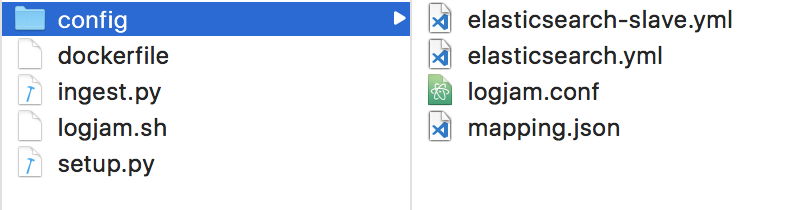
## Logjam: Developer’s Guide

## Where to Begin

If you would like to make modifications to any scripts or configurations we have made, start by pulling this repository onto your local machine and make any necessary edits. After finishing the modifications, push any changes to the server hosting the Docker containers. The dockerfile will detect any changes to the “logjam” directory structure (the directory it resides in) and it will integrate the modifications into the Logjam Image the next time you build the image. Because the dockerfile monitors the folder it resides in, you must ensure that all edited files (configs or scripts, both Python and bash) are contained within the same directory level as the dockerfile. As an example, the following is how we have our “logjam” directory laid out. The “logjam” directory contains everything needed for the project and will be stored on the machine that hosts the Docker containers.



The following are important scripts and configuration files you may want to modify in the future.

ingest.py if you would like to make improvements to the script that processes files setup.py if you would like to make changes to how the database and logjam\_category directory are setup elasticsearch.yml Replaces the default elasticsearch config on the master node. Broadcasts the (master) container’s IP, and then looks at the IPs in the zen-discovery section to connect to the slave containers. elasticsearch-slave.yml Replaces the default elasticsearch config on the slaves nodes. Broadcasts the (slave) container’s IP, and then looks at the IPs in the zen-discovery section to connect to the cluster. logjam.conf Contains the Logstash pipeline configuration. mapping.json Contains the custom mapping for the Elasticsearch index.

## Customizing the Logjam Image

Our Logjam image is based off of the sebp elk-docker image, found at http://elk-docker.readthedocs.io/. If you’d like to expand on the Logjam image, you can edit the dockerfile found in our GitHub repository. The dockerfile contains the packages that will be installed via apt-get as well as pip in the container. The dockerfile also ensures that any scripts needed by the container are copied from the local instance to the container.

## Developing with Docker

If you are developing on your local machine and testing inside an ELK container, you can simplify this process by mounting your development directory into the container. This way, the directory in the container will stay up-to-date with the directory on your local machine. You can do this with the following command:

docker run -v <PATH-TO-DEV-DIRECTORY-LOCAL-MACHINE>:<DESIRED-PATH-IN-CONTAINER> <container>

## Ingest.py

Ingest.py is a Python 2.7.14 script that manages Logstash support case ingestion. The script parses arbitrarily structured support case directories for StorageGRID logs relevant to the 21 most commonly identified log categories. These categories are specified by our sponsors and can be changed in this script to fit your needs. For each main feature, separate functions are defined for simple unit testing and code structure. Table 1 provides a high-level overview of each feature function.

|  |  |
| --- | --- |
| Function | Description |
| flatten(start, depth = None) | Recursively looks through directories for .txt and .log files to send to Logstash. Also calls other methods listed when appropriate to get StorageGrid version numbers and case numbers. Also logs the files in the database to prevent duplicates from entering Logstash |
| getSGVersion(key, path) | Obtains the version number from select files sent to it |
| unzip(path, extension) | Unzips the given file based on extension type. If an unsupported or improperly formatted extension is provided, reports an error. |
| updateToErrorFlag(path) | Marks a database entry to signify a file as corrupt. |
| getCategory(path, filename) | Checks a path for a category. If a valid one is not found, returns “other” |

Table 1: Ingest.py Functions Overview

These features directly correspond to how files are ingested in Logstash. Changes to header structure or file format in the Logstash configuration file must be reflected in ingest.py for proper transfer. For each new feature, a separate function should be created. This design structure simplifies unit testing as related functions can be grouped into their own test suites and preserves code readability.

## Logstash

Inside logjam/config you will find the logjam.conf file. This is a YAML file which defines the Logstash pipeline for Logjam.

The Logstash pipeline is configured to use the file input plugin to watch the various category directories. Most files are delimited by new lines, but the various command categories instead look for the <\-+> pattern. Then the grok plugin extracts metadata from the filename before the Elasticsearch plugin sends the event to the Elasticsearch index.

When making changes to logstash.conf it may be expedient to set Logstash to reload config files automatically. More information on this can be found here: https://www.elastic.co/guide/en/logstash/current/reloading-config.html. Otherwise, Logstash will need to be restarted before changes made to logstash.conf take effect.

## Elasticsearch

**Cluster:**

The cluster IP addresses are manually configured in logjam/config/elasticsearch.yml for the the master node and logjam/config/elasticsearch-slave.yml. network.publish\_host is the address the node will publish to the cluster so that the rest of the cluster knows how to reach that node. discovery.zen.ping.unicast.hosts is the list of addresses that the node should ping when attempting to join the cluster. Currently both nodes are set to use the best\_compression index codec which can be removed to improve index speed at the cost of a marginal increase in index size.

**Mappings:**

The Elasticsearch index uses a custom mapping which is defined in logjam/config/mapping.json. This mapping differs from the default mapping Elasticsearch generates in that it does not generate unnecessary text fields for category, casenumber, and categorized time fields.

## Kibana

If you are running the ELK stack in an AWS instance, in order to see Kibana on your local machine, you will need to set up the a tunnel between the AWS port 5601 (which Kibana listens on) and your local machine’s port 5601. To do this, you can run the following in a unix environment: ssh -i <PATH-TO-AWS-PUBLIC-KEY> admin@<AWS-INSTANCE-NAME>.amazonaws.com -L 5601:<AWS-INSTANCE-NAME>.amazonaws.com:5601