# System Architecture Overview

## Purpose

This document provides a high‑level system architecture overview for **PineCone Pro Supplies’** next‑generation Enterprise Resource Planning (ERP) and Inventory Management System (IMS). The goal is to replace legacy spreadsheets and siloed systems with a unified, modular platform that scales with the business. The overview summarizes the architecture principles, major components, deployment model and operational considerations.

## 1. Architecture vision & principles

**Modular microservices** – Decompose the monolithic ERP into domain‑focused microservices (product, inventory, order, purchasing, shipping, tax/accounting and analytics). Each service is independently deployable, loosely coupled and owns its data. Services communicate via RESTful APIs and asynchronous messages to achieve elasticity and resilience. The architecture leverages cloud design patterns such as **publisher‑subscriber**, **competing consumers** and **API gateway routing** to handle workflows and integrate services【800680358373136†L345-L356】.

**Cloud‑native & containerized** – Containerize each microservice using multi‑stage Docker builds. Base images come from the official .NET SDK and ASP.NET runtime for backend services and Node/Nginx for the Angular front‑end【85830806945883†L203-L233】【85830806945883†L510-L527】. Container images are stored in a private Azure Container Registry and deployed on **Azure Kubernetes Service (AKS)** for portability and horizontal scaling【800680358373136†L404-L432】.

**12‑Factor & DevOps** – Follow 12‑factor application principles: configuration via environment variables, stateless services, automated build/test/release pipelines and comprehensive telemetry. Adopt continuous integration and continuous deployment (CI/CD) using Azure Pipelines or GitHub Actions. Helm charts bundle Kubernetes manifests to simplify deployment and versioning【800680358373136†L433-L441】.

**Security by design** – Enforce role‑based access control (RBAC), least‑privilege service accounts and network segmentation. Use **Microsoft Entra ID** for user and workload identities and **Azure Key Vault** for secrets management【800680358373136†L419-L424】. All traffic between services is encrypted.

**Scalability & resilience** – Design for high availability and load: run multiple replicas, use auto‑scaling, implement circuit breakers/retries and decouple services via queues. The AKS cluster distributes pods across availability zones and uses health probes and rolling deployments to maximize uptime.

## 2. High‑level architecture

The system consists of a web‑based front‑end and a set of back‑end microservices deployed in Kubernetes. Users access the system via browsers, POS devices or mobile apps. Internal staff (customer service, warehouse associates and purchasing) and external actors (customers, vendors, carriers and tax services) interact with the platform.

A managed **NGINX ingress controller** acts as the API gateway. Requests from clients first hit an Azure Load Balancer, which directs traffic to the ingress controller. The gateway terminates TLS, authenticates using Entra ID, performs rate limiting and routes requests based on the URL path to the appropriate microservice【800680358373136†L404-L412】.

Each microservice runs in its own **Kubernetes Deployment** with a dedicated **Azure SQL Database** or other data store, and can scale independently. Event‑driven workflows use **Azure Service Bus** topics/queues and patterns such as publisher‑subscriber and competing consumers【800680358373136†L345-L356】.

### Domain microservices

* **Product Information Service (PIM)** – maintains a single source of truth for SKUs, attributes, units of measure, kits/bundles and hazardous classifications; exposes CRUD APIs to other services and the front‑end.
* **Inventory & Warehouse Service** – tracks stock across multiple warehouses and bins, updates available‑to‑promise, performs cycle counts, manages RF scanning and handles lot/serial tracking.
* **Order Management Service** – accepts orders from the web store, B2B portal, point‑of‑sale and Amazon, performs payment and fraud checks and orchestrates fulfilment via shipping and warehouse services.
* **Purchasing & Vendor Service** – manages suppliers, vendor scorecards, purchase order (PO) workflows and automatically generates POs using reorder points and economic order quantity.
* **Shipping & Logistics Service** – integrates with carriers for rate shopping, hazmat documentation and cross‑dock/3PL workflows; coordinates pick‑pack‑ship tasks and produces shipping labels.
* **Tax & Accounting Service** – calculates multi‑state sales tax, records journal entries and synchronizes with the general ledger; provides audit trails and ensures compliance.
* **Returns & RMA Service** – standardizes return authorization and disposition codes, manages refunds or exchanges and updates inventory accordingly.
* **Analytics & Alerting Service** – aggregates operational data into a reporting data store, delivers real‑time dashboards and sends KPI alerts to stakeholders.

Each microservice owns its own database to avoid cross‑service dependencies and to support independent scaling. Transient messaging and events are published to **Azure Service Bus**; caches rely on **Azure Cache for Redis** for read‑heavy operations; files and attachments reside in **Azure Blob Storage**【800680358373136†L413-L416】.

## 3. Technology stack

| Layer | Technology | Details |
| --- | --- | --- |
| **Front‑end** | Angular 14+ SPA | Uses TypeScript, RxJS and Angular Material. Consumes REST APIs via the Angular HttpClient module and reads the API base URL from a JSON configuration file that is injected at runtime【85830806945883†L342-L385】. |
| **Back‑end** | ASP.NET Core 6 microservices | Built with Clean Architecture and Domain‑Driven Design. Exposes RESTful APIs and may publish/consume events from Service Bus. |
| **Databases** | Azure SQL Database; Azure Cosmos DB; Azure Cache for Redis | Each microservice stores its own data. Cosmos DB can be used for high‑volume or semi‑structured data. Redis provides caching; Azure Blob Storage stores binary objects. |
| **Messaging** | Azure Service Bus | Topics and queues implement asynchronous workflows and integration events to decouple producers and consumers. |
| **Authentication & Authorization** | Microsoft Entra ID | Provides single sign‑on for users and managed identities for services【800680358373136†L419-L424】. JWT bearer tokens secure API calls, and RBAC is enforced by the API gateway and services. |
| **Containerization** | Docker multi‑stage builds | Use .NET SDK and runtime images to restore and publish the backend API and Node/Nginx images to build and serve the Angular app【85830806945883†L203-L234】【85830806945883†L510-L527】. Images are stored in Azure Container Registry. |
| **Orchestration & Deployment** | Azure Kubernetes Service (AKS) | Host all microservices and the front‑end. Deployments define replicas; services expose them internally; a managed NGINX ingress controller provides external routing【800680358373136†L404-L412】. |
| **CI/CD** | Azure Pipelines or GitHub Actions; Helm | Build, test and push container images to ACR. Helm charts package Kubernetes manifests and deploy microservices to AKS【800680358373136†L433-L441】. |
| **Monitoring & Logging** | Azure Monitor & Application Insights | Collect metrics, logs and traces. Provide dashboards and alerts for performance and business KPIs【800680358373136†L443-L448】. |

## 4. Deployment & infrastructure

* **Container registry** – A private **Azure Container Registry** hosts Docker images for all microservices and the Angular front‑end. AKS authenticates to ACR via managed identity and pulls images at deploy time【800680358373136†L419-L432】.
* **AKS cluster** – The system runs on an AKS cluster with multiple nodes across availability zones to achieve high availability. Each microservice is defined via a **Kubernetes Deployment** with a desired replica count and auto‑scaling rules. A **LoadBalancer** service and **NGINX ingress controller** expose services externally【800680358373136†L404-L412】.
* **Ingress & API gateway** – The managed NGINX ingress controller terminates TLS, validates JWTs, throttles requests and routes traffic to services based on URL paths. This implements the API gateway pattern and allows adding or removing services without impacting the client.
* **Persistent storage** – Managed databases (Azure SQL or Cosmos DB) are provisioned for each service. Azure Disks provide persistent volumes for stateful workloads. Azure Files can be used for shared storage if needed.
* **Networking** – The AKS cluster is deployed in a dedicated virtual network. Network policies restrict east–west traffic; private endpoints secure connections to Azure SQL, Blob Storage, Service Bus and Redis. Only the ingress controller is exposed on a public IP.

## 5. Data flow & integration patterns

**Synchronous HTTP flow** – When a user submits an order, the Angular front‑end sends a POST request to the Order Management service. The ingress controller routes the request via the load balancer to the Order service, which validates the order, reserves inventory and authorizes payment. It returns a confirmation response.

**Asynchronous event flow** – After acceptance, the Order service publishes an OrderPlaced event to an Azure Service Bus topic. Subscriber services (Inventory, Shipping, Analytics) consume the event using competing consumer patterns. Inventory decrements stock levels; Shipping creates a shipment; Analytics updates dashboards. This pattern decouples services and improves resilience and throughput【800680358373136†L345-L356】.

**Batch & integration workflows** – The Purchasing service periodically generates purchase orders based on reorder points and economic order quantity. It integrates with suppliers via EDI or REST to send POs and receive ASNs. Integrations with tax services, carriers and Amazon FBA use dedicated adapters hosted within the respective microservice.

## 6. Security & compliance

* **Identity & access management** – Microsoft Entra ID provides SSO for users and managed identities for services. The ingress controller validates tokens and forwards claims to services. Service‑to‑service calls use managed identities and Azure role assignments for resources【800680358373136†L419-L424】.
* **RBAC enforcement** – Fine‑grained permissions defined in the RBAC matrix are enforced in each microservice. The gateway performs coarse checks (e.g., verifying the user belongs to a particular role), while services enforce domain‑specific access controls.
* **Data protection** – All API calls use TLS 1.2+. Data at rest is encrypted using platform keys or customer‑managed keys. Secrets are stored in Azure Key Vault and injected into pods as environment variables or mounted volumes.
* **Compliance & auditing** – The architecture supports hazardous‑materials handling regulations, multi‑state tax requirements and financial audit trails. Logs and change history are captured and retained for regulatory review.

## 7. Scalability, availability & performance

* **Horizontal scaling** – Each service scales independently by adjusting its replica count. The AKS autoscaler monitors CPU, memory and queue metrics to add or remove pods automatically. Stateless services enable rapid scaling.
* **High availability** – Pods are distributed across multiple zones. Kubernetes liveness/readiness probes restart unhealthy containers. Rolling updates and canary deployments minimise downtime.
* **Performance considerations** – Databases are provisioned with adequate capacity. Caching via Redis reduces latency. Service Bus ensures high‑volume events are processed asynchronously. Resource‑intensive tasks (e.g., cycle counts, forecasting) run as background jobs.

## 8. Monitoring & observability

* **Metrics & logs** – Azure Monitor collects container, node and cluster metrics. Application Insights gathers request traces, dependencies, failures and performance metrics【800680358373136†L443-L448】. Dashboards visualise key indicators such as order throughput, API latency, inventory accuracy and error rates.
* **Tracing & diagnostics** – Distributed tracing is enabled using Application Insights or OpenTelemetry. Correlation IDs propagate across services to follow a request end‑to‑end. Log analytics surfaces anomalies and supports incident investigations.

## 9. Future considerations & Phase 2

* **Demand forecasting & analytics** – Introduce a forecasting service that uses historical sales and seasonality to predict future demand. The service may leverage Azure Machine Learning and publish forecast data into the Analytics service.
* **Promotion engine & contract pricing** – Add services to manage promotional campaigns, coupon codes and customer‑specific pricing. These integrate with the Order service to apply discounts at order time.
* **Light manufacturing & kitting** – Introduce a kitting service that manages bills of materials and assembly instructions. The service interfaces with the Inventory and Order services to deduct components and produce finished goods.
* **3PL integration & EDI** – Develop adapters for third‑party logistics providers (3PLs) to exchange shipment and inventory updates. Implement EDI for large suppliers to automate purchase orders, invoices and ASNs.
* **Customer service console** – Build a console for customer service representatives that aggregates order history, shipment tracking, returns status and SLA timers. The console subscribes to events and queries the underlying microservices in real time.

This architecture provides a scalable, secure and extensible foundation for PineCone Pro Supplies’ ERP/IMS. It leverages proven Azure services and microservices best practices to deliver a unified platform that meets current requirements and supports future growth.