Deloitte.

Mule ESB

Capability as a lightweight, Java-based Enterprise Service Bus (ESB)

Presented by the Java Community

Solution Integration Center of Excellence

Contents

Abstract

This document looks into the capability of Mule ESB as a lightweight, Java-based ESB and integration platform that allows developers to connect applications together quickly and easily, enabling rapid data exchange.

Mule ESB supports simple integration of existing systems, regardless of different technologies that the applications use, including JMS, Web Services, JDBC, HTTP, FTP and SFTP, and email functionality.

This document highlights several challenges in Project X where Mule ESB is used extensively to address various business scenarios.

Mule ESB

Mule ESB is a widely used open source enterprise service bus that is lightweight, flexible, and easily adaptable to one's existing infrastructure. Additionally, it is robust enough to power even the largest and most demanding enterprise SOA implementations. Mule ESB simplifies the integration of applications and technologies by taking the complexity out of integration and enabling developers to quickly and easily build high-performance, multi-protocol interactions between heterogeneous systems and services.

Mule supports a large variety of transports such as JMS, HTTP(s), Web Services, JDBC and many others, making integration with commonly used protocols effortless.

Because it does not impose architecture limitations, Mule ESB enables rapid response to changing business needs and opportunities. Mule is ideally suited for today's business environment where partner ecosystems are constantly changing and new web-based and mobile channels are emerging.

Why use Mule ESB?

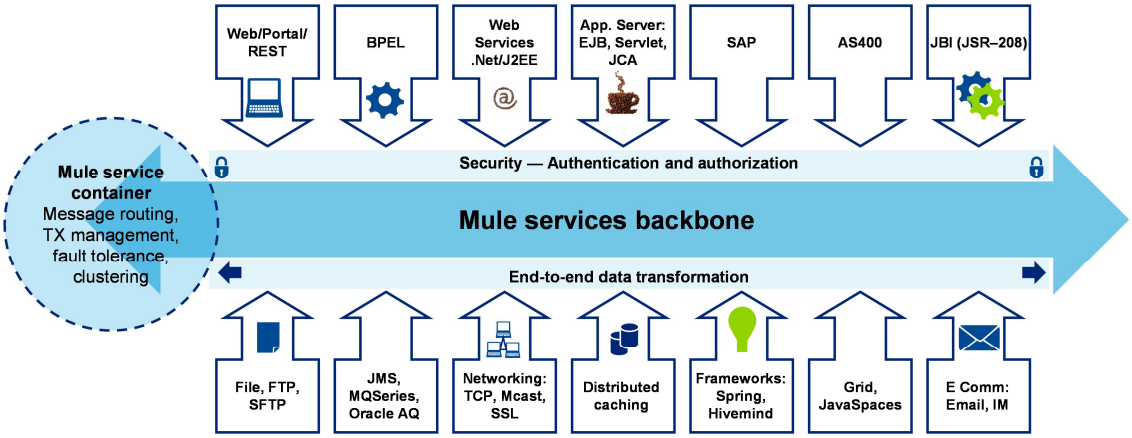
Mule ESB should be considered as a potential solution if one or more of the following statements are true:

1. Need to integrate three or more applications or services
2. Need to plug in more applications in the future
3. Need to use more than one type of communication protocol
4. Need to have polling mechanism for a particular file and SFTP/FTP it to some destination location
5. Need to publish services for consumption by other applications
6. Need to consume services provided by the external applications, such as Adobe

A closer look at Mule ESB as an integration platform

ESBs allow different applications to communicate with each other by acting as a transit system for data between applications within your enterprise or across the Internet. Mule ESB contains powerful capabilities that support this data integration, including:

- **Service creation and hosting** — Expose and host reusable services using Mule ESB as a lightweight service container
- **Service mediation** — Shield services from message formats and protocols, separate business logic from messaging, and enable location-independent service calls
- **Message routing** — Route, filter, aggregate, and re-sequence messages based on content and rules
- **Data transformation** — Exchange data across varying formats and transport protocols



What is a Mule Flow

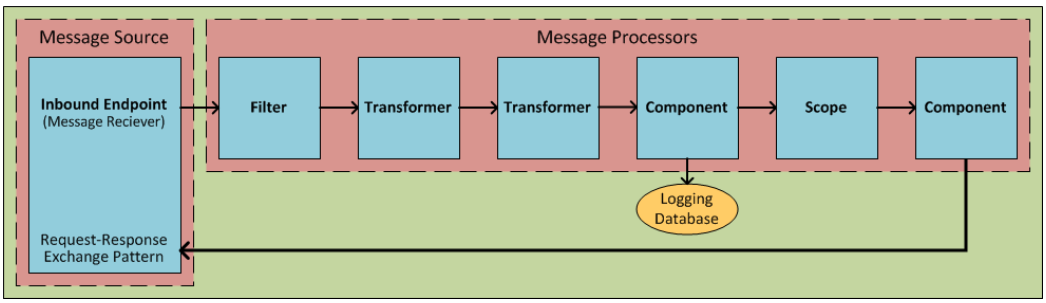
A Mule Flow is a new way to orchestrate services. Flow is a very flexible way to build integrations by simply choosing building blocks from Mule's wide range of functionality. It is further explored in the business scenarios described in this document.

Flows support synchronous and asynchronous child flows, one-way and request-response exchange-patterns, and other architectural options. Flows can be particularly effective for the following use cases:

- Simple integration tasks
- Scheduled data processing
- Integrating Cloud-based and on-premise applications
- Event processing where multiple services need to be orchestrated

At the simplest level, Flows are sequences of message-processing events. As the following diagram illustrates, a message that enters a flow may be:

- validated (filtered)
- enriched (appended)
- transformed into a new format
- processed by custom-coded business logic
- logged to a database
- evaluated to determine what sort of response gets returned to party that submitted the original message



You can refer to **http://www.mulesoft.org** for more details on how to configure a Mule Flow.

Business Scenario 1: Mule for Webservices

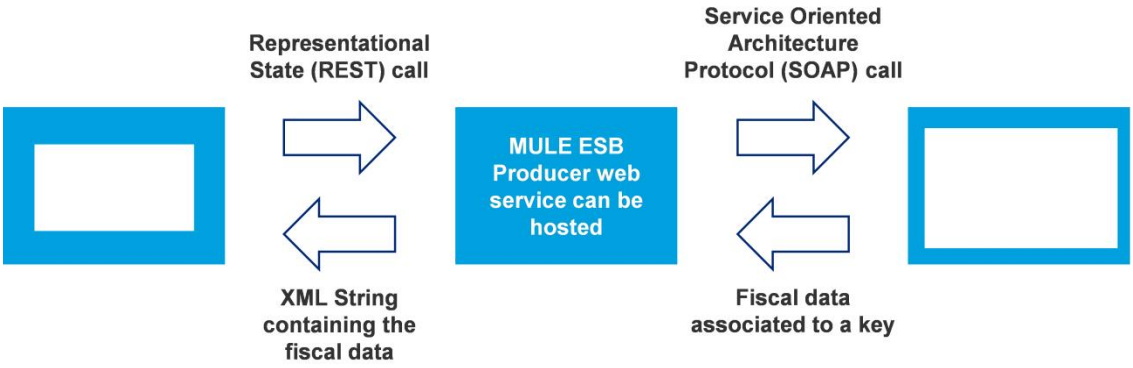
State Accounting Budget and Human Resource System (SABHRS) is an enterprise application implemented by one of the states of the United States (U.S.) to assist state agencies in recording the disposition, use, and receipt of public money and property in accordance with the state law. It also assists in the administration of human resource information, including the generation of a biweekly payroll. For this purpose, SABHRS needed to interact with the Shared Fiscal Services layer (SFSL) to gather certain key fiscal data information.

Problem statement

SABHRS exposed Web Services in order to exchange fiscal data with any third-party subsystem with which it had an established contract. An integration solution was required in order to carry out data transfer between SFSL and SABHRS and consume these Web Services exposed by SABHRS.

Technical solution

Mule ESB is introduced as a transit system which allows easy integration of existing systems regardless of the technologies, here: SABHRS and SFSL.



A batch program in SFSL makes a REST call to Mule ESB server as highlighted in the diagram. A REST call is not tightly coupled to a contract or Web Services Description Language (WSDL), as in the case of SOAP Web Service calls, thus providing scalability of component interactions and generality of interfaces. Unlike SOAP, REST does not require XML parsing and does not require a message header to and from a service provider, which is required for interaction between SFSL and ESB layer. This in turn improves performance. There are 4 steps involved in leveraging MULE ESB to make a REST call.

1. Define a flow in mule-config.xml file

A flow is defined in the Mule configuration file using a Jersey third-party API and annotation-based REST call. This is used for configuring a Java component; i.e., INSndRcvRouterWS java class.

```
<flow name="InterfaceSndRcv">
     <inbound-endpoint address="MULE_URL"/>
     <jersey:resources>
          <component class="INSndRcvRouterWS"/>
     </jersey:resources>
</flow>
```

2. Invocation of a REST call

A Jersey Client class is used to set the MULE_URL appended with PATH, which in turn returns a WebResource Class object as shown in the code snippet below. MultivaluedMapImpl utility class of Jersey sets the Reader object, which is a string and a function name. This returns a MultiValuedMap, which is set in the post method, in order to make a REST call. The path is used to identify a process on the ESB using annotations.

```
Client client = Client.create();
WebResource wsClient = client.resource(MULE_URL+PATH);
MultivaluedMap<String, String> queryParams = new MultivaluedMapImpl();
queryParams.add("READER_OBJECT", <xml string>);
queryParams.add("FUNCTION_NAME", functionName);
String postResp = wsClient.path("PROCESS_TYPE").
post(String.class, queryParams);
```

Here, 'postResp' can be a string with status code information of the REST Web Service call.

3. Invocation of REST on Mule ESB

When a REST call is made, MULE_URL(a mule server location), identifies the flow, which in turn identifies a jersey resource based on the PATH appended with the MULE_URL. PATH is configured using annotations as shown in the code snippet below:

```
@Path(PATH)
public class INSndRcvRouterWS {

@POST
@Path("PROCESS_TYPE")
@Produces(MediaType.TEXT_PLAIN)
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)

public String consumeReq(@FormParam("READER_OBJECT") String readerObj,
     @FormParam("FUNCTION_NAME")String functionName) {

Reader reader = new StringReader(readerObj);
INSndRcvDataController dataController = new INSndRcvDataController();
     String returnCode=dataController.initializeSndAction(reader, functionName);
return returnCode;
}
     }
```

Here, the annotations with @ symbol help in identifying, PROCESS_TYPE, READER_OBJECT, FUNCTION_NAME, PATH, and type of REST call, which is POST in this case. Java class defined on Mule ESB server is used to call/invoke consumer Web Service hosted on any third-party trading partner system directly using Axis 2 or JAX-WS Java API through SOAP calls.

4. Secure consumer Web Services

In order to make secure third-party Web Service calls from Mule ESB, Axis 2 Authenticator for HTTP can be used with Base 64 encryption as shown below:

```
    Authenticator authenticator = new Authenticator();
    List<String> auth = new ArrayList<String>();
    auth.add(Authenticator.BASIC);
    authenticator.setAuthSchemes(auth);
    authenticator.setUsername(Base64.encode(user));
    authenticator.setPassword(Base64.encode(password));
    authenticator.setPreemptiveAuthentication(true);
    service._getServiceClient().getOptions().setProperty(HTTPConstants.AUTHENTICA
TE,  authenticator);
```

Business Scenario 2: Mule with Adobe

Combined Healthcare & Integrated Eligibility System (CHIMES) for SNAP/TANF is an integrated eligibility solution that helps case workers determine benefits provided by multiple programs of a state of the U.S. for applicants effectively and efficiently. State requires online generation of correspondences consisting of forms and notices to be issued to citizens of United States on a need basis.

Problem statement

A method to enable Adobe Life Cycle platform to ingest CHIMES data needs to be developed, in order to generate notices and forms in PDF documents

Technical solution

The data to be displayed on the PDF is exchanged as data packets in the consumer Web Service hosted on the Adobe LiveCycle designer server. The user interface of the online Web application for CHIMES-EA would give a call to the consumer Web Service, the configuration of which is done in the Mule configuration file by generating a client from WSDL and configure the client and HTTP endpoint in a Mule configuration XML built on Spring using CXF Java API as shown below:

1. Generate a CXF client using the WSDL to Java tool from CXF
2. Configure the client in the Mule configuration XML file using the <cxf:jaxws-client> component as shown in the code snippet

```
<flow name="adobe">
     <https:inbound-endpoint connector-ref="httpsConnector"
          address="${adobe.wsdl}" exchange-pattern="request-response">

          <mule-ss:http-security-filter realm="mule-realm"
          securityProviders="basic-auth-provider"/>

          <cxf:jaxws-service serviceClass="<website address cleansed>"/>

     </https:inbound-endpoint>
     <component class="<Website address cleansed>"/>
</flow>
```

3. Configure an endpoint in the Mule configuration XML file that will be the transport to request the service (usually HTTP)

Business Scenario 3: Mule with CCI

The Department of Public Health & Human Services (DPHHS) needed a central service to uniquely identify persons participating in multiple benefit programs. Also, DPHHS needed to streamline its client registration process in order to offer timely and accurate data to staff.

Problem statement

Integration of a common system whose functions will be used by the DPHHS CHIMES-EA Enterprise Architecture (EA) and can be extended to other systems in the future for adding, updating, and finding persons. Currently DPHHS has multiple systems administrating different benefit programs. These systems currently store and process person demographic information separately. Currently DPHHS staff also has to enter the person information in multiple systems to provide the appropriate benefits. A change in person information is not automatically reflected or propagated to all applicable systems, which results in data quality challenges and duplicate individuals.

Technical solution

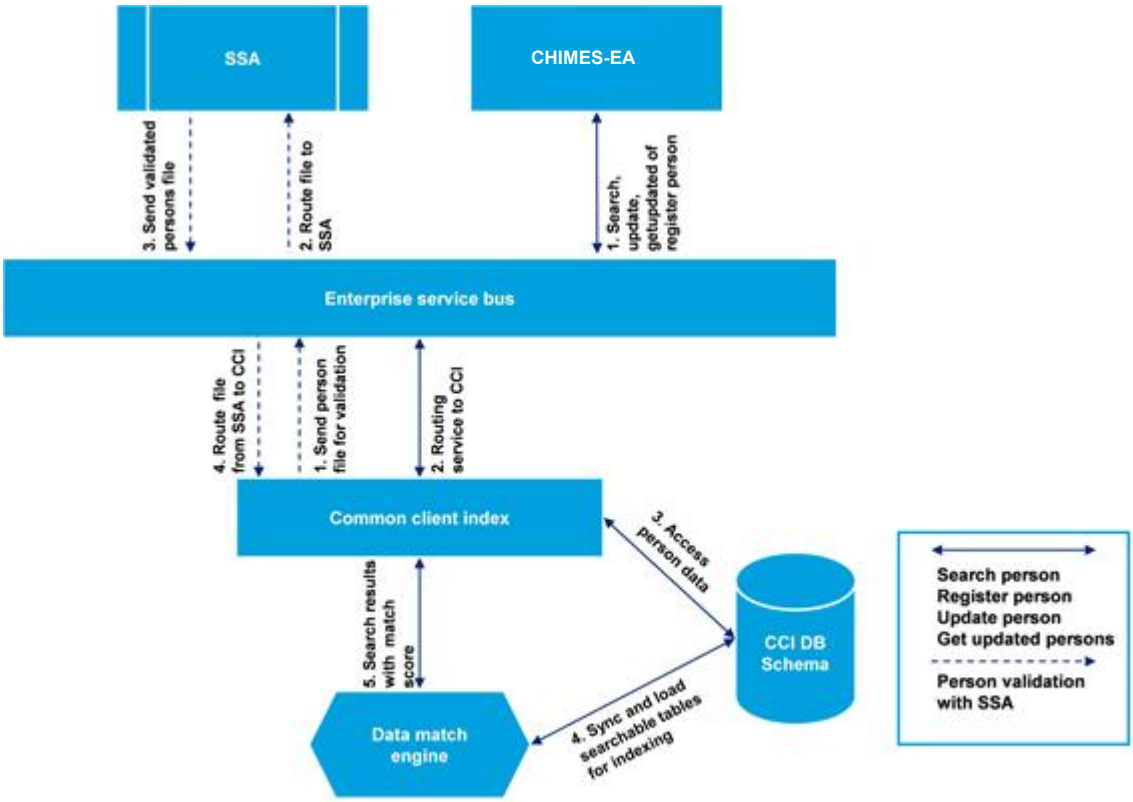
The Common Client Index (CCI) will be developed to enhance the sharing of demographic information and make it extendable to future systems. The EA will contain a common index of individuals and track the participation of these individuals in various systems. The CCI will be realized as a “service” within a service-oriented architecture, not a separate repository of person demographic information. Each system will maintain its own set of person data and business processes. Data will be exchanged based on the access level of the requestor and system functionality.

The ESB will be used for enabling communication within the multiple systems requiring CCI functions. The ESB will be used to facilitate a common communication channel and hub for all CCI requests.

CCI has provided producer Web Services, which can be called through Mule as below:

```
<flow name="CCI">
     <https:inbound-endpoint connector-ref="httpsConnector" address="${cci.wsdl}"
exchange-pattern="request-response">
     <mule-ss:http-security-filter realm="mule-realm" securityProviders="basic-
auth-provider"/>
     <cxf:jaxws-service serviceClass="<Website address cleansed>">
     <cxf:outFaultInterceptors>
     <spring:bean id="outfault"
class="gov.mt.dphhs.cci.util.UnwrapComponentExceptionSoapFaultOutInterceptor"/>
     </cxf:outFaultInterceptors>
     </cxf:jaxws-service>
     </https:inbound-endpoint>
     <component class="<Website address cleansed>"/>
</flow>
```

The below diagram shows seamless integration of Mule ESB and CCI.



Business Scenario 4: Mule for SFTP

DPHHS wanted to come up with a generic service, which would automate the process of transferring files between the different trading partner systems, like Accounts Receivable Management System (ARMS), CHIMES EA, SFSL, etc. DPHHS wanted to leverage Mule ESB to achieve secure file transfer.

Problem statement

DPHHS needed a generic solution to transfer files over SFTP across all the trading partner locations to help avoid errors caused due to manual transfers.

Technical solution

SFTP service will be developed to download files from or upload files to a secured location. The file transfer pattern will allow us to loosely couple two applications together, to optionally use streaming support for larger files, and asynchronously chain other endpoints with an SFTP endpoint. It will also allow us to use Mule ESB's robust error handling in your Mule application.

A simple flow for SFTP can be implemented in a way below:

1. **Create a connector (for inbound endpoint)**

```
<sftp:connector name="muleInboundConnector" maxConnectionPoolSize="1"
     sizeCheckWaitTime="…" pollingFrequency="…">
</sftp:connector>
```

2. **Create a connector (for outbound endpoint)**

```
<sftp:connector name="sftpOutboundConnector"
     identityFile="${sftp.identityFile}" passphrase="<anyvalue>"
     maxConnectionPoolSize="1">
</sftp:connector>
```

3. **Create a flow**

```
<flow name="searchs_loc">
     <sftp:inbound-endpoint connector-ref="muleInboundConnector"
          host="…" port="…" path="…" user="…" password="…">
          <file:filename-regex-filter
               pattern="${<Value of REEGEX>}"/>
     </sftp:inbound-endpoint>
     <sftp:outbound-endpoint connector-ref="sftpOutboundConnector"
          transformer-refs="setProperty01 FilenameTransformer"
          host="${sftp.host.dphhssftp.host.uploading}"
          port="${sftp.host.dphhssftp.port.uploading}"
          path="${searchs.loc.dst.folder}"
          user="${sftp.username.dest.uploading}"
          password="${sftp.password.dest.uploading}">
     </sftp:outbound-endpoint>

     <default-exception-strategy>
          <flow-ref name="moveToErrorandEmail"/>
     </default-exception-strategy></flow>
```

Business Scenario 5: Mule with ARMS

Accounts Receivable Management System (ARMS) is a comprehensive application for debt collection management. ARMS manages the debt collection process by sending reminders by email or SMS to credit managers and customer contacts whenever payments are due or delayed. ARMS is one of the trading partners that SFSL interacts with at one of States of US. ARMS provide information to process accounts receivable claims, post collection activity, forward litigation claims, and provide remittance information.

Problem statement

ARMS (one of the trading partner system) exposed database-stored procedures in order to exchange debtor demographic, debt recovery, and delinquent collection data.

DPHHS needed a service designed for a two-way information exchange between SFSL and ARMS.

Technical solution

1. A batch program in SFSL has been written to make a REST call to Mule ESB server.
2. In the Mule ESB configuration file, a JDBC connector is defined to call stored procedures of ARMS. The authentication details, like database URL, username, and password, are defined in the properties files. For calling a stored procedure, any Object Relation mapping API of Java, like Hibernate or Fast4j can be used. A code snippet is shown below:

```
<spring:bean id="armsJdbcDataSource"
     class="org.enhydra.jdbc.standard.StandardDataSource"
     destroy-method="shutdown">

<spring:property name="driverName"
          value="oracle.jdbc.driver.OracleDriver"/>

<spring:property name="url"
          value="${armsdb.url}"/>

<spring:property name="user" value="${armsdb.username}"/>

<spring:property name="password" value="${armsdb.password}"/>
</spring:bean>

<jdbc:connector name="armsJdbcConnector" dataSource-ref=
     "armsJdbcDataSource"/>
```

Technical Challenges

Applications on the Mule Server fail to deploy abruptly

Say mule.xsd and mule-jersey.xsd present in the lib\jars, refer to 3.1 version. If your import in the mule-config.xml for any application is mentioned as **http://www.mulesoft.org/schema/mule/jersey/3.0/mule-jersey.xsd**, then when the application is loaded, upon mule restart, the application will try to look over Internet for the libraries of 3.0 since the header of the mule-config specified 3.0 version libraries to be referred and the mule core library folder have 3.1 version files.

It may happen that the machine where Mule Server is deployed do not have access to Internet. This will not allow that application to be deployed on the Mule Server and Mule Server will behave in an unexpected manner. So, the fix is to change the import in mule-config.xml to refer to 3.1 version so that its local version of libraries would be referred upon restart. See 3.1 version in the URL:
http://www.mulesoft.org/schema/mule/jersey/3.1/mule-jersey.xsd

Similarly, for mule.xsd in the mule-config.xml

http://www.mulesoft.org/schema/mule/core **http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd** to
http://www.mulesoft.org/schema/mule/core **http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd**

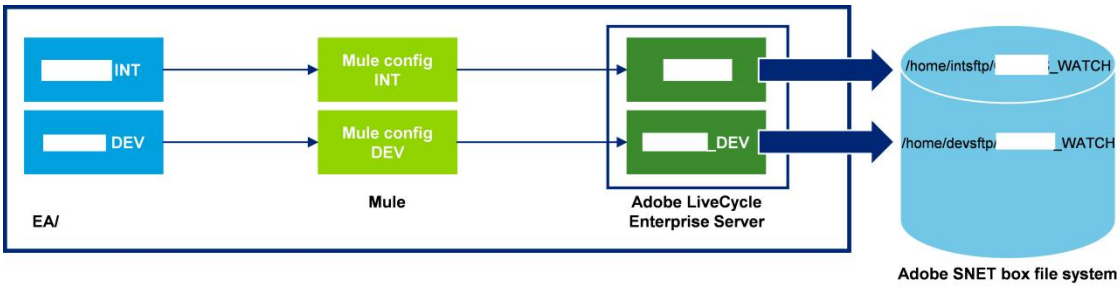
This will ensure the application to use the mule.xsd and mule-jersey.xsd from the mule local jars and not try to fetch it from the URL.

Some applications fail to be deployed on Mule Server

Ports that are used for HTTP and HTTPS should not be same. Also ensure that two application do not use the same port.

Port conflicts when multiple instances of Mule are running

Always ensure that ports are opened and that two ports are not in use by two or more applications.



In the above snapshot, two instances of Mule are deployed on same server. The calling application sends request to specific port, which helps to identify which Mule Server is to be invoked.

Conclusion

Project X successfully leveraged Mule ESB to address several complex business scenarios. The project thus delivers to a degree on high integration, flexibility, decentralized operation, and scalability.

Based on the real time scenarios, our point of view is that Mule is an open source application that helps easy integration of different types of applications which potentially features point to point integration and modularization of the systems architecture, thus resulting in ease in testing and scaling the applications.

References

http://www.mulesoft.org

Contributors

Primary contributors

KunalShah – kunashah@deloitte.com
Kunal Shah is a Senior Consultant in the Systems Integration Service Line in Deloitte. Kunal has 2.4 years of experience in Public Sector projects in Deloitte. Overall 8 years of IT experience with 5.8 years of experience in Internet Banking technology sector [BFSI]. Kunal is working in J2EE domain and holds TOGAF certificate.

Aditi Bhatnagar – adbhatnagar@deloitte.com
Aditi Bhatnagar is a Consultant in the Systems Integration Service Line with 5 years 4 months of experience, developing and maintaining mission-critical applications utilizing Java based technologies and frameworks. She holds 'The Sun Certified Java Programmer, Oracle Certified Expert' – Java EE 6 Enterprise Java Beans Developer and Web Component Developer. Additionally, she has hands on experience of Adobe Flex technology and HL7 heath standard.

Kshama Chopra - kschopra@deloitte.com
Kshama Chopra is a Consultant in the Systems Integration Service Line with 7 years and 3 months of work experience in development, design and maintenance of Java/J2EE based client/server web applications. She holds FFP (Finance Foundation Program) certification and currently working in Public Sector project.

Java Community of Practice

The Java Community of Practice is a community of technology professionals whose primary focus is Java based technology and its applications. The community provides a platform for collaboration, exchange of ideas and encouraging contemporary best practices among its members. Our goal is to continuously enhance our capability to deliver technology solutions across different industries.

More information about the Java Community of Practice as well as additional collaboration and Java resources can be found at our KX site **here**.