**Logjam**

Final Progress Report

Final Requirements, Design, Implementation, Testing, Installation, & Delivery

**NetApp**

**CSC 492 Team 16:**

Jake Ferrero

Joshua Good

Peter Soloninka

Renata Zeitler

North Carolina State University

Department of Computer Science

May 8th , 2018

**Executive Summary**

*Renata Zeitler*

NetApp is a Fortune 500 Information Technology company that provides storage and data management solutions. StorageGRID, produced by NetApp for customers, is a software-defined object storage solution for web applications and rich-content data repositories that uses a distributed shared-nothing architecture. Centered around the open-source Cassandra NoSQL database, it achieves massive scalability and geographic redundancy, while presenting a single global object namespace. Despite the significant benefits this program structure offers, its distributed nature presents unique challenges when addressing customer support cases.

NetApp tasked our team with creating a tool that will allow their Support Engineers to efficiently sort through error logs produced by StorageGRID and present them in a manner that is easy to visualize. Currently, if a NetApp Support Engineer finds a suspicious log, they have to comb through terabytes of unstructured error files, across several StorageGRID nodes, to try and find another similar log- *if one even exists*. This takes an enormous amount of time to accomplish and is highly inefficient for finding the root cause for a single error log. We were responsible for constructing a tool, called Logjam, using the ELK Stack (Elasticsearch, Logstash, Kibana) to rectify this problem by categorizing the log data and creating an interactive visual interface.

The fundamentals of the ELK Stack are as follows: Elasticsearch is used for searching and organizing data provided by Logstash, Logstash is responsible for parsing data via filters to clean the inputted data, and Kibana displays the information supplied by Elasticsearch to conveniently show aggregations. This three-part system ingests, indexes, and visualizes the log data. We have configured Logstash to use plugins to break apart the log files into discrete structured events - this allows Kibana to create more specific visualizations. We acquired a Docker image for the initial ELK startup that was modified to create a setup image containing everything needed to run our end product. We added our own custom configurations and files that are wrapped up in a dockerfile, which can be used to create a Logjam image.

Renata and Josh were responsible for creating a Python script that would scrape desired error logs from StorageGRID unstructured output directories and send them to Logstash. Jake and Peter were responsible for setting up an Elasticsearch cluster and index configurations, and customizing Logstash plugins. This was done across four iterations throughout the Spring 2018 semester.

**Project Description**

*Joshua Good*

**Sponsor Background**

As a global company, NetApp is one of the leading enterprise data storage companies specializing in the Hybrid Cloud, a mixed private storage, on-premise infrastructure, and public cloud service. NetApp’s products offer performance towards maximizing data management, allowing businesses to capitalize on customer analytics. Among these, StorageGRID Webscale, their distributed storage solution, provides a scalable architecture for multi-site ingestion and reading. StorageGRID’s cross-location data transmission, alongside its large storage capacity, serves to manage unstructured data in a rapidly evolving cloud environment.

**Contact Information**

Cloud Business Unit,

StorageGRID Engineering

|  |  |  |
| --- | --- | --- |
| Morgan Mears  Principal Engineer  morgan.mears@netapp.com | Samuel Fink  StorageGRID Developer  samuel.fink@netapp.com | Mauricio Sánchez  StorageGRID Developer  mauricio.sanchez@netapp.com |
| Dilip Tammana  Member Technical Staff  dilip.tammana@netapp.com | Arpita Awasthi  Software Developer  apita.awasthi@netapp.com | Brian Atkins  Senior Software Engineer  brian.atkins@netapp.com |
|  | David Schmitt  Engineering Director, StorageGRID Webscale  david.schmitt@netapp.com |  |

**Problem Statement**

When a StorageGRID customer contacts NetApp support with an issue, Support Engineers often rely on their StorageGRID system logs that contain error statements reported at failure. After creating a unique case number for tracking the issue, Support Engineers investigate these log files and attempt to identify the most relevant information in diagnosing its cause . However, due to StorageGRID’s scale-out distributed architecture, logs are stored across dozens of individual StorageGRID nodes, complicating case filtration. The massive amount of data recorded during each StorageGRID transaction buries these errors, further lengthening the search process. Moreover, the lack of a structured system for organizing legacy cases impedes response times, barring efficient access to past solutions.

**Project Goals and Benefits**

To resolve these issues, Netapp desires a support tool for managing StorageGRID support cases. Our tool, Logjam, shall reduce the turnaround time for customer support, mitigating the overhead associated with manually investigating log files. Support Engineers will be able to view the most prominent errors through frequency analysis, easily discerning the most relevant log collections. Moreover, Logjam will provide Support Engineers the functionality to compare new and legacy case files, repurposing past solutions in efficiently solving new issues with similar symptoms.

**Development Methodology**

Project development followed a bi-weekly sprint process, and beginning on Thursdays. For each sprint, a main feature was identified, then broken into sub-tasks which were then assigned to team members. Each team member demoed their current progress and discussed any current difficulties on Saturday during the weekly internal team meeting. Development continued throughout the sprint, with occasional Slack check-ins for progress, until the weekly Thursday’s sponsor meeting. First-week iteration meetings were composed of a summary of the team’s contributions, including components’ status and collective challenges, and a sponsor review. Additional sponsor suggestions about improving component implementation were then taken into consideration. Second-week iteration meetings demoed the completed feature or sub-components, as well as documented individual team member contributions.

Project management was hosted via GitHub. For each new feature, an Issue was created with any active members, a breakdown of its requirements, and the iteration to which it applies. While multiple team members may have concurrently worked on a single feature at any given time, each member actively contributed to the repository on their own branch. When a feature was completed, the proposed solution was then pushed to the remote repository for review, and the contributing team member created a Pull Request. Any team member could then review the new feature, but this was preferably recommended for other team members working on that feature. If approved, the new feature was merged into the master branch as part of the final build. This process of source control prevented us from creating potential merge conflicts and helped maintain a working version of our current progress.

**Challenges and Resolutions**

The greatest challenge was our inexperience with the chosen tool suite, Elastic Stack. Understanding the relationship between Elastic applications, how plugins operate during ingestion, and pipeline flow was paramount to successful delivery. From a technical perspective, Logjam is built to manage large amounts of log data. For development, we worked off two AWS instances; however, NetApp plans to use Logjam to span across dozens of nodes. The challenge for us was making our tool scalable to this level. We achieved this scalability by minimizing the time complexity of our categorization process. Where possible, categories that used regular expressions are replaced with simple string splitting based on character position.

With regards to indexing log files, one major issue that presented an organizational challenge was StorageGRID version. Support cases may contain multiple versions of StorageGRID, which could incorrectly associate case metadata if not properly categorized. To resolve this issue, NetApp provided several grep commands for extracting the version number from different case files. If a case directory contains multiple StorageGRID versions, the first detected version shall be used.

Another organizational challenge related to indexing files was the difficulty in categorizing highly-nested files. For instance, nesting a “commands” log under both a *base\_os\_logs* and *cassandra* folder complicates the categorization process as the file could be labeled as either a *base\_os\_logs\_commands* or *cassandra\_commands* file. To alleviate this complexity, NetApp suggested a three-pronged matching approach: files should be categorized first by matching canonical name in the final path and second by the path’s trailing portion and canonical name. Otherwise, a substring/pattern matching its contents shall determine its type. After implementing both methods, we determined the latter to be more accurate.

We foresaw only one possible legal challenge when Logjam was delivered to NetApp. Upon transfer, we were unsure whether we will retain the Intellectual Property (IP) rights to our solution, preventing us from using it in our portfolios. After discussing this issue with NetApp, we determined that while the rights to Logjam would transfer, it would not restrict us from presenting our work.

**Resources Needed**

*Jake Ferrero*

* Amazon Web Services (AWS) Instances
  + In order to simulate a multi-node, production environment, our NetApp sponsors provided us with two AWS instances. These instances had 8 cores, 16GB of RAM, and 500GB of storage. The two instances, which were running Debian 4.9.30, were used as a testing ground for our team and were configured with one being a “master” node, containing the log bundles and the full ELK Stack, and the other being a “slave,” having only elasticsearch to load balance.
* GitHub
  + Internally, our sponsors utilize GitHub for version control. In order to comply, we used the technology as well and provided our sponsors with access to our Senior Design repository. We also utilized GitHub’s ‘Projects’ utility as a means of project management. It was used to track which team members were assigned which user stories, their development cost, and their status.
* Docker and “Logjam” image
  + Our project Logjam relies heavily on Docker containers. A container is a stand-alone package containing everything a program needs to run: code, libraries, configurations, etc. For Logjam, we delivered an image that contained our final product and everything it needed to run on any UNIX environment. Our custom Logjam image contains the following: a Python script for both setting up the duplicates database as well as one for data ingestion, a wrapper bash script to make running Logjam easier, the latest ELK Stack (6.2.4), SQLite 3, Python 2.7, and several python packages including conan, patool, pyunpack, and python-magic. We used the latest community version of Docker, Docker CE 17.12.1.

**External Dependencies**

*Jake Ferrero*

* ELK Stack
  + Because our project revolved around Elastic Stack, an external product, it was our responsibility to stay up-to-date on the newest releases and any news regarding the software to ensure that our application was not affected negatively by these updates.
* Amazon Web Services
  + Since our solution was hosted on AWS instances, we were relying on Amazon Web Services remaining operational. While it is unlikely that AWS will go down, it is not improbable. As such, we needed to prepare for the worst. This is why our team included thoroughly detailed documentation in the deliverable.
* Docker Image
  + In addition to the legal challenges mentioned in the ‘Project Description’ section, we were dependent on the Docker image we were using as a base to remain online and up-to-date with current releases of Elastic Stack. However, since we only used this image as a starting point and branched off of it as we progressed through the semester, this should no longer be a problem.

**Requirements**

*Jake Ferrero (primary author)*

*Joshua Good, Peter Soloninka, Renata Zeitler (contributing editors)*

Logjam consists of two main components - ingesting and indexing/visualization. StorageGRID support engineers can point our tool at a directory structure containing a large number of legacy support cases, and it will recursively search through the structure and ingest valid log files, which will then be parsed and enhanced using a custom regex filter. The parsed data will then be indexed, which allows for full-text searches on data, allowing for faster and more efficient queries. Finally, support engineers can visualize query results through generated graphs, as well as isolate specific cases from a larger aggregate of data. Being able to efficiently query indexed logs, as well as visualize aggregate support cases, will help customer support in the future. This process will allow NetApp engineers to explore anomalies in the error logs- anomalies being any error that is out of the ordinary or occurring an alarming amount of times.

**Functional Requirements:**

FR1: Ingest/Indexing

FR1.1: Logjam will ingest recursively zipped StorageGRID log data while preserving unique support case ID (case number) and StorageGrid version.

* + - **Description**

As a StorageGRID Support Engineer, I want to feed zipped log bundles into the Elastic stack so that they can be indexed and searched to find commonalities, such as category or error type, between logs.

* + - **Preconditions/Assumptions**
      * New log data can appear anywhere in the directory structure, but the data will always be in new files.
      * Top level directory names will contain a unique case number.
      * There may be new information that needs to be ingested from old directories.

FR2: Search/Visualization

FR2.1: Visualization displays the frequency of specific log messages based on timestamp, StorageGRID version, category, and categorized time used.

* + - **Description**

As a StorageGRID Support Engineer, I want to be able to see a percentage of logs that exhibit the same anomaly so that I can visualize the frequency of log messages.

* + - **Preconditions/Assumptions**
      * Log data has been correctly ingested and indexed [1.1]

FR2.2: The tools should be able to to drill down to a specific value for a dimension.

* + - **Description**

As a StorageGRID Support Engineer, I want to be able to further investigate aggregations and drill down on specific criteria.

* + - **Preconditions/Assumptions**
      * StorageGRID log data has been correctly ingested and indexed [1.1]

**Non-functional Requirements:**

1. Ingest log bundles at approximately a gigabyte per hour.

**Constraints:**

1. Implement Logjam with Elastic Stack 6.2.
2. Track code on the provided NCSU Senior Design GitHub page[3].
3. Logjam will run over two provided AWS instances.

**Design**

*Peter Soloninka and Renata Zeitler*

Logjam consists of two primary components: a python script, ingest.py, that can be pointed at a local or remote file structure and ingest log bundles; and the Elastic Stack, which will take the ingested case data, parse and index it and provide a means for aggregate case visualization. The Elastic Stack is an open source suite that is widely used for log ingestion and analysis. It has three components: Logstash, Elasticsearch, and Kibana. Logstash parses and enhances raw log data into structured events; Elasticsearch indexes the log data so that it can be quickly searched and aggregated; Kibana provides a web interface to easily query, aggregate, and visualize data from Elasticsearch. Each component of the Elastic Stack will be configured with various plugins, which is described in their respective subsections. This configuration will be organized into a Docker image for easy distribution as shown in Figure 1. The python script ingest.py will run periodically to unpack network mounted log file bundles into category folders. Logstash will watch those category folders for new log data which it will convert into structured events, add metadata, and index in Elasticsearch. The Support Engineers can manually run ingest.py on the AWS instance. Once there are events in Elasticsearch, Kibana provides a web interface to query the log data and produce visualizations. Figure 1 shows the overall flow of Logjam.

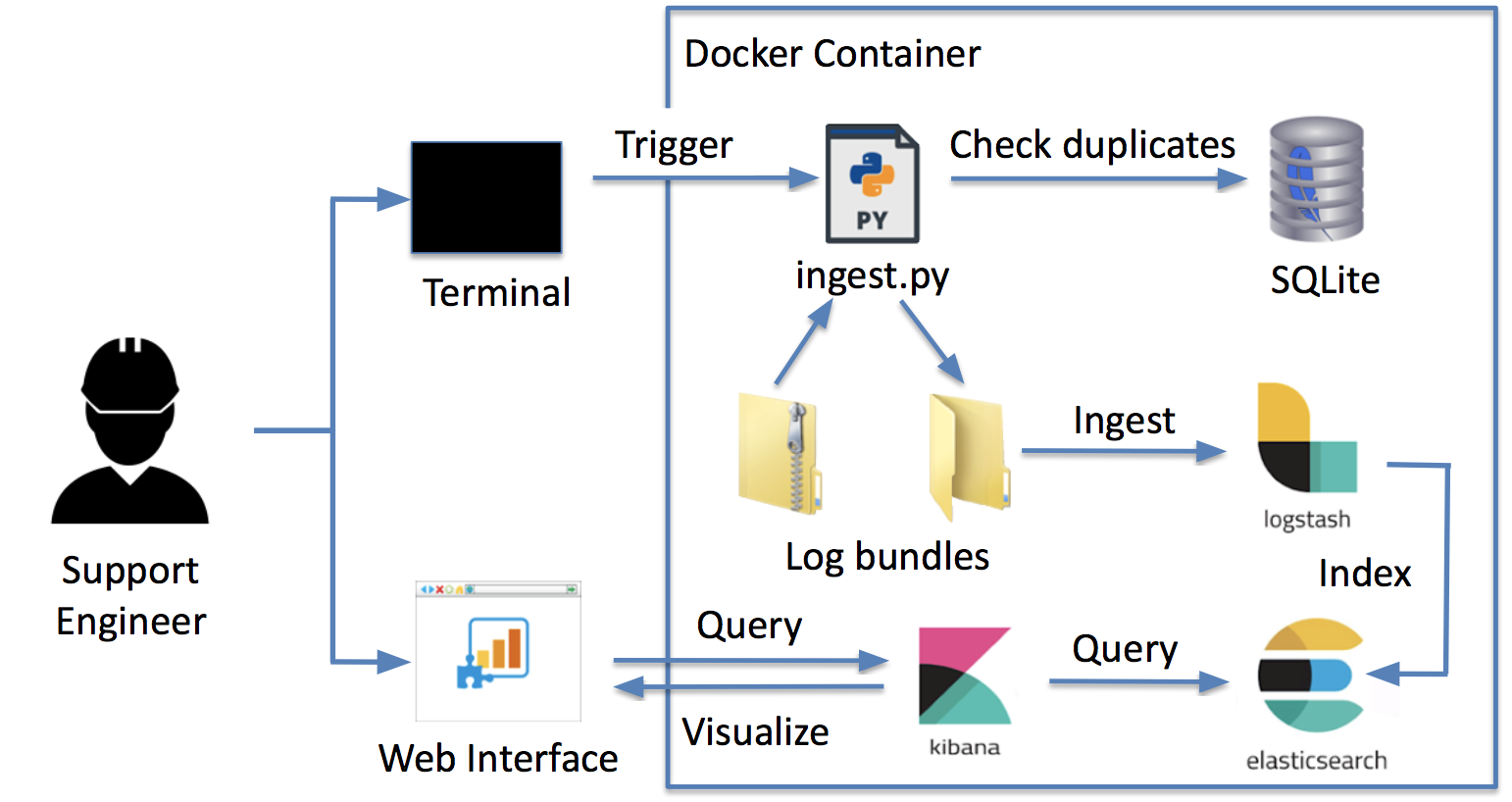


Figure 1: Logjam High-level Design

**ingest.py**

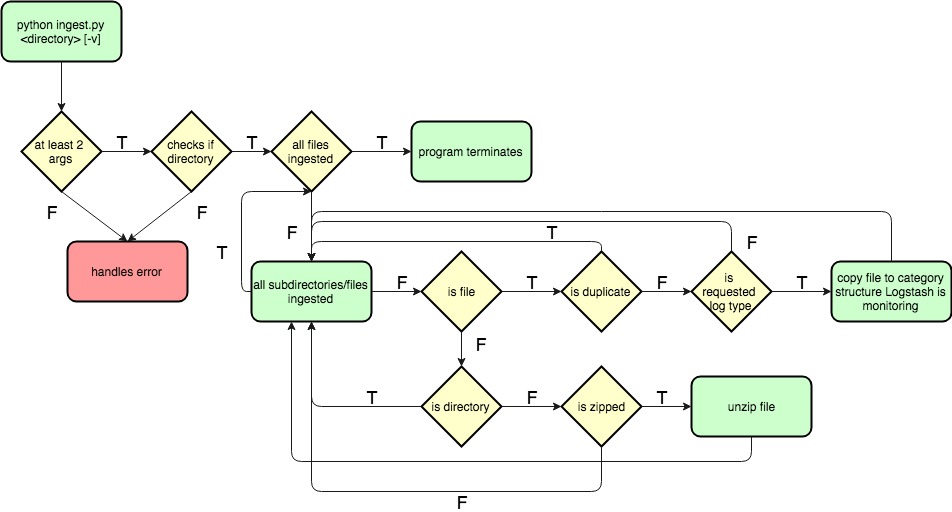
*ingest.py* is a python script that identifies log files to be ingested via Logstash. The script manages support cases by recursively traversing a directory of arbitrarily structured support cases. As these directories may contain extraneous files unrelated to StorageGRID logs, *ingest.py* filters directory content for .log and .txt file types. Below, Figure 2 details the script’s flow:

Figure 2: ingest.py Program Flow

From the command line, a Support Engineer provides the directory to ingest, as well as the optional “-v” verbose option for printing script operations. For each support case, the script parses any zipped archives before filtering relevant log files, extracting their contents to the current directory. Targeted log files are then structured for ingestion (setting their category type, case number, and file path as the filename) and sent to a directory that is monitored by Logstash.

**Logstash**  
 Logstash will watch for new data in each category directory via the file input Logstash plugin. Depending on the category of the log the file input will produce a new event based every time it reaches a certain delimiter, usually a line break. Before sending an event off to Elasticsearch to be indexed, Logstash enhances each event by extracting case number, version, and category from the filename via the grok Logstash plugin. The flow through the Logstash pipeline is shown in Figure 3

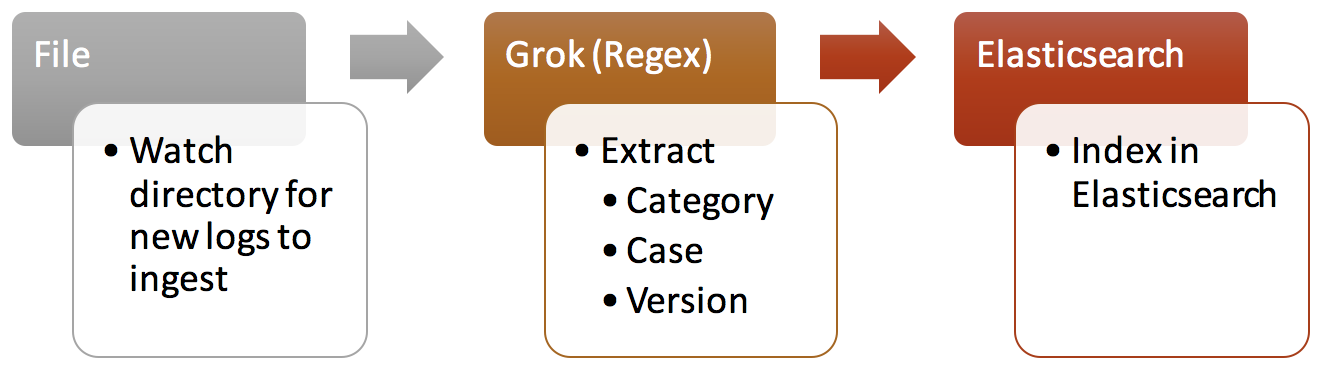


Figure 3: Logstash Low-level Design

**Elasticsearch**

Elasticsearch is where the enhanced log data from Logstash will be stored for quick lookup and visualization in Kibana. All data is stored in a single index with single replication across a two node cluster. Event fields are mapped into the index as defined by a custom mapping (/logjam/config/mapping.json).No plugins are enabled in the Elasticsearch index. Events from different support cases will be distinguished by a case number, which is provided by the top level directory.

**Kibana** We will useKibana for the graphical interface of Logjam, which is shown in Figure 4. It will be configured such that support engineers can query log data in Elasticsearch, and then generate visualizations from those queries. The exact set of Kibana plugins that will be used is still to be determined, but we hope to provide a smooth flow from high-level aggregations to more granular views within one or two mouse clicks.

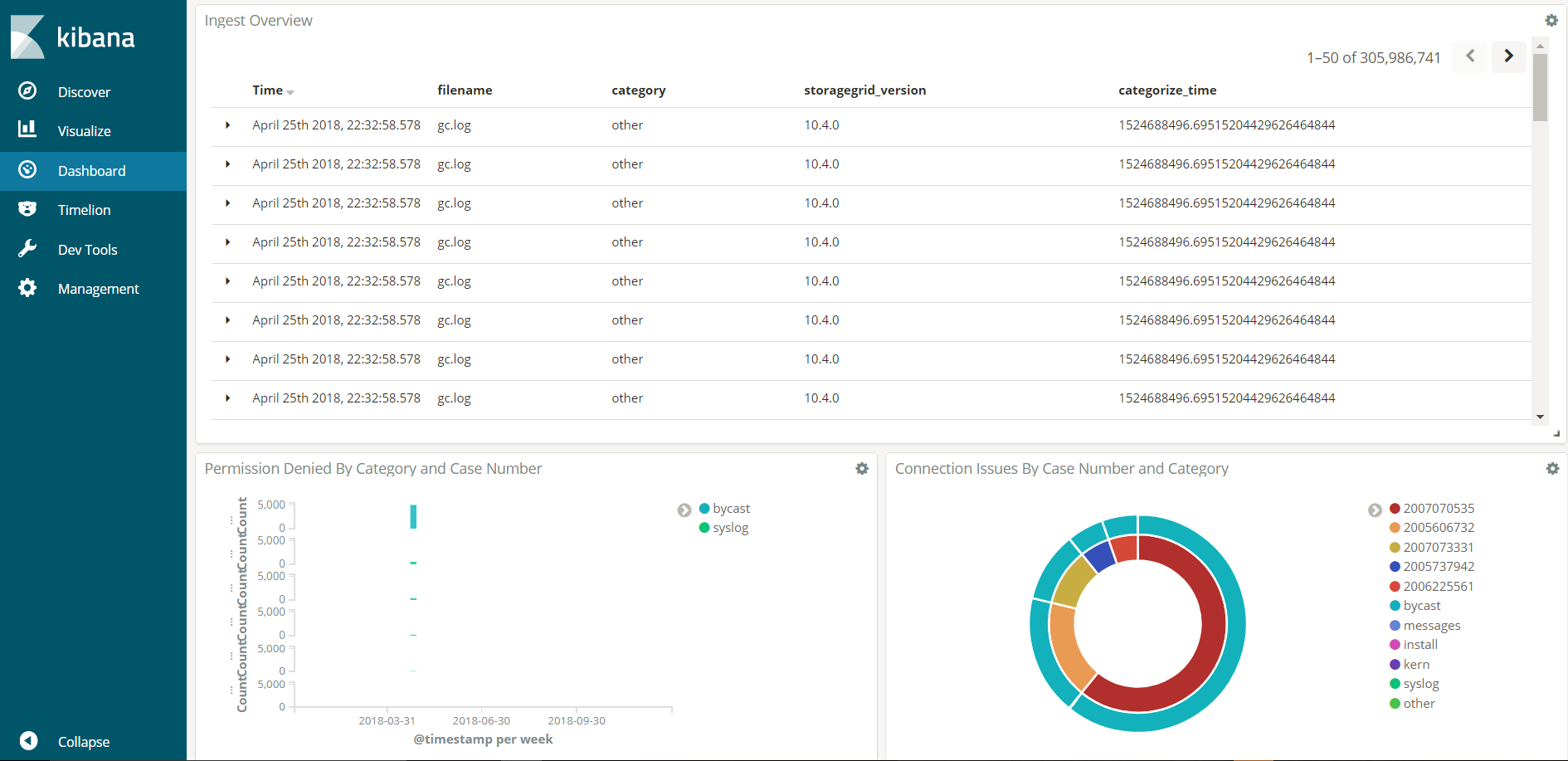


Figure 4: Kibana Default Dashboard

**duplicates.db**

We created a database to prevent duplicate events from being sent to Logstash. If the ingestion script is run again on the same directories that have new error logs, the database will prohibit the same data from being ingested again by comparing the file paths of each of the files to entries already logged in the database. This database also flags if zipped files are corrupted, which will allow the script to skip attempting to unzip the file again on a second pass through. Lastly, the database associates every file path logged with a category it is associated with. This category is used to track where the file is going to in the directory monitored by Logstash. When the files are transferred to the Logstash directory, a unique id is appended to the filename and used as a link from the file to the database. The database is laid out as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | file path | flag | category | uid |
| Type | text | int | text | float |

**Implementation***Peter Soloninka (primary author), Jake Ferrero (contributing editor)*

Our implementation was broken into 4 iterations that were composed of 2 bi-weekly sprints with defined deliverables. The first week of each sprint was focused on planning and design and the second week was focused on implementation and testing.

**Iteration 0: 01/08 - 02/06/2018 - Complete**

This iteration followed a different format than the others. It did not involve implementation, but instead focused on teaming, requirements gathering, high-level design, planning, and documentation. The deliverables for this iteration were the following:

* Requirements document
* High-level design
* Project roadmap

**Iteration 1: 02/06 - 03/01/2018 - Complete** The purpose of this iteration was to create an initial prototype of ingest.py and the Logstash pipeline. Our deliverable was a demo of querying from Elasticsearch with minimal configuration, containing a focus on delivering an initial design of ingest.py and the Logstash pipeline. The second sprint produced a demo-ready prototype of all components of Logstash, except for Kibana. The initial prototype followed our requirements document with the following exceptions:

* Input file structure was simplified to a single level .zip file
* Duplicates were not prevented
* The Logstash filter pipeline was simplified to the following
  + Case number and file structure data was tracked
  + Pattern matching was performed using the Grok filter plugin
  + Each event was from the Bycast category instead of the full twenty provided
  + The timestamp was extracted with the date plugin

**Iteration 2: 03/01 - 03/29/2018 - Complete**  
 The goal of this iteration was to create an end-to-end pipeline that met all client feature requirements and tests that pipeline at scale. Using the prototype created in the first iteration, we implemented the remaining features such as extra categorization, preventing duplicate events, and additional Kibana visualizations.

**Iteration 3: 03/29 - 05/08/2018 - Complete** In this iteration, we hardened required functionality to meet performance and reliability measures, implemented stretch goals, finalized documentation, and created our final presentation for Posters & Pies. The first sprint focused on perfecting Logjam, stretch goals, documentation, and initial Posters & Pies planning, while the second sprint focused on finalizing documentation and the Posters & Pies presentation.

**Project Structure**

The "logjam" directory contains everything needed for the project and will be stored on the machine that hosts the Docker containers. The config directory contains all of the necessary configuration files needed for Logjam, including the logstash pipeline (logjam.conf), two different elasticsearch configurations for the master and slave nodes in the cluster, and a mapping file for the elasticsearch index. The dockerfile is used to create the Logjam image with all of the necessary scripts and configurations pre-installed. The ingest and setup scripts ingest logs bundles and create the duplicates database, respectively. The logjam.sh script serves as a wrapper for the project to allow support engineers a more easy way of interacting with Logjam’s functionality.

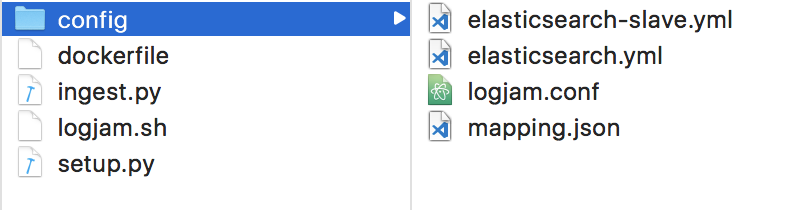


Figure 5: Project Structure

**Project Configuration**

To enable our custom pipeline, logjam.conf must be mounted to the Docker container inside the /etc/logstash/conf directory, and Logstash must be restarted and pointed at that directory. A logjam-test index must be created in Elasticsearch to accept events from Logstash. Otherwise, we are using the default configuration for the sebp/elk Docker image[1]. See the Installation Guide[2] for more details.

**Logjam.sh - Wrapper Script**

Logjam comes with a logjam.sh script that serves as a wrapper for our project and allows users to more easily interact with Logjam’s different functionalities. This includes creating the Logjam Docker image, spawning master and slave nodes for the elasticsearch cluster, and ingesting log bundles. To use the script, one can utilize the following flags:

* -b (build image)
* -c <name> (create elasticsearch master container <name>)
* -r <name> (create elasticsearch slave <name>)
* -i <container> <path to folder to ingest> (ingest log data on <container>)
* -d <container> (delete duplicates database on <container>)
* -e (creates the elasticsearch index)

**Final Status**

We have completed all iterations and implemented the required functionality. The tool can ingest compressed log files at a rate of approximately 1GB per hour.

**Recommendations**

*Peter Soloninka (primary author), Jake Ferrero (contributing editor)*

During this project, we identified several changes that could significantly improve performance and reliability that we did have time to experiment with. These recommendations can be found below.

**Run Logstash on a separate instance**

Currently Logstash runs on the same instance as the master Elasticsearch node, this means that during ingest they will contend for the same CPU and disk resources. Logstash can be run on a seperate machine to improve performance.

**Increase parallelism**

Consider running Logstash on a seperate machine with more CPUs. Logstash can run many workers in parallel so as long as the disk can read fast enough additional CPUs will provide a linear increase in ingest speed up to the capacity of the Elasticsearch cluster.

**Break-up log data more thoroughly (Add back in Grok filters)**

Our initial configuration had additional grok filters to break the message field into multiple fields. This slowed Logstash pipeline down by 10-20%, but also resulted in a lower cardinality for the message field and allowed it to be stored more efficiently. Consider adding these back to reduce index size.

**Configure Elasticsearch for EC2 cluster auto-discovery**

Currently the discovery and broadcast IP addresses for each node in the cluster are set manually. If the production version of Logjam will be run on EC2 the Elasticsearch cluster can be configured with the EC2 discovery plugin so that the IPs do not have to be manually configured.

**Add second pass for StorageGRID version**

The ingest script does not insure that a StorageGrid version number is found before sending each log file to Logstash so some log events may make it into Elasticsearch with that field set to “NA”. Consider creating a script to be run after a full ingest to make sure all version numbers are present.

**Test Plan and Results**

*Renata Zeitler (primary author), Joshua Good (contributing editor)*

**Blackbox**

The expected coverage for Logjam is to be at 80% branch and 100% method coverage for all scripts.

For Blackbox testing, we created 6 complex directories that contained corrupted (and uncorrupted) zip files, a file pertaining to each of the 22 categories (including an “other” category), and invalid file types. By running through these tests, we are able to view the output in Kibana to verify the files have been ingested and indexed properly. Whitebox testing will be described later.

**Setup**

The *ingest-tests* folder contains all testing material relevant to evaluating our ingestion script. Test directories are located within the *example-cases* directory and represent various category structures. These directories contain each type of file (.txt, .log, and messages) as well as a file pertaining to each of the 22 categories for simple testing. The *scripts* folder contains *ingest.py*. All tests are run from the command line.

**Kibana Setup**

If you are running Logjam remotely, you will need to bind Kibana to a port on your local system. Kibana natively runs on port 5601, so it is recommended that you bind it to the same local port, as such: “ssh -i <auth\_key> admin@<remote-host> -L 5601:<remote-host>:5601”. If you are not running Logjam remotely you may ignore the previous setup. Kibana will then be accessible via “localhost:5601”.

**Directory Structures**

The following directory structures are used to test varying inputs to the ingestion script. Each top-level directory represents a sample support case identifier provided by NetApp.

|  |  |
| --- | --- |
| **Structure ID** | **Hierarchy Description** |
| SimpleNotADirectory | Contains a single file labeled “bycast.log”. |
| SimpleOneDirectory | Contains the directory “2006629851” with file “bycast.log”. |
| ComplexNestedDirectories | Contains two directories “2006629851” and “2006642781”, each with one set of nested subdirectories. Each directory’s most-nested child contains a single file, “bycast.log”. |
| NestedDirectoryZipped | Contains the directory “2006767045”, structured with multiple nested directories and file types. “2006767045” includes the file “GID950140\_REV8\_GSPEC.zip” and directory “previous\_case\_2006629754” with file “bycast.log” at its most-nested child folder. |

**Acceptance Test Plan: *Ingest.py***

Test Plan Executed: 5/3/2018

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| 1.1 - Ingest No Command Line Arguments | **Preconditions:**   * **Setup** is complete * The current directory is *logjam*   **Execution:**   1. Run “python ingest.py” | Ingest.py reports: “python ingest.py [directory to ingest] [-v]” and exits. | Pass  python ingest.py [directory to ingest] [-v] |
| 1.2 - Ingest Too Many Command Line Arguments | **Preconditions:**   * **Setup** is complete * The current directory is *logjam*   **Execution:**   1. Run “python ingest.py -f -o -v” | Ingest.py reports: “python ingest.py [directory to ingest] [-v]” and exits. | Pass  python ingest.py [directory to ingest] [-v] |
| 1.3 - Ingest Not A Directory | **Preconditions:**   * **Setup** is complete * *example-cases* contains the *SimpleNotADirectory* folder. * The current directory is *logjam*   **Execution:**   1. Run “python ingest.py ../example-cases/SimpleNotADirectory/bycast.log” | Ingest.py reports: “../example-cases/SimpleNotADirectory/bycast.log is not a directory” and exits. | Pass  ../example-cases/SimpleNotADirectory/bycast.log is not a directory |
| 1.4 - Ingest Simple, Not Nested Directory | **Preconditions:**   * **Setup** is complete * *example-cases* contains the *SimpleOneDirectory* folder. * The current directory is *scripts*   **Execution:**   * Run “python ingest.py ../example-cases/SimpleOneDirectory -v | ingest.py reports the transaction information when moving bycast.log to logjam\_categories/bycast/ via stdout. | See **Test ID: 1.4** in **Blackbox Test Results** |
| 1.5 - Ingest Complex Nested Directories | **Preconditions:**   * **Setup** is complete * *example-cases* contains the *ComplexNestedDirectories* folder. * The current directory is *scripts*   **Execution:**   1. Run “python ingest.py ../example-cases/ComplexNestedDirectories -v | ingest.py reports the transaction for each support case’s Bycast log, including header information. | See **Test ID: 1.5** in **Blackbox Test Results** |
| 1.6 - Ingest Complex Nested Directory With Zipped File | **Preconditions:**   * **Setup** is complete * *example-cases* contains the *NestedDirectoryZipped* folder. * The current directory is *scripts*   **Execution:**   1. Run “python ingest.py ../example-cases/NestedDirectoryZipped -v | ingest.py reports the transaction for the folder containing bycast.log to stdout. It additionally unzips GID950140\_REV8\_GSPEC.zip and parses its contents. | See **Test ID: 1.6** in **Blackbox Test Results** |
| 2.1 - Kibana: View Default Dashboard | **Preconditions:**   * **Setup** is complete * **Kibana setup** is complete   **Execution:**   1. Click **“**Dashboard” on the left-hand menu 2. From the Dashboard list, select “Common Failure/Error Concentrations” 3. In the top-right corner select “Last 15 minutes” and change it to “This Year” | A dashboard displaying the following visualizations appears:   * StorageGRID failures by Category * Ingest Overview * Permission Denied by Category and Case Number * Connection Issues by Case Number and Category * Ext3-fs Errors by Category * Invalid Opcodes by Date and Category * I/O Errors By Category | Pass  A dashboard displaying the aforementioned visualizations is displayed. The user can view concentrations of different errors up to the most recently ingested log file, as well as an overview of all ingested log files. |

**Blackbox Test Results**

|  |  |
| --- | --- |
| **Test ID** | **Actual Results** |
| 1.4 | Pass  *ingest.py* reports:  Ingesting ..\example-cases\SimpleOneDirectory\  Checking if duplicate: ..\example-cases\SimpleOneDirectory\/2006642781/bycast.log  Renamed bycast/bycast.log to C:\Users\Joshua\Documents\CSC492\2018SpringTeam16\logjam\logjam\_categories/bycast/200664278  1-bycast.log-1525403799.00500011444091796875  Adding ..\example-cases\SimpleOneDirectory\/2006642781/bycast.log to db and Logstash  Finished |
| 1.5 | Pass  *ingest.py* reports:  Ingesting ..\example-cases\ComplexNestedDirectories  Checking if duplicate: ..\example-cases\ComplexNestedDirectories/2006629851/sg  This is a directory  Checking if duplicate: ..\example-cases\ComplexNestedDirectories/2006629851/sg/bycast.log  Renamed bycast/bycast.log to <file-path>\logjam\logjam\_categories/  1-bycast.log-1525403049.63199996948242187500  Adding ..\example-cases\ComplexNestedDirectories/2006629851/sg/bycast.log to db and Logstash  Checking if duplicate: ..\example-cases\ComplexNestedDirectories/2006642781/sg  This is a directory  Checking if duplicate: ..\example-cases\ComplexNestedDirectories/2006642781/sg/bycast.log  Renamed bycast/bycast.log to <file-path>\logjam\logjam\_categories/  1-bycast.log-1525403049.70300006866455078125  Adding ..\example-cases\ComplexNestedDirectories/2006642781/sg/bycast.log to db and Logstash  Finished |
| 1.6 | Pass  Ingesting ..\example-cases\NestedDirectoryZipped\  Checking if duplicate: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC.zip  Unzipping ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC.zip  Unzipping: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC.zip  Unzipping 18.2KB  Unzipping 100 % Checking if duplicate: ..\example-cases\NestedDire  ctoryZipped\/2006767045/GID950140\_REV8\_GSPEC/DC3  This is a directory  Checking if duplicate: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/DC3/FZR-DC-GN3.ovf  Flagging ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/DC3/FZR-DC-GN3.ovf  Assumming incorrect filetype: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/DC3/FZR-DC-GN3.ov  f  Checking if duplicate: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/DR3  This is a directory  Checking if duplicate: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/DR3/FZR-DR-GN3.ovf  Flagging ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/DR3/FZR-DR-GN3.ovf  Assumming incorrect filetype: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/DR3/FZR-DR-GN3.ov  f  Checking if duplicate: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/GID950140\_REV8\_GSPEC\_BOM.  csv  Flagging ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/GID950140\_REV8\_GSPEC\_BOM.csv  Assumming incorrect filetype: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/GID950140\_REV8\_GS  PEC\_BOM.csv  Checking if duplicate: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/GID950140\_REV8\_GSPEC\_Soft  wareBOM.csv  Flagging ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/GID950140\_REV8\_GSPEC\_SoftwareBOM.csv  Assumming incorrect filetype: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/GID950140\_REV8\_GS  PEC\_SoftwareBOM.csv  Checking if duplicate: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/GID950140\_REV8\_GSPEC\_VMBO  M.html  Flagging ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/GID950140\_REV8\_GSPEC\_VMBOM.html  Assumming incorrect filetype: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/GID950140\_REV8\_GS  PEC\_VMBOM.html  Checking if duplicate: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/[Content\_Types].xml  Flagging ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/[Content\_Types].xml  Assumming incorrect filetype: ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC/[Content\_Types].x  ml  Adding ..\example-cases\NestedDirectoryZipped\/2006767045/GID950140\_REV8\_GSPEC.zip to db and Logstash  Checking if duplicate: ..\example-cases\NestedDirectoryZipped\/2006767045/previous\_case\_2006629754  This is a directory  Checking if duplicate: ..\example-cases\NestedDirectoryZipped\/2006767045/previous\_case\_2006629754/bycast.log  Renamed bycast/bycast.log to <file-path>\logjam\logjam\_categories/bycast/200676704  5-bycast.log-1525403137.07899999618530273438  Adding ..\example-cases\NestedDirectoryZipped\/2006767045/previous\_case\_2006629754/bycast.log to db and Logstash |

**Whitebox**

For Whitebox testing, PyUnit provides an automated unit-testing framework. The suite allows us to extend the *TestCase* class towards evaluating ingestion script-specific features, including command line arguments, unzipping files, and transferring log data to Logstash. Additionally, the *TestSuite* class provides a grouping module for organizing our test cases, feasibly allowing us to run all cases with a single command.

**Unit Testing Information**

Coverage Tool: coverage.py

**Test Coverage Report**

Unit Testing: PyUnit

115/115 tests : 100%

**Overall Coverage Summary**

Method: 100%

Branch: 84%

Coverage: 88%

**Overall Stats Summary**

|  |  |
| --- | --- |
| Total scripts: | 3 |
| Total executable lines: | 269 |

**Task Plan**

*Joshua Good*

|  |  |  |  |
| --- | --- | --- | --- |
| **Team 16 Task Plan** | | | |
| **Item** | **Owner(s)** | **Due Date** | **Status** |
| Setup team Github repository   * Project board * Issues | All | 1-17-2018 | Complete |
| Renata | Complete |
| All | Complete |
| Contact IT about AWS hosting | Jake | 1-19-2018 | Complete |
| Get test data from Morgan | Jake | 1-19-2018 | Complete |
| Investigate Elastic Stack Suite   * Determine best logstash plugin option * Learn Logstash fundamentals (environment, how to send POST files, etc.) | All | 1-25-2018 | Complete |
| Jake, Peter | Complete |
| All | Complete |
| Revise Requirements   * Rephrase Ingestion feature description * Expand user stories * Solidify query parameters for Kibana visualizations | All | 1-29-2018 | Complete |
| Joshua | Complete |
| Jake, Peter | Complete |
| All | Complete |
| Preliminary Ingest/Indexing design | All | 1-31-2018 | Complete |
| Draft IPR | All | 2-02-2018 | Complete |
| * Executive Summary | Renata | Complete |
| * Project Description | Joshua | Complete |
| * Resources Needed | Jake | Complete |
| * External Dependencies | Jake | Complete |
| * Requirements | Jake | Complete |
| * Design | Peter | Complete |
| * Implementation | Peter | Complete |
| * Test Plan | Renata, Joshua | Complete |
| * Task Plan | Joshua | Complete |
| Draft OPR1 Slides | Joshua | 2-02-2018 | Complete |
| Get Logstash categories from Morgan | Joshua | 2-05-2018 | Complete |
| Draft Development Road Map | All | 2-05-2018 | Complete |
| * Split development into phases | Peter |
| Get bash script from Morgan | Joshua | 2-06-2018 | Complete |
| Draft initial Blackbox Test Plan | Joshua, Renata | 2-08-2018 | Complete |
| Final IPR Draft | All | 2-09-2018 | Complete |
| * Executive Summary | Renata | Complete |
| * Project Description | Joshua | Complete |
| * Resources Needed | Jake | Complete |
| * External Dependencies | Jake | Complete |
| * Requirements | Jake | Complete |
| * Design | Peter | Complete |
| * Implementation | Peter, Renata | Complete |
| * Test Plan | Renata, Joshua | Complete |
| * Task Plan | Joshua | Complete |
| Ingestion script (phase I) | Joshua, Renata | 2-21-2018 | Complete |
| * Read top-level directory via command line arguments | Joshua | Complete |
| * Flatten subdirectories | Renata | Complete |
| Logstash pipeline | Jake, Peter | 2-21-2018 | Complete |
| * Identify line breaks and metadata | Peter | Complete |
| * Parse events via Grok filter | Jake | Complete |
| OPR2 | Jake | 2-25-2018 | Complete |
| Ingestion script (phase 2) | Joshua, Renata | 2-27-2018 | Complete |
| * Remove copy file logic | Joshua, Renata | 2-27-2018 | Complete |
| * Handle varying zipped file formats | Renata | Complete |
| * POST log files to Logstash | Joshua | Complete |
| OPR3 | Peter, Renata, Jake | 3-26-2018 | Complete |
| * Send IPR to Fidelity team contact | Jake |  | Complete |
| * Film Kibana demo | Peter |
| * Prepare slides | Peter, Renata |
| Create standalone Docker image | Jake | 4-03-2018 | Complete |
| Posters and Pies draft poster | Jake, Peter, Renata | 4-06-2018 | Complete |
| Posters and Pies preview presentation | Josh | 4-06-2018 | Complete |
| Kibana Visualizations | Josh | 4-19-2018 | Complete |
| Finalized Poster and Pies presentation | Josh | 4-26-2018 | Complete |
| Posters and Pies demo setup | Jake, Peter | 4-26-2018 | Complete |
| Timestamp Truncation | Jake, Renata | 4-26-2018 | Complete |
| Final Progress Report | All | 5-08-2018 | Complete |
| Final User Guide | All | 5-08-2018 | Complete |
| Final Developer Guide | All | 5-08-2018 | Complete |
| Final Installation Guide | All | 5-08-2018 | Complete |
| Final Product Transfer to NetApp | All | 5-08-218 | Complete |

**Team Contact Information**

Joshua Good, Team Lead and Support Manager, jegood@ncsu.edu, (980) 213 - 6658

Jake Ferrero, Quality Lead, jaferrer@ncsu.edu, (704) 661 - 2565

Peter Soloninka, Development Lead, pasoloni@ncsu.edu, (919) 862-7448

Renata Zeitler, Planning Lead, razeitle@ncsu.edu, (919) 306-6499

**References**

[1] <https://hub.docker.com/r/sebp/elk/>

[2] <https://github.ncsu.edu/engr-csc-sdc/2018SpringTeam16/wiki/Installation-Guide>

[3] <https://github.ncsu.edu/engr-csc-sdc/2018SpringTeam16/>