# AriesView – Recommended Software Tools for Deployment

## 1. Introduction

AriesView has developed a robust foundation with modern frontend technologies (Next.js, React, Tailwind), backend microservices (Node.js, Flask, FastAPI), PostgreSQL with PostGIS, Weaviate for vector storage, and Azure AI/ML integrations.

While this stack is strong, several **critical gaps remain in caching, monitoring, orchestration, and security**. These gaps must be addressed to ensure AriesView can scale reliably, support larger data volumes, and meet enterprise-grade security expectations.

This document presents a prioritized roadmap of additional tools we recommend deploying, organized into **Immediate Priorities (next 2–4 weeks), High Priorities (1–2 months), Medium Priorities (2–3 months), and Strategic Recommendations (3–6 months)**.

## 2. Immediate Priorities (Next 2–4 Weeks)

### 2.1 Redis – Caching & Session Management

**Why Needed:**
Currently, PostgreSQL handles every query, including repeated ones, which creates unnecessary load. By adding Redis, AriesView can:

- Cache financial calculation results (e.g., NOI, IRR, DSCR).
- Store user sessions in memory for faster logins and API responses.
- Cache RAG/AI responses to reduce repeated embedding/LLM calls.
- Reduce PostgreSQL load by an estimated **60–80%**.

**Implementation:**
Redis can be introduced as a container in the existing Docker setup. Minimal code changes

are required — APIs simply fetch from Redis first and fall back to PostgreSQL when necessary. This makes Redis the **fastest to win with the largest impact**.

## 2.2 Prometheus + Grafana – Monitoring & Observability

**Why Needed:**
At present, AriesView has **no visibility** into system performance. This is a critical risk. With Prometheus and Grafana, the team can:

- Monitor API response times and error rates.
- Track database query performance and detect slow queries.
- Monitor Docker container health and resource usage.
- Set up alerts for system failures, bottlenecks, or SLA breaches.

**Implementation:**
Prometheus scrapes metrics from each service, while Grafana provides real-time dashboards. Alerts can be configured to notify the team via Slack or email if performance drops below thresholds. This ensures AriesView is never "flying blind" in production.

## 2.3 ELK Stack (Elasticsearch, Logstash, Kibana) – Centralized Logging

**Why Needed:**
AriesView currently relies on console.log and scattered logs across multiple services. This approach is not scalable for a multi-service architecture. The ELK stack will:

- Aggregate logs from Node.js, Flask, FastAPI, and Docker services into one searchable system.
- Enable engineers to trace errors across the entire pipeline (OCR → RAG → Financial Calculations).
- Provide Kibana dashboards for analyzing system behavior over time.
- Reduce debugging time significantly by centralizing structured logs.

**Implementation:**
Logstash ingests service logs, Elasticsearch stores and indexes them, and Kibana provides visualization. This stack integrates cleanly with Dockerized services.

# 3. High Priorities (Next 1–2 Months)

## 3.1 Apache Airflow – Workflow Orchestration

**Why Needed:**
AriesView pipelines are currently managed manually or via ad-hoc scripts. With OCR → embeddings → RAG → financial analysis workflows, orchestration is essential. Airflow will:

- Schedule financial calculations (e.g., nightly T-12 refresh).
- Manage OCR workflows with retries and error handling.
- Ensure data flows (e.g., from Azure Blob → OCR → Weaviate → PostgreSQL) run in correct order.
- Provide dashboards for pipeline status and SLA monitoring.

**Implementation:**
Airflow DAGs will be created for OCR and financial workflows. This introduces reliability and visibility into every pipeline, reducing the risk of silent failures.

## 3.2 API Gateway – Kong or Azure API Management

**Why Needed:**
AriesView already exposes multiple APIs (Node.js API, Flask financial calcs, RAG service, OCR service). Without a gateway, each service manages authentication and routing separately, creating complexity and security gaps. An API gateway will:

- Provide a single-entry point for all APIs.
- Enforce security, authentication, and rate limiting.
- Enable versioning and documentation.
- Support load balancing and monitoring across services.

**Implementation:**
Kong (open source) or Azure API Management (cloud-native) can be deployed depending on long-term strategy. Either option improves security, governance, and ease of scaling.

### 3.3 HashiCorp Vault – Secrets Management

**Why Needed:**
Currently, API keys and credentials (Azure OpenAI, Firebase, PostgreSQL) are stored in environment files. This is not secure. Vault will:

- Store and encrypt secrets centrally.
- Rotate keys automatically.
- Prevent secrets from being exposed in .env files or Git.

**Implementation:**
Vault integrates with GitLab CI/CD pipelines, ensuring secrets are injected securely during deployments. This is a **must-have for enterprise clients** handling sensitive financial data.

## 4. Medium Priorities (Next 2–3 Months)

### 4.1 RabbitMQ – Message Queue

- Decouples services (OCR jobs, financial calc requests).
- Improves reliability by retrying failed jobs.
- Handles spikes in workload without overloading core services.

### 4.2 Testing Suite – Cypress + K6

- **Cypress**: End-to-end testing of user workflows in the dashboard.
- **K6**: Load testing APIs under simulated high traffic.
  Together, these prevent regressions, ensure stable performance, and catch failures early.

### 4.3 Sentry – Error Tracking

- Provides real-time alerts when errors occur in Node.js, Flask, or frontend.
- Captures stack traces for faster debugging.
- Tracks how errors impact users directly.

# 5. Strategic Recommendations (3–6 Months)

## 5.1 Kubernetes (Azure Kubernetes Service – AKS)

- Orchestrates Docker containers at scale.
- Enables auto-scaling during peak load.
- Supports rolling deployments and zero-downtime updates.
- Natural evolution from the current Docker setup.

## 5.2 Advanced Analytics & AI Ops

- **Apache Spark + dbt**: For large-scale data processing and transformation.
- **MLflow**: For managing AI/ML experiments and underwriting models.
- **Terraform**: Infrastructure as code, enabling reproducible environments.

# 6. Implementation Strategy

- **Weeks 1–2:** Deploy Redis (cache sessions, RAG results, financial calcs).
- **Weeks 3–4:** Add Prometheus + Grafana for monitoring.
- **Month 2:** Deploy ELK for centralized logging.
- **Month 3:** Introduce Airflow for workflows and API Gateway for unified API management.
- **Months 4–6:** Add RabbitMQ, expand testing suite, implement Vault, and prepare for Kubernetes.

# 7. Cost–Benefit Analysis

- **High Impact, Low Effort**:
  - Redis (1 week, ~60% performance boost).
  - Prometheus + Grafana (2 weeks, prevents outages).
- **High Impact, Medium Effort**:
  - Airflow (1 month, orchestrates pipelines end-to-end).
  - API Gateway (2 weeks, secures multi-service APIs).

- **Medium Impact, Low Effort**:
    - Sentry (1-day, immediate error visibility).
    - Basic security scanning (1 week).

## 8. What Not to Implement Yet

- **Apache Spark**: Current data volume does not justify the overhead.
- **Complex ML pipelines**: Stability first, advanced ML later.
- **Advanced BI dashboards**: Should follow once monitoring and orchestration are stable.

## 9. Conclusion

The recommended roadmap begins with **Redis and Monitoring** — delivering immediate performance and visibility.
This is followed by **Airflow, API Gateway, and Vault**, which harden the architecture and workflows.
Finally, AriesView evolves toward **Kubernetes, advanced analytics, and ML lifecycle management**, ensuring the platform is reliable, secure, and scalable for enterprise adoption.

By following this phased approach, AriesView establishes a foundation for long-term growth while delivering quick wins in the near term.