Splunk® Enterprise Distributed Search 7.3.2

Search head clustering architecture

Generated: 10/11/2019 4:47 pm
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:
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 multiplies 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 concurrent search number.
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:
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`.

- 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.