# Spring Integration

## Evolution of spring integration

### Pluming code

If we have to get a file from FTP server. We need to write code for connecting with FTP client and get the file. This code is called pluming code. Which doesn’t deals with business logic but infrastructure logic. Which is tedious and time consuming and not really what our application is all about.

Here come **Spring integration** as rescues where we can use lots of pluming code out of box, where we have many components out of the box using we can connect to FTP server, messaging broker, web services, relational, NoSql databases, email, web sockets and many more.

We can keep integration code separate from business logic. This will help us write clean, testable and maintainable code.

Spring integration is based on **Enterprise Integration Pattern** (where 65 integration design patterns). This design pattern focus on connecting system using **Asynchronous** messaging.

Spring integration is used as basic for **spring-cloud-stream** for building highly scalable event driven microservices.

### Enterprise Integration

Enterprise integration focus on asynchronous messaging.

Integration means connecting system from our organization to systems outside our organization. Integrating systems not often easy because system will be different in many aspects and may will have different data formats.



## Concepts of Enterprise Integration Pattern

* Message
* Message Channel
* Message Endpoint
* Message Transformation
* Message Routing

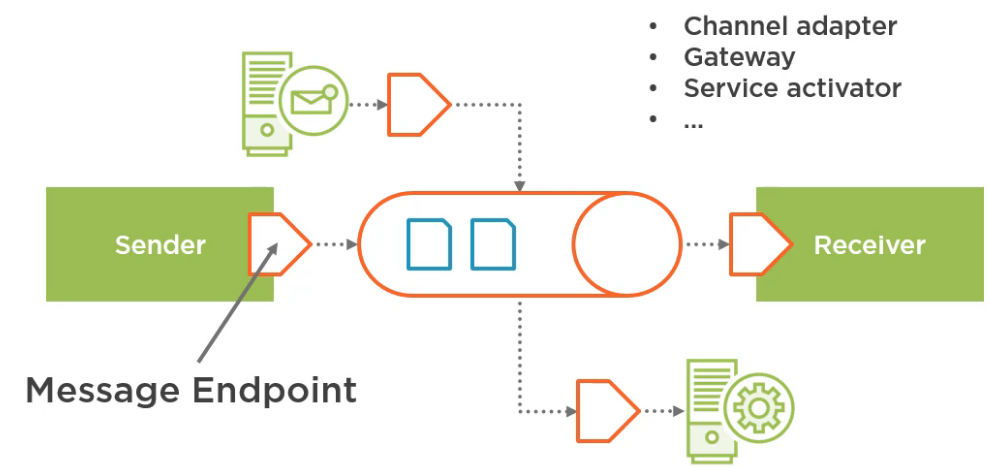
**Message** is a unit of data transmitted from one system to other using a messaging system. Message contains **payload** and **header.**

Messages transmitted over **message channel** from sender to receiver. Queue is an example for message channel

* Point-to-point channel (One sender and one receiver ex. Queue)
* Publish-subscribe channel (Used to broadcast message from one sender to multiple receiver ex. Event notification)



**Message Endpoint** connects to message channel from Sender to Receiver to send message(s).



Differ systems often produces and consumes data in different format, here we need **Message Transformer.**

**Message Routing:** Theremight be a chance where receiver of message from message channel is not fixed. We may need to reside on specific header value of message to which receiver should process the message. Message Router makes such decision.

Messaging system may also decide to drop the message and not to process. Which is decide by **Message Filter (**comes from Message Routing pattern)

