**Integration Architecture**

Integration architecture is a software architecture that facilitates the integration of multiple IT components.

Business objectives is hard to achieve if the systems wont talk to each other. So api’s are built to achieve this.

A software project may consist of different functionalities like

* Web service calls
* File transfers
* Batch processings
* Message processings using Queues
* CRM Applications
* Databases etc.

Integration Architecture enables you to process these application and data objects within your IT architecture while providing and using interfaces.

An Integration architect checks whether the current platform used by the company fulfils the purpose of current and future needs along with the speed of delivery and cost.

Below are some of the functionalities of Integration Architect.

* Understand business needs
* Architecture pattern decision
* Effective Integration platform to reduce point to point connections between services
* Decision of effective tools
* 3rd party vendors
* Reusable services
* Security of api’s
* Data modeling
* Data storing
* Ability to build services to scale to multiple users and sizes
* Improving customer experience
* Adopt to new technologies
* Infrastructure
* Deployments

References:

<https://www.leanix.net/en/integration-architecture>

<https://www.enterpriseintegrationpatterns.com/patterns/messaging/Chapter1.html>

**SOA: Service Oriented Architecture**

It is a layered architectural pattern for building software systems based on loosely coupled service components.

It is more like request response pattern using webservices between different services.

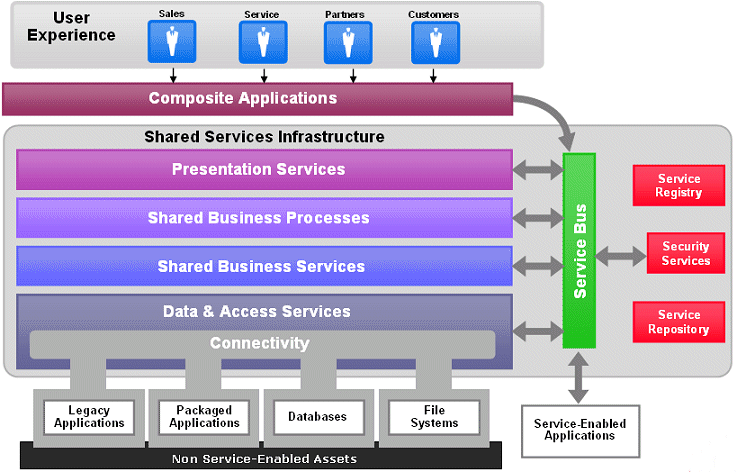
Each service is self contained (independent functionality) and uses abstraction so the end user don’t get the technical details of the service.

Monolithic application into modular services and interacting with each other via ESB.

Example:

A customer placing a new order may require the coordination of many systems. The business needs to validate the customer ID, verify the customer’s good standing, check inventory, fulfill the order, get a shipping quote, compute sales tax, send a bill, etc. This process can easily span across five or six different systems. From the customer’s perspective, it is a single business transaction.

Services should be discoverable, reusable and easily maintainable.



Pros:

* Reusable services
* Scalability, Discoverability
* Platform independence. Because of ESB services can easily integrate with other 3rd party systems.

Cons:

* As they are mainly soap based it requires a high speed server to increase performance.
* It will be a overload if the number of services increases.
* If ESB fails, whole application clogs.

References:

<https://www.techrepublic.com/blog/decision-central/deciphering-the-term-soa/>

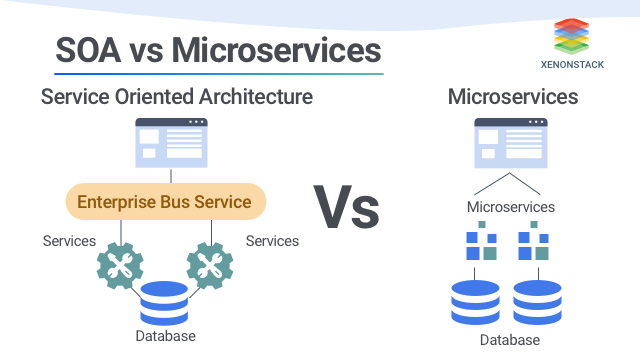
<https://searchapparchitecture.techtarget.com/definition/service-oriented-architecture-SOA>

<https://www.ibm.com/cloud/learn/soa>

<https://www.opengroup.org/soa/source-book/soa/p1.htm>

<https://www.cio.com/article/2439274/soa-definition-and-solutions.html>

**Microservices vs SOA:**



While Microservices and SOA have application modularity in common, they differ in how the services are deployed and the size of modules.

Architecture type is a particular way of developing software, web or mobile applications as suites of independent called  *microservices*. These services are created to serve only one specific business function like search, user info management etc.

Services are smaller than SOA services and independent of other services. It does not focus on reusability of services.

No need of ESB for interaction of services uses simple messaging system.

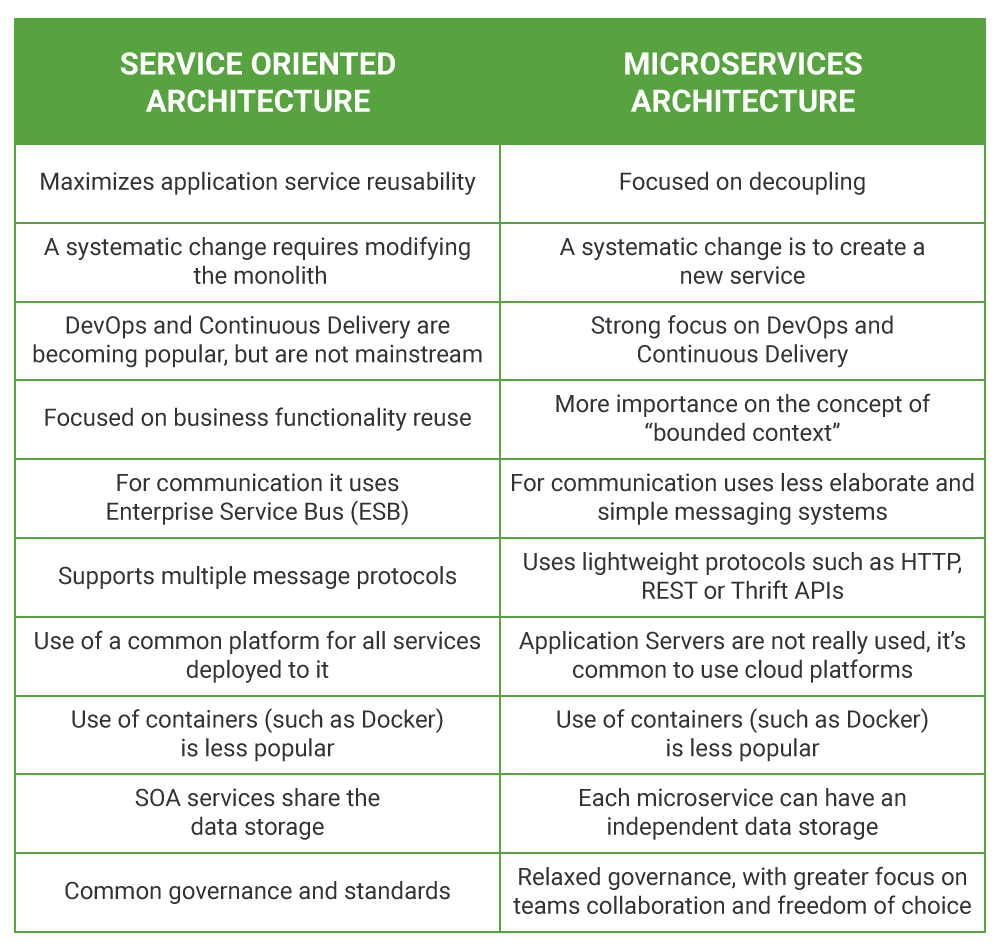
ESB has logic to decide which service to be called and end up becoming complex.

Eg:

A Bank application selling different products like savings account, debit card, credit card, insurance etc.

IN SOA if Customer places an order for credit card. ESB takes the order request from application and handles the communication with the appropriate service and processes it

Where as in Microservices architecture, an event is generated after customer placing an order and the respective service will respond and processes the order. So no need of maintaining ESB.



References:

<https://www.xenonstack.com/insights/service-oriented-architecture-vs-microservices/>

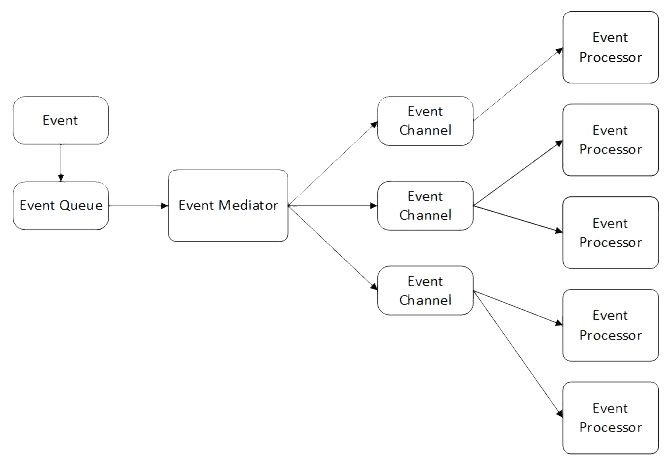
<https://insights.daffodilsw.com/blog/microservices-vs-service-oriented-architecture-soa-fundamental-differences>

<https://newizze.com/soa-vs-microservices-comparison-for-2019/>

**EDA: Event Driven Architecture:**

Event driven architecture goes hand in hand with Microservices. When an action occurs, an event is created and this event is then used to drive decisions across anything that is waiting for that event to occur.

Asynchronous way of sending data by queuing mechanism.



### **Event channel**

This is the second logical layer. An event channel is a mechanism of propagating the information collected from an event generator to the event engine.

**Event Mediator**

Which takes the messages from queue and process it and transform it and routes to respective channel.

Uses Queues, publisher-subscriber pattern etc.

Publisher is the one who triggers events and subscriber consumes them

Publish once and the data sent to the multiple subscribers.

Unlike Rest Api, services that create requests do not need to know the details of the services consuming the requests.

Eg: Notifying user who placed order on sales day that the item is out of stock.

If a system is down notifying different events to raise ticket, to take backup etc.

Example 2:

An e-commerce application that uses this approach would work as follows:

1. The Order Service creates an Order in a pending state and publishes an OrderCreated event.
2. The Customer Service receives the event and attempts to reserve credit for that Order. It then publishes either a Credit Reserved event or a CreditLimitExceeded event.
3. The Order Service receives the event from the Customer Service and changes the state of the order to either approved or cancelled.

**When to use this architecture**

* Multiple subsystems must process the same events.
* Real-time processing with minimum time lag.
* Complex event processing, such as pattern matching or aggregation over time windows.
* High volume and high velocity of data, such as IoT.

**Benefits**

* Producers and consumers are decoupled.
* No point-to-point integrations. It's easy to add new consumers to the system.
* Consumers can respond to events immediately as they arrive.
* Highly scalable and distributed.
* Subsystems have independent views of the event stream.

**Challenges**

* Guaranteed delivery. In some systems, especially in IoT scenarios, it's crucial to guarantee that events are delivered.
* Processing events in order or exactly once. Each consumer type typically runs in multiple instances, for resiliency and scalability. This can create a challenge if the events must be processed in order (within a consumer type), or if the processing logic is not idempotent.

**References:**

<https://aws.amazon.com/event-driven-architecture/>

<https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven>

<https://www.redhat.com/en/blog/importance-event-driven-architecture-digital-world>

<https://medium.com/@ktsravindu/event-driven-architecture-f3cbae2853d9>

**ETL: Extract, transform, Load:**

Data Integration process for Extracting data from sources like CSV, JSON, XML etc and transforming the data schema compatible with the end sources and load to the end systems.

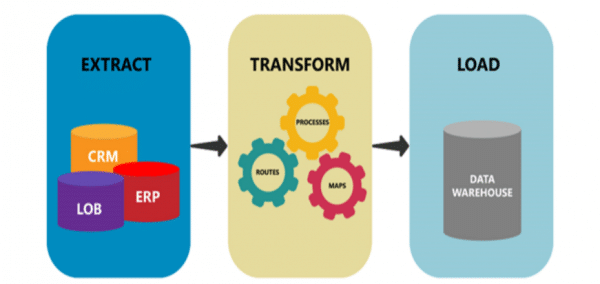
Automated process of transforming data from source system to the target systems.

##### **Phases of the ETL process**

1. **Data extraction process**: this is the first stage of ETL, where the data located in the different company repositories are extracted. In this step, the data is converted into a **single format** and prepared for transformation. Once we have all the data collected and raw, we have to do a **treatment of cleaning or purification** that will help us avoid mistakes that dirty them.
2. **Data transformation process**: transforms the data into the structure we have defined in our data warehouse. The transformation step includes validation actions on **business rules**, **technical validations** (duplicates, integrity, nulls...), code normalization and homogenization, format changes, as well as sorting, filtering, crosses and aggregates.
3. **Data loading**: the loading phase is the last one of the process, the transformed data begin to be loaded **in the data warehouse** where they are registered, they can be audited and they always have a history over time.

ETL Tools in Market:

* [Informatica PowerCenter](https://hevodata.com/blog/best-etl-tools-data-warehouse/#informatica)
* [IBM InfoSphere DataStage](https://hevodata.com/blog/best-etl-tools-data-warehouse/#datastage)
* [Hevo Data](https://hevodata.com/blog/best-etl-tools-data-warehouse/#hevodata)
* [Talend](https://hevodata.com/blog/best-etl-tools-data-warehouse/#talend)
* [Pentaho](https://hevodata.com/blog/best-etl-tools-data-warehouse/#pentaho)
* [AWS Glue](https://hevodata.com/blog/best-etl-tools-data-warehouse/#glue)
* [StreamSets](https://hevodata.com/blog/best-etl-tools-data-warehouse/#streamsets)
* [Blendo](https://hevodata.com/blog/best-etl-tools-data-warehouse/#blendo)
* [Google Data flow](https://hevodata.com/blog/best-etl-tools-data-warehouse/#google_data_flow)
* [Azure Data Factory](https://hevodata.com/blog/best-etl-tools-data-warehouse/#azure_data_factory)
* [Apache Nifi](https://hevodata.com/blog/best-etl-tools-data-warehouse/#apache_nifi)



Example:

If a data analyst wants to analyze data from different systems like CRM, ERP and other systems, instead of accessing data from multiple systems multiple times all this data will be pushed to Datawarehouse. So that analyst can have structured data all at one place.

Advantages of ETL include:

* Good for bulk data movements with complex rules and transformations
* Make maintenance and traceability much easier than hand-coding
* Good for data warehouse environment

Disadvantages of ETL include:

* You must be a data oriented developer or database analyst to use
* Not ideal for near real-time or on-demand data access, where fast response is required
* Takes months to put into place
* Difficult to keep up with changing requirements

**References:**

<https://www.xplenty.com/blog/what-is-etl-architecture/#what>

<https://medium.com/hashmapinc/etl-understanding-it-and-effectively-using-it-f827a5b3e54d>

<https://www.talend.com/resources/etl-architecture/#:~:text=ETL%20architecture%20overview&text=The%20data%20needs%20to%20undergo,a%20log%20file%20or%20database>.

<https://www.complexsql.com/etl-definition/>

**Can Mule be used as ETL?**

With mule data weave and batch processing, we can do ETL work like transforming data from different sources and send it to the end systems.

**Difference between ETL Tool like Informatica PowerCenter and Mule?**

In ETL tool, there are separate components for transforming data and matching columns of database tables easily etc. With Mule we can transform but cannot do everything offered by ETL Tool.

ArchiMate:

<https://online.visual-paradigm.com/diagrams/templates/archimate-diagram/infrastructure/>

<https://sparxsystems.com/resources/tutorials/archimate/index.html>

<https://architecture-center.com/blog/144-the-practical-approach-to-archimate-example-use-cases.html>