SPLUNK LOGGING WITH .NET

Whitepaper by **Chinmaya Pattanayak**

Table of contents

1. Introduction

1.1. Why Splunk for Log Management

Index Machine Data

Easy to Index and store machine data which is independent of format or location. For example malware analysis information, security logs, change events, data from APIs and message queues and multi-line logs from custom applications etc. can be indexed without having any predefined schema.

Search Correlate and Investigate

Easy to search real-time and historical data using Splunk search interface by using easy-to-learn search commands to filter the search results. We can correlate data based on time, external data, location or joins across multiple data sources. The search assistant offers intelligence based suggestions to help user's building a better search query.

Drill-down Analysis

Using Splunk's tool based search interface and unique field extraction capability to find any value across any field from any data, it makes analyst's life much easier to drill down the issue in seconds to minutes rather than hours or days.

Monitor and Alert

We can make Splunk searches into real-time alerts and auto notifications via email or we can generate a service desk ticket automatically.

Reports and Dashboards

Splunk provides various tools to generate customized charts, reports and dashboards of your real-time data which you can access from desktops or mobile devices.

1.2. Real-Time Enterprise Log Management to Search, Diagnose and Report

Log data is a definitive record of what's happening in every business, organization or agency and it's often an untapped resource when it comes to troubleshooting and supporting broader business objectives.

Splunk provides the industry-leading software to consolidate and index any log and machine data, including structured, unstructured and complex multi-line application logs. You can collect, store, index, search, correlate, visualize, analyze and report on any machine-generated data to identify and resolve operational and security issues in a faster, repeatable and more affordable way. It's an enterprise ready, fully integrated solution for log management data collection, storage and visualization.

Ad hoc queries and reporting across historical data can also be accomplished without third-party reporting software. Splunk software supports log data enrichment by providing flexible access to relational databases, field delimited data in comma-separated value (.CSV) files or to other enterprise data stores such as Hadoop or NoSQL. Splunk software supports a wide range of log management use cases including log consolidation and retention, security, IT operations troubleshooting, application troubleshooting and compliance reporting.

- Index, search and correlate any data for complete insight across your infrastructure
- Drill down and up and pivot across data to quickly find the needle in the haystack
- Turn searches into real-time alerts, reports or dashboards with a few mouse clicks
- Securely make operational data available without requiring access to production systems
- Scale from a single server to global datacenters
- Deploy and search across on premise, hybrid-cloud and private/public-cloud based installations

2. Splunk Logging Considerations in .NET

Splunk logging for .NET enables you to configure logging to HTTP Event Collector in Splunk Enterprise 6.3 and later or Splunk Cloud, as well as UDP or TCP logging of events from within your .NET applications, via a .NET TraceListener or a SLAB event sink.

2.1. Choosing a logging destination

HTTP Event Collector is ideal if you want to log data from your .NET application in any of the following scenarios:

- You want to send events directly to Splunk Enterprise rather than requiring writing to disk and installing a forwarder.
- You want to send data securely to Splunk Enterprise, with the option of an HTTPS connection and a unique token.
- You expect to send data at a high volume and frequency.

Alternately, you can log to a TCP or UDP input either directly or by first logging to a file and then using a Splunk Universal Forwarder to monitor the file and send data any time the file is updated. Doing so gives you the features of the Universal Forwarder, plus added robustness from having persistent files.

Programmatically we can log to a UDP or TCP input directly from our .NET applications. Splunk Logging provides us .NET Libraries to configure and log events and errors to Splunk Servers via UDP or TCP inputs.

TCP vs UDP

UDP: UDP is a connection-less and unreliable transport protocol:

- It doesn't enforce delivery
- It's not encrypted
- There's no accounting for lost datagrams
- Direct UDP input is higher performance than reading files from disk. However, because UDP is a "best effort" protocol, you might not get messages if the network is clogged or has a hiccup. Therefore, using UDP for syslog is not reliable, not guaranteed and not recommended if you are concerned about potential data loss or security, or if you require the data for compliance.

TCP: TCP is connection oriented and reliable

- Data packets are received in orderly fashion
- Chances of data loss is very low
- It is slower than UDP
- TCP should be used when no data loss is a requirement and the data must be correct free from errors.

2.2. Trace Listener and SLAB

Semantic Logging Application Block (SLAB)

- Strongly typed events, more control over format of log messages, hence easy to filter
- Structured messages hence easier to query and analyze log files
- Messages can be logged to multiple destinations such as flat file or database
- We can use custom event source classes that extends EventSource class for logging consistent messages.
- While it is possible to use the EventSource class in the .NET framework (without the Semantic Logging Application Block) and Event Tracing for Windows (ETW) to write log messages from our application, using the Semantic Logging Application Block makes it easier to incorporate the functionality provided by the EventSource class, and makes it easier to manage the logging behavior of the application.

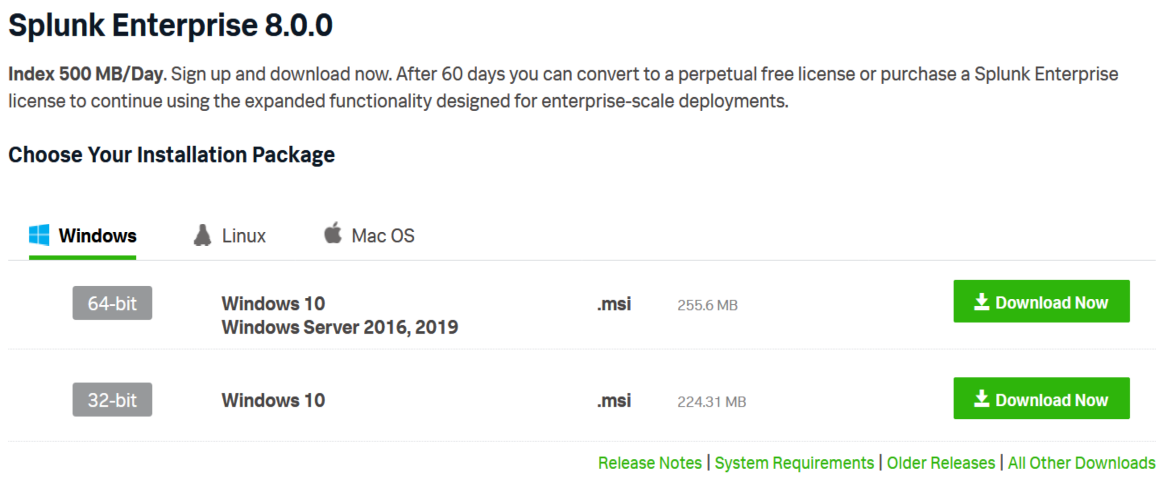
TraceListners

- Messages can be logged into multiple destinations by configuring the listeners
- Full control over message formats is not as much as like SLAB, however we can have a limited control over message format using Enterprise Library.
- TcpTraceListner and UdpTraceListner are extended classes of TraceListner (System.Diagnostics) class. We can configure these trace listener to send messages over a TCP or UDP port.
- We can configure Splunk Enterprise for a particular port (for TCP/UDP) to receive log messages. Whenever a message is available at that port Splunk automatically consumes the message and stores it within its server.

3. Splunk Installation and Configuration

3.1. Steps to install Splunk Enterprise on Windows

1. Download Splunk Enterprise from below link
   https://www.splunk.com/en_us/download/splunk-enterprise.html
2. For Installation of Windows 10, select and download the proper version.



Splunk Enterprise 8.0.0

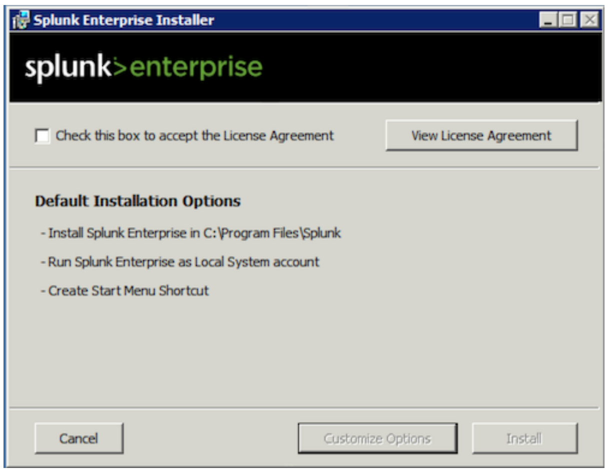
Index 500 MB/Day. Sign up and download now. After 60 days you can convert to a perpetual free license or purchase a Splunk Enterprise license to continue using the expanded functionality designed for enterprise-scale deployments.

Choose Your Installation Package

64-bit	Windows 10 Windows Server 2016, 2019	.msi	255.6 MB	Download Now
32-bit	Windows 10	.msi	224.31 MB	Download Now

Release Notes | System Requirements | Older Releases | All Other Downloads

3. To start the installer, double-click the splunk.msi file. The installer runs and displays the Splunk Enterprise Installer panel.


Splunk Enterprise Installer
splunk>enterprise
Check this box to accept the License Agreement
View License Agreement

Default Installation Options

- Install Splunk Enterprise in C:\Program Files\Splunk
- Run Splunk Enterprise as Local System account
- Create Start Menu Shortcut

Cancel
Customize Options
Install

4. Click next and Finish the installation.

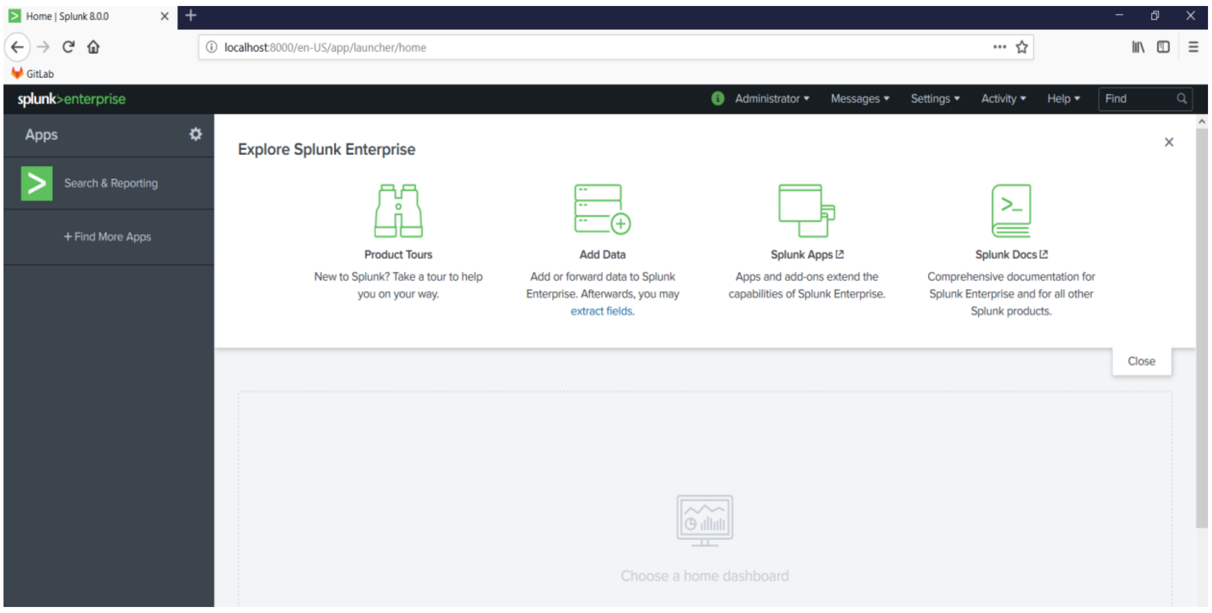
Reference: https://docs.splunk.com/Documentation/Splunk/8.0.0/Installation/InstallonWindows

Once finished, open the command prompt window in **administrative** mode, navigate to the bin folder of Splunk installation directory within the command prompt and run the following commands one by one.

```
C:\Program Files\Splunk\bin splunk stop

C:\Program Files\Splunk\bin splunk start
```

5. Splunk runs on default port 8000. Open (http://localhost:8000) in browser. Splunk Web will open.

6. Logon to Splunk Web with the credentials given by you during installation and the below screen will appear (home screen for S1plunk).



New to Splunk? Take a tour to help you on your way.

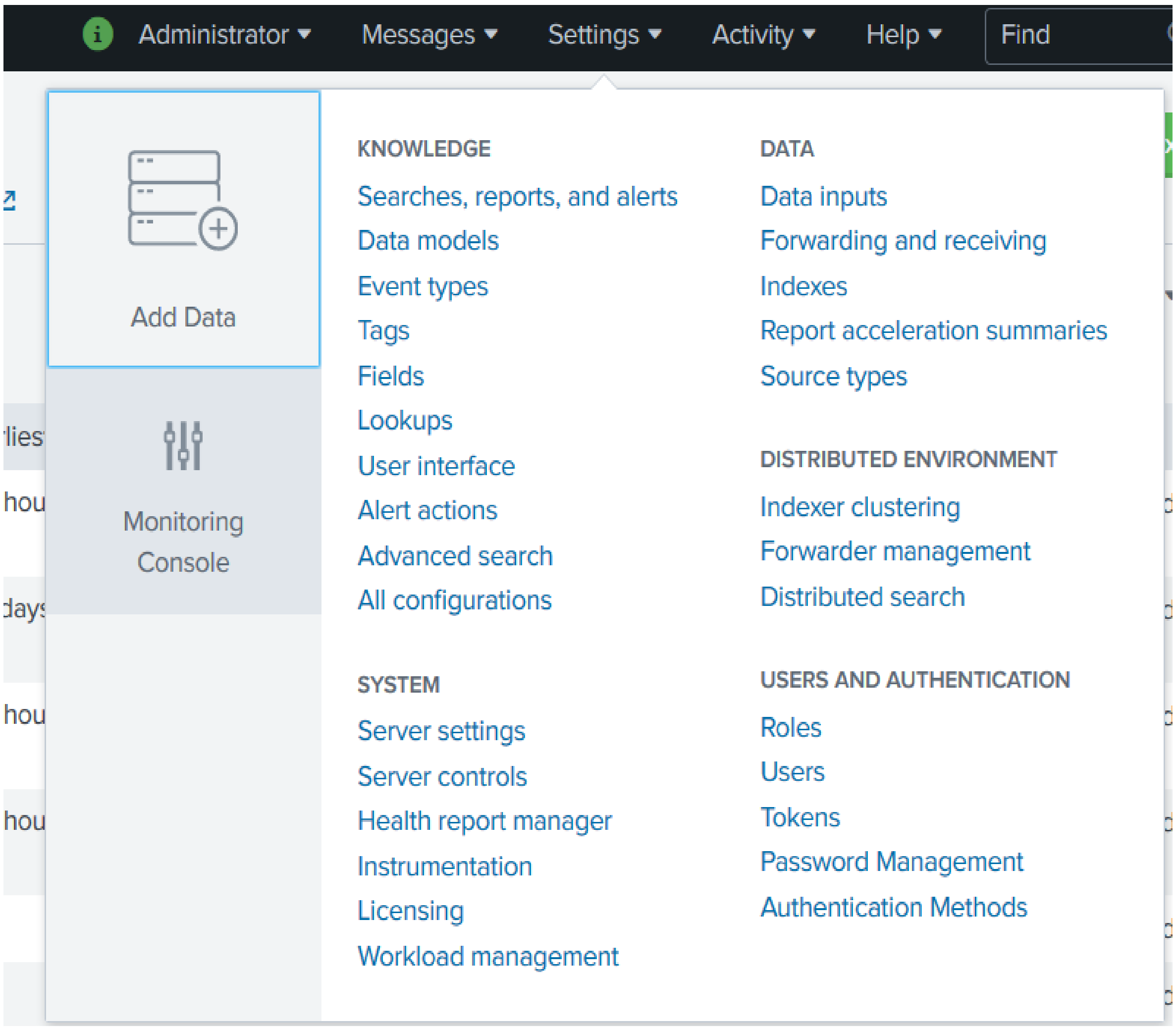
Add or forward data to Splunk Enterprise. Afterwards, you may extract fields.

Apps and add-ons extend the capabilities of Splunk Enterprise.

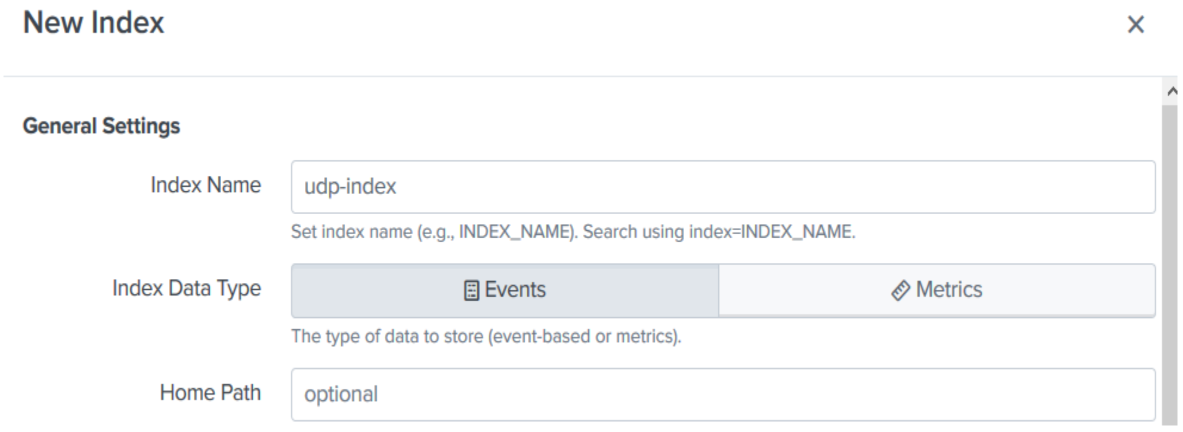
Comprehensive documentation for Splunk Enterprise and for all other Splunk products.

3.2. Steps to Configure Splunk for Udp TraceListener

1. Open Splunk web (Url e.g. http://localhost:8000)
2. Go to Settings → Indexes from Top Right Menu



3. Create a new Index (click New Index on top right side corner of page) by just giving the index name.

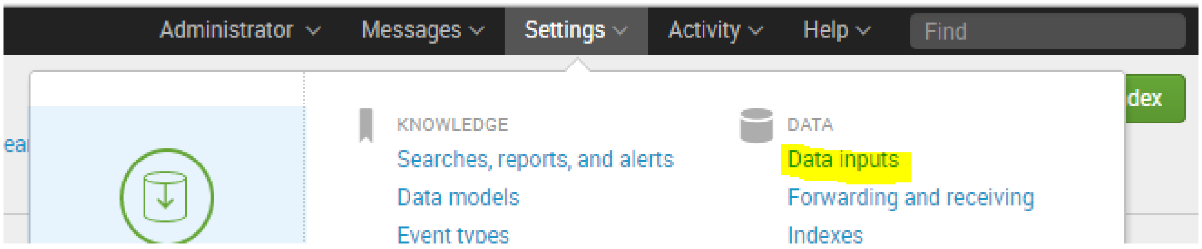


New Index

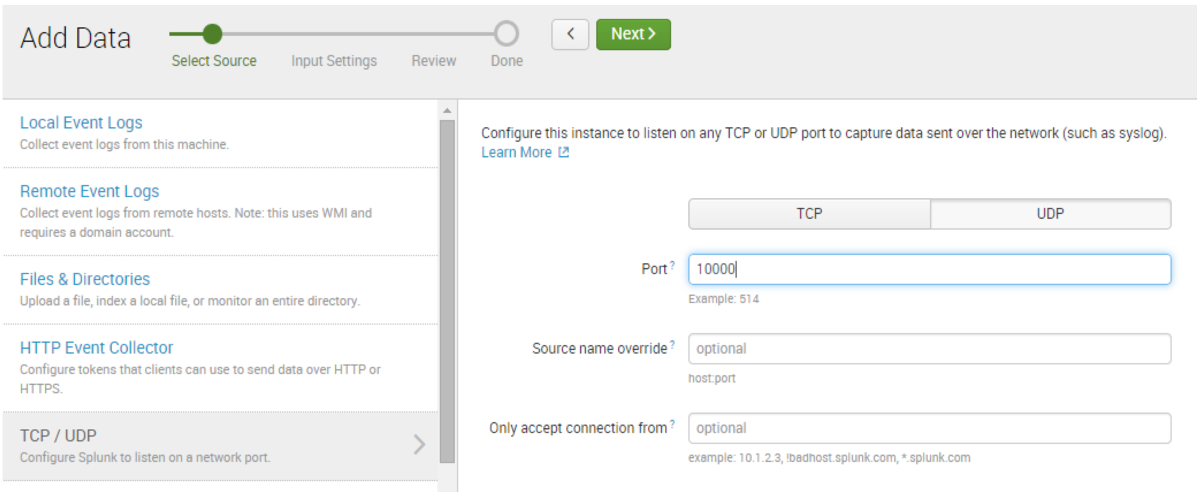
General Settings

Index Name	udp-index
	Set index name (e.g., INDEX_NAME). Search using index=INDEX_NAME.
Index Data Type	[✓ Events] Metrics
	The type of data to store (event-based or metrics).
Home Path	optional

4. After creating the index, go to Settings → Data Inputs



5. Then go to UDP → Click Add New. The below screen will appear.



Local Event Logs
Collect event logs from this machine.

Remote Event Logs
Collect event logs from remote hosts. Note: this uses WMI and requires a domain account.

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.

Configure this instance to listen on any TCP or UDP port to capture data sent over the network (such as syslog). Learn More

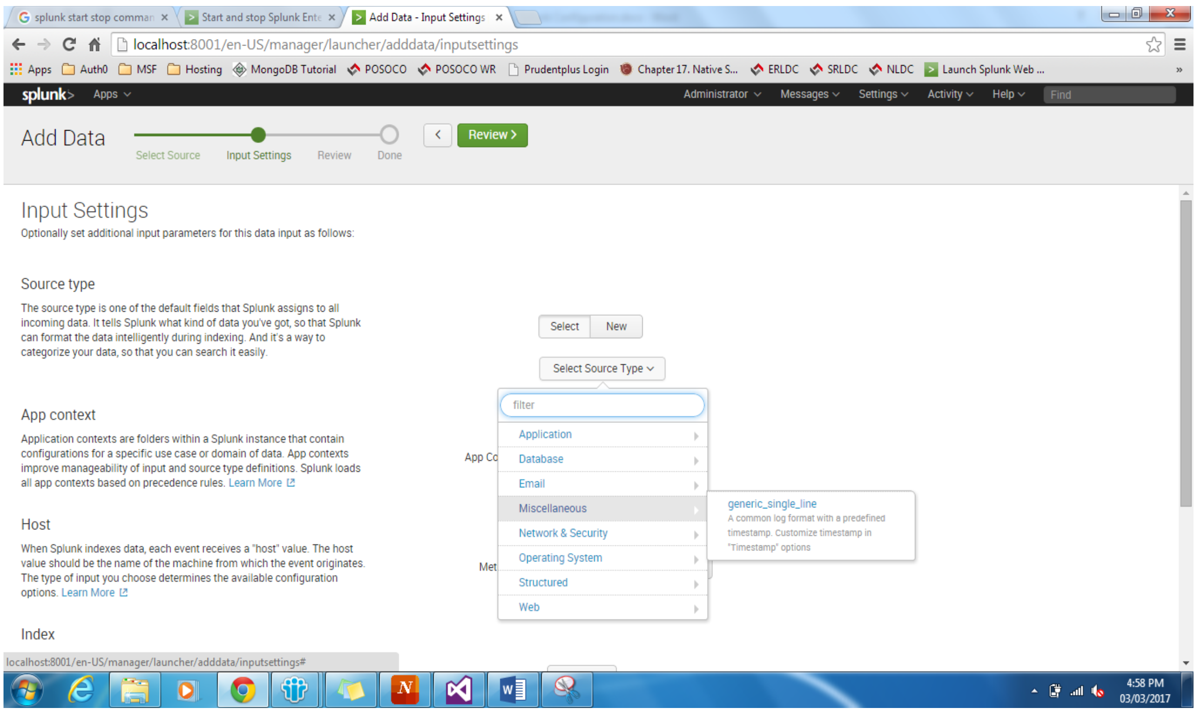
TCP | UDP

Port: 10000
Example: 514

Source name override: optional
host:port

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

6. Provide the port number (the same port number should be used in application to log messages) and click next.
7. In the next screen select the appropriate source type (for now we are using Miscellaneous → generic_single_line)



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.

App context
Application contexts are folders within a Splunk instance that contain configurations for a specific use case or domain of data. App contexts improve manageability of input and source type definitions. Splunk loads all app contexts based on precedence rules. Learn More

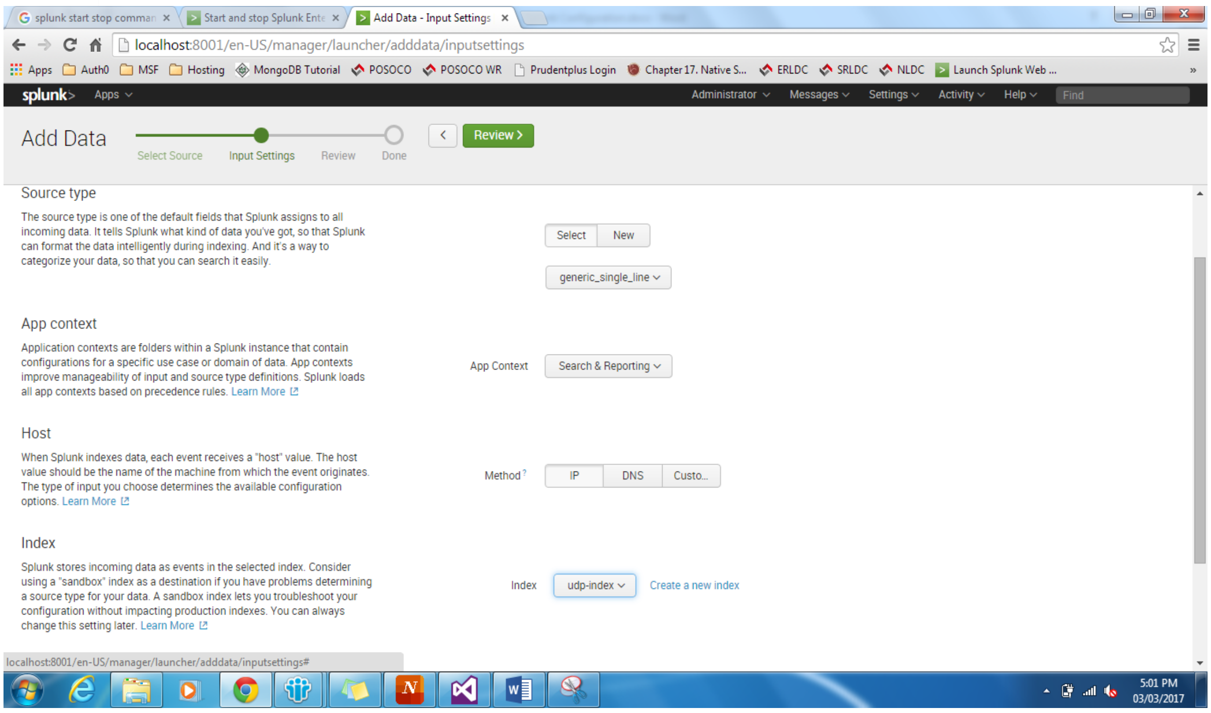
Host
When Splunk indexes data, each event receives a "host" value. The host value should be the name of the machine from which the event originates. The type of input you choose determines the available configuration options. Learn More

Index

generic_single_line
A common log format with a predefined timestamp. Customize timestamp in "Timestamp" options

localhost:8001/en-US/manager/launcher/adddata/inputsettings#

8. Select other options and select the index as the newly created custom index (e.g. udp-index)



localhost:8001/en-US/manager/launcher/adddata/inputsettings

Add Data — Select Source · Input Settings · Review · Done — Review >

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 — generic_single_line

App context

Application contexts are folders within a Splunk instance that contain configurations for a specific use case or domain of data. App contexts improve manageability of input and source type definitions. Splunk loads all app contexts based on precedence rules. Learn More

App Context: Search & Reporting

Host

When Splunk indexes data, each event receives a "host" value. The host value should be the name of the machine from which the event originates. The type of input you choose determines the available configuration options. Learn More

Method: IP | DNS | Custo...

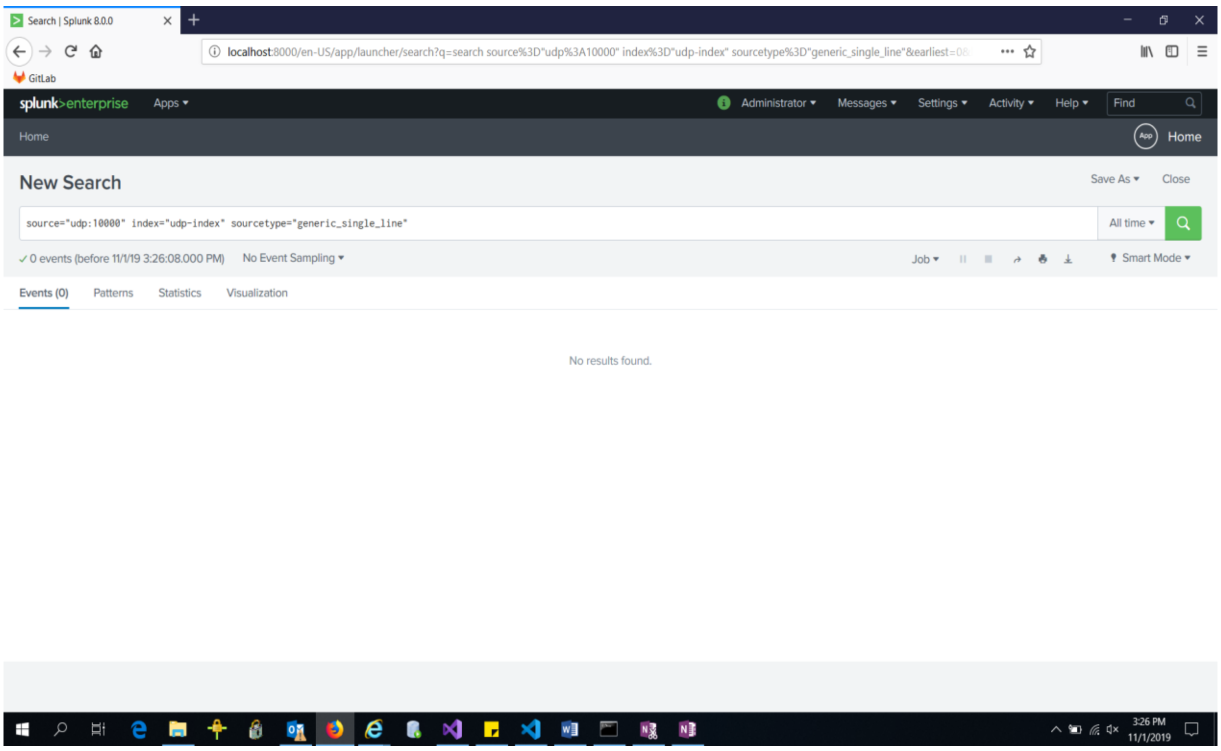
Index

Splunk stores incoming data as events in the selected index. Consider using a "sandbox" index as a destination if you have problems determining a source type for your data. A sandbox index lets you troubleshoot your configuration without impacting production indexes. You can always change this setting later. Learn More

Index: udp-index — Create a new index

9. Then Click Review and then Submit. Now Splunk is configured to receive messages from the specified Udp Port (e.g. 10000)

10. Now click on **Start Searching** and the search results will appear accordingly.



localhost:8000/en-US/app/launcher/search?q=search source%3D"udp%3A10000" index%3D"udp-index" sourcetype%3D"generic_single_line"&earliest=0&

New Search

```
source="udp:10000" index="udp-index" sourcetype="generic_single_line"
```

All time

✓ 0 events (before 11/1/19 3:26:08.000 PM) No Event Sampling

Events (0) · Patterns · Statistics · Visualization

No results found.

Configure Splunk to display Multi-Line Events as a single Event.

When we log an event or exception to Splunk, the event/exception may have multiple parameters such as Machine Name, Application Name, Location, Date etc. By default these parameters along with the exception message are grouped together and shown as a single line event which is actually too messy to read.

To separate all the parameters and show them as individual lines for a single event you can configure Splunk as below.

1. Open Command Prompt as Administrator
2. Stop Splunk
   Navigate to Splunk bin folder of Installation directory.
   Command – cd C:\Program Files\Splunk\bin *stop splunk*
3. Once the splunkd service gets stopped, Open **props.conf** file from the installation directory
   Location: C:\Program Files\Splunk\etc\system\local\props.conf

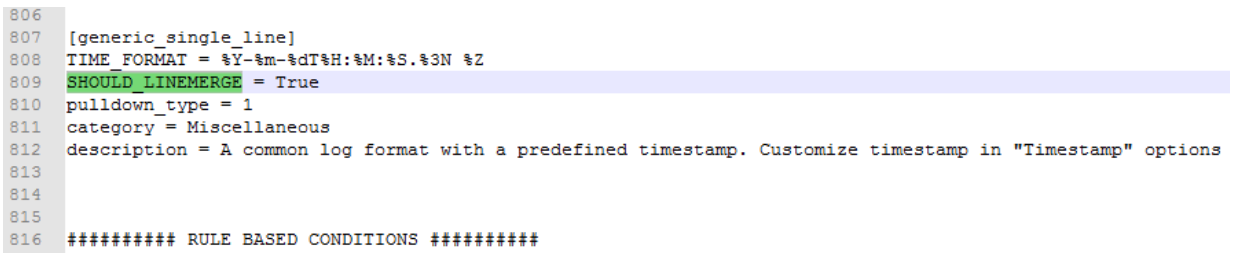
   If file does not exist, copy the file from
   C:\Program Files\Splunk\etc\system\default\props.conf **to the above directory.**

   Note: DO NOT Modify the props.conf file present within
   C:\Program Files\Splunk\etc\system\default\

4. Identify required section and set the value of **SHOULD_LINEMERGE** property to **true**

   Example:-

   As we have selected the source as **generic_single_line** during UDP configuration, we have to modify the below block in order to get the messages collectively as a single event.



```

[generic_single_line]
TIME_FORMAT = %Y-%m-%dT%H:%M:%S.%3N %Z
SHOULD_LINEMERGE = True
pulldown_type = 1
category = Miscellaneous
description = A common log format with a predefined timestamp. Customize timestamp in "Timestamp" options



########## RULE BASED CONDITIONS ##########
```

5. Save and close the file.
6. Start Splunk using the command prompt
   Command: cd C:\Program Files\Splunk\bin *start splunk*

   https://docs.splunk.com/Documentation/Splunk/8.0.0/Data/Configureeventlinebreaking
   http://docs.splunk.com/Documentation/Splunk/latest/admin/propsconf

4. Logging to Splunk from .NET application

In this section you will know how to log an exception to the Splunk server via UDP port. We will be using both TraceListener and SLAB methods for logging. Also we will use EnterpriseLibrary Logging Application Block for Logging the events to Splunk.

4.1. Log event using Udp TraceListener

Let's Create a Sample Console Application in Visual Studio 2019 and name it as "SplunkLoggingSample".

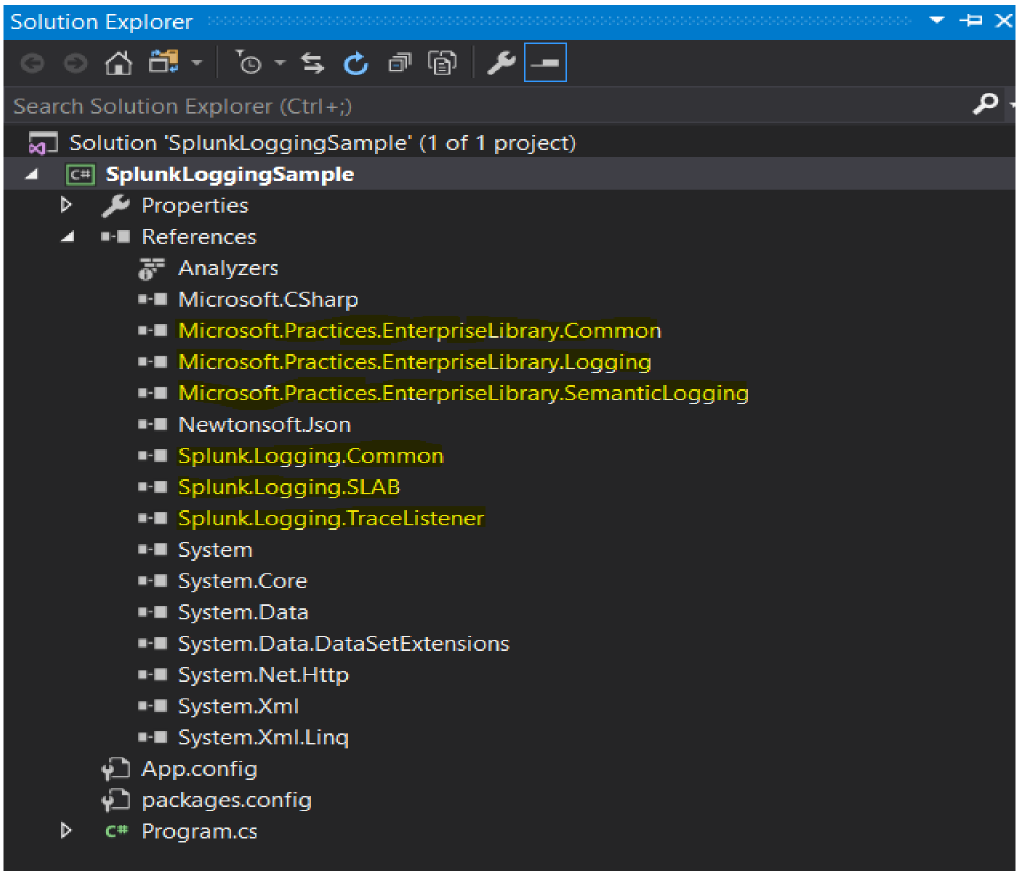
Once the application has been created add the below Nuget Packages to the solution. Open the Nuget Package Manager Window and search "splunk" within the search bar. Then Install the following packages.

1. Splunk.Logging.TraceListner
2. Splunk.Logging.SLAB

As we are going to use Enterprise Library to log the messages, we need to install packages for it as well. Search "enterprise" and install below package.

1. EnterpriseLibrary.Logging

Once installed all the packages if you expand the References section in Solution Explorer, you will see the below installed Libraries which are essential in writing our logging code.



```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SplunkLoggingSample' (1 of 1 project)
  SplunkLoggingSample
    Properties
    References
      Analyzers
      Microsoft.CSharp
      Microsoft.Practices.EnterpriseLibrary.Common
      Microsoft.Practices.EnterpriseLibrary.Logging
      Microsoft.Practices.EnterpriseLibrary.SemanticLogging
      Newtonsoft.Json
      Splunk.Logging.Common
      Splunk.Logging.SLAB
      Splunk.Logging.TraceListener
      System
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Net.Http
      System.Xml
      System.Xml.Linq
    App.config
    packages.config
    Program.cs
```

Once the required packages are installed we are good to go with the coding.

Add the following Configuration in your application’s **App.config** file.

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="loggingConfiguration" type="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.LoggingSettings, Microsoft.Practices.EnterpriseLibrary.Logging"/>
  </configSections>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="Newtonsoft.Json" publicKeyToken="30ad4fe6b2a6aeed" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-11.0.0.0" newVersion="11.0.0.0" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
  <appSettings>
    <add key="HostName" value="127.0.0.1"/>
    <add key="Port" value="10000"/>
  </appSettings>
  <loggingConfiguration name="Logging Application Block" tracingEnabled="true" defaultCategory="General" logWarningsWhenNoCategoriesMatch="true">
    <listeners>
      <add name="Flat File Trace Listener" type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.RollingFlatFileTraceListener, Microsoft.Practices.EnterpriseLibrary.Logging"
           listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.RollingFlatFileTraceListenerData, Microsoft.Practices.EnterpriseLibrary.Logging"
           fileName="C:\Chinmaya\PoC\cpdevtfs\SplunkLoggingSample\SplunkLoggingSample\TraceData.log"
           footer="-----------------------------------" formatter="Text Formatter"
           header="" rollInterval="Day"
           traceOutputOptions="DateTime, Timestamp" filter="All" />
      <add name="Udp Trace Listener" type="Splunk.Logging.UdpTraceListener, Splunk.Logging.TraceListener"
           listenerDataType="SplunkLoggingSample.CustomTraceListenerData, SplunkLoggingSample"
           filter="All" host="127.0.0.1" port="10000" />
    </listeners>
    <formatters>
      <add type="Microsoft.Practices.EnterpriseLibrary.Logging.Formatters.TextFormatter, Microsoft.Practices.EnterpriseLibrary.Logging"
           template="Timestamp: {timestamp(local)}{newline}  Message: {message}{newline}  Category: {category}{newline}  Priority: {priority}{newline}  Severity: {severity}"
           name="Text Formatter" />
    </formatters>
    <categorySources>
      <add switchValue="All" autoFlush="true" name="General">
        <listeners>
          <add name="Flat File Trace Listener" />
          <add name="Udp Trace Listener" />
        </listeners>
      </add>
    </categorySources>
    <specialSources>
      <allEvents switchValue="All" name="All Events">
        <listeners>
          <add name="Udp Trace Listener" />
        </listeners>
      </allEvents>
      <notProcessed switchValue="All" name="Unprocessed Category">
        <listeners>
          <add name="Udp Trace Listener" />
        </listeners>
      </notProcessed>
      <errors switchValue="All" name="Logging Errors & Warnings">
        <listeners>
          <add name="Udp Trace Listener" />
        </listeners>
      </errors>
    </specialSources>
  </loggingConfiguration>
</configuration>
```

App.config explained:-

- The new section "loggingConfiguraion" contains all the configuration for our enterprise library logging application block.
- We have added a listener named as "Udp Trace Listener" which has a host name and port number (it is same as we configured the splunk server earlier – see section 3.2, configure splunk for udp trace listener). At this address (host and port) the events will be received.
- The listenerDataType holds the Fully Qualified class name of our Custom Trace Listener Data class.
- We have added a formatter, this is not required for our case. Formatters are required in case where we have a tracelistener for Flat File Logging.
- Again we have defined the trace switches within the categorySources section. Trace switch controls which trace listener would be used when depending upon the requirement.

Now let's create a custom Trace Listener class which extends the UdpTraceListener class of Splunk.Logging Library, which will use the above host name and port number to process message.

```
using Microsoft.Practices.EnterpriseLibrary.Common.Configuration;
using Microsoft.Practices.EnterpriseLibrary.Logging.Formatters;
using Splunk.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SplunkLoggingSample
{
    [ConfigurationElementType(typeof(CustomTraceListenerData))]
    public class CustomTraceListener : UdpTraceListener
    {
        string host;
        int port;

        public CustomTraceListener(string _host, int _port) :base(_host, _port)
        {
            this.host = _host;
            this.port = _port;
        }
    }
}
```

To initialize this class we need to add another customTraceListenerData class which extends the base TraceListenerData class from Enterprise Library.

```csharp
using Microsoft.Practices.EnterpriseLibrary.Common.Configuration;
using Microsoft.Practices.EnterpriseLibrary.Logging.Configuration;
using Microsoft.Practices.EnterpriseLibrary.Logging.Formatters;
using System.Configuration;
using System.Diagnostics;

namespace SplunkLoggingSample
{
    [ConfigurationElementType(typeof(CustomTraceListenerData))]
    public class CustomTraceListenerData : TraceListenerData
    {
        private const string hostProperty = "host";
        private const string portProperty = "port";

        /// <summary>
        /// Default Constructor
        /// </summary>
        public CustomTraceListenerData()
        {

        }

        [ConfigurationProperty(hostProperty, IsRequired = true)]
        public string HostName
        {
            get { return (string)base[hostProperty]; }
            set { base[hostProperty] = value; }
        }

        [ConfigurationProperty(portProperty, IsRequired = true)]
        public int PortNumber
        {
            get { return (int)base[portProperty]; }
            set { base[portProperty] = value; }
        }

        /// <summary>
        /// Initialize the Custom TraceListener with hostname and port
        /// </summary>
        /// <param name="settings"></param>
        /// <returns>Custome TraceListener</returns>
        protected override TraceListener CoreBuildTraceListener(LoggingSettings settings)
        {
            return new CustomTraceListener(HostName, PortNumber);
        }
    }
}
```

Now our TraceListeners are properly configured and ready to use. Whenever we will write an event using the Enterprise Library Logging Application Block (i.e. LogEntry object to the Logger.Write() method) the event will be automatically transferred to the Splunk server and through the TraceListeners we have used.
Now In the Main() method we just need to create the Log event and Write it to the Logger Class.

```csharp
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SplunkLoggingSample
{
    public class Program
    {
        static void Main(string[] args)
        {
            //Initialize the LogWriter
            Logger.SetLogWriter(new LogWriterFactory().Create());

            //Prepare the LogEntry Object
            LogEntry entry = new LogEntry
            {
                Message = "Log Started with Enterprise Library and Udp Trace Listener.",
                EventId = 101,
                Severity = TraceEventType.Critical,
                Priority = 2,
                Title = "Udp TraceListener Logging"
            };

            //Can add additional info as key-value pair
            entry.ExtendedProperties.Add("Employee Number", 12345);

            //Log via Trace Listener
            Logger.Write(entry);

            Console.ReadKey();
        }
    }
}
```

The moment *Logger.Write(entry)* will be executed, event will be logged to Splunk server.

4.2. Log event using SLAB

In-order to write events with SLAB we need to create another class (say SLABLogging). The app.config LoggingConfiguration section is not relevant for this now.

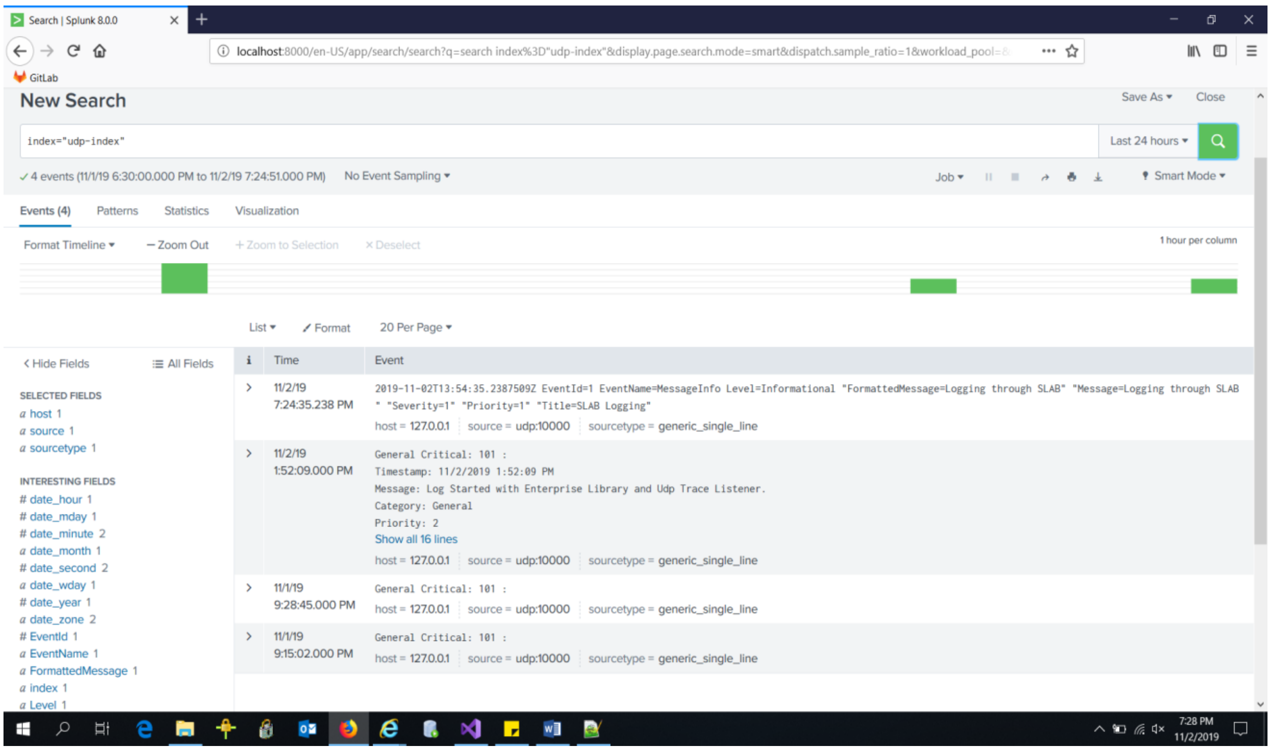
For SLAB we need to create an event source and event sink. The source from which event will be created and sink where event will be received. We will use the same Udp port as event sink and define the custom event source.

```
using Microsoft.Practices.EnterpriseLibrary.SemanticLogging;
using Splunk.Logging;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Diagnostics.Tracing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SplunkLoggingSample
{
    public class SLABLogging
    {
        private static string host = ConfigurationManager.AppSettings["HostName"].ToString()
;
        private static int port = Convert.ToInt32(ConfigurationManager.AppSettings["Port"]);


        public static void LogMessage(string message, int severity, int priority, string tit
le)
        {
            var listener = new ObservableEventListener();
            listener.Subscribe(new UdpEventSink(host, port));

            var eventSource = new SimpleEventSource();
```

```
                listener.EnableEvents(eventSource, EventLevel.LogAlways, Keywords.All);

                eventSource.Message(message, severity, priority, title);
            }
        }

        [EventSource(Name = "SLABEventSource")]
        public class SimpleEventSource : EventSource
        {
            public class Keywords { }
            public class Tasks { }

            [Event(1, Message = "{0}", Level = EventLevel.Informational)]
            internal void Message(string Message, int Severity, int Priority, string Title)
            {
                this.WriteEvent(1, Message, Severity, Priority, Title);
            }
        }
    }
```

When we call the above LogMessage method and pass the relevant parameters to this, event will be logged into Splunk Server.

Go to the Splunk server and search within Search & Reporting appropriate search criterias like *index="udp-index"* and you will see the messages logged.



index="udp-index"

✓ 4 events (11/1/19 6:30:00.000 PM to 11/2/19 7:24:51.000 PM)

SELECTED FIELDS
a host 1
a source 1
a sourcetype 1

INTERESTING FIELDS
# date_hour 1
# date_mday 1
# date_minute 2
a date_month 1
# date_second 2
a date_wday 1
# date_year 1
a date_zone 2
EventId 1
a EventName 1
a FormattedMessage 1
a index 1
a Level 1

i	Time	Event
>	11/2/19 7:24:35.238 PM	2019-11-02T13:54:35.2387509Z EventId=1 EventName=MessageInfo Level=Informational "FormattedMessage=Logging through SLAB" "Message=Logging through SLAB " "Severity=1" "Priority=1" "Title=SLAB Logging" host = 127.0.0.1 source = udp:10000 sourcetype = generic_single_line
>	11/2/19 1:52:09.000 PM	General Critical: 101 : Timestamp: 11/2/2019 1:52:09 PM Message: Log Started with Enterprise Library and Udp Trace Listener. Category: General Priority: 2 Show all 16 lines host = 127.0.0.1 source = udp:10000 sourcetype = generic_single_line
>	11/1/19 9:28:45.000 PM	General Critical: 101 : host = 127.0.0.1 source = udp:10000 sourcetype = generic_single_line
>	11/1/19 9:15:02.000 PM	General Critical: 101 : host = 127.0.0.1 source = udp:10000 sourcetype = generic_single_line

7:28 PM
11/2/2019

5. Appendix

5.1. Reference

http://docs.splunk.com/Documentation/Splunk/8.0.0/Installation/InstallonWindows

http://docs.splunk.com/Documentation/Splunk/8.0.0/SearchTutorial/StartSplunk#Start_Splunk_Enterprise_on_Windows

http://docs.splunk.com/Documentation/Splunk/latest/Data/Monitornetworkports