Care Connect

Doctor Appointment System with Kubernetes and Microservices

Prepared by:
Deven Kapadia (MT2024075)
Shubh Patel (MT2024148)

Date:
May 19, 2025

Contents

1 Introduction

CareConnect is a web-based Doctor Appointment System designed to streamline healthcare appointment scheduling and management. The platform enables patients to book appointments with doctors, manage their profiles, and receive notifications, while providing administrative tools for healthcare providers. The primary goal of the project was to implement a scalable, resilient system using a microservices architecture deployed on Kubernetes (K8s). This report details the project's objectives, architecture, Kubernetes implementation, challenges, results, and future recommendations.

1.1 Objectives

- Develop a user-friendly appointment booking system for patients and doc- tors.
- Implement a microservices-based architecture to ensure modularity and scalability.
- Deploy the system on Kubernetes to achieve high availability, scalability, and efficient resource management.
- Integrate supporting services like RabbitMQ for messaging, PostgreSQL for data storage, and ELK Stack for logging.

2 Project Overview

2.1 System Description

CareConnect consists of multiple microservices, each handling specific function- alities such as user authentication, doctor management, appointment schedul- ing, email notifications, backup and archiving. The frontend is built with React, providing a responsive user interface, while the backend leverages Spring Boot for robust API development. The system is containerized using Docker and orchestrated via Kubernetes.

2.2 Technologies Used

- **Frontend**: React.js, HTML, CSS, JavaScript
- **Backend**: Spring Boot (Java), REST APIs
- **Background worker**: Python
- **Database**: PostgreSQL
- **Message Queue**: RabbitMQ
- **Logging**: ELK Stack (Elasticsearch, Logstash, Kibana)
- **Containerization**: Docker
- **Orchestration**: Kubernetes (K8s)
- **Other Tools**: Git, Maven, Ingress-Nginx, pgAdmin

2.3 Team

- Deven Kapadia (MT2024075)
- Shubh Patel (MT2024148)

3 Architecture

CareConnect adopts a microservices architecture, with the following key services:

- **Auth Service**: Handles user authentication and JWT-based security.
- **User Service**: Manages patient and doctor profiles.
- **Doctor Service**: Oversees doctor-specific functionalities like availability and scheduling.
- **Admin Service**: Provides administrative tools for system management.
- **Email Service**: Consumes RabbitMQ queue and sends appointment confirmations and reminders.
- **Archive Service**: Manages historical appointment data and performs regular DB backups.
- **Frontend Service**: Serves the React-based user interface.

Each service is independently deployable, communicates via REST APIs, and is containerized using Docker. RabbitMQ facilitates asynchronous messaging for email notifications, while PostgreSQL stores persistent data. The ELK Stack provides centralized logging for monitoring and debugging.

4 Kubernetes Implementation

The core focus of CareConnect was to leverage Kubernetes for orchestrating the microservices. The Kubernetes configuration is detailed in the **k8s** folder of the repository, which includes YAML files for deployments, services, and other re- sources.

4.1 Kubernetes Components

- **Namespace**: A dedicated namespace (careconnect) isolates CareConnect resources (namespace.yaml).
- **Deployments**: Each microservice has a dedicated deployment (e.g., deployment-auth- specifying replicas, container images, and resource limits.
- **Services**: ClusterIP services (e.g., service-auth.yaml) expose microser- vices within the cluster.
- **Ingress**: An Ingress resource (ingress.yaml) with Nginx Ingress Controller routes external traffic.
- **Persistent Storage**: Persistent Volume (PV) and Persistent Volume Claim (PVC) configurations (pv.yaml, pvc.yaml) ensure PostgreSQL data persistence.
- **Horizontal Pod Autoscaling (HPA)**: An HPA configuration (hpa.yaml) scales pods based on CPU/memory usage.
- **Secrets**: Sensitive data are managed using Kubernetes Secrets (secrets.yaml).
- **Supporting Services**: RabbitMQ (rabbitmq-service.yaml), ELK Stack (elk-deployment.yaml), and pgAdmin (pgadmin.yaml).

4.2 Deployment Workflow

- **Containerization**: Services are packaged into Docker containers.
- **K8s Cluster Setup**: Deployed on a Kubernetes cluster (e.g., Minikube or AWS EKS).
- **Configuration**: YAML files define the desired state for resources.
- **Scripts**: Shell scripts (start.sh, stop.sh) automate cluster management.
- **Monitoring**: ELK Stack provides real-time logs.

4.3 Benefits of Kubernetes

- **Scalability**: HPA ensures efficient load handling.
- **Resilience**: Pod restarts minimize downtime.
- **Portability**: Enables deployment across cloud providers.

5 Methodology

The project followed an Agile methodology with iterative development cycles:

- **Planning**: Defined microservices and Kubernetes architecture.
- **Development**: Implemented services using Spring Boot and React.
- **Containerization**: Built Docker images.
- **Kubernetes Deployment**: Configured and tested K8s resources.
- **Testing**: Conducted integration tests.
- **Monitoring**: Set up ELK Stack and pgAdmin.

5.1 Challenges

- **Kubernetes Complexity**: Configuring Ingress and HPA required iterative testing.
- **Microservices Coordination**: Ensured via retry mechanisms and circuit breakers.
- **Resource Constraints**: Optimized Docker images for Minikube.

5.2 Security Features

1. **Secure Configuration Management**
   All sensitive Spring Boot backend configurations—such as database credentials, RabbitMQ credentials, and the JWT secret—are sourced from environment variables. These variables are injected at runtime using **Kubernetes Secrets**, ensuring that no credentials are hardcoded into the application.

2. **Secure Background Workers**
   Background worker services also use environment variables for sensitive configurations, which are likewise populated via Kubernetes Secrets. This guarantees consistent security practices across all components.

3. **Authentication and Request Filtering**

   - The **User Service** allows unauthenticated access only for public routes like registration and login.
   - All other routes in the system are protected using **header-based request filtering**, ensuring that only authenticated users with valid JWT can reach the controllers.

4. **Internal Service Communication**
   All services are exposed using **ClusterIP** and are only accessible internally via **Ingress**, which acts as a secure gateway. This isolates services from direct external

access and adds an additional layer of protection.

5. **Database Security**
   - The PostgreSQL database is **not exposed externally** and can only be accessed from within the cluster.
   - It is initialized during application boot with a predefined schema and seed data.
   - Database credentials are securely retrieved from Kubernetes Secrets during startup.
6. **Secure Message Broker (RabbitMQ)**
   RabbitMQ credentials are also managed through Kubernetes Secrets, maintaining consistent credential handling practices.
7. **CI/CD Credential Management**
   All credentials used in the CI/CD pipeline (e.g., for GitHub, Docker Hub, and Ansible Vault) are securely stored and accessed via **Jenkins Credentials**. This prevents the leakage of sensitive information in build scripts or logs.

5.3 Database Schema

<u>Enum Types</u>

Enum Type	**Values**
appointment_status	PENDING, CONFIRMED, COMPLETED, CANCELLED, REQUESTED
backup_status	SUCCESS, FAILURE, PENDING
gender_type	MALE, FEMALE, OTHER
payment_status	PENDING, COMPLETED, FAILED
user_role	PATIENT, DOCTOR, ADMIN

<u>Tables</u>

<u>User:</u>

Column Name	**Data Type**	**Constraints**
user_id	UUID	Primary Key
email	VARCHAR(255)	Not Null, Unique
password	VARCHAR(255)	Not Null
role	user_role	Not Null
name	VARCHAR(255)	Not Null
created_at	TIMESTAMP	
updated_at	TIMESTAMP	

Patients:

Column Name	Data Type	Constraints
patient_id	UUID	Primary Key
user_id	UUID	Foreign Key →app_users(user_id)
first_name	VARCHAR(100)	Not Null
last_name	VARCHAR(100)	Not Null
date_of_birth	DATE	Not Null
gender	gender_type	Not Null
created_at	TIMESTAMP	Not Null
updated_at	TIMESTAMP	Not Null

Doctors:

Column Name	Data Type	Constraints
doctor_id	UUID	Primary Key, Foreign Key → app_users(user_id)
specialization	VARCHAR(100)	Not Null
started_year	INTEGER	Not Null
consultation_fee	NUMERIC(10,2)	Not Null
about	TEXT	Not Null
image	VARCHAR(255)	Not Null
degree	VARCHAR(255)	Not Null
address	VARCHAR(255)	Not Null
created_at	TIMESTAMP	Not Null
updated_at	TIMESTAMP	Not Null

Payments:

Column Name	Data Type	Constraints
payment_id	UUID	Primary Key
amount	NUMERIC(10,2)	Not Null
payment_status	payment_status	Default: 'PENDING'
transaction_id	VARCHAR(100)	Not Null
created_at	TIMESTAMP	Not Null
updated_at	TIMESTAMP	Not Null,

Appointments:

Column Name	Data Type	Constraints
appointment_id	UUID	Primary Key
patient_id	UUID	Not Null, FK → patients(patient_id)
user_id	UUID	Not Null, FK → app_users(user_id)
doctor_id	UUID	Not Null, FK → doctors(doctor_id)
payment_id	UUID	FK → payments(payment_id), ON DELETE SET NULL
status	appointment_status	Default: 'PENDING'
notes	TEXT	
date	VARCHAR(255)	Not Null
time	VARCHAR(255)	Not Null
created_at	TIMESTAMP	Not Null
updated_at	TIMESTAMP	Not Null

Archived_appointments:

Column Name	Data Type	Constraints
appointment_id	UUID	Primary Key
patient_id	UUID	FK → patients(patient_id)
user_id	UUID	FK → app_users(user_id)
doctor_id	UUID	FK → doctors(doctor_id)
payment_id	UUID	FK → payments(payment_id)
status	appointment_status	
notes	TEXT	
date	VARCHAR(255)	Not Null
time	VARCHAR(255)	Not Null
created_at	TIMESTAMP	Not Null
updated_at	TIMESTAMP	Not Null

Backup_logs:

Column Name	Data Type	Constraints
backup_id	UUID	Primary Key
backup_time	TIMESTAMP	Not Null
status	backup_status	Default: 'PENDING'
error_message	TEXT	
created_at	TIMESTAMP	Not Null
updated_at	TIMESTAMP	Not Null

Feedback:

Column Name	Data Type	Constraints
feedback_id	UUID	Primary Key
comments	TEXT	
email	VARCHAR(255)	
name	VARCHAR(100)	
created_at	TIMESTAMP	Not Null
updated_at	TIMESTAMP	Not Null

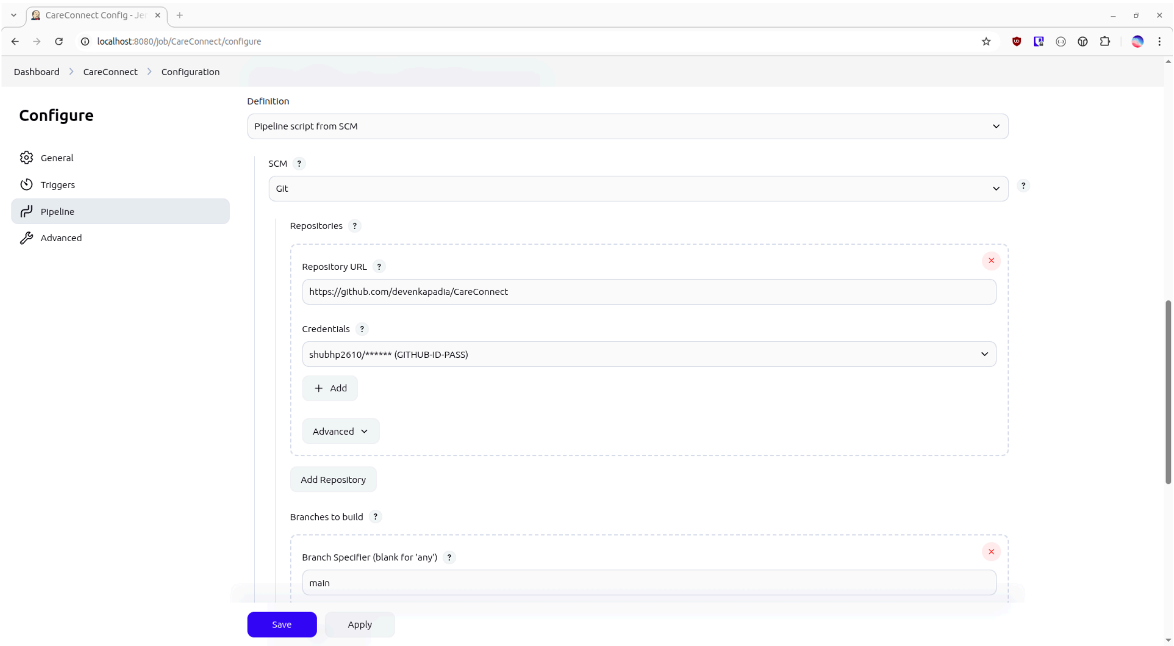
Unavailable_slots:

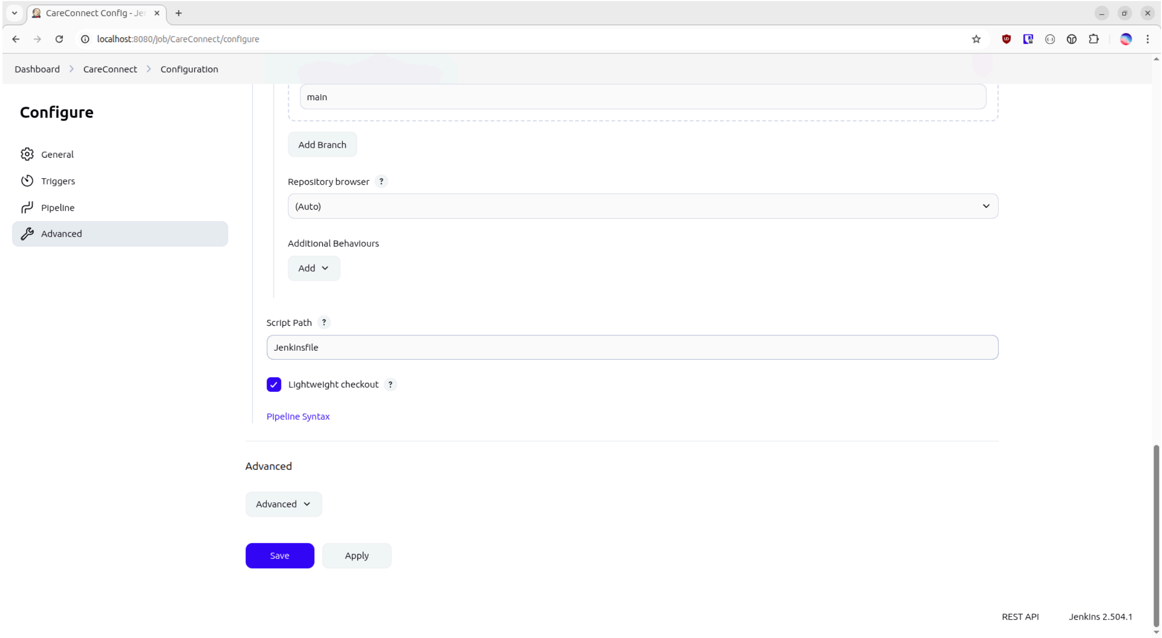
Column Name	Data Type	Constraints
id	UUID	Primary Key
doctor_id	UUID	Not Null, FK → doctors(doctor_id)
date	DATE	Not Null
start_time	TIME	Not Null
end_time	TIME	Not Null

6 Results

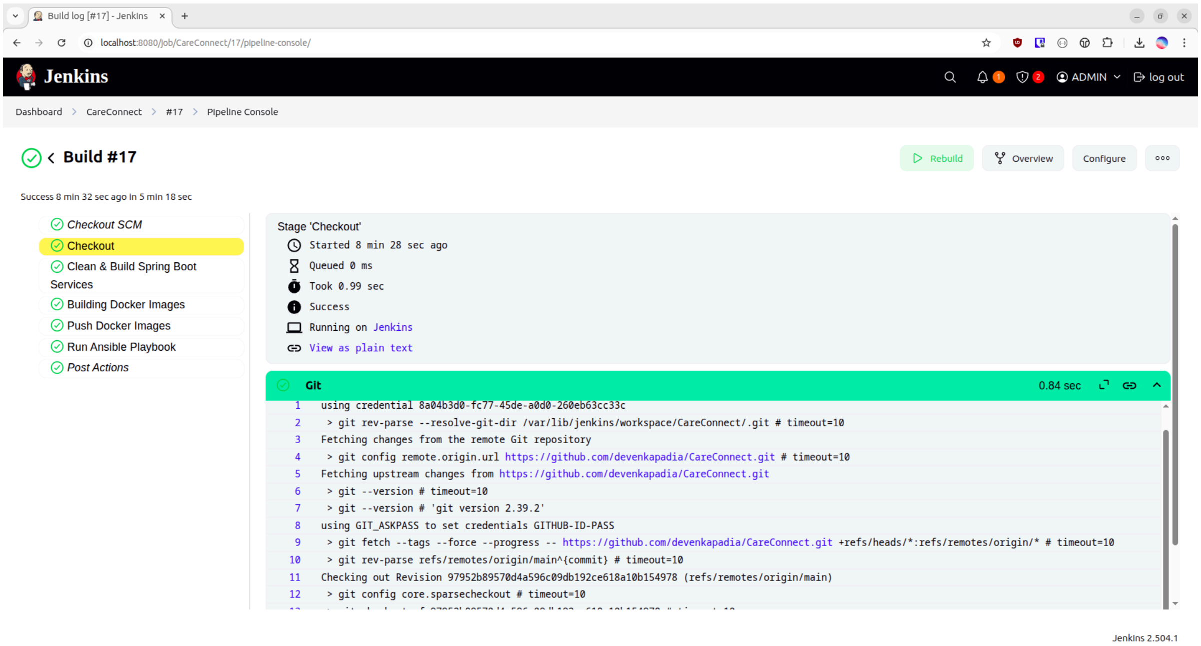
- Deployed a fully functional Doctor Appointment System on Kubernetes.
- Achieved high availability with automated scaling via HPA.
- Implemented a secure microservices architecture.
- Integrated ELK Stack for real-time logging.
- Confirmed intuitive UI/UX through user testing.

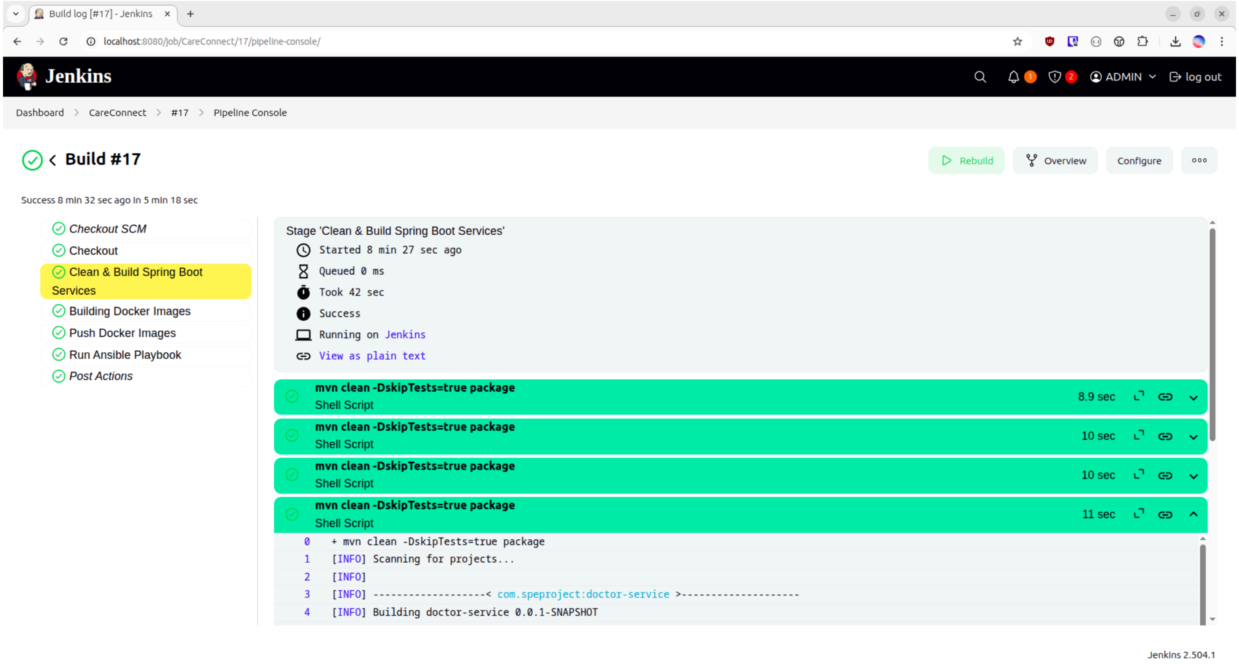
Jenkins project configuration

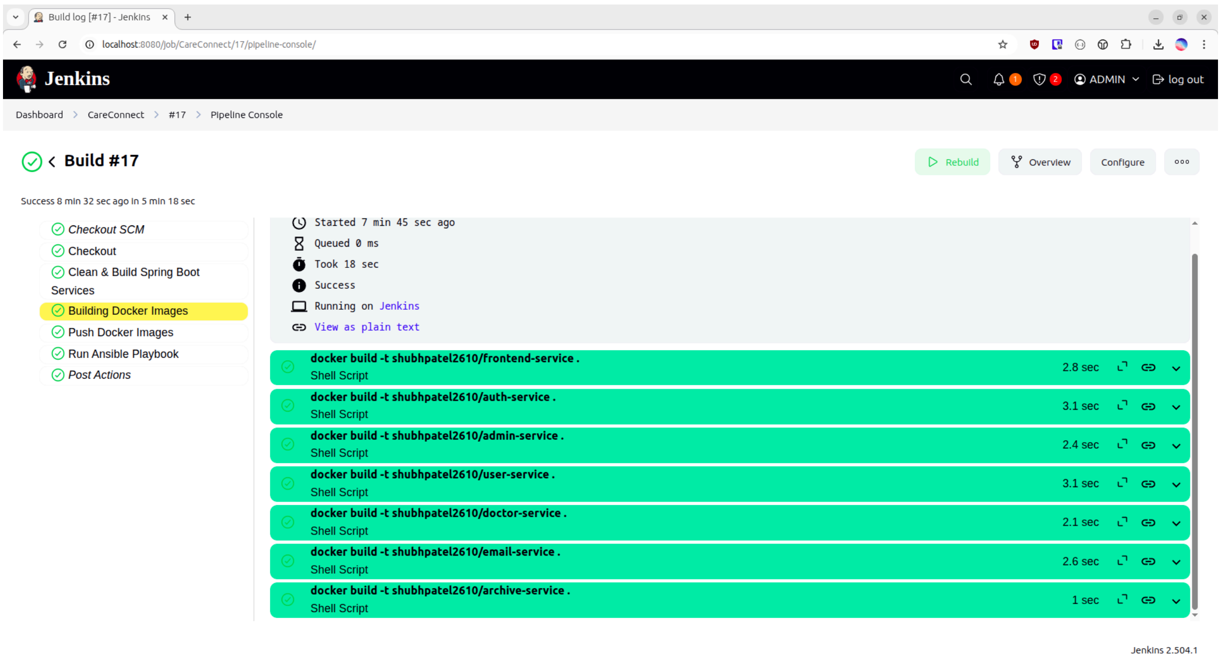


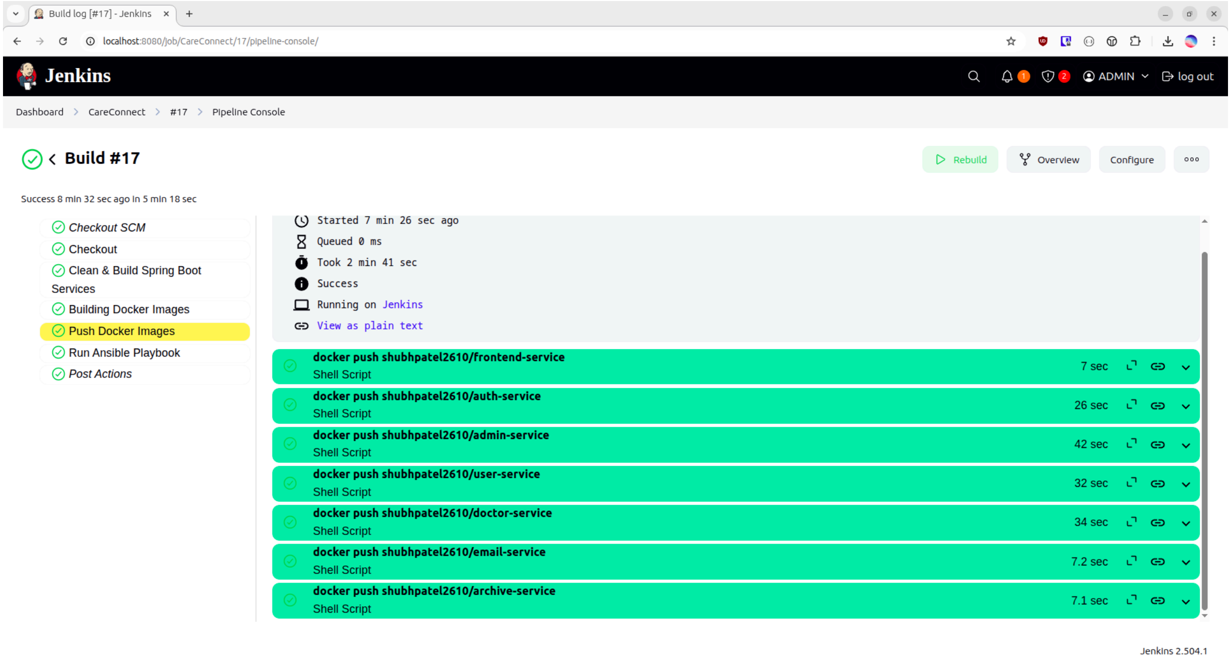


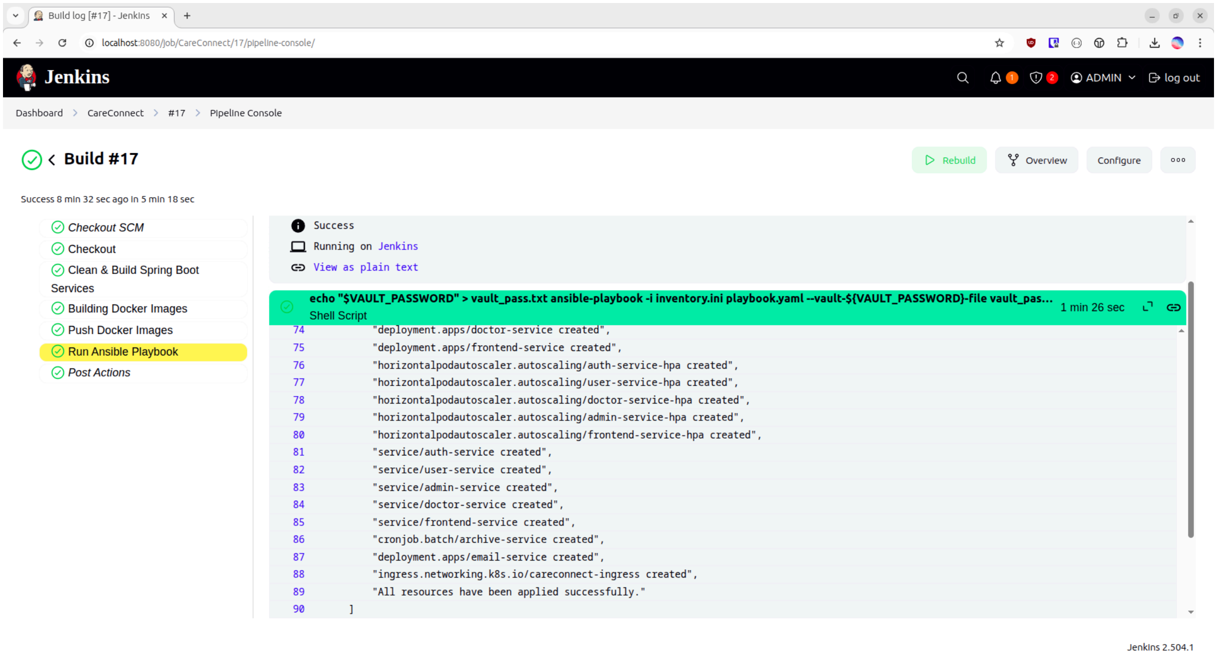
Jenkins Build Stages



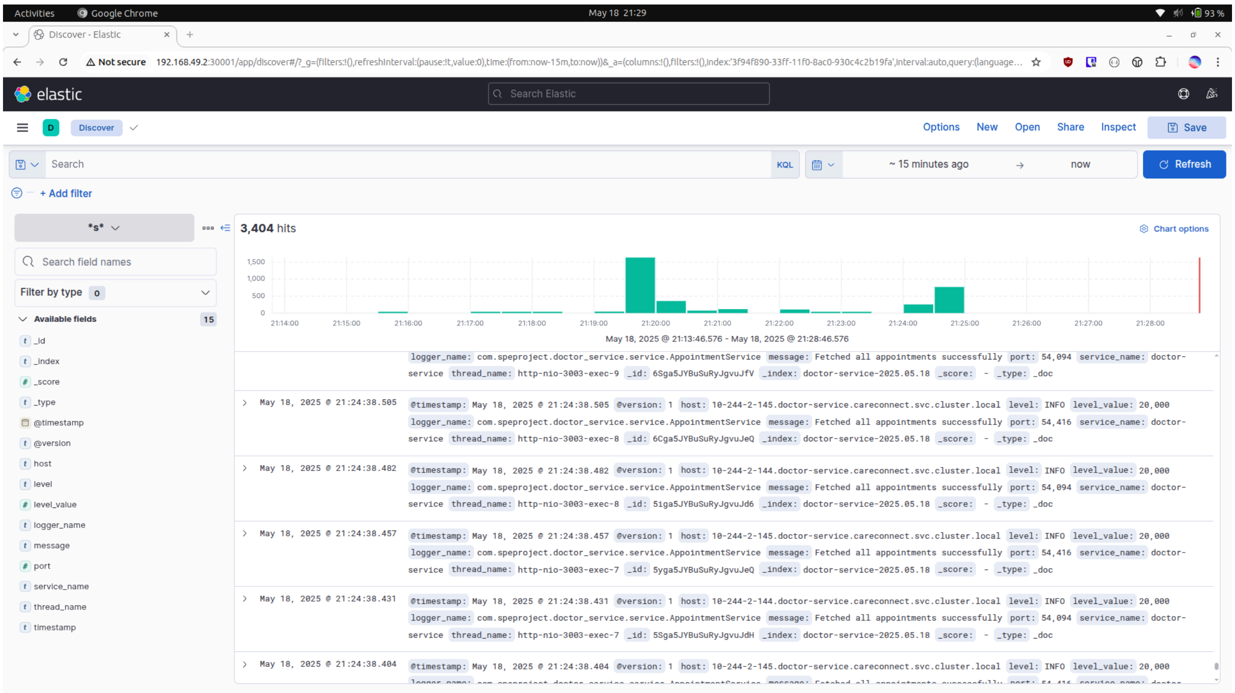

Build log [#17] - Jenkins
localhost:8080/job/CareConnect/17/pipeline-console/
Jenkins
ADMIN
log out
Dashboard > CareConnect > #17 > Pipeline Console
Build #17
Rebuild
Overview
Configure
Success 8 min 32 sec ago in 5 min 18 sec
Checkout SCM
Checkout
Clean & Build Spring Boot Services
Building Docker Images
Push Docker Images
Run Ansible Playbook
Post Actions
Stage 'Clean & Build Spring Boot Services'
Started 8 min 27 sec ago
Queued 0 ms
Took 42 sec
Success
Running on Jenkins
View as plain text
mvn clean -DskipTests=true package
Shell Script
8.9 sec
mvn clean -DskipTests=true package
Shell Script
10 sec
mvn clean -DskipTests=true package
Shell Script
10 sec
mvn clean -DskipTests=true package
Shell Script
11 sec
0 + mvn clean -DskipTests=true package
1 [INFO] Scanning for projects...
2 [INFO]
3 [INFO] -------------------< com.speproject:doctor-service >-------------------
4 [INFO] Building doctor-service 0.0.1-SNAPSHOT
Jenkins 2.504.1


Build log [#17] - Jenkins
localhost:8080/job/CareConnect/17/pipeline-console/
Jenkins
ADMIN
log out
Dashboard > CareConnect > #17 > Pipeline Console
Build #17
Rebuild
Overview
Configure
Success 8 min 32 sec ago in 5 min 18 sec
Checkout SCM
Checkout
Clean & Build Spring Boot Services
Building Docker Images
Push Docker Images
Run Ansible Playbook
Post Actions
Started 7 min 45 sec ago
Queued 0 ms
Took 18 sec
Success
Running on Jenkins
View as plain text
docker build -t shubhpatel2610/frontend-service .
Shell Script
2.8 sec
docker build -t shubhpatel2610/auth-service .
Shell Script
3.1 sec
docker build -t shubhpatel2610/admin-service .
Shell Script
2.4 sec
docker build -t shubhpatel2610/user-service .
Shell Script
3.1 sec
docker build -t shubhpatel2610/doctor-service .
Shell Script
2.1 sec
docker build -t shubhpatel2610/email-service .
Shell Script
2.6 sec
docker build -t shubhpatel2610/archive-service .
Shell Script
1 sec
Jenkins 2.504.1

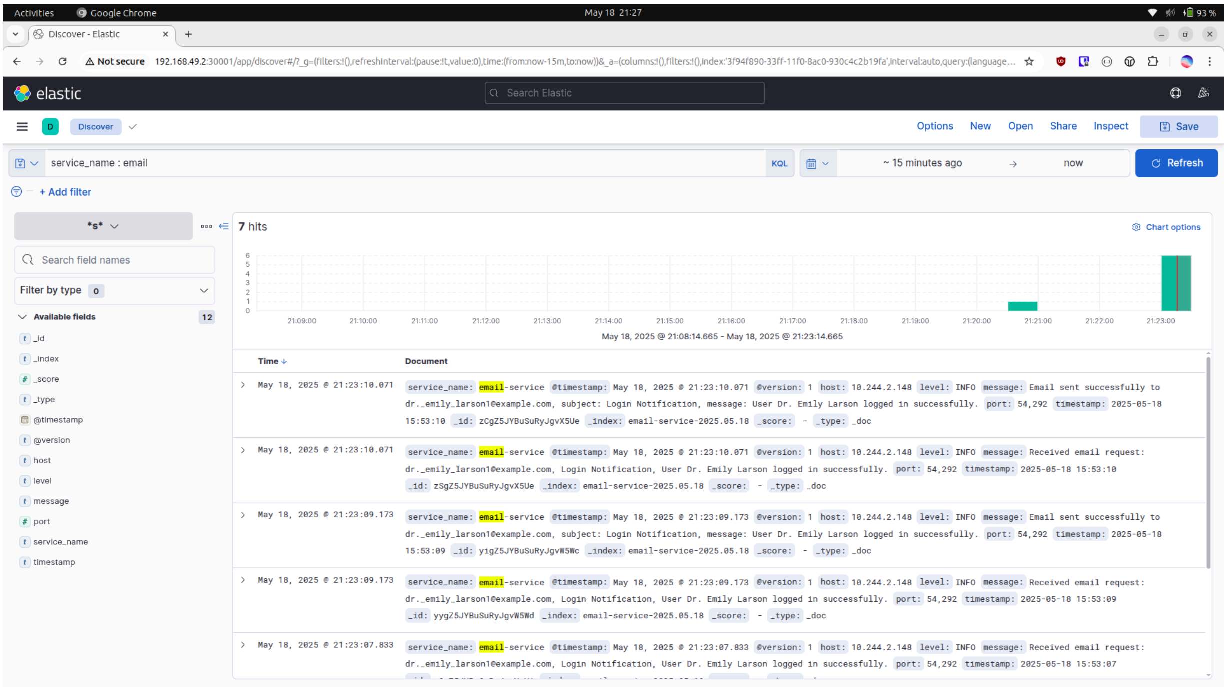

Build log [#17] - Jenkins
localhost:8080/job/CareConnect/17/pipeline-console/
Jenkins
ADMIN
log out
Dashboard > CareConnect > #17 > Pipeline Console
Build #17
Rebuild
Overview
Configure
Success 8 min 32 sec ago in 5 min 18 sec
Checkout SCM
Checkout
Clean & Build Spring Boot Services
Building Docker Images
Push Docker Images
Run Ansible Playbook
Post Actions
Started 7 min 26 sec ago
Queued 0 ms
Took 2 min 41 sec
Success
Running on Jenkins
View as plain text
docker push shubhpatel2610/frontend-service
Shell Script
7 sec
docker push shubhpatel2610/auth-service
Shell Script
26 sec
docker push shubhpatel2610/admin-service
Shell Script
42 sec
docker push shubhpatel2610/user-service
Shell Script
32 sec
docker push shubhpatel2610/doctor-service
Shell Script
34 sec
docker push shubhpatel2610/email-service
Shell Script
7.2 sec
docker push shubhpatel2610/archive-service
Shell Script
7.1 sec
Jenkins 2.504.1


Build log [#17] - Jenkins
localhost:8080/job/CareConnect/17/pipeline-console/
Jenkins
ADMIN
log out
Dashboard > CareConnect > #17 > Pipeline Console
Build #17
Rebuild
Overview
Configure
Success 8 min 32 sec ago in 5 min 18 sec
Checkout SCM
Checkout
Clean & Build Spring Boot Services
Building Docker Images
Push Docker Images
Run Ansible Playbook
Post Actions
Success
Running on Jenkins
View as plain text
echo "$VAULT_PASSWORD" > vault_pass.txt ansible-playbook -i inventory.ini playbook.yaml --vault-${VAULT_PASSWORD}-file vault_pas...
Shell Script
1 min 26 sec
74 "deployment.apps/doctor-service created",
75 "deployment.apps/frontend-service created",
76 "horizontalpodautoscaler.autoscaling/auth-service-hpa created",
77 "horizontalpodautoscaler.autoscaling/user-service-hpa created",
78 "horizontalpodautoscaler.autoscaling/doctor-service-hpa created",
79 "horizontalpodautoscaler.autoscaling/admin-service-hpa created",
80 "horizontalpodautoscaler.autoscaling/frontend-service-hpa created",
81 "service/auth-service created",
82 "service/user-service created",
83 "service/admin-service created",
84 "service/doctor-service created",
85 "service/frontend-service created",
86 "cronjob.batch/archive-service created",
87 "deployment.apps/email-service created",
88 "ingress.networking.k8s.io/careconnect-ingress created",
89 "All resources have been applied successfully."
90]
Jenkins 2.504.1

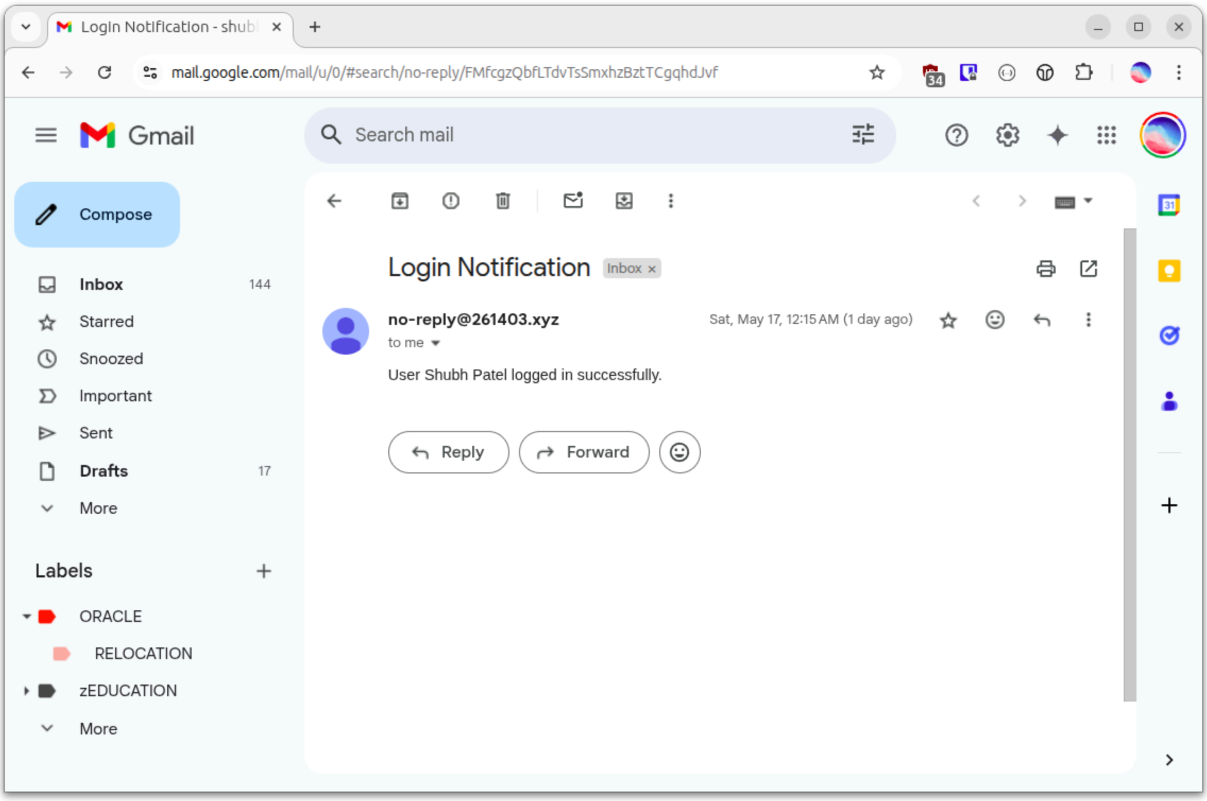
Logs display in elastic search



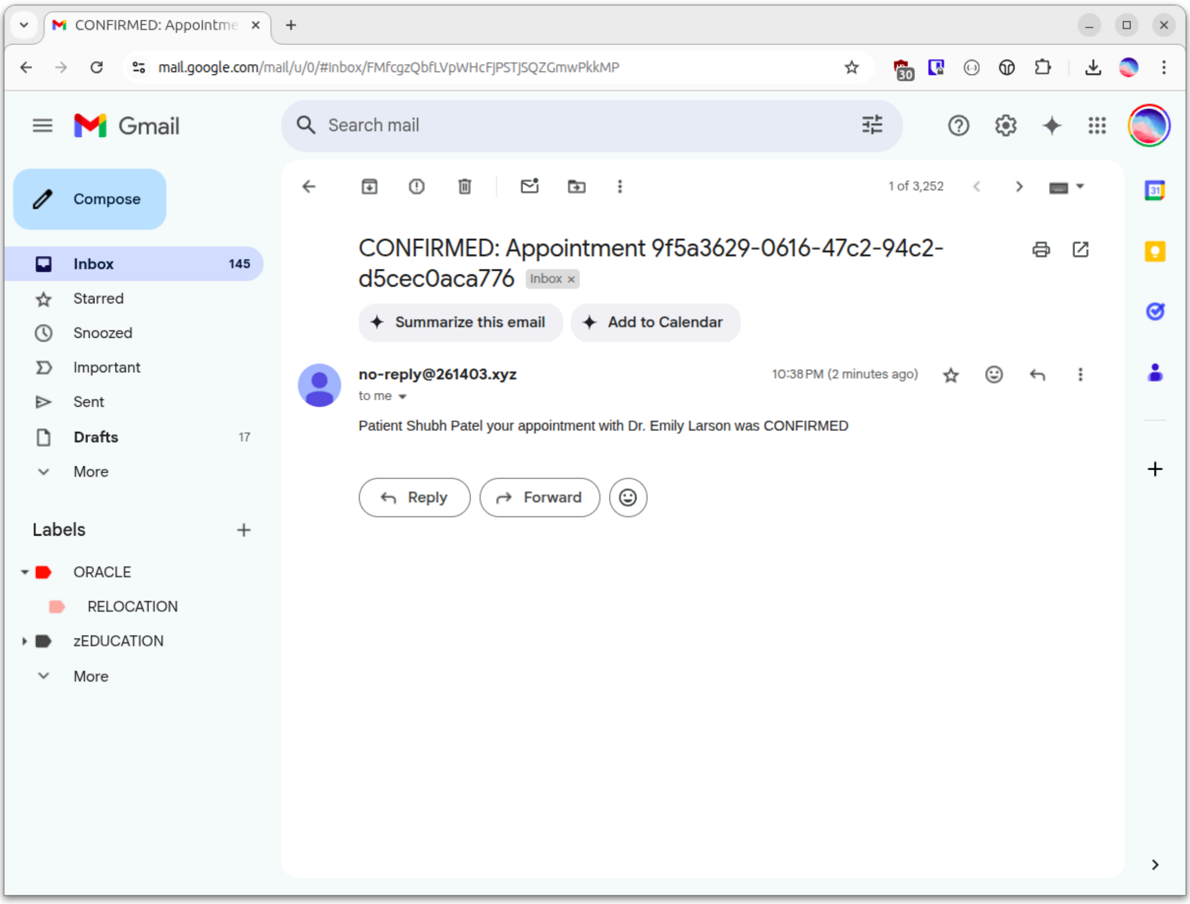
Email-service logs



Login Alert


Login Notification - shubh
mail.google.com/mail/u/0/#search/no-reply/FMfcgzQbfLTdvTsSmxhzBztTCgqhdJvf
Gmail
Search mail
Compose
Inbox 144
Starred
Snoozed
Important
Sent
Drafts 17
More
Labels
ORACLE
RELOCATION
zEDUCATION
More
Login Notification
Inbox
no-reply@261403.xyz
to me
Sat, May 17, 12:15 AM (1 day ago)
User Shubh Patel logged in successfully.
Reply
Forward

Confirmation email



Login page

User appointment booking

User appointments

Doctor appointments

Doctor Unavailability

Admin dashboard

7 Conclusion

CareConnect marks a significant achievement in developing a scalable, user-centric Doctor Appointment System, leveraging a microservices architecture and Kubernetes orchestration to deliver a robust and resilient platform. By implementing modular services such as authentication, user management, and appointment scheduling alongside a React frontend and Spring Boot backend, the project successfully streamlined healthcare appointment processes for patients and providers. Kubernetes played a pivotal role, enabling high availability through Horizontal Pod Autoscaling, efficient resource management via persistent storage, and seamless traffic routing with Ingress, as detailed in the repository's k8s configurations. Despite challenges like Kubernetes complexity and microservices coordination, the team's iterative approach and integration of tools like ELK Stack and RabbitMQ ensured a reliable, monitorable system. CareConnect's cloud-native design not only meets current demands but also lays a versatile foundation for future enhancements, such as telemedicine or AI-driven optimizations, positioning it as a forward-thinking solution in healthcare technology.

8 Recommendations

- **Enhance Monitoring**: Integrate Prometheus and Grafana.
- **Expand Features**: Add real-time chat or telemedicine.
- **Optimize Performance**: Implement caching (e.g., Redis).
- **Cloud Deployment**: Transition to AWS EKS or Google GKE.

9 References

- Kubernetes Documentation: https://kubernetes.io/docs/
- Spring Boot Documentation: https://spring.io/projects/spring-boot
- React Documentation: https://reactjs.org/docs/