Unit 4 – Managing Cloud Integration

Developing with SAP Integration Suite

C_CPI_14

Agenda

- Introducing Cloud Integration

- Business Scenario

- Explaining Development Cycle

- Message Monitoring and Logging

- Camel Data Model and Simple Expression Language

Introducing Cloud Integration

- Supports end-to-end process integration through exchange of messages

- Based on open source framework Camel from Apache Software Foundation

- Core capabilities of SAP BTP Integration Suite

- Low Code / No Code approach
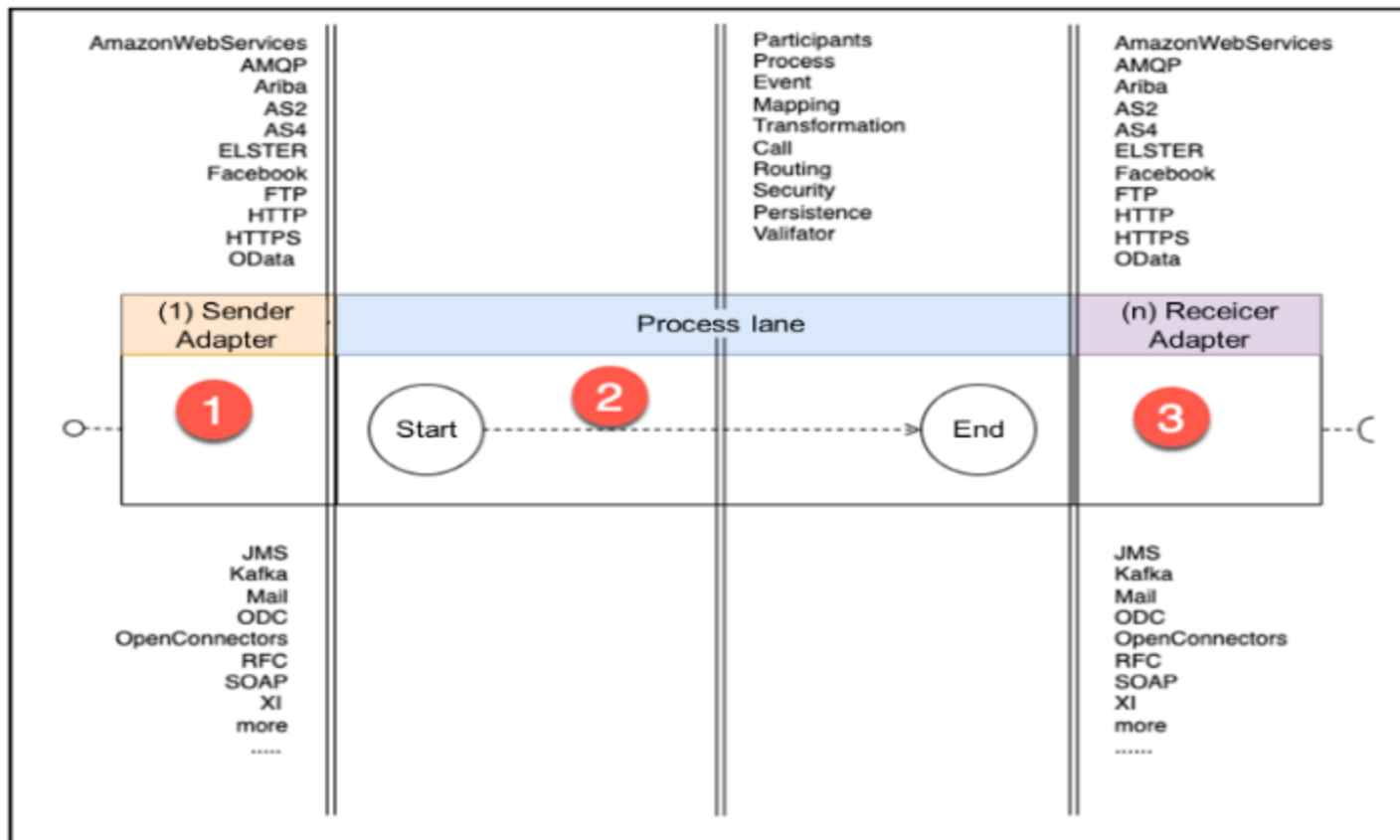
Key Features of Cloud Integration

- Integration flow has 0-1 sender adapter
- Message is delivered via an endpoint
  - If an adapter is configured
- Process is started via Start event
- Different ways messages can be processed
- Receiver adapters can be configured
- Message processing can be synchronous or asynchronous

Apache Camel - Challenge

- Companies have data in various systems

- Need to move data between various systems

- Writing a program entails understanding…
  - Protocols of the various systems
  - Data formats of the various systems
  - And so much more

Apache Camel
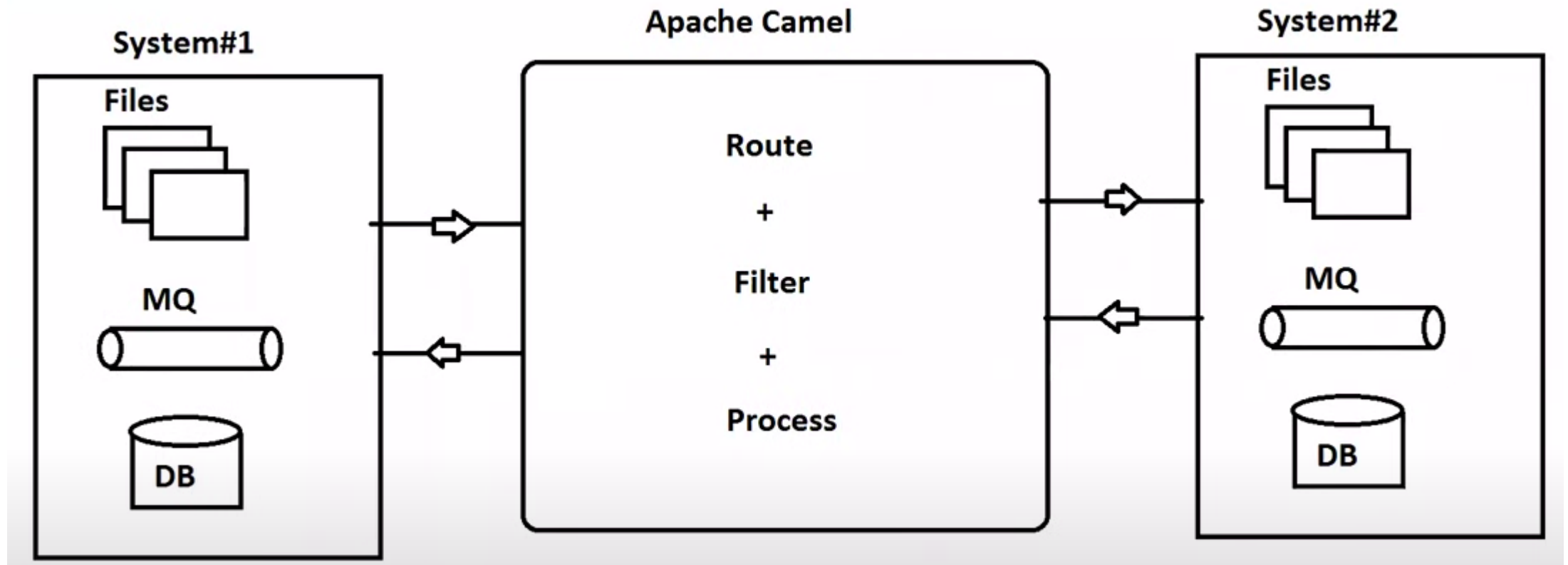
Apache Camel

- Java library that helps you write integrations and run them
- Define your integration flows
  - Where you want to pull data from
  - What you can or cannot do with the data
  - Where the data needs to go
  - And a lot more…
- Comes with built-in set of patterns you can use in integration flows
  - Splitter pattern – split the message based on how you want
  - Content based routing pattern – route messages based on content
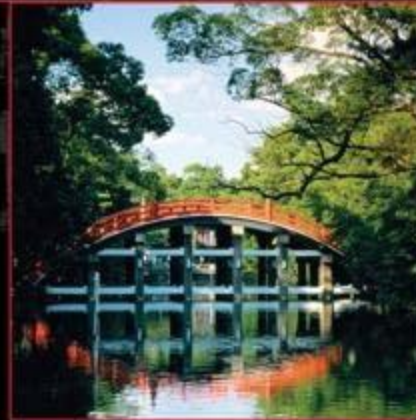 - And a lot more…

Enterprise Integration Patterns

Designing, Building, and Deploying Messaging Solutions

Gregor Hohpe
Bobby Woolf

With Contributions by
Kyle Brown
Conrad F. D'Cruz
Martin Fowler
Sean Neville
Michael J. Rettig
Jonathan Simon

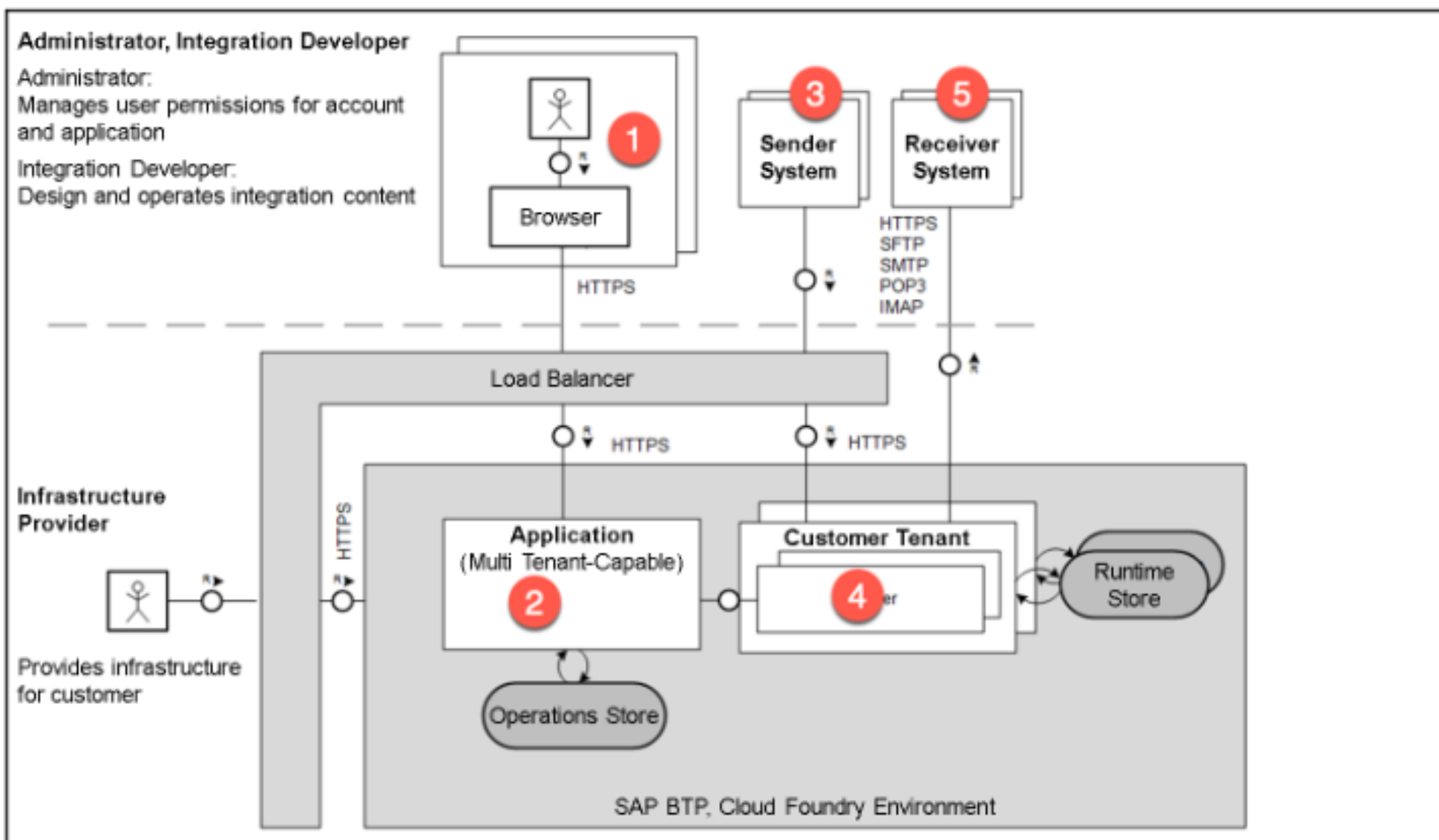*Forewords by John Crupi and Martin Fowler*

SAP Cloud Integration

- Adds enterprise features to Apache Camel

- Engineered for cloud
  - Multitenancy
  - Rolling software updates
  - Horizontal scalability

- Strong focus on security including data isolation

- Used by SAP SaaS solutions
  - SAP S/4HANA
  - SAP SuccessFactors and so on…

The Entire Implementation

Demo: Building sample integration

- Building integration from scratch
- Predefined integration content

Business Scenario

- Company A sells goods to customers
- Some products cannot be delivered on time
- Inform customers who ordered these products about delay

Task flow

Send a list of Product IDs (SOAP)	Is Product ID in DB ? Do this for all products	Get all Sales Orders for Product ID Do this for all products in DB	Get Sales Header for Sales Order Do this for all Sales Orders	Select Customer ID and write to Data Store Do this for all Sales Headers

| Laptop iPhone TV	Laptop iPhone TV	Laptop (Sales orders) • 501 • 502 TV (Sales orders) • 601 • 602 • 603	501 => Header 1 502 => Header 2 601 => Header 3 602 => Header 4 603 => Header 5	Header 1 => Customer 1 Header 2 => Customer 2 Header 3 => Customer 3 Header 4 => Customer 4 Header 5 => Customer 5 Write to Data Store

Demo of working solution…

- Demo of the working solution

- Explore Cloud Integration

  - Discover area
  - Design area
  - Monitor area
  - Settings area

Basic concepts of Cloud Integration flow…

Message

Fundamental entity **containing the data** being carried and routed in Camel

- Messages have a body (a payload), headers, and optional attachments
- Messages are uniquely identified with an identifier of type `java.lang.String`
- *Headers*
  - Headers are values associated with the message
    - ⇨ Sender identifier, hints about content encoding, authentication information,…
  - Headers are name-value-pairs
    - ⇨ Name is a unique, case-insensitive string
    - ⇨ Value is of type `java.lang.Object`
- *Attachments*
  - Optional – typically used for Web service and e-mail components
- *Body*
 - Type: `java.lang.Object` → any kind of content is allowed
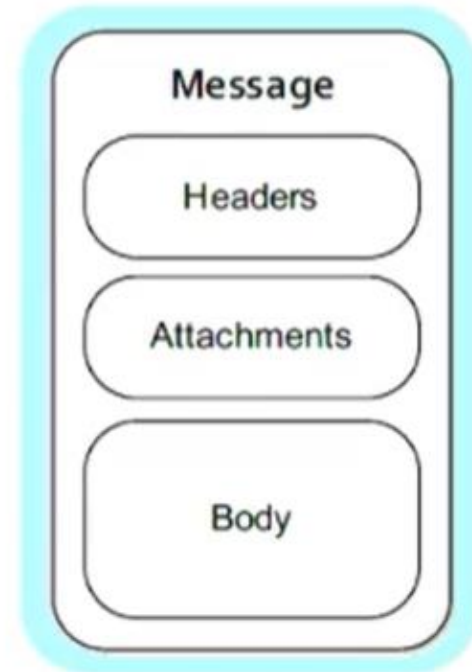
Basic concepts of Cloud Integration flow...

Exchange

The **message's container** during routing

- Provides support for various interaction types between systems, known as Message Exchange Patterns (MEP)
  - `InOnly`: a one-way message (e.g. JMS messaging)
  - `InOut`: a request-response message (e.g. HTTP-based transports)
- *Exchange ID*: a unique ID that identifies the exchange
- *MEP*
  - `InOnly`: exchange contains an "in message" **only**
  - `InOut`: exchange contains an "in message" **and** an "out message" containing the reply message for the caller
- *Exception*: If an error occurs during runtime, the Exception field will be filled
- *Properties*: Similar to message headers, but they last for the duration of the entire exchange; they contain global-level information; you can store and retrieve properties at any point during the lifetime of an exchange
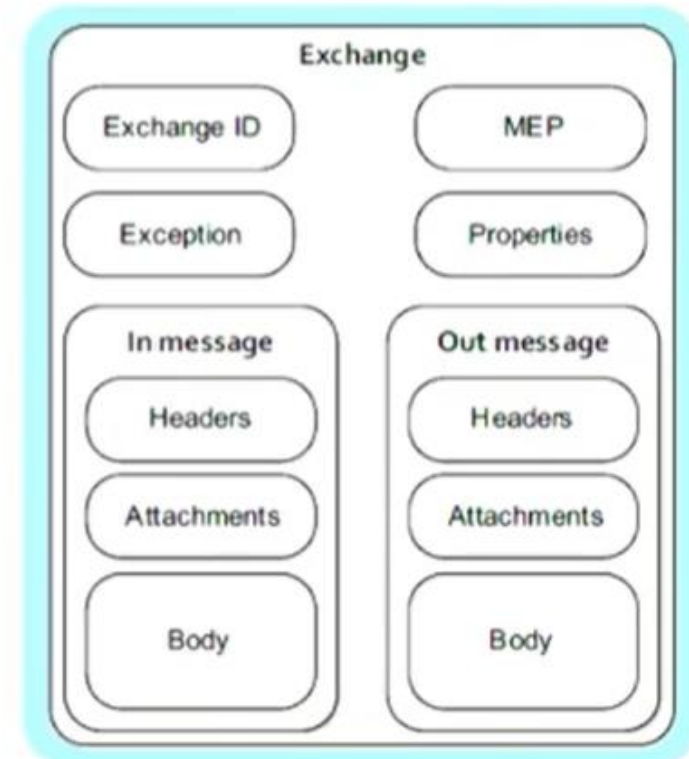
MESSAGE EXCHANGE PATTERNS AND THE EXCHANGE OBJECT

The Camel API is influenced by APIs such as JBI specification, CXF which defines a concept called Message Exchange Patterns (MEP for short).

The MEP defines the messaging style used such as one-way (InOnly) or request-reply (InOut), which means you have IN and optionally OUT messages. This closely maps to other APIs such as WS, WSDL, REST, JBI and the likes.

The Exchange API provides two methods to get a message, either `getIn` or `getOut`. Obviously the `getIn` gets the IN message, and the `getOut` gets the OUT message.

FLOW OF AN EXCHANGE THROUGH A ROUTE
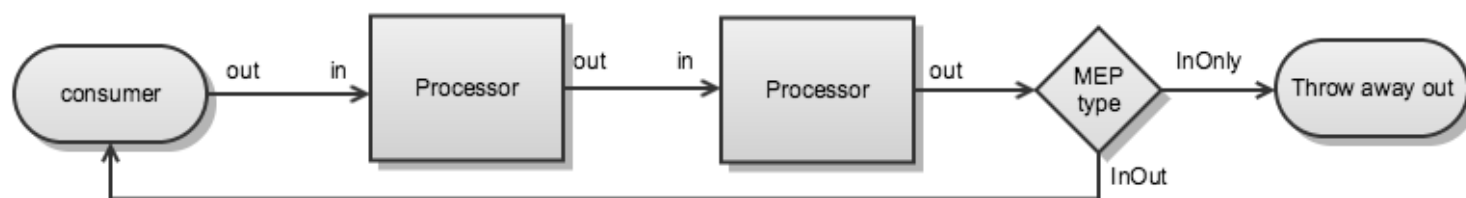
- The out message from each step is used as the in message for the next step
- if there is no out message then the in message is used instead
- For the InOut MEP the out from the last step in the route is returned to the producer. In case of InOnly the last out is thrown away

> **TIP**
>
> **Beware of `getOut` to check if there is an out message**
>
> `exchange.getOut` creates an out message if there is none. So if you want to check if there is an out message then you should use `exchange.hasOut` instead.

Manipulating Exchange Parameters

- Exchange params (including payload): set by incoming messages
- But these params can also be manually manipulated
  - Content Modifier component
  - Groovy SDK
  - JavaScript SDK
  - PDF in Message Mapping
  - XSLT Mapping
  - And more…

Simple Expression Language

- Used to parameterize Exchange Parameters
- General scheme is ${} placeholder containing built-in variable or Exchange parameter
- For example
  - ${in.body}
  - ${property.someproperty}
  - ${header.someheader}
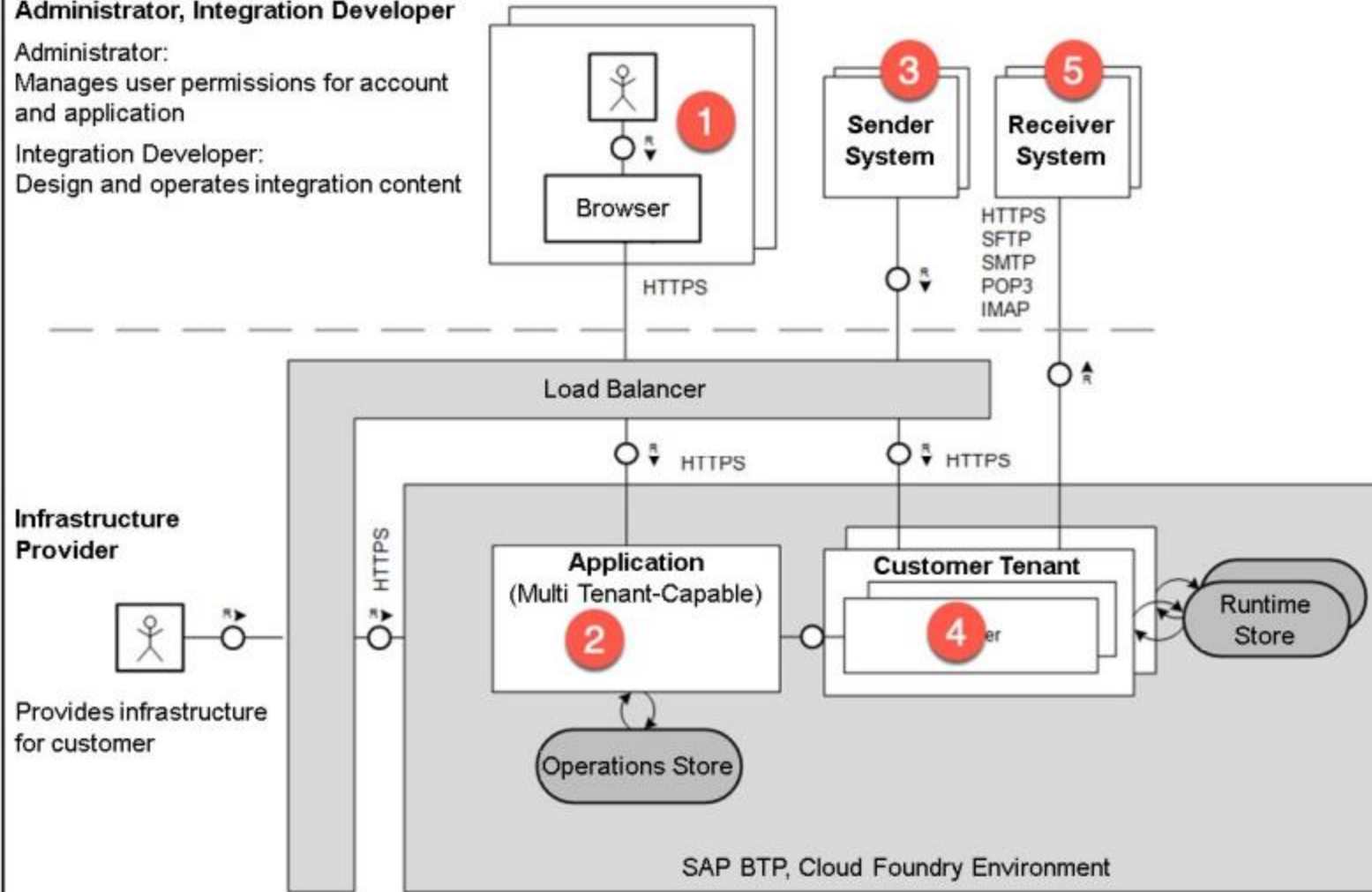
Figure 78: The Entire Implementation

Business Scenario - Task flow

Send a list of Product IDs **(SOAP)**

→

Is Product ID in DB ? **Do this for all products**

→

Get all Sales Orders for Product ID **Do this for all products in DB**

→

Get Sales Header for Sales Order **Do this for all Sales Orders**

→

Select Customer ID and write to Data Store **Do this for all Sales Headers**

Business Scenario – Steps involved

1. Create Integration Package and Integration Flow with Timer

2. Add Content Modifier (mock data – Product list)

3. Add General Splitter (split Products for iteration)

4. Add Content Modifier (add Product ID to Exchange Property)

5. Add Request Reply (is Product ID in database)

6. Add Router (If Product ID exists - CONTINUE, else END)