member joins or rejoins the cluster.

You must use the deployer, not the deployment server, to distribute apps to cluster members. Use of the deployer eliminates the possibility of conflict with the run-time updates that the cluster replicates automatically by means of the mechanism described in Configuration updates that the cluster replicates.

For details of your cluster's app deployment process, view the Search Head Clustering: App Deployment dashboard in the monitoring console. See Use the monitoring console to view search head cluster status.

Which configurations do the deployer manage?

The deployer has these main roles:
• It handles migration of app and user configurations into the search head cluster from non-cluster instances and search head pools.
• It deploys baseline app configurations to search head cluster members.
• It provides the means to distribute non-replicated, non-runtime configuration updates to all search head cluster members.

You do not use the deployer to distribute search-related runtime configuration changes from one cluster member to the other members. Instead, the cluster automatically replicates such changes to all cluster members. For example, if a user creates a saved search on one member, the cluster automatically replicates the search to all other members. See Configuration updates that the cluster replicates. To distribute all other updates, you need the deployer.

Configurations move in one direction only: from the deployer to the members. The members never upload configurations to the deployer. It is also unlikely that you will ever need to force such behavior by manually copying files from the cluster members to the deployer, because the members continually replicate all runtime configurations among themselves.

Types of updates that the deployer handles

These are the specific types of updates that require the deployer:

• New or upgraded apps.
• Configuration files that you edit directly.
• All non-search-related updates, even those that can be configured through the CLI or Splunk Web, such as updates to indexes.conf or inputs.conf.
• Settings that need to be migrated from a search head pool or a standalone search head. These can be app or user settings.

You use the deployer to deploy configuration updates only. You cannot use it for initial configuration of the search head cluster or for version upgrades to the Splunk Enterprise instances that the members run on.

Types of updates that the deployer does not handle

You do not use the deployer to distribute certain runtime changes from one cluster member to the other members. These changes are handled automatically by configuration replication. See How configuration changes propagate across the search head cluster.

Because the deployer manages only a subset of configurations, note the following:
• The deployer does not represent a "single source of truth" for all configurations in the cluster.
• You cannot use the deployer, by itself, to restore the latest state to cluster members.

**App upgrades and runtime changes**

Because of how configuration file precedence works, changes that users make to apps at runtime get maintained in the apps through subsequent upgrades.

Say, for example, that you deploy the 1.0 version of some app, and then a user modifies the app's dashboards. When you later deploy the 1.1 version of the app, the user modifications will persist in the 1.1 version of the app.

As explained in Configuration updates that the cluster replicates, the cluster automatically replicates most runtime changes to all members. Those runtime changes do not get subsequently uploaded to the deployer, but because of the way configuration layering works, those changes have precedence over the configurations in the unmodified apps distributed by the deployer. To understand this issue in detail, read the rest of this topic, as well as the topic Configuration file precedence in the Admin Manual.

**Custom apps and deleted files**

The mechanism for deploying an upgraded version of an app does not recognize any deleted files or directories except for those residing under the default and local subdirectories. Therefore, if your custom app contains an additional directory at the level of default and local, that directory and all its files will persist from upgrade to upgrade, even if some of the files, or the directory itself, are no longer present in an upgraded version of the app.

To delete such files or directories, you must delete them manually, directly on the cluster members.

Once you delete the files or directories from the cluster members, they will not reappear the next time you deploy an upgrade of the app, assuming that they are not present in the upgraded app.

**When does the deployer distribute configurations to the members?**

The deployer distributes configurations to the cluster members under these circumstances:
When you invoke the `splunk apply shcluster-bundle` command, the deployer pushes any new or changed configurations to the members. See Deploy a configuration bundle.

When a member joins or rejoins the cluster, it checks the deployer for configuration updates. A member also checks for updates whenever it restarts. If any updates are available, it pulls them from the deployer.

**Set up the deployer**

The actions in this subsection are integrated into the procedure for deploying the search head cluster, described in the topic Deploy a search head cluster. If you already set up the deployer during initial deployment of the search head cluster, you can skip this section.

**Choose an instance to be the deployer**

Each search head cluster needs one deployer. The deployer must run on a Splunk Enterprise instance outside the search head cluster.

Depending on the specific components of your Splunk Enterprise environment, the deployer might be able to run on an existing Splunk Enterprise instance with other responsibilities, such as a deployment server or the master node of an indexer cluster. Otherwise, you can run it on a dedicated instance. See Deployer requirements.

**Deploy to multiple clusters**

The deployer sends the same configuration bundle to all cluster members that it services. Therefore, if you have multiple search head clusters, you can use the same deployer for all the clusters only if the clusters employ exactly the same configurations, apps, and so on.

If you anticipate that your clusters might need different configurations over time, set up a separate deployer for each cluster.

**Set a secret key on the deployer**

You must configure the secret key on the deployer and all search head cluster members. The deployer uses this key to authenticate communication with the cluster members. To set the key, specify the `pass4SymmKey` attribute in either the `[general]` or the `[shclustering]` stanza of the deployer's `server.conf` file. For example:
The key must be the same for all cluster members and the deployer. You can set the key on the cluster members during initialization.

You must restart the deployer instance for the key to take effect.

If there is a mismatch between the value of pass4SymmKey on the cluster members and on the deployer (for example, you set it on the members but neglect to set it on the deployer), you will get an error message when the deployer attempts to push the configuration bundle. The message will resemble this:

```
Error while deploying apps to first member: ConfDeploymentException:
Error while fetching apps baseline on target=https://testitls1l:8089:
Non-200/201 status_code=401; {"messages":{{"type":"WARN","text":"call not properly authenticated"}}}
```

**Set the search head cluster label on the deployer**

The search head cluster label is useful for identifying the cluster in the monitoring console. This parameter is optional, but if you configure it on one member, you must configure it with the same value on all members, as well as on the deployer.

To set the label, specify the shcluster_label attribute in the [shclustering] stanza of the deployer’s server.conf file. For example:

```
[shclustering]
shcluster_label = shcluster1
```

See Set cluster labels in *Monitoring Splunk Enterprise*.

**Point the cluster members to the deployer**

Each cluster member needs to know the location of the deployer. Splunk recommends that you specify the deployer location during member initialization. See **Deploy a search head cluster**.

If you do not set the deployer location at initialization time, you must add the location to each member’s server.conf file before using the deployer:

```
[shclustering]
conf_deploy_fetch_url = <URL>:<management_port>
```
The `conf_deploy_fetch_url` attribute specifies the URL and management port for the deployer instance.

If you later add a new member to the cluster, you must set `conf_deploy_fetch_url` on the member before adding it to the cluster, so it can immediately contact the deployer for the current configuration bundle, if any.

**Choose a deployer push mode**

The deployer push mode determines how the deployer distributes the configuration bundle to search head cluster members. Before you push the configuration bundle, choose the push mode that best fits your specific apps and use cases. The default push mode is `merge_to_default`.

You can set the push mode on the global level or on the app level. See [Set the deployer push mode](#).

The push mode applies to app directories only, not user directories.

The following tables describe how the deployer handles app configurations for each push mode. For more information on the push modes, see the entry for `deployer_push_mode` in the `app.conf` spec file.

For details on the way the deployer bundles apps in each mode, see [What exactly does the deployer send to the cluster?](#).

**Mode: full**

Do not use `full` mode for built-in apps such as the Search app. Use `local_only` mode instead.

<table>
<thead>
<tr>
<th>Mode: full</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On the deployer</td>
<td>Bundles all of the app’s contents located in the app’s <code>/default</code>, <code>/local</code>, and other directories, and pushes the bundle to the cluster.</td>
</tr>
<tr>
<td>On the members</td>
<td>Copies the non-local and non-user configurations to each member’s app folder and overwrites the existing contents. Merges local and user configurations with the corresponding folders on the member, such that the existing configuration on the member takes precedence.</td>
</tr>
<tr>
<td>Use cases</td>
<td>Use this mode to push app configurations to both <code>/local</code> and <code>/default</code> app directories on the members. For example, if you have</td>
</tr>
</tbody>
</table>
### Mode: full

A saved search that exists only in `/local` on the members, pushing the `/local` and `/default` app configurations to their respective directories on the members maintains the saved search configuration, and lets you subsequently delete the saved search on the members using Splunk Web.

Use this mode to migrate apps from a single search head to a new search head cluster. This retains the exact `/local` and `/default` directory configurations as they appear on the original search head.

Use this mode if you have a configuration on the deployer in the app's `/local` directory, and you want to push it to the members and then delete it from the deployer.

If you have unencrypted secrets in the app's `/default` directory on your deployer, **full** mode causes the deployer to push those along with the rest of the directory to the members. Other push modes do not cause this behavior.

### Mode: local_only

If an app is new, push the `/default` folder and `/local` folders the first time you push the bundle, using the **full** or **merge_to_default** mode. On subsequent pushes, you can choose the **local_only** mode.

<table>
<thead>
<tr>
<th>Mode: local_only</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On the deployer</td>
<td>Bundles the app's <code>/local</code> configuration and its metadata and pushes it to the cluster.</td>
</tr>
<tr>
<td>On the members</td>
<td>Merges the app's <code>/local</code> configuration from the deployer with the app's <code>/local</code> configuration on the member, such that the member's existing configuration takes precedence.</td>
</tr>
<tr>
<td>Use cases</td>
<td>Use this mode to modify only those apps that already exist on the members.</td>
</tr>
<tr>
<td></td>
<td>Use this mode to modify the <code>/local</code> configuration for a built-in app, such as the Search app.</td>
</tr>
</tbody>
</table>

When you push a built-in app, the deployer automatically applies the **local_only** mode. You can override the **local_only** mode for built-in apps by explicitly setting a different push mode in the app's `local/app.conf` file.
**Mode: default_only**

<table>
<thead>
<tr>
<th>Mode: default_only</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On the deployer</td>
<td>Bundles and pushes the app's <code>/default</code> and other non-<code>/local</code> directories to the cluster. The <code>/local</code> directory is not included in the bundle.</td>
</tr>
<tr>
<td>On the members</td>
<td>Overwrites the app's <code>/default</code> and other non-<code>/local</code> directories on each member. The <code>/local</code> subdirectory is unaffected.</td>
</tr>
<tr>
<td>Use cases</td>
<td>Use this mode if you want to explicitly abandon changes made in an app's <code>/local</code> directory. For example, if an app on the deployer has pre-existing configurations in the <code>/local</code> directory, and you delete those configurations on the members, using <code>default_only</code> mode prevents those configurations from re-appearing on the next deployer push.</td>
</tr>
</tbody>
</table>

**Mode: merge_to_default**

`merge_to_default` is the default push mode for all apps, except built-in apps, such as the Search app, which default to the `local_only` mode.

<table>
<thead>
<tr>
<th>Mode: merge_to_default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On the deployer</td>
<td>Merges all settings from files in the app's <code>/local</code> directory into corresponding files in the app's <code>/default</code> directory, and pushes the merged default files to the app's <code>/default</code> directory on each member. During the merging process, settings from the <code>/local</code> directory take precedence over any corresponding settings in the <code>/default</code> directory. If a particular attribute is defined in both places, the merged file retains the definition from the <code>/local</code> directory.</td>
</tr>
<tr>
<td>On the members</td>
<td>Overwrites the existing configuration in the members' <code>/default</code> app directories. No files are deployed to the members' <code>/local</code> app directories. This ensures that deployed settings never overwrite local or replicated runtime settings on the members. Otherwise, for example, app upgrades would wipe out runtime changes.</td>
</tr>
</tbody>
</table>
In **merge_to_default** mode, the deployer also merges the app's `/metadata/local.meta` and `/metadata/default.meta` files into a single `/metadata/default.meta` file on the members.

<table>
<thead>
<tr>
<th>Mode: merge_to_default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use cases</strong></td>
<td>Use this mode if you have a configuration on the deployer in the app's <code>/local</code> directory, and you want to push it to the members and then delete it from the deployer.</td>
</tr>
</tbody>
</table>

**Caution:** Before you choose this mode, read about certain limitations in the management of app-level knowledge object settings deployed with this push mode. See Effect of merge_to_default push mode on management of app-level knowledge objects.

### Set the deployer push mode

Before you push the configuration bundle to cluster members, make sure the push mode is properly set on the deployer.

You can set the push mode globally for all apps and locally for specific apps. App-specific deployer push mode settings take precedence over the global deployer push mode policy, so that you can have one global policy but set exceptions for specific apps.

To set a global deployer push mode, and optionally set a local deployer push mode for one app only:

1. Set the global deployer push mode in the `[shclustering]` stanza in `$SPLUNK_HOME/etc/system/local/app.conf`. For example:

   ```
   [shclustering]
   deployer_push_mode = full
   ```

2. (optional) Set the deployer push mode for one app only in the `[shclustering]` stanza in `$SPLUNK_HOME/etc/shcluster/apps/<app>/local/app.conf` for that specific app. You might need to add the `[shclustering]` stanza to the `app.conf` file if it is not already present. For example:

   ```
   [shclustering]
   deployer_push_mode = local_only
   ```

3. Push the bundle to the search head cluster.
If the `deployer_push_mode` is not explicitly set in `app.conf` for a given app, then that app follows the global `deployer_push_mode` setting.

**What the configuration bundle contains**

The configuration bundle is the set of files that the deployer distributes to the cluster. It consists of two types of configurations:

- App configurations
- User configurations

You determine the contents of the configuration bundle by copying the apps or user configurations to a special location on the deployer.

You can push arbitrary configuration updates by creating a new app directory and putting those configurations in that directory.

The deployer pushes the configuration bundle to the cluster using different methods depending on the selected push mode and on the type of configuration files that it is pushing. See What exactly does the deployer send to the cluster?

**Where to place the configuration bundle on the deployer**

On the deployer, the configuration bundle resides under the `$SPLUNK_HOME/etc/shcluster` directory. The set of files under that directory constitutes the configuration bundle.

The directory has this structure:

```
$SPLUNK_HOME/etc/shcluster/
    apps/
       <app-name>/
       <app-name>/
       ...
    users/
```

Note the following general points:

- The configuration bundle must contain at least one subdirectory under either `/apps` or `/users`. The deployer will error out if you attempt to push a configuration bundle that contains no app or user subdirectories.
- The deployer only pushes the contents of subdirectories under `/shcluster`. It does not push any standalone files directly under `/shcluster`. For example, it will not push the file `/shcluster/file1`. To
deploy standalone files, create a new apps directory under /apps and put the files in the local subdirectory. For example, put file1 under $SPLUNK_HOME/etc/shcluster/apps/newapp/local.

- The /shcluster location is only for files that you want to distribute to cluster members. The deployer does not use the files in that directory for its own configuration needs.

Note the following points regarding apps:

- Put each app in its own subdirectory under /apps. You must untar the app.
- The configuration bundle must contain all previously pushed apps, as well as any new ones. If you delete an app from the bundle, the next time you push the bundle, the app will get deleted from the cluster members.
- To update an app on the cluster members, put the updated version in the configuration bundle. Simply overwrite the existing version of the app.
- To delete an app that you previously pushed, remove it from the configuration bundle. When you next push the bundle, each member will delete it from its own file system. Note: If you need to remove an app, inspect its app.conf file to make sure that state = enabled. If state = disabled, the deployer will not remove the app even if you remove it from the configuration bundle.
- When the deployer pushes the bundle, it pushes the full contents of all apps that have changed since the last push. Even if the only change to an app is a single file, it pushes the entire app. If an app has not changed, the deployer does not push it again.

Note the following points regarding user settings:

- To push user-specific files, put the files under the /users subdirectories where you want them to reside on the members.
- The deployer will push the content under /shcluster/users only if the content includes at least one configuration file. For example, if you place a private lookup table or view under some user subdirectory, the deployer will push it only if there is also at least one configuration file somewhere under /shcluster/users.
- You cannot subsequently delete user settings by deleting the files from the deployer and then pushing the bundle again. In this respect, user settings behave differently from app settings.

What exactly does the deployer send to the cluster?

The deployer pushes the configuration bundle to the cluster as a set of tarballs.
There are several types of tarballs:

- App tarballs. Each app has one or two associated tarballs, depending on the push mode, which contain the app's configurations:
  - A local tarball, which the deployer pushes to the captain, which then replicates the tarball's contents to the members.
  - A default tarball, which the deployer pushes directly to the members.
- A user tarball, which the deployer pushes to the captain, which then replicates the tarball's contents to the members. There is exactly one user tarball for the entire system. The user tarball contains the contents of the $SPLUNK_HOME/etc/shcluster/users directory on the deployer.

On the initial push to a new cluster, the deployer distributes all tarballs to the cluster. On subsequent pushes, it distributes only tarballs for new apps and for apps that have changed since the last push. If even a single file has changed in an app, the deployer redistributes the entire app. It does not redistribute unchanged apps. It also redistributes the user tarball if the set of user configurations changes.

**The tarball creation and distribution process**

The deployer stages the configuration bundle in a special location on its file system, $SPLUNK_HOME/var/run/splunk/deploy. The staging area contains the set of tarballs for all current apps, plus the one user tarball. These tarballs are created or updated only when you invoke the `splunk apply shcluster-bundle` command.

When you invoke the `splunk apply shcluster-bundle` command, a two-step process occurs:

1. The deployer creates tarballs for any apps that are new or have changed since the last time the command was invoked. Note the following:
   - The current push mode determines the contents of the newly created app tarballs, including whether the app will have a local or a default tarball, or both.
   - For new apps, the deployer adds the tarballs to the staging area.
   - For changed apps, the deployer overwrites each existing tarball with a new tarball of the same type. For example, if the deployer is operating in local_only push mode, it generates only a local tarball for a changed app. The deployer then overwrites the app's existing local tarball, if any. If the app also has an existing default tarball, which was created under a different push mode, the deployer
ignores it, leaving the tarball in place.
♦ The deployer also creates a new user tarball, if the user directory has changed.

2. The deployer pushes the new and changed tarballs to the cluster. It sends new or changed default tarballs to all existing cluster members. At the same time, it sends new or changed local or user tarballs to the captain, which replicates their contents to the members.

**Tarball distribution to new or rejoining members**

If a new member joins the cluster, the deployer sends the member the full set of default tarballs currently in its staging area, ensuring that the member has the same set of default configurations as existing members.

If a down member rejoins the cluster, the deployer sends the member those default tarballs that it created or changed while the member was in a down state.

As always, the captain also replicates its baseline configurations to both new and rejoined members, including any configurations originating from local or user tarballs previously pushed to the captain from the deployer. See Configuration updates that the cluster replicates.

**The effect of push mode on app tarball creation**

Depending on the push mode, the deployer creates either one or two tarballs per new or changed app. The push mode also affects the contents of the app tarballs.

Deployer push modes do not affect the user tarball in any way.

<table>
<thead>
<tr>
<th>Deployer push mode</th>
<th>Content included from app</th>
<th>Configuration bundle contents, per app</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>all app directories</td>
<td>Two tarballs:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Local tarball with contents of /local directory and /metadata/local.meta, if present</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Default tarball with contents of all directories except /local (/default, /lookup, and so on). Also contains /metadata/default.meta, if present.</td>
</tr>
<tr>
<td>Deployer push mode</td>
<td>Content included from app</td>
<td>Configuration bundle contents, per app</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>local_only</td>
<td>/local app directory</td>
<td>One tarball:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Local tarball with contents of /local and /metadata/local.meta, if present</td>
</tr>
<tr>
<td>default_only</td>
<td>all app directories except /local</td>
<td>One tarball:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Default tarball with contents of all directories except /local (/default, /lookup, and so on). Also contains /metadata/default.meta, if present.</td>
</tr>
<tr>
<td>merge_to_default</td>
<td>all app directories</td>
<td>One tarball:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Default tarball with contents of all directories (/default, /lookup, /local, and so on). The contents of the /local directory are merged into the /default directory. Also contains /metadata/default.meta, if present.</td>
</tr>
</tbody>
</table>

When constructing the bundle, the deployer treats an app’s /metadata/local.meta and /metadata/default.meta files the same way that it treats the /local and /default directories. For example, in the merge_to_default push mode, the deployer merges the app’s /metadata/local.meta and /metadata/default.meta files into a single /metadata/default.meta file and includes it in the default tarball.

**The effect of push mode change on tarball creation**

You can change the push mode globally (deployer-wide) or on a per-app basis.

When you change the push mode globally, there is no immediate affect on the tarballs already in staging. Tarballs get updated only when the apps themselves are updated, through the addition or change of at least one file, followed by an invocation of the `splunk apply shcluster-bundle` command. A global change to push mode occurs in the deployer’s configuration without touching the apps themselves. Therefore, a global push mode change does not itself cause any tarballs to get updated.
When you change the push mode for a single app, you do so by editing the app's `app.conf` file, thereby updating the app itself. The next time that you invoke the `splunk apply shcluster-bundle` command, the deployer will create one or two new tarballs (depending on the push mode) for that app in staging. It will then distribute those tarballs to the cluster.

A change in push mode to `local_only` for an app, whether configured globally or on a per-app basis, means that the deployer will, in the future, create only local tarballs for that app. However, any existing default tarball for that app, created during the prior push mode, persists in staging and will get pushed to new or rejoining members in the normal fashion. By this mechanism, new and rejoining members always have the same default baseline as existing members.

**Where deployed configurations live on the cluster members**

On cluster members, deployed apps and user configurations reside under `$SPLUNK_HOME/etc/apps` and `$SPLUNK_HOME/etc/users`, respectively.

**App configurations**

When it deploys apps, the deployer bundles each app's configurations into tarballs and pushes the tarballs to the cluster. The deployer push mode determines the contents of the tarballs and how they get pushed. See **What exactly does the deployer send to the cluster?**.

For each app that it deploys, the deployer creates one or two tarballs, depending on the push mode:

- A default tarball
- A local tarball

The deployer pushes default tarballs directly to the cluster members. The members untar each default tarball and apply its settings locally. The settings in the default tarball overwrite directories already on the members. For example, if the default tarball contains a `/default` directory for appA, that `/default` directory from the tarball overwrites any `/appA/default` directory currently on the members.

The deployer pushes local tarballs to the captain. The captain untars each local tarball and then replicates its settings to all the cluster members through its normal method for replicating configurations, as described in **Configuration updates that the cluster replicates**. Each member merges the `/local` settings that it receives from the captain with the app’s `/local` settings already existing on the
member. If conflicts occur, values already on the member take precedence. See the section User configurations for an example of how this merging mechanism works.

**User configurations**

The deployer distributes user configurations to the captain only. The captain then replicates the settings to all the cluster members through its normal method for replicating configurations, as described in Configuration updates that the cluster replicates.

The user configurations reside in the normal user locations on the cluster members. They are not subject to deployer push mode settings. They behave just like any runtime settings created by cluster users through Splunk Web.

The deployment of user configurations is of value mainly for migrating settings from a standalone search head or a search head pool to a search head cluster. See Migrate from a search head pool to a search head cluster.

When you migrate user configurations to an existing search head cluster, the deployer respects the values for attributes that already exist on the cluster. It does not overwrite values for any existing attributes within existing stanzas.

For example, say the cluster members have an existing file

```
$SPLUNK_HOME/etc/users/admin/search/local/savedsearches.conf
```

containing this stanza:

```
[my search]
search = index=_internal | head 1
```

and on the deployer, there's the file

```
$SPLUNK_HOME/etc/shcluster/users/admin/search/local/savedsearches.conf
```

with these stanzas:

```
[my search]
search = index=_internal | head 10
enableSched = 1

[my other search]
search = FOOBAR
```

This will result in a final merged configuration on the members:

```
[my search]
search = index=_internal | head 10
enableSched = 1
```
search = index=_internal | head 1
enableSched = 1

[my other search]
search = FOOBAR

The [my search] stanza, which already existed on the members, keeps the existing setting for its search attribute, but adds the migrated setting for the enableSched attribute, because that attribute did not already exist in the stanza. The [my other search] stanza, which did not already exist on the members, gets added to the file, along with its search attribute.

Deploy a configuration bundle

To deploy a configuration bundle, you push the bundle from the deployer to the cluster members using the `splunk apply shcluster-bundle` command.

**Push the configuration bundle**

Once you push the bundle, do not begin any operations that change the search head captain until the bundle push operation is complete.

To push the configuration bundle to the cluster members:

1. Put new or changed apps and any user configuration changes in subdirectories under `shcluster/` on the deployer.
2. Untar any app.
3. Ensure you have selected the correct deployer push mode.
4. Ensure that the search head cluster is in a healthy state and that the captain is online. To check the cluster status, run the following command:

   ```bash
   splunk show shcluster-status --verbose
   ```

   In the output, verify that the value for the captain's `service_ready_flag` is 1. For more details on this command and an example of its output, see [Initiate a searchable rolling restart from the command line](#).

5. Run the `splunk apply shcluster-bundle` command on the deployer:

   ```bash
   splunk apply shcluster-bundle -target <URI>:<management_port>
   -auth <username>:<password>
   ```

   Note the following:

   - The `-target` parameter specifies the URI and management port for any member of the cluster, for example, `https://10.0.1.14:8089`. You specify only one cluster member but the deployer pushes to all members. This parameter is required.
- The `-auth` parameter specifies credentials for the deployer instance.

In response to `splunk apply shcluster-bundle`, the deployer displays this message:

```
Warning: Depending on the configuration changes being pushed, this command
might initiate a rolling-restart of the cluster members. Please refer to the
documentation for the details. Do you wish to continue? [y/n]:
For information on which configuration changes trigger restart, see
$SPLUNK_HOME/etc/system/default/app.conf. It lists the configuration files
that do not trigger restart when changed. All other configuration changes
trigger restart.
```

6. To proceed, respond to the message with `y`.

   You can eliminate the message by appending the flag `--answer-yes` to the `splunk apply shcluster-bundle` command. For example:

   ```
splunk apply shcluster-bundle --answer-yes -target <URI>:<management_port> -auth <username>:<password>
   ```

   This is useful if you are including the command in a script or otherwise automating the process.

If you attempt to push a very large tarball (>200 MB), the operation might fail due to various timeouts. Delete some of the contents from the tarball's app, if possible, and try again.

**How the cluster applies the configuration bundle**

The deployer and the cluster members execute the command as follows:

1. The deployer stages the app configuration bundle in a special location on its file system, `$SPLUNK_HOME/var/run/splunk/deploy`. The bundle consists of apps that are new or have changed since the last push. For each such app, the deployer copies the app's configurations into directories dictated by the push mode. It then packages those directories in tarballs. See What exactly does the deployer send to the cluster?.

2. The deployer pushes each app tarball, one-by-one, to either the cluster members or the captain:

   - The deployer pushes default tarballs directly to the members, which apply the tarballs locally.
   - The deployer pushes local tarballs to the captain, which untars them and replicates their settings to the members.
3. The deployer separately creates and pushes the users tarball, if any user configurations have changed since the last push. It pushes the users tarball to the captain, which untars it and replicates its settings to the members.

4. At the end of the bundle push, a rolling restart occurs if necessary. During a rolling restart, approximately 10% of the members restart at a time, until all have restarted. See Restart the search head cluster. 

Note: During a rolling restart, all members, including the captain, restart. Restart of the captain triggers the election process, which can result in a new captain. After the final member restarts, the cluster requires approximately 60 seconds to stabilize. During this interval, error messages might appear. You can ignore these messages. They should desist after 60 seconds.

Control the restart process

You should usually let the cluster automatically trigger any rolling restart, as necessary. However, if you need to maintain control over the restart process, you can run a version of `splunk apply shcluster-bundle` that stops short of the restart. If you do so, you must later initiate the restart yourself. The configuration bundle changes will not take effect until the members restart.

To run `splunk apply shcluster-bundle` without triggering a restart, use this version of the command:

```bash
splunk apply shcluster-bundle -action stage && splunk apply shcluster-bundle -action send
```

The members will receive the bundle, but they will not restart. Splunk Web will display the message "Splunk must be restarted for changes to take effect."

To initiate a rolling restart later, invoke the `splunk rolling-restart` command from the captain:

```bash
splunk rolling-restart shcluster-members
```

Push an empty bundle

In most circumstances, it is a bad idea to push an empty bundle. By doing so, you cause the cluster members to delete all the apps previously distributed by the deployer. For that reason, if you attempt to push an empty bundle, the deployer assumes that you have made a mistake and it returns an error message, similar to this one:
Error while deploying apps to first member: Found zero deployable apps to send; /opt/splunk/etc/shcluster is likely empty; ensure that the command is being run on the deployer. If intentionally attempting to remove all apps from the search head cluster use the "force" option.

WARNING: using this option with an empty shcluster directory will delete all apps previously deployed to the search head cluster; use with extreme caution!

You can override this behavior with the -force true flag:

```
splunk apply shcluster-bundle --answer-yes -force true -target <URI>:<management_port> -auth <username>:<password>
```

Each member will then delete all previously deployed apps from its $SPLUNK_HOME/etc/apps directory.

If you need to remove an app, inspect its app.conf file to make sure that state = enabled. If state = disabled, the deployer will not remove the app even if you remove it from the configuration bundle.

**Check the status of the latest deployer bundle**

You can check the status of the bundle that the deployer most recently pushed. Use the following command with the URI for each search head cluster member that you want to check:

```
splunk list shcluster-bundle -member_uri <URI>:<management_port> -auth <username>:<password>
```

This command returns the deployer_push_mode and deployer_push_status, as well as other useful information about the bundle on that search head.

**Allow a user without admin privileges to push the configuration bundle**

By default, only admin users (that is, those assigned a role containing the admin_all_objects capability) can push the configuration bundle to the cluster members. Depending on how you manage your deployment, you might want to allow users without full admin privileges to push apps or other configurations to the cluster members. You can do so by overriding the controlling stanza in the default restmap.conf file.

The default restmap.conf file includes a stanza that controls the bundle push process:
You can specify a different capability in this stanza, either an existing capability or one that you define specifically for the purpose. If you assign that capability to a new role, users given that role can then push the configuration bundle. You can optionally specify both the existing `admin_all_objects` capability and the new capability, so that existing admin users retain the ability to push the bundle.

To create a new special-purpose capability and then assign that capability to the bundle push process:

1. On the deployer, create a new `authorize.conf` file under `$SPLUNK_HOME/etc/system/local`, or edit the file if it already exists at that location. Add the new capability to that file. For example:

   ```
   [capability::conf_bundle_push]
   ```

2. In the same `authorize.conf` file, create a role specific to that capability. For example:

   ```
   [role_deployer_push]
   conf_bundle_push=enabled
   ```

3. On the deployer, create a new `restmap.conf` file under `$SPLUNK_HOME/etc/system/local`, or edit the file if it already exists at that location. Change the value of the `capability.post` setting to include both the `conf_bundle_push` capability and the `admin_all_objects` capability. For example:

   ```
   [apps-deploy:apps-deploy]
   match=/apps/deploy
   capability.post=conf_bundle_push OR admin_all_objects
   authKeyStanza=shclustering
   ```

You can now assign the `role_deployer_push` role to any non-admin users that need to push the bundle.

You can also assign the `capability.post` setting to an existing capability, instead of creating a new one. In that case, create a role specific to the existing capability and assign the appropriate users to that role.

For more information on capabilities, see the chapter Users and role-based access control in *Securing Splunk Enterprise.*
Preserve lookup files across app upgrades

Any app that uses lookup tables typically ships with stubs for the table files. Once the app is in use on the search head the tables get populated by runtime processes, such as searches. When you later upgrade the app, you can choose to preserve the populated lookup tables or overwrite them with stubs.

Set the `deployer_lookups_push_mode` in `app.conf` to specify how the deployer handles lookup tables when upgrading apps. This setting determines the behavior of the `-preserve-lookups` option when you push the configuration bundle using the `splunk apply shcluster-bundle` command.

1. Choose the `deployer_lookups_push_mode` that best fits your use case for each app:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preserve_lookups</td>
<td>This option preserves lookup tables only if you run the <code>splunk apply shcluster-bundle</code> command on the deployer with the <code>-preserve-lookups</code> flag set to true. If you do not set the <code>-preserve-lookups</code> flag to true, the populated lookup tables are overwritten on upgrade.</td>
</tr>
<tr>
<td>always_preserve</td>
<td>This option always preserves lookup tables, regardless of how you set the <code>-preserve-lookups</code> flag.</td>
</tr>
<tr>
<td>always_overwrite</td>
<td>This option always overwrites lookup tables with stub files, regardless of how you set the <code>-preserve-lookups</code> flag.</td>
</tr>
</tbody>
</table>

2. Set your chosen `deployer_lookups_push_mode` globally in the `[shclustering]` stanza of the `system/local/app.conf` file on the deployer. For example:

   ```
   [shclustering]
   deployer_lookups_push_mode = preserve_lookups
   ```

3. (optional) If you want to use a different setting for a specific app, set the `deployer_lookups_push_mode` locally in the `[shclustering]` stanza of the `local/app.conf` file under the specific app.

   If the `deployer_lookups_push_mode` is not explicitly set in `app.conf` under the specific app, the app uses the global `deployer_lookups_push_mode` setting.

4. Push the bundle to the search head cluster. If you are using the `preserve_lookups` mode, set the `-preserve-lookups` flag to true to
preserve the populated lookup tables for your apps:

splunk apply shcluster-bundle -target <URI>:<management_port>
-preserve-lookups true -auth <username>:<password>

To ensure that a stub persists on members only if there is no existing table file of the same name already on the members, this feature can temporarily rename a table file with a .default extension. For example, lookup1.csv becomes lookup1.csv.default. Therefore, if you have been manually renaming table files with a .default extension, you might run into problems when using this feature. Contact Splunk Support before proceeding.

**Effect of merge_to_default push mode on management of app-level knowledge objects**

For apps pushed to members through the merge_to_default push mode, be aware of the following restrictions. You cannot delete the app’s baseline knowledge objects through Splunk Web, the CLI, or the REST API after you deploy an app to the members. You also cannot move, share, or unshare those knowledge objects.

This limitation applies only to the app’s baseline knowledge objects - those that were distributed from the deployer to the members. It does not apply to the app’s runtime knowledge objects, if any. For example, if you deploy an app and then subsequently use Splunk Web to create a new knowledge object in the app, you can manage that object with Splunk Web or any other of the usual methods.

The limitation on managing baseline knowledge objects applies to lookup tables, dashboards, reports, macros, field extractions, and so on. The only exception to this rule is for app-level lookup table files that do not have a permission stanza in default.meta. Such a lookup file can be deleted through a member’s Splunk Web.

To delete an app-level baseline knowledge object, redeploy an updated version of the app that does not include the knowledge object.

This condition does not apply to user-level knowledge objects pushed by the deployer. User-level objects can be managed by all the usual methods.

The limitation on managing baseline knowledge objects occurs only when using the merge_to_default push mode. With this push mode, the deployer moves all local app configurations to the default directories before it pushes the app to the members. Default configurations cannot be moved or otherwise managed. On the other hand, any runtime knowledge objects reside in the app's local directory
and therefore can be managed in the normal way. For more information on where deployed configurations reside, see App configurations.

You can work around this limitation by changing the deployer_push_mode to full or local_only. To determine which deployer push mode best fits your use case, see Choose a deployer push mode.

**A member might have out-of-sync configurations when it rejoins the cluster**

When a down member rejoins the cluster, it checks the deployer for app updates. The deployer then distributes any new or changed default app tarballs to the rejoining member, ensuring that all members have the same set of default configurations.

However, when a member rejoins the cluster, it must also resync its baseline replicated configurations with the captain's baseline. Since the deployer distributes local app tarballs to the captain, which then replicates them to the members, it is possible for the rejoining member to have out-of-sync local app configurations, particularly if the rejoining member was down for an extended period of time. For details on this situation and how to resync baseline configurations when necessary, see Replication synchronization issues.

**Consequence and remediation of deployer failure**

The deployer distributes the configuration bundle to the cluster members under these circumstances:

- When you invoke the `splunk apply shcluster-bundle` command, the deployer pushes the apps and users configurations.
- When a member joins or rejoins the cluster, it checks the deployer for apps updates. A member also checks for updates whenever it restarts. If any apps updates are available, it pulls them from the deployer.

This means that if the deployer is down:

- You cannot push new configurations to the members.
- A member that joins or rejoins the cluster, or restarts, cannot pull the latest set of apps tarballs.

The implications of the deployer being down depend, therefore, on the state of the cluster members. These are the main cases to consider:
• The deployer is down but the set of cluster members remains stable.
• The deployer is down and a member attempts to join or rejoin the cluster.

The deployer is down but the set of cluster members remains stable

If no member joins or rejoins the cluster while the deployer is down, there are no important consequences to the functioning of the cluster. All member configurations remain in sync and the cluster continues to operate normally. The only consequence is the obvious one, that you cannot push new configurations to the members during this time.

The deployer is down and a member attempts to join or rejoin the cluster

In the case of a member attempting to join or rejoin the cluster while the deployer is down, there is the possibility that the apps configuration on that member will be out-of-sync with the apps configuration on the other cluster members:

• A new member will not be able to pull the current set of apps tarballs.
• A member that left the cluster before the deployer failed and rejoined the cluster after the deployer failed will not be able to pull any updates made to the apps portion of the bundle during the time that the member was down and the deployer was still running.

In these circumstances, the joining/rejoining member will have a different set of apps configurations from the other cluster members. Depending on the nature of the bundle changes, this can cause the joining member to behave differently from the other members. It can even lead to failure of the entire cluster. Therefore, you must make sure that this circumstance does not develop.

How to remedy deployer failure

Remediation is two-fold:

1. Prevent any member from joining or rejoining the cluster during deployer failure, unless you can be certain that the set of configurations on the joining member is identical to that on the other members. For example, if the rejoining member went down subsequent to the deployer failure.
2. Bring up a new deployer:
   1. Configure a new deployer instance. See Configure the deployer.
   2. Restore the contents of $SPLUNK_HOME/etc/shcluster to the new instance from backup.
   3. If necessary, update the conf_deploy_fetch_url values on all search head cluster members.
4. Push the restored bundle contents to all members by running the `splunk apply shcluster-bundle` command.
Manage search head clustering

Add a cluster member

There are several categories of members that you might need to add to a cluster:

- **A new member.** In this case, you want to expand the cluster by adding a new member.
- **A member that was previously removed from the cluster.** In this case, you removed the member with the `splunk remove shcluster-member` command and now want to add it back.
- **A member that left the cluster without being removed from it.** This can happen if, for example, the instance shut down unexpectedly.

This topic treats each of these categories separately through a set of high-level procedures, each of which references one or more detailed steps.

Add a new member

Install a new Splunk Enterprise instance and add it to the cluster:

1. Install a new instance of Splunk Enterprise on its own machine or virtual machine. See Hardware and operating system requirements.
2. Initialize the instance. See Initialize the instance.
3. Add the instance to the cluster. See Add the instance.

Add a member that was previously removed from the cluster

These procedures are for Splunk Enterprise instances that were previously members of this cluster but were removed from it with the `splunk remove shcluster-member` command. See "Remove a cluster member."

Add a removed member

To add a removed member:

1. Clean the instance to remove any existing configurations that could interfere with the cluster. See "Clean the instance."
2. Add the instance to the cluster. "Add the instance."

**Add a member that was both removed and disabled**

To add a member that was both removed and disabled:

1. Clean the instance to remove any existing configurations that could interfere with the cluster. See "Clean the instance."

2. Initialize the instance. See "Initialize the instance."

3. Add the instance to the cluster. "Add the instance."

**Add a member that left the cluster without being removed from it**

A typical reason for a member falling into this category is a temporary failure of the cluster member.

For members that left the cluster without being explicitly removed from it:

1. Start the instance with the `splunk start` command.

2. Depending on how long the member has been down, you might need to run the `splunk resync shcluster-replicated-config` command to download the current set of configurations.

See "Handle failure of a cluster member" for information on the `splunk resync shcluster-replicated-config` command, along with a discussion of other issues related to dealing with a failed member.

**Detailed steps**

The high-level procedures for adding a cluster member use the detailed steps in this section. Depending on the particular situation that you are handling, you might need to use only a subset of these steps. See the high-level procedures, earlier in this topic, to determine which of these steps your situation requires.

**Clean the instance**

*Note:* This step is not necessary if you are adding a new instance that contains only the default set of configurations.
If you are adding an existing instance to the cluster, you must first stop the instance and run the `splunk clean all` command:

```
splunk stop
splunk clean all
splunk start
```

The `splunk clean all` command deletes configuration updates that could interfere with the goal of maintaining the necessary identical configurations and apps across all cluster members. It does not delete any existing settings under the `[shclustering]` stanza in `server.conf`.

**Caution:** This step deletes most previously configured settings on the instance.

For a discussion of configurations that must be shared by all members, see "How configuration changes propagate across the search head cluster."

For more information on the `splunk clean` command, access the online CLI help:

```
splunk help clean
```

**Initialize the instance**

If the member is new to the cluster, you must initialize it before adding it to the cluster:

```
splunk init shcluster-config -auth <username>:<password> -mgmt_uri <URI>:<management_port> -replication_port <replication_port> -replication_factor <n> -conf_deploy_fetch_url <URL>:<management_port> -secret <security_key> -shcluster_label <label>
splunk restart
```

**Note the following:**

- See "Deploy a search head cluster" for details on the `splunk init shcluster-config` command, including the meaning of the various parameters.
- The `conf_deploy_fetch_url` parameter specifies the URL and management port for the deployer instance. You must set it when adding a new member to an existing cluster, so that the member can immediately contact the deployer for the latest configuration bundle, if any. See "Use the deployer to distribute apps and configuration updates."
This step is for new members only. Do not run it on members rejoining the cluster.

**Add the instance**

The final step is to add the instance to the cluster. You can run the `splunk add shcluster-member` command either on the new member or from any current member of the cluster. The command requires different parameters depending on where you run it from.

**When running the splunk add command on the new member itself, use this version of the command:**

```
splunk add shcluster-member -current_member_uri <URI>:<management_port>
```

Note the following:

- `current_member_uri` is the management URI and port of any current member of the cluster that this node is joining. This parameter allows the new node to communicate with the cluster.

**When running the splunk add command from a current cluster member, use this version of the command:**

```
splunk add shcluster-member -new_member_uri <URI>:<management_port>
```

Note the following:

- `new_member_uri` is the management URI and port of the new member that you are adding to the cluster. This parameter must be identical to the `-mgmt_uri` value you specified when you initialized this member.

**Post-add activity**

After the member joins or rejoins the cluster, it applies all replicated and deployed configuration updates:

1. It contacts the deployer to get the configuration bundle.

2. It contacts the captain and downloads the replicated configuration tarball.

See "How configuration changes propagate across the search head cluster."
Remove a cluster member

To remove a member from a cluster, run the `splunk remove shcluster-member` command on any cluster member.

**Important:** You must use the procedure documented here to remove a member from the cluster. Do not just stop the member.

To disable a member so that you can then re-use the instance, you must also run the `splunk disable shcluster-config` command.

To rejoin the member to the cluster later, see Add a member that was previously removed from the cluster. The exact procedure depends on whether you merely removed the member from the cluster or both removed and disabled the member.

Remove the member

**Caution:** Do not stop the member before removing it from the cluster.

1. Remove the member.

To run the `splunk remove` command on the member that you are removing, use this version:

```
splunk remove shcluster-member
```

To run the `splunk remove` command from another member, use this version:

```
splunk remove shcluster-member -mgmt_uri <URI>:<management_port>
```

Note the following:

- `mgmt_uri` is the management URI of the member being removed from the cluster.

2. Stop the member.

After removing the member, wait about two minutes for configurations to be updated across the cluster, and then stop the instance:

```
splunk stop
```
By stopping the instance, you prevent error messages about the removed member from appearing on the captain.

By removing the instance from the search head cluster, you automatically remove it from the KV store. To confirm that this instance has been removed from the KV store, run `splunk show kvstore-status` on any remaining cluster member. The instance should not appear in the set of results. If it does appear, there might be problems with the health of your search head cluster.

**Remove and disable the member**

If you intend to keep the instance alive for use in some other capacity, you must disable it after you remove it:

**Caution:** Do not stop the member first.

1. Remove the member:

   `splunk remove shcluster-member`

2. Disable the member:

   `splunk disable shcluster-config`

3. Clean the KVStore:

   `splunk clean kvstore --cluster`

**Configure a cluster member to run ad hoc searches only**

A search head in a cluster typically services both ad hoc search requests from users and scheduled searches assigned by the captain. You can limit a cluster member to ad hoc search requests only. If you designate a member as an ad hoc search head, the captain will not assign it any scheduled searches.

You can designate an ad hoc search head in two ways:

- You can specify that a particular member run only ad hoc searches at all times.
• You can specify that a member run only ad hoc searches while it is the captain.

**Note:** Although you can specify that a member run only ad hoc searches, you cannot specify that it run only scheduled searches. Any cluster member can always run an ad hoc search. You can, of course, prevent user access to a search head through any number of means.

**Configure a member to run ad hoc searches only**

Depending on your specific deployment, you might want to reserve certain search heads for ad hoc use only. Ad hoc search heads will never run scheduled searches. To specify an ad hoc search head, set the `adhoc_searchhead` attribute in the member’s `server.conf` file:

```plaintext
[shclustering]
adhoc_searchhead = true
```

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

**Configure the captain to run ad hoc searches only**

You can designate the captain member as an ad hoc search head. This prevents members from running scheduled searches while they are serving as captain, so that the captain can dedicate its resources to controlling the activities of the cluster. When the captain role moves to another member, then the previous captain will resume running scheduled searches and the new captain will now run ad hoc searches only.

**Important:** Make this change on all cluster members, so that the behavior is the same no matter which member is functioning as captain.

To designate the captain as an ad hoc search head, set the `captain_is_adhoc_searchhead` attribute in `server.conf` on each member:

```plaintext
[shclustering]
captain_is_adhoc_searchhead = true
```

You must restart each member for the change to take effect. Unlike most configuration changes related to search head clustering, you can use the `splunk rolling-restart` command to restart all members. See Restart the search head cluster.
For an overview of search head clustering configuration, see Configure the search head cluster.

**Control captaincy**

You have considerable control over which members become captain, through these methods:

- You can designate members as either "preferred captains" or "not preferred captains." When the cluster assigns captaincy, it attempts to assign it to a member with a preferred captain designation.
- You can transfer captaincy from one member to another.

In addition, by default, the cluster attempts to prevent an out-of-sync member from becoming captain. An out-of-sync member is one whose set of replicated configurations is out of sync with that of the current or most recent captain.

See Search head cluster captain for details on the captain's role in a search head cluster.

**Use cases**

It can be useful to control captaincy to handle a number of situations. For example:

- You have one member that you want to always use as captain. Or conversely, you have one member that you never want to be captain.
- You do not want the captain to perform any user-initiated ad hoc jobs. You can achieve this by designating one specific member as captain and keeping your third-party load balancer ignorant of that member.
- You want to repair the state of the cluster. A quick way to do this is to switch to a new captain, because members join a new captain in a clean state.

The twin tools of preferred captaincy and captaincy transfer give you flexibility when you need to control captaincy. Although neither one can guarantee that you always maintain complete control over the location of your captain, they do limit the likelihood that the captain will reside on a member that is not optimal for your needs. And captaincy transfer offers the ability to transfer the captain to a new member as needed.
Specify captaincy preference

You can designate some members as preferred captains and others as non-preferred captains. When the cluster assigns captaincy through the election process, it attempts to assign it to a member with a preferred captain designation.

Designate captaincy preference

To specify a member's preference for captaincy, set the `preferred_captain` attribute in that member's `server.conf` file:

```
preferred_captain = true|false
```

This attribute defaults to true, which means that, by default, all members are preferred captains.

To limit the likelihood that the cluster will assign captaincy to a particular member, set that member's `preferred_captain` attribute to false:

```
preferred_captain = false
```

The cluster attempts to respect the captaincy preference.

Limitations of captaincy preference

The cluster tries to assign captaincy to a member with `preferred_captain=true`. However, it might not always be possible to assign captaincy to a preferred-captain member. For example, if none of the preferred-captain members are reachable over the network, then captaincy might be assigned to a member with `preferred_captain=false`.

During an election for a new captain, a non-preferred-captain member can briefly become the captain before captaincy transfers to a preferred-captain member. If no preferred-captain members are available, the non-preferred-captain member remains captain until a preferred-captain member becomes available.

Prevent out-of-sync members from becoming captain

By default, the cluster attempts to prevent an out-of-sync member from becoming captain.
**What is an out-of-sync member?**

An out-of-sync member is a member that cannot sync its own set of replicated configurations with the common baseline set of replicated configurations maintained by the current or most recent captain. You do not want an out-of-sync member to become captain.

The captain maintains the baseline set of configurations for all members. When a configuration change occurs on one member, the member sends the change to the captain, which then replicates the change to all the other members. Therefore, it is essential that the baseline set of configurations on the captain be up-to-date.

If a member's set of configurations differs from the captain's baseline set, the member is considered to be out-of-sync. This can occur, for example, if the member lost network connectivity with the cluster for an extended period of time. When the member returns to the cluster, it needs to resync with the baseline set of configurations. If a large number of configuration changes occurred while the member was not in contact with the cluster, the resync can require manual intervention.

While a member is out-of-sync, it must not become captain. If an out-of-sync member does become captain, its set of configurations becomes the baseline that gets replicated to all other members. This situation would result in the loss of configuration changes made on other members.

See [Replication synchronization issues](#).

**Set out-of-sync behavior**

The `prevent_out_of_sync_captain` attribute in `server.conf` determines whether the cluster considers out-of-sync status when evaluating a member's eligibility for captain.

By default, this attribute is set to true. That is, the cluster attempts to prevent the member from becoming captain if it is out-of-sync. It is extremely unlikely that you will need to change this default behavior.

This attribute must be set to the same value on all members.
How the cluster determines member eligibility for captain

When electing a captain, the cluster considers the out-of-sync state to be more important the preferred-captain state. That is, if all preferred-captain members are out-of-sync, the cluster attempts to elect as captain a non-preferred-captain member, rather than a preferred-captain member that is out-of-sync. Briefly, here is the order that the cluster uses to determine member eligibility for captain:

1. Preferred-captain members that are not out-of-sync
2. Non-preferred-captain members that are not out-of-sync
3. A preferred-captain member that is out-of-sync
4. A non-preferred-captain member that is out-of-sync

This order assumes that you maintain the default behavior of prevent_out_of_sync_captain=true.

Transfer captaincy

You can transfer captaincy from one member to another.

The use of captaincy transfer does not interfere with the normal captain election process, which always proceeds in response to the circumstances described in Captain election. If an election occurs and results in the captain moving to a member other than the one you want it to reside on, you can then invoke captaincy transfer to relocate the captain.

Change the captain

To transfer captaincy to a different member, run this command from any member:

```
splunk transfer shcluster-captain -mgmt_uri <URI>:<management_port> -auth <username>:<password>
```

Note the following:

- The `-mgmt_uri` parameter specifies the URI and management port for the member that you want to transfer captaincy to. You must use the fully qualified domain name.
- You can run this command from any member. You are not limited to running it from the current captain or the intended captain.
- You do not need to restart any member after running the command.
To confirm that the captaincy transfer was successful, run the `splunk show shcluster-status` command from any member:

```
splunk show shcluster-status -auth <username>:<password>
```

Among other information returned, this command identifies the current captain.

You can also transfer captaincy through the search head clustering dashboard in Settings. See Use the search head clustering dashboard.

**Some ways to employ captaincy transfer in scripts**

The `splunk transfer shcluster-captain` command can be useful for scripting certain cluster behavior. For example:

- To ensure that captaincy stays with a particular member, you can implement a cron job that monitors the captain on a periodic basis. If the check detects a change in captain, it can automatically run the `splunk transfer shcluster-captain` command to return captaincy to the preferred member.
- To implement rolling-restart-style functionality (for example, if deploying cluster updates through some third-party tool), you can transfer captaincy to another member prior to restarting the current captain.

**Captaincy transfer and rolling-restarts**

As of 6.3, the rolling-restart process automatically invokes captaincy transfer to prevent captaincy from changing during the restart process. Because of this action, the member that was captain prior to the restart ordinarily continues as captain after the restart. See Restart the search head cluster.

**Captaincy transfer and static captain**

Captaincy transfer is available only with a dynamic captain. For information on the use of a static captain for disaster recovery, see Use static captain to recover from loss of majority.

**Handle failure of a search head cluster member**

When a member fails, the cluster can usually absorb the failure and continue to function normally.
When a failed member restarts and rejoins the cluster, the cluster can frequently complete the process automatically. In some cases, however, your intervention is necessary.

When a member fails

If a search head cluster member fails for any reason and leaves the cluster unexpectedly, the cluster can usually continue to function without interruption:

- The cluster’s high availability features ensure that the cluster can continue to function as long as a majority (at least 51%) of the members are still running. For example, if you have a cluster configured with seven members, the cluster will function as long as four or more members remain up. If a majority of members fail, the cluster cannot successfully elect a new captain, which results in failure of the entire cluster. See Search head cluster captain.

- All search artifacts resident on the failed member remain available through other search heads, as long as the number of machines that fail is less than the replication factor. If the number of failed members equals or exceeds the replication factor, it is likely that some search artifacts will no longer be available to the remaining members.

- If the failed member was serving as captain, the remaining nodes elect another member as captain. Since members share configurations, the new captain is immediately fully functional.

- If you are employing a load balancer in front of the search heads, the load balancer should automatically reroute users on the failed member to an available search head.

When the member rejoins the cluster

A failed member automatically rejoins the cluster, if its instance successfully restarts. When this occurs, its configurations require immediate updating so that they match those of the other cluster members. The member needs updates for two sets of configurations:

- The replicated changes, which it gets from the captain. See Updating the replicated changes.

- The deployed changes, which it gets from the deployer. See Updating the deployed changes.
See How configuration changes propagate across the search head cluster for information on how configurations are shared among cluster members.

**Updating the replicated changes**

When the member rejoins the cluster, it contacts the captain to request the set of intervening replicated changes. In some cases, the recovering member can automatically resync with the captain. However, if the member has been disconnected from the cluster for a long time, the resync process might require manual intervention.

See Replication synchronization issues for details on the recovery synchronization process, including how to perform a manual resync.

**Updating the deployed changes**

When the member rejoins the cluster, it automatically contacts the deployer for the latest configuration bundle. The member then applies any changes or additions that have been made since it last downloaded the bundle.

See Use the deployer to distribute apps and configuration updates.

**Use static captain to recover from loss of majority**

A cluster normally uses a dynamic captain, which can change over time. The dynamic captain is chosen by periodic elections, in which a majority of all cluster members must agree on the captain. See "Captain election."

If a cluster loses the majority of its members, therefore, it cannot elect a captain and cannot continue to function. You can work around this situation by reconfiguring the cluster to use a static captain in place of the dynamic captain.

A static captain does not change over time. Unlike a dynamic captain, the cluster does not conduct an election to select the static captain. Instead, you designate a member as the static captain, and that member remains the captain until you designate another member as captain.

**Shortcomings of the static captain**

The static captain has one fundamental shortcoming: It becomes a single point of failure for the cluster. If the captain fails, the cluster fails. The cluster cannot, on
its own, replace a static captain. Rather, manual intervention is necessary.

Because of this shortcoming, Splunk recommends that you use the static captain capability only for disaster recovery. Specifically, you can employ the static captain to recover from a loss of majority, which renders the cluster incapable of electing a dynamic captain.

In addition, the static captain does not check whether enough members are running to meet the replication factor. This means that, under some conditions, you might not have a full complement of search artifact copies.

**Note:** You should only employ static captain when absolutely necessary. While the process of converting to static captain is usually simple and fast, the process of later reverting back to a dynamic captain is somewhat more involved.

**Use cases for static captain**

Here are some situations where it makes sense to switch to a static captain:

- A single-site cluster loses the majority of its members. You can revive the cluster by designating one of its members as a static captain.

- The cluster is deployed across two sites. The majority site fails. Without a majority, the members in the second, minority site cannot elect a captain. You can revive the cluster by designating one of the members on the minority site as a static captain.

In all cases, once the precipitating issue has been resolved, you should revert the cluster to use a dynamic captain.

**Caution:** Do not use the static captain to handle a network interruption that stops communication between two sites. During a network interruption, the site with a majority of members continues to function as usual, because it can elect a dynamic captain as necessary. However, the site with a minority of members cannot elect a captain and therefore will not function as a cluster. If you attempt to revive the minority site by configuring its members to use a static captain, you will then have two clusters, one with a dynamic captain and the other with a static captain. When the network heals, you will not be able to reconcile the configuration changes between the sites.
Switch to a static captain

To switch to a static captain, reconfigure each cluster member to use a static captain:

1. On the member that you want to designate as captain, run this CLI command:

   splunk edit shcluster-config -mode captain -captain_uri <URI>:<management_port> -election false

2. On each non-captain member, run this CLI command:

   splunk edit shcluster-config -mode member -captain_uri <URI>:<management_port> -election false

Note the following:

- The `-mode` parameter specifies whether the instance should function as a captain or solely as a member. The captain always functions as both captain and a member.
- The `-captain_uri` parameter specifies the URI and management port of the captain instance.
- The `-election` parameter indicates the type of captain that this cluster uses. By setting `-election` to “false”, you indicate that the cluster uses a static captain.

You do not need to restart the captain or any other members after running these commands. The captain immediately takes control of the cluster.

To confirm that the cluster is now operating with a static captain, run this CLI command from any member:

   splunk show shcluster-status -auth <username>:<password>

   The `dynamic_election` flag will be set to 0.

Revert to the dynamic captain

When the precipitating situation has resolved, you should revert the cluster to control by a single, dynamic captain. To switch to dynamic captain, you reconfigure all the members that you previously configured for static captain. How exactly you do this depends on the type of scenario you are recovering from.
This topic provides reversion procedures for the two main scenarios:

- Single-site cluster with loss of majority, where you converted the remaining members to use static captain. Once the cluster regains a majority, you should convert the members back to dynamic.

- Two-site cluster, where the majority site went down and you converted the members on the minority site to use static captain. Once the majority site returns, you should convert all members to dynamic.

**Return single-site cluster to dynamic captain**

In the scenario of a single-site cluster with loss of majority, you should revert to dynamic mode once the cluster regains its majority:

1. As members come back online, convert them one-by-one to point to the static captain:

   ```sh
   splunk edit shcluster-config -election false -mode member -captain_uri <URI>:<management_port>
   ```

   Note the following:

   - The `-captain_uri` parameter specifies the URI and management port of the static captain instance.

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

As you point each rejoining member to the static captain, it attempts to download the replication delta. If the purge limit has been exceeded, the system will prompt you to perform a manual resync, as explained in "How the update proceeds."

**Caution**: During the time that it takes for the remaining steps of this procedure to complete, your users should not make any configuration changes.

2. Once the cluster has regained its majority, convert all members back to dynamic captain use. Convert the current, static captain last. To accomplish this, run this command on each member:

   ```sh
   splunk edit shcluster-config -election true -mgmt_uri <URI>:<management_port>
   ```

   Note the following:
The `-election` parameter indicates the type of captain that this cluster uses. By setting `-election` to "true", you indicate that the cluster uses a dynamic captain.

The `-mgmt_uri` parameter specifies the URI and management port for this member instance. You must use the fully qualified domain name. This is the same value that you specified when you first deployed the member with the `splunk init` command.

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

3. Bootstrap one of the members. This member then becomes the first dynamic captain. It is recommended that you bootstrap the member that was previously serving as the static captain.

```
splunk bootstrap shcluster-captain -servers_list
"<URI>:<management_port>,<URI>:<management_port>,..." -auth
<username>:<password>
```

For information on these parameters, see "Bring up the cluster captain."

**Return two-site cluster to dynamic captain**

In the scenario of a two-site cluster with loss of the majority site, you should revert to dynamic mode once the majority site comes back online:

1. When the majority site comes back online, convert its members to use the static captain. Point each majority site member to the static captain:

```
splunk edit shcluster-config -election false -mode member -captain_uri
<URI>:<management_port>
```

Note the following:

- The `-captain_uri` parameter specifies the URI and management port of the static captain instance.

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

As you point each rejoining member to the static captain, it attempts to download the replication delta. If the purge limit has been exceeded, the system will prompt you to perform a manual resync, as explained in "How the update proceeds."

2. Wait for all the majority-site members to get the replicated configs from the static captain. This typically takes a few minutes.
**Caution:** During the time that it takes for the remaining steps of this procedure to complete, your users should not make any configuration changes.

3. Convert all members back to dynamic captain use. Convert the current, static captain last. To accomplish this, run this command on each member:

```
splunk edit shcluster-config -election true -mgmt_uri <URI>:<management_port>
```

Note the following:

- The `-election` parameter indicates the type of captain that this cluster uses. By setting `-election` to "true", you indicate that the cluster uses a dynamic captain.
- The `-mgmt_uri` parameter specifies the URI and management port for this member instance. You must use the fully qualified domain name. This is the same value that you specified when you first deployed the member with the `splunk init` command.

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

4. Bootstrap one of the members. This member then becomes the first dynamic captain. It is recommended that you bootstrap the member that was previously serving as the static captain.

```
splunk bootstrap shcluster-captain -servers_list "<URI>:<management_port>,<URI>:<management_port>,..." -auth <username>:<password>
```

For information on these parameters, see "Bring up the cluster captain."

---

**Put a search head cluster member into detention**

You can put a search head cluster member into manual detention to allow for activities such as search head cluster rolling upgrades, rolling restart, or maintenance operations. When a search head cluster member is in manual detention, it stops accepting all new searches from the search scheduler or from users. Existing ad-hoc and scheduled search jobs run to completion. New scheduled searches are distributed by the captain to search head cluster members that are up and not in detention. You can run new ad-hoc searches against other members of the search head cluster. The search head in detention continues to participate in most cluster operations, such as captain election and conf replication, with the exception of search artifact replication.
You can put a search head cluster member in detention via the CLI, REST endpoint, or via the server.conf file.

When you manually put a search head cluster member into the detention state, it remains in detention until you remove it from detention, and the detention state persists through a restart.

This capability is limited to members in a search head cluster. It is not available to stand-alone search heads.

**Use cases**

Manual detention is useful for cases where you need a search head to be a functional member of a cluster, but you need to perform maintenance of some kind on the search head:

- **Rolling upgrades.** You can put a search head cluster member in detention as a part of a rolling upgrade. A rolling upgrade is a phased upgrade of all cluster members, so that searches can run without disruption during the upgrade process.
- **Search head cluster maintenance.** You can put a search head cluster member in detention to perform maintenance. Once the search head cluster member is in detention and all in-progress searches are completed, the member can be removed from the search head cluster for maintenance operations like hardware replacement or OS upgrade.
- **Search head diagnostics.** You can use manual detention to prevent searches from being sent to a poorly performing search head while you run diagnostics.
- **Searchable rolling restarts.** Manual detention is used by default in searchable rolling restarts. No action is required.

For information on searchable rolling restarts, see Restart the search head cluster. For information on rolling upgrades, see Use rolling upgrade.

**How existing searches are handled**

If a search is running on search head cluster member when it is placed in detention, the following behavior occurs:

- **On a search head that is in manual detention but not a part of a searchable rolling restart.** These searches will run to completion.
- **On a search head that is a part of a searchable rolling restart.** By default, these searches run for 180 seconds. Or, you can set a timeout.
period using the `decommission_search_jobs_wait_secs` attribute in the `[shclustering]` stanza of the search head's `server.conf` file. This attribute determines the amount of time, in seconds, that a cluster member waits for existing searches to complete before restarting.

- **On a search head that is a part of a rolling upgrade.** During rolling upgrade of a search head cluster, you can put a single search head into manual detention and wait for the existing search jobs to run to completion before you shut down the search head.

You can run the following CLI command to confirm that all searches are complete:

```
splunk list shcluster-member-info | grep "active"
```

The following output indicates that all historical and realtime searches are complete:

```
active_historical_search_count:0
active_realtime_search_count:0
```

Or send a GET request against:

```
/services/shcluster/member/info
```

See the documentation for editing the `decommission_search_jobs_wait_secs` attribute in the `server.conf` files here: search head clustering configuration. See the documentation for searchable rolling restarts here: How searchable rolling restart works.

**Put a search head cluster member into detention via Splunk Web**

To put a search head cluster member into detention from Splunk Web, complete the following steps:

1. Log in to any search head cluster member.
2. Click **Setting > Search head clustering.**
   The Search head clustering dashboard opens.
3. On the **Actions** tab for the cluster member you want to put in detention, click **Manual Detention.**
4. Click the **Manual Detention** toggle switch. A success message displays, and the status for the cluster member changes from **Up** to **ManualDetention.**
Put a search head cluster member into detention via the CLI

To put a search head cluster member into detention, run the CLI command
`splunk edit shcluster-config` with the `-manual_detention` parameter.

You can set the `-manual_detention` parameter to one of the following values:

- **on.** The search head cluster member enters detention and does not accept any new searches. It also does not receive replicated search artifacts from other members of the cluster. The search head continues to perform other duties associated with search head clustering, such as voting for a captain.

- **off.** The search head cluster member accepts new searches, replicates search artifacts, and performs duties associated with search head clustering. This is the default setting.

For example:

```
splunk edit shcluster-config -manual_detention on
```

```
splunk edit shcluster-config -manual_detention off
```

The search head must be in the "up" state before you put it in detention. Verify the state of the search head before you attempt to put it in manual detention.

To put a search head cluster member in detention from any other node, run the following command by specifying the 'target_uri' as an additional parameter to the CLI. The 'target_uri' is the 'mgmt_uri' of the target node to be put in manual detention.

```
splunk edit shcluster-config -manual_detention <on/off> -target_uri <>
```

For example:  `splunk edit shcluster-config -manual_detention on -target_uri https://test.sv.splunk.com:8095f`

For information on monitoring the status of a clustered search head, see Distributed Search Dashboards

Put a search head cluster member into detention via the REST endpoint

You can use the REST endpoint
shcluster/member/control/control/set_manual_detention to put a search head cluster member into manual detention.

For details, see the REST API documentation for shcluster/member/control/control/set_manual_detention.

**Put a search head cluster member into detention via the server.conf file**

To put a search head into manual detention, you can modify the manual_detention attribute in the [shclustering] stanza of the search head's server.conf file. You set the value to on. For example:

```
[shclustering]
disabled = 0
mgmt_uri = https://tsen-centos62x64-5:8089
id = C09EC4A9-8426-46F3-8385-693998B1EA5E
manual_detention = on
```

In order for changes to take effect, you must restart the search head cluster member when you use the server.conf file to put it into detention.

See the documentation for cluster configuration in the server.conf files here: search head clustering configuration.

**Restart the search head cluster**

You can restart the entire cluster with the `splunk rolling-restart` command. The command performs a phased restart of all cluster members, so that the cluster as a whole can continue to perform its functions during the restart process.

The deployer also automatically initiates a rolling restart, when necessary, after distributing a configuration bundle to the members. For details on this process, see "Push the configuration bundle".

When changing configuration settings in the [shclustering] stanza of server.conf, you must restart all members at approximately the same time to maintain identical settings across all members. Do not use the `splunk rolling-restart` command to restart the members after such configuration changes, except when configuring the captain_is_adhoc_searchhead attribute. Instead, run the `splunk restart` command on each member.
For more information, see Configure the search head cluster.

How rolling restart works

When you initiate a rolling restart, the captain issues a restart message to approximately 10% (by default) of the members at a time. Once those members restart and contact the captain, the captain then issues a restart message to another 10% of the members, and so on, until all the members, including the captain, have restarted.

If there are fewer than 10 members in the cluster, the captain issues the restart to one member at a time.

The captain is the final member to restart. After the captain member restarts, it continues to function as the captain.

After all members have restarted, it requires approximately 60 seconds for the cluster to stabilize. During this interval, error messages might appear. You can safely ignore these messages. Error messages will stop within 60 seconds.

During a rolling restart, there is no guarantee that all knowledge objects will be available to all members.

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 any search head cluster member.
2. Click Setting > Search head clustering. The Search head clustering dashboard opens.
3. Click Begin Rolling Restart.
4. Click Restart. This initiates the rolling restart across all cluster members.

Initiate a rolling restart from the command line

Invoke the `splunk rolling-restart` command from any member:

```
splunk rolling-restart shcluster-members
```
Specify the percentage of members to restart at a time

By default, the captain issues the restart command to 10% of the members at a time. The restart percentage is configurable through the `percent_peers_to_restart` attribute in the `[shclustering]` stanza of `server.conf`. For convenience, you can configure this attribute with the CLI `splunk edit shcluster-config` command. For example, to change the restart behavior so that the captain restarts 20% of the peers at a time, use this command:

```
splunk edit shcluster-config -percent_peers_to_restart 20
```

Do not set the value to greater than 20%. This can cause issues during the captain election process.

After changing the `percent_peers_to_restart` attribute, you still need to run the `splunk rolling-restart` command to initiate the actual restart.

Restart fails if cluster cannot maintain a majority

A cluster with a dynamic captain requires that a majority of members be running at all times. See "Captain election." This requirement extends to the rolling restart process.

If restarting the next set of members (governed by the `percent_peers_to_restart` attribute) would cause the number of active members to fall below 51% (for example, because some other members have failed), the restart process halts, in order to maintain an active majority of members. The captain then makes repeated attempts to restart the process, in case another member has rejoined the cluster in the interim. These attempts continue until the `restart_timeout` period elapses (by default, 10 minutes). At that point, the captain makes no more attempts, and the remaining members do not go through the rolling-restart process.

The `restart_timeout` attribute is settable in `server.conf`.

Use searchable rolling restart

Splunk Enterprise 7.1 and later provides a searchable option for rolling restarts. The searchable option lets you perform a rolling restart of search head cluster members 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, health checks automatically run to confirm that the cluster is in a healthy state. If the health checks succeed, the captain selects a cluster member and puts that member into manual detention. While in detention, the member stops accepting new search jobs, and waits for in-progress searches to complete. New searches continue to run on remaining members in the search head cluster. For more information, see Put a search head in detention mode.

After a configurable wait time or completion of all in-progress searches (whichever happens first), the captain restarts the member, and the member rejoins the cluster. The process repeats until all cluster members have been restarted. Finally, the captain puts itself into detention mode, and transfers the captaincy to one of the restarted members. The old captain is then restarted, at which point it regains the captaincy, and the rolling restart is complete.

Things to note about the behavior of searchable rolling restarts:

- The captain restarts cluster members one at a time.
- Health checks automatically run to confirm that the cluster is in a healthy state before the rolling restart begins.
- While in manual detention, a member:
  - cannot receive new searches (new scheduled searches are executed on other members).
  - cannot execute ad hoc searches.
  - cannot receive new search artifacts from other members.
  - continues to participate in cluster operations.
- The member 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 setting of 180secs covers the majority of searches in most cases. You can adjust this setting based on the average search runtime.
- Searchable rolling restart applies to both historical and real-time searches.

Initiate a searchable rolling restart

You can initiate a searchable rolling restart from Splunk or from the command line.
Initiate a searchable rolling restart from Splunk Web

1. Log in to any cluster member.
2. Click **Settings > Search head clustering**.
   The Search head clustering dashboard opens.
3. Click **Begin Rolling Restart**.
4. In the Rolling Restart modal, select the **Searchable** option.

![Rolling Restart Modal]

5. (Optional) The searchable option automatically runs cluster health checks. To override health check failures and proceed with the searchable rolling restart, select the **Force** option.

![Rolling Restart Modal]

Use the Force option with caution. This option can impact searches.

6. Click **Restart**.
   This initiates the searchable rolling restart.

Initiate a searchable rolling restart from the command line

To perform a searchable rolling restart from the command line:

1. (Optional) Run health checks to determine if the search head cluster is in a healthy state to perform a searchable rolling restart.
2. Use the CLI command to initiate the searchable rolling restart (includes health checks). Optionally, use the force option to override health checks.

1. (Optional) Run preliminary health check

You can use the `splunk show shcluster-status` command with the `--verbose` option to view information about the health of the search head cluster. This can help you determine if the cluster is in an appropriately healthy state to initiate a searchable rolling restart.

It is not mandatory to run a health check before you initiate a searchable rolling restart. Searchable rolling restart automatically runs a health check when initiated.

To view information about the health of the cluster, run the following command on any cluster member:

```
splunk show shcluster-status --verbose
```

Here is an example of the output from the above command:

```
Captain:
    decommission_search_jobs_wait_secs : 180
dynamic_captain : 1
elected_captain : Tue Mar 6 23:35:52
    id :
FEC6F789-8C30-4174-BF28-674CE4E4FAE2
    initialized_flag : 1
    label : sh3
    max_failures_to_keep_majority : 1
    mgmt_uri :
https://sroback180306192122accme_sh3_1:8089
    min_peers_joined_flag : 1
    rolling_reboot : restart
    rolling_restart_flag : 0
    rolling_upgrade_flag : 0
    service_ready_flag : 1
    stable_captain : 1
Cluster Master(s):
    https://sroback180306192122accme_master1_1:8089
splunk_version: 7.1.0
Members:
    sh3
    label : sh3
```
The output shows a stable, dynamically elected captain, enough members to support the replication factor, no out-of-sync nodes, and all members running a compatible Splunk Enterprise version (7.1.0 or later). This indicates the cluster is in a healthy state to perform a searchable rolling restart.

**Health check output details**

The table shows output values for the criteria used to determine the health of the search head cluster.

<table>
<thead>
<tr>
<th>Health Check</th>
<th>Description</th>
</tr>
</thead>
</table>

181
<table>
<thead>
<tr>
<th><strong>Output Value</strong></th>
<th><strong>Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic_captain</td>
<td>1</td>
</tr>
<tr>
<td>stable_captain</td>
<td>1</td>
</tr>
<tr>
<td>service_ready_flag</td>
<td>1</td>
</tr>
<tr>
<td>out_of_sync</td>
<td>0</td>
</tr>
<tr>
<td>splunk_version</td>
<td>7.1.0 or later.</td>
</tr>
</tbody>
</table>

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 any cluster member, invoke the `splunk rolling-restart shcluster-members` command using the `searchable` option.

```
splunk rolling-restart shcluster-members -searchable true
```

The searchable option automatically runs cluster health checks. If you want to proceed with a searchable rolling restart, despite health check failures, you can override health checks and initiate the searchable rolling restart, using the `force` option. For example:

```
splunk rolling-restart shcluster-members -searchable true \ 
-force true \ 
-decommission_search_jobs_wait_secs <positive integer>
decommission_search_jobs_wait_secs specifies the amount of time, in seconds, that a search head cluster member waits for existing searches to complete before restart. If you do not specify a value for this option, the command uses the default value of 180 secs in `server.conf`. If you specify a value of zero, rolling restart runs in non-searchable mode.
```
Use CLI or REST to set rolling restart behavior in server.conf

You can use the CLI or REST API to set the `rolling_restart` attribute in the `shclustering` stanza of `local/server.conf`.

The `rolling_restart` attribute supports these modes:

- **restart**: Initiates a rolling restart in classic mode (no guarantee of search continuity).
- **searchable**: Initiates a rolling restart with minimum search interruption.
- **searchable_force**: Overrides health check failures and initiates a rolling restart with minimum search interruption.

When you set `rolling_restart` to `searchable` or `searchable_force` mode, you can optionally set a custom value for the `decommission_search_jobs_wait_secs` attribute. This attribute determines the amount of time, in seconds, that a member waits for existing searches to complete before restart. The default is 180 secs.

When using the CLI or REST API to set rolling restart attributes, a cluster restart is not required.

**Use the CLI to set rolling restart**

To set the `rolling_restart` mode, invoke the `splunk edit shcluster-config -rolling_restart` command on any cluster member. For example:

```
splunk edit shcluster-config -rolling_restart searchable
-decommission_search_jobs_wait_secs 300
```

**Use REST to set rolling restart**

To set the `rolling_restart` mode, send a POST request to the `shcluster/config/config` endpoint. For example:

```
curl -k -u admin:pass https://<host>:<mPort>/services/shcluster/config
-d rolling_restart=searchable
-d decommission_search_jobs_wait_secs=300
```

For endpoint details, see `shcluster/config/config` in the *REST API Reference Manual*.

For more information on search head clustering configuration, see `server.conf.spec`. 

183
Set searchable rolling restart as the default mode for deployer bundle push

Deployer bundle pushes that require a restart use the default `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 deployer bundle push.

To set searchable rolling restart as the default mode for deployer bundle push, use the following attributes in the `[shclustering]` stanza of `server.conf`:

```
rolling_restart = searchable | searchable_force
```

By default, `rolling_restart` is set to `restart`.

For more information on deployer bundle push, see Use the deployer to distribute apps and configuration updates.

Monitor the restart process

To check the progress of the rolling restart, run this variant of the `splunk` `rolling-restart` on any cluster member:

```
splunk rolling-restart shcluster-members -status 1
```

The command returns the status of any members that have started or completed the restart process. For example:

```
Peer | Status          | Start Time      | End Time       | GUID
1. server-centos65x64-4 | RESTARTING | Mon Apr 20 11:52:21 2015 | N/A | 7F10190D-F00A-47AF-8688-8DD26F1A8A4D
2. server-centos65x64-3 | RESTART-COMPLETE | Mon Apr 20 11:51:54 2015 | Mon Apr 20 11:52:16 2015 | E78F5ECE-1EC0-4E51-9EF7-5939B793763C
```

Although you can run this command from any member, if you run it from a member that is currently restarting, the command cannot execute and must be retried from another member. For that reason, it is recommended that you run it from the captain. The captain is always the last member to restart, so the command will not fail until the end of the process, if you run it from the captain.

Back up and restore search head cluster settings

Search head clusters can usually recover from member failures without the need to manually restore configuration settings. Several of the other topics in this
chapter provide guidance on recovering from various sorts of member failure, in particular, see:

- Handle failure of a cluster member
- Use static captain to recover from loss of majority

In a functioning search head cluster, each member continually replicates changes to its state to the other members. This makes it possible to rebuild your cluster even if only one member remains intact.

However, to deal with catastrophic failure of a search head cluster, such as the failure of a data center, you can periodically back up the cluster state, so that you can later restore that state to a new or standby cluster, if necessary.

In addition, to deal with failure of the deployer, you can backup and restore the deployer's configuration bundle.

As with any backup-and-recovery scheme, test that these procedures work for you before you need them to work for you.

**Backup the search head cluster settings**

A backup of all search head cluster configurations requires two backups:

- The search head cluster state
- The deployer's configuration bundle

**Backup the search head cluster state**

On a cluster member, preferably the current captain:

1. Back up the most recent set of replicated configurations, located at
   $SPLUNK_HOME/var/run/splunk/snapshot/$LATEST_TIME-$CHECKSUM.bundle.
2. Back up the $SPLUNK_HOME/etc/system/local/server.conf file.
   **Note:** The only setting from this file that you will use when restoring from this backup is the id setting under the [shclustering] stanza. This setting is a unique identifier for the cluster, shared by all cluster members.
3. Back up the KV store:
   - splunk backup kvstore
   This command creates an archive file in the $SPLUNK_HOME/var/lib/splunk/kvstorebackup directory. See Back up the KV store in the *Admin Manual*.
4. Create a tarball containing the set of backups. This is your search head cluster configuration backup. Store it somewhere safe.

**Backup the deployer's configuration bundle**

Back up the deployer's `$SPLUNK_HOME/etc/shcluster` directory. This directory contains the configuration bundle that gets deployed to all cluster members.

**Restore the search head cluster settings**

You can restore the settings to either a new or an existing, standby cluster. The procedure documented here assumes that you are restoring to a standby cluster, but you can apply the main points of the procedure to a new cluster.

To restore a cluster's settings, restore two sets of configurations:

- The deployer's configuration bundle
- The search head cluster state

All members of both the old and new clusters, along with their deployers, must be running the same version of Splunk Enterprise, down to the maintenance level.

**Restore the deployer's configuration bundle**

This procedure assumes that you are restoring to a new deployer. If the old deployer is intact, you can reuse it by just pointing the new cluster members to it.

A deployer can only service a single cluster. The old cluster must be permanently inactive before you can use the existing deployer with the new cluster.

1. Stop all members of the standby search head cluster.
2. Copy the backup of the configuration bundle to the new deployer's `$SPLUNK_HOME/etc/shcluster` directory, overwriting the existing contents, if any.
3. Run the `splunk apply shcluster-bundle` command on the deployer:

   ```
   splunk apply shcluster-bundle -answer-yes -target <URI>:<management_port> -auth <username>:<password>
   See Push the configuration bundle.
   ```

Do not restart the standby cluster members at this point.
Restore the search head cluster state

1. Confirm that all members of the standby search head cluster are still stopped.
2. Untar the set of backups to a temporary location.
3. On each standby cluster member:
   1. Restore the replicated configurations:
      1. Move the replicated bundle $LATEST\_TIME\_CHECKSUM.bundle from the temporary location to $SPLUNK\_HOME/etc.
      2. Untar $LATEST\_TIME\_CHECKSUM.bundle.
   
   You must be working in the $SPLUNK\_HOME/etc directory when you untar $LATEST\_TIME\_CHECKSUM.bundle. The files in $LATEST\_TIME\_CHECKSUM.bundle are relative to $SPLUNK\_HOME/etc.
   
   3. To confirm that the files untarred properly, check for the presence of files in their proper location; for example, look for $SPLUNK\_HOME/etc/system/replication/ops.json.
4. Restore the KV store configurations. Follow the instructions in Restore the KV store data in the Admin Manual.
5. Restore the search head cluster id field. Edit $SPLUNK\_HOME/etc/system/local/server.conf and change the id setting in the shclustering stanza to use the value from the backup.

- Start all cluster members.
- Wait a few minutes for captain election to complete and for the deployer configuration bundle to be applied.
Troubleshoot search head clustering

Use the search head clustering dashboard

The search head clustering dashboard allows you to view the cluster configuration and perform some management actions on the cluster.

To access the search head clustering dashboard:

1. Click **Settings** on the upper right side of Splunk Web.
2. In the **Distributed Environment** group, click **Search head clustering**.

The dashboard provides basic information about the cluster, such as:

- The list of cluster members
- The status of each member
- The current captain
- The time of the last heartbeat to the captain

Several actions are available from the dashboard, including:

- **Begin rolling restart.** This action initiates a rolling restart of the cluster members. See "Restart the search head cluster."
- **Transfer captain.** This action is available for each member not currently the captain. It transfers captaincy to that member. See "Transfer captaincy."

**Note:** To transfer captaincy or perform rolling restart from the dashboard, all search head cluster members must be at release 6.6 or later.

Use the CLI to view information about a search head cluster

A number of CLI commands provide status information on the search head cluster.

You can also use the monitoring console to get more information about the cluster. See "Use the monitoring console to view search head cluster status and troubleshoot issues."
Show cluster status

To check the overall status of your search head cluster, run this command from any member:

```
splunk show shcluster-status -auth <username>:<password>
```

The command returns basic information on the captain and the cluster members. Key information that it provides includes:

- (Captain section.) The `dynamic_captain` field indicates whether the cluster uses a **dynamic captain**. A value of 1 specifies a dynamic captain.
- (Captain section.) The `id` field specifies the cluster GUID. This GUID is different from the GUID of any cluster members, including the captain.
- (Captain section.) The `label` field specifies the cluster label. The monitoring console uses the label identifier.
- (Each member’s section.) The `status` field specifies the status of each member, such as up, down, detention, restarting. Some status values require clarification:
  - **Detention.** A cluster member enters detention when it runs out of disk space. While in detention, the captain will not assign scheduled searches or artifact copies to it. To remediate, you must increase the disk space available to the instance.
  - **Down.** When a member leaves the cluster, because of some failure or because you remove it from the cluster, it enters the down state.
  - **Pending.** This indicates that the member is attempting to rejoin the cluster. This is a transitional state. The status changes to **Up** when the member successfully rejoins the cluster.
- (Each member’s section.) The `last_conf_replication` field indicates when the member last pulled a set of configurations from the captain. See View replication status.

Show member configuration

To check the configuration of a cluster member, run this command on the member itself:

```
splunk list shcluster-config -auth <username>:<password>
```

Alternatively, you can run this variant on another member:
splunk list shcluster-config -uri <URI>:<management_port> -auth <username>:<password>

Note the following:

- The -uri parameter specifies the URI and management port for the member whose configuration you want to check.

**List cluster members**

To get a list of all cluster members, run this command from any member:

```bash
splunk list shcluster-members -auth <username>:<password>
```

This command returns all members of the cluster, along with their configurations.

**Note:** The command continues to list members that have left the cluster until captaincy transfers.

**List member information**

To list information about a member, run this command on the member itself:

```bash
splunk list shcluster-member-info -auth <username>:<password>
```

Alternatively, you can run this variant on another member:

```bash
splunk list shcluster-member-info -uri <URI>:<management_port> -auth <username>:<password>
```

Note the following:

- The -uri parameter specifies the URI and management port for the member whose configuration you want to know.

**List search artifacts**

To list the set of artifacts stored on the cluster, run this command on the captain:

```bash
splunk list shcluster-artifacts
```

To list the set of artifacts stored on a particular member, run this command on the member itself:
splunk list shcluster-member-artifacts

List scheduler jobs

To list the set of scheduler jobs, run this command on the captain:

splunk list shcluster-scheduler-jobs -auth <username>:<password>

Use the monitoring console to view search head cluster status and troubleshoot issues

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

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

Search head clustering dashboards in the monitoring console

There are several search head clustering dashboards under the *Search* menu:

- Search Head Clustering: Status and Configuration
- Search Head Clustering: Configuration Replication
- Search Head Clustering: Artifact Replication
- Search Head Clustering: Scheduler Delegation
- Search Head Clustering: App Deployment

These dashboards provide a wealth of information about your search head cluster, such as:

- Cluster member instance names and status
- Identification of current captain and captain election activity
- Configuration replication performance
- Artifact replication details
- Scheduler activity
- Deployer activity

View the dashboards themselves for more information. In addition, see Search head clustering dashboards in *Monitoring Splunk Enterprise*.
Note: You can also use the CLI to get basic information about the cluster. See Use the CLI to view information about a search head cluster.

Troubleshoot the search head cluster

As part of its continuous monitoring of the search head cluster, the monitoring console provides a variety of information useful for troubleshooting. For example:

- The Search Head Clustering: Status and Configuration dashboard shows:
  - Search concurrency for various types of searches, with details on running versus limit
  - Status, including captaincy and state
  - Heartbeat information (discussed elsewhere in this topic)
  - Configuration baseline consistency (discussed elsewhere in this topic)
  - Artifact count
  - Election activity
- The Search Head Clustering: Configuration Replication dashboard shows:
  - Warning and error patterns
  - Configuration replication activity
- The Search Head Clustering: Artifact Replication dashboard shows:
  - Warning and error patterns
  - Artifact replication activity
- The Search Head Clustering: Scheduler Delegation dashboard shows:
  - Scheduler delegation activity
- The Search Head Clustering: App Deployment dashboard shows:
  - Status of app deployments

Troubleshoot heartbeat issues

The Search Head Clustering: Status and Configuration dashboard provides insight into the heartbeats that the cluster members send to the captain. Specifically, it shows, for each member:

- The time that the member last sent a heartbeat to the captain
- The time that the captain last received a heartbeat from the member

These times should be the same or nearly the same. Significant differences in the sent and received times indicate likely problems.

You can also access heartbeat information through the REST API. See the REST API documentation for shcluster/captain/members/{name}. 
The role of the heartbeat

Members send a heartbeat to the captain on a regular basis. By default, the member sends a heartbeat every five seconds.

The frequency is defined by the heartbeat_period attribute in the [shclustering] stanza of server.conf on each member. All members must set this attribute to the same value.

The heartbeat is the fundamental communication from the member to the captain. It indicates that the member is alive and part of the cluster. The heartbeat also contains a variety of information, such as:

- Search artifacts
- Dispatched searches
- Alerts and suppressions
- Completed summarization jobs
- Member load information

When the captain receives the heartbeat, it notes that the member is in the "up" state.

After the captain receives a heartbeat from every node, it consolidates all the transmitted information and, in turn, sends members information such as:

- Search artifact logs
- List of overall alerts and suppressions
- Dispatched searches

Impact of heartbeat failure

The captain expects to get a heartbeat from each member on a regular basis, as specified in the heartbeat_timeout attribute in the [shclustering] stanza of server.conf.

By default, the timeout is set to 60 seconds.

The captain only knows about the existence of a member through its heartbeat. If it never receives a heartbeat, it will not know that the member exists.

If, within the specified timeout period, the captain does not get a heartbeat from a member that has previously sent a heartbeat, the captain marks the member as "down". The captain does not dispatch new searches to members in the "down" state.
Causes of heartbeat failure

If the captain does not receive a heartbeat from a member, it usually indicates one of the following situations:

- Member is down or unavailable.
- Network partition between captain and member.
- HTTP request failures. These are visible in `splunkd_access.log` on the captain.

Note: By default, Splunk Enterprise logs only heartbeat failures in `splunkd_access.log`. To enable logging for heartbeat successes as well, configure `access_logging_for_heartbeats=true` in the `[shclustering]` stanza of `server.conf` on the captain. If you want this configuration change to persist across captaincy transfer, make the change on all members, not just the current captain.

Troubleshoot configuration baseline consistency

The Search Head Clustering: Status and Configuration dashboard includes information on the consistency of the configuration baseline. This information helps to determine whether configuration changes are being properly replicated across the set of cluster members.

To find this information, go to the Snapshots section of the dashboard and view the Status table. There is one row for each member. The table includes two columns that pertain to baseline consistency:

- **Configuration Baseline Consistency.** This column contains a ratio that compares the consistency of each member’s baseline to the baselines for all other members. For more details, click the ratio. A table to the right then compares the member’s baseline consistency against each individual member.

- **Number of Unpublished Changes.** This column indicates whether there are any sets of configuration changes on the member that have not yet been replicated to the captain. In particular, it notes whether a member is out-of-sync with the captain.

When a baseline mismatch is detected, at least one member requires manual intervention to regain baseline consistency. Examine the consistency comparison
table to identify the member that is not in sync with a majority of the other members. To restore consistency, perform a manual resync on the member, using the `splunk resync shcluster-replicated-config` command. See Perform a manual resync.

For a discussion of configuration replication, see Configuration updates that the cluster replicates.

**Deployment issues**

**Crash when adding new member**

If a member crashes when you add it to a cluster, determine whether the instance was previously a member of another cluster. If that is the case, you probably did not properly remove it from its previous cluster.

It is recommended that you always use new instances when adding members to a cluster, but if you choose to re-use an instance, you must follow the instructions in "Add a new member."

**Runtime considerations**

**Delays due to coordination between cluster members**

Coordination between the captain and other cluster members sometimes creates latency of up to 1.5 minutes. For example, when you save a search job, Splunk Web might not update the job's state for a short period of time. Similarly, it can take a minute or more for the captain to orchestrate the complete deletion of jobs.

In addition, when an event triggers the election of a new captain, there will be an interval of one to two minutes while the election completes. During this time, search heads can service only ad hoc job requests.

**Limit to number of active alerts**

The search head cluster can handle approximately 5000 active, unexpired alerts. To stay within this boundary, use alert throttling or limit alert retention time. See the Alerting Manual.
Site failure can prevent captain election

If the cluster is deployed across two sites and the site with a majority of members goes down or is otherwise inaccessible, the cluster cannot elect a new captain.

To remediate this situation, you can temporarily deploy a static captain. See "Use static captain to recover from loss of majority."

Handle Raft issues

If the Raft metadata that underlies search head clustering gets into a bad state on a member, you can often correct the problem by cleaning the member's var/run/splunk/_raft folder. See Fix Raft issues on a member.

If the cluster is unable to elect a captain and maintain a healthy state due to Raft issues, you can clean the Raft folder on all members and then bootstrap the cluster. See Fix the entire cluster.

Fix Raft issues on a member

The primary symptom of a Raft issue is that the member's status appears as "down" when you run splunk show shcluster-status on the captain. To confirm the Raft issue, look in the member's splunkd.log file for an error message that starts with the string "ERROR SHCRaftConsensus".

File corruption in a member's _raft folder is a common cause of Raft issues. You can fix the problem by cleaning the folder on the member. The folder then repopulates from the captain.

To fix a Raft issue, clean the member's _raft folder. Run the splunk clean raft command on the member:

1. Stop the member:
   splunk stop
2. Clean the member's raft folder:
   splunk clean raft
3. Start the member:
   splunk start
The `_raft` folder will be repopulated from the captain.

**Fix the entire cluster**

If captain election fails even though a majority of members are available, raft metadata corruption is a likely cause. To confirm, you can examine the members' `splunkd.log` files for errors that start with the string "ERROR SHCRaftConsensus".

You can resolve the issue by cleaning the folder on all members and then bootstrapping the cluster:

1. Stop all members.
2. Run `splunk clean raft` on each member:
   ```bash
   splunk clean raft
   ```
3. Start all members.
4. Select one member to be captain and bootstrap it:
   ```bash
   splunk bootstrap shcluster-captain -servers_list "<URI>:<management_port>,<URI>:<management_port>,..." -auth <username>:<password>
   ```
5. If you are using search peer replication, you must re-add the search peers to one member. See [Replicate the search peers across the cluster](#).
Search head pooling

Overview of search head pooling

<table>
<thead>
<tr>
<th>This feature has been deprecated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search head pooling has been deprecated as of Splunk Enterprise version 6.2. This means that although it continues to function, it might be removed in a future version.</td>
</tr>
<tr>
<td>As an alternative, you can deploy search head clustering. See About search head clustering.</td>
</tr>
<tr>
<td>For a list of all deprecated features, see the topic Deprecated features in the Release Notes.</td>
</tr>
</tbody>
</table>

**Important:** Search head pooling is an advanced feature. It’s recommended that you contact the Splunk sales team to discuss your deployment before attempting to implement it.

You can set up multiple search heads so that they share configuration and user data. This is known as **search head pooling**. The main reason for having multiple search heads is to facilitate horizontal scaling when you have large numbers of users searching across the same data. Search head pooling can also reduce the impact if a search head becomes unavailable. This diagram provides an overview of a typical deployment with search head pooling:
You enable search head pooling on each search head that you want to be included in the pool, so that they can share configuration and user data. Once search head pooling has been enabled, these categories of objects will be available as common resources across all search heads in the pool:

- **configuration data** -- configuration files containing settings for **saved searches** and other **knowledge objects**.
- **search artifacts**, records of specific search runs.
- **scheduler state**, so that only one search head in the pool runs a particular **scheduled report**.

For example, if you create and save a search on one search head, all the other search heads in the pool will automatically have access to it.

Search head pooling makes all files in `${SPLUNK_HOME}/etc/{apps,users}` available for sharing. This includes '*.conf files, *.meta files, view files, search scripts, lookup tables, etc.

**Key implementation issues**

Note the following:

- Most shared storage solutions don't perform well across a WAN. Since search head pooling requires low-latency shared storage capable of serving a high number of operations per second, implementing search head pooling across a WAN is not supported.
All search heads in a pool must be running the same version of Splunk Enterprise. Be sure to upgrade all of them at once. See "Upgrade your distributed Splunk Enterprise deployment" in the Installation Manual.

The purpose of search head pooling is to simplify the management of groups of dedicated search heads. Do not implement it on groups of indexers doubling as search heads. That is an unsupported configuration. Search head pooling has a significant effect on indexing performance.

The search heads in a pool cannot be search peers of each other.

Search head pooling and knowledge bundles

The set of data that a search head distributes to its search peers is known as the knowledge bundle. For details, see What search heads send to search peers.

By default, only one search head in a search head pool sends the knowledge bundle to the set of search peers. This optimization is controllable by means of the useSHPBundleReplication attribute in distsearch.conf.

As a further optimization, you can mount knowledge bundles on shared storage, as described in About mounted bundles. By doing so, you eliminate the need to distribute the bundle to the search peers. For information on how to combine search head pooling with mounted knowledge bundles, read Use mounted bundles with search head pooling.

For more information

See the other topics in this chapter for more information on search head pooling:

- "Create a search head pool"
- "Use a load balancer with the search head pool"
- "Other pooling operations"
- "Manage configuration changes"
- "Deployment server and search head pooling"
- "Select timing for configuration refresh"

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about search head pooling.
Create a search head pool

To create a pool of search heads, follow these steps:

1. Set up a shared storage location accessible to each search head.
2. Configure each individual search head.
3. Stop the search heads.
4. Enable pooling on each search head.
5. Copy user and app directories to the shared storage location.
6. Restart the search heads.

The steps are described below in detail:

1. Set up a shared storage location accessible to each search head

So that each search head in a pool can share configurations and artifacts, they need to access a common set of files via shared storage:

   - **On *nix platforms**, set up an NFS mount.
   - **On Windows**, set up a CIFS (SMB) share.

**Important:** The Splunk user account needs read/write access to the shared storage location. *When installing a search head on Windows, be sure to install it as a user with read/write access to shared storage. The Local System user does not have this access.* For more information, see "Choose the user Splunk should run as" in the Installation manual.

2. Configure each search head

   a. Set up each search head individually, specifying the search peers in the usual fashion. See "Add search peers to the search head".

   b. Make sure that each search head has a unique `serverName` attribute, configured in `server.conf`. See "Manage distributed server names" for detailed information on this requirement. If the search head does not have a unique
serverName, a warning will be generated at start-up. See "Warning about unique serverName attribute" for details.

c. Specify the necessary authentication. You have two choices:

- Specify user authentication on each search head separately. A valid user on one search head is not automatically a user on another search head in the pool. You can use LDAP to centrally manage user authentication, as described in "Set up user authentication with LDAP".

- Place a common authentication configuration on shared storage, to be used by all pool members. You must restart the pool members after any change to the authentication.

Note: Any authentication change made on an individual pool member (for example, via Splunk Web) overrides for that pool member only any configuration on shared storage. You should, therefore, generally avoid making authentication changes through Splunk Web if a common configuration already exists on shared storage.

3. Stop the search heads

Before enabling pooling, you must stop splunkd. Do this for each search head in the pool.

4. Enable pooling on each search head

Use the CLI command splunk pooling enable to enable pooling on a search head. The command sets certain values in server.conf. It also creates subdirectories within the shared storage location and validates that Splunk Enterprise can create and move files within them.

Here's the command syntax:

```
splunk pooling enable <path_to_shared_storage> [--debug]
```

Note:

- On NFS, `<path_to_shared_storage>` should be the NFS’s share mountpoint.
- On Windows, `<path_to_shared_storage>` should be the UNC path of the CIFS/SMB share.
• The `--debug` parameter causes the command to log additional information to `btool.log`.

Execute this command on each search head in the pool.

The command sets values in the `[pooling]` stanza of the `server.conf` file in `$SPLUNK_HOME/etc/system/local`.

You can also directly edit the `[pooling]` stanza of `server.conf`. For detailed information on `server.conf`, look here.

**Important:** The `[pooling]` stanza must be placed in the `server.conf` file directly under `$SPLUNK_HOME/etc/system/local/`. This means that you cannot deploy the `[pooling]` stanza via an app, either on local disk or on shared storage. For details see the `server.conf` spec file.

5. **Copy user and app directories to the shared storage location**

Copy the contents of the `$SPLUNK_HOME/etc/apps` and `$SPLUNK_HOME/etc/users` directories on an existing search head into the empty `/etc/apps` and `/etc/users` directories in the shared storage location. Those directories were created in step 4 and reside under the `<path_to_shared_storage>` that you specified at that time.

For example, if your NFS mount is at `/tmp/nfs`, copy the apps subdirectories that match this pattern:

```bash
$SPLUNK_HOME/etc/apps/*
```

into

```bash
/tmp/nfs/etc/apps
```

This results in a set of subdirectories like:

```bash
/tmp/nfs/etc/apps/search
/tmp/nfs/etc/apps/launcher
/tmp/nfs/etc/apps/unix
[...]
```

Similarly, copy the user subdirectories:
$SPLUNK_HOME/etc/users/*

into

/tmp/nfs/etc/users

**Important:** You can choose to copy over just a subset of apps and user subdirectories; however, be sure to move them to the precise locations described above.

### 6. Restart the search heads

After running the `splunk pooling enable` command, restart `splunkd`. Do this for each search head in the pool.

### Use a load balancer with the search head pool

You will probably want to run a load balancer in front of your search heads. That way, users can access the pool of search heads through a single interface, without needing to specify a particular one.

Another reason for using a load balancer is to ensure access to search artifacts and results if one of the search heads goes down. Ordinarily, RSS and email alerts provide links to the search head where the search originated. If that search head goes down (and there's no load balancer), the artifacts and results become inaccessible. However, if you've got a load balancer in front, you can set the alerts so that they reference the load balancer instead of a particular search head.

### Configure the load balancer

There are a couple issues to note when selecting and configuring the load balancer:

- The load balancer must employ layer-7 (application-level) processing.
  - Configure the load balancer so that user sessions are "sticky" or "persistent". This ensures that the user remains on a single search head throughout their session.
Generate alert links to the load balancer

To generate alert links to the load balancer, you must edit alert_actions.conf:

1. Copy alert_actions.conf from a search head to the appropriate app directory in the shared storage location. In most cases, this will be /<path_to_shared_storage>/etc/apps/search/local.

2. Edit the hostname attribute to point to the load balancer:

hostname = <proxy host>:<port>

For details, see alert_actions.conf in the Admin manual.

The alert links should now point to the load balancer, not the individual search heads.

Other pooling operations

Besides the splunk pooling enable CLI command, there are several other commands that are important for managing search head pooling:

- splunk pooling validate
- splunk pooling disable
- splunk pooling display

You must stop splunkd before running splunk pooling enable or splunk pooling disable. However, you can run splunk pooling validate and splunk pooling display while splunkd is either stopped or running.

Validate that each search head has access to shared resources

The splunk pooling enable command validates search head access when you initially set up search head pooling. If you ever need to revalidate the search head's access to shared resources (for example, if you change the NFS configuration), you can run the splunk pooling validate CLI command:

splunk pooling validate [--debug]
Disable search head pooling

You can disable search head pooling with this CLI command:

```
splunk pooling disable [--debug]
```

Run this command for each search head that you need to disable.

**Important:** Before running the `splunk pooling disable` command, you must stop `splunkd`. After running the command, you should restart `splunkd`.

Display pooling status

You can use the `splunk pooling display` CLI command to determine whether pooling is enabled on a search head:

```
splunk pooling display
```

This example shows how the system response varies depending on whether pooling is enabled:

```
$ splunk pooling enable /foo/bar
$ splunk pooling display
Search head pooling is enabled with shared storage at: /foo/bar
$ splunk pooling disable
$ splunk pooling display
Search head pooling is disabled
```

Manage configuration changes

**Important:** Once pooling is enabled on a search head, you must notify the search head whenever you directly edit a configuration file.

Specifically, if you add a stanza to any configuration file in a local directory, you must run the following command:

```
splunk btool fix-dangling
```

**Note:** This is not necessary if you make changes by means of Splunk Web or the CLI.
Deployment server and search head pooling

With search head pooling, all search heads access a single set of configurations, so you don’t need to use a deployment server or a third party deployment management tool like Puppet to push updates to multiple search heads. However, you might still want to use a deployment tool with search head pooling, in order to consolidate configuration operations across all Splunk Enterprise instances.

If you want to use the deployment server to manage your search head configuration, note the following:

1. Designate one of the search heads as a deployment client by creating a deploymentclient.conf file in $SPLUNK_HOME/etc/system/local and specifying its deployment server. You only need to designate one search head as a deployment client.

2. In deploymentclient.conf, set the repositoryLocation attribute to the search head’s shared storage mountpoint. You must also set serverRepositoryLocationPolicy=rejectAlways, so that the locally set repositoryLocation gets used as the download location.

3. In serverclass.conf on the deployment server, define a server class for the search head client.

For detailed information on the deployment server, see "About deployment server" in the Updating Splunk Enterprise Instances manual.

Select timing for configuration refresh

In version 5.0.2 and earlier, the defaults for synchronizing from the storage location were set to very frequent intervals. This could lead to excessive time spent reading configuration changes from the pool, particularly in deployments with large numbers of users (in the hundreds or thousands).

The default settings have been changed to less frequent intervals starting with 5.0.3. In server.conf, the following settings affect configuration refresh timing:

```shell
# 5.0.3 defaults
[pooling]
poll.interval.rebuild = 1m
```
poll.interval.check = 1m

The previous defaults for these settings were 2s and 5s, respectively.

With the old default values, a change made on one search head would become available on another search head at most seven seconds later. There is usually no need for updates to be propagated that quickly. By changing the settings to values of one minute, the load on the shared storage system is greatly reduced. Depending on your business needs, you might be able to set these values to even longer intervals.

**Upgrade a search head pool**

All search heads in a pool must be running the same version of Splunk Enterprise.

For the upgrade procedure, see "Upgrade your distributed Splunk Enterprise deployment" in the *Installation Manual*. Read this procedure carefully before attempting to upgrade your search head pool. You must follow the steps precisely to ensure that the pool remains fully functional.
Distributed search in action

How authorization works in distributed searches

The authorization settings that a search peer uses when processing distributed searches are different from those that it uses for its local activities, such as administration and local search requests:

- When processing a distributed search, the search peer uses the settings contained in the knowledge bundle that the search head distributes to all the search peers when it sends them a search request. These settings are created and managed on the search head.
- When performing local activities, the search peer uses the authorization settings created and stored locally on the search peer itself.

When managing distributed searches, it is therefore important that you distinguish between these two types of authorization. You need to be particularly aware of how authorization settings get distributed through the knowledge bundle when you're managing a system with search head pooling or mounted bundles.

For background information, read about these key concepts:

- **Splunk Enterprise authorization**: The topic "About role-based user access" in the Securing Splunk Enterprise manual
- **Mounted bundles**: The chapter "Mount the knowledge bundle" in this manual
- **Search head pooling**: The chapter "Search head pooling" in this manual

Manage authorization for distributed searches

All authorization settings are stored in one or more authorize.conf files. This includes settings configured through Splunk Web or the CLI. It is these authorize.conf files that get distributed from the search head to the search peers. On the knowledge bundle, the files are usually located in either /etc/system/{local, default} and/or /etc/apps/<app-name>/ {local, default}.

Since search peers automatically use the settings in the knowledge bundle, things normally work fine. You configure roles for your users on the search head, and the search head automatically distributes those configurations to the search peers when it distributes the search itself.
With search head pooling, however, you must take care to ensure that the search heads and the search peers all use the same set of authorize.conf file(s). For this to happen, you must make sure:

- All search heads in the pool use the same set of authorize.conf files
- The set of authorize.conf files that the search heads use goes into the knowledge bundle so that they get distributed to the search peers.

This topic describes the four main scenarios, based on whether or not you’re using search head pooling or mounted bundles. It describes the scenarios in order from simple to complex.

**Four scenarios**

What you need to do with the distributed search authorize.conf files depends on whether your deployment implements search head pooling or mounted bundles. The four scenarios are:

- No search head pooling, no mounted bundles
- No search head pooling, mounted bundles
- Search head pooling, no mounted bundles
- Search head pooling, mounted bundles

The first two scenarios "just work" but the last two scenarios require careful planning. For the sake of completeness, this section describes all four scenarios.

**Note:** These scenarios address authorization settings for distributed search only. Local authorization settings function the same independent of your distributed search deployment.

**No search head pooling, no mounted bundles**

Whatever authorization settings you have on the search head get automatically distributed to its search peers as part of the replicated knowledge bundle that they receive with distributed search requests.

**No search head pooling, mounted bundles**

Whatever authorization settings you have on the search head get automatically placed in the mounted bundle and used by the search peers during distributed search processing.
Search head pooling, no mounted bundles

The search heads in the pool share their /apps and /users directories but not their /etc/system/local directories. Any authorize.conf file in an /apps subdirectory will be automatically shared by all search heads and included in the knowledge bundle when any of the search heads distributes a search request to the search peers.

The problem arises because authorization changes can also get saved to an authorize.conf file in a search head's /etc/system/local directory (for example, if you update the search head's authorization settings via Splunk Web). This directory does not get shared among the search heads in the pool, but it still gets distributed to the search peers as part of the knowledge bundle. Because of how the configuration system works, any copy of authorize.conf file in /etc/system/local will have precedence over a copy in an /apps subdirectory. (See "Configuration file precedence" in the Admin manual for details.)

Therefore, a copy of authorize.conf that gets distributed to the search peers from a single search head's /etc/system/local directory has precedence over any copies distributed from the search head pool's shared directory. Unless you account for this situation, the search peers can end up using different authorization settings for different searches, depending on which search head distributed the search to them. For most situations, this is not what you want to occur.

To avoid this problem, you need to make sure that any changes made to a search head's /etc/system/local/authorize.conf file get propagated to all search heads in the pool. One way to handle this is to move any changed /etc/system/local/authorize.conf file into an app subdirectory, since all search heads in the pool share the /apps directory.

Search head pooling, mounted bundles

This is similar to the previous scenario. The search heads in the pool share their /apps and /users directories but not their /etc/system/local directories. Any authorize.conf file in an /apps subdirectory will be automatically shared by all search heads. It will also be included in the mounted bundle that the search peers use when processing a search request from any of the search heads.

However, authorization changes can also wind up in an authorize.conf file in a search head's /etc/system/local directory (for example, if you update the search head's authorization settings via Splunk Web). This directory does not get automatically shared among the search heads in the pool. It also does not get
automatically distributed to the mounted bundle that the search peers use. Therefore, you must provide some mechanism that ensures that all the search heads and all the search peers have access to that version of authorize.conf.

The simplest way to handle this is to move any changed /etc/system/local/authorize.conf file into an app subdirectory, since both the pooled search heads and all the search peers share the /apps directory.

**How users can control distributed searches**

From the user standpoint, specifying and running a distributed search is essentially the same as running any other search. Behind the scenes, the search head distributes the query to its search peers and consolidates the results when presenting them to the user.

Users can limit the search peers that participate in a search. They also need to be aware of the distributed search configuration to troubleshoot.

**Perform distributed searches**

In general, you specify a distributed search through the same set of commands as for a local search. However, several additional commands and options are available specifically to assist with controlling and limiting a distributed search.

A search head by default runs its searches across its full set of search peers. You can limit a search to one or more search peers by specifying the splunk_server field in your query. See Retrieve events from indexes in the Search Manual.

The search command localop is also of use in defining distributed searches. It enables you to limit the execution of subsequent commands to the search head. See the description of localop in the Search Reference for details and an example.

In addition, the lookup command provides a local argument for use with distributed searches. If set to true, the lookup occurs only on the search head; if false, the lookup occurs on the search peers as well. This is particularly useful for scripted lookups, which replicate lookup tables. See the description of lookup in the Search Reference for details and an example.
Troubleshoot distributed search

Use the monitoring console to view distributed search status

You can use the monitoring console to monitor most aspects of your deployment. This topic discusses the Search Activity: Deployment dashboard, which provides insight into your distributed searches.

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

The Search Activity: Deployment dashboard provides a range of useful information about your distributed search environment and processes, such as:

- Search activity for each search head
- Historical charts that provide information on search concurrency and CPU usage over time

The monitoring console provides other dashboards that show search activity for single instances.

View the dashboards themselves for more information. In addition, see Search activity: Deployment in *Monitoring Splunk Enterprise*.

General troubleshooting issues

Clock skew between search heads and search peers can affect search behavior

You must keep the clocks on your search heads and search peers in sync, via NTP (network time protocol) or some similar means. The nodes require close clock alignment, so that time comparisons are valid across systems. If the clocks are out-of-sync by more than a few seconds, distributed search cannot work correctly, resulting in search failures or premature expiration of search artifacts.

When you add a search peer to a search head, the search head checks that the clocks are in sync. This check ensures that the system time, independent of the timezone, agrees across the nodes of a distributed search environment. If the
nodes are out of sync, the search head rejects the search peer and displays a banner message like this:

The time difference between this system and the intended peer at uri=https://servername:8089/ was too big. Please bring the system clocks into agreement.

Note: The search head does not run this check if you add the search peer by direct edit of distsearch.conf.

Searches can fail if configurations in a knowledge bundle have not yet been replicated to search peers

Configuration changes can take a short time to propagate from search heads to search peers. As a result, during the time between when configuration changes are made on the search head and when they’re replicated to the search peers (typically, not more than a few minutes), distributed searches can either fail or provide results based on the previous configuration.

Types of configuration changes that can cause search failures are those that involve new apps or changes to authentication.conf or authorize.conf. Examples include:

- changing the allowed indexes for a role and then running a search as a user within that role
- creating a new app and then running a search from within that app

Any failures will be noted in messages on the search head.

Types of changes that can provide results based on the previous configuration include changing a field extraction or a lookup table file.

To remediate, run the search again.

Network problems can reduce search performance

A 6.0 or later search head by default asks its search peers to generate a remote timeline. This can result in slow searches if the connection between the search head and the search peers is unstable.

The workaround is to add the following setting to limits.conf on the search head:
remote_timeline_fetchall = false

After making this change, you must restart the search head.

Handle slow search peers

A search normally continues to run until all search peers return the requested data. This can sometimes create a problem in deployments with very large numbers of search peers (100+). If one of the peers is much slower than the others in returning its portion of the data (for example, due to network issues), the searches can continue for abnormally long periods of time while awaiting the final results from that peer.

If such a situation arises and you want to trade data fidelity for search performance, you can direct the search head to end long-running searches without waiting for a slow peer to finish sending all its data. To do this, you enable the search head's [slow_peer_disconnect] stanza in limits.conf. By default, this capability is disabled. You can toggle the capability without restarting the search head.

The heuristics that determine when to disconnect a search from a slow peer are complex and tunable by means of several parameters in the [slow_peer_disconnect] stanza. If you feel the need to use this capability, contact Splunk Professional Services for guidance in adjusting the heuristics for your specific deployment needs.

Quarantine a search peer

You can quarantine a search peer to prevent it from partaking in future searches. This is of value if the peer is experiencing problems, for example, due to a bad disk or network card. It can also be useful to quarantine a search peer while you upgrade it.

By quarantining, instead of stopping, a bad search peer, you can perform live troubleshooting on the peer.

You can override a quarantine for a specific search, if necessary. See How to override a quarantine.
What happens when you quarantine a search peer

When you quarantine a search peer, you prevent it from taking part in new searches. It continues to attempt to service any currently running searches.

The quarantine operation affects only the relationship between the search peer and its search head. The search peer continues to receive and index incoming data in its role as an indexer. If the peer is a member of an indexer cluster, it also continues to replicate data from other peer nodes.

If you need to fully halt the activities of the indexer, you must bring it down.

How to quarantine a search peer

To quarantine a search peer, run this CLI command from the search head:

```
splunk edit search-server -auth <user>:<password> <host>:<port> -action quarantine
```

Note the following:

- Use the `-auth` flag to provide credentials for the search head only.
- `<host>` is the host name or IP address of the search peer's host machine.
- `<port>` is the management port of the search peer.

For example:

```
splunk edit search-server -auth admin:password 10.10.10.10:8089 -action quarantine
```

In a search head cluster, this command affects only the search head that it is run on. To quarantine a peer for all cluster members, you must run this command on each member.

You can also quarantine a search peer through the Search peers page on the search head's Splunk Web. See View search peer status in Settings.

How to unquarantine a search peer

To remove a search peer from quarantine, run this command from the search head:

```
splunk edit search-server -auth <user>:<password> <host>:<port> -action unquarantine
```

216
unquarantine

Note the following:

- Use the `-auth` flag to provide credentials for the search head only.
- `<host>` is the host name or IP address of the search peer’s host machine.
- `<port>` is the management port of the search peer.

For example:

```bash
splunk edit search-server -auth admin:password 10.10.10.10:8089 -action unquarantine
```

**How to override a quarantine**

When a peer is quarantined, it does not ordinarily participate in searches. You can, however, override the quarantine on a search-by-search basis. To do so, the search must target the peer directly with the `splunk_server` field. For example:

```bash
index=_internal splunk_server=idx-tk421-03 (log_level=WARN OR log_level=ERROR)
```

**Note:** If the peer is a member of a distributed search group, you cannot override the quarantine by specifying the `splunk_server_group` field of its search group. You must specify the peer directly with the `splunk_server` field.

**Search head pooling configuration issues**

When implementing search head pooling, there are a few potential issues you should be aware of, mainly having to do with coordination among search heads.

**Authentication and authorization changes made in Splunk Web apply only to a single search head**

Authentication and authorization changes made through a search head’s Splunk Web apply only to that search head and not to other search heads in that pool. Each member of the pool maintains its local configurations in `$SPLUNK_HOME/etc/system/local`. To share configurations across the pool, set them up in shared storage, as described in "Configure search head pooling".
Clock skew between search heads and shared storage can affect search behavior

It's important to keep the clocks on your search heads and shared storage server in sync, via NTP (network time protocol) or some similar means. If the clocks are out-of-sync by more than a few seconds, you can end up with search failures or premature expiration of search artifacts.

Permission problems on the shared storage server can cause pooling failure

On each search head, the user account Splunk runs as must have read/write permissions to the files on the shared storage server.

Performance analysis

A large percentage of search head pooling issues boil down to insufficient performance.

When deploying or investigating a search head pooling environment, it's important to consider these factors:

- **Storage**: The storage backing the pool must be able to handle a very high number of IOPS. IOPS under 1000 will probably never work well.
- **Network**: The communication path between the backing store and the search heads must be high bandwidth and extremely low latency. This probably means your storage system should be on the same switch as your search heads. WAN links are not going to work.
- **Server Parallelism**: Because searching results in a large number of processes requesting a large number of files, the parallelism in the system must be high. This can require tuning the NFS server to handle a larger number of requests in parallel.
- **Client Parallelism**: The client operating system must be able to handle a significant number of requests at the same time.

To validate an environment, a typical approach would be:

- Use a storage benchmarking tool, such as Bonnie++, while the file store is not in use to validate that the IOPS provided are robust.
- Use network testing methods to determine that the roundtrip time between search heads and the storage system is on the order of 10ms.
• Perform known simple tasks such as creating a million files and then deleting them.
• Assuming the above tests have not shown any weaknesses, perform some IO load generation or run the actual Splunk Enterprise load while gathering NFS stat data to see what's happening with the NFS requests.

**NFS client concurrency limits can cause search timeouts or slow search behavior**

The search performance in a search head pool is a function of the throughput of the shared storage and the search workload. The combined effect of concurrent search users and concurrent scheduled searches running will yield a total IOPs that the shared volume needs to support. IOP requirements will also vary by the kind of searches run. To adequately provision a device to be shared between search heads, you need to know the number of concurrent users submitting searches and the number of jobs/apps that will be executed simultaneously.

If searches are timing out or running slowly, you might be exhausting the maximum number of concurrent requests supported by the NFS client. To solve this problem, increase your client concurrency limit. For example, on a Linux NFS client, adjust the `tcp_slot_table_entries` setting.

**NFS latency for large user count can incur configuration access latency or slow dispatch reaping**

Splunk Enterprise synchronizes the search head pool storage configuration state with the in-memory state when it detects changes. Essentially, it reads the configuration into memory when it detects updates. When dealing either with overloaded search pool storage or with large numbers of users, apps, and configuration files, this synchronization process can reduce performance. To mitigate this, the minimum frequency of reading can be increased, as discussed in "Select timing for configuration refresh".

**Warning about unique serverName attribute**

Each search head in the pool must have a unique `serverName` attribute. Splunk Enterprise validates this condition when each search head starts. If it finds a problem, it generates this error message:

```markdown
serverName "<xxx>" has already been claimed by a member of this search head pool
in <full path to pooling.ini on shared storage>
```
There was an error validating your search head pooling configuration. For more information, run 'splunk pooling validate'.

The most common cause of this error is that another search head in the pool is already using the current search head's serverName. To fix the problem, change the current search head's serverName attribute in .system/local/server.conf.

There are a few other conditions that also can generate this error:

- The current search head's serverName has been changed.
- The current search head's GUID has been changed. This is usually due to /etc/instance.cfg being deleted.

To fix these problems, run

```
splunk pooling replace-member
```

This updates the pooling.ini file with the current search head's serverName->GUID mapping, overwriting any previous mapping.

**Artifacts and incorrectly displayed items in Splunk Web after upgrade**

When upgrading pooled search heads, you must copy all updated apps - even those that ship with Splunk Enterprise (such as the Search app) - to the search head pool's shared storage after the upgrade is complete. If you do not, you might see artifacts or other incorrectly-displayed items in Splunk Web.

To fix the problem, copy all updated apps from an upgraded search head to the shared storage for the search head pool, taking care to exclude the local sub-directory of each app.

**Important:** Excluding the local sub-directory of each app from the copy process prevents the overwriting of configuration files on the shared storage with local copies of configuration files.

Once the apps have been copied, restart Splunk Enterprise on all search heads in the pool.
Distributed search error messages

This table lists some of the more common search-time error messages associated with distributed search:

<table>
<thead>
<tr>
<th>Error message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>status=down</td>
<td>The specified remote peer is not available.</td>
</tr>
<tr>
<td>status=not a splunk server</td>
<td>The specified remote peer is not a Splunk Enterprise server.</td>
</tr>
<tr>
<td>duplicate license</td>
<td>The specified remote peer is using a duplicate license.</td>
</tr>
<tr>
<td>certificate mismatch</td>
<td>Authentication with the specified remote peer failed.</td>
</tr>
</tbody>
</table>