



# **SPE Major Project- Workout Buddy**

14.12.2023

---

Agastya Thoppur and Anwit Damale

IMT2020528 and IMT2020532

Github- <https://github.com/imt2020532/SPEMernProject/>

## Problem Statement

Workout Buddy is an application that can be used by a wide variety of users who want to keep track of their basic fitness goals and progress. It provides users with a straightforward UI that allows them to record cardiovascular or strength training exercise sets that are stored and displayed specifically for their unique user ID.

## Tech stack used

The MERN stack (MongoDB, Express.js, React.js, Node.js) was used to build this web application. The MERN stack was employed for this project because of its efficiency, scalability, flexibility and ability to provide full stack JavaScript development. Each component of the MERN stack plays a crucial role in providing robust web applications.

1. MongoDB serves as a versatile noSQL database that can store and manage large amounts of both structured and unstructured data.
2. Express.js is a simplistic web application framework that goes hand-in-hand with Node.js. It handles server-side logic, application routing and middleware.
3. React.js is an extremely popular and powerful front-end JavaScript library that is capable of building easily usable and interactive UIs.
4. Node.js acts as the server-side runtime that fuels the backend by allowing developers to execute server-side JavaScript code.

MongoDB Atlas is a cloud database service that was initially employed for this project due to issues with the MongoDB database Docker image, but is now considered a backup/secondary database service for this web application.

## Set-up steps

The setup steps for this application are defined by the Jenkins pipeline script.

1. Pull Mongo Docker Image- The standard MongoDB database image is pulled from Mongo's public docker repository.
2. Git Clone- The newest version of the codebase stored in the master branch of the Github repository for the web application is cloned to your machine.
3. Build Backend docker image- The container derived from the "backend" image is built.
4. Build Frontend docker image- The container derived from the "frontend" image is built.
5. Push Backend image to Dockerhub- Using the developer's Dockerhub credentials, the backend image is pushed to Dockerhub.
6. Push Frontend image to Dockerhub- Using the developer's Dockerhub credentials, the frontend image is pushed to Dockerhub.
7. Clean Docker Images- Deletes the older, unnecessary docker images from your system.
8. Ansible Deployment- Creates an ansible playbook specifying the localhost as the target. This playbook copies the Docker Compose file from the host machine to the remote host, pulls all docker images specified using Docker Compose, and runs all the images as containers in detached mode.

The full pipeline script is written below.

```
pipeline {  
    environment {
```

```
    backend = 'backimage' // Specify your backend Docker image
name/tag

    frontend = 'frontimage' // Specify your frontend Docker image
name/tag

    mongolImage = 'mongo:latest' // Specify the Mongo Docker image

    mongoContainerName = 'mydatabase' // Specify the name for your
mongo container

    MONGO_PORT = '27017'

    docker_image = "
}
```

```
agent any
```

```
stages {
```

```
    stage('Stage 0: Pull Mongo Docker Image') {
```

```
        steps {
```

```
            echo 'Pulling Mongo Docker image from DockerHub'
```

```
            script {
```

```
                docker.withRegistry("", 'DockerHubCred') {
```

```
                    docker.image("${mongolImage}").pull()
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
stage('Stage 1: Git Clone') {  
  steps {  
    echo 'Cloning the Git repository'  
    git branch: 'master', url:  
'https://github.com/imt2020532/SPEMernProject.git'  
  }  
}
```

```
stage('Stage 2: Build backend Docker Image') {  
  steps {  
    echo 'Building backend Docker image'  
    dir('backend') {  
      sh "docker build -t anwit/${backend} ."  
    }  
  }  
}
```

```
stage('Stage 3: Build frontend Docker image') {  
  steps {  
    echo 'Building frontend Docker image'  
    dir('frontend') {  
      sh "docker build -t anwit/${frontend} ."  
    }  
}
```

```
}  
}
```

```
}
```

```
stage('Stage 4: Push backend Docker image to DockerHub') {
```

```
  steps {
```

```
    echo 'Pushing backend Docker image to DockerHub'
```

```
    script {
```

```
      docker.withRegistry("", 'DockerHubCred') {
```

```
        sh "docker push anwit/${backend}"
```

```
      }
```

```
    }
```

```
  }
```

```
}
```

```
stage('Stage 5: Push frontend Docker image to DockerHub') {
```

```
  steps {
```

```
    echo 'Pushing frontend Docker image to DockerHub'
```

```
    script {
```

```
      docker.withRegistry("", 'DockerHubCred') {
```

```
        sh "docker push anwit/${frontend}"
```

```
      }
```

```
    }
```

```
  }
```

```
}
```

```
stage('Stage 6: Clean docker images') {  
    steps {  
        script {  
            sh 'docker container prune -f'  
            sh 'docker image prune -f'  
        }  
    }  
}  
  
stage('Stage 7: Ansible Deployment') {  
    steps {  
        ansiblePlaybook becomeUser: null,  
        colorized: true,  
        credentialsId: 'localhost',  
        disableHostKeyChecking: true,  
        installation: 'Ansible',  
        inventory: 'deployment/inventory',  
        playbook: 'deployment/deploy.yml',  
        sudoUser: null  
    }  
}  
}
```



## Functionalities

1. Sign up/Login Authentication using JWT- Users can sign up or login to this web app by simply entering their email ID and password. The app also ensures that the user's password is longer than 8 characters. Secure authentication is carried out using JWT (JSON Web Token), which is an open standard that allows information to be securely transmitted between parties as a JSON object. Essentially, a user's confidential login information such as user ID and role is stored in a JWT and encrypted with a secret key. Everytime the user logs in, this secret key is used to verify the password and user roles. These tokens also have expiration times, where the server will deny requests with expired tokens.
2. Data persistence for each unique user- A logged in user can see his workout information on the website. Any changes/additions made by a logged in user will persist on account of the app using Docker Volumes along with a MongoDB database. Everytime a user logs in, his data is fetched from the Docker volume and stored in the database on this user's running container. The frontend fetches this data from the MongoDB database and displays it accordingly. Neither Docker Volumes nor local, containerized MongoDB databases are required if MongoDB Atlas can be used. Since it is a cloud database service that runs 24/7, it would also be a good alternative to using Docker volumes and local containerized databases.
3. Easy and swift data recording for each unique user- Once logged in, a user can enter his exercise information. This information will be made visible instantly on the home screen, and can be easily



edited or deleted. Another feature is that the user can navigate to the “Cardio” Screen, which records a cardio exercise with certain parameters and outputs the number of calories the user has burnt in that session.

## Application flow and UI

### Workout Buddy

[Login](#) [Signup](#)

The screenshot shows the 'Sign Up' form in the 'Workout Buddy' application. The form is centered on a light gray background. It has a title 'Sign Up' in bold. Below the title, there are two input fields: 'Email address:' with the value 'anwit.damale@iiitb.ac.in' and 'Password:' with a masked value '\*\*\*\*\*'. A green 'Sign up' button is located below the password field. The entire form is enclosed in a white box with a thin gray border.

### Workout Buddy

[Login](#) [Signup](#)

This screenshot shows the 'Sign Up' form after an attempt to register with a weak password. The form structure is identical to the previous one, but it includes an additional error message. Below the 'Sign up' button, there is a pink rectangular box with the text 'Password not strong enough' in a pink font. The 'Email address' field still contains 'anwit.damale@iiitb.ac.in' and the 'Password' field is masked with '\*\*\*\*\*'.

## Workout Buddy

[Login](#) [Signup](#)

### Log In

Email address:

Password:

Workout Buddy

anwit.damale@iiitb.ac.in [Log out](#)

Deadlift

Load (kg): 30

Reps: 8

less than a minute ago

Delete

Edit

Benchpress

Load (kg): 20

Reps: 10

less than a minute ago

Delete

Edit

Situps

Load (kg): 5

Reps: 10

less than a minute ago

Delete

Edit

Pushups

Load (kg): 5

Reps: 15

1 minute ago

Delete

Edit

Add a New Workout

Exercise Title:

Load (in kg):

Reps:

Add Workout

[Go to Cardio Page](#)

Workout Buddy

anwit.damale@iiitb.ac.in [Log out](#)

BURNING...

Weight (kg):

Duration (minutes):

Activity Type:

Jog

Calculate Calories

You burned approximately 0.76 calories!

## DevOps and CI/CD practices employed

1. Version Control and Source Code Management- A version control system was adopted to allow for collaboration, coordination between team members and maintenance of source code version

history. The Version Control System used for this project was Git/Github.

2. Unit Testing- JUnit Unit Tests were constructed specifically to testout the core functionalities of the application, allowing basic logical errors to be detected swiftly after every build.
3. Continuous Integration/Continuous Deployment- Implementing CI/CD principles using tools such as Jenkins can ensure reduced monotonic developer workload and can accelerate building, testing and delivery.
4. Automated Build and Deployment- This streamlines the process of tedious processes such as compiling code, running tests and deploying web applications. Ansible is an open-source tool that is used to manage software projects across multiple servers by defining tasks in ansible playbooks.
5. Containerization- Docker and Dockerhub are used to encapsulate the three main components of this web application (frontend code, backend code and database). These components (along with their dependencies) are wrapped into lightweight, runnable containers. Since the application requires multiple containers to be usable, a container orchestration tool is also required. In this case, Docker compose is used.
6. Monitoring and Logging- ELK stack is used to monitor application behavior in real-time and capture different events and errors that may occur during the usage of the application.

## Version Control and SCM with Github

Version Control/SCM tools allow developers to keep track of previous codebase versions while working on newer versions/separate branches of the source code. For this project, this is accomplished using Git/Github. The following steps were followed to maintain a central

repository with the main project codebase, also ensuring that both teammates can work on their own workspaces using local repositories.

1. Created a public repository on Github (link available on first page).
2. `$ git init`
3. `$ git add .`
4. `$ git commit -m "First commit"`
5. `$ git push origin master`

The remote workspace as shown on github looks like this-

The screenshot shows the GitHub repository page for 'SPEMernProject'. The repository is public and has 1 watch, 0 forks, and 0 stars. The main branch is 'master'. The repository contains 15 commits. The file list shows the following structure:

File	Author	Time
backend	omg	3 hours ago
data/db	Initial Commit 2	18 hours ago
deployment	Initial Commit 2	17 hours ago
frontend	please!	47 minutes ago
node_modules	omg	5 hours ago
docker-compose.yml	fixes	15 hours ago
package-lock.json	omg	5 hours ago
package.json	omg	5 hours ago

The right sidebar shows the repository's activity, including 0 stars, 1 watching, and 0 forks. It also lists the repository's releases, packages, and languages. The languages section shows the following distribution:

Language	Percentage
JavaScript	88.1%
CSS	5.2%
HTML	4.4%
Dockerfile	2.3%

## CI/CD with Jenkins

Any update to the codebase, no matter how small, has to undergo a series of complicated procedures before being ready for deployment. These procedures (building, different rounds of testing) can be automated and triggered instantly. A Jenkins pipeline carries out a series of these steps that must be carried out to integrate/deploy an application. These steps are defined by a pipeline script, which is

triggered either manually by a developer or through Webhooks that are attached to the version control system (Github).

A screenshot of the Jenkins build stages is provided below-

#### Stage View

	Stage 0: Pull Mongo Docker Image	Stage 1: Git Clone	Stage 2: Build backend Docker Image	Stage 3: Build frontend Docker image	Stage 4: Push backend Docker image to DockerHub	Stage 5: Push frontend Docker image to DockerHub	Stage 6: Clean docker images	Stage 7: Ansible Deployment
Average stage times: (Average full run time: ~5min 1s)	2min 42s	4s	16s	29s	14s	19s	2s	41s
#27 Dec 14 20:14 1 commit	1min 46s	7s	17s	9s	10s	23s	1s	42s
#26 Dec 14 18:26 1 commit	3min 50s	3s	11s	9s	11s	19s	1s	40s
#25 Dec 14 17:43 No Changes	2min 1s	2s	12s	6s	11s	9s	1s	28s

## Building/Packaging using the MERN stack

The MERN stack is universally renowned for its ability to build modern, full-stack web applications. The Node Package Manager (npm) is used along with any Node.js environment. It is used to facilitate the compilation, installation, execution and overall dependency management of Node.js projects.

The node server.js command ensures that the backend is ready and can execute the necessary specified JavaScript code.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
○ anwit@chimoo:~/Downloads/MERN-Auth-Tutorial-lesson-17/backend$ node server.js
connected to db & listening on port!!

```

Once the server is ready and the compilation is successful, npm start must be executed to get the server into production mode.

```
Compiled successfully!

You can now view frontend in the browser.

Local:      http://localhost:3000
On Your Network: http://172.16.134.32:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

```
anwit@chimooo:~/Downloads/MERN-Auth-Tutorial-lesson-17/frontend$ npm start
```

## Containerization with Docker and Docker compose

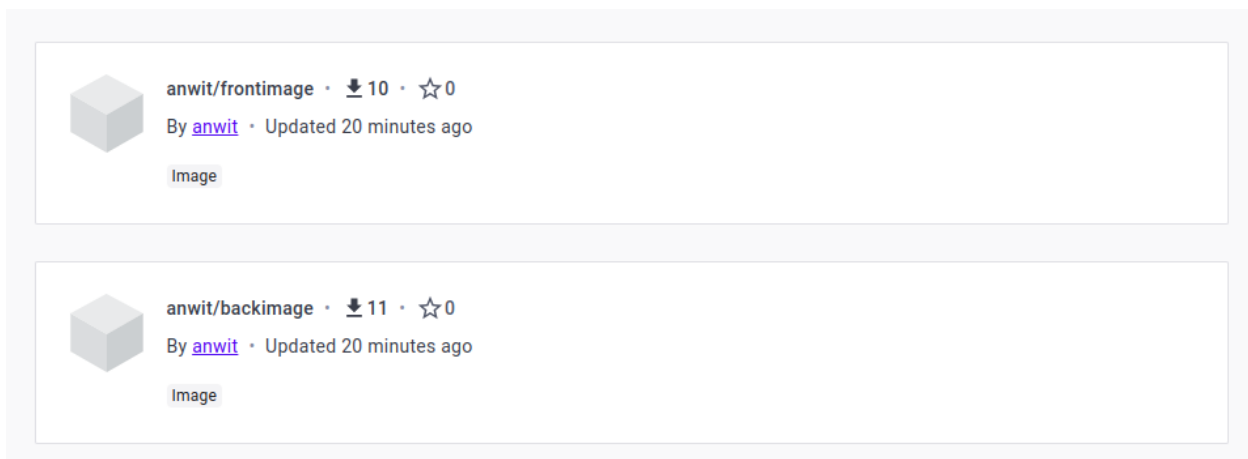
As mentioned above, the frontend code, backend code, and the MongoDB database are stored in three separate containers. Docker enables applications (or components of an application) along with their dependencies to be encapsulated and run in separate, lightweight containers. These containers can be used across different environments since they contain all the dependencies and libraries needed to run an application. A lot of modern applications are multi-container applications, which is where Docker compose comes into the picture. Docker compose uses a YAML file to define the services, volumes and other resources required for an application. This YAML file also facilitates the orchestration of multiple containers, allowing developers and users alike to run applications with just one command. The below image illustrates the different containers and images used to build the multi-container web application.

```

anwit@chinooo:~$ sudo docker container ls
[sudo] password for anwit:
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
56dc7585cafa   anwit/frontimage "docker-entrypoint.s..." 2 hours ago    Up 2 hours    0.0.0.0:3000->3000/tcp, :::3000->3000/tcp  deployme
nt_frontend_1  2a5ea0cae16c   anwit/backimage "docker-entrypoint.s..." 2 hours ago    Up 2 hours    0.0.0.0:4000->4000/tcp, :::4000->4000/tcp  deployme
nt_backend_1   0ae64cc27138   mongo                  "docker-entrypoint.s..." 2 hours ago    Up 2 hours    0.0.0.0:27017->27017/tcp, :::27017->27017/tcp  deployme
nt_mydatabase_1
anwit@chinooo:~$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
anwit/frontimage   latest    43ec0da18458   2 hours ago    311MB
anwit/backimage    latest    873a24bfb4f1   7 hours ago    143MB
anwit/calculator2023 latest    6b89f531154f   21 hours ago   656MB
custom-jenkins-image latest    8ff6cc68ebd5   21 hours ago   487MB
mongo           latest    5acb2131d51f   12 days ago    757MB
hello-world      latest    9c7a54a9a43c   7 months ago    13.3kB
anwit@chinooo:~$

```

The frontend and backend images can be pulled from Dockerhub. The MongoDB image is pulled from MongoDB's public Dockerhub.



Screenshot of the Docker Compose YML file-

```

Code Blame 35 lines (31 loc) · 616 Bytes Code 55% faster with GitHub Copilot
1  version: '3.5'
2  services:
3
4    backend:
5      image: anwit/backimage
6      ports:
7        - "4000:4000"
8      volumes:
9        - ./backend/logs:/app/logs # Map the log directory in the container to the ./backend/logs directory on the host
10     depends_on:
11       - mydatabase
12     networks:
13       - my-network
14
15    frontend:
16      image: anwit/frontimage
17      ports:
18        - "3000:3000"
19      depends_on:
20        - backend
21      networks:
22        - my-network
23
24    mydatabase:
25      image: mongo
26      ports:
27        - "27017:27017"
28      volumes:
29        - ./data/db:/data/db:rw
30      networks:
31        - my-network
32
33    networks:
34      my-network:
35        driver: bridge

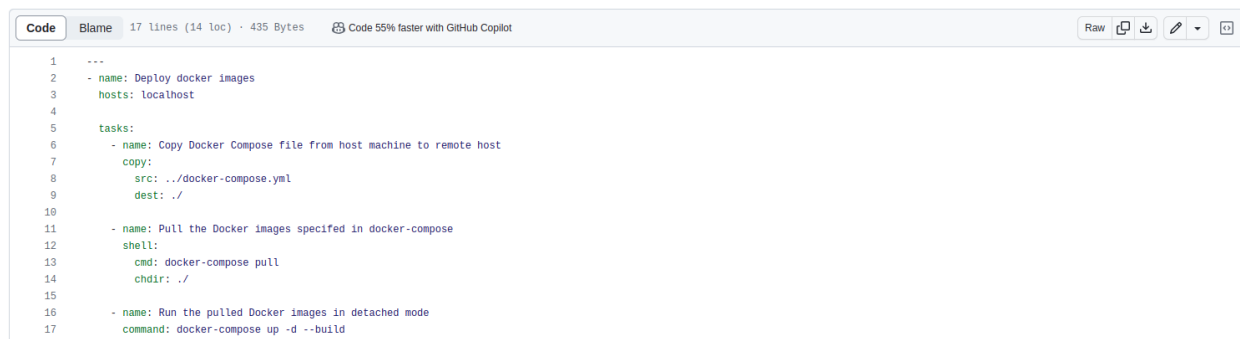
```



## Deployment with Ansible

Ansible is an automation tool that is used for configuration management and automated application deployment. It carries out tasks that are specified in files known as ansible playbooks. This format of task automation is human-readable and easy to use by both developers and system administrators.

Screenshot of Ansible Deployment Playbook-

A screenshot of a code editor showing an Ansible playbook. The editor has a light blue header with tabs for 'Code', 'Blame', and '17 lines (14 loc) · 435 Bytes'. There is also a GitHub Copilot logo and a 'Code 55% faster with GitHub Copilot' message. On the right side of the header, there are icons for 'Raw', 'Download', 'Edit', and 'Copy'. The code itself is a YAML file with the following content:

```
1 ---
2 - name: Deploy docker images
3   hosts: localhost
4
5   tasks:
6     - name: Copy Docker Compose file from host machine to remote host
7       copy:
8         src: ../docker-compose.yml
9         dest: ../
10
11     - name: Pull the Docker images specified in docker-compose
12       shell:
13         cmd: docker-compose pull
14         chdir: ../
15
16     - name: Run the pulled Docker images in detached mode
17       command: docker-compose up -d --build
```

## Monitoring, Logging and Visualization using ELK stack

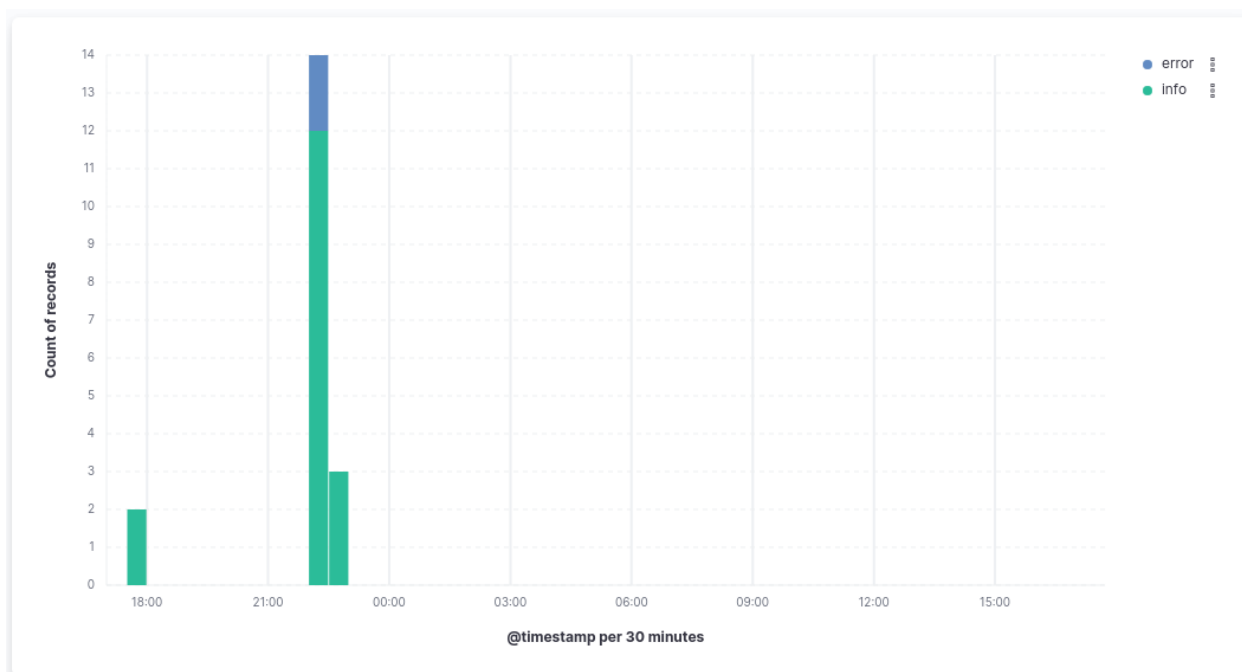
ELK stack is an incredibly useful log file analysis and visualization tool that is widely used in the monitoring and troubleshooting of applications that have been deployed across various systems. It has three main components, Elasticsearch, Logstash and Kibana.

1. Elasticsearch is an analytics engine that is used to store and index large volumes of log data in documents (similar to an RDBMS).
2. Logstash is a data processing pipeline that makes use of grok filters to input, filter and output log data to Elasticsearch.

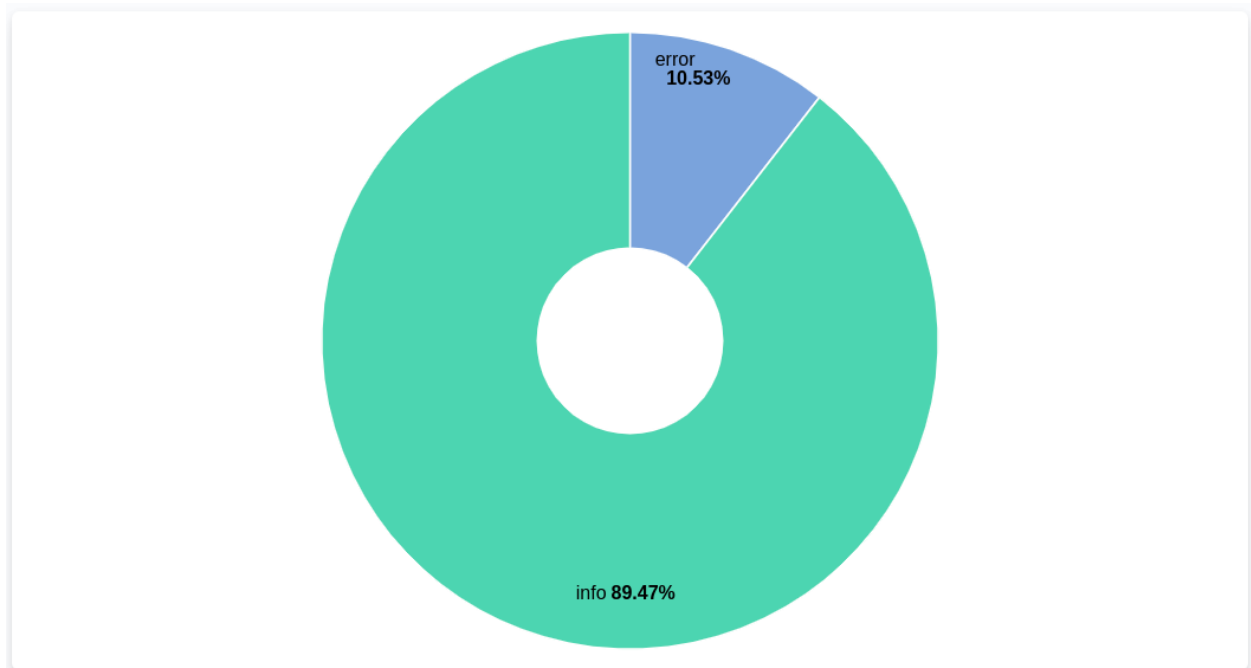
3. Kibana is a visualization tool that works hand-in-hand with ElasticSearch. In fact, it can be thought of as a dashboard for ElasticSearch.

The three of these open-source tools enable developers to centralize logs from different sources of the application base and analyze them in real-time by creating meaningful visualizations.

The bar graph shows how many log entries were created per 30 minutes-



The pie chart below indicates the percentages of infos vs errors logged-



## Testing using chai and mocha

- First step is to install chai and mocha.
- `npm install mocha`
- `npm install chai-http`
- Then adding the appropriate command "`mocha ./test/myBackendTest.js`" in the script inside the package.json of the backend.
- Create a folder called test inside the backend folder and create a file `myBackendTest.js`.
- And as you can see in the backend one json object is returned passing the 1st test case and failing the next because of not fulfilling the requirements of the incoming object.

```

JS server.js    {} package.json frontend    JS myBackendTest.js    {} package.json backend X
backend > {} package.json > {} dependencies
  1  {
  2    "name": "backend",
  3    "version": "1.0.0",
  4    "description": "",
  5    "main": "server.js",
  6    "scripts": {
  7      "test": "mocha ./test/myBackendTest.js",
  8      "start": "node server.js",
  9      "dev": "node server.js"
 10  },

```

```

anwit@chimooo:~/Downloads/MERN-Auth-Tutorial-lesson-17/backend$ node server.js
connected to db & listening on port!!
/api/workouts POST

```

```

anwit@chimooo:~/Downloads/MERN-Auth-Tutorial-lesson-17/backend$ npm test
> backend@1.0.0 test
> mocha ./test/myBackendTest.js

Backend Tests
  ✓ should return a JSON object

1 passing (32ms)

/home/anwit/Downloads/MERN-Auth-Tutorial-lesson-17/backend/node_modules/mocha/lib/runner.js:950
    throw err;
    ^
AssertionError: expected { Object (_events, _eventsCount, ...) } to have status code 200 but got 401
    at /home/anwit/Downloads/MERN-Auth-Tutorial-lesson-17/backend/test/myBackendTest.js:22:29
    at Test.Request.callback (/home/anwit/Downloads/MERN-Auth-Tutorial-lesson-17/backend/node_modules/supera
gent/lib/node/index.js:857:12)
    at /home/anwit/Downloads/MERN-Auth-Tutorial-lesson-17/backend/node_modules/superagent/lib/node/index.js:
1070:18
    at IncomingMessage.<anonymous> (/home/anwit/Downloads/MERN-Auth-Tutorial-lesson-17/backend/node_modules/
superagent/lib/node/parsers/json.js:21:7)
    at IncomingMessage.emit (node:events:525:35)
    at endReadableNT (node:internal/streams/readable:1358:12)
    at processTicksAndRejections (node:internal/process/task_queues:83:21) {
  showDiff: true,
  actual: 401,
  expected: 200
}
anwit@chimooo:~/Downloads/MERN-Auth-Tutorial-lesson-17/backend$

```