Nazar

Software Architecture Document

Performance Data Storage & Analysis Platform

ARC42 Template

A system performance monitoring and analysis platform designed to help DevOps engineers and system administrators monitor server infrastructure health through intelligent anomaly detection and alerting.

Author

Md Asif Iqbal Ahmed

asif.ahmed9414@gmail.com

Version: 1.2

Version History

Version	Date	Author	Changes
1.0	2026-02-01	Md Asif Iqbal Ahmed	Initial architecture document
1.1	2026-02-02	Md Asif Iqbal Ahmed	Added ADR-003 alternatives and rationale for push-based collection
1.2	2026-02-02	Md Asif Iqbal Ahmed	Added ADR-008 for SSE-based real-time dashboard updates

Contents

List of Figures

List of Tables

1 Introduction and Goals

Nazar is a performance monitoring platform that collects server metrics, detects anomalies, and alerts users when issues arise.

1.1 Requirements Overview

ID	Functionality	Description
F1	Metric Collection	Gather CPU, memory, disk, and network metrics via agents or REST API
F2	Time-Series Storage	Store metrics with automatic retention and compression
F3	Statistical Analysis	Detect threshold violations using historical baselines
F4	ML Anomaly Detection	Learn normal patterns and flag unusual behavior
F5	Alerting	Send notifications via dashboard, Slack, or email
F6	Web Dashboard	Visualize metrics, explore history, configure rules
F7	REST API	Ingest external metrics and enable integrations

Table 1: Main Functional Requirements

1.2 Stakeholders

Who?	Goals and Concerns
DevOps Engineers	Fast anomaly detection, low false positives, CI/CD integration
System Administrators	Full infrastructure visibility, reliable alerts, low overhead
Development Teams	Correlate performance with deployments, custom metrics
Operations Managers	High-level dashboards, SLA tracking, capacity planning

Table 2: Stakeholder Concerns

1.3 Quality Goals

Quality Goal	Motivation	Related Req.
Q1: Performance	Handle thousands of metrics per second with fast query response	F1, F2, F6
Q2: Reliability	Stay available during incidents when monitoring matters most	F1, F2, F5
Q3: Accuracy	Minimize false positives and false negatives in anomaly detection	F3, F4, F5

Table 3: Top Quality Goals

2 Constraints

2.1 Technical Constraints

Constraint	Explanation
Python	Backend language with strong ML ecosystem
TimescaleDB	SQL-compatible time-series database
Docker	Containerized deployment

Table 4: Technical Constraints

2.2 Conventions

Convention	Explanation
Code style	PEP 8
API design	RESTful, JSON
Versioning	Semantic versioning

Table 5: Conventions

3 Context and Scope

3.1 System Context Diagram

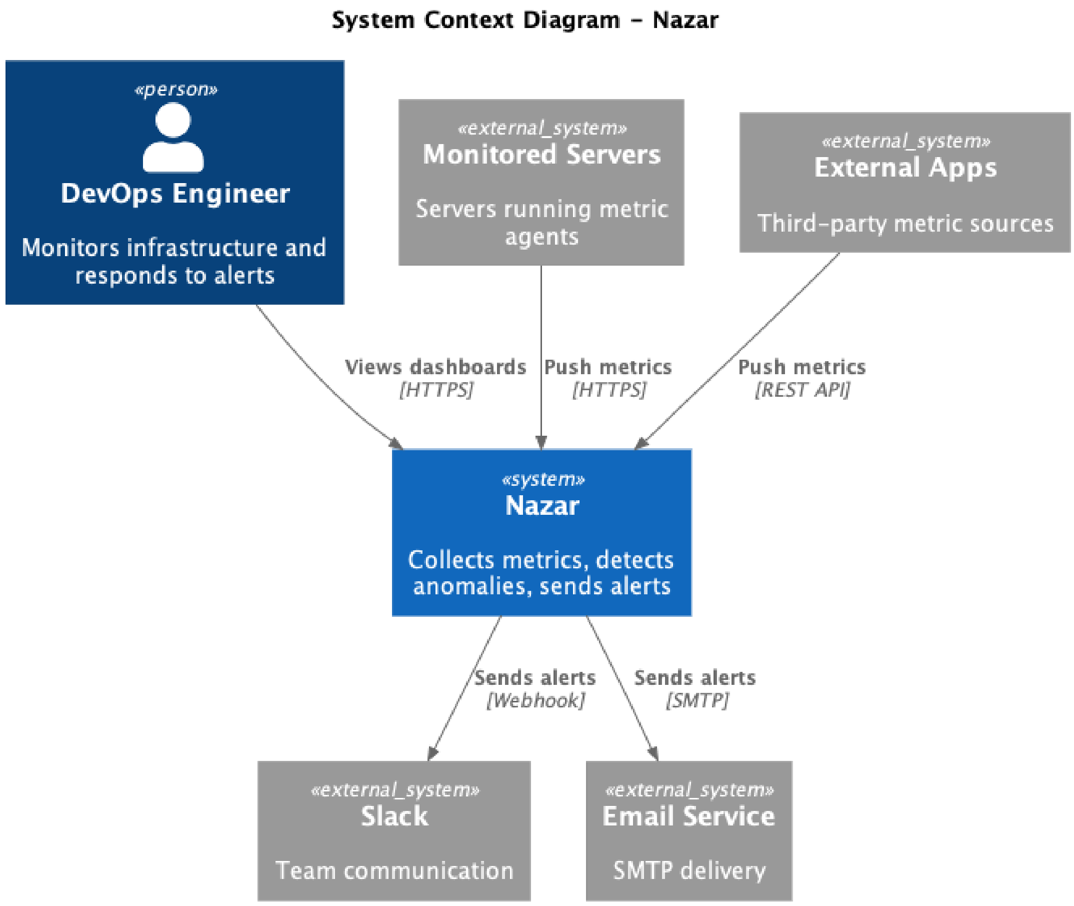

Figure 1: System Context Diagram

3.2 External Entities

Entity	**Type**	**Description**
Monitored Servers	System	Servers running metric collection agents
DevOps Engineer	Actor	Views dashboards, configures alerts
Slack	System	Receives alert notifications via webhook
Email Service	System	Delivers alert emails via SMTP
External Apps	System	Push custom metrics via REST API

Table 6: External Entities

4 Solution Strategy

4.1 Technology Decisions

Decision	Technology	Justification
Backend Framework	FastAPI	Async support, auto API docs, Python native
Time-Series DB	TimescaleDB	SQL compatible, PostgreSQL ecosystem
Message Broker	RabbitMQ	Reliable delivery, decouples ingestion from processing
Frontend	React	Component-based, large ecosystem
Containerization	Docker	Portable, consistent deployment

Table 7: Technology Decisions

4.2 Architectural Patterns

Pattern	Justification
Event-Driven	Decouple metric ingestion from analysis via message queues
Modular Monolith	Simple for solo development, easy to split later
Repository Pattern	Abstract database access for testability

Table 8: Architectural Patterns

4.3 Quality Goal Mapping

Quality Goal	Scenario	Solution Approach
Q1: Performance	1000+ metrics/sec ingestion	Async FastAPI, RabbitMQ buffering, TimescaleDB hypertables
Q2: Reliability	System stays up during incidents	Health checks, message persistence, graceful degradation
Q3: Accuracy	Minimize false alerts	Configurable thresholds, baseline learning, ML tuning

Table 9: Quality Goal Mapping

5 Building Block View

5.1 Level 1: Container Diagram

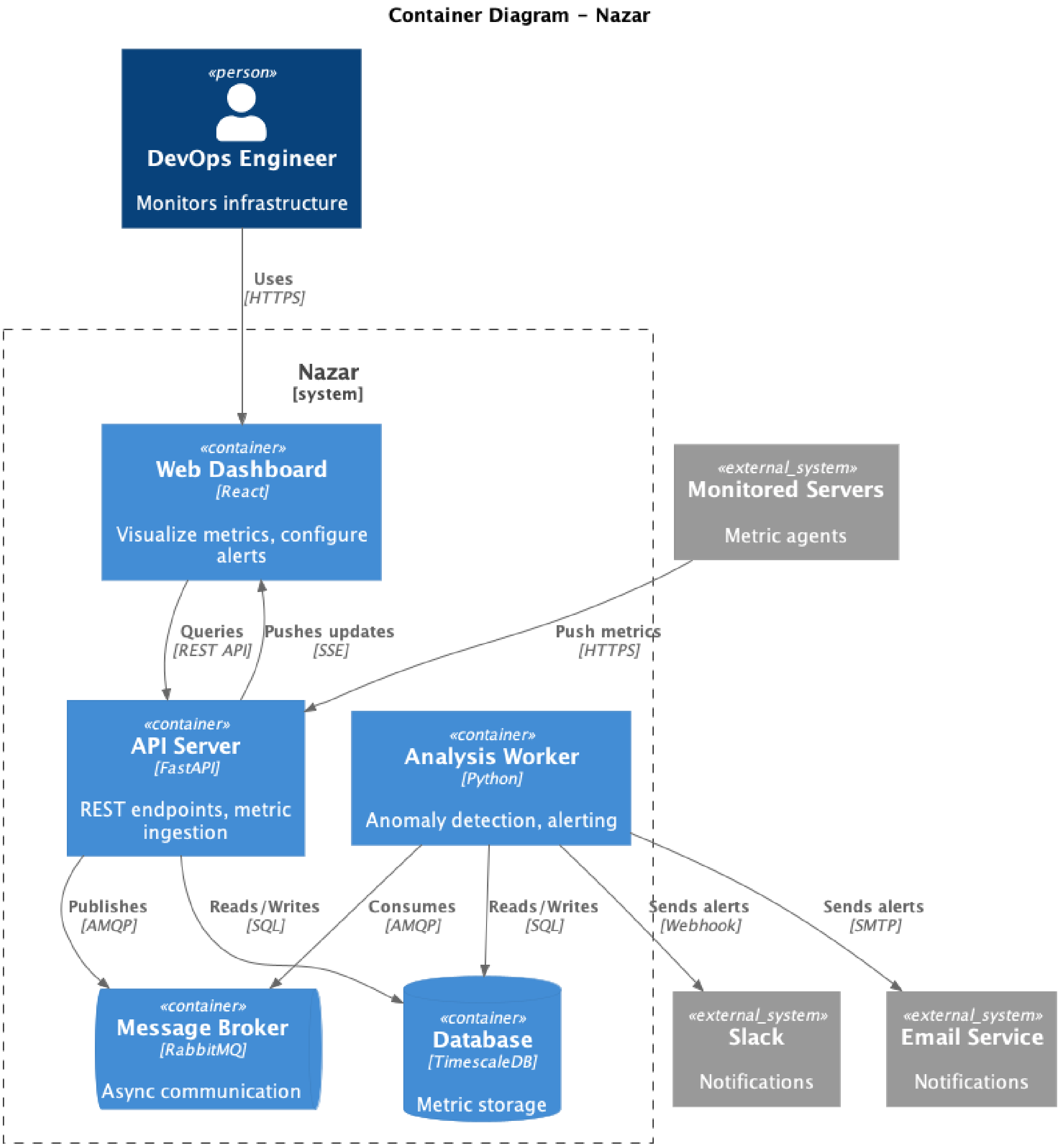

Figure 2: Level 1 - Container Diagram

Container	Purpose
Web Dashboard	React frontend for metric visualization and alert configuration
API Server	FastAPI backend handling REST endpoints, metric ingestion, and SSE streaming
Analysis Worker	Python service for anomaly detection and alert triggering
Message Broker	RabbitMQ for async communication between API and Worker
Database	TimescaleDB for time-series metric storage

Table 10: Level 1 Building Blocks

5.2 Level 2: API Server Components

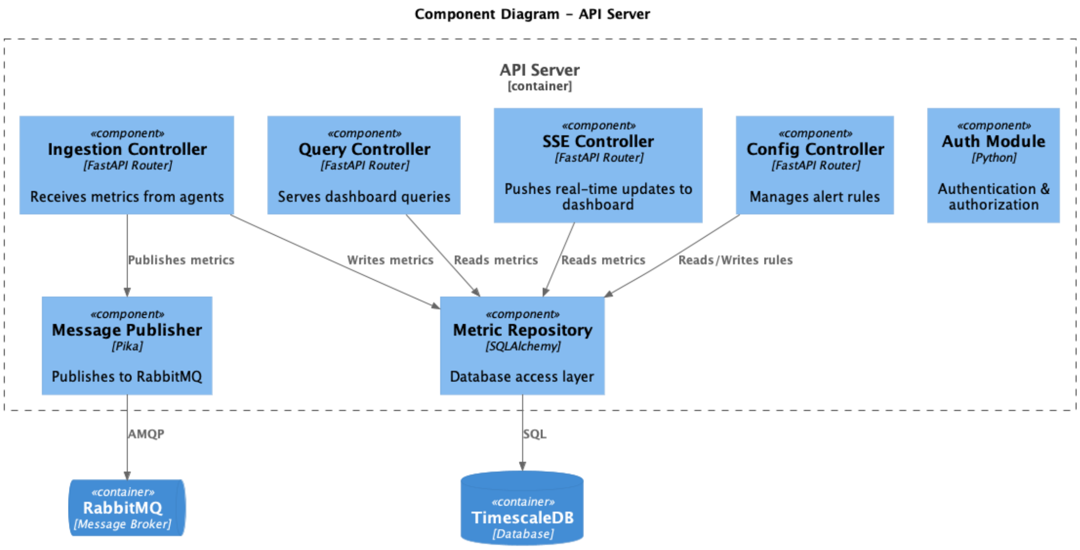

Figure 3: Level 2 - API Server Components

Component	Purpose
Ingestion Controller	Receives metrics from agents and external sources
Query Controller	Serves metric queries for the dashboard
SSE Controller	Pushes real-time metric updates to the dashboard
Config Controller	Manages alert rules and thresholds
Message Publisher	Publishes metrics to RabbitMQ
Metric Repository	Abstracts database access

Table 11: API Server Components

5.3 Level 2: Analysis Worker Components

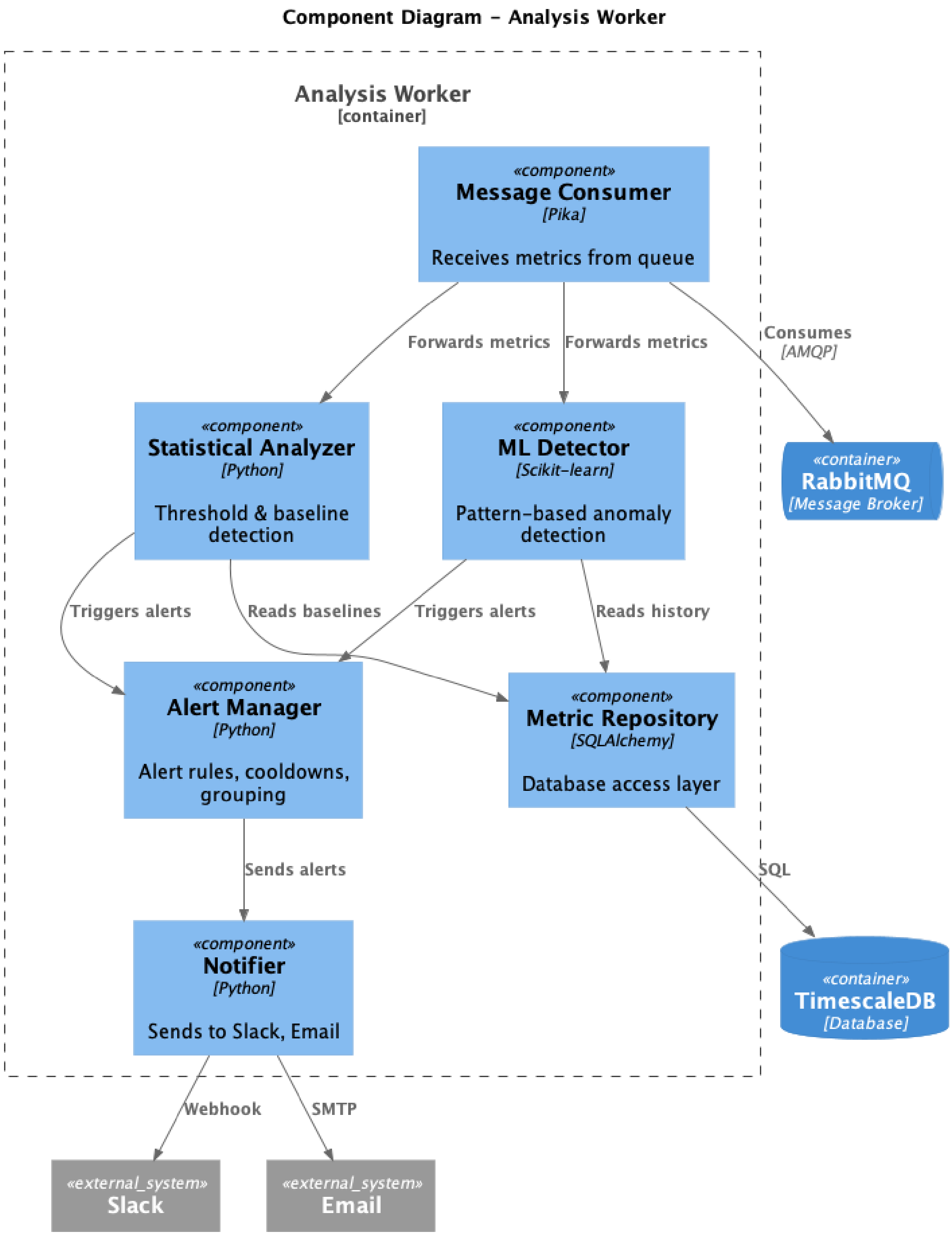

Figure 4: Level 2 - Analysis Worker Components

Component	Purpose
Message Consumer	Receives metrics from RabbitMQ
Statistical Analyzer	Detects threshold violations and baseline deviations
ML Detector	Machine learning based anomaly detection
Alert Manager	Handles alert rules, cooldowns, and grouping
Notifier	Sends alerts to Slack and Email

Table 12: Analysis Worker Components

6 Runtime View

This section describes the behavior of Nazar's building blocks through three key runtime scenarios.

6.1 Runtime Scenario 1: Metric Collection and Storage

This scenario shows how metrics flow from monitored servers into the system.

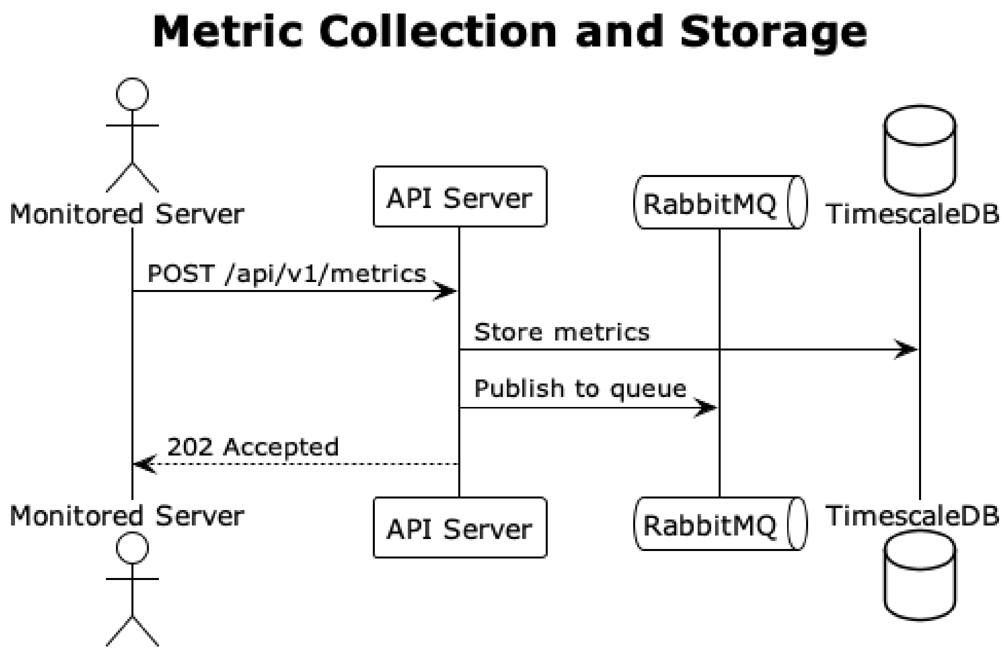

Figure 5: Metric Collection and Storage Flow

6.1.1 Interaction Steps

1. Monitored server pushes metrics to API Server via HTTPS POST
2. API Server stores metrics in TimescaleDB
3. API Server publishes metrics to RabbitMQ for async processing
4. API Server returns 202 Accepted to the server

Component	Role
Monitored Server	Collects and pushes system metrics
API Server	Receives, validates, and routes metrics
TimescaleDB	Persists time-series metric data
RabbitMQ	Queues metrics for analysis processing

Table 13: Scenario 1: Participating Components

6.2 Runtime Scenario 2: Anomaly Detection and Alerting

This scenario shows how anomalies are detected and alerts are delivered.

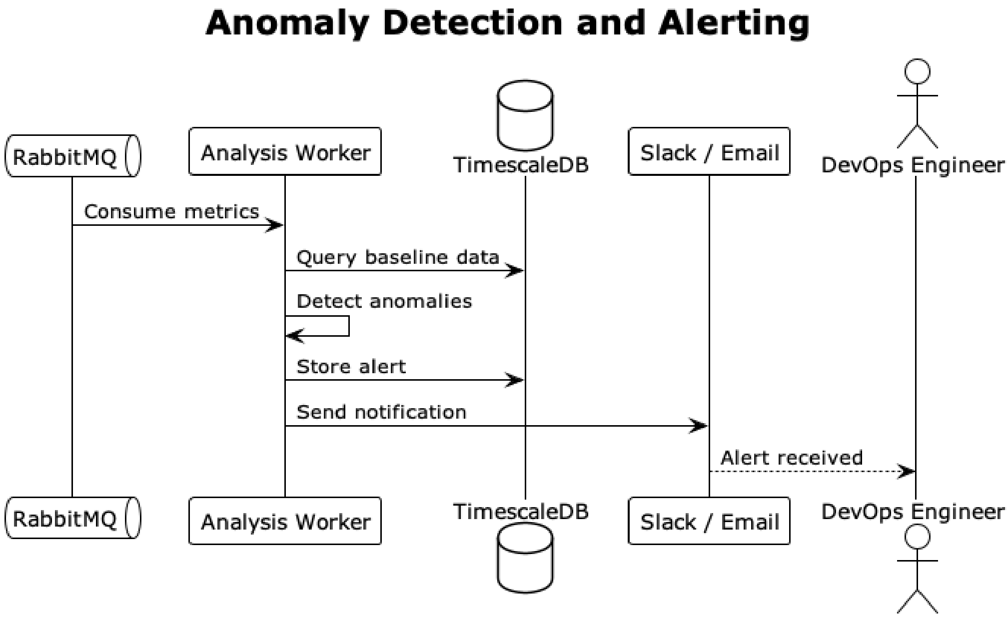

Figure 6: Anomaly Detection and Alerting Flow

6.2.1 Interaction Steps

1. Analysis Worker consumes metrics from RabbitMQ
2. Worker queries TimescaleDB for historical baseline data
3. Worker analyzes metrics and detects anomalies
4. Worker stores alert record in TimescaleDB
5. Worker sends notifications via Slack and Email
6. DevOps Engineer receives alert notification

Component	Role
RabbitMQ	Delivers metric messages to worker
Analysis Worker	Detects anomalies and triggers alerts
TimescaleDB	Provides baseline data and stores alerts
Slack / Email	Delivers notifications to users

Table 14: Scenario 2: Participating Components

6.3 Runtime Scenario 3: Real-time Dashboard Updates

This scenario shows how the dashboard receives real-time updates via Server-Sent Events (SSE).

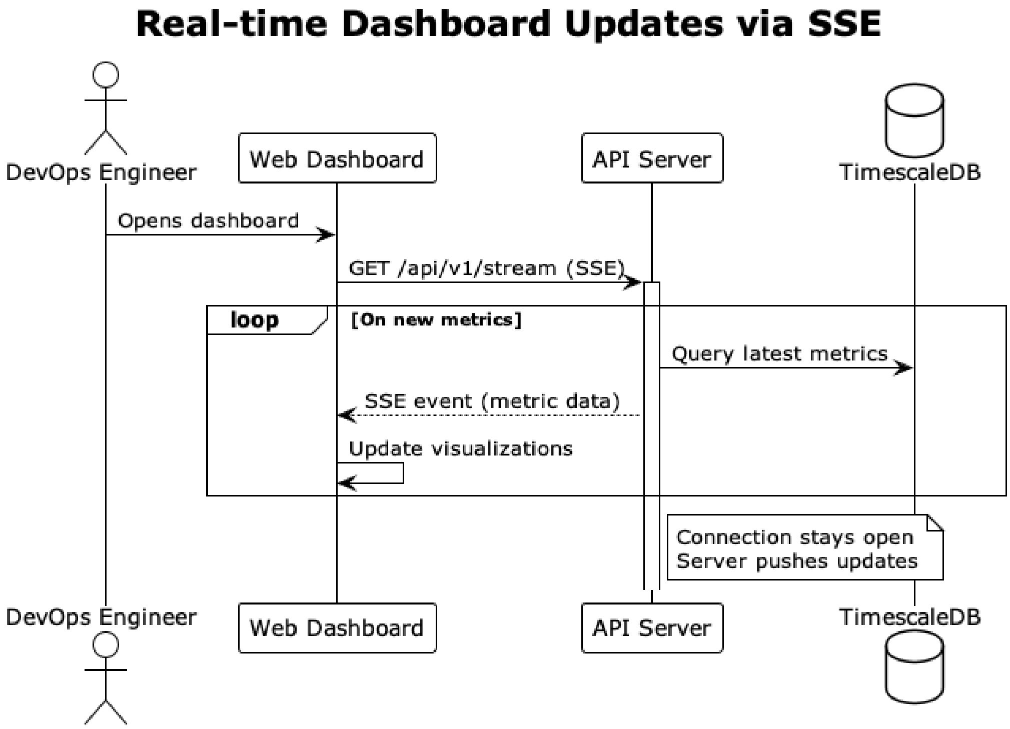

Figure 7: Real-time Dashboard Updates via SSE

6.3.1 Interaction Steps

1. DevOps Engineer opens the Web Dashboard
2. Dashboard establishes SSE connection to API Server
3. API Server queries TimescaleDB for latest metrics
4. API Server pushes metric data as SSE events
5. Dashboard updates visualizations in real-time
6. Connection remains open for continuous updates

Component	Role
Web Dashboard	Displays metrics and maintains SSE connection
API Server	Streams real-time updates via SSE
TimescaleDB	Provides metric data for streaming

Table 15: Scenario 3: Participating Components

7 Crosscutting Concepts

7.1 Patterns and Tactics

Pattern/Tactic	Purpose
Event-Driven	Decouples ingestion from analysis via RabbitMQ
Repository Pattern	Abstracts database operations
Async Processing	Fast API response by queuing work
Batch Processing	Groups metrics for efficient storage
Circuit Breaker	Handles external service failures gracefully

Table 16: Patterns and Tactics

7.2 Cross-cutting Concerns

Concern	Approach
Logging	Structured JSON logs to stdout
Security	API key auth, HTTPS, input validation
Error Handling	Retry with backoff, dead letter queues
Configuration	Environment variables

Table 17: Cross-cutting Concerns

7.3 Variability Points

Element	Variation
Notifier	Pluggable channels (Slack, Email, etc.)
ML Detector	Swappable algorithms (Isolation Forest, LSTM)
Metric Sources	Extensible via REST API or agents
Alert Rules	User-defined thresholds per metric

Table 18: Variability Points

8 Architectural Decisions

8.1 ADR-001: Modular Monolith

Context	Solo developer project; need simple but organized architecture
Alternatives	Clean Architecture, Microservices
Decision	Modular monolith - simpler than Clean Architecture for solo project
Consequences	+ Simpler deployment and debugging + Can split into microservices later if needed - Must maintain discipline on module boundaries

Table 19: ADR-001: Modular Monolith

8.2 ADR-002: Event-Driven Architecture

Context	Metric ingestion and analysis have different performance characteristics
Decision	Use event-driven architecture with message queues between components
Consequences	+ Components scale independently + Ingestion not blocked by slow analysis - Eventual consistency, harder to debug

Table 20: ADR-002: Event-Driven Architecture

8.3 ADR-003: Push-based Metric Collection

Context	Need to collect metrics from servers
Alternatives	Pull-based (server scrapes targets like Prometheus)
Decision	Push-based collection where agents POST metrics to API
Rationale	Pull requires inbound network access to monitored servers. Push works through firewalls and NAT without opening ports. Agents can batch and retry on failure.
Consequences	+ Works through firewalls, no inbound ports needed + Agents control send frequency + Easier to add new servers (no central config) - Server must handle burst traffic

Table 21: ADR-003: Push-based Metric Collection

8.4 ADR-004: Message Broker Selection

Context	Need async processing; also wanted to learn message broker technology
Alternatives	Direct processing (no broker), Kafka
Decision	RabbitMQ - good learning opportunity, simpler than Kafka
Consequences	+ Reliable message delivery with acknowledgments + Buffers load spikes - Additional infrastructure component to manage

Table 22: ADR-004: Message Broker Selection

8.5 ADR-005: Time-Series Database Selection

Context	Need efficient storage for timestamped metrics with fast range queries
Alternatives	InfluxDB, plain PostgreSQL
Decision	TimescaleDB - SQL compatible, leverages existing PostgreSQL knowledge
Consequences	+ Automatic partitioning and compression + Full PostgreSQL ecosystem - Additional complexity vs plain PostgreSQL

Table 23: ADR-005: Time-Series Database Selection

8.6 ADR-006: Anomaly Detection Approach

Context	Simple thresholds miss subtle anomalies; pure ML can be hard to tune
Decision	Use hybrid approach: statistical baselines + ML (Isolation Forest)
Consequences	+ Catches both obvious and subtle anomalies + Statistical methods are interpretable - More complex than threshold-only approach

Table 24: ADR-006: Anomaly Detection Approach

8.7 ADR-007: Alert Delivery Strategy

Context	Users need timely alerts without notification fatigue
Decision	Multi-channel delivery (Slack, Email) with cooldowns and grouping
Consequences	+ Reaches users on preferred channels + Cooldowns prevent alert spam - More complex notification logic

Table 25: ADR-007: Alert Delivery Strategy

8.8 ADR-008: Server-Sent Events for Dashboard

Context	Dashboard needs real-time metric updates without constant polling
Alternatives	Polling (client requests every N seconds), WebSocket (bidirectional), SSE (server-to-client push)
Decision	Server-Sent Events (SSE) for pushing metrics and alerts to dashboard
Rationale	Dashboard only needs one-way server-to-client push. SSE is simpler than WebSocket, has automatic reconnection, works over standard HTTP, and is sufficient for our use case. WebSocket would add complexity for bidirectional capability we don't need.
Consequences	+ Simpler than WebSocket + Automatic reconnection built into browser API + Works with standard HTTP infrastructure - Cannot send client-to-server messages (not needed)

Table 26: ADR-008: Server-Sent Events for Dashboard

9 Quality Requirements

9.1 Quality Attribute Scenarios

9.1.1 QAS-1: Performance - Metric Ingestion

Source	Monitored servers
Stimulus	1000+ metrics per second
Artifact	API Server
Environment	Normal operation
Response	Accept and queue all metrics
Measure	95% of requests complete in < 100ms

Table 27: QAS-1: Metric Ingestion Performance

9.1.2 QAS-2: Reliability - System Availability

Source	Infrastructure incident
Stimulus	Component failure or high load
Artifact	Entire system
Environment	Production
Response	Continue collecting metrics, queue alerts
Measure	No metric data loss during partial failures

Table 28: QAS-2: System Availability

9.1.3 QAS-3: Accuracy - Anomaly Detection

Source	Analysis Worker
Stimulus	Unusual metric pattern
Artifact	Anomaly detection module
Environment	After baseline learning period
Response	Generate alert with appropriate severity
Measure	< 5% false positive rate

Table 29: QAS-3: Anomaly Detection Accuracy

9.2 Quality Tree

Quality	Attribute	Scenario	Priority
Performance	Throughput	QAS-1: Handle 1000+ metrics/sec	H, H
Reliability	Availability	QAS-2: No data loss during failures	H, M
Accuracy	Detection	QAS-3: Low false positive rate	H, M

Table 30: Quality Tree

Priority: (Business Importance, Technical Risk) - H=High, M=Medium

10 Glossary

10.1 Terms

Term	Definition
Metric	A measurement of system performance (CPU, memory, disk, network)
Anomaly	A data point that deviates significantly from normal behavior
Baseline	Normal behavior pattern calculated from historical data
Alert	Notification triggered when anomaly is detected
Time-Series DB	Database optimized for timestamped data with fast writes and range queries
Agent	Software on monitored servers that collects and sends metrics
Isolation Forest	ML algorithm used for anomaly detection

Table 31: Glossary

10.2 Acronyms

Acronym	Meaning
ADR	Architecture Decision Record
QAS	Quality Attribute Scenario
AMQP	Advanced Message Queuing Protocol (RabbitMQ)

Table 32: Acronyms