Splunk Helping in Productivity

Sumit Nagal  |  Principal Engineer , Intuit  | @sumitnagal

27-Sept-2017|  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.
Agenda

▶ Overview
▶ In Quality
▶ In Development
▶ Improving Velocity
▶ Key Takeaways
▶ Q &A
Overview

Journey with Splunk in productivity
Overview
Jenkins and Splunk Background

- Many Jenkins
- Many Splunk
- Thousands of build, test and deploy data per day
- Splunk only for Application Logs
- Collaboration with development Tools challenge
- Data is scattered
- Splunk is Operation Tool
Journey Line
How we started using Splunk for productivity

Jenkins App
- Splunk App
- Jenkins App
- Conf 2016
- Jenkins Conf

Quality
- Splunk Validator
- Jira Validator
- Overops
- Forensic Dashboard

2016-July

Operation

2016-Sept

2016-Dec

Velocity
- Visibility Jenkins Farm
- Collaboration Tools Integration
- Next Gen Dashboard
- Jenkins App integration

Development
- Build Optimization
- Splunk maven plugin
- Build Dashboard
- Webinar

2017- Mar

Management

Developer
Quality

Quality productivity – pulling data from Splunk
“Scaling the Automation with Quality.”
Quality

Problem

Test Failures

Root Cause Analysis

Process

Opportunity

Accountability

Traceability

Transparency

Solution

Jira Validator

Splunk Validator

Integrated with Test
Quality – Solution
Scaling the Automation with Quality

Jira/Splunk Validator

Test Integration with Splunk / Jira Validator
Quality - Solution

Scaling the Automation with Quality

GitHub

Test Repo

Test

Flaky Bucket

Engineer

Pass

Splunk Validator

Close OR update Jira

JIRA Validator

Failure

Performance

Exception

Open Jira

Code Issue

Splunk Validator

JIRA Validator

Test Issue

Fail

Splunk Validator
Quality – Performance Time

How we are identity opportunity, reporting and taking action – pulling from Splunk
Quality – Application Error

How we are identity opportunity, reporting and taking action – pulling from Splunk
Development

Developer productivity – pushing data in Splunk
“Making developer 10X Productive.”
Developer

Problem

Build Time

Build Failures

Root Cause Process

Opportunity

Visibility

Problem identification

Self Healing

Solution

Splunk Data uploader

Error Categorization

User contribution
Developer – Solution
Making developer 10X Productive

Splunk Data uploader

Build Dashboard, User contribution system
Developer - Solution

Making developer **10X** Productive

![Diagram](image-url)
Development - Build Insight

How we are getting build error across developer, PR and CI - pushing to Splunk

Total Success %: 90.48%
Total Failure %: 9.52%
Build Time: 1,244 Sec
Development - Build Insight
How we are getting build error across developer, PR and CI - pushing to Splunk
Development - Build Error Insight

How we are getting build error across developer, PR and CI - pushing to Splunk
Improving Velocity

Improving Velocity – Leveraging Splunk Jenkins App
“If You can’t Measure it, You Can’t Improve it.”
Journey Line – Jenkins App
How we started using Splunk for productivity

- Local Jenkins
- Local Splunk
- Jenkins Conf

2016-Aug

- Splunk App
- Jenkins App
- Conf 2016
- 3 instances

2016-Sept

- Local Jenkins
- Local Splunk
- Jenkins Conf

2016-Jan

- Performance Fixes
- Coverage Fix
- Dashboard improvement
- Webinar

2017- Mar

- Production roll out
- Visibility Jenkins Farm - SBG
- Collaboration Tools Integration
- Next Gen DashBoard

Production

Pre prod

Tools Instance

POC
Improving Velocity

Problem

- Many Data Sources
- Identify Gaps
- Right Owner

Opportunity

- Single Source
- Exposing Data
- Define Ownership

Solution

- Splunk Jenkins App
- Splunk Dashboard
- Integration with Collaboration tools
Improving Velocity – Solution
If You can’t Measure it, You Can’t Improve it

Integration with Collaboration Tools

Splunk Jenkins App with Velocity data
Velocity - Overall

How we are identity opportunity, reporting and taking action – using Jenkins App
Velocity - Test
How we are identity detail information - using Jenkins App
### Velocity – Jenkins App

How we are connecting to Jenkins App

#### Test Results

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Job Status</th>
<th>Build Status</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>test1</td>
<td>Success</td>
<td>Passed</td>
<td>Passed</td>
</tr>
<tr>
<td>test2</td>
<td>Cancelled</td>
<td>Failed</td>
<td>Failed</td>
</tr>
<tr>
<td>test3</td>
<td>Suspended</td>
<td>Running</td>
<td>Running</td>
</tr>
<tr>
<td>test4</td>
<td>Failed</td>
<td>Failed</td>
<td>Failed</td>
</tr>
<tr>
<td>test5</td>
<td>Success</td>
<td>Passed</td>
<td>Passed</td>
</tr>
</tbody>
</table>

*Note: This is a screenshot of Jenkins test results.*
Overall Improvement
Key Takeaways

Scaling the Automation
- Manual to Auto-magic
- Context to Core

Measure & Improve Velocity
- Visibility
- Ownership / Accountability

Development
- Time (from 50+ to 20+ min)
- Success (80% to 98%)
- Issues (Many to Few)

Productivity

Data is Key
- Collaboration of Data
- Integration of Tools

- Time (from 50+ to 20+ min)
- Success (80% to 98%)
- Issues (Many to Few)
Using Splunk to Improve Splunk’s Build Process

Eddie Shafaq
Splunk Infrastructure
First joined Splunk in August 2011 as a Systems Administrator. Aided in expanding engineering support in "exotic operating system" (AIX, HPUX, S390X and PowerLinux). Served as a member of release engineering to address operational and infrastructure support for products team. Currently serving an infrastructure leadership role around Core Engineering and Release Engineering services.

Bill Houston
Splunk Infrastructure
Bill started his career as an analog hardware engineer designing professional recording equipment. Currently he is a senior release engineer at Splunk working on improvements to the Jenkins CI systems. Prior to Splunk he spent 16 years at Adobe working in various roles; the last four were spent using Jenkins to build and test Adobe Flash.
The Goal Set by Management

- Improve Developer Productivity
  - Our contribution: Get CI test results to developers faster
The system had **15 dedicated Linux agents** to perform continuous integration testing.

Each job ran for approximately **54 minutes**, performing a build of Splunk and running a set of validation tests.

That meant it could perform an approximate average of **17 jobs per hour**.
The system had **15 dedicated Linux agents** to perform continuous integration testing.

Each job ran for approximately **54 minutes**, performing a build of Splunk and running a set of validation tests.

That meant it could perform an approximate average of **17 jobs per hour**.
System Before Optimization

If more than 17 triggers were received in a **one hour** period the excess triggers were queued waiting for a Linux agent to run on

Under “normal” circumstances the system operated with minimum delays, however during peak load periods when the pressure on developers was the highest…

We experienced **significant delays** resulting in frustration and phone calls as the engineers waited for results of the validation test jobs they were required to run before they could commit their work
It was Obvious

Robert Hight runs a record
339.83 MPH at Sonoma Raceway
July 2017

We Needed To Speed Things Up!
Our Analysis
Our Analysis

Understanding the Situation

There were four factors that affected the delay developers experienced while waiting for test results

- Build time – how long it takes to build the Splunk executables
- Test time – how long it takes to perform the required set of tests
- Queue time – how long before the test actually started to run
- Notification – how long before developers know the test results

We need to quantify each of those factors and determine what we could do to mitigate their effects on the overall time
How We Collected The Data

Our jobs are well connected to Splunk
The Splunk Plugin for Jenkins
Makes it Easy to Send Your CI/CD Data to Splunk

https://wiki.jenkins.io/display/JENKINS/Splunk+plugin+for+Jenkins
The Splunk Plugin for Jenkins
Easy to install

Splunk for Jenkins Configuration

Enable
Indexer hostname
HTTP Input Port
HTTP Input Token
SSL Enabled
Jenkins Master Hostname
Raw Events Supported

Enable
Indexer hostname
HTTP Input Port
HTTP Input Token
SSL Enabled
Jenkins Master Hostname
Raw Events Supported

Test Connection
The Splunk Plugin for Jenkins

Easy to install

Splunk for Jenkins Configuration

Enable

Indexer hostname

splunk_indexer.yourco.com

HTTP Input Port

8088

HTTP Input Token

0205A7BB-S388-4A89-B068-55DD7C8FEE54

SSL Enabled

Y

Jenkins Master Hostname

jenkins_master

Raw Events Supported

Y

Test Connection
The Splunk Plugin for Jenkins
Easy to customize
The Splunk HTTP Event Collector
Simple to send custom data to your Splunk instance

In Bash

Format your data as a JSON string:
```
jsonData="{"time": 12345, "index": "YourIndex", "sourcetype": "YourSourceType", "source": "YourSource", "event": {"YourFieldName": "SomeData", more json formatted data goes here}}"
```
Include as much json formatted information as you need in the event section

Then execute a curl call:
```
curl --tlsv1.2 --header "Authorization: <Splunk_auth_token_goes_here>" \
--header "Content-Type: application/json" \
--request 'POST' \
--data $jsondata \
https://YourSplunkInstance/services/collector/event
```
Its that simple…
Queue Times Before Optimization

Peak load period had significant delays

150 minutes!
The Search We Use to Analyze Jobs

Of course this won’t work for you, but...

```
index="jenkins_console"  host="aJenkins.ourco.com" source="*Linux ut pr*" ("make -j48 || exit 0" OR 
"Install the project..." OR "Core build is done" OR "run the tests again" OR "Starting backend unit 
tests" OR "Package and publish Splunk" OR "Starting Linux 64 test" OR "fetch the jenkins scripts 
directory" OR ("nodes run &gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt; STARTING ACTION" AND "Write splunk-version.txt") OR 
("INSTALLING COMMAND" AND "Running the contrib command") OR "Done all requested steps") | rex 
field=source "job/Pull_Request_Tests/job/Linux_ut_pr/\(?&lt;build_number&gt;.*\)/console"  | eval 
buildStep=case(searchmatch("fetch the jenkins scripts directory"),"start", searchmatch("Starting Linux 
64 test"),"clone", searchmatch("Running the contrib command"),"chroot", searchmatch("Write splunk-
version.txt"),"contrib", searchmatch("make -j48 || exit 0"),"setup", searchmatch("Core build is 
done"),"build_1", searchmatch("Install the project..."), "build_2", searchmatch("Starting backend unit 
tests"), "package", searchmatch("run the tests again"), "tests 1", searchmatch("Package and publish 
Splunk"), "tests_2", searchmatch("Done all requested steps"), "publish") | chart values(_time) by 
build number, buildStep  | eval gc = round(('clone' - 'start')/60) | eval cs = round((chroot 
- 'clone')/60) | eval "cb" = round((contrib - chroot)/60) | eval "bs" = round((setup - contrib)/60) | 
eval "cub" = round((build 1 - setup)/60) | eval "cbc" = round((build 2 - build 1)/60) | eval "ts" = 
round((package - build 2)/60) | eval "pst" = round((tests 1 - package)/60) | eval "sst" = 
round((tests 2 - 'tests 1')/60) | eval "pub" = round(('publish' - 'tests 2')/60) | search cb &lt; 5 
| search sst &gt; 0 | search pst &lt; 25 | chart values(pub) as Publishing, values(sst) as "Sequential 
Smoke Tests", values(pst) as "Parallel Smoke Tests", values(ts) as "Test Setup", values(cbc) as "Core 
Build Continues", values(cub) as "Core and UI build", values(bs) as "Build Setup", values(cb) as 
"Contrib Build", values(cs) as "Chroot Setup", values(gc) as "Git Clone" by build number
```
The Search We Use to Analyze Jobs

Collect the specific log file lines we will use in our analysis

index="jenkins_console" host="aJenkins.ourco.com" source="*Linux_ut_pr*" ("make -j48 || exit 0" OR "Install the project..." OR "Core build is done" OR "run the tests again" OR "Starting backend unit tests" OR "Package and publish Splunk" OR "starting Linux 64 test" OR "fetch the jenkins scripts directory" OR ("nodes run >>>> STARTING ACTION" AND "Write splunk-version.txt") OR ("STARTING COMMAND" AND "Running the contrib command") OR "Done all requested steps")
The Search We Use to Analyze Jobs

Extract the job number

`| rex field=source "job/Pull_Request_Tests/job/Linux_ut_pr/(?&lt;build_number&gt;.*\)/console`
Build a table of time stamps by job number and job step

```sh
| eval buildStep=case(searchmatch("fetch the jenkins scripts directory"),"start", searchmatch("starting Linux 64 test"),"clone", searchmatch("Running the contrib command"),"chroot", searchmatch("Write splunk-version.txt"),"contrib", searchmatch("make -j48 || exit 0"),"setup", searchmatch("Core build is done"),"build 1", searchmatch("Install the project...") , "build 2", searchmatch("Starting backend unit tests"), "package", searchmatch("run the tests again"), "tests 1", searchmatch("Package and publish Splunk"), "tests 2", searchmatch("Done all requested steps"), "publish")

| chart values(_time) by build_number, buildStep limit=50
```
The Search We Use to Analyze Jobs

Calculate the deltas between job steps

```
| eval gc = round(('clone' - 'start')/60) | eval cs = round((chroot - 'clone')/60) | eval "cb" = round((contrib - chroot)/60) | eval "bs" = round((setup - contrib)/60) | eval "cub" = round((build 1 - setup)/60) | eval "cbc" = round((build 2 - build 1)/60) | eval "ts" = round((package - build 2)/60) | eval "pst" = round((tests 1 - package)/60) | eval "sst" = round(('tests 2' - 'tests 1')/60) | eval "pub" = round(('publish' - 'tests 2')/60) | search cb &lt; 5 | search sst &gt; 0 | search pst &lt; 25
```
The Search We Use to Analyze Jobs

Build the final table with user friendly names for display

| chart values(pub) as Publishing, values(sst) as "Sequential Smoke Tests", values(pst) as "Parallel Smoke Tests", values(ts) as "Test Setup", values(cbc) as "Core Build Continues", values(cub) as "Core and UI build", values(bs) as "Build Setup", values(cb) as "Contrib Build", values(cs) as "Chroot Setup", values(gc) as "Git Clone" by build number limit=50 |
Our Analysis

How long does each step of a job take?

- We used the previous search to chart the time each step took
Our Analysis

How long does each step of a job take?

• We Identified the two portions of the job that took the longest
• The Splunk build (in orange) and the validation tests (in red)
Speeding up the Splunk Build
Distcc Architecture

Physical Machines and VMs:

- All DistCC Build Clients and servers use the same build toolchain and chroot
- 15 Build agents
- 12 compile nodes
- DistCC server is used ONLY for compile
- `make -j48`

12 x 20 Core VMs
DistCC VS Normal Build
Building Splunk with DistCC

- 24 Min Build
  - `make -j 24`
  - Web UI –j 1
  - Optimal 24 core VM agent

- 19 Min Build
  - `make -j 48`
  - Web UI –j 1
  - Optimal 24 core VM agent

- 8 Min Build
  - `make -j 48`
  - Web UI –j 6
  - Optimal 24 core VM agent
  - 12 DistCC hosts

24m ➔ 19m ➔ 8m
Build Time Improvement - Results

Dramatic reduction in the overall build time
Build Time Improvement - Results

Dramatic reduction in the core compile time
Speeding up Testing
Our Analysis
Reducing the Build Testing Time

Analysis:
- We sequentially run two different test configurations on build agents – parallel and sequential
- Parallel tests scheduled across 8 Splunk test instances
- Sequential tests ran one at a time
- We used Splunk to measure the overall timing of each test configuration as well as the individual tests

Mitigation:
- Nothing could be done about the sequential tests
- Increased the number of parallel test instances to 12 and measured over a few days
- Increased the number of parallel test instances to 16 and measured over a few days
Increasing Test Parallelization

Improvement in the overall test time as parallel instances were increased

8->12
12->16

~23m → ~19m → ~17m
The Final Overall Result

Overall job time reduced to ~35 minutes

- Build time reduced to under 8 minutes
- Test time reduced to under 22 minutes
Managing the Agents
Our Analysis
Breaking Down The Timing

Reducing the Pull Request Queue time

Analysis:
• Developers are only waiting for results from pull request tests
• Triggers on commits to some branches and all pull requests

Mitigation:
• Use more agents for the pull request tests
• Manage the allocation of agents to tasks -> shift resources to pull requests when the queue starts to climb
• Add 5 “standby agents” with reduced capabilities that are powered up on demand -> smaller footprint on VM hosts
Our Analysis

Breaking Down The Timing

Reducing the Pull Request Queue time

Analysis:
- Developers are only waiting for results from pull request tests
- Triggers on commits to some branches and all pull requests

Mitigation:
- Use more agents for the pull request tests
- Manage the allocation of agents to tasks -> shift resources to pull requests when the queue starts to climb
- Add 5 “standby agents” with reduced capabilities that are powered up on demand -> smaller footprint on VM hosts
Before Optimization

Peak load period → significant delays
After Optimization

Recent similar trigger conditions

12 minutes!
Improving Notifications
Our Analysis

Speeding up Developer Notification

Analysis:
• Test results only available as comments in pull request UI
• Developers need to go to Bitbucket frequently to check for results
• All employees are connected via Atlassian HipChat

Mitigation:
• Added personal HipChat messages
  • On receipt of trigger → includes the current queue time
  • When job is completed → includes test results
Overall Improvement

Key Takeaways

Parallelize Builds
Improved efficiency in builds by optimizing parallelization

DistCC in Build System
Incorporated DistCC in build system

Alerting Developers
Notified Developers about build and merge status

Splunk
Analysis, Alerting and Reporting

Parallelize Tests
Improved efficiency in test by optimizing parallelization
The Search We Used to Analyze Jobs

Here is what the full output of the search we presented earlier looks like
The Jenkins App for Splunk
Seamlessly collect, monitor and analyze Your Jenkins Data
Don't forget to rate this session in the .conf2017 mobile app
Appendix

Some reference for above Deck to know more
Splunk – Jenkins Configuration

JaCoCo plugin
No description available.

Splunk Plugin

"Splunk plugin for Jenkins provides deep insights into your Jenkins master and slave infrastructure, job and build details such as console logs, status, artifacts, and an incredibly efficient way to analyze test results. The plugin is used together with {color:#474444} Splunk App for Jenkins {color} that provides out-of-the-box dashboards and search capabilities to enable organizations to run a high performing Jenkins cluster and bring operational intelligence into the software development life cycle."
Splunk – Jenkins Configuration - 2

Event Source:
Splunk App for Jenkins Link
Max Events Batch Size
Retries on Error
Ignored Job Names Pattern
Customize Event Processing Script
Customize Event Processing Command

splunk$ curl -X GET http://<jenkins_host>:8080/jenkins/bluebot/poller/logs -d 'instance=1&before=0&count=100'

[...]

// send job metadata and jUnit reports with page size set to 50 (each event contains max 50 test cases)
splunkjs.meta(20)
// send coverage, event meta contains max 50 classe metrics
splunkjs.metaCoverageReports(50)
// send all logs from workspace to splunk, with each file size limits to 1MB
/splunk.meta.archive("/var/log", null, false, "1MB")
// exit

splunk$
Splunk/Jira Validator Flow

```xml
<dependency>
  <groupId>com.intuit.qbo.qa</groupId>
  <artifactId>SplunkValidator</artifactId>
  <version>1.0.10</version>
</dependency>

@Test(enabled = true)
@SplunkValidator(testFunctionality = "companyCreationforUS", enabledValidation=true, enabledJiraValidation=true, jiraProject="SBGCOMP"
public void USCompanyCreation() throws Exception {
    String i    _ = RuntimeSplunkConfig.getInstance().getTestToken();
    System.out.println("JIRA_TTD = " + iuit_tid);

<xml version="1.0" encoding="UTF-8">
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <listeners/>
  <test name="Test">
    <classes>
      <class name="healthcheck.CompanyCreationTest" />
    </classes>
  </test>
</suite>
```
Jira defect
Splunk Configuration
Splunk/Jira Validator Flow

Successful Test

Failed Test

Closed Test

Splunk/Jira Validator Flow

Delaying splunk query for:10000 milli-seconds
Querying For:QUERY.PCI
Splunk QUERY: search index=qbo* source=type=qboapplog TST-QBO-65-qa-testQBCompanyCreation-QBCompanyCreation_NewUser-201703161306186672-23419 signedexistinguser QboAuthToken httpstatuscode=500 PerfStat| stats count
Time Range: From--15m - To --now
PATH: /servicesNS/rest_tbarea/intu_sbg_qbo_search/search/jobs
intu_sbg_qbo_search_from Splunk:
count: 0
Expected:
count: 0

[Splunk Verification]
Mismatch:0, against:2
Mismatch:0, against:3
Mismatch:0, against:1
Mismatch:0, against:1
Splunk.Verification_Test_Failed=TST-QBO-qbo-bats-4-e2e-en_US-testCreateInvoiceFromInvoiceList-TestCreateInvoiceFromInvoiceList
Action=Updated bug comments --> for JiraId=https://jira.intuit.com/browse/SBGCOMP-486

Querying For:QUERY.ERROR
Splunk QUERY: search index=qbo* source=type=qboapplog TST-QBO-65-qa-testQBCompanyCreation-QBCompanyCreation_NewUser-201703161306186672-23419 exception| rex fields_rew "https://kb.to/1?takipi_url=(\"\")| eval takipi_url="https://kb.to/1?takipi_url | rex fields_rew "[^\r\n\s]+" | stats count by errGenericErr, intu_tid, cluster_time, takipi_url | appendpipe | stats count | where count>0 | fields_count
Time Range: From--15m - To --now
PATH: /servicesNS/rest_tbarea/intu_sbg_qbo_search/search/jobs
intu_sbg_qbo_search_from Splunk:
count: 0
Expected:
count: 0

[Splunk Verification]
Updated / closed jira https://jira.intuit.com/browse/SBGCOMP-364
Splunk Attributes:-------------------
12:35:30 entire-build-execution: 13 minutes, 27 seconds *** >>> !!! FOCUS ON SPEEDING THIS UP !!! *** <<<

Uploading result After Build

<table>
<thead>
<tr>
<th>Time</th>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
</table>

Result in Splunk

splunk>
Splunk Data uploader - 2

```xml
<parent>
  <groupId>com.intuit.sb.qbo</groupId>
  <artifactId>qbo-build-modules-list</artifactId>
  <version>17.08.8-SNAPSHOT</version>
</parent>

<artifactId>qbo-build-upload-splunk-data</artifactId>
<packaging>jar</packaging>
<name>QBO Splunk Data Upload Executor</name>

<build>
  <plugins>
    <plugin>
      <groupId>${project.groupId}</groupId>
      <artifactId>qbo-splunk-data-uploader-maven-plugin</artifactId>
      <version>${custom.maven.plugins.version}</version>
    </plugin>
  </plugins>
  <executions>
    <execution>
      <phase>install</phase>
      <goals>
        <goal>upload</goal>
      </goals>
      <configuration>
        <rootSourceDir>${src.home}</rootSourceDir>
        <host>${env.SPLUNK_DATA_UPLOADER_HOST}</host>
        <port>${env.SPLUNK_DATA_UPLOADER_PORT}</port>
      </configuration>
    </execution>
  </executions>
</build>
```
Improving Velocity References
Coverage Process

Unit Coverage Schedule
- Daily Execution Schedule
- This is used when you have a git repo and you are publishing a single number for repository.

Jenkins Build Job
- The job build and test and make source, compile code and jaccio binary from Unit Test ready for consumption.
- Workspace Available

Jenkins Coverage Job
- Trigger
- Get Source, compile and Unit Test jaccio binary, generate dynamic build file based on architecture enforced, uses and generate coverage based on domain & component.

Jenkins publishing Job
- Plug-in dashboard
- This job will publish Data in json meta data to plug-in registry, where it will be display

Dashboards
- Splunk Dashboard

Test Coverage Schedule
- Daily Execution Schedule
- This is used when you have to capture JHB base test execution and its coverage number.

Jenkins Build Job
- Trigger
- This job build and test and make source, compile code and jaccio binary from Unit Test ready for consumption.
- Workspace Available

Jenkins Deploy/Test Job
- Trigger
- This job will deploy and Test artifact on given environment, where we have configure jaccio byte code instrumentation using chef recipe.

Jenkins Capture Coverage from Environment
- Trigger
- This job will capture jaccio binary from given environment, using jaccio client library.

Jenkins Coverage Job
- Trigger
- Get Source, compile and Unit Test jaccio binary, generate dynamic build file based on architecture enforced, uses and generate coverage based on domain & component.

Jenkins publishing Job
- Plug-in dashboard
- This job will publish Data in json meta data to plug-in registry, where it will be display

Dashboards
- Splunk Dashboard