VIVEKANAND EDUCATION SOCIETY'S
INSTITUTE OF TECHNOLOGY

Department of Computer Engineering

Report on

Managing Container Logs and build an ELK service using Docker

Faculty Incharge
Richard Joesph

Submitted by
Group No. 5
Ishaan Malhi (D17A, Roll no : 39)
Manish Manghwani (D17A, Roll No : 41)
Deepika Khatri (D17A, Roll no : 31)

(2016-17)

Table of Contents

1. Abstract

Logging is one of the essential parts in any setup because logs are extremely useful when it comes to troubleshooting problems happening with your production app. When you decide to move your apps to the docker-managed environments, we realise that some of the approaches we were using don't work anymore — now the logs are staying inside the container and we need to find a way to make them persistent. It goes without saying that the easiest way is to put the logs to the volume but what if we have many instances of the app? In this case we have to use a centralised storage for logs.

We've decided to store our logs in one place using the ELK stack. ELK is a centralised log server and log management web interface which includes Elasticsearch (as a storage and a search engine), Logstash (a tool for log processing), and Kibana (UI for the Elasticsearch).

2. Introduction

What is Docker?

Docker is the world's leading software container platform. Developers use Docker to eliminate "works on my machine" problems when collaborating on code with co-workers. Operators use Docker to run and manage apps side-by-side in isolated containers to get better compute density. Enterprises use Docker to build agile software delivery pipelines to ship new features faster, more securely and with confidence for both Linux and Windows Server apps.

What is a Container?

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure. Using containers, everything required to make a piece of software run is packaged into isolated containers. Unlike VMs, containers do not bundle a full operating system - only libraries and settings required to make the software work are needed. This makes for efficient, lightweight, self-contained systems and guarantees that software will always run the same, regardless of where it's deployed.
Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware,containers are more portable and efficient.

Docker For Developers

Docker automates the repetitive tasks of setting up and configuring development environments so that developers can focus on what matters: building great software.
Developers using Docker don't have to install and configure complex databases nor worry about switching between incompatible language toolchain versions. When an app is dockerized, that complexity is pushed into containers that are easily built, shared and run. Onboarding a co-worker to a new codebase no longer means hours spent installing software and explaining setup procedures. Code that ships with Dockerfiles is simpler to work on: Dependencies are pulled as neatly packaged Docker images and anyone with Docker and an editor installed can build and debug the app in minutes.

What is the ELK Stack?

The ELK stack consists of **E**lasticsearch, **L**ogstash, and **K**ibana. Although they've all been built to work exceptionally well together, each one is a separate project that is driven by the open-source vendor Elastic—which itself began as an enterprise search platform vendor. It has now become a full-service analytics software company, mainly because of the success of the ELK stack. Wide adoption of Elasticsearch for analytics has been the main driver of its popularity.

Data constantly flows into your systems, but it can quickly grow to be large and stale. As your data set grows larger, your analytics will slow up, resulting in sluggish insights. And this is likely to be a serious business problem. So, the BIG question for your big data is: how can you maintain valuable business insights?

Not long ago, an epiphany ran through the industry: analytics is, in essence, a search problem that needs coupling with good visualizations. So, there was a marriage: Lucene with all its search goodness was brought together with the distributed-computing goodness that is Elasticsearch. Logstash came onto the scene to normalize all kinds of time-series data. Pop in Kibana's ultra-simple visualization tool, and you have a complete analytics tool that can rival very expensive and scalable solutions from Oracle, Palantir, Tableau, Splunk, Microsoft, and others.

The ELK stack makes it much easier -- and much faster -- to search and analyze large data sets as your data grows larger and larger.

We should mention that ELK is quite versatile. Use the stack as a stand-alone application, or integrate with your existing applications to get the most current data. With Elasticsearch, you get all the features to make real-time decisions-all the time. You can use each of these tools separately, or with other products. For example, Kibana often goes together with Solr/Lucene. Although none of these is a project of the Apache Foundation, each part of the stack falls under the Apache 2 License. Elasticsearch owns both the intellectual property and the trademarks.

- **Elasticsearch — The Log Search Tool**
  Elasticsearch is a juggernaut solution for your data extraction problems. A single developer can use it to find the high-value needles underneath all of your data haystacks.
- **Real-time data and real-time analytics.** The ELK stack gives you the power of real-time data insights, with the ability to perform super-fast data extractions from virtually all structured or unstructured data sources. Real-time extraction, and real-time analytics. Elasticsearch is the engine that gives you both the power and the speed.
- **Scalable, high-availability, multi-tenant.** With Elasticsearch, you can start small and expand it along with your business growth-when you are ready. It is built to scale horizontally out of the box. As you need more capacity, simply add another node and let the cluster reorganize itself to accommodate and exploit the extra hardware. Elasticsearch clusters are resilient, since they automatically detect and remove node failures. You can set up multiple indices and query each of them independently or in combination.
- **Full text search.** Under the cover, Elasticsearch uses Lucene to provide the most powerful full-text search capabilities available in any open-source product. The search features come with multi-language support, an extensive query language, geolocation support, and context-sensitive suggestions, and autocompletion.
- **Document orientation.** You can store complex, real-world entities in Elasticsearch as structured JSON documents. All fields have a default index, and you can use all the indices in a single query to get precise results in the blink of an eye.

Logstash — Routing Your Log Data

Logstash is a tool for log data intake, processing, and output. This includes virtually any type of log that you manage: system logs, webserver logs, error logs, and app logs. As administrators,

we know how much time can be spent normalizing data from disparate data sources. We know, for example, how widely Apache logs differ from NGINX logs.

Rather than normalizing with time-taking ETL (Extract, Transform, and Load), we recommend that you switch over to the fast track. Instead, you could spend much less time training Logstash to normalize the data, getting Elasticsearch to process the data, and then visualize it with Kibana. With Logstash, it's super easy to take all those logs and store them in a central location. The only prerequisite is a Java runtime, and it takes just two commands to get Logstash up and running.
Using Elasticsearch as a backend datastore and Kibana as a frontend dashboard, Logstash will serve as the workhorse for storage, querying and analysis of your logs. Since it has an arsenal of ready-made inputs, filters, codecs, and outputs, you can grab hold of a very powerful feature-set with a very little effort on your part.
Think of Logstash as a pipeline for event processing: it takes precious little time to choose the inputs, configure the filters, and extract the ***relevant, high-value*** data from your logs. Take a few more steps, make it available to Elasticsearch and you get super-fast queries against your mountains of data.

Kibana — Visualizing Your Log Data

Kibana is your log-data dashboard. Get a better grip on your large data stores with point-and-click pie charts, bar graphs, trendlines, maps and scatter plots. You can visualize trends and patterns for data that would otherwise be extremely tedious to read and interpret. Eventually, each business line can make practical use of your data collection as you help them customize their dashboards. Save it, share it, and link your data visualizations for quick and smart communication.

How Can I Use the ELK Stack to Manage my Log Data?

Your critical business questions have answers in logs of your applications and systems, but most potential users of the data in those logs assume that the accessibility barrier is too high. But the answers are in there -- answers to questions such as:

- How many account signups this week?
- What is the effectiveness of our ad campaign?
- What is the best time to perform system maintenance?
- Why is my database performance slow?

The answers are in your data, but there are a number of challenges.

The Challenge of Disparate Log Data

If you're like most of us, your log data universe is inconsistent, inaccessible, and inconsistent. We understand problems such as these quite well:

- **No Consistency —** The variety of systems and absence of standards means that it's difficult to be a jack-of-all trades.
    - Logging is different for each app, system, or device
    - Specific knowledge is necessary for interpreting various types of logs
    - Variation in format makes it challenging to search
    - Many types of time formats
- **No centralization —** Simply put, log data is everywhere:
    - Logs in many locations on various servers

    - Many locations of various logs on each server
    - SSH + GREP doesn’t scale or reach
- **Accessibility of Log Data —** Much of the data is difficult to locate and manage. Although some of the log data may be highly valuable, many admins face these steep challenges:
    - Access is often difficult
    - High expertise to mine data
    - Logs can be difficult to find
    - Immense size of Log Data

The ELK stack can help you manage each of these challenges, and more. ELK is best for time-series data-anything with a timestamp-such as you'll find in most web server logs, transaction logs, and stock data listings. To be intelligible, these logs usually need substantial clean-up.

For example, you'll need to normalize in cases where you have some logs in which the timestamp is the North American dd/mm/yy, and others are the European mm/dd/yy. Logstash handles normalizations like this and many others with ease.

3. Implementation

We have deployed a scaffolded Ruby on Rails app and configured it to output its logs to the console. You will need a configured Rails environment, Docker and Docker Compose.

1. Putting the Rails app into Docker container
To get started, we need to have a working Rails app, we can use any existing app you have or create a new one:

$ rails new -d postgresql app

2. In order to support Docker environment, change the development environment in *app/config/database.yml* :

```
development: &default
adapter: postgresql
encoding: unicode
database: postgres
username: postgres
host: db
```

3. Setup JS runtime for the app — just find the following line in the Gemfile and uncomment it:

gem 'therubyracer', platforms: :ruby

4. Add Dockerfile to the *app* folder with the following content:

```
FROM ruby:2.3
ENV RAILS_ENV development
WORKDIR /app
ADD Gemfile Gemfile.lock /app/
RUN bundle install -j5 --retry 10
ADD . /app
```

5. Create a directory called *db* and add Dockerfile for postgres container:

```
FROM postgres
ENV POSTGRES_USER 'postgres'
ENV POSTGRES_DB 'app_development'
```

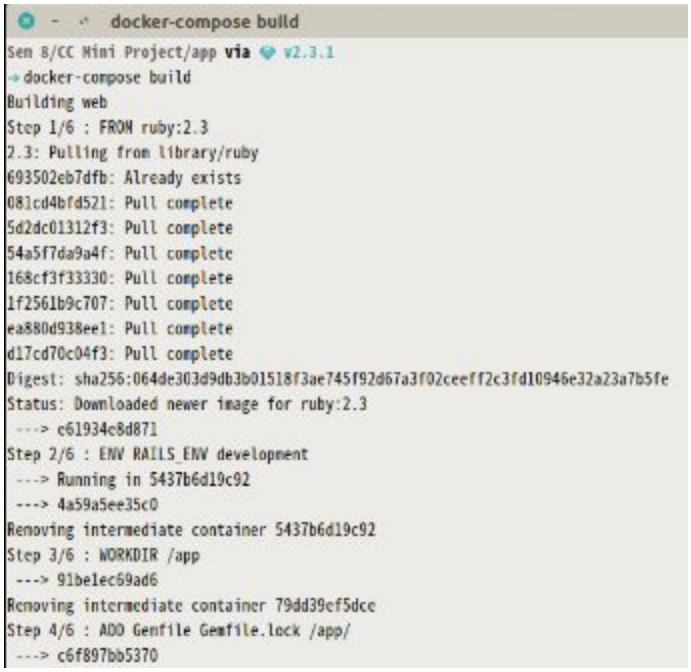

Figure 1 : Docker Compose for RAILS

Figure 2 : Docker Compose for RAILS

6. When the environment variables are set up, the postgres image will create a database. Create file *docker-compose.yml* in the project root directory:

```
version: '2'
services:
 web:
   build:
     context: ./app
     dockerfile: Dockerfile
   environment:
     RAILS_ENV: development
   ports:
     - '3000:3000'
   command: rails s -b 0.0.0.0

 db:
   build:
     context: ./db
     dockerfile: Dockerfile
```

7. Now, build and run the containers using the following commands:

```
$ docker-compose build
$ docker-compose up
```

As a result we are able to see our app running on http://localhost:3000.

```
College work/Sem 8/CC Mini Project
→ docker-compose build
Building web
Step 1/6 : FROM ruby:2.3
 ---> e61934e8d871
Step 2/6 : ENV RAILS_ENV development
 ---> Using cache
 ---> 4a59a5ee35c0
Step 3/6 : WORKDIR /app
 ---> Using cache
 ---> 91belec69ad6
Step 4/6 : ADD Gemfile Gemfile.lock /app/
 ---> Using cache
 ---> c6f897bb5370
Step 5/6 : RUN bundle install -j5 --retry 10
 ---> Using cache
 ---> 919e7d20668b
Step 6/6 : ADD . /app
 ---> 67572583b383
Removing intermediate container 86bc919013d3
Successfully built 67572583b383
Building db
Step 1/3 : FROM postgres
 ---> 4e18b2c30f8d
Step 2/3 : ENV POSTGRES_USER 'postgres'
 ---> Using cache
 ---> 540a80b16711
Step 3/3 : ENV POSTGRES_DB 'app_development'
 ---> Using cache
 ---> 86f8ef7299e1
Successfully built 86f8ef7299e1
elasticsearch uses an image, skipping
Building kibana
```

Figure 3 : Docker Compose for ELK

```
Status: Downloaded newer image for kibana:latest
 ---> 815950a52a6e
Step 2/6 : RUN apt-get update && apt-get install -y netcat
 ---> Running in 0331bd7eb1d1
Get:1 https://artifacts.elastic.co stable InRelease [310 B]
Ign https://artifacts.elastic.co stable InRelease
Get:2 https://artifacts.elastic.co stable Release.gpg [473 B]
Get:3 https://artifacts.elastic.co stable Release [4360 B]
Get:4 https://artifacts.elastic.co stable/main amd64 Packages [11.0 kB]
Get:5 http://security.debian.org jessie/updates InRelease [63.1 kB]
Ign http://deb.debian.org jessie InRelease
Get:6 http://security.debian.org jessie/updates/main amd64 Packages [448 kB]
Get:7 http://deb.debian.org jessie-updates InRelease [145 kB]
Get:8 http://deb.debian.org jessie Release.gpg [2373 B]
Get:9 http://deb.debian.org jessie Release [148 kB]
Get:10 http://deb.debian.org jessie-updates/main amd64 Packages [17.6 kB]
Get:11 http://deb.debian.org jessie/main amd64 Packages [9049 kB]
Fetched 9889 kB in 16s (612 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following extra packages will be installed:
  netcat-traditional
The following NEW packages will be installed:
  netcat netcat-traditional
0 upgraded, 2 newly installed, 0 to remove and 1 not upgraded.
Need to get 75.3 kB of archives.
After this operation, 194 kB of additional disk space will be used.
```

Figure 4 : Docker Compose for ELK

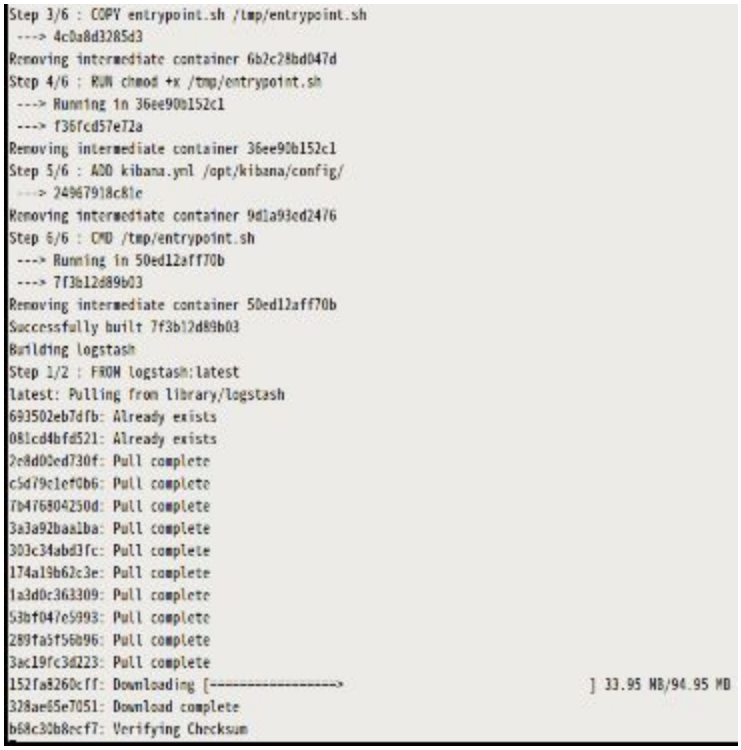

Figure 5 : Docker Compose for ELK

8. Setting up ELK stack

Create a folder called *logstash* in the root directory and put the Dockerfile with the following content into it:

FROM logstash:latest

ADD logstash.conf /etc/logstash/conf.d/

```
Step 6/6 : CMD /tmp/entrypoint.sh
 ---> Using cache
 ---> 7f3b12d89b03
Successfully built 7f3b12d89b03
Building logstash
Step 1/2 : FROM logstash:latest
latest: Pulling from library/logstash
693502eb7dfb: Already exists
081cd4bfd521: Already exists
2e8d00ed730f: Pull complete
c5d79e1ef0b6: Pull complete
7b476804250d: Pull complete
3a3a92baa1ba: Pull complete
303c34abd3fc: Pull complete
174a19b62c3e: Pull complete
1a3d0c363309: Pull complete
53bf047e5993: Pull complete
289fa5f56b96: Pull complete
3ac19fc3d223: Pull complete
152fa8260cff: Pull complete
328ae65e7051: Pull complete
b68c30b8ecf7: Pull complete
Digest: sha256:41963687ba3dc31b866ed03f96fa4753496ddf16271b3e2d1d1c709d43c3500c
Status: Downloaded newer image for logstash:latest
 ---> 413f93f5f1f2
Step 2/2 : ADD logstash.conf /etc/logstash/conf.d/
 ---> 659b68a6019f
Removing intermediate container 19cc8cbd4db2
Successfully built 659b68a6019f
```

Figure 6 : Building logstash

9. We also should provide a config file called *logstash.conf* (it should be placed into the same directory as our Dockerfile). We are going to consume logs from the GELF Docker log driver, so we are adding it as our input, forwarding our logs to the *elasticsearch* output, and setting up a host for it:

```
input {
  gelf {}
}

output {
  elasticsearch {
    hosts => "elasticsearch:9200"
  }
}
```

10.Create a folder called *kibana* in the root directory and put the Dockerfile with the following content into it:

```
FROM kibana:latest
RUN apt-get update && apt-get install -y netcat
COPY entrypoint.sh /tmp/entrypoint.sh
RUN chmod +x /tmp/entrypoint.sh
ADD kibana.yml /opt/kibana/config/
CMD ["/tmp/entrypoint.sh"]
```

11. Kibana should always start after the Elasticsearch so we have to add a custom entry point for our Dockerfile:

```
#!/usr/bin/env bash
while true; do
 nc -q 1 elasticsearch 9200 2>/dev/null && break
done
exec kibana
```

12. Our *kibana.yml* config file will look like this:

```
port: 5601
host: "0.0.0.0"
elasticsearch_url: "http://elasticsearch:9200"
elasticsearch_preserve_host: true
kibana_index: ".kibana"
default_app_id: "discover"
request_timeout: 300000
shard_timeout: 0
verify_ssl: true
bundled_plugin_ids:
- plugins/dashboard/index
- plugins/discover/index
- plugins/doc/index
- plugins/kibana/index
- plugins/markdown_vis/index
- plugins/metric_vis/index
- plugins/settings/index
- plugins/table_vis/index
- plugins/vis_types/index
- plugins/visualize/index
```

13. The final step is to add services with our containers to the *docker-compose.yml*:

```
logstash:
 build: logstash/
 command: logstash -f /etc/logstash/conf.d/logstash.conf
 ports:
   - "12201:12201/udp"
elasticsearch:
 image: elasticsearch:latest
 command: elasticsearch -Des.network.host=0.0.0.0
 ports:
   - "9200:9200"
   - "9300:9300"
kibana:
 build: kibana/
 ports:
   - "5601:5601"
```

```
Creating network "app_default" with the default driver
Creating app_web_1
Creating app_db_1
Attaching to app_db_1, app_web_1
db_1    | The files belonging to this database system will be owned by user "postgres".
db_1    | This user must also own the server process.
db_1    |
db_1    | The database cluster will be initialized with locale "en_US.utf8".
db_1    | The default database encoding has accordingly been set to "UTF8".
db_1    | The default text search configuration will be set to "english".
db_1    |
db_1    | Data page checksums are disabled.
db_1    |
db_1    | fixing permissions on existing directory /var/lib/postgresql/data ... ok
db_1    | creating subdirectories ... ok
db_1    | selecting default max_connections ... 100
db_1    | selecting default shared_buffers ... 128MB
db_1    | selecting dynamic shared memory implementation ... posix
db_1    | creating configuration files ... ok
db_1    | running bootstrap script ... ok
db_1    | performing post-bootstrap initialization ... ok
db_1    | syncing data to disk ...
db_1    | WARNING: enabling "trust" authentication for local connections
db_1    | You can change this by editing pg_hba.conf or using the option -A, or
db_1    | --auth-local and --auth-host, the next time you run initdb.
db_1    | ok
db_1    |
db_1    | Success. You can now start the database server using:
db_1    |
db_1    |     pg_ctl -D /var/lib/postgresql/data -l logfile start
db_1    |
db_1    | ****************************************************
db_1    | WARNING: No password has been set for the database.
```

Figure 7 : Docker Compose Up

```
db_1    | /usr/local/bin/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*
db_1    |
db_1    | LOG:  received fast shutdown request
db_1    | LOG:  aborting any active transactions
db_1    | waiting for server to shut down....LOG:  autovacuum launcher shutting down
db_1    | LOG:  shutting down
db_1    | LOG:  database system is shut down
db_1    |  done
db_1    | server stopped
db_1    |
db_1    | PostgreSQL init process complete; ready for start up.
db_1    |
db_1    | LOG:  database system was shut down at 2017-03-15 18:47:36 UTC
db_1    | LOG:  MultiXact member wraparound protections are now enabled
db_1    | LOG:  database system is ready to accept connections
db_1    | LOG:  autovacuum launcher started
web_1   | Started GET "/" for 172.19.0.1 at 2017-03-15 18:47:51 +0000
web_1   | Cannot render console from 172.19.0.1! Allowed networks: 127.0.0.1, ::1, 127.0.0.0/127.255.255.255
web_1   | Processing by Rails::WelcomeController#index as HTML
web_1   |   Parameters: {"internal"=>true}
web_1   |   Rendering /usr/local/bundle/gems/railties-5.0.2/lib/rails/templates/rails/welcome/index.html.erb
web_1   |   Rendered /usr/local/bundle/gems/railties-5.0.2/lib/rails/templates/rails/welcome/index.html.erb (30.1ms)
web_1   | Completed 200 OK in 72ms (Views: 49.2ms | ActiveRecord: 0.0ms)
web_1   |
web_1   |
^CGracefully stopping... (press Ctrl+C again to force)
Stopping app_db_1 ... done
Stopping app_web_1 ... done
```

Figure 8 : Docker Compose Up

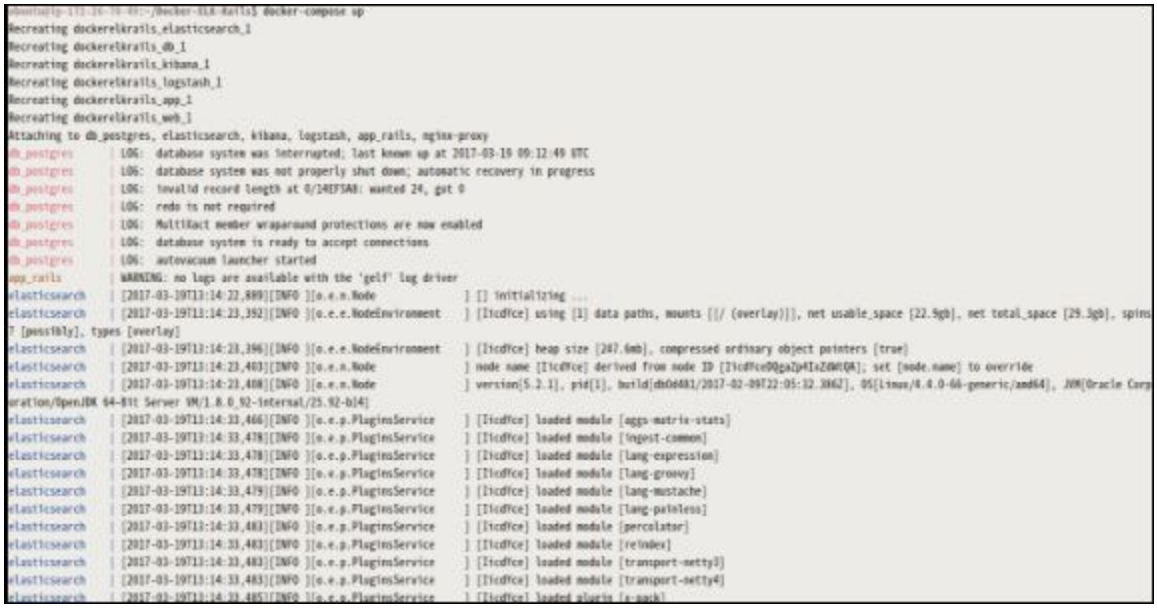

Figure 9 : Docker Compose Up

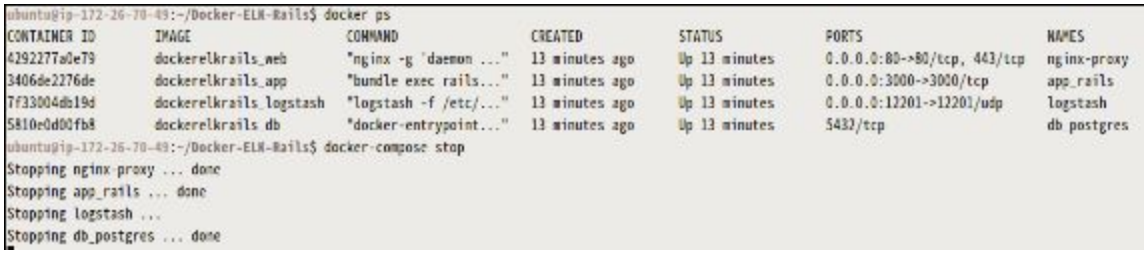

Figure 10 : Docker ps

14. After building and running the services we'll be able to access Kibana on http://localhost:5601. If we want the logs to be persistent — we just need to put our Elasticsearch data folder (*/usr/share/elasticsearch/data*) to a volume.

15. Sending Rails logs to Logstash

In order to forward our container logs to logstash we should just turn on GELF driver for our *web* container and set up a host URL:

```
web:
 ...
 logging:
   driver: gelf
   options:
     gelf-address: 'udp://localhost:12201'
```

Note: why do we use *localhost* instead of *logstash* here? The code which configures logging driver will be executed on the host machine which is not part of default network created by docker.

16. After restarting our services, open the Rails app, open Kibana, navigate to the settings tab and create a index:

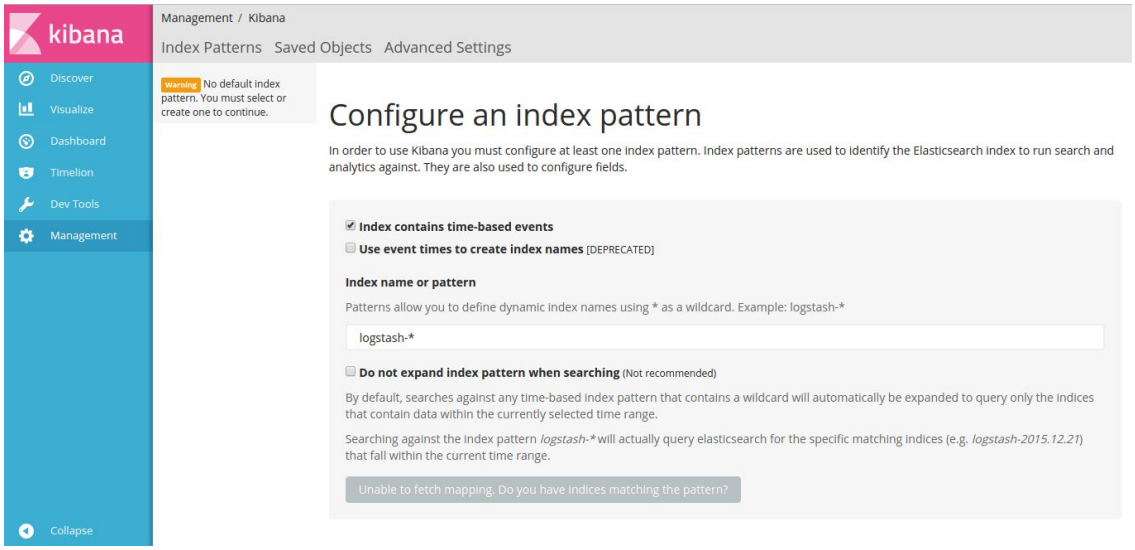

Figure 11 : Creating an index in Kibana

When index is created, we'll be able to see our first log entries in the Kibana UI. However, our current logging is not super helpful since we just sending plaintext from the Rails log:

Figure 12 : Log entry

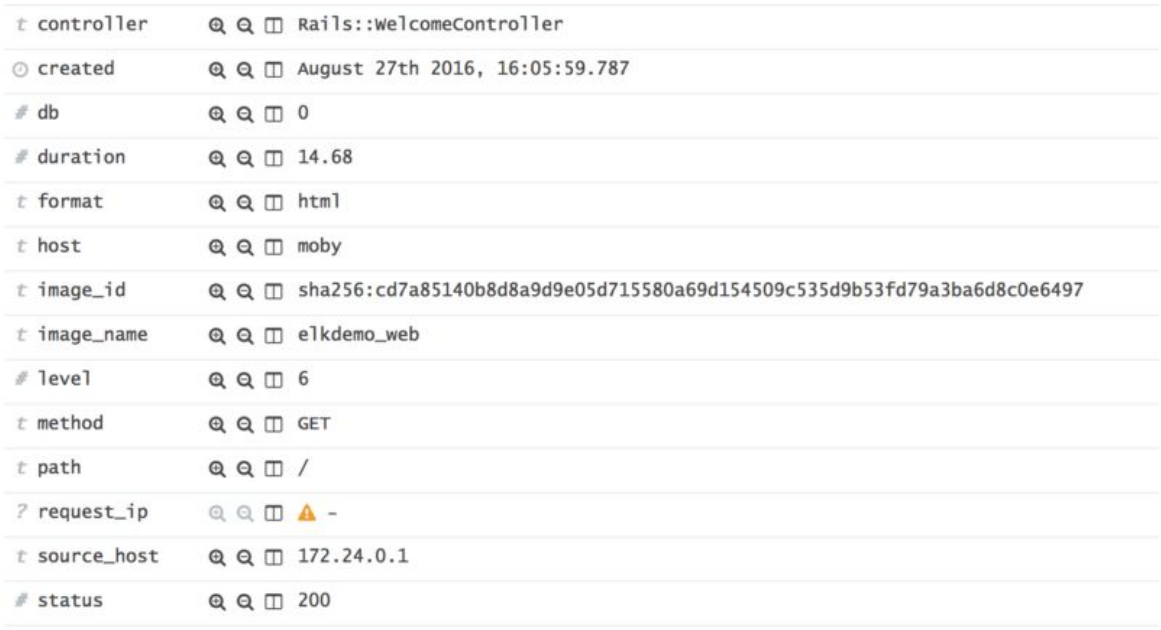

Figure 13 : Log entry

17. Instead we should present our log entries in JSON format and put as much useful data as we can. We shall use the Lograge gem, we should add two gems to our Gemfile: *lograge* and *logstash-event*. After that we should create a new initializer called *lograge.rb*, which will set up an appropriate format for our logs and send them to stdout where docker will grab them and send to the GELF driver:

```
Rails.application.configure do
 config.lograge.formatter = Lograge::Formatters::Logstash.new
 config.lograge.logger = ActiveSupport::Logger.new(STDOUT)
End
```

18. In order to turn on lograge for the environment (now we want it for the *development.rb*) — just add the following line to the corresponding environment file:
config.lograge.enabled = true

19. Adding custom fields is very simple, if we need something from the controller context — just overload the *append_info_to_payload* method in the ApplicationController and put everything you need to the payload, and after that we'll be able to add it to log. For instance, if we want to send an IP where the request came from, your *append_info_to_payload* will be:

```
class ApplicationController < ActionController::Base
  ...
  def append_info_to_payload(payload)
    super
    payload[:request_ip] = request.ip
  end
End
```

And *lograge.rb* should contain:

```
config.lograge.custom_options = lambda do |event|
 {
  request_ip: event.payload[:request_ip]
 }
end
```

20.The final step is to modify our *logstash.conf* file to parse the incoming JSON from the *message* field, just add the following code after the *input* section:

```
filter {
 json {
  source => "message"
  remove_field => "message"
 }
}
```

After rebuilding and starting your services we'll see that all the data coming from the Rails container is structured and indexed by Elasticsearch:

21. Search for all the GET requests : We should open the *Discover* tab and type a query "method:GET" to the search field. We'll see all the log entries with the GET HTTP method with the highlights:

Figure 14 : Log entry in JSON format

4. Results and Graphs

Visualization of Log Entries using Bar Graphs

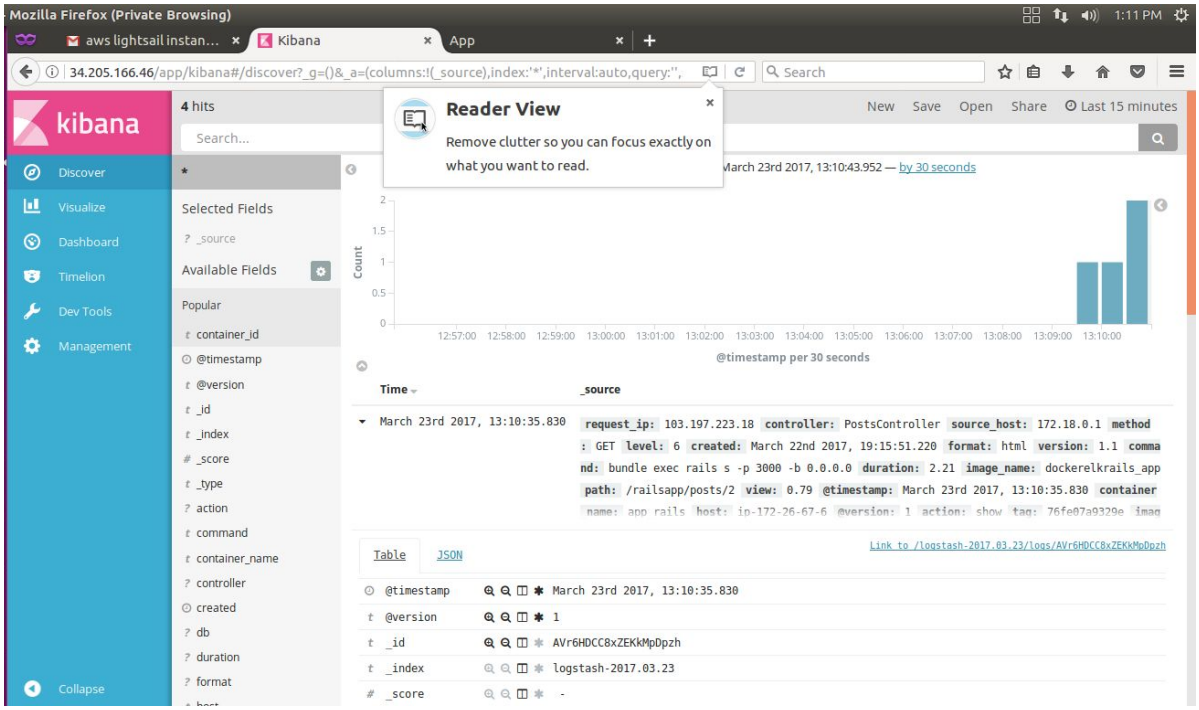

Figure 15 : Bar Graph showing number of requests per day.

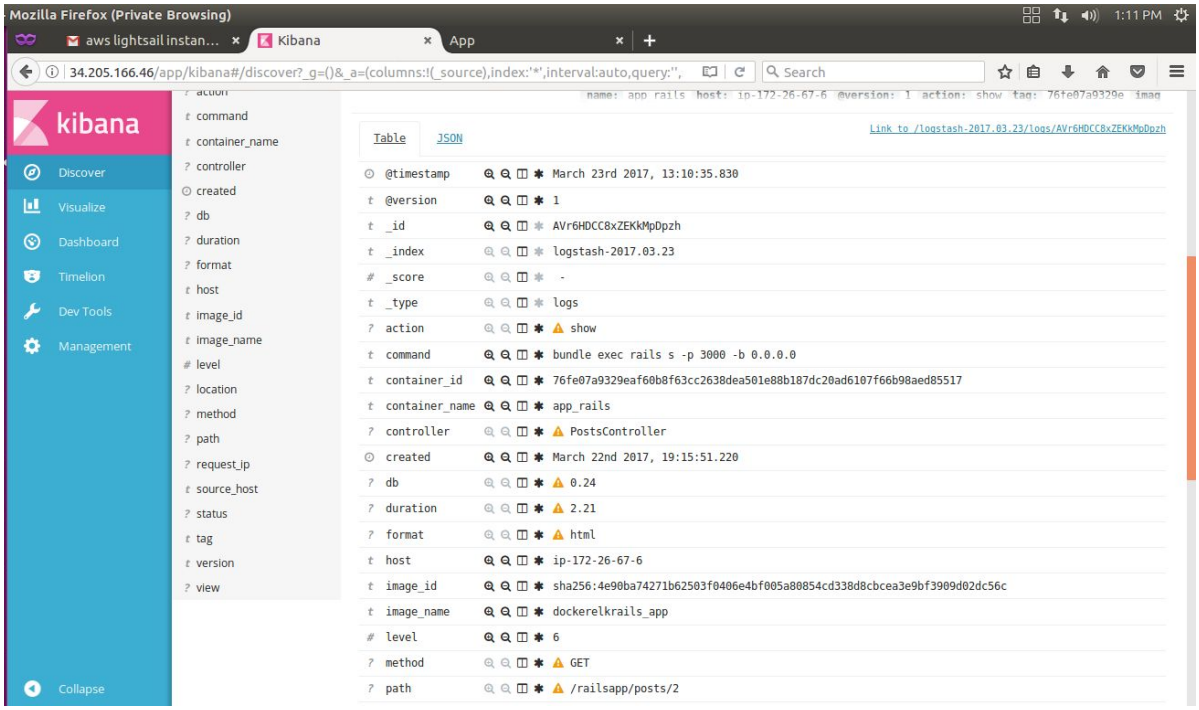

Figure 16a : Table entry of a request

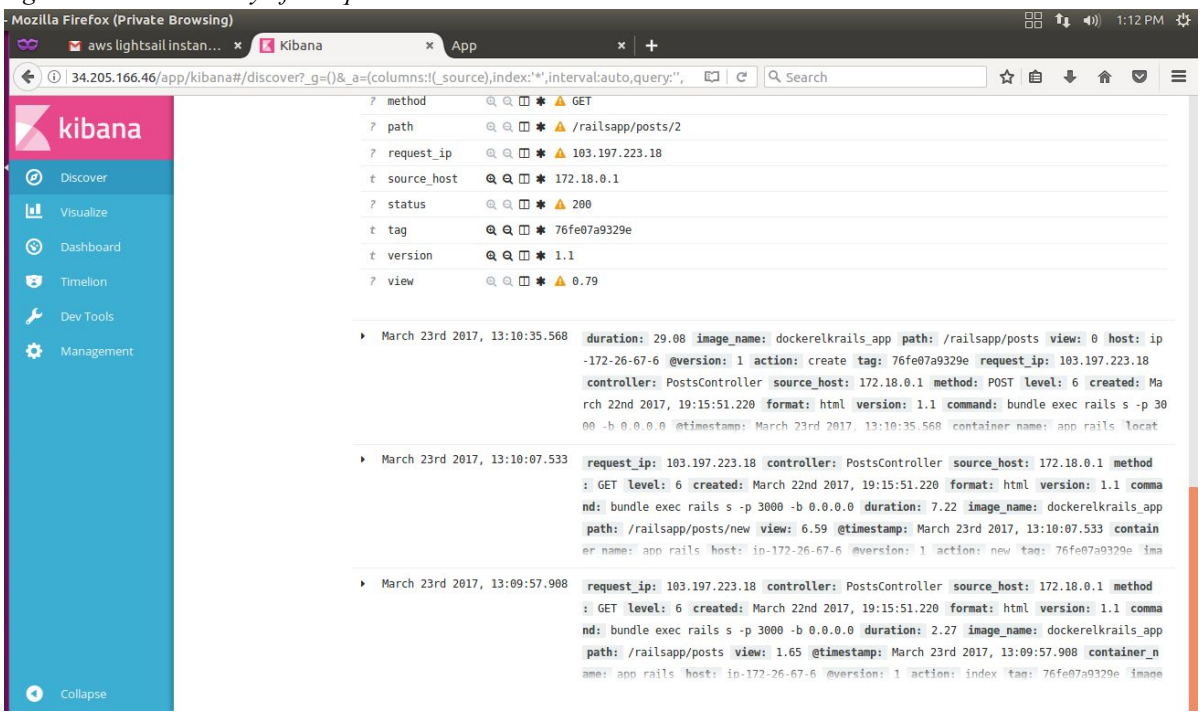

Figure 16b : Table entry of a request

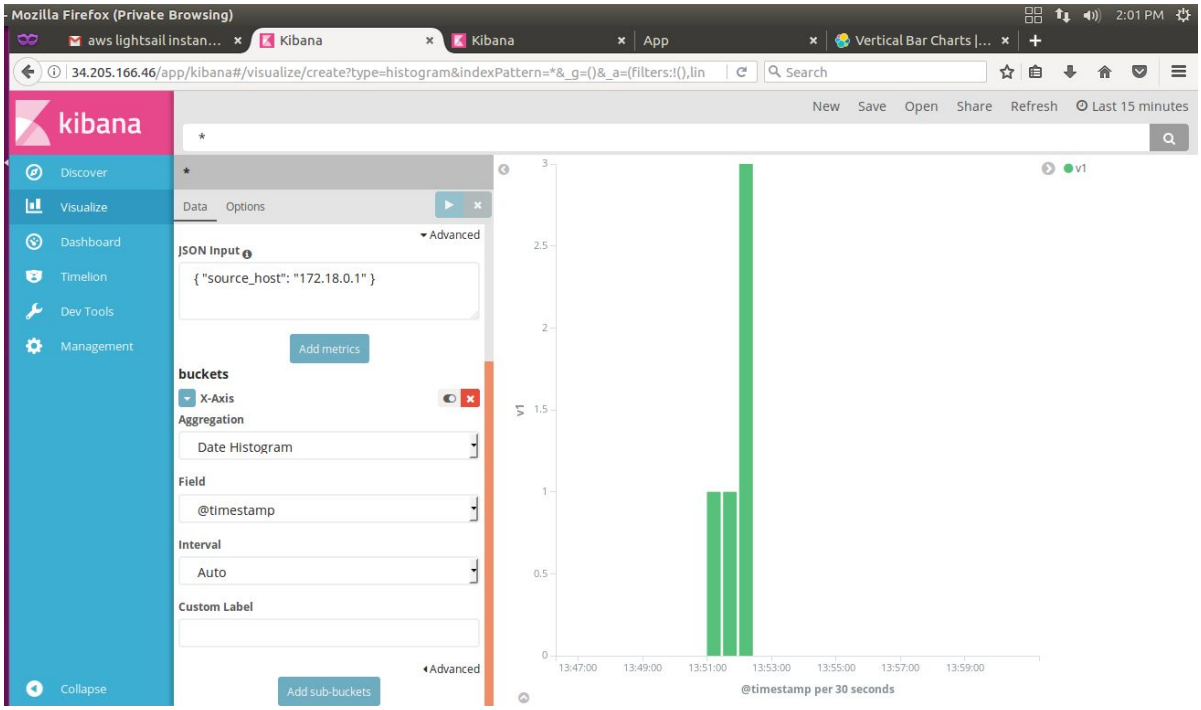

Figure 17 : Bar Graphs depicting number of requests from ip=172.18.0.1 per 30 seconds.

5. Conclusion

User can access our Rails app on the <IP>/railsapp/posts and Kibana on <IP>/. The Ruby on Rails app sends logs to Logstash via 12201/udp port (GELF input endpoint) and Kibana and Logstash talk to Elasticsearch via port 9200 (JSON REST API port). Nginx serves as a reverse-proxy server and web server and routes requests to Kibana or the Rails app based on the endpoint that you visit.

The application was successfully deployed on AWS Lightsail server with 2GB memory, 1 CPU and 40GB SSD.

[

6. References

- Docker Monitoring with the ELK Stack: A Step-by-Step Guide https://logz.io/learn/docker-monitoring-elk-stack/
- How to setup centralized logging on your Django apps using Logstash and Amazon EC2 https://csparpa.github.io/blog/2016/10/django-centralized-logging-on-ec2-with-logstash.html
- How to set up ELK for Rails log management using Docker and Docker Compose https://medium.com/@AnjLab/how-to-set-up-elk-for-rails-log-management-using-docker-and-docker-compose-a6edc290669f