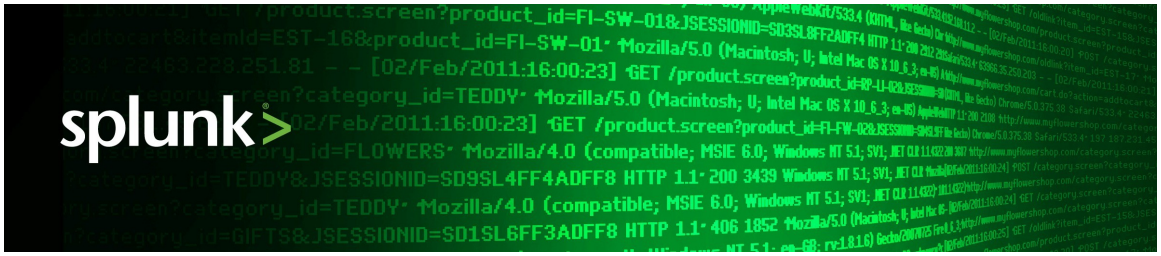


Splunk® Enterprise Forwarding Data 6.5.0

Generated: 11/09/2016 11:18 am

Table of Contents

Introduction to forwarding....1
About forwarding and receiving....1
Types of forwarders....2

Plan your deployment....7
Forwarder deployment topologies....7
Compatibility between forwarders and indexers....11

Deploy heavy and light forwarders....13
Enable forwarding on a Splunk Enterprise instance....13
Heavy and light forwarder capabilities....14
Enable a receiver....17
Deploy a heavy forwarder....19
Deploy a light forwarder....21

Configure forwarders....24
Configure data collection on forwarders with inputs.conf....24
Configure forwarders with outputs.conf....25

Upgrade forwarders....34
Upgrade heavy and light forwarders....34

Perform advanced configuration....35
Set up load balancing....35
Configure a forwarder to use a SOCKS proxy....39
Configure an intermediate forwarder....42
Protect against loss of in-flight data....45
Route and filter data....51
Forward data to third-party systems....67

Troubleshoot forwarding....73
Troubleshoot forwarder/receiver connection....73

Introduction to forwarding

About forwarding and receiving

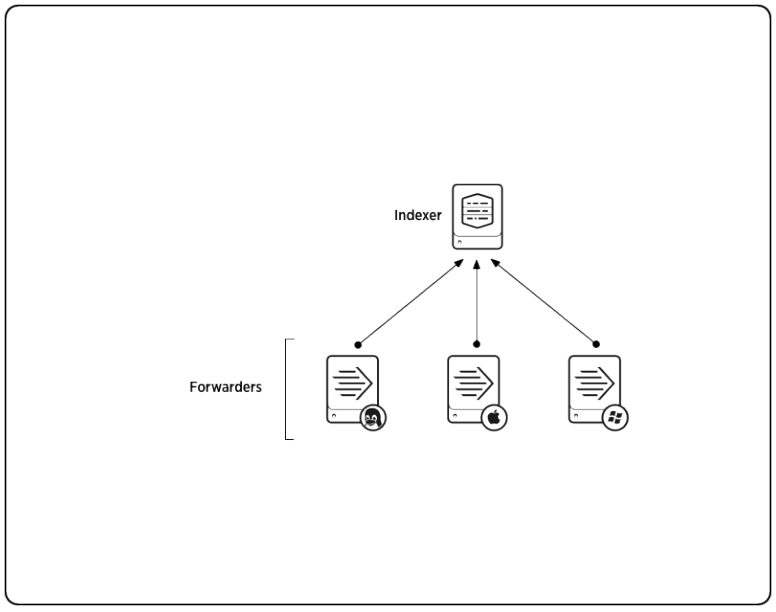
You can forward data from one Splunk Enterprise instance to another Splunk Enterprise instance or even to a non-Splunk system. The Splunk instance that performs the **forwarding** is called a **forwarder**.

There are several types of forwarder. See Types of forwarders to learn about each of them.

A Splunk instance that **receives** data from one or more forwarders is called a **receiver**. The receiver is often a Splunk **indexer**, but can also be another forwarder.

Sample forwarding layout

This diagram shows three forwarders that send data to a single receiver (an indexer), which then indexes the data and makes it available for searching:



Forwarders represent a much more robust solution for data forwarding than raw network feeds, with their capabilities for:

- Tagging of metadata (source, source type, and host)
- Configurable buffering
- Data compression
- SSL security
- Use of any available network ports

The forwarding and receiving capability makes possible all sorts of interesting topologies. You can build environments to handle functions like **data consolidation**, **load balancing**, and **data routing**.

Learn more about forwarding and receiving

- To learn more about the fundamentals of Splunk Enterprise distributed deployment, see the Distributed Deployment Manual.
- For more information on the types of deployment topologies that you can create with forwarders, see Forwarder deployment topologies in this manual.
- To learn about what intermediate forwarding is, see Intermediate forwarding in this manual.
- To learn about the different types of forwarders available, see Types of forwarders.
- To learn about universal forwarders, see the *Universal Forwarder* manual.

Types of forwarders

There are three types of forwarders:

- The **universal forwarder** contains only the components that are necessary to forward data. Learn more about the universal forwarder in the *Universal Forwarder* manual.
- A **heavy forwarder** is a full Splunk Enterprise instance that can index, search, and change data as well as forward it. The heavy forwarder has some features disabled to reduce system resource usage.
- A **light forwarder** is also a full Splunk Enterprise instance, with more features disabled to achieve as small a resource footprint as possible. The light forwarder has been deprecated as of Splunk Enterprise version 6.0. The universal forwarder supersedes the light forwarder for nearly all purposes and represents the best tool for sending data to indexers.

The universal forwarder

The sole purpose of the universal forwarder is to forward data. Unlike a full Splunk instance, you cannot use the universal forwarder to index or search data. To achieve higher performance and a lighter footprint, it has several limitations:

- The universal forwarder cannot search, index, or produce alerts with data.
- The universal forwarder does not **parse** data. You cannot use it to route data to different Splunk indexers based on its contents.
- Unlike full Splunk Enterprise, the universal forwarder does not include a bundled version of Python.

The universal forwarder can get data from a variety of inputs and forward the data to a Splunk deployment for indexing and searching. It can also forward data to another forwarder as an intermediate step before sending the data onward to an indexer.

The universal forwarder is a separately downloadable piece of software. Unlike the heavy and light forwarders, you do not enable it from a full Splunk Enterprise instance. Learn more about the universal forwarder in the *Universal Forwarder* manual.

To learn how to download, install, and deploy a universal forwarder, see Install the universal forwarder software in the *Universal Forwarder* manual.

Heavy and light forwarders

While the universal forwarder is the preferred way to forward data, you might need to use heavy or light forwarders if you need to analyze or make changes to the data before you forward it, or you need to control where the data goes based on its contents. Unlike the universal forwarder, both heavy and light forwarders are full Splunk Enterprise instances with certain features disabled. Heavy and light forwarders differ in capability and the corresponding size of their resource footprints.

A **heavy forwarder** (sometimes referred to as a "regular forwarder") has a smaller footprint than an indexer but retains most of the capability, except that it cannot perform distributed searches. Some of its default functionality, such as Splunk Web, can be disabled, if necessary, to reduce the size of its footprint. A heavy forwarder parses data before forwarding it and can route data based on criteria such as source or type of event.

One key advantage of the heavy forwarder is that it can index data locally, as well as forward data to another Splunk instance. You must activate this feature. See Configure forwarders with outputs.conf in this manual for details.

A **light forwarder** has a smaller footprint with much more limited functionality. It forwards only unparsed data. The universal forwarder, which provides very similar functionality, supersedes it. The light forwarder has been deprecated but continues to be available mainly to meet legacy needs.

When you install a universal forwarder, you can migrate checkpoint settings from any (version 4.0 or greater) light forwarder that resides on the same host. See About the universal forwarder in the *Universal Forwarder* manual for a more detailed comparison of universal and light forwarders.

For detailed information on the capabilities of heavy and light forwarders, see Heavy and light forwarder capabilities in this manual.

Forwarder comparison

This table summarizes the similarities and differences among the three types of forwarders:

Features and capabilities	Universal forwarder	Light forwarder	Heavy forwarder
Type of Splunk Enterprise instance	Dedicated executable	Full Splunk Enterprise, with most features disabled	Full Splunk Enterprise, with some features disabled
Footprint (memory, CPU load)	Smallest	Small	Medium-to-large (depending on enabled features)
Bundles Python?	No	Yes	Yes
Handles data inputs?	All types (but scripted inputs might require Python installation)	All types	All types
Forwards to Splunk Enterprise?	Yes	Yes	Yes
	Yes	Yes	Yes

Forwards to 3rd party systems?			
Serves as intermediate forwarder?	Yes	Yes	Yes
Indexer acknowledgment (guaranteed delivery)?	Optional	Optional (version 4.2 and later)	Optional (version 4.2 and later)
Load balancing?	Yes	Yes	Yes
Data cloning?	Yes	Yes	Yes
Per-event filtering?	No	No	Yes
Event routing?	No	No	Yes
Event parsing?	Sometimes	No	Yes
Local indexing?	No	No	Optional, by setting `indexAndForward` attribute in `outputs.conf`
Searching/alerting?	No	No	Optional
Splunk Web?	No	No	Optional

For detailed information on specific capabilities, see the rest of this topic, as well as the other forwarding topics in the manual.

Types of forwarder data

Forwarders can transmit three types of data:

- Raw
- Unparsed
- Parsed

The type of data a forwarder can send depends on the type of forwarder it is, as well as how you configure it. Universal forwarders and light forwarders can send raw or unparsed data. Heavy forwarders can send raw or parsed data.

With raw data, the forwarder sends the data unaltered over a TCP stream. it does not convert the data into the Splunk communications format. The forwarder

collects the data and sends it on. This is particularly useful for sending data to a non-Splunk system.

With unparsed data, a universal forwarder performs minimal processing. It does not examine the data stream, but it does tag the stream with metadata to identify source, source type, and host. It also divides the data stream into 64-kilobyte blocks and performs some rudimentary timestamping on the stream that the receiving indexer can use in case the events themselves have no discernible timestamps. The universal forwarder does not identify, examine, or tag individual events except when you configure it to parse files with structure data (such as comma-separated value files.)

With parsed data, a heavy forwarder breaks the data into individual events, which it tags and then forwards to a Splunk indexer. It can also examine the events. Because the data has been parsed, the forwarder can perform conditional routing based on event data, such as field values.

The parsed and unparsed formats are both referred to as **cooked** data, to distinguish them from raw data. By default, forwarders send cooked data (universal forwarders send unparsed data and heavy forwarders send parsed data.) To send raw data instead, set the `sendCookedData=false` attribute/value pair in outputs.conf.

Forwarders and indexes

Forwarders forward and route data on an index-by-index basis. By default, they forward all external data, as well as data for the `_audit` internal index. In some cases, they also forward data for the `_internal` internal index. You can change this behavior as necessary. For details, see Filter data by target index.

Plan your deployment

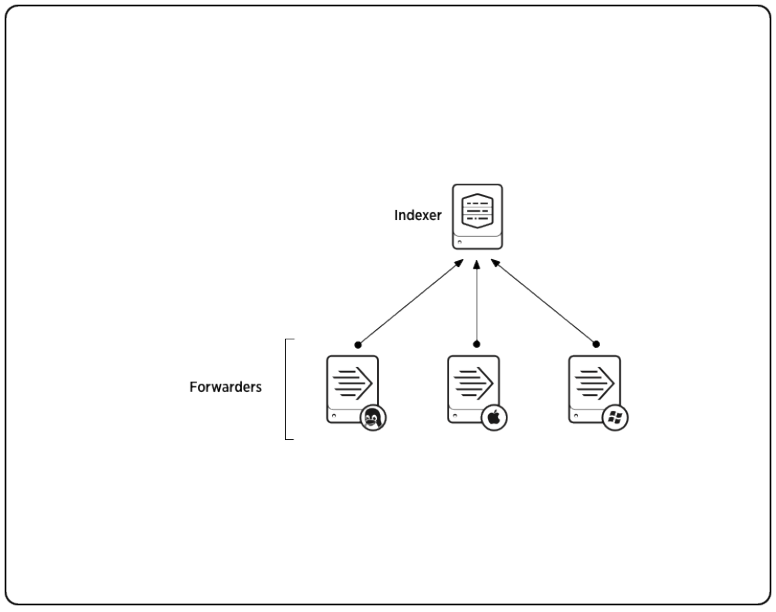
Forwarder deployment topologies

You can deploy forwarders in a wide variety of scenarios. This topic provides an overview of some of the most useful topologies that you can create with forwarders. For detailed information on how to configure various deployment topologies, see Consolidate data from multiple hosts.

Data consolidation

Data consolidation is one of the most common topologies, with multiple forwarders sending data to a single Splunk instance. The scenario typically involves universal forwarders forwarding unparsed data from workstations or production servers to a central Splunk Enterprise instance for consolidation and indexing. In other scenarios, heavy forwarders can send parsed data to a central Splunk indexer.

Here, three universal forwarders are sending data to a single indexer:



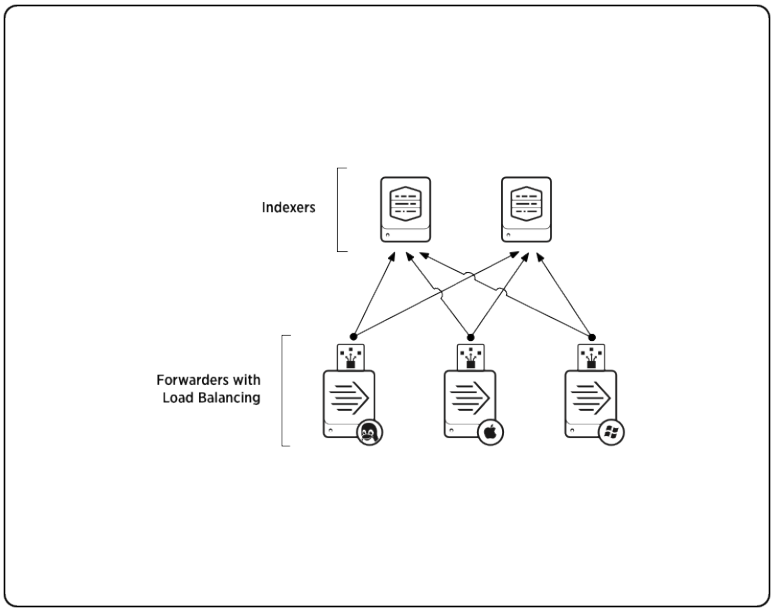
For more information on data consolidation, read Consolidate data from multiple hosts.

Load balancing

Load balancing simplifies the process of distributing data across several indexers to handle considerations such as high data volume, horizontal scaling for enhanced search performance, and fault tolerance. In load balancing, the forwarder routes data sequentially to different indexers at specified intervals.

Forwarders perform automatic load balancing, in which the forwarder switches receivers at set time intervals. If parsing is turned on (for a heavy forwarder), the switching will occur at event boundaries.

In this diagram, three universal forwarders perform load balancing between two indexers:



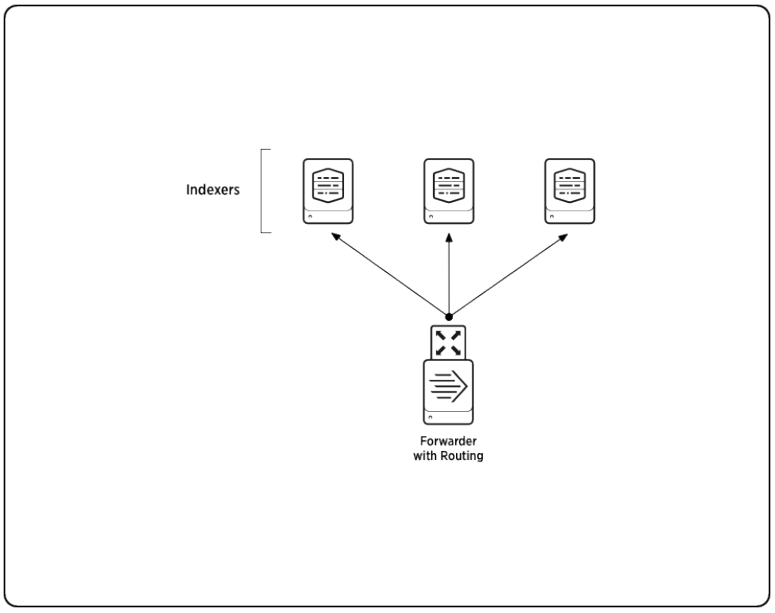
For more information on load balancing, read "Set up load balancing".

Routing and filtering

In **data routing**, a forwarder routes events to specific hosts, based on criteria such as source, source type, or patterns in the events themselves. Routing at the event level requires a heavy forwarder.

A forwarder can also filter and route events to specific queues, or discard them altogether by routing to the null queue.

Here, a heavy forwarder routes data to three indexers based on event patterns:

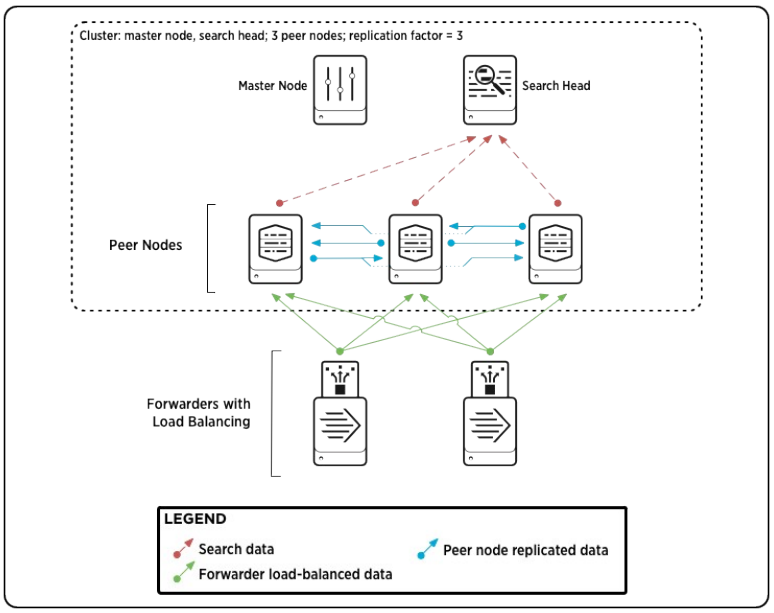


For more information on routing and filtering, see Route and filter data in this manual.

Forwarders and indexer clusters

You can use forwarders to send data to peer nodes in an indexer cluster. A Splunk best practice is to use load-balanced forwarders for that purpose.

This diagram shows two load-balanced forwarders sending data to a cluster:

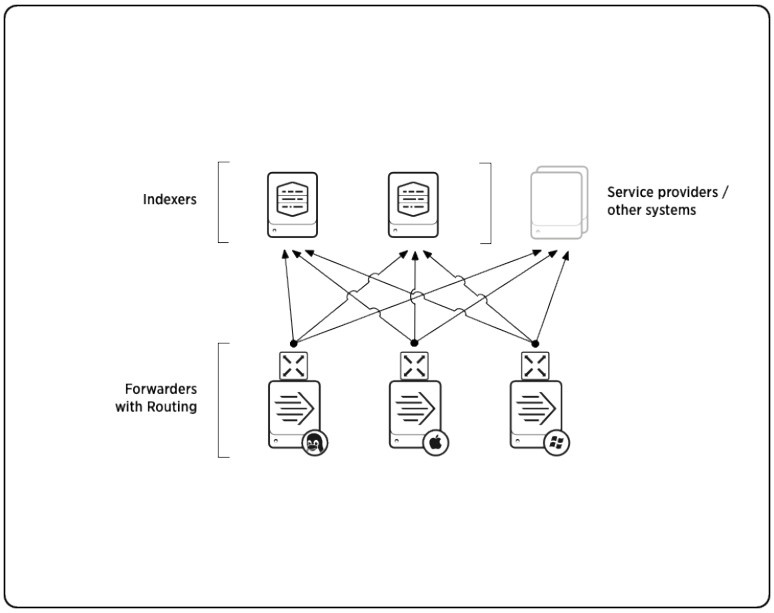


To learn more about forwarders and indexer clusters, see Use forwarders to get your data in *Managing Indexers and Clusters of Indexers*. To learn more about indexer clusters in general, see About indexer clusters and index replication.

Forward to non-Splunk systems

With a heavy forwarder, you can send raw data to a third-party system such as a syslog aggregator. You can combine this with data routing, sending some data to a non-Splunk system and other data to one or more Splunk instances.

In this diagram, three forwarders route data to two Splunk instances and a non-Splunk system:



For more information on forwarding to non-Splunk systems, see Forward data to third-party systems.

Intermediate forwarding

To handle some advanced use cases, you might want to insert an intermediate forwarder between a group of forwarders and the indexer. In this type of scenario, the originating forwarders send data to a consolidating forwarder, which then forwards the data on to an indexer. In some cases, the intermediate forwarders also index the data.

Typical use cases are situations where you need an intermediate index, either for "store-and-forward" requirements or to enable localized searching. (In this case, you would need to use a heavy forwarder.) You can also use an intermediate forwarder if you have some need to limit access to the indexer machine; for instance, for security reasons.

To enable intermediate forwarding, see Configure an intermediate forwarder.

Compatibility between forwarders and indexers

Following are the specific version compatibility restrictions between forwarders and their receiving indexers. For best results, ensure that the version of the indexers is the same or higher than the version of the forwarders from which they

receive data.

Forwarder-indexer compatibilty

- A forwarder that is version 6.0 or later can send data to an indexer that is version 5.0 or later.
- An indexer that is version 6.0 can receive data from a forwarder that is version 4.3 or later.

The following features are available only if both indexers and forwarders are at version 6.x or higher:

- Dynamic file headers.
- Forwarding of structured data.
- Timezone transmission by the forwarder. Additionally, forwarders do not maintain the timezone transmission feature across intermediate forwarding tiers when those tiers consist solely of light or universal forwarders.

Deploy heavy and light forwarders

Enable forwarding on a Splunk Enterprise instance

You can set up heavy and light forwarders on full Splunk Enterprise instances.

Set up forwarding and receiving: heavy or light forwarders

For instructions on how to enable receiving on a Splunk Enterprise instance, see Enable a receiver. If you do not see data on the indexers after you enable forwarding and receiving, see Troubleshoot forwarder/receiver connection.

1. Designate hosts that will act as forwarders and receivers.
2. Install Splunk Enterprise on all of these hosts.
3. On each receiver, use Splunk Web or the CLI to enable receiving.
4. On each forwarder, use Splunk Web or the CLI to enable forwarding. See Deploy a heavy forwarder or Deploy a light forwarder.
5. On each forwarder, use Splunk Web or the CLI, or edit `inputs.conf` to specify data inputs.
6. On each forwarder, use Splunk Web or the CLI, or edit `outputs.conf` to specify where the forwarders should send data.
7. On each forwarder, restart Splunk Enterprise to commit the configuration changes and start forwarding.
8. On the receivers, search for data to confirm that forwarding occurs as you expect. For example:

```
host=<forwarder host name>
```

Set up the universal forwarder

The universal forwarder is a separate product with a separate installation package and documentation. See the *Universal Forwarder* manual for details about the universal forwarder software.

1. Designate hosts that will act as forwarders and receivers.
2. Install Splunk Enterprise on the receiver hosts.
3. On each receiver, use Splunk Web or the CLI to enable receiving.
4. Download the universal forwarder software for the operating system that the forwarder hosts run. For example, if the forwarder hosts run Windows, download the Windows universal forwarder.

5. Install the universal forwarder software on the forwarder hosts. If the hosts run Windows, you can configure parts of the universal forwarder during the installation.
6. After you install the universal forwarder, configure it to send data to a Splunk Enterprise, Splunk Light, or Splunk Cloud indexer.
7. Configure the data inputs that you want to forward.
8. Start the universal forwarder.
9. On the receivers, search for data to confirm that forwarding occurs as you expect. For example:

```
host=<forwarder host name>
```

Find additional information about the universal forwarder in the Universal Forwarder manual

The *Universal Forwarder* manual has detailed information on how to install, configure, and troubleshoot problems with the universal forwarder software. For detailed instructions on how to install the forwarder, see Install the universal forwarder software in the *Universal Forwarder* manual. For more information on how to use the universal forwarder to send data, see one of the following topics in the *Universal Forwarder* manual:

- How to forward data to Splunk Light
- How to forward data to Splunk Cloud
- How to forward data to Splunk Enterprise

Heavy and light forwarder capabilities

This topic describes the capabilities that come with heavy and light forwarders as well as what capabilities are disabled by default.

Heavy forwarder details

The heavy forwarder has all Splunk Enterprise functions and modules enabled by default, with the exception of the distributed search module. The file `$SPLUNK_HOME/etc/apps/SplunkForwarder/default/default-mode.conf` includes this stanza:

```
[pipeline:distributedSearch]
disabled = true
```

For a detailed view of the exact configuration, see the configuration files for the SplunkForwarder application in `$SPLUNK_HOME/etc/apps/SplunkForwarder/default`.

Light forwarder details

The deprecated light forwarder disables most features of Splunk Enterprise. Specifically, the light forwarder:

- Disables event signing and checking whether the disk is full (`$SPLUNK_HOME/etc/apps/SplunkLightForwarder/default/default-mode.conf`).
- Limits internal data inputs to `splunkd` and metrics logs only (`$SPLUNK_HOME/etc/apps/SplunkLightForwarder/default/inputs.conf`).
- Disables all indexing (`$SPLUNK_HOME/etc/apps/SplunkLightForwarder/default/indexes.conf`).
- Does not use `transforms.conf` and does not fully parse incoming data, but the `CHARSET, CHECK_FOR_HEADER, NO_BINARY_CHECK, PREFIX_SOURCETYPE,` and `sourcetype` properties from `props.conf` are used.
- Disables the Splunk Web interface (`$SPLUNK_HOME/etc/apps/SplunkLightForwarder/default/web.conf`).
- Limits throughput to 256KBps (`$SPLUNK_HOME/etc/apps/SplunkLightForwarder/default/limits.conf`).
- Disables the following modules in `$SPLUNK_HOME/etc/apps/SplunkLightForwarder/default/default-mode.conf`:

```
      [pipeline:indexerPipe]
      disabled_processors= indexandforward, diskusage,
signing,tcp-output-generic-processor, syslog-output-generic-processor,
http-output-generic-processor, stream-output-processor

      [pipeline:distributedDeployment]
      disabled = true

      [pipeline:distributedSearch]
      disabled = true

      [pipeline:fifo]
      disabled = true

      [pipeline:merging]
      disabled = true

      [pipeline:typing]
      disabled = true
```

```
[pipeline:udp]
disabled = true

[pipeline:tcp]
disabled = true

[pipeline:syslogfifo]
disabled = true

[pipeline:syslogudp]
disabled = true

[pipeline:parsing]
disabled_processors=utf8, linebreaker, header, sendOut

[pipeline:scheduler]
disabled_processors = LiveSplunks
```

These modules include the deployment server (but not the deployment client), distributed search, named pipes/FIFOs, direct input from network ports, and the scheduler.

The defaults for the light forwarder can be tuned to meet your needs by overriding the settings in `$SPLUNK_HOME/etc/apps/SplunkLightForwarder/default/default-mode.conf` on a case-by-case basis.

Purge old indexes

When you convert an indexer instance to a light forwarder, among other things, you disable indexing. In addition, you lose access to any data that was previously indexed on that instance. However, the data still exists.

If you want to purge that data from your system, you must first disable the SplunkLightForwarder app, then run the CLI `clean` command, and then renable the app. For information on the `clean` command, see Remove indexed data from Splunk in the *Managing Indexers and Clusters of Indexers* manual.

Considerations for forwarding structured data

When you forward structured data (data with source types that use the `INDEXED_EXTRACTIONS` feature) you must perform any parsing, extraction, or filtering changes on the forwarder, not the indexer. See Forward data extracted from header files in the *Getting Data In* manual.

Enable a receiver

To enable forwarding and receiving, you configure both a **receiver** and a **forwarder**. The receiver is the Splunk instance that receives the data; the forwarder sends the data to the receiver.

Depending on your forwarding needs, you might have multiple receivers for each forwarder. Conversely, a single receiver can accept data from many forwarders.

The receiver is either a Splunk **indexer** (the typical case) or another forwarder (referred to as an "intermediate forwarder".)

A Splunk best practice is to set up receivers first, then set up forwarders to send data to those receivers.

Set up receiving

Before you enable a Splunk instance (either an indexer or a forwarder) as a receiver, you must install it. You can then enable receiving on the instance with Splunk Web, the CLI, or the inputs.conf configuration file.

Set up receiving with Splunk Web

Use Splunk Web to set up a receiver:

1. Using Splunk Web, log into the receiver as admin.

2. Click **Settings > Forwarding and receiving**.

3. Click **Add new**.

4. Specify the TCP port you want the receiver to listen on (the **listening port**, also known as the **receiving port**).

For example, if you enter "9997," the receiver listens for connections from forwarders on port 9997. You can specify any unused port. You can use a tool like `netstat` to determine what ports are available on your system. Make sure the port you select is not in use by splunkweb or splunkd.

5. Click **Save.** Splunk software starts to listen for incoming data on the port you specified.

Set up receiving with Splunk CLI

1. From a shell or command prompt, change to the `$SPLUNK_HOME/bin` directory:

```
cd $SPLUNK_HOME/bin
```

2. Run the CLI command to enable receiving:

```
splunk enable listen <port> -auth <username>:<password>
```

For `<port>`, substitute the port you want the receiver to listen on (the receiving port). For example, if you enter "9997," the receiver will receive data on port 9997. You can specify any unused port. You can use a tool like `netstat` to determine what ports are available on your system. Make sure the port you select is not in use by splunkweb or splunkd.

The `splunk enable listen` command creates a `[splunktcp]` stanza in `inputs.conf`. For example, if you set the port to "9997", it creates the stanza `[splunktcp://9997]`.

Set up receiving with the configuration file

You can enable receiving on your Splunk Enterprise instance by configuring `inputs.conf` in `$SPLUNK_HOME/etc/system/local`.

1. With a text editor, open `inputs.conf` in `$SPLUNK_HOME/etc/system/local`.

Note: You might need to create this file if it does not exist.

2. Add a `[splunktcp]` stanza that specifies the receiving port. In this example, the receiving port is 9997:

```
[splunktcp://9997]
disabled = 0
```

Note: The forms `[splunktcp://9997]` and `[splunktcp://:9997]` (one colon or two) are semantically equivalent. Use either one.

Deploy a heavy forwarder

You can enable a heavy forwarder on a full Splunk Enterprise instance

To enable forwarding and receiving, you must configure both a **receiver** and a **forwarder**. The receiver is the Splunk instance that receives the data; the forwarder sends data to the receiver.

A Splunk best practice is to set up the receiver first, as described in Enable a receiver. You can then set up forwarders to send data to that receiver.

Setting up a **heavy** forwarder is a two step process:

1. Install a full Splunk Enterprise instance.

2. Enable forwarding on the instance.

Set up forwarding

You can use Splunk Web or the CLI to enable forwarding for a Splunk instance.

You can also enable, as well as configure, forwarding by creating an `outputs.conf` file on the Splunk instance. Although setting up forwarders with `outputs.conf` requires more initial knowledge, there are advantages to performing all forwarder configurations in a single location. Most advanced configuration options are available only through `outputs.conf`. In addition, if you enable and configure a number of forwarders, you can easily accomplish this by editing a single `outputs.conf` file and making a copy for each forwarder. See Configure forwarders with outputs.conf for more information.

Set up heavy forwarding with Splunk Web

1. Log into Splunk Web as admin on the instance that will be forwarding data.

2. Click the **Settings > Forwarding and receiving**.

3. Click **Add new**.

4. Enter the hostname or IP address for the receiving Splunk instance(s), along with the **receiving port** specified when the receiver was configured. For example, you might enter: `receivingserver.com:9997`. To implement load-balanced forwarding, you can enter multiple hosts as a comma-separated

list.

5. Click **Save.**

Configure heavy forwarders to index and forward data

A heavy forwarder has a key advantage over light and universal forwarders in that it can index your data locally, as well as forward the data to another index. However, local indexing is turned off by default. If you want to store data on the forwarder, you must enable that capability - either in the manner described above or by editing `outputs.conf`.

1. Log into Splunk Web as admin on the instance that will be forwarding data.

2. Click the **Settings > Forwarding and receiving**.

3. Select **Forwarding defaults.**

4. Select **Yes** to store and maintain a local copy of the indexed data on the forwarder.

All other configuration must be done in `outputs.conf`.

Set up heavy forwarding with the CLI

With the CLI, enable forwarding on the Splunk Enterprise instance as follows, then configure forwarding to a specified receiver.

1. From a command or shell prompt, navigate to `$SPLUNK_HOME/bin/`.

2. Type in the following to enable forwarding:

```
splunk enable app SplunkForwarder -auth <username>:<password>
```

3. Restart Splunk Enterprise.

Start forwarding activity from the CLI

The following procedure sends data to the receiving indexer that you specify. Before you can send data to a receiver,

1. From a shell or command prompt, go to the `$SPLUNK_HOME/bin` directory.

2. To start forwarding activity, specify the receiver with the `splunk add forward-server` command:

```
splunk add forward-server <host>:<port> -auth <username>:<password>
```

3. After invoking either of these commands, restart the forwarder.

Stop forwarding activity from the CLI

To end forwarding activity, enter:

```
splunk remove forward-server <host>:<port> -auth <username>:<password>
```

Disable forwarding from the CLI

Even if you stop forwarding activity, the instance remains configured as a forwarder. To revert the forwarder to a full Splunk Enterprise instance, use the `disable` command, as described earlier in this topic.

1. From a command or shell prompt, go to the `$SPLUNK_HOME/bin` directory.

2. Type in the following to disable forwarding:

```
splunk disable app SplunkForwarder -auth <username>:<password>
```

By disabling forwarding, this command reverts the forwarder to a full Splunk Enterprise instance.

Deploy a light forwarder

Important: The light forwarder has been deprecated in Splunk Enterprise version 6.0. For a list of all deprecated features, see Deprecated features in the Release Notes.

You can install a light forwarder on a full Splunk Enterprise instance. For information on how to install a **universal forwarder**, which is the recommended replacement for the light forwarder, see Install the universal forwarder software in the *Universal Forwarder* manual.

To enable forwarding and receiving, configure both a **receiver** and a **forwarder**. The receiver receives the data and the forwarder sends data to the receiver.

A Splunk best practice is to set up the receiver first. You can then set up forwarders to send data to that receiver.

Setting up a **light** forwarder is a two-step process:

1. Install a full Splunk Enterprise instance.
2. Enable forwarding on the instance.

Note: When you configure a Splunk instance as a light forwarder, select the forwarder license. For more information, see Types of Splunk licenses.

Set up forwarding

You can use the CLI as a quick way to enable forwarding.

You can also enable, as well as configure, forwarding by creating an `outputs.conf` file on the Splunk instance. Although setting up forwarders with `outputs.conf` requires a bit more initial knowledge, there are obvious advantages to performing all forwarder configurations in a single location. Most advanced configuration options are available only through `outputs.conf`. In addition, if you will be enabling and configuring a number of forwarders, you can easily accomplish this by editing a single `outputs.conf` file and making a copy for each forwarder. See the topic "Configure forwarders with outputs.conf" for more information.

Set up light forwarding with the CLI

To set up light forwarding, perform the following steps:

1. From a shell or command prompt, navigate to the `$SPLUNK_HOME/bin/` directory and run the following command: `splunk enable app SplunkLightForwarder -auth <username>:<password>`
2. Restart the forwarder.

To disable the light forwarder mode, run the following command:

```
splunk disable app SplunkLightForwarder -auth <username>:<password>
```

This command reverts the forwarder to a full Splunk Enterprise instance.

Start forwarding activity from the CLI

1. From a shell or command prompt, navigate to the `$SPLUNK_HOME/bin/` directory.
2. To start forwarding activity, specify the receiver with the `splunk add forward-server` command: `splunk add forward-server <host>:<port> -auth <username>:<password>`

To end forwarding activity, enter:

```
splunk remove forward-server <host>:<port> -auth <username>:<password>
```

Note: Although this command ends forwarding activity, the instance remains configured as a forwarder. To revert the instance to a full Splunk Enterprise instance, use the `disable` command, as described earlier in this topic.

After invoking either of these commands, restart the forwarder.

Configure forwarders

Configure data collection on forwarders with inputs.conf

You can configure data inputs on a forwarder by editing the `inputs.conf` configuration file.

In nearly all cases, edit `inputs.conf` in the `$SPLUNK_HOME/etc/system/local` directory. If you have an app installed and want to make changes to its input configuration, edit `$SPLUNK_HOME/etc/apps/<appname>/local/inputs.conf`. For example, if you have the Splunk Add-on for Unix and Linux installed, you would make edits in `$SPLUNK_HOME/etc/apps/Splunk_TA_nix/local/inputs.conf`.

Do not make changes to the inputs.conf in `$SPLUNK_HOME/etc/system/default`. When you upgrade, the installation overwrites that file, which removes any changes you made.

See What data can I index? for information on the data that forwarders can collect.

Whenever you make a change to a configuration file, you must restart the forwarder for the change to take effect.

Edit inputs.conf

1. Using your operating system file management tools or a shell or command prompt, navigate to `$SPLUNK_HOME/etc/system/local`.
2. Open `inputs.conf` for editing. You might need to create this file if it does not exist.
3. Add your data inputs.
4. Once you have added your inputs, save the file and close it.
5. Restart the forwarder.
6. On the receiving indexer, log in and load the Search and Reporting app.
7. Run a search and confirm that you see results from the forwarder that you set up the data inputs on:

```
host=<forwarder host name or ip address> source=<data source>
earliest=1h
```

If you don't see any results, visit the Troubleshooting page for possible resolution.

Configure forwarders with outputs.conf

The outputs.conf file defines how forwarders send data to receivers.

While you can specify some output configurations through Splunk Web (heavy/light forwarders only) or the CLI, most advanced configuration settings require that you edit `outputs.conf`. The topics describing various topologies, such as load balancing and data routing, provide detailed examples on configuring `outputs.conf` to support those topologies.

Although `outputs.conf` is a critical file for configuring forwarders, it only addresses where the forwarder should send data. To specify what data the forwarder should collect, you must separately configure the inputs. For details on configuring inputs, see Add data and configure inputs in *Getting Data In*.

Types of outputs.conf files

A single forwarder can have multiple `outputs.conf` files (for instance, one located in an apps directory and another in `/system/local`). No matter how many `outputs.conf` files the forwarder has and where they reside, the forwarder combines all their settings, using the rules of location precedence, as described in Configuration file precedence in the *Admin Manual*.

Default versions

Splunk Enterprise ships with a single default `outputs.conf` file, located in `$SPLUNK_HOME/etc/system/default`.

The universal folder has two default outputs.conf files. See Configure forwarding with outputs.conf in the *Universal Forwarder* manual.

Do not touch default versions of any configuration files, for reasons explained in About configuration files.

Custom versions

When you configure forwarding, changes get saved in custom versions of `outputs.conf`. There are several ways you can specify forwarding behavior:

- While installing the forwarder (on Windows universal forwarder only)
- By running CLI commands
- By using Splunk Web (on heavy and light forwarders only)
- By directly editing an `outputs.conf` file

The forwarder automatically creates or edits custom versions of `outputs.conf` in response to the first three methods. The locations of those versions vary, depending on the type of forwarder and other factors.

When you enable a heavy/light forwarder through Splunk Web or the CLI, an `outputs.conf` file gets created in the directory of the active app on the instance. For example, if you're working in the search app, Splunk Enterprise creates the file in `$SPLUNK_HOME/etc/apps/search/local/`. You can then edit it there.

In addition to any `outputs.conf` files that you create and edit indirectly (for example, through the CLI), you can also create or edit an `outputs.conf` file directly. A Splunk best practice is to work with just a single copy of the file, which you place in `$SPLUNK_HOME/etc/system/local/`. (If a copy of the file already exists in that directory, because of configuration changes made through the CLI, just edit that copy.) For purposes of distribution and management simplicity, you can combine settings from all non-default versions into a single custom `outputs.conf` file.

After you make changes to `outputs.conf`, you must restart the forwarder for the changes to take effect.

For detailed information on `outputs.conf`, see the outputs.conf spec file.

Configuration levels

There are two types of output processors: `tcpout` and `syslog`. You can configure them at three levels of stanzas:

- **Global.** (Optional) At the global level, you specify any attributes that you want to apply globally, as well as certain attributes only configurable at the system-wide level for the output processor.
- **Target group.** A target group defines settings for one or more receiving indexers. There can be multiple target groups per output processor. Most configuration settings can be specified at the target group level.
- **Single server.** (Optional) You can specify configuration values for single hosts (receivers) within a target group. .

Configurations at the more specific level take precedence. For example, if you specify `compressed=true` for a target group, the forwarder will send the hosts in that target group compressed data, even if `compressed` is set to "false" for the global level.

Note: This discussion focuses on the `tcpout` processor, which uses the `[tcpout]` header. For information on the syslog output processor, see Forward data to third-party systems.

Global stanza

Here you set any attributes that you want to apply globally. This stanza is optional. However, there are several attributes that you can set only at the global level, including `defaultGroup` and `indexAndForward`.

The global stanza for the `tcpout` procesor is specified with the `[tcpout]` header.

Here's an example of a global tcpout stanza:

```
[tcpout]
defaultGroup=indexer1
indexAndForward=true
```

This global stanza includes two attribute/value pairs:

- **defaultGroup=indexer1** This tells the forwarder to send all data to the "indexer1" target group. See Default target groups for more information.
- **indexAndForward=true** This tells the forwarder to index the data locally as well as forward it to receiving indexers in the target groups. If set to "false" (the default), the forwarder forwards data but does not index it. This attribute is only available for heavy forwarders; universal and light forwarders cannot index data.

Default target groups

To set default groups for automatic forwarding, include the `defaultGroup` attribute at the global level, in your `[tcpout]` stanza:

```
[tcpout]
defaultGroup= <target_group1>, <target_group2>, ...
```

The defaultGroup specifies one or more target groups that you define later with `tcpout:<target_group>` stanzas. The forwarder sends all events to the specified groups.

If you do not want to forward data automatically, don't set the `defaultGroup` attribute.

For some examples of using the `defaultGroup` attribute, see Route and filter data.

Target group stanza

The target group identifies a set of receivers. It also specifies how the forwarder sends data to those receivers. You can define multiple target groups.

Here's the basic pattern for the target group stanza:

```
[tcpout:<target_group>]
server=<receiving_server1>, <receiving_server2>, ...
<attribute1> = <val1>
<attribute2> = <val2>
...
```

To specify a receiving host in a target group, use the format `<hostname_or_ip_address>:<port>`, where `<port>` is the receiving server's **receiving port**. For example, `myhost.Splunk.com:9997`. You can specify multiple receivers and the forwarder will load balance among them.

See Define typical deployment topologies, later in this topic, for information on how to use the target group stanza to define several deployment topologies.

Single-server stanza

You can define a specific configuration for an individual receiving indexer. However, the receiver must also be a member of a target group.

When you define an attribute at the single-server level, it takes precedence over any definition at the target group or global level.

Here is the syntax for defining a single-server stanza:

```
[tcpout-server://<ipaddress_or_servername>:<port>]
```

```
<attribute1> = <val1>
<attribute2> = <val2>
...
```

Example

The following `outputs.conf` example contains three stanzas for sending tcpout to Splunk Enterprise receivers:

- Global settings. In this example, there is one setting, to specify a `defaultGroup`.
- Settings for a single target group that consists of two receivers. This example specifies a load-balanced target group consisting of two receivers.
- Settings for one receiver within the target group. In this stanza, you can specify any specific settings for the `mysplunk_indexer1` receiver.

```
[tcpout]
defaultGroup=my_indexers

[tcpout:my_indexers]
server=mysplunk_indexer1:9997, mysplunk_indexer2:9996

[tcpout-server://mysplunk_indexer1:9997]
```

Define typical deployment topologies

This section describes how to configure a forwarder to support several typical deployment topologies.

Load balancing

To perform **load balancing**, specify one target group with multiple receivers. In this example, the target group consists of three receivers:

```
[tcpout:my_LB_indexers]
server=10.10.10.1:9997,10.10.10.2:9996,10.10.10.3:9995
```

The forwarder balances load between the specified three receivers. If one receiver goes down, the forwarder automatically switches to the next available receiver.

Data cloning

To perform **data cloning**, specify multiple target groups, each in its own stanza. In data cloning, the forwarder sends copies of all its events to the receivers in two or more target groups. Data cloning usually results in similar, but not necessarily exact, copies of data on the receiving indexers. An example of how you set up data cloning follows:

```
[tcpout]
defaultGroup=indexer1,indexer2

[tcpout:indexer1]
server=10.1.1.197:9997

[tcpout:indexer2]
server=10.1.1.200:9997
```

The forwarder will send duplicate data streams to the servers specified in both the `indexer1` and `indexer2` target groups.

Data cloning with load balancing

You can combine load balancing with data cloning. For example:

```
[tcpout]
defaultGroup=cloned_group1,cloned_group2

[tcpout:cloned_group1]
server=10.10.10.1:9997, 10.10.10.2:9997, 10.10.10.3:9997

[tcpout:cloned_group2]
server=10.1.1.197:9997, 10.1.1.198:9997, 10.1.1.199:9997,
10.1.1.200:9997
```

The forwarder sends full data streams to both `cloned_group1` and `cloned_group2`. The data will be load-balanced within each group, rotating among receivers every 30 seconds (the default frequency).

For syslog and other output types, you must explicitly specify routing. See Route and filter data in this manual.

Commonly used attributes

The `outputs.conf` file provides a large number of configuration options that offer considerable control and flexibility in forwarding. Of the attributes available, several are of particular interest:

Attribute	Default	Where configured	Value
defaultGroup	n/a	global stanza	A comma-separated list of one or more target groups. Forwarder sends all events to all specified target groups. Don't set this attribute if you don't want events automatically forwarded to a target group.
indexAndForward	false	global stanza	If set to "true", the forwarder will index all data locally, in addition to forwarding the data to a receiving indexer. **Note:** This attribute is only available for heavy forwarders. A universal forwarder cannot index locally.
server	n/a	target group stanza	Required. Specifies the server(s) that will function as receivers for the forwarder. This must be set to a value using the format `<hostname_or_ip_address>:<port>`, where `<port>` is the receiving port.
disabled	false	any stanza level	Specifies whether the stanza is disabled. If set to "true", it is equivalent to the stanza not being there.
sendCookedData	true	global or target group stanza	Specifies whether the forwarder processes the data before forwarding.
compressed	false	global or target group stanza	Specifies whether the forwarder sends compressed data.

ssl....	n/a	any stanza level	Set of attributes for configuring SSL. See About securing data from forwarders in *Securing Splunk Enterprise* for information on how to use these attributes.
useACK	false	global or target group stanza	Specifies whether the forwarder waits for indexer acknowledgment confirming that the data has been written to the file system. See Protect against loss of in-flight data.
dnsResolutionInterval	300	global or target group stanza	Specifies the base time interval in seconds at which indexer DNS names will be resolved to IP addresses. See DNS resolution interval in this manual.

The `outputs.conf.spec` file, which you can find here, along with several examples, provides details for these and all other configuration options. In addition, most of these settings are discussed in topics dealing with specific forwarding scenarios.

DNS resolution interval

The `dnsResolutionInterval` attribute specifies the base time interval (in seconds) at which receiver DNS names will be resolved to IP addresses. This value is used to compute the run-time interval as follows:

```
run-time interval = dnsResolutionInterval + (number of receivers in
'server' attribute - 1) * 30
```

The run-time interval extends by 30 seconds for each additional receiver specified in the `server` attribute (for each additional receiver across which the forwarder is load balancing.) The `dnsResolutionInterval` attribute defaults to 300 seconds.

For example, if you leave the attribute at the default setting of 300 seconds and the forwarder load-balancs across 20 indexers, DNS resolution occurs about every 14 minutes:

```
(300 + ((20 - 1) * 30)) = 870 seconds = 14 minutes
```

If you change `dnsResolutionInterval` to 600 seconds, and keep the number of load-balanced indexers at 20, DNS resolution occurs every 19.5 minutes:

```
(600 + ((20 - 1) * 30)) = 1170 seconds = 19.5 minutes
```

Upgrade forwarders

Upgrade heavy and light forwarders

To upgrade a heavy or light forwarder, upgrade the Splunk instance that supports it. There are no specific or additional instructions when you perform an upgrade on a heavy or light forwarder beyond confirming that forwarding continues to work after the upgrade.

See How to upgrade Splunk Enterprise in the *Installation Manual* for installation instructions.

Perform advanced configuration

Set up load balancing

With **load balancing**, a forwarder distributes data across several receiving Splunk instances. Each receiver gets a portion of the total data, and together the receivers hold all the data. To then access the full set of forwarded data, set up distributed searching across all the receivers. For information on distributed search, see About distributed search in the Distributed Search manual.

Load balancing improves performance by letting forwarders send data to several receivers at once. In addition, its automatic switchover capability ensures resiliency in the face of machine outages. If a receiver goes down, the forwarder simply begins sending data to the next available receiver.

Load balancing can also be of use when getting data from network devices like routers. To handle syslog and other data generated across port 514, a single heavy forwarder can monitor port 514 and distribute the incoming data across several indexers.

When you implement load balancing between forwarders and receivers, you must use the functionality that comes with the forwarder. Do not use an external load balancer, as load balancers between forwarders and receivers does not work properly.

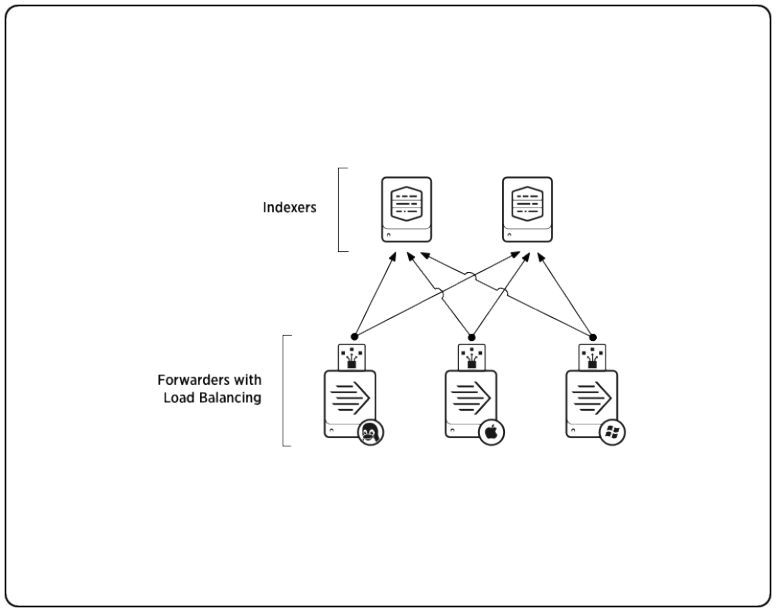
How load balancing works

A forwarder automatically routes data to different indexers based on a specified time interval that you can configure. For example, assume you have a load-balanced group consisting of three indexers: A, B, and C. At some specified interval, such as every 30 seconds, the forwarder switches the data stream to another indexer in the group, selected at random. So, the forwarder might switch from indexer B to indexer A to indexer C, and so on. If one indexer is down, the forwarder immediately switches to another.

To expand on this a bit, there is a data stream for each of the inputs that the forwarder is configured to monitor. The forwarder determines if it is safe for a data stream to switch to another indexer. Then, at the specified interval, it switches the data stream to the newly selected indexer. If it cannot switch the data stream to the new indexer safely, it keeps the connection to the previous indexer open and continues to send the data stream until it has been safely sent.

Universal forwarders have a slight disadvantage in that they can't switch indexers when monitoring TCP network streams of data unless they encounter an End of File (EOF) marker in the stream or an indexer goes down. When this happens, the universal forwarder switches to the next indexer in the list. Because the universal forwarder does not parse the data and identify event boundaries before forwarding (unlike a heavy forwarder), it has no way of knowing when it's safe to switch to the next indexer unless it receives an EOF.

This diagram shows a typical load-balancing scenario, in which three forwarders are sending load-balanced data across a set of two receiving indexers:



Targets for load balancing

When you configure the set of target receivers, you can employ either DNS or static lists.

DNS lists provide greater flexibility and simplified scale-up, particularly for large deployments. Through DNS, you can change the set of receivers without needing to re-edit `outputs.conf` on each forwarder.

The main advantage of a static list is that it allows you to specify a different port for each receiver. This is useful if you need to perform load balancing across multiple receivers running on a single host. Each receiver can listen on a separate port.

Static list target

To use a static list for the target, specify each of the receivers in the target group `[tcpout]` stanza in the forwarder `outputs.conf` file. In the following example, the target group consists of three receivers, specified by IP address and receiver port number:

```
[tcpout: my_LB_indexers]
server=10.10.10.1:9997,10.10.10.2:9996,10.10.10.3:9995
```

The universal forwarder will balance load between the three receivers listed. If one receiver goes down, the forwarder automatically switches to another one on the list.

DNS list target

To use a DNS list, edit `outputs.conf` on the forwarder to specify a single host in the target group `[tcpout]` stanza. For example:

```
[tcpout:my_LB_indexers]
server=splunkreceiver.mycompany.com:9997
```

In your DNS server, create a DNS A record for each host IP address, referencing the server name you specified in `outputs.conf`. For example:

```
splunkreceiver.mycompany.com   A   10.10.10.1
splunkreceiver.mycompany.com   A   10.10.10.2
splunkreceiver.mycompany.com   A   10.10.10.3
```

The forwarder will use the DNS list to balance loads, sending data in intervals, switching among the receivers specified. If a receiver is not available, the forwarder skips it and sends data to another one on the list.

If you have a topology with many forwarders, the DNS list method lets you update the set of receivers by making changes in just a single location, without touching the `outputs.conf` files on the forwarders.

Configure load balancing for horizontal scaling

To configure load balancing, first determine your needs, particularly your horizontal scaling and failover requirements. Then develop a topology based on

those needs, possibly including multiple forwarders, as well as receivers and a search head to search across the receivers.

Assuming a topology of three forwarders and three receivers, set up DNS-based load balancing with these steps:

1. Install and enable a set of three Splunk Enterprise instances as receivers. This example uses a DNS list to designate the receivers, so they must all listen on the same port. For example, if the port is 9997, enable each receiver by going to its `$SPLUNK_HOME/bin/` location and using this CLI command:

```
./splunk enable listen 9997 -auth <username>:<password>
```

2. Install the set of forwarders.

3. Set up a DNS list with an A record for each receiver's IP address:

```
splunkreceiver.mycompany.com   A   10.10.10.1
splunkreceiver.mycompany.com   A   10.10.10.2
splunkreceiver.mycompany.com   A   10.10.10.3
```

4. Create a single `outputs.conf` file for use by all the forwarders. This one specifies the DNS server name used in the DNS list and the port the receivers are listening on:

```
[tcpout]
defaultGroup=my_LB_indexers

[tcpout:my_LB_indexers]
disabled=false
autoLBFrequency=40
server=splunkreceiver.mycompany.com:9997
```

This `outputs.conf` file uses the `autoLBFrequency` attribute to set a load-balance frequency of 40 seconds. Every 40 seconds, the forwarders will switch to another receiver. The default frequency, which rarely needs changing, is 30 seconds.

5. Distribute the `outputs.conf` file to all the forwarders. You can use the **deployment server** to handle the distribution.

The steps are similar if you're using a static list instead of DNS.

Specify load balancing from the CLI

You can also use the CLI to specify load balancing. You do this when you start forwarding activity to a set of receivers, using this syntax:

```
./splunk add forward-server <host>:<port> -method autobalance
```

where `<host>:<port>` is the host and receiver port of the receiver.

This example creates a load-balanced group of four receivers:

```
./splunk add forward-server indexer1:9997 -method autobalance
./splunk add forward-server indexer2:9997 -method autobalance
./splunk add forward-server indexer3:9997 -method autobalance
./splunk add forward-server indexer4:9997 -method autobalance
```

Configure a forwarder to use a SOCKS proxy

This topic discusses how to configure a forwarder with a Socket Secure version 5 (SOCKS5) proxy server as a target with the intent of forwarding data to an indexer beyond the proxy server.

By default, a Splunk forwarder requires a direct network connection to any receiving indexers. If a firewall blocks connectivity between the forwarder and the indexer, the forwarder cannot send data to the indexer.

Starting with version 6.3 of Splunk Enterprise, you can configure a forwarder to use a SOCKS5 proxy host to send data to an indexer. You can do this by specifying attributes in a stanza in the `outputs.conf` configuration file on the forwarder. After you configure and restart the forwarder, it connects to the SOCKS5 proxy host, and optionally authenticates to the server on demand if you provide credentials. The proxy host establishes a connection to the indexer and the forwarder begins sending data through the proxy connection.

Any type of Splunk forwarder can send data through a SOCKS5 proxy host.

This implementation of the SOCKS5 client complies with the Internet Engineering Task Force (IETF) Request for Comments (RFC) Memo #1928. For information on this memo, see "Network Working Group: Request for Comments: 1928" (http://www.ietf.org/rfc/rfc1928.txt) on the IETF website.

To configure a SOCKS5 proxy connection, edit stanzas in `outputs.conf` and specify certain attributes to enable the proxy. For a list of valid proxy attributes, see "Proxy configuration values." You cannot configure proxy servers in Splunk Web.

Configure a SOCKS5 proxy connection with configuration files

1. Make a copy of `$SPLUNK_HOME/etc/system/default/outputs.conf` and place it into $SPLUNK_HOME/etc/system/local</code>.

2. Open `$SPLUNK_HOME/etc/system/local/outputs.conf` for editing.

3. Define forwarding servers or output groups in `outputs.conf` by creating `[tcpout]` or `[tcpout-server]` stanzas. See "Configure forwarders with outputs.conf."

4. In the stanza for connections that should have SOCKS5 proxy support, add attributes for SOCKS that fit your proxy configuration. You must specify at least the `socksServer` attribute to enable proxy support.

5. Save the file and close it.

6. Restart the forwarder.

7. On the receiving indexer, user the Search and Reporting app to confirm that the indexer received the data.

Proxy configuration values

Use the following attributes to configure SOCKS5 on the forwarder:

Attribute	Description	Default
`socksServer`	Tells the forwarder the host name or IP address and port of the SOCKS5 proxy it should connect to for forwarding data. You can specify one of `host:port` or `IP address:port`. You must specify both the host name or the IP address and the port. You must specify this attribute to enable SOCKS5 support.	N/A
`socksUsername`	(Optional) Tells the forwarder to use this username	N/A

	to authenticate to the SOCKS5 proxy host if it demands authentication during the connection phase.	
`socksPassword`	(Optional) Tells the forwarder to provide this password when authenticating into a SOCKS5 proxy host that demands authentication during the connection phase. The forwarder obfuscates this password when it loads the configuration that is associated with the stanza. However, there are some security considerations. See "Security considerations".	N/A
`socksResolveDNS`	(Optional) Tells the forwarder whether or not it should use DNS to resolve the host names of indexers in the output group before passing that information on to the SOCKS5 proxy host. When you set this attribute to `true`, the forwarder sends the name of the indexers to the SOCKS5 proxy host as is, and the SOCKS5 proxy host must then resolve the indexer host names through DNS. Set to `true` if, for example, the forwarder and the proxy server are on different networks served by different DNS servers. When you set it to `false`, the forwarder attempts to resolve the indexer host names through DNS itself, and if it is successful, sends the resolved IP addresses of the indexers to the SOCKS5 proxy host. This attribute only applies if you specify host names for indexers in the `[tcpout]` or `[tcpout-server]` stanzas. If you specify IP addresses, DNS resolution does not happen.	`false`

Examples of SOCKS5 support

Here are some examples of `outputs.conf` stanzas with SOCKS5 proxy support enabled:

This example establishes a connection to a SOCKS5 proxy host that forwards the data to indexers beyond the host:

```
[tcpout]
defaultGroup = proxy_indexers

[tcpout:proxy_indexers]
server = indexer1.slapstick.com:9997, indexer2.slapstick.com:9997
socksServer = prx.slapstick.com:1080
```

This example uses credentials to authenticate into the proxy host before attempting to send data, and tells the proxy host to resolve DNS to determine the indexers to connect for sending data:

```
[tcpout]
defaultGroup = socksCredentials

[tcpout:socksCredentials]
server = indexer3.slapstick.com:9997
socksServer = prx.slapstick.com:1081
socksUsername = proxysrv
socksPassword = letmein
socksResolveDNS = true
```

Security considerations

Note the following caveats when using this feature:

- SOCKS5 proxy support only exists between the forwarder and the indexer inclusive. There is no support for the usage of SOCKS with any other Splunk features, apps, or add-ons.
- The SOCKS5 protocol sends authentication credentials in clear text. Due to this implementation, these credentials are vulnerable to a man-in-the-middle attacker. This means that an attacker can secretly relay and possibly change communication between the SOCKS client and the SOCKS proxy host. This is a caveat of the SOCKS protocol, not the implementation of this feature in Splunk software.
- For the most secure results, use the SOCKS attributes only on forwarders which are inside networks that a SOCKS proxy host protects. Deploying a forwarder in an unprotected environment can result in the interception of SOCKS credentials by a third party, even though the forwarder has SOCKS proxy support enabled.

Configure an intermediate forwarder

This topic provides instructions on how to set up an intermediate forwarder tier.

Intermediate forwarding is where a forwarder receives data from one or more forwarders and then sends that data on to another indexer. This kind of setup is useful when, for example, you have many hosts in different geographical regions and you want to send data from those forwarders to a central host in that region before forwarding the data to an indexer. All forwarder types can act as an immediate forwarder.

For instructions on how to set up intermediate forwarding on a universal forwarder, see Configure an intermediate forwarder in the *Universal Forwarder* manual.

Set up intermediate forwarding with Splunk Web

1. In Splunk Web, log into the Splunk instance that you want to configure as an intermediate forwarder.

2. In the system bar, choose **Settings > Forwarding and receiving**.

3. Under "Receive data", click **Add new**. The "Receive data > Add New" page loads.

4. In the **Listen on this port** field, enter the port number that the instance should listen on for incoming forwarder connections.

5. Click **Save.** The forwarder starts listening on the specified port and Splunk Web displays the "Receive data" page.

6. Under "Receive data", click **Forwarding and receiving**. Splunk Web displays the "Forwarding and receiving" page again.

7. Under "Forward data", on the "Configure forwarding" line, click **Add New**. The "Forward data > Add New" page loads.

8. In the "Host" field, enter the host name or IP address and port of the indexer that should receive the forwarded data.

Note: Do not use the port you specified earlier for this instance unless you configured the same port number on the receiver.

9. Click **Save.** Splunk Web saves the configuration and the forwarder attempts to connect to the specified host and port.

10. Restart the forwarder. From the system bar, click **Settings > Server controls.**

11. Click **Restart Splunk**.

Repeat these instructions on additional hosts to set up a tier of intermediate forwarders.

Set up intermediate forwarding with configuration files

1. Open a command or shell prompt on the host you want to act as an intermediate forwarder.

2. Edit `inputs.conf` to configure the forwarder to receive data, as described in Configure data collection on forwarders with inputs.conf.

3. Configure the forwarder to send data to the receiving indexer, as described in "Configure forwarders with outputs.conf."

4. (Optional) Edit `inputs.conf` on the intermediate forwarder to configure any local data inputs.

5. Restart the forwarder.

Repeat these steps to add more forwarders to the tier.

Configure forwarders to use the intermediate forwarding tier

To set up additional forwarders to send their data to the intermediate forwarding tier:

1. If you have not already, install the universal forwarder.

2. Configure the forwarder to send data to the intermediate forwarder.

3. (Optional) Configure local data inputs on the forwarder.

4. Restart the forwarder.

Test the configuration

To confirm that the intermediate tier works properly:

1. Using Splunk Web, log into the receiving indexer.

2. Open the Search and Reporting app.

3. Run a search that contains a reference to one of the hosts that you configured to send data to the intermediate forwarder. For example:

```
host=<name or ip address of forwarder> index=_internal
```

If you do not see events, then the host has not been configured properly. See Troubleshoot forwarder/receiver connection for possible solutions.

Protect against loss of in-flight data

If you have Splunk Enterprise, you can guard against loss of data when **forwarding** to an **indexer** by enabling the **indexer acknowledgment** capability. With indexer acknowledgment, the **forwarder** will resend any data not acknowledged as "received" by the indexer.

You enable indexer acknowledgment on the forwarder, in `outputs.conf`. The feature is disabled by default.

Note: Both forwarders and indexers must be version 4.2 or above for indexer acknowledgment to function. Otherwise, the transmission will proceed without acknowledgment.

Indexer acknowledgment and indexer clusters

When using forwarders to send data to peer nodes in an indexer cluster, you should ordinarily enable indexer acknowledgment. To learn more about forwarders and clusters, read "Use forwarders to get your data" in the *Managing Indexers and Clusters of Indexers* manual.

How indexer acknowledgment works when everything goes well

The forwarder sends data continuously to the indexer, in blocks of approximately 64kB. The forwarder maintains a copy of each block in memory, in its wait queue, until it gets an acknowledgment from the indexer. While waiting, it continues to send more data blocks.

If all goes well, the indexer:

1. Receives the block of data.

2. Parses the data.

3. Writes the data to the file system as events (raw data and index data).

4. Sends an acknowledgment to the forwarder.

The acknowledgment tells the forwarder that the indexer received the data and successfully wrote it to the file system. Upon receiving the acknowledgment, the forwarder releases the block from memory.

If the wait queue is of sufficient size, it doesn't fill up while waiting for acknowledgments to arrive. But see this section for possible issues and ways to address them, including how to increase the wait queue size.

How indexer acknowledgment works when there's a failure

When there's a failure in the round-trip process, the forwarder does not receive an acknowledgment. It will then attempt to resend the block of data.

Why no acknowledgment?

These are the reasons that a forwarder might not receive acknowledgment:

- Indexer goes down after receiving the data -- for instance, due to machine failure.
- Indexer is unable to write to the file system -- for instance, because the disk is full.
- Network goes down while acknowledgment is en route to the forwarder.

How the forwarder deals with failure

After sending a data block, the forwarder maintains a copy of the data in its wait queue until it receives an acknowledgment. In the meantime, it continues to send additional blocks as usual. If the forwarder doesn't get acknowledgment for a block within 300 seconds (by default), it closes the connection. You can change the wait time by setting the `readTimeout` attribute in `outputs.conf`.

If the forwarder is set up for **auto load balancing**, it then opens a connection to the next indexer in the group (if one is available) and sends the data to it. If the

forwarder is not set up for auto load balancing, it attempts to open a connection to the same indexer as before and resend the data.

The forwarder maintains the data block in the wait queue until acknowledgment is received. Once the wait queue fills up, the forwarder stops sending additional blocks until it receives an acknowledgment for one of the blocks, at which point it can free up space in the queue.

Other reasons the forwarder might close a connection

There are actually three conditions that can cause the forwarder to close the network connection:

- Read timeout. The forwarder doesn't receive acknowledgment within 300 (default) seconds. This is the condition described above.
- Write timeout. The forwarder is not able to finish a network write within 300 (default) seconds. The value is configurable in `outputs.conf` by setting `writeTimeout`.
- Read/write failure. Typical causes include the indexer's machine crashing or the network going down.

In all these cases, the forwarder will then attempt to open a connection to the next indexer in the load-balanced group, or to the same indexer again if load-balancing is not enabled.

The possibility of duplicates

It's possible for the indexer to index the same data block twice. This can happen if there's a network problem that prevents an acknowledgment from reaching the forwarder. For instance, assume the indexer receives a data block, parses it, and writes it to the file system. It then generates the acknowledgment. However, on the round-trip to the forwarder, the network goes down, so the forwarder never receives the acknowledgment. When the network comes back up, the forwarder then resends the data block, which the indexer will parse and write as if it were new data.

To deal with such a possibility, every time the forwarder resends a data block, it writes an event to its `splunkd.log` noting that it's a possible duplicate. The admin is responsible for using the log information to track down the duplicate data on the indexer.

Here's an example of a duplicate warning:

```
10-18-2010 17:32:36.941 WARN  TcpOutputProc - Possible duplication of
events with
channel=source::/home/jkerai/splunk/current-install/etc/apps/sample_app
/logs/maillog.1|host::MrT|sendmail|, streamId=5941229245963076846,
offset=131072
subOffset=219 on host=10.1.42.2:9992
```

Enable indexer acknowledgment

You configure indexer acknowledgment on the forwarder. Set the `useACK` attribute to `true` in `outputs.conf`:

```
[tcpout:<target_group>]
server=<server1>, <server2>, ...
useACK=true
```

By default, `useACK` is set to `false`.

Note: You can set `useACK` either globally or by target group, at the `[tcpout]` or `[tcpout:<target_group>]` stanza levels. You cannot set it for individual receiving indexers at the `[tcpout-server: ...]` stanza level.

For more information, see the outputs.conf spec file.

Indexer acknowledgment and forwarded data throughput

The forwarder uses a wait queue to manage the indexer acknowledgment process. This queue has a default maximum size of 21MB, which is generally sufficient. In rare cases, however, you might need to manually adjust the wait queue size.

If you want more information about the wait queue, read this section. It describes how the wait queue size is configured. It also provides detailed information on how the wait queue functions.

How the wait queue size is configured

You do not set the wait queue size directly. Instead, you set the size of the in-memory output queue, and the wait queue size is automatically set to three times the output queue size. To configure the output queue size, use the `maxQueueSize` attribute in `outputs.conf`.

The default for the `maxQueueSize` attribute is `auto`. Splunk recommends that you keep this setting. It optimizes the queue sizes, based on whether indexer acknowledgment is enabled:

- When `useACK=true`, the output queue size is 7MB and the wait queue size is 21MB.
- When `useACK=false`, the output queue size is 500KB.

You can set `maxQueueSize` to specific values if necessary. See the outputs.conf spec file for further details on `maxQueueSize`.

Note the following points regarding the `maxQueueSize=auto` recommendation:

- When you turn on indexer acknowledgment, the increase in queue size takes effect only after you restart the forwarder.
- The `auto` setting is only available for forwarders of version 5.0.4 and above. For earlier version forwarders running indexer acknowledgment, you need to explicitly set the `maxQueueSize` attribute to 7MB.

Why the wait queue matters

If you enable indexer acknowledgment, the forwarder uses a wait queue to manage the acknowledgment process. Because the forwarder sends data blocks continuously and does not wait for acknowledgment before sending the next block, its wait queue will typically maintain many blocks, each waiting for its acknowledgment. The forwarder will continue to send blocks until its wait queue is full, at which point it will stop forwarding. The forwarder then waits until it receives an acknowledgment, which allows it to release a block from its queue and thus resume forwarding.

A wait queue can fill up when something is wrong with the network or indexer; however, it can also fill up even when the indexer is functioning normally. This is because the indexer only sends the acknowledgment after it has written the data to the file system. Any delay in writing to the file system will slow the pace of acknowledgment, leading to a full wait queue.

There are a few reasons that a normal functioning indexer might delay writing data to the file system (and so delay its sending of acknowledgments):

- The indexer is very busy. For example, at the time the data arrives, the indexer might be dealing with multiple search requests or with data coming from a large number of forwarders.

- The indexer is receiving too *little* data. For efficiency, an indexer only writes to the file system periodically -- either when a write queue fills up or after a timeout of a few seconds. If a write queue is slow to fill up, the indexer will wait until the timeout to write. If data is coming from only a few forwarders, the indexer can end up in the timeout condition, even if each of those forwarders is sending a normal quantity of data. Since write queues exist on a per hot bucket basis, the condition occurs when some particular bucket is getting a small amount of data. Usually this means that a particular index is getting a small amount of data.

To ensure that throughput does not degrade because the forwarder is waiting on the indexer for acknowledgment, you should ordinarily retain the default setting of `maxQueueSize=auto`. In rare cases, you might need to increase the wait queue size, so that the forwarder has sufficient space to maintain all blocks in memory while waiting for acknowledgments to arrive. On the other hand, if you have many forwarders feeding a single indexer and a moderate number of data sources per forwarder, you might be able to conserve a few megabytes of memory by using a smaller size.

When the receiver is a forwarder, not an indexer

You can also use indexer acknowledgment when the data transmission occurs via an intermediate forwarder; that is, where an originating forwarder sends the data to an intermediate forwarder, which then forwards it to the indexer. For this scenario, if you want to use indexer acknowledgment, it is recommended that you enable it along all segments of the data transmission. That way, you can ensure that the data gets delivered along the entire path from originating forwarder to indexer.

Assume you have an originating forwarder that sends data to an intermediate forwarder, which in turn forwards that data to an indexer. To enable indexer acknowledgment along the entire line of transmission, you must enable it twice: first for the segment between originating forwarder and intermediate forwarder, and again for the segment between intermediate forwarder and indexer.

If you enable both segments of the transmission, the intermediate forwarder waits until it receives acknowledgment from the indexer and then it sends acknowledgment back to the originating forwarder.

However, if you enable just one of the segments, you only get indexer acknowledgment over that part of the transmission. For example, say indexer acknowledgment is enabled for the segment from originating forwarder to intermediate forwarder but not for the segment from intermediate forwarder to

indexer. In this case, the intermediate forwarder sends acknowledgment back to the originating forwarder as soon as it sends the data on to the indexer. It then relies on TCP to safely deliver the data to the indexer. Because indexer acknowledgment is not enabled for this second segment, the intermediate forwarder cannot verify delivery of the data to the indexer. This second case has limited value and is not recommended.

Route and filter data

This topic describes how to filter and route event data to Splunk instances. See "Forward data to third-party systems" in this manual for information on routing to non-Splunk systems.

This topic also describes how to perform selective indexing and forwarding, in which you index some data locally on a heavy forwarder and forward the non-indexed data to one or more separate indexers. See "Perform selective indexing and forwarding" later in this topic for details.

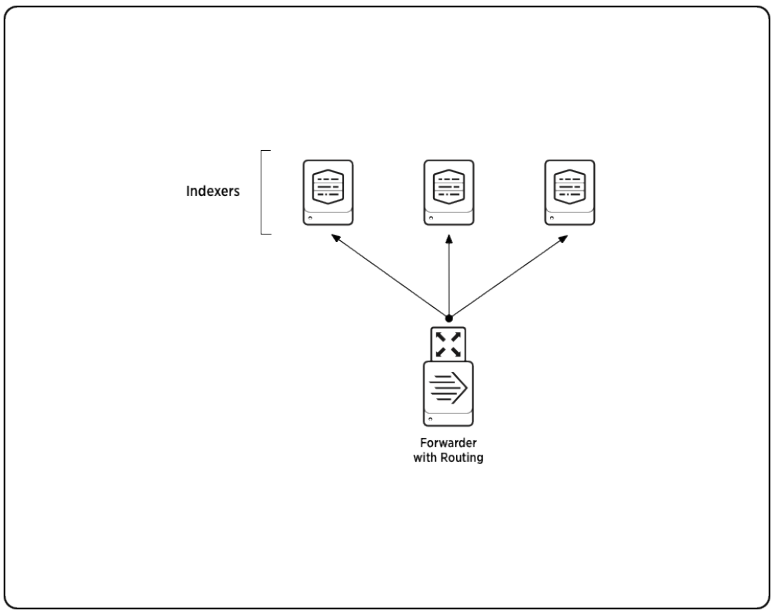
Routing and filtering capabilities of forwarders

Heavy forwarders can filter and route data to specific receivers based on criteria such as source, source type, or patterns in the events themselves. For example, it can send all data from one group of hosts to one indexer and all data from a second group of hosts to a second indexer. Heavy forwarders can also look inside the events and filter or route accordingly. For example, you could use a heavy forwarder to inspect WMI event codes to filter or route Windows events. This topic describes a number of typical routing scenarios.

Besides routing to receivers, forwarders can also filter and route data to specific queues or discard the data altogether by routing to the null queue.

Only heavy forwarders can route or filter all data at the event level. Universal forwarders and light forwarders do not have the ability to inspect individual events except in the case of extracting fields from files with structured data, but they can still forward data based on a host, source, or source type. They can also route based on the data's input stanza, as described below, in the subtopic, "Route inputs to specific indexers based on the data's input". Some input types have the ability to filter out some types of data while acquiring them (see below).

A simple illustration of a forwarder routing data to three indexers follows:



Configure routing

These instructions can only be performed on a heavy forwarder.

1. Determine the criteria to use for routing by answering the following questions:

- How will you identify categories of events?
- Where will you route the events?

2. On the instance that does the routing, open a shell or command prompt.

3. Edit props.conf to add a TRANSFORMS-routing attribute to determine routing based on event metadata:

```
[<spec>]
TRANSFORMS-routing=<transforms_stanza_name>
```

In this `props.con` stanza:

- `<spec>` can be:
  - `<sourcetype>`, the source type of an event
  - `host::<host>`, where <host> is the host for an event
  - `source::<source>`, where <source> is the source for an event
- If you have multiple TRANSFORMS attributes, use a unique name for each. For example: "TRANSFORMS-routing1", "TRANSFORMS-routing2",

and so on.
- <transforms_stanza_name> must be unique.

Use the <transforms_stanza_name> specified here when creating an entry in transforms.conf (below).

Examples later in this topic show how to use this syntax.

3. Edit transforms.conf to specify target groups and to set additional criteria for routing based on event patterns:

```
[<transforms_stanza_name>]
REGEX=<routing_criteria>
DEST_KEY=_TCP_ROUTING
FORMAT=<target_group>,<target_group>,....
```

In this transforms.con stanza:

- <transforms_stanza_name> must match the name you defined in props.conf.
- In <routing_criteria>, enter the regular expression rules that determine which events get routed. This line is required. Use "REGEX = ." if you don't need additional filtering beyond the metadata specified in props.conf.
- DEST_KEY should be set to _TCP_ROUTING to send events via TCP. It can also be set to _SYSLOG_ROUTING or _HTTPOUT_ROUTING for other output processors.
- Set FORMAT to a <target_group> that matches a target group name you defined in outputs.conf. If you specify more than one target group, use commas to separate them. A comma separated list clones events to multiple target groups.

Examples later in this topic show how to use this syntax.

4. Edit outputs.conf to define the target group(s) for the routed data:

```
[tcpout:<target_group>]
server=<ip>:<port>
```

In this outputs.conf stanza:

- Set <target_group> to match the name you specified in transforms.conf.
- Set the IP address and port to match the receiving server.

The use cases described in this topic generally follow this pattern.

Filter and route event data to target groups

In this example, a heavy forwarder filters three types of events and routes them to different target groups. The forwarder filters and routes according to these criteria:

- Events with a source type of "syslog" to a load-balanced target group
- Events containing the word "error" to a second target group
- All other events to a default target group

1. On the instance that is to do the routing, open a command or shell prompt.

2. Edit `props.conf` in `$SPLUNK_HOME/etc/system/local` to set two TRANSFORMS-routing attributes — one for syslog data and a default for all other data:

```
[default]
TRANSFORMS-routing=errorRouting

[syslog]
TRANSFORMS-routing=syslogRouting
```

3. Edit `transforms.conf` to set the routing rules for each routing transform:

```
[errorRouting]
REGEX=error
DEST_KEY=_TCP_ROUTING
FORMAT=errorGroup

[syslogRouting]
REGEX=.
DEST_KEY=_TCP_ROUTING
FORMAT=syslogGroup
```

In this example, if a syslog event contains the word "error", it routes to `syslogGroup`, not `errorGroup`. This is due to the settings you previously specified in `props.conf`. Those settings dictated that all syslog events should be filtered through the `syslogRouting` transform, while all non-syslog (default) events should be filtered through the `errorRouting` transform. Therefore, only non-syslog events get inspected for errors.

4. Edit `outputs.conf` to define the target groups:

```
[tcpout]
defaultGroup=everythingElseGroup

[tcpout:syslogGroup]
server=10.1.1.197:9996, 10.1.1.198:9997

[tcpout:errorGroup]
server=10.1.1.200:9999

[tcpout:everythingElseGroup]
server=10.1.1.250:6666
```

`syslogGroup` and `errorGroup` receive events according to the rules specified in `transforms.conf`. All other events get routed to the default group, `everythingElseGroup`.

Replicate a subset of data to a third-party system

This example uses data filtering to route two data streams. It forwards:

- All the data, in cooked form, to a Splunk Enterprise indexer (10.1.12.1:9997)
- A replicated subset of the data, in raw form, to a third-party host (10.1.12.2:1234)

The example sends both streams as TCP. To send the second stream as syslog data, first route the data through an indexer.

1. Edit `props.conf`:

```
[syslog]
TRANSFORMS-routing = routeAll, routeSubset
```

2. Edit `transforms.conf`:

```
[routeAll]
REGEX=(.)
DEST_KEY=_TCP_ROUTING
FORMAT=Everything

[routeSubset]
REGEX=(SYSTEM|CONFIG|THREAT)
```

```
DEST_KEY=_TCP_ROUTING
FORMAT=Subsidiary,Everything
```

3. Edit `outputs.conf`:

```
[tcpout]
defaultGroup=nothing

[tcpout:Everything]
disabled=false
server=10.1.12.1:9997

[tcpout:Subsidiary]
disabled=false
sendCookedData=false
server=10.1.12.2:1234
```

Filter event data and send to queues

Although similar to forwarder-based routing, queue routing can be performed by an indexer, as well as a heavy forwarder. It does not use the `outputs.conf` file, only `props.conf` and `transforms.conf`.

You can eliminate unwanted data by routing it to `nullQueue`, the Splunk equivalent of the Unix `/dev/null` device. When you filter out data in this way, the data is not forwarded and doesn't count toward your indexing volume.

See "Caveats for routing and filtering structured data" later in this topic.

Discard specific events and keep the rest

This example discards all `sshd` events in `/var/log/messages` by sending them to `nullQueue`:

1. In `props.conf`, set the TRANSFORMS-null attribute:

```
[source::/var/log/messages]
TRANSFORMS-null= setnull
```

2. Create a corresponding stanza in `transforms.conf`. Set `DEST_KEY` to "queue" and `FORMAT` to "nullQueue":

```
[setnull]
```

```
REGEX = \[sshd\]
DEST_KEY = queue
FORMAT = nullQueue
```

3. Restart Splunk Enterprise.

Keep specific events and discard the rest

This scenario is opposite to the previous scenario. In this example, you use two transforms to keep *only* the `sshd` events. One transform routes `sshd` events to `indexQueue`, while another routes all other events to `nullQueue`.

Note: In this example, the order of the transforms in `props.conf` matters. The null queue transform must come first; if it comes later, it will invalidate the previous transform and route all events to the null queue.

1. In `props.conf`:

```
[source::/var/log/messages]
TRANSFORMS-set= setnull,setparsing
```

2. In `transforms.conf`:

```
[setnull]
REGEX = .
DEST_KEY = queue
FORMAT = nullQueue

[setparsing]
REGEX = \[sshd\]
DEST_KEY = queue
FORMAT = indexQueue
```

3. Restart Splunk Enterprise.

Filter WMI and Event Log events

You can filter WinEventLog directly at the forwarder level.

Otherwise to filter on WMI events, use [WMI:WinEventLog:Security] source type stanza in `props.conf`. The following example uses regular expressions to filter out two Windows event codes, 592 and 593:

1. Edit `props.conf`:

```
[WinEventLog:Security]
TRANSFORMS-wmi=wminull
```

2. Edit `transforms.conf`:

```
[wminull]
REGEX=(?m)^EventCode=(592|593)
DEST_KEY=queue
FORMAT=nullQueue
```

3. Restart Splunk Enterprise.

Filter data by target index

Forwarders have a **forwardedindex** filter that lets you specify whether data gets forwarded, based on the data's target index. For example, if you have one data input targeted to "index1" and another targeted to "index2", you can use the filter to forward only the data targeted to index1, while ignoring the index2 data. The forwardedindex filter uses **whitelists** and **blacklists** to specify the filtering. For information on setting up multiple indexes, see the topic Create custom indexes in the *Managing Indexers and Clusters of Indexers* manual.

Note: The forwardedindex filter is only applicable under the global `[tcpout]` stanza. This filter does not work if you create it under any other stanza in `outputs.conf`.

Use the `forwardedindex.<n>.whitelist|blacklist` attributes in `outputs.conf` to specify which data should get forwarded on an index-by-index basis. You set the attributes to regexes that filter the target indexes.

Default behavior

By default, the forwarder forwards data targeted for all external indexes, including the default index and any user-created indexes. Regarding data for internal indexes, the default behavior varies according to who is doing the forwarding:

- The **universal forwarder** forwards the data for the `_audit` internal index only. It does not forward data for other internal indexes. Its default `outputs.conf` file located in `$SPLUNK_HOME/etc/apps/SplunkUniversalForwarder/default` specifies that behavior with these attributes:

```
[tcpout]
forwardedindex.0.whitelist = .*
forwardedindex.1.blacklist = _.*
forwardedindex.2.whitelist = _audit
```

- **Heavy forwarder and full Splunk instances with forwarding enabled** (for example, a search head with forwarding enabled) forward the data for the `_audit` and `_internal` internal indexes. Their default `outputs.conf` file, located in `$SPLUNK_HOME/etc/system/default`, specifies that behavior with these attributes:

```
[tcpout]
forwardedindex.0.whitelist = .*
forwardedindex.1.blacklist = _.*
forwardedindex.2.whitelist = (_audit|_internal)
```

In most deployments, you do not need to override the default settings. See outputs.conf for more information on how to whitelist and blacklist indexes. For more information on default and custom `outputs.conf` files and their locations, see "Types of outputs.conf files".

Forward all external and internal index data

If you want to forward all internal index data, as well as all external data, you can override the default forwardedindex filter attributes:

```
#Forward everything
[tcpout]
forwardedindex.0.whitelist = .*
# disable these
forwardedindex.1.blacklist =
forwardedindex.2.whitelist =
```

Forward data for a single index only

If you want to forward only the data targeted for a single index (for example, as specified in `inputs.conf`), and drop any data that is not a target for that index, configure `outputs.conf` in this way:

```
[tcpout]
#Disable the current filters from the defaults outputs.conf
forwardedindex.0.whitelist =
forwardedindex.1.blacklist =
```

```
forwardedindex.2.whitelist =

#Forward data for the "myindex" index
forwardedindex.0.whitelist = myindex
```

This first disables all filters from the default `outputs.conf` file. It then sets the filter for your own index. Be sure to start the filter numbering with 0: `forwardedindex.0`.

Note: If you've set other filters in another copy of `outputs.conf` on your system, you must disable those as well.

You can use the CLI `btools` command to ensure that there aren't any other filters located in other `outputs.conf` files on your system:

```
splunk cmd btool outputs list tcpout
```

This command returns the content of the `tcpout` stanza, after all versions of the configuration file have been combined.

Use the forwardedindex attributes with local indexing

On a heavy forwarder, you can index locally. To do that, you must set the `indexAndForward` attribute to "true". Otherwise, the forwarder just forwards your data and does not save it on the forwarder. On the other hand, the `forwardedindex` attributes only filter forwarded data; they do not filter any data that gets saved to the local index.

In a nutshell, local indexing and forwarder filtering are entirely separate operations, which do not coordinate with each other. This can have unexpected implications when you performing blacklist filtering:

- If you set `indexAndForward` to "true" and then filter out some data through `forwardedindex` blacklist attributes, the forwarder does not forward the blacklisted data, but it does still index the data.
- If you set `indexAndForward` to "false" (no local indexing) and then filter out some data, the forwarder drops the filtered data entirely (because it neither forwards nor indexes the filtered data.)

Route inputs to specific indexers based on the data's input

In this scenario, you use `inputs.conf` and `outputs.conf` to route data to specific indexers, based on the data's input. Universal and light forwarders can perform

this kind of routing.

Here's an example that shows how this works.

1. In `outputs.conf`, you create stanzas for each receiving indexer:

```
[tcpout:systemGroup]
server=server1:9997

[tcpout:applicationGroup]
server=server2:9997
```

2. In `inputs.conf`, you use `_TCP_ROUTING` to specify the stanza in `outputs.conf` that each input should use for routing:

```
[monitor://.../file1.log]
_TCP_ROUTING = systemGroup

[monitor://.../file2.log]
_TCP_ROUTING = applicationGroup
```

The forwarder routes data from `file1.log` to `server1` and data from `file2.log` to `server2`.

Perform selective indexing and forwarding

You can index and store data locally, as well as forward the data onwards to a receiving indexer. There are two ways to do this:

- **Index all the data before forwarding it.** To do this, enable the `indexAndForward` attribute in `outputs.conf`.
- **Index a subset of the data before forwarding it or other data.** This is called **selective indexing**. With selective indexing, you can index some of the data locally and then forward it on to a receiving indexer. Alternatively, you can choose to forward only the data that you don't index locally.

Do not enable the `indexAndForward` attribute in the `[tcpout]` stanza if you also plan to enable selective indexing.

Configure selective indexing

To use selective indexing, modify both your `inputs.conf` and `outputs.conf` files:

1. In `outputs.conf`:

a. Add the `[indexAndForward]` stanza:

```
[indexAndForward]
index=true
selectiveIndexing=true
```

The presence of this stanza, including the `index` and `selectiveIndexing` attributes, turns on selective indexing for the forwarder. It enables local indexing for any input (specified in `inputs.conf`) that has the `_INDEX_AND_FORWARD_ROUTING` attribute. Use the entire `[indexAndForward]` stanza exactly as shown here.

Note: This is a global stanza, which only needs to appear once in `outputs.conf`.

b. Include the usual target group stanzas for each set of receiving indexers:

```
[tcpout:<target_group>]
server = <ip address>:<port>, <ip address>:<port>, ...
...
```

The named `<target_group>` is used in `inputs.conf` to route the inputs, as described below.

2. In `inputs.conf`:

a. Add the `_INDEX_AND_FORWARD_ROUTING` attribute to the stanzas of each input that you want to index locally:

```
[input_stanza]
_INDEX_AND_FORWARD_ROUTING=<any_string>
...
```

The presence of the `_INDEX_AND_FORWARD_ROUTING` attribute tells the heavy forwarder to index that input locally. You can set the attribute to any string value you want. The forwarder just looks for the attribute itself; the string value has no effect at all on behavior.

b. Add the `_TCP_ROUTING` attribute to the stanzas of each input that you want to forward:

```
[input_stanza]
_TCP_ROUTING=<target_group>
...
```

The `<target_group>` is the name used in `outputs.conf` to specify the target group of receiving indexers.

The next several sections show how to use selective indexing in a variety of scenarios.

Index one input locally and then forward the remaining inputs

In this example, the forwarder indexes data from one input locally but does not forward it. It also forwards data from two other inputs but does not index those inputs locally.

1. In `outputs.conf`, create these stanzas:

```
[tcpout]
defaultGroup=noforward
disabled=false

[indexAndForward]
index=true
selectiveIndexing=true

[tcpout:indexerB_9997]
server = indexerB:9997

[tcpout:indexerC_9997]
server = indexerC:9997
```

Since the `defaultGroup` is set to the non-existent group "noforward" (meaning that there is no `defaultGroup`), the forwarder only forwards data that has been routed to explicit target groups in `inputs.conf`. It drops all other data.

2. In `inputs.conf`, create these stanzas:

```
[monitor:///mydata/source1.log]
_INDEX_AND_FORWARD_ROUTING=local
```

```
[monitor:///mydata/source2.log]
_TCP_ROUTING=indexerB_9997

[monitor:///mydata/source3.log]
_TCP_ROUTING=indexerC_9997
```

The result is that the forwarder:

- indexes the `source1.log` data locally but does not forward it (because there is no explicit routing in its input stanza and there is no default group in `outputs.conf`).
- forwards the `source2.log` data to indexerB but does not index it locally.
- forwards the `source3.log` data to indexerC but does not index it locally.

Index one input locally and then forward all inputs

This example is nearly identical to the previous one. The difference is that here, you index just one input locally, but then you forward all inputs, including the one you've indexed locally.

1. In `outputs.conf`, create these stanzas:

```
[tcpout]
defaultGroup=noforward
disabled=false

[indexAndForward]
index=true
selectiveIndexing=true

[tcpout:indexerB_9997]
server = indexerB:9997

[tcpout:indexerC_9997]
server = indexerC:9997
```

This `outputs.conf` is identical to the previous example.

2. In `inputs.conf`, create these stanzas:

```
[monitor:///mydata/source1.log]
_INDEX_AND_FORWARD_ROUTING=local
_TCP_ROUTING=indexerB_9997

[monitor:///mydata/source2.log]
```

```
_TCP_ROUTING=indexerB_9997

[monitor:///mydata/source3.log]
_TCP_ROUTING=indexerC_9997
```

The only difference from the previous example is that here, you have specified the `_TCP_ROUTING` attribute for the input that you are indexing locally. The forwarder routes both `source1.log` and `source2.log` to the `indexerB_9997` target group, but it only locally indexes the data from `source1.log`.

Another way to index one input locally and then forward all inputs

You can achieve the same result as in the previous example by setting the `defaultGroup` to a real target group.

1. In `outputs.conf`, create these stanzas:

```
[tcpout]
defaultGroup=indexerB_9997
disabled=false

[indexAndForward]
index=true
selectiveIndexing=true

[tcpout:indexerB_9997]
server = indexerB:9997

[tcpout:indexerC_9997]
server = indexerC:9997
```

This `outputs.conf` sets the `defaultGroup` to `indexerB_9997`.

2. In `inputs.conf`, create these stanzas:

```
[monitor:///mydata/source1.log]
_INDEX_AND_FORWARD_ROUTING=local

[monitor:///mydata/source2.log]
_TCP_ROUTING=indexerB_9997

[monitor:///mydata/source3.log]
_TCP_ROUTING=indexerC_9997
```

Even though you haven't set up an explicit routing for `source1.log`, the forwarder

still forwards it to the `indexerB_9997` target group, since `outputs.conf` specifies that group as the `defaultGroup`.

Selective indexing and internal logs

Once you enable selective indexing in `outputs.conf`, the forwarder only indexes locally those inputs with the `_INDEX_AND_FORWARD_ROUTING` attribute. This applies to the internal logs in the `/var/log/splunk` directory (specified in the default `etc/system/default/inputs.conf`). By default, the forwarder does not index those logs. If you want to index them, you must add their input stanza to your local `inputs.conf` file (which takes precedence over the default file) and include the `_INDEX_AND_FORWARD_ROUTING` attribute:

```
[monitor://$SPLUNK_HOME/var/log/splunk]
index = _internal
_INDEX_AND_FORWARD_ROUTING=local
```

Caveats for routing and filtering structured data

Splunk software does not parse structured data that has been forwarded to an indexer

When you forward structured data to an indexer, Splunk software does not parse this data once it arrives at the indexer, even if you have configured `props.conf` on that indexer with `INDEXED_EXTRACTIONS` and its associated attributes. Forwarded data skips the following queues on the indexer, which precludes any parsing of that data on the indexer:

- `parsing`
- `aggregation`
- `typing`

The forwarded data must arrive at the indexer already parsed. To achieve this, you must also set up `props.conf` on the forwarder that sends the data. This includes configuration of `INDEXED_EXTRACTIONS` and any other parsing, filtering, anonymizing, and routing rules. Universal forwarders are capable of performing these tasks solely for structured data. See "Forward data extracted from header files".

Forward data to third-party systems

Splunk forwarders can forward raw data to non-Splunk systems over a plain TCP socket or packaged in standard syslog. Because they are forwarding to a non-Splunk system, they can send only raw data.

By editing `outputs.conf`, `props.conf`, and `transforms.conf`, you can configure a heavy forwarder to route data conditionally to third-party systems, in the same way that it routes data conditionally to other Splunk instances. You can filter the data by host, source, or source type. You can also use regular expressions to further qualify the data.

Data forwarding to third-party systems is one of several search result export methods that Splunk software offers. For information about the other export methods available to you, see Export search results in the *Search Manual*.

TCP data

You can use any kind of forwarder, such as a universal forwarder, to forward TCP data to a third-party system:

1. Configure the third party receiving host to expect incoming data on a TCP port.

2. Edit outputs.conf to specify the receiving host and port.

To route the data, you must use a heavy forwarder, which has the ability to parse data.

3. Edit props.conf to determine what data to route.

4. Edit transforms.conf to determine where to route the data based on what you configured in `props.conf`.

Edit configuration files

To forward data, edit `outputs.conf`:

- Specify target groups for the receiving servers.
- Specify the IP address and TCP port for each receiving server.
- Set `sendCookedData` to `false`, so that the forwarder sends raw data.

To route and filter the data on heavy forwarders only, also edit `props.conf` and `transforms.conf`:

- In `props.conf`, specify the host, source, or sourcetype of your data stream. Specify a transform to perform on the input.
- In `transforms.conf`, define the transform and specify `_TCP_ROUTING`. You can also use regular expressions to further filter the data.

Forward all data

This example shows how to send all the data from a forwarder to a third-party system. Since you are sending all the data, you only need to edit `outputs.conf`:

```
[tcpout]

[tcpout:fastlane]
server = 10.1.1.35:6996
sendCookedData = false
```

Forward a subset of data

This example shows how to use a heavy forwarder to filter a subset of data and send the subset to a third-party system. Light and universal forwarders cannot route or filter data.

1. Edit `props.conf` and `transforms.conf` to specify the filtering criteria.

In `props.conf`, apply the `bigmoney` transform to all host names beginning with `nyc`:

```
[host::nyc*]
TRANSFORMS-nyc = bigmoney
```

In `transforms.conf`, configure the `bigmoney` transform to specify `TCP_ROUTING` as the `DEST_KEY` and the `bigmoneyreader` target group as the `FORMAT`:

```
[bigmoney]
REGEX = .
DEST_KEY=_TCP_ROUTING
FORMAT=bigmoneyreader
```

2. In `outputs.conf`, define both a `bigmoneyreader` target group for the non-Splunk server and a default target group to receive any other data:

```
[tcpout]
defaultGroup = default-clone-group-192_168_1_104_9997

[tcpout:default-clone-group-192_168_1_104_9997]
server = 192.168.1.104:9997

[tcpout:bigmoneyreader]
server=10.1.1.197:7999
sendCookedData=false
```

The forwarder will send all data from host names beginning with `nyc` to the non-Splunk server specified in the `bigmoneyreader` target group. It will send data from all other hosts to the server specified in the `default-clone-group-192_168_1_104_9997` target group.

Note: If you want to forward only the data specifically identified in `props.conf` and `transforms.conf`, set `defaultGroup=nothing`.

Syslog data

You can configure a heavy forwarder to send data in standard syslog format. The forwarder sends the data through a separate output processor. The syslog output processor is not available for universal or light forwarders.

The syslog output processor sends RFC 3164-compliant events to a TCP/UDP-based server and port, making the payload of any non-compliant data RFC 3164-compliant.

By default, Splunk software does not change the content of an event to make its character set compliant with the third-party server. You can specify a `SEDCMD` configuration in `props.conf` to address data that contains characters that the third-party server cannot process. This option is useful for removing newline characters from Windows Event Log events. See Anonymize data through a sed script in *Getting Data In*.

You can also filter the data with `props.conf` and `transforms.conf`. When you do so, you need to specify _SYSLOG_ROUTING as the DEST_KEY.

Forward syslog data to a third-party host

1. Identify the third-party receiving host.

2. On the forwarder that is to send data to the third-party host, open `$SPLUNK_HOME/etc/system/local/outputs.conf` for editing.

3. In the `outputs.conf` file, add a stanza that specifies the receiving host in a `syslog` target group.

```
[syslog]
defaultGroup=syslogGroup

[syslog:syslogGroup]
server = 10.1.1.197:514
```

If you define multiple event types for syslog data, you must include the string "syslog" in all of the event type names.

Forward syslog data

In `outputs.conf`, specify the `syslog` target group:

```
[syslog:<target_group>]
<attribute1> = <val1>
<attribute2> = <val2>
...
```

The target group stanza requires this attribute:

Required Attribute	Default	Value
server	n/a	This must be in the format `<hostname_or_ipaddress>:<port>`. This is a combination of the IP address or servername of the syslog server and the port on which the syslog server is listening. Note that syslog servers use port 514 by default.

These attributes are optional:

Optional Attribute	Default	Value

type	udp	The transport protocol. Must be set to "tcp" or "udp".
priority	<13> - this signifies a facility of 1 ("user") and a severity of 5 ("notice")	Syslog priority. This must be an integer 1 to 3 digits in length, surrounded by angle brackets; for example: <34>. This value will appear in the syslog header. Mimics the number passed via syslog interface call; see outputs.conf for more information. Compute the priority value as (<facility> * 8) + <severity>. If facility is 4 (security/authorization messages) and severity is 2 (critical conditions), priority value will be: (4 * 8) + 2 = 34, which you specify in the conf file as <34>.
syslogSourceType	n/a	This must be in the format `sourcetype::syslog`, the source type for syslog messages.
timestampformat	""	The format used when adding a timestamp to the header. This must be in the format: <%b %e %H:%M:%S>. See "Configure timestamps" in the Getting Data In manual for details.

Send a subset of data to a syslog server

This example shows how to configure a heavy forwarder to forward data from hosts whose names begin with "nyc" to a syslog server named "loghost.example.com" over port 514:

1. Edit `props.conf` and `transforms.conf` to specify the filtering criteria.

In `props.conf`, apply the `send_to_syslog` transform to all host names beginning with `nyc`:

```
[host::nyc*]
TRANSFORMS-nyc = send_to_syslog
```

In `transforms.conf`, configure the `send_to_syslog` transform to specify `_SYSLOG_ROUTING` as the `DEST_KEY` and the `my_syslog_group` target group as the `FORMAT`:

```
[send_to_syslog]
REGEX = .
DEST_KEY = _SYSLOG_ROUTING
FORMAT = my_syslog_group
```

2. In `outputs.conf`, define the `my_syslog_group` target group for the non-Splunk server:

```
[syslog:my_syslog_group]
server = loghost.example.com:514
```

Troubleshoot forwarding

Troubleshoot forwarder/receiver connection

If forwarding does not work, or does not work correctly, you can follow these troubleshooting steps to determine why.

Receiver does not accept new connections on its receiving port

If the internal queue on the receiving indexer gets blocked, the indexer shuts down the receiving/listening (`splunktcp`) port after a specified interval of being unable to insert data into the queue. Once the queue is again able to start accepting data, the indexer reopens the port.

However, sometimes (on Windows machines only) the indexer is unable to reopen the port once its queue is unblocked. To remediate, you must restart the indexer.

If you have this issue, you can set the receiver `stopAcceptorAfterQBlock` attribute in inputs.conf to a higher value, so that it does not close the port as quickly. This attribute determines the amount of time the indexer waits before closing the port. The default is 300 seconds (five minutes).

If you use load-balanced forwarders, they switch their data stream to another indexer in the load-balanced group based on their time-out interval, set in outputs.conf with the `writeTimeout` attribute. This results in automatic failover when the receiving indexers have blocked queues.

Confusing the receiving and management ports

As part of setting up a forwarder, specify the receiver's `hostname/IP_address` and `port`. The forwarder uses these to send data to the receiver. Be sure to specify the port that was designated as the receiving port at the time the receiver was configured. If you mistakenly specify the receiver's management port, the receiver will generate an error similar to this:

```
splunkd.log:03-01-2010 13:35:28.653 ERROR TcpInputFd - SSL Error =
error:140760FC:SSL routines:SSL23_GET_CLIENT_HELLO:unknown protocol
splunkd.log:03-01-2010 13:35:28.653 ERROR TcpInputFd - ACCEPT_RESULT=-1
```

```
VERIFY_RESULT=0
splunkd.log:03-01-2010 13:35:28.653 ERROR TcpInputFd - SSL Error for fd
from HOST:localhost.localdomain, IP:127.0.0.1, PORT:53075
splunkd.log:03-01-2010 13:35:28.653 ERROR TcpInputFd - SSL Error =
error:140760FC:SSL routines:SSL23_GET_CLIENT_HELLO:unknown protocol
splunkd.log:03-01-2010 13:35:28.653 ERROR TcpInputFd - ACCEPT_RESULT=-1
VERIFY_RESULT=0
splunkd.log:03-01-2010 13:35:28.653 ERROR TcpInputFd - SSL Error for fd
from HOST:localhost.localdomain, IP:127.0.0.1, PORT:53076
splunkd.log:03-01-2010 13:35:28.653 ERROR TcpInputFd - SSL Error =
error:140760FC:SSL routines:SSL23_GET_CLIENT_HELLO:unknown protocol
splunkd.log:03-01-2010 13:35:28.654 ERROR TcpInputFd - ACCEPT_RESULT=-1
VERIFY_RESULT=0
splunkd.log:03-01-2010 13:35:28.654 ERROR TcpInputFd - SSL Error for fd
from HOST:localhost.localdomain, IP:127.0.0.1, PORT:53077
splunkd.log:03-01-2010 13:35:28.654 ERROR TcpInputFd - SSL Error =
error:140760FC:SSL routines:SSL23_GET_CLIENT_HELLO:unknown protocol
splunkd.log:03-01-2010 13:35:28.654 ERROR TcpInputFd - ACCEPT_RESULT=-1
VERIFY_RESULT=0
```

Closed receiver socket

If a receiving indexer queues become full, it closes the receiver socket to prevent additional forwarders from connecting to it. If a forwarder with load-balancing enabled can no longer forward to that receiver, it sends its data to another indexer on its list. If the forwarder does not employ load-balancing, it holds the data until you resolve the problem.

The receiver socket reopens automatically when the queue gets unclogged.

Typically, a receiver gets behind on the data flow because it can no longer write data due to a full disk or because it is itself attempting to forward data to another Splunk Enterprise instance that is not accepting data.

The following warning message will appear in `splunkd.log` if the socket gets blocked:

```
Stopping all listening ports. Queues blocked for more than N seconds.
```

This message will appear when the socket reopens:

```
Started listening on tcp ports. Queues unblocked.
```

Disable receiving

To disable receiving through the CLI, run the `splunk disable listen` command:

```
splunk disable listen -port <port> -auth <username>:<password>
```

You can also disable receiving by deleting the `[splunktcp]` stanza from `inputs.conf`.

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has around configuring forwarding.