SWE 265P Reverse Engineering and Modeling

Architecture

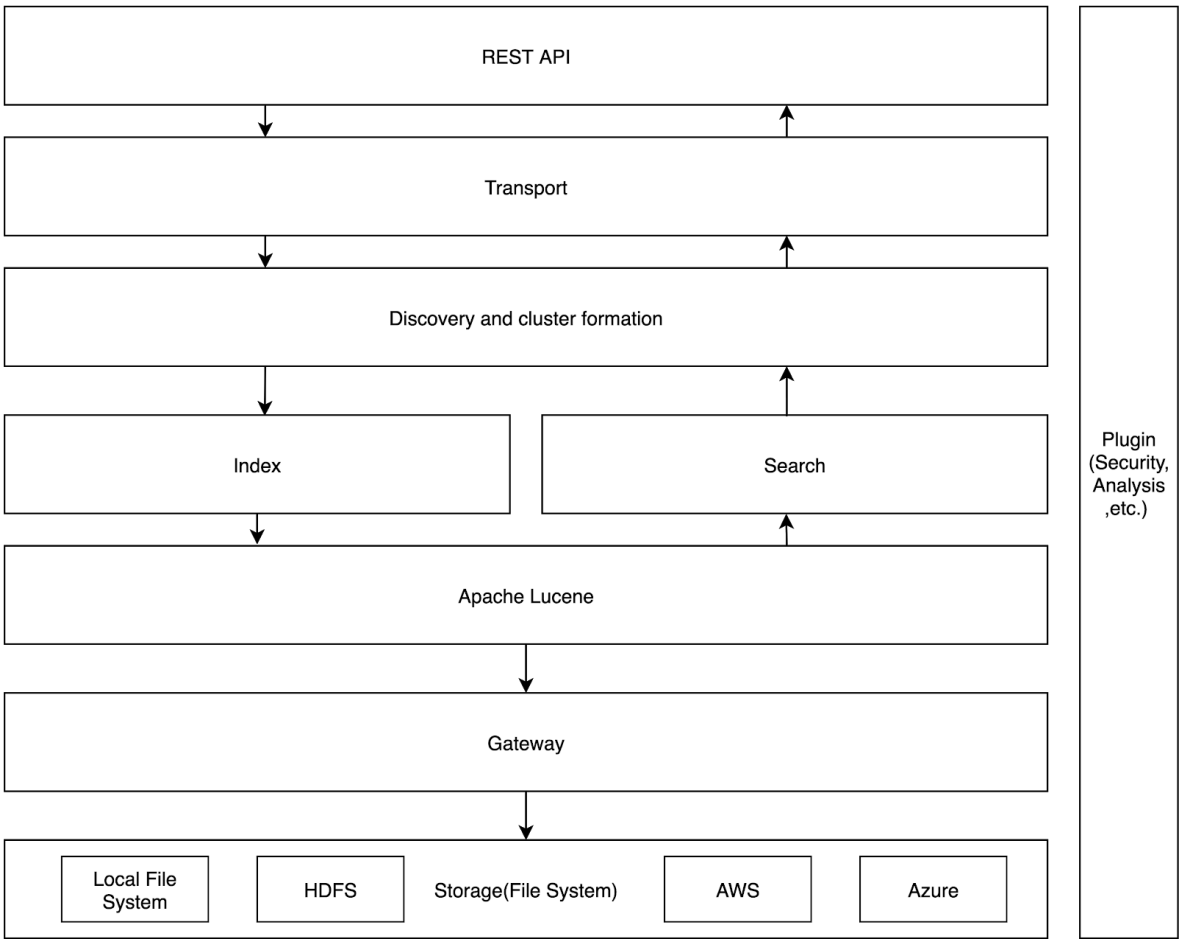

Fig 1.ElasticSearch Architecture diagram

The source code of elasticsearch is packed in packages. We first dug into its folder structure which is shown below.

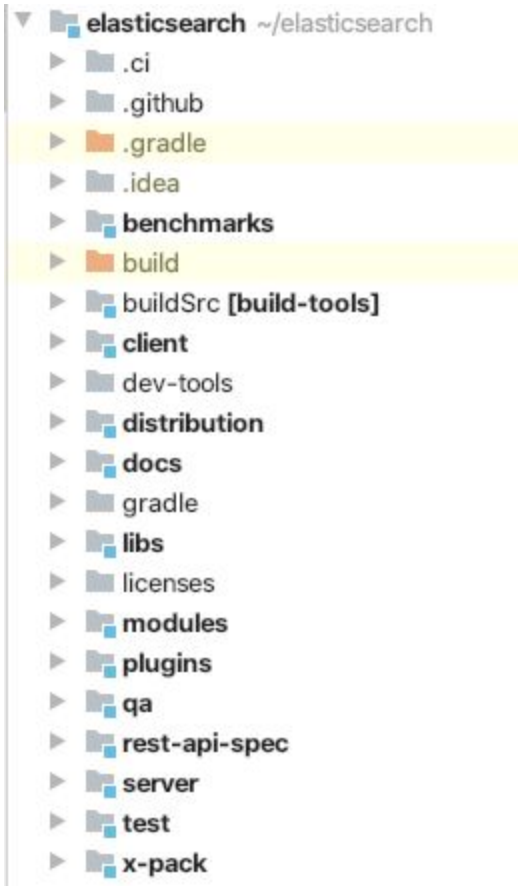

Fig 2. Folder structure of elasticsearch

We looked into each package, except for those trivial parts such as benchmarks, distribution, etc,. What we really care about are client and server.
The client package mainly contains the REST API related code, the most important package is the server package.

The folder structure of the server package is shown below:

Fig 3. Folder structure of elasticsearch

As we can see, it contains apache.lucene and elasticsearch. Apache Lucene is the module that does the actual indexing and searching job.

Going down into the org.elasticsearch package shown below.

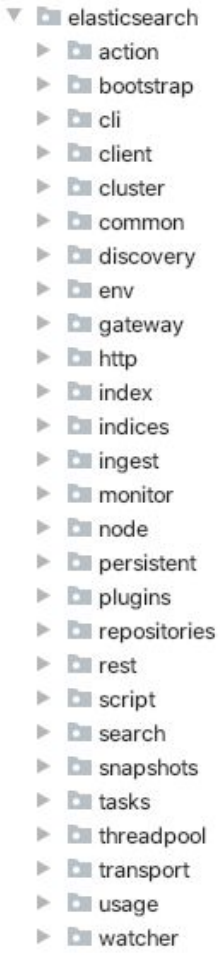

Fig 4. Folder structure of org.elasticsearch

There are familiar packages such as transport, index, and search. Based on what we have learned in the previous homework, we could draw the following initial diagram.

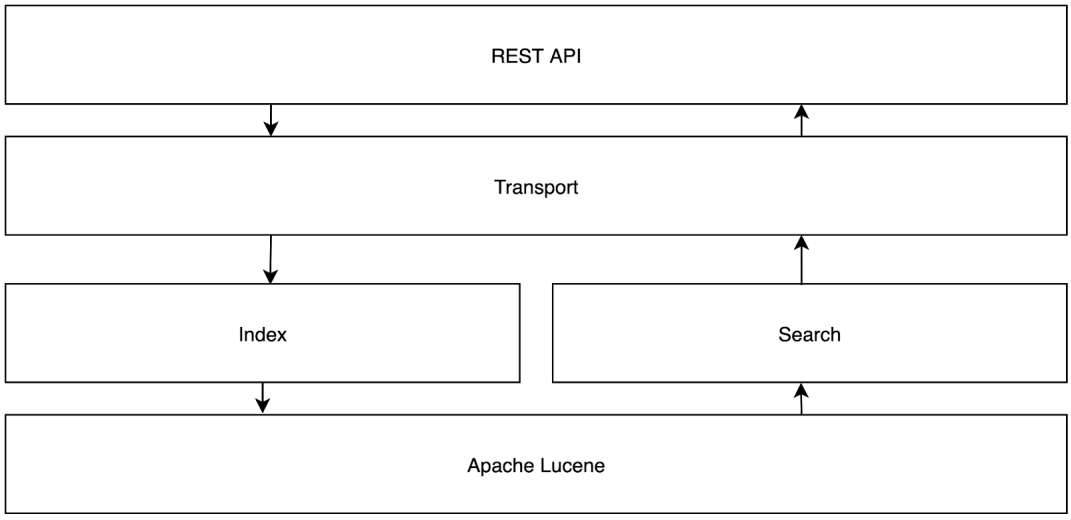

Fig 5. Initial partial architectural diagram

- REST API: Provides the Rest API for external users.
- Transport: Manages the transport networking layer of elasticsearch, which is used internally to realize the transportations between nodes within the system.
- Index: Implements the creation and management of all indices globally. Controls the index-related configure.
- Search:
- Apache Lucene: Implements the indexing and searching feature at the lowest level.

There are also packages we don't know what they do, but we found some helpful information in the official documentation[1], which briefly describes the responsibility and functionality of each module. The documentation also elaborated the relationship and interaction between different components. We verified some of our previous thoughts and refined the architecture we drew according to the documentation by adding the following important modules:

- Gateway: Responsible for data persistence and recovery
- Storage: Data storage. We have found support for local file systems, HDFS, Amazon S3, Azure and Google Cloud.

- Discovery and cluster formation: Responsible for discovering nodes, master node election, updating cluster states
- Plugins: Plugins that extend ElasticSearch functionality. For instance, the analyzer plugin and security plugin.

In conclusion, we recovered Elasticsearch's as-implemented architecture based on its folder structure and features. We figured out the relationship between models and decided which ones

to group, which should be separate apart, and how different parts interact with each other in accordance with our domain knowledge and previous study on the system. After drawing the draft of the architecture, we modified and adjusted it with the help of the official documentation.

Social Context

Key developers and State of the project

The key developers including employees of Elastic company and GitHub contributors since it is an open-source project. The project is still in iterative development, bug fixing, and releasing phase. There are altogether 50783 commits and 2211 open issues. During the last week(Feb 19 - Feb 26), 68 contributors pushed 136 commits to the master branch and 370 commits to all branches. 59 pull requests were proposed and 223 were merged. There were 61 new issues opened and 75 issues closed[2]. The latest version 7.6.0 was released two weeks ago on Feb 11[3].

Standards and Guidelines

As a contributor, we should adhere to the project standards in different aspects.
Elasticsearch uses the Eclipse JDT formator with the Spotless Gradle plugin to format java files in the project. We can find the coding standard in Java Language Formatting Guidelines**[4]**. Includes: 1) 4 spaces java indent; 2) 140 characters Line width; 3) Keep lines of code surrounded by // tag::NAME and // end::NAME comments no more than 76 characters long 4) No Wildcard imports; etc.

The testing standard is also documented in TESTING**[5]**. And for documentation, Elasticsearch provides two versions of Javadoc standard**[6]**, the short version and the long version. The short version regulates that 1) Javadoc should always be added to new code; 2) Adding Javadoc to existing code is encouraged; 3) Document the reasons rather than the process; 4) Don't document trivial or obvious things.

The issue guidelines and the pull request standards are also provided[7][8]. When opening an issue, we should describe the feature with all of the environmental information (OS, JVM, tools, etc.). The problem, the expected and the actual behavior in the scenario, how to reproduce the problem, and other related information should be provided in detail. When submitting a pull request, the contributor should make sure to 1) Sign the Contributor License Agreement; 2) Follow the contributor guidelines; 3) Build locally with gradle check before submission; 4) Pull requests against master would be better; 5) Check whether the OS and architecture of the submission are supported. 6) Follow the guidelines about contributing as a part of a class.

Process of contributing

The Elastic company also regulates the process of contributing to the project[9].
For bug reporting, the contributor should 1) Test against the latest version of Elasticsearch since the problem may have been fixed; 2) Check whether there are already similar issues in the

issue list; 3) Provide test cases as curl commands to reproduce the bug; 4) Provide as much information as impossible in order to find and fix the bugs easier and faster.
For feature requests, we can open an issue that describes the feature, its functionality, and its mechanism.
For Code and Documentation changes, the contributing process is: 1) Find or open an issue about the change; 2) Fork the project to his/her own repository; 3) Test the change to make sure nothing is broken because of it; 4) Sign the Contributor License Agreement; 4) Rebase the change; 5) Submit a pull request and mention the issue about the change;

Tools

During the process, JDK13 and Gradle wrapper are required to build the project. Some Elasticsearch artifacts and test cases require Docker. Eclipse(>=4.13) and IntelliJ(>=2017.2) are recommended IDEs. Some built-in plugins of the IDEs are also being used, such as the coverage analysis tools. GitHub is used as the source code repository as well as the issue tracker. Eclipse JDT formator with the Spotless Gradle plugin is used for code format reviewing.

Interesting Pull Requests

There are around 30k pull requests on GitHub and here are some interesting ones.

- **Allow specifying highlighter parameters on a per-field level basis #356**
  This is the first pull request on Github in 2010. According to the official website of Elastic, one of the key developers who now is the CEO of Elastic built a search engine for his wife's list of recipes. But on GitHub, the first pull request was a feature enhancement that improved the performance of the highlighter.

- **Upgrade to Gradle 6.2.1 #52732**
  This pull request updates the version of Gradle to the latest patch release which fixes a few bugs. It only modified the file gradle/wrapper/gradle-wrapper.properties to change the version of the Gradle with a single line. This pull request inspires us to pay attention to the versions of the dependencies of our system and it is worth a pull request.

- **exec the java process #368**
  This pull request fixed a problem by making java process exec in foreground mode but it will cause more other problems like the logger will produce logs both to files and console. So this pull request was closed. It reminds us that when we are going to fix a bug, we should avoid producing more bugs.

- **Fix readme typo #882**
  This pull request found a typo of the readme file and corrected it from 'fist' to 'first'. It reminds us that it is important to focus on the details and a typo fixing worth for a pull request.

- **Proposal for a new Super Hero ;-) #1385**
  This is a pull request from a big fan of the key developer & CEO of Elastic, Shay Banon. It proposed to add Shay's name to a config file. Here's how it says in the pull request: "If MJ Watson is in, so Shay must be in too. MJ has perhaps saved Peter multiple times, but she never changed my life as Shay did with ES."

Interesting Issues

- **Tune the number of connections per remote cluster. #52665**
  This issue describes a feature that allows us to define the number of connections for every cluster. The current setting cluster.remote.connections_per_cluster applies to all clusters. In this case, someone has a small cluster and a big cluster, the connection of the big cluster is limited by the small cluster.

- **Query DSL: Terms Filter #1**
  The first issue in history! It proposed to add a terms filter that allows multiple terms in query for a specific field. For example:

```
{
    filteredQuery : {
        query : {
            term : { "name.first" : "shay" }
        },
        filter : {
            terms : {
                "name.last" : ["banon",
"kimchy"]
            }
        }
    }
}
```

- **Rename `user.username` to `user.name` foSetSecurityUserProcessor #51799**
  This issue suggests changing a variable name that does not conform to the naming convention documented in Elastic common schema.

- **Ranking Evaluation API: Allow multiple metrics per request #51680**
  This issue proposed to allow multiple ranking eval metrics for every query. Sometimes people want multiple ranking evaluation metrics in order to better understand the relevance. Elasticsearch currently only allows one metric per request. If we want to apply a different metric, we have to rerun the same request. It would be nice if es can add the multiple metrics feature.

- **Add Cache-Control or Expires header to has_privileges request when using API Key #51860**
  The **has_privileges** API is used to determine if a user or an API key has certain privileges. API keys can have an expiration time, and currently when making requests

using an API key, it does not return any information about whether the API key has expired. The author of this issue suggests if and Cache-Control or Expires header can be returned, it would be much more convenient for the client-side to cache the API key.

Reference

[1] https://www.elastic.co/guide/en/elasticsearch/reference/current/modules.html
[2] https://github.com/elastic/elasticsearch/releases
[3] https://github.com/elastic/elasticsearch/pulse
[4] https://github.com/elastic/elasticsearch/blob/master/CONTRIBUTING.md#Java-Language-Formatting-Guidelines
[5] https://github.com/elastic/elasticsearch/blob/master/TESTING.asciidoc
[6] https://github.com/elastic/elasticsearch/blob/master/CONTRIBUTING.md#javadoc
[7] https://github.com/elastic/elasticsearch/blob/master/.github/ISSUE_TEMPLATE.md
[8] https://github.com/elastic/elasticsearch/blob/master/.github/PULL_REQUEST_TEMPLATE.md
[9] https://github.com/elastic/elasticsearch/blob/master/CONTRIBUTING.md