Building a Modern Enterprise Ecosystem

2019-05-18

Jarkko Enden
Head of Technology
Nortal

Version	Date	Description	Author
1.0	2018-10-02	First version sent to comments round	JE
1.1	2018-10-24	Updated high level architecture diagram	JE
1.2	2018-10-29	Updated diagrams	JE
1.9	2019-05-14	Updated to new Domain-Driven Design based architecture, added Service Layer diagram	JE

Table of Contents

1. Introduction

Many companies are struggling to cope with today’s fast-paced and ever-changing business landscape. Current IT systems are often seen as a burden and a blocker that slows down business transformation. In some cases, this can lead to an almost complete inability to introduce significant new technology without a justifiable fear that it will break down the current system landscape comprising of legacy systems. Neither a Big Bang nor a gradual transformation may seem a viable option.

This is often related to several separate, but interconnected issues:

- Many systems are near the end of their life cycle
- Systems are overlapping in functionality or even lack a clearly defined purpose
- Products are heavily customized and can no longer be upgraded without significant costs and risks
- Systems integrations are performed point-to-point or orchestrated poorly
- Various hacks and workarounds are used to get around the limitations of the current systems

While there is no silver bullet to magically get out of this bind, with the right strategy, partnership and tools technology can become a key enabler that drives business change and introduces new opportunities that have seemed too far-fetched to date.

Nortal proposes a new solution architecture based on a truly modular ecosystem approach. The ecosystem allows gradual and continuous improvement while bringing concrete benefits to your business. We firmly believe that an open, efficient software architecture is a necessary ingredient in being successful in the modern business world.

Our mission is to simplify and optimize complex processes to create a **Seamless Society**.

Key elements of the solution:

- Open ecosystem, free of vendor lock
- Truly modular architecture to reflect changing business needs
- Flexible and close co-operation between the customer and all ecosystem vendors

2. Ecosystem Overview

2.1. Product-friendly Architecture

While globally sold products and multi-tenant cloud services come with their own baggage, building a complex ecosystem without relying on any existing market-ready solutions is too slow, expensive and risky. We love microservices and they are an essential part of our approach, but they are not an answer for everything.

Systems used for planning and running production vary across industries. In the manufacturing industry, the core production systems are Manufacturing Execution Systems (MES), while in healthcare they are Electronic Medical Record (EMR) systems or Hospital Information Systems (HIS). The same open ecosystem approach is valid regardless of the systems running your production, so you can keep reading.

There are very few industries where sales and marketing are not an essential part of the business. Hence, a robust CRM is one of the key elements of a thriving ecosystem. Often the CRM has been bought as an after-thought and only used by sales people as a standalone or poorly integrated tool. To unlock the power of a modern CRM, such as Microsoft Dynamics or Salesforce, the CRM needs to be re-introduced as a key element driving the ecosystem.

ERP systems may have a bad or legacy rep in some circles, but businesses still need to run their finance in a regulated way. Using microservices and custom code to build all finance modules is an endeavor that sounds interesting, but is not one we recommend embarking on. A solid ERP product (or a small collection of separate products) is still the best option for the finance stuff, both in the short and long term.

The chosen products and cloud services provide a comprehensive and robust foundation for the ecosystem, but it is also essential to make these products and modules work together flawlessly. Nortal’s Service Layer architecture provides layers between the different products and business domains and acts as dynamic glue that makes the ecosystem thrive. And yes, it embraces and uses microservices.

Service Layers provide the necessary tools for avoiding complex tailoring of the products and enable adding, upgrading and removing products and modules without introducing breaking changes to other ecosystem components. The architecture also provides a uniform and well-defined integration experience for all applications, which makes it easier to introduce new applications to the ecosystem. When these are translated to business benefits, this approach to IT architecture enables better adaptation to existing and future business needs, provides vastly improved lifecycle management and greatly reduces the risk of vendor lock-in.

2.2. Key Business Values

The key to successfully realizing the benefits of the new solution does not depend only on selecting specific products or models, but on understanding of a holistic approach in implementing a business-critical system based on correctly described problem areas. With the ecosystem solution, we expect to see various business and ICT benefits:

- **Combine business continuity with agility**
    - Reach agility in reacting to changes in the business environment and a low threshold for testing new business ideas in practice
    - Small agile steps towards the future ensures almost instant ROI from the first steps

  - Solve business problems in accelerated time by introducing new products and modules
- **Adapt to existing and future business needs**
  - Concentrate on the future and manage your business proactively instead of reacting to issues afterwards
  - Add, upgrade and replace parts of the ecosystem as needed
  - Use multiple vendors or build things independently of vendors
- **Improve IT lifecycle management**
  - Upgrade and swap products without interruptions to other ecosystem applications
  - Take full control of the IT architecture via standardized and open integration and data extraction interfaces
  - Avoid Big Bang transformations of end-of-life legacy systems
- **Reduce risk of vendor lock-in**
  - Block any vendor from rising towards a monopoly position towards the customer
  - Reach cost savings and better service due to increased competition between vendors
- **Increase automation levels**
  - Introduce advanced analytics and AI solutions in a controlled manner
  - Allow resources to concentrate on value adding work instead of mundane tasks and manual information transfer inside the organization

2.3. Transformation to Future State

The key to a successful ramp up in a large, business-critical project is the skill to divide the implementation, testing and deployment into sufficiently small, manageable roadmap of units. As the units in the business development project get bigger, the amount of work that does not add value to the final solution tends to grow exponentially. At the same time, the threshold for confirming that the results of the work are fit for production use rises.

Successful implementation requires agile development practices, which enable the solution to be constructed either in short Sprints or in continuous development and release trains. Agile methods make it possible to ensure that the direction, quality, and speed of the transformation meet the predefined goals throughout the project. These practices, combined with DevOps-based continuous integration and deployment, allow small change sets to be published frequently. This approach mitigates risks and reduces training needs, friction and change resistance within the organization.

Ecosystem development can be divided to work packages of various sizes. Instead of forcing the same development process for each work package and team, we recommend allowing for some variations in the methods. The same processes may not be convenient for product configuration -oriented and custom software development work. Naturally appropriate governance models must still be set for the ecosystem development program.

2.4. Vendor Ecosystem

After the core components are in place and the foundation is robust, you are able to introduce a vendor ecosystem for further development of the solution. By providing a uniform user experience for all application developers (API architecture, authorization logic, error handling, etc.) and novel ways of integrating with the core products on all levels, the innovation power of startups and smaller vendors can be leveraged in unforeseen ways.

3. Solution Architecture

3.1. Domain-Driven Design and API-Driven Architecture

The following principles are used as general guidelines for developing the ecosystem. The guiding principle is that all modern applications must be driven by APIs. Data cannot be hidden within any individual system.

One of the keys to a successful and maintainable architecture is Domain-Driven Design (DDD). Domain modeling is often overlooked and things are built in silos to fit the need at hand. This approach will lead to difficult problems and large refactoring needs later on in the ecosystem lifecycle.

Further reading on Domain-Driven Design:

- https://en.wikipedia.org/wiki/Domain-driven_design
- https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/microservice-ddd-cqrs-patterns/ddd-oriented-microservice

Core principles:

1. PRODUCT-BASED ARCHITECTURE

- Product-based but product-agnostic architecture
- Leverage proven COTS products where applicable
- Favor products that expose high-quality APIs
- Avoid complex tailoring of COTS products

2. MICROSERVICES ARCHITECTURE

- Smart endpoints and dumb pipes
- Lightweight API Management over heavy ESB
- Keep number of services manageable
- Extend products beyond their original capabilities

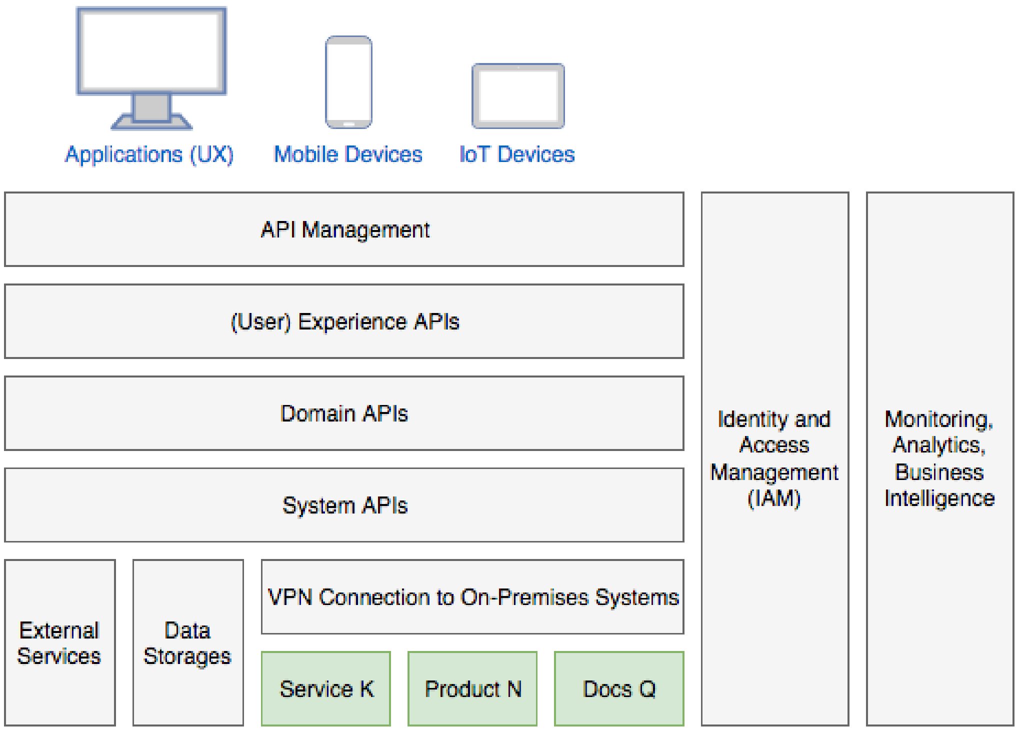
3. DOMAIN-BASED ARCHITECTURE

- Define architectural domains
- Expose Domain APIs
- Speak the same language as API consumers
- Hide the complexity of underlying products

4. CLOUD-BASED ARCHITECTURE

- Do not fear the cloud, embrace it
- Highly-certified best-of-breed data centers
- Improved flexibility, productivity and stability
- Enhanced security against cyber attacks

3.1.1. Layered API Architecture



System APIs are the native APIs that provide access to core back-end systems, such as on-premises legacy systems, external services, and different data storages. The idea is to provide a clean and reusable API surface for the API user, while at the same time securing the user from the complexity or changes to the underlying systems. System APIs are for internal use only and point-to-point integrations using System APIs should be avoided.

Domain APIs are responsible for modelling and encapsulating the business entities and processes of the organization. Domain APIs shape the data by orchestrating (split, aggregate, route etc.) and calling one or more System APIs. Domain APIs are designed to be generic and can be consumed by both internal and external systems. Domain APIs are produced by the Service Layers.

Experience APIs are designed for a specific user experience or audience, such as a customer portal, mobile application or IoT device. Experience APIs are the means by which data can be reconfigured so that it is most easily consumed by its intended audience.

3.1.2. Domain-Driven Design –based microservice architecture

When you are basing your layered architecture on DDD principles, the dependencies between layers should follow the principles set in the following diagram:

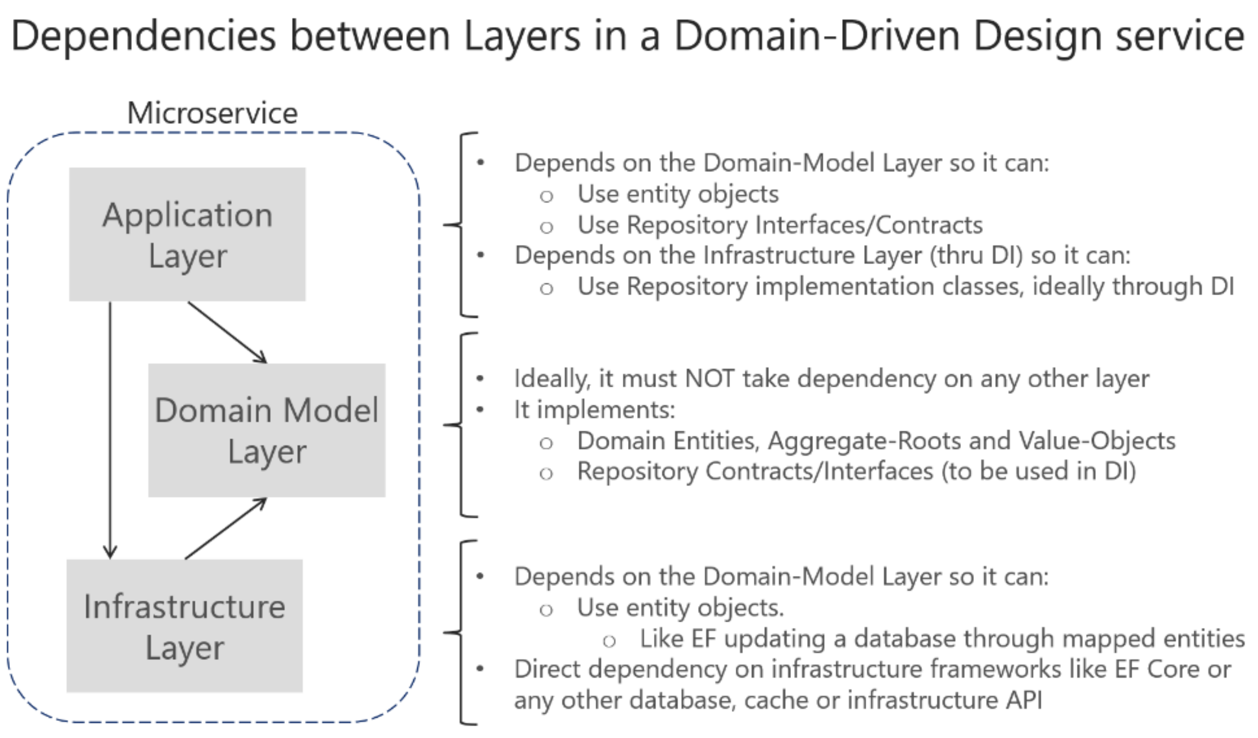

Dependencies between Layers in a Domain-Driven Design service
Microservice
Application Layer
Domain Model Layer
Infrastructure Layer
• Depends on the Domain-Model Layer so it can:
o Use entity objects
o Use Repository Interfaces/Contracts
• Depends on the Infrastructure Layer (thru DI) so it can:
o Use Repository implementation classes, ideally through DI
• Ideally, it must NOT take dependency on any other layer
• It implements:
o Domain Entities, Aggregate-Roots and Value-Objects
o Repository Contracts/Interfaces (to be used in DI)
• Depends on the Domain-Model Layer so it can:
o Use entity objects.
o Like EF updating a database through mapped entities
• Direct dependency on infrastructure frameworks like EF Core or any other database, cache or infrastructure API

Diagram: Internal layers and dependencies in a DDD-based microservice

Source: https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/microservice-ddd-cqrs-patterns/ddd-oriented-microservice

3.1.3. API Documentation

OpenAPI standard is used for describing and documenting custom-developed APIs. API documentation is online and based on source code (i.e. always up to date).

Swagger and API Management tools offer a web-based user interface for quick testing of the APIs.

3.1.4. Benefits of Domain-Driven Design and API-Driven Architecture

By combining Domain-Driven Design with API-Driven Architecture, you are able to:

- Identify different business domains and produce well-defined interfaces (Domain APIs) for the domains.
- Identify dependencies between different domain concepts and entities and define bounded contexts that serve as a blueprint for you microservice-based architecture.
- Make the different products and other ecosystem elements as loosely coupled as possible. As long as the Domain APIs remain intact, the business will flow regardless of future product selections.
- Provide business entities modelled according to your business rules instead of the rules of individual products. This way we can better serve your business and add new components to the architecture.
- Base the solution on a robust product-based foundation. One key benefit of "Commercial Of-The-Shelf" (COTS) products is that they are reliable as long as they do not contain modifications. Internal customization of the products should be avoided. Instead of forcing a product to perform your custom business logic, we can place it in the Service Layers, keeping the product intact and ready for upgrades.

3.2. Service Layer Architecture

Proven products are used where applicable, but customized **Service Layers** are an important part of our ecosystem approach. Service Layers offer a way to capture and define different domains within the business, and make them collaborate efficiently. In addition to integrating different ecosystem domains and components, Service Layers offer a home for common data models and customer-specific business logic. While product customization has always been problematic, it has become increasingly difficult and cumbersome with modern cloud-based solutions.

Service Layers are built using robust and proven backend technologies. Data models and business logic are grouped into logical units and implemented as microservices. Common utility libraries are distributed via package managers so they can be leveraged in all microservices. Continuous integration and deployment pipelines are set up early to enable smooth development and deployments. API Management services are used for centralized management of API documentation, access control, throttling rates and service endpoints. Infrastructure-as-Code (IaC) solutions are used for automating all aspects of cloud environment setup.

The diagram below depicts both an enterprise architecture for a specific scenario (healthcare ecosystem), as well as the internal architecture of microservices provided by the Service Layer. The scenario contains two different microservices with inter-service dependencies, core products used as the main building blocks of the ecosystem and a data analytics layer.

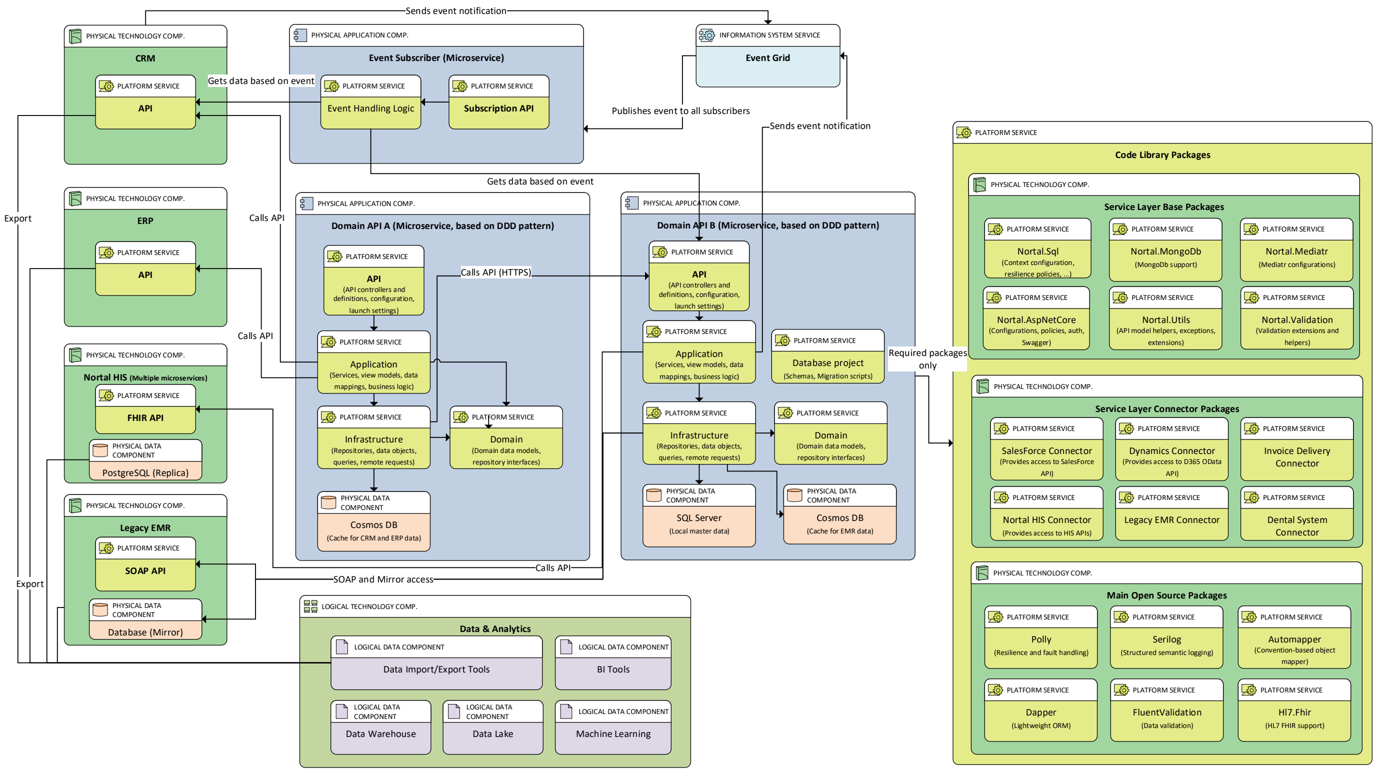

Sends event notification
PHYSICAL TECHNOLOGY COMP.
CRM
PLATFORM SERVICE
API
Gets data based on event
PHYSICAL APPLICATION COMP.
Event Subscriber (Microservice)
Event Handling Logic
Subscription API
INFORMATION SYSTEM SERVICE
Event Grid
Publishes event to all subscribers
Sends event notification
Gets data based on event
Export
Calls API
ERP
Domain API A (Microservice, based on DDD pattern)
API
(API controllers and definitions, configuration, launch settings)
Calls API (HTTPS)
Application
(Services, view models, data mappings, business logic)
Infrastructure
(Repositories, data objects, queries, remote requests)
Domain
(Domain data models, repository interfaces)
PHYSICAL DATA COMPONENT
Cosmos DB
(Cache for CRM and ERP data)
Domain API B (Microservice, based on DDD pattern)
Database project
(Schemas, Migration scripts)
SQL Server
(Local master data)
Cosmos DB
(Cache for EMR data)
Required packages only
Nortal HIS (Multiple microservices)
FHIR API
PostgreSQL (Replica)
Legacy EMR
SOAP API
Database (Mirror)
Calls API
SOAP and Mirror access
Export
LOGICAL TECHNOLOGY COMP.
Data & Analytics
LOGICAL DATA COMPONENT
Data Import/Export Tools
BI Tools
Data Warehouse
Data Lake
Machine Learning
Code Library Packages
Service Layer Base Packages
Nortal.Sql
(Context configuration, resilience policies, ...)
Nortal.MongoDb
(MongoDb support)
Nortal.Mediatr
(Mediatr configurations)
Nortal.AspNetCore
(Configurations, policies, auth, Swagger)
Nortal.Utils
(API model helpers, exceptions, extensions)
Nortal.Validation
(Validation extensions and helpers)
Service Layer Connector Packages
SalesForce Connector
(Provides access to SalesForce API)
Dynamics Connector
(Provides access to D365 OData API)
Invoice Delivery Connector
Nortal HIS Connector
(Provides access to HIS APIs)
Legacy EMR Connector
Dental System Connector
Main Open Source Packages
Polly
(Resilience and fault handling)
Serilog
(Structured semantic logging)
Automapper
(Convention-based object mapper)
Dapper
(Lightweight ORM)
FluentValidation
(Data validation)
Hl7.Fhir
(HL7 FHIR support)

Diagram: Service layer internal architecture and sample enterprise architecture

3.2.1. Service Layer development principles and core features

- DevOps (CI and CD) all the way
- Microservices architecture with a manageable number of services

- Reusable base libraries published via a package manager
- Asynchronous internal pipelines
- DI (Dependency Injection) -friendly architecture
- Internal architecture according to Domain-Driven Design principles
- Clear responsibilities between different layers
- Convention-based model mapper for data transformations
- Fast and robust in-memory and out-of-memory caching
- Policy-based resilience and error handling
- Support for different authentication and authorization providers
- Support for globalization
- Support for data versioning and auditing
- Support for semantic logging and different log providers
- Online auto-generated API documentation (Swagger/OpenAPI)
- Comprehensive test automation
- Infrastructure-as-Code (IaC) tools used when suitable (namely Terraform in cloud solutions)

3.3. Cloud Architecture

Benefits of a cloud-based solution compared to an on-premise solution:

- More robust and secure operations environment (highly certified data centers and personnel)
- Faster setup time
- Automatic behind-the-scenes version upgrades
- Practically infinite scalability
- Lower TCO (Total Cost of Ownership)
- Predictable service fees, no unexpected maintenance costs
- Advanced PaaS services that cannot be leveraged in on-premise solutions

Connectivity between cloud and on-premises can be built to achieve a hybrid solution that can access data from both cloud and on-premise.

Different cloud service types are discussed in the next chapters.

3.3.1. Infrastructure-as-a-Service (IaaS)

With IaaS, you provision the virtual machines (VMs) that you need, along with associated network and storage components. Then you deploy software and applications you want onto those VMs. This model is the closest to a traditional on-premises environment, except that the cloud provider manages the infrastructure. You manage the individual VMs. While other service types are on the rise, IaaS is still a useful scenario for many areas, especially when combined with container technologies.

3.3.2. Platform-as-a-Service (PaaS)

PaaS provides a managed hosting environment, where you can deploy your application without needing to manage VMs or networking resources. Instead of creating and managing individual VMs, you define an initial configuration and the service will provision, configure, and manage the requested resources. API Management and database-as-a-service are examples of PaaS services.

Our approach is to leverage PaaS services as much as possible. VM-based IaaS will only be used when applicable PaaS services do not exist or their use is not suitable for technical or business reasons.

3.3.3. Functions-as-a-Service (FaaS)

FaaS goes even further in removing the need to worry about the hosting environment. Instead of creating compute instances and deploying code to those instances, you simply deploy your code, and the service automatically runs it. You do not need to administer the compute resources. These services make use of serverless architecture, and seamlessly scale up or down to whatever level necessary to handle the traffic. Azure Functions and AWS Lambda are examples of FaaS service.

FaaS services are still quite new and evolving rapidly. More experience related to versioning, configuration management and other core areas needs to be gathered before utilizing them in business critical components. FaaS is on the technology radar, but will not be used extensively until they reach a more mature state.

3.3.4. Software-as-a-Service (SaaS)

SaaS provides a complete software solution that you purchase on a pay-as-you-go basis from a cloud service provider. You rent the use of an app for your organization, and your users connect to it over the Internet, usually with a web browser. All of the underlying infrastructure, middleware, app software, and app data are located in the service provider’s data center. The service provider manages the hardware and software, and with the appropriate service agreement, will ensure the availability and the security of the app and your data as well. Power BI is an example of a SaaS service.

3.4. Systems Integrations

The number of systems integrations in large enterprises tends to be very high. The traditional solution for managing the integrations has been to purchase a centralized ESB (Enterprise Service Bus) and concentrate all integration logic there. However, integrations are more complex in the real world than in sales pitches, which leads to the ESB becoming the default location for business logic that was not originally planned to be there. Slowly the ESB evolves into an uncontrollable monolithic monster that drives your business.

Modern cloud-based solutions avoid the monolithic ESB solution by providing the functionalities in smaller packages. Our approach is to leverage these lightweight-ESB PaaS components, such as API Management, for what they handle well, but place the business logic in the Service Layers.

API Management is used to manage all APIs in the ecosystem. APIs can be published to external parties, partners and internal developers. API Management provides a portal for API publishers as well as a self-service portal for developers. In addition to managing and documenting the APIs, API Management acts as a centralized gateway that controls access to the APIs. It supports modern authentication and authorization standards for securing APIs.

Cloud-based message queues, such as Azure Service Bus, offer reliable messaging between ecosystem applications. They support asynchronous operations, message brokering and publish/subscribe capabilities.

To avoid polling, an event routing and delivery service, such as Azure Event Grid, is recommended. HTTP-based reactive event handling helps to build efficient solutions through intelligent filtering and routing. The solution is ideal for real-time reactive scenarios where multiple workflows are run in parallel.

Azure Logic Apps provides a wide range of integration tools from job schedulers to visual workflow editors and on-premise connectors. It is the closest thing to a traditional ESB, but with a key difference: you can choose when you want to use it and only pay for it when you need it.

3.5. Logging and Monitoring

Performance and other issues will have a major negative impact to business. With multiple components and frequent releases, issues are bound to happen at some point. The aim is to be able to predict problems before they occur or find and fix problems before users even notice them.

This requires advanced diagnostics capabilities and a centralized logging solution. Azure Application Insights and Log Analytics can be used as the foundation for the monitoring platform. On-premise logs can also be forwarded there for further analysis.

It is important to set ground rules for ecosystem applications. All applications should be able to send logs to the centralized logging as early as possible.

3.6. Updates and Version Upgrades

One of the benefits related to the new ecosystem and associated DevOps practices is that they allow for frequent updates and upgrades. Update processes and responsibilities for Nortal's software can be fully decided during the project. For multi-tenant cloud services, such as Microsoft Dynamics 365, updates and version upgrades are managed by the cloud service vendor.

Updates are planned in a way that cause minimal downtime to production systems. Any update or upgrade that causes downtime is planned meticulously and communicated clearly to all stakeholders. DevOps practices are leveraged in development, but all production updates are subjected to a strict acceptance procedure.

3.7. Reporting and Business Intelligence

Existing reports built against legacy systems can still be used, but we propose to build new business intelligence capabilities on **Microsoft's Power BI** platform. Users can create modern reports and publish them for consumption on the web and across mobile devices. Everyone can create personalized dashboards with a 360-degree view of their business.

4. Case study: Healthcare Ecosystem

4.1. Introduction

Due to major changes in the health care sector in Finland, there are new market opportunities for private healthcare companies, but due to the current IT landscape, which is mainly based on monolithic legacy products, the enterprise architecture no longer answers healthcare providers' current or future business needs.

Many healthcare systems were originally built for billing purposes and then upgraded to offer structured electronic medical records, e-booking, customer relationship management, integrations with national and regional healthcare systems, etc. The solutions are either completely monolithic or only internally modular. Very few systems support a truly modular approach where the system gives room to other vendors' solutions and does not try to master all data and business processes.

To solve customers' current issues, reach key objectives and prepare for the future, Nortal offers a new ecosystem approach for healthcare. It is based on proven products, limited product customizations and the Service Layer approach for complementing the products and building robust integrations.

4.2. Solution Overview

We are able to work with any products that provide some kind of APIs (or database access), but we recommend leveraging modern products that provide comprehensive well-documented REST APIs. The core products chosen for our largest customer cases in healthcare are Nortal HIS, Microsoft Dynamics 365 Sales and Customer Engagement, and Microsoft Dynamics 365 Finance and Operations.

Nortal HIS is the leading hospital information system in the Baltic countries. It is a comprehensive solution for running a clinic or hospital and has been used in production since 2007. The system provides fully web-based user interfaces and FHIR-compliant REST APIs for managing data and orchestrating processes. Nortal has been able to combine several technology generations in a single solution. The first 4th generation modules are in production and we are currently developing the solution further to answer Finnish and global market needs.

Microsoft Dynamics 365 for Sales and Customer Engagement is a cloud-based CRM/xRM system that supports management of accounts, contacts, products, sales agreements and their different relations. It provides tools for sales pipeline management, including prospect to cash pipelines, as well as advanced reporting via Power BI. In addition to out-of-the-box features, it offers a powerful dynamic data and forms engine that can be used to build customer-specific solutions without touching or overriding original product code.

Microsoft Dynamics 365 for Finance and Operations is a cloud-based, modular Enterprise Resource Planning (ERP) system for finance management, project accounting, supply chain and warehouse management, manufacturing, retail operations and talent management. It is an integrated system where everyday transactions automatically produce accounting transactions, greatly reducing the amount of manual work.

Both Microsoft Dynamics 365 for Sales and Microsoft Dynamics 365 for Finance and Operations are part of the Microsoft 365 product line.

4.3. High-level Enterprise Architecture

High-level enterprise architecture for a healthcare provider is depicted in the diagram below.

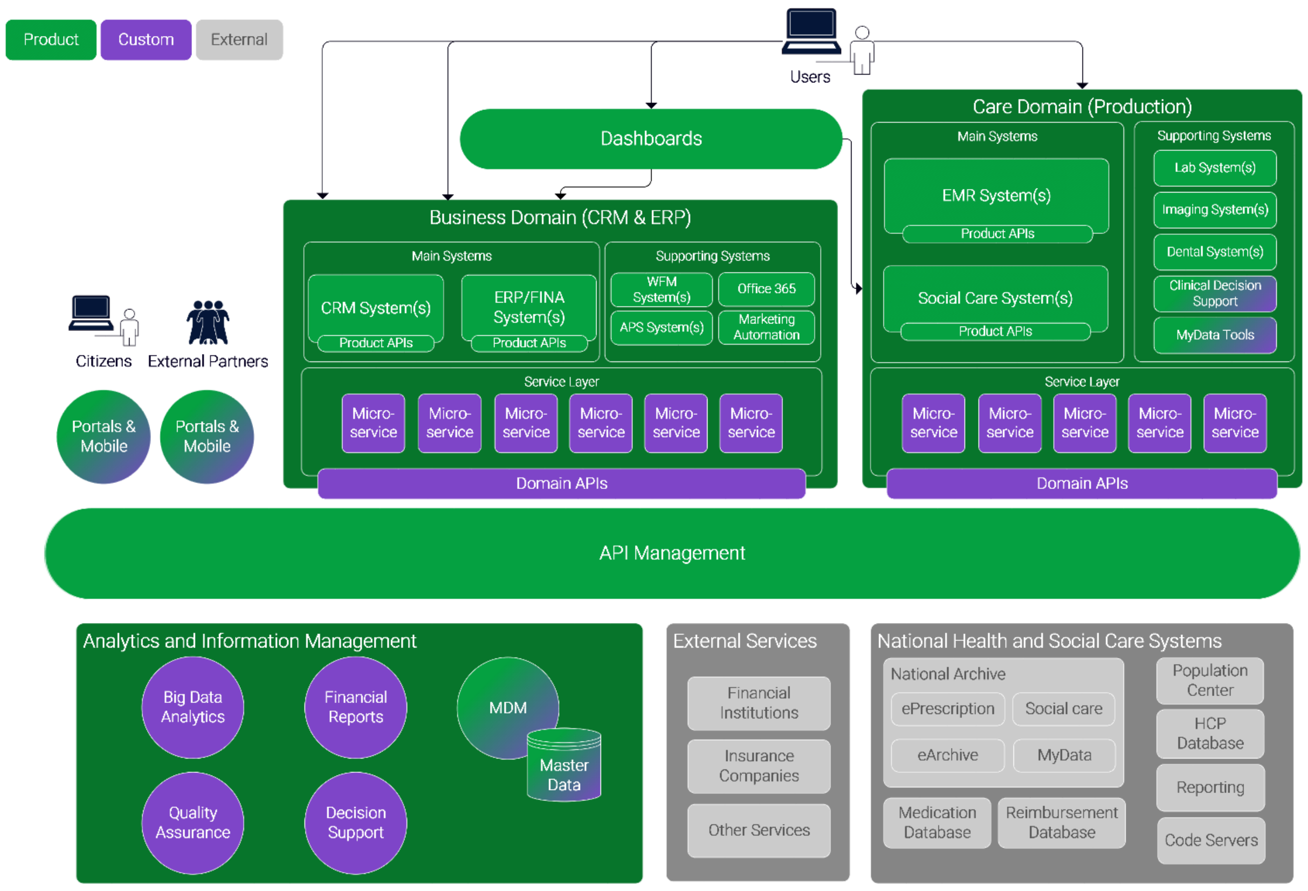


Diagram: Ecosystem Enterprise Architecture

Green boxes depict Nortal's core offering. Modules indicated by purple boxes can be provided together with our partners.

4.4. Functional Architecture

The diagram below depicts our approach for dividing the responsibilities between the different systems.

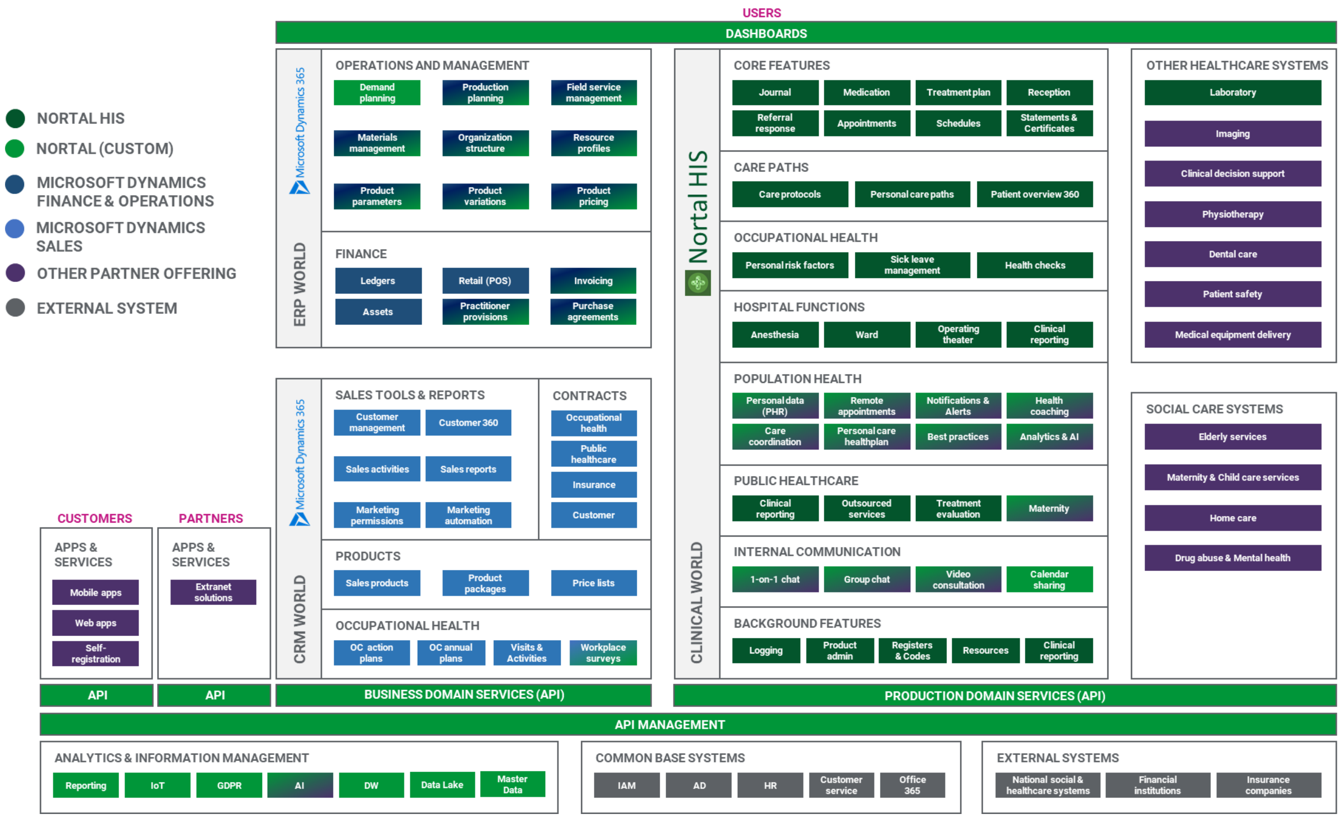


Diagram: Ecosystem Functional Architecture

4.5. Solution Details

For more information on Nortal's novel healthcare ecosystem or other solutions, please contact your local Nortal representative.