AI VOID FIELD GUIDE

Technical Case Studies

In-depth technical case studies exploring real-world architecture decisions, implementation challenges, and engineering solutions from production systems.

AI VOID

2026-06-03 15:31 IST

Contents

CHAPTER 01

EuroDocs: Lessons from Self-Hosting a Kubernetes-Based Collaboration Platform: Technical Case Study

Executive Summary

This case study details the journey of "EuroDocs," a fictional but realistic European technology company, in developing and self-hosting a collaborative document editing platform designed as a privacy-focused alternative to mainstream solutions like Google Docs. Driven by stringent European data sovereignty regulations and a commitment to open-source principles, EuroDocs embarked on a challenging project to build a robust, scalable, and secure platform from the ground up, leveraging Kubernetes as its core infrastructure. This document outlines the architectural decisions, key implementation hurdles, operational complexities, and the invaluable lessons learned from taking a complex, stateful application from development to production within a self-managed cloud-native environment.

Project Genesis: A European Vision for Data Sovereignty

The impetus for EuroDocs stemmed from a growing demand for digital collaboration tools that explicitly adhered to European data protection standards, particularly GDPR. Many existing solutions, while feature-rich, often involved data processing outside the EU or relied on proprietary infrastructure that lacked transparency. EuroDocs aimed to provide a secure, high-performance, and fully auditable alternative, ensuring all user data remained within EU borders under strict control.

Initial Constraints:

- **Data Residency:** Absolute requirement for all data to reside and be processed exclusively within the European Union.
- **Security & Privacy:** End-to-end encryption, strict access controls, and a privacy-by-design approach.

- **Real-time Collaboration:** Seamless, low-latency co-editing experience comparable to industry leaders.
- **Scalability:** Ability to support thousands of concurrent users and millions of documents.
- **Cost-Efficiency:** Optimize infrastructure costs by avoiding vendor lock-in and leveraging open-source components.
- **Operational Control:** Full control over the entire software and infrastructure stack.

Core Requirements and Design Principles

The EuroDocs platform needed to support core document functionalities while meeting stringent non-functional requirements.

Functional Requirements:

- **Document Creation & Editing:** Rich text editing, formatting, image embedding.
- **Real-time Co-editing:** Multiple users editing the same document concurrently with immediate synchronization.
- **Version History:** Automatic and manual document versioning with rollback capabilities.
- **Access Control:** Granular permissions for documents and folders.
- **Sharing & Collaboration:** Secure document sharing, commenting, and review workflows.
- **Search:** Fast and accurate full-text search across all documents.

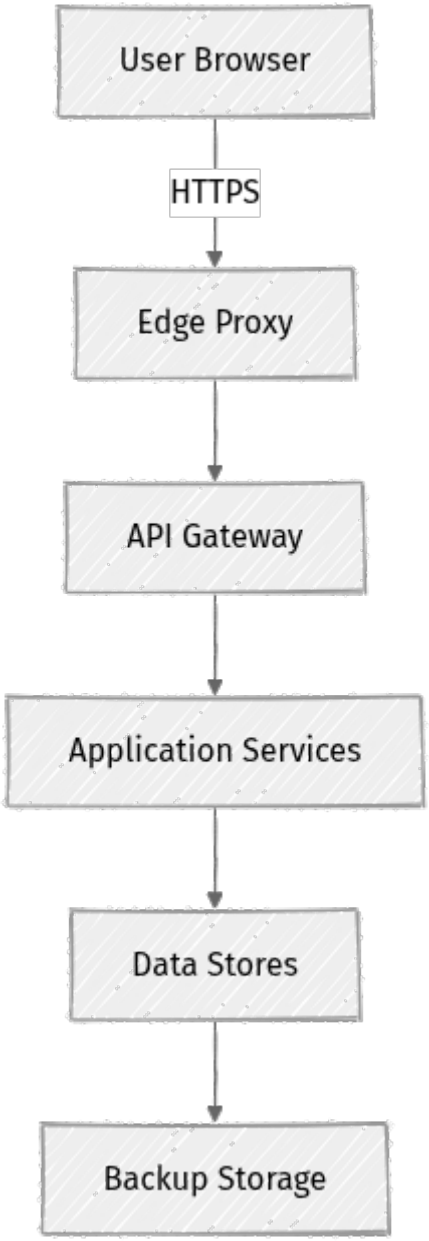
Non-Functional Requirements:

- **Availability:** Target 99.9% uptime.
- **Performance:** <100ms latency for real-time edits, <500ms for document load.
- **Scalability:** Horizontal scaling for both application services and data storage.
- **Security:** Robust authentication, authorization, data encryption (at rest and in transit), regular security audits.
- **Maintainability:** Modular architecture, automated deployments, comprehensive monitoring.

- **Compliance:** Full GDPR and relevant EU data protection regulation adherence.

Architectural Blueprint: Cloud-Native Collaboration

The decision to adopt a microservices architecture orchestrated by Kubernetes was central to meeting EuroDocs' scalability, resilience, and operational control requirements. This approach allowed for independent development, deployment, and scaling of individual services.



Key Architectural Components:

- **Kubernetes Cluster:** Self-hosted on a bare-metal or private cloud infrastructure within the EU. Managed with tools like Kubeadm and later Rancher for simplified operations.
- **Load Balancer & Ingress:** Nginx Ingress Controller exposed via a cloud provider's Load Balancer (or HAProxy for bare-metal).
- **API Gateway:** Routes HTTP requests to appropriate microservices, handles authentication pre-checks.
- **WebSocket Gateway:** Manages persistent WebSocket connections for real-time collaboration.
- **Microservices:**
    - **Authentication Service:** Handles user registration, login, token management (JWT).
    - **Document Service:** Manages document metadata, permissions, versioning, and storage interactions.
    - **Collaboration Service:** The core of real-time editing, using Operational Transformation (OT) or Conflict-free Replicated Data Types (CRDTs).
    - **Search Service:** Indexes document content and metadata, powered by Elasticsearch.
- **Data Stores:**
    - **PostgreSQL:** Primary relational database for user data, document metadata, and permissions.
    - **MinIO (S3-compatible Object Storage):** For storing document binaries and large assets.
    - **Redis:** High-performance caching for frequently accessed data and WebSocket session management.
    - **Elasticsearch:** For full-text search indexing.

Key Implementation Areas

Real-time Collaboration Engine

Implementing real-time co-editing was the most complex technical challenge. After evaluating both Operational Transformation (OT) and Conflict-free Replicated Data Types (CRDTs), EuroDocs opted for a CRDT-based approach due to its inherent eventual consistency and simpler merge logic for distributed systems.

- **Technology Stack:**
    - **Backend:** Golang microservice for the Collaboration Service.
    - **CRDT Library:** Yjs (implemented in JavaScript, adapted for backend logic).
    - **Communication:** WebSockets for bidirectional, low-latency communication.
    - **State Management:** Redis Pub/Sub for broadcasting changes across instances, PostgreSQL for persisting document states and history.

Conceptual Flow for Real-time Editing:

1. User makes an edit in the browser.
2. Client-side Yjs generates a CRDT update.
3. Update sent via WebSocket to the Collaboration Service.
4. Collaboration Service applies the update to its local CRDT document replica.
5. Update is broadcasted via Redis Pub/Sub to all other connected Collaboration Service instances and directly to other clients editing the same document.
6. Other clients receive the update and apply it to their local Yjs document, updating the UI.
7. Periodically, the full document state (or a snapshot of changes) is persisted to PostgreSQL for durability and versioning.

Document Storage and Versioning

Documents were stored in MinIO, with metadata and version pointers in PostgreSQL.

- **MinIO:** Provided S3-compatible object storage, crucial for large document binaries, offering high availability and scalability within the self-hosted environment.

- **PostgreSQL:** Stored document metadata (title, owner, permissions), version history (pointers to MinIO objects), and CRDT-specific state for recovery.
- **Versioning:** Each save or significant CRDT state persistence created a new version. Old versions were retained based on a configurable retention policy, allowing users to revert to previous states.

Deployment Automation with Kubernetes

A GitOps approach was adopted for managing Kubernetes deployments, ensuring infrastructure and application configurations were version-controlled and auditable.

- **CI/CD Pipeline:** GitLab CI/CD was used to automate:
    1. Code commit triggers build and test.
    2. Successful tests lead to Docker image creation and push to a private registry.
    3. Helm charts, stored in a Git repository, define application deployments.
    4. Argo CD (a GitOps controller) continuously monitors the Helm chart repository and the Kubernetes cluster, automatically synchronizing any divergences.

Example Kubernetes Deployment (simplified):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: collab-service
  labels:
    app: collab-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: collab-service
  template:
    metadata:
      labels:
        app: collab-service
    spec:
      containers:
      - name: collab-service
        image: eurodocs/collab-service:1.2.0
        ports:
        - containerPort: 8080
        env:
        - name: REDIS_HOST
          value: "redis-master"
```

```
        - name: PG_HOST
          value: "postgresql"
        resources:
          requests:
            cpu: "200m"
            memory: "512Mi"
          limits:
            cpu: "1000m"
            memory: "1Gi"
```

Scaling and Performance Optimization

To handle anticipated load, several strategies were employed:

- **Horizontal Pod Autoscaling (HPA):** Configured for microservices based on CPU utilization and custom metrics (e.g., WebSocket connections per Collaboration Service instance).
- **Database Sharding (Planned):** While initial PostgreSQL deployment was a single instance with read replicas, sharding was planned for massive scale-out of document metadata.
- **Caching:** Extensive use of Redis for session management, frequently accessed document parts, and user permissions to reduce database load.
- **Optimized CRDT Sync:** Implemented delta-based synchronization for Yjs updates, sending only necessary changes over WebSockets, minimizing network traffic.
- **Load Testing:** Regular load tests using tools like k6 and Locust simulated thousands of concurrent users to identify bottlenecks and validate scaling configurations.

Ensuring Reliability and Data Sovereignty

Reliability and data sovereignty were non-negotiable.

- **High Availability:**
  - Kubernetes control plane deployed in a highly available configuration (3 masters).
  - Worker nodes spread across different availability zones (physical data centers in the EU).
  - Stateful services (PostgreSQL, Elasticsearch, MinIO) deployed with replication and failover mechanisms using Kubernetes StatefulSets and operators (e.g., Crunchy Data PostgreSQL Operator, Elasticsearch Operator).
- **Backup and Disaster Recovery:**
  - Automated daily backups of all persistent volumes (PostgreSQL, Elasticsearch, MinIO) to an immutable, geographically separate EU-based object storage.
  - Point-in-time recovery for PostgreSQL.
  - Regular disaster recovery drills to validate recovery procedures.
- **Data Residency:** All infrastructure, including compute, storage, and backups, was strictly confined to EU-based data centers, with clear contractual agreements and audits to confirm compliance.

Challenges and Hard-Won Lessons

Self-hosting a complex, stateful application on Kubernetes presented numerous challenges.

1. Kubernetes Operational Complexity (Day 2 Operations)

Challenge: While Kubernetes simplified deployment, managing the cluster itself (upgrades, patching, troubleshooting) proved demanding for a small team. State of the art tooling like ArgoCD helped with application deployment, but the underlying cluster required significant expertise. **Solution:** Invested heavily in training for the DevOps team. Standardized on Rancher for cluster lifecycle

management, which provided a more user-friendly interface and automation for common tasks. Developed robust monitoring (Prometheus, Grafana) and logging (Loki, Promtail) to quickly diagnose issues.

2. Managing Stateful Applications

Challenge: Running databases (PostgreSQL, Elasticsearch) and object storage (MinIO) directly on Kubernetes, especially with persistent volumes and high availability, was more complex than stateless microservices. Ensuring data durability during node failures and seamless upgrades required careful orchestration. **Solution:** Leveraged Kubernetes Operators (e.g., Crunchy Data PostgreSQL Operator, MinIO Operator) which encapsulate operational knowledge for specific applications, automating tasks like backup, restore, scaling, and failover. This significantly reduced the manual burden and improved reliability.

3. Real-time Performance Tuning

Challenge: Achieving sub-100ms latency for real-time edits with many concurrent users required meticulous tuning of the WebSocket stack, CRDT synchronization, and underlying network. **Solution:**

- **Network Optimization:** Configured kernel parameters (e.g., `net.core.somaxconn`, `net.ipv4.tcp_max_syn_backlog`) on Kubernetes nodes.
- **Resource Allocation:** Fine-tuned CPU and memory requests/limits for the Collaboration Service pods to prevent throttling and ensure consistent performance.
- **CRDT Optimization:** Implemented custom logic to batch small CRDT updates to reduce WebSocket message overhead and prioritize critical updates.

4. Security Hardening in a Multi-tenant Environment

Challenge: Protecting sensitive user data and preventing unauthorized access in a multi-tenant environment required a layered security approach. **Solution:**

- **Network Policies:** Implemented Kubernetes Network Policies to restrict traffic between microservices to only necessary ports and protocols.
- **Secrets Management:** Used HashiCorp Vault, integrated with Kubernetes, for secure storage and rotation of database credentials, API keys, and other sensitive information.
- **Image Scanning:** Integrated Trivy into the CI/CD pipeline to scan Docker images for known vulnerabilities before deployment.

- **Role-Based Access Control (RBAC):** Strictly enforced RBAC within Kubernetes and for application-level access.

5. Compliance and Legal Aspects

Challenge: Navigating the evolving landscape of EU data protection laws and demonstrating compliance was an ongoing effort. **Solution:**

- **Data Protection Officer (DPO):** Appointed a dedicated DPO to oversee compliance.
- **Regular Audits:** Engaged third-party auditors to verify data residency, security controls, and processing activities.
- **Transparent Policies:** Published clear and comprehensive privacy policies, data processing agreements, and terms of service.

Achieved Results and Business Impact

Despite the challenges, EuroDocs successfully launched its platform, demonstrating the viability of a self-hosted, Kubernetes-native approach for sensitive applications.

- **Operational Stability:** Achieved an average uptime of 99.9% over the first year of production, meeting the target SLA.
- **Performance:** Consistently delivered real-time editing latency below 80ms for users within the EU, even during peak loads of up to 5,000 concurrent users. Document load times averaged 350ms.
- **Compliance:** Fully compliant with GDPR and other relevant EU data regulations, successfully passing multiple third-party audits. This was a significant differentiator in the market.
- **Cost Efficiency:** While initial setup costs were higher due to expertise investment, long-term operational costs were estimated to be 25-30% lower than comparable commercial cloud-managed services due to avoided vendor fees and optimized resource utilization.
- **User Adoption:** Attracted over 50,000 active users within the first 18 months, primarily from organizations and individuals prioritizing data privacy.

Key Takeaways for Self-Hosting Cloud-Native Platforms

The EuroDocs journey provided critical insights for any organization considering self-hosting complex, stateful applications on Kubernetes:

1. **Invest in Expertise Early:** Kubernetes and cloud-native operations require specialized skills. Budget for training, certifications, and potentially experienced hires or consultants.
2. **Operators are Your Friends for Stateful Workloads:** For databases, message queues, and other stateful services, Kubernetes Operators are invaluable. They automate complex day-2 operations, reducing toil and increasing reliability.
3. **Prioritize Observability:** Robust monitoring, logging, and tracing are non-negotiable. They are the eyes and ears of your system, crucial for troubleshooting and performance optimization in a distributed environment.
4. **Security by Design is Paramount:** Integrate security practices throughout the development and deployment lifecycle, from image scanning to network policies and secrets management.
5. **Automate Everything Possible:** From CI/CD pipelines to infrastructure provisioning and cluster upgrades, automation reduces human error and increases operational efficiency. GitOps is a highly recommended approach.
6. **Load Testing is Continuous:** Don't just test once. Integrate load testing into your CI/CD to catch performance regressions early and validate scaling strategies.
7. **Data Sovereignty Requires Holistic Planning:** It's not just about where your servers are. It encompasses data processing agreements, supply chain audits, and legal counsel.

EuroDocs' success demonstrates that with strategic planning, significant investment in expertise, and a commitment to robust engineering practices, self-hosting a complex, privacy-focused cloud-native platform is not only feasible but can also yield significant strategic advantages.

References

- Kubernetes from Dev to Production: Lessons learned from self ...
- Lessons learned from hosting an European alternative to Google Docs
- Yjs: A Framework for Offline-First P2P Collaborative Applications - Conceptual reference for CRDT implementation.
- GitOps with Argo CD - Industry standard for GitOps practices.

Transparency Note

This case study is a fictionalized account inspired by common challenges and solutions encountered in real-world Kubernetes and cloud-native deployments, particularly those with strong data sovereignty requirements. The company "EuroDocs" and specific metrics are illustrative, but the technical architectures, challenges, and lessons learned are derived from general industry knowledge and the provided search context on self-hosting European alternatives to Google Docs with Kubernetes.