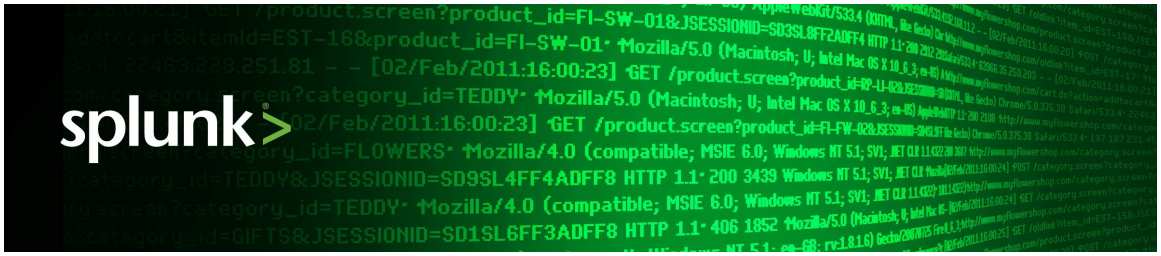

Splunk® Enterprise Managing Indexers and Clusters of Indexers 5.0.1

How Splunk stores indexes

Generated: 1/14/2017 10:31 pm

How Splunk stores indexes

As Splunk indexes your data, it creates a bunch of files. These files contain two types of data:

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

Together, these files constitute the Splunk **index**. The files reside in sets of directories organized by age. Some directories contain newly indexed data; others contain previously indexed data. The number of such directories can grow quite large, depending on how much data you're indexing.

Why you might care

You might not care, actually. Splunk handles indexed data by default in a way that gracefully ages the data through several stages. After a long period of time, typically several years, Splunk removes old data from your system. You might well be fine with the default scheme it uses.

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

How Splunk ages data

Each of the index directories is known as a **bucket**. To summarize so far:

- A Splunk "index" contains compressed raw data and associated index files.
- A Splunk index resides across many age-designated index directories.
- An index directory is a bucket.

A bucket moves through several stages as it ages:

- hot
- warm
- cold
- frozen

As buckets age, they "roll" from one stage to the next. Newly indexed data goes into a hot bucket, which is a bucket that's both searchable and actively being written to. After the hot bucket reaches a certain size, it becomes a warm bucket

("rolls to warm"), and a new hot bucket is created. Warm buckets are searchable, but are not actively written to. There are many warm buckets.

Once Splunk has created some maximum number of warm buckets, it begins to roll the warm buckets to cold based on their age. Always, the oldest warm bucket rolls to cold. Buckets continue to roll to cold as they age in this manner. After a set period of time, cold buckets roll to frozen, at which point they are either archived or deleted. By editing attributes in indexes.conf, you can specify the bucket aging policy, which determines when a bucket moves from one stage to the next.

Here are the stages that buckets age through:

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

The collection of buckets in a particular stage is sometimes referred to as a database or "db": the "hot db", the "warm db", the "cold db", etc.

Note: Hot buckets *always* roll when `splunkd` gets restarted.

What the index directories look like

Each bucket occupies its own subdirectory within a larger database directory. Splunk organizes the directories to distinguish between hot/warm/cold buckets. In addition, the bucket directory names are based on the age of the data.

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

Bucket type	Default location	Notes

Hot	`$SPLUNK_HOME/var/lib/splunk/defaultdb/db/*`	There can be multiple hot subdirectories. Each hot bucket occupies its own subdirectory, which uses this naming convention: `hot_v1_<localid>`
Warm	`$SPLUNK_HOME/var/lib/splunk/defaultdb/db/*`	There are separate subdirectories for each warm bucket. These are named as described below in "Warm/cold bucket naming convention".
Cold	`$SPLUNK_HOME/var/lib/splunk/defaultdb/colddb/*`	There are multiple cold subdirectories. When warm buckets roll to cold, they get moved into this directory, but are not renamed. **Note:** In a **cluster**, all *replicated* copies of buckets reside in the `colddb` directory. It doesn't matter whether they're hot, warm, or cold. The original copies of cluster buckets, however,

		reside in their normal locations, as described in this table. For example, if clusterpeer1 creates a new hot bucket and then replicates it to clusterpeer2, the original copy of the hot bucket on clusterpeer1 will reside in the `db` directory, while the replicated copy of the hot bucket on clusterpeer2 will reside in the `colddb` directory. For detailed information on clusters and buckets, read "Buckets and clusters" in the cluster section of this manual.
Frozen	N/A: Frozen data gets deleted or archived into a directory location you specify.	Deletion is the default; see "Archive indexed data" for information on how to archive the data instead.
Thawed	`$SPLUNK_HOME/var/lib/splunk/defaultdb/thaweddb/*`	Location for data that has been archived and later thawed. See "Restore archived data" for information on

		restoring archived data to a thawed state.

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

Important: All index locations must be writable.

Warm/cold bucket naming convention

Bucket names identify the time range for the data they contain. The naming convention varies slightly, depending on whether the bucket rolled to warm while its indexer was functioning as a cluster peer:

- For non-clustered buckets: `db_<newest_time>_<oldest_time>_<localid>`
- For clustered original bucket copies: `db_<newest_time>_<oldest_time>_<localid>_<guid>`
- For clustered replicated bucket copies: `rb_<newest_time>_<oldest_time>_<localid>_<guid>`

where:

- `<newest_time>` and `<oldest_time>` are timestamps indicating the age of the data within. The timestamps are expressed in UTC epoch time (in seconds). For example: `db_1223658000_1223654401_2835` is a warm, non-clustered bucket containing data from October 10, 2008, covering the exact period of 9am-10am.
- `<localid>` is an ID for the bucket, generated on the indexer on which the bucket originated.
- `<guid>` is the `guid` of the peer node (indexer) on which the bucket originated. The `guid` is located in the peer's `$SPLUNK_HOME/etc/instance.cfg` file.

In a cluster, the original buckets and its replicated copies have identical names, except for the prefix (`db` for the original bucket; `rb` for the copies).

Note: In a cluster, when data is streaming from the originating peer to a target peer, the data first goes into a temporary directory on the target peer, which is identified by the originating peer's `<localid>` and `<guid>`, like this: `<localid>_<guid>`. This is true, independent of the type of bucket the data is being streamed from. When the replication has completed, the directory is rolled into a warm bucket, identified by the `rb_` prefix, as described above. For an introduction to cluster architecture and replicated data streaming, read "Basic

cluster architecture".

Important: The bucket naming convention is subject to change.

Buckets and Splunk administration

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

For information on setting a retirement and archiving policy, see "Set a retirement and archiving policy". You can base the retirement policy on either size or age of data.

For information on how to archive your indexed data, see "Archive indexed data". For information on archive signing, see "Configure archive signing" in the Security Manual. To learn how to restore data from archive, read "Restore archived data".

To learn how to back up your data, read "Back up indexed data". That topic also discusses how to manually roll hot buckets to warm (so that you can then back them up). Also, see "Best practices for backing up" on the Community Wiki.

For information on setting limits on disk usage, see "Set limits on disk usage".

For a list of configurable bucket settings, see "Configure index storage".

Troubleshoot your buckets

This section tells you how to deal with an assortment of bucket problems.

Recover after a crash

Splunk usually handles crash recovery without your intervention. If an indexer goes down unexpectedly, some recently received data might not be searchable. When you restart Splunk, it will automatically run the `fsck` command in the background. This command diagnoses the health of your buckets and rebuilds search data as necessary.

It is highly unlikely that you will need to run `fsck` manually. This is a good thing, because to run it manually, you must stop Splunk, and the command can take several hours to complete if your indexes are large. During that time your data will be inaccessible. However, if Splunk Support directs you to run it, the rest of this section tells you how to do so. (Also, you will need to run `fsck`

manually to perform recovery for any 4.2.x indexers. Only Splunk indexers at version 4.3 or above run it automatically.)

To run `fsck` manually, you'll need to first stop Splunk. Then run `fsck` against any affected buckets. To run `fsck` against buckets in all indexes, use this command:

```
splunk fsck --repair --all
```
This will rebuild all types of buckets (hot/warm/cold/thawed) in all indexes.

Note: The `fsck` command only rebuilds buckets created by version 4.2 or later of Splunk.

To learn more about the `fsck` command, including a list of all options available, enter:

```
splunk fsck
```
Warning: The `fsck --repair` command can take as long as several hours to run, depending on the size of your indexes. That's why you want to let Splunk run it in the background automatically, if possible. Also, if you can determine that you only need to rebuild a few buckets, you can run the `rebuild` command on just those buckets, as described in the next section, "Rebuild a bucket."

If you just want to diagnose the state of your indexes (without taking any immediate remedial action), run `fsck` without the `--repair` flag:

```
splunk fsck --all
```
For single bucket you cannot use fsck, instead use the rebuild option explained below.

Rebuild a bucket

If the index and metadata files in a bucket (version 4.2 and later) somehow get corrupted, you can rebuild the bucket from the raw data file alone. Use this command:

```
splunk rebuild <bucket directory>
```
Splunk automatically deletes the old index and metadata files and rebuilds them. You don't need to delete any files yourself.

Important: You must stop Splunk before running the `rebuild` command.

A few notes:

- Rebuilding a bucket does not count against your license.
- The time required to rebuild a bucket can be significant. Depending on various system considerations, such as your hardware specifications, it can take anywhere from half an hour to a few hours to rebuild a 10 GB bucket.

Recover invalid pre-4.2 hot buckets

A hot bucket becomes an invalid hot (`invalid_hot_<ID>`) bucket when Splunk detects that the metadata files (`Sources.data, Hosts.data, SourceTypes.data`) are corrupt or incorrect. Incorrect data usually signifies incorrect time ranges; it can also mean that event counts are incorrect.

Splunk ignores invalid hot buckets. Data does not get added to such buckets, and they cannot be searched. Invalid buckets also do not count when determining bucket limit values such as `maxTotalDataSizeMB`. This means that invalid buckets do not negatively affect the flow of data through the system, but it also means that they can result in disk storage that exceeds the configured maximum value.

To recover an invalid hot bucket, use the `recover-metadata` command:

1. Make backup copies of the metadata files, `Sources.data, Hosts.data, SourceTypes.data`.

2. Rebuild the metadata from the raw data information:

```
splunk cmd recover-metadata path_to_your_hot_buckets/invalid_hot_<ID>
```

3. If successful, rename the bucket as it would normally be named.

Rebuild index-level bucket manifests

It is rare that you might have reason to rebuild index-level manifests, but if you need to, Splunk provides a few commands that do just that.

Caution: You should only use these commands if Splunk support directs you to. *Do not rebuild the manifests on your own.*

The two index-level manifest files are `.bucketManifest` and `.metaManifest`. The `.bucketManifest` file contains a list of all buckets in the index. You might need to rebuild this if, for example, you manually copy a bucket into an index. The `.metaManifest` file contains a list of buckets that have contributed to the index-level metadata file.

The following command rebuilds the `.bucketManifest` and `.metaManifest` files and all `*.data` files in the homePath for the main index **only**. It does **not** rebuild metadata for individual buckets:

```
splunk _internal call /data/indexes/main/rebuild-metadata-and-manifests
```

If you only want to rebuild the `.metaManifest` and `homePath/*.data` files, use this command instead:

```
splunk _internal call /data/indexes/main/rebuild-metadata
```

If you only want to rebuild the `.bucketManifest` file, use this command:

```
splunk _internal call /data/indexes/main/rebuild-bucket-manifest
```

You can use the asterisk (*) wildcard to rebuild manifests for all indexes. For example:

```
splunk _internal call /data/indexes/*/rebuild-metadata-and-manifests
```

For more information

For more information on index storage and buckets, read these topics:

- "Configure index storage"
- "Use multiple partitions for index data"
- "Configure index storage size"
- "Buckets and clusters".

In addition, see "indexes.conf" in the Admin manual and "Understanding buckets" on the Community Wiki.