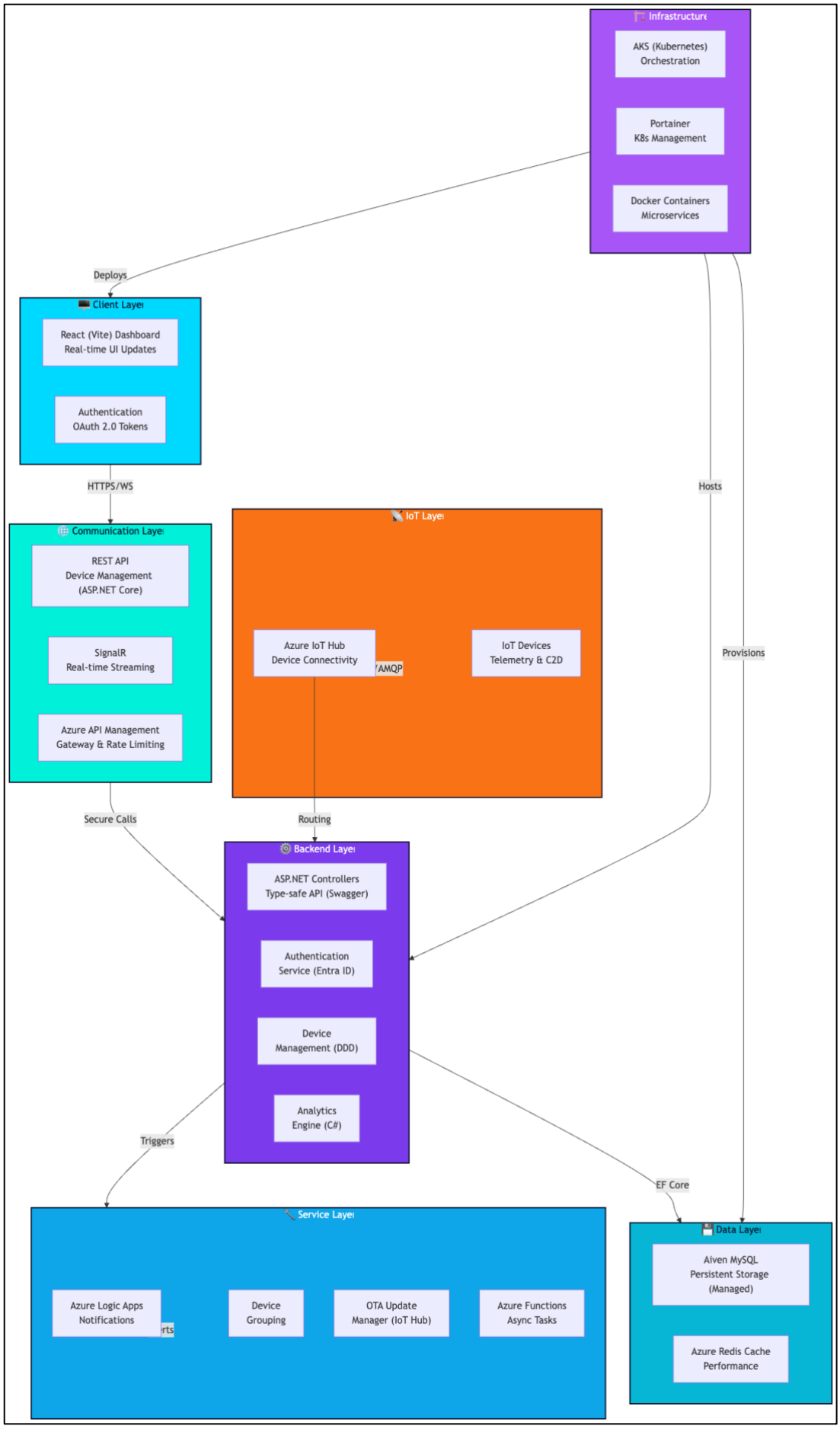
Smart Factory IoT - System Architecture

Overview

The Smart Factory IoT platform is a comprehensive real-time industrial monitoring and control system. It features a React (Vite) frontend for interactive dashboards, a .NET backend for robust API and business logic, and integration with Azure IoT Hub for device connectivity. The architecture emphasizes robustness through microservices and containerization, security via Microsoft Entra ID and RBAC, extensibility using DDD and SOLID principles, and high performance with caching, asynchronous processing, and optimized queries.

System Architecture Diagram


Infrastructure
AKS (Kubernetes)
Orchestration
Portainer
K8s Management
Docker Containers
Microservices
Deploys
Client Layer
React (Vite) Dashboard
Real-time UI Updates
Authentication
OAuth 2.0 Tokens
HTTPS/WS
Hosts
Communication Layer
REST API
Device Management
(ASP.NET Core)
SignalR
Real-time Streaming
Azure API Management
Gateway & Rate Limiting
IoT Layer
Azure IoT Hub
Device Connectivity
AMQP
IoT Devices
Telemetry & C2D
Provisions
Secure Calls
Routing
Backend Layer
ASP.NET Controllers
Type-safe API (Swagger)
Authentication
Service (Entra ID)
Device
Management (DDD)
Analytics
Engine (C#)
Triggers
EF Core
Service Layer
Azure Logic Apps
Notifications
Device
Grouping
OTA Update
Manager (IoT Hub)
Azure Functions
Async Tasks
Data Layer
Aiven MySQL
Persistent Storage
(Managed)
Azure Redis Cache
Performance

Component Details

Client Layer

- **React (Vite) Dashboard**: Interactive UI for real-time monitoring, built with Vite for fast development and bundling. Deployed to Vercel for serverless hosting and automatic scaling.
- **Authentication**: OAuth 2.0 with Microsoft Entra ID for secure token-based auth.
- **State Management**: React Context + TanStack Query for data fetching.

Communication Layer

- **REST API**: Device CRUD operations, configuration via ASP.NET Core Web API.
- **SignalR**: Real-time sensor data streaming integrated with Azure SignalR Service for scalability.
- **Azure API Management**: Gateway for API versioning, rate limiting, and security policies.

Backend Layer

- **ASP.NET Controllers**: Centralized API endpoint management with OpenAPI/Swagger for documentation.
- **Authentication Service**: Entra ID integration with JWT validation.
- **Device Management**: CRUD operations and device grouping using Domain-Driven Design (DDD) aggregates and repositories.
- **Analytics Engine**: OEE calculations and reporting in C#.

Data Layer

- **Aiven MySQL**: Managed MySQL database for persistent storage, provisioned via Terraform for cross-cloud compatibility.
- **Azure Redis Cache**: In-memory caching for high-performance reads.

Service Layer

- **Azure Logic Apps**: Workflow automation for notifications (Email/SMS via Microsoft Graph API).
- **Device Grouping**: Batch operations and analytics aggregation.
- **OTA Update Manager**: Firmware updates via Azure IoT Hub direct methods.
- **Azure Functions**: Serverless async tasks for data processing, triggered by IoT Hub or queues.

IoT Layer

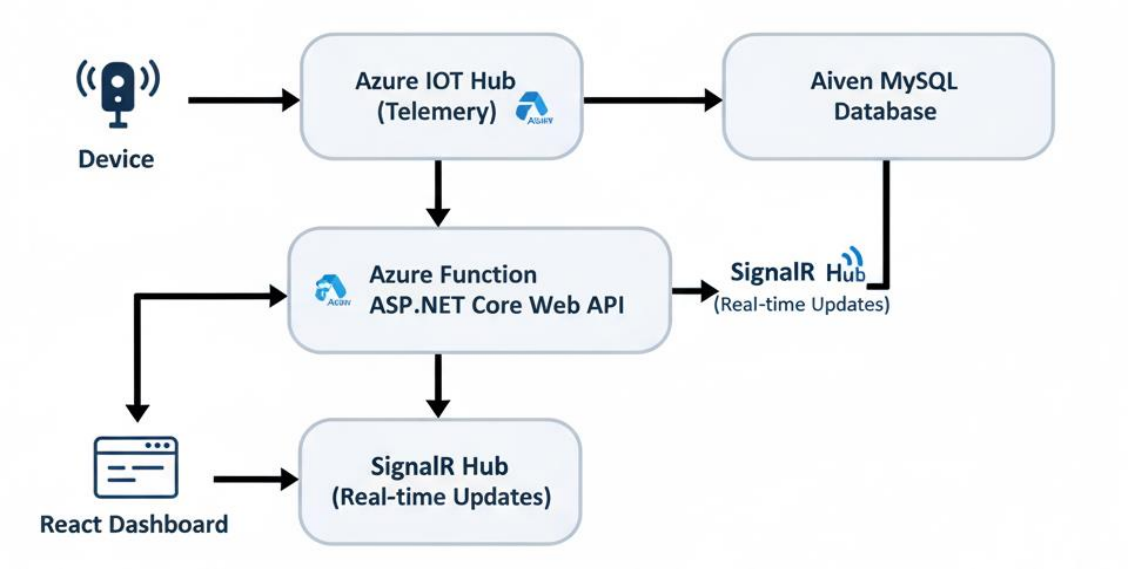
- **Azure IoT Hub**: Central hub for device registration, telemetry ingestion (D2C), commands (C2D), and twins for metadata sync.
- **IoT Devices**: Connect via MQTT/AMQP, with SDKs for secure communication.

Infrastructure Layer

- **Docker**: Containerization of .NET backend microservices for portability.
- **Kubernetes (AKS)**: Orchestration for backend services, ensuring high availability and auto-scaling.
- **Portainer**: Web-based management for Kubernetes clusters, simplifying operations.

Data Flow

Real-time Monitoring Flow



1. Ingestion Layer (The Entry Point)

- **Device:** A physical sensor or simulated device (e.g., Raspberry Pi, ESP32) that collects data like temperature or pressure.
- **Azure IoT Hub:** The central cloud gateway. It manages secure, bi-directional communication and ingests the raw telemetry data stream via protocols like MQTT, AMQP, or HTTPS.

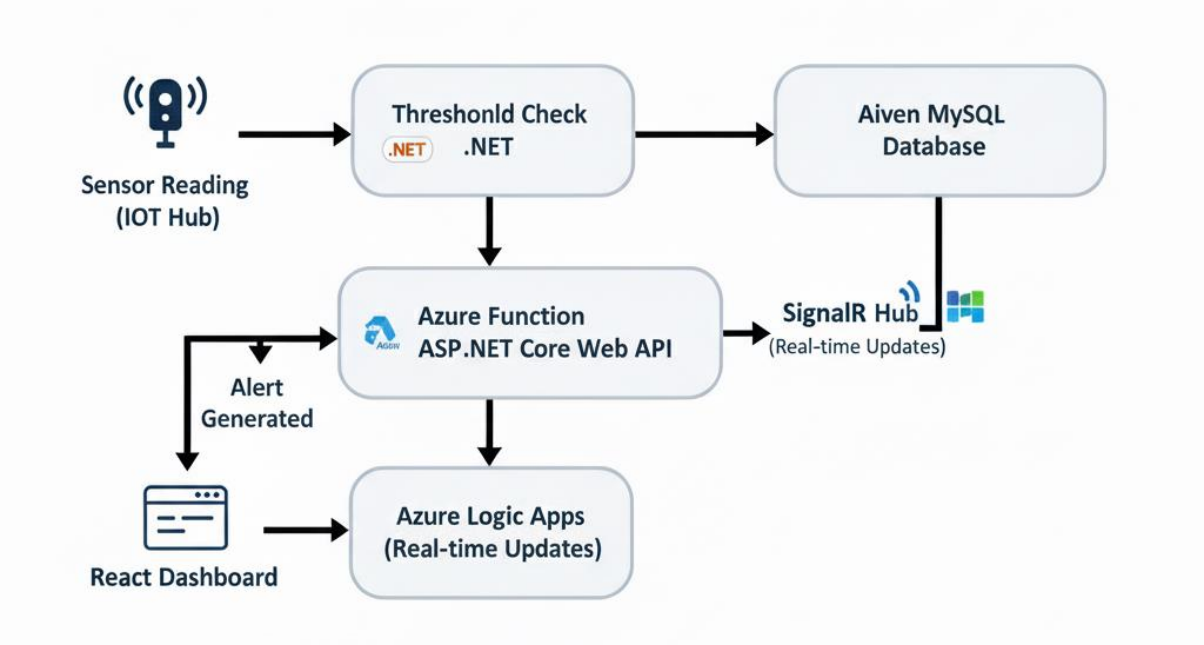
2. Processing & Storage Layer (The Brain)

- **Azure Function / ASP.NET:** A "hot-path" processor. As telemetry arrives at the IoT Hub, it triggers a serverless **Azure Function** or a **.NET worker**.
- **Aiven MySQL:** The processed data is persisted here. This managed database serves as the "Source of Truth" for historical reporting and data integrity.

3. Presentation Layer (The User Experience)

- **SignalR:** This is the "Live" connector. Instead of the browser asking for new data, SignalR **pushes** the latest telemetry values from the backend to the user instantly.
- **React Dashboard:** The front-end UI. It receives SignalR updates to refresh charts and gauges in real-time, providing an interactive, low-latency monitoring experience.

Alert Flow



1. Detection & Evaluation

- **Sensor Reading (IoT Hub):** Telemetry data (like temperature, vibration, or pressure) is ingested from the edge into **Azure IoT Hub**.
- **Threshold Check (.NET):** A **.NET background worker** or **Azure Function** processes the stream in real-time. It compares the incoming values against predefined business rules (e.g., *Is the temperature $> 100°C$?*).

2. Orchestration & Logic

- **Alert Generated:** If the threshold is breached, the .NET service generates an alert payload.
- **Azure Logic Apps:** This acts as the **workflow engine**. Instead of hard-coding notification logic, Logic Apps receives the alert and manages the "if-this-then-that" flow. It can handle retries, branching logic (e.g., *if critical, page the manager; if warning, send an email*), and connections to hundreds of services.

3. Communication & Delivery

- **User Notification (Graph API):** The final step uses the **Microsoft Graph API** to deliver the message directly into the user's ecosystem. This typically manifests as:
  - A **Microsoft Teams** channel message.
  - An **Outlook** email.
  - A calendar event or a task created in **Microsoft To Do**.

Technology Stack

Layer	Technology		Purpose
Frontend	Frontend	React 19 (Vite)	UI Framework & Bundling
Frontend	Frontend	TypeScript	Type Safety
Frontend	Frontend	Tailwind CSS	Styling
Frontend	Frontend	TanStack Query	Data Fetching
Backend	Backend	.NET 8 (ASP.NET Core)	Runtime & Web Framework
Backend	Backend	Entity Framework Core	ORM for MySQL
Backend	Backend	SignalR	Real-time
Database	Database	Aiven MySQL	Managed DB
Azure IoT Hub	IoT	Azure IoT Hub	Device Connectivity
Azure IoT Hub	Infra	Docker/Kubernetes (AKS)	Containerization & Orchestration
Portainer	Infra	Portainer	K8s Management
Portainer	IaC	Azure Bicep/Terraform	Resource Provisioning
Monitoring	CI/CD	GitHub Actions	Automation
Monitoring	Monitoring	Azure Monitor/App Insights	Logging & Alerts
Deployment	Deployment	**Vercel** (Frontend), **Render** (Backnd Fallback)	Hosting

Key Features

1. Real-time Monitoring

- Live device status via SignalR and IoT Hub.
- Telemetry streaming with low latency.
- Automatic reconnection.

2. Alert Management

- Threshold-based alerts processed in .NET.
- Severity levels with Logic Apps notifications.

3. Device Management

- CRUD with grouping, using DDD.
- Health monitoring via IoT Hub.

4. Analytics

- OEE calculation in C#.
- Historical analysis with cached queries.

5. Firmware Management

- OTA via IoT Hub and Blob Storage.
- Status tracking.

Security Architecture

Authentication

- OAuth 2.0 with Entra ID.
- Secure hashing (bcrypt) as fallback.
- Token refresh.

Authorization

- RBAC integrated with Entra ID.
- Managed Identities for services.
- Policy-driven access.

Data Protection

- TLS encryption.
- Key Vault for secrets.
- Input validation.

Scalability Considerations

Horizontal Scaling

- Stateless .NET services in K8s.
- IoT Hub scaling units.
- Redis for distributed caching.

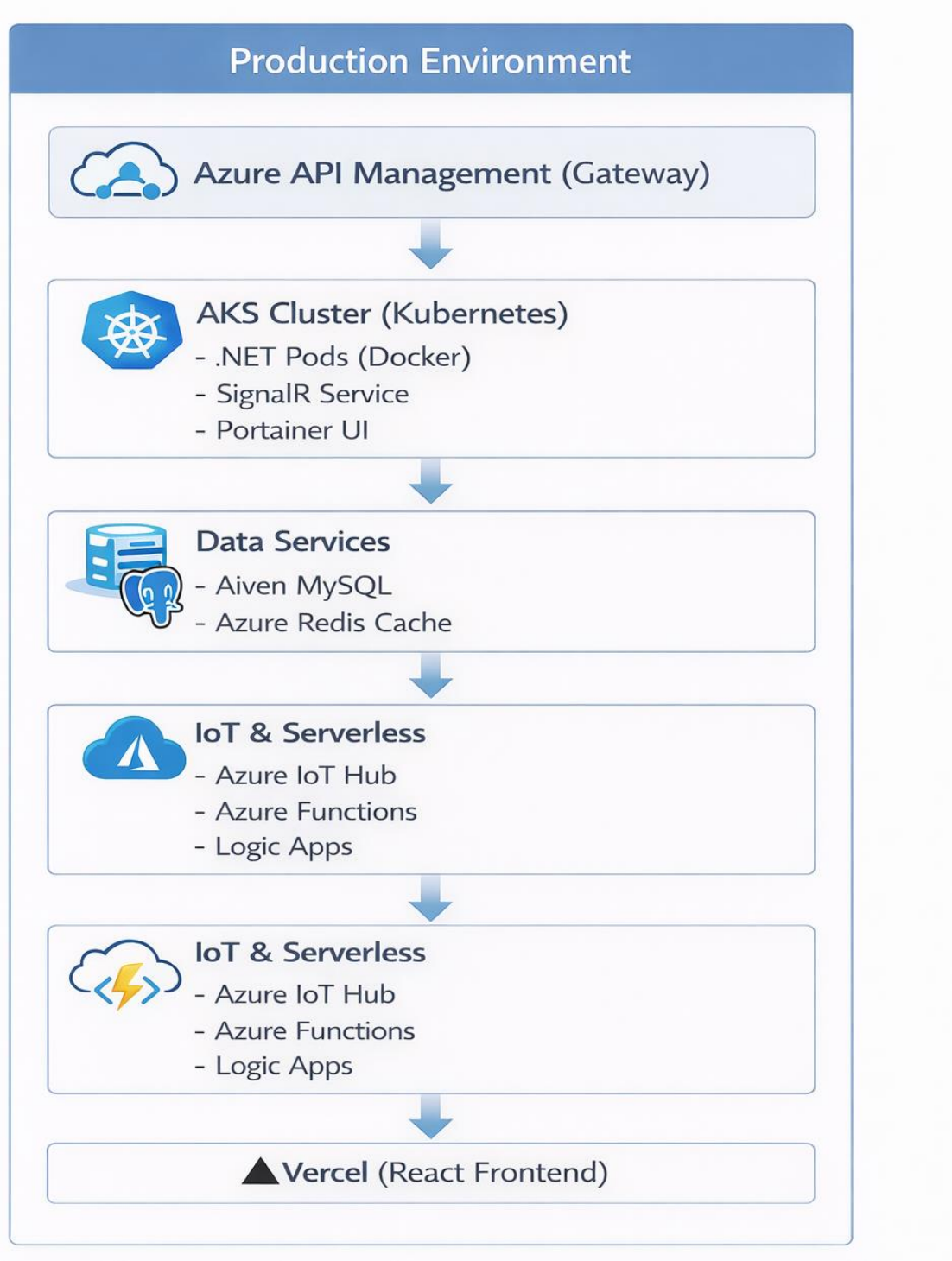
Performance Optimization

- EF Core query optimization.
- AsyncAPI for events.
- Load balancing in AKS.

Monitoring & Observability

- App Insights for telemetry.
- Log Analytics for queries.
- Alerts via Azure Monitor.

Deployment Architecture



IaC: Provisioned with Bicep for Azure resources, Terraform for Aiven DB.

CI/CD: GitHub Actions for build/deploy, with TDD via xUnit.

Development Workflow

1. **Local Development**
    - Docker Compose for local stack.
    - Vite dev server.
    - EF Core migrations.
2. **Testing**
    - Unit/Integration tests (xUnit).
    - TDD practices.
    - Swagger for API testing.
3. **Deployment**
    - GitHub Actions pipeline.
    - Bicep/Terraform apply.
    - Kubernetes manifests via Helm.

SOLID Principles Implementation

Single Responsibility

- Each microservice/Domain has one purpose.

Open/Closed

- Extensible via interfaces.

Liskov Substitution

- Consistent behaviors.

Interface Segregation

- Focused interfaces.

Dependency Inversion

- DI in .NET.

Performance Metrics

- **API Response Time**: < 200ms (p95)
- **SignalR Latency**: < 100ms
- **Dashboard Load Time**: < 2s
- **DB Query Time**: < 50ms (p95)
- **Memory Usage**: < 512MB/pod
- **CPU Usage**: < 50% under load

Future Enhancements

1. **Machine Learning**
    - Azure ML for predictive maintenance.
2. **Advanced Analytics**
    - Power BI integration.
3. **Integration**
    - Kafka for event streaming.
4. **Scalability**
    - Multi-region AKS.