**Interview Prep:**

**MEDIATION:**

The separation of the business logic from the sending and transformation of messages allows for great flexibility in how you set up your architecture and makes it much simpler to customize the business logic without having to worry about the various formats in which a message might arrive.

Logging, auditing, and security monitoring can all be performed without any modification of the existing code.

**Primitive:**

Whatever Component or connector involved in Mediation is called Primitive.

Example-Message Filter, Validator, HTTP and Other Component which helps to separate business logic.

**Fan-Out:**

Fan-Out is nothing but usage of scatter gather like sending multiple requests or payload messages in parallel to multiple targets.

**Fan-In:**

After sending the request to Multiple targets Using Fan Out Concept, combining all the response from Multiple Target is Called Fan-In.

Fan out and Fan In always Ships together. An Example is Scatter-Gather in Mule.

**API Led Connectivity:**

The main purpose of API-led connectivity is to enable the integration flows to be reused by many parties and to be reused inside the integration platform. In this approach, the APIs are based on three distinct layers: System, Process, and Experience.

**System Layer:**

This is the foundational layer of the three-layer architecture. A System API defines the RAML to describe how to interact with the domain. The main benefit of the system layer API's is that One can modify the System API logic without affecting the other APIs (Process and Experience). For example, if a System API is using SAP and, in the future, SAP needs to be replaced with Salesforce, this replacement can be done easily modifying only the System API without touching anything in Process and Experience layers.

**Process Layer:**

Process layer APIs are responsible for shaping the data which it gets from multiple system API's. The main purpose of Process APIs is to strictly encapsulate the business process logic independent of the source systems (System APIs) from which the data originates. For example, in a purchase order process, it needs to interact with various domains of the organization. The Process API (purchase order/order fulfilment) interacts with the already available System APIs to implement the logic.

**Experience Layer:**

Experience APIs are simple. Basically, they involve only the transformation of data, so as to meet a wide range of clients that accept data in diverse formats. For example, our Order Purchase API (Process Layer) has exposed data in the JSON format, but we have a client application that accepts only XML format, or vice versa. This simple transformation logic is implemented in the Experience Layer and the implementations are exposed as Experience APIs.

**JMS:**

JMS enables the application to communicate through the exchange of messages. A JMS connector is capable of sending and receiving messages to and from the

queues and topics. JMS allows two models of messaging:

1. **QUEUES**: it enables one to one communication. The sender will deliver a message to the queue and a single receiver picks up the message from the queue. The receiver does not need to listen to the queue at the time the message is sent to the queue.

2. **TOPICS**: It enables one to many communications, also known as Publish-Subscribe communication. The Publisher will deliver a message to the topic and it will be delivered to all the Subscribers who are actively listening to that topic.

**Operations in JMS:**

**On new Message**: Acts as a Listener for incoming messages.

**Publish**: Allows a user to send a message to a JMS destination.

**Consume**: Allows the user to consume a message from the destination.

**Publish Consume**: Allows the user to send a message to a JMS destination and waits for a response.

**Ack**: JMS Connector provides different message acknowledgement modes to configure for the **Ack**, **Consume**, **On New Message**, and **Recovery session** operations:

* Automatic acknowledgement on success
* Immediate acknowledgement
* Manual acknowledgement

**Recover Session**: Allows the user to perform a session recover when the AckMode#MANUAL mode is elected while consuming the Message. As per JMS Spec, performing a session recover automatically redelivers all the consumed messages that had not been acknowledged before this recover.

**VM Connector:**

VM Connector is used for intra-app (within the app) and inter-app communication through either Transient or Persistent asynchronous queues.

* **Transient queues:** This type of queue is volatile, meaning the data would be lost if the system crashed or restarted. Transient queues are faster than Persistent queues.
* **Persistent queues:** This type of queue is more reliable; data would be persisted if the system crashed, failed, or restarted. These queues are slower than Transient queues.

**VM Connector is mainly used for the following:**

* To transfer the message from one flow to another.
* To distribute workload (load balancing) across a cluster.
* To communicate with different apps running in the same Mule domain.
* To perform simple queueing.

**VM Connector provides the following operations:**

* **Publish**: To publish the data into the queue. It is asynchronous.
* **Consume**: To pull data from the queue. If there is no message in the queue, then the consume operation waits for up to the configured queue Timeoutvalue, after which the VM: QUEUE\_TIMEOUTerror is thrown.
* **Listener**: To listen to the queue and pick the message.
* **Publish Consume**: To publish the message and then wait for a response from the consuming operation. This operation publishes the content into the queue and waits for a response up to the configured queue Timeout value. If the response is not received in the given time, then the VM: QUEUE\_TIMEOUTerror is thrown. It is synchronous.

**Difference between VM and JMS:**

They all use different queuing/messaging protocols/platforms underneath.

VM is Mule's internal transport for messaging/queueing. The VM transport is for intra-JVM communication between Mule flows. So, that means when you use a VM in your flow, you can communicate between different flows in the application. It can only be used by Mule applications. When creaing a VM queue it can only be accessed by the Mule application that creates it (Cloudhub for example) OR it can be reused by same Mule apps running in a domain project or cluster. No existing broker infrastructure needs to be setup. Supports persistent and transient.

Anypoint MQ is Mulesoft's Cloud Messaging platform. This can be used by other applications - not just Mule. It can also be used across multiple Mule apps regardless of domains or cluster, well suited for Cloudhub applications. No infra setup, all in the cloud. Think Amazon SQS but a lot better and great integration with Anypoint Platform.

JMS uses the Java Messaging Service protocol and requires an external JMS broker such as ActiveMQ. Can be used by any application that supports JMS connectivity.

**Mule Domain Project:**

A Mule Domain Project is implemented to configure the resources that are shared among different projects. These resources can be used by all the projects associated with this domain. Mule applications can be associated with only one domain, but a domain can be associated with multiple projects. Note: Mule Domain Project and the associated Mule applications must be in the same workspace. You cannot define Flows, SubFlows, or any Message Processors inside a Domain Project.

A Mule Domain Project will only have the Global Elements and the Configuration XML tabs in it. We can reference to the Mule Domain Project in our mule projects by doing the following:

* Right click on the project name
* Go to Properties
* Click on Mule Project and there will be an option to select the Domain for this project. From the dropdown select the name of the Domain Project you have created and click on Apply and Close.

**Deploying the Domain Project in the Mule Standalone Server:**

Right-click on the Domain project -> Export -> Mule -> Anypoint Studio Project to Mule Deployable Archive, choose a path, and click Finish.

Repeat the same step for the Mule application.

Once both the executable jars are created, open the file explorer and navigate to the folder where your Mule EE Standalone Server is there. Copy the executable jar for the Domain project and save it inside the “domains” folder and copy the executable jar for the Mule application and save it inside the “apps” folder.

Go to the “bin” folder and click on the “mule” batch file. The server should start and successfully be deployed.

**Securing Properties in Mule:**

**Steps:**

Create a new mule project in Anypoint Studio.

* Drag and drop a HTTP Listener on to the canvas  🡪  Add Salesforce Module  🡪 Select Query operation from the Salesforce Module and drop it on to the canvas
* Go to the src/main/resources folder and right click on it  🡪 Click on File 🡪 write the name of the Configuration file (ex: Myconfig.properties) and click on Finish
* In the Configuration Properties File write the Salesforce username, password, security token, and login Url :
  + [sf.username=lakshmiveda@eniquesolutions.com](mailto:sf.username=lakshmiveda@eniquesolutions.com)
  + sf.password=Srambhs@07
  + sf.tokennumber=tpV8Xqq0uM1riW3Na7U2azP0a
  + sf.loginurl=https://login.salesforce.com/?locale=in

Now download the Secure Properties Tool jar file available in the official MuleSoft documentation (Secure Configuration Properties) and put in a separate location on your disk (ex: D drive)

Go to the folder where you have placed the secure-properties-tool.jar file and type cmd there 🡪 Command Prompt will open.

Type the command: java -cp secure-properties-tool.jar 🡪 It will display the syntax of how to encrypt or decrypt a String or a File

In our project, I am securing the Password and the Security token. Write the command given below to encrypt the password and token:

**Syntax:**

Java -cp secure-properties-tool.jar string <encrypt|decrypt> <algorithm> <mode> <key> <value>

Java -jar secure-properties-tool.jar string encrypt Blowfish CBC MyMulesoft “Srambhs@07”

Here I have given my password as the value and then press enter 🡪 we can see the encrypted value 🡪 copy that value and in the configuration properties file paste the encrypted value for the password as: sf.password=![encrypted string for the password].

Repeat the above process to encrypt the Security Token as well and follow the same syntax using the exclamation mark and the square brackets to store the encrypted value in the configuration properties file.

Now, Go to the main mule project and set the Salesforce connector configuration: As we have given our salesforce login credentials in the config file, we have just refer to them dynamically. So, we just give as following

* username: ${sf.username}
* Password: ${secure::sf.password} (we have to use secure keyword for encrypted values)
* SecurityToken: ${secure::sf.tokennumber}
* Authentication url: ${sf.loginurl}

Now, we need to specify where we have our configuration file. For that Go to Global Elements 🡪 Click on Create 🡪 Global Configurations 🡪Configuration Properties 🡪Enter your Configuration Properties file name (say MyConfig.properties) 🡪 Click on OK

We also need to add Secure Configuration Properties to the Global Elements, for that Click on Add from Exchange in the Mule palette and search for Mule Secure Configuration Properties Extension and add it. Now Go to Global Elements and click on Create 🡪 Connector Configurations 🡪 Select Secure Properties Config 🡪Give the Properties file name (ex MyConfig.properties) and the Key name which we have given while encrypting our password and token (ex MyMulesoft). We also need to specify the algorithm (ex Blowfish) and mode (ex CBC) and Click on OK.

Now we need to write a Query for fetching information we need from the Salesforce.

Save the project and Run it. Once the project gets deployed, we can check it from the Postman.

**Log4J:**

MuleSoft provides application logs and a runtime log to help you monitor and troubleshoot your apps and the Mule server.

Every app that you build in Studio comes with its own log4j2.xml file. The log contains information about any errors raised in the app (unless you have app logic to handle those errors). It also contains anything you want to explicitly log, if you build the logic in the app.

log4j is a tool to help the programmer output log statements to a variety of output targets.

In case of problems with an application, it is helpful to enable logging so that the problem can be located. With log4j it is possible to enable logging at runtime without modifying the application. The log4j package is designed so that log statements can remain in shipped code without incurring a high-performance cost.

log4j is designed with three goals in mind: reliability, speed and flexibility.

**What Is Splunk?**

Splunk is a tool used for logging, analysing, reporting, visualizing, monitoring, or searching the machine data in real time.

**Splunk Main Components**

**Splunk Search Heads**

Splunk Search Head is a Splunk instance that distributes searches across the indexers.

**Splunk Indexers**

Indexers are a Splunk component used to index and store incoming data from forwarders. Splunk instances transform the incoming data into events and stores them into indexes to perform search operations efficiently.

**Splunk Forwarders**

Forwarders are used to collect the data from various sources in a secure, reliable way and forward data to Splunk for indexing and analysis.

**Download and Install Splunk Enterprise On-Premise**

For installing the Splunk on-premise or on your machine, you need to navigate to Splunk Free download and create an account by filling out a form.

There are 2 options: either you can download an on-premise Splunk Enterprise trial for 60 days or a Splunk Cloud trial for 15 days.

Once you create a Splunk account, it will give you the option to download Splunk Enterprise and Splunk Cloud.

Download Splunk Enterprise, during the installation process, you need to create a password for admin users and you can select local accounts during installation. For windows, it will create windows services “**Splunkd Service.**”

Once the Splunk is installed, you can browse to [http://localhost:8000](http://localhost:8000/), and it will navigate you to Splunk Web GUI.

To Login into Splunk, you need to provide username “**admin**” and password that you have created during installation.

**Creating HTTP Event Collector Token**

The HTTP Event Collector (HEC) lets you send data and application events to a Splunk deployment over the HTTP and Secure HTTP (HTTPS) protocols. HEC uses a token-based authentication model.

* For creating tokens, navigate to **Settings → Data inputs → HTTP Event Collector.**
* Once you select the **HTTP Event Collector**, it will navigate you to the screen from where you can create a **New Token**. Click on the **New Token** button.
* Once you click on the **New Token** button, it will navigate to form and start filling out the details like **Name.** Click on **Next**at the top of the web page for filling further details.
* On the next screen, you can select **SourceType**as **log4j,** as we will be using **log4j**as a source for sending data to Splunk from the MuleSoft application.
* On the same screen, select all index main, summary, and history.
* Now click on the **Review**button on top of the screen. Finally, review the details and click on **Submit** Button.
* Once you click on the **Submit**button, it will create a token that we will be using for authentication purposes.
* You can again navigate to **Settings → Data inputs → HTTP Event Collector** to see the newly created token. Also, make sure that token is enabled. In case it is not enabled, you can click on **Global Settings** and enable the token.
* Also, notice the HTTP Port Number **8088**and it will be required for sending logs from the application to Splunk. We will be using URL [http://localhost:8088](http://localhost:8088/) and the token that we have created above for sending logs to Splunk from the MuleSoft application.
* Edit the Token and select **Default Index** as **main**.

**Setting Up Mail Server on Splunk**

* For setting up a mail server, navigate to **Settings → Server Settings → Mail Server Settings.**
* You need to provide Mail host, username, and password. If you are using **smtp.gmail.com**, use port number **465** for SSL and **587** for TLS.
* You can define email format in **Mail Server Settings,** and we will keep the default for this article. Click **Save**.

**Integrating MuleSoft Application with Splunk Enterprise Using Anypoint Studio**

For integrating a MuleSoft application with Splunk, we will be adding **HttpAppender**in **log4j2.xml** located at **src/main/resources**

1

<Appenders>

         <Http name="Splunk" url="http://localhost:8088/services/collector/raw">

             <Property name="Authorization" value="Splunk 42e1dcf7-6e14-4a1b-8485-c5fa15edb47d" ></Property>

             <PatternLayout pattern="[%d{MM-dd HH:mm:ss}] %-5p %c{1} [%t]: %m%n"></PatternLayout>

        </Http>

</Appenders>

For connecting to Splunk from a MuleSoft application, you need to provide the URL where we will be sending logs to Splunk. In our case, the URL is “<http://localhost:8088/services/collector/raw>”, and you need to add an authorization property and value Token we have created above with Prefix Splunk.

We need to define loggers in log4j2.xml.

<AsyncRoot level="INFO">

   <AppenderRef ref="Splunk" ></AppenderRef>

</AsyncRoot>

Also, make sure **AppenderRef**matches the name of **HTTP Appender**.

There is a MuleSoft application created that will be logging information, errors, and other information into Splunk.

Once you have started the Mule application and have sent the request, it will start logging events into Splunk.

**Searching Events with Splunk**

For viewing the events, you can go to the Splunk home page and select **Search & Reporting**.

It will navigate you to the search page. On that page, you can use a filter for searching the events and can view a **Data Summary**. The data summary will show you all the hosts, sources, and source types from where the events are coming from.

These hosts, sources, and source Types can be used to search for events.

**Integrating MuleSoft Application with Splunk Enterprise Using Anypoint Platform and Anypoint Server Group or Cluster**

For integrating Splunk Enterprise on-premise with Anypoint Server Group or Cluster, you need to create a cluster or server group on Anypoint Platform Runtime Manager.

**Adding Server on Anypoint Platform Runtime Manager**

For adding a server to Anypoint Platform Runtime Manager, navigate to **Runtime Manager → Servers → Add Servers**.

Once you click on **Add Servers**, it will give the command that you need to run on your machine to cd into /bin folder of Mule runtime.

After running the command, you will see the server will be added to Anypoint Platform runtime manager, and it will be a deployed agent on your local machine under**/conf**folder. Agent details can be found in **mule-agent.yml**. This will establish secure communication between Anypoint Platform Runtime Manager and your local machine Mule runtime.

**Creating Server Group**

For creating a server group on Anypoint Runtime Manager, navigate to **Runtime Manager → Servers → Create Group**.

Define the group name and add the servers that you need in that server group. To add a server into a group, it must be in a running state. Click Create Group, and it will create a server group.

You can add more than one server into Server Group.

**Deploying MuleSoft Application to Server Group**

For deploying an application to the server group, navigate to **Runtime Manager → Applications → Deploy application**

For deploying the application to a server group, you need to select **Deployment Target** as your server **Group Name**. Click on **Deploy Application**. This will deploy applications to all the servers within that particular server group.

**Enabling Splunk on Server Group Using Anypoint Runtime Manager**

For enabling Splunk on Server Group, navigate to **Runtime Manager → Servers → Click on your server group → Manage Group**

After clicking on **Manage Group**, click on the Plugins tab and you will see Splunk and ELK plugin.

Click on the Setting icon in front of Splunk to configure it.

Configure Splunk Event Tracking as:

You need to provide the host and port number of Splunk. In our case, the port number is 8088, and the Token is what we have created at the top of article.

Splunk Source is mule-events and it will be useful for searching events coming from Anypoint Runtime Manager. Click **Apply**.

This tracking will be applicable for all applications deployed within this server group.

We will be executing an application deployed on the server group and it will start logging the data into Splunk, which can be used for generating alerts, reporting, visualizing, troubleshooting, and analytics.

Now you can see the new **Source=mule-events** and **sourceType=mule** have been added to the data summary.

Similarly, you can enable Splunk logging for servers under the cluster.

**Generating the Reports and Email Alerts with Splunk**

There are default reports available in Splunk. To view reports, navigate to **Settings → Searches, reports, and alerts.**

We will run Errors in the last 24 hours report by clicking on run. This will show all the errors or exceptions that occurred in the last 24 hours.

For scheduling the report, click on **Edit → Edit Schedule**.

Once you've clicked on **Edit Schedule**, it will open a pop-up window and start filling the details like schedule, CRON expression, and Triggered Actions (send email) as shown in the below screenshot. Click **Save**.

Another way of scheduling PDF: Navigate to **Splunk Homepage → Search & Reporting → Reports**

Click on **Errors in the last 24 hours → Add to dashboard**.

It will open a pop-up and fill in the Dashboard title. Click **Save**.

Once saved, it will open a pop-up window. Click on **View Dashboard**.

Now click on **Export → Schedule PDF Delivery**.

This will open a pop-up window, and you can schedule a PDF delivery. We have set up a CRON scheduler to run reports every one minute, and this will send an email every one minute with all errors that occurred in the last 24 hours. Click **Save**.

**FTP Connector Configuration:**

Anypoint Connector for FTP (FTP Connector) provides access to files and folders on an FTP server. Its main features include:

* The ability to read files or fully list directory contents on demand.
* Support for common FTP operations such as creating directories and copying, moving, renaming, and deleting files.
* Support for locking files.
* File matching functionality.
* A design that is fully consistent with the File and SFTP connectors. The same set of operations is available on all these connectors, and they behave similarly.

To set up the connector configuration for FTP, we need to provide the following:

* Working Directory (ex: empdir)
* Username (ex: the user which we create in the FTP server (File Zilla), in this case say, ftpuser)
* Password (ex: the password which we create in the FTP server while defining the user, say, ftpuser)
* Host (if the FTP server is downloaded in our machine, then it will be localhost)
* Port (ex: for FileZilla it is “20”)

**SAP Connector:**

MuleSoft SAP Connector enables Mule runtime engines to support SAP integration, as a certified SAP Java connector that leverages SAP Java Connector (JCo) libraries.

**Usage of SAP Connector**

* Execute the BAPI functions over the following type of RFCs (Remote Function Calls).
  + Synchronous Remote Function Call (sRFC).
  + Asynchronous Remote Function Call (aRFC).
* Sends and Receive IDocs over Transactional Remote Function Call (tRFC) and Queued Remote Function Call (qRFC).
* Transform SAP Object from and to XML.
* Act as a JCo server to be called a BAPI over sRFC and aRFC.

By default, SAP connector is not available in the AnyPoint Studio palette. You need to search and download SAP connector from an exchange.

**Establishing the connection with SAP Application Server:**

For establishing the connection, you will be requiring below **JCo** libraries that you can get it from the SAP marketplace.

Two Multi-Platform Libraries

* sapjco3.jar
* sapidoc3.jar

One JCO native library

* sapjco3.dll (Windows)
* libsapjco3.jnilib (Mac OS X)
* libsapjco3.so (Linux)

Now, you can start connector configuration and first make sure that you provide all three JCO libraries in connector configuration.

First, we will configure IDoc Library by clicking on **Configure -> Use local file.** Browse sapidoc3.jar.

Click **OK**. Similarly, configure the JCo library and JCo native library.

To connect SAP Application Server, you need to provide Username, Password, SAP system number (e.g., 00, 01), SAP client ID (e.g., 500, 800), Application server host.

After doing all this configuration, you can simply click on Test connection to check whether all configuration is correct and we are able to connect SAP.

Click **OK**. This completes the SAP connector configuration. Now you need to provide Key (i.e., Function that you have to call from SAP to retrieve data) and Metadata in Key will be populated automatically. Click on the search icon and select the Function that you need to execute.

**Considerations while deploying to Cloudhub:**

When you are deploying your application to CloudHub Runtime Manager, you need to make sure that JCo native library is libsapjco3.so added at build path and in pom.xml because Cloudhub workers are running on Linux operating system otherwise it will give error ***java.lang.ExceptionInInitializerError: JCo initialization failed with java.lang.UnsatisfiedLinkError: no sapjco3 in java.library.path***.

**Batch Processing:**

The name itself says a batch (more than one) and processing (process). Batch processing is nothing but process more than one record simultaneously. In simple words, process messages in batches **asynchronously**.

Batch processing splits messages into individual records, performs actions upon each record, and then reports on the results and potentially pushes the processed output to other systems or queues.

* Splits the messages into individual records.
* Performs actions upon each record. For example, transformation, enriching data, etc.,
* Creates reports on the results. For example, how many records succeeded and failed.
* Pushes the processed output to other systems. For example, pushing output to database or salesforce.

In simple words, When the batch job starts executing, Mule splits the incoming message into records, stores them in a persistent queue, and schedules those records in blocks of records to process.

By default, the runtime stores 100 records in each batch step and assigns **16 threads** to each job.

**Batch Job Phases:**

There are basically three phases in Batch job. People sometimes argue that there are 4 phases right? Yeah, they do consider the **input** phase as well, so let’s close the argument.

Here are the phases in batch processing:

1. **Load and Dispatch**

During this phase, runtime performs all the work behind the scenes such as create batch job instances, convert payload into collection of records and split the collection into individual records.

1. **Process**

      In this phase, Mule process all individual records **asynchronously**and also allows you to

      filter records. There are many things you can perform in this phase such as filtering only

      failure records, setting accept expressions (Watch this movie if you are 18+) and you can

      push records in bulk using batch aggregator.

1. **On Complete**

       During this phase, you can configure the runtime to create summary of the records

       processed for the batch job instance. Which means, it will give us how many records

       succeeded, how many failed, and how many records loaded, etc.,

Refer to the following link for detailed explanation: <https://vanchiv.com/what-is-batch-processing-in-mule-4/>

**What is SOAP?**

* SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol
* SOAP is an application communication protocol
* SOAP is a format for sending and receiving messages
* SOAP is platform independent
* SOAP is based on XML

**SOAP Building Blocks:**

A SOAP message is an ordinary XML document containing the following elements:

* An Envelope element that identifies the XML document as a SOAP message
* A Header element that contains header information
* A Body element that contains call and response information
* A Fault element containing errors and status information
* A SOAP message MUST be encoded using XML
* A SOAP message MUST use the SOAP Envelope namespace

**SOAP Skeleton:**

<?xml version="1.0"?>  
  
<soap:Envelope  
xmlns:soap=<http://www.w3.org/2003/05/soap-envelope/> ---🡪 Namespace  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  
<soap:Header>  
...  
</soap:Header>  
  
<soap:Body>  
...  
  <soap:Fault>  
  ...  
  </soap:Fault>  
</soap:Body>  
  
</soap:Envelope>

The required SOAP Envelope element is the root element of a SOAP message. This element defines the XML document as a SOAP message.

The namespace defines the Envelope as a SOAP Envelope.

The encodingStyle attribute is used to define the data types used in the document.

The optional SOAP Header element contains application-specific information (like authentication, payment, etc) about the SOAP message.

If the Header element is present, it must be the first child element of the Envelope element.

**Note:** All immediate child elements of the Header element must be namespace-qualified.

The required SOAP Body element contains the actual SOAP message intended for the ultimate endpoint of the message.

Immediate child elements of the SOAP Body element may be namespace-qualified.

The optional SOAP Fault element is used to indicate error messages.

The SOAP Fault element holds errors and status information for a SOAP message.

If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message. Fault element contains:

Fault code, fault string, fault factor, detail

**SOAP BINDING:**

SOAP bindings are mechanisms which allow SOAP messages to be effectively exchanged using a transport protocol.

Most SOAP implementations provide bindings for common transport protocols, such as HTTP or SMTP.

HTTP is synchronous and widely used. A SOAP HTTP request specifies at least two HTTP headers: Content-Type and Content-Length.

SMTP is asynchronous and is used in last resort or particular cases.

Java implementations of SOAP usually provide a specific binding for the JMS (Java Messaging System) protocol.

**What is SOAP Action?**

soap action is the url of the operation, which you are going to perform.

Suppose there are 3 operations in a wsdl namely deposit, withdraw and cancel.  
if u select deposit operation, soap action will automatically populate a url with target namespace and operation, which indicates the operation that you have selected.

Soap action is used to identify operations. For every operation, soap action will have some unique value. This value is used by service to identify which operation it has to execute.

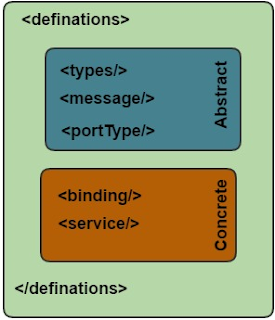
**ABSTRACT & CONCRETE WSDL:**

An abstract WSDL document describes what the web service does, but not how it does it or how to contact it. An abstract WSDL document defines:

* the operations provided by the web service.
* the input, output and fault messages used by each operation to communicate with the web service, and their format.

A concrete WSDL document adds the information about how the web service communicates and where you can reach it. A concrete WSDL document contains the abstract WSDL definitions, and also defines:

* the communication protocols and data encodings used by the web service.
* the port address that must be used to contact the web service.



**WSDL:**

The **Web Services Description Language** (WSDL) is an XML language for describing web services. **WSDL** has five major elements–**types, message, portType, binding, and service.**All these major elements may be defined 0 or more times in a WSDL document, except for <types>, which may be 0 or 1 time. Here’s a short description of each:

**types:** This is where XML types to be used in the WSDL document are defined. Traditionally, this has meant using XML Schema.

**message:** This is the section where the input or output parts of an operation are defined, i.e. the “**parameters**” or “**return types** “.

**portType:** Here is where the operations that a web service offers are defined in terms of messages (input and output, with faults).

**binding:** This is the “how” of a service, specifying the binding of the operations defined in the portType(s) to specific protocols, such as SOAP.

**service:** This is the “where” of the service, specifying the address where a bound operation may be found.

**Example of WSDL document:**

 <?xml version="1.0" encoding="UTF-8" ?>  
<definitions targetNamespace="urn:Emp" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="urn:Emp"  
             xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"  
             xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
             xmlns:weo="http://www.example.org">  
  **<types>**  
    <xsd:schema targetNamespace="urn:Emp/types" elementFormDefault="qualified"/>  
    <xsd:schema>  
      <xsd:import schemaLocation="../Schema/Emp.xsd" namespace="http://www.example.org"/>  
    </xsd:schema>  
  **</types>**  
  **<portType name="EmpDetailsPort">**  
    <operation name="EmpRequestOperation">  
      <input message="tns:EmpDetailsPort\_EmpRequest"/>  
      <output message="tns:EmpDetailsPort\_EmpResponse"/>  
    </operation>  
  **</portType>**  
  **<message name="EmpDetailsPort\_EmpRequest">**  
    <part name="part" element="weo:EmpRequest"/>  
  </message>  
  <message name="EmpDetailsPort\_EmpResponse">  
    <part name="part" element="weo:EmpResponse"/>  
  **</message>**  
  **<binding name="EmpDetailsPortSOAP11Binding" type="tns:EmpDetailsPort">**  
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>  
    <operation name="EmpRequestOperation">  
      <soap:operation style="document" soapAction="urn:Emp/EmpRequestOperation"/>  
      <input>  
        <soap:body use="literal" parts="part"/>  
      </input>  
      <output>  
        <soap:body use="literal" parts="part"/>  
      </output>  
    </operation>  
  **</binding>**  
  **<service name="EmpDetailsPort">**  
    <port name="EmpDetailsPortPort" binding="tns:EmpDetailsPortSOAP11Binding">  
      <soap:address location="http://www.example.com"/>  
    </port>  
  **</service>**  
</definitions>