Best Practices for Building Splunk Apps and Technology Add-ons

Presented by: Jason Conger

Splunk, Inc.
250 Brannan Street, 2nd Floor
San Francisco, CA 94107

+1.415.568.4200(Main)
+1.415.869.3906 (Fax)
www.splunk.com

splunk>

Table of Contents

Overview

The purpose of this guide is to provide guidance on building Splunk Add-Ons and Applications. The recommendations provided within this document may not be appropriate for every environment. Therefore, all best practices within this document should be evaluated in an isolated test environment prior to being implemented in production.

Creating a Directory Structure for Your Application or Add-on

Splunk Applications and Add-ons are basically a file system directory containing a set of configurations for collecting data and/or analyzing data. To get started on your Splunk application, create a directory in $SPLUNK_HOME/etc/apps where $SPLUNK_HOME is one of the following by default:

Windows	%ProgramFiles%\Splunk\etc\apps
*nix	/opt/Splunk/etc/apps
Mac	/Applications/Splunk/etc/apps

All configurations will go in this directory to make the application or add-on self-contained and portable.

Directory Naming Conventions for Your Application or Add-on

For applications (dashboards, forms, saved searches, alerts, etc.):

> *Vendor*-app-*product*
>
> Example: acme-app-widget

For add-ons (data collection mechanisms with no user interface):

> TA_*vendor-product*
>
> Example : TA_acme-widget

TA stands for Technology Add-on

Note: after uploading an application to Splunk Apps, the directory name cannot be changed. The actual name of the application displayed on the Splunk start screen and on Splunk Apps is controlled by a file named app.conf and is independent of the directory name mentioned above.

Getting Data Into Splunk

The first thing that needs to happen to create a Splunk Application is to get data into Splunk. There are various methods to get data into Splunk including, but not limited to, the following:

- Reading log files on disk
- Sending data to Splunk over the network via TCP or UDP
- Pulling data from APIs
- Sending scripted output to Splunk such as bash, PowerShell, batch, etc.
- Microsoft Windows perfmon, Event Logs, registry, WMI, etc.

These methods are covered in detail at the Splunk Docs site
http://docs.splunk.com/Documentation/Splunk/latest/Data/WhatSplunkcanmonitor

Data can be local or remote to the Splunk instance.

Local Data

A local resource is a fixed resource that your Splunk Enterprise server has direct access to, meaning you are able to access it - and whatever is contained within it - without having to attach, connect, or perform any other intermediate action (such as authentication or mapping a network drive) in order to have that resource appear available to your system. A Splunk instance can reach out to other remote systems or receive data from remote systems over the network.

Remote Data

A remote resource is any resource where the above definition is not satisfied. Splunk Universal Forwarders can be installed on remote systems to gather data locally and send the gathered data over the network to a central Splunk instance.

Best Practices for Reading Existing Log Files

Existing log files are easily read using Splunk. Information about how to direct Splunk to monitor existing log files and directories can be found here -> http://docs.splunk.com/Documentation/Splunk/latest/Data/Monitorfilesanddirectories

Best Practices for Logging Data to be consumed by Splunk

If you have control of how data is logged, the following best practices can help Splunk better recognize events and fields with little effort:

- Start the log line event with a time stamp
- Use clear key-value pairs
- When using key-value pairs, leverage the Common Information Model
- Create events that humans can read
- Use unique identifiers
- Log in text format
- Use developer-friendly formats
- Log more than just debugging events
- Use categories
- Keep multi-line events to a minimum
- Use JSON (Java Script Object Notation) format

More information about these best practices can be found here ->

http://dev.splunk.com/view/logging-best-practices/SP-CAAADP6

Best Practices for Event Breaking

For each of the inputs, at a minimum, ensure you set the following props.conf attributes. They help tremendously with event breaking and timestamp recognition performance:

TIME_PREFIX, MAX_TIMESTAMP_LOOKAHEAD, TIME_FORMAT, LINE_BREAKER, SHOULD_LINEMERGE, TRUNCATE, KV_MODE

Sample events:

```
2014-07-15 10:51:06.700 -0400 "GET
servicesNS/admin/launcher/search/typeahead?prefix=index%3D_internal&count=50&
output_mode=json&max_time= HTTP/1.0" 200 73 - - - 1ms
2014-07-15 10:51:06.733 -0400 "GET
servicesNS/admin/launcher/search/typeahead?prefix=index%3D_internal&count=50&
output_mode=json&max_time=  HTTP/1.0" 200 73 - - - 2ms
2014-07-15 10:51:06.833 -0400 "GET
servicesNS/admin/launcher/search/typeahead?prefix=index%3D_internal&count=50&
output_mode=json&max_time= HTTP/1.0" 200 73 - - - 1ms
```

Sample props.conf

```
[sourcetypeA]

TIME_PREFIX = ^

MAX_TIMESTAMP_LOOKAHEAD = 25

TIME_FORMAT = %Y-%m-%d %H:%M:%S.%3N %z

LINE_BREAKER = ([\r\n]+)\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2}.\d{3}

SHOULD_LINEMERGE = False

TRUNCATE = 5000

KV_MODE = None

ANNOTATE_PUNCT = false
```

For each of the above settings, detailed descriptions can be found on the manual for props.conf but here's a brief explanation:

- TIME_PREFIX: Leads Splunk to the exact location to start looking for a timestamp pattern. The more precise this is, the faster the timestamp processing.
- MAX_TIMESTAMP_LOOKAHEAD: Tells splunk how far after TIME_PREFIX the timestamp pattern extends.
- TIME_FORMAT: Tells splunk the exact format of the timestamp (instead of having Splunk to guess what it is by iterating over *many* possible timeformats)
- LINE_BREAKER: Tells splunk how to break the stream into events. This setting requires a capture group and the breaking point is immediately after it. Capture group content is tossed away. In this specific example it reads: "break after one or more carriage returns or newlines followed by a pattern that looks like a timestamp"
- SHOULD_LINEMERGE: Tells Splunk not to engage line merging (break on newlines and merge on timestamps), which is known to be a huge resource hog, especially for multiline events. Instead, by defining LINE_BREAKER we're telling it to break on a definite pattern.
- TRUNCATE: maximum line length (or event length) in bytes. This defaults to 10K. Its prudent to have something non-default depending on expected event length. Super-long events tend to indicate a logging system problem.

- KV_MODE: Specify exactly what you want Splunk to engage at search time. If you do not have events in KV pairs, or any other poly/semi/structured format, disable it.
- ANNOTATE_PUNCT: Unless you expect PUNCT to be used in your searches, disable its extraction as it adds index-time overhead.

Scripted Input Best Practices

Scripted inputs allow you to gather data from sources where data does not exist on disk. Examples include calling APIs or gathering in-memory data. Any script that the operating system can run, Splunk can use for scripted inputs. Any output from the script to stdout (the screen by default) will end up in the Splunk index.

Do not hard code paths in scripts

When referencing file paths in the Splunk directory, use the special $SPLUNK_HOME environment variable. This environment variable will be automatically expanded to the correct Splunk path based on the operating system on which Splunk is running.

Example (Python):

```
os.path.join(os.environ["SPLUNK_HOME"],'etc','apps',APP_NAME)
```

Example (PowerShell):

```
Join-Path -path (get-item env:\SPLUNK_HOME).value "Splunk\etc\apps"
```

Use Splunk Entity Class or position files as a placeholder

Oftentimes, you may be calling an API with a scripted or modular input. In order to only query a specific range of values, use either the Splunk Entity Class or a position file to keep track of where the last run left off so that the next run will pick up at this position.

Depending on where your input runs will dictate whether you should use the Splunk Entity class or use a position file. For more detailed information, refer to the following blog post -> http://blogs.splunk.com/2014/09/22/pick-up-where-you-left-off-in-scripted-and-modular-inputs/

For position files, avoid using files that start with a dot as operating systems usually treat these types of files as special files.

Do use acme.pos

Do not use .pos

Use error trapping and logging

The following example demonstrates how to use the Python Logging Facility. Information logged with logging.error() will end up in splunkd.log as well as a special "_internal" index that can used for troubleshooting.

Example with Python Logging Module:

```
import logging

try:

      Some code that may fail like opening a file

except IOError, err:

      logging.error('%s - ERROR - File may not exist %s\n' %
(time.strftime("%Y-%m-%d %H:%M:%S"), str(err)))

      pass
```

Using stderr

Just like anything written to stdout will end up in the Splunk index, anything written to stderr from a scripted input will behave like logging.error() from above.

Example (Python):

```
try:

      Some code that may fail like opening a file

except IOError, err:

      sys.stderr.write('%s - ERROR - File may not exist %s\n' %
(time.strftime("%Y-%m-%d %H:%M:%S"), str(err)))

      pass
```

Example (PowerShell):

```
try

{

    Some code that may fail like opening a file

}

catch

{

    Write-Error('{0:MM/dd/yyyy HH:mm:ss} GMT - {1} {2}' -f (Get-
Date).ToUniversalTime(), "Could not create position file: ",
$_.Exception.Message)

    exit

}
```

Test Scripts using Splunk CMD

To see the output of a script as if it was run by the Splunk system, use the following:

Mac:

```
/Applications/Splunk/bin/splunk cmd python
/Applications/Splunk/etc/apps/<your app>/bin/<your script>
```

Windows:

```
C:\Program Files\Splunk\bin\splunk.exe cmd C:\Program
Files\Splunk\etc\apps\<your app>\bin\<your script>
```

More useful command line tools to use with Splunk can be found here ->
http://docs.splunk.com/Documentation/Splunk/latest/Troubleshooting/CommandlinetoolsforusewithSupport

Use configuration files to store user preferences

Configuration files store specific settings that will vary for different environments. Examples include REST endpoints, API levels, or any specific setting. Configuration files are stored in either of the following locations and cascade:

```
$SPLUNK_HOME/etc/apps/<your_app>/default
$SPLUNK_HOME/etc/apps/<your_app>/local
```

For example, suppose there is a configuration file called acme.conf in both the default and local directories. Settings from the local directory will override settings in the default directory.

Use Splunk methods to read cascaded settings

The Splunk cli_common library contains methods that will read combined settings from configuration files.

Example (Python):

```
import splunk.clilib.cli_common


def __init__(self,obj):
```

```
    self.object = obj
    self.settings = splunk.clilib.cli_common.getConfStanza("acme", "default")
```

Use script methods to construct file paths

Example (Python):

```
abs_file_path = os.path.join(script_dir, rel_path)
```

Example (PowerShell):

```
$positionFile = Join-Path $positionFilePath $positionFileName
```

Modular Inputs

Modular Inputs allows you to extend the Splunk Enterprise framework to define a custom input capability. Your custom input definitions are treated as if they were part of Splunk Enterprise native inputs. The inputs appear automatically on the **Settings > Data Inputs** page. From a Splunk Web perspective, your users interactively create and update your custom inputs using Settings, just as they do for Splunk Enterprise native inputs.

Modular Inputs vs. Scripted Inputs

Modular inputs can be used just about anywhere scripted inputs are used. Scripted inputs are quick and easy, but may not be the easiest for an end user. Modular inputs require more upfront work, but are easier for end user interaction.

For more information on modular inputs as well as comparisons between scripted inputs and modular inputs, follow this link -> http://docs.splunk.com/Documentation/Splunk/latest/AdvancedDev/ModInputsIntro

Analyzing Data with Splunk

Once data is in a Splunk index, the data can be analyzed via searches, reports, and visualizations.

Dashboard and Form Best Practices

There are multiple options for building Splunk dashboards – Simple XML, Advanced XML, and the Splunk Web Framework. Most dashboard needs are covered in Simple XML; however, if you require advanced visualizations that are outside the scope of the visualizations included with Simple XML, the Splunk Web Framework allows you to use any HTML/JavaScript libraries alongside Splunk data.

More information on Simple XML can be found here ->
http://docs.splunk.com/Documentation/Splunk/latest/Viz/Visualizationreference

More information on the Splunk Web Framework -> http://dev.splunk.com/view/web-framework/SP-CAAAER6

Splunk 6.x Dashboard Examples

The Splunk 6.x Dashboards Examples app provides numerous examples of how to visualize your data in Simple XML format. The app can be downloaded here -> http://apps.splunk.com/app/1603/

Search Best Practices

The Splunk Search Processing Language (SPL) is the heart of all Splunk analytics. A firm understanding of the SPL is critical to creating good analytics. Several examples of more common search commands can be found on this quick start guide -> http://dev.splunk.com/web_assets/developers/pdf/splunk_reference.pdf

Parameterize index names

Parameterize index names so that they can be changed later without modifying existing searches. This can be done as a macro or eventtype.

marcros.conf example:

```
[acme_index]

definition = index=acme
```

Example search using macro:

```
`acme_index` sourcetype=widiget | stats count
```

eventtypes.conf example:

```
[acme_eventtype]

search = index=acme sourcetype="widget"
```

Example search using eventtype:

```
eventtype=acme_eventtype | stats count
```

Use scheduled saved searches to build lookup files

By building lookup files using a scheduled saved search, lookup files will be automatically replicated in a distributed environment.

Example (from saved searches.conf):

```
[Lookup - WinHosts]

action.email.inline = 1

alert.suppress = 0

alert.track = 0

auto_summarize.dispatch.earliest_time = -1d@h

cron_schedule = 0 0 * * *

description = Updates the winHosts.csv lookup file

dispatch.earliest_time = -26h@h

dispatch.latest_time = now

enableSched = 1

run_on_startup = true

search = `acme_index` sourcetype=WinHostMon | stats latest(_time) AS _time
latest(OS) AS OS latest(Architecture) AS Architecture latest(Version) AS
Version latest(BuildNumber) AS BuildNumber latest(ServicePack) AS ServicePack
latest(LastBootUpTime) AS LastBootUpTime latest(TotalPhysicalMemoryKB) AS
TotalPhysicalMemoryKB latest(TotalVirtualMemoryKB) as TotalVirtualMemoryKB
latest(NumberOfCores) AS NumberOfCores by host | inputlookup append=T
winHosts.csv | sort _time | stats latest(_time) AS _time latest(OS) AS OS
latest(Architecture) AS Architecture latest(Version) AS Version
latest(BuildNumber) AS BuildNumber latest(ServicePack) AS ServicePack
latest(LastBootUpTime) AS LastBootUpTime latest(TotalPhysicalMemoryKB) AS
TotalPhysicalMemoryKB latest(TotalVirtualMemoryKB) as TotalVirtualMemoryKB
latest(NumberOfCores) AS NumberOfCores by host | outputlookup winHosts.csv
```

Use TERM() to Search for IP Addresses

When you search for a term that contains minor segmenters, Splunk defaults to treating it as a phrase: It searches for the conjunction of the subterms (the terms between minor breaks) and post-filters the results. For example, when you search for the IP address 127.0.0.1, Splunk searches for: 127 AND 0 AND 1

If you search for TERM(127.0.0.1), Splunk treats the IP address as a single term to match in your raw data.

Knowledge Object Scope

Knowledge Object definition -> http://docs.splunk.com/Splexicon:Knowledgeobject

Knowledge Objects can be scoped to individuals, apps, or global. The scope of the knowledge objects is controlled via default.meta or local.meta. Your app should not ship with a local.meta file, so all scoping should be defined in default.meta.

It is a best practice to scope all knowledge objects to the application only. However, if you are creating a TA that is not visible and/or will be used by multiple other applications, the scope of the object should be set to Global.

Packaging Your App for Distribution

After you build your Splunk application, you can share your extensions to Splunk Enterprise on Splunk Apps and make them available to everyone in the Splunk community or distribute them directly to your costumers.

Detailed instructions on the process for packaging your application for redistribution can be found here ->

http://docs.splunk.com/Documentation/Splunk/latest/AdvancedDev/PackageApp

Important Information Regarding app.conf

The name of your application, version, App ID, etc. is stored in a file called app.conf. The value for **id** in app.conf must match the folder name in which your apps lives in $SPLUNK_HOME/etc/apps. Once this id is set and uploaded to Splunk Apps, the id cannot be changed unless you create a separate application to upload.

Application Naming Conventions

Certain naming conventions should be followed. What you name your application or add-on is not impacted by how you named your directory structure.

A detailed list of naming convention parameters can be found here ->

http://docs.splunk.com/Documentation/Splunkbase/latest/Splunkbase/Namingguidelines

Definitions

Splunk Instance – the server(s) where the Splunk software is installed. This can be a single server consisting of the Splunk Indexer, Search Head, and Deployment Server. Or, this can be a collection of servers in a distributed deployment. Generally, a Splunk Forwarder is not considered to be part of a Splunk Instance.

Splunk Forwarder – a piece of software running on a remote system that collects and forwards data to a Splunk Instance.

Technology Add-on (TA) – a set of Splunk configurations, scripts, modular inputs, etc. The purpose of a TA is to collect data and/or add knowledge to the collected data.

Revision History

Rev.	Change Description	Updated By	Date
1.0	Initial Document	Business Development	July 15, 2014
1.1	Added Do's and Don'ts table	Business Development	August 8, 2014
1.2	Added Knowledge Object Scope	Business Development	September 6, 2014
1.3	Added Appendix B – Application/Add-on checklist	Business Development	October 2, 2014

Appendix A – Do's and Don'ts

Application

Do	Don't
Use setup.xml or a django form to allow the end user to configure the app	Make users manually enter information such as API credentials into configuration files.
Encrypt user input passwords. **http://blogs.splunk.com/2011/03/15/storing-encrypted-credentials/**	Store clear text passwords in .conf files.
Parameterize indexes so that they can be easily changed	Hard code indexes in your searches
Use the CIM add-on **http://docs.splunk.com/Documentation/CIM/latest/User/Overview**	
Place all .conf files in default $SPLUNK_HOME/etc/apps/<your_app>/default	Leave any content in $SPLUNK_HOME/etc/apps/<your_app>/local
Set default permissions in: $SPLUNK_HOME/etc/apps/<your_app>/metadata/default.meta	Have a local.meta file located in: $SPLUNK_HOME/etc/apps/<your_app>/metadata

Data Collection

Do	Don't
Support multiple platforms.	Code for a single OS.
Use scripting language utilities such as os.path.join() and the special environment variable $SPLUNK_HOME to construct paths in scripts.	Hard code script paths.
Use key=value pairs in writing to log files (if you have control of the logging output).	Use name abbreviations.
Throttle how much data is collected at one time from an API.	Overwhelm a system by pulling exorbitant amounts of data at one time from an API.
Use logging and error trapping in scripts.	

Packaging Applications

Do	Don't
Follow the guidelines found at **http://docs.splunk.com/Documentation/Splunk/latest/AdvancedDev/PackageApp**	Leave any hidden files in the app such as Mac's ._ files.
Include a screen shot of your application in the correct location.	
Let the user choose which inputs are enabled for their environment.	Enable all inputs by default if not necessary.
Use a build automation tool such as Apache Ant if necessary to ensure a clean build/package.	Leave anything in: $SPLUNK_HOME/etc/apps/<app>/local directory $SPLUNK_HOME/etc/apps/<app>/metadata/local.meta
Ensure the appropriate settings are set in app.conf	
Document your app with a README.txt file	
Test your application on a clean system	

Appendix B – Application/Add-on Checklist

Application Setup Instructions

- ☐ README located in the root directory of your application with basic instructions.
- ☐ Detailed instructions located on a dashboard within the application.
- ☐ Instructions do not direct the user to store clear text passwords anywhere.
- ☐ Setup screen (optional) that prompts the user for setup parameters.
    - ☐ Setup mechanism encrypts passwords.

Application Packaging

- ☐ APP.CONF specifies:
    - ☐ id - this cannot be changed once created.
    - ☐ version - this field can contain trailing text such as “beta”.
    - ☐ description
- ☐ The “local” directory in the application is either empty or does not exist.
- ☐ Remove metadata/local.meta
- ☐ Ensure metadata/default.meta exports and permissions are set correctly.
- ☐ Remove any files from the lookups directory that are not static. For instance, some scheduled saved searches generate files in this directory that are specific to the environment.
- ☐ No hidden files contained in the application directory structure.
- ☐ All XML files are valid.
  http://docs.splunk.com/Documentation/Splunk/latest/AdvancedDev/AdvancedSchemas#Validation

Application Performance

- ☐ Intervals on inputs.conf reviewed. For example, inventory data should be polled less frequently than performance data.
- ☐ Scripted or modular inputs are verified to not have a negative impact on the host system.
- ☐ Amount of data requested from 3rd party APIs is throttled (if applicable). For example, requesting 1 million records via REST may be bad.
- ☐ Saved searches are optimized. For example, dispatch.earliest_time and dispatch.latest_time should be set by default in savedsearches.conf.
- ☐ Measured resources when running the application:
  - ☐ Load average
  - ☐ % CPU
  - ☐ Memory usage

Application Portability

- ☐ Searches do not contain hard coded index names.
- ☐ Application conforms to the Splunk Common Information Model (CIM) (optional).
- ☐ Eventgen.conf created (optional).
  - ☐ Eventgen sample data stored in the samples directory.
  - ☐ Eventgen data anonymized.

Application Security

- ☐ Application does not open outgoing connections.
- ☐ Application does not use IFRAME.
- ☐ Application and/or Add-ons do not have pre-compiled Python code.
- ☐ Application does not contain executable files.

Technology Add-Ons

- ☐ Technology add-ons are stored in your application folder under appserver/addons.
- ☐ Scripted or modular inputs are verified to not have a negative impact on the host system.
- ☐ No hard coded paths in scripted or modular inputs.
- ☐ Logging mechanism used in data collection.
- ☐ Error trapping used in data collection.

Testing

- ☐ Tested Application on a clean install of Splunk to ensure everything is self-contained.
- ☐ Tested Application on Splunk running in *nix
- ☐ Tested Application on Splunk running in Windows®
- ☐ Tested Add-ons on multiple platforms (optional).
- ☐ Application tested with multiple Splunk account roles.
- ☐ Application tested on multiple versions of Splunk (optional).
- ☐ Open Application in non-Flash browser.
- ☐ Open Application in browsers supported by Splunk.

Legal

- ☐ No use of Splunk in trademark infringing way.
- ☐ Developer's agreement to "Developer Distribution License"
- ☐ App has a valid EULA