Document Clustering System with Docker

Technical Mathematics for Big Data

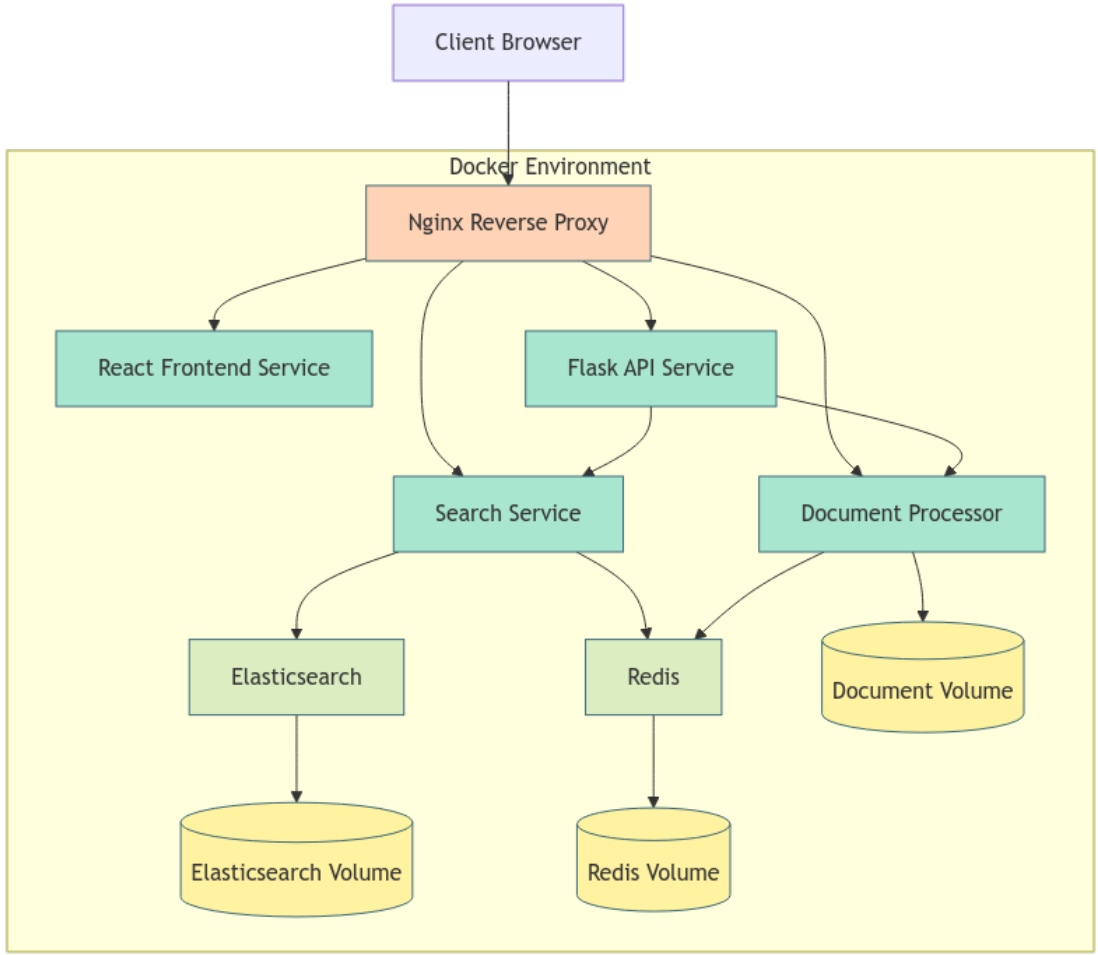
Oyedotun Oluwasegun Michael (#123168)
Silvia Mastracci (#123177)
Oleksandr Solovei (#126784)

January 16, 2025

Project Overview

- Document Clustering System built with microservices
- Containerized solution using Docker technology

- React-based UI
- Flask microservices
- Elasticsearch engine
- Document analysis system



Docker Setup and Operations

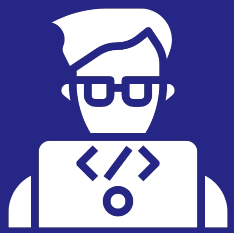
Docker Compose Config

```
services:
  nginx:
    image: nginx:alpine
    ports:
      - "4321:80"
  frontend:
    build: ./frontend
    expose:
      - "3000"
  api:
    build: ./api
    expose:
      - "8000"
  document-processor: (...)
```

Common Commands

```
# Build and start
docker-compose up --build

# Stop services
docker-compose down

# View logs
docker-compose logs -f

# Rebuild specific service
docker-compose build svc-name

# Show running containers
docker-compose ps
```

Docker Benefits



For Development

- **Consistent Environment**
  no "works on my machine"
- **Isolated Deps**
  no version conflicts
- **Rapid Dev Cycle**
  fast startup, easy rollbacks

For Performance

- **Resource Efficiency**
  lightweight architecture
- **Scalability**
  horizontal scaling
- **Maintenance**
  simple updates, min downtime

For Security

- **Vulnerability Management**
  image scanning, regular updates
- **Instance Isolation**
  for processes and networks
- **Security Features**
  access & resource limits

Core Components of Docker

Docker Client & Server (Engine): The client sends commands to the server (Docker Engine), which builds, runs, and manages containers.
Docker Images: A Docker image is a blueprint for creating containers, built using a Dockerfile.
Docker Containers: A Docker container is a running instance of a Docker image, providing isolated environments for applications.
Docker Registries: A Docker registry stores and distributes Docker images. Examples include Docker Hub and private registries.

Using Docker for Clustering

structure

```
clustering-project/
 app/   # Source code for both document and image clustering
    document_clustering/    # Document clustering code
       main.py           # Entry point for document clustering
       clustering.py  # Core clustering logic (e.g., K-means)
       utils.py      # Text preprocessing (tokenization, etc)
       requirements.txt    # Document clustering dependencies
    image_clustering/       # Image clustering code
       main.py           # Entry point for image clustering
       clustering.py  # Core clustering logic (e.g., DBSCAN)
       utils.py        # Image preprocessing (resizing, CNN)
       requirements.txt  # Image clustering dependencies (tens
 data/                        # Stores input and output files
    documents/                # Raw text documents
    images/                   # Raw image files
```

Using Docker for Clustering

```
   processed/  # Processed output (clusters, visualizations)
Dockerfile   # Configuration for containerization
docker-compose.yml  # Multi-container setup for both clusteri
README.md                    # Project documentation
requirements.txt   # Common dependencies (e.g.,numpy,pandas)
```

- **‘app/‘**: Contains source code for both tasks:
  - **‘document clustering/‘**: Includes Python scripts like `main.py`, `clustering.py` (e.g., K-means), and `utils.py` (text preprocessing). Dependencies include `scikit-learn`, `nltk`, and `spaCy`.
  - **‘image clustering/‘**: Includes scripts like `main.py`, `clustering.py`, and `utils.py` (image preprocessing). Dependencies include `opencv`, `PIL`, and `tensorflow`.
- **‘data/‘**: Stores input and output data:
  - **‘documents/‘**, **‘images/‘** (raw files) and **‘processed/‘** (output from clustering).

Dockerfile Components for Document and Image Clustering

- **Base Image**:
  `FROM python:3.8-slim`
  A lightweight Python image to start from for both document and image clustering.
- **Working Directory**:
  `WORKDIR /app`
  Defines the working directory inside the container.
- **Copy Application Files**:
  `COPY . /app`
  Copies project files (document_clustering/ or image_clustering/) into the container.
- **Install Dependencies**:
  `RUN pip install -r requirements.txt`

Dockerfile Components for Document and Image Clustering

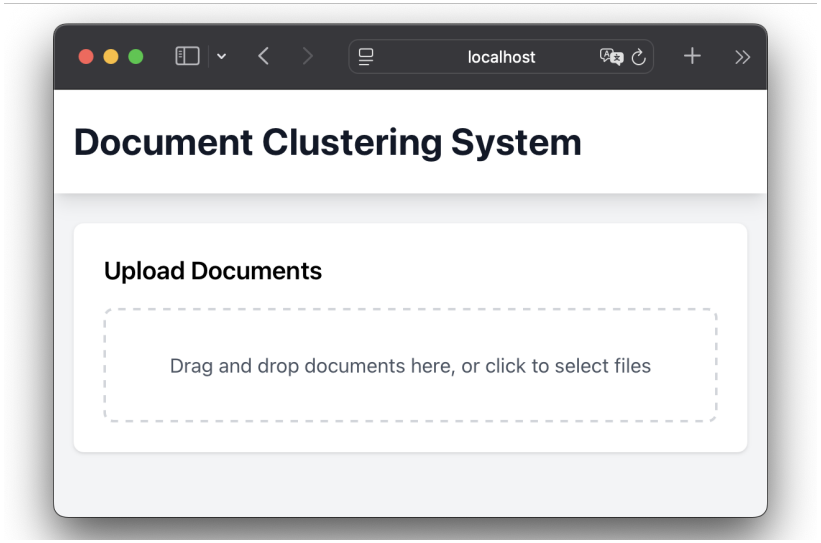
Installs necessary dependencies for the respective clustering task (e.g., `scikit-learn` for document clustering or `opencv`, `tensorflow` for image clustering).

- **Expose Ports (Optional)**:
  `EXPOSE 8000`
  If exposing a web app or API, define the port to be accessed outside the container.
- **Run the Application**:
  Document Clustering: `CMD ["python", "document_clustering/main.py"]`
  Image Clustering: `CMD ["python", "image_clustering/main.py"]`
  Executes the clustering application when the container starts.

Project Implementation and System Features

- **Multi-stage builds**
  - Optimized image sizes
  - Reduced attack surface
- **Docker Compose**
  - Service orchestration
  - Environment configuration
  - Network management
- **Volume Management**
  - Persistent data storage
  - Efficient data sharing

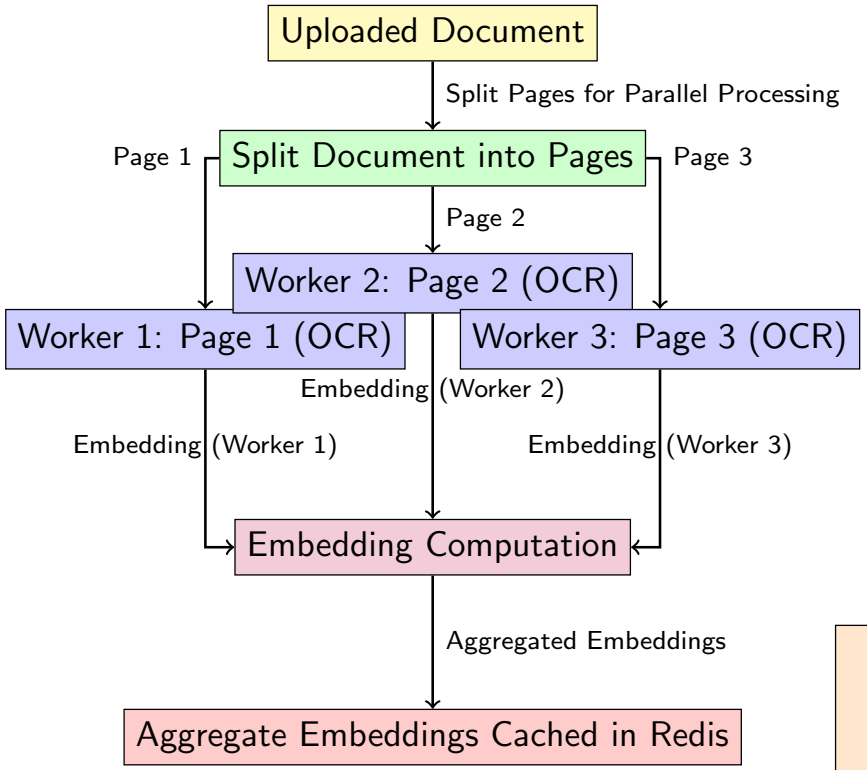
		Name	Tag	Image ID
☐	●	redis	alpine	1bf97f21f01b
☐	●	docker.elastic.co/elasticsearch/elasticsearch	7.17.9	59b37f77bd8b
☐	●	nginx	alpine	814a8e88df97
☐	●	docker-project-search	latest	4f5fbb64e1de
☐	●	docker-project-document-processor	latest	d8329a9c19bc
☐	●	docker-project-api	latest	1806fbbc4c1a



- **Single Port Access**
  - All services through one port
  - Nginx reverse proxy
- **Monitoring**
  - Health checks
  - Automated recovery
- **Data Management**
  - Elasticsearch integration
  - Redis caching

Performance Optimization with Parallelization

Document Processing Flowchart

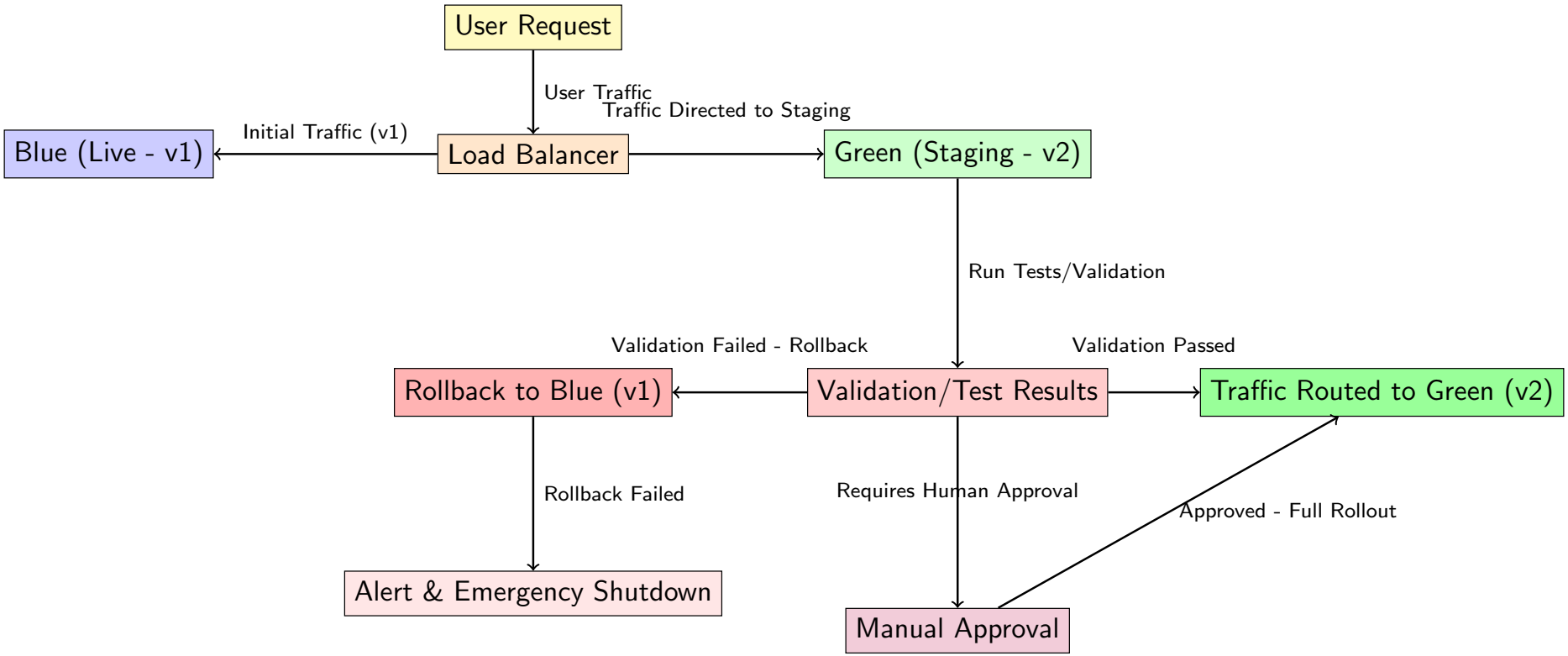


Note: Redis stores computed page embeddings for faster retrieval and parallel processing. It enables efficient distributed task communication.

Future Enhancements

- Implement `Celery` + Redis for distributed task queues.
- Add more workers to scale up parallel computations.

Cloud Deployment & Blue-Green Strategy



Key Benefits:

- Zero-downtime upgrades.
- Easy rollback if staging fails.
- Safer deployments with validation.

Cloud Development Blue-Green Strategy

Cloud Platforms:

- AWS ECS/EKS: Docker container orchestration.
- Azure AKS: Managed Kubernetes clusters.
- DigitalOcean App Platform: Simplified deployments.

Next Steps:

- Test load balancer configuration locally with Nginx.

Migration to Kubernetes

Why Kubernetes?

- Self-healing, auto-scaling, and automated rollouts.
- Handles complex deployments with better orchestration.

Migration Plan

- Convert `docker-compose.yml` to Kubernetes manifests using:

  `kubectl apply -f deployment.yml`
- Use Helm charts for templating reusable manifests.

Key Kubernetes Components:

Component	Purpose
Pods	Smallest deployable unit containing containers.
Deployments	Manage replica sets of pods to ensure availability and updates.
Services	Expose pods to external traffic and other components.

Conclusion

Thank You

Document Clustering System
Docker-based Microservices Architecture

Questions?