Splunk

Splunk is a proprietary data mining product. **From Wikipedia**:

> Splunk is software to search, monitor and analyze machine-generated data by applications, systems and IT infrastructure at scale via a web-style interface. Splunk captures, indexes and correlates real-time data in a searchable repository from which it can generate graphs, reports, alerts, dashboards and visualizations.
> Splunk aims to make machine data accessible across an organization and identifies data patterns, provides metrics, diagnoses problems and provides intelligence for business operation. Splunk is a horizontal technology used for application management, security and compliance, as well as business and web analytics.

Splunk is licensed based on MB of data indexed per day. The free license allows up to 500 MB of data per day, but it is missing a few features such as access control, alerts / monitoring and PDF generation

Splunk provides a fairly high-level search interface to data. Raw data is parsed by sets of regular expressions (many of them built-in) to extract fields; these fields then allow a query language that has fairly unique semantics but will be recognisable to user familiar with SQL or other structured data querying languages.

Splunk's online documentation is open to the public and reasonably comprehensive. Much of it is in Unix-like man pages, particularly for the search and configuration reference files. This article will focus on lesser known features or failures of Splunk, and how to run it healthily in Arch Linux.

Contents

Installation

There is now a `splunk (https://aur.archlinux.org/packages/splunk/)`AUR package in the **AUR** which will create the `splunk` user and group, install Splunk, and install a systemd unit file.

There is also a `splunkforwarder (https://aur.archlinux.org/packages/splunkforwarder/)`AUR package which will install the Splunk Universal Forwarder.

Manual

Log into **splunk.com (http://www.splunk.com)** to get the download link for Splunk or the Splunk Universal Forwarder and wget it:

```
$ wget -O splunk.tgz <url goes here>
```

Extract the tarball:

```
$ tar -xvf splunk.tgz
```

For a simple deployment, it is conventional to move the extracted directory to `/opt/`.

Splunk's installation directory is commonly referred to as $SPLUNKHOME. You may set it in `.bashrc` and add it to your path:

```
export SPLUNK_HOME=/opt/splunk
export PATH=$PATH:$SPLUNK_HOME/bin
```

It has a reasonably robust CLI interface, and all the configuration is stored in `.ini` style configuration files.

Running Splunk

Splunk has two main components: the `splunkd` daemon and the `splunkweb` service, a `cherrypy` web application.

If using the AUR package, you can run both by **starting** the systemd `splunk` service.

Alternatively run with the Splunk binary:

```
# splunk start
```

Performance

The conventional wisdom in the Splunk community is that Splunk's performance is heavily IO-bound, but this may be an assumption based on traditional use cases for Splunk. There are certain powerful operations with a single-threaded implementation that spend most of their time occupying a single core while barely hitting the disk.

It is easy to see what Splunk is doing if you monitor these:

```
$ iostat -d -x 5
$ top
```

A sign that you have a bottleneck caused by Splunk's implementation details - rather than your own hardware - is a pattern where you mostly see a single core at 100% with little-to-no disk usage, with sporadic spikes of activity by splunkd on an extra core as it hits the disk for more events.

If you are having trouble getting Splunk to utilise your hardware, consider the following factors:

Search Semantics

Much of Splunk's search functionality is powered a **MapReduce** implementation. It is powerful and very useful in a distributed environment, but the high-level search language abstractions can mask a number of mistakes that essentially force a `reduce` operation early in the pipeline, which removes Splunk's ability to parallelise its operations, whether in a distributed environment or on a single instance.

A simple rule of thumb is that any operation which (in a naive implementation) would need to see every 'event' to do its work will not be parallelised. This applies particularly to the **transaction (http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/tran**

saction) command, which is one of Splunk's most useful features.

Distributed Environment

Splunk is designed to be run in a distributed environment; the assumption is generally that each instance is on a separate machine, but on a machine with four or more logical cores and a fast disk (such as a solid-state drive), it is possible to improve performance significantly by setting up several Splunk instances.

If you run multiple Splunk instances on a single machine, there are a couple of settings you need to pay attention to:

- serverName - in the [general] stanza of `server.conf`
- mgmthostport and httpport for splunkd and splunkweb respectively - in the [settings] stanza of `web.conf`

You may set up a third instance as a 'search head' which dispatches searches to the indexers ('search peers'), or you can set both indexers to be aware of the other.

If you are using a dedicated search head, you may as well disable the web interface on the indexers:

```
# splunk disable webserver
# splunk restart
```

Indexing

Multiple indexers means splitting the data between them. Either set up their `inputs.conf` to monitor different subsets of your source data, or set up a separate 'forwarder' instance that uses the auto load-balancing features to round-robin between them.

Do not try to make two indexers read from the same index via a static path in `indexes.conf` or a symlink - this will push responsibility for deduplicating results onto the search head and mitigate the advantage of distributing the work in the first place.

Debugging and Administration

Splunk's **CLI (http://docs.splunk.com/Documentation/Splunk/latest/Admin/Aboutthecli)** is under-utilised.

It is very useful for debugging your config files:

```
# splunk btool props list
```

Or for adding one-off files for testing, rather than having to configure `inputs.conf` to monitor a directory:

```
# splunk add oneshot <file> -sourcetype mysourcetype -host myhost -index myindex
```

Take care to use a special test index when testing - it is generally not possible to remove data from an indexing once it has been added without wiping it entirely.

Custom Commands

Per **this (http://docs.splunk.com/Documentation/Splunk/latest/Developer/SearchScripts)**, Splunk allows the user to call out to arbitary Python or Perl scripts during the search pipeline. This is useful for overcoming the limitations of Splunk's framework, looking up external data sources, and so on. It is also a shortcut to building macros that will automatically push data to other locations or perform arbitrary jobs outside of what Splunk is capable of.

The Splunk documentation, as well as the interface, is sprinkled with warnings that using custom commands will seriously affect search performance. In reality, as long as the search command is not doing something stupid, a custom command generally has a very low footprint, and is executed in a separate process that can use CPU and memory resources while Splunk is mostly bound to a single core. Splunk will repeatedly spawn custom commands with chunks of the data (unless `streaming = false`, in which case the command gets the entire data set) and do its own work while waiting for the external script to output its results and exit.

Splunk comes with a pre-packaged Python 2.7.2 binary, and will not execute commands with the system Python installation. This can make it difficult to use packages installed via `pip` or `easy_install`, or your own libraries.

There is nothing to stop you from using calls like `fork` and/or `execv` to get around this limitation and load the system Python installation. Alternatively, use it to process the data in a faster environment, whether with a compiled program or just a faster Python interpreter such as pypy.

Configuration

The guide to **commands.conf (http://docs.splunk.com/Documentation/Splunk/latest/Admin/commandsconf)** is somewhat misleading. In particular:

```
streaming = [true|false]
 * Specify whether the command is streamable.
 * Defaults to false.
```

The 'streaming' here actually just tells Splunk whether it is safe for it to repeatedly spawn your command with arbitrarily-sized (often in the realm of 50K rows) discrete chunks of the data it is passing through; it will not tell the default `splunk.Intersplunk` library to actually provide a streaming interface to the data as you work with it.

Library API

There is no real documentation for the Splunk library available to the built-in interpreter. Try inspecting the module directly:

```
$SPLUNK_HOME/bin/splunk cmd python
#(in python interpreter)
import splunk.Intersplunk
help(splunk.Intersplunk)
```

The source for splunk.Intersplunk shows that it essentially parses the entire set of input from the process' stdin before offering the data to the command as such. Unless the command needs to have the entire data set to do its work - generally only a small subset of use cases - this is extremely inefficient.

The library is easy to replace. The data passed in from Splunk contains several header lines with `key`:`value` pairs, followed by a newline, followed by a header row and the data proper. In Python, read in the header rows and store or discard them then use a `csv.Reader` or `csv.DictReader` object - to handle the data a row at a time, with a `csv.Writer` or `csv.DictWriter` to push resulting rows back into the Splunk search pipeline.

Retrieved from "https://wiki.archlinux.org/index.php?title=Splunk&oldid=492803"