

Splunk Admin Guide – Hands On

Frank Anaya

Table of Contents

Your Splunk.com account	15
Obtaining a Splunk.com account	15
Installing Splunk on Windows	16
Logging in the first time	17
Run a simple search	19
NOTE	19
Intro - Splunk Web Framework Fundamentals	22
Introducing the Splunk Web Framework	22
A quick note about advanced XML	23
Architecture of the Splunk Web Framework	23
Description of the architecture	24
The Splunk web interface	26
Simple XML	27
SimpleXML extensions	28
HTML	29
SplunkJS libraries	29
splunkd	30
Creating a Splunk app	31
Populating data with Eventgen	34
NOTE	34
NOTE	34
Installing an add-on	35
SPLUNK CLI Command Guide	38
Configuring Eventgen	40
Viewing the Destinations app	42
Creating your first dashboard	44
Summary Session 1	48
Splunk Universal Forwarders	50
Leveraging your forwarders	51
Lab: Using the Universal Forwarder to gather data	53

Getting ready	53
TIP	53
How to do it...	53
TIP	54
TIP	54
How it works.....	56
There's more...	56
ADD THE RECEIVING INDEXER VIA OUTPUTS.CONF	56
TIP	57
Heavy Forwarder management	58
NOTE	62
Managing your Heavy Forwarder	63
MANUAL ADMINISTRATION	63
DEPLOYMENT SERVER	63
Important configuration files	64
Splunk and big data	66
Streaming data	66
Latency of data.....	67
Sparseness of data.....	67
Splunk data sources	68
Machine data	68
Web logs	68
Data files	69
Social media data	69
Other data types.....	69
Splunk Config Files - Introduction	70
Locating Splunk configuration files.....	70
The configuration merging logic	73
The merging order	73
THE MERGING ORDER OUTSIDE OF SEARCH	73
THE MERGING ORDER WHEN SEARCHING	74
The configuration merging logic.....	75

CONFIGURATION MERGING – EXAMPLE 1.....	75
CONFIGURATION MERGING – EXAMPLE 2.....	76
CONFIGURATION MERGING – EXAMPLE 3.....	77
TIP	80
CONFIGURATION MERGING – EXAMPLE 4.....	80
Using btool	82
Summary	84
Creating indexes	85
Buckets	87
Data inputs	89
Splunk events and fields	93
Extracting new fields	94
Getting More Data In New Data :)	99
Introduction	99
Indexing files and directories	101
Getting ready	101
How to do it...	102
TIP	103
TIP	104
How it works	105
TIP	105
There's more....	105
ADDING A FILE OR DIRECTORY DATA INPUT VIA THE CLI	105
NOTE	106
ADDING A FILE OR DIRECTORY INPUT VIA INPUTS.CONF	106
NOTE	107
ONE-TIME INDEXING OF DATA FILES VIA THE SPLUNK CLI	107
TIP	107
INDEXING THE WINDOWS EVENT LOGS	107
Getting data through network ports	109
Getting ready	109
How to do it...	109

How it works.....	112
TIP	112
There's more.....	112
ADDING A NETWORK INPUT VIA THE CLI.....	113
ADDING A NETWORK INPUT VIA INPUTS.CONF	113
TIP	113
Using scripted inputs.....	115
Getting ready	115
How to do it.....	115
How it works.....	118
Using modular inputs	119
Getting ready	119
How to do it.....	119
How it works.....	123
TIP	124
There's more.....	124
Using the Universal Forwarder to gather data.....	125
Getting ready	125
TIP	125
How to do it.....	126
TIP	126
TIP	126
How it works.....	128
There's more.....	128
ADD THE RECEIVING INDEXER VIA OUTPUTS.CONF	129
TIP	129
Loading an Access Log.....	130
Getting ready	130
How to do it.....	130
How it works.....	134
Defining field extractions.....	135
Getting ready	135

How to do it...	135
How it works...	138
Defining event types and tags	139
NOTE	139
Getting ready	139
How to do it...	140
How it works...	141
There's more...	142
ADDING EVENT TYPES AND TAGS VIA EVENTTYPES.CONF AND TAGS.CONF	
.....	142
TIP	142
More Splunk .conf files	143
props.conf	143
COMMON ATTRIBUTES	143
STANZA TYPES	148
PRIORITIES INSIDE A TYPE	150
ATTRIBUTES WITH CLASS	151
inputs.conf	152
COMMON INPUT ATTRIBUTES	152
NOTE	153
FILES AS INPUTS	153
NETWORK INPUTS	159
NATIVE WINDOWS INPUTS	161
SCRIPTS AS INPUTS	162
transforms.conf	163
CREATING INDEXED FIELDS	164
MODIFYING METADATA FIELDS	166
LOOKUP DEFINITIONS	169
USING REPORT	173
CHAINING TRANSFORMS	175
DROPPING EVENTS	177
fields.conf	178

outputs.conf	179
indexes.conf.....	179
authorize.conf	181
savedsearches.conf.....	182
times.conf.....	182
commands.conf.....	183
web.conf.....	183
User interface resources.....	183
Views and navigation	183
Appserver resources	184
Metadata	185
Summary	188
Search Processing Language.....	190
Anatomy of a search	190
Search pipeline.....	191
Time modifiers	193
NOTE	194
Session 3-2: Diving into Data – Search and Report for Admins	199
Introduction	199
TIP	200
TIP	201
NOTE	203
Making raw event data readable.....	205
Getting ready	205
How to do it.....	205
How it works.....	207
TIP	208
There's more.....	208
TABULATING EVERY FIELD.....	208
REMOVING FIELDS, THEN TABULATING EVERYTHING ELSE	208
TIP	209
Finding the most accessed web pages.....	210

Getting ready	210
How to do it.....	210
How it works.....	211
There's more.....	212
SEARCHING FOR THE TOP 10 ACCESSED WEB PAGES.....	212
SEARCHING FOR THE MOST ACCESSED PAGES BY USER.....	212
NOTE	213
Finding the most used web browsers	214
Getting ready	214
How to do it.....	214
How it works.....	215
There's more.....	216
SEARCHING FOR THE WEB BROWSER DATA FOR THE MOST USED OS TYPES	216
Identifying the top-referring websites	217
Getting ready	217
How to do it.....	217
How it works.....	218
There's more.....	218
SEARCHING FOR THE TOP 10 USING STATS INSTEAD OF TOP.....	219
NOTE	219
Charting web page response codes.....	220
Getting ready	220
How to do it.....	220
How it works.....	221
There's more.....	222
TOTALING SUCCESS AND ERROR WEB PAGE RESPONSE CODES.....	222
Displaying web page response time statistics	224
Getting ready	224
How to do it.....	224
How it works.....	225
There's more.....	226

DISPLAYING WEB PAGE RESPONSE TIME BY ACTION	226
Listing the top viewed products.....	228
Getting ready	228
How to do it.....	228
How it works.....	229
There's more.....	230
SEARCHING FOR THE PERCENTAGE OF CART ADDITIONS FROM PRODUCT VIEWS.....	230
Charting the application's functional performance	232
Getting ready	232
How to do it.....	232
How it works.....	233
TIP	234
There's more.....	234
Charting the application's memory usage	235
Getting ready	235
How to do it.....	235
How it works.....	236
Counting the total number of database connections.....	238
Getting ready	238
How to do it.....	238
How it works.....	239
Introduction	241
LAB-Calculating the average session time on a website	244
Getting ready	244
How to do it.....	244
How it works.....	249
There's more.....	250
TIP	250
STARTS WITH A WEBSITE VISIT, ENDS WITH A CHECKOUT	250
DEFINING MAXIMUM PAUSE, SPAN, AND EVENTS IN A TRANSACTION	251
NOTE	252

Calculating the average execution time for multi-tier web requests.....	253
Getting ready	253
How to do it...	253
How it works.....	258
There's more....	259
NOTE	260
CALCULATING THE AVERAGE EXECUTION TIME WITHOUT USING A JOIN	260
Displaying the maximum concurrent checkouts.....	261
Getting ready	261
How to do it...	261
How it works.....	264
NOTE	265
Session 4- Dashboards and Visualizations – Making Data Shine.....	267
Introduction	267
TIP	271
Creating an Operational Intelligence dashboard.....	273
Getting ready	273
How to do it...	273
How it works.....	275
TIP	275
There's more.....	275
CHANGING DASHBOARD PERMISSIONS	275
TIP	276
Using a pie chart to show the most accessed web pages.....	277
Getting ready	277
How to do it...	277
How it works.....	281
There's more....	282
SEARCHING FOR THE TOP 10 ACCESSED WEB PAGES	282
Displaying the unique number of visitors	283
Getting ready	283
How to do it...	283

NOTE	286
How it works	287
There's more	287
COLORING THE VALUE BASED ON RANGES	289
ADDING TRENDS AND SPARKLINES TO THE VALUES	290
NOTE	290
Using a gauge to display the number of errors	291
Getting ready	291
How to do it	291
How it works	293
There's more	294
NOTE	294
Charting the number of method requests by type and host	295
Getting ready	295
How to do it	295
How it works	297
Creating a timechart of method requests, views, and response times	298
Getting ready	298
How to do it	298
How it works	300
There's more	301
METHOD REQUESTS, VIEWS, AND RESPONSE TIMES BY HOST	301
Using a scatter chart to identify discrete requests by size and response time	303
Getting ready	303
How to do it	303
How it works	305
There's more	306
USING TIME SERIES DATA POINTS WITH A SCATTER CHART	306
Creating an area chart of the application's functional statistics	308
Getting ready	308
How to do it	308
How it works	309

Using a bar chart to show the average amount spent by category	312
Getting ready	312
How to do it...	312
How it works	314
Creating a line chart of item views and purchases over time	316
Getting ready	316
How to do it...	316
How it works	318
Building an Operational Intelligence Application	319
Introduction	319
TIP	320
Creating an Operational Intelligence application	322
Getting ready	322
How to do it...	322
How it works	324
There's more	324
CREATING AN APPLICATION FROM ANOTHER APPLICATION	324
DOWNLOADING AND INSTALLING A SPLUNK APP	325
TIP	326
Adding dashboards and reports	327
Getting ready	327
How to do it...	327
How it works	331
TIP	331
There's more	332
CHANGING PERMISSIONS OF SAVED REPORTS	332
Organizing the dashboards more efficiently	334
Getting ready	334
How to do it...	334
How it works	336
NOTE	336
There's more	337

MODIFYING THE SIMPLE XML DIRECTLY	337
TIP	338
Dynamically drilling down on activity reports	339
Getting ready	339
How to do it.....	339
How it works.....	342
TIP	343
There's more.....	343
DISABLING THE DRILLDOWN FEATURE IN TABLES AND CHARTS.....	343
TIP	343
Creating a form for searching web activity.....	344
Getting ready	344
How to do it.....	344
How it works.....	349
There's more.....	350
ADDING A SUBMIT BUTTON TO YOUR FORM	350
Linking web page activity reports to the form	351
Getting ready	351
How to do it.....	351
How it works.....	353
There's more.....	354
ADDING AN OVERLAY TO THE SESSIONS OVER TIME CHART	354
Displaying a geographical map of visitors.....	355
Getting ready	355
How to do it.....	355
How it works.....	358
TIP	359
There's more.....	359
ADDING A MAP PANEL USING SIMPLE XML.....	359
MAPPING DIFFERENT DISTRIBUTIONS BY AREA.....	360
Scheduling PDF delivery of a dashboard.....	361
Getting ready	361

How to do it.....	361
How it works.....	364
TIP	364

Session 1 - Splunk in Action

Splunk, whose name was inspired by the process of exploring caves, or *spelunking*, helps analysts, operators, programmers, and many others explore many types of data, including raw machine data from their organizations, by collecting, analyzing, and acting on them. This multinational company, cofounded by Michael Baum, Rob Das, and Erik Swan, has a core product called Splunk Enterprise. This product manages searches, inserts, deletes, filters, and analyzes big data that is generated by machines, as well as many other types of data.

NOTE

Throughout the book, we will be covering the fundamental, barebones concepts of Splunk so you can learn quickly and efficiently. We reserve any deep discussion of concepts to Splunk's online documentation. Where necessary, we provide links to help provide you with the practical skills, and examples, so you can get started quickly. All images and exercise materials used in this book are available online.

With very little time, you can achieve direct results using Splunk, which you can access through a free enterprise trial license. While this license limits you to 500 MB of data ingested per day, it will allow you to quickly get up to speed with Splunk and learn the essentials of this powerful software.

The exercises in this section may look challenging at first, but if you follow what we've written closely, we believe you will quickly learn the fundamentals you need to use Splunk effectively. Together, we will make the most of the Trial License and give you a visible result that you can use to create valuable insights for your company (and, if you like, proudly show to your friends and coworkers).

Your Splunk.com account

First you will need to register for a Splunk.com account. This is the account that you will use if you decide to purchase a license later. Go ahead and do this now. From here on, the password you use for your Splunk.com account will be referred to as your Splunk.com password.

Obtaining a Splunk.com account

To obtain your Splunk.com account, perform the following steps:

1. Go to the Splunk signup page at <http://www.splunk.com>.
2. In the upper right hand corner, click on **My Account | Sign Up**.
3. Enter the information requested.
4. Create a username and password.

You will then need to download the Splunk Enterprise software. Go to <http://download.splunk.com> and select the Splunk Enterprise free download. Choose your operating system, being careful to select 32- or 64-bit (whichever is appropriate in your case; most should select 64-bit, which most computers today use). For Windows, download the [*.msi](#) file. For Mac OS X, download the [*.dmg](#) file. In this book, we will work with Version 6.4.1 or later.

The installation is very straightforward. Follow the steps for your particular operating system, whether it be Windows or Mac OS X.

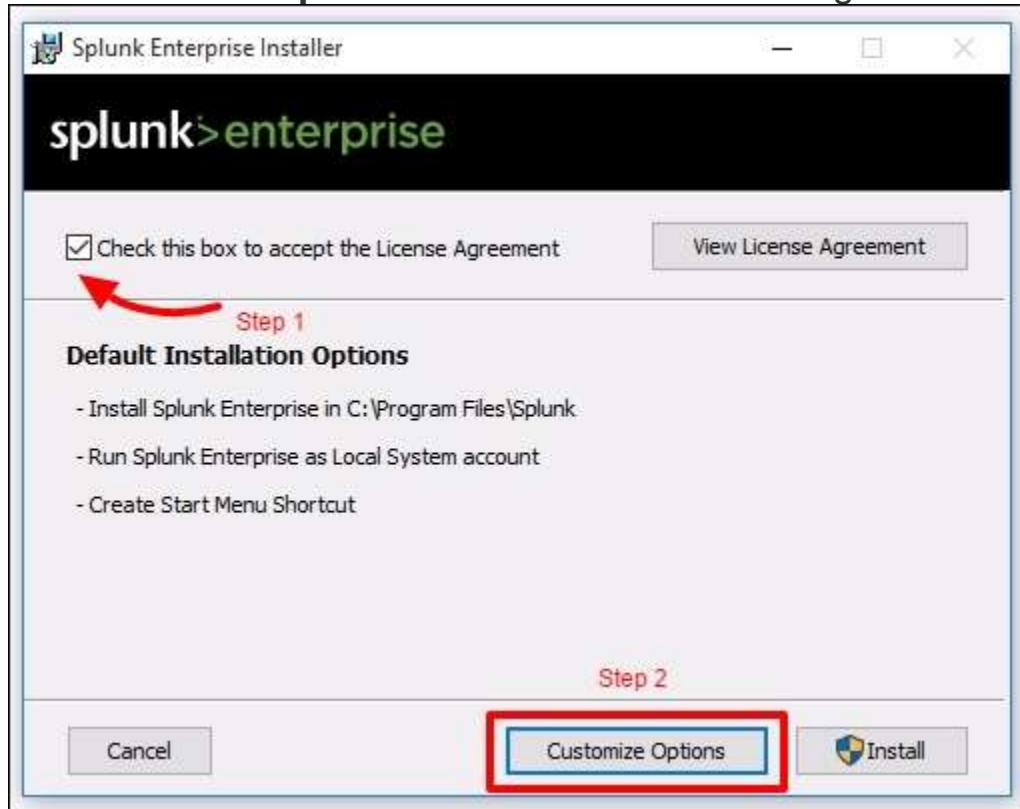
NOTE

Make sure that there is no previous installation of Splunk in your system. If there is, uninstall the old version before proceeding with the next steps.

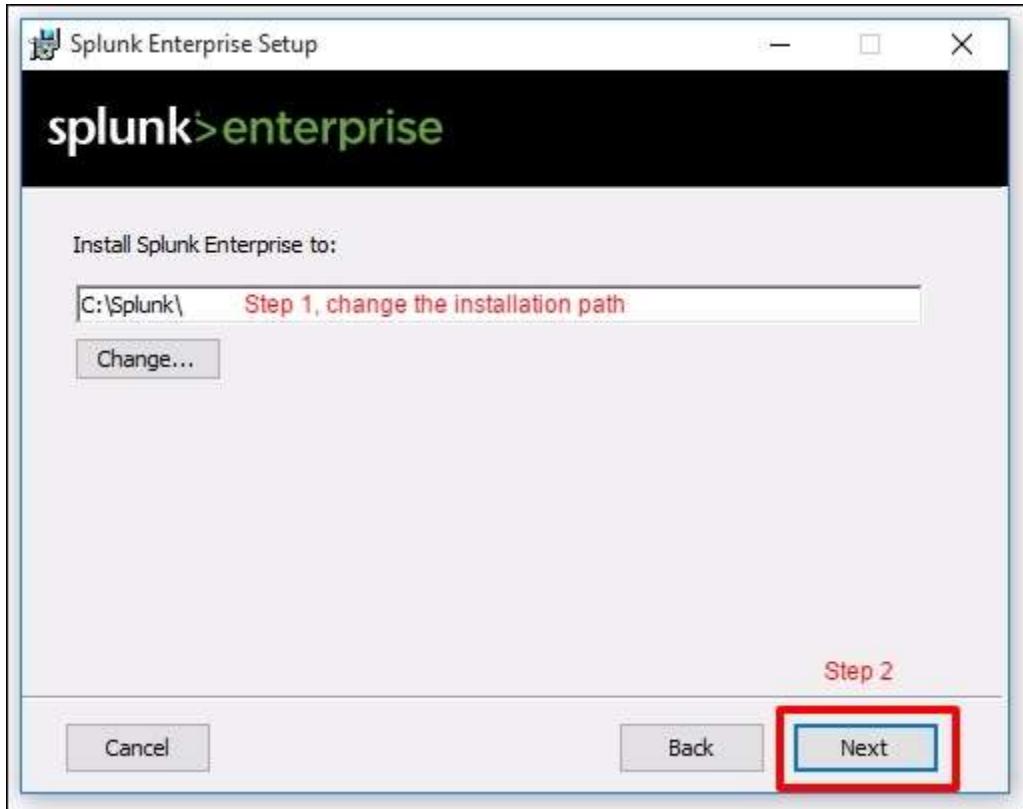
Installing Splunk on Windows

These are the instructions you need to follow to install Splunk on your Windows desktop. Take your time and do not rush the installation. Many chapters in this book will rely on these steps:

1. Run the installer that you downloaded.
2. Check the box to accept the License Agreement and then click on **Customize Options** as shown in the following screenshot:



3. Change the **installation path** to <c:\splunk>. You will thank us later as it simplifies issuing **Splunk CLI (command-line interface)** commands. This is also a best practice used by modern Windows administrators. Remember to eliminate white spaces in directory names as well, as it causes complications with scripting. Click on **Next** to continue as seen in the following screenshot:



4. Install Splunk Enterprise as the **Local System** and then click on **Next**.
5. Leave the checkbox selected to **Create Start Menu Shortcut**.
6. Click on **Install**.
7. Wait for the installation to complete.
8. Click on **Finish** to complete the installation. It will attempt to launch Splunk for the first time in your default browser.

NOTE

Throughout the book, you will see references to `$SPLUNK_HOME`. This will be the installation directory of Splunk. In Windows, as a convention used in this book, `$SPLUNK_HOME` will be at `C:\Splunk`.

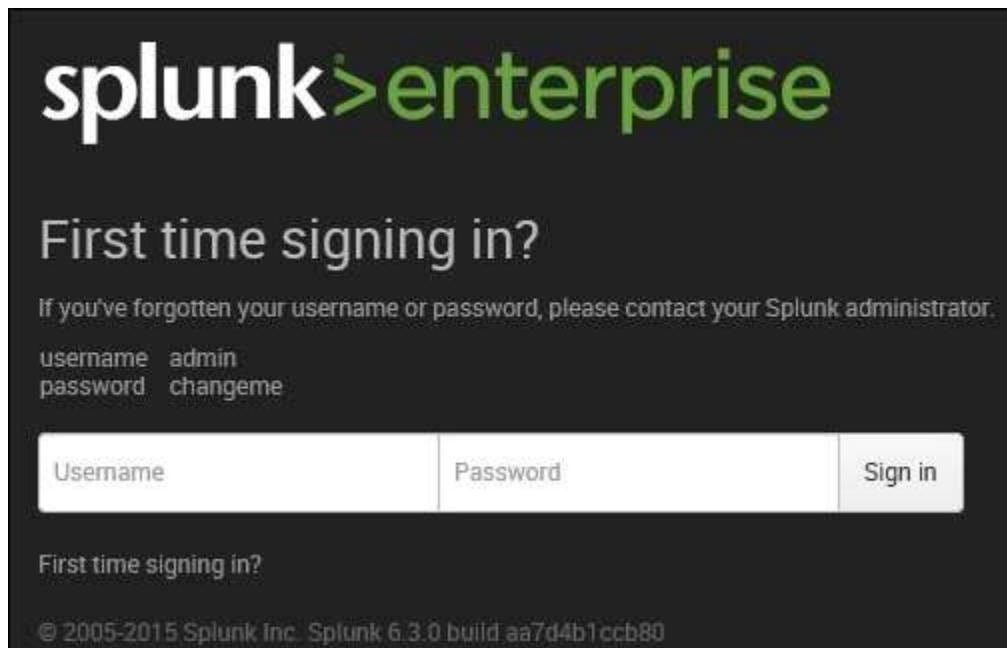
Logging in the first time

Launch the application the first time in your default browser. You can also manually access the Splunk web page via the [HTTP://localhost:8000](http://localhost:8000) URL.

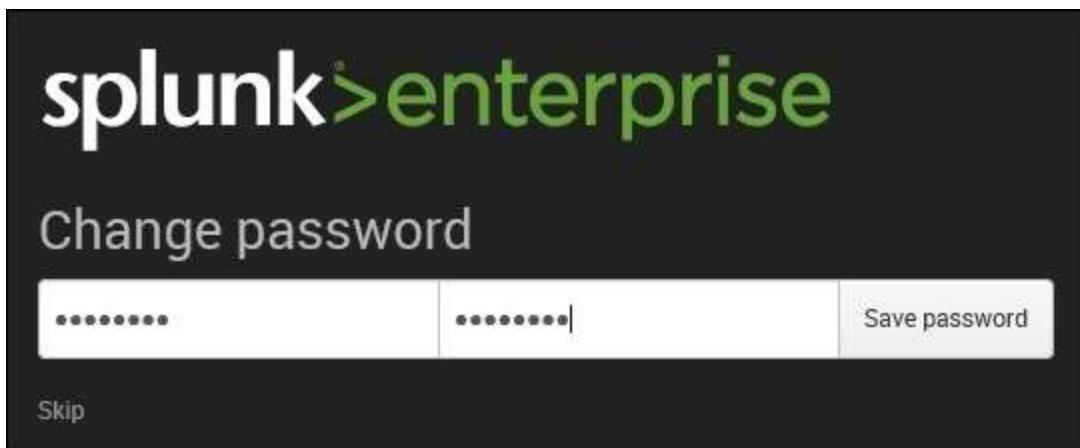
NOTE

Splunk requires you to use a modern browser. It supports most versions of Google Chrome, Firefox, and newer versions of Internet Explorer. It may not support older versions of Internet Explorer.

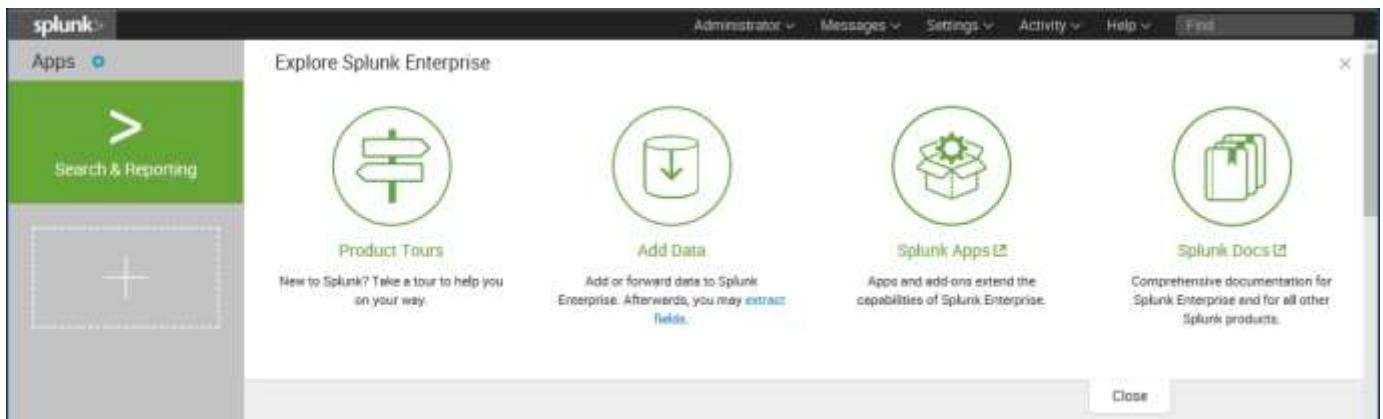
Log in with the default username and password (**admin : changeme**) as indicated in the following screenshot:



The next step is to change the default administrator password, while keeping the default username. Do not skip this step. Make security an integral part of your day-to-day routine. Choose a password that will be secure:



Assuming all goes well, you will now see the default Splunk **Search & Reporting** dashboard:



Run a simple search

You are finally ready to run your very first Splunk search query:

1. Go ahead and create your first Splunk search query. Click on the **Search & Reporting** app. You will be introduced to Splunk's very own internal index: this is Splunk's way of *splunking* itself (or collecting detailed information on all its underlying processes).
2. In the **New Search** input, type in the following search query (more about the **Search Processing Language (SPL)** in, [Chapter 3, Search Processing Language](#)):
 3. `SPL> index=_internal sourcetype=splunkd`

NOTE

The `SPL>` prefix will be used as a convention in this book to indicate a `Search` command as opposed to the `c:\>` prefix which indicates a Windows command.

4. This search query will have as an output the raw events from the `metrics.log` file that is stored in the `_internal` index. A log file keeps track of every event that takes place in the system. The `_internal` index keeps track of every event that occurs and makes it easily accessible.
5. Take a look at these raw events, as shown in the following screenshot. You will see fields listed on the left side of the screen. The important **Selected Fields** are `host`, `source`, and `sourcetype`. We will go into more detail about these later, but suffice it to say that you will frequently search on one of these, as we have done here. As you can see from the highlighted fields, we indicated that we were looking for events where `sourcetype=splunkd`. Underneath **Selected Fields**, you will see **Interesting Fields**. As you can tell, the purposes of many of these fields are easy to guess:

The screenshot shows the Splunk interface with the following details:

- Header:** 2,138 events (10/31/15 5:48:48.000 AM to 10/31/15 6:03:48.000 AM)
- Toolbar:** Job, Smart Mode
- Panel Tabs:** Events (2,138), Patterns, Statistics, Visualization
- Format Timeline:** 1 minute per column
- Event List:**

	Time	Event
< Hide Fields	10/31/15 6:03:42.401	10-31-2015 06:03:42.401 -0400 INFO Metrics - group=thruput, name=thruput, instantaneous_kbps=1.048615, instantaneous_eps=4.129061, average_kbps=0.943057, total_k_processed=31135.000000, kb=32.506836, ev=128.000000 host = WIN-DT11F5NUKEN source = C:\Splunk\var\log\splunk\metrics.log sourcetype = splunkd
Selected Fields	10/31/15 6:03:42.401	10-31-2015 06:03:42.401 -0400 INFO Metrics - group=thruput, name=syslog_output, instantaneous_kbps=0.000000, instantaneous_eps=0.000000, average_kbps=0.000000, total_k_processed=0.000000, kb=0.000000, ev=0.000000 host = WIN-DT11F5NUKEN source = C:\Splunk\var\log\splunk\metrics.log sourcetype = splunkd
Interesting Fields	10/31/15 6:03:42.401	10-31-2015 06:03:42.401 -0400 INFO Metrics - group=thruput, name=index_thruput, instantaneous_kbps=1.048615, instantaneous_eps=3.516154, average_kbps=0.942937, total_k_processed=31131.000000, kb=32.506836, ev=109.000000 host = WIN-DT11F5NUKEN source = C:\Splunk\var\log\splunk\metrics.log sourcetype = splunkd
# date_minute 15	10/31/15 6:03:42.401	10-31-2015 06:03:42.401 -0400 INFO Metrics - group=queue, name=winparsing, max_size_kb=500, current_size_kb=0, current_size=0, largest_size=0, smallest_size=0 host = WIN-DT11F5NUKEN source = C:\Splunk\var\log\splunk\metrics.log sourcetype = splunkd

Intro - Splunk Web Framework Fundamentals

So you've installed Splunk, got things running, and now what? Hopefully, that is where this guide will come in and help you get the ball rolling, making fresh, interactive, useful, and dynamic applications using the Splunk Web Framework. We are hoping that we can actually get you creating some interesting applications without the usual log, index, search, graph, and report documentation that seems to be out in abundance.

Introducing the Splunk Web Framework

Welcome to the Splunk Web Framework, which has been set up as an essential support structure for Splunk users to build custom reports, dashboards, and apps on Splunk and with Splunk. This means that there is a supporting environment that can be used to develop end-to-end applications with no need to install anything other than Splunk. The Splunk Web Framework allows the user to start from the basics using a drag-and-drop interface, and makes them able to get underneath the hood and interact and customize the code directly. Further still, developers don't even need to develop with Splunk as their platform of choice to display their data. They are free to simply interface with Splunk API calls, search for data, and then display this returned data directly on their own websites and applications.

As of Splunk version 6, there was a major overhaul to the Splunk Web Framework. The framework is now integrated directly into Splunk Enterprise 6, so now you don't need to install anything else to start using the web framework. Previously, in Splunk 5, you needed to use a standalone version of the web framework. So unless you're using an old version of Splunk, you will be able to get going and working with the framework straight away. All your apps from previous versions of Splunk should work on Splunk 6, including apps created in Advanced XML, so it is well worth the upgrade to get an improved interface and functionality that it brings.

A quick note about advanced XML

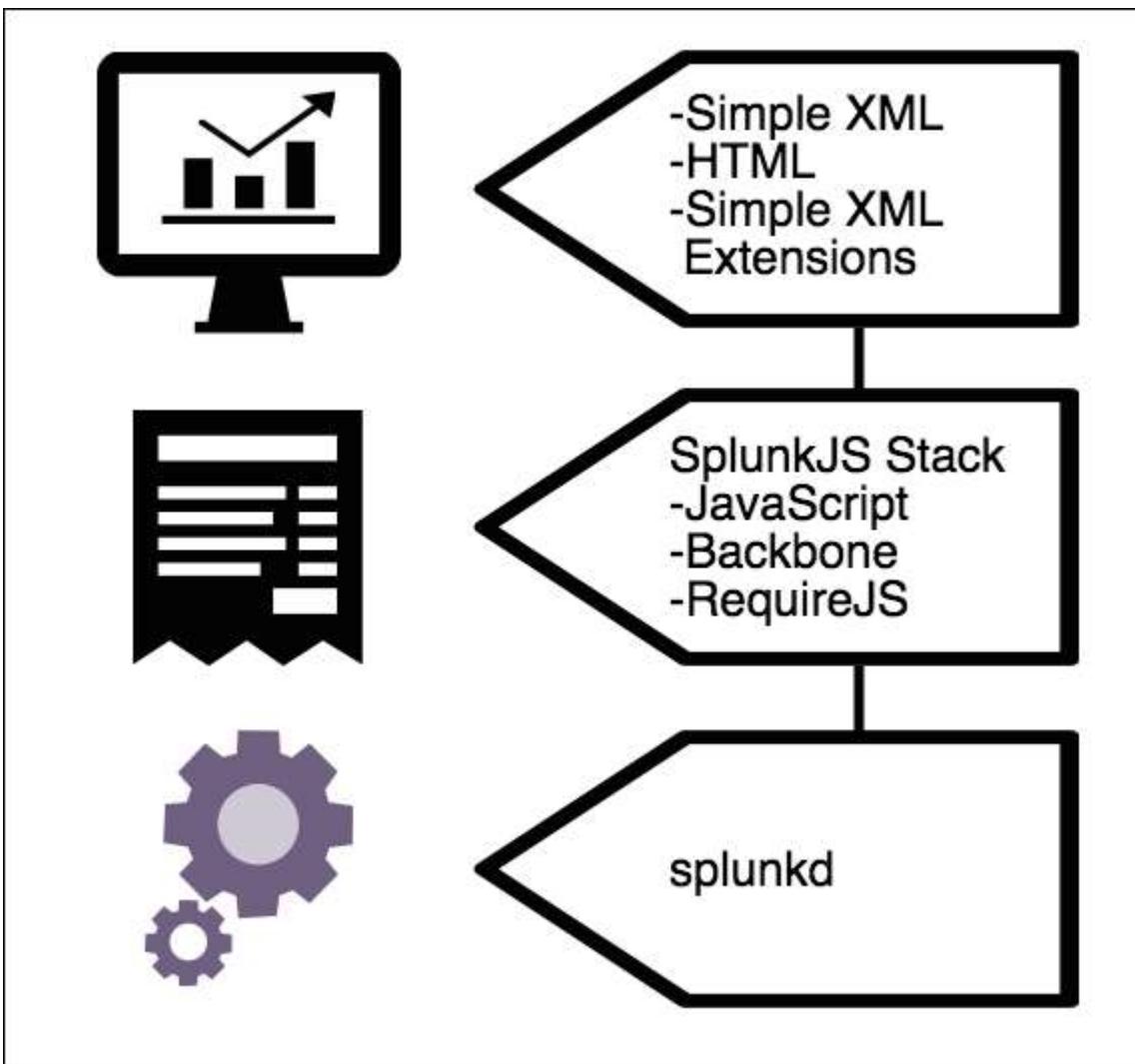
Let's get this out of the way early. You may have heard about Advanced XML, or you may have even seen some dashboards or views created in your environment that have been set up using Advanced XML. As of Splunk Enterprise 6.3, the Advanced XML feature has been deprecated. Although apps and dashboards using Advanced XML will continue to work and Splunk will continue to support and fix bugs, there will no longer be any feature enhancements to the Advanced XML feature of the Splunk Web Framework.

A date has not yet been set for the removal of Advanced XML from Splunk Enterprise. All future development should be done using other features of the Splunk Web Framework, and all existing apps or dashboards that use Advanced XML should be migrated away from Advanced XML and onto one of the other options available in the Splunk Web Framework.

Architecture of the Splunk Web Framework

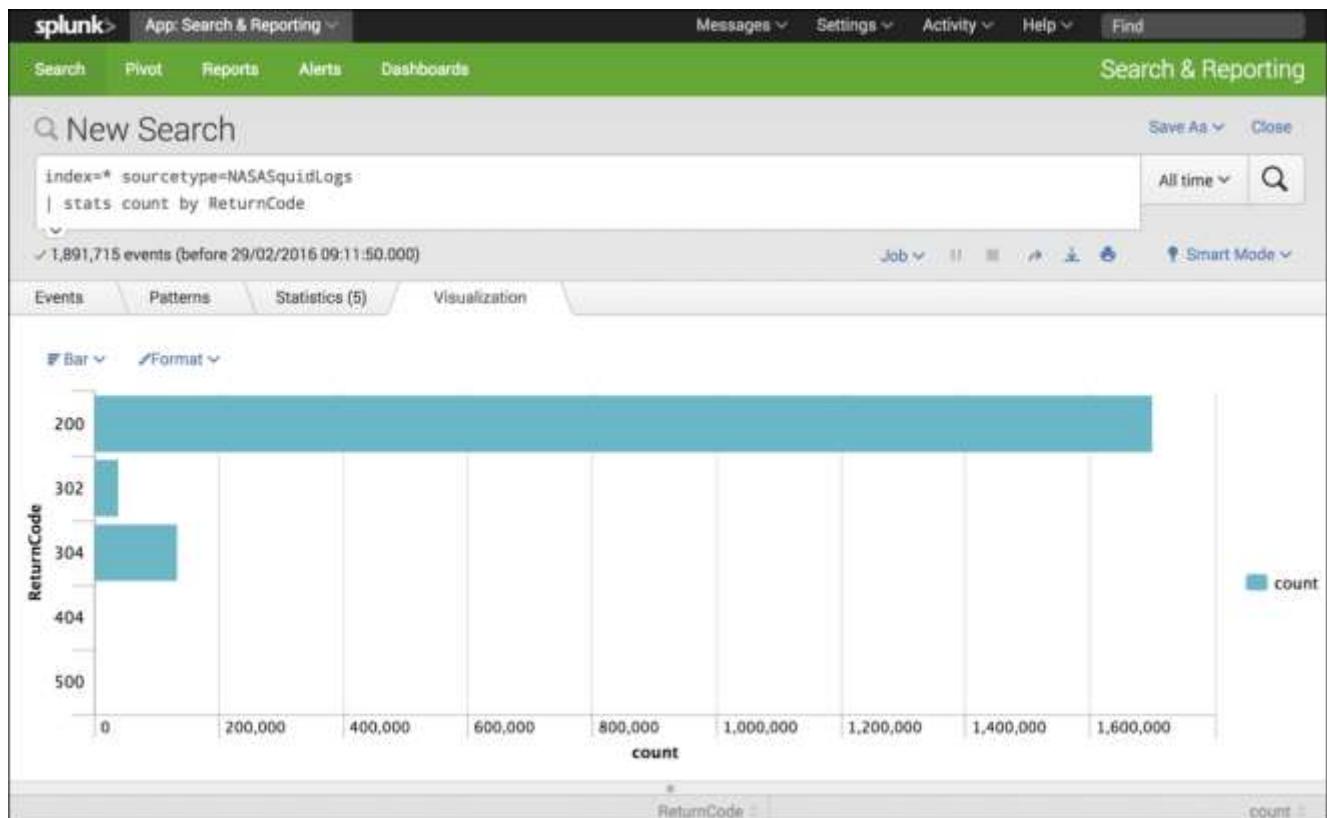
The Splunk Web Framework is now built directly on the core Splunk daemon, splunkd. Originally, splunkd only handled indexing, searching, and forwarding, but as of version 6.2, it also operates the Splunk Web Interface. Making this change was practical because it gave the framework the tools you need to build web applications directly on Splunk, as well as use the data that Splunk provides to display on your own website.

Within the framework, you have an app that will include numerous dashboard elements within the app. Within the dashboards, you will then have numerous panel and visualization elements that will make up your dashboard:



Description of the architecture

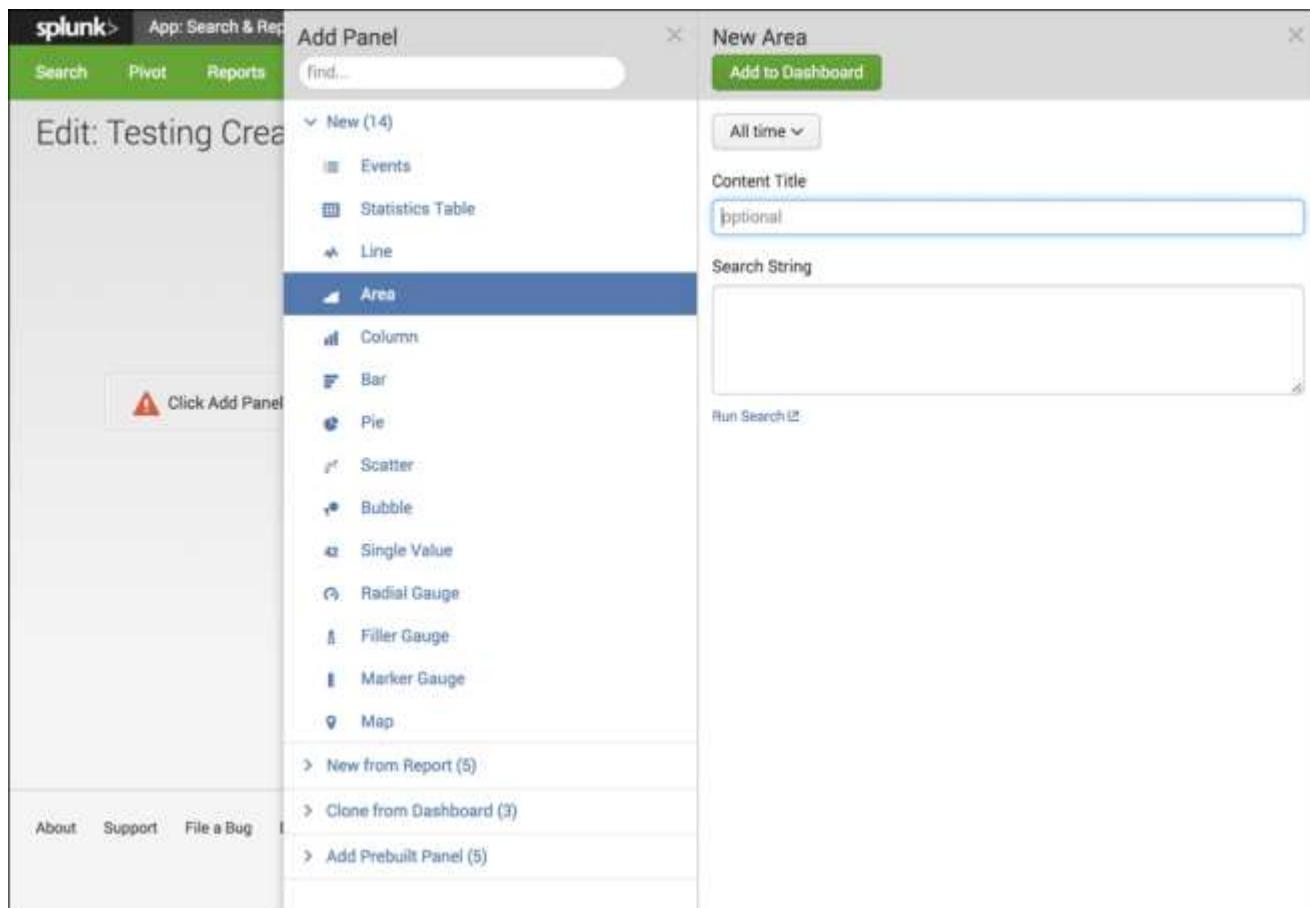
The preceding diagram provides a clear breakdown of the architecture, and its three distinct layers. It shows splunkd, which is built on C/C++ for speed and stability, as a server that provides the indexing and searching capabilities to the SplunkJS stack, which delivers the display and interface supporting the SimpleXML, HTML, and external web displays. Each layer builds on the others, providing further enhanced functionality.



The Splunk web interface

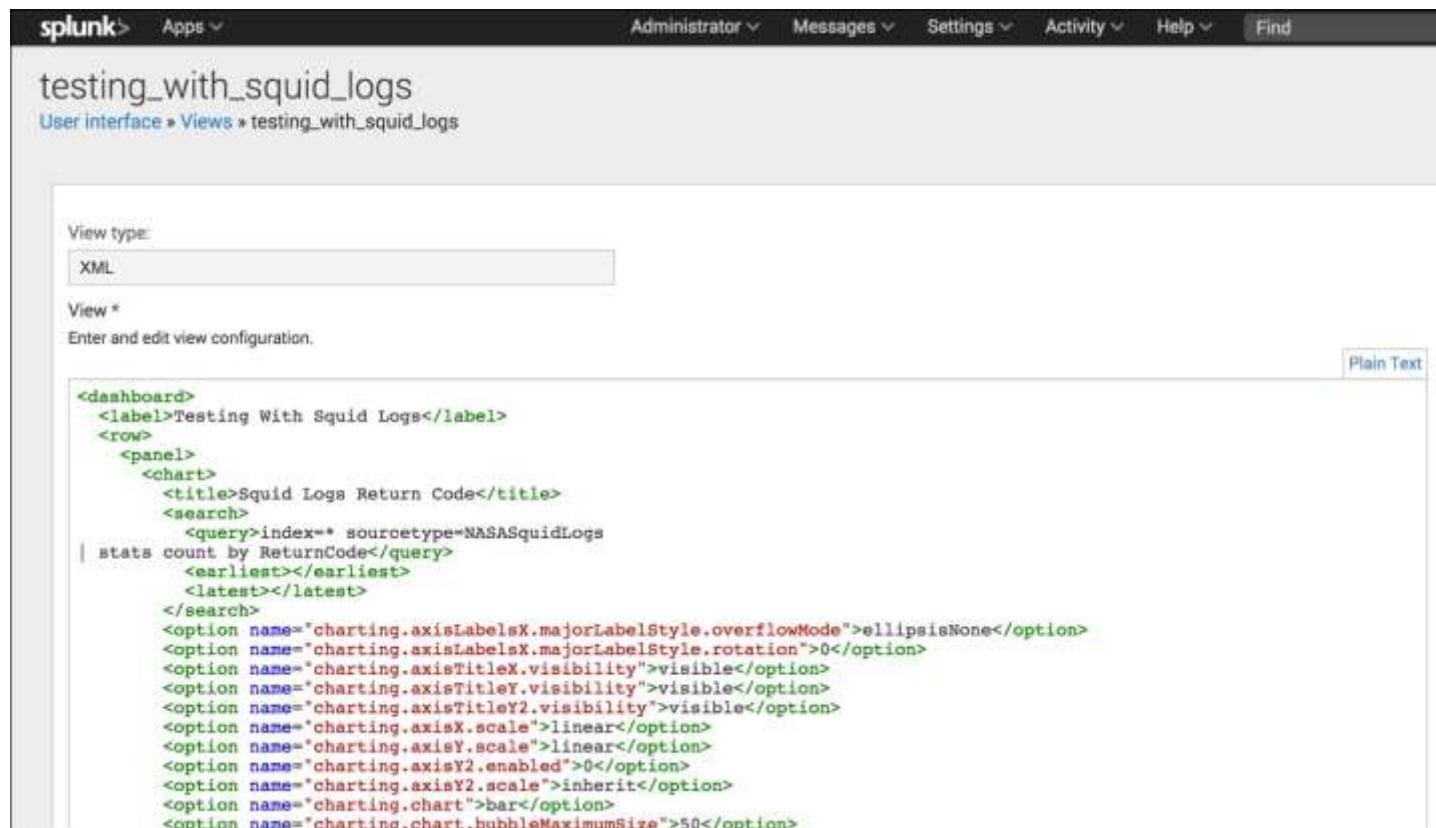
By now, I am sure you at least know that Splunk has a web interface. If you are competent with using Splunk, you would already be familiar with using the web interface for searching, configuring, and administration of Splunk. As part of the Splunk Web Framework, the web interface also provides an easy-to-use graphical user interface, which allows you to drag and drop tools and functionality with no prior programming knowledge or experience. It provides rapid development on the framework and allows you to visualize dashboard panels with ease.

The dashboard editor is the main interface and is part of the SimpleXML layer of the Splunk Web Framework; it allows you to build dashboards within Splunk Web. Here you can visualize your events and statistical information as dashboard panels and views and provide charting functionality. It even allows you to start providing form-based controls and an interface with the user.



Simple XML

Simple XML expands the functionality of the framework further and allows the user to fine-tune the dashboard panels with more layout and display options. Splunk's **Extensible Markup Language(XML)** is the underlying code that is developed when using the web interface and dashboard editor. Simple XML code can be edited and manipulated directly from Splunk's built-in editor, or you can use your own code editor to configure the easy-to-learn syntax. The directory structure within Splunk is also straightforward and easy to learn, and it helps you manipulate the environment in ways that you can't actually do within the web interface.



The screenshot shows the Splunk web interface with the following details:

- Header: splunk > Apps ▾
- Header: Administrator ▾ Messages ▾ Settings ▾ Activity ▾ Help ▾ Find
- Page Title: testing_with_squid_logs
- Breadcrumb: User interface > Views > testing_with_squid_logs
- View Type: XML (selected in a dropdown menu)
- View Configuration: Enter and edit view configuration.
- Code Editor Content:

```
<dashboard>
  <label>Testing With Squid Logs</label>
  <row>
    <panel>
      <chart>
        <title>Squid Logs Return Code</title>
        <search>
          <query>index=* sourcetype=NASASquidLogs
| stats count by ReturnCode</query>
          <earliest></earliest>
          <latest></latest>
        </search>
        <option name="charting.axisLabelsX.majorLabelStyle.overflowMode">ellipsisNone</option>
        <option name="charting.axisLabelsX.majorLabelStyle.rotation">0</option>
        <option name="charting.axisTitleX.visibility">visible</option>
        <option name="charting.axisTitleY.visibility">visible</option>
        <option name="charting.axisTitleY2.visibility">visible</option>
        <option name="charting.axisX.scale">linear</option>
        <option name="charting.axisY.scale">linear</option>
        <option name="charting.axisY2.enabled">0</option>
        <option name="charting.axisY2.scale">inherit</option>
        <option name="charting.chart">bar</option>
        <option name="charting.chart.bubbleMaximumSize">50</option>
```
- Plain Text button (top right of the code editor)

From the preceding example, you can see that the syntax of the Simple XML code is straightforward and relatively easy to learn. The code provides a multitude of options to tweak and fine-tune all aspects of the display of the different types of panels provided. It is definitely worth learning to use this function of the Splunk Web Framework. Although the drag-and-drop interface allows you to develop rich and interesting dashboard panels,

sooner or later you will start to want to configure the display in a way that you can only do in SimpleXML.

Each visualization type has a long list of properties that can be managed and changed through SimpleXML code. Although simple, you still need to adhere to the white space and open and close tags within the code. If not, you could end up with no display provided.

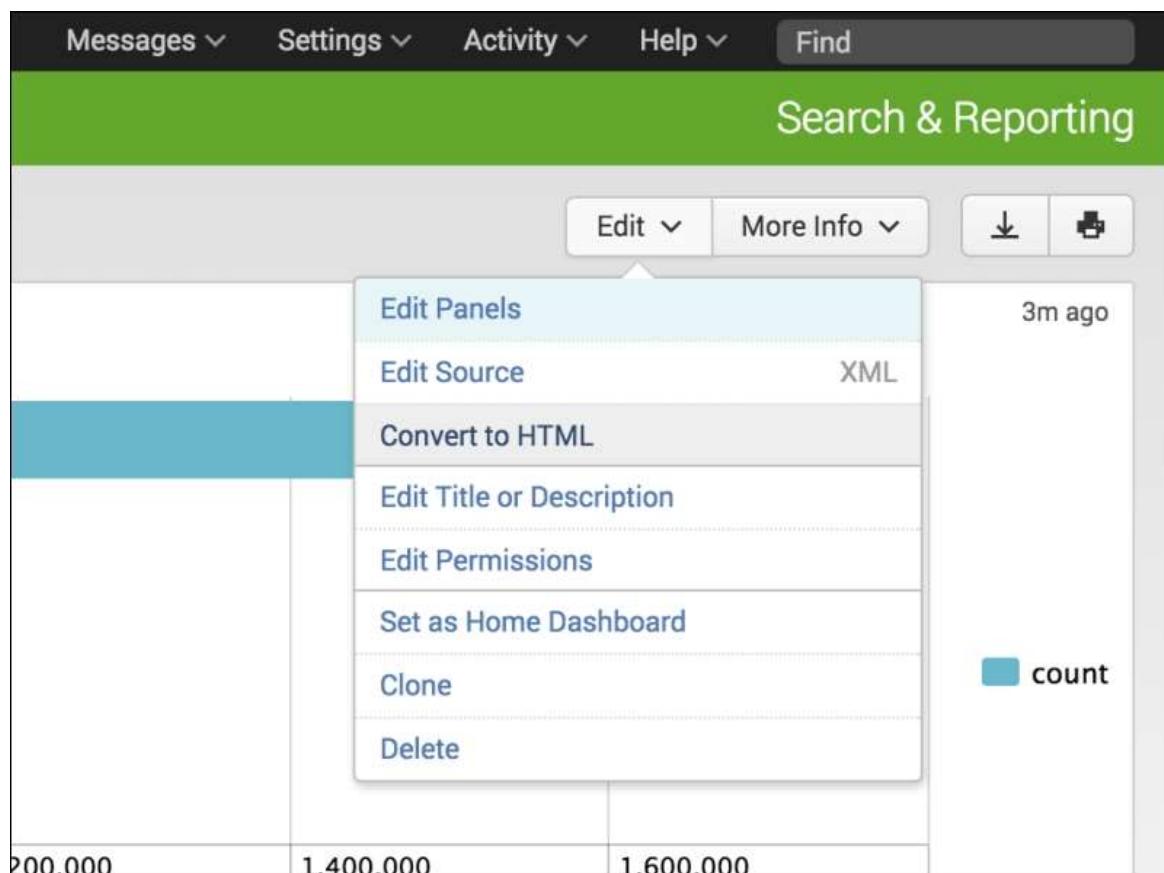
SimpleXML extensions

SimpleXML also allows you to create extensions to utilize CSS and JavaScript files so that you can further modify and enhance the behavior and appearance of a dashboard that was created through the code editor or web interface. You can modify the layouts further, add new visualizations, and customize the way that the end user interacts with the dashboards.

Working with SimpleXML extensions will get you working directly in the server directory structure of your Splunk deployment. Once you have added your CSS or JavaScript files to the server, it is simply a matter of editing your code to then use the files needed.

HTML

By now, you can see that each level of the framework expands functionality of the interface further, and utilizing HTML dashboards allows you to expand your functionality even further. Splunk comes with a converter that allows you to convert your Simple XML dashboards into HTML, and allows you to use the built-in code editor, edit, and configure the HTML dashboard further. As with Simple XML, you are also able to use your favorite code editor, allowing developers with knowledge of HTML, CSS, and JavaScript to transfer their knowledge and work directly in Splunk by using it as a platform to generate their HTML-based environment.



SplunkJS libraries

[SplunkJS](#) provides a framework of tools and libraries that allows developers to build and manage dashboards and organize dependencies, as well as integrate Splunk components into their own web applications. The libraries allow you to manage views and search managers to allow you to work with

searches and interact with Splunk data. [SplunkJS](#) removes the developer from the Splunk Web Interface but gives the ability to both build Splunk Apps for Splunk and build web applications using Splunk data.

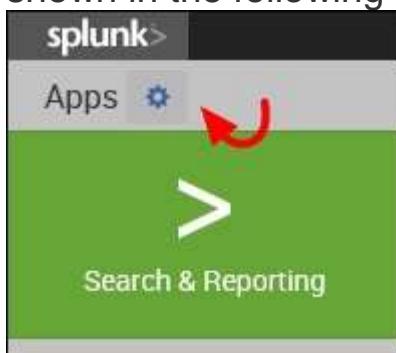
splunkd

This is the main system process that Splunk uses to handle all of the indexing, searching, forwarding, and web interface that you work with in Splunk Enterprise. Although we will need to restart Splunk and the splunkd process occasionally, this book will not be focusing on splunkd, as this would be more of a server administration focus.

Creating a Splunk app

It is good practice to create a custom Splunk app to isolate all the changes you make in Splunk. You may never have created an app before, but you will quickly see it is not very difficult. Here we will create a basic app called **Destinations** that we will use throughout this book:

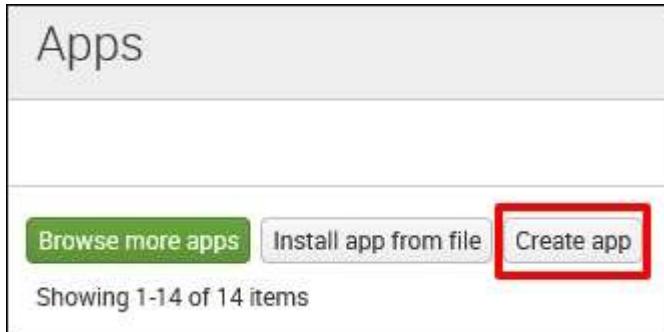
1. Let's access the **Manage Apps** page. There are two ways to do this; you may either click on the **Apps** icon at the *home page* as shown in the following screenshot:



2. Or select **Manage Apps** from the app dropdown in the top navigation bar of the **Search & Reporting** app:



3. At the **Manage Apps** page, click on the **Create app** icon as shown in the following screenshot:



- Finally, populate the forms with the following information to complete the app creation. When you are done, click on the **Save** button to create your first Splunk app:

Add new

Apps » Add new

Name

Give your app a friendly name for display in Splunk Web.

Folder name *

This name maps to the app's directory in \$SPLUNK_HOME/etc/apps/.

Version

App version.

Visible
 No Yes
Only apps with views should be made visible.

Author

Name of the app's owner.

Description

Enter a description for your app.

Template

These templates contain example views and searches.

Upload asset

Can be any html, js, or other file to add to your app.

5. You have just created your very first Splunk app. Notice that it now appears in the list of apps and it has a status of **Enabled**, meaning it is ready to be used:

Name	Folder name	Version	Update checking	Visible	Sharing	Status
SplunkForwarder	SplunkForwarder		Yes	No	App Permissions	Disabled Enable
SplunkLightForwarder	SplunkLightForwarder		Yes	No	App Permissions	Disabled Enable
Webhook Alert Action	alert_webhook	6.3.0	Yes	No	App Permissions	Enabled Disable
Apps Browser	appsbrowser	6.3.0	Yes	Yes	App Permissions	Enabled
Destinations	destinations	None	Yes	Yes	Global Permissions	Enabled Disable
framework	framework		Yes	No	App Permissions	Enabled Disable
Getting started	gettingstarted	1.0	Yes	Yes	App Permissions	Disabled Enable

We will use this bare bones app to complete the exercises in this book, but first we need to make a few important changes:

1. Click the **Permissions** link as show in the preceding screenshot.
2. In the next window, under the **Sharing for config file-only objects** section, select **All apps**.

These steps will ensure that the application will be accessible to the Eventgen add-on that will be installed later in the guide. Use the following screenshot as a guide:



Splunk permissions are always composed of three columns: **Roles**, **Read**, and **Write**. A role refers to certain authorizations or permissions that can be taken on by a user. Selecting **Read** for a particular role grants the set of users in the role permission to view the object. Selecting **Write** will allow the set of users to modify the object. In the preceding screenshot, everyone (all users) will have access to view the Destinations app, but only the admin (you) and a power user can modify it.

Populating data with Eventgen

Machine data is the information produced by the many functions carried out by computers and other mechanical machines. If you work in an environment that is rich in machine data, you will most likely have many sources of readily-available machine inputs for Splunk. However, to facilitate learning in this book, we will use a Splunk add-on called the **Splunk Eventgen** to easily build real-time and randomized web log data. This is the type of data that would be produced by a web-based e-commerce company.

NOTE

Here's an important tip. Make it a habit to always launch your command prompt in Administrator mode. This allows you to use commands that are unhindered by Windows security:

1. Right-click on the Windows Start menu icon and select **Search**. In Windows 7, you can click on the Windows icon and the search window will be directly above it. In Windows 10, there is a search bar named **Cortana** next to the Windows icon that you can type into. They both have the same underlying function.
2. In the search bar, type `cmd`.
3. In the search results, look for `command.exe` (Windows 7) or a command prompt (Windows 10), right-click on it, then select **Run as administrator**.

NOTE

Familiarize yourself with this step. Throughout the rest of the class, you will be frequently asked to open a command prompt in Administrator mode. You will know if you are in Administrator mode, as it will say Administrator: Command Prompt in the title of the command prompt window.

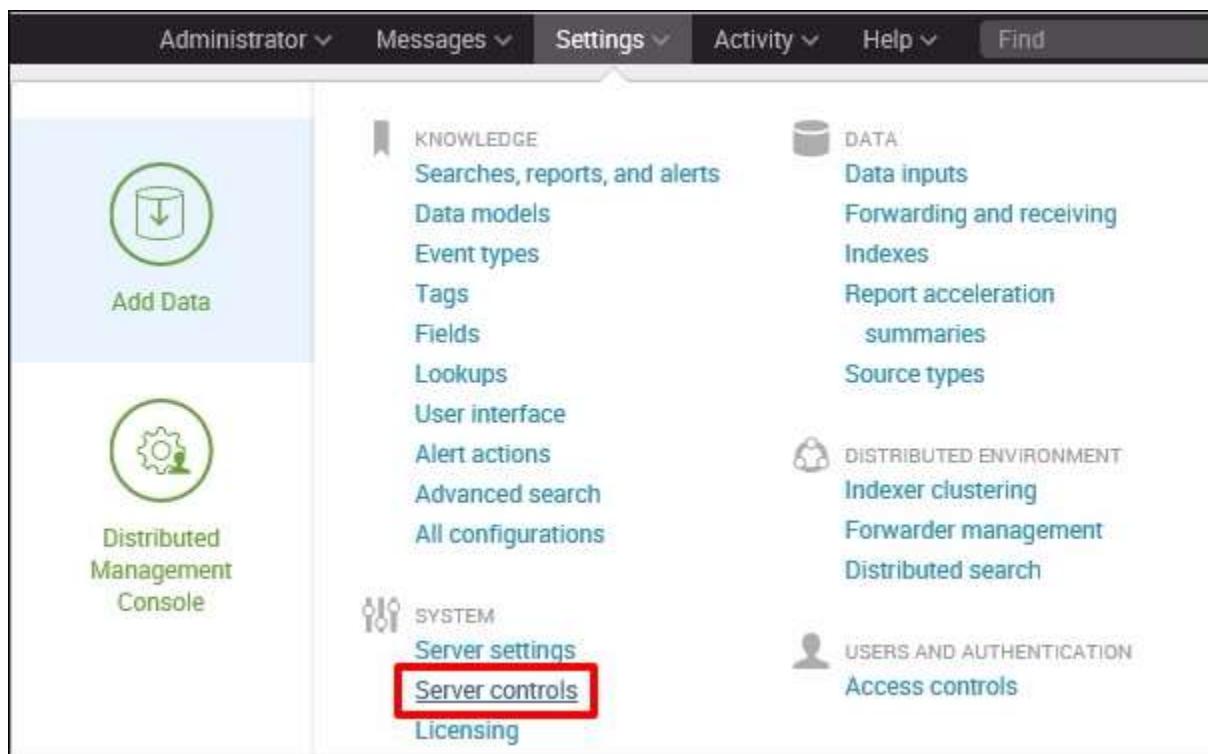
Installing an add-on

A Splunk add-on extends and enhances the base functionality of Splunk. They also typically enrich data from source for easier analysis. In this section, you will be installing your first add-on called **Splunk Eventgen** that will help us pre-populate Splunk with real-time simulated web data:

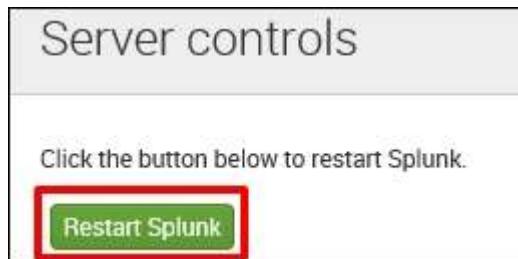
1. First we need to install the Eventgen add-on.
2. You may also download the ZIP file from the Eventgen's public repository, <http://github.com/splunk/eventgen>, and extract it onto your machine. The download ZIP button is in the lower-right corner of the GitHub repository page.



3. After extracting the ZIP file, copy the entire `eventgen` directory into the `$SPLUNK_HOME/etc/apps/` folder. You may need to rename it from `eventgen-develop` to `SA-EventGen` if you manually downloaded the ZIP file. The trailing slashes are important.
4. These are the contents of the recently-copied `SA-Eventgen` folder as shown in the following screenshot:
5. Restart Splunk by selecting the **Settings** dropdown, and under the **SYSTEM** section, click on **Server controls**:



6. On the **Server controls** page, click on the **Restart Splunk** button as shown in the following screenshot. Click **OK** when asked to confirm the restart:



7. The web interface will first notify you that Splunk is restarting in the background, then it will tell you that the restart has been successful. Every time Splunk is restarted, you will be prompted to log in with your credentials. Go ahead and log in.
8. Go to the **Manage Apps** page and confirm that the [SA-EventGen](#) application is installed:

The screenshot shows the Splunk Add-on Manager interface. At the top, there are three buttons: 'Browse more apps' (green), 'Install app from file' (grey), and 'Create app' (grey). Below these, it says 'Showing 1-17 of 17 items'. A table follows, with columns 'Name', 'Folder name', and 'Version'. There is one item listed: 'SA-Eventgen' under both 'Name' and 'Folder name', and '@build.version@' under 'Version'.

Name	Folder name	Version
SA-Eventgen	SA-Eventgen	@build.version@

Congratulate Yourself. You have successfully installed a Splunk add-on.

SPLUNK CLI Command Guide

There are several different ways to stop, start, or restart Splunk. The easiest way is to do it from the web interface, as demonstrated in the preceding section. The web interface, however, only allows you to restart your Splunk instance. It does not offer any other control options.

In Windows, you can also control Splunk through the **Splunkd Service** as shown in the following screenshot. The *d* in the service name, denoting *daemon*, means a background process. Note that the second service, **splunkweb**, is not running. Do not try to start **splunkweb** as it is deprecated and is only there for legacy purposes. The Splunk web application is now bundled in **Splunkd Service**:

Software Protection	Enables the ...	Automatic (D...	Network S...
Splunkd Service	Splunkd is t... Running	Automatic	Local Syste...
splunkweb (legacy purposes only)	The splunk...	Automatic	Local Syste...
Spot Verifier	Verifies pote...	Manual (Trig...	Local Syste...

The best way to control Splunk is by using the **command-line interface(CLI)**. It may require a little effort to do it, but using the CLI is an essential skill to learn. Remember to always use command prompts in Administrator mode.

In the console or command prompt, type in the following command and hit **Enter** on your keyboard:

```
C:\> cd \Splunk\bin
```

Here `cd` is a command that means *change directory*.

While in the `C:\Splunk\bin` directory, issue the following command to restart Splunk:

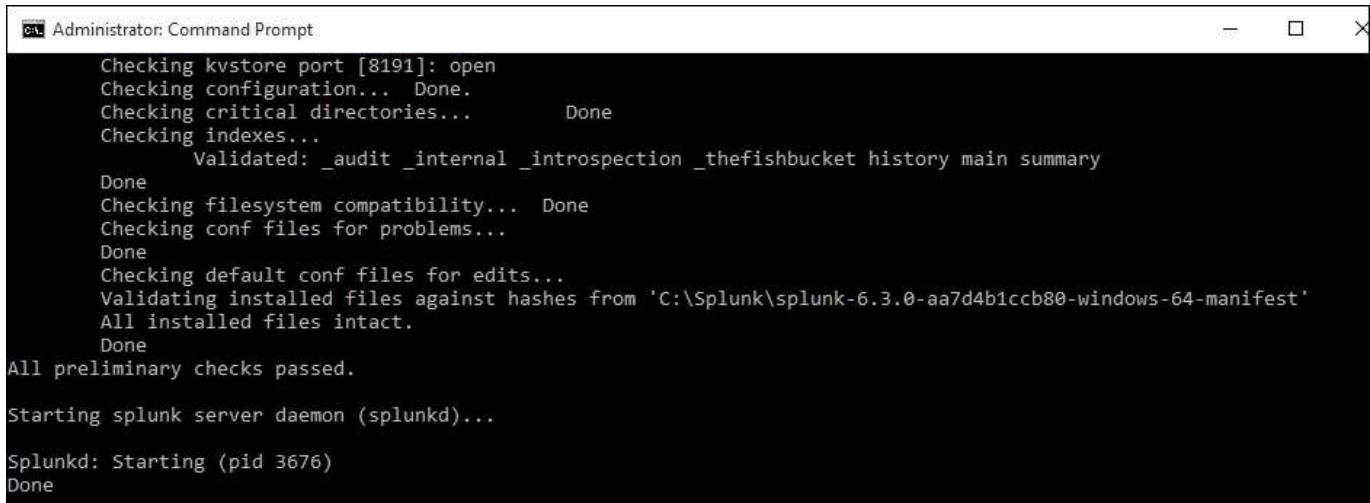
```
C:\> C:\Splunk\bin> splunk restart
```

After issuing this command, `splunkd` will go through its restart process. Here are the other basic parameters that you can pass to the Splunk application to control Splunk:

- `splunk status`: Tells you if splunkd is running or not
- `splunk stop`: Stops splunkd and all its processes
- `splunk start`: Starts splunkd and all its processes
- `splunk restart`: Restarts splunkd and all its processes

Doing this in the console gives the added benefit of verbose messages. A verbose message is a message with a lot of information in it. Such messages can be useful for making sure the system is working correctly or troubleshooting any errors.

A successful restart of splunkd has the following output (which may vary):



```
Administrator: Command Prompt
Checking kvstore port [8191]: open
Checking configuration... Done.
Checking critical directories... Done
Checking indexes...
    Validated: _audit _internal _introspection _thefishbucket history main summary
Done
Checking filesystem compatibility... Done
Checking conf files for problems...
Done
Checking default conf files for edits...
Validating installed files against hashes from 'C:\Splunk\splunk-6.3.0-aa7d4b1ccb80-windows-64-manifest'
All installed files intact.
Done
All preliminary checks passed.

Starting splunk server daemon (splunkd)...

Splunkd: Starting (pid 3676)
Done
```

Configuring Eventgen

We are almost there. Proceed by first downloading the exercise materials that will be used in this class. Open an Administrator command prompt and make sure you are in the root of the `c:` drive.

The Eventgen configuration you will need for the exercises in this book has been packaged and is ready to go. We are not going into the details of how to configure Eventgen. If you are interested in learning more about Eventgen, visit the project page at <http://github.com/splunk/eventgen>.

Follow these instructions to proceed:

1. Extract the project ZIP file into your local machine. Open an administrator console and CD into the directory where you extracted the file.
2. Create a new `samples` directory in the Destinations Splunk app. The path of this new directory will be `$SPLUNK_HOME/etc/apps/destinations/samples`:
`C:\> mkdir c:\splunk\etc\apps\destinations\samples`
3. Copy all the `*.sample` files from `/labs/chapter01/eventgen` of the extracted project directory into the newly-created `samples` directory. You can also copy and paste using the GUI if you prefer it:

```
C:\> copy splunk-essentials\labs\chapter01\eventgen\*.sample  
c:\Splunk\etc\apps\destinations\samples\
```

4. Now copy the `eventgen.conf` into the `$SPLUNK_HOME/etc/apps/destinations/local` directory. You can also copy and paste using the GUI if you prefer it:

```
C:\> copy splunk-  
essentials\labs\chapter01\eventgen\eventgen.conf  
c:\Splunk\etc\apps\destinations\local\
```

5. Grant the `SYSTEM` account full access permissions to the `eventgen.conf` file. This is a very important step. You can either do it using the following `icacls` command or change it using the Windows GUI:

```
6. C:\> icacls c:\Splunk\etc\apps\destinations\local\eventgen.conf
```

7. **/grant SYSTEM:F**

A successful output of this command will look like this:

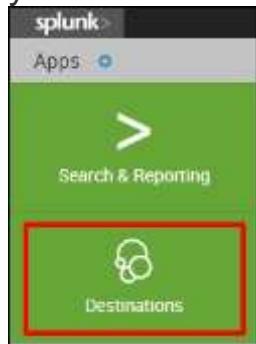
```
processed file:  
c:\Splunk\etc\apps\destinations\local\eventgen.conf  
Successfully processed 1 files; Failed processing 0 files
```

8. Restart Splunk.

Viewing the Destinations app

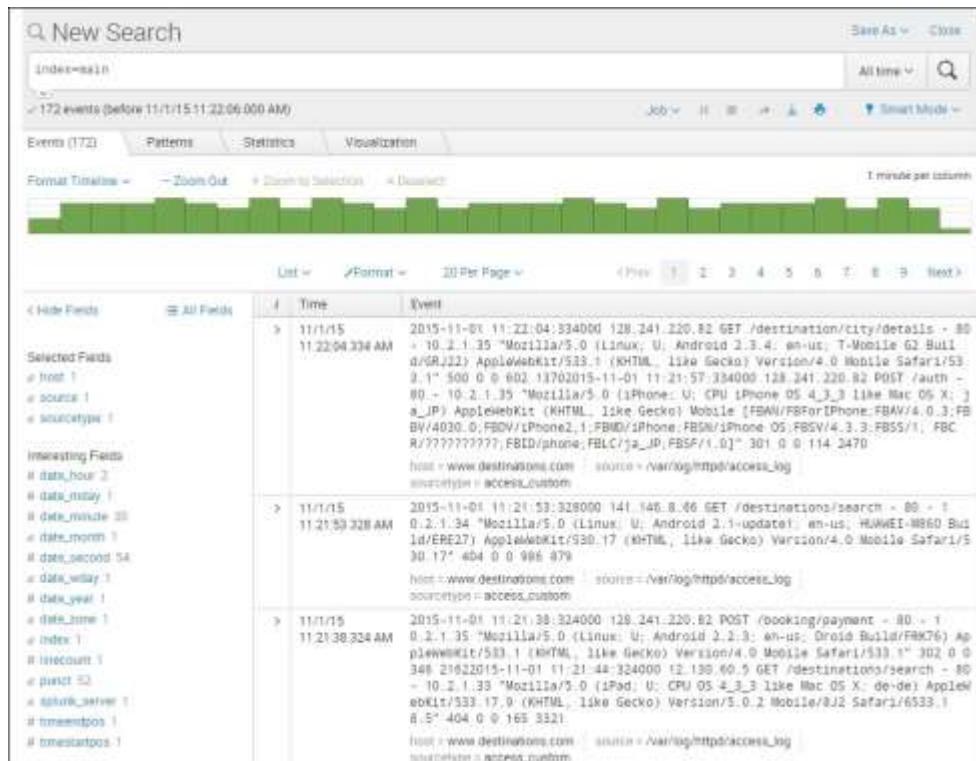
Next we will see our Destinations app in action! Remember that we have configured it to draw events from a prototype web company. That is what we did when we set it up to work with Eventgen. Now let's look at some of our data:

1. After a successful restart, log back in to Splunk and proceed to your new Destinations app:



2. In the **Search** field, type this search query and select **Enter**:

3. **SPL> index=main**



Examine the event data that your new app is enabling to come into Splunk. You will see a lot of references to browsers, systems, and so forth: the kinds of information that make a web-based e-commerce company run.

Try changing the time range to **Real-time (5 minute window)** to see the data flow in before your eyes:

The screenshot shows the Splunk search interface with a focus on the time range selection. At the top right, there is a dropdown menu labeled "All time" with a red box around it. Below this is a magnifying glass icon. The main area is titled "Presets" and contains three columns of time range options:

Real-time	Relative	Other
30 second window	Today	All time
1 minute window	Week to date	
5 minute window (highlighted with a red box)	Business week to date	
30 minute window	Month to date	
1 hour window	Year to date	
All time (real-time)	Yesterday	
	Last 15 minutes	
	Last 60 minutes	
	Last 4 hours	
	Last 24 hours	
	Last 7 days	
	Last 30 days	
	Previous week	
	Previous business week	
	Previous month	
	Previous year	

Congratulations! You now have real-time web log data that we can use in subsequent chapters.

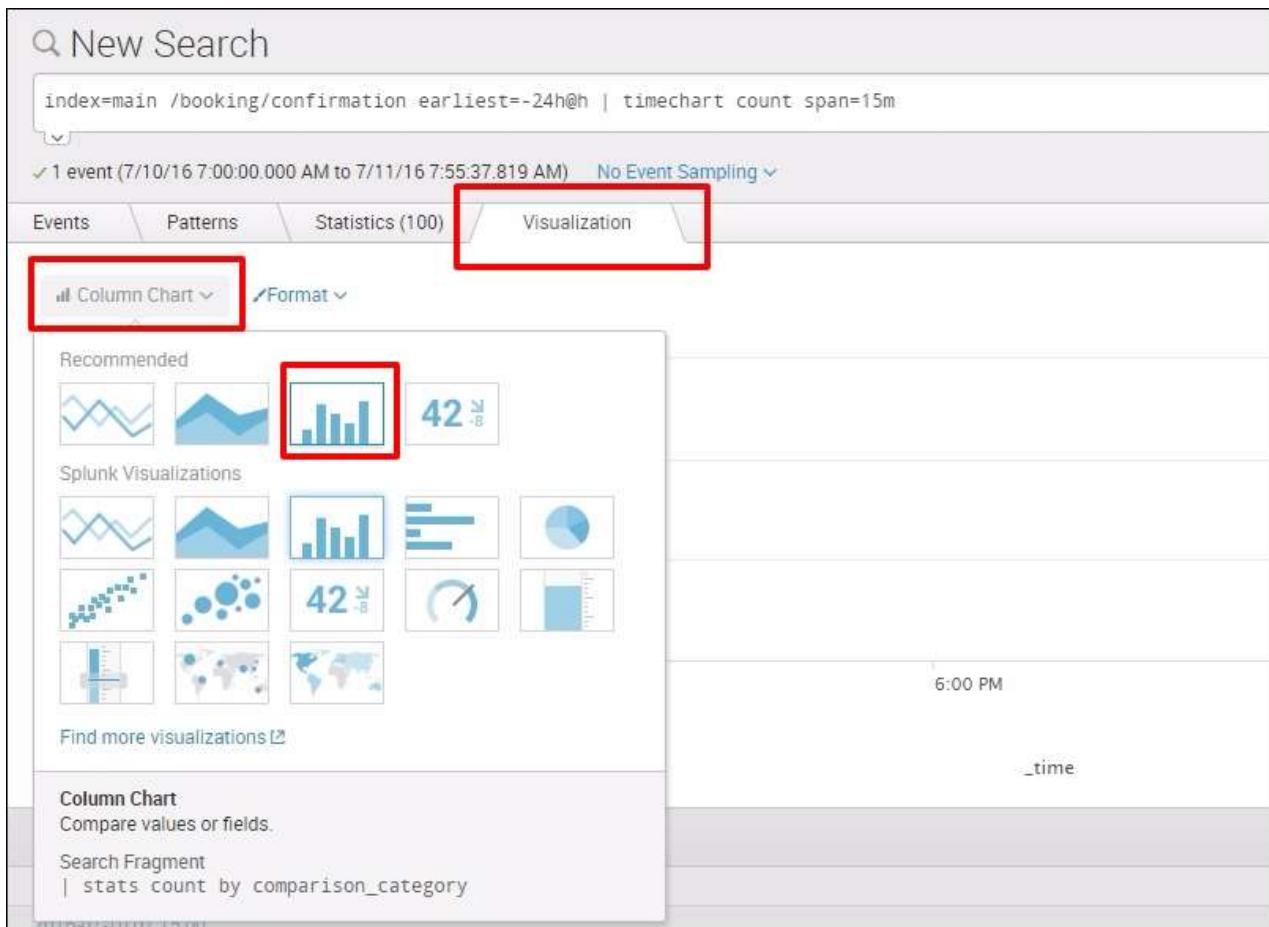
Creating your first dashboard

Now that we have data ingested, it is time to use it in order to derive something meaningful out of it. You are still in the Destinations app, correct? We will show you the basic routine when creating new dashboards and dashboard panels.

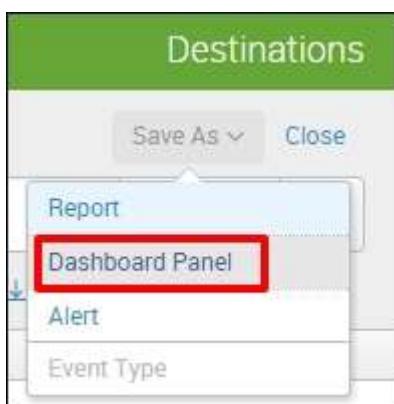
Copy and paste the following search query in the **Search Field**, then hit **Enter**:

```
SPL> index=main /booking/confirmation earliest=-24h@h |  
timechart  
count span=15m
```

After the search results render, click on the **Visualization** tab. This will switch your view into visualization so you can readily see how your data will look. By default, it should already be using the **Column Chart** as shown in the following screenshot. If it does not, then use the screenshot as a guide on how to set it:



Now that you can see your **Column Chart**, it is time to save it as a dashboard. Click on **Save As** in the upper-right corner of the page, then select **Dashboard Panel** as shown in the following screenshot:



Now let's fill up that dashboard panel information, as seen in the following screenshot. Make sure to select the **Shared in App** in the **Dashboard Permissions** section:

Save As Dashboard Panel ×

Dashboard	New	Existing
Dashboard Title	Bookings Dashboard	
Dashboard ID?	bookings_dashboard	
Can only contain letters, numbers and underscores.		
Dashboard Description	Bookings Dashboard	
Dashboard Permissions	Private	Shared in App
Panel Title	Confirmed Bookings Last 24 Hrs	
Panel Powered By	Q Inline Search	
Panel Content	Statistics	Column
Cancel		Save

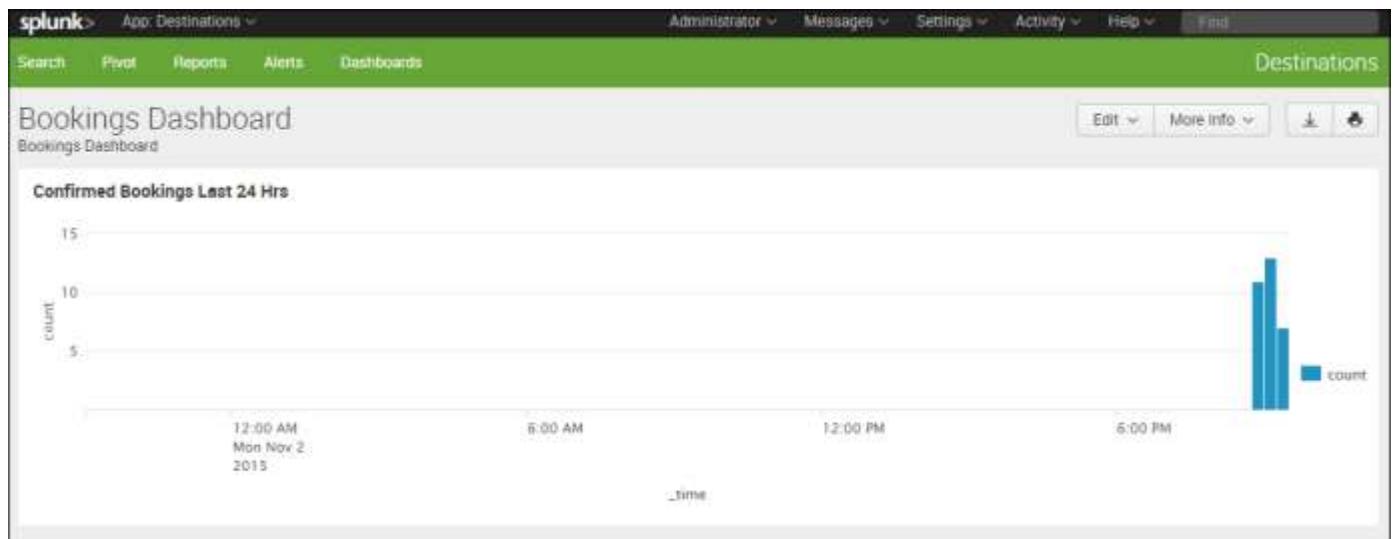
Finish up by clicking **View Dashboard** in the next prompt:

Your Dashboard Panel Has Been Created ×

The panel has been created and added to bookings_dashboard. You may now view the dashboard.

View Dashboard

You have created your very first Splunk dashboard with a panel that tells you the number of confirmed bookings in the last 24 hours at 15-minute intervals. Time to show it to your boss!



Take that well-deserved coffee break. You now have a fully-functional Splunk installation with live data. Leave Splunk running for 2 hours or so. After a few hours, you can stop Splunk if you need to rest for a bit to suppress indexing and restart it when you're ready to proceed into the next chapters. Do you recall how to control Splunk from the command line?

```
C:\> C:\Splunk\bin> splunk stop  
C:\Splunk\bin> splunk start
```

Session 2 Forwarder Data

Introduction to Forwarder Data Inputs

Getting data into Splunk can be a long process, and it is often a very important and overlooked process on a Splunk journey. If we do it poorly, there is lots of clean up that we have to do, which is usually much more complicated than just sitting down to plan out how we want our data to get to Splunk, and how we want it to look. This process is known as data scrubbing, or data cleaning. This is the process of breaking events at the proper line, as well as extracting some fields, or masking data before and after Splunk writes it to disk.

Topics that will be covered in this section:

- Heavy Forwarder management
 - What is a Heavy Forwarder?
 - Managing the deployment server
 - Installing modular inputs
- Data formatting
 - Event management
 - Knowledge management
- Pre/post indexing techniques to clean data before indexing
 - Pre-indexed field extractions
 - Data masking

We're going to focus on only a few pieces of a Splunk system in this guide:

- Universal Forwarders
 - Thin clients that can be installed on a device
- Heavy Forwarders
 - Thick clients that can be installed on a device.
- Indexers
 - The data store for Splunk

The Universal Forwarder is the most popular forwarding tier to install on a machine and begin receiving data.

Splunk Universal Forwarders

It is good to mention that on large-scale Splunk implementations, data gathering should, as much as possible, be done using these. Their usefulness lies in the fact that they are lightweight applications that can run on many different operating systems and can quickly and easily forward data to the Splunk indexer.

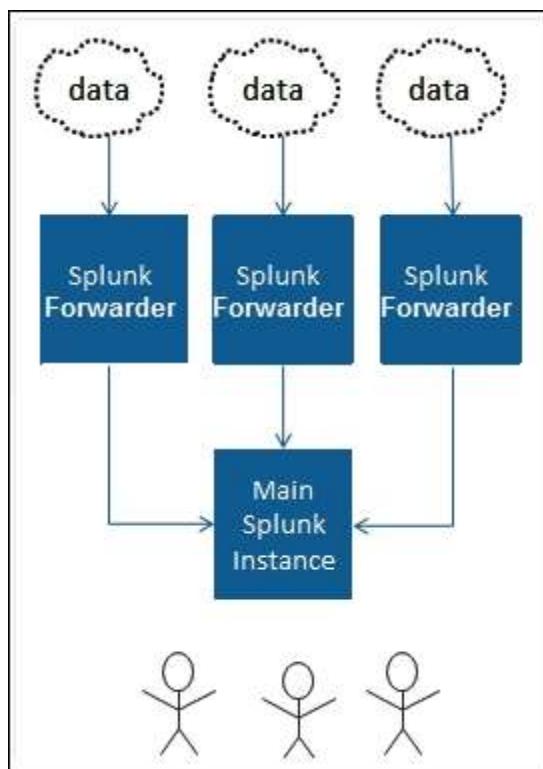
In production environments, with many different types of deployment and using many different machines, each machine where data resides will have a Universal Forwarder.

When the implementation is large and includes many different machines, Universal Forwarders can be managed using the forwarder manager.

These forwarders and the ability to manage them easily are one of the reasons for Splunk's growing popularity. Sizeable organizations find it much easier to be able to bring in, understand, and use their data for decision-making when they can use the capabilities of Splunk's Universal Forwarders. It is useful to note that the adjective Universal represents the fact that Splunk can be used with almost any type of data imaginable, thus multiplying the usefulness of these Universal Forwarders.

Leveraging your forwarders

Let's be forward here—what is a Splunk forwarder? A forwarder is an instance of Splunk that has a specific purpose, to input data and forward it to other instances of Splunk. In other words, forwarders have limited capabilities by design. Most forwarders don't include Splunk Web and don't have users logging in and running search pipelines; therefore, they require minimal resources and have little impact on performance. So, they can usually reside on the machines where the data originates. The following diagram gives you an idea of how you can configure Splunk using forwarders local to multiple data sources:



As an example, Splunk can be installed and configured as a forwarder on a number of individual servers that are all perhaps generating similar log data which you want to search centrally. You can then install Splunk on its own server where a user (or a user community) can perform searches. The forwarders on the data-generating servers can then be set up to take the data and send it to the main Splunk server, which then consolidates and indexes it and makes it available for searching. Because of their light footprint, the forwarders won't affect the performance of the source servers.

Splunk forwarders can handle exactly the same types of data and can consume this data in the same way as any Splunk instance, with one main difference: they do not index the data themselves. Instead, they input the data and refer it to a central Splunk instance, which will do the indexing and searching.

In a typical Splunk deployment, forwarders (which could number in hundreds or even thousands) will serve as the primary consumers of data. It's only in a single-machine deployment that the same Splunk instance will be the data consumer and indexer and will support user searches.

Forwarders are typically configured by editing Splunk's `inputs.conf` configuration file, either directly or using the CLI (which we discussed in a previous chapter of this book) or using Splunk Web on a test instance to configure the inputs, and then distributing the updated `inputs.conf` file to the forwarder instance.

Another method to do this is by deploying a Splunk app that contains the desired inputs.

Lab: Using the Universal Forwarder to gather data

Most IT environments today range from multiple servers in the closet of your office to hundreds of endpoint servers located in multiple geographically distributed data centers.

When the data we want to collect is not located directly on the server where Splunk is installed, the Splunk UF can be installed on your remote endpoint servers and used to forward data back to Splunk to be indexed.

The UF is similar to the Splunk server in that it has many of the same features, but it does not contain Splunk Web and doesn't come bundled with the Python executable and libraries. Additionally, the UF cannot process data in advance, such as performing line breaking and timestamp extraction.

This recipe will guide you through configuring the Splunk UF to forward data to a Splunk indexer and will show you how to set up the indexer to receive the data.

Getting ready

To step through this recipe, you will need a server with the Splunk UF installed but not configured. You will also need a running Splunk server. No other prerequisites are required.

TIP

To obtain the UF software, you need to go to http://www.splunk.com/en_us/download.html and register for an account if you do not already have one. Then, either download the software directly to your server or download it to your laptop or workstation and upload it to your server via a file-transfer process such as SFTP.

How to do it...

Follow the steps in the recipe to configure the Splunk Forwarder to forward data and the Splunk indexer to receive data:

1. On the server with the UF installed, open a command prompt if you are a Windows user or a terminal window if you are a Unix user.
2. Change to the `$SPLUNK_HOME/bin` directory, where `$SPLUNK_HOME` is the directory in which the Splunk Forwarder was installed.

For Unix, the default installation directory will be `/opt/splunkforwarder/bin`. For Windows, it will be `C:\Program Files\SplunkUniversalForwarder\bin`.

TIP

If using Windows, omit `./` in front of the Splunk command in the upcoming steps.

3. Start the Splunk Forwarder if not already started, using the following command:

```
./splunk start
```

4. Accept the license agreement.
5. Enable the UF to autostart, using the following command:

```
./splunk enable boot-start
```

6. Set the indexer that this UF will send its data to. Replace the host value with the value of the indexer as well as the username and password for the UF:

```
./splunk add forward-server <host>:9997 -auth  
<username>:<password>
```

The username and password to log in to the Forwarder (default is `admin:changeme`) is `<username>:<password>`.

TIP

Additional receiving indexers can be added in the same way by repeating the command in the previous step with a different indexer host or IP. Splunk will automatically load balance the forwarded data if more than one receiving indexer is specified in this manner. Port 9997 is the default Splunk TCP port and should only be changed if it cannot be used for some reason.

On the receiving Splunk indexer servers:

1. Log in to your receiving Splunk indexer server. From the home launcher, in the top right-hand corner click on the **Settings** menu item and then select the **Forwarding and receiving** link.

The screenshot shows the Splunk Settings menu interface. At the top, there are navigation links: Administrator, Messages, Settings (which is currently selected), Activity, Help, and Find. Below these, the main menu is organized into several categories:

- DATA**: Data inputs, Forwarding and receiving (circled in red), Indexes, Report acceleration, summaries, Source types.
- DISTRIBUTED ENVIRONMENT**: Indexer clustering, Forwarder management, Distributed search.
- SYSTEM**: Server settings, Server controls, Licensing.
- USERS AND AUTHENTICATION**: Access controls.

On the left side, there are two large green circular icons: "Add Data" (with a database icon) and "Distributed Management Console" (with a gear icon).

2. Click on the **Configure receiving** link.

This screenshot shows the "Receive data" configuration page. It has a header "Receive data" and a sub-instruction "Configure this instance to receive data forwarded from other instances." Below this is a prominent blue button labeled "Configure receiving", which is circled in red.

3. Click on **New**.

4. Enter **9997** in the **Listen on this port** field.

This screenshot shows the "Configure receiving" setup page. It has a header "Configure receiving" and a sub-instruction "Set up this Splunk instance to receive data from forwarder(s.)". Below this is a form field labeled "Listen on this port *". Inside the field, the value "9997" is entered. A note below the field states: "For example, 9997 will receive data on TCP port 9997."

5. Click on **Save** and restart Splunk. The UF is installed and configured to send data to your Splunk server, and the Splunk server is configured to receive data on the default Splunk TCP port 9997.

How it works...

When you tell the forwarder which server to send data to, you basically add a new configuration stanza into an `outputs.conf` file behind the scenes. On the Splunk server, an `inputs.conf` file will contain a `[splunktcp]` stanza to enable receiving. The `outputs.conf` file on the Splunk forwarder will be located in `$SPLUNK_HOME/etc/system/local`, and the `inputs.conf` file on the Splunk server will be located in the local directory of the app you were in (the launcher app in this case) when configuring receiving.

Using forwarders to collect and forward data has many advantages. The forwarders communicate with the indexers on TCP port 9997 by default, which makes for a very simple set of firewall rules that need to be opened. Forwarders can also be configured to load balance their data across multiple indexers, increasing search speeds and availability. Additionally, forwarders can be configured to queue the data they collect if communication with the indexers is lost. This can be extremely important when collecting data that is not read from logfiles, such as performance counters or syslog streams, as the data cannot be re-read.

There's more...

While configuring the settings of the UF can be performed via the command-line interface of Splunk, as outlined in this recipe, there are several other methods to update the settings quickly and allow for customization of the many configuration options that Splunk provides.

ADD THE RECEIVING INDEXER VIA OUTPUTS.CONF

The receiving indexers can be directly added to the `outputs.conf` configuration file on the UF.

Edit `$SPLUNK_HOME/etc/system/local/outputs.conf`, add your input, and then, restart the UF. The following example configuration is provided, where two receiving indexers are specified. The `[tcpout-server]` stanza can be

leveraged to add output configurations specific to an individual receiving indexer:

```
[tcpout]
defaultGroup = default-autolb-group

[tcpout:default-autolb-group]
disabled = false
server = mysplunkindexer1:9997,mysplunkindexer2:9997

[tcpout-server://mysplunkindexer1:9997]
[tcpout-server://mysplunkindexer2:9997]
```

TIP

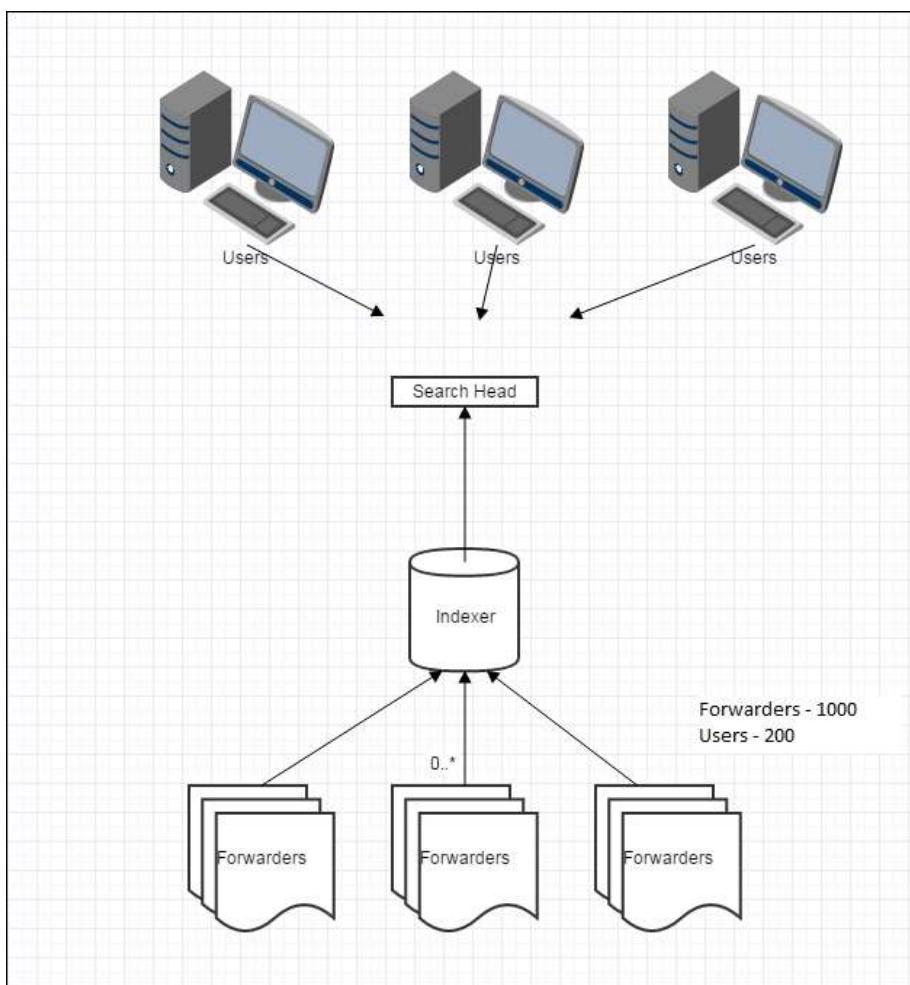
If nothing has been configured in `inputs.conf` on the UF, but `outputs.conf` is configured with at least one valid receiving indexer, the Splunk forwarder will only send internal forwarder health-related data to the indexer. It is, therefore, possible to configure a forwarder correctly and be detected by the Splunk indexers, but not actually send any real data.

Heavy Forwarder management

A Heavy Forwarder is usually used in instances where an administrator needs some more functionality, such as event routing (errors go to the error index, and info messages go to the info index), data routing, or collecting API information to be sent to the indexers. Heavy Forwarders can also be used for advanced, detailed filtering of data to reduce indexing volume.

Indexers are the heart of a Splunk system, and you can think of them as a big database. They will almost always be the beefiest of your machines, because they are doing the lion's share of the heavy lifting.

This is a diagram depicting a typical Splunk deployment:



Within a Splunk deployment a new admin often asks the question: *When would you ever need a Heavy Forwarder?*

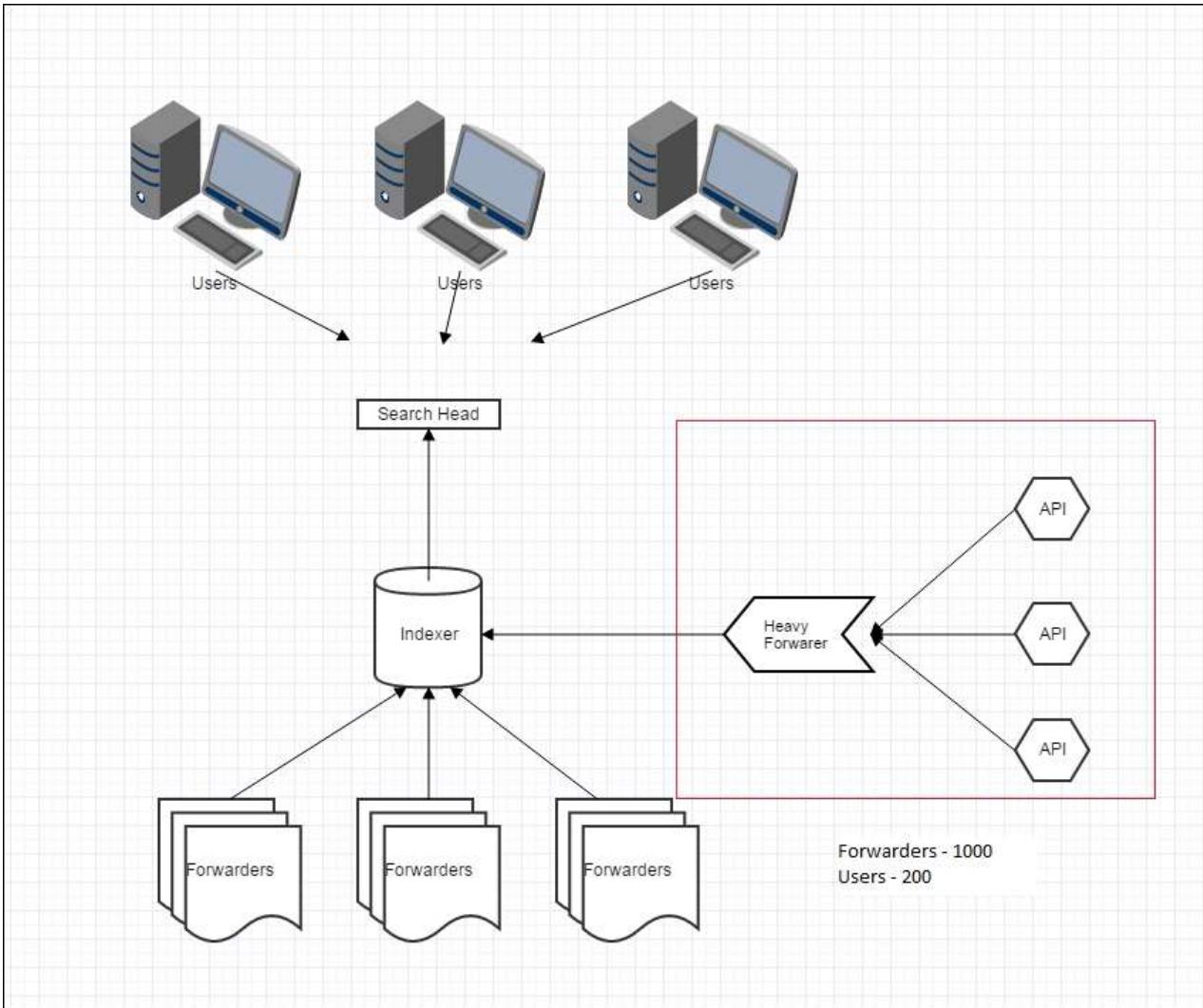
The answer to this question is complicated; however, I'll give a single use case where it is good to use a Heavy Forwarder.

A Heavy Forwarder should be deployed in circumstances in which data needs to be parsed, re-routed, consolidated, or *cooked* prior to delivering it to the indexer instead of merely forwarding raw data. Using a Heavy Forwarder can reduce the processing load on Splunk indexers. It can also perform more granular event filtering, reducing the overall data ingest. Finally, using a Heavy Forwarder to cook data allows us to transform inbound data, performing actions such as the obfuscation of fields before that data hits an indexer.

If a Splunk deployment looks like the preceding diagram, then the single **Indexer** is doing a lot of work. It's writing data to a disk rapidly, as well as answering search queries, and the **Search Head** is also under a lot of stress from all the **Users** trying to query things.

If someone wants to add API inputs to this architecture, it would be advisable to introduce a Heavy Forwarder to this architecture.

This is so you don't overwork either your **Indexer** or your **Search Head** by adding the API input to what they are doing, and reducing the user experience:



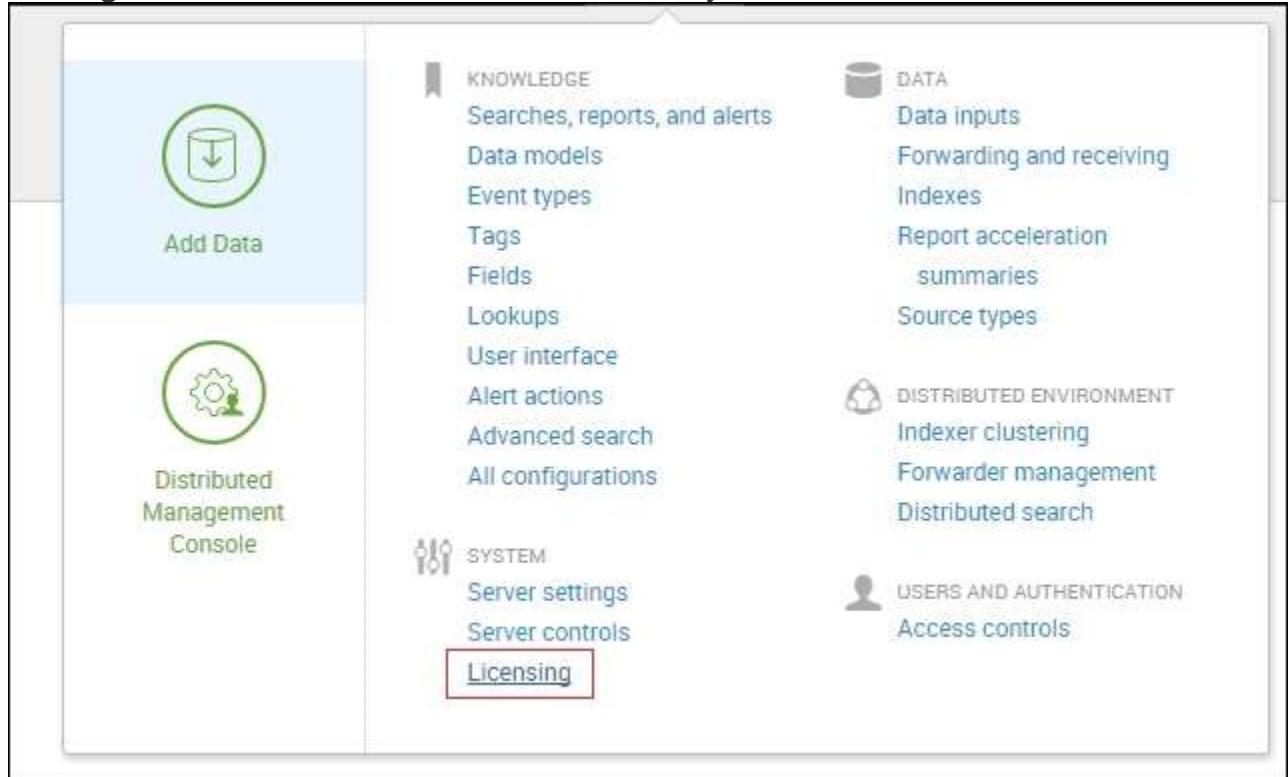
Adding the Heavy Forwarder to your configuration will take the load off querying the API and cooking the data, so your search head doesn't have to do the queries and the indexer doesn't have to cook the data.

It is also used as a consolidation point for data collection in a lot of architecture; that is, the Heavy Forwarder collects data for active directories, exchange networks, API's, and other sources, and cooks the data and sends it to the indexers.

The next question that is often asked is: *How do I install a Heavy Forwarder?*

Here are the steps (it's really easy):

1. Get a new machine.
2. Go to <http://www.splunk.com/>.
3. Download Splunk (the full version), and install it on your new machine.
4. Set the license from trial to forwarder.
5. Configure its `outputs.conf` to send data to your indexers:



6. First go to **Settings | Licensing**:

This server is acting as a standalone license server [Change to slave](#)

Trial license group [Change license group](#)

This server is configured to use licenses from the Trial license group

[Add license](#) [Usage report](#)

7. Then change the license group to **Forwarder license**:

Change license group

The type of license group determines what sorts of licenses can be used in the pools on this license server.

[Learn more](#)

Enterprise license

This license adds support for multi-user and distributed deployments, alerting, role-based security, single sign-on, scheduled PDF delivery, and unlimited data volumes.

There are no valid Splunk *Enterprise licenses* installed. You will be prompted to install a license if you choose this option.

Forwarder license

Use this group when configuring Splunk as a forwarder. [Learn more](#)

Free license

Use this group when you are running Splunk Free. This license has a 500MB/day daily indexing volume. [Learn more](#)

Enterprise Trial license

This is your included download trial. **IMPORTANT:** If you switch to another license, you cannot return to the Trial. You must install an Enterprise license or switch to Splunk Free.

[Cancel](#) [Save](#)

NOTE

When you do this, you will no longer have to use a username to login to this machine. If you need authentication, you will have to use an enterprise license.

8. Then go into **Forwarding defaults** and make sure you check **No** on the **Store a local copy of forwarded events?** section.
9. Go to **Forwarding and receiving**:

Forwarding and receiving

Forward data

Set up forwarding between two or more Splunk instances.

Forwarding defaults

[Configure forwarding](#)

10. Then click on **Forwarding defaults** and check **No** on **Store a local copy of forwarded events?**

Store a local copy of forwarded events?

Yes No

This saves a copy of all indexed data on this Splunk instance and forwards copies to other instances.

[Cancel](#) [Save](#)

And like magic, you have a Heavy Forwarder.

Managing your Heavy Forwarder

There are two ways to manage a Heavy Forwarder:

- Manual administration
- Deployment server

MANUAL ADMINISTRATION

Manual administration simply needs more effort, and is much less scalable. If you only have one Heavy Forwarder, then this may be the choice for you, but when you spin up four or five Heavy Forwarders it's a lot of effort to configure and manage them manually.

DEPLOYMENT SERVER

The Splunk deployment server is usually the preferred way to manage a large scale Splunk Forwarder deployment, as it is a central point of

administration for both Universal and Heavy Forwarders. Setting this up is very easy.

In the file `deploymentclient.conf`, set the following settings:

```
[deployment-client]
disabled = 0

[target-broker:deploymentServer]
targetUri = <deploymentserver>:<managementport>
```

Then restart the Splunk service, and check your deployment server list of Forwarders for the machine you installed it on.

Important configuration files

You will need to create a new app for your Heavy Forwarder in the `deployment-apps` directory of your deployment server, and then a couple of files:

- `outputs.conf`: Tells the Forwarder which indexers to connect to
- `props.conf`: This is where configurations for extractions and so on go
- `transforms.conf`: This is where we configure our masking (if needed)

Then after we create these files and configure our outputs, we simply add the new app, whitelist the hostname in `serverclass.conf` of the deployment server in `etc/system/local`, and reload the appropriate server class to distribute the app.

After you do this, make sure that you see the package deployed within the deployment server's UI to that host:

Session 3 – Getting Data In

Frank Anaya

Bringing in Data

The process of collecting data with Splunk is enhanced, as its system makes it easy to get data from many types of computerized systems, which are responsible for much of the data produced today. Such data is frequently referred to as machine data. And since much of this is streaming data, Splunk is especially useful, as it can handle streaming data quickly and efficiently. Additionally, Splunk can collect data from many other sources.

In this session, you will learn about Splunk and its role in big data, as well as the most common methods of ingesting data into Splunk. The chapter will also introduce essential concepts such as forwarders, indexes, events, event types, fields, sources, and source types. It is paramount that you learn this early on as it will empower you to gather more data. In this session we will cover the following topics:

- Splunk and big data
- Splunk data sources
- Splunk indexes
- Inputting data into Splunk
- Splunk events and fields

Splunk and big data

Splunk is useful for datasets of all types, and it allows you to use big data tools on datasets of all sizes. But with the recent focus on big data, its usefulness becomes even more apparent. Big data is a term used everywhere these days, but one that few people understand. In this part of the chapter, we will discuss aspects of big data and the terms that describe those aspects.

Streaming data

Much of the data that is large and comes quickly does not need to be kept. For instance, consider a mechanical plant; there can be many sensors that collect data on all parts of the assembly line. The

significance of this data is primarily to be able to alert someone to a possible upcoming problem (through noticing a bad trend) or to a current problem (by drawing attention to a metric that has exceeded some designated level); and much of it does not need to be kept for a long period of time. Often this type of data loses its importance once its timeliness expires and its main usefulness may just be in providing a sample measurement that can be used for historical records. Fast-moving data such as this is called streaming data, and Splunk, with its ability to create alerts, allows organizations to use this data to make sure they prevent, or act quickly on, problems that can occur.

Latency of data

The term **latency**, in regards to data, refers to the delay in how speedily it is entered into the system for analysis. Splunk is able to analyze data in real time with no latency issues when deployed on hardware that is sufficient to handle the indexing and searching workload. For example, if an alert goes off, a system can be immediately shut down if there is no latency in the data. If a denial of a service attack (a cyberattack that can dramatically hurt an e-commerce company's bottom line) is taking place, Splunk can be quickly used to figure out what is happening almost immediately.

Sparseness of data

Splunk is also excellent for dealing with sparse data. Much data in retailing environments is considered sparse. Consider a store that has many products but where most people just buy a few of them on any given shopping trip. If the store's database has fields specifying how many items of a particular type have been purchased by each customer, most of the fields would be empty if the time interval under consideration was short. We would say then that the data is sparse. In Splunk, the sparseness of data in a search ranges from dense (meaning that a result is obtained 10 percent of the time or more) to sparse (from 0.01 to 1 percent of the time). This can also extend to super sparse, or, for a better definition, trying to find a needle in a haystack (which is less than 0.01 percent), and even to rare, which is just a handful of cases.

Splunk data sources

Splunk was invented as a way to keep track of and analyze machine data coming from a variety of computerized systems. It is a powerful platform for doing just that. But since its invention, it has been used for a myriad of different data types, including machine data, log data (which is a type of machine data), and social media data. The various types of data that Splunk is often used for are explained in the next few sections.

Machine data

As mentioned previously, much of Splunk's data is machine data. Machine data is data created each time a machine does something, even if it is as seemingly insignificant as a tick on a clock. Each tick has information about its exact time (down to the second) and source, and each of these becomes a field associated with the event (the tick). The term "machine data" can be used in reference to a wide variety of data coming from computerized machines, from servers to operating systems to controllers for robotic assembly arms. Almost all machine data includes the time it was created or when the actual event took place. If no timestamp is included, then Splunk will need to find a date in the source name or filename based on the file's last modification time. As a last resort, it will stamp the event with the time it was indexed into Splunk.

Web logs

Web logs are invaluable sources of information for anyone interested in learning about how their website is used. Deep analysis of web logs can answer questions about which pages are visited most, which pages have problems (people leaving quickly, discarded shopping carts, and other aborted actions), and many others. Google, in early 2014, was registering as many as 20 billion websites each day; you can find more information about this at http://www.roche.com/media/roche_stories/roche-stories-2014-01-22.htm.

Data files

Splunk can read in data from basically all types of files containing clear data, or as they put it, any data. Splunk can also decompress the following types of files: `tar`, `gz`, `bz2`, `tar.gz`, `tgz`, `tbz`, `tbz2`, `zip`, and `z`, along with many others.

Social media data

An enormous amount of data is produced by social media every second. Consider the fact that 1.13 billion people (<https://zephoria.com/top-15-valuable-facebook-statistics/>) login to Facebook each day and they spend, on average, 20 minutes at a time interacting with the site. Any Facebook (or any other social media) interaction creates a significant amount of data, even those that don't include the more data-intensive acts, such as posting a picture, audio file, or a video. Other social media sources of data include popular sites such as Twitter, LinkedIn, Pinterest, and Google+ in the U.S., and QZone, WeChat, and Weibo in China. As a result of the increasing number of social media sites, the volume of social media data created continues to grow dramatically each year.

Other data types

Almost any type of data works in Splunk. Some of these types include scripted inputs and modular inputs. Sometimes you may want to include a script that sets data up so that it is indexed the way you want. Or you may want to include data coming from a source that is unusual in some way, and you want to make sure that the fields are set up the way they should be. For these reasons, it is nice to know that you can use Python scripts, Windows batch files, shell scripts, and other utilities to make sure your inputs are formatted correctly. You will see the other data types listed when we add data to Splunk shortly.

Splunk Config Files - Introduction

If you have spent any length of time in the filesystem investigating Splunk, you must have seen many different files ending in .conf. In this section, we will give you a quick overview of the most common .conf files. The official documentation is the best place to look for a complete reference to files and attributes.

The quickest way to find the official documentation is with your favorite search engine by searching for `splunk filename.conf`. For example, a search for `splunk props.conf` pulled up (and will pull up) the Splunk documentation for `props.conf` first in every search engine I tested.

Everything that controls Splunk lives in its configuration files sitting in the filesystem of each instance of Splunk. These files are unencrypted, easily readable, and easily editable. Almost all of the work that we have done so far has been accomplished through the web interface, but everything actually ends up in these configuration files.

While the web interface does a lot, there are many options that are not represented in the admin interface. There are also some things that are simply easier to accomplish by editing the files directly.

In this section, we will cover the following topics:

- Locating configuration files
- Merging configurations
- Debugging configurations
- Common configurations and their parameters

Locating Splunk configuration files

Splunk's configuration files live in `$SPLUNK_HOME/etc`. This is reminiscent of Unix's `/etc` directory but is instead contained within Splunk's directory structure.

This has the advantage that the files don't have to be owned by root. In fact, the entire Splunk installation can run as an unprivileged user (assuming you don't need to open a port below 1024 or read files only readable by another user).

The directories that contain configurations are as follows:

- **\$SPLUNK_HOME/etc/system/default**: These are the default configuration files that ship with Splunk. Never edit these files as they will be overwritten each time you upgrade.
- **\$SPLUNK_HOME/etc/system/local**: This is the location of the global configuration overrides specific to this host. There are very few configurations that need to live here—most configurations that do live here are created by Splunk itself. In almost all cases, you should make your configuration files inside an app.
- **\$SPLUNK_HOME/etc/apps/\$app_name/default**: This is the proper location for configurations in an app that will be shared either through Splunkbase or otherwise.
- **\$SPLUNK_HOME/etc/apps/\$app_name/local**: This is where most configurations should live and where all the nonprivate configurations created through the web interface will be placed.
- **\$SPLUNK_HOME/etc/users/\$user_name/\$app_name/local**: When a search configuration is created through the web interface, it will have a permission setting of *Private* and will be created in a user- or app-specific configuration file. Once permissions are changed, the configuration will move to the corresponding directory named `$app_name/local`.

There are a few more directories that contain files that are not `.conf` files. We'll talk about those later in this section, under the *User interface resources* in this session.

The structure of a Splunk configuration file

The `.conf` files used by Splunk look very similar to `.ini` files. A simple configuration looks as follows:

```
#settings for foo
[foo]
bar=1
la = 2
```

Let's look at the following couple of definitions:

- **stanza:** A stanza is used to group attributes. Our stanza in this example is `[foo]`. A common synonym for this is "section". Keep in mind the following key points:
 - A stanza name must be unique in a single file
 - The order does not matter
- **attribute:** An attribute is a name-value pair. Our attributes in this example are `bar` and `la`. A common synonym is parameter. Keep in mind the following key points:
 - The attribute name must not contain a whitespace or the equals sign.
 - Each attribute belongs to the stanza defined previously; if the attribute appears before all stanzas, the attribute belongs to the stanza `[default]`.
 - The attribute name must be unique in a single stanza but not in a configuration.
 - Each attribute must have its own line and can only use one line. Spaces around the equals sign do not matter.

These are a few rules that may not apply in other implementations:

- Stanza and property names are case sensitive
- The comment character is `#`
- Bare attributes at the top of a file are added to the `[default]` stanza
- Any attributes in the `[default]` stanza are added to all stanzas that do not have an attribute with that name already

The configuration merging logic

Configurations in different locations merge behind the scenes into one super configuration. Luckily, the merging happens in a predictable way, is fairly easy to learn, and there is a tool to help us preview this merging.

The merging order

The merging order is slightly different depending on whether the configuration is being used by the search engine or another part of Splunk. The difference is whether there is an active user and app.

THE MERGING ORDER OUTSIDE OF SEARCH

Configurations being used outside of search are merged in a fairly simple order. These configurations include the files to read, the indexed fields to create, the indexes that exist, deployment server and client configurations, and other settings. These configurations merge in this order:

1. `$SPLUNK_HOME/etc/system/default`: This directory contains the base configurations that ship with Splunk.

Never make changes in `$SPLUNK_HOME/etc/system/default` as your changes will be lost when you upgrade Splunk.
2. `$SPLUNK_HOME/etc/apps/*/default`: Configurations are overlaid in the reverse ASCII order by app directory name, that is, *a beats z*.
3. `$SPLUNK_HOME/etc/apps/*/local`
4. `$SPLUNK_HOME/etc/system/local`
 - The configurations in this directory are applied last.
 - Outside of search, these configurations cannot be overridden by an app configuration. Apps are a very convenient way to compartmentalize control and distribute configurations. This is particularly relevant if you use the deployment server, which we will cover in [Chapter 12, Advanced Deployments](#).

Don't edit configurations in `$SPLUNK_HOME/etc/system/local`—even if you have a very specific reason. An app is almost always the correct place for configuration.

A little pseudo code to describe this process might look as follows:

```
$conf = new Configuration('$SPLUNK_HOME/etc/')

$conf.merge( 'system/default/$conf_name' )
for $this_app in reverse(sort(@all_apps)):
    $conf.merge( 'apps/$this_app/default/$conf_name' )
    for $this_app in reverse(sort(@all_apps)):
        $conf.merge( 'apps/$this_app/local/$conf_name' )
$conf.merge( 'system/local/$conf_name' )
```

THE MERGING ORDER WHEN SEARCHING

When you are searching, configuration merging is slightly more complicated. When you are running a search, there is always an active user and app, and they come into play. The logical order looks like this:

1. `$SPLUNK_HOME/etc/system/default`
2. `$SPLUNK_HOME/etc/system/local`
3. `$SPLUNK_HOME/etc/apps/not app`
 - Each app, other than the current app, is looped through in the ASCII order of the directory name (not the visible app name). Unlike merging outside of search, here *z beats a*.
 - All configuration attributes that are shared globally are applied, first from default and then from local.
4. `$SPLUNK_HOME/etc/apps/app`

All configurations from default and then local are merged.
5. `$SPLUNK_HOME/etc/users/user/app/local`

Maybe a little pseudo code would make this clearer:

```
$conf = new Configuration('$SPLUNK_HOME/etc/')

$conf.merge( 'system/default/$conf_name' )
$conf.merge( 'system/local/$conf_name' )
for $this_app in sort(@all_apps):
```

```
if $this_app != $current_app:  
    $conf.merge_shared( 'apps/$this_app/default/$conf_name' )  
    $conf.merge_shared( 'apps/$this_app/local/$conf_name' )  
    $conf.merge( 'apps/$current_app/default/$conf_name' )  
    $conf.merge( 'apps/$current_app/local/$conf_name' )  
    $conf.merge(  
        'users/$current_user/$current_app/local/$conf_name' )
```

The configuration merging logic

Now that we know which configurations will merge in what order, let's cover the logic for how they actually merge. The logic is fairly simple:

- The configuration name, stanza name, and attribute name must match exactly
- The last configuration added wins

The best way to understand configuration merging is through examples.

CONFIGURATION MERGING – EXAMPLE 1

Say we have the base configuration `default/sample1.conf`:

```
[foo]  
bar=10  
la=20
```

And, say we merge a second configuration, `local/sample1.conf`:

```
[foo]  
bar=15
```

The resulting configuration would be as follows:

```
[foo]  
bar=15  
la=20
```

The things to notice here are as follows:

- The second configuration does not simply replace the prior configuration
- The value of `bar` is taken from the second configuration
- The lack of a `la` property in the second configuration does not remove the value from the final configuration

CONFIGURATION MERGING – EXAMPLE 2

Say we have the `default/sample2.conf` base configuration:

```
[foo]
bar = 10
la=20
[pets]
cat = red
Dog=rex
```

And say we merge a second configuration, `local/sample2.conf`:

```
[pets]
cat=blue
dog=fido
fish = bubbles
[foo]
bar= 15
[cars]
ferrari =0
```

The resulting configuration would be as follows:

```
[foo]
bar=15
la=20
[pets]
cat=blue
dog=rex
Dog=fido
```

```
fish=bubbles
[cars]
ferrari=0
```

The things to notice in this example are as follows:

- The order of the stanzas does not matter
- The spaces around the equals sign do not matter
- `Dog` does not override dog as all stanza names and property names are case sensitive
- The `cars` stanza is added fully

CONFIGURATION MERGING – EXAMPLE 3

Let's do a little exercise, merging four configurations from different locations.

In this case, we are not in search, so we will use the rules from the *Merging order outside of search* section.

Let's walk through a few sample configurations:

For `$SPLUNK_HOME/etc/apps/d/default/props.conf`, we have the following code:

```
[web_access]
MAX_TIMESTAMP_LOOKAHEAD = 25
TIME_PREFIX = ^\[[
[source::*.log]
BREAK_ONLY_BEFORE_DATE = true
```

For `$SPLUNK_HOME/etc/system/local/props.conf`, we have the following code:

```
BREAK_ONLY_BEFORE_DATE = false
[web_access]
TZ = CST
```

For `$SPLUNK_HOME/etc/apps/d/local/props.conf`, we have the following code:

```
[web_access]
```

```
TZ = UTC  
[security_log]  
EXTRACT-<name> = \[(?P<user>.*?)\]
```

For `$SPLUNK_HOME/etc/apps/b/default/props.conf`, we have the following code:

```
[web_access]  
MAX_TIMESTAMP_LOOKAHEAD = 20  
TIME_FORMAT = %Y-%m-%d $H:%M:%S  
[source::*/access.log]  
BREAK_ONLY_BEFORE_DATE = false
```

I've thrown a bit of a curveball here by placing the files out of merging order.

These configurations would actually merge in this order:

```
$SPLUNK_HOME/etc/apps/d/default/props.conf  
$SPLUNK_HOME/etc/apps/b/default/props.conf  
$SPLUNK_HOME/etc/apps/d/local/props.conf  
$SPLUNK_HOME/etc/system/local/props.conf
```

Walking through each merge, the configuration would look as follows:

1. We will start with `$SPLUNK_HOME/etc/apps/d/default/props.conf`:

```
[web_access]  
MAX_TIMESTAMP_LOOKAHEAD = 25  
TIME_PREFIX = ^\[  
[source::*.log]  
BREAK_ONLY_BEFORE_DATE = true
```

2. We will then merge `$SPLUNK_HOME/etc/apps/b/default/props.conf`:

```
[web_access]  
MAX_TIMESTAMP_LOOKAHEAD = 30  
TIME_PREFIX = ^\[  
TIME_FORMAT = %Y-%m-%d $H:%M:%S  
[source::*.log]  
BREAK_ONLY_BEFORE_DATE = true  
[source::*/access.log]
```

```
BREAK_ONLY_BEFORE_DATE = false
```

Even though `[source::*.log]` and `[source::*/access.log]` both match a file called `access.log`, they will not merge in the configuration because the stanza names do not match exactly. This logic is covered later in the *Stanza types* section under the `props.conf` heading.

3. Then, we will merge `$SPLUNK_HOME/etc/apps/d/local/props.conf`:

```
[web_access]
MAX_TIMESTAMP_LOOKAHEAD = 30
TIME_PREFIX = ^\[
TIME_FORMAT = %Y-%m-%d $H:%M:%S
TZ = UTC
[source::*.log]
BREAK_ONLY_BEFORE_DATE = true
[source::*/access.log]
BREAK_ONLY_BEFORE_DATE = false
[security_log]
EXTRACT-<name> = \[(?P<user>.*?)\]
```

4. We will finally merge the globally overriding `$SPLUNK_HOME/etc/system/` configuration:

```
local/props.conf file:
[default]
BREAK_ONLY_BEFORE_DATE = false
[web_access]
MAX_TIMESTAMP_LOOKAHEAD = 25
TIME_PREFIX = ^\[
TIME_FORMAT = %Y-%m-%d $H:%M:%S
TZ = CST
BREAK_ONLY_BEFORE_DATE = false
[source::*.log]
BREAK_ONLY_BEFORE_DATE = true
[source::*/access.log]
Configuring Splunk
[ 288 ]
BREAK_ONLY_BEFORE_DATE = false
[security_log]
```

```
EXTRACT-<name> = \[ (?P<user>.*?) \]
BREAK_ONLY_BEFORE_DATE = false
```

The setting with the biggest impact here is the `BREAK_ONLY_BEFORE_DATE = false` bare attribute. It is first added to the `[default]` stanza and then added to all stanzas that do not already have any value.

TIP

As a general rule, avoid using the `[default]` stanza and bare word attributes. The final impact may not be what you expect.

CONFIGURATION MERGING – EXAMPLE 4

In this case, we are in search, so we will use the more complicated merging order.

Assuming that we are currently working in the `d` app, let's merge the same configurations again. For simplicity, we are assuming that all attributes are shared globally. We will merge the same configurations listed previously in *Configuration merging – example 3*.

With `d` as our current app, we will now merge in this order:

```
$SPLUNK_HOME/etc/system/local/props.conf
$SPLUNK_HOME/etc/apps/b/default/props.conf
$SPLUNK_HOME/etc/apps/d/default/props.conf
$SPLUNK_HOME/etc/apps/d/local/props.conf
```

Walking through each merge, the configuration will look as follows:

1. We will start with `$SPLUNK_HOME/etc/system/local/props.conf`:

```
BREAK_ONLY_BEFORE_DATE = false
[web_access]
TZ = CST
```

2. Now, we will merge the default for apps other than our current app (which, in this case, is only one configuration): `$SPLUNK_HOME/etc/apps/b/default/props.conf`:

```
BREAK_ONLY_BEFORE_DATE = false
[web_access]
```

```

MAX_TIMESTAMP_LOOKAHEAD = 20
TIME_FORMAT = %Y-%m-%d $H:%M:%S
TZ = CST
[source::*/access.log]
BREAK_ONLY_BEFORE_DATE = false

```

3. Next, we will merge the default of our current app: `$SPLUNK_HOME/etc/apps/d/default/props.conf`:

```

BREAK_ONLY_BEFORE_DATE = false
[web_access]
MAX_TIMESTAMP_LOOKAHEAD = 25
TIME_PREFIX = ^\[
TIME_FORMAT = %Y-%m-%d $H:%M:%S
TZ = CST
[source::*/access.log]
BREAK_ONLY_BEFORE_DATE = false
[source::*.log]
BREAK_ONLY_BEFORE_DATE = true

```

4. Now, we will merge our current app

`local $SPLUNK_HOME/etc/apps/d/local/props.conf:`

```

BREAK_ONLY_BEFORE_DATE = false
[web_access]
MAX_TIMESTAMP_LOOKAHEAD = 25
TIME_PREFIX = ^\[
TIME_FORMAT = %Y-%m-%d $H:%M:%S
TZ = UTC
[source::*/access.log]
BREAK_ONLY_BEFORE_DATE = false
[source::*.log]
BREAK_ONLY_BEFORE_DATE = true
[security_log]
EXTRACT-<name> = \[ (?P<user>.*?) \]

```

5. And finally, we will apply our default stanza to stanzas that don't already have the attribute:

```

BREAK_ONLY_BEFORE_DATE = false
[web_access]
MAX_TIMESTAMP_LOOKAHEAD = 25

```

```
TIME_PREFIX = ^\[  
TIME_FORMAT = %Y-%m-%d $H:%M:%S  
TZ = UTC  
BREAK_ONLY_BEFORE_DATE = false  
[source::*/access.log]  
BREAK_ONLY_BEFORE_DATE = false  
[source::*.log]  
BREAK_ONLY_BEFORE_DATE = true  
[security_log]  
EXTRACT-<name> = \[(?P<user>.*?)\]  
BREAK_ONLY_BEFORE_DATE = false
```

I know this is fairly confusing, but with practice, it will make sense. Luckily, **btool**, which we will cover in the next section, makes it easier to see this.

Using btool

To help preview merged configurations, we call on btool, a command-line tool that prints the merged version of configurations. The Splunk site has one of my favorite documentation notes of all time, as follows:

btool is not tested by Splunk and is not officially supported or guaranteed. That said, it's what our Support team uses when trying to troubleshoot your issues.

With that warning in mind, btool has never steered me wrong. The tool has a number of functions, but the only one I have ever used is list, as follows:

```
$SPLUNK_HOME/bin/splunk cmd btool props list
```

This produces 5,277 lines of output, which I won't list here. Let's list the `impl_splunk_gen` stanza by adding it to the end of the command line, as shown here:

```
/opt/splunk/bin/splunk cmd btool props list impl_splunk_gen
```

This will produce an output such as this:

```
[impl_splunk_gen]
ANNOTATE_PUNCT = True
BREAK_ONLY_BEFORE =
BREAK_ONLY_BEFORE_DATE = True
... truncated ...
LINE_BREAKER_LOOKBEHIND = 100
LOOKUP-lookupusers = userslookup user AS user OUTPUTNEW
MAX_DAYS_AGO = 2000
... truncated ...
TRUNCATE = 10000
TZ = UTC
maxDist = 100
```

Our configuration file

at `$SPLUNK_HOME/etc/apps/ImplementingSplunkDataGenerator/local/props.conf` contains only the following lines:

```
[impl_splunk_web]
LOOKUP-web_section = flatten_summary_lookup url AS url
OUTPUTNEW
EXTRACT-url = \s[A-Z]+\s(?P<url_from_app_local>.*?)\s
EXTRACT-foo = \s[A-Z]+\s(?P<url_from_app>.*?)\s
```

So, where did the rest of this configuration come from? With the use of the `-debug` flag, we can get more details:

```
/opt/splunk/bin/splunk cmd btool props list impl_splunk_gen -debug
```

This produces the following query:

```
Implementi [impl_splunk_gen]
system ANNOTATE_PUNCT = True
system BREAK_ONLY_BEFORE =
system BREAK_ONLY_BEFORE_DATE = True
... truncated ...
system LINE_BREAKER_LOOKBEHIND = 100
Implementi LOOKUP-lookupusers = userslookup user AS user
OUTPUTNEW
```

```
system MAX_DAYS_AGO = 2000
... truncated ...
system TRUNCATE = 10000
Implementi TZ = UTC
system maxDist = 100
```

The first column, truncated though it is, tells us what we need to know. The vast majority of these lines are defined in the system, most likely in `system/default/props.conf`.

The remaining items from our file are labeled `Implementi`, which is the beginning of our app directory, `ImplementingSplunkDataGenerator`. If you ever have a question about where some setting is coming from, btool will save you a lot of time. Also, check out the *Splunk on Splunk* app at Splunkbase for access to btool from the Splunk web interface.

Summary

This section provided an overview of how configurations work and a commentary on the most common aspects of Splunk configuration. This is by no means a complete reference for these configurations, which I will leave to the official documentation. I find that the easiest way to get to the official documentation for a particular file is to query your favorite search engine for splunk `configname.conf`. distributed deployments and look at how they are efficiently configured. What you have learned in this section will be vital to understanding what is considered to be a best practice.

Creating indexes

Indexes are where Splunk Enterprise stores all the data it has processed. It is essentially a collection of databases that is, by default, located at `$SPLUNK_HOME/var/lib/splunk`. Before data can be searched, it needs to be indexed, a process we describe here.

There are two ways to create an index, through the Splunk portal or by creating an `indexes.conf` file. You will be shown here how to create an index using the Splunk portal, but you should realize that when you do that, it simply generates an `indexes.conf` file.

You will be creating an index called `wineventlogs` to store Windows Event Logs. To do this, take the following steps:

1. In the Splunk navigation bar, go to **Settings**.
2. In the **Data** section, click on **Indexes**, which will take you to the Indexes page.
3. Click on **New Index**.
4. Now fill out the information for this new index as seen in the following screenshot, carefully going through steps 1 to 4.
5. Be sure to **Save** when you are done.

The screenshot shows the 'New Index' configuration dialog in the Splunk interface. The 'Index Name' field is set to 'wineventlogs'. The 'Max Size of Entire Index' field is set to '100' with a dropdown menu for 'MB'. The 'Max Size of Hot/Warm/Cold Bucket' field is set to 'auto'. The 'App' field is set to 'Destinations'. Red boxes highlight the 'Index Name' field, the 'Max Size of Entire Index' field, the 'Max Size of Hot/Warm/Cold Bucket' field, and the 'App' field, corresponding to the steps described in the text above.

You will now see the new index in the list as shown here:

wineventlog	Edit	Delete	Disable	destinations	1 MB	100 MB	0
-------------	------	--------	---------	--------------	------	--------	---

The preceding steps have created a new `indexes.conf` file. Now go ahead and inspect this file. In your administrator command prompt (assuming that you are at the root of the `C:` drive), type the following command:

```
C:\> notepad  
c:\splunk\etc\apps\destinations\local\indexes.conf
```

Every index has specific settings of its own. Here is how your index looks when automatically configured by the portal. In production environments, this is how Splunk administrators manage the indexes. Note that the max size value of 100 that you specified is also indicated in the configuration.

```
[wineventlogs]  
coldPath = $SPLUNK_DB\wineventlogs\colddb  
homePath = $SPLUNK_DB\wineventlogs\db  
maxTotalDataSizeMB = 100  
thawedPath = $SPLUNK_DB\wineventlogs\thaweddb
```

The complete `indexes.conf` documentation can be found at <http://docs.splunk.com/Documentation/Splunk/latest/admin/indexesconf>.

Buckets

You may have noticed that there is a certain pattern in this configuration file, in which folders are broken into three locations: `coldPath`, `homePath`, and `thawedPath`. This is a very important concept in Splunk. An index contains compressed raw data and associated index files that can be spread out into age-designated directories. Each piece of this index directory is called a **bucket**.

A bucket moves through several stages as it ages. In general, as your data gets older (think colder) in the system, it is pushed to the next bucket. And, as you can see in the following list, the thawed bucket contains data that has been resurrected from an archive. Here is a breakdown of the buckets in relation to each other:

- **hot**: This is newly indexed data and open for writing (`hotPath`)
- **warm**: This is data rolled from hot with no active writing (`warmPath`)
- **cold**: This is data rolled from warm (`coldPath`)
- **frozen**: This is data rolled from cold and deleted by default but it can be archived (`frozenPath`)
- **thawed**: This is data restored from the archive (`thawedPath`)

Now going back to the `indexes.conf` file, realize that the `homePath` will contain the hot and warm buckets, the `coldPath` will contain the cold bucket, and the `thawedPath` will contain any restored data from the archive. This means you can put buckets in different locations to efficiently manage disk space.

In production environments, Splunk admins never use the portal to generate indexes. They make their changes in the `indexes.conf` file. It is best practice to always use a temporary index for new data that you are unfamiliar with. Go ahead and create a new index stanza in `indexes.conf` with the following information:

```
[temp]
coldPath = $SPLUNK_DB\temp\colddb
homePath = $SPLUNK_DB\temp\db
maxTotalDataSizeMB = 100
```

```
thawedPath = $SPLUNK_DB\temp\thaweddb
```

1. Click on **Save**.
2. **Exit** the `indexes.conf` file when you are done.
3. Restart Splunk, which you must do for this change to take effect.
4. After restarting Splunk, go to the `http://localhost:8000/en-US/manager/destinations/data/indexes` page to confirm that the new index named `temp` has been created.

Data inputs

As you may have noticed, any configuration you make in the Splunk portal corresponds to a `*.conf` file written to the disk. The same goes for the creation of data inputs; it creates a file called `inputs.conf`. Now that you have an index to store your machine's Windows Event Logs, let us go ahead and create a data input for it, with the following steps:

1. Go to the Splunk home page.
2. Click on your Destinations app. Make sure you are in the Destinations app before you execute the next steps.
3. In the Splunk navigation bar, select **Settings**.
4. Under the **Data** section, click on **Data inputs**.
5. On the **Data inputs** page, click on the **Local event log collection** type as shown in the following screenshot:

The screenshot shows the 'Type' section of the Data inputs configuration page. It includes three options: 'Local event log collection' (selected and highlighted with a red box), 'Remote event log collections', and 'Files & directories'. The 'Local event log collection' option is described as 'Collect event logs from this machine.'

6. In the next page select the **Application** and **System** log types.
7. Change the index to `wineventlog`. Compare your selections with the following screenshot:

The screenshot shows the 'Selected log(s)' section of the Data inputs configuration page. It displays two selected log types: 'Application' and 'System' (highlighted with a red box). Below this, there is a note: 'Select the Windows Event Logs you want to index from the list.' The 'Available log(s)' list on the left includes Application, Security, Setup, System, ForwardedEvents, Els_Hyphenation/Analytic, EndpointMapper, FirstUXPerf-Analytic, and AirSpaceChannel. At the bottom, the 'Index' section is set to 'wineventlog' (highlighted with a red box), and the 'Save' button is visible.

8. Click **Save**.
9. On the next screen, confirm that you have successfully created the data input, as shown in the following screenshot:

The screenshot shows a table with one item. The columns are: Event Log collection name, Log(s), Host(s), Index, Status, and Actions. The data is as follows:

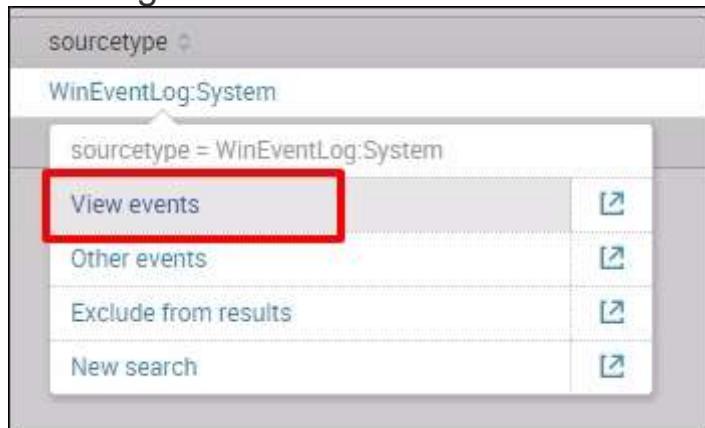
Event Log collection name	Log(s)	Host(s)	Index	Status	Actions
localhost	Application, System	localhost	wineventlogs	Enabled	Disable

Before we proceed further, let's make sure that the data input works. To do so, follow these steps:

1. Go to the Destinations search page: <http://localhost:8000/us/app/destinations/search>.
2. In the search window, run the following Splunk search query:

 3. `SPL> index=wineventlogs | top sourcetype`

4. Check the search results to see that there are now two actively-indexed source types in your Splunk system, that is, **WinEventLog:System** and **WinEventLog:Application**.
5. View the events of each of these source types by clicking the source type name and selecting **View events** as shown in the following screenshot:



We have introduced a new concept here called **source types**. A source type is a type of classification of data that has been automatically made for you when you created the data input through the Splunk portal. In the same index, due to the steps we just took precedingly, Splunk internally classified the **System** and **Applications** event logs so you can access the events separately. There will be more about source types in [Chapter 3, Search Processing Language](#).

Go ahead and inspect the `inputs.conf` file:

1. In an administrator prompt, open the file with this command:

```
C:\> notepad  
c:\splunk\etc\apps\destinations\local\inputs.conf
```

2. Compare your results with this generated `inputs.conf` file:

```
[WinEventLog://Application]  
checkpointInterval = 5  
current_only = 0  
disabled = 0  
index = wineventlogs  
start_from = oldest  
  
[WinEventLog://System]  
checkpointInterval = 5  
current_only = 0  
disabled = 0  
index = wineventlogs  
start_from = oldest
```

You can add directly into the `inputs.conf` file by following the same format. Use the temp index for testing. Here is an example input file based on a log file that is generated by the Splunk application. Note that this is going to be redundant to what is already being ingested through the `_internal` index and will only serve as an example.

```
[monitor://C:\Splunk\var\log\splunk\eventgen.log]  
disabled = 0  
sourcetype = eventgen  
index = temp
```

Changes to the `inputs.conf` may require a Splunk restart. You can access that information with the following search command:

```
SPL> index=temp sourcetype=eventgen
```

The complete documentation for the `inputs.conf` file can be found at <https://docs.splunk.com/Documentation/Splunk/latest/Admin/Inputsconf>.

If you closely followed the instructions in these guides, you should now have exactly four data sources in your very own Splunk system that will be used in the remainder of the class. This is how you can query raw data from these data sources.

Splunk events and fields

All throughout this chapter you have been running Splunk search queries that have returned data. It is important to understand what events and fields are before we go any further, for an understanding of these is essential to comprehending what happens when you run Splunk on the data.

In Splunk, a single piece of data is known as an **event** and is like a record, such as a log file or other type of input data. An event can have many different attributes or fields or just a few attributes or fields. When you run a successful search query, you will see that it brings up events from the data source. If you are looking at live streaming data, events can come in very quickly through Splunk.

Every event is given a number of default fields. For a complete listing, go to <http://docs.splunk.com/Documentation/Splunk/6.3.2/Data/Aboutdefaultfields>. We will now go through some of these default fields.

- **Timestamp:** A timestamp is applied at the exact time the event is indexed in Splunk. Usually, Splunk can figure out what timestamp to assign from the raw data it receives. For example, as a shopper clicks the final purchase button on an e-commerce website, data is collected about precisely when the sale occurred. Splunk can usually automatically detect this from the raw data.
- **Host:** The host field tells us what the hostname, IP address, or full domain name of the data is.
- **Index:** The index field describes where the event is located, giving the specific name of the index.
- **Source and sourcetype:** The source field tells us where the data came from, specifically the file, data stream, or other data input. The sourcetype is the format of the data input from which the data came. Common sourcetypes are `access_combined`, `access_custom`, and `cisco_syslog`.
- **Linecount:** The linecount is simply the number of lines contained in the event.

These default fields are name/value pairings that are added to events when Splunk indexes data. Think of fields as a quick way to categorize and group events. Fields are the primary constituents of all search queries. In later chapters you will learn more about fields and how to create custom fields from events.

Extracting new fields

Most raw data that you will encounter will have some form of structure. Just like a CSV (comma-separated value file) or a web log file, it is assumed that each entry in the log corresponds to some sort of format. Splunk 6.3+ makes custom field extraction very easy, especially for delimited files. Let's take the case of our Eventgen data and look at the following example. If you look closely, the `_raw` data is actually delimited by white spaces:

```
2016-01-21 21:19:20:013632 130.253.37.97 GET /home - 80 -
10.2.1.33 "Mozilla/5.0 (iPad; U; CPU OS 4_3_3 like Mac OS X;
en-us) AppleWebKit/533.17.9 (KHTML, like Gecko)
Version/5.0.2 Mobile/8J3 Safari/6533.18.5" 200 0 0 186 3804
```

Since there is a distinct separation of fields in this data, we can use Splunk's out-of-the-box field extraction tool to automatically classify these fields. In your Destinations app Search page, run the following search command:

```
SPL> index=main sourcetype=access_custom
```

This sourcetype `access_custom` refers to a type of file format that is generated by a server as it creates a web log file. After the data populates, click on the **Extract New Fields** link in the left column of the page as shown in the following screenshot:

The screenshot shows the 'Extract Fields' page in Splunk. At the top left is a 'Hide Fields' link and a 'All Fields' button. Below these sections are two lists: 'Selected Fields' and 'Interesting Fields'. The 'Selected Fields' list contains four items: 'host' (count 1), 'source' (count 1), 'sourcetype' (count 1), and 'index' (count 1). The 'Interesting Fields' list contains many more items, including date-related fields like 'date_hour', 'date_mday', 'date_minute', 'date_month', 'date_second', 'date_wday', 'date_year', and various time-related fields such as 'linecount', 'punct', 'splunk_server', 'timeendpos', and 'timestartpos'. At the bottom of the page is a red-bordered button labeled 'Extract New Fields'.

In the resulting **Extract Fields** page, select one of the events that is shown in the `_raw` events area. Try to select an entry with the longest text. As soon as you do this, the text will appear highlighted at the top of the page, as per the following screenshot:

Extract Fields

Select sample Select method Select fields Save **Next >** Existing fields >

Select Sample Event
Choose a source or source type, select a sample event, and click Next to go to the next step. The field extractor will use the event to extract fields. Learn more [I prefer to write the regular expression myself >](#)

Source type `access_custom`

```
2016-07-19 05:42:34:656523,164.218.0.0,GET,/destination/HOU/details,-,80,-,10.2.1.33,Mozilla/5.0 (Windows NT 6.2; WOW64)  
AppleWebKit/537.36 (KHTML; like Gecko) Chrome/30.0.1599.66 Safari/537.36,302,0,0,672,2162
```

Click on the **Next** button to proceed. In the page that appears, click on the **Delimiters** icon as indicated in the following screenshot:

Regular Expression
Splunk Enterprise will extract fields using a Regular Expression.

Delimiters
Splunk Enterprise will extract fields using a delimiter (such as commas, spaces, or characters). Use this method for delimited data like comma-separated values (CSV files).

Click on **Next**. On the next page, click on the **Comma** delimiter as shown in the following screenshot:

Rename Fields

Select a delimiter. In the table that appears, rename fields by clicking on field names or values. Learn more [\[?\]](#)

Delimiter

Space Comma Tab Pipe Other

field1	field2	field3	field4	field5	field6
2016-02-04 05:27:36.719023	74.125.19.106	GET	/destination/MIA/details	-	80

As soon as you select the **Comma** delimiter, Splunk will automatically allow you to modify each field and add a label to it. Click on the pencil icon for each field to input the label. When you're done, click on the **Rename Field** icon. For example, to edit **field3**, use the following screenshot:

The screenshot shows the Splunk interface with the 'Comma' delimiter selected. A specific event is highlighted, and the 'Field Name' for 'field3' is being renamed to 'ip_address'. A 'Rename Field' button is visible at the bottom right of the dialog.

Do the same for the remaining fields using the following guide. These fields will be needed in future chapters. You can skip those that are not included in the following list:

- **field1:** `datetime`
- **field2:** `client_ip`
- **field3:** `http_method`
- **field4:** `http_uri`
- **field8:** `server_ip`
- **field9:** `http_user_agent`
- **field10:** `http_status_code`
- **field14:** `http_response_time`

When you have completed the preceding task, click on **Next** to proceed. In the next window, label the **Extractions Name** as **eventgen** and select the **All apps** permission type. Refer to the following screenshot:

Save
Name the extraction and set permissions.

Extractions Name	REPORT	eventgen	
Owner	admin		
App	destinations		
Permissions	Owner	App	All apps

Click on **Finish** to complete the process. Now that you have extracted new fields, these will now be readily available in your search queries as exemplified below. In the next chapter, you will be shown how to use these new fields to filter search results. An example of the kind of search you can do on them is shown here:

```
SPL> index=main | top http_uri
```

If you want to go ahead and try this out now, just to prove that you have already made changes that will help you to understand the data you are bringing in, be our guest!

Getting More Data In New Data :)

In this session, we will continue to cover the basic ways to get data into **Splunk**. You will learn about the following recipes:

- Indexing files and directories
- Getting data through network ports
- Using scripted inputs
- Using modular inputs
- Using the Universal Forwarder to gather data
- Loading the sample data for this book
- Defining field extractions
- Defining event types and tags

Introduction

The machine data that facilitates operational intelligence comes in many different forms and from many different sources. Splunk is able to collect and index data from many different sources, including log files written by web servers or business applications, syslog data streaming in from network devices, or the output of custom-developed scripts. Even data that looks complex at first can be easily collected, indexed, transformed, and presented back to you in real time.

This **session will walk you through the basic recipes that will act as the building blocks to get the data you want into Splunk**. The session will further serve as an introduction to the sample datasets that we will use to build our own operational intelligence Splunk app. The datasets will be coming from a hypothetical, three-tier, e-commerce web application and will contain web server logs, application logs, and database logs.

Splunk Enterprise can index any type of data; however, it works best with time-series data (data with timestamps). When Splunk Enterprise indexes data, it breaks it into events, based on timestamps and/or event size, and puts them into indexes. Indexes are data stores that Splunk has engineered to be very fast, searchable, and scalable across a

distributed server environment; they are commonly referred to as indexers. This is also why we refer to the data being put into Splunk as being indexed.

All data indexed into Splunk is assigned a source type. The source type helps identify the data format type of the event and where it has come from. Splunk has a number of preconfigured source types, but you can also specify your own. The example source types include `access_combined`, `cisco_syslog`, and `linux_secure`. The source type is added to the data when the indexer indexes it into Splunk. It is a key field that is used when performing field extractions and when conducting many searches to filter the data being searched.

The **Splunk community plays a big part in making it easy to get data into Splunk.** The ability to extend Splunk has provided the opportunity for the development of inputs, commands, and applications that can be easily shared. If there is a particular system or application you are looking to index data from, there is most likely someone who has developed and published relevant configurations and tools that can be easily leveraged by your own Splunk Enterprise deployment.

Splunk Enterprise is designed to **make the collection of data very easy**, and it will not take long before you are being asked or you yourself try to get as much data into Splunk as possible—at least as much as your license will allow for!

Indexing files and directories

File and directory-based inputs are the most commonly used ways of getting data into Splunk. The primary need for these types of inputs will be to index logfiles. Almost every application or system produces a logfile, and it is generally full of data that you want to be able to search and report on.

Splunk is able to continuously monitor for new data being written to existing files or new files being added to a directory, and it is able to index this data in real time. Depending on the type of application that creates the logfiles, you would set up Splunk to either monitor an individual file based on its location or scan an entire directory and monitor all the files that exist within it. The latter configuration is more commonly used when the logfiles being produced have unique filenames, such as filenames containing a timestamp.

This recipe will show you how to configure Splunk to continuously monitor and index the contents of a rolling logfile located on the Splunk server. The recipe specifically shows how to monitor and index a Red Hat Linux system's messages logfile ([/var/log/messages](#)). However, the same principle can be applied to a logfile on a Windows system, and a sample file is provided. Do not attempt to index the Windows event logs this way, as Splunk has specific Windows event inputs for this.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server and access to read the [/var/log/messages](#) file on Linux. No other prerequisites are required. If you are not using Linux and/or do not have access to the [/var/log/messages](#) location on your Splunk server, use the [cp01_messages.log](#) file that is provided and upload it to an accessible directory on your Splunk server.

How to do it...

Follow the steps in the recipe to monitor and index the contents of a file:

1. Log in to your Splunk server.
2. From the menu in the top right-hand corner, click on the **Settings** menu and then click on the **Add Data** link.



3. If you are prompted to take a quick tour, click on **Skip**.
4. In the **How do you want to add data?** section, click on **monitor**.



5. Click on the **Files & Directories** section.

Add Data

Select Source Set Source Type Input Settings Review Done

Files & Directories  

Upload a file, index a local file, or monitor an entire directory.

HTTP Event Collector

Configure tokens that clients can use to send data over HTTP or HTTPS.

TCP / UDP

Configure Splunk to listen on a network port.

Scripts

Get data from from any API, service, or database with a script.

File or Directory?

6. In the **File or Directory** section, enter the path to the logfile ([/var/log/messages](#) or the location of the [cp01_messages.log](#) file), ensure **Continuously Monitor** is selected, and click on **Next**.

File or Directory?  [/var/log/messages](#)

On Windows: c:\apache\apache.error.log or \\hostname\apache\apache.error.log. On Unix: /var/log or /mnt/www01/var/log.

Continuously Monitor Index Once

Whitelist?

Blacklist?

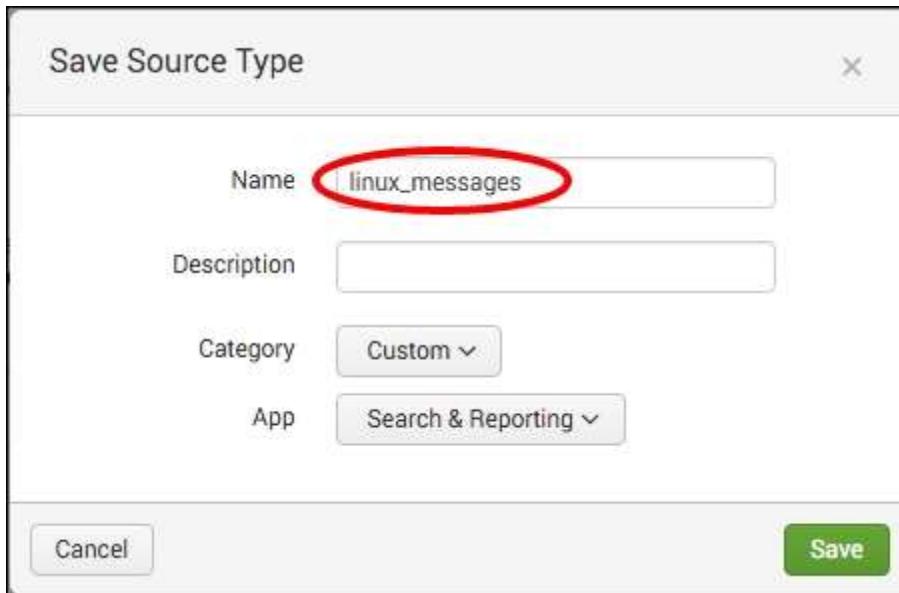
TIP

If you are just looking to do a one-time upload of a file, you can select **Index Once** instead. This can be useful to index a set of data that you would like to put into Splunk, either to backfill some missing or incomplete data or just to take advantage of its searching and reporting tools.

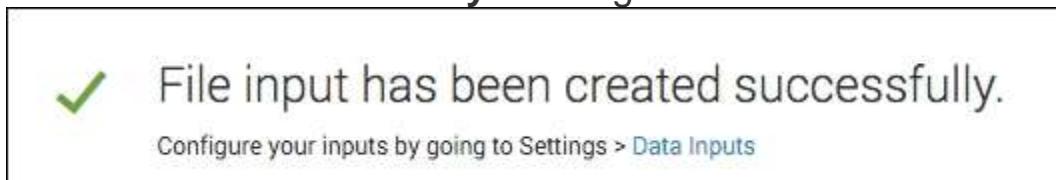
7. Assuming that you are using the provided file or the native [/var/log/messages](#) file, the data preview will show the correct

line breaking of events and timestamp recognition. Click on the **Next** button.

8. A **Save Source Type** box will pop up. Enter `linux_messages` as the **Name** and then click on **Save**.



9. On the **Input Settings** page, leave all of the default settings, and click **Review**.
10. Review the settings and if everything is correct, click **Submit**.
11. If everything was successful, you should see a **File input has been created successfully** message.



12. Click on the **Start searching** button. The **Search & Reporting** app will open with the search already populated based on the settings supplied earlier in the recipe.

TIP

In this recipe, we could have simply used the common syslog source type or let Splunk choose a source type name for us; however, starting a new source type is often a better choice. The syslog format can look completely different depending on the data source. As knowledge objects, such as field extractions, are built on top of source types, using a

single syslog source type for everything can make it challenging to search for the data you need.

How it works...

When you add a new file or directory data input, you are basically adding a new configuration stanza into an `inputs.conf` file behind the scenes. The Splunk server can contain one or more `inputs.conf` files, and these files are either located in `$SPLUNK_HOME/etc/system/local` or in the local directory of a Splunk app.

Splunk uses the monitor input type and is set to point to either a file or a directory. If you set the monitor to point to a directory, all the files within that directory will be monitored. When Splunk monitors files, it initially starts by indexing all the data that it can read from the beginning. Once complete, Splunk maintains a record of where it last read the data from, and if any new data comes into the file, it reads this data and advances the record. The process is nearly identical to using the `tail` command in Unix-based operating systems. If you are monitoring a directory, Splunk also provides many additional configuration options, such as blacklisting files you don't want Splunk to index.

TIP

For more information on Splunk's configuration files, visit <http://docs.splunk.com/Documentation/Splunk/latest/Admin/Aboutconfigurationfiles>.

There's more...

While adding inputs to monitor files and directories can be done through the web interface of Splunk, as outlined in this recipe, there are other approaches to add multiple inputs quickly. These allow for customization of the many configuration options that Splunk provides.

ADDING A FILE OR DIRECTORY DATA INPUT VIA THE CLI

Instead of going via the GUI, you can add a file or directory input via the **Splunk CLI (command-line interface)**. Navigate to your `$SPLUNK_HOME/bin` directory and execute the following command (replacing the file or directory to be monitored with your own):

For Unix:

```
./splunk add monitor /var/log/messages -sourcetype  
linux_messages
```

For Windows:

```
splunk add monitor c:\filelocation\cp01_messages.log -  
sourcetype linux_messages
```

There are a number of different parameters that can be passed along with the file location to monitor.

NOTE

See the Splunk documentation for more on data inputs using the CLI (<http://docs.splunk.com/Documentation/Splunk/latest/Data/MonitorfilesanddirectoriesusingtheCLI>).

ADDING A FILE OR DIRECTORY INPUT VIA INPUTS.CONF

Another common method of adding the file and directory inputs is to manually add them to the `inputs.conf` configuration file directly. This approach is often used for large environments or when configuring Splunk forwarders to monitor for files or directories on endpoints.

Edit `$SPLUNK_HOME/etc/system/local/inputs.conf` and add your input. After your inputs are added, Splunk will need to be restarted to recognize these changes:

For Unix:

```
[monitor:///var/log/messages]  
sourcetype = linux_messages
```

For Windows:

```
[monitor://c:\filelocation\cp01_messages.log]
sourcetype = linux_messages
```

NOTE

Editing `inputs.conf` directly is often a much faster way of adding new files or directories to monitor when several inputs are needed. When editing `inputs.conf`, ensure that the correct syntax is used and remember that Splunk will need a restart for modifications to take effect. Additionally, specifying the source type in the `inputs.conf` file is best practice to assign source types.

ONE-TIME INDEXING OF DATA FILES VIA THE SPLUNK CLI

Although you can select **Upload and Index a file** from the Splunk GUI to upload and index a file, there are a couple of CLI functions that can be used to perform one-time bulk loads of data.

Use the `oneshot` command to tell Splunk where the file is located and which parameters to use, such as the source type:

```
./splunk add oneshot XXXXXXXX
```

Another way is to place the file you wish to index into the Splunk `spool` directory, `$SPLUNK_HOME/var/spool/splunk`, and then add the file using the `spool` command:

```
./splunk spool XXXXXXXX
```

TIP

If using Windows, omit `./` that is in front of the Splunk commands mentioned earlier.

INDEXING THE WINDOWS EVENT LOGS

Splunk comes with special `inputs.conf` configurations for some source types, including monitoring the Windows event logs. Typically, the Splunk **Universal Forwarder (UF)** would be installed on a Windows

server and configured to forward the Windows events to the Splunk indexer(s). The configurations for `inputs.conf` to monitor the Windows security, application, and event logs in real time are as follows:

```
[WinEventLog://Application]
disabled = 0
[WinEventLog://Security]
disabled = 0
[WinEventLog://System]
disabled = 0
```

By default, the event data will go into the main index, unless another index is specified.

Getting data through network ports

Not every machine has the luxury of being able to write logfiles. Sending data over network ports and protocols is still very common. For instance, sending logs via syslog is still the primary method to capture network device data such as firewalls, routers, and switches.

Sending data to Splunk over network ports doesn't need to be limited to network devices. Applications and scripts can use socket communication to the network ports that Splunk is listening on. This can be a very useful tool in your back pocket, as there can be scenarios where you need to get data into Splunk but don't necessarily have the ability to write to a file.

This recipe will show you how to configure Splunk to receive syslog data on a UDP network port, but it is also applicable to the TCP port configuration.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server. No other prerequisites are required.

How to do it...

Follow the steps in the recipe to configure Splunk to receive network UDP data:

1. Log in to your Splunk server.
2. From the menu in the top right-hand corner, click on the **Settings** menu and then click on the **Add Data** link.



3. If you are prompted to take a quick tour, click on **Skip**.
4. In the **How do you want to add data?** section, click on **monitor**.

Add Data
How do you want to add data?

- upload**
File from my computer
Locate log files
Local structured files (e.g. CSV)
Tutorial for adding data ↗
- monitor**
File and ports on this Splunk instance
Files - WMI - TCP/UDP - Scripts
Solar inputs for external data sources
- forward**
data from Splunk forwarder
Files - TCP/UDP - Scripts
Help me install the universal forwarder ↗

5. Click on the **TCP / UDP** section.

Add Data Select Source Input Settings Review Done < Next >

Files & Directories Upload a file, index a local file, or monitor an entire directory.	
HTTP Event Collector Configure tokens that clients can use to send data over HTTP or HTTPS.	
TCP / UDP Configure Splunk to listen on a network port.	
Scripts Get data from from any API, service, or database with a script.	

6. Ensure the **UDP** option is selected and in the **Port** section, enter [514](#). On Unix/Linux, Splunk must be running as root to access privileged ports such as 514. An alternative would be to specify a

higher port such as port 1514 or route data from 514 to another port using routing rules in iptables. Then click on **Next**.

TCP UDP

Port? **514** Example: 514

Source name override? optional
host:port

Only accept connection from? optional
example: 10.1.2.3, !badhost.splunk.com, *.splunk.com

7. In the **Source type** section, select **From** list from the **Set sourcetype** drop-down list, and then, select **syslog** from the **Select Source Type** drop-down list and click **Review**.

Input Settings
Optionally set additional input parameters for this data input as follows:

Source type
The source type is one of the default fields that Splunk assigns to all incoming data. It tells Splunk what kind of data you've got, so that Splunk can format the data intelligently during indexing. And it's a way to categorize your data, so that you can search it easily.

Select New

syslog ▾

8. Review the settings and if everything is correct, click **Submit**.
9. If everything was successful, you should see a **UDP input has been created successfully** message.

✓ UDP input has been created successfully.
Configure your inputs by going to Settings > Data Inputs

10. Click on the **Start searching** button. The **Search & Reporting** app will open with the search already populated based on the settings supplied earlier in the recipe. Splunk is now

configured to listen on UDP port 514. Any data sent to this port now will be assigned the syslog source type. To search for the syslog source type, you can run the following search:

```
source="udp:514" sourcetype="syslog"
```

Understandably, you will not see any data unless you happen to be sending data to your Splunk server IP on UDP port 514.

How it works...

When you add a new network port input, you basically add a new configuration stanza into an `inputs.conf` file behind the scenes. The Splunk server can contain one or more `inputs.conf` files, and these files are either located in the `$SPLUNK_HOME/etc/system/local` or the local directory of a Splunk app.

To collect data on a network port, Splunk will set up a socket to listen on the specified TCP or UDP port and will index any data it receives on that port. For example, in this recipe, you configured Splunk to listen on port 514 for UDP data. If data was received on that port, then Splunk would index it and assign a syslog source type to it.

Splunk also provides many configuration options that can be used with network inputs, such as how to resolve the host value to use on the collected data.

TIP

For more information on Splunk's configuration files, visit <http://docs.splunk.com/Documentation/Splunk/latest/Admin/Aboutconfigurationfiles>.

There's more...

While adding inputs to receive data from network ports can be done through the web interface of Splunk, as outlined in this recipe, there are

other approaches to add multiple inputs quickly; these inputs allow for customization of the many configuration options that Splunk provides.

ADDING A NETWORK INPUT VIA THE CLI

You can also add a file or directory input via the Splunk CLI. Navigate to your `$SPLUNK_HOME/bin` directory and execute the following command (just replace the protocol, port, and source type you wish to use):

For Unix:

```
./splunk add udp 514 -sourcetype syslog
```

For Windows:

```
splunk add udp 514 -sourcetype syslog
```

There are a number of different parameters that can be passed along with the port. See the Splunk documentation for more on data inputs using the CLI

(<http://docs.splunk.com/Documentation/Splunk/latest/Data/MonitorfilesanddirectoriesusingtheCLI>).

ADDING A NETWORK INPUT VIA INPUTS.CONF

Network inputs can be manually added to the `inputs.conf` configuration files. Edit `$SPLUNK_HOME/etc/system/local/inputs.conf` and add your input. You will need to restart Splunk after modifying the file:

```
[udp://514]
sourcetype = syslog
```

TIP

It is best practice to not send syslog data directly to an indexer. Instead, always place a forwarder between the network device and the indexer. The Splunk forwarder would be set up to receive the incoming syslog data (`inputs.conf`) and will load balance the data across your Splunk indexers (`outputs.conf`). The forwarder can also be configured to cache the syslog data in the event communication to the indexers is lost.

Using scripted inputs

Not all data that is useful for operational intelligence comes from logfiles or network ports. Splunk will happily take the output of a command or script and index it along with all your other data.

Scripted inputs are a very helpful way to get that hard-to-reach data. For example, if you have third-party-supplied command-line programs that can output data you would like to collect, Splunk can run the command periodically and index the results. Typically, scripted inputs are often used to pull data from a source, whereas network inputs await a push of data from a source.

This recipe will show you how to configure Splunk on an interval to execute your command and direct the output into Splunk.

Getting ready

To step through this recipe, you will need a running Splunk server and the provided scripted input script suited to the environment you are using. For example, if you are using Windows, use the `cp01_scripted_input.bat` file. This script should be placed in the `$SPLUNK_HOME/bin/scripts` directory. No other prerequisites are required.

How to do it...

Follow the steps in the recipe to configure a scripted input:

1. Log in to your Splunk server.
2. From the menu in the top right-hand corner, click on the **Settings** menu and then click on the **Add Data** link.



3. If you are prompted to take a quick tour, click on **Skip**.
4. In the **How do you want to add data?** section, click on **monitor**.



5. Click on the **Scripts** section.

Add Data

Select Source Input Settings Review Done

Files & Directories
Upload a file, index a local file, or monitor an entire directory.

HTTP Event Collector
Configure tokens that clients can use to send data over HTTP or HTTPS.

TCP / UDP
Configure Splunk to listen on a network port.

Scripts
Get data from from any API, service, or database with a script.

6. A form will be displayed with a number of input fields. In the **Script Path** drop-down, list select the location of the script. All scripts must be located in a Splunk `bin` directory, either

in `$SPLUNK_HOME/bin/scripts` or an appropriate bin directory within a Splunk app, such as `$SPLUNK_HOME/etc/apps/search/bin`.

7. In the **Script Name** drop-down list, select the name of the script. In the **Commands** field, add any command-line arguments to the auto-populated script name.
8. Enter the value in the **Interval** field (in seconds) in which the script is to be run (the default value is **60.0** seconds) and then click **Next**.

Configure this instance to execute a script or command and to capture its output as event data. Scripted inputs are useful when the data that you want to index is not available in a file to monitor. [Learn More](#)

Script Path	<input type="text" value="\$SPLUNK_HOME/bin/scripts"/>
Script Name	<input type="text" value="cp01_scripted_input.py"/>
Command?	<input type="text" value="\$SPLUNK_HOME/bin/scripts/cp01_scripted_input.py"/>
Interval?	<input type="text" value="60.0"/>
Source name override?	<input type="text" value="optional"/>

9. In the **Source Type** section, you have the option to either select a predefined source type or select **New** and enter your desired value. For the purpose of this recipe, select **New** as the source type and enter `cp01_scripted_input` as the value for the source type. Then click **Review**.

Select	New
Source Type	<input type="text" value="cp01_scripted_input"/>
Source Type Category	<input type="text" value="Custom"/>
Source Type Description	<input type="text"/>

Data will be indexed into Splunk's default index, which is main. To change the destination index, you can select the desired index from the drop-down list in the **Index** section.

10. Review the settings. If everything is correct, click **Submit**.
11. If everything was successful, you should see a **Script input has been created successfully** message.



Script input has been created successfully.

Configure your inputs by going to Settings > Data Inputs

12. Click on the **Start searching** button. The **Search & Reporting** app will open with the search already populated based on the settings supplied earlier in the recipe. Splunk is now configured to execute the scripted input you provided every 60 seconds, in accordance with the specified interval. You can search for the data returned by the scripted input using the following search over all time:

`sourcetype=cp01_scripted_input`

How it works...

When adding a new scripted input, you are directing Splunk to add a new configuration stanza into an `inputs.conf` file behind the scenes. The Splunk server can contain one or more `inputs.conf` files, located either in `$SPLUNK_HOME/etc/system/local` or the local directory of a Splunk app.

After creating a scripted input, Splunk sets up an internal timer and executes the command that you have specified, in accordance with the defined interval. It is important to note that Splunk will only run one instance of the script at a time, so if the script gets blocked for any reason, it will cause the script to not be executed again, until after it has been unblocked.

Since Splunk 4.2, any output of the scripted inputs that are directed to `stderr` (causing an error) are captured to the `splunkd.log` file, which can be useful when attempting to debug the execution of a script. As Splunk indexes its own data by default, you can search for that data and alert on it if necessary.

For security reasons, Splunk does not execute scripts located outside of the bin directories mentioned earlier. In order to overcome this limitation,

you can use a wrapper script (such as a shell script in Linux or batch file in Windows) to call any other script located on your machine.

Using modular inputs

Since Splunk 5.0, the ability to extend data input functionality has existed such that custom input types can be created and shared while still allowing for user customization to meet needs.

Modular inputs build further upon the scripted input model. Originally, any additional functionality required by the user had to be contained within a script. However, this presented a challenge, as no customization of this script could occur from within Splunk itself. For example, pulling data from a source for two different usernames needed two copies of a script or meant playing around with command-line arguments within your scripted input configuration.

By leveraging the modular input capabilities, the developers are now able to encapsulate their code into a reusable app that exposes parameters in Splunk and allows for configuration through processes familiar to Splunk administrators.

This recipe will walk you through how to install the **Command Modular Input**, which allows for periodic execution of commands and subsequent indexing of the command output. You will configure the input to collect the data output by the `vmstat` command in Linux and the `systeminfo` command in Windows.

Getting ready

To step through this recipe, you will need a running Splunk server with a connection to the Internet. No other prerequisites are required.

How to do it...

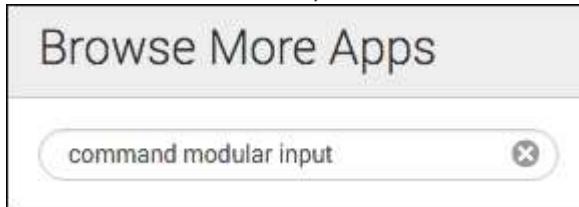
Follow the steps in this recipe to configure a modular input:

1. Log in to your Splunk server.

- From the **Apps** menu in the upper left-hand corner of the home screen, click on the gear icon.



- The **Apps settings** page will load. Then click on the **Browse More Apps** button.
- In the search field, enter `command modular input` and press *Enter*.



- In the search results, click on the **Install** button for **Command Modular Input**.

CMD Command Modular Input

This is a Splunk Modular Input for executing commands and indexing the output. It is quite simply just a wrapper around whatever system commands/programs that you want to periodically execute and capture the output from ie: (top, ps, iostat, tshark, tcpdump etc...). It will work on all supported Splunk platforms.

Category: Utilities | Author: Damien Dallimore | Downloads: 1061 | Released: 3 years ago | Last Updated: 2 years ago | View on Splunkbase

A screenshot of a Splunkbase page for the "Command Modular Input" app. The page has a blue header with the text "CMD" and "Command Modular Input". Below the header is a description of the app: "This is a Splunk Modular Input for executing commands and indexing the output. It is quite simply just a wrapper around whatever system commands/programs that you want to periodically execute and capture the output from ie: (top, ps, iostat, tshark, tcpdump etc...). It will work on all supported Splunk platforms.". At the bottom of the page, there is a summary: "Category: Utilities | Author: Damien Dallimore | Downloads: 1061 | Released: 3 years ago | Last Updated: 2 years ago | View on Splunkbase". On the right side of the page, there is a green "Install" button with a red oval highlighting it.

- Enter your Splunk.com credentials, check the checkbox to accept the terms and conditions, and click on **Login and Install**. Splunk should return with a message saying that the app was installed successfully.

Login X

Enter your Splunk.com username and password to download the app

Username

Password

[Forgot your password?](#)

This app is provided by a third party and your rights to use the app is in accordance with the license provided by that third-party licensor. Splunk is not responsible for any third-party apps and does not provide any warranty or support. If you have any questions, complaints or claims with respect to this app, please contact the licensor directly, whose contact information can be found on the download page.

[Splunk Software License Agreement](#)

[Splunk Websites Terms and Conditions of Use](#)

I have read the terms and conditions of the license and agree to be bound by them. I accept that Splunk will securely send my login credentials over the Internet to splunk.com

[Cancel](#) [Login and Install](#)

7. From the menu in the top right-hand corner, click on the **Settings** menu and then click on the **Add Data** link.



- If you are prompted to take a quick tour, click on **Skip**.
- In the **How do you want to add data?** section, click on **monitor**.



- Click on the **Command** section.

The screenshot shows the 'Add Data' configuration process at the 'Select Source' step. The 'Command' section is highlighted with a red circle.

- Files & Directories**: Upload a file, index a local file, or monitor an entire directory.
- HTTP Event Collector**: Configure tokens that clients can use to send data over HTTP or HTTPS.
- TCP / UDP**: Configure Splunk to listen on a network port.
- Scripts**: Get data from from any API, service, or database with a script.
- Command**: Command input wrapper for executing commands and indexing the output.

- In the **Mod Input Name** field, enter a name for the input of **SystemInfo**. If you are using Linux, enter `/usr/bin/vmstat` in the **Command Name** field. If you are using Windows, enter `C:\Windows\System32\systeminfo.exe` in the **Command Name** field.

Mod Input Name *	<input type="text" value="SystemInfo"/>
Name of this command input	
Command Name *	<input type="text" value="/usr/bin/vmstat"/>
Name of the system command if on the PATH (ps), or if not, the full path to the command (/bin/ps). Environment variables in the format \$VARIABLE\$ can be included and they will be substituted ie: \${\$SPLUNK_HOME\$}	

Use the full path if the command to be executed cannot be found on the system [PATH](#).

12. In the **Command Arguments** field, enter any argument that needs to be passed to the command listed in the **Command Name** field. In the **Command Execution Interval** field, enter a value in seconds for how often the command should be executed (in this case, we will use 60 seconds). If the output is streamed, then leave this field empty and check the **Streaming Output** field.

Command Arguments	<input type="text"/>
Arguments string for the command. Environment variables in the format \$VARIABLE\$ can be included and they will be substituted ie: \$SPLUNK_HOME\$	
Streaming Output ?	<input type="checkbox"/>
Whether or not the command output is streaming(std out remains open) or not(results received and std out is closed). If it is streaming then "Execution Interval" won't really be relevant.	
Command Execution Interval	<input type="text" value="60"/>
Interval time in seconds to execute the command, defaults to 60 seconds	

13. In the **Source type** section, you have the option to either select a predefined source type or select **Manual** and enter a value. For the purpose of this recipe, select **Manual** as the source type and enter [cp01_modular_input](#) as the value for the source type.
14. Click **Next**.
15. If everything was successful, you should see a **Modular input has been created successfully** message.



16. Click on the **Start searching** button. The **Search & Reporting** app will open with the search already populated based on the settings supplied earlier in the recipe. Splunk is now configured to execute the modular input you provided, every 60 seconds, in accordance with the specified interval. You can search for the data returned by the scripted input using the following search over all time:

[sourcetype=cp01_modular_input](#)

How it works...

Modular inputs are bundled as Splunk apps and, once installed, contain all the necessary configuration and code to display them in the **Data inputs** section of Splunk. In this recipe, you installed a modular input application that allows for periodic execution of commands. You configured the command to execute every minute and index the results of the command each time, giving the results a source type of `cp01_modular_input`.

Modular inputs can be written in a number of languages and need to follow only a set of interfaces that expose the configuration options and runtime behaviors. Depending on the design of the input, they will either run persistently or run on an interval and will send data to Splunk as they receive it.

TIP

You can find several other modular inputs, including REST API, SNMP, and PowerShell, on the Splunk Apps site (<http://splunkbase.splunk.com>).

There's more...

To learn how to create your own modular input, refer to the *Modular Inputs* section of the *Developing Views and Apps for Splunk Web* manual located at <http://docs.splunk.com/Documentation/Splunk/latest/AdvancedDev/ModInputsIntro>.

Using the Universal Forwarder to gather data

Most IT environments today range from multiple servers in the closet of your office to hundreds of endpoint servers located in multiple geographically distributed data centers.

When the data we want to collect is not located directly on the server where Splunk is installed, the Splunk UF can be installed on your remote endpoint servers and used to forward data back to Splunk to be indexed.

The UF is similar to the Splunk server in that it has many of the same features, but it does not contain Splunk Web and doesn't come bundled with the Python executable and libraries. Additionally, the UF cannot process data in advance, such as performing line breaking and timestamp extraction.

This recipe will guide you through configuring the Splunk UF to forward data to a Splunk indexer and will show you how to set up the indexer to receive the data.

Getting ready

To step through this recipe, you will need a server with the Splunk UF installed but not configured. You will also need a running Splunk server. No other prerequisites are required.

TIP

To obtain the UF software, you need to go to http://www.splunk.com/en_us/download.html and register for an account if you do not already have one. Then, either download the software directly to your server or download it to your laptop or workstation and upload it to your server via a file-transfer process such as SFTP.

How to do it...

Follow the steps in the recipe to configure the Splunk Forwarder to forward data and the Splunk indexer to receive data:

1. On the server with the UF installed, open a command prompt if you are a Windows user or a terminal window if you are a Unix user.
2. Change to the `$SPLUNK_HOME/bin` directory, where `$SPLUNK_HOME` is the directory in which the Splunk Forwarder was installed.

For Unix, the default installation directory will be `/opt/splunkforwarder/bin`. For Windows, it will be `C:\Program Files\SplunkUniversalForwarder\bin`.

TIP

If using Windows, omit `./` in front of the Splunk command in the upcoming steps.

3. Start the Splunk Forwarder if not already started, using the following command:

```
./splunk start
```

4. Accept the license agreement.
5. Enable the UF to autostart, using the following command:

```
./splunk enable boot-start
```

6. Set the indexer that this UF will send its data to. Replace the host value with the value of the indexer as well as the username and password for the UF:

```
./splunk add forward-server <host>:9997 -auth  
<username>:<password>
```

The username and password to log in to the Forwarder (default is `admin:changeme`) is `<username>:<password>`.

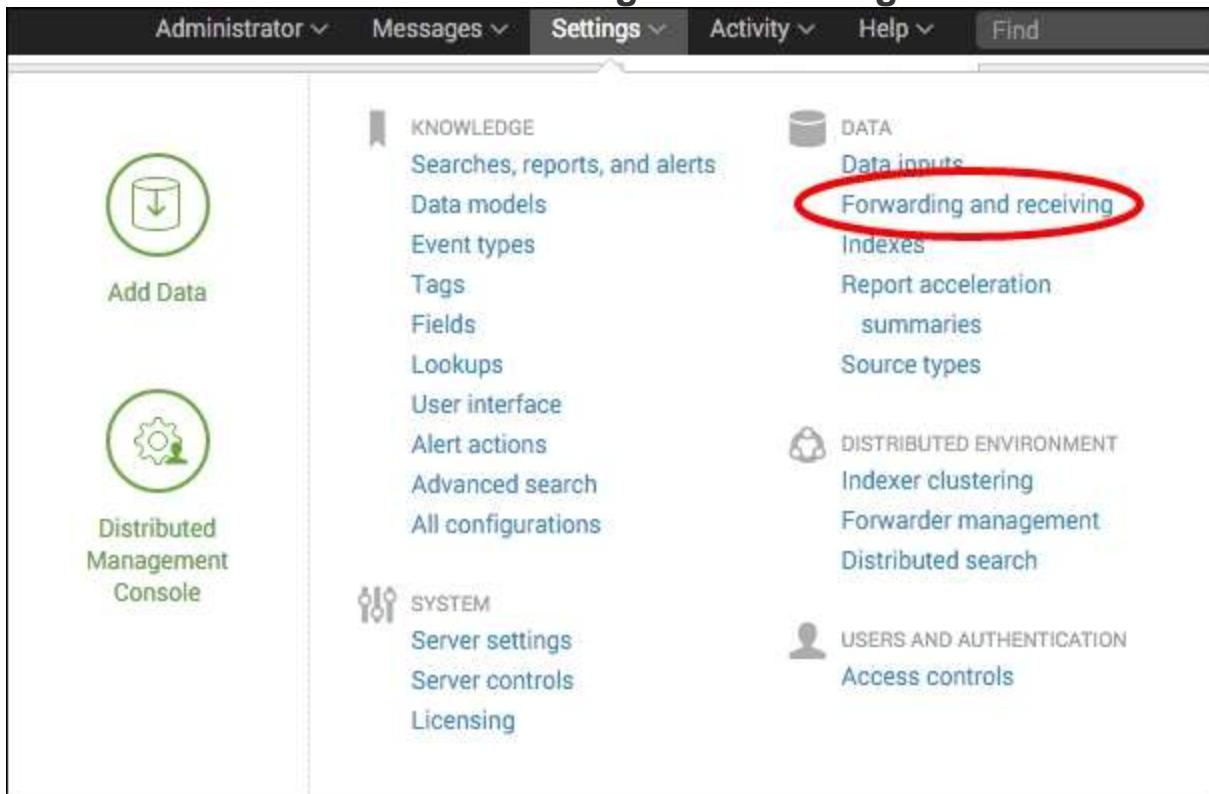
TIP

Additional receiving indexers can be added in the same way by repeating the command in the previous step with a different indexer host or IP. Splunk will automatically load balance the forwarded data if more than one receiving indexer is specified in this manner. Port 9997 is the default

Splunk TCP port and should only be changed if it cannot be used for some reason.

On the receiving Splunk indexer servers:

1. Log in to your receiving Splunk indexer server. From the home launcher, in the top right-hand corner click on the **Settings** menu item and then select the **Forwarding and receiving** link.



2. Click on the **Configure receiving** link.

A screenshot of the 'Receive data' configuration page. It has a green header 'Receive data' and a sub-header 'Configure this instance to receive data forwarded from other instances.' Below this is a large input field containing the text 'Configure receiving', which is circled in red.

3. Click on **New**.
4. Enter **9997** in the **Listen on this port** field.

Configure receiving

Set up this Splunk instance to receive data from forwarder(s).

Listen on this port *

9997

For example, 9997 will receive data on TCP port 9997.

5. Click on **Save** and restart Splunk. The UF is installed and configured to send data to your Splunk server, and the Splunk server is configured to receive data on the default Splunk TCP port 9997.

How it works...

When you tell the forwarder which server to send data to, you basically add a new configuration stanza into an `outputs.conf` file behind the scenes. On the Splunk server, an `inputs.conf` file will contain a `[splunktcp]` stanza to enable receiving. The `outputs.conf` file on the Splunk forwarder will be located in `$SPLUNK_HOME/etc/system/local`, and the `inputs.conf` file on the Splunk server will be located in the local directory of the app you were in (the launcher app in this case) when configuring receiving.

Using forwarders to collect and forward data has many advantages. The forwarders communicate with the indexers on TCP port 9997 by default, which makes for a very simple set of firewall rules that need to be opened. Forwarders can also be configured to load balance their data across multiple indexers, increasing search speeds and availability. Additionally, forwarders can be configured to queue the data they collect if communication with the indexers is lost. This can be extremely important when collecting data that is not read from logfiles, such as performance counters or syslog streams, as the data cannot be re-read.

There's more...

While configuring the settings of the UF can be performed via the command-line interface of Splunk, as outlined in this recipe, there are several other methods to update the settings quickly and allow for customization of the many configuration options that Splunk provides.

ADD THE RECEIVING INDEXER VIA OUTPUTS.CONF

The receiving indexers can be directly added to the `outputs.conf` configuration file on the UF. Edit `$SPLUNK_HOME/etc/system/local/outputs.conf`, add your input, and then, restart the UF. The following example configuration is provided, where two receiving indexers are specified. The `[tcpout-server]` stanza can be leveraged to add output configurations specific to an individual receiving indexer:

```
[tcpout]
defaultGroup = default-autolb-group

[tcpout:default-autolb-group]
disabled = false
server = mysplunkindexer1:9997,mysplunkindexer2:9997

[tcpout-server://mysplunkindexer1:9997]
[tcpout-server://mysplunkindexer2:9997]
```

TIP

If nothing has been configured in `inputs.conf` on the UF, but `outputs.conf` is configured with at least one valid receiving indexer, the Splunk forwarder will only send internal forwarder health-related data to the indexer. It is, therefore, possible to configure a forwarder correctly and be detected by the Splunk indexers, but not actually send any real data.

Loading an Access Log

While most of the data you will index with Splunk will be collected in real time, there might be instances where you have a set of data that you would like to put into Splunk, either to backfill some missing or incomplete data, or just to take advantage of its searching and reporting tools.

This recipe will show you how to perform one-time bulk loads of data from files located on the Splunk server. We will also use this recipe to load the data samples that will be used throughout the subsequent sessions as we build our operational intelligence app in Splunk.

There are two files that make up our sample data.

- The first is `access_log`, which represents the data from our web layer and is modeled on an Apache web server.
- The second file is `app_log`, which represents the data from our application layer and is modeled on `log4j` log data from our custom middleware application.

Getting ready

To step through this recipe, you will need a running Splunk server and you should have a copy of the sample data generation app (`OpsDataGen.spl`) for this book.

How to do it...

Follow the given steps to load the sample data generator on your system:

1. Log in to your Splunk server using your credentials.
2. From the **Apps** menu in the upper left-hand corner of the home screen, click on the gear icon.



3. The **Apps settings** page will load. Then click on the **Install app from file** button.



4. Select the location of the `OpsDataGen.spl` file on your computer, and then, click on the **Upload** button to install the application.

Upload an app

If you have a .spl or .tar.gz app file to install, you can upload it using this form.

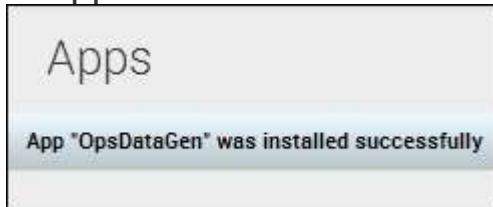
You can replace an existing app via the Splunk CLI. [Learn more.](#)

File

Choose File **OpsDataGen.spl**

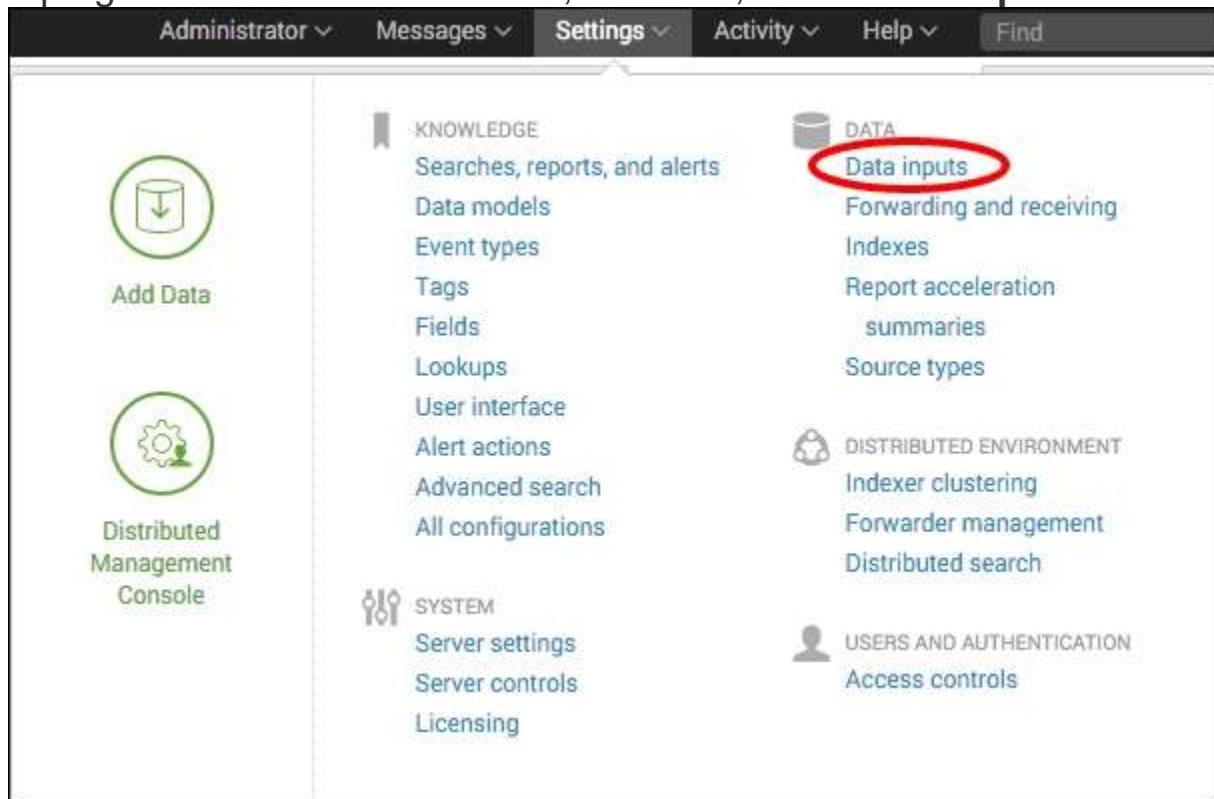
Upgrade app. Checking this will overwrite the app if it already exists.

5. After installation, a message should appear in a blue bar at the top of the screen, letting you know that the app has installed successfully. You should also now see the `OpsDataGen` app in the list of apps.



6. By default, the app installs with the data-generation scripts disabled. In order to generate data, you will need to enable either a

Windows or Linux script, depending on your Splunk operating system. To enable the script, select the **Settings** menu from the top right-hand side of the screen, and then, select **Data inputs**.



7. From the **Data inputs** screen that follows, select **Scripts**.
8. On the **Scripts** screen, locate the **OpsDataGen** script for your operating system and click on **Enable**.
 - For Linux, it will
be `$SPLUNK_HOME/etc/apps/OpsDataGen/bin/AppGen.path`
 - For Windows, it will
be `$SPLUNK_HOME/etc/apps/OpsDataGen/bin/AppGen-win.path`

The following screenshot displays both the Windows and Linux inputs that are available after installing the **OpsDataGen** app. It also displays where to click to enable the correct one based on the operating system Splunk is installed on.

Command	Interval	Source type
<code>\$SPLUNK_HOME/etc/apps/OpsDataGen/bin/AppGen.path</code>	Linux	AppGenLogs
<code>\$SPLUNK_HOME/etc/apps/OpsDataGen/bin/AppGen-win.path</code>	Windows	AppGenLogs

9. Select the **Settings** menu from the top right-hand side of the screen, select **Data inputs**, and then select **Files & directories**.
10. On the **Files & directories** screen, locate the two **OpsDataGen** inputs for your operating system and for each click on **Enable**.
 - For Linux, it will be `$SPLUNK_HOME/etc/apps/OpsDataGen/data/access_log` and `$SPLUNK_HOME/etc/apps/OpsDataGen/data/app_log`
 - For Windows, it will be `$SPLUNK_HOME\etc\apps\OpsDataGen\data\access_log` and `$SPLUNK_HOME\etc\apps\OpsDataGen\data\app_log`

The following screenshot displays both the Windows and Linux inputs that are available after installing the **OpsDataGen** app. It also displays where to click to enable the correct one based on the operating system Splunk is installed on.

Full path to your data	Set host	Source type	Set the destination index	Number of files	App	Status
<code>\$SPLUNK_HOME/etc/apps/OpsDataGen/data/access_log</code>	Constant Value	access_combined	main	Linux	OpsDataGen	Disabled
<code>\$SPLUNK_HOME/etc/apps/OpsDataGen/data/app_log</code>	Constant Value	log4j	main	Linux	OpsDataGen	Disabled
<code>\$SPLUNK_HOME\etc\apps\OpsDataGen\data\access_log</code>	Constant Value	access_combined	main	Windows	OpsDataGen	Disabled
<code>\$SPLUNK_HOME\etc\apps\OpsDataGen\data\app_log</code>	Constant Value	log4j	main	Windows	OpsDataGen	Disabled

11. The data will now be generated in real time. You can test this by navigating to the Splunk search screen and running the following search over an All time (real-time) time range:

```
index=main sourcetype=log4j OR sourcetype=access_combined
```

12. After a short while, you should see data from both the source types flowing into Splunk. The data generation is now working, as displayed in the following screenshot:

The screenshot shows the Splunk web interface with a search bar containing the query "index=main sourcetype=log4j OR sourcetype=access_combined". Below the search bar, it says "20 of 20 events matched · No Event Sampling". The main area displays a table of search results with columns for Time, Event, and several dropdown menus for filtering fields like host, source, and sourcetype.

< Hide Fields		All Fields	Time	Event
Selected Fields			> 4/19/16 3:11:26.000 AM	143.77.9.0 - - [19/Apr/2016:03:11:26 +0000] "GET /home HTTP/1.1" 200 1407 "http://www.yahoo.com" "Mozilla/5.0 (Windows NT 5.1) Gecko/20100101 Firefox/14.0 Opera/12.0" "JSESSIONID=52B00F6511172E7F451B96A680591619" 49 host = ip-172-31-12-177 source = /opt/splunk/etc/apps/OpsDataGen/data/access.log sourcetype = access_combined
Interesting Fields			> 4/19/16 3:11:26.000 AM	47.91.235.14 - - [19/Apr/2016:03:11:26 +0000] "GET /home HTTP/1.1" 200 2971 "http://www1.samplesite.ca/updatecart" "Mozilla/5.0 (Windows NT 5.1) Gecko/20100101 Firefox/14.0 Opera/12.0" "JSESSIONID=5A0C47840F2A39536CF5142717F7E1E6" 33 host = ip-172-31-12-177 source = /opt/splunk/etc/apps/OpsDataGen/data/access.log sourcetype = access_combined
# bytes 6			> 4/19/16 3:11:26.000 AM	47.91.235.14 - - [19/Apr/2016:03:11:26 +0000] "GET /updatecart?orderId=1461035435&item=4728475&qty=2 HTTP/1.1" 302 2137 "https://www1.samplesite.ca/viewCart" "Mozilla/5.0 (Windows NT 5.1) Gecko/20100101 Firefox/14.0 Opera/12.0" "JSESSIONID=5A0C47840F2A39536CF5142717F7E1E6" 40

How it works...

In this case, you installed a Splunk application that leverages a scripted input. The script we wrote generates data for two source types.

The `access_combined` source type contains sample web access logs, and the `log4j` source type contains application logs. These data sources will be used throughout the recipes in this course. Applications will also be discussed in more detail later on.

Defining field extractions

Splunk has many built-in features, including knowledge on several common source types, which lets it automatically know which fields exist within your data. Splunk, by default, also extracts any key-value pairs present within the log data and all the fields within the JSON-formatted logs. However, often the fields within raw log data cannot be interpreted out of the box, and this knowledge must be provided to Splunk in order to make these fields easily searchable.

The sample data that we will be using in subsequent sessions contains data we wish to present as fields to Splunk. Much of the raw log data contains key-value fields that Splunk will extract automatically, but there is one field we need to tell Splunk how to extract, **representing the page response time**. To do this, we will be adding a custom field extraction, which will tell Splunk how to extract the field for us.

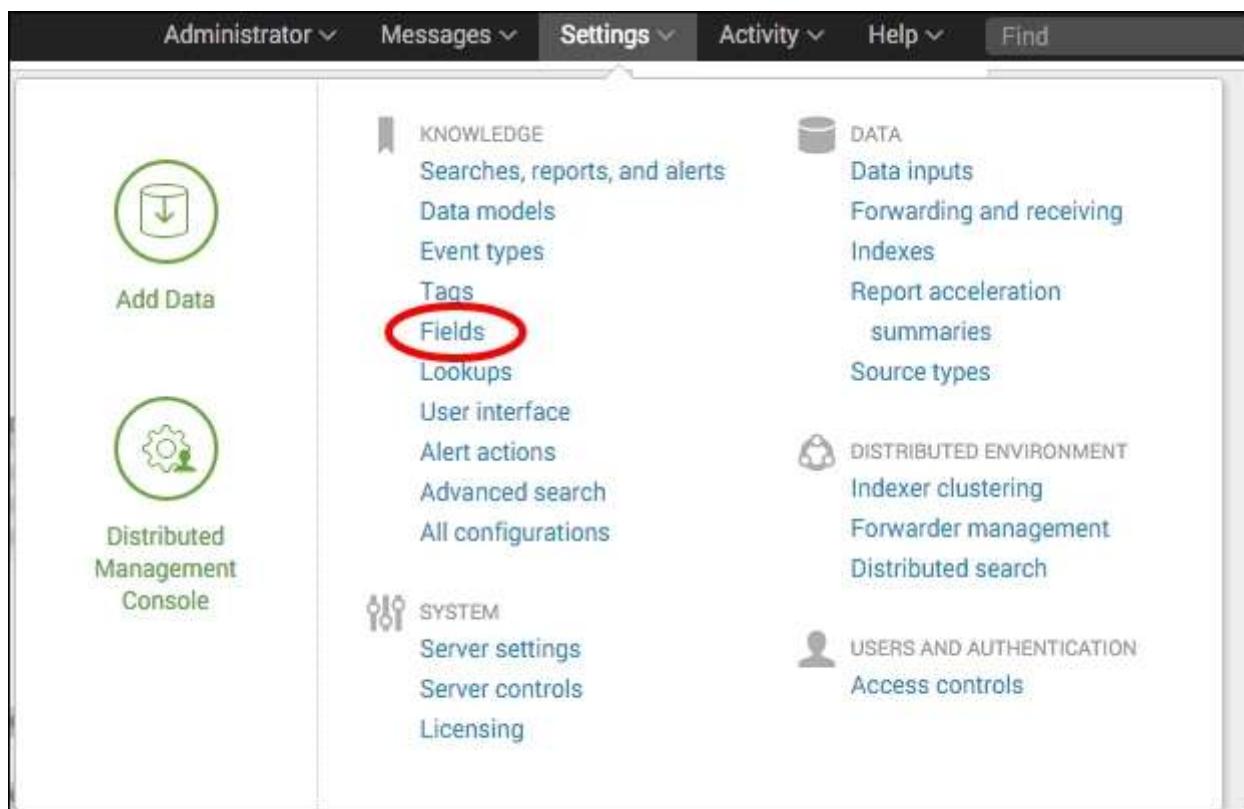
Getting ready

To step through this recipe, you will need a running Splunk server with the operational intelligence sample data loaded. No other prerequisites are required.

How to do it...

Follow the given steps to add a custom field extraction for response:

1. Log in to your Splunk server.
2. In the top right-hand corner, click on the **Settings** menu and then click on the **Fields** link.



3. Click on the **Field extractions** link.

Field extractions

View and edit all field extractions. Add new field extractions and update permissions.

4. Click on **New**.

5. In the **Destination app** field, select the **search** app, and in the **Name** field, enter `response`. Set the **Apply to** drop-down list to **sourcetype** and the **named** field to `access_combined`. Set the **Type** drop-down list to **Inline**, and for the **Extraction/Transform** field, carefully enter the `(?i)^(?:[^"]*"){8}\s+(?P<response>.+)` **regex**.

Destination app *

Name *

Apply to * named *

sourcetype access_combine

Type *

Inline

Extraction/Transform *

```
(?i)^(?:[^"]*"){8}is+(?P<response>.*)
```

6. Click on **Save**.
 7. On the **Field extractions** listing page, find the recently added extraction, and in the **Sharing** column, click on the **Permissions** link.
- 
8. Update the **Object should appear in** setting to **All apps**. In the **Permissions** section, for the **Read** column, check **Everyone**, and in the **Write** column, check **admin**. Then, click on **Save**.

Object should appear in

Keep private This app only (search) All apps

Permissions

Roles	Read	Write
Everyone	<input checked="" type="checkbox"/>	<input type="checkbox"/>
admin	<input type="checkbox"/>	<input checked="" type="checkbox"/>
can_delete	<input type="checkbox"/>	<input type="checkbox"/>
power	<input type="checkbox"/>	<input type="checkbox"/>
splunk-system-role	<input type="checkbox"/>	<input type="checkbox"/>
user	<input type="checkbox"/>	<input type="checkbox"/>

9. Navigate to the Splunk search screen and enter the following search over the **Last 60 minutes** time range:

```
index=main sourcetype=access_combined
```

10. You should now see a field called **response extracted** on the left-hand side of the search screen under the **Interesting Fields** section.

How it works...

All field extractions are maintained in the `props.conf` and `transforms.conf` configuration files. The stanzas in `props.conf` include an extraction class that leverages regular expressions to extract field names and/or values to be used at search time. The `transforms.conf` file goes further and can be leveraged for more advanced extractions, such as reusing or sharing extractions over multiple sources, source types, or hosts.

Defining event types and tags

Event types in Splunk are a way of categorizing common types of events in your data in order to make them easier to search and report on. One advantage of using event types is that they can assist in applying a common classification to similar events. Event types essentially turn chunks of search criteria into field/value pairs. Tags help you search groups of event data more efficiently and can be assigned to any field/value combination, including event types.

For example, Windows logon events could be given an event type of `windows_logon`, Unix logon events be given an event type of `unix_logon`, and VPN logon events could be given an event type of `vpn_logon`. We could then tag these three event types with a tag of `logon_event`. A simple search for `tag="logon_event"` would then search across the Windows, Unix, and VPN source types and return all the logon events. Alternatively, if we want to search only for Windows logon events, we will search for `eventtype=windows_logon`.

This recipe will show how to define event types and tags for use with the sample data. Specifically, you will define an event type for successful web server events.

NOTE

For more information on event types and tags in Splunk, check out:

- <http://docs.splunk.com/Documentation/Splunk/latest/Knowledge/Abouteventtypes>
- <http://docs.splunk.com/Documentation/Splunk/latest/Knowledge/Abouttagsandaliases>

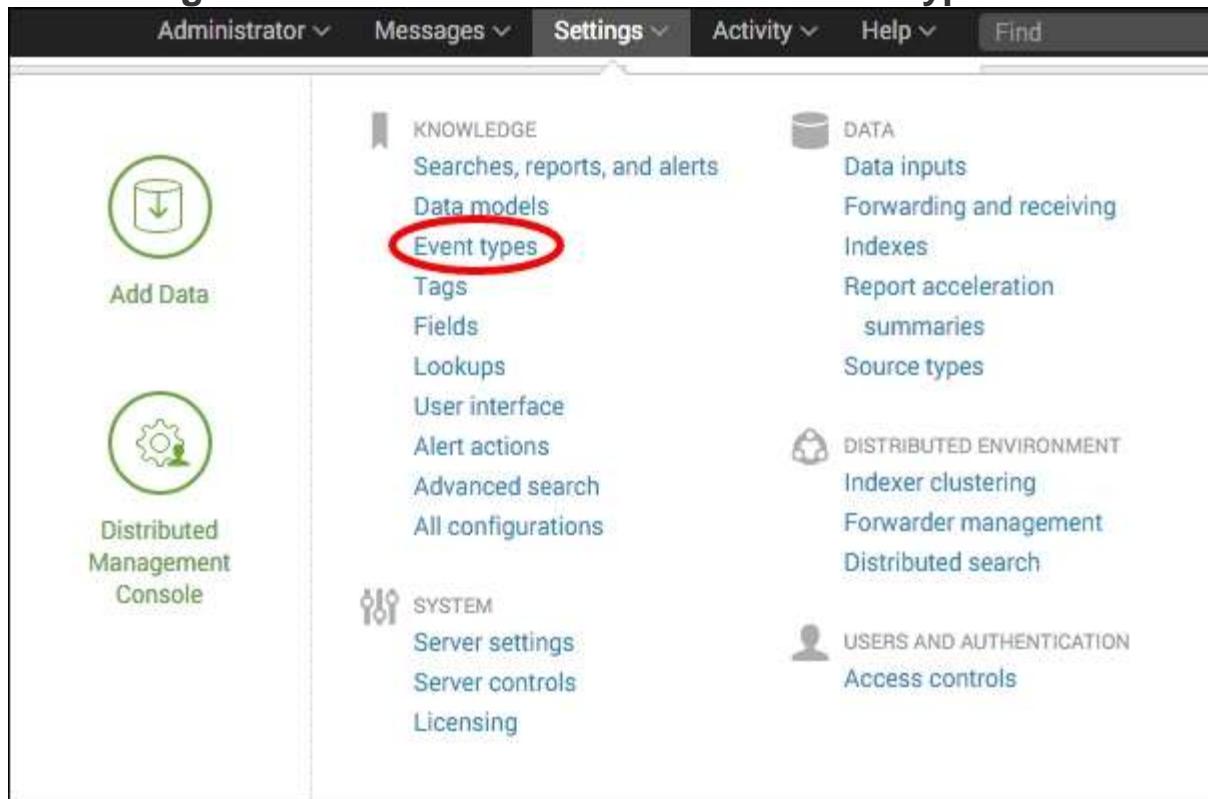
Getting ready

To step through this recipe, you will need a running Splunk server with the operational intelligence sample data loaded. No other prerequisites are required.

How to do it...

Follow the given steps to define an event type and associated tag:

1. Log in to your Splunk server.
2. From the home launcher in the top right-hand corner, click on the **Settings** menu item and then click on the **Event types** link.



3. Click on the **New** button.
4. In the **Destination App** drop-down list, select **search**. Enter `HttpRequest-Success` in the **Name** field. In the **Search string** text area, enter `sourcetype=access_combined status=2*`. In the **Tag(s)** field, enter `webserver`, and then click on **Save**.

Destination App *	<input type="text" value="search"/>
Name *	<input type="text" value="HttpRequest-Success"/>
Search string *	<input type="text" value="sourcetype=access_combined status=2*"/>
Tag(s)	<input type="text" value="webserver"/> <small>Enter a comma-separated list of tags.</small>

5. The event type is now created. To verify that this worked, you should now be able to search by both the event type and the tag that you created. Navigate to the Splunk search screen in the **Search & Reporting** app and enter the following search over the **Last 60 minutes** time range to verify that the `eventtype` is working:

```
eventtype="HttpRequest-Success"
```

6. Enter the following search over the **Last 60 minutes** time range to verify that the tag is working:

```
tag="webserver"
```

How it works...

Event types are applied to events at search time and introduce an `eventtype` field with user-defined values that can be used to quickly sift through large amounts of data. An event type is essentially a Splunk search string that is applied against each event to see if there is a match. If the event type search matches the event, the `eventtype` field is added, with the value of the field being the user-defined name for that event type.

The common tag value allows for a grouping of event types. If multiple event types had the same tag, then your Splunk search could just search for that particular tag value, instead of needing to list out each individual event type value.

Event types can be added, modified, and deleted at any time without the need to change or reindex your data, as they are applied at search time.

Event types are stored in `eventtypes.conf` in either the `$SPLUNK_HOME/etc/system/local/` or a custom app directory.

There's more...

While adding of event types and tags can be done through the web interface of Splunk, as outlined in this recipe, there are other approaches to add them in bulk quickly and allow for customization of the many configuration options that Splunk provides.

ADDING EVENT TYPES AND TAGS VIA EVENTTYPES.CONF AND TAGS.CONF

Event types in Splunk can be manually added to the `eventtypes.conf` configuration files. Edit or create `$SPLUNK_HOME/etc/system/local/eventtypes.conf` and add your event type. You will need to restart Splunk after this:

```
[HttpRequest-Success]
search = status=2*
```

Tags in Splunk can be manually added to the `tags.conf` configuration files. Edit or create `$SPLUNK_HOME/etc/system/local/tags.conf` and add your tag. You will need to restart Splunk after this:

```
[eventtype=HttpRequest-Success]
webserver = enabled
```

TIP

In this recipe, you tagged an event type. However, tags do not always need to be associated with event types. You can tag any field/value combination found in an event. To create new tags independently, click on the **Settings** menu and select **Tags**.

More Splunk .conf files

props.conf

The stanzas in `props.conf` define which events to match based on host, source, and sourcetype. These stanzas are merged into the master configuration based on the uniqueness of stanza and attribute names, as with any other configuration, but there are specific rules governing when each stanza is applied to an event and in what order. Stated as simply as possible, attributes are sorted by type, then by priority, and then by the ASCII value.

We'll cover those rules under the *Stanza types* section. First, let's look at common attributes.

COMMON ATTRIBUTES

The full set of attributes allowed in `props.conf` is vast. Let's look at the most common attributes and try to break them down by the time when they are applied.

Search-time attributes

The most common attributes that users will make in `props.conf` are field extractions. When a user defines an extraction through the web interface, it ends up in `props.conf`, as shown here:

```
[my_source_type]
EXTRACT-foo = \s(?:<bar>\d+)ms
EXTRACT-cat = \s(?:<dog>\d+)s
```

This configuration defines the fields bar and dog for the `my_source_type` source type. Extracts are the most common search-time configurations. Any of the stanza types listed under the *Stanza*

types section can be used, but the source type is definitely the most common one.

Other common search-time attributes include:

- `REPORT-foo = bar`: This attribute is a way to reference stanzas in `transforms.conf`, but apply them at search time instead of index time. This approach predates `EXTRACT` and is still useful in a few special cases.

We will cover this case later under the `transforms.conf` section.

- `KV_MODE = auto`: This attribute allows you to specify whether Splunk should automatically extract fields in the form of `key=value` from events. The default value is `auto`. The most common change is to disable automatic field extraction for performance reasons by setting the value to `none`. Other possibilities are `multi`, `JSON`, and `XML`.
- `LOOKUP-foo = mylookup barfield`: This attribute lets you wire up a lookup to automatically run for a set of events. The lookup itself is defined in `transforms.conf`.

Index-time attributes

As discussed in [Chapter 3, Tables, Charts, and Fields](#), it is possible to add fields to the metadata of events. This is accomplished by specifying a transform in `transforms.conf`, and an attribute in `props.conf`, to tie the transformation to specific events.

The attribute in `props.conf` looks as follows: `TRANSFORMS-foo = bar1,bar2`.

This attribute references stanzas in `transforms.conf` by name, in this case, `bar1` and `bar2`. These transform stanzas are then applied to the events matched by the stanza in `props.conf`.

Parse-time attributes

Most of the attributes in `props.conf` actually have to do with parsing events. To successfully parse events, a few questions need to be answered, such as these:

- When does a new event begin? Are events multiline? Splunk will make fairly intelligent guesses, but it is best to specify an exact setting. Attributes that help with this include:
 - `SHOULD_LINEMERGE = false`: If you know that your events will never contain the newline character, setting this to `false` will eliminate a lot of processing.
 - `BREAK_ONLY_BEFORE = ^\d\d\d\d-\d\d-\d\d`: If you know that new events always start with a particular pattern, you can specify it using this attribute.
 - `TRUNCATE = 1024`: If you are certain you only care about the first *n* characters of an event, you can instruct Splunk to truncate each line. What is considered a line can be changed with the next attribute.
 - `LINE_BREAKER = ([\r\n]+)(?=^\d{4}-\d\d-\d\d)`: This is the most efficient approach to multiline events is to redefine what Splunk considers a line. This example says that a line is broken on any number of newlines followed by a date of the form 1111-11-11. The big disadvantage to this approach is that, if your log changes, you will end up with garbage in your index until you update your configuration. Try the `props` helper app available at Splunkbase for help in making this kind of configuration.
- Where is the date? If there is no date, see `DATETIME_CONFIG` further down this bullet list. The relevant attributes are as follows:
 - `TIME_PREFIX = ^\[:`: By default, dates are assumed to fall at the beginning of the line. If this is not true, give Splunk some help and move the cursor past the characters preceding the date. This pattern is applied to each line, so if you have redefined `LINE_BREAKER` correctly, you can be sure only the beginnings of actual multiline events are being tested.
 - `MAX_TIMESTAMP_LOOKAHEAD = 30`: Even if you change no other setting, you should change this one. This setting says how far

after `TIME_PREFIX` to test for dates. With no help, Splunk will take the first 150 characters of each line and then test regular expressions to find anything that looks like a date. The default regular expressions are pretty lax, so what it finds may look more like a date than the actual date. If you know that your date is never more than n characters long, set this value to `n` or `n+2`. Remember that the characters retrieved come after `TIME PREFIX`.

- What does the date look like? These attributes will be of assistance here:
 - `TIME_FORMAT = %Y-%m-%d %H:%M:%S.%3N %:z`: If this attribute is specified, Splunk will apply `strptime` to the characters immediately following `TIME_PREFIX`. If this matches, then you're done. This is, by far, the most efficient and least error-prone approach. Without this attribute, Splunk actually applies a series of regular expressions until it finds something that looks like a date.
 - `DATETIME_CONFIG = /etc/apps/a/custom_datetime.xml`: As mentioned, Splunk uses a set of regular expressions to determine the date. If `TIME_FORMAT` is not specified, or won't work for some strange reason, you can specify a different set of regular expressions or disable time extraction completely by setting this attribute to `CURRENT`(the indexer clock time) or `NONE` (file modification time, or if there is no file, clock time). I personally have never had to resort to a custom `datetime.xml` file, though I have heard of it being done.
 - The Data preview function available when you are adding data through the manager interface builds a good, usable configuration. The configuration generated does not use `LINE_BREAKER`, which is definitely safer but less efficient.

Here is a sample stanza that uses LINE BREAKER for efficiency:

```
[mysourcetype]
TIME_FORMAT = %Y-%m-%d %H:%M:%S.%3N %:z
MAX_TIMESTAMP_LOOKAHEAD = 32
TIME_PREFIX = ^\[[
SHOULD_LINEMERGE = False
LINE_BREAKER = ([\r\n]+)(?=\[\d{4}-\d{1,2}-\d{1,2}\]\s+|
```

```

d{1,2}:\d{1,2}:\d{1,2})
TRUNCATE = 1024000
This configuration would apply to log messages that looked like
this:
[2011-10-13 13:55:36.132 -07:00] ERROR Interesting message.
More information.
And another line.
[2011-10-13 13:55:36.138 -07:00] INFO All better.
[2011-10-13 13:55:37.010 -07:00] INFO More data
and another line.

```

Let's walk through how these settings affect the first line of this sample configuration:

- `LINE_BREAKER` states that a new event starts when one or more newline characters are followed by a bracket and series of numbers and dashes in the `[1111-11-11 11:11:11]` pattern.
- `SHOULD_LINEMERGE=False` tells Splunk not to bother trying to recombine multiple lines.
- `TIME_PREFIX` moves the cursor to the character after the `[` character.
- `TIME_FORMAT` is tested against the characters at the current cursor location. If it succeeds, we are done.
- If `TIME_FORMAT` fails, `MAX_TIMESTAMP_LOOKAHEAD` characters are read from the cursor position (after `TIME_PREFIX`) and the regular expressions from `DATE_CONFIG` are tested.
- If the regular expressions fail against the characters returned, the time last parsed from an event is used. If there is no last time parsed, the modification date from the file would be used, if known; otherwise, the current time would be used.

This is the most efficient and precise way to parse events in Splunk, but also the most brittle. If your date format changes, you will almost certainly have junk data in your index. Only use this approach if you are confident the format of your logs will not change without your knowledge.

Input-time attributes

There are only a couple of attributes in `props.conf` that matter at the input stage, but they are generally not needed:

- `CHARSET = UTF-16LE`: When reading data, Splunk has to know the character set used in the log. Most applications write their logs using either `ISO-8859-1` or `UTF-8` is handled by the default settings just fine. Some Windows applications write logs in 2-byte Little Endian, which is indexed as garbage. Setting `CHARSET = UTF-16LE` takes care of the problem. Check the official documentation for a list of supported encodings.
- `NO_BINARY_CHECK = true`: If Splunk believes that a file is binary, it will not index the file at all. If you find that you have to change this setting to convince Splunk to read your files, it is likely that the file is in an unexpected character set. You might try other `CHARSET` settings before enabling this setting.

STANZA TYPES

Now that we have looked at common attributes, let's talk about the different types of stanzas in `props.conf`. Stanza definitions can take the following three forms:

- `[foo]`
 - This is the exact name of a source type and is the most common type of stanza to be used; the source type of an event is usually defined in `inputs.conf`
 - Wildcards are not allowed
- `[source:::/logs/.../*.log]`
 - This matches the source attribute, which is usually the path to the log where the event came from
 - `*` matches a file or directory name
 - `...` matches any part of a path
- `[host:::*nyc*]`
 - This matches the host attribute, which is usually the value of the hostname on a machine running Splunk Forwarder
 - `*` is allowed

Types follow this order in taking precedence:

1. Source.
2. Host.
3. Source type.

For instance, say an event has the following fields:

```
sourcetype=foo_type  
source=/logs/abc/def/gh.log  
host=dns4.nyc.mycompany.com
```

Given this configuration snippet and our preceding event, we have the following code:

```
[foo_type]  
TZ = UTC  
[source:::/logs/.../*.log]  
TZ = MST  
[host::*nyc*]  
TZ = EDT
```

`TZ = MST` would be used during parsing because the source stanza takes precedence.

To extend this example, say we have this snippet:

```
[foo_type]  
TZ = UTC  
TRANSFORMS-a = from_sourcetype  
[source:::/logs/.../*.log]  
TZ = MST  
BREAK_ONLY_BEFORE_DATE = True  
TRANSFORMS-b = from_source  
[host::*nyc*]  
TZ = EDT  
BREAK_ONLY_BEFORE_DATE = False  
TRANSFORMS-c = from_host
```

The attributes applied to our event would, therefore, be as shown here:

```
TZ = MST  
BREAK_ONLY_BEFORE_DATE = True  
TRANSFORMS-a = from_sourcetype
```

```
TRANSFORMS-b = from_source  
TRANSFORMS-c = from_host
```

PRIORITIES INSIDE A TYPE

If there are multiple source or host stanzas that match a given event, the order in which settings are applied also comes into play. A stanza with a pattern has a priority of `0`, while an exact stanza has a priority of `100`. Higher priorities win. For instance, say we have the following stanza:

```
[source:::/logs/abc/def/gh.log]  
TZ = UTC  
[source:::/logs/.../*.log]  
TZ = CDT
```

Our `TZ` value will be `UTC` since the exact match of `source:::/logs/abc/def/gh.log` has a higher priority.

When priorities are identical, stanzas are applied by the ASCII order. For instance, say we have this configuration snippet:

```
[source:::/logs/abc/.../*.log]  
TZ = MST  
[source:::/logs/.../*.log]  
TZ = CDT
```

The attribute `TZ=CDT` would win because `/logs/.../*.log` is first in the ASCII order.

This may seem counterintuitive since `/logs/abc/.../*.log` is arguably a better match. The logic for determining what makes a better match, however, can quickly become fantastically complex, so the ASCII order is a reasonable approach.

You can also set your own value of priority, but luckily, it is rarely needed.

ATTRIBUTES WITH CLASS

As you dig into configurations, you will see attribute names of the FOO-bar form.

The word after the dash is generally referred to as the class. These attributes are special in a few ways:

- Attributes merge across files as with any other attribute
- Only one instance of each class will be applied according to the rules described previously
- The final set of attributes is applied in the ASCII order by the value of the class. Once again, say we are presented with an event with the following fields:
 - sourcetype=foo_type
 - source=/logs/abc/def/gh.log
 - host=dns4.nyc.mycompany.com

And, say that this is the configuration snippet:

```
[foo_type]
TRANSFORMS-a = from_sourcetype1, from_sourcetype2
[source::/logs/.../*.log]
TRANSFORMS-c = from_source_b
[source::/logs/abc/.../*.log]
TRANSFORMS-b = from_source_c
[host::*nyc*]
TRANSFORMS-c = from_host
The surviving transforms would then be:
TRANSFORMS-c = from_source_b
TRANSFORMS-b = from_source_c
TRANSFORMS-a = from_sourcetype1, from_sourcetype2
```

To determine the order in which the transforms are applied to our event, we will sort the stanzas according to the values of their classes, in this case, `c`, `b`, and `a`. This gives us:

```
TRANSFORMS-a = from_sourcetype1, from_sourcetype2
```

```
TRANSFORMS-b = from_source_c  
TRANSFORMS-c = from_source_b
```

The transforms are then combined into a single list and executed in this order:

```
from_sourcetype1, from_sourcetype2, from_source_c,  
from_source_b
```

The order of transforms usually doesn't matter, but it is important to understand it if you want to chain transforms and create one field from another. We'll try this later, in the *transforms.conf* section.

inputs.conf

This configuration, as you might guess, controls how data makes it into Splunk.

By the time this data leaves the input stage, it still isn't an event but has some basic metadata associated with it: `host`, `source`, `sourcetype`, and optionally `index`. This basic metadata is then used by the parsing stage to break the data into events according to the rules defined in `props.conf`:

Input types can be broken down into files, network ports, and scripts. First, we will look at attributes that are common to all inputs.

COMMON INPUT ATTRIBUTES

These common bits of metadata are used in the parsing stage to pick the appropriate stanzas in `props.conf`.

- `host`: By default, `host` will be set to the hostname of the machine producing the event. This is usually the correct value, but it can be overridden when appropriate.
- `source`: This field is usually set to the path, file, or network port that an event came from, but this value can be hardcoded.
- `sourcetype`: This field is almost always set in `inputs.conf` and is the primary field to determine which set of parsing rules in `props.conf` to apply to these events.

NOTE

It is very important to set `sourcetype`. In the absence of a value, Splunk will create automatic values based on the source, which can easily result in an explosion of `sourcetype` values.

- `index`: This field says what index to write events to. If it is omitted, the default `index` will be used.

All of these values can be modified using transforms, the only caveat being that these transforms are applied after the parsing step. The practical consequence of this is that you cannot apply different parsing rules to different events in the same file, for instance, different time formats on different lines.

FILES AS INPUTS

The vast majority of events in Splunk come from files. Usually, these events are read from the machine where they are produced and as the logs are written. Very often, the entire input's stanza will look as follows:

```
[monitor:///logs/interesting.log*]
sourcetype=interesting
```

This is often all that is needed. This stanza says:

- Read all logs that match the `/logs/interesting.log*` pattern, and going forward, watch them for new data
- Name the source type "interesting"
- Set the source to the name of the file in which the log entry was found
- Default the host to the machine where the logs originated
- Write the events to the default index

These are usually perfectly acceptable defaults. If `sourcetype` is omitted, Splunk will pick a default source type based on the filename, which you don't want—your source type list will get very messy very fast.

Using patterns to select rolled logs

You may notice that the previous stanza ended in *. This is important because it gives Splunk a chance to find events that were written to a log that has recently rolled. If we simply watch `/logs/interesting.log`, it is likely that events will be missed at the end of the log when it rolls, particularly on a busy server.

There are specific cases where Splunk can get confused, but in the vast majority of cases, the default mechanisms do exactly what you would hope for. See the *When to use crcSalt* section further on for a discussion about special cases.

Using blacklist and whitelist

It is also possible to use a blacklist and whitelist pattern for more complicated patterns. The most common use case is to blacklist files that should not be indexed, for instance, `.gz` and `.zip` files. This can be done as follows:

```
[monitor:///opt/B/logs/access.log*]
sourcetype=access
blacklist=.*.gz
```

This stanza would still match `access.log.2012-08-30`, but if we had a script that compressed older logs, Splunk would not try to read `access.log.2012-07-30.gz`.

Conversely, you can use a whitelist to apply very specific patterns, as shown here:

```
[monitor:///opt/applicationserver/logs]
sourcetype=application_logs
whitelist=(app|application|legacy|foo)\.log(\.\d{4})?
blacklist=.*.gz
```

This whitelist would match `app.log`, `application.log`, `legacy.log.2012-08-13`, and `foo.log`, among others. The blacklist will negate any `.gz` files.

Since a log is a directory, the default behavior will be to recursively scan that directory.

Selecting files recursively

The layout of your logs or your application may dictate a recursive approach.

For instance, say we have these stanzas:

```
[monitor:///opt/*/logs/access.log*]  
sourcetype=access  
[monitor:///opt/.../important.log*]  
sourcetype=important
```

The character `*` will match a single file or directory, while `...` will match any depth. This will match the files you want, with the caveat that all of `/opt` will continually be scanned.

Splunk will continually scan all directories from the first wildcard in a monitor path.

If `/opt` contains many files and directories, which it almost certainly does, Splunk will use an unfortunate amount of resources scanning all directories for matching files, constantly using memory and CPU. I have seen a single Splunk process watching a large directory structure use 2 gigabytes of memory. A little creativity can take care of this, but it is something to be aware of.

The takeaway is that if you know the possible values for `*`, you are better off writing multiple stanzas. For instance, assuming our directories in `/opt` are `A` and `B`, the following stanzas will be far more efficient:

```
[monitor:///opt/A/logs/access.log*]  
sourcetype=access  
[monitor:///opt/B/logs/access.log*]  
sourcetype=access
```

It is also perfectly acceptable to have stanzas matching files and directories that simply don't exist. This causes no errors, but be careful not to include patterns that are so broad that they match unintended files.

Following symbolic links

When scanning directories recursively, the default behavior is to follow symbolic links. Often this is very useful, but it can cause problems if a symbolic link points to a large or slow filesystem. To control this behavior, simply do this:

```
followSymlink = false
```

It's probably a good idea to put this on all of your monitor stanzas until you know you need to follow a symbolic link.

Setting the value of the host from the source

The default behavior of using the hostname from the machine forwarding the logs is almost always what you want. If, however, you are reading logs for a number of hosts, you can extract the hostname from the source using `host_regex` or `host_segment`. For instance, say we have the path:

```
/nfs/logs/webserver1/access.log
```

To set host to `webserver1`, you could use:

```
[monitor:///nfs/logs/*/access.log*]  
sourcetype=access  
host_segment=3
```

You could also use:

```
[monitor:///nfs/logs/*/access.log*]  
sourcetype=access  
host_regex=/(.*)/access\.\log
```

The `host_regex` variable could also be used to extract the value of the host from the filename. It is also possible to reset the host using a transform, with the caveat that this will occur after parsing, which means any settings in `props.conf` that rely on matching the host will already have been applied.

Ignoring old data at installation

It is often the case that, when Splunk is installed, months or years of logs are sitting in a directory where logs are currently being written. Logs that are appended to infrequently may also have months or years of events that are no longer interesting and would be wasteful to index.

The best solution is to set up archive scripts to compress any logs older than a few days, but in a large environment, this may be difficult to do. Splunk has two settings that help ignore older data, but be forewarned: once these files have been ignored, there is no simple way to change your mind later. If, instead, you compress older logs and blacklist the compressed files as explained in the *Using blacklist and whitelist* section, you can simply decompress, at a later stage, any files you would like to index. Let's look at a sample stanza:

```
[monitor:///opt/B/logs/access.log*]
sourcetype = access
ignoreOlderThan = 14d
```

In this case, `ignoreOlderThan` says to ignore, forever, all events in any files, the modification date of which is older than 14 days. If the file is updated in the future, any new events will be indexed.

The `followTail` attribute lets us ignore all events written until now, instead starting at the end of each file. Let's look at an example:

```
[monitor:///opt/B/logs/access.log*]
sourcetype = access
followTail = 1
```

Splunk will note the length of files matching the pattern, but `TailfollowTail` instructs Splunk to ignore everything currently in these

files. Any new events written to the files will be indexed. Remember that there is no easy way to alter this if you change your mind later.

It is not currently possible to say *ignore all events older than x*, but since most logs roll on a daily basis, this is not commonly a problem.

When to use crcSalt

To keep track of what files have been seen before, Splunk stores a checksum of the first 256 bytes of each file it sees. This is usually plenty as most files start with a log message, which is almost guaranteed to be unique. This breaks down when the first 256 bytes are not unique on the same server.

I have seen two cases where this happens, as follows:

1. The first case is when logs start with a common header containing information about the product verion, for instance:

```
2. ======  
 =  
3. == Great product version 1.2 brought to you by Great company ==  
 == Server kernel version 3.2.1 ==
```

4. The second case is when a server writes many thousands of files with low time resolution, for instance:

```
5. 12:13:12 Session created  
    12:13:12 Starting session
```

To deal with these cases, we can add the path to the log to the checksum, or salt our crc. This is accomplished as shown here:

```
[monitor:///opt/B/logs/application.log*]  
sourcetype = access  
crcSalt = <SOURCE>
```

It says to include the full path to this log in the checksum.

This method will only work if your logs have a unique name. The easiest way to accomplish this is to include the current date in the name of the

log when it is created. You may need to change the pattern for your log names so that the date is always included and the log is not renamed.

Do not use **crcSalt** if your logs change names!

If you enable crcSalt in an input where it was not already enabled, you will re-index all the data! You need to ensure that the old logs are moved aside or uncompressed and blacklisted before enabling this setting in an existing configuration.

Destructively indexing files

If you receive logfiles in batches, you can use the batch input to consume `logs` and then delete them. This should only be used against a copy of the logs.

See the following example:

```
[batch:///var/batch/logs/*/access.log*]
sourcetype=access
host_segment=4
move_policy = sinkhole
```

This stanza would index the files in the given directory and then delete the files. Make sure this is what you want to do!

NETWORK INPUTS

In addition to reading files, Splunk can listen to network ports. The stanzas take the following form:

```
[protocol://<remote host>:<local port>]
```

The remote host portion is rarely used, but the idea is that you can specify different input configurations for specific hosts. The usual stanzas look as follows:

- `[tcp://1234]`: This specifies that we will listen to port 1234 for TCP connections. Anything can connect to this port and send data in.

- `[tcp-ssl://importanthost:1234]`: This listens on TCP using SSL, and we can apply this stanza to the `importanthost` host. Splunk will generate self-signed certificates the first time it is launched.
- `[udp://514]`: This is generally used to receive `syslog` events. While this does work, it is generally considered a best practice to use a dedicated syslog receiver, such as `rsyslog` or `syslogng`. See [Chapter 12, Advanced Deployments](#), for a discussion on this subject.
- `[splunktcp://9997]` or `[splunktcp-ssl://9997]`: In a distributed environment, your indexers will receive events on the specified port. It is a custom protocol used between Splunk instances. This stanza is created for you when you use the **Manager** page at **Manager | Forwarding and receiving | Receive data**.

For TCP and UDP inputs, the following attributes apply:

- `source`: If it is not specified, the source will default to `protocol:port`, for instance, `udp:514`.
- `sourcetype`: If it is not specified, `sourcetype` will also default to `protocol:port`, but this is generally not what you want. It is best to specify a source type and create a corresponding stanza in `props.conf`.
- `connection_host`: With network inputs, what value to capture for `host` is somewhat tricky. Your options essentially are:
 - `connection_host = dns` uses reverse DNS to determine the hostname from the incoming connection. When reverse DNS is configured properly, this is usually your best bet. This is the default setting.
 - `connection_host = ip` sets the host field to the IP address of the remote machine. This is your best choice when reverse DNS is unreliable.
 - `connection_host = none` uses the hostname of the Splunk instance receiving the data. This option can make sense when all traffic is going to an interim host.
 - `host = foo` sets the hostname statically.
 - It is also common to reset the value of the host using a transform, for instance, with syslog events. This happens after parsing, though, so it is too late to change things such as time zone based on the host.

- `queueSize`: This value specifies how much memory Splunk is allowed to set aside for an input queue. A common use for a queue is to capture spiky data until the indexers can catch up.
- `persistentQueueSize`: This value specifies a persistent queue that can be used to capture data to the disk if the in-memory queue fills up. If you find yourself building a particularly complicated setup around network ports, I would encourage you to talk to Splunk support as there may be a better way to accomplish your goals.

NATIVE WINDOWS INPUTS

One nice thing about Windows is that system logs and many application logs go to the same place.

Unfortunately, that place is not a file, so native hooks are required to access these events. Splunk makes those inputs available using stanzas of the `[WinEventLog:LogName]`. form. For example, to index the `Security` log, the stanza simply looks like this:

```
[WinEventLog:Security]
```

There are a number of supported attributes, but the defaults are reasonable. The only attribute I have personally used is `current_only`, which is the equivalent of `followTail` for monitor stanzas. For instance, this stanza says to monitor the `Application` log, but also to start reading from now:

```
[WinEventLog:Application]
current_only = 1
```

This is useful when there are many historical events on the server.

The other input available is **Windows Management Instrumentation(WMI)**. With WMI, you can accomplish the following:

- Monitor native performance metrics as you would find in Windows Performance Monitor
- Monitor the Windows Event Log API
- Run custom queries against the database behind WMI
- Query remote machines

Even though it is theoretically possible to monitor many Windows servers using WMI and a few Splunk forwarders, this is not advised. The configuration is complicated, does not scale well, introduces complicated security implications, and is not thoroughly tested. Also, reading Windows Event Logs via WMI produces different output than the native input, and most apps that expect Windows events will not function as expected.

The simplest way to generate the `inputs.conf` and `wmi.conf` configurations needed for Windows Event Logs and WMI is to install Splunk for Windows on a Windows host and then configure the desired inputs through the web interface. See the official Splunk documentation for more examples.

SCRIPTS AS INPUTS

Splunk will periodically execute processes and capture the output. For example, here is input from the `ImplementingSplunkDataGenerator` app:

```
[script://./bin/implSplunkGen.py 2]
interval=60
sourcetype=impl_splunk_gen_sourcetype2
source=impl_splunk_gen_src2
host=host2
index=implSplunk
```

Things to notice in this example are as follows:

- The present working directory is the root of the app that contains `inputs.conf`.
- Files that end with `.py` will be executed using the Python interpreter included with Splunk. This means the Splunk Python modules are available.

To use a different Python module, specify the path to Python in the stanza.

- Any arguments specified in the stanza will be handed to the script as if it was executed at the command line.

- The interval specifies how often, in seconds, this script should be run:
 - If the script is still running, it will not be launched again.
 - Long-running scripts are fine. Since only one copy of a script will run at a time, the interval will instead indicate how often to check whether the script is still running.
 - This value can also be specified in the `cron` format.

Any programming language can be used as long as it can be executed at the command line. Splunk simply captures the standard output from whatever is executed.

Included with Splunk for Windows are scripts to query WMI. One sample stanza looks as follows:

```
[script://$SPLUNK_HOME\bin\scripts\splunk-wmi.path]
```

The things to note are as follows:

- Windows paths require backslashes instead of slashes
- `$SPLUNK_HOME` will expand properly

transforms.conf

The `transforms.conf` configuration is where we specify transformations and lookups that can then be applied to any event. These transforms and lookups are referenced by name in `props.conf`.

For our examples in the later subsections, we will use this event:

```
2012-09-24T00:21:35.925+0000 DEBUG [MBX] Password reset
called.

[old=1234, new=secret, req_time=5346]
```

We will use it with these metadata values:

```
sourcetype=myapp
source=/logs/myapp.session_foo-jA5MDkyMjEwMTIK.log
host=vlbmba.local
```

CREATING INDEXED FIELDS

One common task accomplished with `transforms.conf` is the creation of new indexed fields. Indexed fields are different than extracted fields in that they must be created at index time and can be searched for whether the value is in the raw text of the event or not. It is usually preferable to create extracted fields instead of indexed fields. See [Chapter 3, Tables, Charts, and Fields](#), for a deeper discussion about when indexed fields are beneficial.

Indexed fields are only applied to events that are indexed after the definition is created. There is no way to backfill a field without re-indexing.

Creating a loglevel field

The format of a typical stanza in `transforms.conf` looks as follows:

```
[myapp_loglevel]
REGEX = \s([A-Z]+\s
FORMAT = loglevel:::$1
WRITE_META = True
```

This will add to our events the field `loglevel=DEBUG`. This is a good idea if the values of `loglevel` are common words outside of this location, for instance `ERROR`.

Walking through this stanza, we have the following:

- `[myapp_loglevel]`: The stanza can be any unique value, but it is in your best interest to make the name meaningful. This is the name referenced in `props.conf`.
- `REGEX = \s([A-Z]+\s`: This is the pattern to test against each event that is handed to us. If this pattern does not match, this transform will not be applied.
- `FORMAT = loglevel:::$1`: Create the `loglevel`. Under the hood, all indexed fields are stored using a `::` delimiter, so we have to follow that form.

- `WRITE_META = True`: Without this attribute, the transform won't actually create an indexed field and store it with the event.

Creating a session field from the source

Using our event, let's create another field, `session`, which appears only to be in the value of the source:

```
[myapp_session]
SOURCE_KEY = MetaData:Source
REGEX = session_(.*?)\.\log
FORMAT = session::$1
WRITE_META = True
```

Note the `SOURCE_KEY`.attribute. The value of this field can be any existing metadata field or another indexed field that has already been created. See the *Attributes with class* subsection within the `props.conf` section for a discussion about the transform execution order. We will discuss these fields in the `Modifying metadata fields` subsection.

Creating a tag field

It is also possible to create fields simply to tag events that would be difficult to search for otherwise. For example, if we wanted to find all events that were slow, we could search for:

```
sourcetype=myapp req_time>999
```

Without an indexed field, this query would require parsing every event that matches `sourcetype=myapp` over the time that we are interested in. The query would then discard all events whose `req_time` value was 999 or less.

If we know ahead of time that a value of `req_time>999` is bad, and we can come up with a regular expression to specify what "bad" is, we can tag these events for quicker retrieval. Say we have this `transforms.conf` stanza:

```
[myapp_slow]
```

```
REGEX = req_time=\d{4,}
FORMAT = slow_request::1
WRITE_META = True
```

This `REGEX` will match any event containing `req_time=` followed by four or more digits.

After adding `slow_request` to `fields.conf` (see the `fields.conf` section), we can search for `slow_request=1` and find all slow events very efficiently. This will not apply to events that were indexed before this transform existed. If the events that are slow are uncommon, this query will be much faster.

Creating host categorization fields

It is common to have parts of a hostname mean something in particular. If this pattern is well known and predictable, it may be worthwhile to pull the value out into fields. Working from our fictitious host `value.vlbmba.local` (which happens to be my laptop), we might want to create fields for the owner and the host type. Our stanza might look similar to this:

```
[host_parts]
SOURCE_KEY = MetaData:Host
REGEX = (...) (...)\.
FORMAT = host_owner::$1 host_type::$2
WRITE_META = True
```

With our new fields, we can now easily categorize errors by whatever information is encoded into the hostname. Another approach would be to use a lookup, which has the advantage of being retroactive. This approach has the advantage of faster searches for the specific fields.

MODIFYING METADATA FIELDS

It is sometimes convenient to override the main metadata fields. We will look at one possible reason for overriding each base metadata value.

Remember that transforms are applied after parsing, so changing metadata fields via transforms cannot be used to affect which `props.conf` stanzas are applied for date parsing or line breaking.

For instance, with `syslog` events that contain the hostname, you cannot change the time zone because the date has already been parsed before the transforms are applied. The keys provided by Splunk include:

- `_raw` (this is the default value for `SOURCE_KEY`)
- `MetaData:Source`
- `MetaData:Sourcetype`
- `MetaData:Host`
- `_MetaData:Index`

Overriding the host

If your hostnames are appearing differently from different sources: for instance, `syslog` versus Splunk Forwarders, you can use a transform to normalize these values. Given our hostname `v1bmba.local`, we may want to only keep the portion to the left of the first period. The stanza would look as follows:

```
[normalize_host]
SOURCE_KEY = MetaData:Host
DEST_KEY = MetaData:Host
REGEX = (.*?)\.
FORMAT = host::$1
```

This will replace our hostname with `v1bmba`. Note these two things:

- `WRITE_META` is not included because we are not adding to the metadata of this event; we are instead overwriting the value of a core metadata field
- `host::` must be included at the beginning of the format

Overriding the source

Some applications will write a log for each session, conversation, or transaction. One problem this introduces is an explosion of source

values. The values of the source will end up in `$SPLUNK_HOME/var/lib/splunk/*/db/Sources.data`—one line per unique value of the source. This file will eventually grow to a huge size, and Splunk will waste a lot of time updating it, causing unexplained pauses. A new setting in `indexes.conf`, called `disableGlobalMetadata`, can also eliminate this problem.

To flatten this value, we could use a stanza such as this:

```
[myapp_flatten_source]
SOURCE_KEY = MetaData:Source
DEST_KEY = MetaData:Source
REGEX = (.session_).*.log
FORMAT = source::$1x.log
```

This would set the value of source to `/logs/myapp.session_x.log`, which would eliminate our growing source problem. If the value of session is useful, the transform in the *Creating a session field from source* section could be run before this transform to capture the value. Likewise, a transform could capture the entire value of the source and place it into a different metadata field.

A huge number of logfiles on a filesystem introduces a few problems, including running out of nodes and the memory used by the Splunk process of tracking all of the files. As a general rule, a cleanup process should be designed to archive older logs.

Overriding sourcetype

It is not uncommon to change the `sourcetype` field of an event based on the contents of the event, particularly from syslog. In our fictitious example, we want a different source type for events that contain `[MBX]` after the log level so that we can apply different extracts to these events. The following examples will do this work:

```
[mbx_sourcetype]
DEST_KEY = MetaData:Sourcetype
REGEX = \d+\s[A-Z]+\s\([MBX]\)
```

```
FORMAT = sourcetype::mbx
```

Use this functionality carefully as it easy to go conceptually wrong, and this is difficult to fix later.

Routing events to a different index

At times, you may want to send events to a different index, either because they need to live longer than other events or because they contain sensitive information that should not be seen by all users. This can be applied to any type of event from any source, whether it be a file, network, or script.

All that we have to do is match the event and reset the index.

```
[contains_password_1]
DEST_KEY = _MetaData:Index
REGEX = Password reset called
FORMAT = sensitive
```

The things to note are as follows:

- In this scenario, you will probably make multiple transforms, so make sure to make the name unique
- DEST_KEY starts with an underscore
- FORMAT does not start with index::
- The index sensitive must exist on the machine indexing the data, or else the event will be lost

LOOKUP DEFINITIONS

A simple lookup simply needs to specify a filename in transforms.conf, as shown here:

```
[testlookup]
filename = test.csv
```

Assuming that `test.csv` contains the `user` and `group` columns and our events contain the field `user`, we can reference this lookup using the `lookup` command in search, as follows:

```
* | lookup testlookup user
```

Otherwise, we can wire this lookup to run automatically in `props.conf`, as follows:

```
[mysourcetype]
LOOKUP-testlookup = testlookup user
```

That's all you need to get started, and this probably covers most cases. See the *Using lookups to enrich data* section in [Chapter 7, Extending Search](#), for instructions on creating lookups.

Wildcard lookups

In [Chapter 10, Summary Indexes and CSV Files](#), we edited `transforms.conf` but did not explain what was happening. Let's take another look. Our transform stanza looks as follows:

```
[flatten_summary_lookup]
filename = flatten_summary_lookup.csv
match_type = WILDCARD(url)
max_matches = 1
```

Walking through what we added, we have the following terms and their descriptions:

- `match_type = WILDCARD(url)`: This says that the value of the `url` field in the lookup file may contain wildcards. In our example, the URL might look like `/ contact/*` in our CSV file.
- `max_matches = 1`: By default, up to 10 entries that match in the lookup file will be added to an event, with the values in each field being added to a multivalue field. In this case, we only want the first match to be applied.

CIDR wildcard lookups

CIDR wildcards look very similar to text-based wildcards but use Classless Inter-Domain Routing (CIDR) rules to match lookup rows against an IP address.

Let's try an example.

Say we have this lookup file:

```
ip_range, network, datacenter
10.1.0.0/16, qa, east
10.2.0.0/16, prod, east
10.128.0.0/16, qa, west
10.129.0.0/16, prod, west
```

It has this corresponding definition in `transforms.conf`:

```
[ip_address_lookup]
filename = ip_address_lookup.csv
match_type = CIDR(ip_range)
max_matches = 1
```

And, there are a few events such as these:

```
src_ip=10.2.1.3 user=mary
src_ip=10.128.88.33 user=bob
src_ip=10.1.35.248 user=bob
```

We could use `lookup` to enrich these events as follows:

```
src_ip="*"
| lookup ip_address_lookup ip_range as src_ip
| table src_ip user datacenter network
```

This would match the appropriate IP address and give us a table such as this one:

	src_ip	user	datacenter	network
1	10.2.1.3	mary	east	prod
2	10.128.88.33	bob	west	qa
3	10.1.35.248	bob	east	qa

The query also shows that you could use the same lookup for different fields using the `as` keyword in the `lookup` call.

Using time in lookups

A temporal lookup is used to enrich events based on when the event happened. To accomplish this, we specify the beginning of a time range in the lookup source and then specify a format for this time in our lookup configuration. Using this mechanism, `lookup` values can change over time, even retroactively.

Here is a very simple example to attach a version field based on time. Say we have the following CSV file:

```
sourcetype,version,time
impl_splunk_gen,1.0,2012-09-19 02:56:30 UTC
impl_splunk_gen,1.1,2012-09-22 12:01:45 UTC
impl_splunk_gen,1.2,2012-09-23 18:12:12 UTC
```

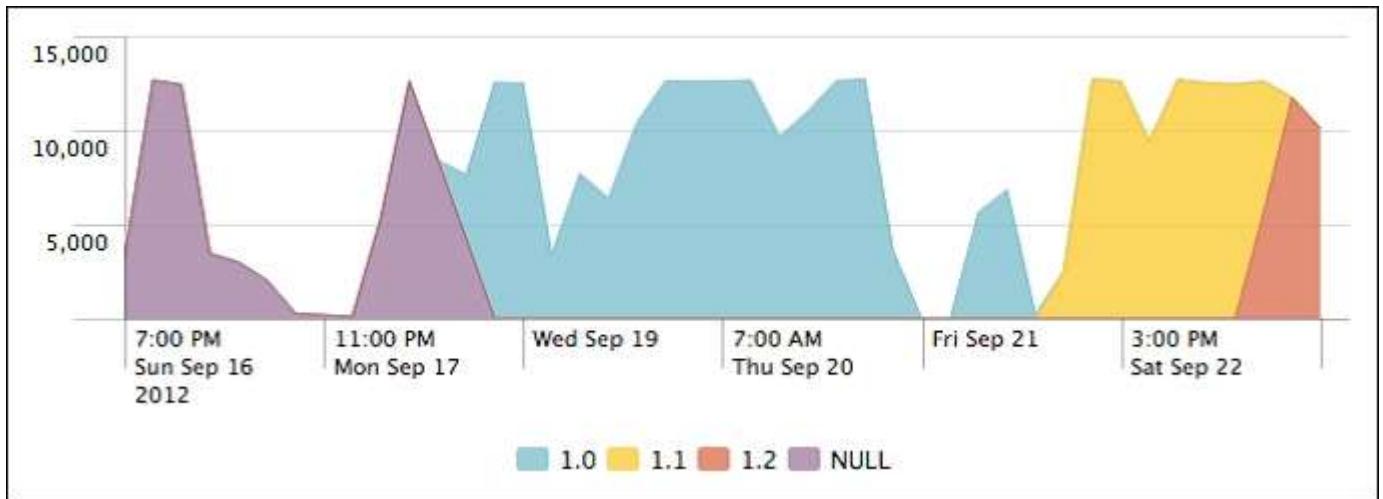
We then use the lookup configuration in `transforms.conf` to specify which field in our lookup will be tested against the time in each event, and what the format of the time field will be:

```
[versions]
filename = versions.csv
time_field = time
time_format = %Y-%m-%d %H:%M:%S %Z
```

With this in place, we can now use our lookup in search, as shown here:

```
sourcetype=impl_splunk_gen error
| lookup versions sourcetype
| timechart count by version
```

This would give us a chart of errors (by version) over time, as shown here:



Other use cases include tracking deployments across environments and tracking activity from disabled accounts.

USING REPORT

Attributes of the format `REPORT-foo` in `props.conf` call stanzas in `transforms.conf` at search time, which means that they cannot affect metadata fields. `EXTRACT` definitions are more convenient to write as they live entirely in a single attribute in `props.conf`, but there are a couple of things that can only be done using a `REPORT` attribute paired with a transform defined in `transforms.conf`.

Creating multivalue fields

Assuming some value might occur multiple times in a given event, an `EXTRACT` definition can only match the first occurrence. For example, say we have the event:

```
2012-08-25T20:18:09 action=send a@b.com c@d.com e@f.com
```

We could pull the first e-mail address using the following extraction:

```
EXTRACT-email = (?i) (?P<email>[a-zA-Z0-9._]+@[a-zA-Z0-9._]+)
```

This would set the field `email` to `a@b.com`. Using a `REPORT` attribute and the `transform` stanza, we can capture all of the e-mail addresses using the `MV_ADD` attribute. The `props` stanza would look as follows:

```
REPORT-mvemail = mvemail
```

The `transforms.conf` stanza would then look as follows:

```
[mvemail]
REGEX = (?i) ([a-zA-Z0-9._]+@[a-zA-Z0-9._]+)
FORMAT = email:::$1
MV_ADD = true
```

The `MV_ADD` attribute also has the effect that, if some other configuration has already created the `email` field, all values that match will be added to the event.

Creating dynamic fields

Sometimes, it can be useful to dynamically create fields from an event. For instance, say we have an event, such as:

```
2012-08-25T20:18:09 action=send from_335353("a@b.com")
to_223523("c@d.com") cc_39393("e@f.com") cc_39394("g@h.com")
```

It would be nice to pull from, to, and cc as fields, but we may not know all of the possible field names. This stanza in `transforms.conf` would create the fields we want, dynamically:

```
[dynamic_address_fields]
REGEX=\s(\S+)\_\s+\("(.*)"\)
FORMAT = $1:::$2
MV_ADD=true
```

While we're at it, let's put the numeric value after the field name into a value:

```
[dynamic_address_ids]
REGEX=\s(\S+)\_(\S+)\(".
FORMAT = $1:::$2
```

```
MV_ADD=true
```

This gives us multivalue fields such as the ones in the following screenshot:

action	cc	from	to
send	a@f.com g@h.com 39393 39394	a@b.com	c@d.com 335353 223523

One thing that we cannot do is add extra text to the `FORMAT` attribute. For instance, in the second case, it would be nice to use a `FORMAT` attribute such as this one:

```
FORMAT = $1_id:::$2
```

Unfortunately, this will not function as we hope and will instead create the field `id`.

CHAINING TRANSFORMS

As covered before in the *Attributes with class* section, transforms are executed in a particular order. In most cases, this order does not matter, but there are occasions when you might want to chain transforms together, with one transform relying on a field created by a previous transform.

A good example is the source flattening that we used previously in the *Overriding source* section. If this transform happened before our transform in the *Creating a session field from source* section, our session field would always have the value `x`.

Let's reuse two transforms from the previous sections and then create one more transform. We will chain them to pull the first part of session into yet another field. Say we have these transforms:

```
[myapp_session]
SOURCE_KEY = MetaData:Source
REGEX = session_(.*?)\.\log
FORMAT = session::$1
```

```

WRITE_META = True
[myapp_flatten_source]
SOURCE_KEY = MetaData:Source
DEST_KEY = MetaData:Source
REGEX = (*.session_).*\.log
FORMAT = source::$1x.log
[session_type]
SOURCE_KEY = session
REGEX = (.*?)-
FORMAT = session_type::$1
WRITE_META = True

```

To ensure that these transforms run in order, the simplest thing would be to place them in a single `TRANSFORMS` attribute in `props.conf`, as shown here:

```

[source:*session_*.log]
TRANSFORMS-s =
myapp_session,myapp_flatten_source,session_type

```

We can use the source from our sample event specified inside `transforms.conf` as follows:

```
source=/logs/myapp.session_foo-jA5MDkyMjEwMTIK.log
```

Walking though the transforms, we have the following terms and their descriptions:

- `myapp_session`: Reading from the metadata field, `source`, this creates the indexed field `session` with the `foo-jA5MDkyMjEwMTIK` value
- `myapp_flatten_source`: This resets the metadata field, `source`, to `/logs/myapp.session_x.log`
- `session_type`: Reading from our newly indexed field, `session`, this creates the `session_type` field with the value `foo`

This same ordering logic can be applied at search time using the `EXTRACT` and `REPORT` Stanzas. This particular case needs to be calculated as indexed fields if we want to search for these values since the values are part of a metadata field.

DROPPING EVENTS

Some events are simply not worth indexing. The hard part is figuring out which ones these are and making very sure you're not wrong. Dropping too many events can make you blind to real problems at critical times and can introduce more problems than tuning Splunk to deal with the greater volume of data in the first place.

With that warning stated, if you know what events you do not need, the procedure for dropping events is pretty simple. Say we have an event such as this one:

```
2012-02-02 12:24:23 UTC TRACE Database call 1 of 1,000.  
[...]
```

I know absolutely that, in this case and for this particular source type, I do not want to index `TRACE` level events.

In `props.conf`, I will create a stanza for my source type, as shown here:

```
[mysourcetype]  
TRANSFORMS-droptrace=droptrace
```

Then, I will create the following transform in `transforms.conf`:

```
[droptrace]  
REGEX=^\d{4}-\d{2}-\d{2}\s+\d{1,2}:\d{2}:\d{1,2}\s+[A-Z]+\sTRACE  
DEST_KEY=queue  
FORMAT=nullQueue
```

Splunk compares `nullQueue` to `nulldevice`, which (according to the product documentation) tells Splunk not to forward or index the filtered data.

This `REGEX` attribute is purposely as strict as I can make it. It is vital that I do not accidentally drop other events, and it is better for this brittle pattern to start failing and to let through `TRACE` events rather than for it to do the opposite.

fields.conf

We need to add to `fields.conf` any indexed fields we create, or else they will not be searched efficiently, or may even not function at all. For our examples in the `transforms.conf` section, `fields.conf` would look as follows:

```
[session_type]
INDEXED = true
[session]
INDEXED = true
[host_owner]
INDEXED = true
[host_type]
INDEXED = true
[slow_request]
INDEXED = true
[loglevel]
INDEXED = true
```

These stanzas instruct Splunk not to look in the body of the events for the value being queried. Take, for instance, the following search:

```
host_owner=vlb
```

Without this entry, the actual query would essentially be:

```
vlb | search host_owner=vlb
```

With the expectation that the value `vlb` is in the body of the event, this query simply won't work. Adding the entry to `fields.conf` fixes this. In the case of loglevel, since the value is in the body, the query will work, but it will not take advantage of the indexed field, instead only using it to filter events after finding all events that contain the bare word.

outputs.conf

This configuration controls how Splunk will forward events. In the vast majority of cases, this configuration exists on Splunk Forwarders, which send their events to Splunk indexers. An example would look similar to this:

```
[tcpout]
defaultGroup = nyc
[tcpout:nyc]
autoLB = true
server = 1.2.3.4:9997,1.2.3.6:9997
```

It is possible to use transforms to route events to different server groups, but it is not commonly used as it introduces a lot of complexity that is generally not needed.

indexes.conf

Put simply, `indexes.conf` determines where data is stored on the disk, how much is kept, and for how long. An index is simply a named directory with a specific structure. Inside this directory structure, there are a few metadata files and subdirectories; the subdirectories are called buckets and actually contain the indexed data.

A simple stanza looks as follows:

```
[implSplunk]
homePath = $SPLUNK_DB/implSplunk/db
coldPath = $SPLUNK_DB/implSplunk/colddb
thawedPath = $SPLUNK_DB/implSplunk/thaweddb
```

Let's walk through these attributes:

- `homePath`: This is the location for recent data.
- `coldPath`: This is the location for older data.

- `thawedPath`: This is a directory where buckets can be restored. It is an unmanaged location. This attribute must be defined, but I for one, have never actually used it.

An aside about the terminology of buckets is probably in order. It is as follows:

- `hot`: This is a bucket that is currently open for writing. It lives in `homePath`.
- `warm`: This is a bucket that was created recently but is no longer open for writing. It also lives in `homePath`.
- `cold`: This is an older bucket that has been moved to `coldPath`. It is moved when `maxWarmDBCount` has been exceeded.
- `frozen`: For most installations, this simply means deleted. For customers who want to archive buckets, `coldToFrozenScript` or `coldToFrozenDir` can be specified to save buckets.
- `thawed`: A thawed bucket is a frozen bucket that has been brought back. It is special in that it is not managed, and it is not included in all time queries. When using `coldToFrozenDir`, only the raw data is typically kept, so Splunk rebuild will need to be used to make the bucket searchable again.

How long data stays in an index is controlled by these attributes:

- `frozenTimePeriodInSecs`: This setting dictates the oldest data to keep in an index. A bucket will be removed when its newest event is older than this value. The default value is approximately 6 years.
- `maxTotalDataSizeMB`: This setting dictates how large an index can be. The total space used across all hot, warm, and cold buckets will not exceed this value. The oldest bucket is always frozen first. The default value is 500 gigabytes. It is generally a good idea to set both of these attributes. `frozenTimePeriodInSecs` should match what users expect. `maxTotalDataSizeMB` should protect your system from running out of disk space.

Less commonly used attributes include:

- `coldToFrozenDir`: If specified, buckets will be moved to this directory instead of being deleted. This directory is not managed by Splunk, so it is up to the administrator to make sure that the disk does not fill up.
- `maxHotBuckets`: A bucket represents a slice of time and will ideally span as small a slice of time as is practical. I would never set this value to less than 3, but ideally, it should be set to 10.
- `maxDataSize`: This is the maximum size for an individual bucket. The default value is set by the processor type and is generally acceptable. The larger a bucket, the fewer the buckets to be opened to complete a search, but the more the disk space needed before a bucket can be frozen. The default is auto, which will never top 750 MB. The setting `auto_high_volume`, which equals 1 GB on 32-bit systems and 10 GB on 64-bit systems, should be used for indexes that receive more than 10 GB a day.

authorize.conf

This configuration stores definitions of capabilities and roles. These settings affect search functions and the web interface. They are generally managed through the interface at **Manager | Access controls**, but a quick look at the configuration itself may be useful.

A role stanza looks as follows:

```
[role_power]
importRoles = user
schedule_search = enabled
rtsearch = enabled
srchIndexesAllowed = *
srchIndexesDefault = main
srchDiskQuota = 500
srchJobsQuota = 10
rtSrchJobsQuota = 20
```

Let's walk through these settings:

- `importRoles`: This is a list of roles to import capabilities from. The set of capabilities will be the merging of capabilities from imported roles and added capabilities.
- `schedule_search` and `rtsearch`: These are two capabilities enabled for the role power that were not necessarily enabled for the imported roles.
- `srchIndexesAllowed`: This determines which indexes this role is allowed to search. In this case, all are allowed.
- `srchIndexesDefault`: This determines the indexes to search by default. This setting also affects the data shown in **Search | Summary**. If you have installed the `ImplementingSplunkDataGenerator` app, you will see the `impl_splunk_*` source types on this page even though this data is actually stored in the `implsplunk` index.
- `srchDiskQuota`: Whenever a search is run, the results are stored on the disk until they expire. The expiration can be set explicitly when creating a saved search, but the expiration is automatically set for interactive searches. Users can delete old results from the **Jobs** view.
- `srchJobsQuota`: Each user is limited to a certain number of concurrently running searches. The default is three. Users with the power role are allowed 10, while those with the admin role are allowed 50.
- `rtSrchJobsQuota`: Similarly, this is the maximum number of concurrently running real-time searches. The default is six.

savedsearches.conf

This configuration contains saved searches and is rarely modified by hand.

times.conf

This configuration holds definitions for time ranges that appear in the time picker.

commands.conf

This configuration specifies commands provided by an app.

web.conf

The main settings changed in this file are the port for the web server, the SSL certificates, and whether to start the web server at all.

User interface resources

Most Splunk apps consist mainly of resources for the web application. The app layout for these resources is completely different from all other configurations.

Views and navigation

Like `.conf` files, view and navigation documents take precedence in the following order:

- `$SPLUNK_HOME/etc/users/$username/$appname/local`: When a new dashboard is created, it lands here. It will remain here until the permissions are changed to **App** or **Global**.
- `$SPLUNK_HOME/etc/apps/$appname/local`: Once a document is shared, it will be moved to this directory.
- `$SPLUNK_HOME/etc/apps/$appname/default`: Documents can only be placed here manually. You should do this if you are going to share an app. Unlike `.conf` files, these documents do not merge.

Within each of these directories, views and navigation end up under the directories `data/ui/views` and `data/ui/nav`, respectively. So, given a view `foo`, for the user `bob`, in the app `app1`, the initial location for the document will be as follows:

```
$SPLUNK_HOME/etc/users/bob/app1/local/data/ui/views/foo.xml
```

Once the document is shared, it will be moved to the following location:

```
$SPLUNK_HOME/etc/apps/app1/local/data/ui/views/foo.xml
```

Navigation follows the same structure, but the only navigation document that is ever used is called `default.xml`, for instance:

```
$SPLUNK_HOME/etc/apps/app1/local/data/ui/nav/default.xml
```

You can edit these files directly on the disk instead of through the web interface, but Splunk will probably not realize the changes without a restart—unless you use a little trick. To reload changes to views or navigation made directly on the disk, load the URL `http://mysplunkserver:8000/debug/refresh`, replacing `mysplunkserver` appropriately. If all else fails, restart Splunk.

Appserver resources

Outside of views and navigation, there are a number of resources that the web application will use. For instance, applications and dashboards can reference CSS and images, as we did in [Chapter 8, Working with Apps](#). These resources are stored under `$SPLUNK_HOME/etc/apps/$appname/appserver/`. There are a few directories that appear under this directory, as follows:

- `static`: Any static files that you would like to use in your application are stored here. There are a few magic documents that Splunk itself will use, for instance, `appIcon.png`, `screenshot.png`, `application.css`, and `application.js`. Other files can be referenced using includes or templates. See the *Using ServerSideInclude in a complex dashboard* section in [Chapter 8, Working with Apps](#), for an example of referencing includes and static images.
- `event_renderers`: Event renderers allow you to run special display code for specific event types. We will write an event renderer in [Chapter 13, Extending Splunk](#).
- `templates`: It is possible to create special templates using the *makotemplate* language. It is not commonly done.
- `modules`: This is where new modules that are provided by apps are stored. Examples of this include the Google Maps and Sideview Utils modules. See <http://dev.splunk.com> for more information

about building your own modules or use existing modules as an example.

Metadata

Object permissions are stored in files located at `$SPLUNK_HOME/etc/apps/$appname/metadata/`. The two possible files are `default.meta` and `local.meta`.

These files have certain properties:

- They are only relevant to the resources in the app where they are contained
- They do merge, with entries in `local.meta` taking precedence
- They are generally controlled by the admin interface
- They can contain rules that affect all configurations of a particular type, but this entry must be made manually

In the absence of these files, resources are limited to the current app.

Let's look at `default.meta` for `is_app_one`, as created by Splunk:

```
# Application-level permissions
[]
access = read : [ * ], write : [ admin, power ]
### EVENT TYPES
[eventtypes]
export = system
### PROPS
[props]
export = system
### TRANSFORMS
[transforms]
export = system
### LOOKUPS
[lookups]
Chapter 10
```

```

[ 329 ]
export = system
### VIEWSTATES: even normal users should be able to create
shared
viewstates
[viewstates]
access = read : [ * ], write : [ * ]
export = system

```

Walking through this snippet, we have the following terms and their descriptions:

- The `[]` stanza states that all users should be able to read everything in this app but that only users with the admin or power roles should be able to write to this app.
- The `[eventtypes]`, `[props]`, `[transforms]`, and `[lookups]` states say that all configurations of each type in this app should be shared by all users in all apps, by default. `export=system` is equivalent to `Global` in the user interface.
- The `[viewstates]` stanza gives all users the right to share `viewstates` globally. A viewstate contains information about dashboard settings made through the web application, for instance, chart settings. Without this, chart settings applied to a dashboard or saved search would not be available.

Looking at `local.meta`, we see settings produced by the web application for the configurations we created through the web application.

```

[indexes/summary_impl_splunk]
access = read : [ * ], write : [ admin, power ]
[views/errors]
access = read : [ * ], write : [ admin, power ]
export = system
owner = admin
version = 4.3
modtime = 1339296668.151105000
[savedsearches/top%20user%20errors%20pie%20chart]

```

```

export = none
owner = admin
version = 4.3
modtime = 1338420710.720786000
[viewstates/flashtimeline%3Ah2v14xkb]
owner = nobody
version = 4.3
modtime = 1338420715.753642000
[props/impl_splunk_web/LOOKUP-web_section]
access = read : [ * ]
export = none
owner = admin
version = 4.3
modtime = 1346013505.279379000

```

Hopefully, you get the idea. The web application will make very specific entries for each object created. When distributing an application, it is generally easier to make blanket permissions in `metadata/default.meta` as appropriate for the resources in your application.

For an application that simply provides dashboards, no metadata at all will be needed as the default for all resources (apps) will be acceptable. If your application provides resources to be used by other applications, for instance, lookups or extracts, your `default.meta` file might look like this:

```

### PROPS
[props]
export = system
### TRANSFORMS
[transforms]
export = system
### LOOKUPS
[lookups]
export = system

```

This states that everything in your `props.conf` and `transforms.conf` files, and all lookup definitions, are merged into the logical configuration of every search.

Summary

In this session, we learned about some terms that need to be understood about big data, such as what the terms streaming data, data latency, and data sparseness mean. We also covered the types of data that can be brought into Splunk. Then we studied what an index is, made an index for our data, and put in data from our Destinations app. We talked about what fields and events are. And finally, we saw how to extract fields from events and name them so that they can be more useful to us. In the chapters to come, we'll learn more about these important features of Splunk.

BONUS – A POWER USER SEARCH PRIMER

Search Processing Language

In the previous chapters, you learned how to collect and index data to prepare it for searching, and how to do a simple search. In this chapter, we will cover more about how to use search and other commands to analyze our data. In a nutshell, we will cover the following topics:

- Anatomy of a search
- Search pipeline
- Time modifiers
- Filtering searches
- Search command-`stats`
- Search command-`top/rare`
- Search commands-`chart` and `timechart`
- Search command-`eval`
- Search command-`rex`

Anatomy of a search

Search Processing Language (SPL), a special-purpose processing language, was developed to enable fast searching on machine-generated data that has been indexed into Splunk. The language was originally set up to be based on Unix piping and **Standard Query Language (SQL)**. We'll explain piping later. SQL is the language most widely used to query databases. The Search Processing Language (SPL, as opposed to SQL) is a library of all search processing commands and their functions, arguments, and clauses. With a search command you can group different events, filter data based on a constraint, extract fields using regular expressions, perform statistical calculations, and other tasks.

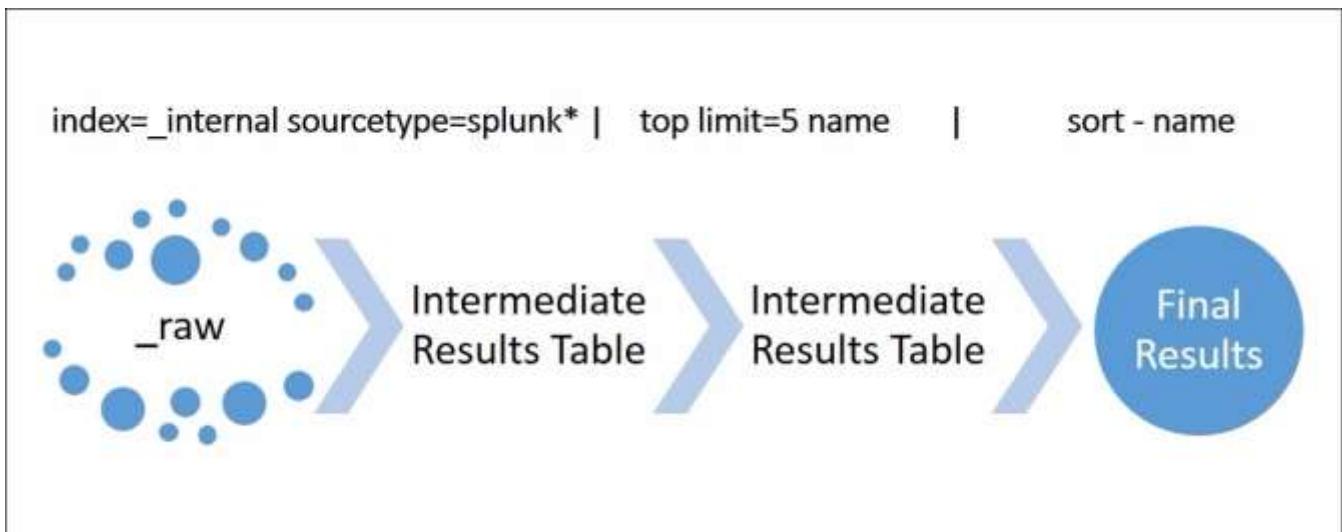
Let us dissect a search query so you can understand exactly how it works. This will also help you to understand what pipes are. As you will see, a pipe basically takes the data that has come from an earlier step and after it has been acted on, filtered, or extracted, sends it on to the next step in processing.

We'll use the Destinations app here to show you a simple example:

1. Go to the Splunk home page.
2. Click on your Destinations app.
3. In your Destinations app's **Search** page, type in the following:

```
SPL> index=_internal sourcetype=splunk* | top limit=5  
name  
| sort - name
```

The following diagram will allow you to visualize the data as it goes through one delimiting pipe (|) and another; in this case, from the internal index of Splunk, to limiting it to the top five names, to sorting by name, which then gets sent to the **Final Results** table. We will go through this step by step, as shown in the following screenshot:



Search pipeline

The arrows in the preceding visualization, each of which represents a pipe, mean that the resulting data from the previous command will be used against the next command in the series. To fully understand this command, let's go through the pipes one by one. Type the following commands in succession into the search window, but pause in every one of them and observe how the data changes.

The following command will display the raw events:

```
SPL> index=_internal sourcetype=splunk*
```

This command used the raw events in the previous results table, keyed them on the name field, tabulated it by the number of events that the particular field has, then limited the result to five:

```
SPL> index=_internal sourcetype=splunk* | top limit=5 name
```

Finally, the results table of the `| top` command is passed on to another command `| sort` for sorting transformation.

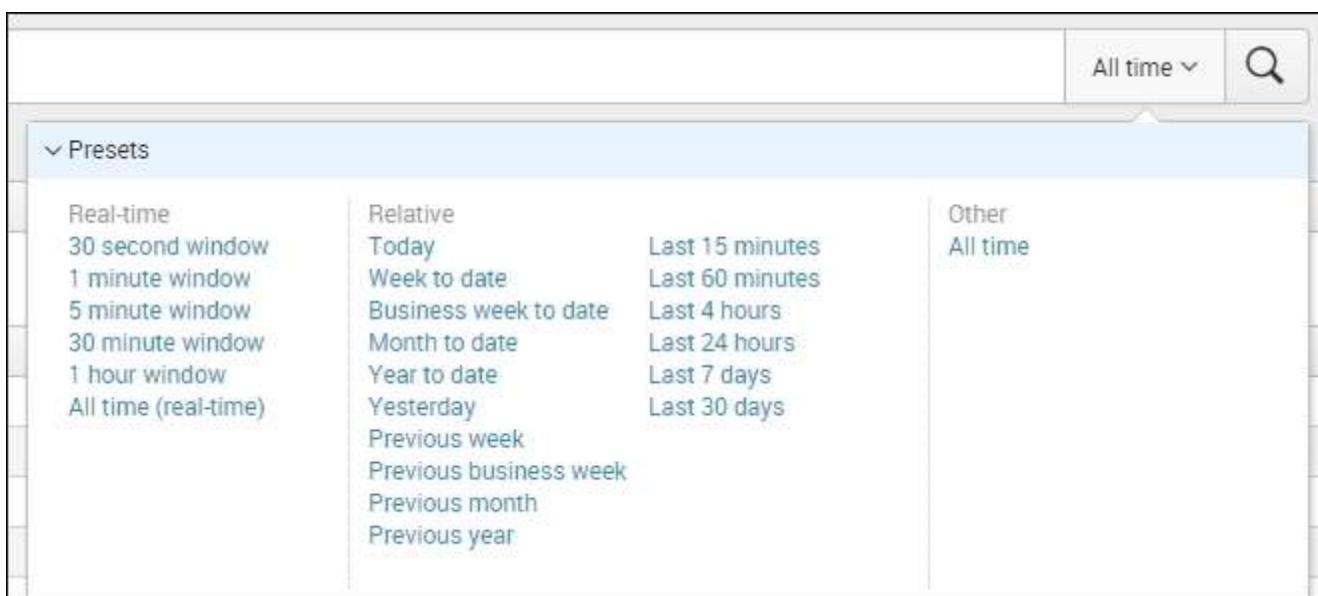
```
SPL> index=_internal sourcetype=splunk* | top limit=5 name |  
sort - name
```

This chaining of commands is called the **search pipeline**.

Time modifiers

Every time you execute a search, always be aware that you are running a query against a set of data that is bound by date and time. The time-range picker is on the right side of the search bar. Splunk comes with predetermined time modifiers, as seen in the following screenshot. You can also use the time-range picker to set up a custom date/time range or other advanced ranges

(<https://docs.splunk.com/Splexicon:Timerangepicker>):



There are two types of time modifier: real-time and relative. In the preceding screenshot, the predetermined real-time modifiers are in the leftmost column, and the relative time modifiers are in the middle column.

Real-time modifiers mean that Splunk will run an ongoing, real-time search based on the specified time. For example, a real-time search that is in a 5-minute window will continuously display data within the last five minutes. If new data comes in, it will push out the oldest event within the time frame.

NOTE

Real-time searches are resource-intensive. Use them sparingly. Other ways to efficiently show real-time data will be shown in later chapters.

Relative time modifiers are just that; they collect data based on relative time measures, and will find data within the specified timeframe.

What you do not see when you are using the time modifier drop-down is that in the background, Splunk is defining the earliest time and the latest time in specific variables.

The **last 15 minutes** preset, for example, is equivalent to this SPL modifier:

```
SPL> earliest=-15m latest=now
```

The presets built into Splunk automatically insert the `latest=now` modifier. Run this search command in your Destinations app **Search** bar:

```
SPL> index=main earliest=-30m latest=now | timechart count span=5m
```

Notice that even if you have not changed the time modifier selected in the drop-down menu (which will not change unless you use it), the data will show that your earliest event is 30 minutes ago and your last data is recent. In other words, what you put in the search bar overrides the time modifier drop-down menu.

You can use a number of alternative ways to identify each of the time units; the most commonly supported time units listed by Splunk are:

- **Second**: s, sec, secs, second, seconds
- **Minute**: m, min, minute, minute, minutes
- **Hour**: h, hr, hrs, hour, hours
- **Day**: d, day, days
- **Week**: w, week, weeks
- **Month**: mon, month, months
- **Quarter**: q, qtr, qtrs, quarter, quarters
- **Year**: y, yr, yrs, year, years

Diving into Data – Search and Report for Admins

Table of Contents

Session 3-2: Diving into Data – Search and Report for Admins	199
Introduction	199
TIP	200
TIP	201
NOTE	203
Making raw event data readable.....	205
Getting ready	205
How to do it.....	205
How it works.....	207
TIP	208
There's more.....	208
TABULATING EVERY FIELD.....	208
REMOVING FIELDS, THEN TABULATING EVERYTHING ELSE	208
TIP	209
Finding the most accessed web pages.....	210
Getting ready	210
How to do it.....	210
How it works.....	211
There's more.....	212
SEARCHING FOR THE TOP 10 ACCESSED WEB PAGES.....	212
SEARCHING FOR THE MOST ACCESSED PAGES BY USER	212
NOTE	213
Finding the most used web browsers	214
Getting ready	214
How to do it.....	214
How it works.....	215
There's more.....	216
SEARCHING FOR THE WEB BROWSER DATA FOR THE MOST USED OS TYPES	216
Identifying the top-referring websites	217
Getting ready	217

How to do it.....	217
How it works.....	218
There's more.....	218
SEARCHING FOR THE TOP 10 USING STATS INSTEAD OF TOP.....	219
NOTE	219
Charting web page response codes.....	220
Getting ready	220
How to do it.....	220
How it works.....	221
There's more.....	222
TOTALING SUCCESS AND ERROR WEB PAGE RESPONSE CODES.....	222
Displaying web page response time statistics	224
Getting ready	224
How to do it.....	224
How it works.....	225
There's more.....	226
DISPLAYING WEB PAGE RESPONSE TIME BY ACTION	226
Listing the top viewed products.....	228
Getting ready	228
How to do it.....	228
How it works.....	229
There's more.....	230
SEARCHING FOR THE PERCENTAGE OF CART ADDITIONS FROM PRODUCT VIEWS.....	230
Charting the application's functional performance	232
Getting ready	232
How to do it...	232
How it works.....	233
TIP	234
There's more.....	234
Charting the application's memory usage	235
Getting ready	235

How to do it.....	235
How it works.....	236
Counting the total number of database connections.....	238
Getting ready	238
How to do it...	238
How it works.....	239

Session 3-2: Diving into Data – Search and Report for Admins

In this session, we will cover the basic ways to search data in Splunk. We will cover the following recipes:

- Making raw event data readable
- Finding the most accessed web pages
- Finding the most used web browsers
- Identifying the top-referring websites
- Charting web page response codes
- Displaying web page response time statistics
- Listing the top-viewed products
- Charting the application's functional performance
- Charting the application's memory usage
- Counting the total number of database connections

Introduction

In the previous session, we learned about the various ways to get data into Splunk. In this session, we will dive right into the data and get our hands dirty.

The ability to search machine data is one of Splunk's core functions, and it should come as no surprise that many other features and functions of Splunk are heavily driven-off searches. Everything from basic reports and dashboards to data models and fully featured Splunk applications are powered by Splunk searches behind the scenes.

Splunk has its own search language known as the **Search Processing Language (SPL)**. This SPL contains hundreds of search commands, most of which also have several functions, arguments, and clauses. While a basic understanding of SPL is required in order to effectively search your data in Splunk, you are not expected to know all the commands! Even the most seasoned ninjas do not know all

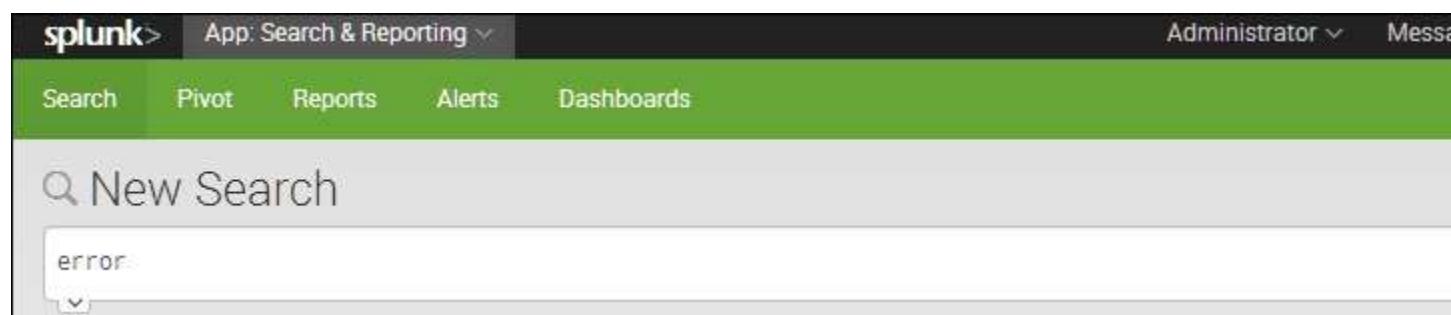
the commands and regularly refer to the Splunk manuals, website, or Splunk Answers (<http://answers.splunk.com>).

TIP

To get you on your way with SPL, be sure to check out the search command cheat sheet and download the handy quick reference guide available at <http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/SplunkEnterpriseQuickReferenceGuide>.

Searching

Searches in Splunk usually start with a base search, followed by a number of commands that are delimited by one or more pipe (`|`) characters. The result of a command or search to the left of the pipe is used as the input for the next command to the right of the pipe. Multiple pipes are often found in a Splunk search to continually refine data results as needed. As we go through this session, this concept will become very familiar to you.

A screenshot of the Splunk web interface. The top navigation bar shows 'splunk > App: Search & Reporting'. On the right, there are user dropdowns for 'Administrator' and 'Messages'. Below the bar, a green navigation menu has links for 'Search', 'Pivot', 'Reports', 'Alerts', and 'Dashboards'. A large search bar below the menu contains the text 'error'. To the left of the search bar is a magnifying glass icon.

Splunk allows you to search for anything that might be found in your log data. For example, the most basic search in Splunk might be a search for a keyword such as `error` or an IP address such as `10.10.12.150`. However, searching for a single word or IP over the terabytes of data that might potentially be in Splunk is not very efficient. Therefore, we can use the SPL and a number of Splunk commands to really refine our searches. The more refined and granular the search, the faster the time to run and the quicker you get to the data you are looking for!

TIP

When searching in Splunk, try to filter as much as possible before the first pipe (|) character, as this will save CPU and disk I/O. Also, pick your time range wisely. Often, it helps to run the search over a small time range when testing it and then extend the range once the search provides what you need.

Boolean operators

There are three different types of Boolean operators available in Splunk. These are `AND`, `OR`, and `NOT`. Case sensitivity is important here, and these operators must be in uppercase to be recognized by Splunk. The `AND` operator is implied by default and is not needed, but does no harm if used.

For example, searching for the term `error` or `success` would return all the events that contain either the word `error` or the word `success`. Searching for `error success` would return all the events that contain the words `error` and `success`. Another way to write this can be `error AND success`. Searching web access logs for `error OR success NOT mozilla` would return all the events that contain either the word `error` or `success`, but not those events that also contain the word `mozilla`.

Common commands

There are many commands in Splunk that you will likely use on a daily basis when searching data within Splunk. These common commands are outlined in the following table:

Command	Description
<code>chart/timechart</code>	This command outputs results in a tabular and/or time-based output for use by Splunk charts.
<code>dedup</code>	This command de-duplicates results based upon specified fields, keeping the most recent match.
<code>eval</code>	This command evaluates new or existing fields and values. There are many different functions available for <code>eval</code> .
<code>fields</code>	This command specifies the fields to keep or remove in search results.

Command	Description
<code>head</code>	This command keeps the first <code>x</code> (as specified) rows of results.
<code>lookup</code>	This command looks up fields against an external source or list, to return additional field values.
<code>rare</code>	This command identifies the least common values of a field.
<code>rename</code>	This command renames the fields.
<code>replace</code>	This command replaces the values of fields with another value.
<code>search</code>	This command permits subsequent searching and filtering of results.
<code>sort</code>	This command sorts results in either ascending or descending order.
<code>stats</code>	This command performs statistical operations on the results. There are many different functions available for <code>stats</code> .
<code>table</code>	This command formats the results into a tabular output.
<code>tail</code>	This command keeps only the last <code>x</code> (as specified) rows of results.
<code>top</code>	This command identifies the most common values of a field.
<code>transaction</code>	This command merges events into a single event based upon a common transaction identifier.

Time modifiers

The drop-down time range picker in the **Graphical User Interface (GUI)** to the right of the Splunk search bar allows users to select from a number of different preset and custom time ranges. However, in addition to using the GUI, you can also specify time ranges directly in your search string using the earliest and latest time modifiers. When a time modifier is used in this way, it automatically overrides any time range that might be set in the GUI time range picker.

The `earliest` and `latest` time modifiers can accept a number of different time units: seconds (`s`), minutes (`m`), hours (`h`), days (`d`), weeks (`w`), months (`mon`), quarters (`q`), and years (`y`). Time modifiers can also make use of the `@` symbol to round down and snap to a specified time.

For example, searching for `sourcetype=access_combined earliest=-1d@d latest=-1h` will search all the `access_combined` events from midnight a day ago until an hour ago from now. Note that the snap (@) will round down such that if it were 12 p.m. now, we would be searching from midnight a day and a half ago until 11 a.m. today.

Working with fields

Fields in Splunk can be thought of as keywords that have one or more values. These fields are fully searchable by Splunk. At a minimum, every data source that comes into Splunk will have the `source`, `host`, `index`, and `sourcetype` fields, but some source might have hundreds of additional fields. If the raw log data contains key-value pairs or is in a structured format such as JSON or XML, then Splunk will automatically extract the fields and make them searchable. Splunk can also be told how to extract fields from the raw log data in the backend `props.conf` and `transforms.conf` configuration files.

Searching for specific field values is simple. For example, `sourcetype=access_combined status!=200` will search for events with a `sourcetype` field value of `access_combined` that has a status field with a value other than `200`.

NOTE

Splunk has a number of built-in pre-trained `sourcetypes` that ship with Splunk Enterprise that might work with out-of-the-box, common data sources. These are available at <http://docs.splunk.com/Documentation/Splunk/latest/Data/Listofpretrainedsourcetypes>.

In addition, **Technical Add-Ons (TAs)**, which contain event types and field extractions for many other common data sources such as Windows events, are available from the Splunk app store at <https://splunkbase.splunk.com>.

Saving searches

Once you have written a nice search in Splunk, you may wish to save the search so that you can use it again at a later date or use it for a dashboard. Saved searches in Splunk are known as **Reports**. To save a

search in Splunk, you simply click on the **Save As** button on the top right-hand side of the main search bar and select **Report**.

The screenshot shows the Splunk interface with the following details:

- Top Bar:** Splunk logo, App: Search & Reporting, Administrator, Messages, Settings, Activity, Help.
- Header:** Search, Pivot, Reports, Alerts, Dashboards, Search & Reporting.
- Search Bar:** sourcetype=access_combined*, 39,543 events (before 3/7/14 5:37:06.000 PM).
- Context Menu (Save As):** Report (circled in red), Dashboard Panel, Alert, Event Type.
- Bottom Buttons:** Events (39,543), Statistics, Visualization, Format Timeline, Zoom Out, Zoom to Selection, Deselect, 1 day per column.

Making raw event data readable

When a basic search is executed in Splunk from the search bar, the search results are displayed in a raw event format by default. To many users, this raw event information is not particularly readable, and valuable information is often clouded by other less valuable data within the event. Additionally, if the events span several lines, only a few events can be seen on the screen at any one time.

In this recipe, we will write a Splunk search to demonstrate how we can leverage Splunk commands to make raw event data readable, tabulating events and displaying only the fields we are interested in.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and search results area.

How to do it...

Follow the given steps to search and tabulate the selected event data:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application from the drop-down menu located in the top left-hand side of the screen.



3. Set the time range picker to **Last 24 hours** and type the following search into the Splunk search bar:

```
index=main sourcetype=access_combined
```

Then, click on **Search** or hit **Enter**.

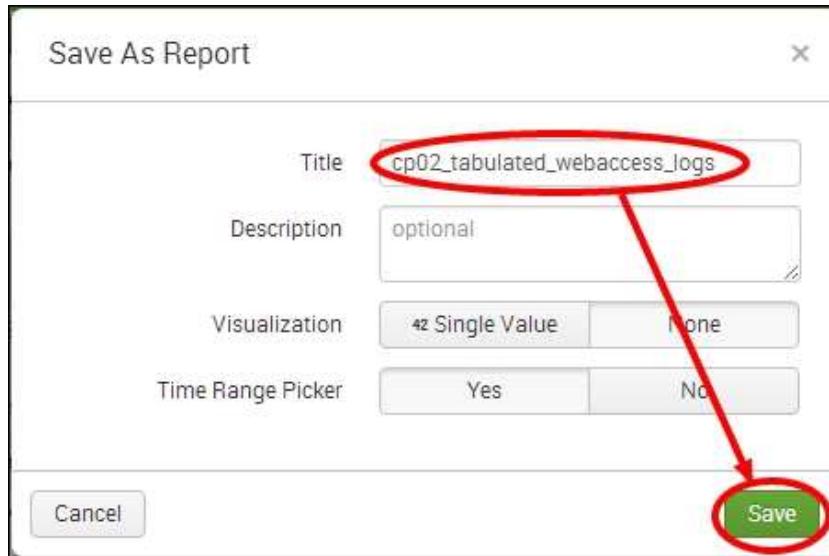
4. Splunk will return the results of the search and display the raw search events under the search bar.
5. Let's rerun the search, but this time we will add the `table` command as follows:

```
index=main sourcetype=access_combined | table _time,
referer_domain, method, uri_path, status, JSESSIONID, useragent
```

6. Splunk will now return the same number of events, but instead of presenting the raw events to you, the data will be in a nicely formatted table, displaying only the fields we specified. This is much easier to read!

_time	referer_domain	method	uri_path	status	JSESSIONID	useragent
2016-04-20 02:35:20	https://www1.samplesite.ca	GET	/viewCart	200	3EC32502F7653FA4FE3745FCC8043429	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0
2016-04-20 02:39:18	https://www1.samplesite.ca	POST	/removeItem	200	3EC32502F7653FA4FE3745FCC8043429	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0
2016-04-20 02:39:12	https://www1.samplesite.ca	GET	/viewCart	200	3EC32502F7653FA4FE3745FCC8043429	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0
2016-04-20 02:39:08	https://www1.samplesite.ca	POST	/addItem	200	3EC32502F7653FA4FE3745FCC8043429	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0

7. Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_tabulated_webaccess_logs` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.



How it works...

Let's break down the search piece by piece:

Search fragment	Description
index=main	All the data in Splunk is held in one or more indexes. While not strictly necessary, it is a good practice to specify the index (<code>es</code>) to search, as this will ensure a more precise search.
sourcetype=access_combined	This tells Splunk to search only the data associated with the <code>access_combined</code> sourcetype, which, in our case, is the web access logs.
table _time, referer_domain, method, uri_path, action, JSESSIONID, useragent	Using the <code>table</code> command, we take the result of our search to the left of the pipe and tell Splunk to return the data in a tabular format. Splunk will only display the fields specified after the <code>table</code> command in the table of results.

In this recipe, you used the `table` command. The `table` command can have a noticeable performance impact on large searches. It should be used towards the end of a search, once all the other processing on the data by the other Splunk commands has been performed.

TIP

The `stats` command is more efficient than the `table` command and should be used in place of `table` where possible. However, be aware that `stats` and `table` are two very different commands.

There's more...

The `table` command is very useful in situations where we wish to present data in a readable format. Additionally, tabulated data in Splunk can be downloaded as a CSV file, which many users find useful for offline processing in spreadsheet software or for sending to others. There are some other ways we can leverage the `table` command to make our raw event data readable.

TABULATING EVERY FIELD

Often, there are situations where we want to present every event within the data in a tabular format, without having to specify each field one by one. To do this, we simply use a wildcard (*) character as follows:

```
index=main sourcetype=access_combined | table *
```

REMOVING FIELDS, THEN TABULATING EVERYTHING ELSE

While tabulating every field using the wildcard (*) character is useful, you will notice that there are a number of Splunk internal fields, such as `_raw`, that appear in the table. We can use the `fields` command before the `table` command to remove the fields as follows:

```
index=main sourcetype=access_combined | fields - sourcetype,  
index, _raw, source date* linecount punct host time*  
eventtype | table *
```

If we do not include the minus (-) character after the `fields` command, Splunk will keep the specified fields and remove all the other fields.

TIP

If you regularly need to remove a number of fields in your searches, you can write a macro to do this and then simply call the macro from your search. Macros are covered later in this book.

Finding the most accessed web pages

One of the data samples we loaded in [Session 3-1, Play Time – Getting Data In](#), contained access logs from our web server. These have a Splunk sourcetype of `access_combined` and detail all pages accessed by the users of our web application. We are particularly interested in knowing which pages are being accessed the most, as this information provides great insight into how our e-commerce web application is being used. It could also help influence changes to our web application such that rarely visited pages are removed, or our application is redesigned to be more efficient.

In this recipe, we will write a Splunk search to find the most accessed web pages over a given period of time.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and the time range picker to the right of it.

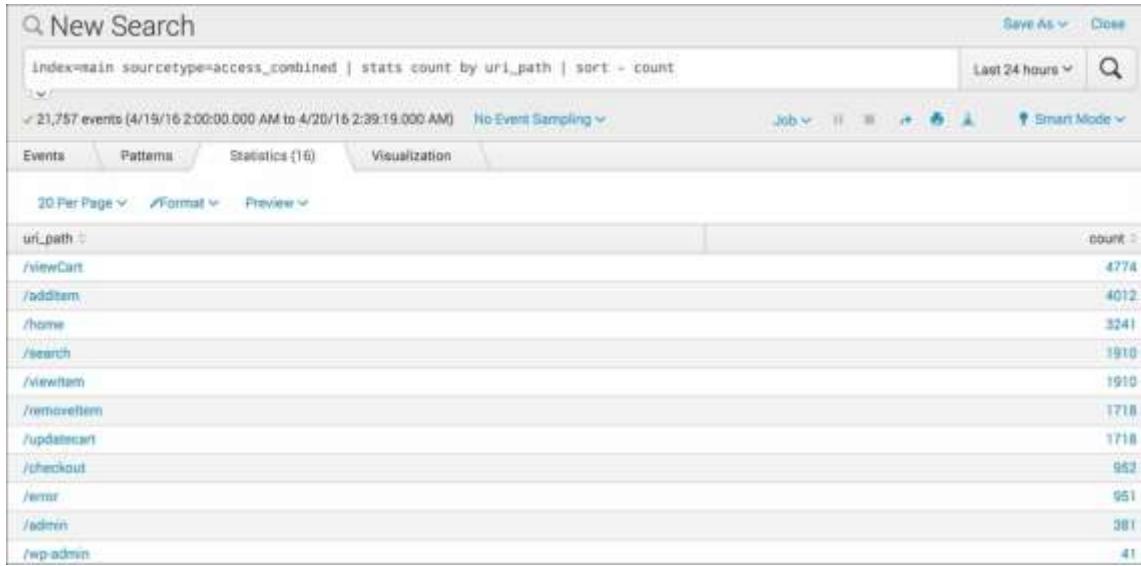
How to do it...

Follow the given steps to search for the most accessed web pages:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Set the range picker to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined | stats count by uri_path  
| sort - count
```

4. Splunk will return a list of pages, and a new field named **count** displays the total number of times a page has been accessed.



5. Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_most_accessed_webpages` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main</code>	All the data in Splunk is held in one or more indexes. While not strictly necessary, it is a good practice to specify the index(es) to search, as this will ensure a more precise search.
<code>sourcetype=access_combined</code>	This tells Splunk to search only the data associated with the <code>access_combined</code> sourcetype, which, in our case, is the web access logs.
<code> stats count by uri_path</code>	Using the <code>stats</code> command, we take the result of our search to the left-hand side of the pipe and tell Splunk to count the instances of each <code>uri_path</code> .

Search fragment	Description
	The <code>uri_path</code> field is the name of the field associated with the website page.
<code> sort - count</code>	Using the <code>sort</code> command, we take the <code>count</code> field generated by <code>stats</code> and tell Splunk to sort the results of the previous command in descending (-) order, such that the most visited web page appears at the top of the results.

There's more...

We can further build upon the base search to provide different variations of the results.

SEARCHING FOR THE TOP 10 ACCESSED WEB PAGES

We can modify the search from this recipe and replace the `stats` command with the `top` command. By default, this will display the top 10 web pages:

```
sourcetype=access_combined index=main | top uri_path
```

Here, we modified the search and replaced the `stats` command with the `top` command. By default, this displays the top 10 web pages. If we want to get the top 20 web pages, we can specify a limit value, as follows:

```
sourcetype=access_combined index=main | top limit=20  
uri_path
```

SEARCHING FOR THE MOST ACCESSED PAGES BY USER

We can modify the search from this recipe and can use the distinct count (`dc`) function of the `stats` command to display a list of users and the unique pages they visited:

```
sourcetype=access_combined index=main | stats dc(uri_path)  
by user | sort - user
```

The distinct count function ensures that if a user visits the same page multiple times, it is only counted as one visit. The user who visited the most number of unique pages will be at the top of the list, as we used a descending sort.

NOTE

For more information on the various functions that can be used with the `stats` command, check out <http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/CommonStatsFunctions>.

Finding the most used web browsers

Users visiting our website use a variety of devices and web browsers. By analyzing the web access logs, we can understand which browsers are the most popular and, therefore, which browsers our site must support at the least. We can also use this same information to help identify the types of devices that people are using.

In this recipe, we will write a Splunk search to find the most used web browsers over a given period of time. We will then make use of both the `eval` and `replace` commands to clean up the data a bit.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and the time range picker to the right of it.

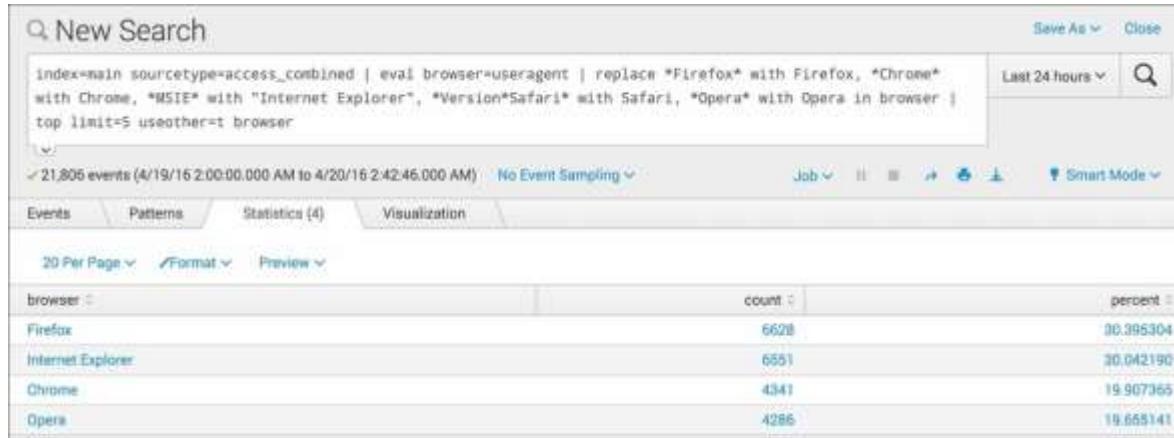
How to do it...

Follow the given steps to search for the most used web browsers:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined | eval browser=useragent  
| replace *Firefox* with Firefox, *Chrome* with Chrome, *MSIE*  
with "Internet Explorer", *Version*Safari* with Safari, *Opera*  
with Opera in browser | top limit=5 useother=t browser
```

4. Splunk will return a tabulated list of the top five most used web browsers on our site, by count and percent.



- Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_most_used_webbrowsers` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this session.
<code> eval browser=useragent</code>	Using the <code>eval</code> command, we evaluate a new field called <code>browser</code> and populate it with the contents of the <code>useragent</code> field.
<code> replace *Firefox* with Firefox, *Chrome* with Chrome, *MSIE* with "Internet Explorer", *Version*Safari* with Safari, *Opera* with Opera in browser</code>	Using the <code>replace</code> command, we use wildcards (*) within the content of the <code>browser</code> field to replace the values with shortened browser names. Note that the values that contain spaces require quotes around them, for example, "Internet Explorer".
<code> top limit=5 useother=t browser</code>	Using the <code>top</code> command, we tell Splunk to find the top five web browsers and classify everything else under the value of other.

In this recipe, we used both the `eval` and `replace` commands for illustrative purposes. This approach absolutely works, but a better approach can be to use Splunk's lookup functionality to look up the `useragent` value and return the browser name and version. Lookups are covered later in this book.

There's more...

Often, the same field values can be used in different ways to provide additional insight. In this case, the `useragent` field can be used to inform the types of devices that access our site.

SEARCHING FOR THE WEB BROWSER DATA FOR THE MOST USED OS TYPES

Let's modify the search to display the types of user operating systems that access our website:

```
index=main sourcetype=access_combined | eval os=useragent |  
replace *Windows* with Windows, *Macintosh* with Apple,  
*Linux* with Linux in os | top limit=3 useother=t os
```

When the search is run, you should see results similar to the following screenshot:



The search is similar, but this time we decided to pull the OS-related information from the `useragent` field and used it to compare access between major OS types.

Identifying the top-referring websites

Our web access logs continue to give us great information about our website and the users visiting the site. Understanding where our users are coming from provides insight into potential sales leads and/or which marketing activities might be working better over others. For this information, we look for the `referer_domain` field value within the log data.

In this recipe, we will write a Splunk search to find the top-referring websites.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and the time range picker.

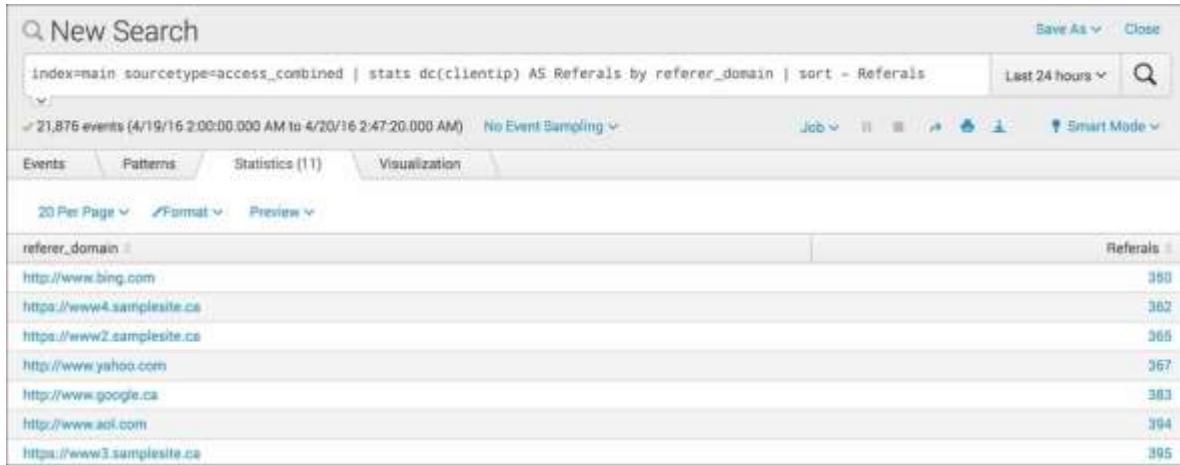
How to do it...

Follow the given steps to search for the top-referring websites:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined | stats dc(clientip) AS  
Referrals by referer_domain | sort - Referrals
```

4. Splunk will return a tabulated list ordered by the number of unique referrals each website has provided.



- Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_top_referring_websites` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this session.
<code> stats dc(clientip) AS Referrals by referer_domain</code>	Using the <code>stats</code> command, we apply the distinct count (<code>dc</code>) function to <code>clientip</code> to count the unique IP addresses by <code>referer_domain</code> and rename the generated count field to <code>Referrals</code> .
<code> sort - Referrals</code>	Using the <code>sort</code> command, we sort by the number of referrals in the descending order.

There's more...

In this recipe, we did not use the `top` command, as this command only provides limited functionality. The `stats` command is far more powerful and has many available functions, including distinct count.

SEARCHING FOR THE TOP 10 USING STATS INSTEAD OF TOP

Using the `stats` command in this recipe, we brought back all the websites present in our web access logs and then sorted them by the number of unique referrals. Should we want to only show the top 10, we can simply add the `head` command at the end of our search, as follows:

```
index=main sourcetype=access_combined | stats dc(clientip)
AS Referals by referer_domain | sort - Referals | head 10
```

The `head` command keeps the first specified number of rows. In this case, as we have a descending sort, by keeping the first 10 rows, we are essentially keeping the top 10. Instead of using the `head` command, we can also use the `limit` parameter of the `sort` command, as follows:

```
index=main sourcetype=access_combined | stats dc(clientip)
AS Referals by referer_domain | sort - Referals limit=10
```

NOTE

There is a great guide in the Splunk documentation to understand all the different functions for `stats`, `chart`, and `timechart`, available at <http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/CommonStatsFunctions>.

Charting web page response codes

Log data often contains seemingly cryptic codes that have all sorts of meanings. This is true of our web access logs, where there is a status code that represents a web page response. This code is very useful as it can tell us whether certain events were successful or not. For example, error codes found in purchase events are less than ideal, and if our website was at fault, then we might have lost a sale.

In this recipe, we will write a Splunk search to chart web page responses against the various web pages on the site.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and the time range picker.

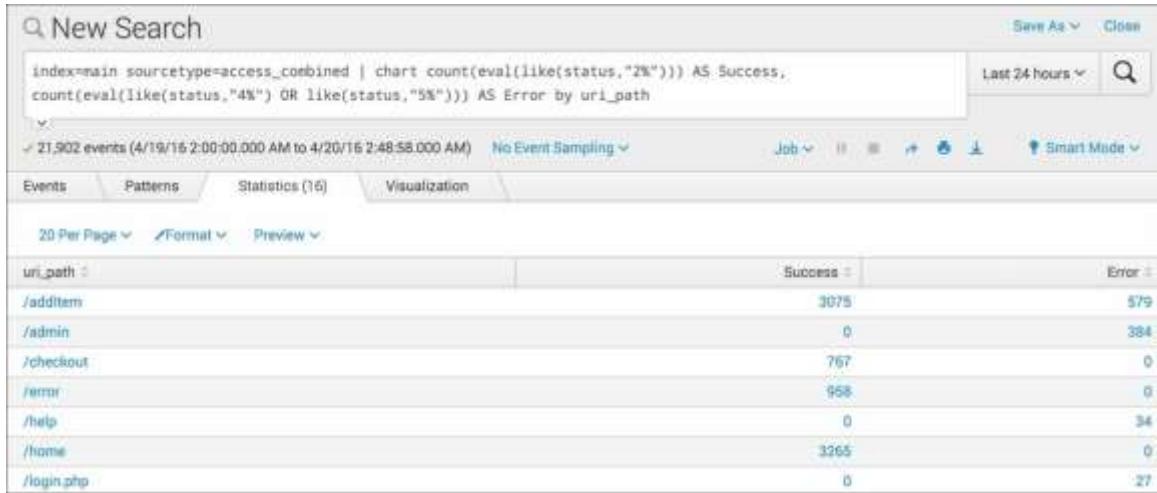
How to do it...

Follow the given steps to chart the web page response codes over time:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined | chart  
count(eval.like(status, "2%")) AS Success,  
count(eval.like(status, "4%") OR like(status, "5%")) AS Error by  
uri_path
```

4. Splunk will return a tabulated list of web pages, detailing for each page how many events were successful and how many generated errors.



5. Click on the **Visualization** tab, and you will see this data represented in a column (by default) chart.
6. Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_webpage_response_codes` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

In this recipe, we selected to search by the `uri_path` field. This field represents the various web pages on the site. Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this session.
<code> chart count(eval(iflike(status,"2%")) AS Success, count(eval(iflike(status,"4%") OR like(status,"5%")) AS Error) by uri_path</code>	Stripping away the complexity for a moment, this is very similar to performing stats count by <code>uri_path</code> . However, in this case, we are using the <code>chart</code> command and only counting success and error status codes. As the status field is essentially just a code, we are evaluating whether the code represents success or error. We

Search fragment	Description
	do this using an inline <code>eval</code> command with the <code>like</code> function. The <code>like</code> function allows us to specify the start of the status field value and then wildcard it with a <code>%</code> sign. Any status code beginning with <code>2</code> represents success events, and any status code beginning with either <code>4</code> or <code>5</code> represents an error.

There's more...

Hopefully, you can start to see the power of the SPL, as we start to ramp up the complexity a bit. We can now take this search a little further to provide a bit more insight.

TOTALING SUCCESS AND ERROR WEB PAGE RESPONSE CODES

We can further amend the search to show only the `addItem` and `checkout` web pages events, which seems a little more relevant to sales intelligence. Additionally, using the `addcoltotals` command, we can add up the total success and error events:

```
index=main sourcetype=access_combined uri_path="/addItem" OR
uri_path="/checkout" | chart count(eval(like(status,"2%")))
AS Success, count(eval(like(status,"4%") OR
like(status,"5%"))) AS Error by uri_path | addcoltotals
label=Total labelfield=uri_path
```

When this updated search is run, you should see results similar to the following screenshot:

New Search

Save As ▾ Close

```
index=main sourcetype=access_combined uri_path="/addItem" OR uri_path="/checkout" | chart
count(eval.like(status,"2%")) AS Success, count(eval.like(status,"4%") OR like(status,"5%")) AS Error by uri_path |
addcoltotals label=Total labelfield=uri_path
```

Last 24 hours ▾

✓ 5,001 events (4/19/16 2:00:00.000 AM to 4/20/16 2:50:23.000 AM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns Statistics (3) Visualization

20 Per Page ▾ Format ▾ Preview ▾

uri_path	Success	Error
/addItem	3079	579
/checkout	768	0
Total	3847	579

We use `labelfield=uri_path` and `label=Total` to tell Splunk to place a value of **Total** in the `uri_path` field column.

Displaying web page response time statistics

No one likes to wait for a web page to load, and we certainly do not want users of our web application waiting either! Within our web access logs, there is a field named `response` that tracks the total time the page has taken to load in milliseconds.

In this recipe, we will track the average page load time over the past week at different times of the day.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and the time range picker.

How to do it...

Follow the given steps to search and calculate the web page response time statistics over the past week:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 7 days** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
sourcetype=access_combined | timechart span=6h avg(response) AS avg_response | eval avg_response=round(avg_response/1000,2)
```

4. Splunk will return a tabulated list, detailing the average response time for every 6-hour period, going back a week.

New Search		Save As ▾	Close
<pre>sourcetype=access_combined timechart span=6h avg(response) AS avg_response eval avg_response=round(avg_response/1000,2)</pre>		Last 7 days ▾	Search
<input checked="" type="checkbox"/> 149,052 events (4/13/16 2:00:00.000 AM to 4/20/16 2:52:05.000 AM) No Event Sampling ▾		Job ▾	Smart Mode ▾
Events	Patterns	Statistics (29)	Visualization
20 Per Page ▾	Format ▾	Preview ▾	< Prev 1 2 Next >
_time		avg_response	
2016-04-13 00:00		0.04	
2016-04-13 06:00		0.05	
2016-04-13 12:00		0.04	
2016-04-13 18:00		0.04	
2016-04-14 00:00		0.04	
2016-04-14 06:00		0.04	

- This is great, but hard to visualize in a tabular form. Click on the **Visualization** tab and you will see this data represented as a chart.
- Click on the chart type link in the upper-left of the chart (next to the **Format** link) and select **Line** if not already selected. Splunk now presents this data in a nice line chart, and we can now see the average response time at different times of the day much more clearly.



- Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_webpage_response_times` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
index=main sourcetype=access_combined	You should now be familiar with this search from the earlier recipes in this session.
timechart span=6h avg(response) AS avg_response	Using the <code>timechart</code> command, we specify a span of 6 hours. We then use the <code>avg</code> function on the response field. Splunk will add up all the response times in the 6-hour period and then calculate the average response time during that period.
eval avg_response=round(avg_response/1000,2)	Using the <code>eval</code> command, we calculate the average response time in seconds by dividing the average time (which is in milliseconds) by <code>1000</code> , to give us the time in seconds. The number <code>2</code> at the end is part of the round function and tells Splunk to round to <code>2</code> decimal places.

There's more...

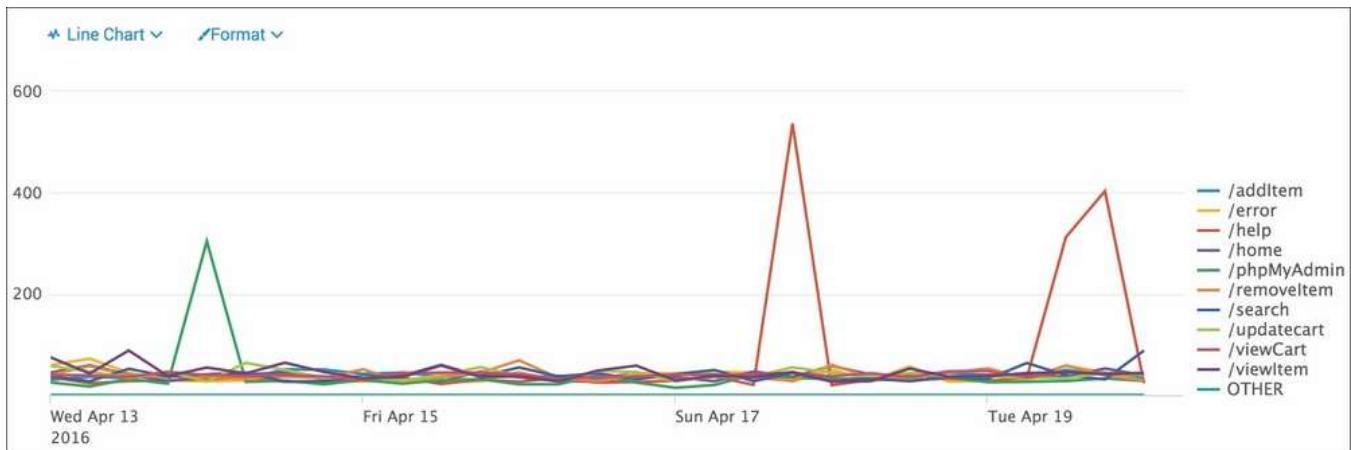
The `timechart` command offers some great functionality. Searches like this can be extended further to graphically compare several weeks against one another to spot anomalies and other issues.

DISPLAYING WEB PAGE RESPONSE TIME BY ACTION

We can further amend the search to offer granular information on average response time by the type of action being performed. This might pinpoint some actions that are less responsive than the others. For example, we might want to ensure that the checkout page remains at an optimal load time. For the following search to work, you must complete the *Defining field extractions* recipe in [Session 3-1, Play Time – Getting Data In](#), to extract the response field:

```
sourcetype=access_combined uri_path=* | timechart span=6h
avg(response) by uri_path | foreach * [eval
<<FIELD>>=round(<<FIELD>>/1000,2)]
```

We are now searching for web page events and then we will calculate the average time by page (`uri_field`). This results in a table of multiple columns, where each column represents a different web page. When we visualize this on a line graph, we now see many different lines on the same chart—pretty cool! You will notice that we used a pretty advanced Splunk search command, earlier named `foreach`. This is essentially a for type loop that cycles through each of the column fields in the table and applies a calculation to convert the average time by page from milliseconds to seconds while rounding the value to two decimal places:



Listing the top viewed products

Our web access logs capture the product IDs (the item field in the logs) that the users view and add to their shopping carts. Understanding the top products that people view can help influence our sales and marketing strategy, and even product direction. Products viewed on an e-commerce website might not always necessarily translate into sales of that product though.

In this recipe, we will write a Splunk search to chart the top 10 products that users successfully view and compare against the number of successful shopping cart additions for each product. For example, if a product has a high number of views but the product is not added to carts and subsequently purchased, this could indicate that something is not right—perhaps the pricing of the product is too high.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and the time range picker.

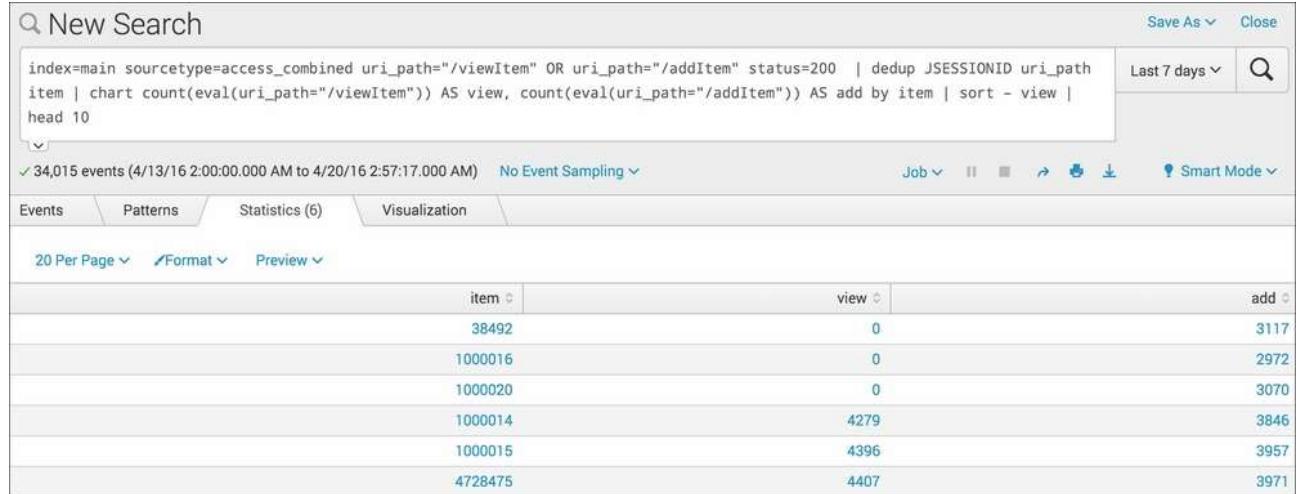
How to do it...

Follow the given steps to search for the top products being searched over the past week:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 7 days** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined uri_path="/viewItem" OR
uri_path="/addItem" status=200 | dedup JSESSIONID uri_path
item | chart count(eval(uri_path="/viewItem")) AS view,
count(eval(uri_path="/addItem")) AS add by item | sort - view |
head 10
```

- Splunk will return a tabulated list of items (products), detailing the number of times a product was successfully viewed versus the number of times that product was actually added to the shopping cart.



- Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_top_products_viewed` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

In this recipe, our search returned a count by item of how many items were viewed versus how many were added to the cart. In this case, the item field represents a unique item ID that pertains to a specific product. Let's break down the search piece by piece:

Search fragment	Description
<code>index=main</code> <code>sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this session.
<code>uri_path="/ViewItem" OR</code> <code>uri_path="/addItem" status=200</code>	Following best practice of making our search as granular as possible, we only search for events that contain <code>uri_paths</code> related to viewing items and adding items, and have a successful status code of <code>200</code> . This type of granularity greatly limits the number of

Search fragment	Description
	records we search, making our search a lot faster.
dedup JSESSIONID uri_path item	Using the <code>dedup</code> command, we de-duplicate our data by the <code>JSESSIONID</code> , the <code>uri_path</code> , and the <code>item</code> values. Why? Well, because a user in a given session could view a product many times in that session before adding it, so we want to ensure that we only count one view and one addition per user session of a product.
chart count(eval(uri_path="/viewItem")) AS view, count(eval(uri_path="/addItem")) AS add by item	Using the <code>chart</code> and <code>eval</code> commands, we count the number of views and adds by item.
sort - view head 10	Using the <code>sort</code> command, we sort in descending order on the <code>view</code> field, such that the items with the most number of views are at the top. We then leverage the <code>head</code> command to keep only the first 10 rows of data, leaving us with the top 10 searched products.

There's more...

This recipe provides us with some insight into product views and subsequent shopping cart additions that might then lead on to a sale. However, we can keep adding to the search to make it even easier to understand the relationship between the two.

SEARCHING FOR THE PERCENTAGE OF CART ADDITIONS FROM PRODUCT VIEWS

We can further amend the search from this recipe to evaluate a new column that calculates the percentage of product views added to the

cart. We do this using the `eval` command and some basic math, as follows:

```
index=main sourcetype=access_combined uri_path="/ViewItem"
OR uri_path="/addItem" status=200 | dedup JSESSIONID
uri_path item | chart count(eval(uri_path="/ViewItem")) AS
view, count(eval(uri_path="/addItem")) AS add by item | sort
- view | head 10 | eval
cart_conversion=round(add/view*100).%"
```

We firstly evaluate a new field called `cart_conversion` and then calculate the percentage, dividing purchase by view and multiplying by `100`. We use the round function of `eval` to eliminate the decimal places and then tack on the `%` character at the end. Now, we can easily see what percentage of views lead to cart additions.

Charting the application's functional performance

Another of the data samples we loaded in [Session 3-1, Play Time – Getting Data In](#), contained application logs from our application server. These have a Splunk sourcetype of `log4j` and detail the various calls that our application makes to the backend database in response to user web requests, in addition to providing insight into memory utilization and other health-related information. We are particularly interested in tracking how our application is performing in relation to the time taken to process user-driven requests for information.

In this recipe, we will write a Splunk search to find out how our application is performing. To do this, we will analyze database call transactions and chart the maximum, mean, and minimum transaction durations over the past week.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and the time range picker.

How to do it...

Follow the given steps to chart the application's functional performance over the past week:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=log4j | transaction maxspan=4h threadId |
timechart span=6h max(duration) AS max, mean(duration) AS mean,
min(duration) AS min
```

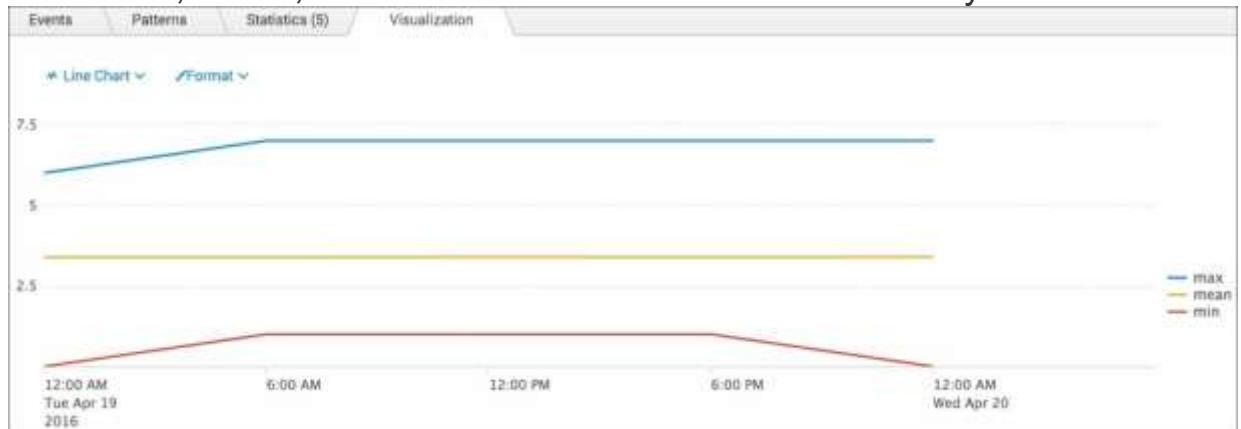
- Splunk will return a tabulated list, detailing the maximum, mean, and minimum database transaction durations for every 6-hour period, going back the last 24 hours.

The screenshot shows a Splunk search interface with the following details:

- Search Bar:** index=main sourcetype=log4j | transaction maxspan=4h threadId | timechart span=6h max(duration) AS max, mean(duration) AS mean, min(duration) AS min
- Time Range:** Last 24 hours
- Results:** 16,642 events (4/19/16 2:00:00.000 AM to 4/20/16 2:59:15.000 AM)
- Event Sampling:** No Event Sampling
- Job Status:** Job
- Visualization Tab:** Selected
- Format:** 20 Per Page, Format, Preview
- Data Table:**

time	max	mean	min
2016-04-19 00:00	6	3.389438	0
2016-04-19 06:00	7	3.386790	1
2016-04-19 12:00	7	3.396663	1
2016-04-19 18:00	7	3.391586	1
2016-04-20 00:00	7	3.395960	0

- This is great, but hard to visualize in a tabular form. Click on the **Visualization** tab and you will see this data represented as a chart.
- Click on the chart type link in the upper-left of the chart (next to the **Format** link) and select **Line** if not already selected. Splunk now presents this data in a nice line chart, and we can now see the maximum, mean, and minimum levels much more clearly.



- Save this search by clicking on **Save As** and then on **Report**. Give the report the name [cp02_application_performance](#) and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
index=main sourcetype=log4j	In this example, we search for our application logs which have the <code>log4j</code> sourcetype.
transaction maxspan=4h threadId	<p>Using the <code>transaction</code> command, we essentially consolidate multiple events with a common <code>threadId</code> into single event, multiline transactions. The <code>maxspan</code> function tells Splunk to only look at events with the same <code>threadId</code> that are within 4 hours of each other.</p> <p>The <code>transaction</code> command also calculates a new field called <code>duration</code>. This is the <code>duration</code> in seconds from the first event in the transaction to the last event in the transaction.</p>
<code> timechart span=6h max(duration) AS max, mean(duration) AS mean, min(duration) AS min</code>	Using the <code>timechart</code> command, we specify a span of 6 hours. We then use the <code>max</code> , <code>mean</code> , and <code>min</code> functions on the duration field. Splunk analyzes the durations in the 6-hour period and then calculates the max, mean, and min durations during this period.

TIP

The `transaction` command is an extremely resource intensive (CPU/memory) search command. When using this command, be sure to use the `maxspan` function where possible, as this helps focus on transactions grouped only within the specified `maxspan` timeframe.

There's more...

In this recipe, we leveraged the `transaction` command. This is a very useful and powerful function, and we will revisit it in more depth and complexity later in the book.

Charting the application's memory usage

In addition to measuring functional performance of database transactions, we are also interested in understanding how our application is performing from a memory usage perspective. Analyzing this type of information can help identify memory leaks in our application or high-memory utilization that might be affecting the user experience and causing our application to slow down.

In this recipe, we will analyze the memory usage of our application over time.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar and the time range picker.

How to do it...

Follow the given steps to chart the application memory usage over the past day:

1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

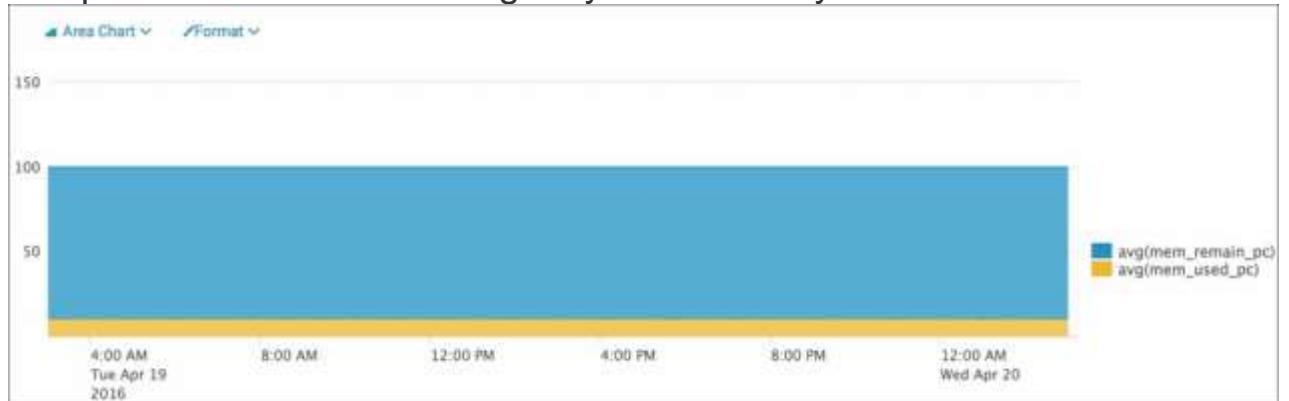
```
index=main sourcetype=log4j perfType="MEMORY" | eval  
mem_used_pc=round((mem_used/mem_total)*100) | eval  
mem_remain_pc=(100-mem_used_pc) | timechart span=15m  
avg(mem_remain_pc) avg(mem_used_pc)
```

4. Splunk will return a tabulated list, detailing all the events that meet our search criteria.

New Search		Save As ↘	Close
<input style="width: 100%; height: 40px; border: 1px solid #ccc;" type="text" value="index=main sourcetype=log4j perfType='MEMORY' eval mem_used_pc=round((mem_used/mem_total)*100) eval mem_remain_pc=(100-mem_used_pc) timechart span=15m avg(mem_remain_pc) avg(mem_used_pc)"/>		Last 24 hours ↘	<input type="button" value="Search"/>
✓ 16,053 events (4/19/16 3:00:00.000 AM to 4/20/16 3:02:05.000 AM) No Event Sampling ↘		Job ↘	<input type="button" value="Smart Mode"/>
Events Patterns Statistics (97)		Visualization	
20 Per Page ↘	Format ↘	Preview ↘	Prev 1 2 3 4 5 Next

_time	avg(mem_remain_pc)	avg(mem_used_pc)
2016-04-19 03:00:00	89.880952	10.119048
2016-04-19 03:15:00	90.113772	9.886228
2016-04-19 03:30:00	89.994063	10.005917
2016-04-19 03:45:00	89.880952	10.119048
2016-04-19 04:00:00	90.048193	9.951807

5. This is great, but hard to visualize in a tabular form. Click on the **Visualization** tab and you will see this data represented in a column (by default) chart.
6. Click on the column link the chart and select **Area**. Then, click on the **Format** link and change **Stack Mode** to **Stacked** and click on **Apply**. Splunk now presents this data in an area chart, allowing us to easily see if there are times during the day when our application might be getting low on memory. In this case, our sample data looks to be using very little memory.



7. Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_application_memory` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
index=main sourcetype=log4j perfType="MEMORY"	In this example, we search for our application logs which have the <code>log4j</code> sourcetype. We also select to view only the memory-related events.
eval mem_used_pc=round((mem_used/mem_total)*100)	Using the <code>eval</code> command, we calculate the percentage of memory used from the <code>mem_used</code> and <code>mem_total</code> fields in our application log.
eval mem_remain_pc=(100-mem_used_pc)	Using the <code>eval</code> command again, we calculate the remaining percentage of memory from the used percentage of memory that we just calculated in the previous step.
timechart span=15m avg(mem_remain_pc) avg(mem_used_pc)	Using the <code>timechart</code> command, we calculate the average remaining percentage of memory and the used percentage of memory for every 15-minute interval over the past day.

Counting the total number of database connections

Our application only allows for a limited number of concurrent database connections currently. As our application user base grows, we need to proactively monitor these connections to ensure that we do not hit our concurrency limit or to know when we need to further scale out the database infrastructure.

In the last recipe of this session, we will monitor database transactions over the past week to identify if there are certain times or days when we might be close to our concurrency limit.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from Session 3-1, *Play Time – Getting Data In*. You should be familiar with the Splunk search bar and the time range picker.

How to do it...

Follow the given steps to search for the total number of database connections over the past 30 days:

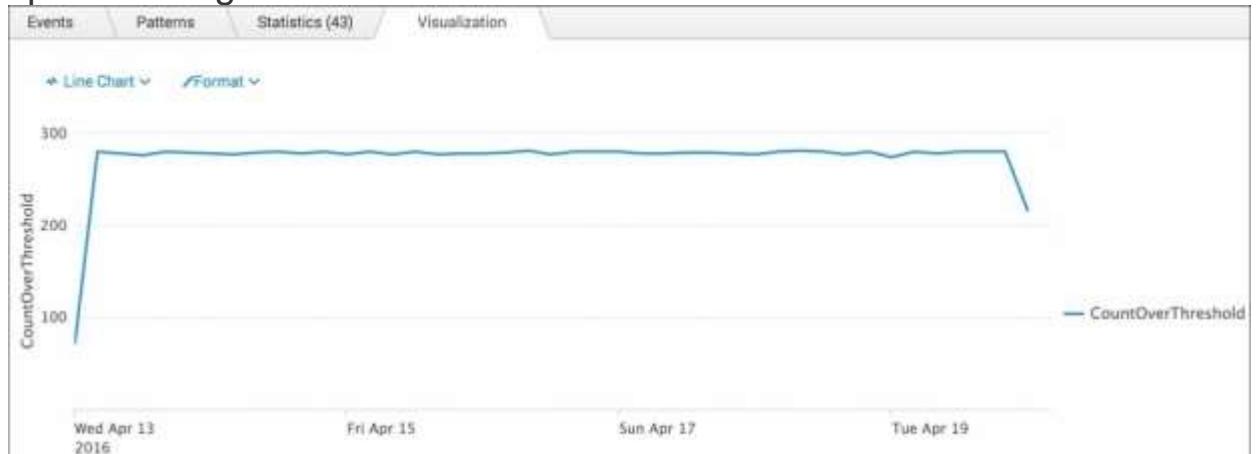
1. Log in to your Splunk server.
2. Select the **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 7 days** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=log4j perfType="DB" | eval  
threshold=con_total/100*70 | where con_used>=threshold |  
timechart span=4h count(con_used) AS CountOverThreshold
```

4. Splunk will return a tabulated list, detailing all the events that meet our search criteria.

New Search		Save As	Close
<pre>index=main sourcetype=log4j perfType="DB" eval threshold=con_total/100*70 where con_used>=threshold timechart span=4h count(con_used) AS CountOverThreshold</pre>		Last 7 days	
<input checked="" type="checkbox"/> 11,713 events (4/13/16 3:00:00.000 AM to 4/20/16 3:05:05.000 AM) No Event Sampling		Job	
Events	Patterns	Statistics (43)	Visualization
20 Per Page			(Prev) Next >
_time			CountOverThreshold
2016-04-13 00:00			70
2016-04-13 04:00			280
2016-04-13 08:00			278
2016-04-13 12:00			276
2016-04-13 16:00			280
2016-04-13 20:00			279
2016-04-14 00:00			278

5. This is great, but hard to visualize in a tabular form. Click on the **Visualization** tab and you will see this data represented in a column (by default) chart.
6. Click on the column link in the chart and select **Line**. Splunk now presents this data in a line chart, allowing us to easily see any spikes during certain times of the week.



7. Save this search by clicking on **Save As** and then on **Report**. Give the report the name `cp02_application_db_connections` and click on **Save**. On the next screen, click on **Continue Editing** to return to the search.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<pre>index=main sourcetype=log4j perfType="DB"</pre>	In this example, we search for our application logs which have the <code>log4j</code> sourcetype. We also select to view only the events related to database (<code>DB</code>).
<pre> eval threshold=con_total/100*70</pre>	Using the <code>eval</code> command, we calculate a new field called <code>threshold</code> , which is 70 percent of the total connections permitted.
<pre> where con_used>=threshold</pre>	Using the <code>where</code> command, we search for only those events that are greater than or equal to the 70 percent threshold we just defined.
<pre> timechart span=4h count(con_used) AS CountOverThreshold</pre>	Finally, we count the number of times over a 4-hour period in which the connection limit is greater than or equal to our threshold.

Diving Deeper – Advanced Searching

In this chapter, we will cover some of the more advanced search commands available within Splunk. We will cover the following recipes:

- Calculating the average session time on a website
- Calculating the average execution time for multi-tier web requests
- Displaying the maximum concurrent checkouts
- Analyzing the relationship of web requests
- Predicting website-traffic volumes
- Finding abnormally sized web requests
- Identifying potential session spoofing

Introduction

In the previous chapter, we learned about Splunk's new data model and Pivot functionality and how they can be used to further intelligence reporting. In this chapter, we will return to Splunk's SPL, diving deeper and making use of some very powerful search commands to facilitate a better understanding and correlation of event data. You will learn how to create transactions, build subsearches and understand concurrency, leverage field associations, and so on.

Looking at event counts, applying statistics to calculate averages, or finding the top values over time only provide a view of the data limited to one angle. Splunk's SPL contains some very powerful search commands that provide the ability to correlate data from different sources and understand or build relationships between the events. Through the building of relationships between data sets and looking at different angles of the data, you can better understand the impact one event might have on another. Additionally, correlating related values can provide a much more contextual value to teams when reviewing or analyzing a series of data.

Identifying and grouping transactions

Single events can be easily interpreted and understood, but these single events are often part of a series of events, where the event might be influenced by the preceding events or might affect the other events to come. By leveraging Splunk's ability to group associated events into transactions based on field values, the data can be presented in such a way that the reader understands the full context of an event and gets what led up to this point. Building transactions can also be useful when needing to understand the time duration between the start and finish of specific events or calculating values within a given transaction and comparing them to the values of others.

Converging data sources

Context is everything when it comes to building successful Operational Intelligence, and when you are stuck analyzing events from a single data source at a time, you might miss out on rich contextual information that other data sources could provide. With Splunk's ability to converge multiple data sources using the [join](#) or [append](#) search commands and search across them as if they were a single source, you can easily enrich the single data source and understand events from other sources that occurred at, or around, the same time.

For example, you might notice there are more timeouts than usual on your website, but when you analyze the website access log, everything appears normal. However, when you look at the application log, you notice that there are numerous failed connections to the database. Even so, by looking at each data source individually, it is hard to understand where the actual issue lies. Using Splunk's SPL to converge the data sources will allow for both the web access and application logs to be brought together into one view, to better understand and troubleshoot the sequence of events that might lead to website timeouts.

Identifying relationships between fields

In the Operational Intelligence world, the ability to identify relationships between fields can be a powerful asset. Understanding the values of a field, and how these values might have a relationship with the other field values within the same event, allows you to calculate the degree of certainty the values will provide in future events. By continually sampling events as they come in over time, you can become more accurate at

predicting values in events as they occur. When used correctly, this can provide tremendous value in being able to actively predict the values of fields within events, leading to a more proactive incident or issue identification.

Predicting future values

Understanding system, application, and user behavior will always prove to be extremely valuable when building any intelligence program; however, the ability to predict future values can provide values more immense than simple modeling actions. The addition of predictive capabilities to an Operational Intelligence program enables the ability to become more proactive to issue identification, forecast system behavior, and plan and optimize thresholds more effectively.

Imagine being able to predict the number of sessions on your website, the number of purchases of a specific product, the response times during peak periods, or general tuning alerting thresholds to values that are substantiated rather than taking an educated guess. All of this is possible with predictive analytics; by looking back over the past events, you can better understand what the future will hold.

Discovering anomalous values

With the volume of data ever increasing, looking for events that are outliers is becoming more difficult and requires different techniques for their detection. The value of identifying these values is that it can lead to the identification of a resource issue, highlight malicious activities hidden within high volumes of events, or simply detect the users attempting to interact with the application in a way they were not designed to. Capitalize on these opportunities to capture the abnormalities and triage them accordingly.

LAB-Calculating the average session time on a website

In the previous chapters, we created methods to assess the various values that show how consumers interact with our website. However, what these values did not outline was how long these consumers spend on our website. By leveraging Splunk's more powerful search commands, we can calculate the average session time of the consumers interacting with our website, which can act as supporting information when articulating data such as engagement rates, resource requirements, or consumer experience.

In this recipe, you will write a Splunk search to calculate the average time of a session on the website over a given period of time. You will then graphically display this value on a dashboard using the single value visualization.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Chapter 1, Play Time – Getting Data In](#). You should be familiar with navigating the Splunk user interface.

How to do it...

Follow the steps in this recipe to calculate the average session time on a website:

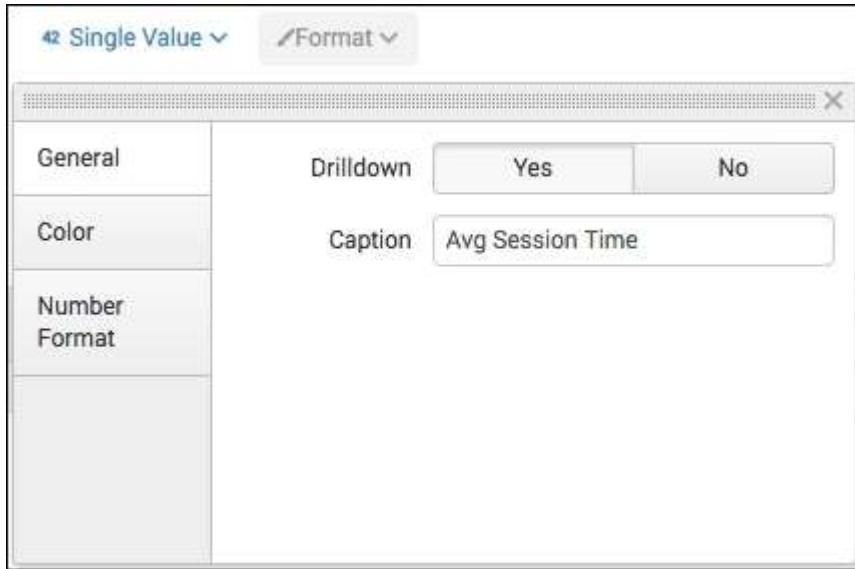
1. Log in to your Splunk server.
2. Select the **Operational Intelligence** application.
3. Ensure the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on the magnifying glass button or hit *Enter*.

```
index=main sourcetype=access_combined | transaction JSESSIONID  
| stats avg(duration) AS Avg_Session_Time
```

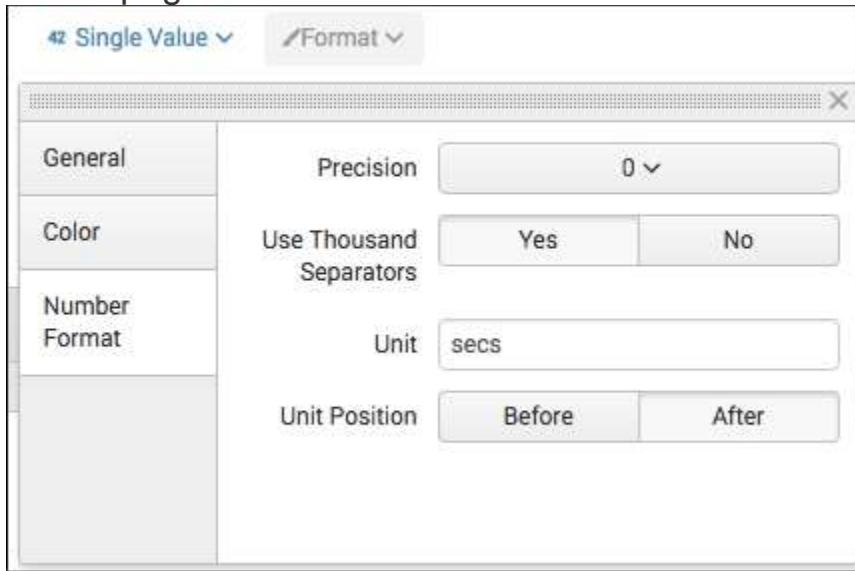
4. Depending on your Splunk server, this might take a little while to run. Splunk will return a single value representing the average duration in seconds for a session on the website.
5. Click on the **Visualization** tab.
6. Since there are a number of visualizations within Splunk, the single value visualization might not be displayed by default within the **Visualization** tab. Click on the dropdown listing the visualization types and select **Single Value**.



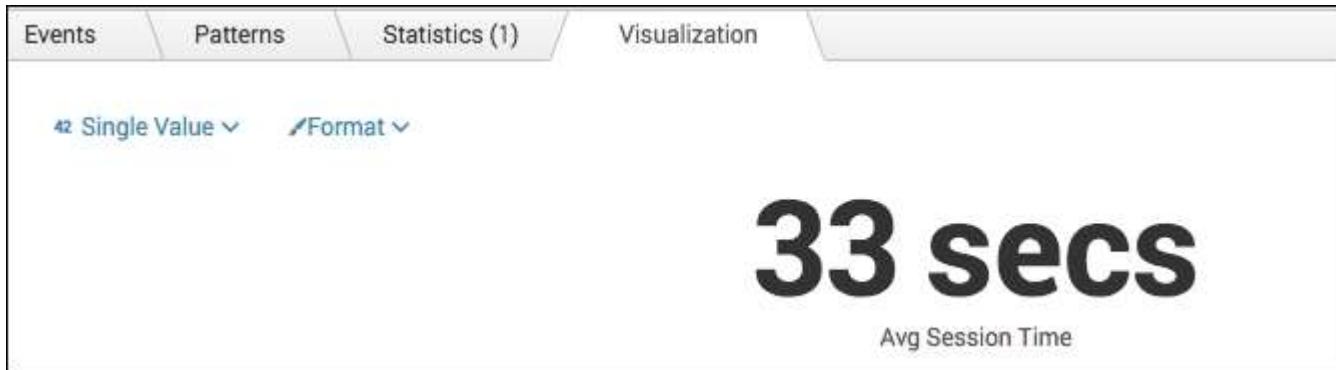
7. You should now see the value represented as single value visualization.
8. Let's add more context to the visualization. Click on **Format**. Enter `Avg Session Time` in **Caption**.



9. Click on **Number Format**. Enter `secs` in **Unit** and then click anywhere on the page.



10. Your single value visualization should now look similar to the following example:



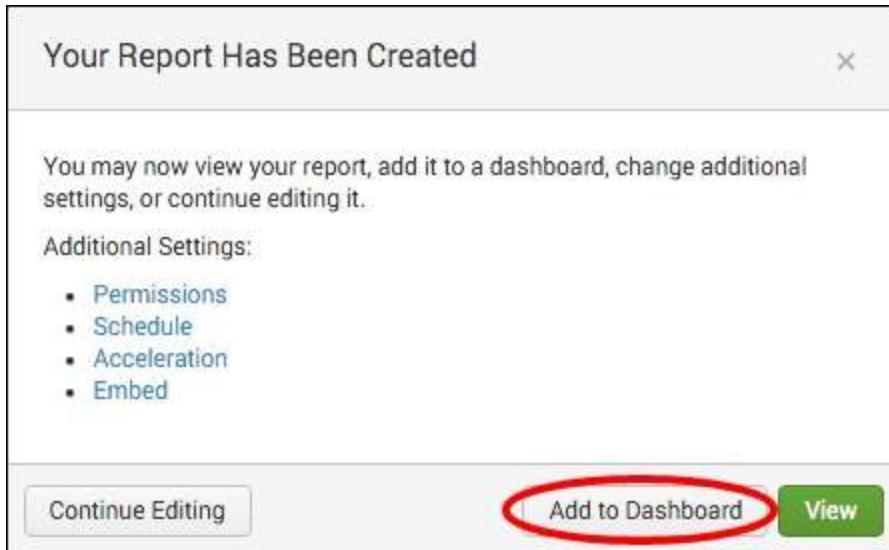
11. Let's save this search as a report. Click on **Save As** and choose **Report** from the drop-down menu.



12. In the **Save As Report** window that appears, enter `cp06_average_session_time` as the title and click on **Save**.

A screenshot of the "Save As Report" dialog box. The "Title" field contains the text "cp06_average_session_time", which is circled in red. The "Description" field is set to "optional". The "Content" section shows a search bar with the number "42". The "Time Range Picker" section has "Yes" selected. At the bottom, there are "Cancel" and "Save" buttons, with "Save" also circled in red.

13. You will receive confirmation that your report has been created. Now, let's add this report to a dashboard. In the next window, click on **Add to Dashboard**.



14. You will create a new dashboard for this report. On the **Save As Dashboard Panel** screen, ensure **New** is selected and enter [Session Monitoring](#) as the **Dashboard Title**. Select **Shared** in **App** for the **Dashboard Permissions** and **Report to Panel Powered By**. Finally, click on **Save** to create the dashboard.

Save As Dashboard Panel

Dashboard	<input checked="" type="radio"/> New	<input type="radio"/> Existing
Dashboard Title	Session Monitoring	
Dashboard ID?	session_monitoring Can only contain letters, numbers and underscores.	
Dashboard Description	optional	
Dashboard Permissions	<input type="radio"/> Private	<input checked="" type="radio"/> Shared in App
Panel Title	optional	
Panel Powered By	<input type="checkbox"/> Inline Search <input checked="" type="checkbox"/> Report	
Panel Content	<input type="checkbox"/> Statistics	<input type="checkbox"/> Single Value
<input type="button" value="Cancel"/> <input checked="" type="button" value="Save"/>		

15. The report is saved and a new **Session Monitoring** dashboard is created. You can now choose to click on **View Dashboard** to see your newly created dashboard with the **Average Session Time** report.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should be familiar with this search from the recipes in chapters. It is used to return events from the website access.
<code> transaction JSESSIONID</code>	Using the transaction command, we group the events together given <code>JSESSIONID</code> to form a single transaction. The <code>JSESSIONID</code> chosen as each visitor to the website is given a random se

Search fragment	Description
	whose value is stored in this field. One of the fields created by the transaction command is the duration field. The duration field contains the amount of time, in seconds, between the first and last transaction.
<code> stats avg(duration) AS Avg_Session_Time</code>	Using the stats command, we calculate the average value of the duration field. Using the <code>AS</code> operator, we rename the resulting field to something more descriptive, such as <code>Avg_Session_Time</code> .

There's more...

The `transaction` command provides many parameters to control the way in which transactions are grouped. Using the `startswith` and `endswith` parameters, you can control what marks the start and end of a transaction based on data inside the events. Using the `maxspan`, `maxpause`, or `maxevents` parameter, you can control the constraints around how long a transaction will be, the amount of time between events before splitting it into a new transaction, or the total number of events within a transaction.

TIP

Where possible, using the parameters available for the `transaction` command is highly encouraged. Using the `transaction` command without any other parameter can result in a processing-intensive (and inefficient) search that takes a while to run.

STARTS WITH A WEBSITE VISIT, ENDS WITH A CHECKOUT

To mark where a transaction begins and ends, you can make use of two parameters available within the `transaction` command, called `startswith` and `endswith`, respectively. In the following example, we modify the search in the recipe to include the `startswith="GET /home"` and `endswith="checkout"` parameters. This constrains the `transaction` command to only group events together with a general website request when the first event begins and last the event is a request

to checkout. Any other event, or transaction, that does not meet these criteria will be discarded and not included in the returned results.

```
index=main sourcetype=access_combined | transaction  
JSESSIONID startswith="GET /home" endswith="checkout" |  
stats avg(duration) AS Avg_Session_Time
```

By making use of these parameters, you can be more explicit on what gets treated as a transaction or focus on specific groupings of data.

DEFINING MAXIMUM PAUSE, SPAN, AND EVENTS IN A TRANSACTION

Three more very useful parameters available, apart from the `transaction` command, are `maxpause`, `maxspan`, and `maxevents`. These parameters allow you to apply more constraints around the duration and size of the transactions and can be used individually or all together for even more precise constrictions.

Adding the `maxpause=30s` parameter to the search in the recipe tells the `transaction` command that there must be no pause between events greater than 30 seconds, otherwise the grouping breaks. By default, there is no limit.

```
index=main sourcetype=access_combined | transaction  
JSESSIONID maxpause=30s | stats avg(duration) AS  
Avg_Session_Time
```

Adding the `maxspan=30m` parameter to the search in the recipe tells the `transaction` command that when building the transaction, the first and last events cannot be greater than 30 minutes, otherwise the grouping breaks. By default, there is no limit.

```
index=main sourcetype=access_combined | transaction  
JSESSIONID maxspan=30m | stats avg(duration) AS  
Avg_Session_Time
```

Adding the `maxevents=300` parameter to the search in the recipe tells the `transaction` command that when building the transaction, the total number of events contained within cannot be greater than 300, otherwise the grouping breaks. By default, the value is 1,000.

```
index=main sourcetype=access_combined | transaction  
JSESSIONID maxevents=300 | stats avg(duration) AS  
Avg_Session_Time
```

As mentioned, all these parameters can be combined to create an even more constrained transaction for specific use cases. Here is an example of a transaction that starts with a home page request, ends with a checkout, is no longer than 30 minutes, has no events where there is a pause greater than 30 seconds, and the maximum number of events contained within is 300:

```
index=main sourcetype=access_combined | transaction  
JSESSIONID startswith="GET /home" endswith="checkout"  
maxpause=30s maxspan=30m maxevents=300 | stats avg(duration)  
AS Avg_Session_Time
```

NOTE

For more information on the transaction command, visit <http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Transaction>.

Calculating the average execution time for multi-tier web requests

With components existing at many different layers to provide varying functionalities, web applications are no longer as straightforward as they once were. Understanding the execution time for a web request across the entire application stack, rather than at a single layer, can be extremely beneficial in correctly articulating the average time that requests take to execute in their entirety. This can lead to identification of issues in relation to increasing website response times.

In this recipe, you will write a Splunk search to calculate the average execution time of a web request that traverses not only the website access logs but also the application logs. You will then graphically display this value on a dashboard using the single value visualization.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Chapter 1, Play Time – Getting Data In](#). You should have also completed the recipes in the previous chapters and be familiar with navigating the Splunk user interface.

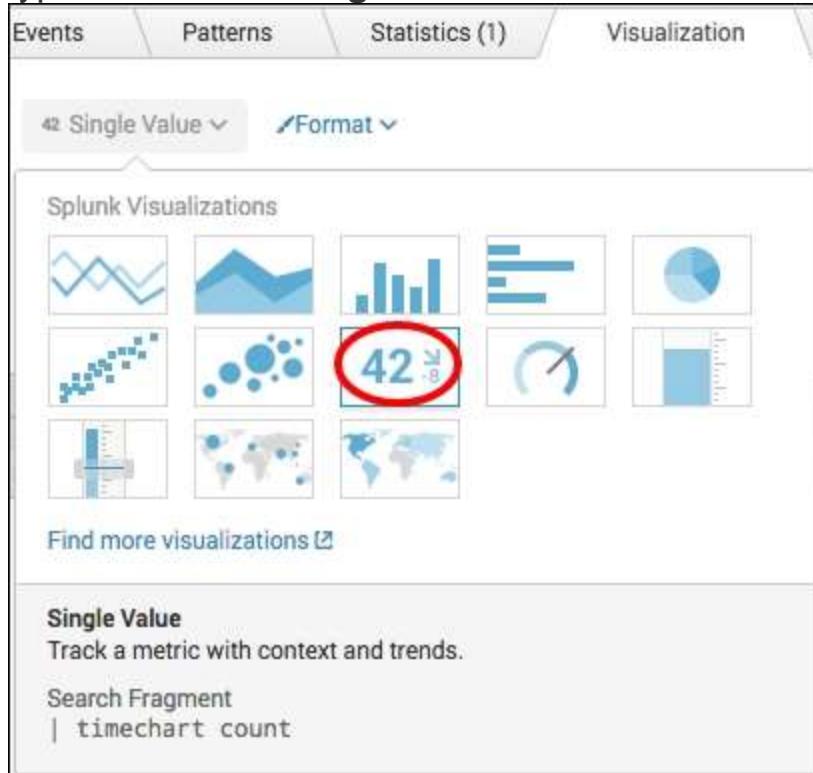
How to do it...

Follow the steps in this recipe to calculate the average execution time for multi-tier web requests:

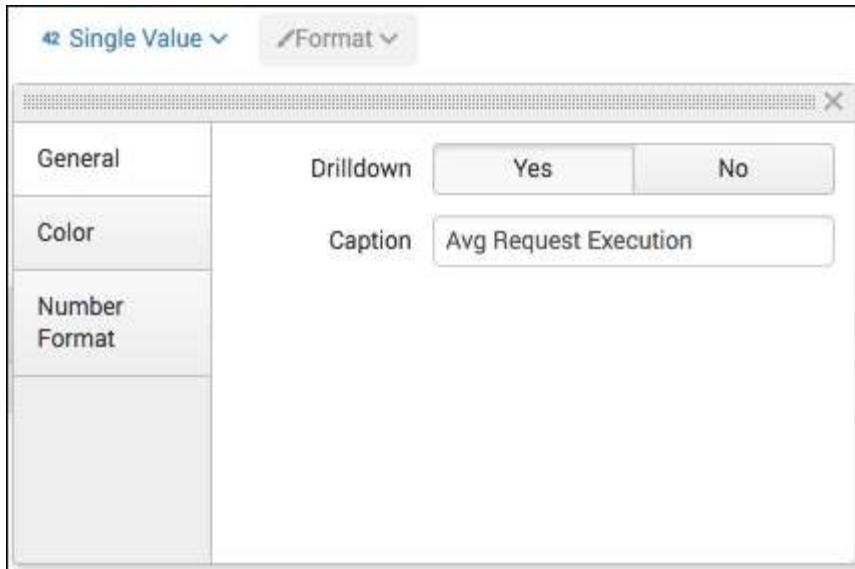
1. Log in to your Splunk server.
2. Select the **Operational Intelligence** application.
3. Ensure the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on the search button or hit *Enter*.

```
index=main sourcetype=access_combined | join JSESSIONID  
usetime=true earlier=false [ search index=main sourcetype=log4j  
| transaction threadId maxspan=5m | eval JSESSIONID=sessionId ]  
| stats avg(duration) AS Avg_Request_Execution_Time
```

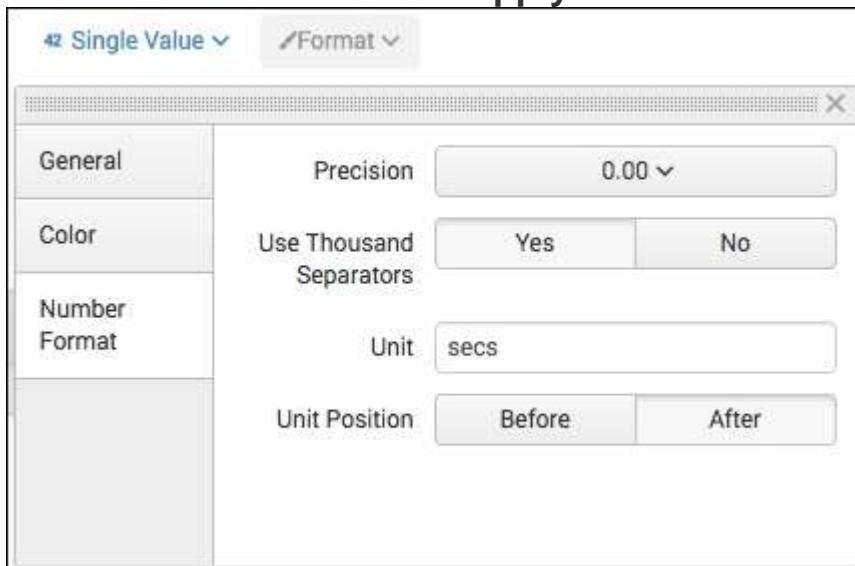
- After a little while, Splunk will return a single value representing the average execution time in seconds for a complete web request on the website.
- Click on the **Visualization** tab.
- Since there are a number of visualizations within Splunk, the single value visualization might not be displayed by default within the **Visualization** tab. Click on the dropdown listing the visualization types and select **Single Value**.



- You should now see the value represented as single value visualization.
- Let's add more context to the visualization. Click on **Format**. On the **General** tab, enter `Avg Request Execution` in **Caption**.



9. Click on the **Number Format** tab and select **0.00** for **Precision** and enter **secs** for **Unit**. Click on **Apply** to save the formatting.



10. Your single value visualization should now look similar to the following example:



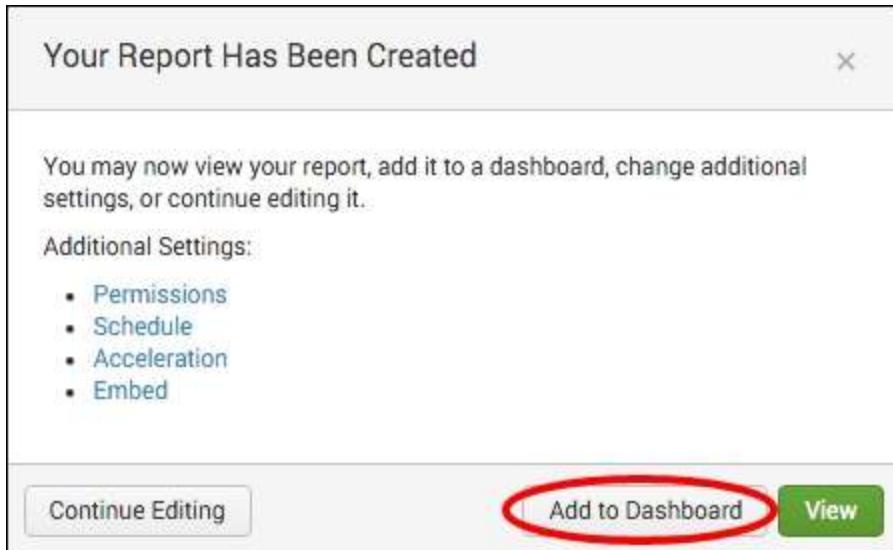
11. Let's save this search as a report. Click on **Save As** and choose **Report** from the drop-down menu.



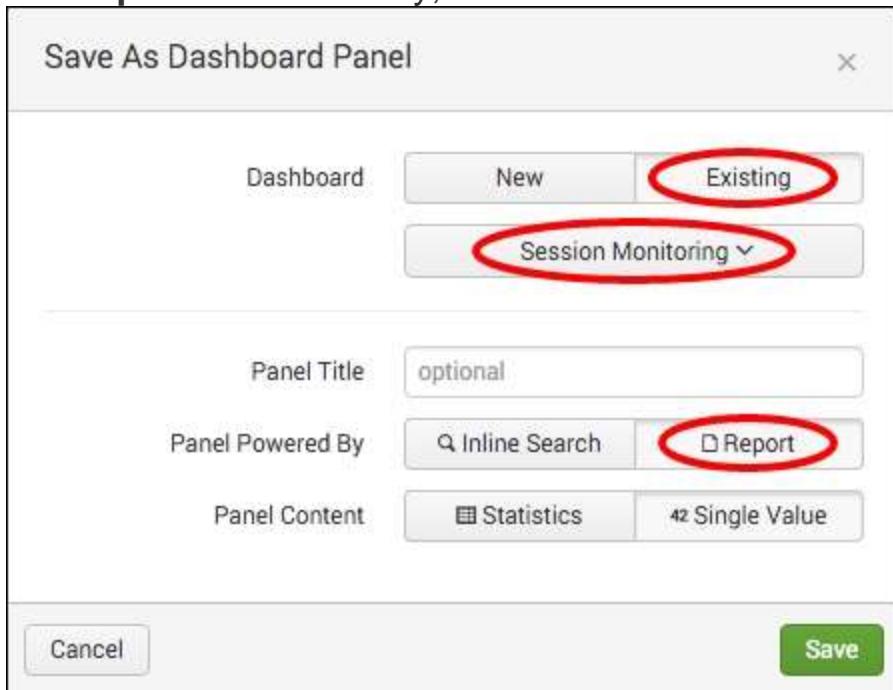
12. In the **Save As Report** window that appears, enter `cp06_average_request_execution_time` as the title and click on **Save**.

Save As Report	
Title	<code>cp06_average_request_execution_time</code>
Description	optional
Content	42 + <input type="button" value="42"/> <input type="button" value="42"/>
Time Range Picker	Yes <input type="button" value="No"/>
<input type="button" value="Cancel"/> <input style="background-color: green; color: white; border-radius: 50%;" type="button" value="Save"/>	

13. You will receive a confirmation that your report has been created. Now, let's add this report to a dashboard. In the next window, click on **Add to Dashboard**.

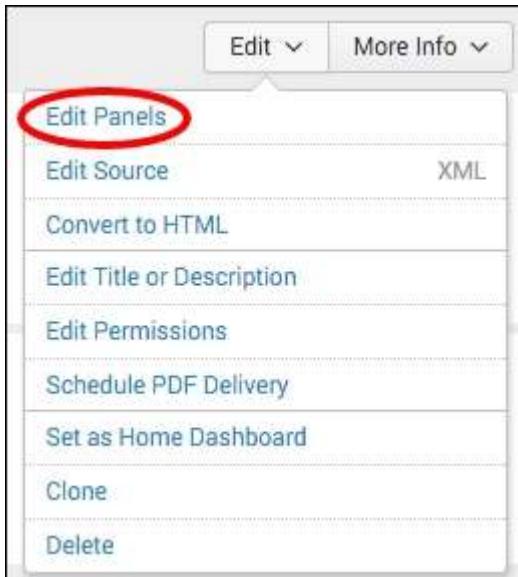


14. You will now add this to the dashboard that was created in the previous recipe named **Session Monitoring**. In the **Save As Dashboard Panel** window, click on the **Existing** button beside the **Dashboard** label. From the drop-down menu that appears, select **Session Monitoring**. For the **Panel Powered By** field, click on **Report**. Finally, click on **Save**.

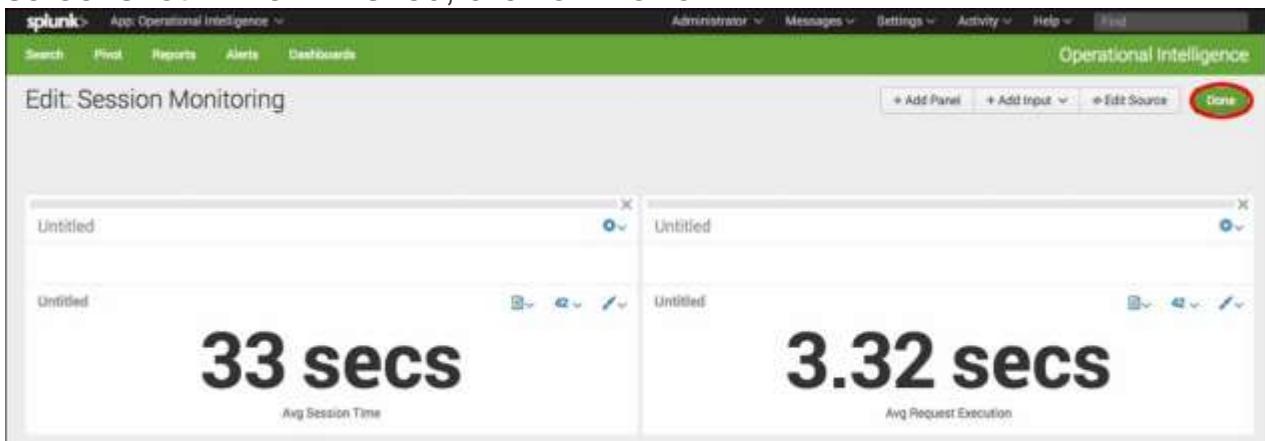


15. Click on **View Dashboard** to see the panel that's been added to your **Session Monitoring** dashboard.

16. Now, let's arrange the panels so they are side by side. Click on **Edit** and choose **Edit Panels** from the drop-down menu.



17. Now, drag the newly added panel so that both single value visualizations are on the same line, as shown in the following screenshot. When finished, click on **Done**.



How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should be familiar with this search from the previous chapters. It is used to return events from the access log.

Search fragment	Description
<pre> join JSESSIONID usetime=true earlier=false [search index=main sourcetype=log4j transaction threadId maxspan=5m eval JSESSIONID=sessionId]</pre>	<p>Using the <code>join</code> command, we execute a subsearch matching events from the web application log. The <code>JSESSIONID</code> field is used as the unique value for events on. Within the subsearch, we leverage the <code>transaction</code> command to group all the application events together by their <code>threadId</code>, which is a unique value for each execution. The <code>maxspan</code> parameter is used with the <code>transaction</code> command to ensure that the application events common to the transaction happen within five minutes of each other. Then, we create a new field named <code>JSESSIONID</code> using <code>eval</code> because it does not have a field named <code>sessionId</code> for web application events, which has a field named <code>sessionId</code> instead. By creating this field, the search can properly associate the events. The <code>usetime</code> and <code>earlier</code> parameters passed to the <code>join</code> command tell it to limit the matches to events that come after the originating web access log event. This ensures that only those web application events that happened after the website access log event will be returned. We know the natural method of execution for our application requires a user interaction with the website before the application triggers a function execution.</p>
<pre> stats avg(duration) AS Avg_Request_Execution_Time</pre>	<p>Using the <code>stats</code> command, we calculate the average of the duration field, since this field has been carried over from the subsearch. Using the <code>transaction</code> command within the subsearch ensures that only those web application events that happened after the website access log event will be returned. Using the <code>AS</code> operator, we rename the resulting field created with the given value to something more meaningful, for example, <code>Avg_Request_Execution_Time</code>.</p>

There's more...

In this recipe, you used the `join` command to join an inner subsearch with an outer main search. This is similar to a join in an SQL database. Another command that is similar to `join` is `append`. The `append` command allows you to string two different searches together, such that the results of the second search are appended to the results of the first search. The maximum value is obtained from append if the searches you append together share common fields; use of the `eval` command or implementation of CIM can help with this.

NOTE

For more information on `join`,
visit <http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Join>.

For more information on the `append` command,
visit <http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Append>.

While both the `join` and `append` commands can be useful, they are not the most efficient commands. This is because both the commands execute multiple searches instead of just one. Often, the `stats` or `transaction` command can be used in creative ways to avoid using `join` or `append`, and to increase search performance as a result.

CALCULATING THE AVERAGE EXECUTION TIME WITHOUT USING A JOIN

Often, there are many ways to write a search that results in providing the same or similar insight. While there is nothing wrong with the search used in this recipe, we can amend the search so that it does not use the `join` command. An example search might be as follows:

```
index=main sourcetype=access_combined OR sourcetype=log4j
| eval action=substr(uri_path,2) | eval
action=lower(if(isnull(action),requestType,action))
| eval
JSESSIONID;if(isnull(JSESSIONID),sessionId,JSESSIONID)
| transaction threadId, JSESSIONID, action maxspan=1m
| stats avg(duration) AS Avg_Request_Execution_Time
```

Here, we search both the web access and application logs in the same search. We evaluated a new field called `action` using similar field values found in the web access (`uri_path`) and application logs (`requestType`). For example, a checkout web request generates a checkout application request. Using the `transaction` command, we transact all the events across both sourcetypes that share a session ID, thread ID, or our new `action` field. We also make the assumption that our requests do not take longer than a minute to execute, and subsequently, we set a `maxspan` of one minute. Setting this tightened criteria makes the `transaction` command more efficient. Splunk will now group all the web requests and subsequent

application events related to the Web requests together into transactions, with duration calculated for each. We then apply the same stats command to work out the average request execution time. This might actually provide a more accurate execution time as we incorporate the timestamp of the web access logs into the transaction duration.

Displaying the maximum concurrent checkouts

Typically, when analyzing web requests, events often overlap with one another due to multiple users issuing requests concurrently. By identifying these overlapping requests, and further understanding the concurrency of events, you will gain a clearer picture of the true demand for both the resources and consumer demands.

In this recipe, you will write a Splunk search to find the number of concurrent checkouts over a given period of time. You will then graphically display this value on a dashboard using the line chart visualization.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Chapter 1, Play Time – Getting Data In](#). You should have also completed the earlier recipes in this chapter and be familiar with navigating the Splunk user interface.

How to do it...

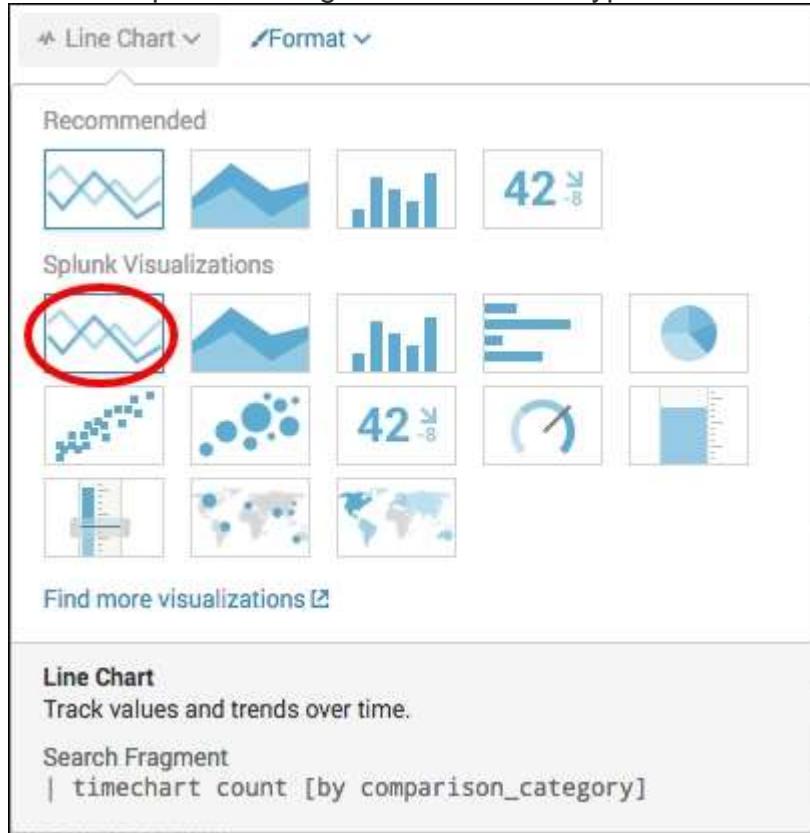
Follow the steps in this recipe to identify the number of concurrent checkouts over a given period of time:

1. Log in to your Splunk server.
2. Select the **Operational Intelligence** application.
3. Ensure the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on the search button or hit *Enter*.

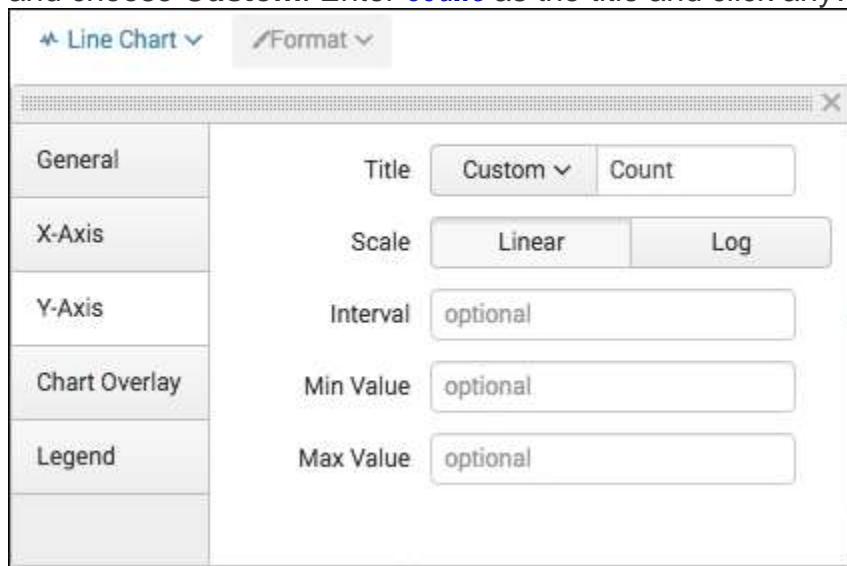
```
index=main sourcetype=access_combined | transaction JSESSIONID  
startswith="GET /home" endswith="checkout" | concurrency  
duration=duration | timechart max(concurrency) AS "Concurrent  
Checkouts"
```

4. After a short while, Splunk will return the values associated with the maximum concurrent checkouts split in 30-minute durations.
5. Click on the **Visualization** tab.

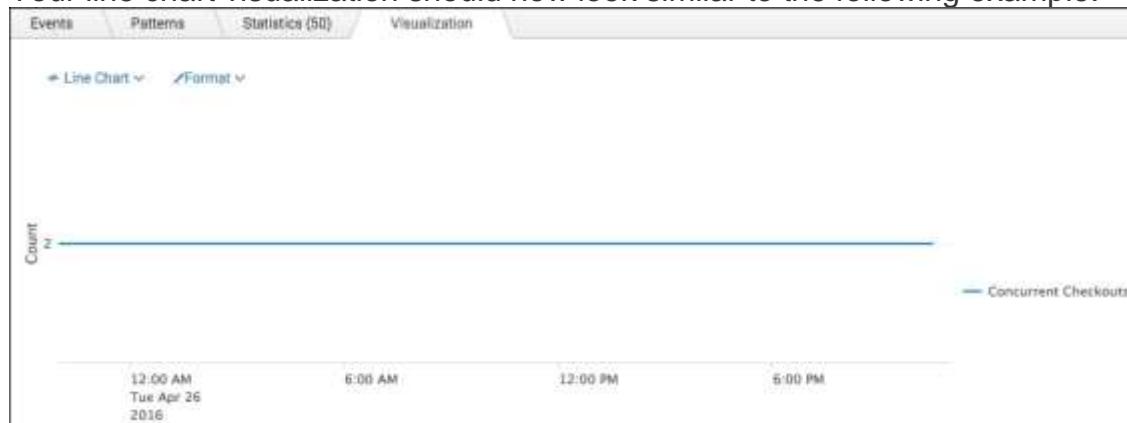
6. Since there are a number of visualizations within Splunk, the line chart visualization might not be displayed by default within the **Visualization** tab. Click on the dropdown listing the visualization types and select **Line**.



7. You should now see the value represented as line chart visualization.
8. Let's add more context to the visualization and correct some values. Click on **Format** and then click on the **Y-Axis** tab. Click on the dropdown **Title** menu and choose **Custom**. Enter **Count** as the title and click anywhere on the page.



Your line chart visualization should now look similar to the following example:



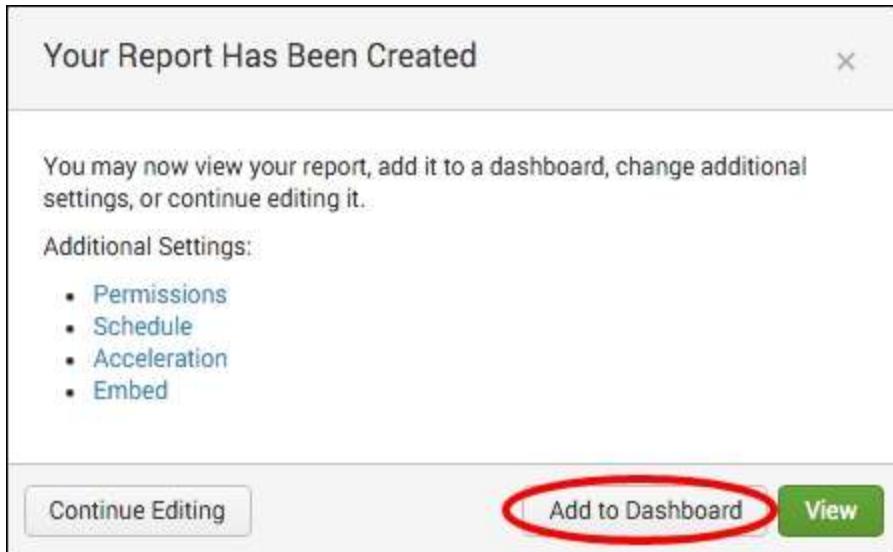
9. Let's save this search as a report. Click on **Save As** and choose **Report** from the drop-down menu.



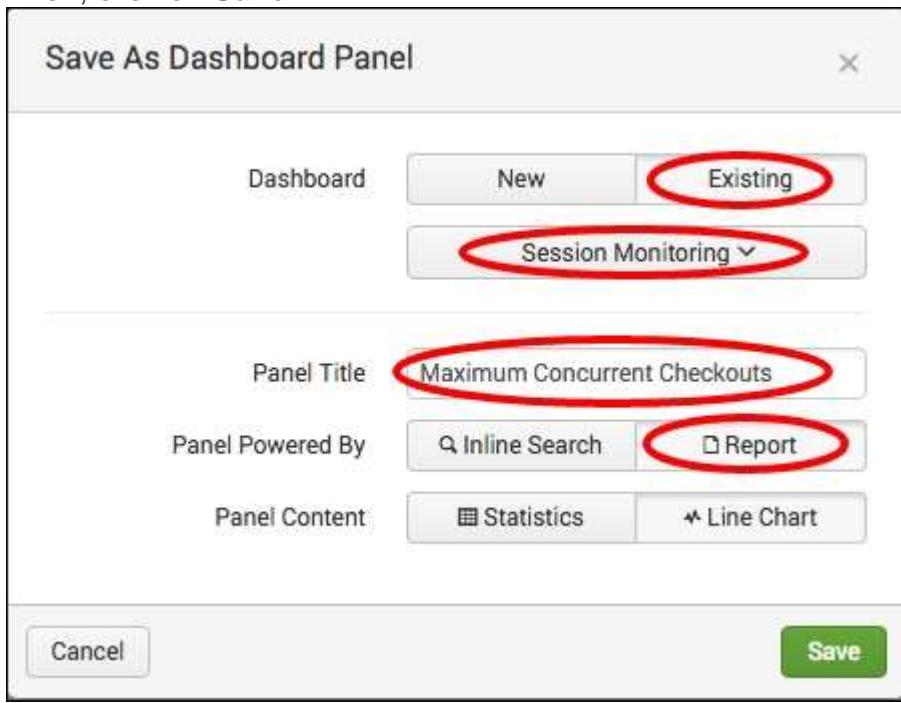
10. In the pop-up box that appears, enter `cp06_concurrent_checkouts` in the **Title** field and then click on **Save**.

A screenshot of a 'Save As Report' dialog box. It has fields for 'Title' (containing 'cp06_concurrent_checkouts'), 'Description' (containing 'optional'), 'Content' (with three icons), and 'Time Range Picker' (with 'Yes' and 'No' buttons). At the bottom are 'Cancel' and 'Save' buttons, with 'Save' circled in red.

11. You will receive a confirmation that your report has been created. Now, let's add this report to a dashboard. In the next window, click on **Add to Dashboard**.



12. You add this report to the **Session Monitoring** dashboard that was created in an earlier recipe. In the **Save As Dashboard Panel** pop-up box, click on the **Existing** button beside the **Dashboard** label. From the drop-down menu that appears, select **Session Monitoring**. Enter **Maximum Concurrent Checkouts** in the **Panel Title** field and ensure the panel is powered by **Report**. Then, click on **Save**.



13. You can now click on **View Dashboard** to see the panel on your **Session Monitoring** dashboard.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should be familiar with this search from the earlier recipes in this chapter. It is used to return events from the website access log.
<code> transaction JSESSIONID startswith="GET /home" endswith="checkout"</code>	Using the <code>transaction</code> command, we group events together, based on their given <code>JSESSIONID</code> , to form a single transaction and apply transaction parameters so that events start with a <code>GET</code> request for the main page and end with checkout.
<code> concurrency duration=duration</code>	The <code>concurrency</code> command is used to find the concurrent number of events, given a duration value, which occurred at the same start time. The duration field used here is generated by the use of the <code>transaction</code> command. A field named <code>concurrency</code> will be created by the <code>concurrency</code> command, and this will store the value of concurrent events.
<code> timechart max(concurrency) AS "Concurrent Checkouts"</code>	The <code>timechart</code> command is leveraged to plot the maximum values of the <code>concurrency</code> field over the given period of time. The <code>AS</code> operator is leveraged to rename the field to a more readable value; for example, <code>Concurrent Checkouts</code> .

The `concurrency` command is a useful way of calculating concurrent events without using too much logic. In this recipe, you were able to use the command to identify the maximum amount of concurrent checkouts throughout the day.

NOTE

For more information on the `concurrency` command, visit <http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Concurrency>.

Session 4 - Managing Indexes and Users – The Experience

Session 4- Dashboards and Visualizations – Making Data Shine

In this chapter, we will learn how to build dashboards and create visualizations of your data. We will cover the following recipes:

- Creating an **Operational Intelligence** dashboard
- Using a pie chart to show the most accessed web pages
- Displaying the unique number of visitors
- Using a gauge to display the number of errors
- Charting the number of method requests by type and host
- Creating a timechart of method requests, views, and response times
- Using a scatter chart to identify discrete requests by size and response time
- Creating an area chart of the application's functional statistics
- Using a bar chart to show the average amount spent by category
- Creating a line chart of item views and purchases over time

Introduction

In the previous chapter, we learned all about Splunk's SPL and how it can be leveraged to search and report your data. In this chapter, we're going to build on this knowledge and use some of Splunk's visualization capabilities to make our data shine! You will learn how to create a dashboard through the Splunk UI and proceed to add to it the reports that were built in the previous chapter. Two more dashboards will then be created as a result of the remaining recipes.

Visualizations are a cornerstone for proper data presentation. By visualizing data in a manner that we as humans are accustomed to, you enable the user to better relate to what is being presented and have a proper understanding of how to react. When using Splunk for Operational Intelligence, you will be hard pressed to find a report that is

not visually represented in some fashion. Everyone from front-line staff to C-level executives looks to Splunk's visualizations to make better sense of the data that their systems and applications produce. Through the creation and use of dashboards, these visualizations can then be arranged and centralized to meet the needs of your organization.

About Splunk dashboards

A dashboard represents the most common type of view within Splunk and provides the means to bring together one or more reports and display them on a single page. Each report is placed on a dashboard as a panel and powered by the search you created. Typically, the panels get populated with data once the dashboard is loaded. A report within a panel can display tabular data or one of the number of visualizations we will cover in this chapter.

Using dashboards for Operational Intelligence

In the world of Operational Intelligence, dashboards are one of the key tools to unlock pivotal information and provide a holistic view of systems and applications through a single pane. Dashboards are built to collectively display information to key audiences such as operators, administrators, or executives. They act as a window to how your environment is performing and allow you to obtain the right information at the right time in one place, in order to make timely and actionable decisions.

Enriching data with visualizations

Data on its own can be hard for us, as humans, to make sense of easily and can be extremely tedious to analyze. Visualizations provide a powerful way to bring data to life. Presenting data in a visual context enables those viewing it to better understand the relationship one value has to another, identify patterns, build correlations between datasets, and plot out trends. Colors that we easily relate to can be applied to visualizations in order to direct attention and highlight specific data points. For example, a value within an acceptable range might be colored green, but when this value increases, it might change to yellow and eventually to red when it's in an unacceptable range. Humans

associate red with bad and green with good; therefore, a red value nicely conveys the need for attention to itself.

Let's now apply this to an Operational Intelligence example. Imagine that you have a distributed environment of web servers that are generating large amounts of erratic data. Inside each of these events is a field that represents the response time of when that event occurred. If you were left having to analyze these events row by row in a table, it could take a very long time to find the events with values outside of the norm. Using visualizations such as a scatter chart, you could plot your event data and easily be able to identify these discrete events that lie outside of the primary cluster of events.

Available visualizations

One of the great benefits of Splunk is that there are numerous out-of-the-box visualizations that can be easily overlaid on your data. The type of visualizations and their common usage is outlined in the following table:

Visualization	Common usage
Line chart	This is commonly used to display data over time. If more than one data series is specified, each line on the chart will have a different color. Line charts can be stacked to help understand the relation of a given series of data to the rest of the plotted series.
Area chart	This works in the same way as a line chart, but the area below the line is shaded in color to emphasize quantities.
Column chart	This displays the data values as vertical columns and is most commonly used when the frequency of values needs to be compared. Column charts can also be stacked to highlight the importance of different data types within the chart.
Bar chart	This displays the data values as horizontal bars and works in the same way as column charts, but with its axis reversed.
Pie chart	This is two-dimensional and displays the data values as segments of a pie. It is most commonly used to highlight or compare the proportion of numerical values.

Visualization	Common usage
Scatter chart	This displays the data values as a series of plotted squares. It is commonly used when trying to identify discrete values in data that fall within the confines of regular events.
Single value	This displays the data as a single value and is mostly used to display a sum or total, for example, the total number of errors in the last hour.
Radial gauge	This resembles a speedometer with a needle to signify the current value across an arced range. It is most commonly used on real-time dashboards to draw attention to the current state of a given metric. Thresholds can be defined to signify which values are acceptable (green), escalating (yellow), and severe (red).
Filler gauge	This resembles a thermometer with a liquid-like indicator. As the value changes, the volume of the liquid rises and falls, as well as changing color. As with the radial gauge, it is most commonly found on real-time dashboards and can have custom thresholds defined.
Marker gauge	Similar to the filler gauge, a marker gauge is already filled with values as defined by the thresholds, and it has a sliding marker to signify what the current value is. It is found most commonly on real-time dashboards.
Marker Map	This is commonly used to illustrate geographical distributions of the data. Data points on the map can be charted to highlight distinct counts of values within the same geographic region.
Choropleth Map	Choropleth maps use shading or coloring of specific bounded areas to illustrate differences in the values between different areas on the map. For example, a map of the US may have states shaded or colored differently, in order to represent differing values per state.
Heat map	Tabular values can have a heat map overlay applied to them so that the highest values are shaded red while the lowest values are shaded blue. It is most commonly used when

Visualization	Common usage
	trying to draw attention visually to the variation of values in a table.
Sparkline	Sparkline visualizations are like mini line charts and are applied within a table to each row. They provide insight into the identification of patterns that might not otherwise have been properly represented by the resulting data in the table.

TIP

For more information on the available visualizations, visit <http://docs.splunk.com/Documentation/Splunk/latest/Viz/Visualizationreference>.

You can also checkout the Splunk Web Framework Toolkit app on Splunkbase for examples (<https://splunkbase.splunk.com/app/1613>).

Best practices for visualizations

Here are some best practices to consider when adding visualizations to your dashboards:

- Use visualizations to provide insight in a way that cannot be easily represented by tabular data.
- It can sometimes be useful to have a chart supported with a table, as a table can make finding absolute numbers easier.
- Provide enough information, but not too much. Do not overload the charts with data that is not pertinent to achieving the goal of the visualization.
- Do not overload the dashboards with visualizations. Spread visualizations out among a few dashboards rather than overloading one specific dashboard. This makes things easier for your users to view as well as improves performance.
- Stacked charts are your friend, especially with area charts. If not stacked, area charts can have instances where large values dominate the chart, obscuring other values.
- Scale visualizations properly. Know when it is best to use linear versus log.

- Label your visualizations clearly so that the audience can understand what they are looking at and do not have to assume.
- Use appropriate visualizations for the task at hand. Here is some guidance:
 - **Comparisons over time:** Use line and column charts
 - **Comparisons among items:** Use bar and column charts
 - **Relationships:** Use scatter charts
 - **Distribution:** Use column or bar charts sorted or a scatter chart
 - **Static composition:** Use column charts stacked at 100 percent or a pie chart
 - **Changing composition:** Use column or area charts stacked, or column or area charts stacked at 100 percent
 - **Geographic statistics:** Use marker or choropleth maps
- Make proper use of colors and thresholds when leveraging single value visualizations and gauge charts.
- You can change the orientation of most visualizations; make use of your best judgment as required.

Creating an Operational Intelligence dashboard

Before this chapter gets into everything that is great about visualizations, it is best to first cover the process of creating a dashboard. In this recipe, you will create a dashboard from scratch using the Splunk Web UI that we will then use for other recipes in this chapter.

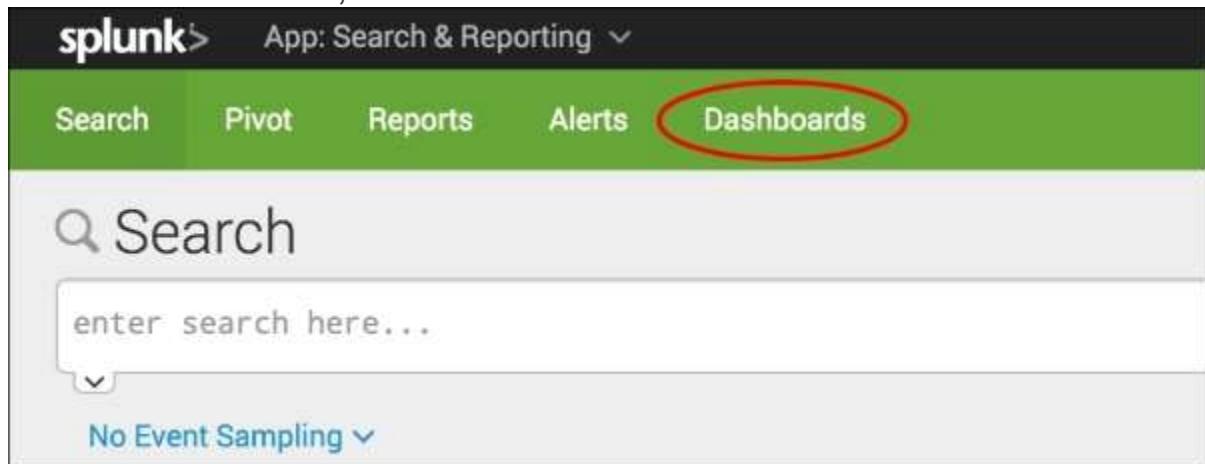
Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#), and all the completed recipes from [Session 3-2, Diving into Data – Search and Report](#). You should be familiar with navigating the Splunk user interface.

How to do it...

Follow the given steps to create an Operational Intelligence dashboard:

1. Log in to your Splunk server.
2. Select the default **Search & Reporting** application.
3. From the menu bar, click on the **Dashboards** link:



4. On the **Dashboards** screen, click on the **Create New Dashboard** button:



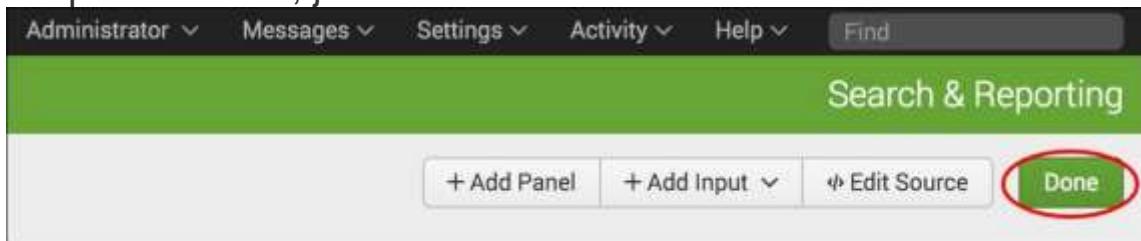
5. The **Create New Dashboard** screen will pop up. Enter [Website Monitoring](#) in the **Title** field. The **ID** field will be automatically populated; leave it as it is. The **Description** field can be left blank for now, but ensure that the **Shared in App** permission is selected. Finally, click on **Create Dashboard**:

A screenshot of a "Create New Dashboard" dialog box. It contains the following fields:

- Title:** Website Monitoring (circled with a red oval)
- ID:** website_monitoring
- Description:** optional
- Permissions:** Shared in App (circled with a red oval)

At the bottom are two buttons: "Cancel" and "Create Dashboard".

6. The newly created **Website Monitoring** dashboard will appear in edit mode. We will add panels to the dashboard in the next few recipes. For now, just click on the **Done** button:



You have now created a blank dashboard ready to be populated with reports and visualizations.

How it works...

When creating a dashboard through the user interface, Splunk builds the underlying dashboard object code in simple XML for you behind the scenes. Following this, the dashboard object will then be used as a container for the reports you add to it. You can always view the source code of the dashboard by clicking on the **Edit** button and then, from the drop-down menu, you can click on **Edit Source**. The simple XML source code for the dashboard will be displayed in the editor. Dashboards and simple XML will be covered in more detail in the next chapter.

TIP

There are several ways in which dashboards can be created in the Splunk user interface. In this recipe, we essentially created an empty dashboard which is ready to be filled with visualized reports. Splunk also allows dashboards to be created at the time of adding reports, as you will see later in this chapter.

There's more...

When creating a dashboard, the default permission is for it to be private (only accessible by the user who created it). You might wish to share this or other dashboards with other users or groups who have an interest in these reports.

CHANGING DASHBOARD PERMISSIONS

To change the permissions on a dashboard, you must first return to the **Dashboards** screen. This can be accomplished by clicking on the **Dashboards** menu, as outlined in step 3 of this recipe. Within the **Dashboards** screen, you will see the **Website Monitoring** dashboard that you created during the course of this recipe. Under the **Actions** column, you will see a clickable link labeled **Edit**; click on this link. A drop-down panel will appear; click on the item labeled **Edit Permissions**. The resulting pop-up window that appears will allow you to define permissions on a role-by-role basis and limit these permissions to be restrictive within the working application, or globally for all applications.

TIP

When creating dashboards through the GUI in Splunk, the default permission level should be private. However, in this recipe, you selected the **Shared in App** permissions button when creating the dashboard. This ensures that the new dashboard is automatically shared with everyone who has permissions to the application and saves you from having to edit permissions after creating it. On the backend, dashboards with application permissions are stored in the respective application directory structure. Dashboards with private permissions are stored in the respective user directory.

Using a pie chart to show the most accessed web pages

The sample data loaded in [Session 3-1, Play Time – Getting Data In](#), provides a wealth of information on how customers are interacting with our online shopping website. In the *Finding the most accessed web pages* recipe in [Session 3-2, Diving into Data – Search and Report](#), we saw how to find the most accessed web pages. The output of that recipe was displayed in a tabular format that could be hard for the viewer to grasp the proportional differences between web page access amounts. We will now take a look at how to use pie charts. By taking the same data and visually presenting it using a pie chart now, we will enable the viewer to more easily identify the proportion of requests between the different web pages. Visual representation of data, even if the data is very simple, can lead to better decision making in times of need.

In this recipe, you will use the report named `cp02_most_accessed_webpages`, which you created in [Session 3-2, Diving into Data – Search and Report](#). You will graphically display the output of the report using a pie chart and add it to the **Website Monitoring** dashboard that we just created.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar, the time range picker, and the search tabs (**Events**, **Statistics**, and **Visualization**).

How to do it...

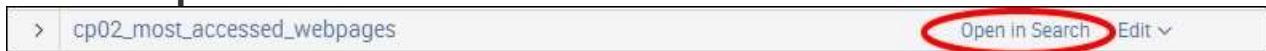
Follow the given steps to use a pie chart to show the most accessed web pages:

1. Log in to your Splunk server.
2. Select the default **Search & Reporting** application.

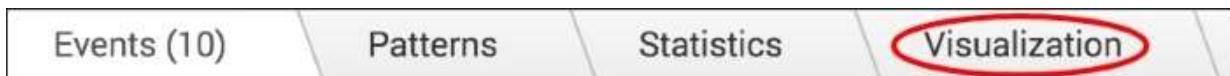
3. From the menu bar, click on the **Reports** link. This will display a list of all the reports we created and saved in Session 3-2, Diving into Data – Search and Report:



4. Locate the report line item named `cp02_most_accessed_webpages` and click on **Open in Search**:



5. Splunk will run the saved report with the search outlined in the following code. This will return a list of pages together with a count field that totals the number of times each page has been accessed.
6. On completion of the search, the results will be displayed within the **Statistics** tab. As we will be creating a pie chart, click on the **Visualization** tab:



7. As there are a number of visualizations within Splunk, the **Pie** visualization may not be displayed by default within the **Visualization** tab. Click on the dropdown to list the visualization types and then select **Pie Chart**:

Pie Chart Format

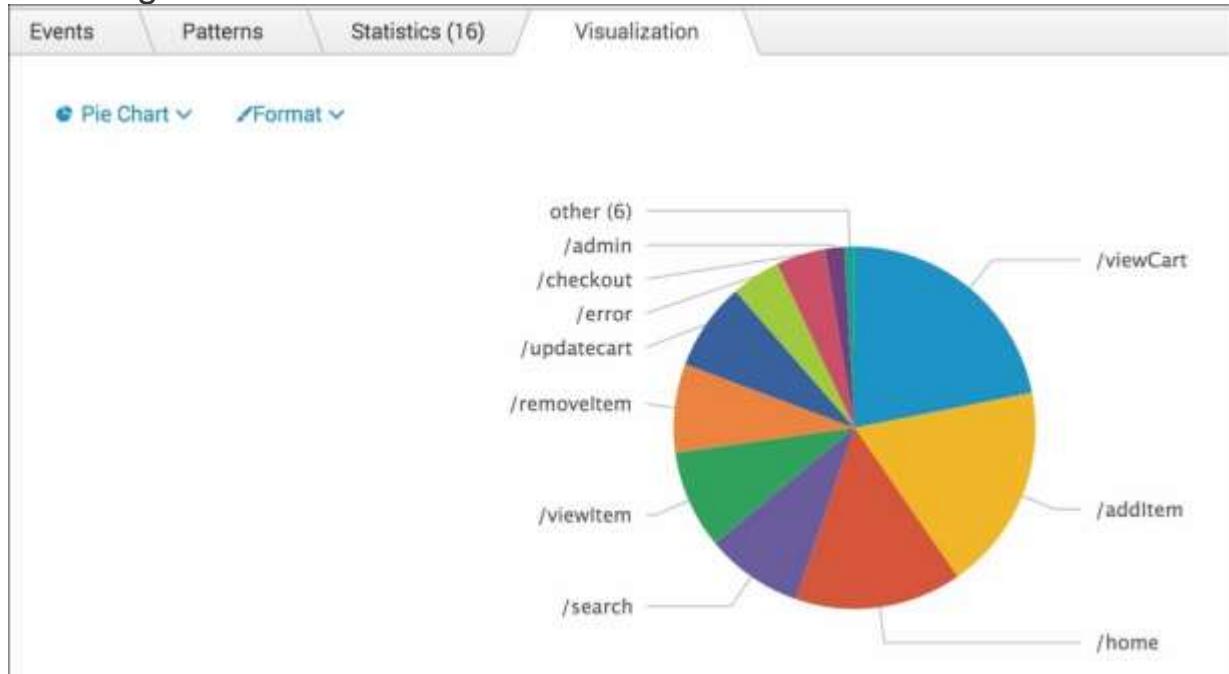
Splunk Visualizations

Find more visualizations ↗

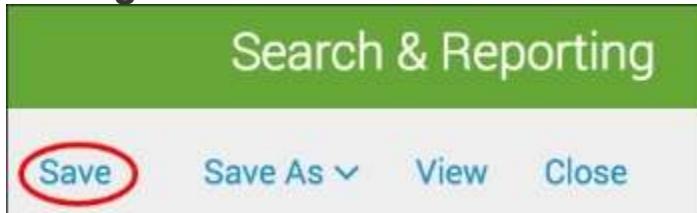
Pie Chart
Compare categories in a dataset.

Search Fragment
| stats count by comparison_category

- Now the data will be visualized as a pie chart, as shown in the following screenshot:



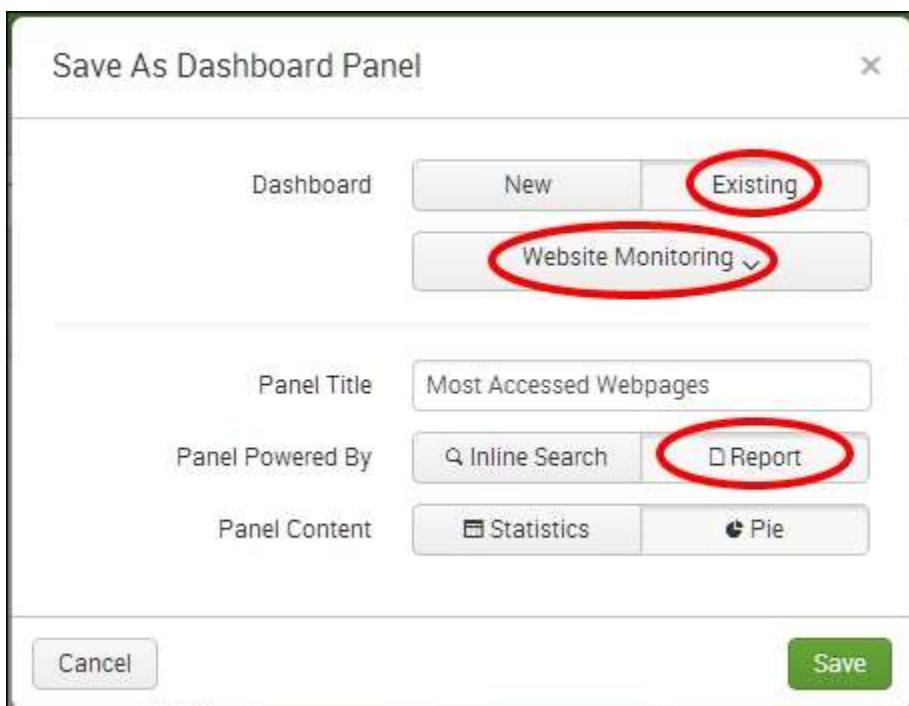
9. Next, save the updated report by clicking the **Save** link, and then click the **Save** button in the form that pops up. Then click **Continue Editing**:



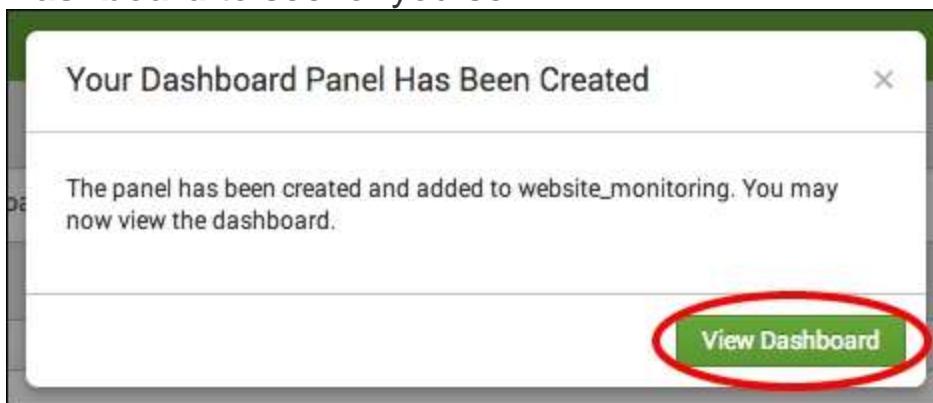
10. Let's add it to the **Website Monitoring** dashboard you created in the first recipe. Click on **Save As**, and then from the drop-down menu, click on **Dashboard Panel**:



11. The **Save As Dashboard Panel** screen will pop up. Select **Existing** to use an existing dashboard and select the **Website Monitoring** dashboard from the list. For **Panel Title**, enter `Most Accessed Webpages` and select **Report** for the **Panel Powered By** field. Then click on **Save**, as shown in the following screenshot:



12. The next screen will confirm that the dashboard has been created and the panel has been added. Click on **View Dashboard** to see for yourself:



How it works...

To review how the search works in detail, refer to the *Finding the most accessed web pages* recipe in [Session 3-2, Diving into Data – Search and Report](#).

The **Visualization** tab simply takes the tabular output, which is essentially a value split by another value, and overlays the given

visualization. In this case, it was a total count of events split by the web page name, which you overlaid with the pie chart visualization.

There's more...

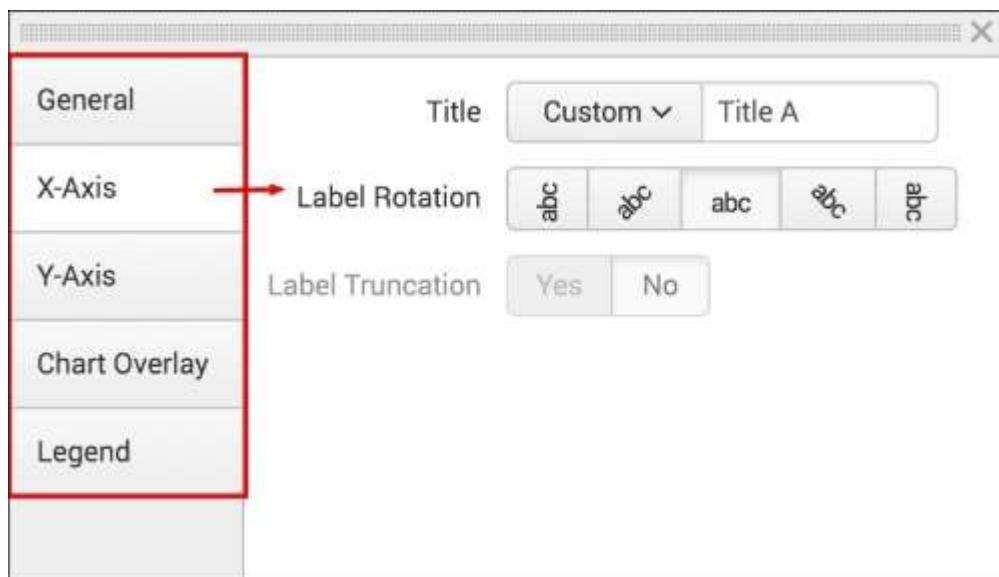
We can further build on the base search to provide different variations of the results and make use of other visualizations.

SEARCHING FOR THE TOP 10 ACCESSED WEB PAGES

If we modify the report search and replace the `stats` command with the `top` command, by default it will display the top 10 web pages:

```
index=main sourcetype=access_combined | top uri_path
```

Here, we modified the report search and replaced the `stats` command with the `top` command. By default, this will display the top 10 web pages. You can then select the **Visualization** tab and choose **Column** to see the results displayed as a column chart. Then, by clicking on **Format**, you can access a menu that allows you to extend the control over the chart by applying specific values, such as customizing the *x* and *y* axes, placement or removal of the legend, and more:



Displaying the unique number of visitors

It is always good to understand the number of page views and identify those that are accessed the most, but sometimes it is even better to understand how many of these page views are from unique visitors. Through the web access logs, we can get an understanding of how many unique visitors we have had to our website. For example, it could be helpful to understand whether times of high load are due to the true number of sessions on the website.

In this recipe, you will write a Splunk search to find the unique number of visitors to a website over a given period of time. You will then graphically display this value on a dashboard using the single value visualization.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar, the time range picker, and the **Visualization** tab. It is not required, but is advisable, that you complete all the recipes up until this point.

How to do it...

Follow the given steps to display the unique number of website visitors:

1. Log in to your Splunk server.
2. Select the default **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined | stats dc(JSESSIONID)
```

4. Splunk will return a single value that represents the distinct count (unique) of values in the field named `JSESSIONID`.
5. Click on the **Visualization** tab.

6. As there are a number of visualizations within Splunk, the **Single Value** visualization might not be displayed by default within the **Visualization** tab. Click on the dropdown that lists the visualization types and select **Single Value**:

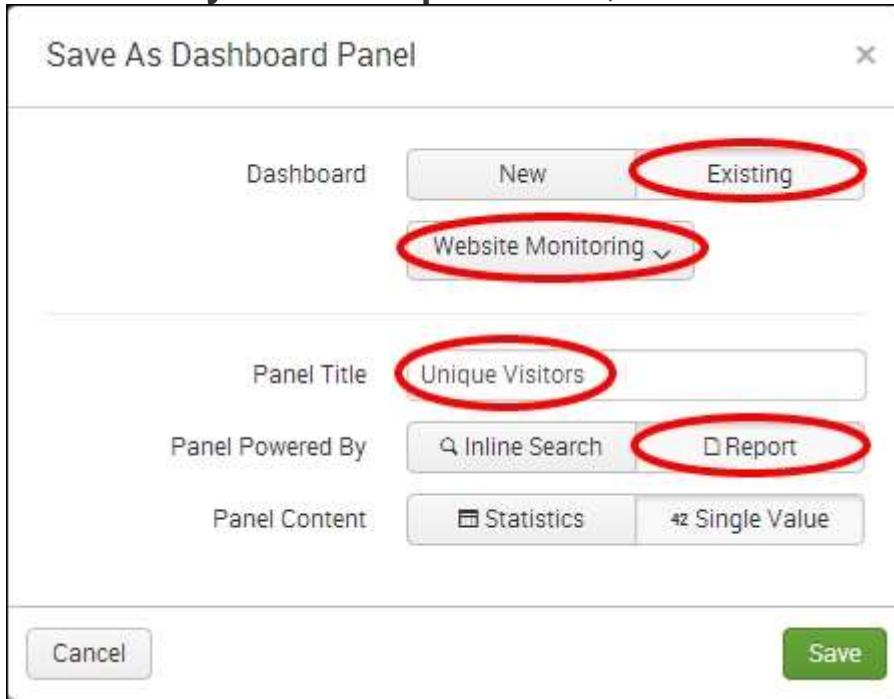
The screenshot shows the Splunk Visualization selector. At the top, there are two dropdown menus: '42 Single Value' and 'Format'. Below them is a grid of nine visualization icons. The third icon from the left in the second row, which is a 'Single Value' visualization showing the number '42' with a small chart below it, is circled in red. Below the grid is a link 'Find more visualizations'. Under the heading 'Single Value', there is a description: 'Track a metric with context and trends.' and a search fragment: 'Search Fragment | timechart count'.

7. Your data should now be visualized as a single value:

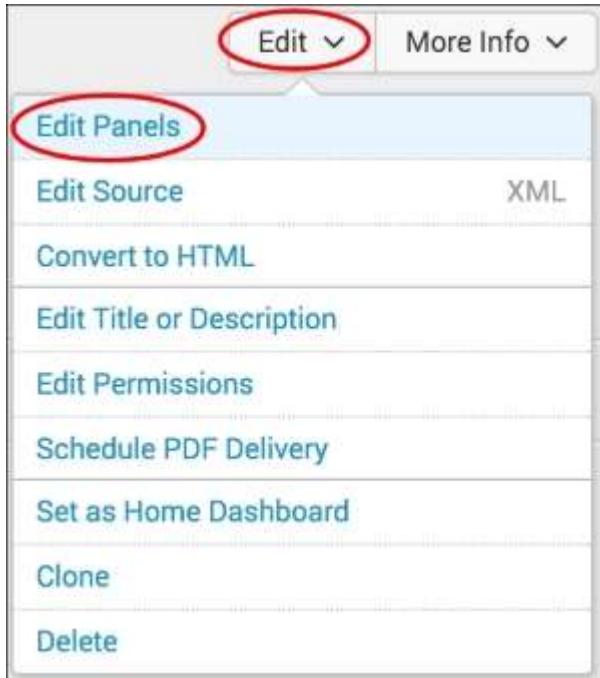
The screenshot shows the Splunk search results page. At the top, there are tabs for 'Events', 'Patterns', 'Statistics (1)', and 'Visualization'. The 'Visualization' tab is active. Below the tabs, there are two dropdown menus: '42 Single Value' and 'Format'. The main area displays a large, bold, black number '2,708'.

8. Save this search by clicking on **Save As** and then on **Report**. Name the report `cp03_unique_visitors` and click on **Save**. On the next screen, click on **Add to Dashboard**.
9. You will now add this to the **Website Monitoring** dashboard. Select the button labeled **Existing**, and from the drop-down menu that

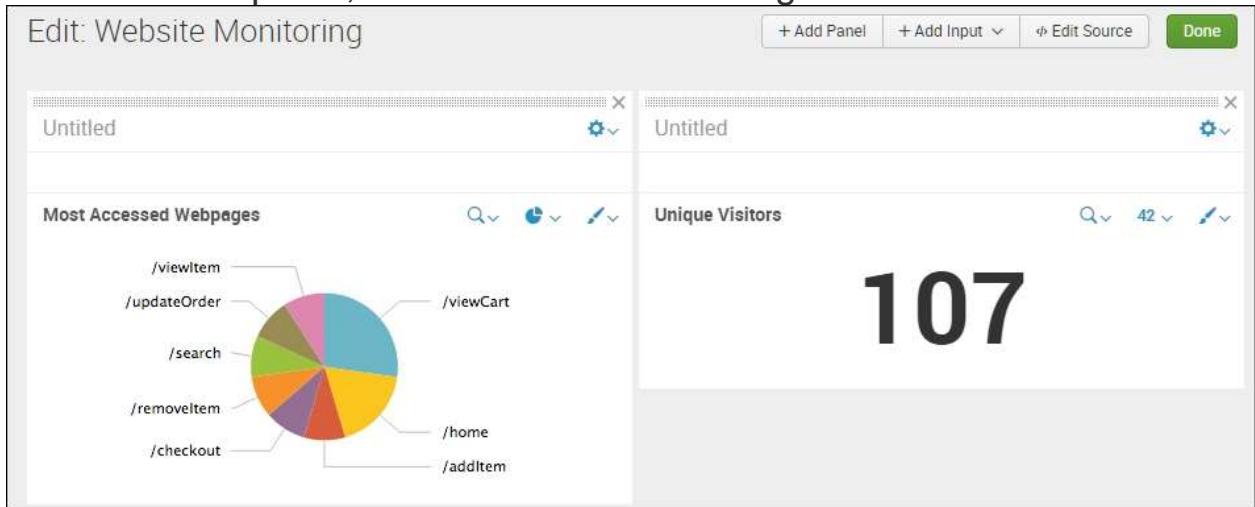
appears, select the **Website Monitoring** dashboard. For the **Panel Title** field value, enter `Unique Visitors` and select for the **Panel Powered By** field as **Report**. Then, click on **Save**:



10. The next screen will confirm that the dashboard has been created and the panel has been added. Click on **View Dashboard** to see for yourself. The single value visualization should be placed below the pie chart you created in the previous recipe.
11. You will now arrange the dashboard such that the **Pie Chart** panel and the **Single Value** panel are side by side. Click on the **Edit** button, and from the drop-down menu, select **Edit Panels**:



12. A gray bar will now appear at the top of your panel. Using this bar, click-and-drag the panel to be aligned onto the same row as the **Pie Chart** panel, as shown in the following screenshot:



13. Finally, click on **Done** to save the changes to your dashboard.

NOTE

You will learn more about the functions and features of the Dashboard Editor in the next chapter. For the purpose of this chapter, you will simply be moving panels around on a dashboard.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this book.
<code> stats dc(JSESSIONID)</code>	Using the <code>stats</code> command, you call the distinct count (<code>dc</code>) function to count the total number of unique values for the field named <code>JSESSIONID</code> . The <code>JSESSIONID</code> field is chosen because each visitor to the website is given a random session identifier whose value is stored in this field. The <code>clientip</code> field, for example, was not chosen here as you can have multiple users coming to a website from the same IP address through the use of NAT (short for Network Address Translation).

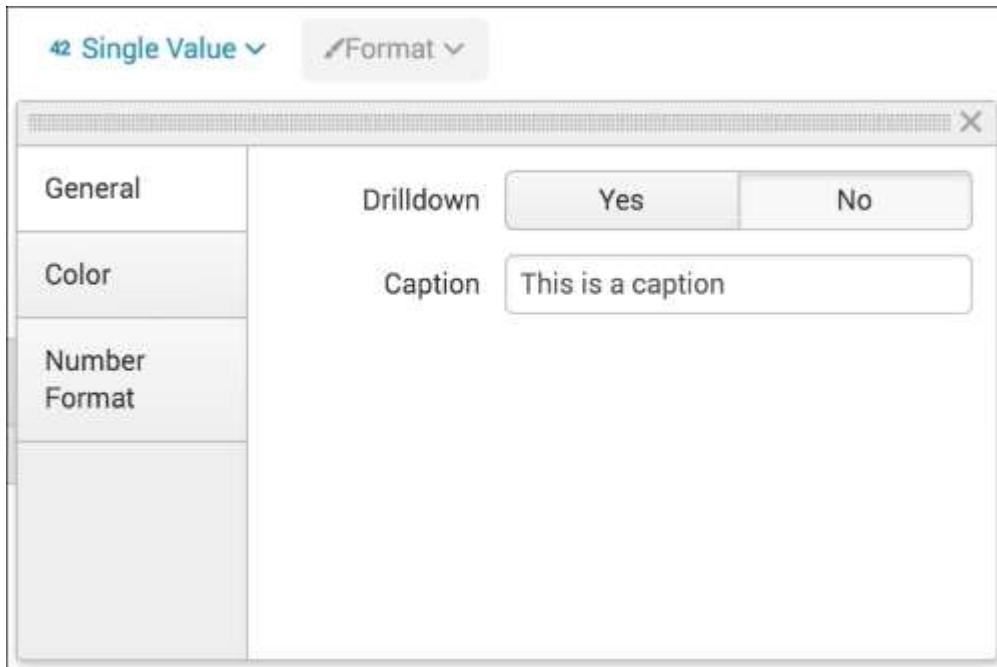
The **Visualization** tab simply takes the numeric output of the `stats` command and overlays the given visualization. In this case, you overlaid the single value visualization on a distinct count of visitor sessions.

There's more...

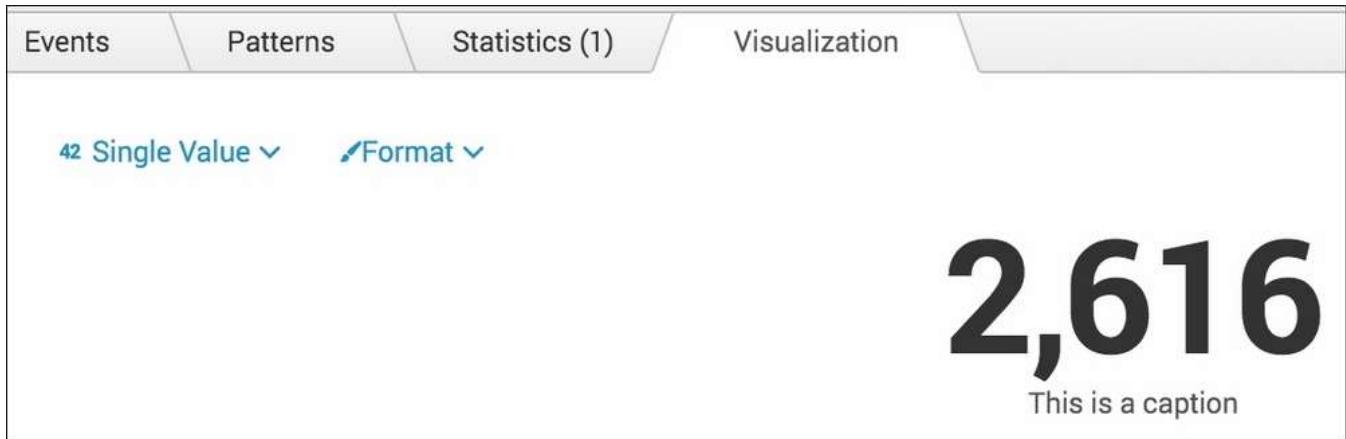
A single value on the dashboard is very useful, but providing some visual colors and context to the value can prove even more useful.

Adding labels to a single value panel

Run the same search from this recipe, and when the search completes, click on the **Visualization** tab and choose the **Single Value** visualization type. Next, click on the **Format** button, and in the drop-down menu that appears, you have the option to enter a **Caption** text value:



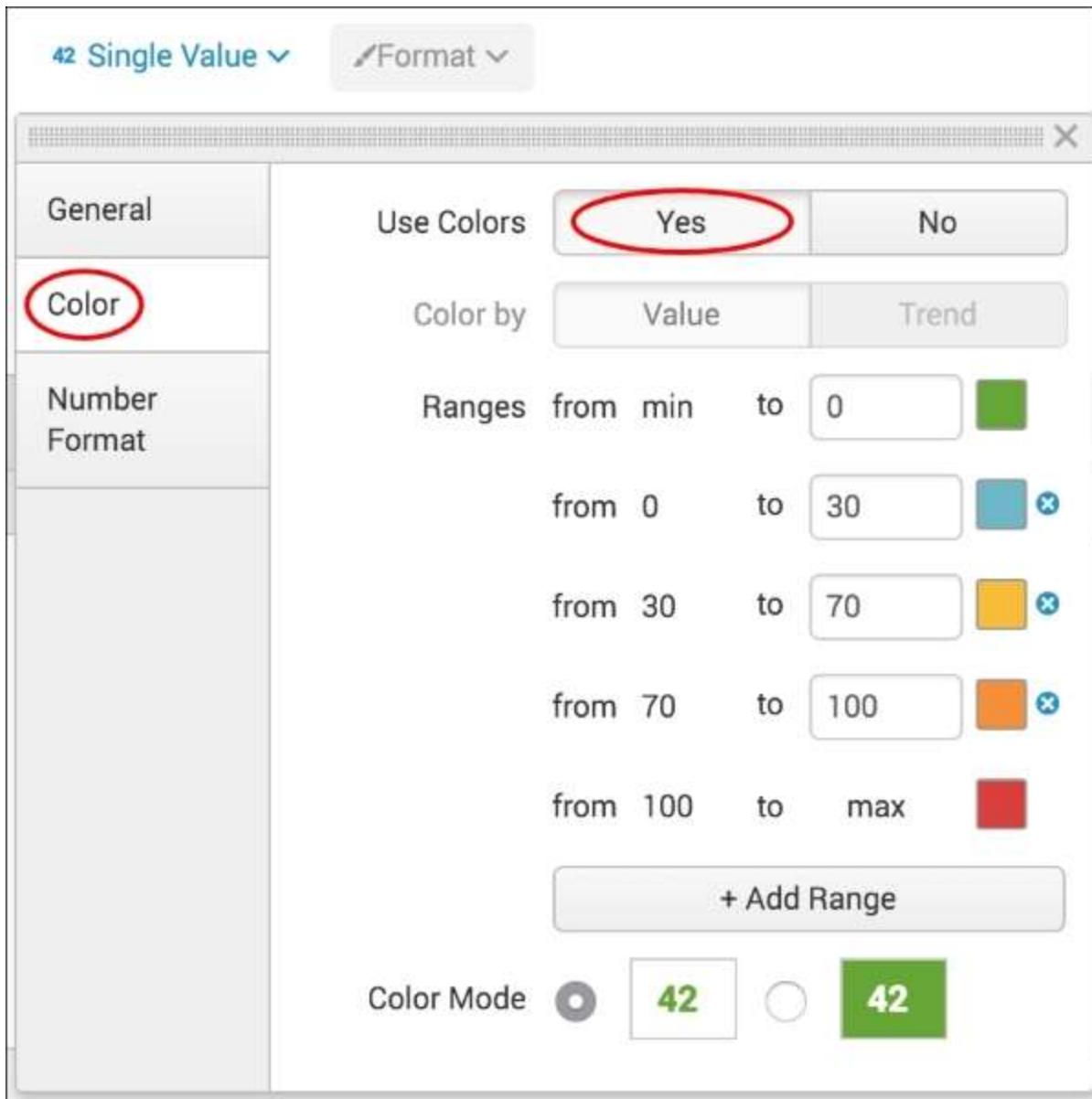
After entering your desired values, click *Enter*. The changes will appear immediately, as shown in the following screenshot:



You can now save this single value report as a panel on a dashboard, as you did earlier, but can leave the **Panel Title** field empty as the description of the value is now part of the data itself.

COLORING THE VALUE BASED ON RANGES

After adding labels, it can be useful to provide some visual color to the numeric value displayed, based on a given range within which the number might be. This can be done directly from the **Single Value** formatting option by clicking on the **Color** tab and selecting **Use Colors** as follows:



From this screen you can set the colors and ranges that you wish to use for your value. You can also select whether you want a color mode

where the number is colored or the background is colored. Once you have made your selection, click anywhere on the page.

ADDING TRENDS AND SPARKLINES TO THE VALUES

The **Single Value** visualization is also able to display trends and sparklines. However, for this to happen, the search must be based on the `timechart` command. We can modify the search from the recipe to illustrate this functionality, as follows:

```
index=main sourcetype=access_combined | timechart span=1h  
dc(JSESSIONID)
```

Run this modified search and when the search completes, click on the **Visualization** tab and choose the **Single Value** visualization type. Next, click on the **Format** button. On the **General** tab, you will now notice that there are many more options available to select from, specifically around trending. Leave the default options selected, but select **Show trend in Percent**. Now select the **Color** tab from the formatting box. Again, you will notice some additional options. Select **Color by Trend** and then select **Apply**. You will now see the **Single Value** visualization represented as a number, with a Sparkline illustrating the last 24 hours and also a downward or upward trend percentage compared against the previous hour:



NOTE

For further information on the formatting options available for the Single Value visualization, visit http://docs.splunk.com/Documentation/Splunk/latest/Viz/Visualizationreference#Single_value_visualizations.

Using a gauge to display the number of errors

Not every user interaction with a website goes smoothly. There are times when the accessed pages report an unsuccessful status code. Understanding this number and being able to apply acceptable low, medium, and high thresholds enables a better understanding of the current user experience when there are a higher number of errors than acceptable.

In this recipe, you will write a Splunk search to find the total number of errors over a given period of time. You will then graphically represent this value on the dashboard using a radial gauge.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar, the time range picker, and the **Visualization** tab. It is not required, but is advisable, that you complete all the recipes up until this point.

How to do it...

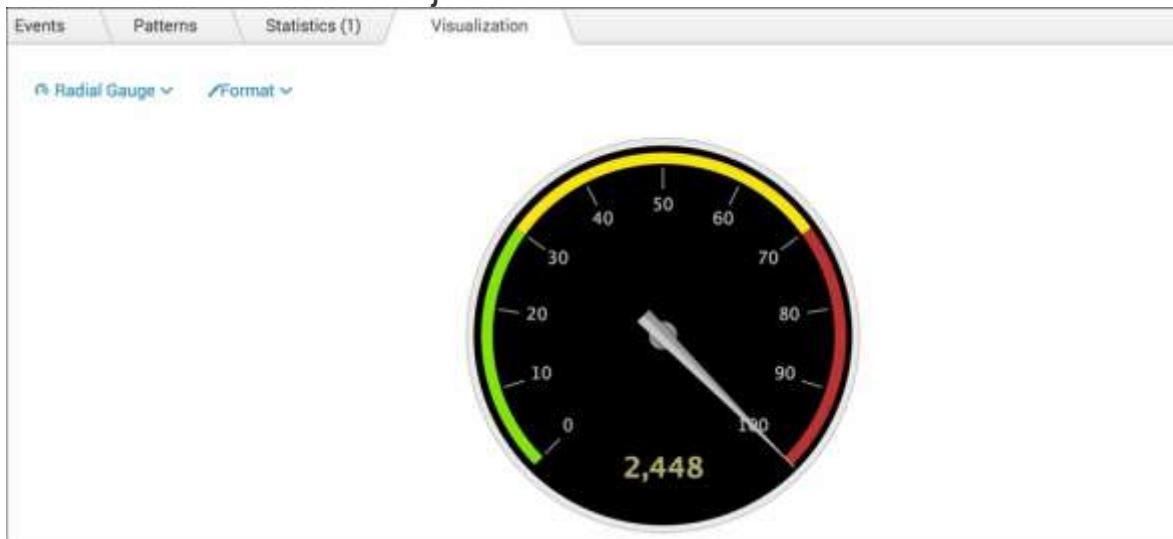
Follow the given steps to use a gauge visualization to display the number of web access errors:

1. Log in to your Splunk server.
2. Select the default **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

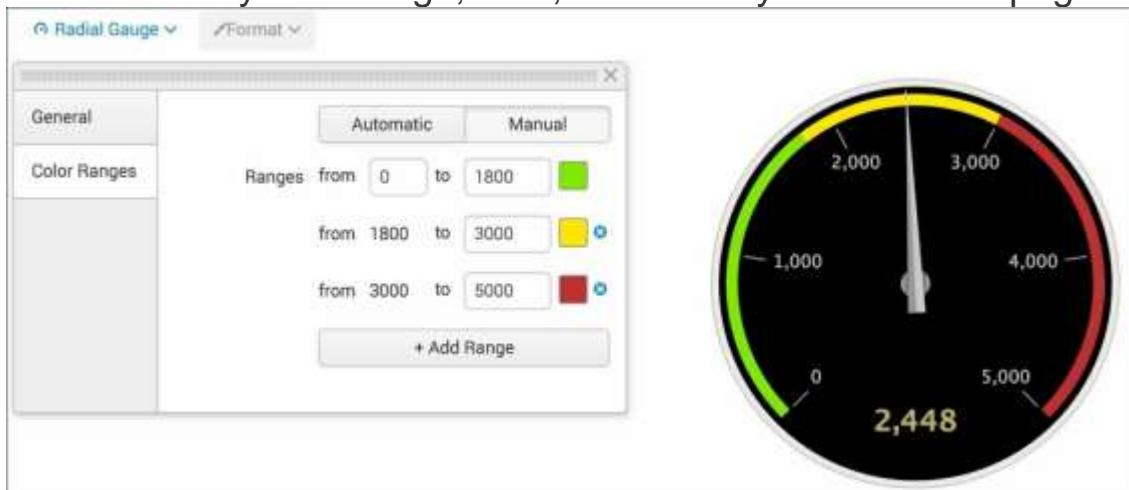
```
index=main sourcetype=access_combined NOT status="200" | stats  
count
```

4. Splunk will return the total count of events where the status was anything but successful.

5. Click on the **Visualization** tab.
6. As there are a number of visualizations within Splunk, the **Single Value** visualization might not be displayed by default within the **Visualization** tab. Click on the dropdown that lists the visualization types and select **Radial Gauge**.
7. Your data should now be visualized as a gauge and the needle on the gauge is likely all the way into the red—this is because the thresholds need to be adjusted:



8. To adjust the thresholds on the radial gauge, click on the **Format** button. Then, from the drop-down menu, click on the **Color Ranges** tab, and finally, click on the **Manual** button.
9. From within the **Manual Color Ranges** screen, you can now adjust the values to acceptable amounts such that the needle sits in the middle of the yellow range; then, click on anywhere on the page:



10. Save this report by clicking on **Save As** and then on **Report**. Name the report `cp03_webaccess_errors` and click on **Save**. On the next screen, click on **Add to Dashboard**.
11. You will now add this report to the **Website Monitoring** dashboard. Select the button labeled **Existing**, and from the drop-down menu that appears, select the **Website Monitoring** dashboard. For the **Panel Title** field value, enter **Total Number of Errors** and select for the panel to be powered by **Report**; then, click on **Save**.
12. The next screen will confirm that the dashboard has been created and the panel has been added. Click on **View Dashboard** to see for yourself. The radial gauge visualization should now be positioned on the dashboard below the previous two panels.
13. Arrange the dashboard so that the radial gauge panel is to the right of the single value panel. Click on the **Edit** button, and from the drop-down menu, select **Edit Panels**. Move the radial gauge panel accordingly.
14. Finally, click on **Done** to save the changes to your dashboard. The dashboard should now look like the following screenshot:



How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined NOT status="200"</code>	You should be familiar with this search from the earlier recipes in this chapter. However, we added the search criteria to not return any event where the <code>status</code> field is equal to <code>200</code> (success).
<code> stats count</code>	Using the <code>stats</code> command, we count the total number of events that are returned.

The **Visualization** tab simply takes the numeric output of the `stats` command and overlays the given visualization. In this case, you overlaid a radial gauge visualization on the total count of events that were not successful.

There's more...

In this recipe, we did not use the other two types of gauges: the filler gauge and the marker gauge. It is advisable to try out the other gauges, as they might be the preferred single value visualization for your intended audience.

NOTE

For more information on the available gauge visualizations, visit <http://docs.splunk.com/Documentation/Splunk/latest/Viz/Visualizationreference#Gauges>.

Charting the number of method requests by type and host

In our environment, where multiple hosts are responding to web requests for customers who browse the website, it is good to get an idea of the current number of each method request split by the host. Methods relate to request/response actions between a customer's web client and our web hosts. Having this type of information can enable us to understand if these requests are properly being balanced across the hosts or if one host is receiving the majority of the load.

In this recipe, you will write a Splunk search to chart the number of method requests split by type and host. You will then graphically represent these values on a dashboard, using a column chart.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from Session 3-1, *Play Time – Getting Data In*. You should be familiar with the Splunk search bar, the time range picker, and the **Visualization** tab. It is not required, but is advisable, that you also complete all the recipes up until this point.

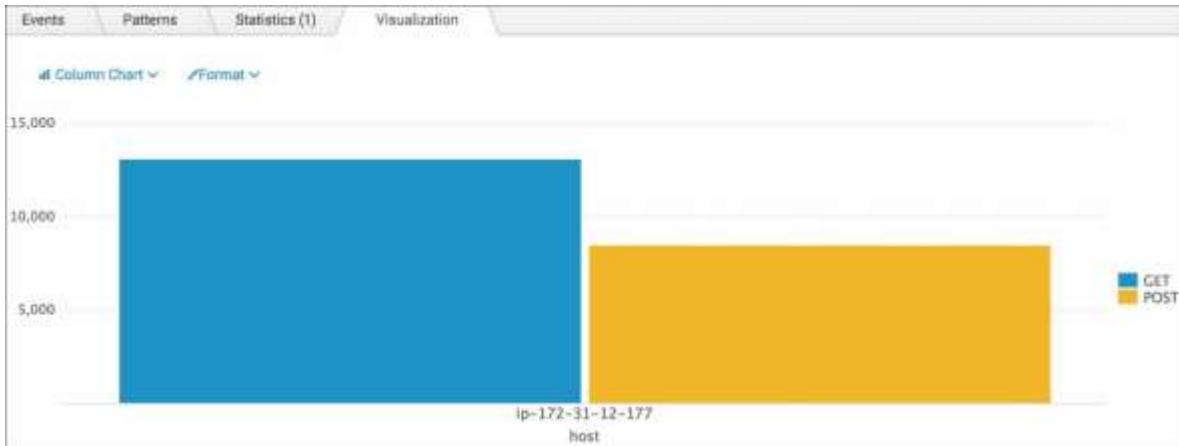
How to do it...

Follow the given steps to chart the number of method requests by type and host:

1. Log in to your Splunk server.
2. Select the default **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

```
index=main sourcetype=access_combined | chart count by host,method
```

4. Splunk will return a tabulated list of the total counts for each method request split by host.
5. Click on the **Visualization** tab.
6. Click on the dropdown that lists the visualization types and select **Column**.
7. Your data should now be visualized as shown in the following screenshot:



8. Save this report by clicking on **Save As** and then on **Report**. Name the report **cp03_methods_by_host** and click on **Save**. On the next screen, click on **Add to Dashboard**.
9. You will now add this to the **Website Monitoring** dashboard. Select the button labeled **Existing**, and from the drop-down menu that appears, select the **Website Monitoring** dashboard. For the **Panel Title** field value, enter **Method Requests by Type and Host** and select **Report** in the **Panel Powered By** field; then, click on **Save**.
10. The next screen will confirm that the dashboard has been created and the panel has been added. Click on **View Dashboard** to see for yourself.
11. Edit the dashboard to position the column chart visualization below the previously added panels.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes.
<code> chart count by host,method</code>	The <code>chart</code> command simply performs a count of events split by host and method. This produces the total count of each method for a given host.

The **Visualization** tab simply takes the tabulated output of the `stats` command and overlays the given visualization. In this case, you overlaid a column chart visualization on the total count for each method split by host.

Creating a timechart of method requests, views, and response times

Having the right single values displayed on a dashboard can be beneficial to understanding key metrics, but can also be limiting in providing true operational intelligence on how the different metrics of our website affect one another. By plotting values such as the number of method requests, the number of total views, and the average response times over a given time range, you can begin to understand if there is any correlation between these numbers. This can be very beneficial in understanding things such as if the average response time of pages is growing due to the number of active `POST` requests to the website or if one type of request is making up for the majority of the total number of requests at that given time.

In this recipe, you will create a Splunk search using the `timechart` command to plot values over a given time period. You will then graphically represent these values using a line chart.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar, the time range picker, and the **Visualization** tab. It is not required, but is advisable, that you also complete all the recipes up until this point.

How to do it...

Follow the given steps to create a timechart of method requests, views, and response times:

1. Log in to your Splunk server.
2. Select the default **Search & Reporting** application.

3. Ensure that the time range picker is set to **Last 7 days** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*.

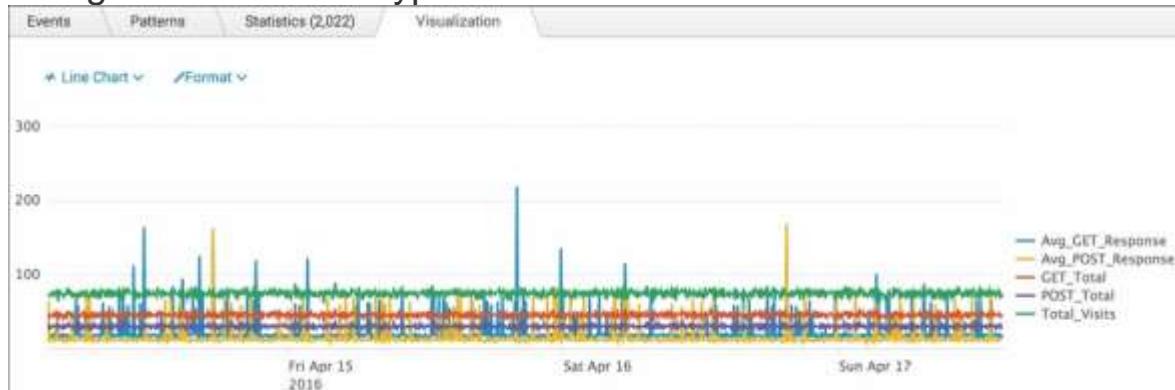
```
index=main sourcetype=access_combined | eval
GET_response=if(method=="GET",response,0) | eval
POST_response=if(method=="POST",response,0) | timechart span=5m
avg(GET_response) AS Avg_GET_Response, avg(POST_response) AS
Avg_POST_Response, count(eval(method=="GET")) AS GET_Total,
count(eval(method=="POST")) AS POST_Total, count AS
Total_Visits
```

4. Splunk will return a time series chart of values for the average response time of **GET** and **POST** requests, the count of **GET** and **POST** requests, and the total count of web page visits:

The screenshot shows the Splunk search results for the provided query. The search bar at the top contains the search command. Below it is a table with the following data:

_time	Avg_GET_Response	Avg_POST_Response	GET_Total	POST_Total	Total_Visits
2016-04-14 03:00:00	17.000000	9.391304	44	25	69
2016-04-14 03:05:00	17.985915	63.591549	41	30	71
2016-04-14 03:10:00	16.888889	0.930556	45	27	72
2016-04-14 03:15:00	18.013699	11.164384	45	28	73
2016-04-14 03:20:00	16.154930	9.323544	44	27	71

5. Click on the **Visualization** tab and select **Line** from the drop-down listing of visualization types to visualize the data as a line chart:



6. Save this report by clicking on **Save As** and then on **Report**. Name the report `cp03_method_view_reponse` and click on **Save**. On the next screen, click on **Add to Dashboard**.
7. You will now add this report to the **Website Monitoring** dashboard. Select the button labeled **Existing**, and

from the drop-down menu that appears, select the **Website Monitoring** dashboard. For the **Panel Title** field value, enter `Website Response Performance` and select **Report in Panel Powered By** field; then, click **Save**.

8. The next screen will confirm that the dashboard has been created and the panel has been added. Click on **View Dashboard** to see for yourself. The line chart visualization should now be positioned on the dashboard below the previously added panels.
9. Arrange the dashboard so that the line chart panel is on the right-hand side of the column chart panel created in the previous recipe. Click on the **Edit** button, and from the drop-down menu, select **Edit Panels**. Move the line chart panel accordingly.
10. Finally, click on **Done** to save the changes to your dashboard.

How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this book.
<code> eval GET_response=if(method=="GET", response, 0)</code>	Using the <code>eval</code> command, we create a new field called <code>GET_response</code> , whose value is based on the return value of the <code>if</code> function. In this case, if the method is <code>GET</code> , then the value returned is the value of the <code>response</code> field; otherwise, the value returned is <code>0</code> .
<code> eval POST_response=if(method=="POST", response, 0)</code>	Using the <code>eval</code> command, we create a new field, called <code>POST_response</code> , whose value is based on the return value of the <code>if</code> function. In this case, if the method is <code>POST</code> ,

Search fragment	Description
<pre>timechart span=5m avg(GET_response) AS Avg_GET_Response, avg(POST_response) AS Avg_POST_Response, count(eval(method=="GET")) AS GET_Total, count(eval(method=="POST")) AS POST_Total, count AS Total_Visits</pre>	<p>then the value returned is the value of the <code>response</code> field; otherwise, the value returned is <code>0</code>.</p> <p>Using the <code>timechart</code> command, we first specify a span of five minutes. Next, we calculate the average value for the given span of <code>GET_response</code> and <code>POST_response</code>. Next, we count the total number of <code>GET</code> and <code>POST</code> events. Finally, the total number of events, both <code>GET</code> and <code>POST</code>, are counted. Note that we make use of the <code>AS</code> operator to rename the fields so that they are meaningful and easy to understand when displayed on our chart.</p>

The **Visualization** tab takes the time series output of the `timechart` command and overlays the given visualization. In this case, you overlaid the line chart visualization.

There's more...

In this recipe, we looked at the values represented as a whole across our web server environment. However, in instances like ours, where web traffic is balanced across multiple servers, it is a good idea to split the values based on their respective hosts.

METHOD REQUESTS, VIEWS, AND RESPONSE TIMES BY HOST

It is very easy to obtain a more granular view of events split by the host where the events are occurring. All we need to do is add the `by` clause to the end of our previous Splunk search, as follows:

```
index=main sourcetype=access_combined | eval  
GET_response=if(method=="GET",response,0) | eval  
POST_response=if(method=="POST",response,0) | timechart  
span=5m avg(GET_response) AS Avg_GET_Response,  
avg(POST_response) AS Avg_POST_Response,  
count(eval(method=="GET")) AS GET_Total,  
count(eval(method=="POST")) AS POST_Total, count AS  
Total_Visits by host
```

As simple as this is, we can now visualize the values broken down by the host on which these values originated. In a distributed environment, this can be most crucial in understanding where latency or irregular volumes exist.

Using a scatter chart to identify discrete requests by size and response time

As shown by the recipes up until this point, there is vast intelligence that can be attained by building visualizations that summarize the current application state, analyze performance data over time, or compare values to one another. However, what about those discrete events that appear off in the distance at odd or random times? These events might not be correctly reflected when looking at a column chart, single value gauge, or pie chart, as to most calculations, they are just a blip in the radar somewhere off in the distance. However, there could be times where these discrete events are indicative of an issue or simply the start of one.

In this recipe, you will write a very simple Splunk search to plot a few elements of web request data in a tabular format. The real power comes next, when you will graphically represent these values using a scatter chart.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar, the time range picker, and the **Visualization** tab. It is not required, but is advisable, that you complete the recipes up until this point.

How to do it...

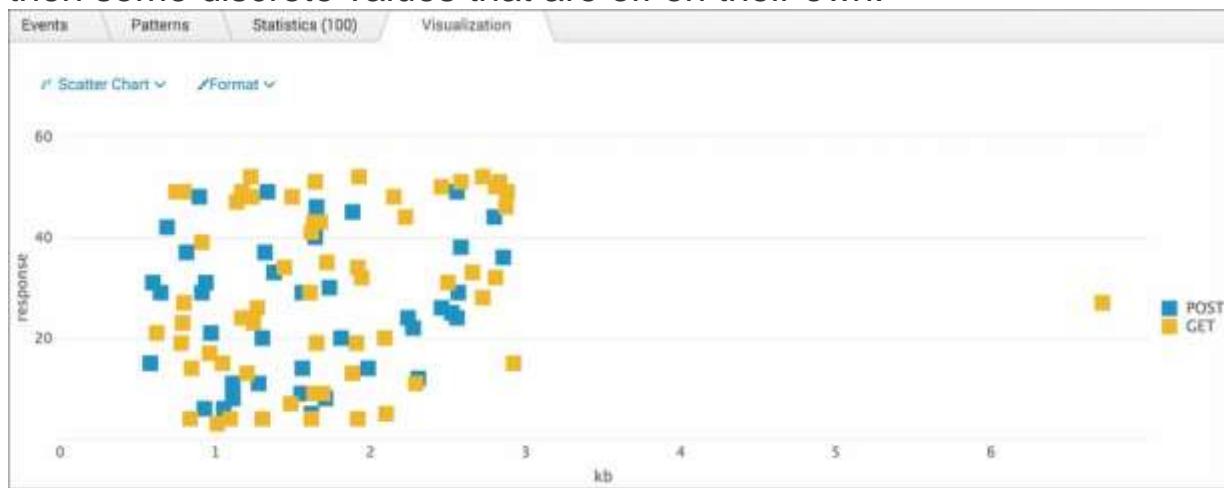
Follow the given steps to use a scatter chart to identify discrete requests by size and response time:

1. Log in to your Splunk server.
2. Select the default **Search & Reporting** application.

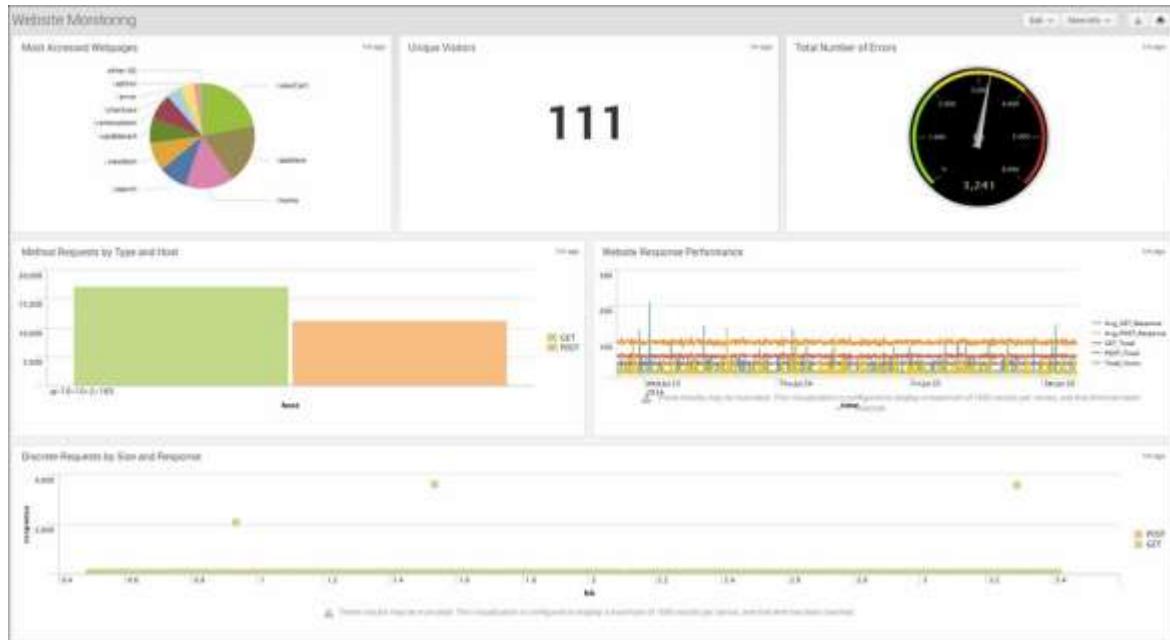
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*:

```
index=main sourcetype=access_combined | eval kb=bytes/1024 |
table method kb response
```

4. Splunk will return a tabulated list of the `method`, `kb`, and `response` fields for each event.
5. Click on the **Visualization** tab and select **Scatter** from the drop-down list of visualization types to see the data represented as a scatter plot chart. You should see the cluster of normal activity and then some discrete values that are off on their own:



6. Save this report by clicking on **Save As** and then on **Report**. Name the report `cp03_discrete_requests_size_response` and click on **Save**. On the next screen, click on **Add to Dashboard**.
7. You will now add this to the **Website Monitoring** dashboard. Select the button labeled **Existing**, and from the drop-down menu that appears, select the **Website Monitoring** dashboard. For the **Panel Title** field value, enter **Discrete Requests by Size and Response** and select **Report** in **Panel Powered By**; then, click on **Save**.
8. The next screen will confirm that the dashboard has been created and the panel has been added. Click on **View Dashboard** to see for yourself. The scatter chart visualization should now be positioned on the dashboard below the previously added panels:



How it works...

Let's break down the search piece by piece:

Search fragment	Description
<code>index=main sourcetype=access_combined</code>	You should now be familiar with this search from the earlier recipes in this book.
<code> eval kb=bytes/1024</code>	Using the <code>eval</code> command, we convert the size of the request from bytes to kilobytes. For presentation purposes, this makes it easier to read and relate.
<code> table method kb response</code>	Using the <code>table</code> command, we plot our data points that will be represented on the scatter chart. The first field, <code>method</code> , presents the data that appears in the legend. The second field, <code>kb</code> , represents the x axis value. Finally, the third

Search fragment	Description
	field, <code>response</code> , represents the yaxis value.

There's more...

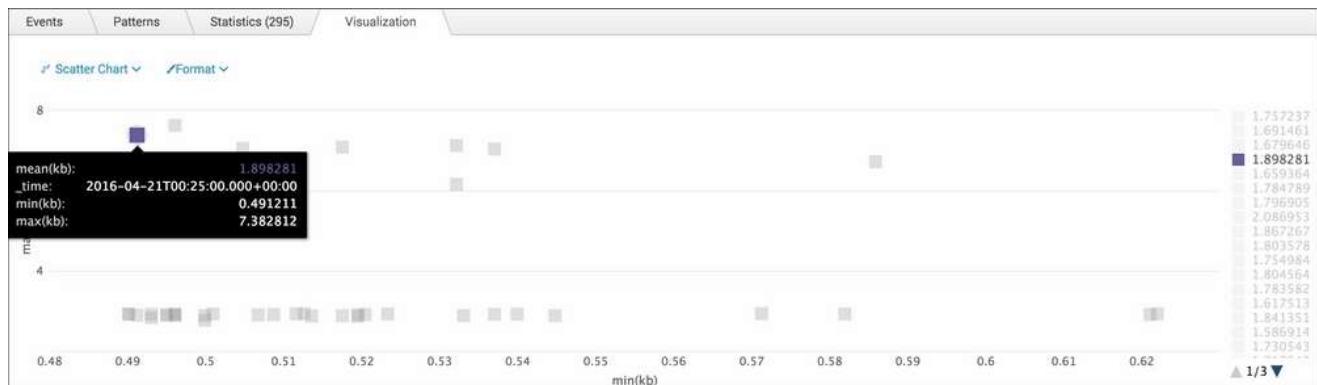
Aside from simply plotting the data points for a scatter chart in tabular form, you can leverage the `timechart` command and its available functions to better identify and provide more context to these discrete values.

USING TIME SERIES DATA POINTS WITH A SCATTER CHART

The Splunk search you ran in this recipe can be modified to make use of the `timechart` command and all the functions it has to offer. Using the **Visualization** tab and scatter chart, run the following Splunk search over the last 24 hours:

```
index=main sourcetype=access_combined | eval kb=bytes/1024 |
timechart span=5m mean(kb) min(kb) max(kb)
```

As you can see, with the `timechart` command, you first bucket the events into 5-minute intervals, as specified by the `span` parameter. Next, the `mean`, `min`, and `max` values of the `kb` field for that given time span are calculated. This way, if there is an identified discrete value, you can see more clearly what drove that span of events to be discrete. An example of this can be found in the following screenshot. In this scatter chart, we have highlighted one discrete value that is far outside the normal cluster of events. You can see why this might have stood out using the `min` and `max` values from this event series:



Creating an area chart of the application's functional statistics

Understanding not only how your web page is performing and responding to requests but also how underlying applications that you rely on is critical to the success of any website. You need to have a constant pulse on how the application is behaving and if any trends are emerging or correlations are being observed between interdependent pieces of data. The experience a customer has with your website is reliant on the constant high performance of all its components.

In this recipe, you will write a Splunk search using the `timechart` command to plot web application memory and response time statistics over a given time period. You will then graphically present these values using an area chart.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from Session 3-1, *Play Time – Getting Data In*. You should be familiar with the Splunk search bar, the time range picker, and the **Visualization** tab. It is not required, but is advisable, that you complete all the recipes up until this point.

How to do it...

Follow the given steps to create an area chart of an application's functional statistics:

1. Log in to your Splunk server.
2. Select the default **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*:

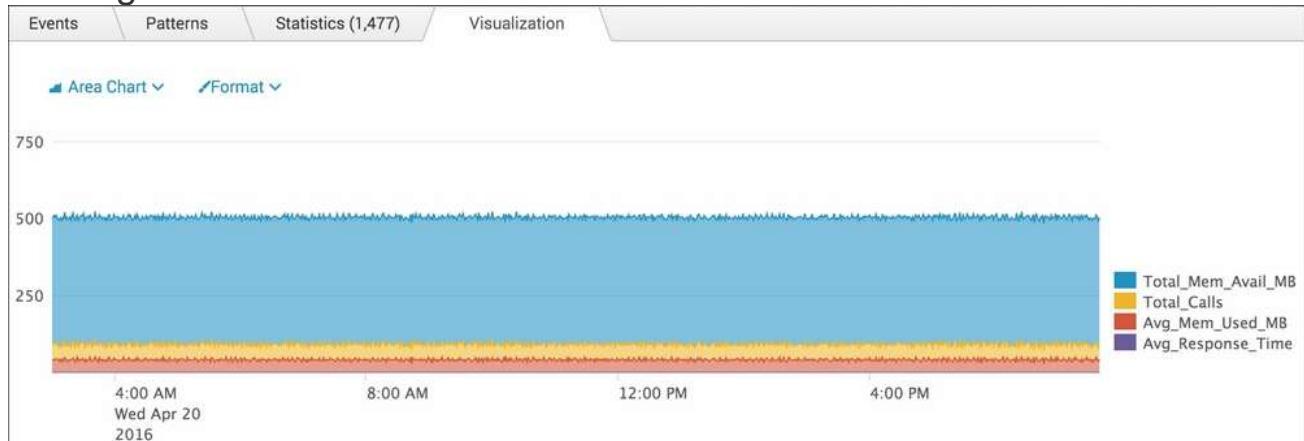
```
index=main sourcetype=log4j | eval  
mem_used_MB=(mem_used/1024)/1024 | eval  
mem_total_MB=(mem_total/1024)/1024 | timechart span=1m
```

```

values(mem_total_MB) AS Total_Mem_Avail_MB, count AS
Total_Calls, avg(mem_used_MB) AS Avg_Mem_Used_MB,
avg(response_time) AS Avg_Response_Time

```

4. Splunk will return a time series chart of values for the average response time of `GET` and `POST` requests, the count of `GET` and `POST` requests, and the total count of web page visits.
5. Click on the **Visualization** tab and select **Area** from the drop-down list of visualization types to see the data represented as an area chart. Note how the data is stacked for better visual representation of the given data:



6. Save this report by clicking on **Save As** and then on **Report**. Name the report `cp03_webapp_functional_stats` and click on **Save**. On the next screen, click on **Add to Dashboard**.
7. You will now add this to the **Website Monitoring** dashboard. Select the button labeled **Existing**, and from the dropdown menu that appears, select the **Website Monitoring** dashboard. For the **Panel Titlefield** value, enter `Web Application Functional Statistics` and select **Report** in **Panel Powered By**; then, click on **Save**.
8. The next screen will confirm that the dashboard has been created and the panel has been added. Click on **View Dashboard** to see for yourself.

How it works...

Let's break down the search piece by piece:

Search Fragment	Description
<pre>index=main sourcetype=log4j</pre>	In this example, we search for our application's logs that have the <code>log4j</code> sourcetype.
<pre> eval mem_used_MB=(mem_used/1024)/1024</pre>	Using the <code>eval</code> command, we calculate the amount of memory currently being used, in megabytes.
<pre> eval mem_total_MB=(mem_total/1024)/1024</pre>	Using the <code>eval</code> command again, we calculate the total amount of memory that is available for use, in megabytes.
<pre> timechart span=1m values(mem_total_MB) AS Total_Mem_Avail_MB, count AS Total_Calls, avg(mem_used_MB) AS Avg_Mem_Used_MB, avg(response_time) AS Avg_Response_Time</pre>	Using the <code>timechart</code> command, we first specify a span of 1 minute for our events. Next, we use the <code>values</code> function to retrieve the value stored in the <code>mem_total_MB</code> field. The <code>count</code> function is then used to calculate the total number of function calls during the given time span. The average function is then called twice, to calculate the average amount of memory used and the average response time for the function call during the

Search Fragment	Description
	given time span. Note that we make use of the <code>AS</code> operator to rename the fields so that they are meaningful and easy to understand when displayed on our chart.

The **Visualization** tab takes the time series output of the `timechart` command and overlays the given visualization. In this case, you overlaid an area chart.

Using a bar chart to show the average amount spent by category

Throughout this chapter, you have been building visualizations to provide insight into the operational performance of our e-commerce website. It can also be useful to understand the customers' views and the factors that might drive them to the website. This type of information is traditionally most useful for product or marketing folks. However, it can also be useful to gain an understanding around whether an item is increasing in popularity and if this could ultimately lead to additional customers and heavier load on the site.

In this recipe, you will write a Splunk search to calculate the average amount of money spent, split out by product category. You will then graphically present this data using a bar chart on a new **Product Monitoring** dashboard.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should be familiar with the Splunk search bar, the time range picker, and the **Visualization** tab. It is not required, but is advisable, that you complete all the recipes up until this point.

How to do it...

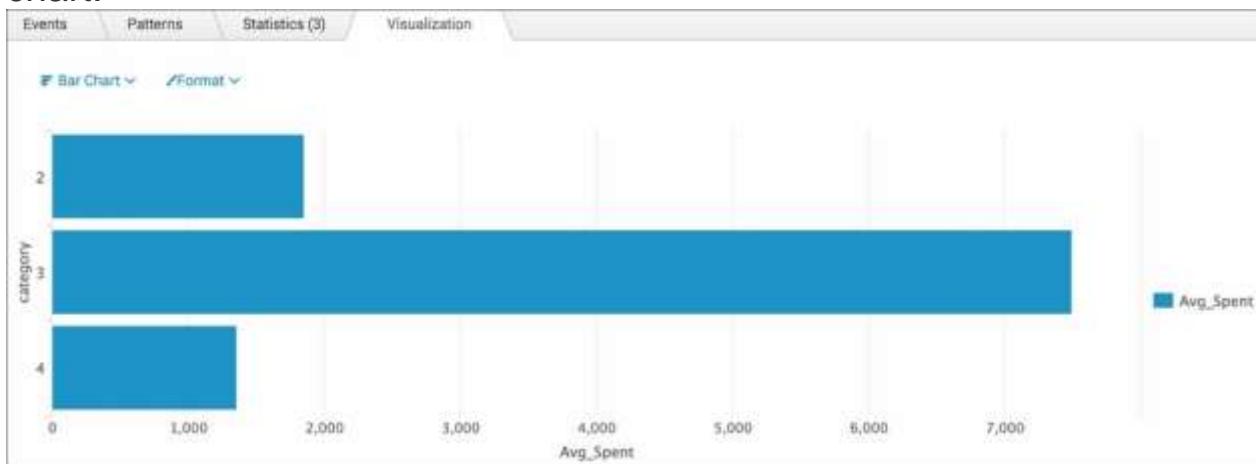
Follow the given steps to use a bar chart to show average amount spent by category:

1. Log in to your Splunk server.
2. Select the default **Search & Reporting** application.

3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*:

```
index=main sourcetype=log4j | transaction sessionId maxspan=30m
| search requestType="checkout" | stats avg(total) AS Avg_Spent
by category
```

4. Splunk will return a tabulated list, detailing the category and the associated average amount spent.
5. Click on the **Visualization** tab and select **Bar** from the drop-down list of visualization types to see the data represented as a bar chart:



6. Save this report by clicking on **Save As** and then on **Report**. Name the report `cp03_average_spent_category` and click on **Save**. On the next screen, click on **Add to Dashboard**.
7. You will now add this to a new **Product Monitoring** dashboard. Select the button labeled **New** and enter the dashboard title **Product Monitoring**. For the **Panel Title** field value, enter `Average Spent by Category` and select **Report** in the **Panel Powered By** field; then, click on **Save**:

Save As Dashboard Panel

Dashboard	<input type="radio"/> New	<input type="radio"/> Existing
Dashboard Title	Product Monitoring	
Dashboard ID ?	product_monitoring	
Dashboard Description	optional	
Dashboard Permissions	<input type="radio"/> Private	<input checked="" type="radio"/> Shared in App
Panel Title	Average Spent by Category	
Panel Powered By	<input type="radio"/> Inline Search	<input checked="" type="radio"/> Report
Panel Content	<input type="checkbox"/> Statistics	<input type="checkbox"/> Bar
Cancel		Save

- The next screen will confirm that the dashboard has been created and the panel has been added. Click on **View Dashboard** to see for yourself.

How it works...

Let's break down the search piece by piece:

Search Fragment	Description
<code>index=main sourcetype=log4j</code>	In this example, we search for our application's logs that have the <code>log4j</code> sourcetype.
<code> transaction sessionId maxspan=30m</code>	Using the <code>transaction</code> command, we group together all the events that

Search Fragment	Description
	share the same <code>sessionId</code> in a 30-minute span.
<code> search requestType="checkout" paymentReceived="Y"</code>	Using the <code>search</code> command, we limit the grouped results to those that have only a <code>checkout</code> event and where the payment was received. In this visualization, a purchase does not qualify for consideration if it did not successfully process.
<code> stats avg(total) AS Avg_Spent_by_category</code>	Using the <code>stats</code> command, we calculate the average total amount spent by category. Note that we make use of the <code>AS</code> operator to rename the field so that it is meaningful and easy to understand when displayed on our chart.

The **Visualization** tab simply takes the time series output of the `stats` command and overlays the given visualization. In this case, you overlaid a bar chart visualization.

Creating a line chart of item views and purchases over time

Continuing on from the last recipe, we further improve our understanding of customer activities by now looking at a chart of item views and actual purchases over a given time period. This will allow you to understand if the customers who view an item actually follow through with purchasing the given item.

In the last recipe of this chapter, you will write a Splunk search to chart item views and purchases over a given time period. You will then graphically present this data using a line chart.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from Session 3-1, *Play Time – Getting Data In*. You should be familiar with the Splunk search bar, the time range picker, and the **Visualization** tab. It is not required, but is advisable, that you complete all the recipes up until this point.

How to do it...

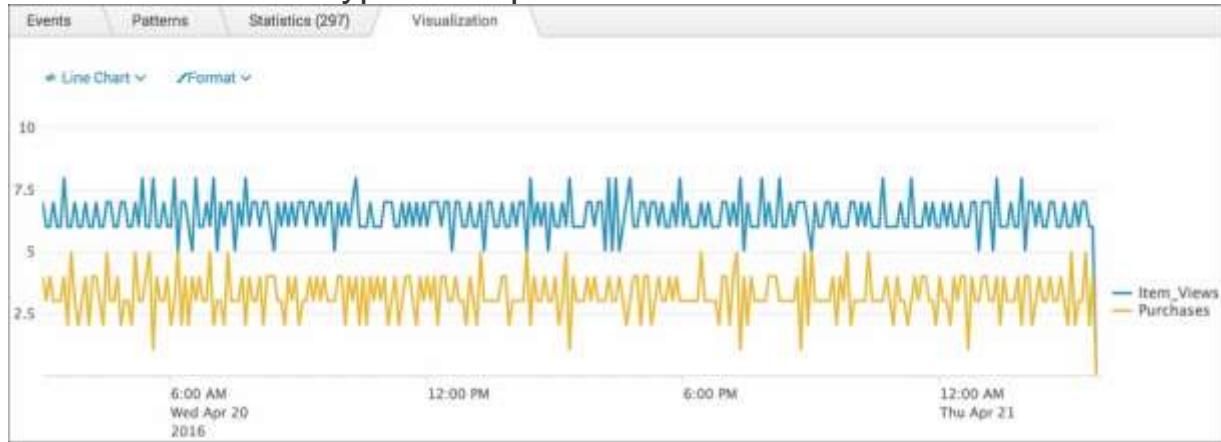
Follow the given steps to create a line chart of item views and purchases over time:

1. Log in to your Splunk server.
2. Select the default **Search & Reporting** application.
3. Ensure that the time range picker is set to **Last 24 hours** and type the following search into the Splunk search bar. Then, click on **Search** or hit *Enter*:

```
index=main sourcetype=access_combined | timechart span=5m  
count(eval(uri_path="/ViewItem")) AS Item_VIEWS,  
count(eval(uri_path="/checkout")) AS Purchases
```

4. Splunk will return a time series-based chart, listing the count of item views and the count of purchases over the given time period.

5. Click on the **Visualization** tab and select **Line** from the drop-down list of visualization types to represent the data as a line chart:



6. Save this report by clicking on **Save As** and then on **Report**. Name the report **cp03_item_views_purchases** and click on **Save**. On the next screen, click on **Add to Dashboard**.
7. You will now add this to the **Product Monitoring** dashboard. Select the button labeled **Existing**, and from the drop-down menu that appears, select the **Product Monitoring** dashboard. For the **Panel Title** field value, enter **Item Views vs. Purchases** and select **Report** in **Panel Powered By**; then, click on **Save**.
8. The next screen will confirm that the dashboard has been created and the panel has been added. Click on **View Dashboard** to see for yourself.
9. Arrange the dashboard so that the line chart panel is to the right of the bar chart panel created in the previous recipe. Click on the **Edit button**, and from the drop-down menu, select **Edit Panels**. Move the line chart panel accordingly.
10. Finally, click on **Done** to save the changes to your dashboard:

Session 3-4: Building an Operational Intelligence Application

Frank Anaya

Building an Operational Intelligence Application

In this session, we will learn how to build and modify a Splunk application. We will cover the following topics:

- Creating an Operational Intelligence application
- Adding dashboards and reports
- Organizing the dashboards more efficiently
- Dynamically drilling down on activity reports
- Creating a form to search web activity
- Linking web page activity reports to the form
- Displaying a geographical map of visitors
- Scheduling PDF delivery of a dashboard

Introduction

In the previous session, we were introduced to Splunk's awesome dashboarding and visualization capabilities. We created several basic dashboards and populated them with various Operational Intelligence-driven visualizations. In this session, we will continue to build on what we have learned in the previous sessions and further advance our Splunk dashboarding knowledge. You will learn how to create a Splunk application and populate it with several dashboards. You will also learn to use some of Splunk's more advanced dashboarding capabilities such as forms, drill downs, and maps.

Splunk applications are best thought of as workspaces designed specifically around certain use cases. In this session, we will be building a new application that focuses specifically on Operational Intelligence. Splunk apps can vary in complexity from a series of saved reports and dashboards through to complex, fully-featured standalone solutions. After logging in to Splunk for the first time, you actually interface with Splunk through the launcher application, which displays a dashboard that lists the other applications installed on the system. The Search and

Reporting application that we have been using in this book so far is an example of another bundled Splunk application.

TIP

Several vendors, developers, and customers have developed applications that can be used to get you started with your datasets. Most of these applications are available for free download from the Splunk app store at <http://splunkbase.splunk.com>.

In this session, you will also start to get to grips with the dashboard forms functionality. The best way to think about forms in Splunk is that they are essentially dashboards with an interface, allowing the users to easily supply values to the underlying dashboard searches. For example, a basic form in Splunk would be a dashboard with a user-selectable time range at the top. The user might then select to run the dashboard over the last 24 hours, and all the searches that power the dashboard visualizations will run over this selected time range.

Forms, by their very nature, require input. Luckily for us, Splunk has a number of common form inputs out-of-the-box that can be readily used by using the dashboard editor or [SimpleXML](#). The available form inputs and an explanation of their common usages are detailed in the following table:

Input	Common usage
Dropdown	This is used to display the lists of user selectable values. Drop-downs can be populated dynamically using Splunk searches and even filtered based on the user selection of another drop-down. The users can also select single values or multiple values.
Radio	This is used for simple yes/no or single selection type values. You can only select one value at a time with the radio buttons, unlike dropdown .
Text	This is a simple textbox allowing the user to type whatever value they want to search for. The textbox is great for searching wildcard type values, such as field values of <code>abc*</code> .
Time	This is a time range picker. This is exactly the same as the time range picker found on the main Splunk search dashboard. You can

Input	Common usage
	add a time range for the entire dashboard or for individual dashboard panels.

Dashboards in Splunk are coded behind the scenes in something known as [SimpleXML](#). This [SimpleXML](#) code can be edited directly or by use of Splunk's interactive, GUI-based dashboard editor. For the most part, this session will focus on using the GUI-based dashboard editor, which allows dashboards to be edited without touching a line of code—nice! However, you will be introduced to direct [SimpleXML](#) editing in order to take advantage of more advanced capabilities and options.

Ok, enough of discussion; let's get started!

Creating an Operational Intelligence application

This recipe will show you how to create an empty Splunk app that we will use as the starting point for building our Operational Intelligence application.

Getting ready

To go through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should have also completed the recipes from the earlier sessions. You should be familiar with navigating the Splunk user interface.

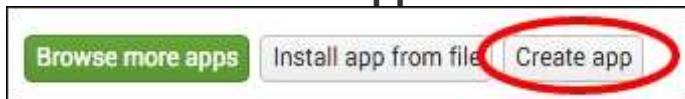
How to do it...

Follow the given steps to create the Operational Intelligence application:

1. Log in to your Splunk server.
2. From the **Apps** menu in the upper left-hand corner of the home screen, click on the gear icon.



3. Click on the **Create app** button.



4. Complete the fields in the box that follows. Name the app [Operational Intelligence](#) and give it a folder name

of `operational_intelligence`. Add a version number and provide an author name. Ensure that **Visible** is set to **Yes** and the barebones template is selected.

Add new

Apps » Add new

Name
Operational Intelligence 

Give your app a friendly name for display in Splunk Web.

Folder name *
operational_intelligence

This name maps to the app's directory in \$SPLUNK_HOME/etc/apps/.

Version
1.0

App version.

Visible
 No Yes

Only apps with views should be made visible.

Author
John Smith

Name of the app's owner.

Description
An application for Operational Intelligence

Enter a description for your app.

Template
barebones 

These templates contain example views and searches.

- When the form is completed, click on **Save**. This should be followed by a blue bar with the message **Successfully saved operational_intelligence**.

Congratulations, you just created a Splunk application!

How it works...

When an app is created through the Splunk GUI, as in this recipe, Splunk essentially creates a new folder (or directory) named `operational_intelligence` within the `$SPLUNK_HOME/etc/apps` directory. Within the `$SPLUNK_HOME/etc/apps/operational_intelligence` directory, you will find four new subdirectories that contain all the configuration files needed for the barebones Operational Intelligence app that we just created.



The eagle-eyed among you would have noticed that there were two templates that could have been selected from when creating the app: `barebones` and `sample_app`. The `barebones` template creates an application with nothing much inside it, and the `sample_app` template creates an application populated with sample dashboards, searches, views, menus, and reports. If you wish, you can also develop your own custom template if you create lots of apps, which might enforce certain color schemes for example.

There's more...

As Splunk apps are just a collection of directories and files, there are other methods to add apps to your Splunk Enterprise deployment.

CREATING AN APPLICATION FROM ANOTHER APPLICATION

It is relatively simple to create a new app from an existing app without going through the Splunk GUI, should you wish to do so. This approach can be very useful when we create multiple apps with different `inputs.conf` files for deployment to Splunk UF.

Taking the app we just created as an example, copy the entire directory structure of the `operational_intelligence` app and name it `copied_app`:

```
cp -r $SPLUNK_HOME$/etc/apps/operational_intelligence/*  
$SPLUNK_HOME$/etc/apps/copied_app
```

Within the directory structure of `copied_app`, we must now edit the `apps.conf` file in the default directory.

Open `$SPLUNK_HOME$/etc/apps/copied_app/default/apps.conf` and change the label field to **My Copied App**, provide a new description, and then save the `conf` file:

```
#  
# Splunk app configuration file  
#  
[install]  
is_configured = 0  
  
[ui]  
is_visible = 1  
label = My Copied App  
  
[launcher]  
author = John Smith  
description = My Copied application  
version = 1.0
```

Now restart Splunk, and the new **My Copied App** application should now be seen in the application menu.

```
$SPLUNK_HOME$/bin/splunk restart
```

DOWNLOADING AND INSTALLING A SPLUNK APP

Splunk has an entire application website with hundreds of applications created by Splunk, other vendors, and even the users of Splunk. These

are great ways to get started with a base application, which you can then modify to meet your needs.

If the Splunk server that you are logged in to has access to the Internet, from the **Apps** page you can click on the **Browse More Apps** button. From here, you can search for apps and install them directly.

An alternative way to install a Splunk app is to visit <http://splunkbase.splunk.com> and search for the app. You will then need to download the application locally. From your Splunk server, on the **Apps** page, click on the **Install app from file** button and upload the app you just downloaded in order to install it.

Once the app has been installed, go and look at the directory structure that the installed application just created. Familiarize yourself with some of the key files and where they are located.

TIP

When downloading applications from the Splunk apps site, it is best practice to test and verify them in a non-production environment first. The Splunk apps site is community driven and, as a result, quality checks and/or technical support for some of the apps might be limited.

Adding dashboards and reports

As we saw in the previous session, dashboards are a great way to present many different pieces of information. Rather than having lots of disparate dashboards across your Splunk environment, it makes a lot of sense to group related dashboards into a common Splunk application, for example, putting operational intelligence dashboards into a common Operational Intelligence application.

In this recipe, you will learn how to move the dashboards and the associated reports you created in the last couple of sessions into our new Operational Intelligence application.

Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#). You should have also completed the recipes from the earlier sessions. You should be familiar with navigating the Splunk user interface.

How to do it...

Follow these steps to move your dashboards into the new application:

1. Log in to your Splunk server.
2. Select the newly created **Operational Intelligence** application.
3. From the top menu, select **Settings** and then select the **User interface** menu item.

The screenshot shows the Splunk Settings menu. On the left, there are three main navigation icons: 'Add Data' (green icon), 'Explore Data' (blue icon), and 'Distributed Management Console' (orange icon). The central column contains several configuration categories: 'KNOWLEDGE' (Searches, reports, and alerts; Data models; Event types; Tags; Fields; Lookups), 'User interface' (Alert actions; Advanced search; All configurations), 'SYSTEM' (Server settings; Server controls; Licensing), and 'DATA' (Data inputs; Forwarding and receiving; Indexes; Report acceleration summaries; Virtual indexes; Source types). A red oval highlights the 'User interface' link under the Knowledge category. To the right, there are sections for 'DISTRIBUTED ENVIRONMENT' (Indexer clustering; Forwarder management; Distributed search) and 'USERS AND AUTHENTICATION' (Access controls).

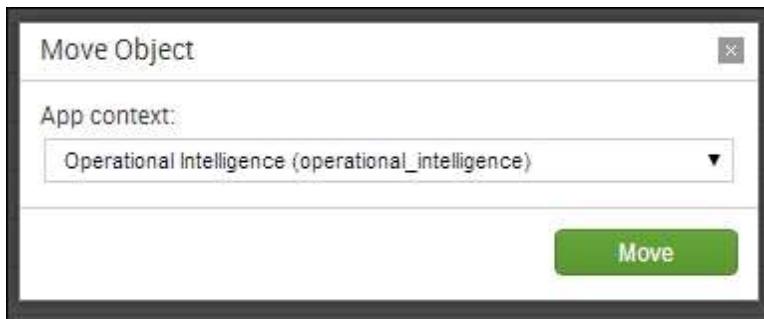
4. Click on the **Views** section.

The screenshot shows the 'User interface' page. At the top, it says 'Create and edit views, dashboards, and navigation menus.' Below this is a list of options: 'Time ranges', 'Views' (which is highlighted with a red oval), 'View PDF scheduling', 'Navigation menus', 'Prebuilt panels', and 'Bulletin messages'.

- In the **App Context** drop-down, select **Searching & Reporting (search)** or whatever application you were in when creating the dashboards in the previous session.



- Locate the `website_monitoring` dashboard row in the list of views and click on the **Move** link to the right of the row.
- In the **Move Object** popup, select the **Operational Intelligence (operational_intelligence)** application that was created earlier and then click on the **Move** button.



- A message bar will then be displayed at the top of the screen to confirm that the dashboard was moved successfully.
- Repeat from step 5 to move the `product_monitoring` dashboard as well.

Successfully moved 'product_monitoring' to 'operational_intelligence'					
View name	Owner	App	Sharing	Status	Actions
product_monitoring	admin	operational_intelligence	App Permissions	Enabled	Open Clone Move Delete
website_monitoring	admin	operational_intelligence	App Permissions	Enabled	Open Clone Move Delete

- After the **Website Monitoring and Product Monitoring** dashboards have been moved, we next want to move all the reports we created in the previous recipes, as these power the dashboards and provide operational intelligence insight. From

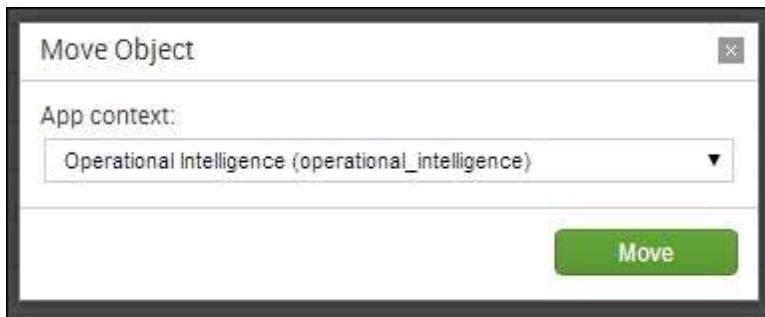
the top menu, select **Settings**, but this time, select **Searches, Reports, and Alerts**.

11. Select the **Search & Reporting (search)** context and filter by [cp0*](#) to view the searches (reports) created in Session 3-2, Diving into Data – Search and Report and Session 3-3, Dashboards and Visualizations – Make Data Shine. Click on the **Move** link of the first [cp0*](#) search in the list.

The screenshot shows the Splunk search interface. The 'App context' dropdown at the top is set to 'Search & Reporting (search)'. A red circle highlights this selection. Below it, a search bar also has 'cp0*' entered. A red arrow points from the 'cp0*' entry in the search bar to the 'cp0*' entry in the 'App context' dropdown. The main area displays a table of search results. The first search result, 'cp0*_application_db_connections', has a 'Move' link highlighted with a red circle. A red arrow points from this link to the 'Move' button in the 'Move Object' dialog box shown below.

Search name #	RSS #	Scheduled #	Display #	Owner #	App #	Alerts #	Sharing #	Status #	Actions
cp0*_application_db_connections	None	None	admin	search	0	Private Permissions	Enabled Disable	Run Advanced edit Clone Move Delete	
cp0*_application_memory	None	None	admin	search	0	Private Permissions	Enabled Disable	Run Advanced edit Clone Move Delete	

12. Select to move the object to the **Operational Intelligence (operational_intelligence)** application and click on the **Move** button.

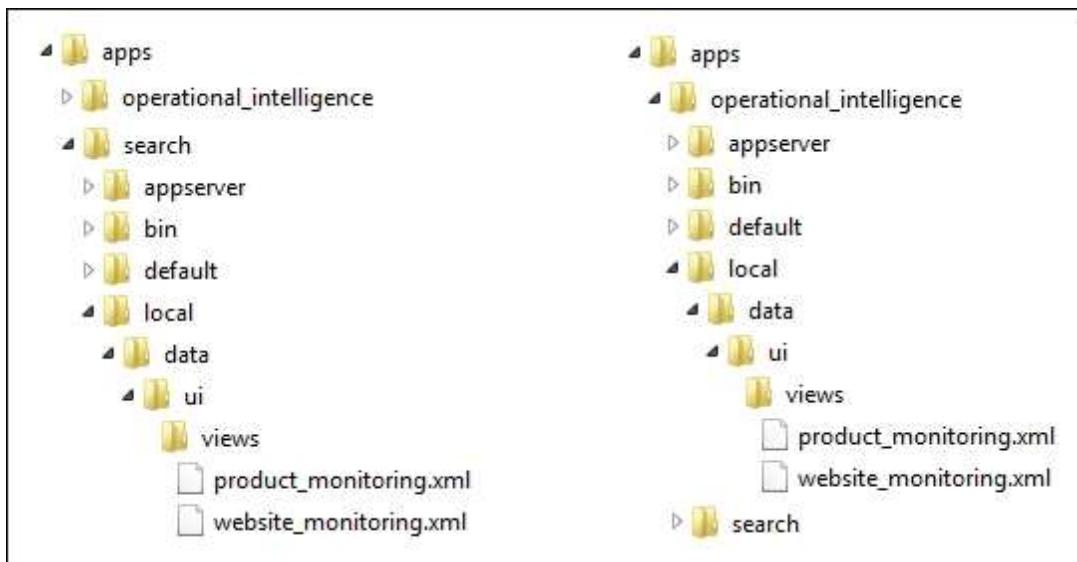


13. A message bar will then be displayed at the top of the screen to confirm that the dashboard was moved successfully.
14. Select the **Search & Reporting (search)** context and repeat from step 11 to move all the other searches over to the new Operational Intelligence application—this seems like a lot but will not take you long!

All the dashboards and reports are now moved to your new Operational Intelligence application.

How it works...

In the previous recipe, we revealed how the Splunk apps are essentially just collections of directories and files. Dashboards are XML files found within the `$SPLUNK_HOME/etc/apps` directory structure. When moving a dashboard from one app to another, Splunk essentially just moves the underlying file from a directory inside one app to a directory in the other app. In this recipe, you moved the dashboards from the **Search & Reporting** app to the **Operational Intelligence** app, as represented in the following screenshot:



As visualizations on the dashboards leverage the underlying saved searches (or reports), you also moved these reports to the new app so that the dashboards maintain permission to access them. Rather than moving the saved searches, we could have changed the permissions of each search to **Global** so that they could be seen from all the other apps in Splunk. However, the other reason we moved the reports was to keep everything contained within a single Operational Intelligence application, which we will continue to build out going forward.

TIP

It is best practice to avoid setting permissions to **Global** for reports and dashboards, as this makes them available to all the other applications when they most likely do not need to be. Additionally, setting global permissions can make

things a little messy from a housekeeping perspective and crowd the lists of reports and views that belong to specific applications. The exception to this rule might be for knowledge objects such as tags, event types, macros, and lookups, which often have advantages in being available across all applications.

There's more...

As you went through this recipe, you probably noticed that the dashboards had application-level permissions but the reports had private-level permissions. The reports are private as this is the default setting in Splunk when they are created. This private-level permission restricts access to your user account and admin users. In order to make the reports available to the other users of your application, you need to change the permissions of the reports to **Shared in App**, as we did when adjusting the permissions of reports.

CHANGING PERMISSIONS OF SAVED REPORTS

Changing the sharing permission levels of your reports from the default **Private to App** is relatively straightforward:

1. Ensure that you are in your newly created Operational Intelligence application.
2. Select the **Reports** menu item to see the list of reports.
3. Click on **Edit** next to the report you wish to change the permissions for. Then, click on **Edit Permissions** from the drop-down list.

The screenshot shows the Splunk interface with the title bar "splunk > App: Operational Intelligence". Below the title bar, there is a navigation bar with links for Search, Pivot, Reports, Alerts, and Dashboards. The "Reports" link is highlighted with a red circle. The main content area is titled "Reports" and displays a table of four reports. The table has columns for Title, Actions, Owner, App, Sharing, and Embedding. The "Actions" column contains links like "Open in Search", "Edit", "Edit Description", "Edit Schedule", "Edit Acceleration", "Clone", "Embed", and "Delete". The "Edit" link for the first report is circled in red. A red arrow points from this circled "Edit" link to a context menu that appears when the link is clicked. The context menu includes options: "Edit", "Edit Description", "Edit Schedule", "Edit Acceleration", "Clone", "Embed", and "Delete". The "Edit Permissions" option is also circled in red within this menu.

Title	Actions	Owner	App	Sharing	Embedding
cp02_application_db_connections	Open in Search Edit ▾	admin	operational_intelligence	Private	Disabled
cp02_application_memory	Open ▾ Edit Description	admin	operational_intelligence	Private	Disabled
cp02_application_performance	Open ▾ Edit Schedule	admin	operational_intelligence	Private	Disabled
cp02_most_accessed_webpages	Open ▾ Edit Acceleration	admin	operational_intelligence	Private	Disabled

4. An **Edit Permissions** pop-up box will appear. In the **Display for** section, change from **Owner** to **App** and then click on **Save**.
5. The box will close and you will see that the sharing permissions in the table now display **App** for the specific report. This report will now be available to all the users of your application.

Organizing the dashboards more efficiently

In this recipe, you will learn how to use Splunk's dashboard editor to use more efficient visualizations and organize the dashboards more efficiently. This feature was introduced in Splunk 6 and enhanced even further in more recent versions.

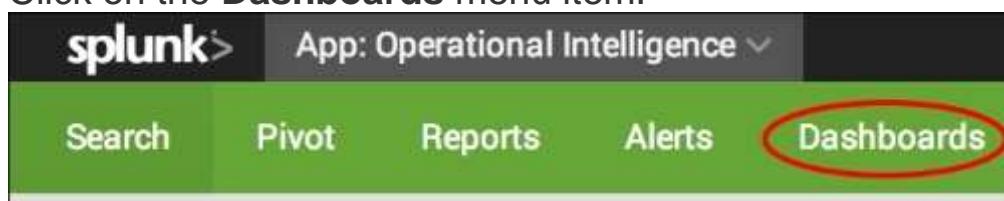
Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#), and you should have completed the earlier recipes in this session. You should also be familiar with navigating the Splunk user interface.

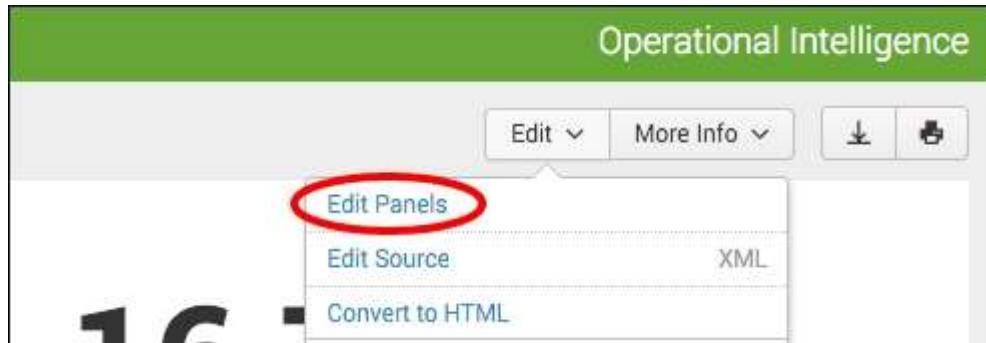
How to do it...

Follow these steps to organize the dashboards more efficiently:

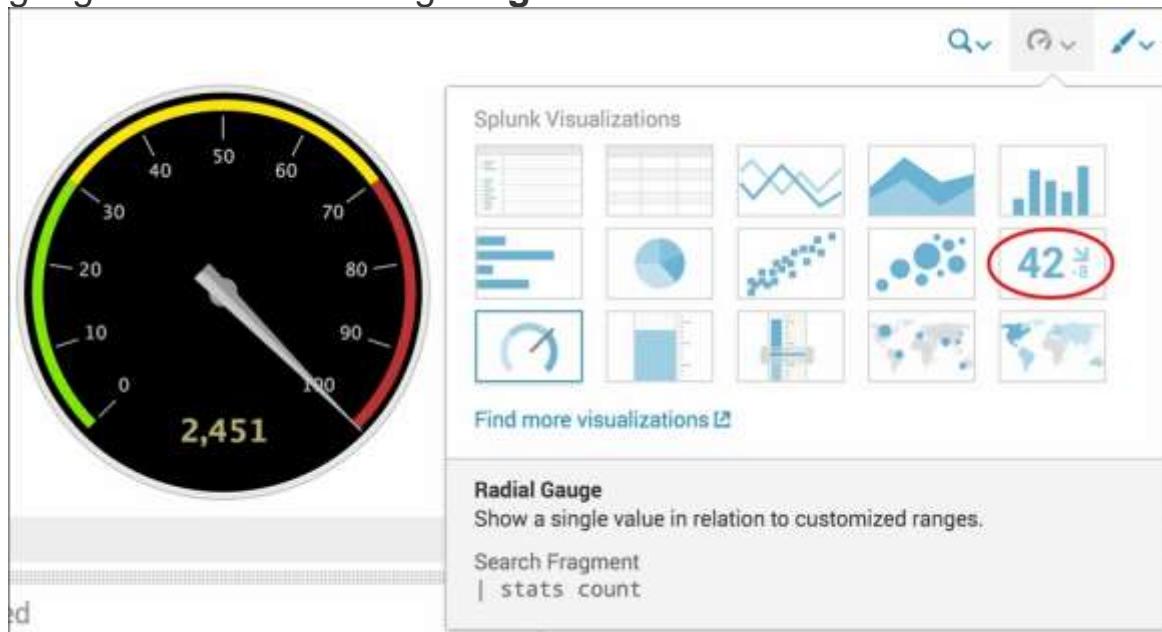
1. Log in to your Splunk server.
2. Select the **Operational Intelligence** application.
3. Click on the **Dashboards** menu item.



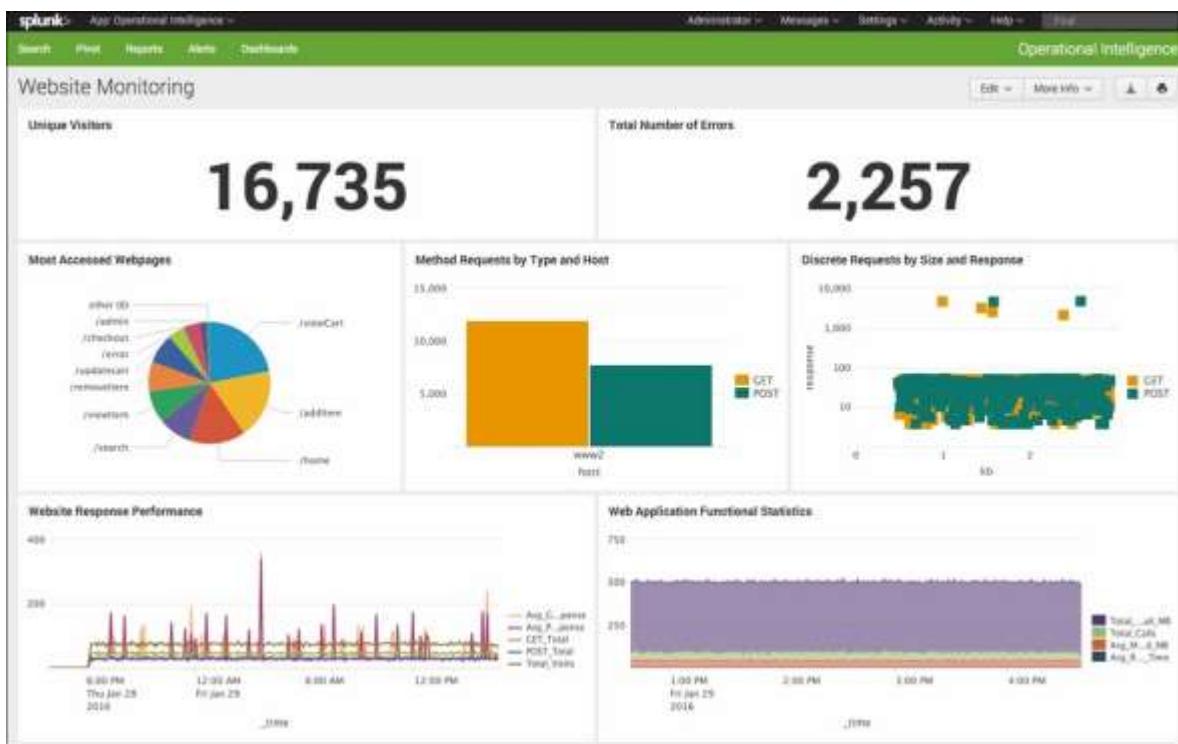
4. You should see the **Product Monitoring** and **Website Monitoring** dashboards that we moved into the **Operational Intelligence** app in the previous recipe. Select the **Website Monitoring** dashboard and it will be displayed.
5. You will note that there are several visualizations on the dashboard; in Splunk, they are known as panels. Select **Edit Panels** from the **Edit** menu.



6. In the **Total Number of Errors** panel, change **Radial Gauge** to **Single Value** visualization by clicking on the radial gauge icon and selecting **Single Value**.



7. Move the other panels around until your dashboard resembles the layout in the following screenshot. Note that we have our single value panels at the top, our various charts in the middle, and then, our time series-based charts at the bottom.



- Click on the **Done** button when complete. I think you will agree that this dashboard looks a lot better than the earlier one! Everything fits on the screen for the most part, and it is much easier on the eye.

How it works...

In this recipe, we started to experience the power of Splunk's amazing dashboard editor. The dashboard editor provides a nice usable interface that essentially shields the user from what is happening underneath the covers. When we edit the dashboard panels and the visualizations in this manner, Splunk actually writes the required [SimpleXML](#) code into the respective view's XML file on your behalf. When you click on **Done**, the file is essentially saved with the newly written XML under the covers. If you are hungry for more, don't worry; we are just getting started with the editor. There is plenty more to come later in this session!

NOTE

Dashboards in Splunk are also called Views. In the management interface and in the backend, they are commonly known as Views, but in the application menu, they are called Dashboards. We may use the terms interchangeably in this book.

There's more...

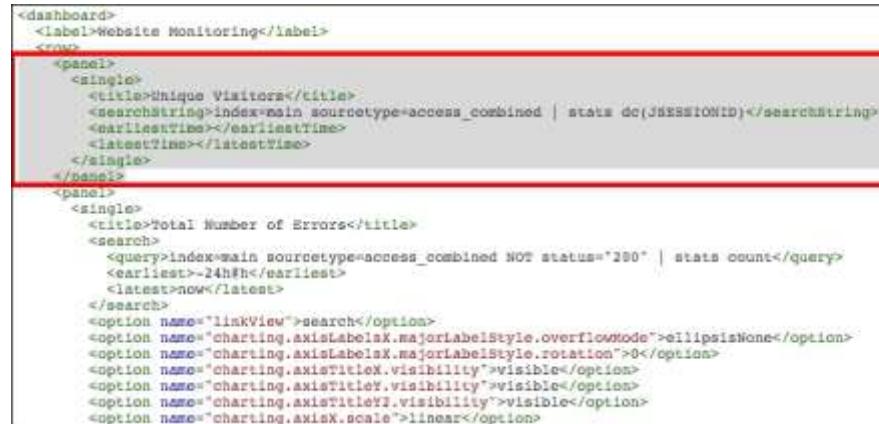
Instead of using the dashboard editor, you can edit the [SimpleXML](#) directly.

MODIFYING THE SIMPLE XML DIRECTLY

Let's take a look at the [SimpleXML](#) that is behind the **Website Monitoring** dashboard. Ensure that the dashboard is displayed on the screen. Then, click on the **Edit** button as you did earlier, but instead of clicking on **Edit Panels**, click on **Edit Source**. The underlying [SimpleXML](#) source code will now be displayed.

Dashboards in [SimpleXML](#) consist of rows, panels, and visualization elements. Dashboards can have many rows (`<row></row>`), but around three rows is advisable. Within each row, you can have multiple panels (`<panel></panel>`), and within each panel, you can have multiple visualization elements (for example, `<chart></chart>` for a chart). On the **Website Monitoring** dashboard, you should see three row elements and multiple panels with a single dashboard element on each panel.

We can edit the [SimpleXML](#) code directly to swap the single elements around on the top row of the dashboard. Simply select the first panel group (The **Unique Visitors** panel XML):



```
<dashboard>
  <label>Website Monitoring</label>
  <rows>
    <panel>
      <single>
        <title>Unique Visitors</title>
        <searchString>index=main sourcetype=access_combined | stats dc(JSESSIONID)</searchString>
        <earliestTime></earliestTime>
        <latestTime></latestTime>
      </single>
    </panel>
    <panel>
      <single>
        <title>Total Number of Errors</title>
        <search>
          <query>index=main sourcetype=access_combined NOT status="200" | stats count</query>
          <earliest>-24h</earliest>
          <latest>now</latest>
        </search>
        <option name="linkView">search</option>
        <option name="charting.axisLabelsX.majorLabelStyle.overflowMode">ellipsisNone</option>
        <option name="charting.axisLabelsX.majorLabelStyle.rotation">0</option>
        <option name="charting.axisTitleX.visibility">visible</option>
        <option name="charting.axisTitleY.visibility">visible</option>
        <option name="charting.axisTitleZ.visibility">visible</option>
        <option name="charting.axisX.scale">linear</option>
      </single>
    </panel>
  </rows>
</dashboard>
```

Next, move it below the second panel group (The **Total Number of Errors** panel XML).

```

<option name="charting.chart.stackMode">default</option>
<option name="charting.chart.style">shiny</option>
<option name="charting.drilldown">all</option>
<option name="charting.layout.splitSeries">0</option>
<option name="charting.layout.allowIndependentYRanges">0</option>
<option name="charting.legend.overflowMode">ellipsisMiddle</option>
<option name="charting.legend.placement">right</option>
<option name="colorBy">value</option>
<option name="colorMode">none</option>
<option name="numberPrecision">0</option>
<option name="showSparkline">1</option>
<option name="showTrendIndicator">1</option>
<option name="trendColorInterpretation">standard</option>
<option name="trendDisplayMode">absolute</option>
<option name="useColors">0</option>
<option name="useThousandsSeparators">1</option>
<option name="drilldown">none</option>
</single>
</panel>
<panel>
<single>
<title>Unique Visitors</title>
<searchString>index=main sourcetype=access_combined | stats dc(JSESSIONID)</searchString>
<earliestTime></earliestTime>
<latestTime></latestTime>
</single>
</panel>
</row>

```

Then, once done, click on the **Save** button. This is an extremely simple example, but you should start to see how we can edit the code directly rather than use the dashboard editor.

Note that there might be some `<option>` data within the panel groups that we have removed in the screenshots to simplify things. However, ensure that you move everything within the panel group.

Familiarizing yourself with Simple XML and learning how to tweak it manually will provide more functionality and probably make the process of creating a dashboard a lot more efficient.

TIP

A great way to learn `SimpleXML` is to modify something using the Splunk dashboard editor and then select to view the code to see what happened in the underlying `SimpleXML` code. Splunk also has a great `SimpleXML` reference that allows quick access to many of the key `SimpleXML` elements. Visit <http://docs.splunk.com/Documentation/Splunk/latest/Viz/PanelreferenceforSimpleXML> for more information.

Dynamically drilling down on activity reports

When viewing a dashboard in Splunk, there is usually a very high probability that you will look at a chart or report and want to know more details about the information that you are looking at.

Splunk dashboards can be configured to let the user drill down into more details. By linking the results or data points to an underlying dashboard or report, information about what the user clicked on can provide them with the next level of detail or the next step in the process they are following.

This recipe will show you how you can configure reports to drill down into subsequent searches and other dashboards so that you can link them together into a workflow that gets the user to the data they are interested in seeing within your Operational Intelligence application.

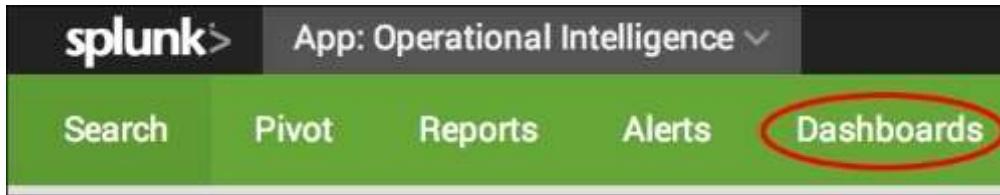
Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#), and you should have completed the earlier recipes in this session. You should also be familiar with navigating the Splunk user interface.

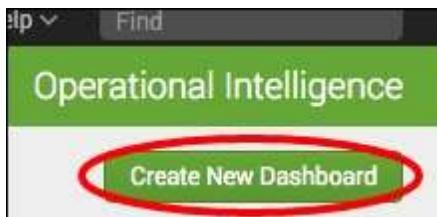
How to do it...

Follow these steps to configure a dashboard report with row drilldown capabilities:

1. Log in to your Splunk server.
2. Select the **Operational Intelligence** application.
3. Click on the **Dashboards** menu.



4. Click on the **Create New Dashboard** button.



5. Name the dashboard **visitor Monitoring** and set the **Permissions** field to **Shared in App**.

A screenshot of a form for creating a new dashboard. The form fields are:

- Title: Visitor Monitoring
- ID?: visitor_monitoring
Can only contain letters, numbers and underscores.
- Description: optional
- Permissions: Private (radio button) Shared in App (radio button)

6. Click on **Create Dashboard**.

7. When the empty dashboard is displayed, click on the **Add Panel** button.

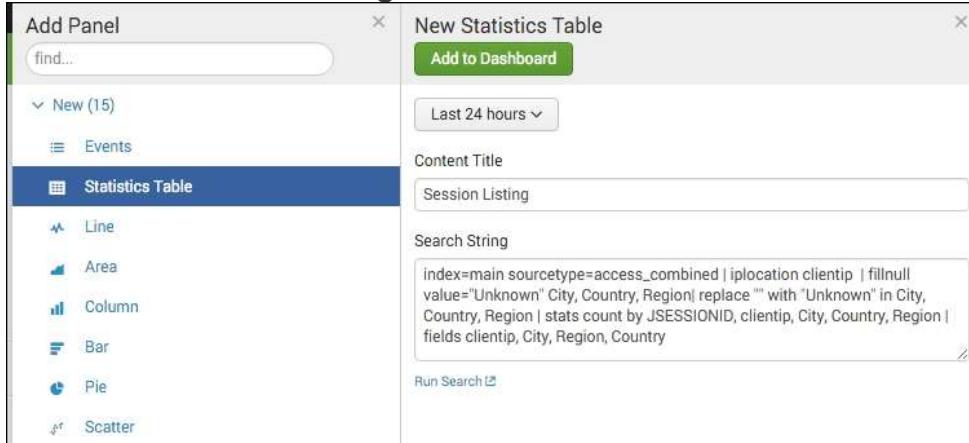


8. From the fly-out panel, expand the **New** section then click on **Statistics Table**.
9. Set the **Content Title** panel to **Session Listing**.
10. Set the **Search String** field to the following:

```
index=main sourcetype=access_combined | iplocation clientip |  
fillnull value="Unknown" City, Country, Region| replace "" with  
"Unknown" in City, Country, Region | stats count by JSESSIONID,
```

```
clientip, City, Country, Region | fields clientip, City,  
Region, Country
```

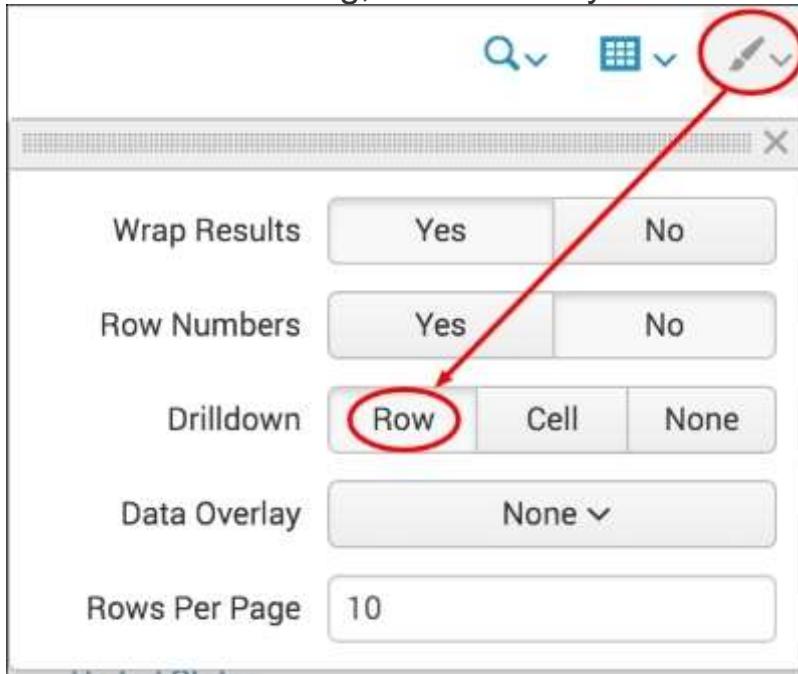
11. Set the time range to **Last 24 hours**.



12. Click on the **Add to Dashboard** button.

13. Click anywhere on the dashboard to make the fly-out panel disappear.

14. Click on the panel edit icon, select the **Row** option for the **Drilldown** setting, and click anywhere on the page.



15. Click on the **Done** button to finish editing the dashboard.



16. Click on a row in the dashboard table, and Splunk will now drill down to the search screen and execute a search that is filtered by the `clientip` value in the row you selected to drill down on.

How it works...

The drilldown feature of dashboards can be utilized to get your users to the next set of data they need. When they click on a table entry or a part of a chart, they set off a search that can drill down into more details about the item they clicked on. The behavior of the drilldown is controlled by the configuration of the panel in the [SimpleXML](#) code but also has a few options displayed by the dashboard editor.

When displaying a table of results, there are three options that can be chosen from:

Option	Description
<code>Row</code>	When a row is clicked on, the search that is launched by the drilldown is based on the <code>x-axis</code> value, which is the first column in the row.
<code>Cell</code>	When a particular cell is clicked on, the search that is launched by the drilldown is based on both the <code>x-axis</code> and <code>y-axis</code> values represented by that cell.
<code>None</code>	The drilldown functionality is disabled. When a user clicks on the table, the page will not change.

When displaying a chart, there are two options for the drilldown behavior that can be chosen from:

Option	Description
<code>On</code>	When a row is clicked on, the search that is launched by the drilldown is based on the values of the portion of that chart.

Option	Description
Off	The drilldown functionality is disabled. When a user clicks on the table, the page will not change.

When the drilldown search is started after the table or chart is clicked on, it is generally derived by taking the original search, backing off the final transforming commands, and then adding the values that were selected, depending on the drilldown setting.

TIP

When a new panel item is added, such as a chart, table, or map, the default drilldown is always turned on by default.

There's more...

The drilldown options can be customized and provide many different options to control the behavior when dashboards are clicked on.

DISABLING THE DRILLDOWN FEATURE IN TABLES AND CHARTS

To disable the drilldown feature, you can specify the **None** option in the **Drilldown** setting of the edit panel form or add/modify the following [SimpleXML](#) option to the panel source:

```
<option name="drilldown">none</option>
```

TIP

A full reference of drilldown options can be found in the Splunk documentation at http://docs.splunk.com/Documentation/Splunk/latest/Viz/PanelreferenceforSimplifiedXML#Panel_visualization_elements.

Creating a form for searching web activity

Presenting users with dashboards is a great way to visualize data, as we have seen. However, often people like to *slice n dice* data in many different ways, and to do this, we need to make our dashboards more interactive. We can do this using the dashboard forms functionality of Splunk, which allows the users to filter the dashboard visualizations and data based upon the criteria that are important to them.

This recipe will build on the tabular **Visitor Monitoring** dashboard you created in the previous recipe to allow for granular filtering of the tabulated results.

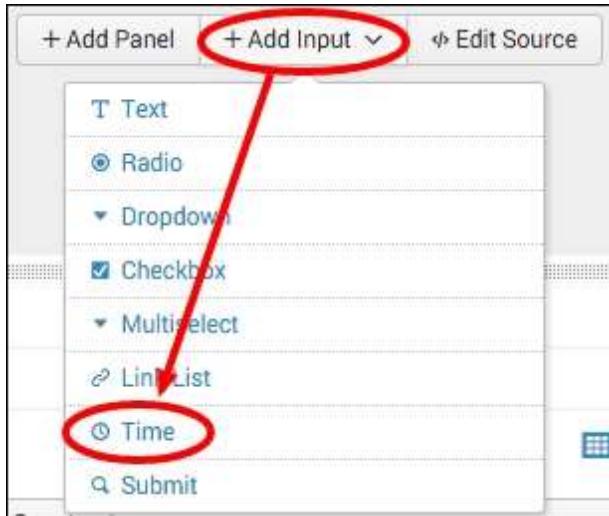
Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#), and you should have completed the earlier recipes in this session. You should also be familiar with navigating the Splunk user interface.

How to do it...

Follow these steps to create a form to filter data on a dashboard:

1. Log in to your Splunk server.
2. Select the **Operational Intelligence** application.
3. Click on the **Dashboards** menu item.
4. Select to view the **Visitor Monitoring** dashboard we created in the previous recipe.
5. Once loaded, click on the **Edit** drop-down and then on **Edit Panels**.
6. Click on **Add Input** and then select **Time**.



7. Click on **Add Input** again, and this time select **Text**.
8. A new text input named **field2** will appear. Above the text input you will see a little pencil icon. Click on the pencil icon to edit the input. A popup will be displayed.
9. Complete the box with the values in the following table:

Label	IP
Search on Change	Checked
Token	ip
Default	*
Token Suffix	*

10. Then, click on **Apply**.

ID	General
T Text	<input checked="" type="radio"/> Radio <input type="checkbox"/> Dropdown <input checked="" type="checkbox"/> Checkbox <input type="checkbox"/> Multiselect <input type="checkbox"/> Link List <input checked="" type="radio"/> Time <input type="checkbox"/> Submit
General Label: IP Search on Change: <input checked="" type="checkbox"/> Token Options Token?: ip Default?: * Initial Value? Token Prefix? Token Suffix?: *	
<input type="button" value="Cancel"/> <input type="button" value="Apply"/>	

11. The box will disappear and you will see that the input is now titled IP.

12. Repeat from step 7 to add and edit the three other textbox fields (one at a time) using the following values for each:

- **field3:**

Label	City
Search on Change	Checked
Token	city
Default	*
Token Suffix	*

- **field4:**

Label	Region
Search on Change	Checked
Token	region
Default	*
Token Suffix	*

- **field5:**

Label	Country
Search on Change	Checked
Token	country
Default	*
Token Suffix	*

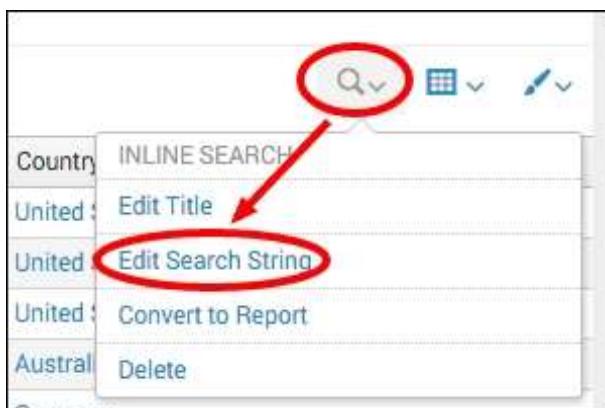
13. Once complete, you should have a total of five fields. Let's now do a bit of rearrangement. Move the **Time input** field to the far right-hand side so that it is the last input field at the top. Additionally, check the **Autorun dashboard** checkbox on the far right-hand side.

The screenshot shows a top navigation bar with tabs: Search, Pivot, Reports, Alerts, and Dashboards. Below this, the title "Edit: Visitor Monitoring" is displayed. Underneath the title are four search input fields: "IP" (with a pencil icon), "City" (with a pencil icon), "Region" (with a pencil icon), and "Country" (with a pencil icon). Each field has a red asterisk (*) in its search box.

14. Click on the pencil icon above the time input and change the default time range from **All Time** to **Last 24 Hours**. Then, click on **Apply**.

This screenshot shows a configuration dialog for a "Time" field. On the left is a sidebar with options: Text, Radio, Dropdown, Checkbox, Multiselect, Link List, and Time (which is selected and highlighted in blue). The main panel has sections for "General" (Label input field), "Search on Change" (checkbox checked), "Token Options" (Token? input field containing "field1"), and "Default" (a dropdown menu currently set to "Last 24 hours"). A red oval highlights the "Last 24 hours" option in the dropdown. At the bottom are "Cancel" and "Apply" buttons.

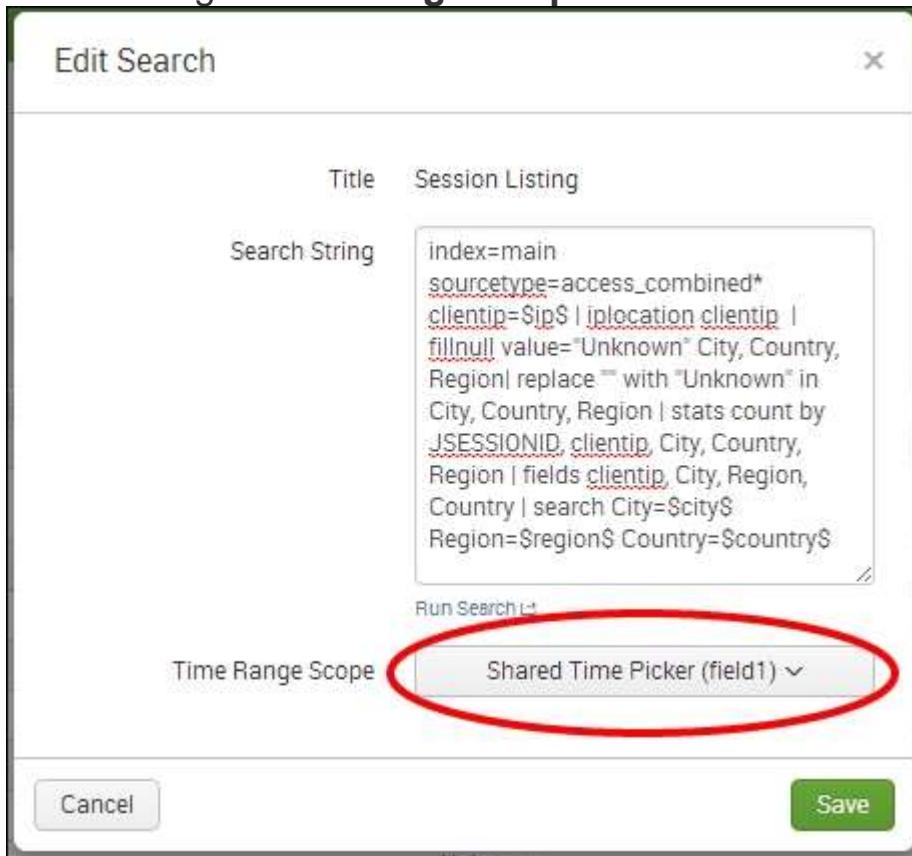
15. Next, click on **Done** on the top right-hand side of the screen to finish editing the form. You should see all your fields nicely labeled across the top with wildcard asterisks (*) in each textbox.
16. Next, we need to link the new fields we just created with the table. Click on **Edit** and then on **Edit Panels**.
17. Select **Edit Search String** in the panel with the table. A pop-up box will appear with the current search string.



18. Replace the existing search string with the following search string. The modifications to the search have been highlighted:

```
index=main sourcetype=access_combined clientip="$ip$" |  
iplocation clientip | fillnull value="Unknown" City, Country,  
Region| replace "" with "Unknown" in City, Country, Region |  
stats count by JSESSIONID, clientip, City, Country, Region |  
fields clientip, City, Region, Country | search City="$city$"  
Region="$region$" Country="$country$"
```

19. Change Time Range Scope to Shared Time Picker (field1).



20. Click on **Save** and then on **Done**.

21. Refresh/reload the dashboard in your browser.
22. That's it! You now have a nice form-driven table. Try testing it out. For example, if you want to filter by all the IP addresses that begin with 134, simply enter `134` into the IP textbox and press *Enter*.

How it works...

In this recipe, we only used the GUI editor, which means that Splunk changed the underlying Simple XML for us. As soon as we added the first input field, Splunk changed the opening Simple XML `<dashboard>` element to a `<form>` element behind the scenes. Each of the five inputs we added is contained within a `<fieldset>` element. For each of the inputs, Splunk creates an `<input>` element, and each input type can have a number of fields: some optional and some required. One of the key fields for each type is the **Token** field, the values of which are then used by searches on the dashboard. We assigned a token name to each input, such as `ip`, `city`, and `country`. Other fields that we populated for the text inputs were the **Default** and **Suffix** field values of `*`. This tells Splunk to search for everything `(*)` by default if nothing is entered into the textbox and to add a wildcard `(*)` suffix to everything entered into the textbox. This means that if we were to search for a city using a value of `Tor`, Splunk would search for all the cities that begin with `Tor` (`Tor*`), such as Toronto. We also checked the **Search on Change** box, which forces a rerun of any searches in the dashboard should we change a value of the input. After completing the editing of the inputs, we selected to autorun the dashboard, which adds `autoRun="true"` in the `<fieldset>` element of the Simple XML and ensures that the dashboard runs as soon as it is loaded with the default values rather than waiting for something to be submitted in the form.

Once we built the form inputs and configured them appropriately, we needed to tell the searches that power the dashboard visualizations to use the tokens from each of the form inputs. The **Token** field for each input contains the value for that input. We edited the search for the table on the dashboard and added additional search criteria to force Splunk to search based upon these tokens. Token names must be encapsulated by `$` signs, so our `ip` token is entered into the search as `ip` and our `country` token is entered as `$country$`. We also told our search to use

the `Shared Time Picker` input rather than its own time range. This allows us to search using the time picker input we added to the form.

The end result is that anything entered into the form inputs is encapsulated into the respective tokens, and the values of these tokens are then passed to the search that powers the table in the dashboard. If we change the value of an input then the value of the input's token in the search changes, the search immediately reruns, and the search results in the table change accordingly.

There's more...

In this recipe, we began to scratch the surface of form building, using only textbox inputs and the time picker. Later in this book, we will leverage the drop-down input, and you will learn how to prepopulate the drop-down values as a result of a search and learn how to filter the drop-down values as a result of other drop-down selections.

ADDING A SUBMIT BUTTON TO YOUR FORM

In this recipe, you probably noticed that there was no **Submit** button. The reason for this was primarily because no **Submit** button was needed. We selected to autorun the dashboard and selected **Search on Change** for each input. However, there are times when you might not want the form to run as soon as something is changed; perhaps, you want to modify multiple inputs and then search. Additionally, many users like the reassurance of a **Submit** button, as it is commonly used on forms across websites and applications.

Adding a **Submit** button is extremely simple. When the dashboard is in the editing mode, simply click on the **Add Input** drop-down and select **Submit**. You will note that a green **Submit** button now appears on the form. If you now edit the text inputs and uncheck the **Search on Change** checkbox for each of them, the form will only be submitted when someone clicks on the **Submit** button.

Linking web page activity reports to the form

Form searches in Splunk do not need to be limited to displaying events and table-driven data. Rich visualizations can also be linked to forms and be updated when the forms are submitted.

This recipe will show you how you can extend a form to include charts and other visualizations that can be driven by the form created to show visitor traffic and location data.

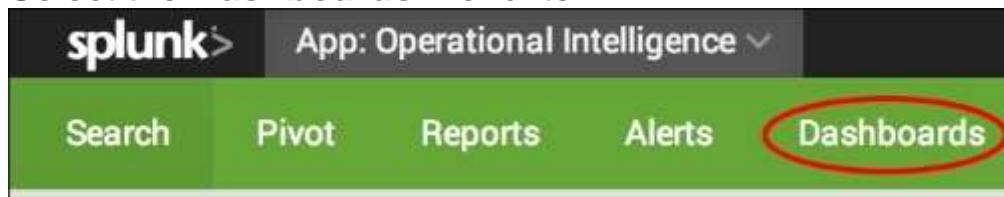
Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#), and you should have completed the earlier recipes in this session. You should also be familiar with navigating the Splunk user interface.

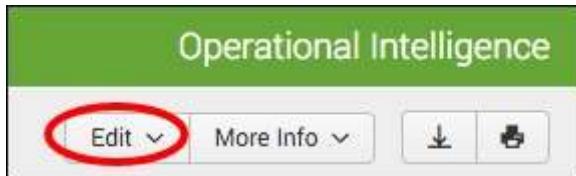
How to do it...

Follow these steps to add a web page activity chart and link it to a form:

1. Log in to your Splunk server.
2. Select the default **Operational Intelligence** application.
3. Select the **Dashboards** menu item.



4. Select the **Visitor Monitoring** dashboard.
5. Click on the **Edit** button and then on **Edit Panels**.



6. Click on the **Add Panel** button.



7. From the fly-out panel, expand the **New** section and then click on **Line**.
8. Set the **Content Title** panel to **Session Over Time**.
9. Set the **Search String** field to the following:

```
index=main sourcetype=access_combined clientip="$ip$" |  
iplocation clientip | fillnull value="Unknown" City, Country,  
Region| replace "" with "Unknown" in City, Country, Region |  
search City="$city$" Region="$region$" Country="$country$" |  
timechart dc(JSESSIONID)
```

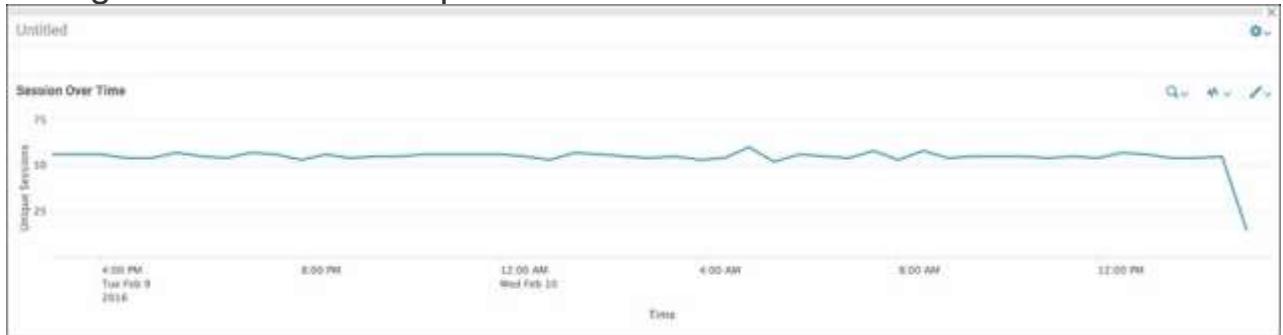
10. Set the **Time Range Scope** field to **Shared Time Picker (field1)**.

The screenshot shows two overlapping panels. The left panel is titled 'Add Panel' and has a sidebar with 'New (15)' expanded, showing options like 'Events', 'Statistics Table', 'Line' (which is selected and highlighted in blue), 'Area', 'Column', 'Bar', 'Pie', and 'Scatter'. The right panel is titled 'New Line' and contains fields for 'Time Range Scope' (set to 'Shared Time Picker (field1)'), 'Content Title' (set to 'Session Over Time'), and 'Search String' (containing the provided ELK search query). A green 'Add to Dashboard' button is visible at the top of the right panel.

11. Click on **Add to Dashboard**.
12. Click anywhere on the dashboard to make the fly-out panel disappear.
13. Click on the edit panel icon in the panel we just added to the dashboard.



14. Update the **X-Axis** label with **Custom Title** set to **Time**.
15. Update the **Y-Axis** label with **Custom Title** set to **Unique Sessions**.
16. Set the **Legend** option to **None**.
17. Click on **Apply**. The pop-up box will disappear with the changes reflected on the panel.



18. Next, click on **Done** to finish editing the dashboard.



19. Filter by an IP of 134 or similar again, and you should see that the chart panel also changes along with the table panel.

How it works...

Adding a chart to the dashboard works in a manner very similar to the way in which the original form was created. You can utilize the field variables defined in the form in the inline search that is used for the chart. Splunk will set them when the form is submitted. The panel can also utilize the time range that was used in the form or contain a separate time range drop-down.

By building a form and several different charts and tables, you can build a very useful form-driven dashboard. One of the great uses of a form-driven dashboard is for investigative purposes. In this example, you can take any of the fields and, for instance, see all sessions that are coming from a particular country and then see the level of activity over the time period you are interested in.

There's more...

Additional customizations can be added to the charts in order to give them more meaning.

ADDING AN OVERLAY TO THE SESSIONS OVER TIME CHART

You can have Splunk overlay a field value on top of your existing chart to provide trendlines and so on. Add the following line to the end of the inline search used for the **Sessions Over Time** search:

```
| eventstats avg(dc(JSESSIONID)) as average | eval  
average=round(average,0)
```

Then, add the following line to the Simple XML of the panel:

```
<option name="charting.chart.overlayFields">average</option>
```

```
<row>  
  <panel>  
    <chart>  
      <title>Session Over Time</title>  
      <search>  
        <query>index=main sourcetype=access_combined clientip="$ip$" | iplocation clientip | fi  
average=round(average,0)</query>  
        <earliest>$field1.earliest$</earliest>  
        <latest>$field1.latest$</latest>  
      </search>  
      <option name="charting.chart.overlayFields">average</option>  
      <option name="charting.chart">line</option>  
      <option name="charting.axisLabelsX.majorLabelStyle.overflowMode">ellipsisNone</option>  
      <option name="charting.axisLabelsX.majorLabelStyle.rotation">0</option>  
    </chart>  
  </panel>  
</row>
```

It will then add a line that charts the average of the session count over top of the actual values. The chart overlay functionality can also be added from the panel editor under the edit panel icon.



Displaying a geographical map of visitors

Operational intelligence doesn't always need to come in the form of pie charts, bar charts, and data tables. With a wide range of operational data being collected from IT systems, there is the opportunity to display this data in ways that can be more meaningful to users or help present it in ways that can be easier to identify trends or anomalies.

One way that always provides great visibility is representing your data using a geographical map. With geolocation data available for many different data types, it becomes very easy to plot them. Using IP addresses from web server logs is a very common use case for this type of visualization. Splunk allows the easy addition of a map to a dashboard with the capability to zoom and update the portion of the map that the user is viewing.

This recipe will show you how you can configure a map panel within a dashboard and link it to a search that contains IP addresses in order to visualize from where in the world the IP traffic is originating.

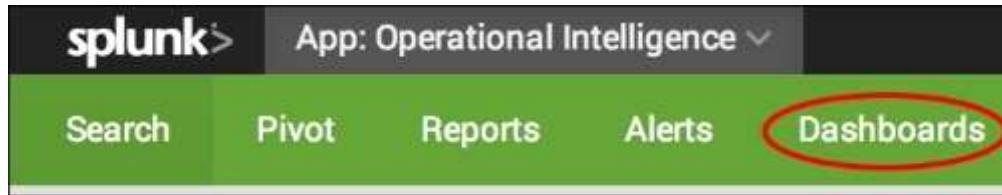
Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#), and you should have completed the earlier recipes in this session. You should also be familiar with navigating the Splunk user interface.

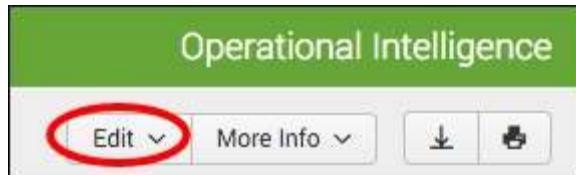
How to do it...

Follow these steps to add a map to your form-driven dashboard:

1. Log in to your Splunk server.
2. Select the **Operational Intelligence** application.
3. Click on the **Dashboards** menu item.



4. Select the **Visitor Monitoring** dashboard.
5. Click on the **Edit** button.



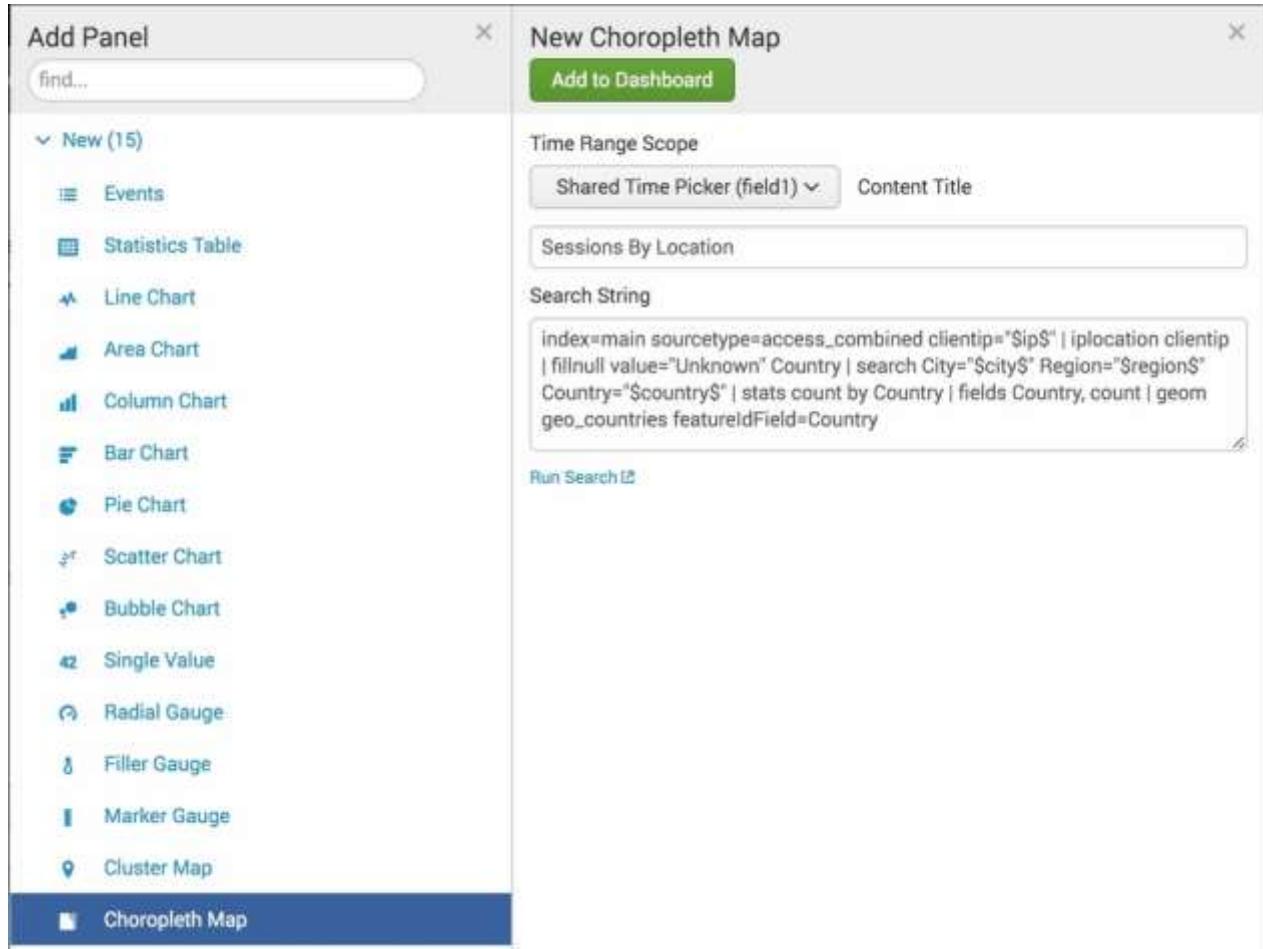
6. Click on the **Add Panel** button.



7. From the fly-out panel, expand the **New** section and then click on **Choropleth Map**.
8. Set the **Content Title** panel to **Sessions By Location**.
9. Set the **Search String** field to the following:

```
index=main sourcetype=access_combined clientip="$ip$" |  
iplocation clientip | fillnull value="Unknown" Country |  
search City="$city$" Region="$region$" Country="$country$" |  
stats count by Country | fields Country, count | geom  
geo_countries featureIdField=Country
```

10. Set the **Time Range Scope** field to **Shared Time Picker (field1)**.



11. Click on the **Add to Dashboard** button.
12. Click anywhere on the dashboard to make the fly-out panel disappear.
13. Click on **Done** to finish editing the dashboard.



14. Filter by an IP of 134 or similar again, and you should now see that the map panel also changes along with the table and chart panels you added earlier.



How it works...

Mapping support has been available since Splunk 4 using a third-party developed application. Since Splunk 6, native map support has been available and can be used easily within your dashboards. It was given a big boost in 6.3 with the addition of geo-lookups and Choropleth maps.

The rendering of the map is done in the same way in which most browser-based maps are generated - using many small images known as tiles that are put together in a grid layout and swapped in and out depending on the zoom level and the visible area being requested. This results in the browser and services not needing to load an entire world's worth of image data into memory. Layers are then rendered on top of the tiles based on either markers or shapes (polygons).

Splunk currently has two mapping types that can be used:

Type	Description
Choropleth	A Choropleth map uses shading to show relative metrics, such as population or election results, for predefined geographic regions.
Marker	Marker maps can plot geographic coordinates as interactive markers. These markers can be configured to represent a metric such as a pie chart with details about that location.

Splunk supports both a native tile server that can be used to serve the actual map images or can be configured to use the external OpenStreetMap service

(<http://www.openstreetmap.org/#map=5/51.500/-0.100>). The native tiles do not have a very granular level of mapping detail but work in situations where there is no external connectivity or there are security reasons for not calling the external service.

In this recipe, the map panel depends on the result of the `geom` command, which looks for the necessary feature ID fields in the search results and adds its own data to that the map can use to render the shapes properly. The `iplocation` command is commonly used to map network traffic-originating locations.

The built-in IP location data within Splunk is provided by Splunk as part of Splunk Enterprise but is not always the most up-to-date data available from the Internet. It's often best practice to purchase a third-party service to get the most accurate and real-time data available, especially when it is used on critical security-monitoring dashboards and searches.

The map panel has many different configuration options that can be used to specify the initial latitude, longitude, and zoom level that should be applied when the map is initially loaded, as well as the minimum and maximum zoom levels. Drilldown in the maps is also supported.

TIP

More details on Mapping in Splunk is available at <http://docs.splunk.com/Documentation/Splunk/latest/Viz/Choroplethmaps>.

A full reference of map drilldown options can be found in the Splunk documentation at http://docs.splunk.com/Documentation/Splunk/latest/Viz/PanelreferenceforSimplifiedXML#Panel_visualization_elements.

There's more...

The map panel option can also be configured in several different ways in Splunk.

ADDING A MAP PANEL USING SIMPLE XML

A map panel can be added directly to a dashboard by adding the following Simple XML when editing the dashboard source:

```
<row><panel>
<map>
    <title>Count by location</title>
    <searchString>index=main sourcetype=access_combined
clientip="$ip$" | iplocation clientip | fillnull
value="Unknown" City, Country, Region| replace "" with
"Unknown" in City, Country, Region | search City="$city$"
Region="$region$" Country="$country$" | geostats
count</searchString>
    <earliestTime>-24h@m</earliestTime>
    <latestTime>now</latestTime>
    <option name="mapping.data.maxClusters">100</option>
    <option name="mapping.drilldown">all</option>
    <option name="mapping.map.center">(0,0)</option>
</map>
</panel></row>
```

MAPPING DIFFERENT DISTRIBUTIONS BY AREA

The `geostats` command takes an aggregation term as its main argument. This term is what is used to render the pie charts that are located on the map. In this recipe, we simply ran `| geostats count`, which is the most commonly used command and simply does a single count. However, you can break out the data by product, and then the pie charts will provide segmented visual information and can be moused over to see the breakdown.

```
MySearch | geostats count by product
```

Scheduling PDF delivery of a dashboard

Getting Operational Intelligence to the users who need it the most can be challenging. They can be users who are not IT savvy, don't have the correct access to the right systems, or are executives about to walk into a client meeting to go over the latest result data.

Sometimes, all a user needs is to have data e-mailed to their inbox every morning so that they can review it on their commute to the office or have an assistant prepare for a morning briefing. Splunk allows the users to schedule a dashboard so that it can be delivered as a PDF document via e-mail to a customizable list of recipients.

This recipe will show you how to schedule the delivery of a dashboard within the Operational Intelligence application as a PDF document to an internal e-mail distribution list.

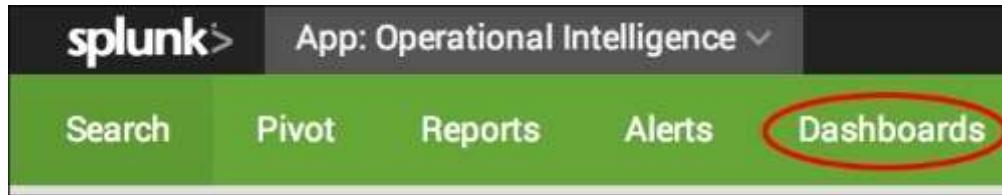
Getting ready

To step through this recipe, you will need a running Splunk Enterprise server, with the sample data loaded from [Session 3-1, Play Time – Getting Data In](#), and you should have completed the earlier recipes in this session. You should also be familiar with navigating the Splunk user interface. You should also have configured your e-mail server to work with Splunk so that Splunk can actually send e-mails to specified addresses.

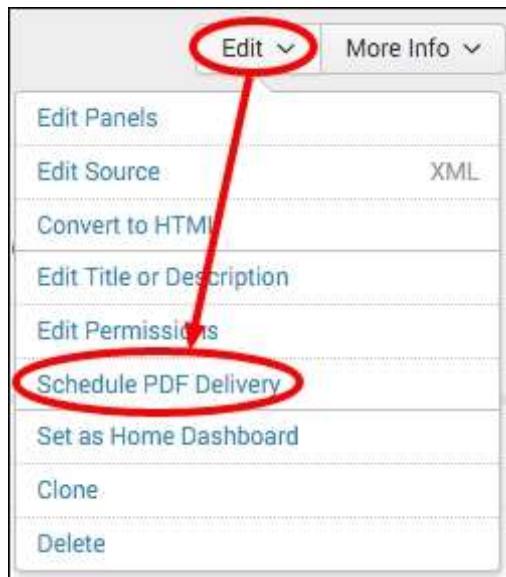
How to do it...

Follow these steps to schedule a PDF delivery of your dashboard:

1. Log in to your Splunk server.
2. Select the **Operational Intelligence** application.
3. Click on the **Dashboards** menu item.



4. From the dashboard listing, select the dashboard you would like to deliver as a PDF document. Only the **Website Monitoring** and **Product Monitoring** dashboards can leverage PDF delivery, as the PDF delivery function is not (currently) compatible with the dashboards driven by form inputs.
5. Once the selected dashboard loads, click on the **Edit** drop-down menu on the top right-hand side of the screen.
6. Click on the **Schedule PDF Delivery** option.



7. On the **Edit PDF Schedule** form, check the **Schedule PDF** box.

A screenshot of a dialog box titled 'Edit PDF Schedule'. It has tabs for 'Dashboard' and 'Product Monitoring'. Under the 'Dashboard' tab, there is a section labeled 'Schedule PDF' with a checkbox. The checkbox is checked and circled with a red oval. At the bottom of the dialog are 'Cancel' and 'Save' buttons.

8. Modify the **Schedule** field to suit your needs. Update the drop-down and select the appropriate schedule type.

Schedule	Run every week ▾		
On	Monday ▾	at	6:00 ▾

9. Enter the list of e-mail addresses you wish to send the PDF to in the **Email To** field, using commas to separate multiple e-mail addresses.
10. Select the priority of the e-mail.
11. Customize the **Subject** field with the content of the message subject you would like the recipients to see.
12. Customize the **Message** field with the content of the message you would like the recipients to see.

Email To	<input type="text"/>
Priority	Normal ▾
Subject	Splunk Dashboard: '\$name\$'
Message	A PDF was generated for \$name\$

13. Update the layout options of the generated PDF by updating the **Paper Size** and **Paper Layout** options.

Paper Size	Letter ▾	
Paper Layout	Portrait	Landscape

14. You can test your PDF and e-mail formatting using the preview options. Click on the **Send Test Email** link to send to the recipients the dashboard as it looks when the link is clicked. Then, click on the **Preview PDF** link to view a version of the PDF as it looks when the link is clicked.

[Send Test Email](#) [Preview PDF](#)

15. Click on the **Save** button, and the PDF delivery of the dashboard is now scheduled.

How it works...

Since Splunk 5 was released, Splunk Enterprise has been natively able to produce PDFs of dashboards and reports. Prior to Version 5, it required a separate add-on app that only worked on Linux servers and required other operating system dependencies. The new integrated PDF features allow quicker and easier access to generate PDFs, either via a schedule and e-mailed or directly from the Web.

There are still some situations that are not able to produce PDFs, such as form-driven dashboards, dashboards created using advanced XML, and Simple XML dashboards that still contain Flash components. There are also some features, such as heat map overlays, that do not render properly.

PDFs are generated when requested by native libraries built into Splunk that render what would normally be output as HTML and encode this into the PDF. It's not an easy feat, as you have to take the page layout and orientation into consideration as the PDF is much more constrained than the browser window.

When delivering a scheduled PDF of a dashboard, you use the same mechanism that scheduled reports and alerts use.

The `sendemail` command is the backbone of the process and allows many different configuration options for the format of the message, including a full range of tokens that can be inserted into the subject and body of the messages that are replaced with job- and schedule-specific details.

TIP

For more information on the configuration options to schedule reports and dashboards, check out <http://docs.splunk.com/Documentation/Splunk/latest/Report/Schedulereports>.