**UNIT 4: Enterprise Application Integration & Enterprise Integration patterns:**

**I. Introduction to Enterprise Integration**

What is enterprise integration?

Every modern business must share data. If you are a large business trying to take advantage of big data, you know that [big data is an integration challenge](https://www.redhat.com/en/topics/big-data). To do this, the applications and devices at the core of your business strategy must be accessible to each other—and, very likely, across multiple [cloud environments](https://www.redhat.com/en/topics/cloud). Enterprise integration encompasses the technologies, processes, and team structures that connect data, applications, and devices from everywhere in your IT organization.

Enterprise integration models have evolved over the years from those with a relatively small number of point-to-point connections, to a centralized model connected through an enterprise service bus (ESB), to a distributed architecture with many reusable endpoints.

Enterprise integration is the use of multiple integration approaches, including [API management](https://www.ibm.com/cloud/learn/api-management), application integration and messaging to leverage enterprise services and assets in order to expose them as APIs or connect them as services. This enables organizations to seamlessly integrate, unify and standardize core business capabilities across diverse IT environments. Specifically, enterprise integration enables you to easily do the following:

* Discover valuable services, applications and data
* Access and expose application functions via APIs
* Connect multiple enterprise services
* Monitor application lifecycles and governance

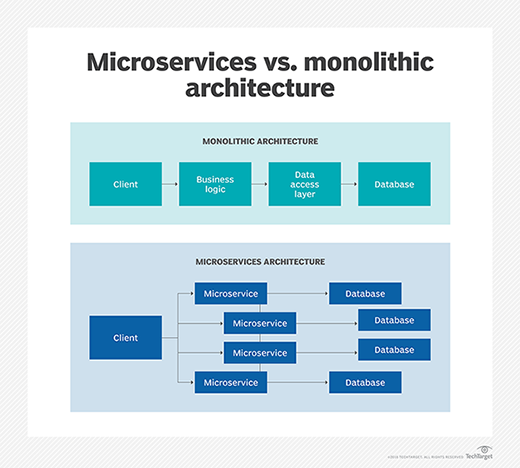
### Why is enterprise application integration important?

Most businesses use several software products from different vendors in their day-to-day operations. These applications can have different uses, data stores and methods of operation. This can create [data silos](https://www.techtarget.com/searchdatamanagement/definition/data-silo) where data is either duplicated between databases or is available in one but not another. That leads to data inconsistency when users need to manually copy and paste data between applications. Users may also need to consult several applications for their daily work.

EAI provides a methodology that helps these disparate applications share data and workflows to reduce manual steps and errors and to provide better insight into organizational data. This becomes more important as an organization grows and as technology advances.

In the past, all business workflows were manual and carried out on paper. As computers automated work for users, the day-to-day assignments of information workers quickly became formatting data and transferring it from one system to another. This need to quickly make data available between applications will only increase as more organizations adopt [microservices](https://www.techtarget.com/searchapparchitecture/definition/microservices), software as a service ([SaaS](https://www.techtarget.com/searchcloudcomputing/definition/Software-as-a-Service)) and platform as a service ([PaaS](https://www.techtarget.com/searchcloudcomputing/definition/Platform-as-a-Service-PaaS)) tools.

EAI can help to remove manual steps and reduce user error.



### How does enterprise application integration work?

EAI is a principal or philosophy toward the programs in use at a business. It is not a specific tool or [framework](https://www.techtarget.com/whatis/definition/framework). It's best thought of as a goal or ideal to reach toward rather than an implementation.

Organizations can be at different levels of EAI, from applications existing separately to full integration where all applications share common data and workflows. More realistically, most will fall somewhere in between, with some applications working together and other not. Businesses are always changing, so any EAI implementation must also be able to change.

EAI does not require that someone reprograms the applications themselves. Instead, organizations typically use [middleware](https://www.techtarget.com/searchapparchitecture/definition/middleware) or [APIs](https://www.techtarget.com/searchapparchitecture/definition/application-program-interface-API) to get applications to work together. EAI also makes it easier to swap applications to suit business needs without impacting productivity or requiring large-scale system designs.

Not [cloud](https://www.techtarget.com/searchcloudcomputing/definition/cloud-computing) or on-premises specific, a well-done EAI implementation can account for applications with different sources and underlying [infrastructure](https://www.techtarget.com/searchdatacenter/definition/infrastructure) needs. EAI attempts to solve the same business issues as using a single enterprise resource planning ([ERP](https://www.techtarget.com/searcherp/The-ultimate-guide-to-ERP)) package, but with several smaller programs working together instead of one large program.

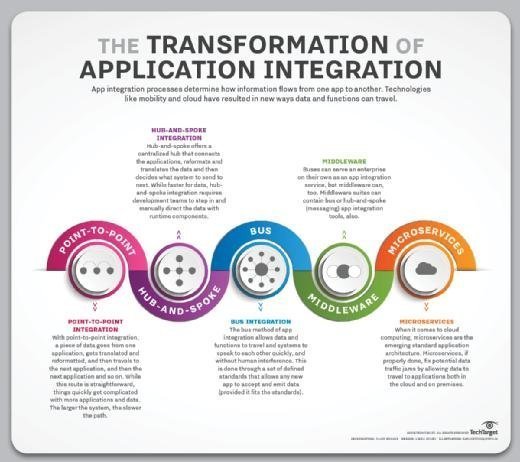
Almost all organizations use some form of EAI. Taking the output of one program and running a Microsoft [Excel](https://www.techtarget.com/searchenterprisedesktop/definition/Excel) macro or a [script](https://www.techtarget.com/whatis/definition/script) to reformat it to enter into another program can be thought of as a simple form of EAI.

### Goals of enterprise application integration

Database integration is a common and relatively easy goal for EAI. Removing duplication of data and ensuring consistency between applications can reduce mistakes and liability. EAI can be achieved by setting different applications to use the same database or using synchronization tools to keep different databases up to date with changes. [Data warehouse](https://www.techtarget.com/searchdatamanagement/definition/data-warehouse) designs can make this possible. All these can also be more easily fed into [big data](https://www.techtarget.com/searchdatamanagement/definition/big-data) systems for better business intelligence ([BI](https://www.techtarget.com/searchbusinessanalytics/definition/business-intelligence-BI)).

Workflow integration can get separate tools to work together to increase productivity by making common tasks easier to accomplish. As an example of EAI, onboarding an employee may require human resources to update payroll, personnel records, desk assignments on a floor plan, door access controls and IT resources. With EAI, all these different tasks can be accomplished as a single workflow instead of in separate programs.

Interface integration allows organizations to present a single consistent user interface and user experience ([UI](https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI) and [UX](https://www.techtarget.com/searchcio/definition/UX-user-experience)) to operators instead of multiple interfaces from different software packages. This can be the most difficult form of EAI to achieve as minor changes to the underlying software can result in redesign work.

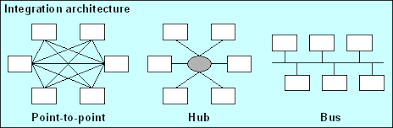
Understanding the transformation of application integration.

### Types of enterprise application integration

An EAI implementation depends on the underlying project tools and goals, but several common design paradigms have emerged.

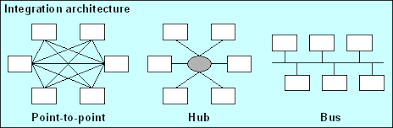
**Point-to-point integration.**This is the simplest form of EAI. In it, data is taken from one source, perhaps reformatted, and then ingested by the next application. These are often simple to implement for small workflows and a few tools. They can quickly grow large and difficult to manage as more applications and integrations are added, however, and can become slow as a backlog or slowdown in one system affects others in the line.

As an example, **an organization may need to update a human resources database with information from an ERP system**.

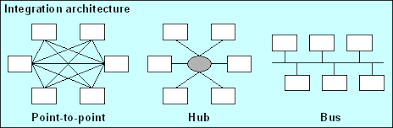


**Hub-and-spoke integration.** This approach uses a central program to facilitate the data and steps between the participation applications. The program can handle the data reformatting and keep workflows moving in the event of an application slowdown. Hub-and-spoke is therefore faster and more reliable than point-to-point but requires development time and effort to set up and maintain.

For example how **airlines operate out of a centralized hub and use regional airports as the spokes from which they offer flights**

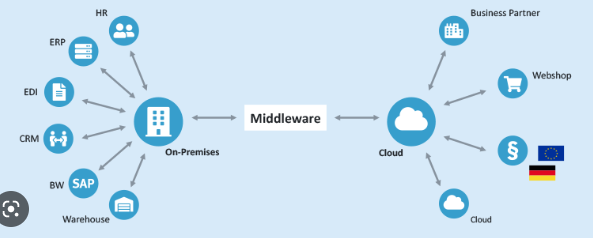


**Bus integration.**This is an evolution of hub-and-spoke design EAI. It is also called an enterprise service bus ([ESB](https://www.techtarget.com/searchapparchitecture/definition/Enterprise-Service-Bus-ESB)). In a common bus design, all participating applications use a set of standards to send and receive data or workflows. This allows for quick and easy integration but requires work during the planning and product selection phase.



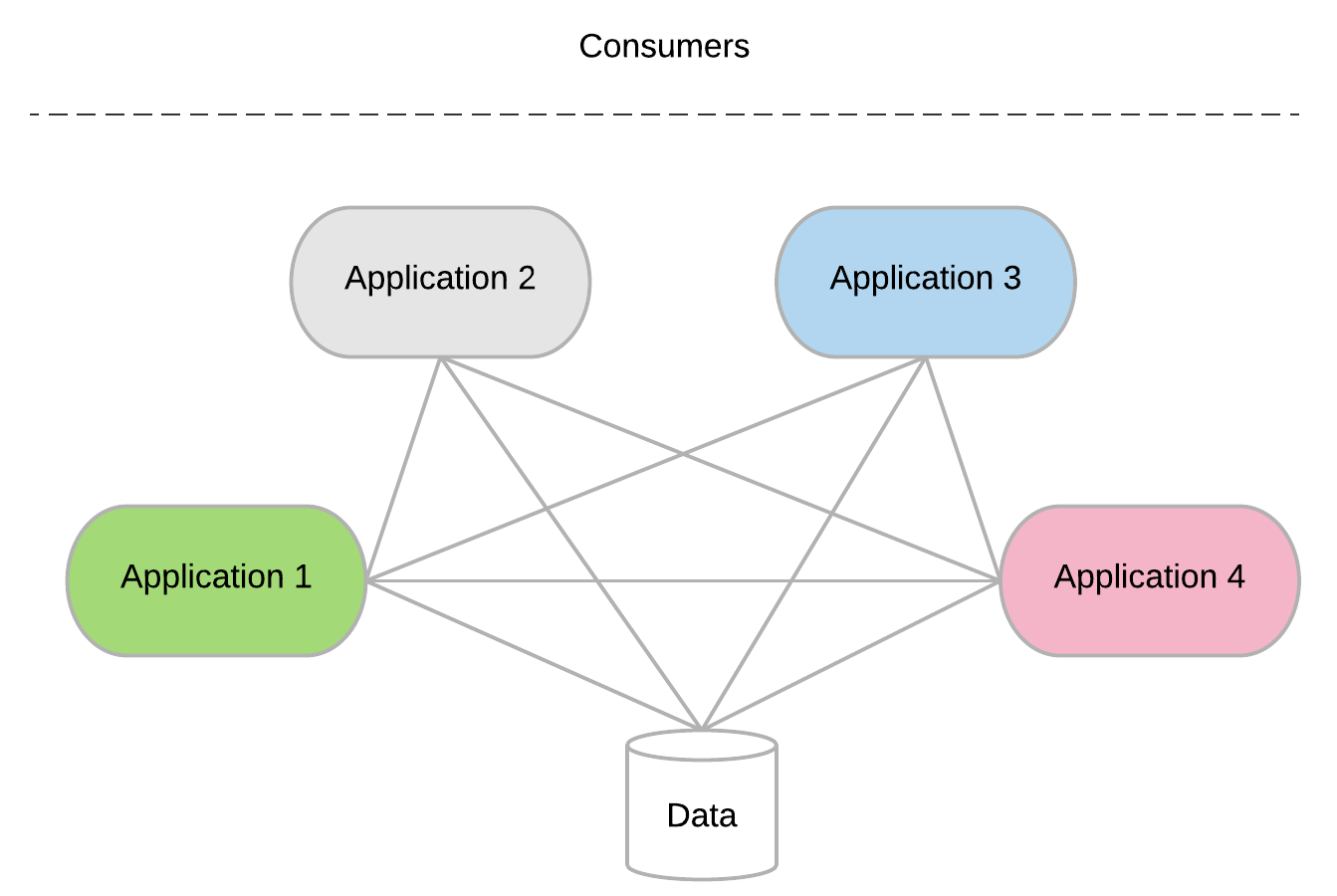
**Middleware integration**. This involves an intermediary program that sits between the end user and the underlying application. Middleware supports interface integration and may have an underlying hub-and-spoke or bus design.

Common middleware examples include **database middleware, application server middleware, message-oriented middleware, web middleware, and transaction-processing monitors**.



**Microservices.**These are small, single-purpose tools that support EAI initiatives. These can be serverless functions or dedicated apps designed to integrate easily or quickly connect programs. Microservices can often be easily offloaded as cloud workloads.

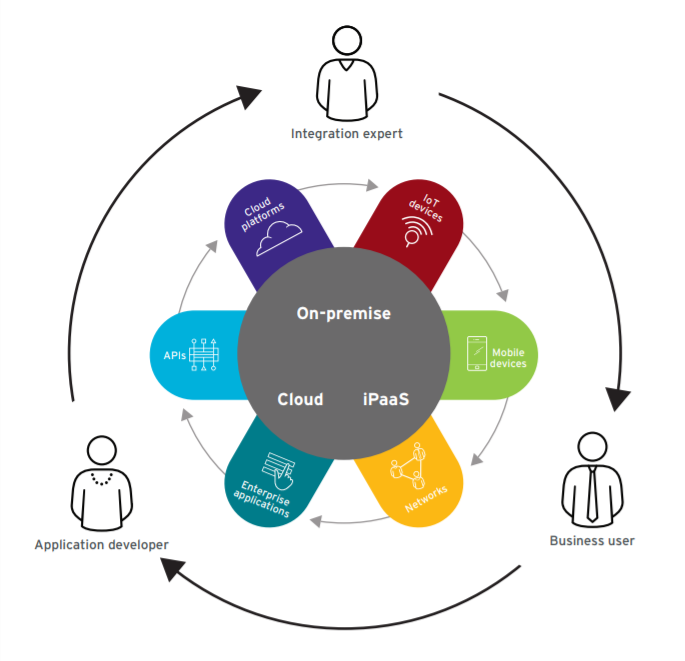
Some of the most innovative and profitable microservices architecture examples among enterprise companies in the world — like **Amazon, Netflix, Uber, and Etsy**



**Red Hat's approach to integration**

Red Hat believes that the traditional approach to integration, with centralized teams controlling monolithic technologies, can impede the development and long-term usefulness of distributed applications. Traditional integration technologies like the ESB have benefits like prioritizing security and data integrity, but they also rely on a single team to define integrations for the entire enterprise.

Today’s loosely coupled, cloud-native application architectures developed through [agile](https://www.redhat.com/en/topics/devops/what-is-agile-methodology) and DevOps methods need an equally agile and scalable approach to integration. Red Hat’s view of agile integration is just that, an approach to connecting your resources that combines integration technologies, agile delivery techniques, and cloud-native platforms to improve the speed and security of software delivery. Specifically, agile integration involves deploying integration technologies like APIs into Linux containers and extending integration roles to cross-functional teams.



Why is enterprise integration important?

Overall, enterprise integration is key to the enhancement of internal processes and business activities as well as the conceptualization, implementation and distribution of critical applications. By sharing important information, simplifying processes and maximizing opportunities, companies can improve their operational scalability and increase their reach and revenue.

Sharing critical information

Enterprise integration eases the flow of data within complex information and operating systems by providing a middleware layer to act as the common interface between each separate application, system and service. It simplifies data exchange between various applications and amongst multiple users who rely on said data, making it easy for an application developer to exchange data or expose interfaces without having to understand the other applications and systems, know where they are running or predict what could go wrong.

Simplifying IT processes

Enterprise integration enables seamless collaboration, combining functionality and information exchange across multiple applications. Their interconnection helps simplify IT processes in a manner that makes life easier for users *and* organizations. Specifically, it enables users to access data faster, and it helps IT organizations efficiently streamline data integration and services. This simplification modernizes the creation and use of enterprise integration patterns like gateway services, [message queues](https://www.ibm.com/cloud/learn/message-queues), file transfer and [enterprise service bus (ESB)](https://www.ibm.com/cloud/learn/esb) by enabling them to be built, deployed, operated and maintained through agile and automated processes.

Maximizing opportunities

Enterprise integration also helps teams work proactively to take advantage of new or changing business needs and opportunities. By gaining control of all data access points, teams can quickly identify and adjust to time-sensitive events like unexpected policy shifts or new application management procedures without needing to change the applications themselves. Ultimately, by addressing their integration needs, teams become empowered to devise, implement and streamline multiple integration solutions by leveraging a common approach for collaboration and information exchange.

What are the key elements of enterprise integration?

* **Application programming interfaces (APIs)** process data transfers between different systems. Situated between an application and web server, they enable companies to share the data and functionality of their applications with third-party developers, business partners and internal departments. With APIs increasingly used to access and expose real-time data, this can be extended to more sources, such as data published as events.
* [**Application integration**](https://www.ibm.com/cloud/learn/application-integration) is the enablement of individual applications — each designed for a specific purpose — to work collaboratively. By making it easier to share data and combine workflows and processes, organizations can benefit from integrations that modernize infrastructures without rework. Furthermore, application integration helps on-premises systems and cloud-based enterprise systems like CRMs and ERPs interact successfully without major changes to existing applications.
* **Messaging**helps provide resilience and performance to IT environments spanning cloud and on-premises systems. Messaging must cross network boundaries to provide reliable delivery while preserving network-wide message integrity, data protection and regulatory compliance via security-rich functions.
* **Events** are records of action or change. When one application or service performs an action or undergoes a change relative to the functionality of another application or service, the first one publishes an event. Other applications or services can detect the event publication. They can then process the event, perform one or more reciprocal action or simply ignore the event.
* **Data**, specifically real-world operational data, enables continuous improvement (CI) of enterprise architecture. Data is also used to assess the criticality and usage of integrations and determine their target state. When analyzed, data reveals recommended target integration patterns (e.g., [service-oriented architecture (SOA)](https://www.ibm.com/cloud/learn/soa), event-driven, message-driven, etc.), consolidation possibilities and other inputs that help define the target integration state.

**The "what" and "how" of enterprise integration**

***What*, as in, "what are you integrating?"**

First and foremost, enterprise integration is a data challenge. So much data exists within organizations now that the term "[big data](https://www.redhat.com/en/page-not-found)" is often used to indicate the size—and also the variety—of data sources. A large volume of data existing in a variety of nonstandard formats can be of significant business value, but first it must be integrated from multiple sources or applications. The [Internet of Things](https://www.redhat.com/en/page-not-found) (IoT) also represents a new opportunity to connect with customers and analyze useful data through everyday devices, but you must filter out the critical data that needs to go to your datacenter. Web applications further add to the complexity of enterprise integration, especially when legacy applications must be integrated with a service-based architecture, like [microservices](https://www.redhat.com/en/topics/microservices).



***How*, as in, "how are you integrating your apps, devices, and data?"**

In the past, a centralized enterprise service bus (ESB) managed by a centralized team could connect every endpoint in your environment. However, a centralized approach to teams and technologies can bottleneck modern systems, which need fast, easy avenues to integrate between distributed components. Depending on your data and service needs, a combination of messaging, application connectors, data streams, enterprise integration patterns, and [application programming interfaces](https://www.redhat.com/en/topics/api) (APIs) that can be deployed faster and iteratively are more suited to [modern application development](https://www.redhat.com/en/solutions/cloud-native-development).

**Messaging**

Messaging is a way for different components in a distributed [application architecture](https://www.redhat.com/en/topics/cloud-native-apps/what-is-an-application-architecture) to communicate. Components can send and receive messages across different languages, compilers, and operating systems as long as each side of the communication understands the common messaging format and protocol.

A [service mesh](https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh) is used to route messages within a microservices architecture.

**Application connectors**

Application connectors are architectural elements that model the rules for how components interact. They are standard class connections customized for certain APIs, so they can be used to quickly integrate new endpoints.

**Data streams**

Data streams provide a constant flow of information that applications can add to or consume from, independent of the transmission of that data. For instance, [Apache Kafka](https://www.redhat.com/en/topics/integration/what-is-apache-kafka) is a distributed data streaming platform that can publish, subscribe to, store, and process streams of records in real time

**Enterprise integration patterns**

EIPs are collections of technology-independent solutions to common integration problems. Patterns also provide a common language for developers and application architects to describe integrations.

**Application programming interfaces**

An API is a set of tools, definitions, and protocols for building application software. It lets your product or service communicate with other products and services without having to know how they’re implemented.

**How does enterprise application integration work?**

EAI is a principal or philosophy toward the programs in use at a business. It is not a specific tool or [framework](https://www.techtarget.com/whatis/definition/framework). It's best thought of as a goal or ideal to reach toward rather than an implementation.

Organizations can be at different levels of EAI, from applications existing separately to full integration where all applications share common data and workflows. More realistically, most will fall somewhere in between, with some applications working together and other not. Businesses are always changing, so any EAI implementation must also be able to change.

EAI does not require that someone reprograms the applications themselves. Instead, organizations typically use [middleware](https://www.techtarget.com/searchapparchitecture/definition/middleware) or [APIs](https://www.techtarget.com/searchapparchitecture/definition/application-program-interface-API) to get applications to work together. EAI also makes it easier to swap applications to suit business needs without impacting productivity or requiring large-scale system designs.

Not [cloud](https://www.techtarget.com/searchcloudcomputing/definition/cloud-computing) or on-premises specific, a well-done EAI implementation can account for applications with different sources and underlying [infrastructure](https://www.techtarget.com/searchdatacenter/definition/infrastructure) needs. EAI attempts to solve the same business issues as using a single enterprise resource planning ([ERP](https://www.techtarget.com/searcherp/The-ultimate-guide-to-ERP)) package, but with several smaller programs working together instead of one large program.

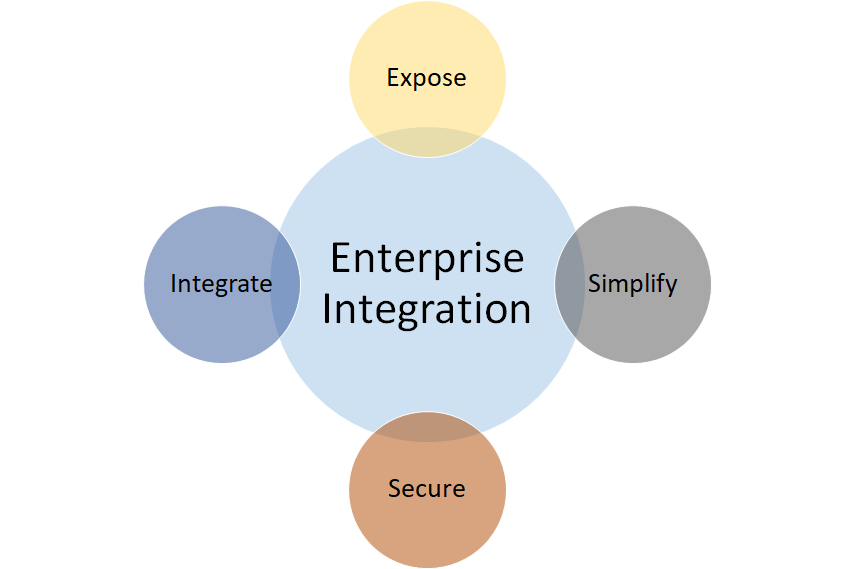
Almost all organizations use some form of EAI. Taking the output of one program and running a Microsoft [Excel](https://www.techtarget.com/searchenterprisedesktop/definition/Excel) macro or a [script](https://www.techtarget.com/whatis/definition/script) to reformat it to enter into another program can be thought of as a simple form of EAI.

**Goals of enterprise application integration**

Database integration is a common and relatively easy goal for EAI. Removing duplication of data and ensuring consistency between applications can reduce mistakes and liability. EAI can be achieved by setting different applications to use the same database or using synchronization tools to keep different databases up to date with changes. [Data warehouse](https://www.techtarget.com/searchdatamanagement/definition/data-warehouse) designs can make this possible. All these can also be more easily fed into [big data](https://www.techtarget.com/searchdatamanagement/definition/big-data) systems for better business intelligence ([BI](https://www.techtarget.com/searchbusinessanalytics/definition/business-intelligence-BI)).

Workflow integration can get separate tools to work together to increase productivity by making common tasks easier to accomplish. As an example of EAI, onboarding an employee may require human resources to update payroll, personnel records, desk assignments on a floor plan, door access controls and IT resources. With EAI, all these different tasks can be accomplished as a single workflow instead of in separate programs.

Interface integration allows organizations to present a single consistent user interface and user experience ([UI](https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI) and [UX](https://www.techtarget.com/searchcio/definition/UX-user-experience)) to operators instead of multiple interfaces from different software packages. This can be the most difficult form of EAI to achieve as minor changes to the underlying software can result in redesign work.



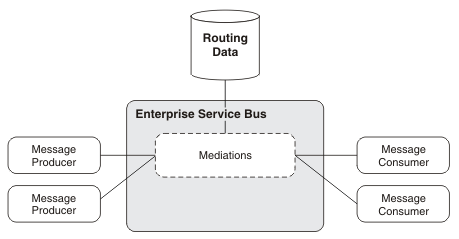
**III. Elements of messaging-based Integration.**

# **Message-based Integration**

Message-based Integration is one of the common categories of patterns for connectivity solutions.

An Enterprise Service Bus can extend an existing messaging infrastructure by providing an environment for building and deploying message-based applications at the infrastructure level. Examples of these applications include routing and transformation services, and logging services. This environment can extend a single underlying messaging infrastructure or provide a bridge between different products and technologies. Adapters can be used to provide access to applications or services that do not have an in-built messaging capability.

The following diagram illustrates an example of a Message-based Integration scenario:



**Asynchronous Message-Based Integration**

Asynchronous message-based integration is rapidly emerging as the primary technology required for enterprise integration. To help you understand how to use it most effectively, we next:

* Evaluate an end-to-end enterprise integration scenario and discuss how to use messaging to facilitate integration
* Compare two different conceptual message-based architectures that you can use for integration, point-to-point integration and hub-and-spoke integration, and discuss the relative benefits and trade-offs of the two architectures
* Summarize the key technological components involved in using messaging for enterprise integration, discussing message storage and management, message routing and propagation, and message transformation, as well as message interoperability and the use of standards

In order to understand how messaging can best be used for e-business integration, we now consider in detail the technical architecture associated with connecting a supplier's supply chain application to a business-to-business trading marketplace or exchange. This section includes:

* [An Example of the Use of Messaging for B2B Integration](https://docs.oracle.com/cd/A87860_01/doc/ois.817/a83729/adois04.htm" \l "1000439)

* [Exchange Integration Scenario: Supplier Perspective](https://docs.oracle.com/cd/A87860_01/doc/ois.817/a83729/adois04.htm" \l "1000455)

* [Exchange Integration Scenario: Exchange Perspective](https://docs.oracle.com/cd/A87860_01/doc/ois.817/a83729/adois04.htm" \l "1000468)

**An Example of the Use of Messaging for B2B Integration**

For example, a supplier must automate the way it interacts with a B2B exchange so that it can automatically

* Post up-to-date pricing and inventory information to the exchange
* Respond rapidly to auctions and requests for information (RFIs) from the exchange

Let us consider how the integration works from both the supplier standpoint and from the exchange standpoint. From an integration point of view, three primary technological requirements enable such communication. These include:

**Communication Between the Supplier and Exchange**

To send a message from its supply chain system to the exchange, the supplier must use a solution that receives a message payload and propagates it to the exchange. The message propagation facility must provide a number of services:

* **Asynchronous communication:** The supplier's systems and the exchange must be connected using asynchronous messaging for three reasons:
  + Supplier and exchange are not tightly coupled in a request-response model because both can continue to process business activities without requiring a response.
  + Communication between supplier and exchange must be resistant to application, system, and network outages.
  + Neither the supplier nor the exchange enables another trading partner to carry out the two-phased commit operation across its systems that is required for a synchronous interaction.
* **Guaranteed delivery**: Because messages exchanged between supplier and exchange are business-critical, the messaging facility must guarantee exactly once, in-order delivery of messages.
* **Message storage and management:**In order to resolve potential disputes between the supplier and its trading partners, the messaging facility must store messages sent by and received from the exchange so that you can audit and track them.

**Message and Data Transformation**

In propagating the message from its supply chain system to the exchange, the supplier must address two message or data transformation issues:

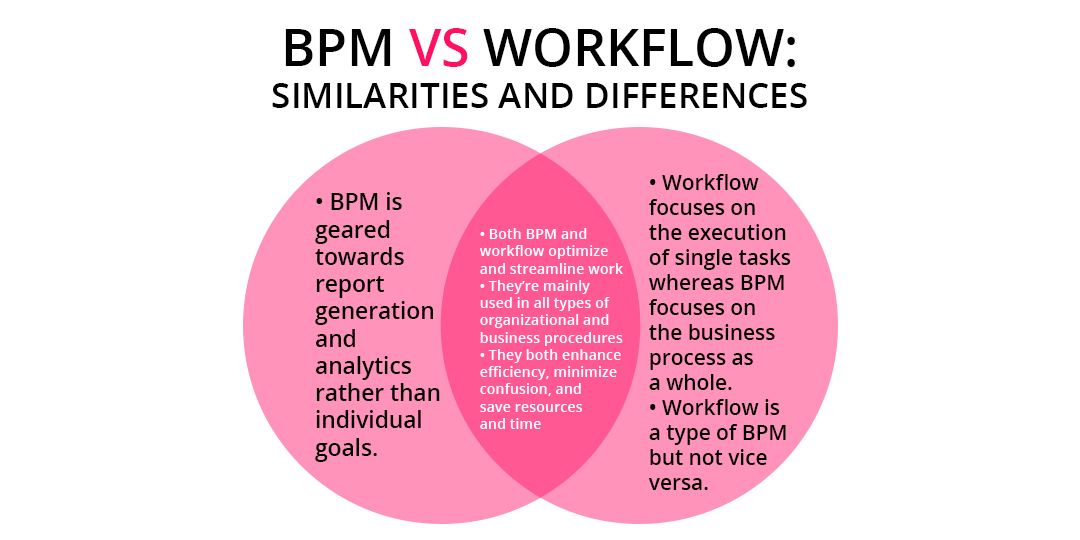
* **Message propagation protocol:**Since communication between the supplier and the exchange and other trading partners occurs over the Internet, the message must be propagated over the standard HTTP-S protocol. Prefer SSL over HTTP for security reasons.
* **Message propagation format:** Although the message is propagated over HTTP-S, the message payload must be sent in a format that both the exchange itself and the other trading partners can understand. Most businesses send the message payload in an XML format agreed to by all the trading partners and the exchange.
* **Message and data transformation:**Because the supplier's supply chain application is likely to store data in a proprietary format, you require a message transformation facility to accept the data from the supply chain application and translate it into the appropriate XML format.

**Business Process Management and Workflow**

Before sending the message to the exchange, the supplier may need to get approval for the price list from an appropriate executive. Similarly, in responding to a purchase request from the exchange, the supplier may need to send the message to the company's Financial Application and to update the inventory status in the Supply Chain Application. To manage this multi-step business process, the supplier must maintain a business process management or workflow facility that coordinates messages between the different applications and the exchange with this:

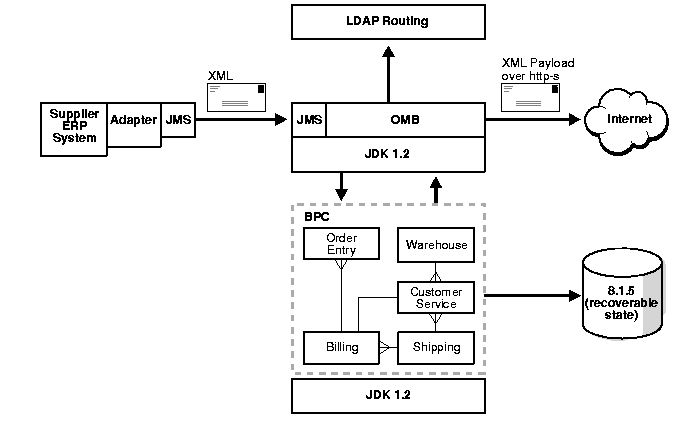
Before sending the message to the exchange, a local workflow process may need to be invoked in order to determine whether there is sufficient inventory associated with each of the items on the new price list. In this case, a business process management facility will be required at the supplier end. The business process management or workflow facility will need to route the new price list from the backoffice application to the company's inventory application to compare inventory status for items on the price list. To manage this multi-step business process, the supplier must maintain a business process management or workflow facility that coordinates messages between the different applications and the exchange. Further, since a single business process may involve coordinating business events between the company's enterprise applications and with an exchanging or an external trading partner, a single business process management or workflow facility should be used for both intra-enterprise and business-to-business interactions.

Now that we understand the primary requirements for this integration scenario, let us examine in greater technical detail the specifics of the integration from the perspective of both the supplier and the exchange.



**Exchange Integration Scenario: Supplier Perspective**

Various steps and integration components link the supplier's supply chain application with the business-to-business exchange. In this scenario this is an Oracle exchange. The steps and components involved in establishing such connectivity are illustrated in Figure 1.

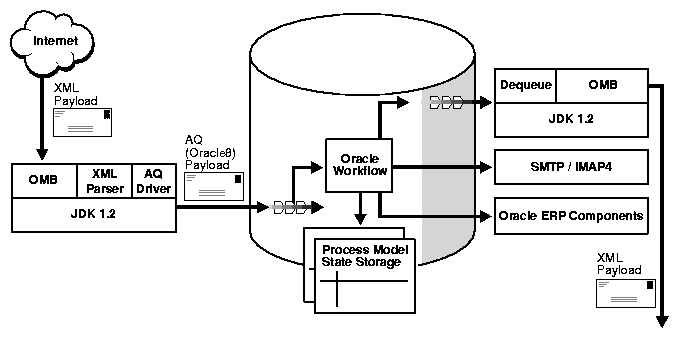


These steps and components include:

* **Adapter technology:** For the application to connect to the exchange, the supply chain application must first use an adapter to post a message to the exchange integration facility. The adapter waits for new inventory status information to be posted on a specific message queue. The adapter receives the message in the specific data format of the application, so the adapter then converts the data into XML.
* **Message propagation infrastructure:** The application adapter first enqueues the message into a message propagation infrastructure through a standard messaging interface. The messaging infrastructure can then provide three additional services:
  + It can route the message to the exchange either statically or based on its subject or content.
  + It can consult an LDAP directory service to determine where to send the message.
  + It can optionally store the message in a persistent store so that the message can be audited or tracked.
* **Local Workflow Processing:** If local processing is necessary, the local business process management facility can dequeue the message from the message propagation infrastructure, process it locally, and, when this is completed, place the message back in the message propagation infrastructure in order to send the message to the exchange.

**Exchange Integration Scenario: Exchange Perspective**

The integration architecture of the Oracle exchange resembles the architecture described in the previous section.



An XML payload is received over the wire on the HTTP-S protocol and goes through the following stages:

* **Message receipt and propagation:** The message management facility receives the message as an XML payload over HTTP-S.
* **Data transformation and parsing services:** When the facility receives the message, it frequently places the message in a persistent store so that it can be audited and tracked for conflict resolution. The message header and payload may often need to be parsed using an XML parser for two purposes:
  + To determine where to route information based on either the subject or topic of the message or the content of the payload. For instance, the message may need to be sent through a specific approval cycle if the supplier is posting inventory levels that are below the threshold for the exchange.
  + To parse the message into a structured format to enable it to be placed in a business process management or workflow facility for processing
* **Business Process Management:**The workflow system dequeues the message from the message propagation infrastructure and carries out an entire workflow process to update the various applications that form part of the exchange. The workflow facility can also:
  + For example, send an e-mail message to notify a small supplier by directly calling its own SMTP/IMAP4 interface
  + Send an XML message outbound: If the workflow system needs to send an XML message outbound to another supplier over HTTP-S, it can serialize the message into the XML payload format appropriate to the target, determine how to route the message by consulting an LDAP directory, and then enqueue the message into the message propagation infrastructure.

**IV. Modern service integration techniques.**

1. API Integration

Application Programming Interface (API) is the most common tool for connecting different applications for service management software. There are many different types of API that are either public, partner, or private. What they all have in common is how they enable interaction between applications. An API uses a common code language to specify functionality and set protocols. This gives your applications the ability to transfer data.

Pros:

Highly Flexible: Even though you are dependent on the developer resources, specific data becomes highly flexible because the integration uses product code.

App Changes Aren’t Disruptive: Service providers offer better functionality that goes uninterrupted since APIs are often limited in scope.

Widely Available: As stated earlier, API is the most common tool for third-party integration. So, it will be unlikely that you run into a service that won’t offer API integration options.

Cons:

Dependent on Vendor: Vendors are responsible for creating APIs. So, you are reliant on the vendor to create APIs for the specific type of information you are trying to pull.

Code-Intensive: Because they are code-based, APIs need an understanding of programming languages to install.

2. Web-hooks

Web hooks or HTTP call backs are an alternative to API integration. They are both tools that link to a web application but have two key differences. For web hooks, implementation is often not code-based. They often have modules that are programmable within a web application. Instead of being request-based, web hooks are event-based. They only trigger when specific events occur within a third-party service.

Pros:

Real-Time Data: Web hooks don’t use a request-based system. They allow your team to view data on a real-time scale.

Supports Automation Efforts: Because data requests are event-based, you don’t have to set up poll timings to your data centers. This can help streamline data flow and automation.

Cons:

Limits Data Manipulation: A webhook requires the service to trigger a data transfer based on an update. In contrast to webhooks, APIs can list, create, edit, or delete an item without triggering a transfer.

3. ISC

Integration Services Component (ISC) lives on a local server unlike code-based integrations.  The ISC creates a bridge with on-premise tools such as directories, asset management tools, and BI tools without the need for file imports.

Pros:

(Near) Out-of-the-Box Solution: The ISC immediately offers many data synchronization options you would likely use.

Wider Range of Functionality: With an ISC, you have complete data access that you can do anything with.  Any data that you can access on the backend with your cloud service will be available.

Cons:

Knowledge of Database Architecture Necessary: If you are unfamiliar with how your local database is set up, implementing an ISC will be challenging.

Requires Access to the Backend of Your Applications: There will be many cases where backend access isn’t there for your team, so you won’t be able to use an ISC in those situations.

4. ORCHESTRATION

The most automated integration option is orchestrations. If you are not familiar with orchestrations, they refer to the process of automating multiple systems and services together. Teams will often use software configuration management tools such as [PowerShell](https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7) to build orchestrations. Software configuration management tools offer various methods such as snap-ins or hosting APIs to connect with applications to manage the automation workflow.

Pros:

Full Automation: Automation across all processes.

Manages Multiple Systems: Ability to manage the integrations of multiple systems collectively.

Cons:

Code-Intensive: You need to have coding skills to manage your software configuration management tool.

Labor-Intensive: Because the workflows are quite complex, the setup can be a drawn-out process. Also, any asset or process changes force you to check how it will affect your orchestrations.

**V. Introduction to WSDL, SOAP.**

# **Web Services Description Language (WSDL)**

Web Services Description Language (WSDL) is a standard specification for describing networked, XML-based services. It provides a simple way for service providers to describe the basic format of requests to their systems regardless of the underlying run-time implementation.

WSDL defines an XML format for describing network services as a set of endpoints that operate on messages that contain either document-oriented or procedure-oriented information. The operations and messages are first described abstractly and then bound to a concrete network protocol and message format in order to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages, regardless of which message formats or network protocols are used to communicate. This means that interfaces are defined abstractly using XML schema and then bound to concrete representations that are appropriate for the protocol.

WSDL allows a service provider to specify the following characteristics of a Web service:

* The name of the Web service and addressing information
* The protocol and encoding style to be used when accessing the public operations of the Web service
* The type information such as operations, parameters, and data types comprising the interface of the Web service

WSDL documents allow developers to expose their applications as network-accessible services on the Internet. Through UDDI and WSIL, other applications can discover WSDL documents and bind with them to execute transactions or perform other business processes.

This development platform supports and encourages the development and use of WS-I compliant WSDL. Business service providers can deploy Java™ beans and enterprise beans as a Web service and generate a WSDL document that describes the service. They can also generate Java and enterprise bean skeletons from an existing WSDL file. A business service client can generate a Java proxy from a WSDL document, thereby providing an easy-to-use Java interface to the Web service. The Java interface hides the network communications details from the client enabling the business service provider to focus on business and process portions of the application.

In addition to providing tools to create Web services, the workbench provides a WSDL editor that allows you to create WSDL files using a graphical interface, a WSDL validator that ensures that your WSDL file is semantically correct and optionally checks for WS-I compliance, and the Web Services Explorer which allows you to dynamically test Web services without requiring you to generate a proxy.

**SOAP**

SOAP is a messaging protocol for exchanging information between two computers based on XML over the internet. SOAP messages are purely written in XML which is why they are platform and language independent.

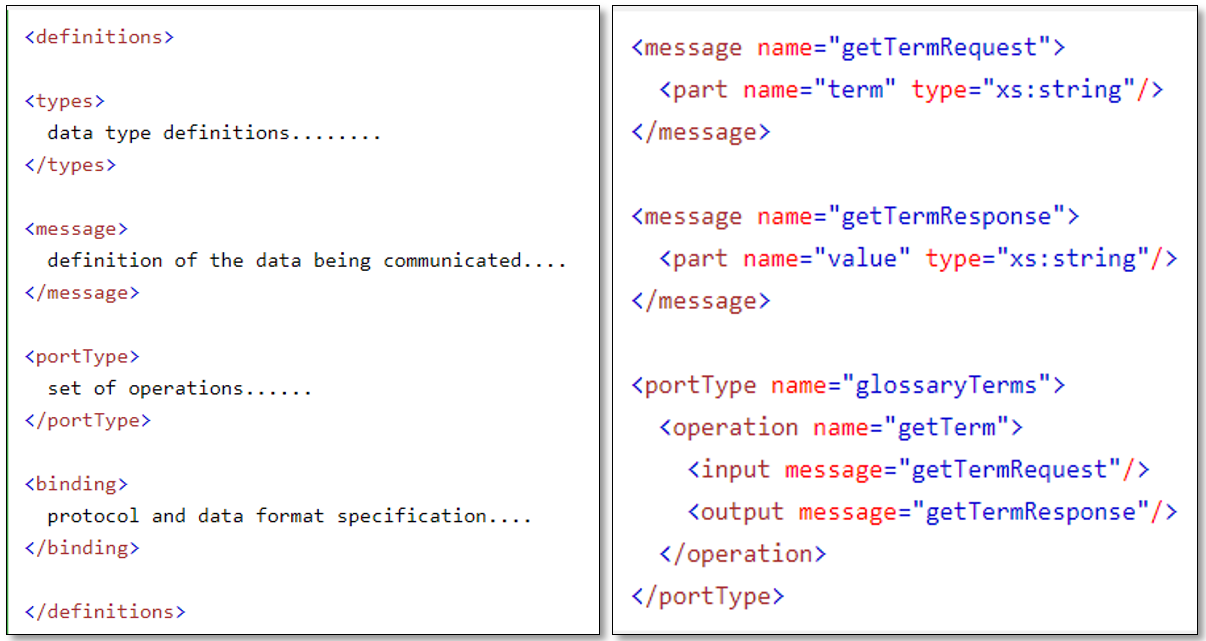
A SOAP message contains:

* An Envelope that indicates the start and end of the message
* A Header that includes attributes used to process the message and is an optional element
* A Body that holds the XML data that is to be sent and it cannot be left out
* A Fault which provides error messages when processing and it is an optional element

Working with SOAP requests and responses could get very complex. Some languages make efficient use of the SOAP shortcuts to reduce the level of complexity and the .Net platform for example hides the XML to a large extent. This is made possible due to the Web Service Definition Language (WSDL). WSDL is an XML file that defines and describes the services that are available in the web service of interest. It describes the naming of services, the specifications and structure of the response sent back. The services in the WSDL are described as a compilation of network ports/endpoints. With the WSDL, the .Net platform is able to auto-generate the proxy classes and functions which can be called from the application.

* The types describe the datatypes the web service use.
* The message defines each operation’s data element
* The portType describes what operations can be done and the messages involved in this operation. This could be one of four types: One way (receives messages but no response), Request-Response (receives request and responds), Solicit-Response (sends request and waits for response) or Notification (sends request, does not require response
* Binding tells the format of data for the type of each port and the protocol.





**VI. Introduction Restful web services integration.**

### REST

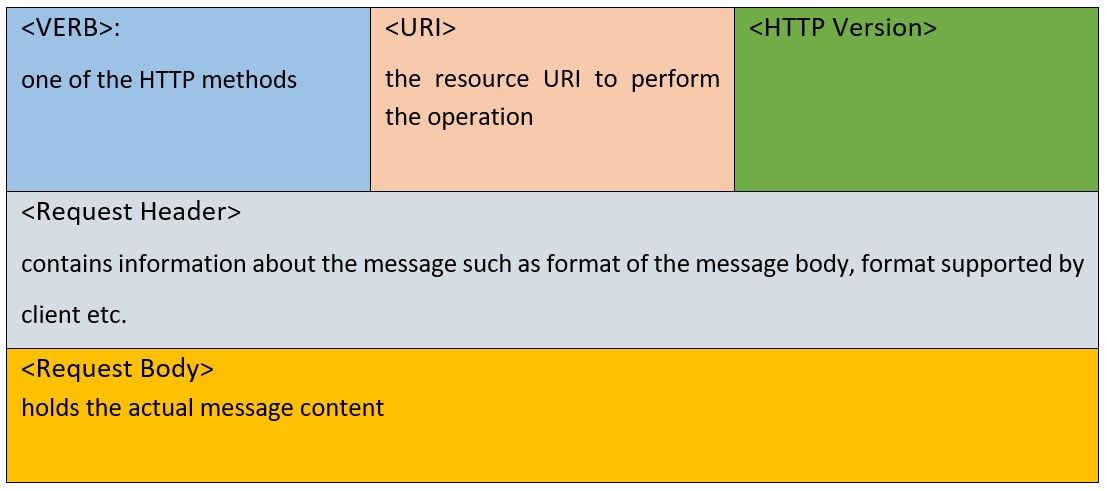
REST is a web standard architecture that achieves data communication using a standard interface such as HTTP or other transfer protocols that use standard Uniform Resource Identifier (URI). The design is such that each component in a RESTful web service is a resource that can be accessed using standard HTTP methods (if the chosen protocol is HTTP). Resources which can be thought of as objects in the concept of Object oriented programming (OOP) are identified by URIs and the resources are represented in several ways such as JSON, XML, Text etc. though JSON is currently the more favoured choice.

RESTful services have the following properties: Representations, Messages, URIs, Uniform interface, Stateless, Links between resources and caching. A quick look into these properties below using HTTP

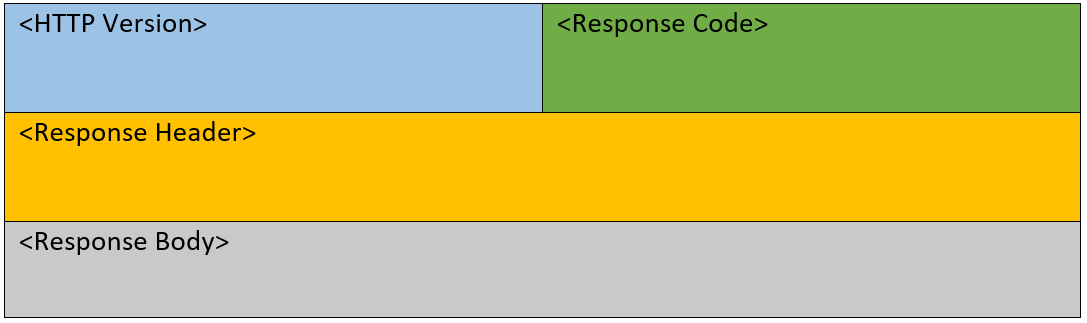
1. Representation- resources are represented in different formats as earlier stated and should be a complete representation of the resource.
2. Messages- This is how the client and server interact. Along with the data, messages contain metadata bout the message. When accessing a RESTful resource using HTTP, the commonly used methods are GET (reads a resource), PUT (creates a resource), DELETE (removes a resource) and POST (updates an existing resource)
3. URIs – Each resource needs at least one URI to identify a resource(s) and the operation is determined by the HTTP verb/action. Hence a URI can be called with different actions.
4. Stateless- Restful web services are stateless and any session state is held on the client not server. This ensures that every client to server request has the necessary information to understand the request and handles each request independently.
5. Links Between Resources- the representation of a resource can have links to other resources.
6. Caching- the data produced when a request is made the first time is stored and used the next time in order to stop regenerating same information for the same request and improves performance. The HTTP headers help to control caching such as age which tracks how long ago the data was fetched from the server, expires date and time the resource representation expires etc.

The above gives an oversight of the six properties of REST and it is important to remember that:

* REST is not coupled to HTTP and is actually protocol independent. It is simply not a mapping of CRUD to the HTTP methods.
* REST makes it relatively easy to integrate with websites and are exposed using XML (one of many ways) for easy consumption.



The Response Header and Body hold similar information to the request, only that the information is different as it is the server response



**VII. Differences between SOAP and REST.**



