Performance and Scale testing of Splunk Enterprise

October 21, 2019

Somu Rajarathinam

Technical Director | Pure Storage

@purelydb | www.somu.us

splunk> .conf19

Performance & Scale testing of Splunk

- What, Why & How scale test Splunk?
- Data Generation at massive scale
- Concurrent Searches framework

What is Scalability Testing?

- Type of non-functional testing
- Determine how application scales with increasing workload
- Identify server-side robustness and degradation if any
- Determine end user experience under increased workload

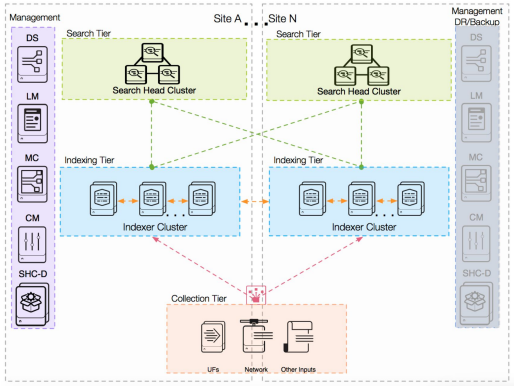
Why Scale test?

Data volume growth

- Business growth
- M & A

Infrastructure upgrade

- Technology shift



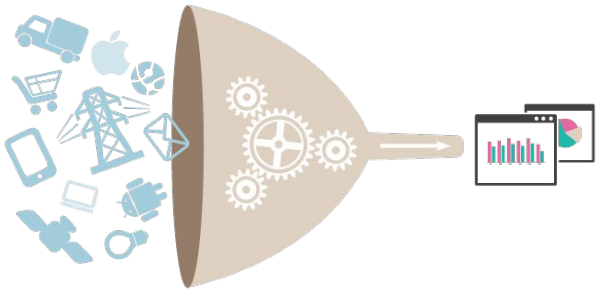
Deployment model change

- Single instance to cluster
- Single site to Multi-site

How to Scale Test?

Three major components of Splunk

Collect (Forwarder)



Store/Ingest (Indexer)

Search (Search Head)

How to Scale Test?

- Ideally with real-world data
- Generate synthetic data for measuring Splunk ingest performance
- Perform search tests from the data generated

Our Requirements

- Ingest test
  - 4 indexes
  - Total 64TB of data to be ingested (16 TB of data per index)
- Search test
  - Concurrent users test (120 users)
  - Sparse and Rare search to test I/O overhead

Synthetic Data Requirements

- Generate a log file for a given GB/day parameter
- Option to generate the log files for a given number of days
- Option to generate the per day log file in parallel to speed up
- Add a keyword for every 1, 10, 100, 1k, 10k, 100k, 1m, 10m, 100m events

- Wanted 1 TB/day per index across 16 days totaling 16 TB (64 TB across 4 indexes)
- apache log format
  - Sourcetypes : access_combined | access_common
  - Timestamp to be sequential within a day
  - Randomize all fields except the timestamp

Data Generation

- Preferred Toolkit – Eventgen
  - Advantages
    - Allows templates
    - Portable between applications
    - Abundant event or transaction types
    - Numerous output formats
  - Challenges
    - Parallelism
      - Out of sequence events
        - CPU overhead with increased generators
    - Massive scale
      - Generator Queue full error

https://github.com/splunk/eventgen

Data Generation at Massive Scale

- Used “golang”
  - Advantages
    - Concurrency
    - Cross-platform
    - Effective memory management through garbage collector
  - Disadvantages
    - Less flexibility
    - Lack of 3rd party modules
- Used Brian Voelker’s random fake data generator gofakeit routines https://github.com/brianvoe/gofakeit
- Wrapped script on bash

Code Snippets - apclog

```
// Generate generates the logs with given options
func Generate(option *Option) error {
    frac_ns := 1e9 * (option.Created - math.Floor(option.Created))
    created := time.Unix(int64(option.Created), int64(frac_ns))
    writer := os.Stdout

    ival := time.Duration(option.Sleep)
    index := option.Index

    // Generates the logs until the certain number of lines is reached
    for line := 0; line < option.Number; line++ {
        every := DeriveEvery(index)
        log := NewLog(option.Format, created, every)
        writer.Write([]byte(log + "\n"))

        created = created.Add(ival)
        index++
    }

        return nil
}

// NewLog creates a log for given format
func NewLog(format string, t time.Time, every string) string {
        switch format {
        case "apache_common":
                return NewApacheCommonLog(t, every)
        case "apache_combined":
                return NewApacheCombinedLog(t, every)
        default:
                return ""
        }
}
```

```
func DeriveEvery(count int64) string {
    if count % 1e12 == 0 {
        return "every1t"
    } else if count % 1e11 == 0 {
        return "every1 every10 every100 every1k every10k every10b every1m every10m eve
ry100m every1b every10b every100b"
    } else if count % 1e10 == 0 {
        return "every1 every10 every100 every1k every10k every10b every1m every10m eve
ry100m every1b every10b"
    } else if count % 1e9 == 0 {
        return "every1 every10 every100 every1k every10k every100k every1m every10m ev
ery100m every1b"
    } else if count % 1e8 == 0 {
        return "every1 every10 every100 every1k every10k every100k every1m every10m ev
ery100m"
    } else if count % 1e7 == 0 {
        return "every1 every10 every100 every1k every10k every100k every1m every10m"
    } else if count % 1e6 == 0 {
        return "every1 every10 every100 every1k every10k every100k every1m"
    } else if count % 1e5 == 0 {
        return "every1 every10 every100 every1k every10k every100k"
    } else if count % 1e4 == 0 {
        return "every1 every10 every100 every1k every10k"
    } else if count % 1e3 == 0 {
        return "every1 every10 every100 every1k"
    } else if count % 1e2 == 0 {
        return "every1 every10 every100"
    } else if count % 1e1 == 0 {
        return "every1 every10"
    } else {
        return "every1"
    }
}
```

```
const (
// ApacheCommonLog :   {host} {user-identifier} {auth-user-id} [{datetime}] "{method}
{request} HTTP/1.0" {response-code} {bytes} {every}
        ApacheCommonLog = "%s - %s %d [%s] \"%s %s\" %d %d %s"
// ApacheCombinedLog : {host} {user-identifier} {auth-user-id} [{datetime}] "{method}
{request} HTTP/1.0" {response-code} {bytes} "{referrer}" "{agent}" {every}
        ApacheCombinedLog = "%s - %s %d [%s] \"%s %s\" %d %d \"%s\" \"%s\" %s"
)

// NewApacheCommonLog creates a log string with apache common log format
func NewApacheCommonLog(t time.Time, every string) string {
        return fmt.Sprintf(
                ApacheCommonLog,
                gofakeit.IPv4Address(),
                gofakeit.Username(),
                gofakeit.Number(0, 1000),
                t.Format(time.RFC3339),
                gofakeit.HTTPMethod(),
                RandResourceURI(),
                gofakeit.StatusCode(),
                gofakeit.Number(0, 30000),
                every,
        )
}

// NewApacheCombinedLog creates a log string with apache combined log format
func NewApacheCombinedLog(t time.Time, every string) string {
        return fmt.Sprintf(
                ApacheCombinedLog,
                gofakeit.IPv4Address(),
                gofakeit.Username(),
                gofakeit.Number(1, 1000),
                t.Format(time.RFC3339),
                gofakeit.HTTPMethod(),
                RandResourceURI(),
                gofakeit.StatusCode(),
                gofakeit.Number(30, 100000),
                gofakeit.URL(),
                gofakeit.UserAgent(),
                every,
        )
}
```

Keywords included into each event that can be used for various search types

gofakeit routines to randomize various fields

splunk> .conf19

Data Generation - apclog

gendata bash script – Wrapper script for apclog that generates data in parallel for given number of days

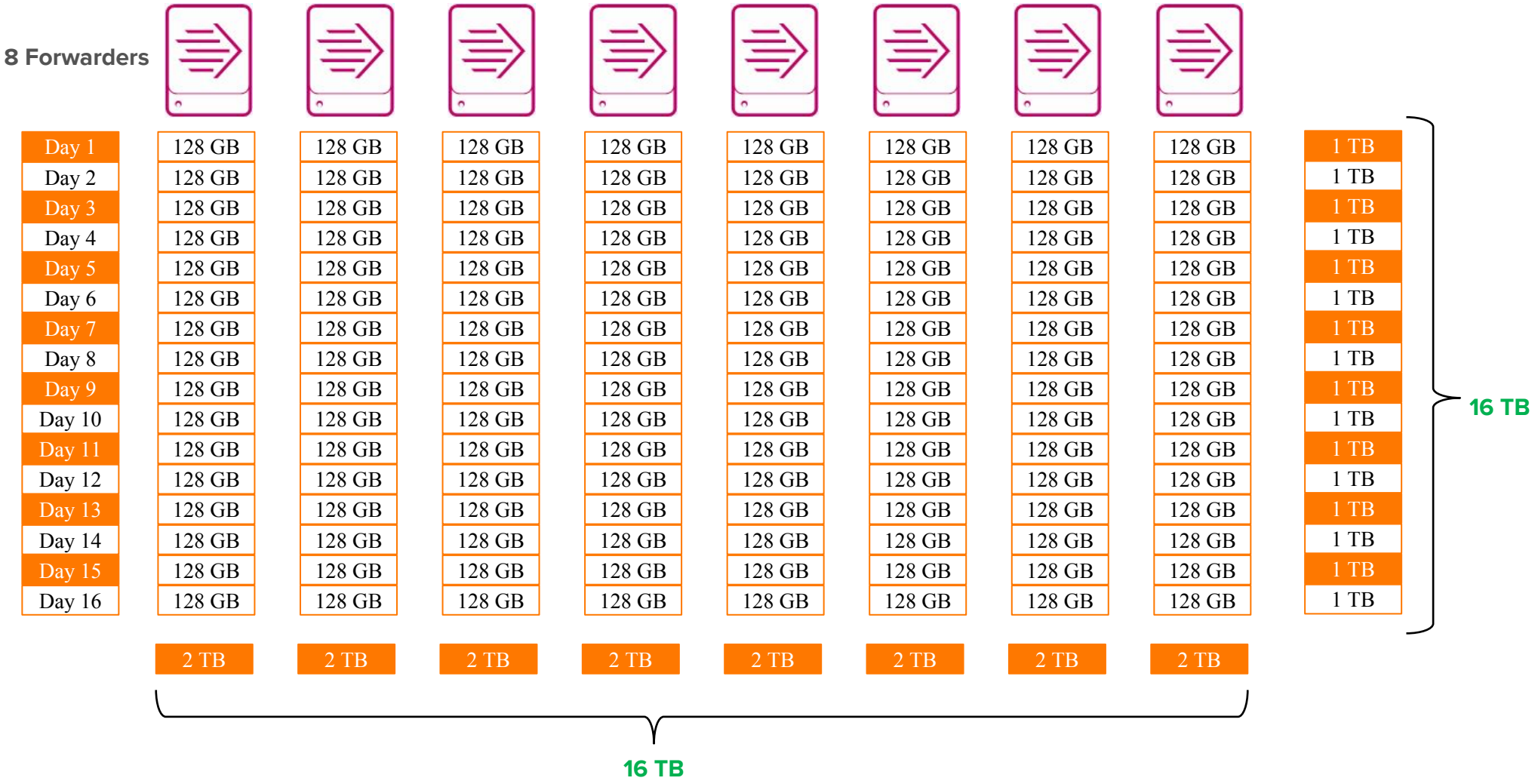
- Divide the daily ingest rate by number of forwarders to determine the data size
- Parallelization is at the day level

```
Usage: gendata.sh GB_per_day_per_forwarder no_of_days no_of_workers
destination
gendata.sh 1024 2 2 /alog/eg01
apclog Options:
  -f, --format string      choose log format. ("apache_common"|"apache_combined") (default "apache_common")
  -n, --number integer     number of events to generate.
  -c, --created numeric    creation start time for each log (in seconds since epoch).
  -s, --sleep numeric      creation time interval for each log (in nanoseconds). It does not actually sleep.
  -i, --index integer      initial index (count) in the overall log sequence
```

```
[root@splunk-eg01 alog]# ./gendata.sh 1024 2 2 /alog/eg01
Generating day 2 -- 09_28_2019
apclog -f apache_combined -c 1569654000 -n 2359467012 -s 18309 -i 1
apclog -f apache_combined -c 1569697200 -n 2359467012 -s 18309 -i 2359467013
Generating day 1 -- 09_29_2019
apclog -f apache_combined -c 1569740400 -n 2359467012 -s 18309 -i 1
apclog -f apache_combined -c 1569783600 -n 2359467012 -s 18309 -i 2359467013
```

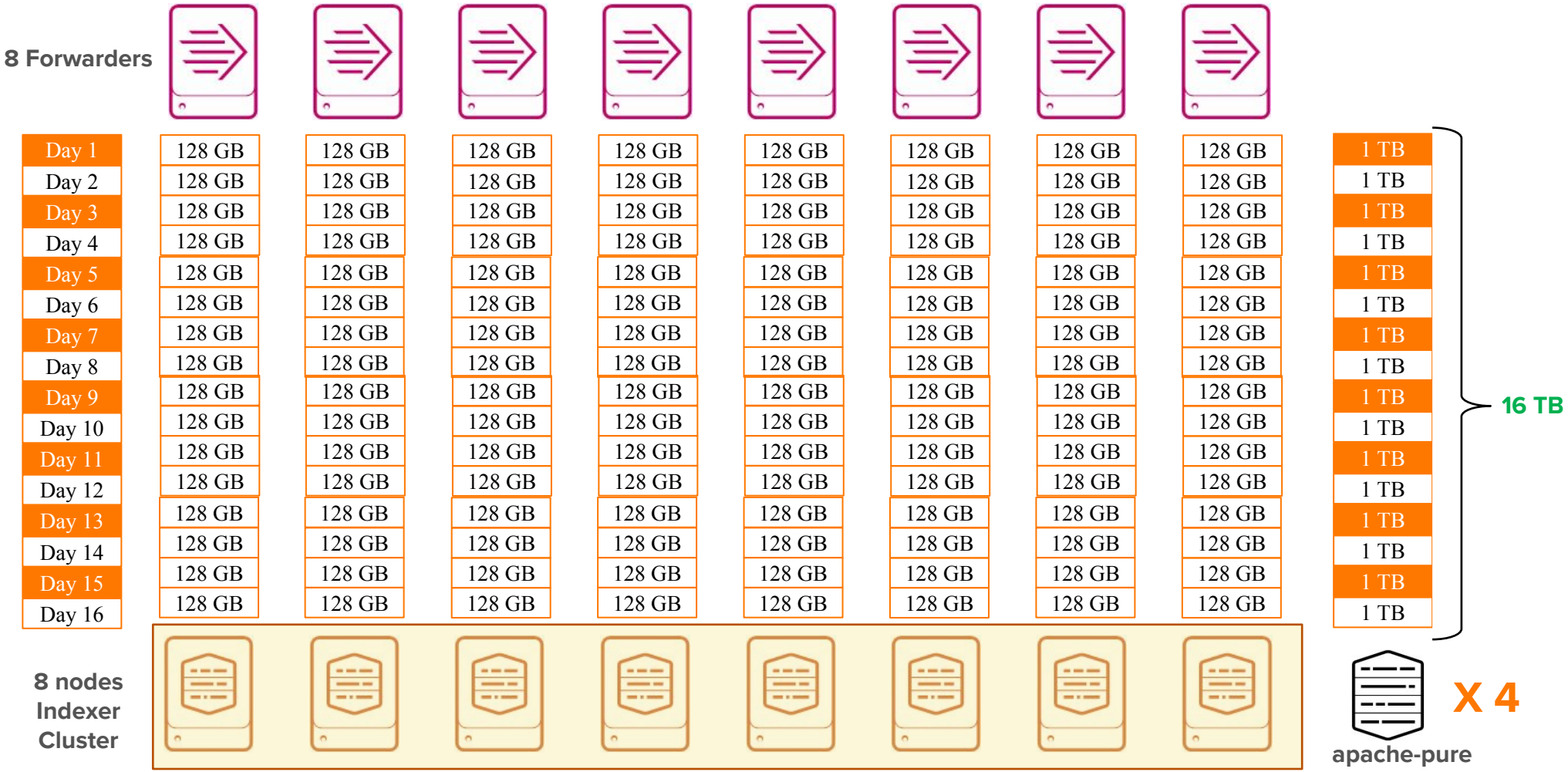
```
# du -sh /alog/eg01/*
1T apache_09_28_2019.log
1T apache_09_29_2019.log
```

Data Generation at Massive Scale



splunk> .conf19

Data Ingest at Massive Scale



Data Generation - Comparison

Method	Threads / Server	Per Server Throughput	Per Hour Throughput	Total Throughput across 8 servers
Eventgen	16 threads / server	8 MB/s	28.125 GB/hr	225 GB/hr
Eventgen	32 threads / server	10 MB/s	35.16 GB/hr	281 GB/hr
GoLang	16 threads / server	364 MB/s	1.25 TB/hr	9.99 TB/hr
GoLang	32 threads / server	851 MB/s	2.92 TB/hr	23.37 TB/hr

Concurrent Searches Framework

Search framework should enable automation of

- Concurrent Searches submission through REST API or CLI
- Collection of search results and metrics right away

Assumption

- Searches have to be listed in a datafile

Search Invocation

- Invoking REST API call through CLI using curl

```
# curl -s -u username:password -k https://searchhead:8089/services/search/jobs -d search="search command"
```

Search Automation Process

1. Read the searches from the file

2. Submit each search through curl and save the job id in an array

3. Run the curl command to get the output for every job that was submitted and print it

Search Jobs Submission

Include the search commands in a datafile

Number of entries in the datafile equals to the number of searches to be performed

Use the keyword from the data ingested based on the type of search to submit

In 4.7 billion events generated for 1 TB/day, there would be

- 4.7 million every1k events
- 470,000 every10k events
- 4700 every1m events
- 47 every100m events
- 4 every1b events

```
'search index=apache-pure1 every10k  starttime="05/22/2019:00:00:00" endtime="05/22/2019:23:59:59"'
'search index=apache-pure1 every10k  starttime="05/23/2019:00:00:00" endtime="05/23/2019:23:59:59"'
'search index=apache-pure1 every10k  starttime="05/24/2019:00:00:00" endtime="05/24/2019:23:59:59"'
'search index=apache-pure1 every10k  starttime="05/25/2019:00:00:00" endtime="05/25/2019:23:59:59"'
'search index=apache-pure2 every10k  starttime="05/22/2019:00:00:00" endtime="05/22/2019:23:59:59"'
'search index=apache-pure2 every10k  starttime="05/23/2019:00:00:00" endtime="05/23/2019:23:59:59"'
'search index=apache-pure2 every10k  starttime="05/24/2019:00:00:00" endtime="05/24/2019:23:59:59"'
'search index=apache-pure2 every10k  starttime="05/25/2019:00:00:00" endtime="05/25/2019:23:59:59"'
'search index=apache-pure3 every10k  starttime="05/22/2019:00:00:00" endtime="05/22/2019:23:59:59"'
'search index=apache-pure3 every10k  starttime="05/23/2019:00:00:00" endtime="05/23/2019:23:59:59"'
'search index=apache-pure3 every10k  starttime="05/24/2019:00:00:00" endtime="05/24/2019:23:59:59"'
'search index=apache-pure3 every10k  starttime="05/25/2019:00:00:00" endtime="05/25/2019:23:59:59"'
```

Search Metrics Collection

Following command provides detailed job data similar to Inspect Job

```
# curl –s –u username:password –k https://searchhead:8089/services/search/jobs/$jobid
```

Useful metrics collected

eventCount

resultCount

scanCount

runDuration

searchTotalBucketsCount

searchTotalEliminatedBucketsCount

index-bucket-hits

index-bucket-miss

```
while [ $ct -lt ${#SIDS[@]} ]; do
  for ((i=0; i < ${#SIDS[@]}; i++)); do
    if [[ "${status[$i]}" =~ "DONE" ]] && [ ${RPTD[$i]} -eq 0 ]; then
      (( ct++ ))
      RPTD[$i]=1
      cmd="curl -s -u user:pwd -k https://srchead:8089/services/search/jobs/${SIDS[$i]} | egrep -i 'updated|resultC
ount|scanCount|eventCount|searchTotalBucketsCount|runduration|searchTotalEliminatedBucketsCount'| sed 's/s:key//g' |sed -r
 's/< name=\"[a-zA-Z]*\">//g'|sed 's/updated//g' |sed 's/<\/>/,/g' |sed 's/<>//g' "
      results[$i]=$( eval $cmd )
      cmd="curl -s -u user:pwd -k https://srchead:8089/services/search/jobs/${SIDS[$i]} |egrep -A3 'command.search.
index.bucketcache.hit|command.search.index.bucketcache.miss'  |grep invocations |sed 's/s:key//g' |sed -r 's/< name=\"[a-z
A-Z]*\">//g'|sed 's/<\/>/,/g' |sed 's/<>//g' "
      bkts[$i]=$( eval $cmd )
      echo $TESTNAME", "${DTIR[$i]}", "${results[$i]}" "${SRCH[$i]} ", " ${bkts[$i]}
    elif [ ${RPTD[$i]} -eq 0 ]; then
       cmd="curl -s -u user:pwd -k https://srchead:8089/services/search/jobs/${SIDS[$i]} | grep 'dispatchState'  |
sed 's/s:key//g' |sed 's/[\<|\>|\/]//g' |sed 's/\"/ /g'|sed 's/name= //'| sed 's/dispatchState //' "
       status[$i]=$( eval $cmd )
    fi
  done
done
```

What's next?

- golang routines can be extended for other sourcetypes
- Further automate the search framework

Apache log generation toolkit can be found at
github.com/rsomu/apclog

For more information

Visit us at **Booth #114** in source=*Pavilion

splunk> .conf19

.conf19

splunk>

Thank You !

Go to the .conf19 mobile app to

RATE THIS SESSION