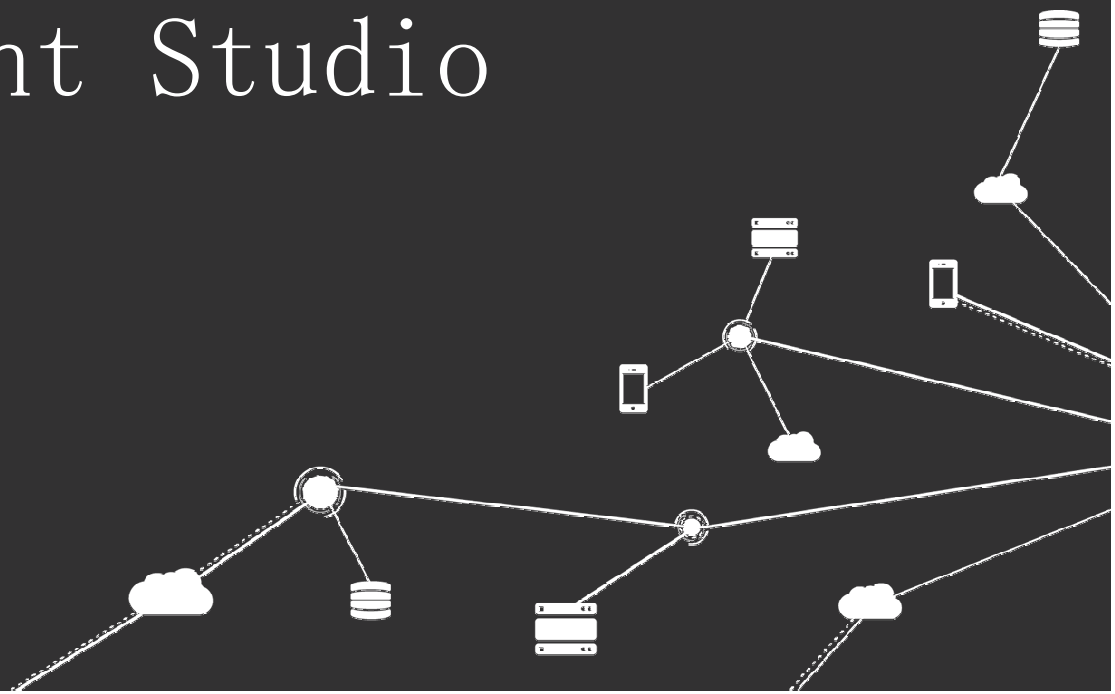
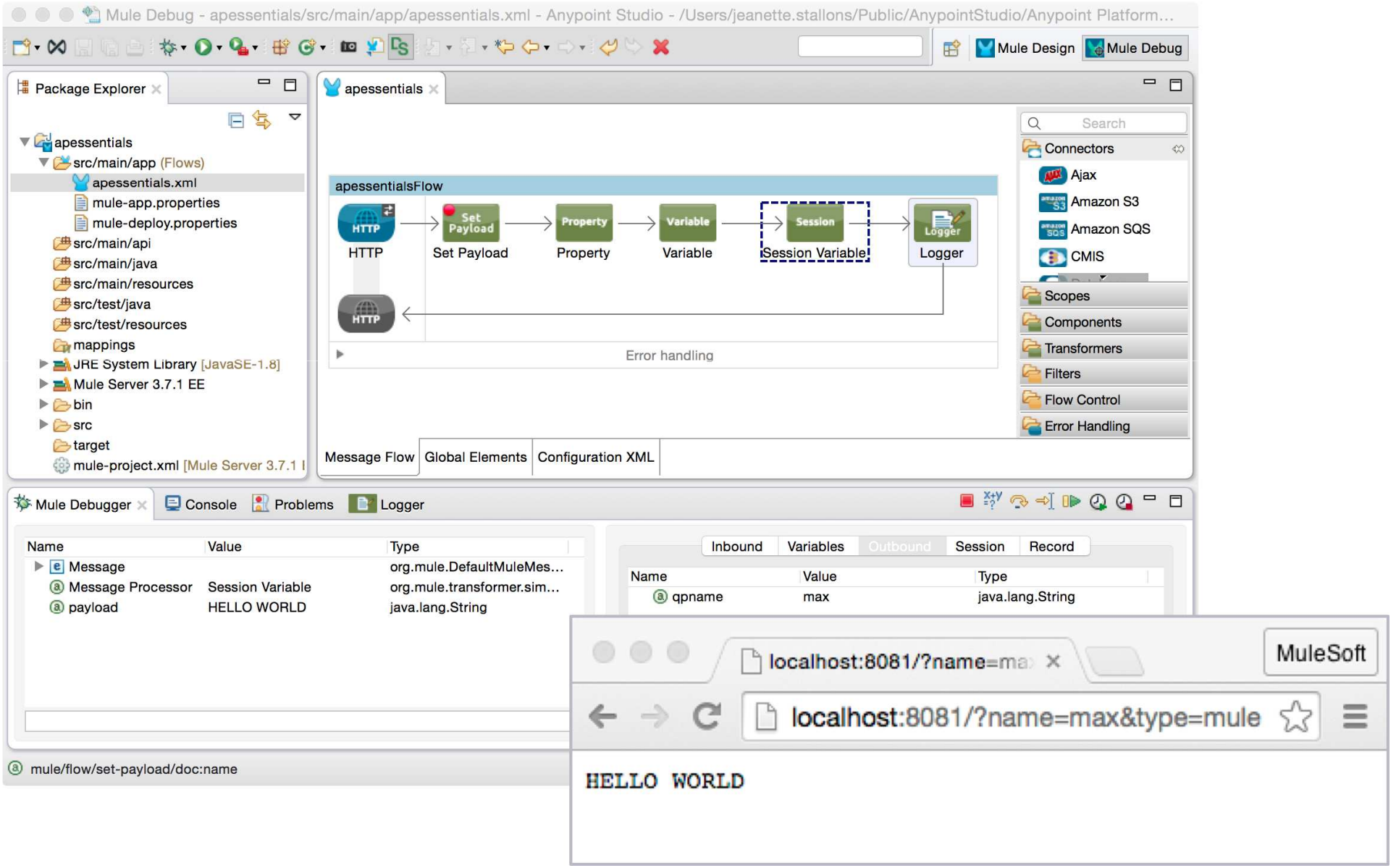
Module 2: Building Integration Applications with Anypoint Studio



Goal

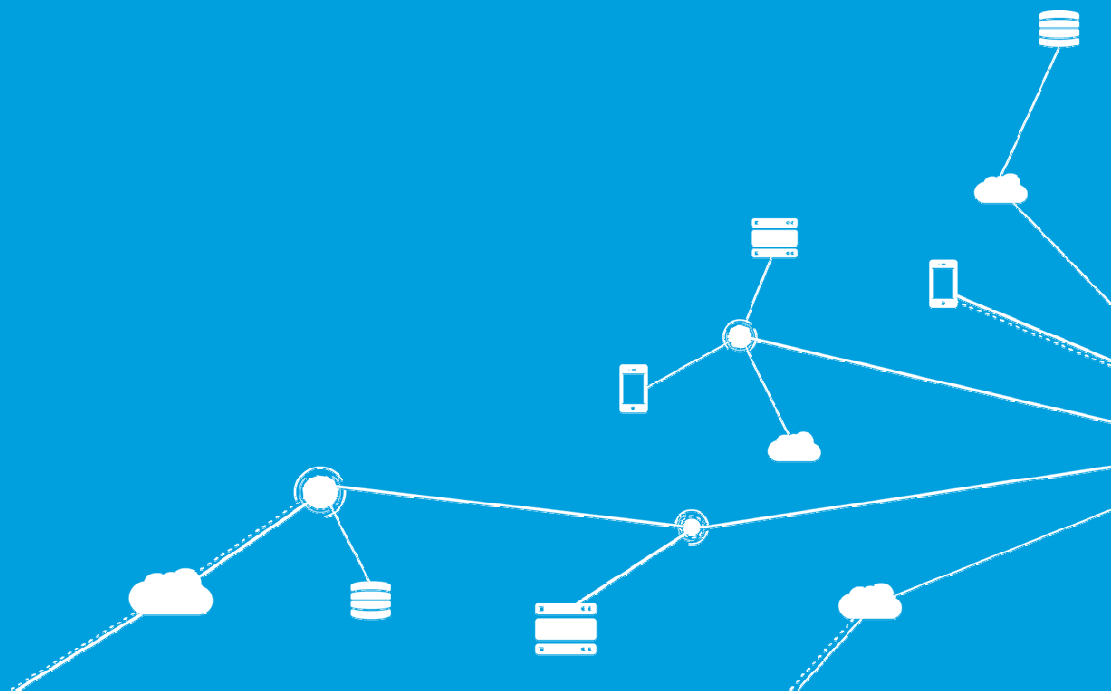

Mule Debug - apessentials/src/main/app/apessentials.xml - Anypoint Studio - /Users/jeanette.stallons/Public/AnypointStudio/Anypoint Platform...
Mule Design
Mule Debug
Package Explorer
apessentials
src/main/app (Flows)
apessentials.xml
mule-app.properties
mule-deploy.properties
src/main/api
src/main/java
src/main/resources
src/test/java
src/test/resources
mappings
JRE System Library [JavaSE-1.8]
Mule Server 3.7.1 EE
bin
src
target
mule-project.xml [Mule Server 3.7.1 I
apessentials
apessentialsFlow
HTTP
Set Payload
Property
Variable
Session Variable
Logger
HTTP
Error handling
Search
Connectors
Ajax
Amazon S3
Amazon SQS
CMIS
Scopes
Components
Transformers
Filters
Flow Control
Error Handling
Message Flow
Global Elements
Configuration XML
Mule Debugger
Console
Problems
Logger
Name
Value
Type
Message
org.mule.DefaultMuleMes...
Message Processor
Session Variable
org.mule.transformer.sim...
payload
HELLO WORLD
java.lang.String
Inbound
Variables
Outbound
Session
Record
Name
Value
Type
qpname
max
java.lang.String
mule/flow/set-payload/doc:name
localhost:8081/?name=ma
MuleSoft
localhost:8081/?name=max&type=mule
HELLO WORLD

Objectives

- In this module, you will learn:
  - About Mule applications, flows, messages, and message processors
  - To use Anypoint Studio to create flows graphically using connectors, transformers, components, scopes, and flow control elements
  - To build, run, test, and debug Mule applications
  - To read and write message properties
  - To write expressions with Mule Expression Language (MEL)
  - To create variables

MuleSoft

Introducing Mule applications



Mule applications

- Accept and process **messages** through a series of **message processors** plugged together in a **flow**
  - A message can be initiated by an events like
    - A consumer request from a mobile device
    - A change to data in a database
    - The creation of a new customer ID in a SaaS application

MuleSoft

Mule applications

- Are written in XML (primarily)
  - Under the hood, are Java applications using Spring
- Can be created and tested visually with Anypoint Studio
  - Available as a stand-alone or as an Eclipse plug-in
- Are deployed to a Mule Server Runtime
  - A standalone app to a Mule runtime(typically)
  - A WAR file with an embedded Mule instance to an app server

MuleSoft

Mule Server Runtime

- A JVM server that
  - Can handle many concurrent requests for different Java (Mule) applications in a single JVM
  - Decouples point-to-point integrations by having all (non-Mule) applications talk to the bus (to a Mule app) instead of directly to each other
  - Decouples applications by using
    - Protocol adaption
      - Communication can be over different protocols, like HTTP or FTP
    - A canonical data format
      - A common format all messages are transformed to

Mule runtime editions

- Enterprise and community editions
  - Mule ESB Server runtime EE
  - Mule ESB Server runtime CE
- CE is open-source
- EE is a hardened code line with support and additional capabilities
- By default, Anypoint Studio uses EE
  - You can install other versions and select which one to use

- http://www.mulesoft.com/platform/soa/mule-esb-enterprise

MuleSoft

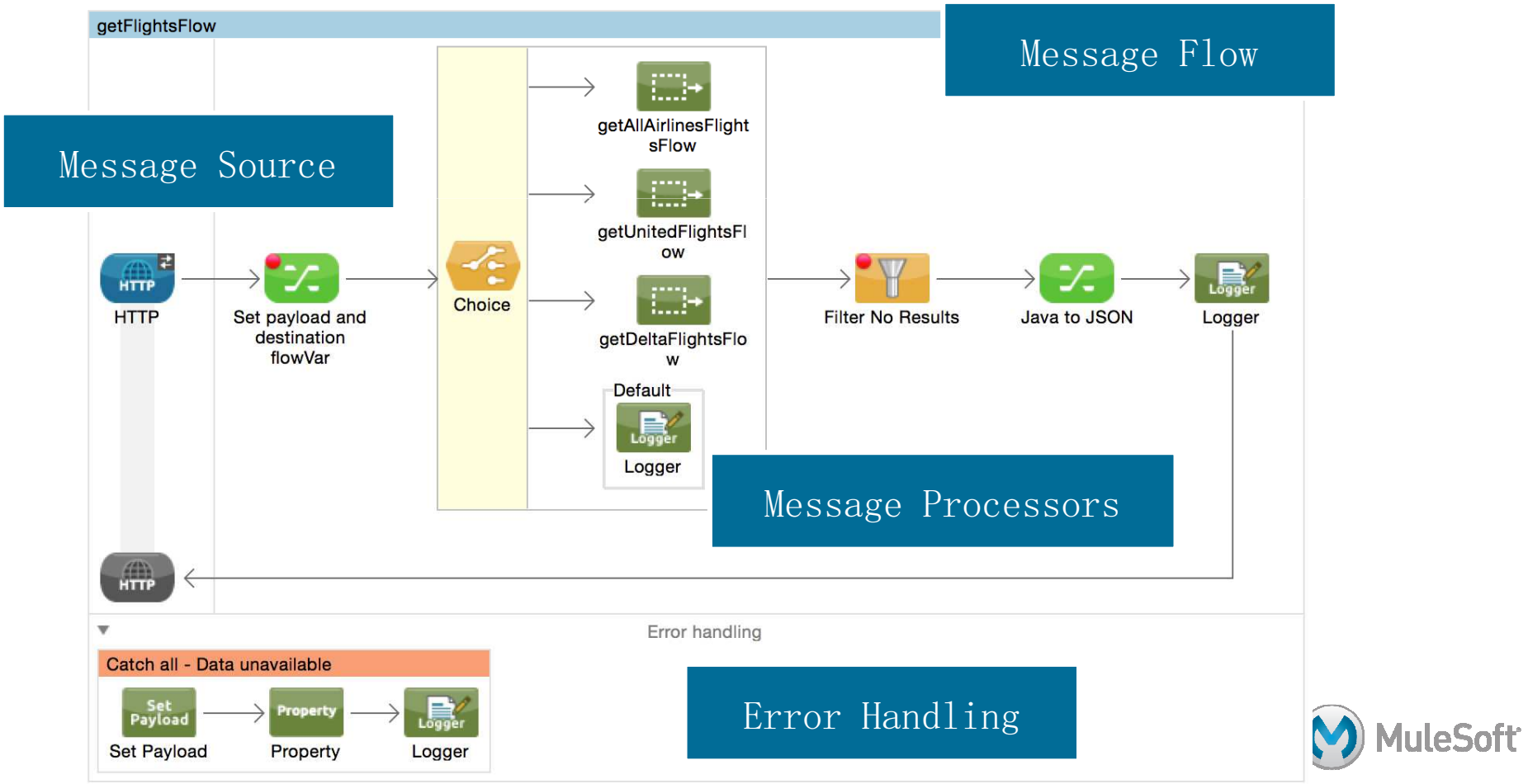
Mule runtime EE

- 24/7 global support
- Additional connectors
- ✧ Visual debugging
- ✧ DataMapper and DataSense
- ✧ Batch module
- Caching and transaction support
- Performance monitoring
- Security module
- Templates
- ✧ Deployment and performance management

MuleSoft

Mule applications and flows

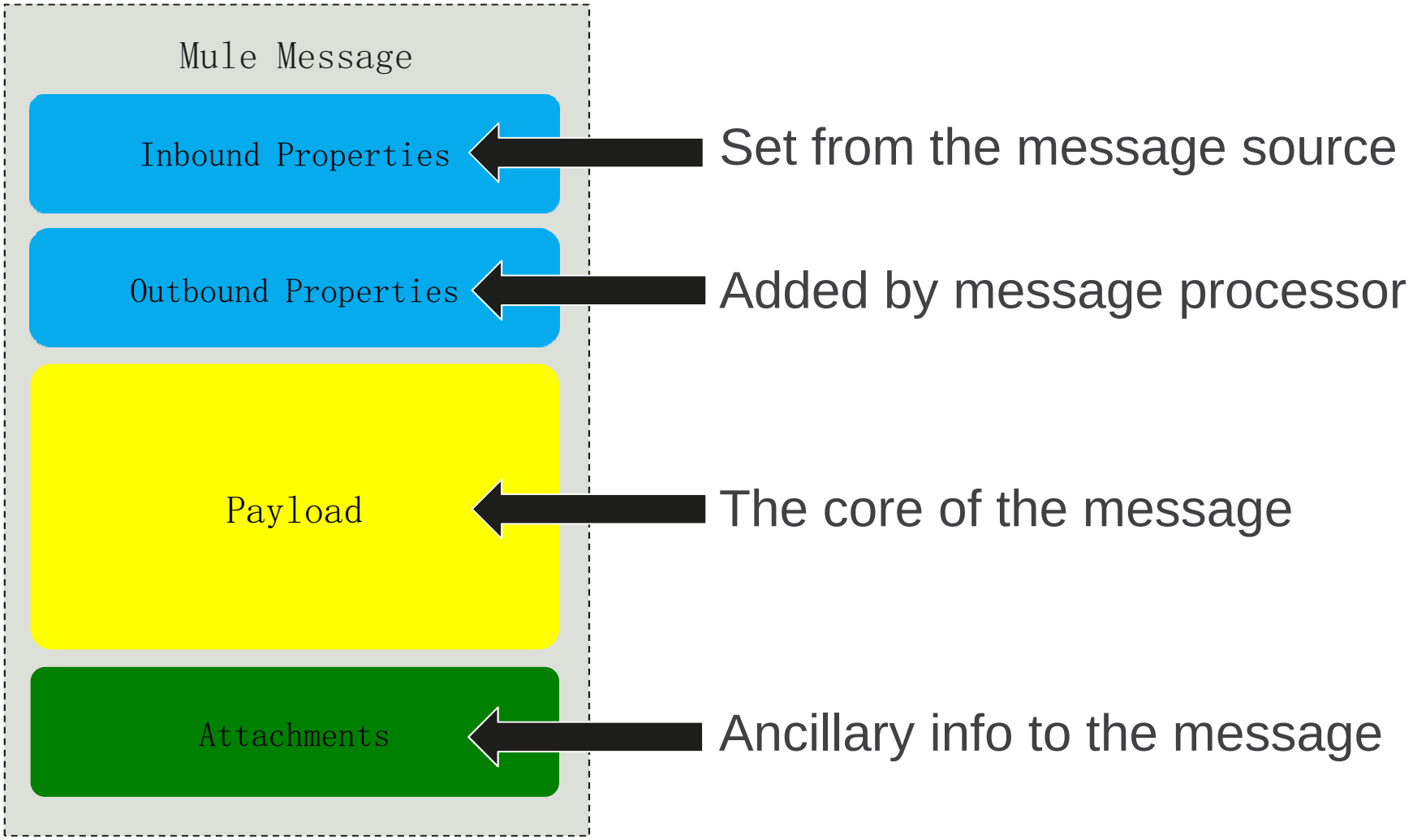
- Mule applications accept and process **messages** through a series of **message processors** plugged together in a **flow**



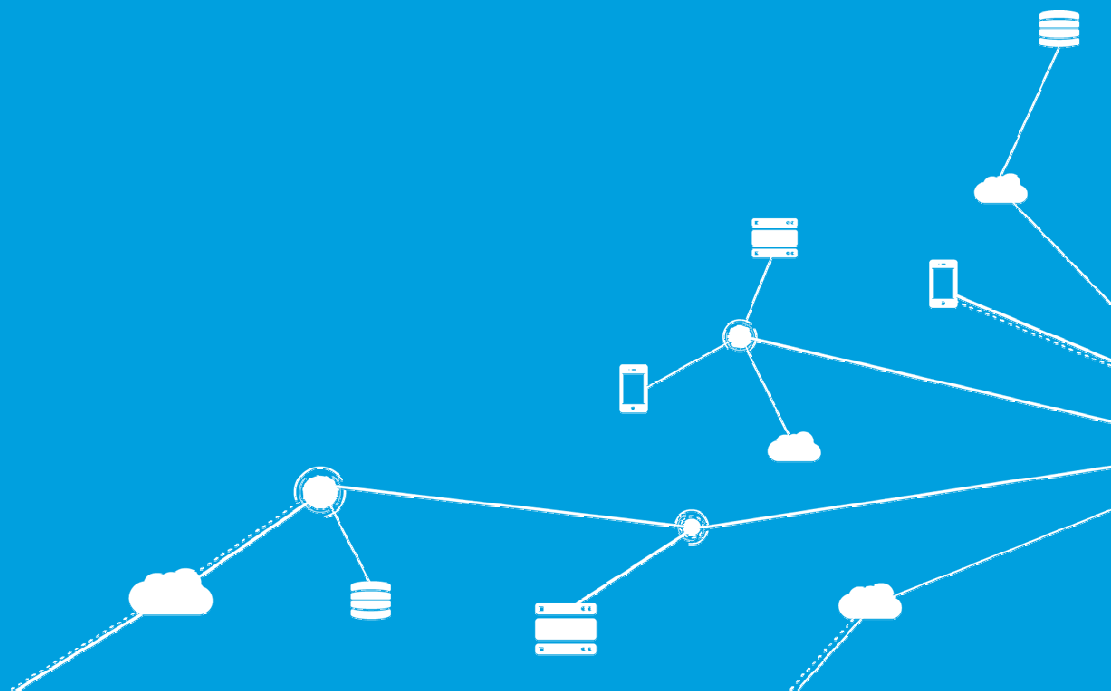
Mule flows

- A typical flow has
  - A message source
    - Accepts a message from an external source triggering the execution of the flow
  - Message processors
    - Transform, filter, enrich, and process the message
- An application can consist of
  - A single flow
  - Multiple flows
  - Multiple flows connected together

Mule messages



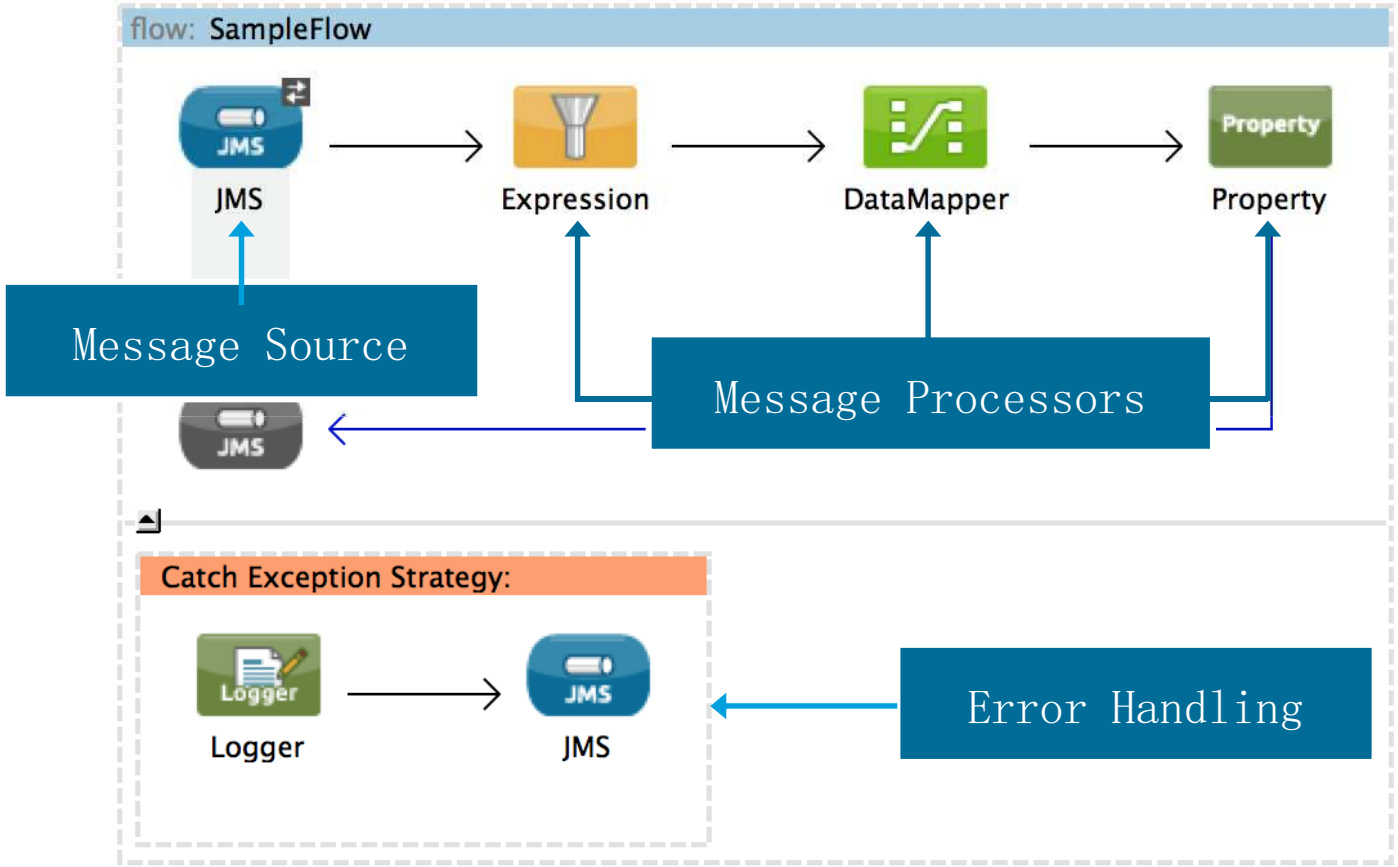
Creating Mule applications with Anypoint Studio



Creating Mule applications with Anypoint Studio

- Anypoint Studio is an Eclipse-based integration development environment
  - Two-way editing between graphical and XML views
  - Visual debugging (EE)
  - Pre-built tooling to connect to
    - Many popular services (Salesforce, Workday, Facebook, more!)
    - Many standard protocols (HTTP, HTTPS, FTP, SMTP, more!)
    - Any SOAP or RESTful API
  - A data transformation framework and language (EE)
  - One-click deployment of applications
  - Templates for common integration patterns (EE)
  - Integration with Maven for continuous build processes

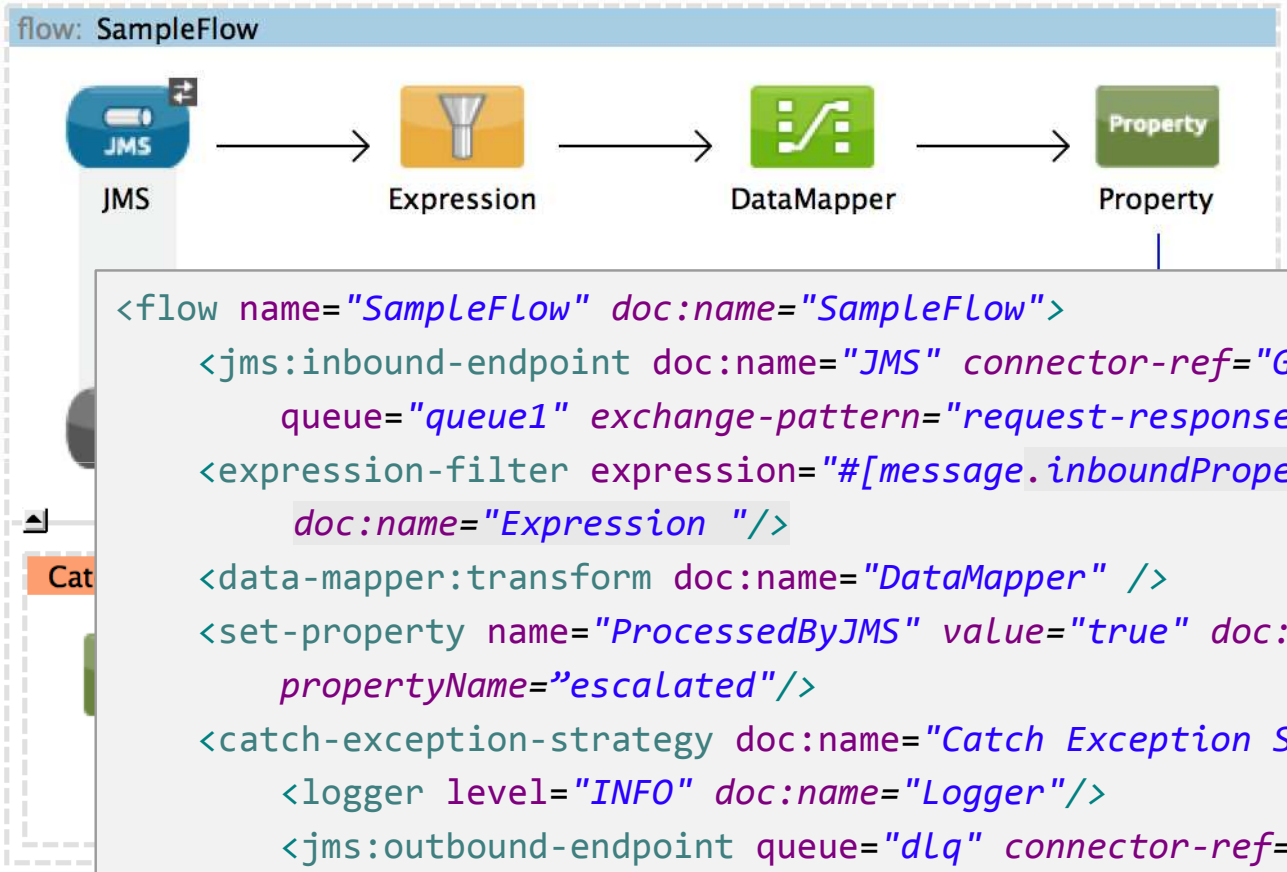
Anatomy of a flow: Visual



* Each flow operates on its own thread pool

MuleSoft

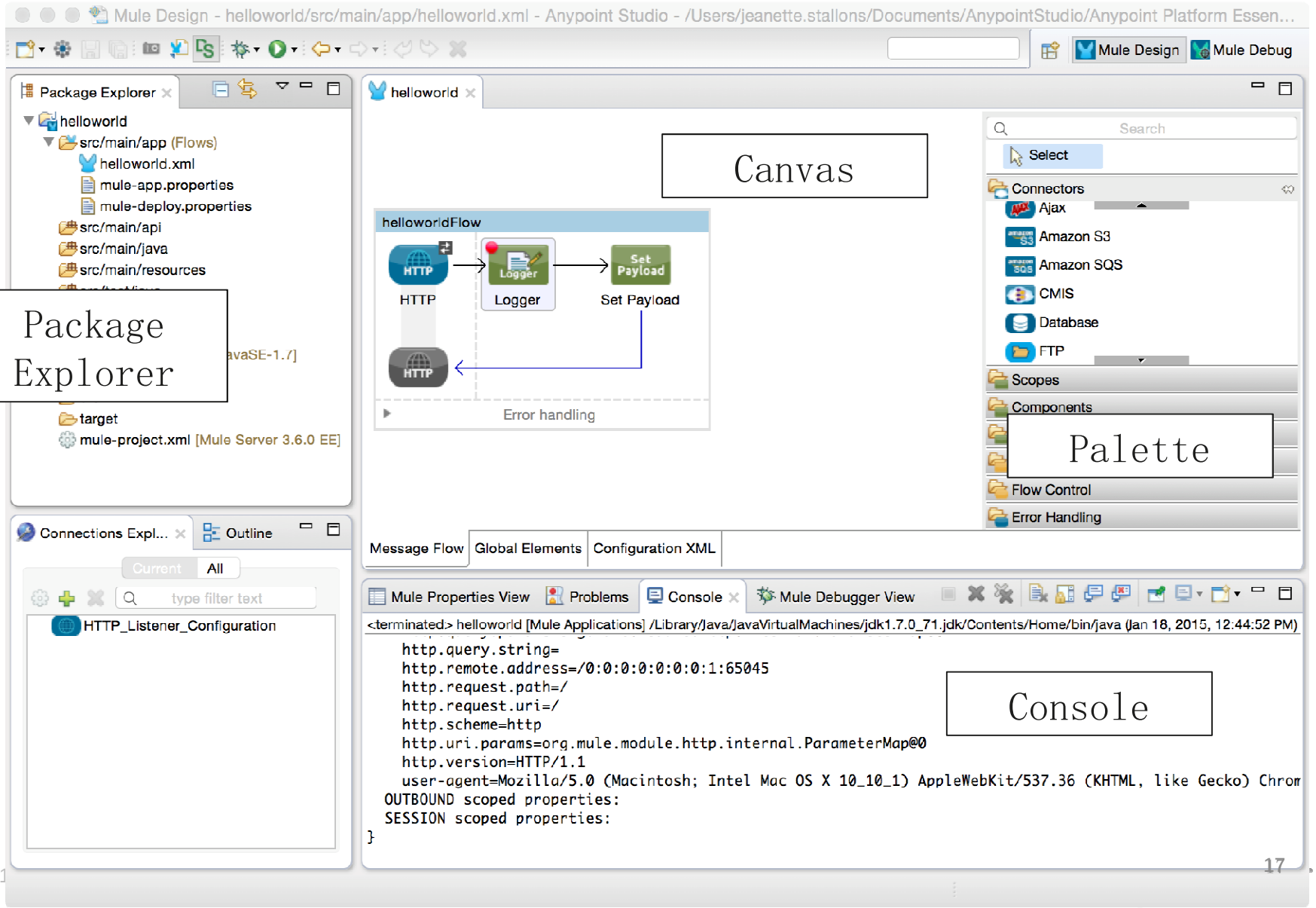
Anatomy of a flow: XML




```
<flow name="SampleFlow" doc:name="SampleFlow">
    <jms:inbound-endpoint doc:name="JMS" connector-ref="GlobalJMS”
        queue="queue1" exchange-pattern="request-response" />
    <expression-filter expression="#[message.inboundProperties]”
         doc:name="Expression "/>
    <data-mapper:transform doc:name="DataMapper" />
    <set-property name="ProcessedByJMS" value="true" doc:name="Property”
        propertyName=”escalated"/>
    <catch-exception-strategy doc:name="Catch Exception Strategy">
        <logger level="INFO" doc:name="Logger"/>
        <jms:outbound-endpoint queue="dlq" connector-ref="glConn”
           doc:name="JMS"/>
    </catch-exception-strategy>
</flow>
```

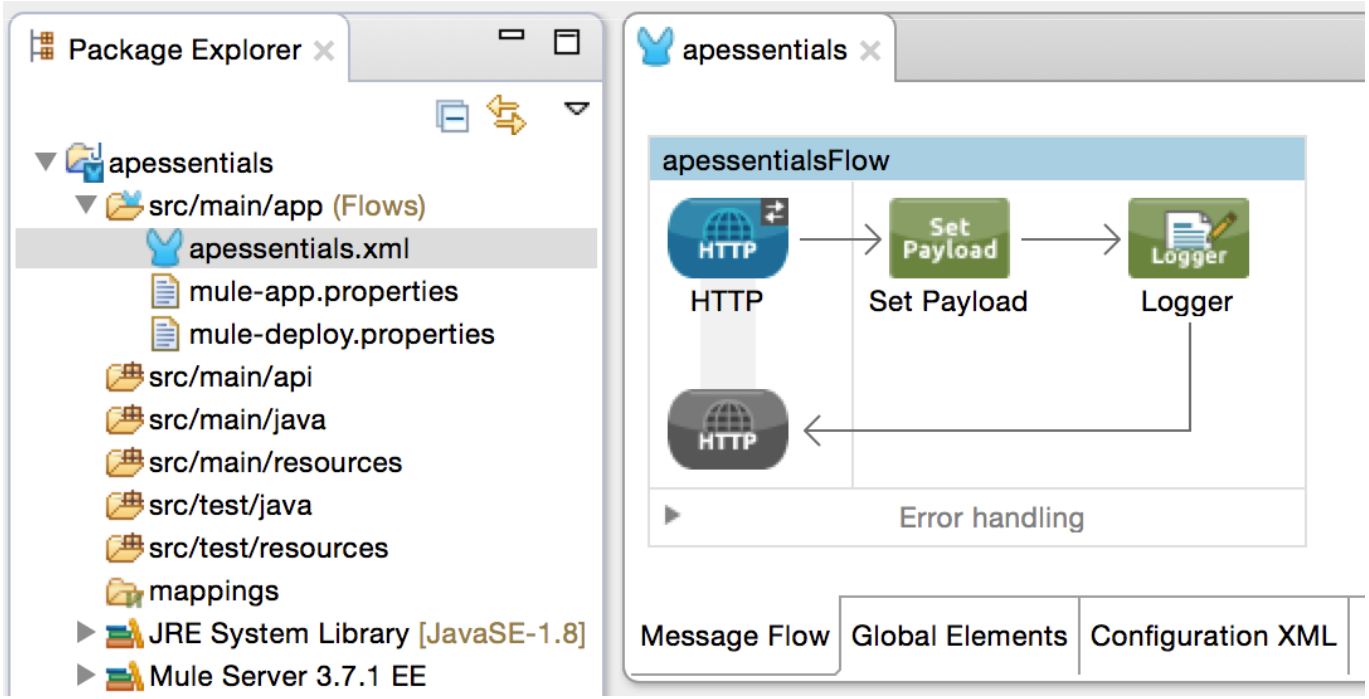

MuleSoft

Anypoint Studio anatomy



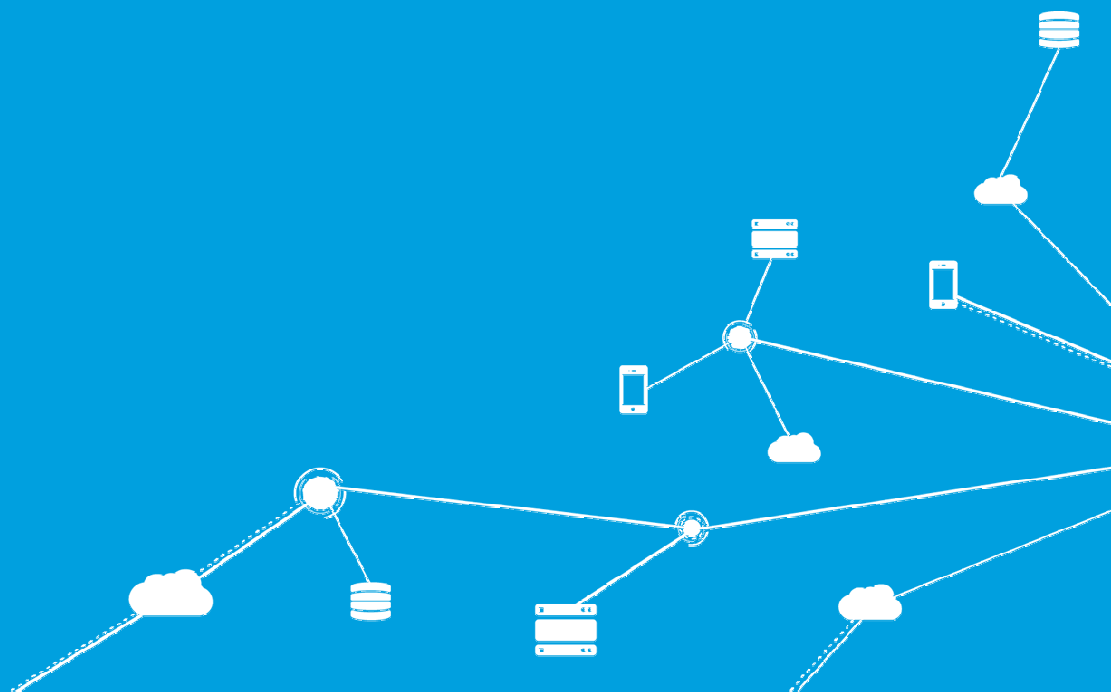
Walkthrough 2-1: Create your first Mule application

- Create a new Mule project with Anypoint Studio
- Add a connector to receive requests at an endpoint
- Display a message in the Anypoint Studio console
- Set the message payload



MuleSoft

Understanding Mule application building blocks



Message sources

- Mule applications accept and process **messages** through a series of **message processors** plugged together in a **flow**

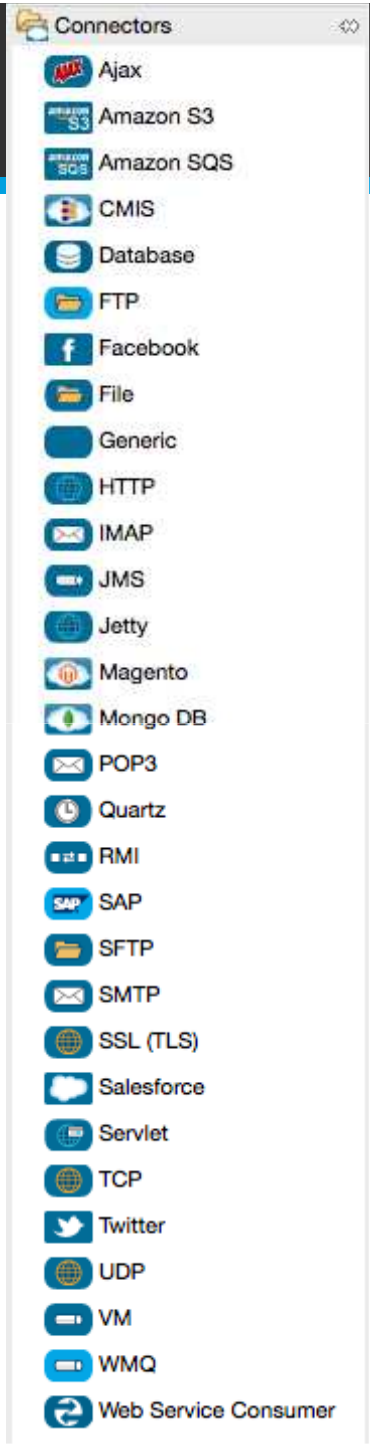
- Message sources are usually Anypoint Connectors
- Connectors provide connectivity to external resources
  - Such as databases, protocols, or APIs
  - Standard protocols like HTTP, FTP, SMTP, AMQP
  - Third-party APIs like Salesforce.com, Twitter, or MongoDB
- The first building block of most flows is a receiver that receives new messages and places them in the queue for processing

MuleSoft

Anypoint Connectors

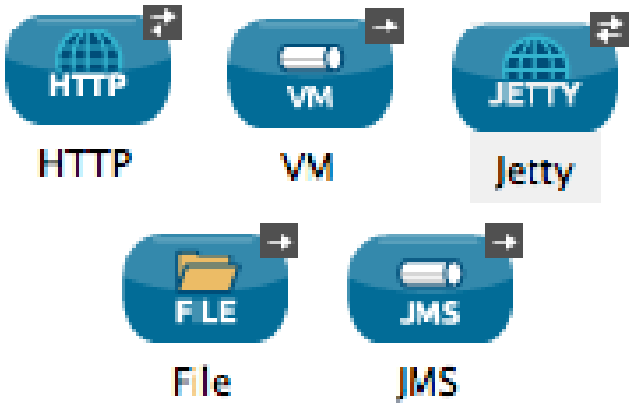
- Anypoint Platform has over 120 pre-built connectors
  - 30 bundled with Anypoint Studio
  - Full library at https://www.mulesoft.com/exchange

- There are 2 main types
  - Endpoint-based connectors
  - Operation-based connectors



Endpoint-based connectors

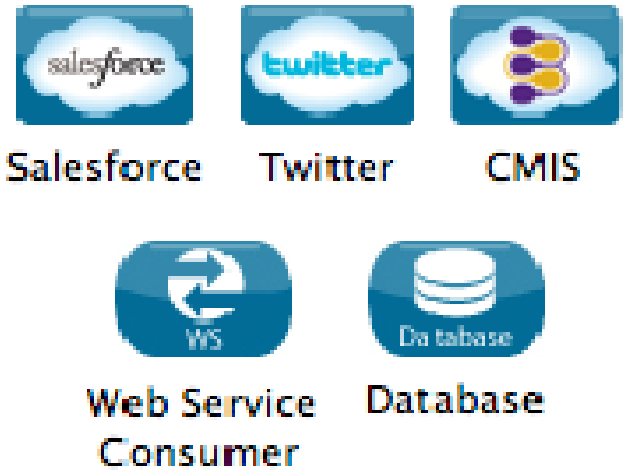
- Are either inbound or outbound endpoints in a flow
- Inbound endpoints serve as a message source for a flow
- Outbound endpoints send information to external systems
  - Can occur mid-flow or at the end of flows (or at the beginning in a poll)



MuleSoft

Operation-based connectors

- Require the specification of an operation for that connector to perform
- Includes most connectors not based on a standard communication protocol



MuleSoft

Terminology: Connector vs endpoint vs transport

- A **connector** is a Mule-specific connection to an external resource of any kind
- An **endpoint** is a flow-level element that is configured to receive and/or send messages from and/or to external resources
- **Transports** provide connectivity to a data source or message channel by implementing one of a variety of standard connection protocols that are bundled with Studio

Connectors

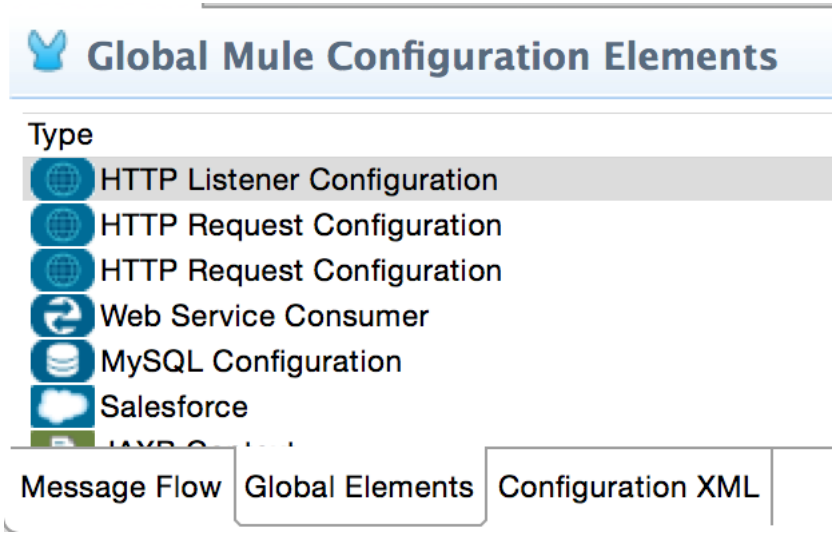
- A **connector** is a Mule-specific connection to an external resource of any kind
- Resources may be
  - A generic protocol like HTTP or FTP
  - A Java-based database
  - A specific third-party API like Salesforce or Workday
- Connectors may be operation-based or endpoint-based
  - This distinction refers to a difference in configuration details, not in functionality
  - Endpoint-based connectors are often constructed around a transport

Endpoints

- An **endpoint** is a flow-level element that is configured to receive and/or send messages from and/or to external resources
- Any connector (whether endpoint-based or operation-based) can function as an endpoint
- But some connectors can only act as inbound endpoints, others can only act as outbound endpoints

Connectors and endpoints and global elements

- When you drag a connector from the Studio palette, an endpoint is created
- For most endpoints, a lot of the configuration is encapsulated in a separate global element
  - A reusable object that can be used by many endpoints
  - Defines a connection to a network resource
  - These are referred to as
    - Connectors
    - Global Mule configuration elements



HTTP connector

- Can send and receive HTTP and HTTPS requests over a selected host, port, and address
- Can be either a listener or a requester depending upon where you add it in a flow
- HTTP Listener connector (inbound)
  - Listens for HTTP requests that arrive at a certain address and provides an HTTP response to these
  - By default, host is set to 0.0.0.0
    - A shortcut to simultaneously listen on all active IP addresses (including localhost)
- HTTP Request connector (outbound)
  - Sends HTTP requests to a certain address and receives the returned response

MuleSoft

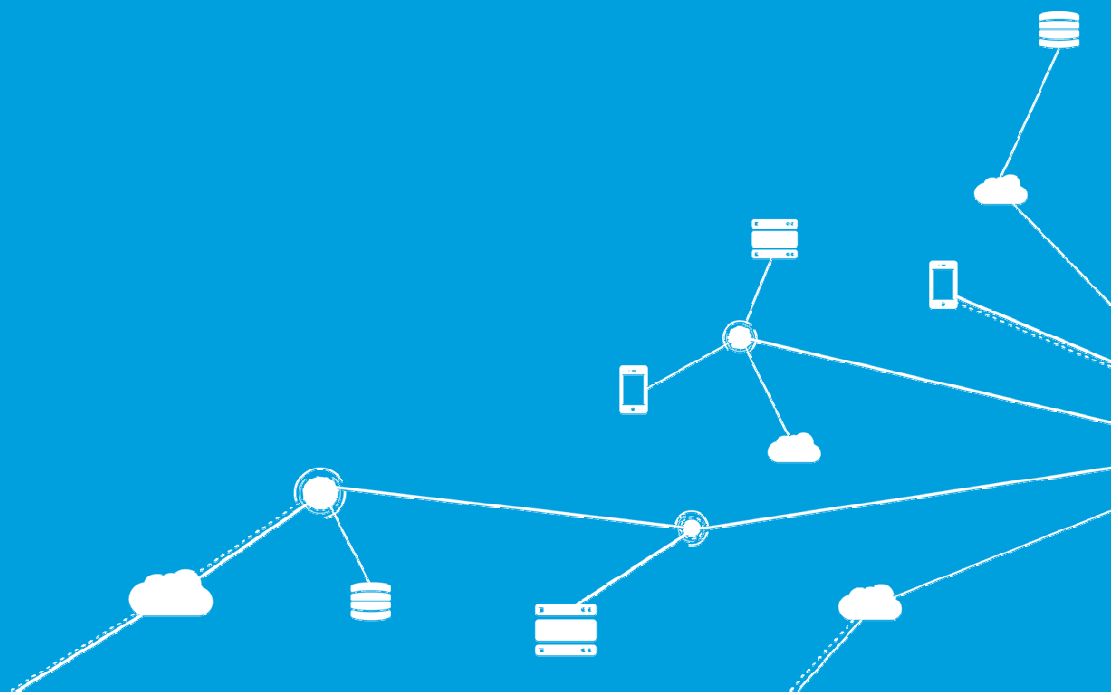
Message processors

- Components
  - Execute specific logic upon a message, including custom-logic in Java, JavaScript, Groovy, Python or Ruby
  - **Logger** component
- Transformers
  - Convert data types and formats so as to "translate" messages between applications or systems
  - **Set Payload** transformer

Message processors

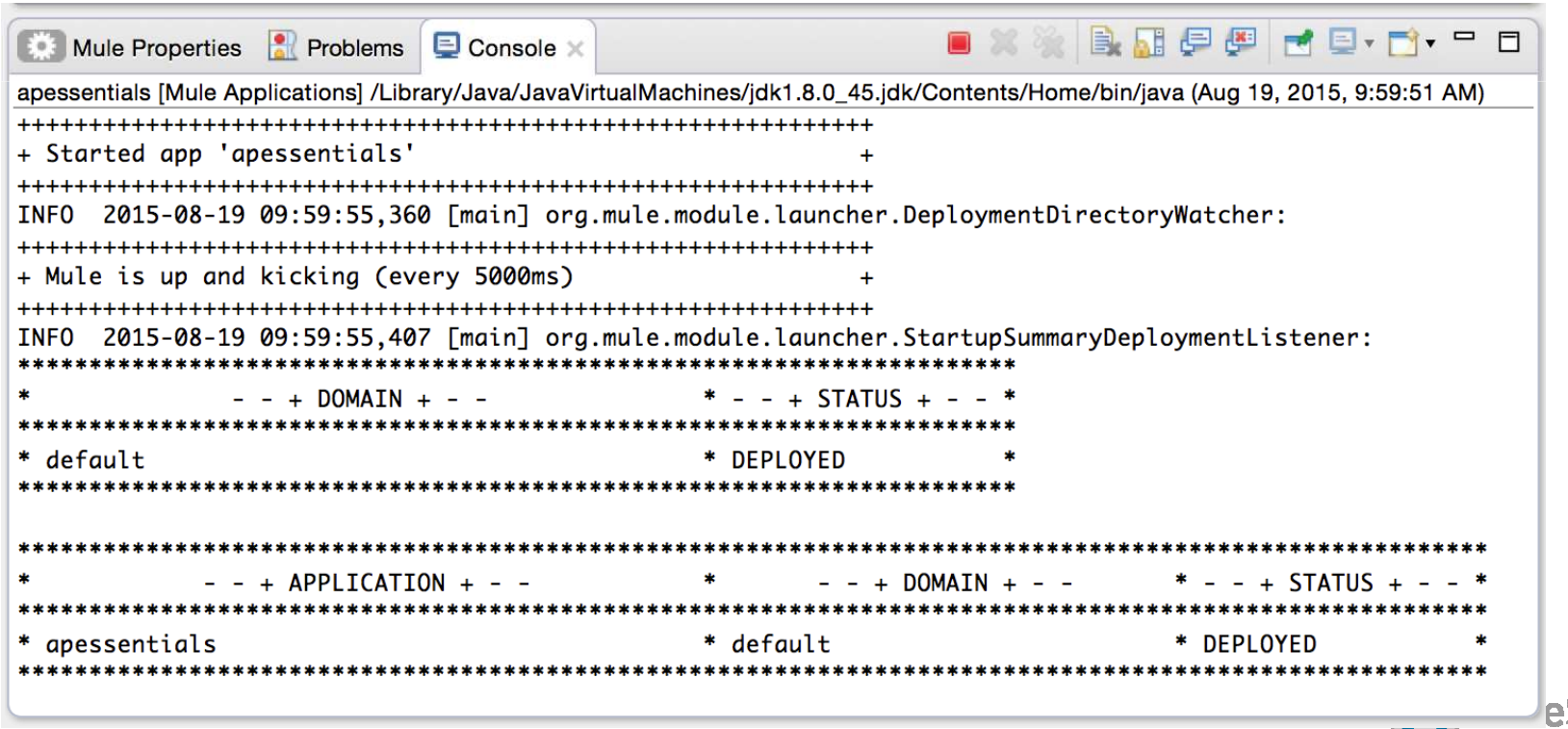
- Scopes
  - Wrap snippets of code to define fine-grained behavior within a flow
- Filters
  - Limit processing of messages based on set criteria
- Flow control
  - Direct messages through different pathways in an application depending upon content or other criteria
- Error handling
  - Handle any errors that occur during message processing

Running, testing, and debugging Mule applications



Running applications

- Anypoint Studio comes complete with an embedded Mule runtime to test applications without leaving it
- The console outputs application logs and information

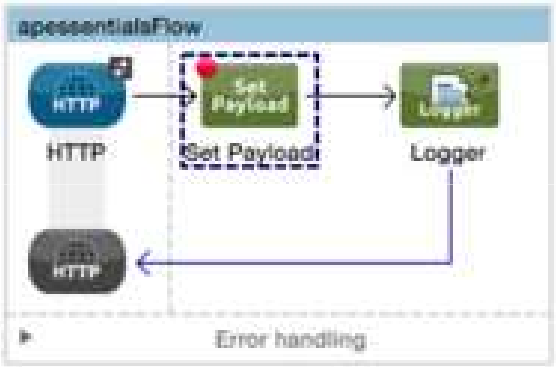


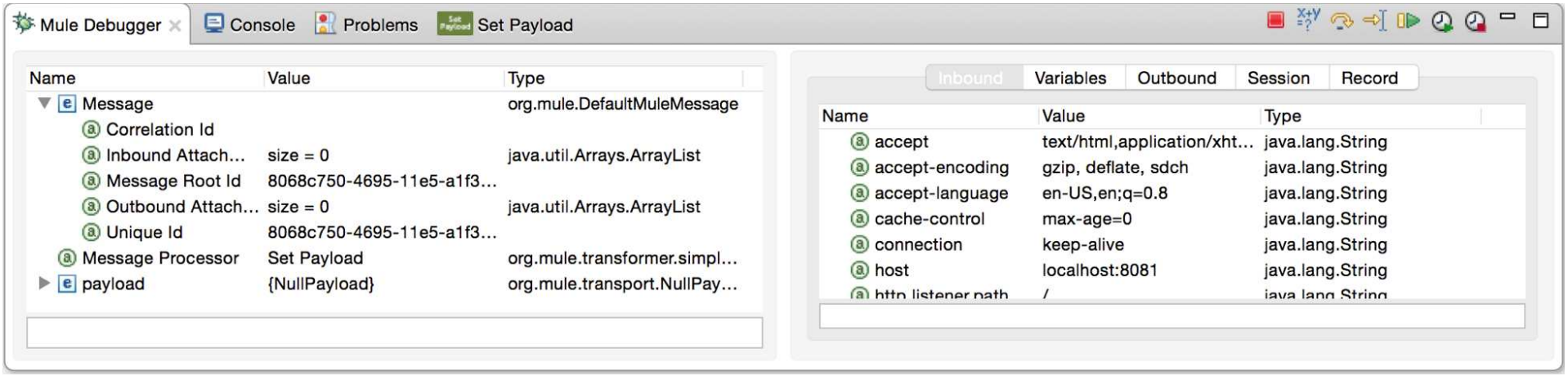
Testing applications

- There are several options for making requests to an endpoint
  - A browser
  - A cURL command-line utility
  - A browser extension like Postman (for Google Chrome)

Debugging applications with the Mule Debugger

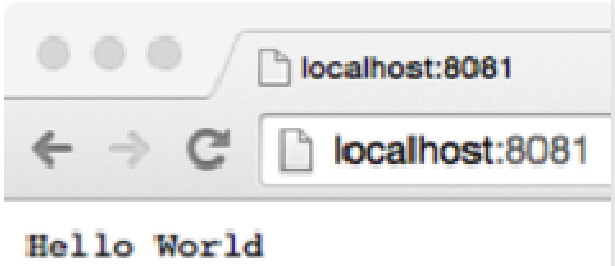
- Can add breakpoints to processors and step through the application
  - Watch message and variable values
  - Watch and evaluate expressions
- Debugger listens for incoming TCP connections on localhost port 6666

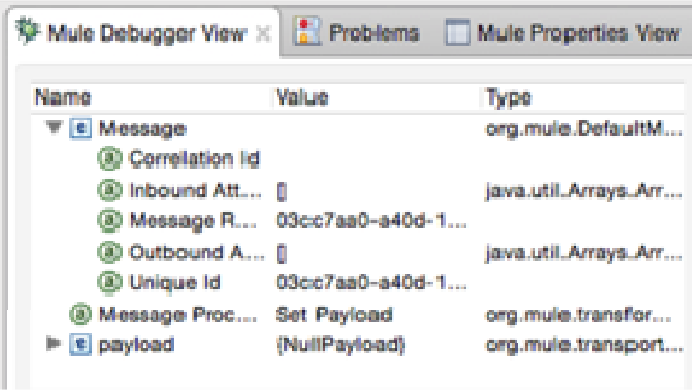


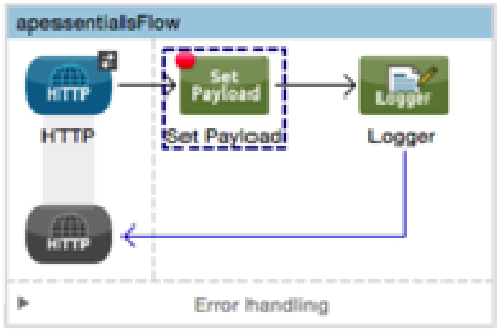


Walkthrough 2-2: Run, test, and debug a Mule application

- Run a Mule application using the embedded Mule runtime
- Make an HTTP request to the endpoint via a web browser or a tool like cURL or Postman
- Receive a response with the text Hello World
- Redeploy an application
- Use the Mule Debugger to debug an application

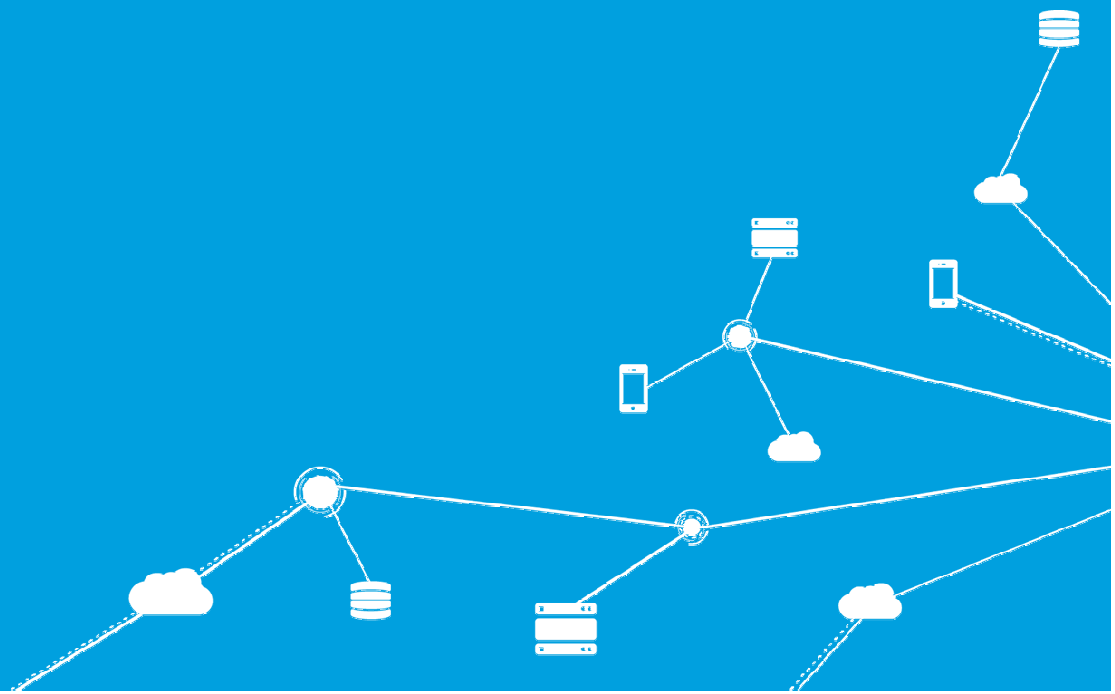




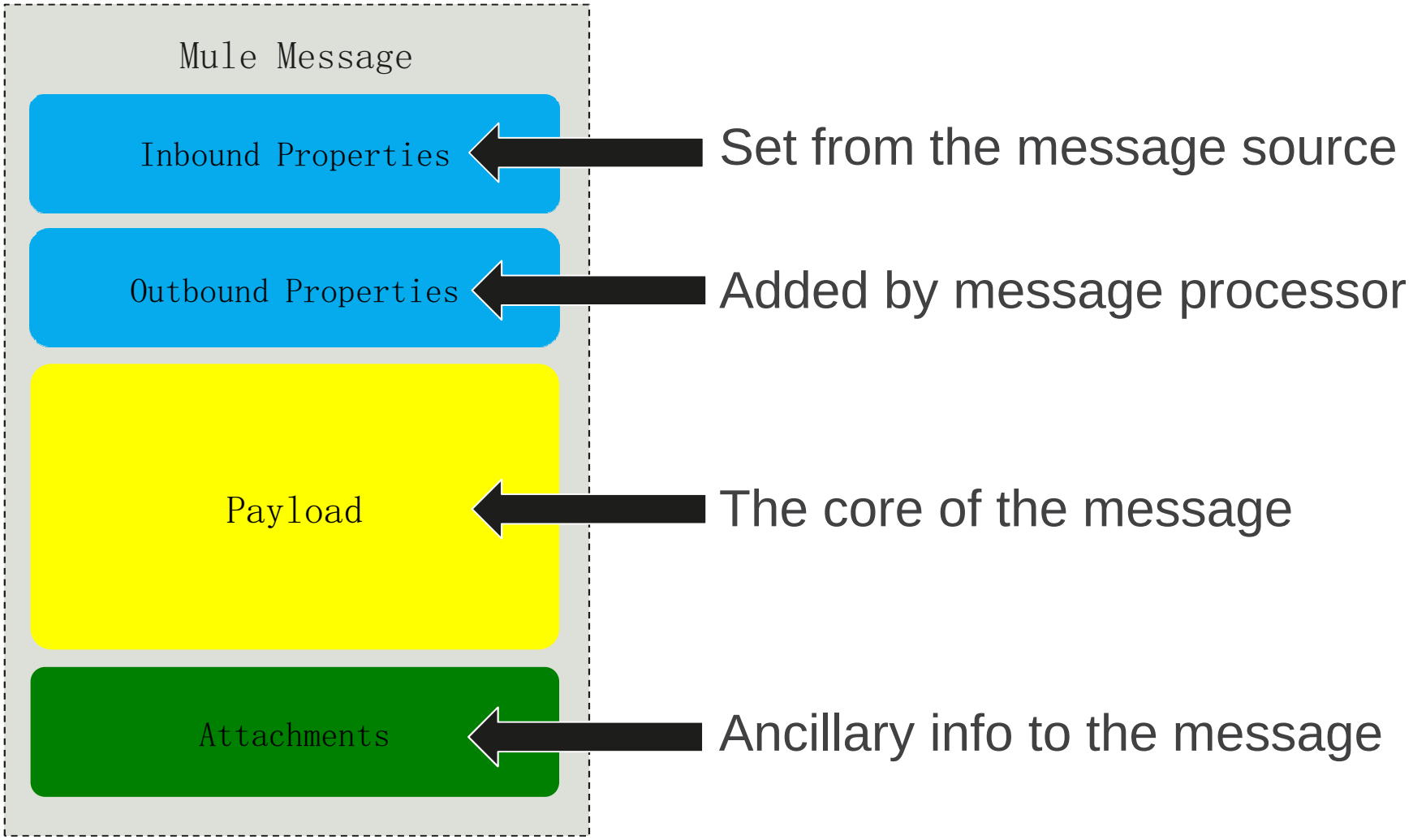


MuleSoft

Reading and writing Mule messages

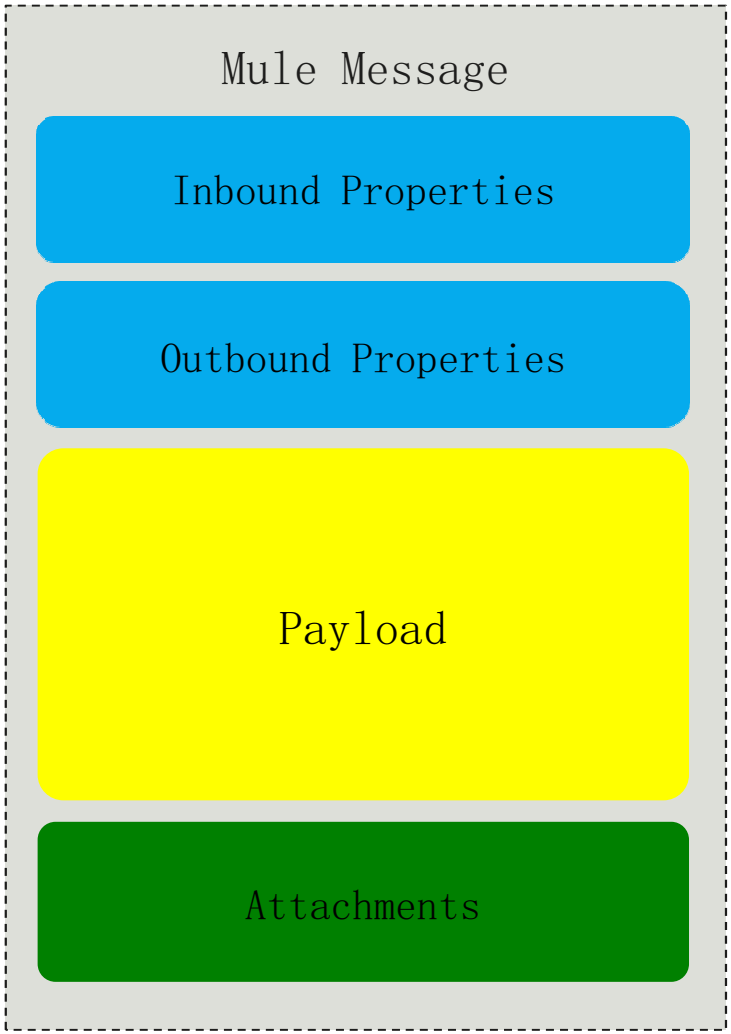


Mule messages



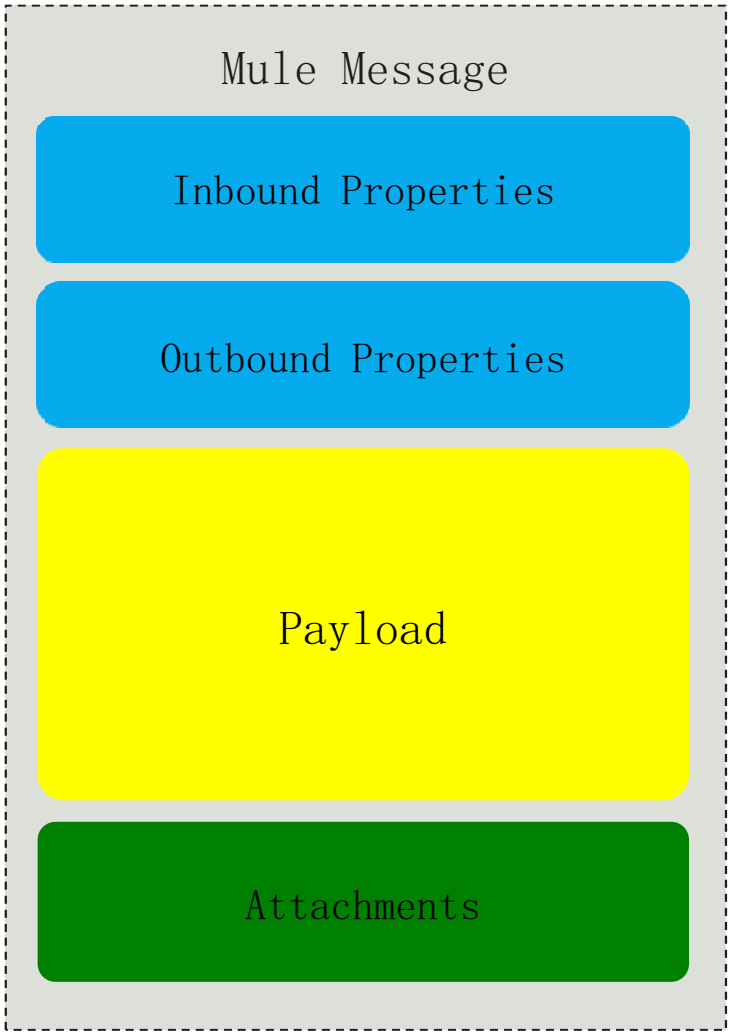
MuleSoft

Message properties



- Inbound properties
  - Set from the message source
  - Read-only access
  - Persist throughout the flow
- Outbound properties
  - Added by message processor
  - Read/write access
  - Can set, remove, copy

Message payload and attachments



- Payload
  - The core of the message
  - Contains primary info to be processed
  - Contains a Java Object
- Attachments
  - Ancillary info to the message
  - Similar to an email attachment

Payload representation

Raw Data

Payload

```
<order>
  <id>ed921739-99c1-4e09</id>
  <custId>49e377ed-bc72-4523</custId>
  <itemsTotal>200.34</itemsTotal>
 </order>
```

org.java.lang.String

Structured Data

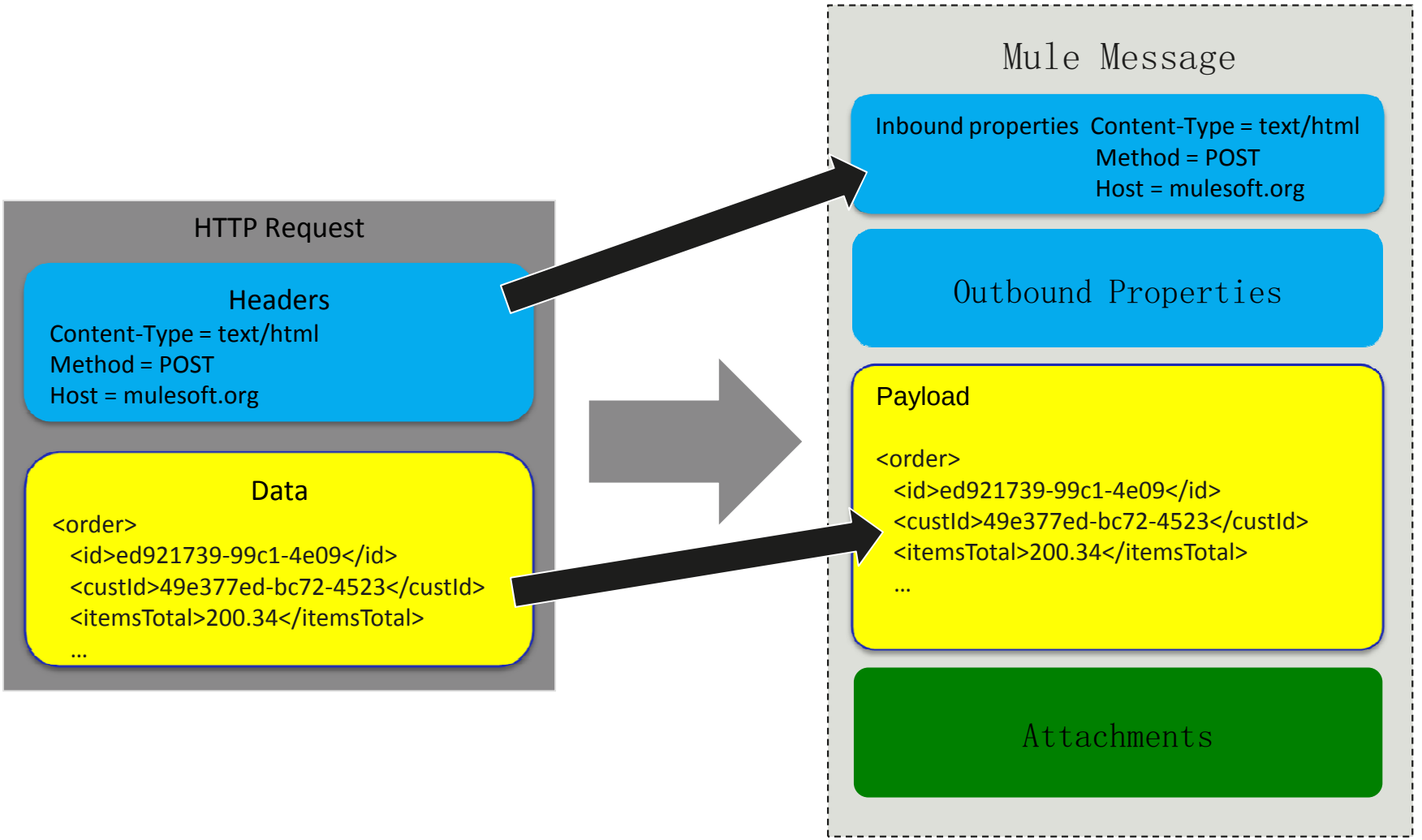
Payload

id: ed921739-99c1-4e09
custId: 49e377ed-bc72-4523
itemsTotal: 200.34

org.java.util.HashMap

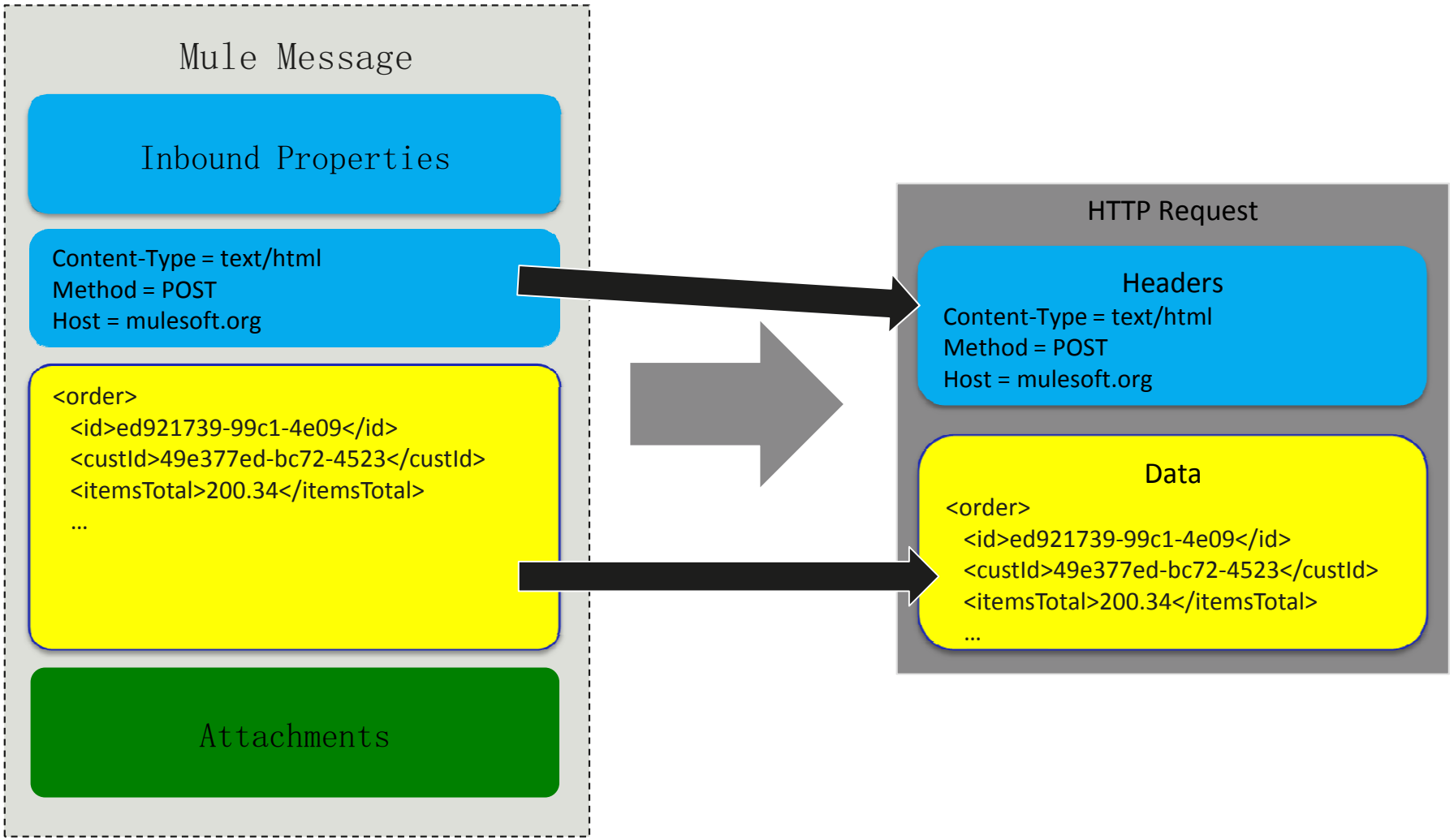
- Raw data often of type
  - String
  - InputStream
  - Byte[] (Byte array)
- Structured data often of type
  - Map
  - Structured Java Object
    - Order, Account, etc.

Inbound message properties



MuleSoft

Outbound message properties



MuleSoft

Setting message properties

Set Payload

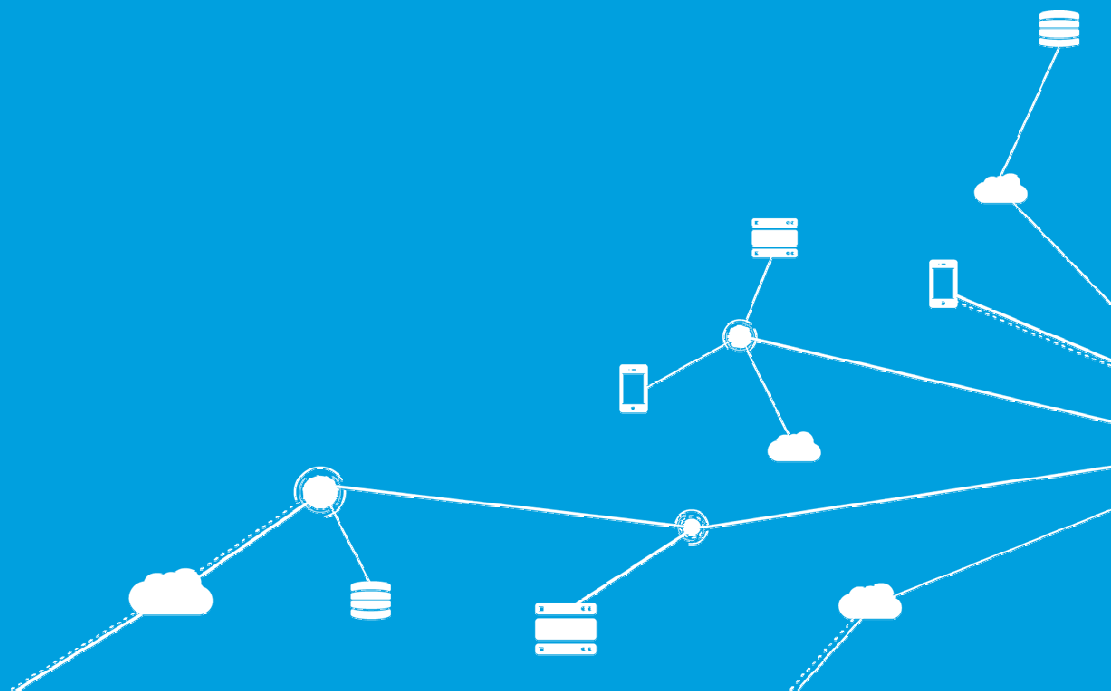
- Sets the value of the message payload

Property

- Sets, removes, or copies properties on the outbound scope of a message
  - message.outboundProperties

43

Writing Mule expressions



The Mule Expression Language (MEL)

- MEL is a lightweight, Mule-specific expression language
- Use it to access and evaluate the data in the payload, properties, and variables of a Mule message
- Accessible and usable from within virtually every message processor in Mule
  - Is used to modify the way the processors act upon the message such as routing or filtering
- Makes use of Mule-specific context objects
- Case-sensitive
- Easy to use with auto-complete everywhere

Basic MEL syntax

`#[]` Encapsulates all Mule expressions

`#[message]` Holds a context object

`#[message.payload]`

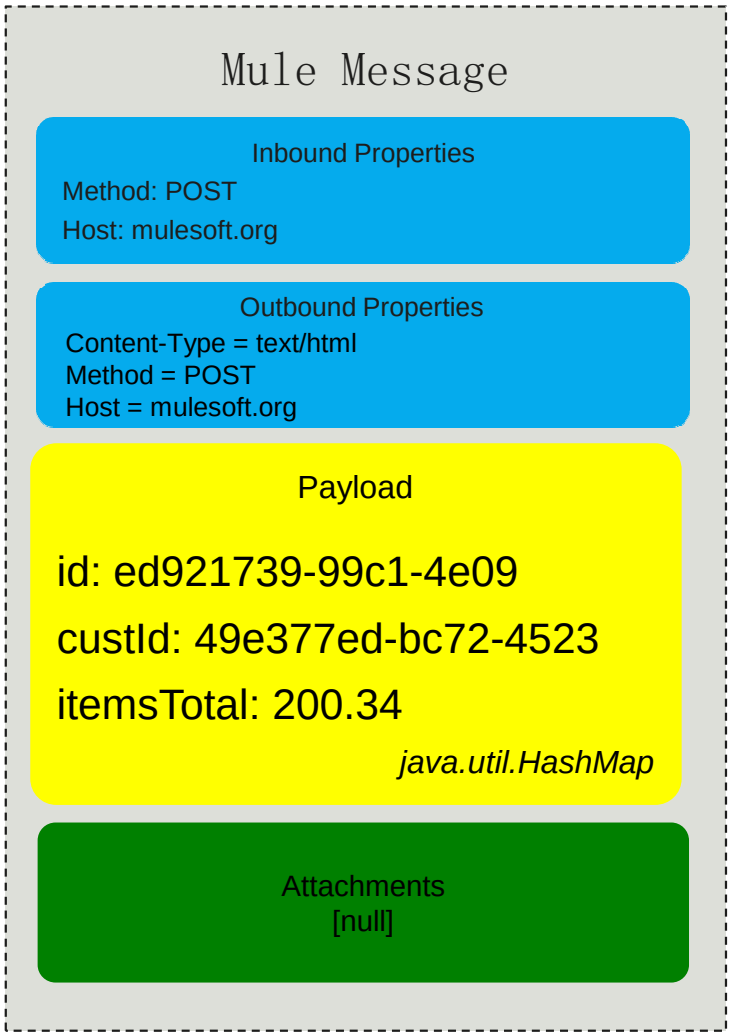
Dot notation to access fields or methods

Context objects

server	Operating system that message processor is running
mule	The Mule instance that the application is running
application	User application the current flow is deployed in
message	The Mule message that the message processer is processing

MuleSoft

Accessing message data




```
#[message.inboundProperties.host]
```

mulesoft.org

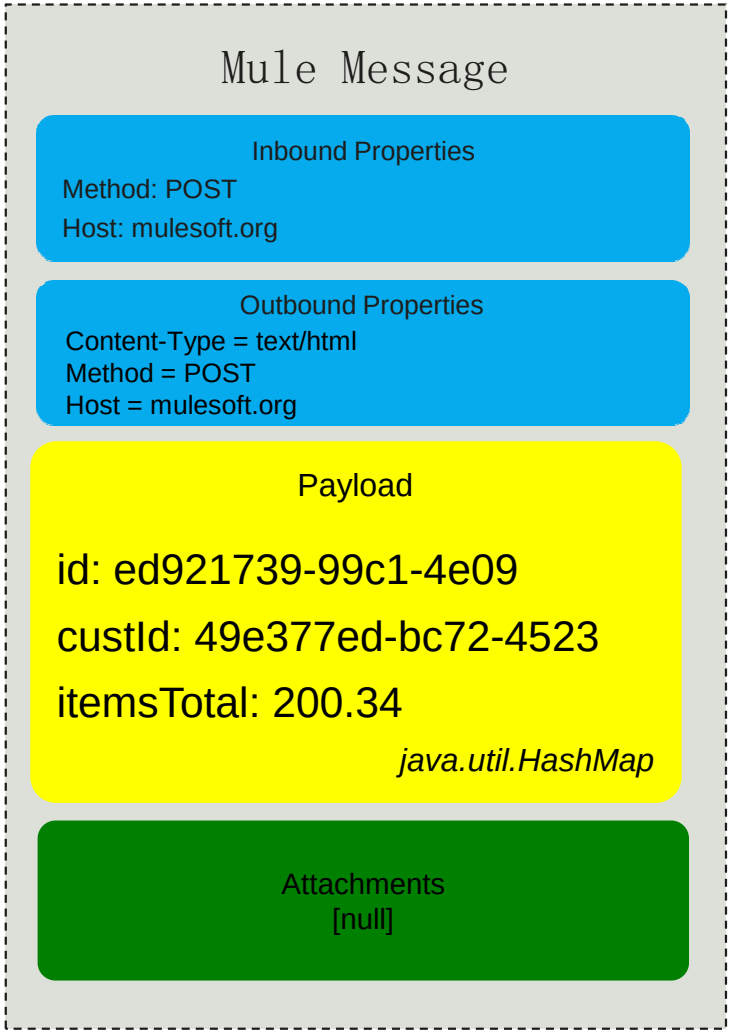
```
#[message.inboundProperties['http.method']]
```

POST

```
#[message.outboundProperties['content-type']]
```


MuleSoft

Accessing message payload data




```
#[message.payload.id]
#[message.payload['id']]
```

ed921739-99c1-4e09

```
#[message.payload.itemsTotal]
```

200.34

```
#[message.payload.toString()]
```

```
#[payload.id]
```

Is a shortcut for #[message.payload]

This shortcut only works with payload

MuleSoft

Accessing relational map data

FirstName	LastName	City	State	
John	Muley	Boston	Ohio	
Mark	Dailer	Cleveland	Ohio	
Bill	Muley	Avon	Ohio	

```
#[message.payload[1]['LastName']]
```

Dailer

```
#[message.payload[0].City]
```

Boston

MuleSoft

Writing expressions

- Operators
  - Arithmetic: +, -, /, *, %
  - Evaluation: ==, !=, >, <, >=, <=, contains, is

    #[message.inboundProperties.'http.query.params'.lastname != null]

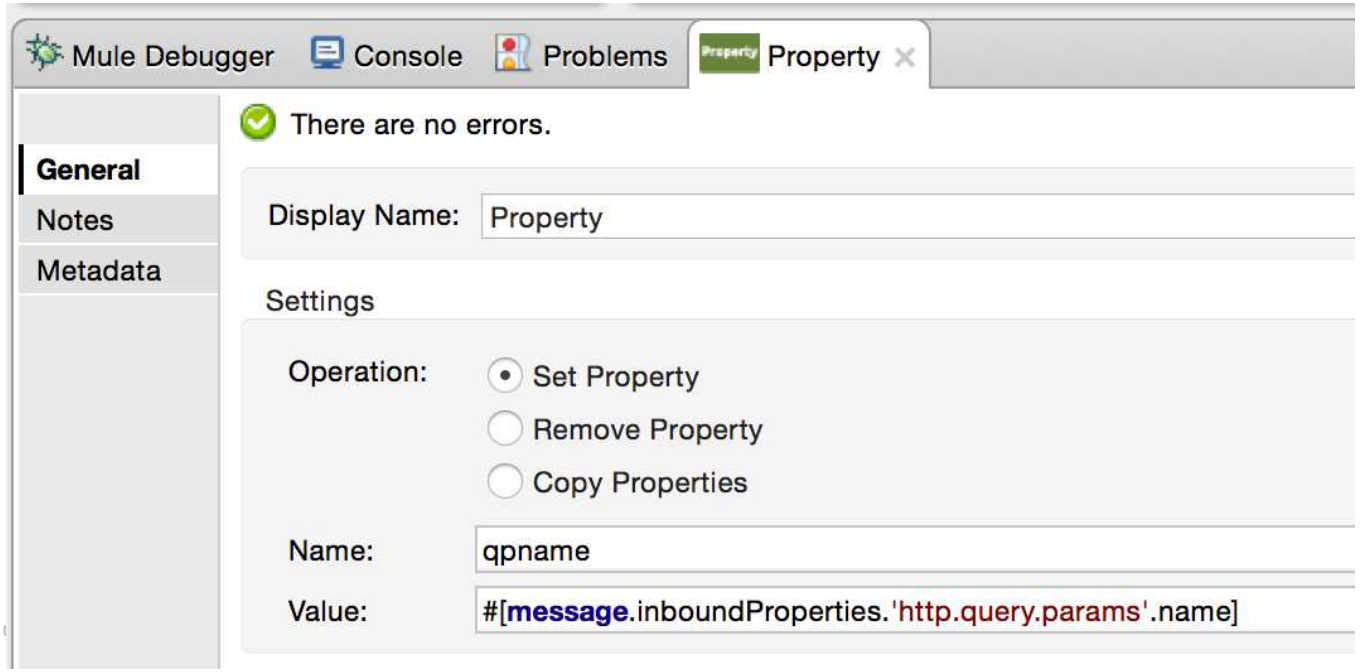
- Testing for emptiness
  - The literal **empty** tests the emptiness of a value
    - Null, boolean false, “”, “ ”, zero, empty collections

- Data extraction
  - XPath: #[xpath('expression')]
  - RegEx: #[regex('expression')]

MuleSoft

Walkthrough 2-3: Read and write message properties

- Write MEL expressions
- Use the Debugger to read inbound and outbound message properties
- Use the Property transformer to set outbound message properties



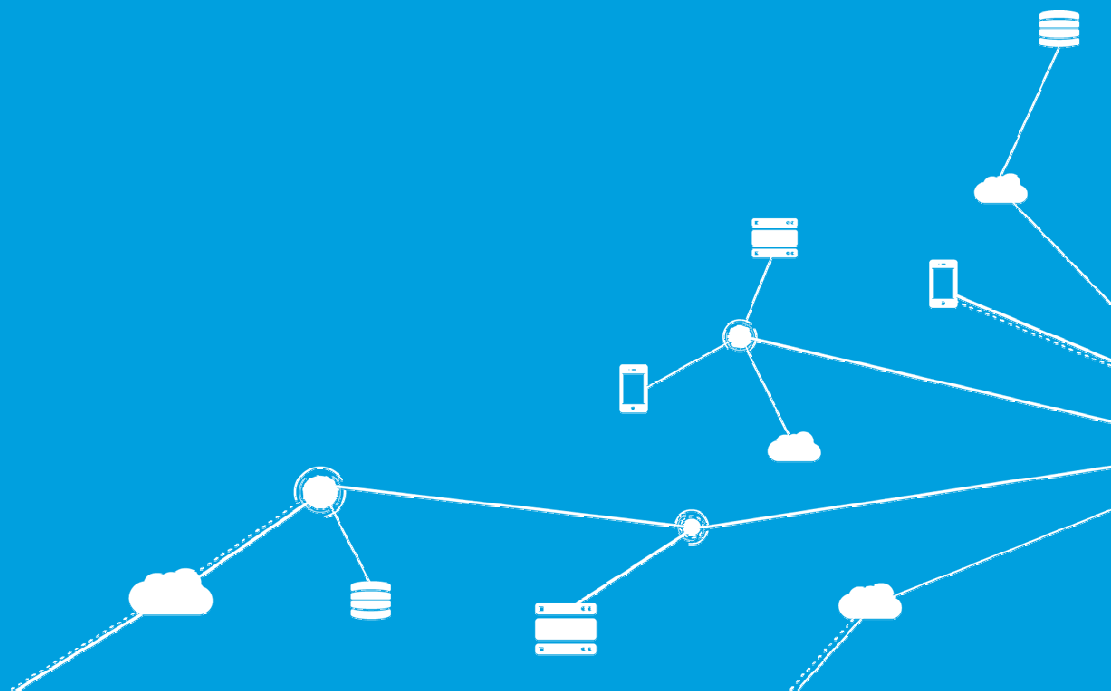
MuleSoft

MEL references

- MEL expression reference
  - http://www.mulesoft.org/documentation/display/current/Mule+Expression+Language+Reference
- MEL language tips
  - http://www.mulesoft.org/documentation/display/current/Mule+Expression+Language+Tips

MuleSoft

Creating variables



Context variables

flowVars

sessionVar

recordVars

```
#[flowVars.ticketNum]
```


MuleSoft

Setting variables

Variable

- Sets or removes **flow variables**
  - Variables on the message tied to the current
  - Reference as flowVars
    - The flowVars reference is optional
    - #[flowVars.foo] or #[foo]

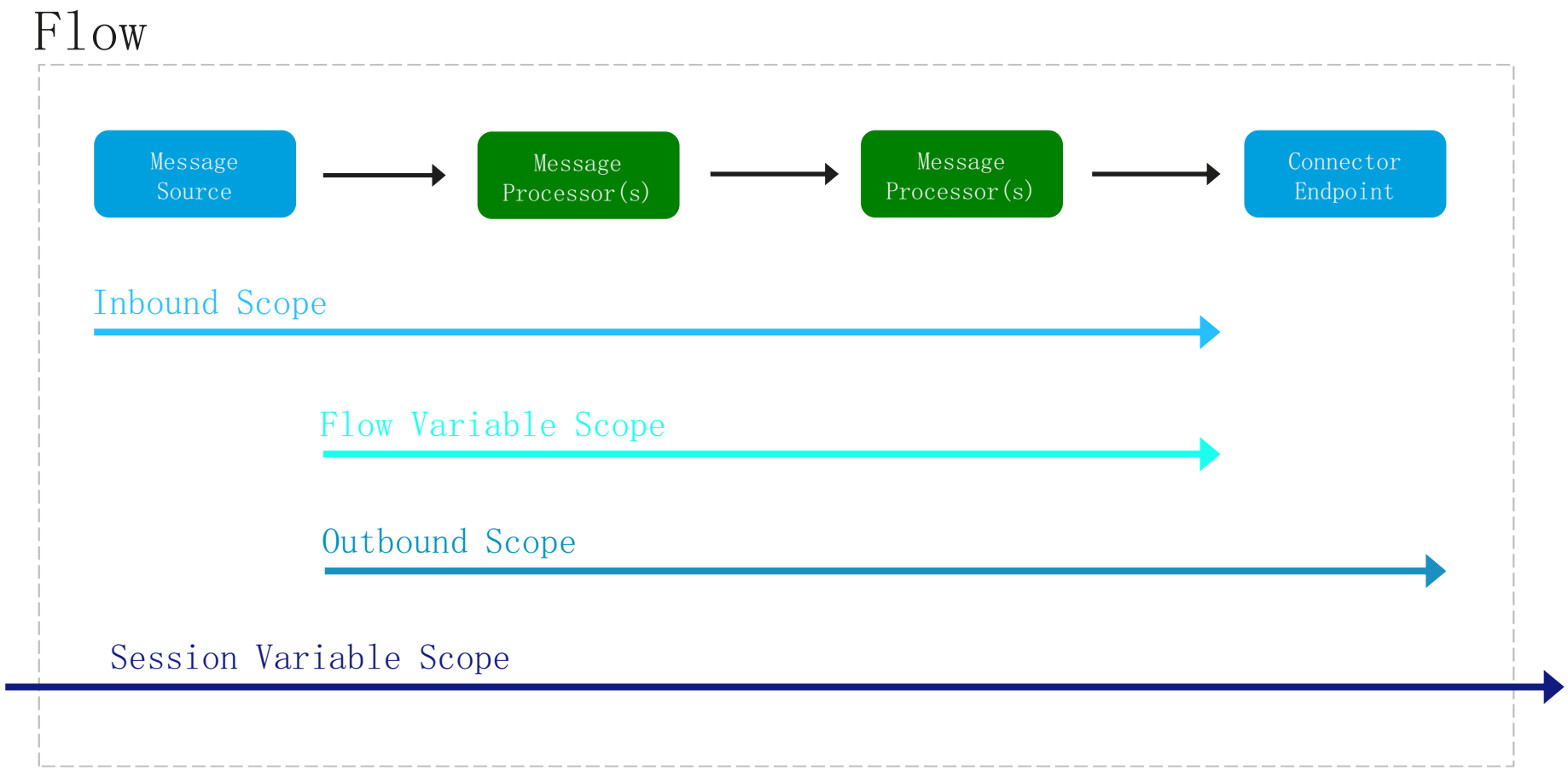
Session

- Sets or removes **session variables**
  - Variables tied to a message for its lifecycle across flows, applications, and servers
  - They are persisted across some but **not all** transport barriers
  - Reference as sessionVars
    - #[sessionVars.foobar]

  56

MuleSoft

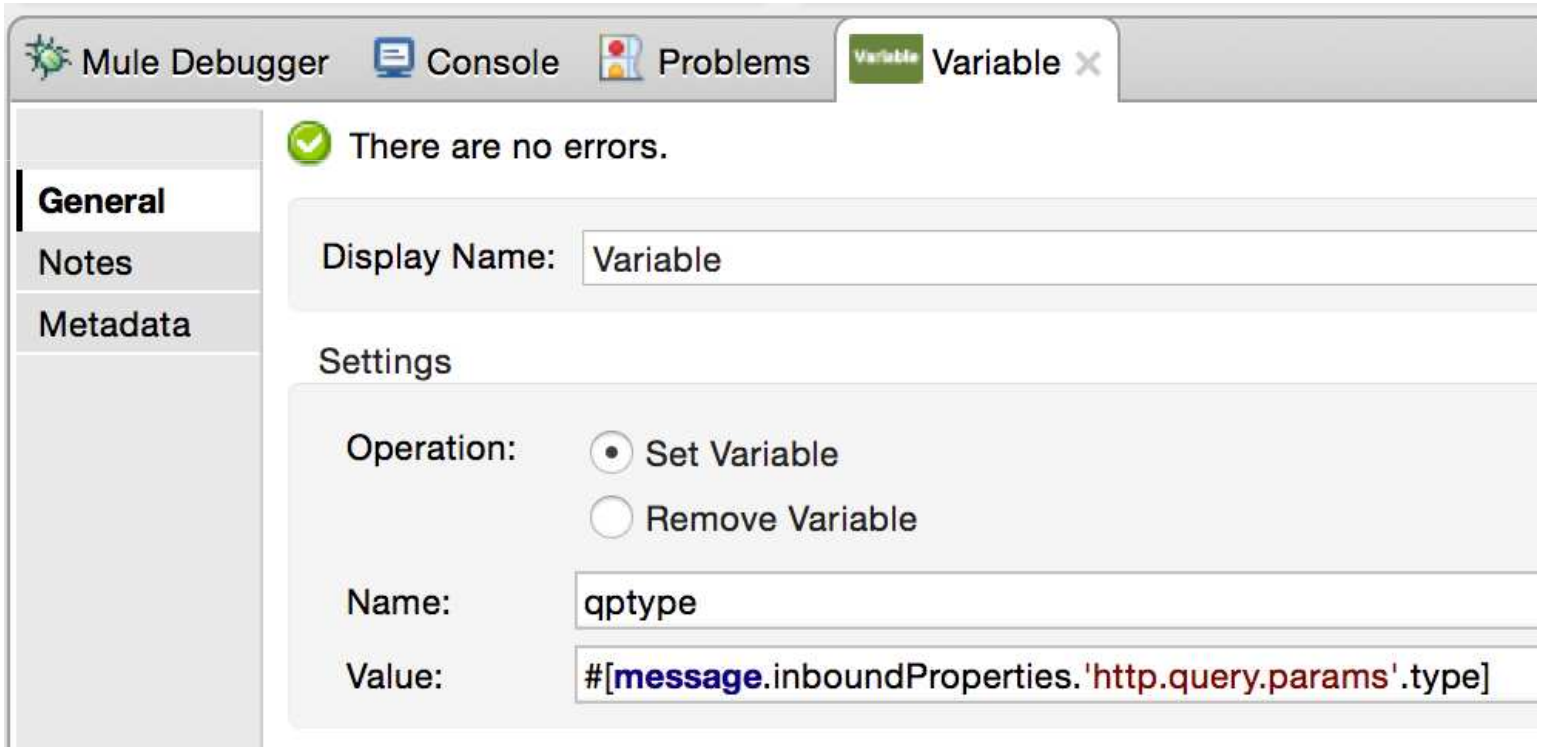
Variable persistence



MuleSoft

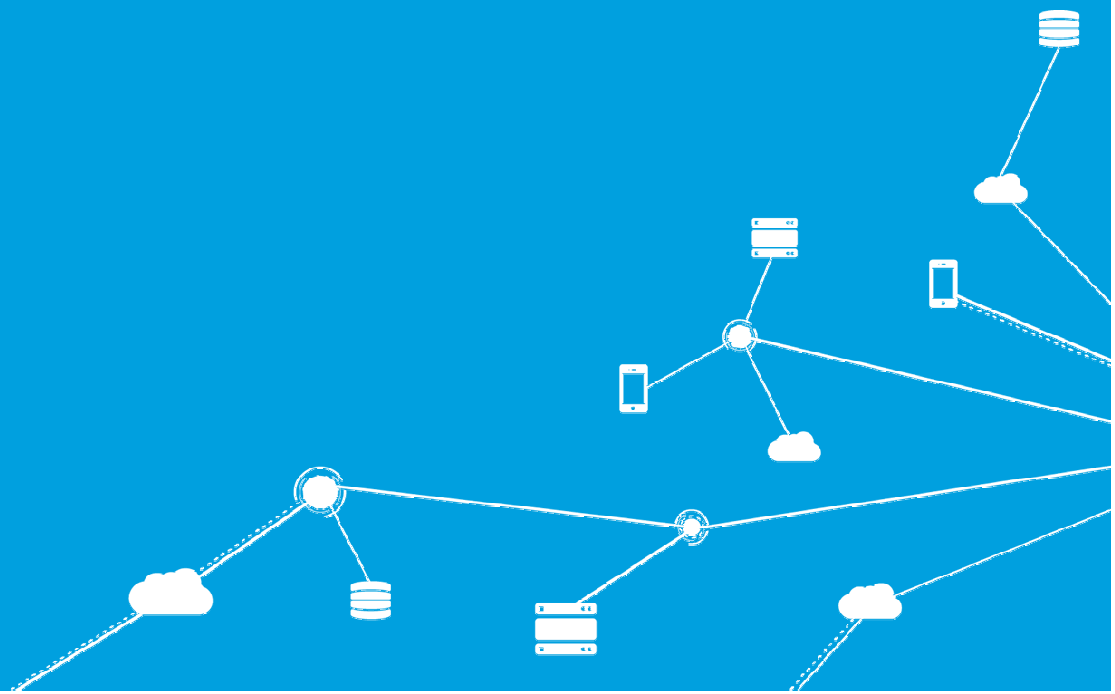
Walkthrough 2-4: Read and write variables

- Use the Variable transformer to create flow variables
- Use the Session transformer to create session variables



MuleSoft

Summary



Summary

- In this module, you learned to build integration applications graphically with Anypoint Studio
  - Two-way editing between graphical and XML views
  - An embedded Mule runtime for testing applications
  - A Visual Debugger (EE) for debugging applications
- Mule applications accept and process messages through a series of message processors plugged together in a flow
- Mule messages have inbound properties, outbound properties, a payload, and attachments

Summary

- Message processors include connectors, transformers, components, scopes, and flow control elements
- Connectors are inbound or outbound and endpoint or operation based
- When you drag out a connector, an endpoint is created
- For most endpoints, a lot of the configuration is encapsulated in a separate, reusable global element

MuleSoft

Summary

- Use the HTTP Listener connector as an inbound endpoint to trigger a flow with an HTTP request
- Use the Property transformer to set, remove, or copy message outbound properties
- Use the Set Payload transformer to set the payload
- Use the Logger component to display data in the console
- Use the Mule Expression Language (MEL) to write expressions #[]
- Use the Variable transformer to create flow variables

MuleSoft