Extending SPL with Custom Search Commands

Jacob Leverich | Director of Engineering

2017/08/11 | Washington, DC
Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC.

The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward looking statements we may make. In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

Splunk, Splunk>, Listen to Your Data, The Engine for Machine Data, Splunk Cloud, Splunk Light and SPL are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names, or trademarks belong to their respective owners. © 2017 Splunk Inc. All rights reserved.
Who am I?

- Splunker for 3 years, based in San Francisco
- Engineering manager in Splunk’s Incubation team
- Implemented Search Command Protocol Version 2
- Die-hard Longhorns fan
Agenda

• Introduction to Custom Search Commands
• How do Custom Search Commands work?
  – High-level concepts
  – Low-level details
• Types of Search Commands
• How to create new Custom Search Commands
• Wrap-up
Introduction to Custom Search Commands
What is a Custom Search Command?

- A user-defined SPL command.
search

| timechart (span=1h) sum(bytes) as bytes_per_hour
| eventstats (avg(bytes_per_hour) as avg, stdev(bytes_per_hour) as sd)
| where bytes_per_hour > avg+2*sd

717,267 events (before 7/31/16 12:31:47.000 PM) No Event Sampling
What is a Custom Search Command?

- A user-defined SPL command.

- Can be used to extend the SPL language!
Who uses Custom Search Commands?

- Partners
  - Concanon, etc.

- Customers
  - Use-case specific analytics

- Splunk!
  - `predict` command
  - IT Service Intelligence
  - Enterprise Security
  - DB Connect
  - Machine Learning Toolkit

- Anyone who wants to extend the Splunk platform
  - Integration with 3rd party services
  - Implementation of custom logic
How do Custom Search Commands work?
How do Custom Search Commands work?

1. When parsing SPL, splunkd interrogates each command.
   “Are you a Custom Search Command?”

2. If so, spawn external process and allow it to parse arguments.

3. During search, pipe search results through external process.
Parsing #1: Split search into commands

| inputlookup geo_attr_us_states.csv | GOCRAZY | head 5
Parsing #2: Look for custom search commands

| inputlookup geo_attr_us_states.csv | GOCRAZY | head 5

commands.conf

[gocrazy]
...

inputlookup geo_attr_us_states.csv

GOCRAZY

head 5
Parsing #3: Spawn external process

| inputlookup geo_attr_us_states.csv | GOCRAZY | head 5

$SPLUNK_HOME/bin/python gocrazy.py
Parsing #4: Let external process parse arguments

| inputlookup geo_attr_us_states.csv | GOCRAZY | head 5

```
$inputlookup geo_attr_us_states.csv
GOCRAZY
head 5
```

```
$SPLUNK_HOME/bin/python gocrazy.py
```
Search: Pipe results through external process

| inputlookup geo_attr_us_states.csv | GOCRAZY | head 5

$SPLUNK_HOME/bin/python gocrazy.py
Recap: high-level concepts

- Enable you to register new SPL commands, extend the language.

- Allow you to intercept and modify search results during a search.
  - CSV in → CSV out

- Implemented as an external process (i.e. a program you write).
  - Typically written in Python.
Custom Commands: low-level details

- How results are exchanged between splunkd and external process
- “Types” of search commands
splunkd ↔ custom command

- There are two “protocols” for custom commands:
  - Version 1, legacy protocol used by Intersplunk.py (available since Splunk 3.0)
  - Version 2, new protocol used by Python SDK (available since 6.3)
  - In both protocols, all communication over stdin/stdout

- Version 2 protocol
  - Spawns external process once, streams results through chunk by chunk
  - Simple commands.conf configuration
    - “chunked=true”
  - Support for platform-specific programs

- Version 1 protocol
  - Spawns external process for each chunk of search results (!)
  - “Transforming” commands limited to 50,000 events
# Search Command protocol comparison

<table>
<thead>
<tr>
<th>Protocol</th>
<th>APIs</th>
<th>Performance</th>
<th>Scalability</th>
<th>Simple configuration</th>
<th>Platform-specific programs</th>
<th>Programming languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1 (legacy)</td>
<td>Intersplunk.py, Python SDK</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>Python</td>
</tr>
<tr>
<td>Version 2</td>
<td>Python SDK</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Python, arbitrary binaries</td>
</tr>
</tbody>
</table>
Search Command Protocol Version 2

• Transaction-oriented
  • splunkd sends a command, external process responds with reply

• Simple bi-directional transport protocol:
  • ASCII transport header
  • JSON metadata payload
  • CSV search results payload

• Every search starts with a “getinfo” command (capability exchange)
• Subsequently, issues “execute” commands with search results
Transport “chunk”

Metadata length
chunked 1.0, 22, 54
{"“action”: “execute”} _raw,a,b,c
hello,0,1,2
everyone,3,4,5
howareyou,6,7,8

Data length
Transport header
Metadata (JSON)
Data payload (CSV)
Example: GOCRAZY

| inputlookup geo_attr_us_states.csv | head 5 | GOCRAZY |

chunked 1.0,22,106
{"action": "execute"}
state_code,state_fips,state_name
AL,01,Alabama
AK,02,Alaska
AZ,04,Arizona
AR,05,Arkansas
CA,06,California

$SPLUNK_HOME/bin/python
gocrazy.py

chunked 1.0,18,106
{"finished": true}
dste_aecot,pste_asfit,mste_aenat
LA,10,aaalbmA
KA,20,laaskA
ZA,40,iaorznA
RA,50,Akaasnsr
AC,60,iCifolarna
“What kind of command are you?”

splunkd

spawns external process

getinfo command

getinfo reply

execute command

execute reply

execute command

execute reply

: closes stdin pipe

kills external process

external process

“Hey! I’m a streaming command!”
“getinfo” command

- Metadata in the getinfo command sent by splunkd:
  - Command arguments
  - Full SPL query string
  - Execution context (app, user)
  - Search sid
  - splunkd URI and auth token (for making REST requests)

- Metadata in the custom command’s reply:
  - Type of search command (streaming/stateful/reporting/etc.)
  - Which fields splunkd should extract (required fields)
  - Whether or not it generates results (e.g. must be first search command)
Sample “getinfo” metadata

```
{
"action": "getinfo",
"streaming_command_will_restart": false,
"searchinfo": {
  "earliest_time": "0",
  "raw_args": [
    "LinearRegression", "petal_length", "from", "petal_width"
  ],
  "session_key": "...",
  "maxresultrows": 50000,
  "args": [
    "LinearRegression", "petal_length", "from", "petal_width"
  ],
  "dispatch_dir": "/Users/jleverich/builds/conf_mlapp_demo/var/run/splunk/dispatch/1475007525.265",
  "command": "fit",
  "latest_time": "0",
  "sid": "1475007525.265",
  "splunk_version": "6.5.0",
  "username": "admin",
  "search": ":%7C%20inputlookup%20iris.csv%20%7C%20fit%20LinearRegression%20petal_length%20from%20petal_width",
  "splunkd_uri": "https://127.0.0.1:8090",
  "owner": "admin",
  "app": "Splunk_ML_Toolkit"
},
"preview": false
}
```
“execute” command

- Metadata in execute command sent by splunkd
  - Whether or not preceding commands are “finished”

- Metadata in the custom command’s reply:
  - Whether or not this command is “finished”

- splunkd and search commands negotiate completion of search
  - Both must indicate “finished” = True
Types of Search Commands
Types of Search Commands

- “Streaming” commands
- “Stateful Streaming” commands
- “Transforming” commands
  - “Events” commands
  - “Reporting” commands
“Streaming” commands

- Process search results one-by-one
  - Can’t maintain global state
  - Must not re-order search results

- Eligible to run at Indexers
  - Can run in parallel on Indexers

- Examples:
  - eval
  - where
  - rex
“Streaming” command example

... | eval foo="bar" | ...

Remote results

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>jumps</td>
<td>dog</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick</td>
<td>over</td>
<td>oops</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>brown</td>
<td>the</td>
<td>too</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>fox</td>
<td>lazy</td>
<td>many</td>
</tr>
</tbody>
</table>

Indexers

Final search results

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>jumps</td>
<td>dog</td>
<td>bar</td>
</tr>
<tr>
<td>quick</td>
<td>over</td>
<td>oops</td>
<td>bar</td>
</tr>
<tr>
<td>brown</td>
<td>the</td>
<td>too</td>
<td>bar</td>
</tr>
<tr>
<td>fox</td>
<td>lazy</td>
<td>many</td>
<td>bar</td>
</tr>
</tbody>
</table>
“Stateful Streaming” commands

- Process search results one-by-one
  - **Can** maintain global state
  - Must not re-order search results

- Only run at Search Head

- Examples:
  - accum
  - streamstats
  - dedup
“Stateful Streaming” command example

... | accum foo | ...

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>jumps</td>
<td>dog</td>
<td>1</td>
</tr>
<tr>
<td>quick</td>
<td>over</td>
<td>oops</td>
<td>1</td>
</tr>
<tr>
<td>brown</td>
<td>the</td>
<td>too</td>
<td>1</td>
</tr>
<tr>
<td>fox</td>
<td>lazy</td>
<td>many</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>jumps</td>
<td>dog</td>
<td>1</td>
</tr>
<tr>
<td>quick</td>
<td>over</td>
<td>oops</td>
<td>2</td>
</tr>
<tr>
<td>brown</td>
<td>the</td>
<td>too</td>
<td>3</td>
</tr>
<tr>
<td>fox</td>
<td>lazy</td>
<td>many</td>
<td>4</td>
</tr>
</tbody>
</table>
“Events” commands

- Process search results as a whole
  - May re-order search results
  - Typically maintain all fields in each event, especially:
    - _raw, _time, index, sourcetype, source, host

- Only run at Search Head
- May run several times for “preview”

- Examples:
  - sort
  - eventstats
“Events” command example

... | sort field_A | ...

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>jumps</td>
<td>dog</td>
<td>1</td>
</tr>
<tr>
<td>quick</td>
<td>over</td>
<td>oops</td>
<td>2</td>
</tr>
<tr>
<td>brown</td>
<td>the</td>
<td>too</td>
<td>3</td>
</tr>
<tr>
<td>fox</td>
<td>lazy</td>
<td>many</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>brown</td>
<td>the</td>
<td>too</td>
<td>3</td>
</tr>
<tr>
<td>fox</td>
<td>lazy</td>
<td>many</td>
<td>4</td>
</tr>
<tr>
<td>quick</td>
<td>over</td>
<td>oops</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>jumps</td>
<td>dog</td>
<td>1</td>
</tr>
</tbody>
</table>
“Reporting” commands

- Process search results as a whole
  - Typically transform the results (e.g. aggregate, project, summarize, etc.)

- Only run at Search Head

- May run several times for “preview”

- Results show up in the “Statistics” tab

- Examples:
  - stats
  - timechart
  - transpose
"Reporting" command example

... | stats count | ...

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>jumps</td>
<td>dog</td>
<td>1</td>
</tr>
<tr>
<td>quick</td>
<td>over</td>
<td>oops</td>
<td>2</td>
</tr>
<tr>
<td>brown</td>
<td>the</td>
<td>too</td>
<td>3</td>
</tr>
<tr>
<td>fox</td>
<td>lazy</td>
<td>many</td>
<td>4</td>
</tr>
</tbody>
</table>

count

4
Beware of large result sets!

- “Events” and “Reporting” commands process results as a whole.
  - May contain 1,000,000s of search results!
  - Write Streaming or Stateful commands instead when possible.

- Build-in capacity limits, or spill results to disk when necessary.
Streaming “pre-op”

- Commands may specify a “pre-op” to prepend in SPL

```plaintext
... | stats count | ... ➔ ... | prestats count | stats count | ...
```

- Communicated to splunkd in getinfo metadata (streaming_preop)
- Useful to parallelize computation, reduce volume of data transfer
- Must be “Streaming” (i.e., may run at Indexers)
Implementing Custom Search Commands with the Splunk SDK for Python
Basic steps to create a search command

1. Create an “App”
2. Deploy the Python SDK for Splunk in the bin directory
3. Write a script for your Custom Search Command
4. Register your command in commands.conf
5. Restart Splunk Enterprise
6. (optional) Export the command to other apps
Create an “App”
Deploy the Python SDK in the bin directory

```bash
cd $SPLUNK_HOME/etc/apps/MyNewApp/bin

pip install -t . splunk-sdk
```
Write a script for your Custom Search Command

```python
import sys
from splunklib.searchcommands import dispatch, StreamingCommand, Configuration

@Configuration()
class FoobarCommand(StreamingCommand):
    def stream(self, records):
        for record in records:
            record['foo'] = 'bar'
            yield record

if __name__ == '__main__':
    dispatch(FoobarCommand, sys.argv, sys.stdin, sys.stdout, __name__)
```

$SPLUNK_HOME/etc/apps/MyNewApp/bin/foobar.py
Register your command in commands.conf

$SPLUNK_HOME/etc/apps/MyNewApp/default/commands.conf

[foobar]
chunked=true
# filename=foobar.py  ## <--- optional
Restart Splunk Enterprise

$SPLUNK_HOME/bin/splunk restart
Export to other apps (optional)
Export to other apps (optional)

<table>
<thead>
<tr>
<th>Name</th>
<th>Config type</th>
<th>Owner</th>
<th>App</th>
<th>Sharing</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>exevents</td>
<td>commands</td>
<td>No owner</td>
<td>custom_search_example</td>
<td>Permissions</td>
<td>Enabled</td>
</tr>
<tr>
<td>exreport</td>
<td>commands</td>
<td>No owner</td>
<td>custom_search_example</td>
<td>Permissions</td>
<td>Enabled</td>
</tr>
<tr>
<td>exstateful</td>
<td>commands</td>
<td>No owner</td>
<td>custom_search_example</td>
<td>Permissions</td>
<td>Enabled</td>
</tr>
<tr>
<td>exstream</td>
<td>commands</td>
<td>No owner</td>
<td>custom_search_example</td>
<td>Permissions</td>
<td>Enabled</td>
</tr>
<tr>
<td>gocrazy</td>
<td>commands</td>
<td>No owner</td>
<td>custom_search_example</td>
<td>Permissions</td>
<td>Enabled</td>
</tr>
<tr>
<td>levenshtein</td>
<td>commands</td>
<td>No owner</td>
<td>custom_search_example</td>
<td>Permissions</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
Export to other apps (optional)
import sys
from splunklib.searchcommands import dispatch, StreamingCommand, Configuration

@Configuration()
class ExStreamCommand(StreamingCommand):
    def stream(self, records):
        for record in records:
            record['foo'] = 'bar'
            yield record

if __name__ == '__main__':
    dispatch(ExStreamCommand, sys.argv, sys.stdin, sys.stdout, __name__)
import sys
from splunklib.searchcommands import dispatch, StreamingCommand, Configuration

@Configuration(local=True)
class ExStatefulCommand(StreamingCommand):
    def stream(self, records):
        for record in records:
            record['foo'] = 'bar'
            yield record

if __name__ == '__main__':
    dispatch(ExStatefulCommand, sys.argv, sys.stdin, sys.stdout, __name__)
import sys
from splunklib.searchcommands import dispatch, EventingCommand, Configuration

@Configuration()
class ExEventsCommand(EventingCommand):
    def transform(self, records):
        l = list(records)
        l.sort(key=lambda r: r['_raw'])
        return l

if __name__ == '__main__':
    dispatch(ExEventsCommand, sys.argv, sys.stdin, sys.stdout, __name__)
import sys
from splunklib.searchcommands import dispatch, ReportingCommand, Configuration

@Configuration()
class ExReportCommand(ReportingCommand):
    @Configuration()
    def map(self, records):
        return records

    def reduce(self, records):
        count = 0
        for r in records:
            count += 1
        return [{'count': count}]

if __name__ == '__main__':
    dispatch(ExReportCommand, sys.argv, sys.stdin, sys.stdout, __name__)
Custom commands are programs that run on Splunk instances.

- **BEWARE UNVALIDATED INPUT!**
  - Sanitize user arguments AND search results

Use role-based access control to restrict access.

Be prepared to handle 1,000,000s of events.

Be excellent to each other.
What Now?

- [https://github.com/splunk/splunk-sdk-python](https://github.com/splunk/splunk-sdk-python)
  - [https://github.com/splunk/splunk-sdk-python/tree/master/examples/searchcommands_app](https://github.com/splunk/splunk-sdk-python/tree/master/examples/searchcommands_app)

- Dev Portal Documentation
  - [http://dev.splunk.com/view/python-sdk/SP-CAAAEU2](http://dev.splunk.com/view/python-sdk/SP-CAAAEU2)

- Contact: Developer Ecosystem Team <devinfo@splunk.com>
Thank You

Don't forget to rate this session in the .conf2017 mobile app
Streaming Commands only serialize required fields

```
{"required_fields": ["fieldX"], ...}
```

**Internal result set**

<table>
<thead>
<tr>
<th>_raw, _time, _cd, _indextime,..., fieldX</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, 1400000000, x:y, 1400000010,..., BOB</td>
</tr>
<tr>
<td>a, 1400000001, x:y, 1400000011,..., JIM</td>
</tr>
<tr>
<td>a, 1400000002, x:y, 1400000012,..., BOB</td>
</tr>
<tr>
<td>a, 1400000003, x:y, 1400000013,..., JIM</td>
</tr>
<tr>
<td>a, 1400000004, x:y, 1400000014,..., JIM</td>
</tr>
<tr>
<td>a, 1400000005, x:y, 1400000015,..., BOB</td>
</tr>
<tr>
<td>a, 1400000006, x:y, 1400000016,..., JIM</td>
</tr>
<tr>
<td>a, 1400000007, x:y, 1400000017,..., BOB</td>
</tr>
<tr>
<td>a, 1400000008, x:y, 1400000018,..., BOB</td>
</tr>
<tr>
<td>a, 1400000009, x:y, 1400000019,..., JIM</td>
</tr>
</tbody>
</table>

**External result set**

<table>
<thead>
<tr>
<th>_chunked_idx, fieldX</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, BOB</td>
</tr>
<tr>
<td>1, JIM</td>
</tr>
<tr>
<td>2, BOB</td>
</tr>
<tr>
<td>3, JIM</td>
</tr>
<tr>
<td>4, JIM</td>
</tr>
<tr>
<td>5, BOB</td>
</tr>
<tr>
<td>6, JIM</td>
</tr>
<tr>
<td>7, BOB</td>
</tr>
<tr>
<td>8, BOB</td>
</tr>
<tr>
<td>9, JIM</td>
</tr>
</tbody>
</table>
“Right outer-join” on required fields

- Supports
  - Removing events
  - Adding events
  - Editing fields
  - Adding fields
- Can’t re-order events
Performance comparison

2.5 million events

<table>
<thead>
<tr>
<th></th>
<th>Splunk</th>
<th>Protocol v1</th>
<th>Protocol v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Echo</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Echo (CSV)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Echo (selected)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>where</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
“Streaming” command example

... | eval foo="bar" | ...

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>jumps</td>
<td>dog</td>
</tr>
<tr>
<td>quick</td>
<td>over</td>
<td>oops</td>
</tr>
<tr>
<td>brown</td>
<td>the</td>
<td>too</td>
</tr>
<tr>
<td>fox</td>
<td>lazy</td>
<td>many</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>jumps</td>
<td>dog</td>
<td>bar</td>
</tr>
<tr>
<td>quick</td>
<td>over</td>
<td>oops</td>
<td>bar</td>
</tr>
<tr>
<td>brown</td>
<td>the</td>
<td>too</td>
<td>bar</td>
</tr>
<tr>
<td>fox</td>
<td>lazy</td>
<td>many</td>
<td>bar</td>
</tr>
</tbody>
</table>