**Unit 3: Introduction to Enterprise Application architectures & Application Architecture Patterns:**

**I. Layer Architecture**

The most common architecture pattern is the layered architecture pattern, otherwise known as the n-tier architecture pattern. This pattern is the de facto standard for most Java EE applications and therefore is widely known by most architects, designers, and developers. The layered architecture pattern closely matches the traditional IT communication and organizational structures found in most companies, making it a natural choice for most business application development efforts.

Pattern Description

Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database ([Figure 1-1](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html#sapr_0101_img)). In some cases, the business layer and persistence layer are combined into a single business layer, particularly when the persistence logic (e.g., SQL or HSQL) is embedded within the business layer components. Thus, smaller applications may have only three layers, whereas larger and more complex business applications may contain five or more layers.

Each layer of the layered architecture pattern has a specific role and responsibility within the application. For example, a presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing specific business rules associated with the request. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request. For example, the presentation layer doesn’t need to know or worry about how to get customer data; it only needs to display that information on a screen in particular format. Similarly, the business layer doesn’t need to be concerned about how to format customer data for display on a screen or even where the customer data is coming from; it only needs to get the data from the persistence layer, perform business logic against the data (e.g., calculate values or aggregate data), and pass that information up to the presentation layer.



Figure 1-1. Layered architecture pattern

One of the powerful features of the layered architecture pattern is the separation of concerns among components. Components within a specific layer deal only with logic that pertains to that layer. For example, components in the presentation layer deal only with presentation logic, whereas components residing in the business layer deal only with business logic. This type of component classification makes it easy to build effective roles and responsibility models into your architecture, and also makes it easy to develop, test, govern, and maintain applications using this architecture pattern due to well-defined component interfaces and limited component scope.

Key Concepts

Notice in [Figure 1-2](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html#sapr_0102_img) that each of the layers in the architecture is marked as being closed. This is a very important concept in the layered architecture pattern. A closed layer means that as a request moves from layer to layer, it must go through the layer right below it to get to the next layer below that one. For example, a request originating from the presentation layer must first go through the business layer and then to the persistence layer before finally hitting the database layer.



Figure 1-2. Closed layers and request access

So why not allow the presentation layer direct access to either the persistence layer or database layer? After all, direct database access from the presentation layer is much faster than going through a bunch of unnecessary layers just to retrieve or save database information. The answer to this question lies in a key concept known as layers of isolation.

The layers of isolation concept means that changes made in one layer of the architecture generally don’t impact or affect components in other layers: the change is isolated to the components within that layer, and possibly another associated layer (such as a persistence layer containing SQL). If you allow the presentation layer direct access to the persistence layer, then changes made to SQL within the persistence layer would impact both the business layer and the presentation layer, thereby producing a very tightly coupled application with lots of interdependencies between components. This type of architecture then becomes very hard and expensive to change.

The layers of isolation concept also means that each layer is independent of the other layers, thereby having little or no knowledge of the inner workings of other layers in the architecture. To understand the power and importance of this concept, consider a large refactoring effort to convert the presentation framework from JSP (Java Server Pages) to JSF (Java Server Faces). Assuming that the contracts (e.g., model) used between the presentation layer and the business layer remain the same, the business layer is not affected by the refactoring and remains completely independent of the type of user-interface framework used by the presentation layer.

While closed layers facilitate layers of isolation and therefore help isolate change within the architecture, there are times when it makes sense for certain layers to be open. For example, suppose you want to add a shared-services layer to an architecture containing common service components accessed by components within the business layer (e.g., data and string utility classes or auditing and logging classes). Creating a services layer is usually a good idea in this case because architecturally it restricts access to the shared services to the business layer (and not the presentation layer). Without a separate layer, there is nothing architecturally that restricts the presentation layer from accessing these common services, making it difficult to govern this access restriction.

In this example, the new services layer would likely reside below the business layer to indicate that components in this services layer are not accessible from the presentation layer. However, this presents a problem in that the business layer is now required to go through the services layer to get to the persistence layer, which makes no sense at all. This is an age-old problem with the layered architecture, and is solved by creating open layers within the architecture.

As illustrated in [Figure 1-3](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html#sapr_0103_img), the services layer in this case is marked as open,  meaning requests are allowed to bypass this open layer and go directly to the layer below it. In the following example, since the services layer is open, the business layer is now allowed to bypass it and go directly to the persistence layer, which makes perfect sense.

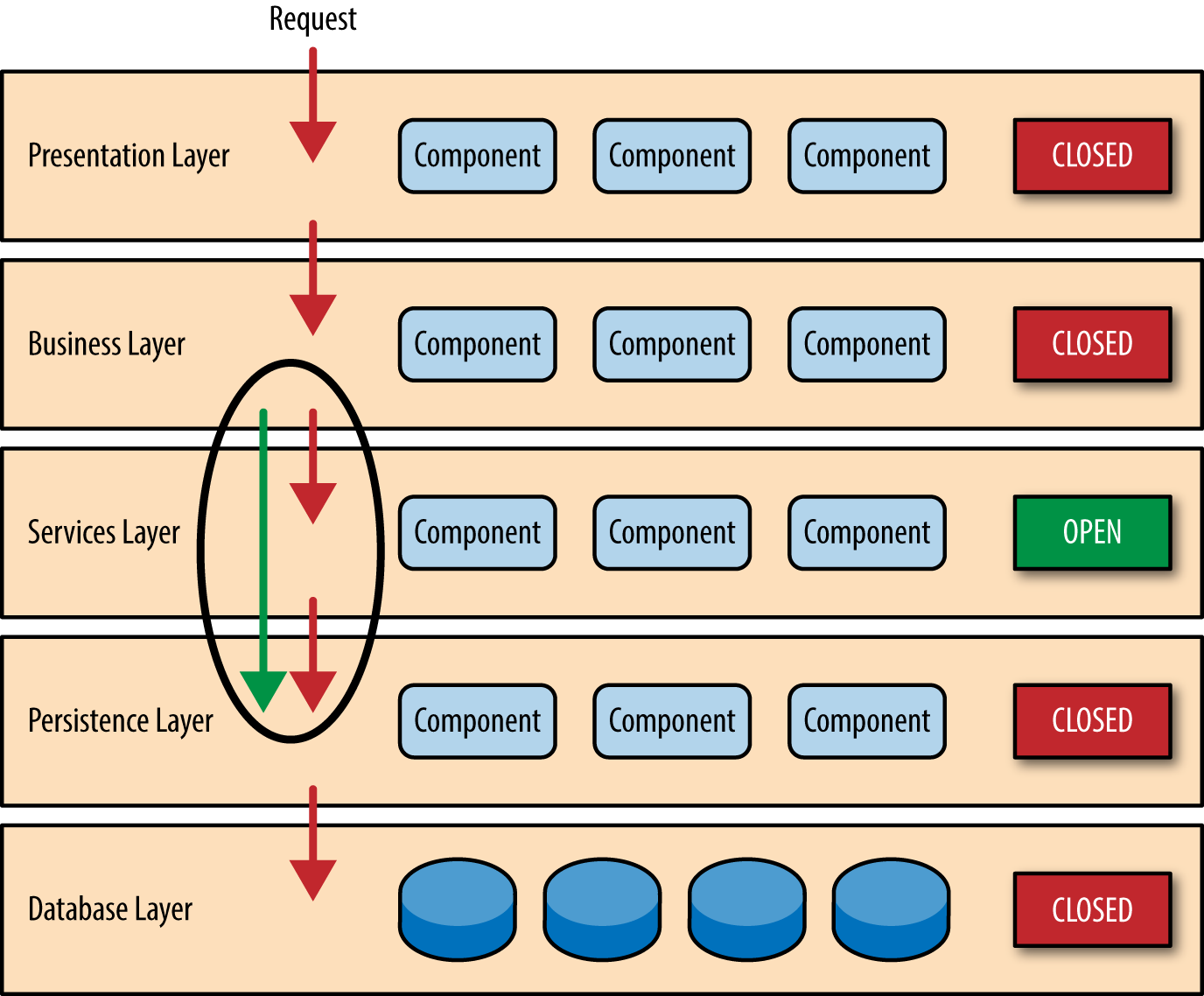


Figure 1-3. Open layers and request flow

Leveraging the concept of open and closed layers helps define the relationship between architecture layers and request flows and also provides designers and developers with the necessary information to understand the various layer access restrictions within the architecture. Failure to document or properly communicate which layers in the architecture are open and closed (and why) usually results in tightly coupled and brittle architectures that are very difficult to test, maintain, and deploy.

**II. Event driven Architecture**

Event-driven architecture (EDA) is a design paradigm in which a software component executes in response to receiving one or more event notifications.

What is it?

﻿Event-driven architecture models your business systems as a flow of events — when an important business event happens, your systems are alerted to that change of state. A simple example could be a customer changing their address: once that state change is registered, your billing systems get notified of the new address. This is in stark contrast to traditional request-based architectures.

A major benefit of this architectural pattern is that it is both scalable and relatively easy to change. EDA’s inherently loosely coupled nature means that it’s relatively easy to make changes in one particular part of your systems, without breaking anything else.

A well-designed EDA will be based on events that are meaningful to the business. The events could be triggered by user activity, external inputs, such as sensor activity, or outputs from an analytics system. What’s important is the way you define those events, so that you’re capturing something important to your organization.

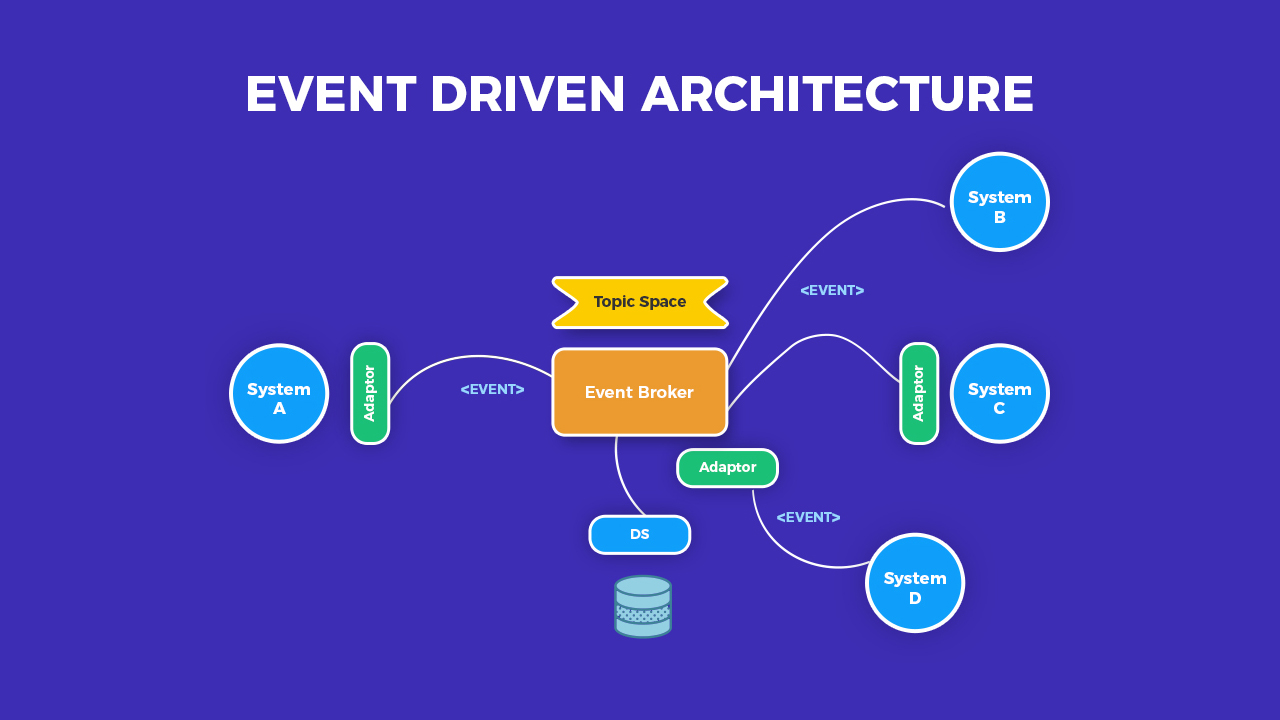
By basing your designs on these triggering events, you gain flexibility; you’re able to add new behaviours without having to redesign the entire system.

How is it being used?

The easiest way to build an event-driven architecture may be to design your systems that way from the outset. Many of today’s renowned digital-native platforms embrace event-driven designs.

But not every organization has the luxury of starting afresh. And it’s perfectly possible to build EDA on top of legacy infrastructure. Many organizations we’ve worked with start with a single EDA-based project and expand from there, as they realize more benefits.

And EDA is compatible with efforts to become an intelligent enterprise. EDA makes it easier to automate actions based on outputs from your analytics systems. And because you define what business events are important to your enterprise, EDA is well suited to incorporate real-time information from a variety of sources, for instance if you wanted to deploy an array of Internet of Things sensors.



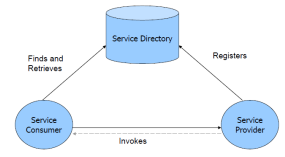
**III. Service oriented Architecture**

Service-Oriented Architecture built applications based on services and loosely-coupled architecture. So they can be reused. A major benefit of Service-Oriented Architecture is that it delivers enterprise agility, by enabling rapid development and modification of the software that supports the business.

Service-Oriented Architecture defined in various ways, “Service-oriented architecture is a Client/Server design approach in which an application consists of software services and software service consumers (also known as clients or service requesters). SOA differs from the more general client/server model in its definitive emphasis on loose coupling between software components, and in its use of separately standing interfaces” (Gartner).

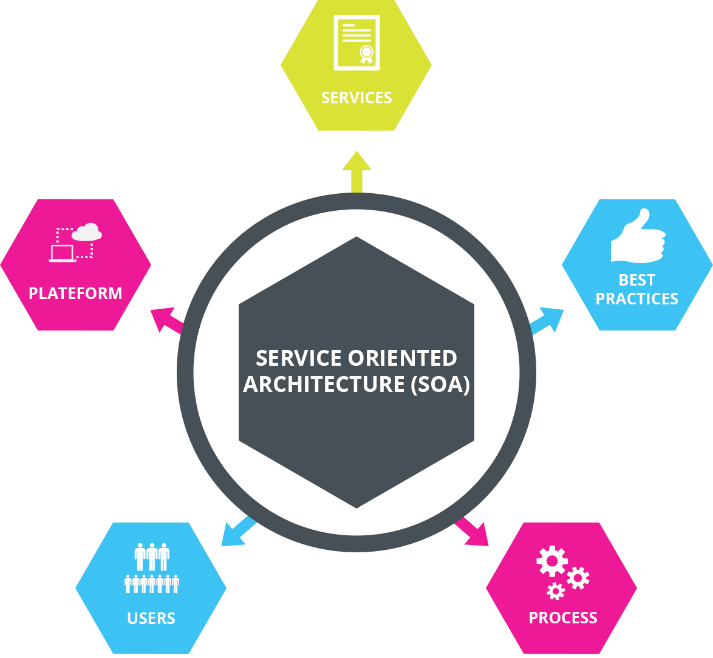
Also Service-Oriented Architecture enable flexible, Combined Business Processes, and business process optimization and the Real Time Enterprise (RTE)

SOA is method of design, development and management of both application and the software infrastructure where: all software is organized into business services that are network accessible and executable, service interface are based on public standards for interoperability. Service-Oriented Architecture is an architectural approach that allows distributed deployment by expose enterprise data and business logic as loosely coupled, discoverable, structured, standards-based, coarse-grained, stateless units of functionality called services. Furthermore allows reusability by choose a services provider and access to existing resources exposed as services. By allowing reusing the existing applications Service-Oriented Architecture enables enterprise to influence existing investments. Another importance is composability by allowing assemble new processes from existing services that are exposed at a desired granularity through well defined, published and standards complaint interface. Also provide interoperability by share capabilities and reuse shared services across a network irrespective of underlying protocols or implementation technology. Key characteristic of SOA: quality of service- response time, security and performance, service is cataloged and discoverable, data are cataloged and discoverable, protocols use only industry standards



A SOA has three major parts; service provider, service consumer, and service directory. Service providers are the parties who build service and make available service. Service consumers are the clients who consume services. Service directory is the place where service providers register the services and consumer search for services. Service directory provide following services: 1. Scalability of services; can add services incrementally. 2. Decouples consumers from providers. 3. Allows for hot updates of services. 4. Provides a look-up service for consumers. 5. Allows consumers to choose between providers at runtime rather than hard-coding a single provider

There are three roles in each of the Service-Oriented Architecture building blocks: service provider; service broker, service registry, service repository; and service requester/consumer.



What can you do with SOA?

There are several things that can be done with SOA:

1. Making a Reliable Service

It could be used to make a reliable service.

It could be used to make a reliable service which contains the following features:

Improved information flow.

Ability to expose internal functionality.

Organizational flexibility.

2. Making Reusable Service

One of the main use of SOA is to make a reusable service. Therefore, SOA concepts could be easily used and implemented to make a service that is not limited to a single component but could be used in multiple components.

3. Configuration Flexibility

It is highly flexible and could be easily configured as per our needs.

4. For Developing new Function Combinations

It could be used for developing new functions combinations rapidly as per need or requirement.

Advantages and Disadvantages

Given below are the advantages and disadvantages mentioned:

Advantages:

**Maintenance is Easy:** Editing and updating any service implemented under [SOA architecture](https://www.educba.com/soa-architecture/) is easy. You don’t need to update your system. A third party maintains the service, and any amendment in this service won’t have an effect on your system. In most cases, the previous API work because it is functioning before.

**Quality of Code Improved:** As services run freelance of our system, they have their own variety of code; therefore, our code is prevented from redundancy. Also, our code becomes error-free.

**Platform Independence:** Services communicate with alternative applications through a common language, which implies it’s freelance of the platform on which that application is running. Services can provide API in different languages, e.g. PHP, JavaScript, etc.

**Scalable:** If any service obtains several users, it is often simply scalable by attaching additional servers. This will create service out there all time to the users.

**Reliable:** Services square measure typically tiny size as compared to the full-fledged application. So it’s easier to correct and check the freelance services.

**Same Directory Structure:** Services have an equivalent directory structure so customers can access the service information from an equivalent directory on every occasion. If any service has modified its location, then the additional directory remains the same. This is very helpful for consumers.

**Independent of Other Services:** Services generated using SOA principles are independent of each other. So services are often utilized by multiple applications at an equivalent time.

Disadvantages:

**High Bandwidth Server:** Therefore, net service sends and receives messages and knowledge often times, so it simply reaches high requests per day. So it involves a high-speed server with plenty of information measures to run an internet service.

**Extra Overload**: In SOA, all inputs square measure its validity before it’s sent to the service. If you are victimization multiple services, then it’ll overload your system with further computation.

High Cost: It is expensive in terms of human resources, development, and technology.

Why Should we Use SOA?

This has multiple advantages, as we have discussed earlier in this article. We can use it for making reliable, better, injectable and reusable services.

Why do we Need SOA?

It could be used for solving various business needs:

For Developing Independent Services: It is required if our business need is to develop multiple services which are independent of each other. However, these services can still communicate with each other.

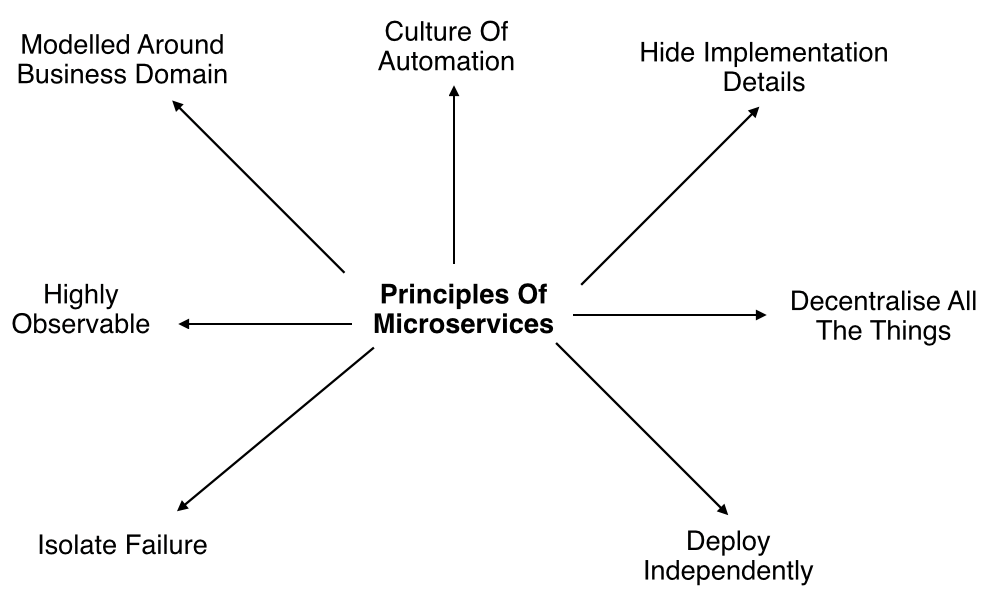
To Expose Data: Exposing the functionality of the software as a service is easier to implement if we are using SOA.

To Develop Reusable Service: If our requirement is to develop a reusable service, then SOA is perfect for this. It could be used to make independent, reliable and reusable services.

**IV. Micro-service**

Micro-service architecture is one of the booming concepts nowadays that promise quick and flawless software changes compared to traditional or monolithic architectures by modularizing complex applications into distributed parts that run parallel without any harm to another part of the applications. The team of developers composes the overall applications by the resulting in the upgradable, interchangeable, and critical scalable parts of the application. When it comes to the rapid application development era, such kind of modular architectural style surely helps to accelerate the business growth by enabling the agile deployment of the rapid client functionalities. But again, decomposing the part of applications can also come with some complexity compare with the monolithic model.

Micro-services Principles



Modelled around business domain: Micro-services architecture lets us separate system capability into different domains. Each domain will focus on one thing and its associated logic and can easily migrate independently to the next version, and also scale independently according to requirement.

Culture of Automation: As we are building, testing, deploying, and monitoring each service separately and there is an increase in the number of deployment units compared to monolithic architecture we should follow the culture of automation by designing it for continuous integration and continuous delivery.

The smaller and compact codebases and their defined scope are generally turned out to quicker deployments, which also allow us to start to explore the benefits of Continuous Deployment and Continuous integration seamlessly.

Hide implementation details: [Micro-services](https://www.dotnettricks.com/learn/microservices) should be architected in such a way that they won’t expose the internal details; neither technical implementation nor the business rules that drive it. This will reduce the coupling and help to do changes and improvements without affecting the overall architecture.

**Decentralization:**In traditional monolithic implementations, the software is designed to use a single database with different tables whereas micro-services are designed in such a way to manage their own database.

**Deploy Independently:** To enjoy the complete benefits of the architecture, Micro-services should be independently deployable. If you are failing to do so, check for any coupling in the application and solve it.

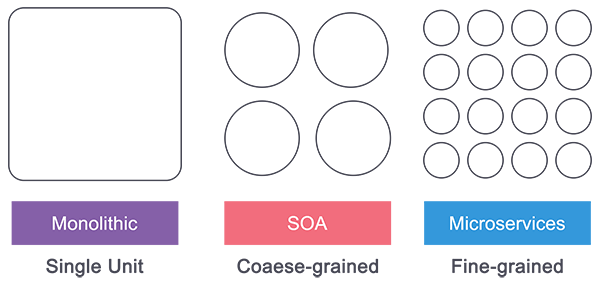
**Failure Isolation:** The impact of a failure is less in Micro-services architecture compares to the monolithic type as it will only affect that particular service and its association while other services can keep running. The associated services should handle such scenarios when the dependent is unresponsive or slow.

The larger or the enterprise applications may remain unaffected mostly by the failure of a single module and due to that, other parts of the application are running concurrently which enhances the availability of the feature to the customers most of the time.

**Highly Observable:** The services should collect as much information to analyze what is happening within each of them like log events and stats.

Monolithic vs. SOA vs. Micro-services

Monolithic architectures are the simplest form of architecture as it is having only one application layer that bundles together all the software components, and is hosted and delivered together. This type has been widely used by many small and mid-sized companies. The main challenge in this system is during scaling up as we need to duplicate the whole system including all the features of other machines which increases the cost. Also, the failure of one feature will affect the whole system making it unreliable.



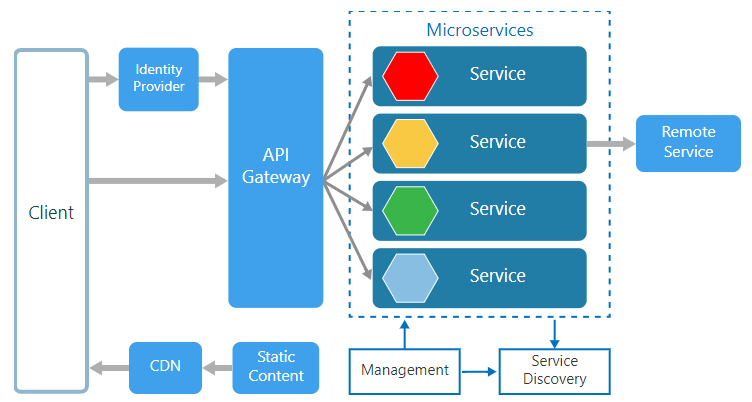
Service-Oriented Architecture (SOA) follows a coarse-grained structure where the features of an application are broken down into smaller components as services comprised of some tasks. This type of architecture allowed us to horizontally scale each service, and also more flexibility and performance at the cost of increasing the complexity of the architecture compared to the monolithic. Each service can be written in different languages and the communication between them can be done with the help of a middleware

Micro-services have technically evolved out of SOA where those features are further broken down into tasks level services making it fine-grained architecture. While Service Oriented Architecture followed a centrally governed architecture where each component is controlled by a central middleware, in Micro-services it’s a decentralized governing system where components talk directly to each other and can be written in different programming languages and communicate without the help of any broker and are done with the help of REST API.

The differences between Micro-services and SOA can be a bit fuzzy while the technical aspects of both of them can be drawn between the Micro-services and the SOA architecture somehow, and most probably around the role of the "Enterprise Service Bus", When it comes to the defining difference, it is pretty easy to consider the difference as one of the scopes. The SOA architecture was one of the extensive efforts to standardize the way all the web services work in an organization to talk to and integrate with each other, and when it comes to the Micro-services architecture, it is a kind of application-specific platform.

**Micro-services Architecture**

An architectural style that structures an application as a collection of small self-contained processes, modelled around a business capability. They don’t share the data structure and will be communicating through APIs. While in a monolithic application all the components are in a single module, in Micro-services we can see all the components are divided into a separate module and communication happens with each other with the help of APIs. In Micro-services Architecture the data is federated where each Micro-services is responsible for its own data model and data.



Being small in size, independent, and loosely coupled each service can be developed and deployed independently by a small team as each service is having its own codebase. Data and state persistence should be taken with each service as it lacks a separate data layer to handle it. The services only communicate with well-defined APIs hiding each service’s internal implementation from the other. Each service can use a different technology stack, language, libraries, or frameworks.

Management: The Management takes care of the placement of services on nodes, checking for failures, and rebalancing services across nodes in case of any failures.

Service Discovery: Maintains a list of services and the nodes where each service is located, and also enables the service to look up to find the endpoint for a particular service.

API Gateway: The entry point for clients where all the calls from the client will be taken, analyze, and forward to appropriate services. In case some calls are needed from multiple services API Gateway will aggregate and will return the aggregated result.

Companies using Micro-services

There are many companies using Micro-services for their products and services and here is a list of a few who shared their experiences.

Comcast Cable

Uber

Netflix

Amazon

eBay

Sound Cloud

Karma

Microsoft

Groupon

Hailo

Gilt

Zalando

Lending Club

AutoScout24

Advantages of Micro-services

* Services can be written in different programming languages and can be accessed by using any framework.
* Independently develop, deploy, redeploy, version, and scale component services in seconds without compromising the integrity of an application
* Better fault isolation keeps other services working even though on got failed.
* Zero downtime upgrades.
* Services can be of from different servers or even different data centers.
* Interaction with other services in a well-defined protocol
* Monitor, capture, and report health diagnostics
* Reliable and self-healing
* Supports continuous integration and delivery
* Easy to transfer knowledge to the new team member
* Easy to integrate with third parties

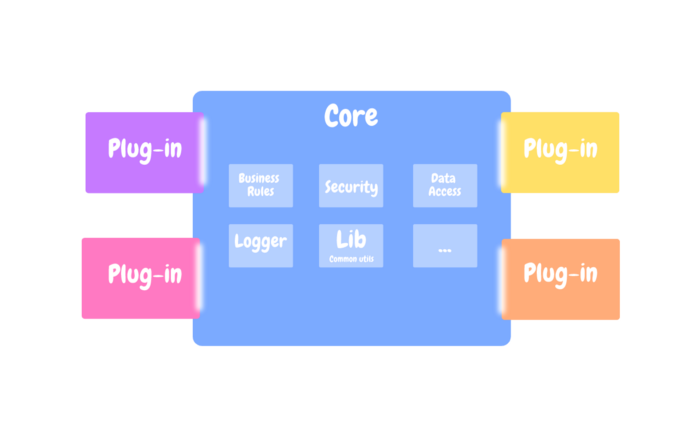
Disadvantages of Micro-services

* The additional complexity for implementation of an inter-process communication mechanism between services.
* Writing automated tests involving multiple services is challenging and It can be difficult to create consistent testing environments.
* Requires a high level of automation to manage multiple instances of different types of services in production.
* Everyone has to manage eventual consistency as maintaining string consistency becomes extremely difficult.
* Managing multiple databases and their transactions is difficult.
* Inter-process calls are slow.
* Debugging will become difficult.
* Complexity in DevOps.
* Production monitoring cost is higher.
* Formal documentation overhead.
* Lack of governance.

**V. Plug-in architecture Layering**

What is a plugin architecture?

A plug-in is a bundle that adds functionality to an application, called the host application, through some well-defined architecture for extensibility. This allows third-party developers to add functionality to an application without having access to the source code.



The plug-in architecture consists of two components: a core systemand plug-in modules.

The main key design here is to allow adding additional features as plugins to the core application, providing extensibility, flexibility, and isolation of application features and customs processing logic.

The specific rules and processing are separate from the core system. At any given point, we can add, remove, and change existing plugins with little or no effect on the rest of the core system or other plug-in modules.

**Core system**

At a high level, it defines how the system operates and the basic business logic. There is no specific implementation, no customization. It is abstracted.

A simple example. The generic workflow, such as how the data flow inside the application is defined. But, the steps involved inside that workflow is up to plugin. And so, all extending plugins will follow that generic flow providing their customized implementation.

Digging deeper a bit, it also handles special cases, applies special rules, and complex conditional processing. These are the things that need to be enforced regardless of the extending plugin.

In addition, it also contains the common code being used (or has to be used) by multiple plugins as a way to get rid of duplicate and boilerplate code, and have one single source of truth.

For example, if two plugins do log the transactions and failures, the core system should provide such a feature logger as part of it. Not to mention things like security, versioning, UI components, database access, caching, etc.

**Plug-ins**

The plug-ins are stand-alone, independent components that contain specialized processing, additional features, and custom code that is meant to enhance or extend the core system to produce additional capabilities.

Generally, plug-in modules should be independent of other plug-in modules. Though, some plug-ins require talking to, or assumes the presence of, other plug-ins. Either way, it is important to keep the communication and the dependency between plug-ins as minimal as possible.

Core←→Plug-ins

The core system needs to know about (1) the extending plug-in modules and (2) how to get to them.

The core system declares extension points that plugins can hook into. These extension points, these hooks, often represent the core system life cycle.

And so, each plugin registers itself to the core, passing some information such name, communication protocol, input/output data handlers, data format, and hooks into these extension points.

There should be a well-defined interface between the core and the plugins.

How the core system connects to these plugins is entirely based on the type of the application building (small product or large business application) and your specific needs (e.g., single deploy or distributed deployment).

Organizing domain logic

**VI. Mapping to database**

Data mapping is crucial to the success of many data processes. One misstep in data mapping can ripple throughout your organization, leading to replicated errors, and ultimately, to inaccurate analysis.

Nearly every enterprise will, at some point, move data between systems. And different systems store similar data in different ways. So to move and consolidate data for analysis or other tasks, a roadmap is needed to ensure the data gets to its destination accurately.

For processes like data integration, data migration, data warehouse automation, data synchronization, automated data extraction, or other data management projects, quality in data mapping will determine the quality of the data to be analyzed for insights.

Understanding data mapping for the modern enterprise

Data mapping is the process of matching fields from one database to another. It's the first step to facilitate data migration, data integration, and other data management tasks.

Before data can be analyzed for business insights, it must be homogenized in a way that makes it accessible to decision makers. Data now comes from many sources, and each source can define similar data points in different ways. For example, the state field in a source system may show Illinois as "Illinois," but the destination may store it as "IL."

Data mapping bridges the differences between two systems, or data models, so that when data is moved from a source, it is accurate and usable at the destination.

Data mapping has been a common business function for some time, but as the amount of data and sources increase, the process of data mapping has become more complex, requiring automated tools to make it feasible for large data sets.

Data mapping is the key to data management

Data mapping is an essential part of many data management processes. If not properly mapped, data may become corrupted as it moves to its destination. Quality in data mapping is key in getting the most out of your data in data migrations, integrations, transformations, and in populating a data warehouse.

**Data migration**

[Data migration](https://www.talend.com/resources/understanding-data-migration-strategies-best-practices/) is the process of moving data from one system to another as a one-time event. Generally, this is data that doesn't change over time. After the migration, the destination is the new source of migrated data, and the original source is retired. Data mapping supports the migration process by mapping source fields to destination fields.

**Data integration**

[Data integration](https://www.talend.com/resources/what-is-data-integration/) is an ongoing process of regularly moving data from one system to another. The integration can be scheduled, such as quarterly or monthly, or can be triggered by an event. Data is stored and maintained at both the source and destination. Like data migration, data maps for integrations match source fields with destination fields.

**Data transformation**

[Data transformation](https://www.talend.com/resources/data-transformation-defined/) is the process of converting data from a source format to a destination format. This can include cleansing data by changing data types, deleting nulls or duplicates, aggregating data, enriching the data, or other transformations. For example, "Illinois" can be transformed to "IL" to match the destination format. These transformation formulas are part of the data map. As data is moved, the data map uses the transformation formulas to get the data in the correct format for analysis.

**Data warehousing**

If the goal is to pool data into one source for analysis or other tasks, it is generally pooled in a [data warehouse](https://www.talend.com/resources/what-is-data-warehouse/). When you run a query, a report, or do analysis, the data comes from the warehouse. Data in the warehouse is already migrated, integrated, and transformed. Data mapping ensures that as data comes into the warehouse, it gets to its destination the way it was intended.

What are the steps of data mapping?

Step 1: Define — Define the data to be moved, including the tables, the fields within each table, and the format of the field after it's moved. For data integrations, the frequency of data transfer is also defined.

Step 2: Map the Data — Match source fields to destination fields.

Step 3: Transformation — If a field requires transformation, the transformation formula or rule is coded.

Step 4: Test — Using a test system and sample data from the source, run the transfer to see how it works and make adjustments as necessary.

Step 5: Deploy — Once it's determined that the data transformation is working as planned, schedule a migration or integration go-live event.

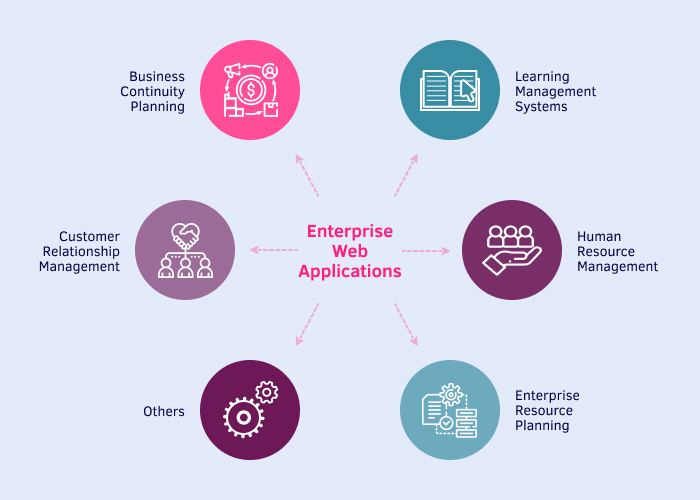
Step 6: Maintain and Update — For ongoing data integration, the data map is a living entity that will require updates and changes as new data sources are added, as data sources change, or as requirements at the destination change.

**VII. Web Presentation**

Enterprise Web Applications allow you to manage and record the internal and external operations and processes of your company or organization. The main advantages are the important cost savings and time savings, by helping the modern company in their digital transformation path.

Enterprise web application development is the process of programming an application for a large-scale business that is aimed at in-house usage and helps to be in control over all operational functions. This software is different from off-the-shelf web technologies because it’s designed for the needs of a specific company, not its clients.

Enterprise web custom solutions can be corporate learning management systems (LMS), human resource management platforms (HRM), enterprise resource planning apps (ERP), customer relationship management software (CRM), business continuity planning programs (BCP), and many others.



3 Different Types of Enterprise Application Features

Enterprise software differs from traditional apps because it must have certain features that help the whole team be informed of all internal operations.

There are three types of such specifications:

1. Basic features

– Cloud-based software

Enterprise web development usually presupposes building apps using cloud computing, because it doesn’t restrict you to a certain location. It means that no matter where you or your team members are or what kind of device they use, they can easily access your corporate app.

Beyond that, a cloud-based management system allows for organizational scalability, so you can hire developers from distant locations and manage them just like your on-premises employees.

– Automation of administration processes

With enterprise website solutions, corporations forget about manual entry and sorting of data. The system does this for them. That’s why your managers will no longer have to update the client information, payment details, accounting data, and other useful information. Moreover, such business applications automatically upload and streamline your Excel sheets, so that you don’t lose the previous data.

– App design adaptability

Intuitive design is one of the advantages of all enterprise apps that let your team members use the software without problems. However, these enterprise-level applications must also be adaptive in terms of adjustments to make this corporate management system integration barrier-free.

2. Optimal features

– Security of corporate data

Custom enterprise web development not only offers automated and adaptable data entry and organization but also safekeeping and exchange of this information. Protective measures of this software cover the defense of your sensitive data against malware, cyberattacks, ransomware, and other malicious activities.

– Recovery of information

Another essential feature of flawless enterprise web application performance is the ability to restore your data in case of an unexpected system shutdown or other circumstances. As a business leader, you can rest assured that your data will not be lost, damaged, or deleted forever.

– Mobile compatibility

How to develop an enterprise web application and make it convenient for the whole team? You should also create a version of this application for smartphones. This way your developers and other employees will be able to enter or obtain the needed information anytime and anywhere.

3. Top features

– Analytical tools

It’s great to have analytics experts in place who report to you about the latest operational tendencies and help you make rational business decisions. Nevertheless, it’s more reliable to have a smart application that does the same thing using embedded data analytics – and people who properly interpret this data and provide important insights.

– Digital payments support

To simplify payments processing and accounting, some enterprise web development services incorporate online payment methods. They enable you or your company managers to make payments at any time and track your expenses in one place. You won’t even have to make any entries yourself since your app will do it for you.

– Social engagement functionality

Enterprise web application engineering can foster more social interactions among your developers or clients. This will help them be in touch and even find common ground, which creates trust. And trust improves your chances of striking a profitable deal with your customers.

**VIII. Concurrency**

As concurrency means adding forces in a complementary and simultaneous way to achieve a common goal, then it is easy to understand that it is a good complementary tactical approach or way of operating with a collaboration strategy. Concurrency is also synonymous with boundary-less organization as it has been demonstrated with concurrent engineering implementation that broken down barriers between disciplines enable collaboration.

Concurrent organization

To survive in a fast-changing world, particularly in different domains, the adaptation capability should be higher than ever before. The adaptation capability combines the learning mode with creativity and innovation potential. Otherwise, survivability doesn't mean that someone or something has to live for eternity but lives for a given life cycle where learning and teaching are daily concurrent operations between the old, current, and new generations. In fact, new generations really constitute the surviving mechanism. While revolution has shown a tendency to push back to chaos (like reengineering), evolution is built on capitalized knowledge (like the learning organization). Then the opportunity is to rapidly progress in operating concurrently. It's like having a piece of the key, but some others own the complementary pieces, and you do not know them, and no one knows he has a piece of that key. Let's imagine that anyone can build up some models of all the pieces he has and put him or her on a global network. Here we have the connection with fractal theory, as some pieces should inevitably attract some others due to the fact they have something in common. Humanity constitutes one of those possible networks even if the interaction potential is quite low at a given period of time. Then it is multiplied along centuries. Stochastic interactions between facts and people enabled past discoveries, but trying to understand which one exactly made the body of the key appearing is like trying to identify which one of many low signals announcing changes was suddenly the enabler (signals announcing stock exchange growth, economic or political crises, thunderstorms, earthquakes).