

Final Project Report - EMS

Bhavil (IMT2021041) | Varshith (IMT2021078)

Project Description

EMS - Employee Management System is a platform made by using the MERN stack. Here, we can add, update or delete employee data.

The architecture of our projects requires three layers:

- Frontend
- Backend
- Database

We have described the technology stack and functionalities of our application in more detail later in this report.

GitHub Repository Link:

https://github.com/Bhavil-13/Employee_Management_System_SPEFinalProject

What is DevOps?

DevOps is a collection of practices and principles designed to enhance communication and collaboration between development and operations teams, ultimately improving the speed and quality of software delivery. It focuses on automating workflows, adopting agile methodologies, and utilizing tools that support continuous integration, delivery, and deployment. By breaking down silos and fostering teamwork, DevOps enables organizations to deliver software faster, with fewer errors, and more efficiently.

Key DevOps practices and principles include:

1. **Continuous Integration (CI):** Developers frequently integrate code into a shared repository, triggering automated build and test processes.
2. **Continuous Delivery (CD):** Code changes are automatically built, tested, and deployed to production environments.
3. **Infrastructure as Code (IaC):** Infrastructure is managed through code stored in version control, enabling versioning, testing, and automated deployment.
4. **Monitoring and Logging:** Real-time monitoring and logging help teams quickly detect and resolve issues.
5. **Collaboration and Communication:** DevOps prioritizes effective collaboration and communication among developers, operations teams, and other stakeholders.

Why DevOps?

DevOps is important because it helps organizations to deliver software quickly, with fewer errors, and with greater efficiency. Here are a few points of importance of DevOps:

- **Faster time-to-market:** DevOps enables organizations to release software more quickly, allowing them to respond to changing market conditions and customer needs more rapidly.
- **Improved quality:** By automating processes and using agile methodologies, DevOps helps to reduce the number of errors in software releases, resulting in higher quality software.
- **Increased collaboration:** DevOps emphasizes collaboration and communication between development and operations teams, which helps to break down silos and improve the overall efficiency of the organization.
- **Scalability:** DevOps practices enable organizations to scale their software delivery processes to meet the needs of their users, without sacrificing quality or speed.
- **Enhanced security:** By incorporating security into the development process, DevOps helps to reduce the risk of security vulnerabilities and data breaches.

Technology Stack used for Project Pipeline:

Purpose	Tools	Description
Source Control Management	Git, GitHub	Version control tools that facilitate collaborative software development by tracking changes and managing code repositories.
Development	MERN Stack (MongoDB, Express, ReactJS, NodeJS)	A comprehensive full-stack web development framework comprising MongoDB for the database, Express for backend framework, ReactJS for the frontend library, and NodeJS as the runtime environment for backend development.
Containerization	Docker, Docker Hub	A platform for building, shipping, and running software applications within lightweight, portable containers.
Deployment (Localhost)	Kubernetes	An open-source tool that automates the provisioning, configuration, and deployment of applications in local or distributed environments.
Monitoring	ELK Stack	A suite of tools (Elasticsearch, Logstash, Kibana) designed for real-time monitoring, logging, and visualization of data from diverse sources to identify and address system issues efficiently.
Build Trigger	GitHub Webhook, Ngrok	GitHub webhooks trigger automated build and test processes upon code changes, while Ngrok provides a public URL for exposing local servers to the internet.
Continuous Integration	Jenkins	A powerful automation server used to streamline continuous integration and delivery pipelines by automating testing and builds.

Jenkins Pipeline

Building the CI/CD pipeline using Jenkins

Install Java (required for jenkins to run)

```
sudo install java
```

```
java --version
```

```
bhavi1@bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA:~$ java --version
openjdk 17.0.13 2024-10-15
OpenJDK Runtime Environment (build 17.0.13+11-Ubuntu-2ubuntu124.04)
OpenJDK 64-Bit Server VM (build 17.0.13+11-Ubuntu-2ubuntu124.04, mixed mode, sharing)
```

Now, install jenkins, and then in your command line, type:

```
sudo systemctl start jenkins
```

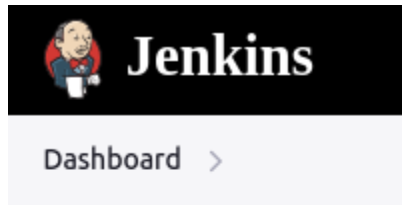
Then, to check if it's running, you can do:

```
sudo systemctl status jenkins
```

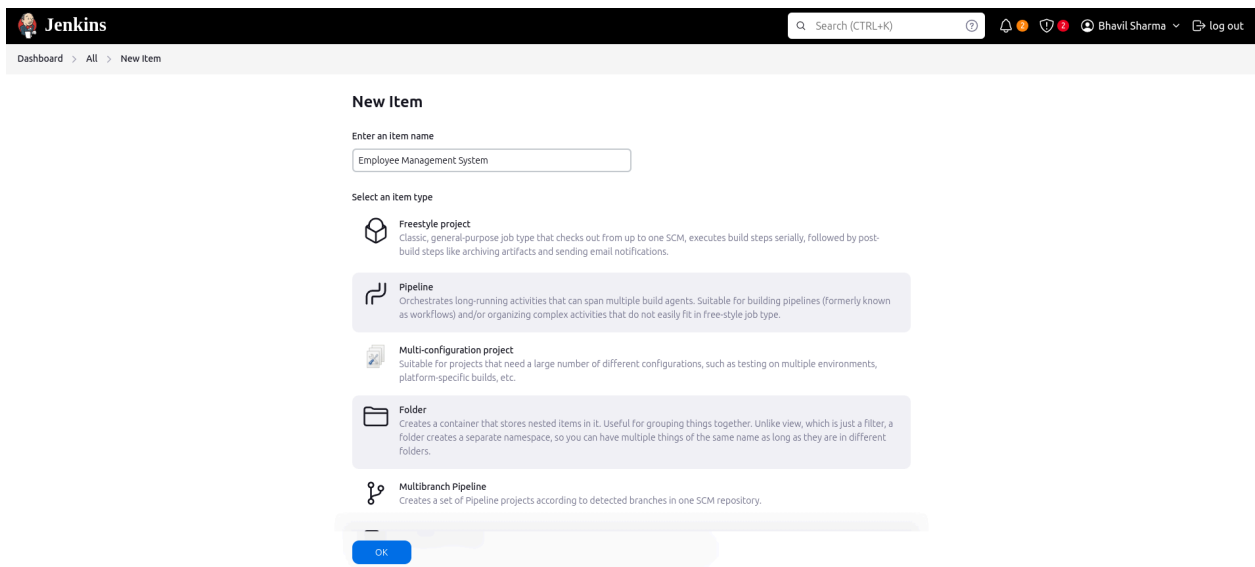
```
bhavi1@bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA:~$ sudo systemctl start jenkins
[sudo] password for bhavi1:
bhavi1@bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
   Active: active (running) since Tue 2024-12-10 21:59:03 IST; 1h 1min ago
     Main PID: 1378 (java)
       Tasks: 56 (limit: 9880)
      Memory: 359.2M (peak: 541.0M swap: 91.6M swap peak: 95.4M)
         CPU: 43.276s
        CGroup: /system.slice/jenkins.service
                └─1378 /usr/lib/jvm/java-17-openjdk-and64/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Dec 10 21:59:02 bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA jenkins[1378]: 2024-12-10 16:29:02.099+0000 [id=40] INFO jenkins.InitReactorRunner$1#onAttained: Started all plugins
Dec 10 21:59:02 bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA jenkins[1378]: 2024-12-10 16:29:02.111+0000 [id=44] INFO jenkins.InitReactorRunner$1#onAttained: Augmented all extensions
Dec 10 21:59:02 bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA jenkins[1378]: 2024-12-10 16:29:02.608+0000 [id=42] INFO h.p.b.g.GlobalTimeoutConfiguration#load: global timeout not set
Dec 10 21:59:03 bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA jenkins[1378]: 2024-12-10 16:29:03.479+0000 [id=37] INFO jenkins.InitReactorRunner$1#onAttained: System config loaded
Dec 10 21:59:03 bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA jenkins[1378]: 2024-12-10 16:29:03.484+0000 [id=39] INFO jenkins.InitReactorRunner$1#onAttained: System config adapted
Dec 10 21:59:03 bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA jenkins[1378]: 2024-12-10 16:29:03.533+0000 [id=49] INFO jenkins.InitReactorRunner$1#onAttained: Loaded all jobs
Dec 10 21:59:03 bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA jenkins[1378]: 2024-12-10 16:29:03.549+0000 [id=49] INFO jenkins.InitReactorRunner$1#onAttained: Configuration for all jobs
Dec 10 21:59:03 bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA jenkins[1378]: 2024-12-10 16:29:03.584+0000 [id=38] INFO jenkins.InitReactorRunner$1#onAttained: Completed initialization
Dec 10 21:59:03 bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA jenkins[1378]: 2024-12-10 16:29:03.629+0000 [id=26] INFO hudson.lifecycle.Lifecycle$onReady: Jenkins is fully up and ready
Dec 10 21:59:03 bhavi1-VivoBook-ASUSLaptop-X515EA-X515EA systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
lines 1-20/20 (END)
```

This will start jenkins on localhost:8080. Login to it, and create a new pipeline in jenkins:



Go to New Item, name your project and select pipeline option:



Add a description, and mention that its a github project by clicking the github project checkbox. Select the build trigger and add the repo link:

Dashboard > Employee Management System > Configuration

Configure

- General
- Advanced Project Options
- Pipeline

☒ GitHub hook trigger for GITScm polling ?

☐ Poll SCM ?

☐ Quiet period ?

☐ Trigger builds remotely (e.g., from scripts) ?

Advanced Project Options

Advanced

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

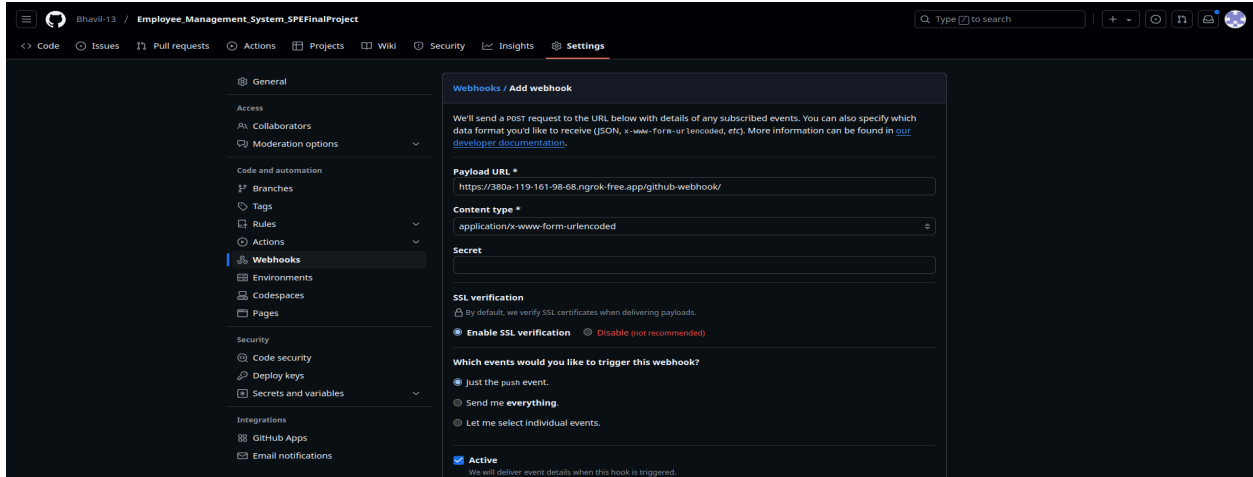
https://github.com/Bhavil-13/Employee_Management_System_SPEFinalProject.git

Save Apply

Provide the GitHub repository URL that Jenkins will use to clone the repository upon a build trigger. Specify the branches you want to build, and include the path to the pipeline script in the repository, which in this case is the **Jenkinsfile** located at the root of the project. Once you've entered this information, click **Save**.

Setting up build trigger - Github Webhook and Ngrok

- **GitHub Webhook:** GitHub provides a webhook feature that can trigger a Jenkins build upon every code push to the repository. This webhook sends a POST request to the Jenkins server, initiating the build. To set up a webhook, follow these steps:
 1. Navigate to your repository on GitHub and click the **Settings** tab.
 2. Select **Webhooks**, then click **Add webhook**.
 3. Enter the **Payload URL** for your Jenkins server. This URL should include your Jenkins server address followed by the path to your webhook URL, for example: **http://your-jenkins-server/github-webhook/**.
 4. Choose the events that will trigger the webhook. To trigger the webhook on every code push, select **Just the push event**.
 5. Check the **Active** checkbox and click **Add webhook**.



Ngrok: Ngrok is a tool that enables you to create a secure tunnel from a public endpoint to a locally running web service. We'll use Ngrok to create a public endpoint for our Jenkins server so GitHub can send webhook requests to it. To set up Ngrok, follow these steps:

1. Download and install Ngrok from the [official website](#).
2. Start Ngrok by running the following command in your terminal: `ngrok http 8080`. This command creates a secure tunnel from a public endpoint to your locally running Jenkins server on port 8080.
3. Copy the **Forwarding** URL from the Ngrok console. This URL is the public endpoint you'll use as the Payload URL in the GitHub webhook configuration.

```
ngrok
🚧 Route traffic by anything: https://ngrok.com/r/iep

Session Status      online
Account             Bhavil-13 (Plan: Free)
Version             3.18.4
Region              India (in)
Web Interface        http://127.0.0.1:4040
Forwarding           https://4fb1-119-161-98-68.ngrok-free.app -> http://localhost:8080

Connections         ttl    opn    rt1    rt5    p50    p90
                   0      0      0.00   0.00   0.00   0.00
```

After setting up the GitHub webhook and Ngrok, you can test the webhook by pushing some code changes to your repository. If everything is configured correctly, you should see a new build trigger appear in your Jenkins dashboard.

Creating jenkins file

To create a pipeline script for Jenkins, you need to create a file called **Jenkinsfile** in the root of your project repository. This file will contain the steps that Jenkins follows to build, test, and deploy your project.

The **Jenkinsfile** is written in Groovy, a Java-based scripting language designed for the Java Virtual Machine (JVM). You will use Groovy to define the steps that Jenkins will execute in your pipeline.

You can customize the **Jenkinsfile** to include any steps necessary for your project. Jenkins also offers numerous plugins and integrations that can help automate your build, test, and deployment processes.

Once you've created the **Jenkinsfile**, configure your Jenkins pipeline to use it by specifying the path to the file in the Jenkins project configuration. Jenkins will then automatically execute your pipeline whenever it is triggered, such as by a code push to your repository.

The screenshot displays the Jenkins web interface for a specific build. The top navigation bar includes the Jenkins logo, a search bar, and user information. The main content area shows the details for 'Build #9 (22 Nov 2024, 10:50:48)'. On the left, a sidebar lists various options like Status, Changes, Console Output, and Pipeline Overview. The main panel shows the build's status as successful, the user who started it (Bhavil Sharma), and the time it took to complete (6.9 seconds). It also displays the Git revision and repository information.

Jenkins Search (CTRL+K) Bhavil Sharma log out

Dashboard > SPE_FinalProject > #9

Build #9 (22 Nov 2024, 10:50:48) Keep this build forever

Started 18 days ago Took 6.9 sec

Add description

Changes

1. edit in addemployeecomponent.js (details / githubweb)

Started by user Bhavil Sharma

This run spent:

- 5 ms waiting;
- 6.9 sec build duration;
- 6.9 sec total from scheduled to completion.

Revision: 696e4106be8bfb0971e1134ff2917123a506704
Repository: <https://github.com/Bhavil-13/spefinalproj.git>

- refs/remotes/origin/main

REST API Jenkins 2.462.2

Build

To build the application, you need to install the required packages listed in the `package.json` files located in the `server` and `client` directories.

Once the packages are installed, you can build the frontend React app by running the `npm run build` command. This will create an optimized version of the React app that is ready for deployment.

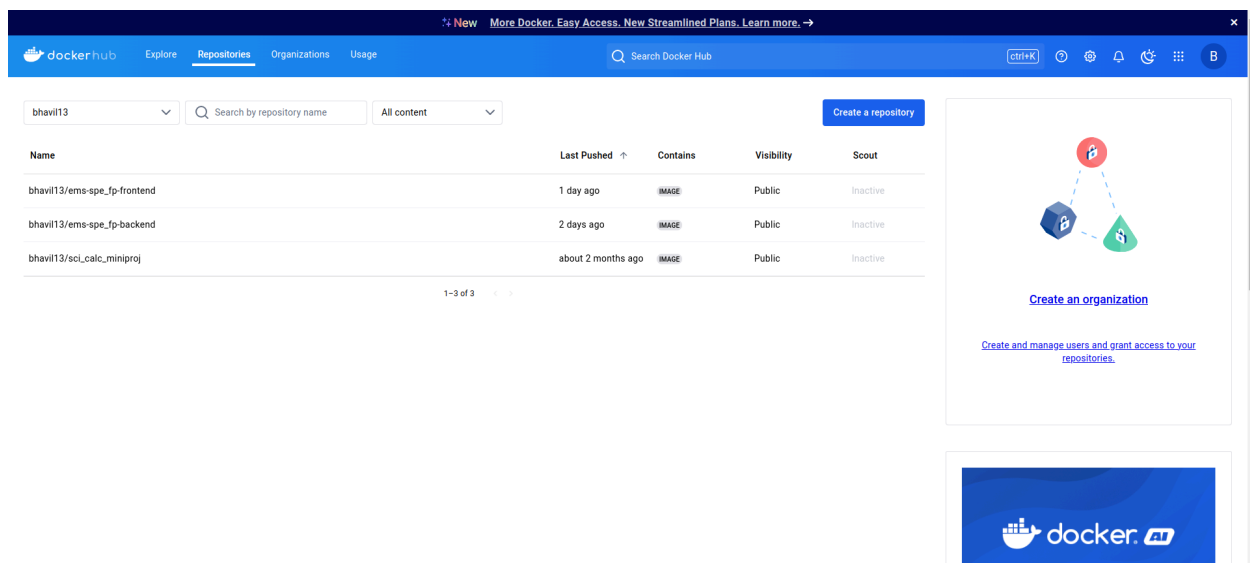
Containerization

1. Installing Docker on Linux

To start with Docker, you first need to install it on your Linux system. You can download the Docker DMG from [Docker's official website](#). After installation, the Docker daemon will start, and you can begin using Docker commands.

2. Dockerizing the MERN App

To containerize your MERN application, you need to create two Dockerfiles: one for the frontend and one for the backend. These Dockerfiles define the steps to build the images for both parts of the app.



Dockerfile for Frontend

The Dockerfile for the frontend will create a container that hosts the compiled frontend code using the nginx web server. The container will also expose port 5173 to allow external access to the served content.

```
frontend > Dockerfile > ...
1 FROM node:18.9.1
2 WORKDIR /app
3 COPY package.json .
4 RUN npm install
5 EXPOSE 5173
6 COPY . .
7 CMD ["npm", "run", "dev"]
```

- **Steps in the Dockerfile:**

- The **EXPOSE** command makes the container listen on port 5173, which allows external processes to communicate with the container over this port.
- The **RUN** command removes the default nginx configuration file, so you can replace it with a custom configuration during the build process.
- The base image for the frontend is specified as **node:18.9.1**, which includes Node.js version 18.9.1 for building the React app.

Dockerfile for Backend

The Dockerfile for the backend defines the steps to set up a Node.js environment for the backend service.

```
backend > Dockerfile > ...
1 FROM node:18.9.1
2 WORKDIR /app
3 COPY package.json .
4 RUN npm install
5 COPY . .
6 EXPOSE 5050
7 CMD ["npm", "start"]
```

- **Steps in the Dockerfile:**

- The base image for the backend is specified as `node:18.16`, which is a pre-built image for the Node.js runtime.
- The working directory for the container is set to `/app`, and the contents of the current directory (your project files) are copied to this directory inside the container.
- The `RUN` command installs the necessary Node.js dependencies from the `package.json` file using `npm install`.
- The `EXPOSE` command specifies that the container will listen on port 5050, which is the default port used by the Node.js server.
- The `CMD` instruction specifies the command to run when the container starts — in this case, `npm start` to start the Node.js server.

Overall, the Dockerfile for the frontend sets up a container to serve the compiled React app using nginx, while the Dockerfile for the backend creates a container to run the Node.js server, both exposing relevant ports for external communication.

Container orchestration:

Docker Compose

Docker Compose is a tool that simplifies the process of defining and running multi-container Docker applications. It uses a YAML file (`docker-compose.yml`) to define the services, networks, and volumes required for an application. Docker Compose allows you to manage and orchestrate multiple containers by defining and starting them all at once.

Basic Docker Compose Commands:

- `docker-compose up`: Starts the containers defined in the `docker-compose.yml` file.
- `docker-compose down`: Stops and removes the containers defined in the `docker-compose.yml` file.
- `docker-compose ps`: Lists all the containers in the Docker Compose project.

Example Workflow:

We began by creating a `docker-compose.yml` file to understand how different images (such as for frontend and backend) and network services work together in a containerized environment. This setup helps manage both the frontend and backend containers with a single command, making it easier to work with multi-container applications.

```
docker-compose.yml
42
43
44
45 services:
46   backend:
47     build: ./backend
48     ports:
49       - "5050:5050"
50     networks:
51       - mern_network
52     environment:
53       MONGO_URI: mongodb://mongo:27017/mydatabase
54     depends_on:
55       - mongodb
56       - elasticsearch # Add Elasticsearch dependency for logging/search integration if need
57
58   frontend:
59     build: ./frontend
60     ports:
61       - "5173:5173"
62     networks:
63       - mern_network
64     environment:
65       REACT_APP_API_URL: http://backend:5050
66
67   mongodb:
68     image: mongo:latest
69     ports:
70       - "27017:27017"
71     networks:
72       - mern_network
73     volumes:
74       - mongo-data:/data/db
75
76   elasticsearch:
77     image: elasticsearch:8.10.2
78     container_name: elasticsearch
79     environment:
80       - discovery.type=single-node
81       - "ES_JAVA_OPTS=-Xms1g -Xmx1g"
82       - xpack.security.enabled=false
83       # - "ES_JAVA_OPTS=-Xms256m -Xmx256m"
84     ports:
85       - "9200:9200" # Elasticsearch HTTP
86       - "9300:9300" # Elasticsearch Kibana
```

```

🔥 docker-compose.yml
45   services:
76     elasticsearch:
84       ports:
85         - "9200:9200" # Elasticsearch HTTP
86         - "9300:9300" # Elasticsearch transport
87       networks:
88         - mern_network
89       volumes:
90         - es_data:/usr/share/elasticsearch/data
91
92     logstash:
93       image: logstash:8.10.2
94       container_name: logstash
95       ports:
96         - "5044:5044"
97       networks:
98         - mern_network
99       volumes:
100        - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf
101       depends_on:
102        - elasticsearch
103
104     kibana:
105       image: kibana:8.10.2
106       container_name: kibana
107       environment:
108        - XPACK_SECURITY_ENABLED=false
109        - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
110       ports:
111        - "5601:5601"
112       networks:
113        - mern_network
114       depends_on:
115        - elasticsearch
116
117   networks:
118     mern_network:
119       driver: bridge
120
121   volumes:
122     mongo-data:
123       driver: local # Persist MongoDB data locally
124     es_data:
125       driver: local # Persist Elasticsearch data locally

```

This `docker-compose.yml` file defines a multi-container application with services for the backend, frontend, MongoDB, Elasticsearch, Logstash, and Kibana. Here's a short explanation of each part:

Services:

1. **backend:**

- Builds the backend from the `./backend` directory.
- Exposes port 5050.
- Connects to the `mern_network`.
- Environment variable `MONGO_URI` connects to MongoDB.
- Depends on MongoDB and Elasticsearch for logging and search functionality.

2. **frontend:**

- Builds the frontend from the `./frontend` directory.
- Exposes port 5173.
- Connects to the `mern_network`.
- Environment variable `REACT_APP_API_URL` points to the backend.

3. **mongodb:**

- Uses the `mongo:latest` image.
- Exposes port 27017.
- Connects to the `mern_network`.
- Persists data using a local volume `mongo-data`.

4. **elasticsearch:**

- Uses the `elasticsearch:8.10.2` image.
- Exposes ports 9200 (HTTP) and 9300 (transport).
- Configures Elasticsearch to run as a single-node and disables security.
- Persists data using a local volume `es_data`.

5. **logstash:**

- Uses the `logstash:8.10.2` image.
- Exposes port 5044.
- Connects to the `mern_network`.
- Mounts a custom `logstash.conf` configuration file.
- Depends on Elasticsearch for log storage.

6. **kibana:**

- Uses the `kibana:8.10.2` image.
- Exposes port 5601.

- Connects to the **mern_network**.
- Depends on Elasticsearch to visualize logs.

Networks:

- **mern_network**: A bridge network for all services to communicate with each other.

Volumes:

- **mongo-data**: Persists MongoDB data locally.
- **es_data**: Persists Elasticsearch data locally.

This file sets up a complete MERN stack with integrated logging and search capabilities using Elasticsearch, Logstash, and Kibana.

On doing:

```
docker compose up -d
```

We get this:

```
=> [backend internal] load build context
=> => transferring context: 250B
=> CACHED [frontend 2/5] WORKDIR /app
=> CACHED [backend 3/5] COPY package.json .
=> CACHED [backend 4/5] RUN npm install
=> CACHED [backend 5/5] COPY .
=> CACHED [frontend 3/5] COPY package.json .
=> CACHED [frontend 4/5] RUN npm install
=> CACHED [frontend 5/5] COPY .
=> [backend] exporting to image
=> => exporting layers
=> => writing image sha256:1ea3efb543cd4d84cd6be2086e7e58e7dff3f5b8fcefbcd5ea4b0f9200757e4d
=> => naming to docker.io/library/employee_management_system_spefinalproject-backend
=> [frontend] exporting to image
=> => exporting layers
=> => writing image sha256:bdb8950bc5e5fae3fbd7b2f4ddec3c88f1198709deb30f4a3342e56b8e2f17c
=> => naming to docker.io/library/employee_management_system_spefinalproject-frontend
=> [frontend] resolving provenance for metadata file
=> [backend] resolving provenance for metadata file
[+] Running 9/9
✔ Network employee_management_system_spefinalproject_mern_network Created
✔ Volume "employee_management_system_spefinalproject_es_data" Created
✔ Volume "employee_management_system_spefinalproject_mongo-data" Created
✔ Container elasticsearch Started
✔ Container employee_management_system_spefinalproject-frontend-1 Started
✔ Container employee_management_system_spefinalproject-mongodb-1 Started
✔ Container kibana Started
✔ Container logstash Started
✔ Container employee_management_system_spefinalproject-backend-1 Started
```

Docker Compose is great for local development and small-scale applications, but it lacks advanced features like auto-scaling, self-healing, and orchestration that are needed for production-level systems. This is where **Kubernetes** comes in.

Kubernetes

Why use Kubernetes instead of Docker Compose?

1. **Auto-Scaling:** Kubernetes can automatically scale your application based on traffic load. If your containers are overwhelmed, Kubernetes can create more replicas to handle the increased load and reduce them when the demand drops.
2. **Self-Healing:** Kubernetes can detect unhealthy containers and automatically restart or replace them. This ensures higher availability and minimizes downtime.
3. **Load Balancing:** Kubernetes has built-in load balancing for distributing traffic across multiple instances of your containers.
4. **Orchestration:** Kubernetes automates the deployment, management, and coordination of multi-container applications. It handles complex deployment strategies (like rolling updates, canary deployments, etc.), which are harder to manage with Docker Compose.
5. **Persistent Storage:** Kubernetes allows for more robust handling of persistent storage using StatefulSets and Persistent Volumes (PVs), unlike Docker Compose's simpler volume management.
6. **Distributed Systems:** Kubernetes supports deploying applications across a cluster of machines, ensuring that workloads are distributed across multiple nodes for improved availability and resilience.

Kubernetes Overview:

- **Pods:** The basic unit of deployment, a pod runs one or more containers.
- **ReplicaSets:** Ensures that a specified number of pod replicas are running.
- **Deployments:** Manages the deployment of ReplicaSets and ensures a desired state.
- **Services:** Provides stable networking for pods and load balancing.
- **Namespaces:** Allow for logical separation of resources in the cluster.

Here's a high-level explanation of how Kubernetes works:

- You specify your application and infrastructure requirements in a Kubernetes manifest file, known as a **resource file**, which outlines the desired state of your application.
- The Kubernetes API is then used to submit this manifest file to the Kubernetes control plane, a set of components that manage the overall state of the cluster.
- The control plane schedules the application components on worker nodes, which are responsible for running the containers that make up your application.
- The Kubernetes control plane continuously monitors the worker nodes and their application components, taking necessary actions to maintain the system's desired state.
- If a worker node fails or becomes unavailable, Kubernetes automatically reschedules the application components to other available nodes to ensure availability.

To run our application on Kubernetes, we defined several resource files in YAML format, each describing a Kubernetes object. A Kubernetes object is a persistent entity in the Kubernetes system, such as a Pod, Service, Deployment, or ConfigMap.

Here are the resource file we created for backend, frontend and mongodb:

- Frontend deployment and service - `frontend-app.yaml`

```
k8s > ! frontend.yaml
30
37 apiVersion: v1
38 kind: Namespace
39 metadata:
40   name: mern-stack
41 ---
42
43 apiVersion: apps/v1
44 kind: Deployment
45 metadata:
46   name: frontend
47   namespace: mern-stack
48 spec:
49   replicas: 1
50   selector:
51     matchLabels:
52       app: frontend
53   template:
54     metadata:
55       labels:
56         app: frontend
57     spec:
58       containers:
59         - name: frontend
60           image: bhavil13/ems-spe_fp-frontend
61           ports:
62             - containerPort: 5173
63           env:
64             - name: REACT_APP_API_URL
65               value: http://backend:5050
66 ---
67 apiVersion: v1
68 kind: Service
69 metadata:
70   name: frontend
71   namespace: mern-stack
72 spec:
73   ports:
74     - port: 5173
75   selector:
76     app: frontend
77
```

- Backend deployment and service - `backend-app.yaml`

```
k8s > ! backend.yaml
36
37
38 apiVersion: v1
39 kind: Namespace
40 metadata:
41   name: mern-stack
42 ---
43
44 apiVersion: apps/v1
45 kind: Deployment
46 metadata:
47   name: backend
48   namespace: mern-stack
49 spec:
50   replicas: 1
51   selector:
52     matchLabels:
53       app: backend
54   template:
55     metadata:
56       labels:
57         app: backend
58     spec:
59       containers:
60       - name: backend
61         image: bhavil13/ems-spe_fp-backend
62         ports:
63         - containerPort: 5050
64         env:
65         - name: MONGO_URI
66           value: mongodb://mongodb:27017/mydatabase
67 ---
68 apiVersion: v1
69 kind: Service
70 metadata:
71   name: backend
72   namespace: mern-stack
73 spec:
74   ports:
75   - port: 5050
76   selector:
77     app: backend
78
```

- MongoDB deployment and service - `mongo-app.yaml`

```
k8s > ! mongoapp.yaml
54
55   apiVersion: v1
56   kind: Namespace
57   metadata:
58     name: mern-stack
59   ---
60
61   apiVersion: apps/v1
62   kind: Deployment
63   metadata:
64     name: mongodb
65     namespace: mern-stack
66   spec:
67     replicas: 1
68     selector:
69       matchLabels:
70         app: mongodb
71     template:
72       metadata:
73         labels:
74           app: mongodb
75       spec:
76         containers:
77         - name: mongodb
78           image: mongo:latest
79           ports:
80             - containerPort: 27017
81           volumeMounts:
82             - name: mongo-data
83               mountPath: /data/db
84         volumes:
85         - name: mongo-data
86           persistentVolumeClaim:
87             claimName: mongo-pvc
88   ---
89   apiVersion: v1
90   kind: Service
91   metadata:
92     name: mongodb
93     namespace: mern-stack
94   spec:
95     ports:
96     - port: 27017
97     selector:
```

```
      app: mongodb
98   ---
99
100  apiVersion: v1
101  kind: PersistentVolumeClaim
102  metadata:
103    name: mongo-pvc
104    namespace: mern-stack
105  spec:
106    accessModes:
107    - ReadWriteOnce
108    resources:
109      requests:
110        storage: 1Gi
```

Now, on doing minikube service frontend, we will be able to see our project:

MongoDB. Create Employee

Employee Records

Name	Position	Level	Action
Bhavil Sharma	Chief of Staff at Zomato	Senior	Edit Delete
Chamanlal	Janitor	Intern	Edit Delete
Skibbidi	Chief Toilet	Senior	Edit Delete

MongoDB. Create Employee

Create/Update Employee Record

Employee Info

This information will be displayed publicly so be careful what you share.

Name

Position

☐ Intern ☐ Junior ☐ Senior

Save Employee Record

localhost:5173/create

Config-Management:

For configuration management for deploying our application to host machines, we are using Ansible. It uses a combination of playbooks and inventory files to define and manage configurations.

A playbook in Ansible is a collection of tasks that are organized in a file and executed on a set of hosts defined in an inventory file. The tasks in a playbook are written in YAML format and can be used to perform a variety of operations, such as installing packages, managing services, copying files, and executing commands. After running the playbook, we get:

```
PLAY [pull updated image] *****
TASK [Gathering Facts] *****
[WARNING]: Platform darwin on host localhost is using the discovered Python interpreter at
/opt/homebrew/bin/python3.11, but future installation of another Python interpreter could change the meaning of that
path. See https://docs.ansible.com/ansible-core/2.14/reference_appendices/interpreter_discovery.html for more
information.
ok: [localhost]

TASK [Check Minikube's status.] *****
ok: [localhost]

TASK [Start Minikube if it's not running.] *****
skipping: [localhost]

TASK [Install kubernetes python package] *****
ok: [localhost]

TASK [Delete any existing mongo deployment] *****
ok: [localhost]

TASK [Create a deployment for mongo] *****
changed: [localhost]

TASK [Delete any existing deployment of Backend Deployment] *****
ok: [localhost]

TASK [Create a deployment for Blacklist Service] *****
changed: [localhost]

TASK [Delete any existing deployment of Frontend Deployment] *****
ok: [localhost]

TASK [Create a deployment for Frontend Service] *****
changed: [localhost]

TASK [Delete any existing deployment of Filebeat] *****
changed: [localhost]

TASK [Create a deployment for Filebeat] *****
changed: [localhost]

TASK [Delete any existing deployment of Logstash] *****
changed: [localhost]

TASK [Create a deployment for Logstash] *****
changed: [localhost]

PLAY RECAP *****
localhost : ok=13  changed=7  unreachable=0  failed=0  skipped=1  rescued=0  ignored=0
```

Monitoring:

Filebeat and Logstash Configuration

Filebeat and Logstash are commonly used together for logging and monitoring in distributed system architectures. Here's how they work:

- Filebeat collects log files or other data sources and forwards the log data to Logstash.
- Logstash processes the incoming log data by parsing the log lines, adding metadata, and applying filters (such as the **grok** pattern).
- Logstash then sends the processed data to Elasticsearch or another data store for indexing and further analysis.
- Kibana can be used to visualize and analyze the log data, whether in real-time or for historical analysis.

Kibana

By using the grok pattern mentioned above, we can track how users interact with social media, capturing actions like like/unlike, friend/unfriend, login, and new user creation, along with the user's email ID. The log level indicates whether the log represents information, an error, a warning, etc., and the timestamp provides the precise time when the action was performed.

With this log data stored in Elastic Cloud, we can leverage Kibana to create visualization dashboards that offer insights into user interactions.

Here's some content for **Elasticsearch**:

Elasticsearch

Elasticsearch is a distributed, RESTful search and analytics engine designed for horizontal scalability, reliability, and real-time search capabilities. It is commonly used to store, search,

and analyze large volumes of data quickly. In the context of logging and monitoring, Elasticsearch plays a central role in storing and indexing log data, allowing for fast retrieval and analysis.

Here's how Elasticsearch fits into the logging pipeline:

- **Data Storage & Indexing:** Elasticsearch stores log data in an indexed format, making it highly efficient for searching and querying large datasets. Logs are ingested through tools like Filebeat and Logstash, and Elasticsearch creates indices for quick search access.
- **Full-Text Search:** Elasticsearch supports powerful full-text search features, allowing users to query logs with complex queries and filters. You can search for specific log entries, keywords, or patterns in large volumes of data in real-time.
- **Real-Time Analytics:** With its distributed architecture, Elasticsearch allows for real-time analytics and fast data retrieval, making it an ideal choice for logging systems where timely insights are crucial.
- **Scalability:** Elasticsearch is built to scale horizontally, which means it can handle large amounts of data as your application or infrastructure grows. It can distribute the load across multiple nodes in a cluster, ensuring high availability and performance.

In combination with Filebeat, Logstash, and Kibana (the ELK stack), Elasticsearch provides a powerful solution for managing, searching, and visualizing log data from various sources, helping to monitor application performance, troubleshoot issues, and gain insights into user behavior and system health.


Elasticsearch:

```

{
  "name": "0888c2e67315",
  "cluster_name": "docker-cluster",
  "cluster_uuid": "3f8PM0GvHGBV-t_132RA",
  "version": {
    "number": "8.10.2",
    "build_flavor": "default",
    "build_type": "docker",
    "build_hash": "6d206dd8ce62365be91aca96427de4622e970ef9e",
    "build_date": "2023-09-19T08:16:24.540000070Z",
    "build_snapshot": false,
    "lucene_version": "9.4.0",
    "minimum_wire_compatibility_version": "7.17.0",
    "minimum_index_compatibility_version": "7.0.0"
  },
  "tagline": "You Know, for Search"
}

```

Kibana:

elastic

Find apps, content, and more.

*/

Clusters


Elasticsearch nodes

Overview

Nodes

Indices

Ingest Pipelines

Elasticsearch node detected

The following nodes are not monitored. Click 'Monitor with Metricbeat' below to start monitoring.

Filter Nodes...

Name ↑	Alerts	Status	Roles
0888c2e67315 172.18.0.2:9300		● Offline	<div>data data_cold</div> <div>data_content data_frozen</div> <div>data_hot +6</div>

Rows per page: 20

Set up monitoring for new node

You are in setup mode. The (P) icon indicates configuration options.

Monitor 0888c2e67315 node with Metricbeat

URL of monitoring cluster

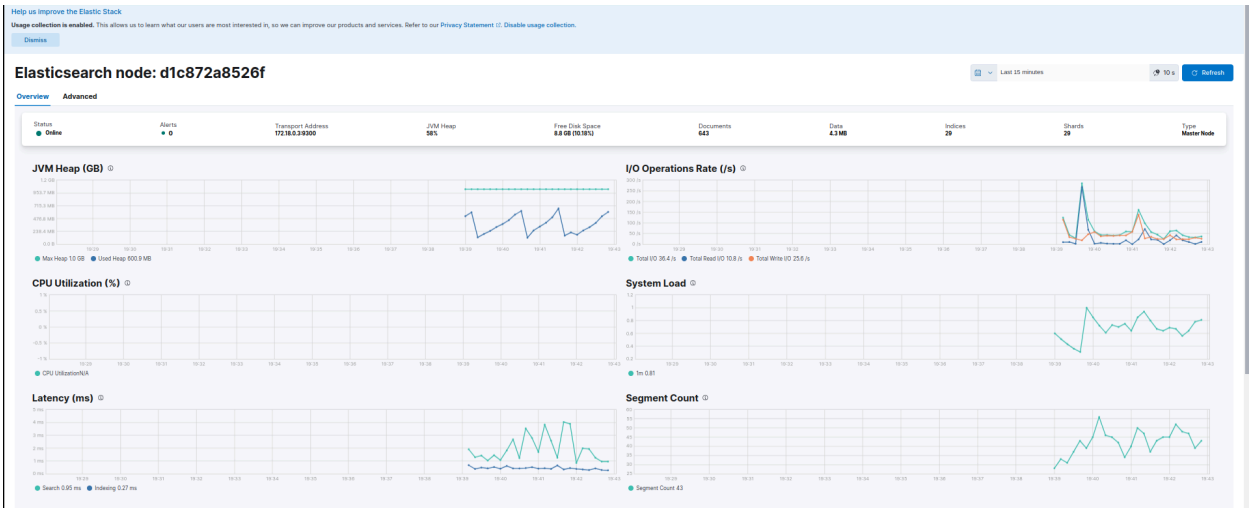
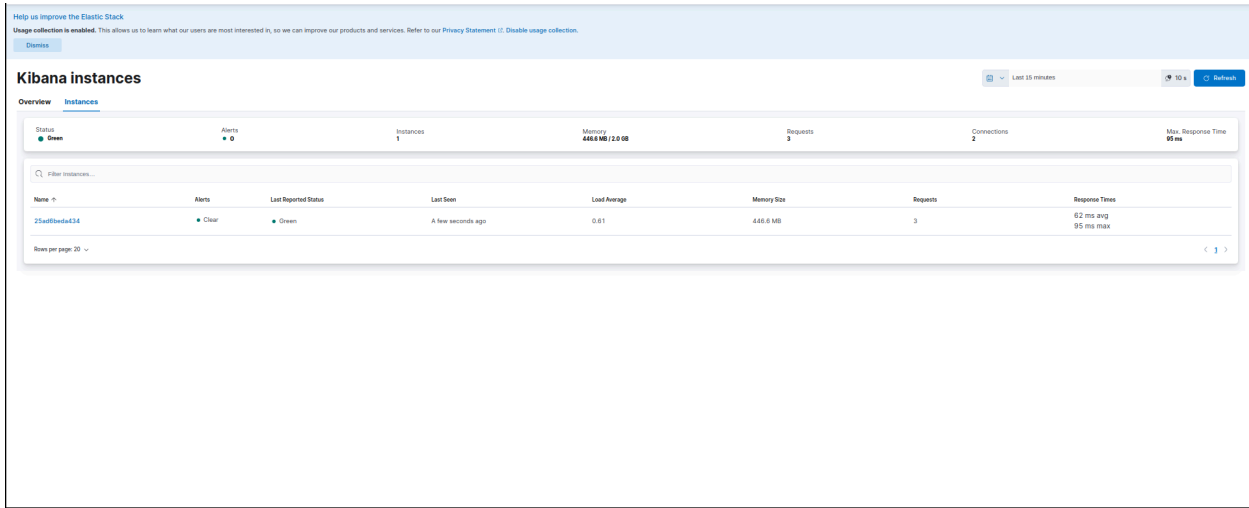
http://localhost:9200

Typically a single URL. If multiple URLs, separate with a comma. The running Metricbeat instance must be able to communicate with these Elasticsearch servers.

Close

Next

Kibana Vizualizations:



Help us improve the Elastic Stack

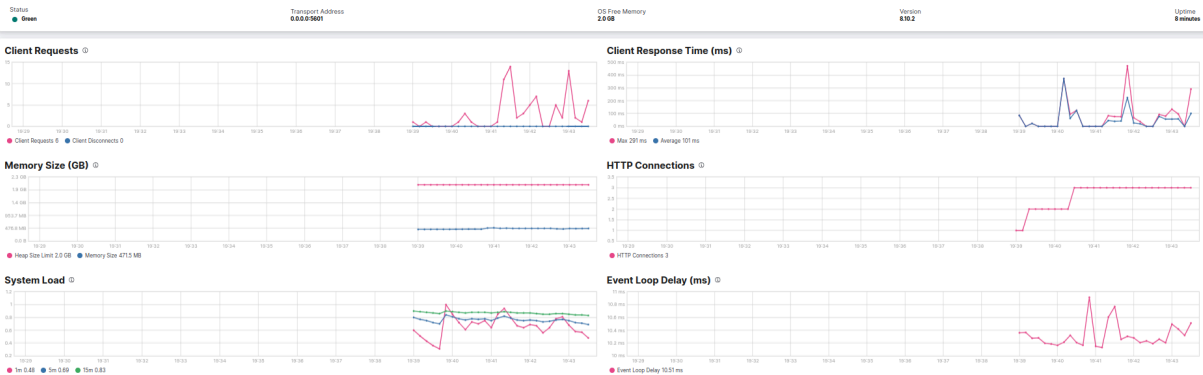
Usage collection is enabled. This allows us to learn what our users are most interested in, so we can improve our products and services. Refer to our [Privacy Statement](#). [Disable usage collection.](#)

[Dismiss](#)

Kibana instance: 25ad6beda434

Last 15 minutes

10 s [Refresh](#)



Kibana overview

Last 15 minutes

10 s [Refresh](#)

