

# SEZG583 Scalable Services

## Assignment: Microservices Development and Deployment

### Real-Time User Event Tracking and Analytics Platform

Course: SEZG583 Scalable Services | Submission: 10 November 2024

**Repository:** <https://github.com/wilp-bits-2024mt03053/scalable-services-assignment-1>

### Group Members

Balaji O M	2024mt03025
Balasubramaniyan	2024mt03053
Deep Pokala	2024mt03042
Jagriti Sharma	2024mt03116

### Table of Contents

1. Application Description
2. PART 1: DESIGN
  - System Architecture
  - Business Capabilities & Services
  - System Operations (Commands & Queries)
  - Service Collaboration
3. PART 2: IMPLEMENTATION
  - Technology Stack
  - Two Key Microservices
  - Service Independence
4. PART 3: DEPLOYMENT
  - Docker Containers - Separate per Service
  - Kubernetes Deployment Manifests
  - Minikube Cluster & Deployment
  - Kubernetes Dashboard
  - Scalability Demonstration: Pod Resilience & Self-Healing
5. Implementation & Verification
6. Snapshots & Verification

## 1. Application Description

### Real-Time User Event Tracking and Analytics Platform

This project implements a scalable, real-time data streaming pipeline designed to capture, process, and analyze user interaction events from a web application. It serves as a comprehensive example of a modern microservices-based architecture using containerization and orchestration.

#### Project Overview:

The system is designed as a multi-stage pipeline that captures user events from a web frontend, processes them in real-time, and stores them in a database for analytics. Each component in the services/ directory is a self-contained microservice with specific responsibility enabling independent development, scaling, and maintenance.

#### Key Features:

- Captures user interactions (clicks, page views, form submissions) from React frontend
- Custom tracking hook (`react-user-tracker`) batches events for efficient transmission
- Real-time event streaming using Apache Kafka
- Event validation and transformation through Python processor
- Persistent storage in PostgreSQL database
- RESTful API (FastAPI) for querying event data with filtering and pagination
- Database administration UI via Adminer
- Complete containerization with Docker - each service in separate container
- Orchestration with Kubernetes and Minikube

#### Data Flow:

1. User interactions on React Frontend App
2. Batched HTTP POST to Event Collector (FastAPI)
3. Events published to Kafka topic (`user-tracking-events`)
4. Event Processor (Python Kafka consumer) reads and validates events
5. Processed events inserted into PostgreSQL database
6. Events API provides query endpoint for analytics
7. Adminer provides database inspection and management

## 2. PART 1: DESIGN

### 2.1 System Architecture

The system follows a layered microservices architecture with clear separation of concerns. Each layer handles specific responsibilities and communicates through well-defined interfaces. The architecture progresses from individual services to a fully orchestrated deployment:

**Service-Level Design:** Each component in the services/ directory is a self-contained microservice with a specific responsibility (e.g., real-time-events-collector for ingestion, real-time-events-processor for processing). This separation of concerns allows for independent development, scaling, and maintenance.

**Dockerization:** Every service includes a Dockerfile that packages its source code, dependencies, and runtime into a lightweight, portable container image. This ensures that each service runs in an identical and isolated environment, whether on a local machine or in a cloud-based cluster.

**Local Development:** The docker-compose.yml file defines the entire multi-service application stack for local development. It orchestrates the containers, networks, and volumes, allowing you to spin up the whole system with a single command (make deploy).

**Kubernetes Deployment:** For a production-like environment, the project uses Kubernetes. The manifests in the kube/ directory define how each service is deployed, configured, and exposed. The deploy-minikube.sh script automates the process of building images and applying these manifests to a local Minikube cluster.

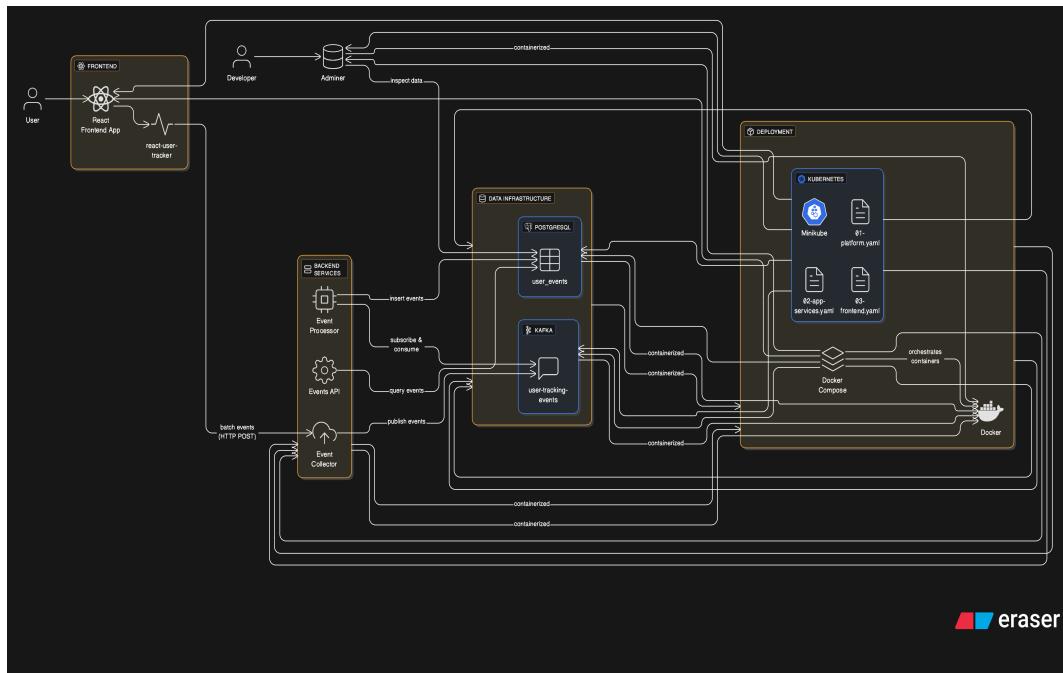


Figure 1: System Architecture Diagram

### 2.2 Business Capabilities & Services

Business Capability	Service	Technology
Event Capture	react-user-tracker	React/TypeScript
Event Ingestion	real-time-events-collector	FastAPI

Business Capability	Service	Technology
Stream Management	Kafka + Zookeeper	Message Broker
Event Processing	real-time-events-processor	Python
Data Storage	PostgreSQL	Database
Query API	real-time-events-service	FastAPI
Data Inspection	Adminer	Web UI

### 2.3 System Operations (Commands & Queries)

**Commands:** POST /collect, Kafka publish, INSERT user\_events

**Queries:** GET /events, SELECT from database

### 2.4 Service Collaboration

Source	Target	Protocol	Data Format
React	Collector	HTTP POST	JSON Array
Collector	Kafka	TCP	JSON
Processor	Kafka	TCP	JSON
Processor	PostgreSQL	SQL INSERT	Structured
API	PostgreSQL	SQL SELECT	Structured

### 3. PART 2: IMPLEMENTATION

#### 3.1 Technology Stack

Component	Technology	Purpose
Frontend	React/Next.js	User interface
Backend Services	Python FastAPI	API endpoints
Message Broker	Apache Kafka 7.0.1	Event streaming
Database	PostgreSQL 13	Persistence
Container	Docker	Isolation
Orchestration	Kubernetes/Minikube	Orchestration

#### 3.2 Two Key Microservices

- 1. Event Collector (real-time-events-collector)** - FastAPI service on port 8000, receives events and publishes to Kafka
- 2. Event Processor (real-time-events-processor)** - Python Kafka consumer, validates and inserts events into database

#### 3.3 Service Independence

Each service in services/ folder has own Dockerfile, requirements.txt, configuration via environment variables, and can be deployed independently

## 4. PART 3: DEPLOYMENT

### 4.1 Docker Containers - Separate per Service

#### Deployment Command:

make deploy

#### 8 Separate Services:

1. Zookeeper (port 2181)
2. Kafka (port 9092)
3. PostgreSQL (port 5432)
4. Adminer (port 8080)
5. Event Collector (port 8000)
6. Event Processor (bg)
7. Events API (port 8001)
8. Frontend (port 3000)

PS C:\Users\deepd\OneDrive\Desktop\scalable-services-assignment-1> docker ps --format "table {{.Names}}\t{{.Image}}\t{{.Status}}\t{{.Ports}}"				
NAME	IMAGE	STATUS	PORTS	
real-time-user-tracker-demo	real-time-user-tracker-demo:latest	Up About an hour	0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp	
real-time-events-processor	real-time-events-processor:latest	Up About an hour		
real-time-events-collector	real-time-events-collector:latest	Up About an hour	0.0.0.0:8000->8000/tcp, [::]:8000->8000/tcp	
adminer	adminer	Up About an hour	0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp	
real-time-events-service	real-time-events-service:latest	Up About an hour	0.0.0.0:8001->8001/tcp, [::]:8001->8001/tcp	
kafka	wurstmeister/kafka:latest	Up About an hour (healthy)	0.0.0.0:9092->9092/tcp, [::]:9092->9092/tcp	
postgres	postgres:13	Up About an hour (healthy)	0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp	
minikube	gcr.io/k8s-minikube/kicbase:v0.0.48	Up 3 hours	127.0.0.1:53547->22/tcp, 127.0.0.1:53548->2376/tcp, 127.0.0.1:53545->5000/tcp, 127.0.0.1:53546->8443/tcp, 127.0.0.1:53549->32443/tcp	

Figure 2: Docker Containers - docker ps Output

```
PS C:\Users\deepd\OneDrive\Desktop\scalable-services-assignment-1> kubectl apply -f ./k8s/scalable-services-assignment-1/
--> LegacyKeyValueFormat: "ENV key=value" should be used instead of legacy "ENV key value" format (line 18)
--> Deployment "real-time-user-tracker" should be updated...
--> Fetching Minikube IP and exporting for Kafka...
--> Applying 01-zookeeper.yaml...
namespaces/scalable-services configured
service/zookeeper created
statefulset.apps/zookeeper created
service/kafka created
statefulset.apps/kafka created
service/events-processor created
service/postgres created
statefulset.apps/postgres created
--> Applying 02-app-services.yaml...
deployment.apps/events-collector created
deployment.apps/events-processor created
service/events-processor created
deployment.apps/events-api created
service/adminer created
deployment.apps/frontend created
horizontalpodautoscaler.autoscaling/events-collector-hpa created
horizontalpodautoscaler.autoscaling/events-processor-hpa created
horizontalpodautoscaler.autoscaling/events-api-hpa created
--> Applying 03-frontend.yaml...
deployment.apps/frontend created
deployment.apps/frontend created
horizontalpodautoscaler.autoscaling/frontend-hpa created
=====
@ WAITING FOR ALL RESOURCES TO BE READY
=====
--> Waiting for StatefulSet "zookeeper" to be ready...
Horizontal pod auto-scaler: 1 new pods have been updated...
partitioned roll out complete: 1 new pods have been updated...
--> @ StatefulSet "zookeeper" is ready.
--> Waiting for StatefulSet "kafka" to be ready...
Horizontal pod auto-scaler: 1 new pods have been updated...
partitioned roll out complete: 1 new pods have been updated...
--> @ StatefulSet "kafka" is ready.
--> Waiting for StatefulSet "postgres" to be ready...
partitioned roll out complete: 1 new pods have been updated...
--> @ StatefulSet "postgres" is ready.
--> Waiting for Deployment "events-collector" to be ready...
deployment "events-collector" successfully rolled out
--> @ Deployment "events-collector" is ready.
--> Waiting for Deployment "events-processor" to be ready...
deployment "events-processor" successfully rolled out
--> @ Deployment "events-processor" is ready.
```

Figure 3: Services Startup Phase

```
=====
@ WAITING FOR ALL RESOURCES TO BE READY
=====
--> Waiting for StatefulSet 'cookeeper' to be ready...
Waiting for 1 pods to be ready...
partitioned roll out complete: 1 new pods have been updated...
--> ☑ StatefulSet 'zookeeper' is ready.
--> Waiting for StatefulSet 'kafka' to be ready...
Waiting for 1 pods to be ready...
partitioned roll out complete: 1 new pods have been updated...
--> ☑ StatefulSet 'kafka' is ready.
--> Waiting for StatefulSet 'postgres' to be ready...
Waiting for 1 pods to be ready...
partitioned roll out complete: 1 new pods have been updated...
--> ☑ StatefulSet 'postgres' is ready.
--> Waiting for Deployment 'events-collector' to be ready...
deployment "events-collector" successfully rolled out
--> ☑ Deployments 'events-collector' is ready.
--> Waiting for Deployment 'events-processor' to be ready...
deployment "events-processor" successfully rolled out
--> ☑ Deployment 'events-processor' is ready.
--> Waiting for Deployment 'events-api' to be ready...
deployment "events-api" successfully rolled out
--> ☑ Deployment 'events-api' is ready.
--> Waiting for deployment 'adminer' to be ready...
Waiting for deployment "adminer" rollout to finish: 0 of 1 updated replicas are available...
deployment "adminer" successfully rolled out
--> ☑ Deployment 'adminer' is ready.
--> Waiting for Deployment 'frontend' to be ready...
deployment "frontend" successfully rolled out
--> ☑ Deployment 'frontend' is ready.
--> All resources are ready!
--> Deployment to Minikube is complete!
-- Access Instructions:
Frontend App: minikube service frontend -n scalable-services
Adminer Tool: minikube service adminer -n scalable-services
Events API: minikube service events-api -n scalable-services --url
```

Figure 4: Services Ready - All Healthy

## 4.2 Kubernetes Deployment Manifests

**01-platform.yaml:** Zookeeper, Kafka, PostgreSQL StatefulSets

**02-app-services.yaml:** Collector, Processor, API Deployments with HPA

**03-frontend.yaml:** Frontend Deployment with LoadBalancer

## 4.3 Minikube Cluster & Deployment

### Deployment Steps:

1. minikube start --cpus=4 --memory=6144
2. kubectl create namespace scalable-services
3. ./deploy-minikube.sh deploy
4. kubectl get all -n scalable-services
5. minikube dashboard

```
deep_vayavaya@mnt:/Users/deepd/scalable-services-assignment-1$ kubectl get deployments -n scalable-services
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
adminer     1/1     1           1           83m
events-api  1/1     1           1           83m
events-collector 1/1     1           1           83m
events-processor 1/1     1           1           83m
frontend    1/1     1           1           83m
deep_vayavaya@mnt:/Users/deepd/scalable-services-assignment-1$ kubectl get svc -n scalable-services
NAME        TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
adminer     NodePort   10.98.169.33 <none>        8080:30080/TCP   84m
events-api  NodePort   10.106.130.81 <none>        8001:30081/TCP   84m
events-collector  ClusterIP  10.103.226.112 <none>        8000/TCP         84m
frontend    NodePort   10.103.66.176 <none>        3000:30030/TCP   84m
kafka       NodePort   10.98.128.96 <none>        9092:32191/TCP,9094:30092/TCP 84m
postgres    ClusterIP  10.101.83.126 <none>        5432/TCP         84m
zookeeper   ClusterIP  10.96.24.29  <none>        2181/TCP         84m
deep_vayavaya@mnt:/Users/deepd/scalable-services-assignment-1$ kubectl get pods -n scalable-services
NAME                  READY   STATUS    RESTARTS   AGE
adminer-84f4d644d5-xkk85  1/1    Running   0          84m
events-api-6dff7b9fb-d-q9t8m  1/1    Running   0          84m
events-collector-6499774785-5b87c  1/1    Running   1 (83m ago)  84m
events-processor-5b79f74b64-vnpvxh  1/1    Running   1 (76m ago)  84m
frontend-7b5b9978d7-42tf  1/1    Running   0          84m
kafka-0                1/1    Running   0          84m
postgres-0              1/1    Running   0          84m
zookeeper-0             1/1    Running   0          84m
deep_vayavaya@mnt:/Users/deepd/scalable-services-assignment-1$
```

Figure 5: kubectl get all - Kubernetes Resources

The screenshot shows the Kubernetes dashboard interface. The top navigation bar includes the Kubernetes logo, the namespace 'scalable-services' dropdown, a search bar, and a '+' button. The main area is titled 'Workloads'.

**Deployments:**

Name	Images	Labels	Pods	Created
frontend	real-time-user-tracker-demo:latest	-	1 / 1	an hour ago
adminer	adminer:latest	-	1 / 1	an hour ago
events-api	real-time-events-service:latest	-	1 / 1	an hour ago
events-collector	real-time-events-collector:latest	-	1 / 1	an hour ago
events-processor	real-time-events-processor:latest	-	1 / 1	an hour ago

**Pods:**

Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
frontend-7b5b9978d7-42tf	real-time-user-tracker-demo:latest	app: frontend pod-template-hash: 7b5b9978d7	minikube	Running	0	-	-	an hour ago

Figure 6: Kubernetes Deployments - All Running

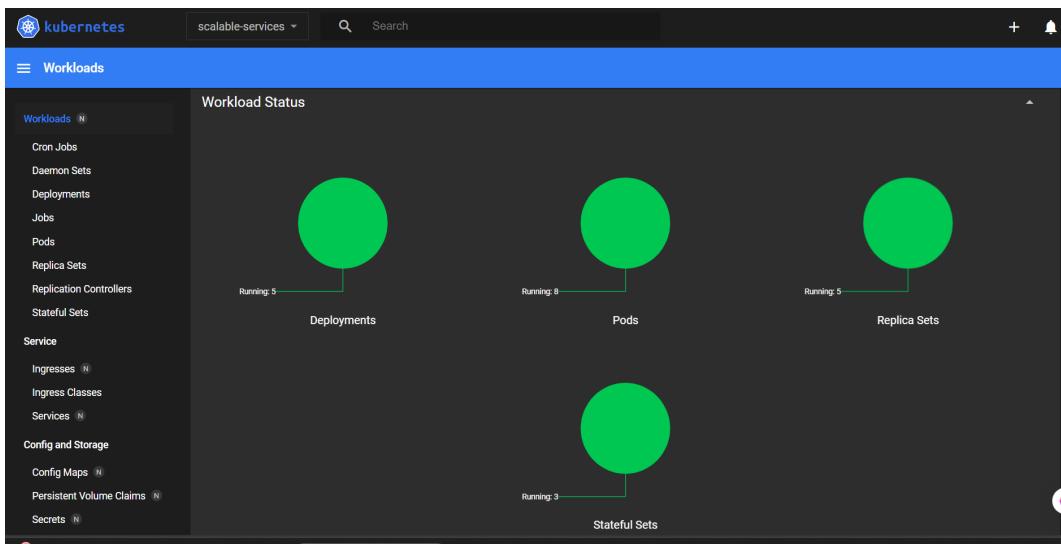


Figure 7: Kubernetes Workloads - Pods & Services

```
deep_vayavaya@Vayavaya:/mnt/c/Users/deepd/scalable-services-assignment-1$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

deep_vayavaya@Vayavaya:/mnt/c/Users/deepd/scalable-services-assignment-1$
```

Figure 8: Minikube Status

## 4.4 Kubernetes Dashboard

Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
dashboard-metrics-scraper-77bf4d6c4c-zh7wz	docker.io/kubernetes-suit/metrics-scraper:v1.0.0-rc2+e2367604998707604a0433efc2d56fb9520b1982b22a29356092ce20698765c	k8s-app: dashboard-metrics-scraper	minikube	Running	0	-	-	10 minutes ago
kubernetes-dashboard-855c9754f9-cjfx	docker.io/kubernetes-suit/dashboard:v2.7.0@sha256:2e500d29e9df5f4a086b908eb8df7ecac57d2ab9d65b24f588bd449841ef93	gcp-auth-skip-secret: true	minikube	Running	0	-	-	10 minutes ago
frontend-7b5b9978d7-42tft	real-time-user-tracker:demo:latest	app: frontend	minikube	Running	0	-	-	an hour ago
events-processor-5b7974b64-vnpnx	real-time-events-processor:latest	app: events-process	minikube	Running	1	-	-	an hour ago

Figure 9: Kubernetes Dashboard - Cluster Overview

## 4.5 Scalability Demonstration: Pod Resilience & Self-Healing

Kubernetes demonstrates its self-healing capabilities and scalability through its automatic pod recovery mechanism. This section verifies that the platform maintains service availability even when individual pods fail.

### Demonstration Steps:

#### Step 1: List Running Pods

Command: `kubectl get pods -n scalable-services`

This lists all running pods in the scalable-services namespace, showing the initial deployment state with all services running.

#### Step 2: Kill a Pod

Command: `kubectl delete pod <pod-name> -n scalable-services`

This forcibly terminates a specific pod (e.g., a frontend or API pod). In a real-world scenario, this simulates a pod crash due to resource exhaustion, application error, or node failure.

#### Step 3: List Pods Again

Command: `kubectl get pods -n scalable-services`

By running this command immediately after deletion, we observe that Kubernetes automatically creates a replacement pod. The Deployment controller detects that the desired replica count has been violated and instantly schedules a new pod to restore the desired state.

### Key Observations:

- Self-Healing:** Kubernetes automatically replaces failed pods without manual intervention.
- Replica Management:** The Deployment maintains the specified number of replicas (e.g., 3 replicas for frontend).
- Continuous Availability:** Other pods continue to serve requests while the replacement pod starts.
- Scalability:** This mechanism allows the system to scale horizontally by increasing replica counts, and Kubernetes automatically distributes them across nodes.

```
deep_vayavaya@mntc:/Users/deepd/scalable-services-assignment-1$ kubectl get pods -n scalable-services
NAME                      READY   STATUS    RESTARTS   AGE
adminer-84f4d644d5-xkk85   1/1    Running   0          3d2h
events-api-6dff7b9fb9-q9t8m 1/1    Running   0          3d2h
events-collector-6499774785-5b87c 1/1    Running   1 (3d2h ago) 3d2h
events-processor-5b79f74b64-vnpvh 1/1    Running   1 (3d2h ago) 3d2h
frontend-7b5b9978d7-42tf7   1/1    Running   0          3d2h
kafka-0                     1/1    Running   0          3d2h
postgres-0                  1/1    Running   0          3d2h
zookeeper-0                 1/1    Running   0          3d2h
deep_vayavaya@mntc:/Users/deepd/scalable-services-assignment-1$
```

Figure 10: Pod Scaling Demonstration - Killing and Replacing Pods

```
bash: pod name: no such file or directory
deep_vayavaya@mntc:/Users/deepd/scalable-services-assignment-1$ kubectl delete pod adminer-84f4d644d5-xkk85 -n scalable-services
pod "adminer-84f4d644d5-xkk85" deleted from scalable-services namespace
deep_vayavaya@mntc:/Users/deepd/scalable-services-assignment-1$ kubectl get pods -n scalable-services
NAME                      READY   STATUS    RESTARTS   AGE
adminer-84f4d644d5-qndbl   1/1    Running   0          21s
events-api-6dff7b9fb9-q9t8m 1/1    Running   0          3d2h
events-collector-6499774785-5b87c 1/1    Running   1 (3d2h ago) 3d2h
events-processor-5b79f74b64-vnpvh 1/1    Running   1 (3d2h ago) 3d2h
frontend-7b5b9978d7-42tf7   1/1    Running   0          3d2h
kafka-0                     1/1    Running   0          3d2h
postgres-0                  1/1    Running   0          3d2h
zookeeper-0                 1/1    Running   0          3d2h
deep_vayavaya@mntc:/Users/deepd/scalable-services-assignment-1$
```

Figure 11: Autoscaling Pods - Horizontal Pod Autoscaler in Action

## 5. Implementation & Verification

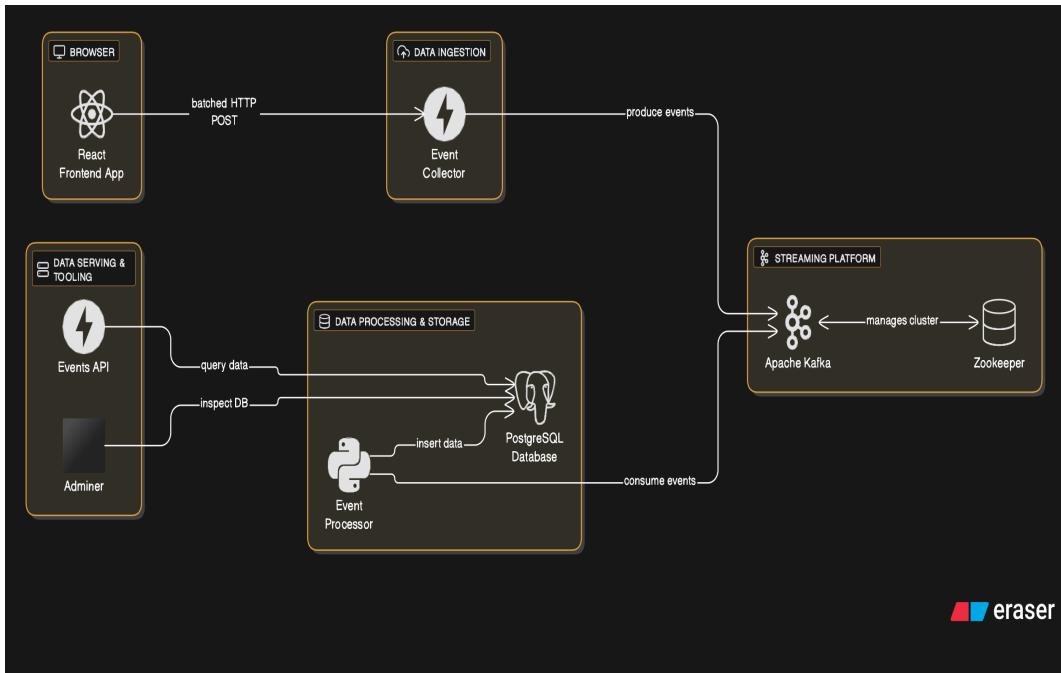


Figure 12: Data Flow Pipeline

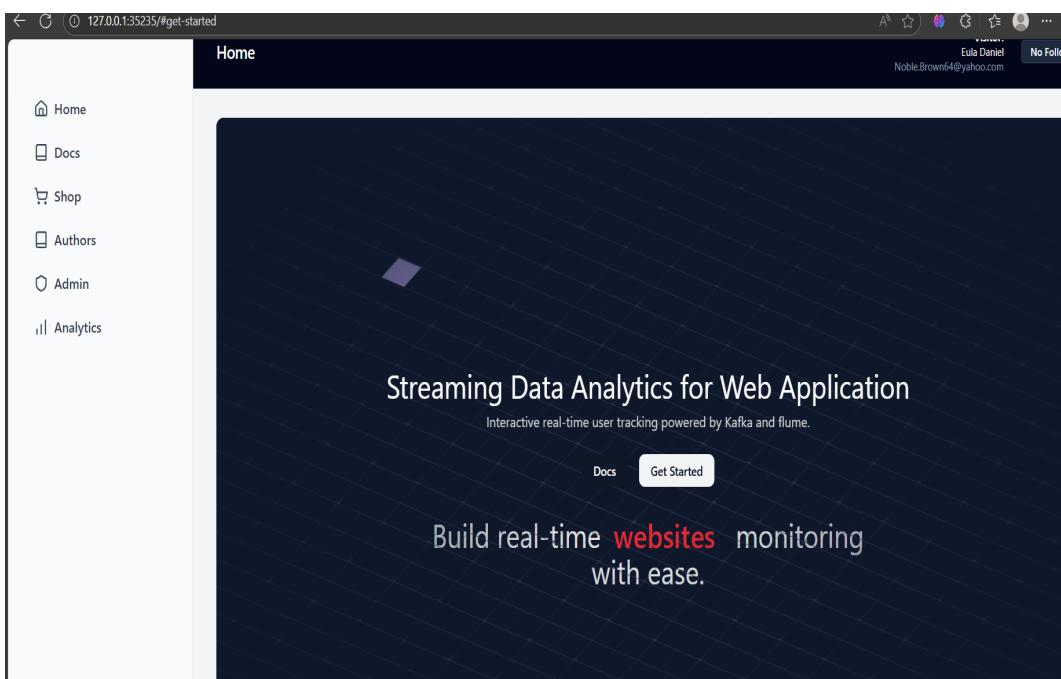


Figure 13: Frontend Application UI

```
✓ Deployment to Minikube is complete!

🌐 Access Instructions:
-----
Frontend App: minikube service frontend -n scalable-services
Adminer Tool: minikube service adminer -n scalable-services
Events API: minikube service events-api -n scalable-services --url
-----

deep_vayavaya@mntc:/Users/deepd/scalable-services-assignment-1$ minikube service frontend -n scalable-services
NAME        NAMESPACE   TARGET PORT   URL
frontend   scalable-services  3000          http://192.168.49.2:30030

Starting tunnel for service frontend...
NAME        NAMESPACE   TARGET PORT   URL
frontend   scalable-services          http://127.0.0.1:35235

Starting tunnel for service frontend...
Opening service scalable-services/frontend in default browser...
http://127.0.0.1:35235
! Because you are using a Docker driver on linux, the terminal needs to be open to run it.
```

Figure 14: Deployed Frontend - Live

Language: English

PostgreSQL » postgres » tracking\_db » Schema: public

user Logout

**Adminer 5.4.1**

Schema: public

DB: tracking\_db Schema: public

Alter schema Database schema Routines Sequences User types

Tables and views

Search data in tables (1)

LIKE %%

Table Engine Collation Data Length? Index Length? Data Free Auto Increment Rows? Comment?

user\_events table en\_US.utf8 81,920 16,384 ? ? ~ 62

1 in total

Selected (0) Move to other database (0)

Vacuum Optimize Truncate Drop public Move

Create table Create view

Routines

Create procedure Create function

Sequences

Create sequence

User types

Create type

Figure 15: Adminer Database Tool

```
c:\> stopped tunnel for service frontend.
deep_yayavaya@Yayava:/mnt/c/Users/deepd/scalable-services-assignment-1$ minikube service adminer -n scalable-services
NAMEPORTURL
scalable-servicesadminer8080http://192.168.49.2:30080

Starting tunnel for service adminer./
NAMEPORTURL
scalable-servicesadminer34761http://127.0.0.1:34761

Starting tunnel for service adminer.
Opening service scalable-services/adminer in default browser...
http://127.0.0.1:34761
Because you are using a Docker driver on linux, the terminal needs to be open to run it.
```

Figure 16: Database Schema & Tables

```

Deep_Vayavya@Deep_Vayavya:/mnt/c/Users/deepd/scalable-services-assignment-1$ kubectl get all -n scalable-services
NAME                         READY   STATUS    RESTARTS   AGE
pod/adminer                   1/1     Running   0          61m
pod/events-api                1/1     Running   0          61m
pod/events-collector          1/1     Running   1 (60m ago) 61m
pod/events-processor          1/1     Running   1 (53m ago) 61m
pod/frontend                  1/1     Running   0          61m
pod/kafka-0                   1/1     Running   0          61m
pod/postgres-0                1/1     Running   0          61m
pod/zookeeper-0              1/1     Running   0          61m

NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)      AGE
service/adminer  NodePort   10.98.169.33 <none>       8080:30880/TCP 61m
service/events-api  NodePort   10.106.130.81 <none>       8081:30881/TCP 61m
service/events-collector  ClusterIP  10.103.226.112 <none>       8080/TCP      61m
service/frontend           NodePort   10.103.66.176 <none>       3000:30030/TCP 61m
service/kafka               NodePort   10.98.128.96 <none>       9092:32191/TCP,9094:30092/TCP 61m
service/postgres            ClusterIP  10.101.83.126 <none>       5432/TCP      61m
service/zookeeper           ClusterIP  10.96.24.29  <none>       2181/TCP      61m

NAME          READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/adminer  1/1    1          1          61m
deployment.apps/events-api 1/1    1          1          61m
deployment.apps/events-collector 1/1    1          1          61m
deployment.apps/events-processor 1/1    1          1          61m
deployment.apps/frontend    1/1    1          1          61m

NAME          DESIRED  CURRENT  READY  AGE
replicaset.apps/adminer-84f4d644d5  1        1        1        61m
replicaset.apps/events-api-6dff7b9fbfd  1        1        1        61m
replicaset.apps/events-collector-6499774785  1        1        1        61m
replicaset.apps/events-processor-5b79f74b64  1        1        1        61m
replicaset.apps/frontend-7b5b9978d7  1        1        1        61m

NAME          READY  AGE
statefulset.apps/kafka  1/1    61m
statefulset.apps/postgres  1/1    61m
statefulset.apps/zookeeper  1/1    61m

NAME          REFERENCE  TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
horizontalpodautoscaler.autoscaling/events-api-hpa  Deployment/events-api  cpu: <unknown>/50% 1        5        1        61m
horizontalpodautoscaler.autoscaling/events-collector-hpa  Deployment/events-collector  cpu: <unknown>/50% 1        5        1        61m
horizontalpodautoscaler.autoscaling/events-processor-hpa  Deployment/events-processor  cpu: <unknown>/50% 1        5        1        61m
horizontalpodautoscaler.autoscaling/frontend-hpa  Deployment/frontend  cpu: <unknown>/50% 1        5        1        61m

Deep_Vayavya@Deep_Vayavya:/mnt/c/Users/deepd/scalable-services-assignment-1$
```

Figure 17: Kubernetes Services Running