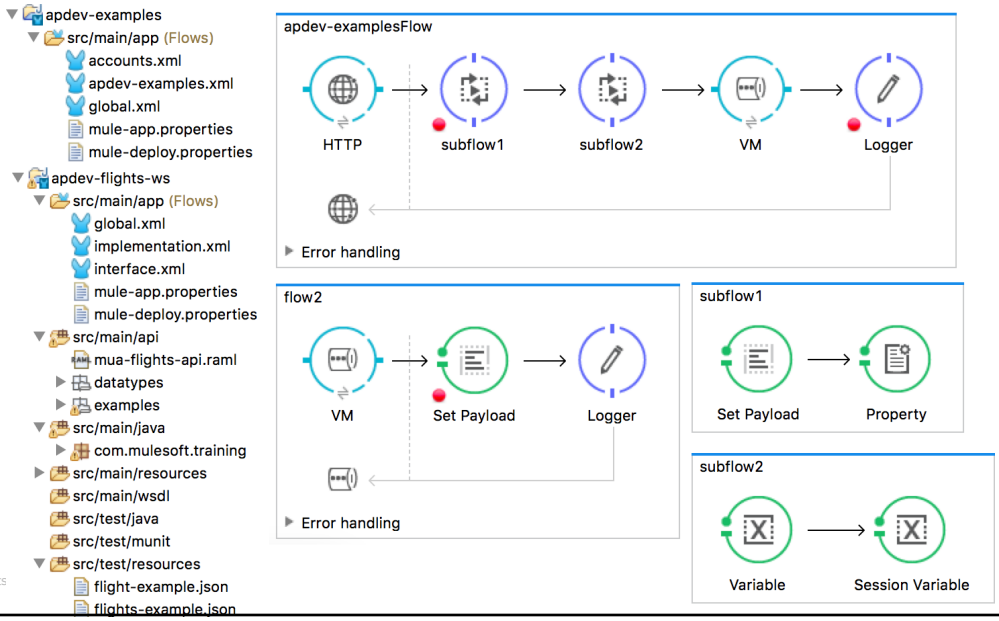
Module 7: Structuring Mule Applications

Goal



At the end of this module, you should be able to

MuleSoft

- Create and reference flows and subflows
- Pass messages between flows using the Java Virtual Machine (VM) transport
- Investigate variable persistence through subflows and flows and across transport barriers
- Encapsulate global elements in a separate configuration file
- Explore the files and folder structure of Mule projects

Encapsulating processors into separate flows and subflows

Break up flows into separate flows and subflows

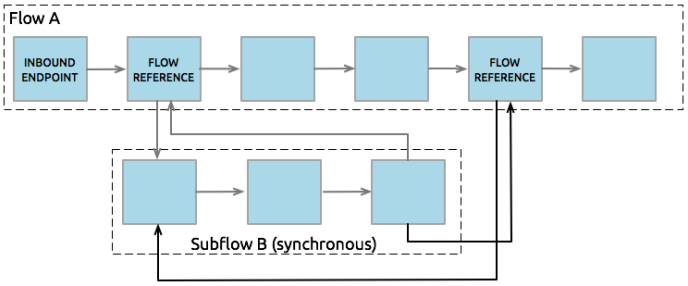
MuleSoft

- Makes the graphical view more intuitive
  - You don't want long flows that go off the screen
- Makes XML code easier to read
- Enables code reuse
- Provides separation between an interface and implementation
  - We already saw this
- Makes them easier to test

Subflows

MuleSoft

- Subflows are executed exactly as if the processors were still in the calling flow
  - Always run **synchronously** in the same thread
  - **Inherit the processing and exception strategies** of the flow that triggered its execution



Flows

MuleSoft

- Flows, on the other hand, have much more flexibility in how they are used
  - Can have **their own processing and exception** strategies
  - Can be **synchronous** or **asynchronous**
- Flows without message sources are sometimes called **private** flows

What is a flow processing strategy?

MuleSoft

- A flow **processing strategy** determines how Mule implements message processing for a given flow
  - Should the message be processed synchronously (on the same thread) or asynchronously (on a different thread)?
  - If asynchronously, what are the properties of the pool of threads used to process the messages?
  - If asynchronously, how will messages wait for their turn to be processed in the second thread?

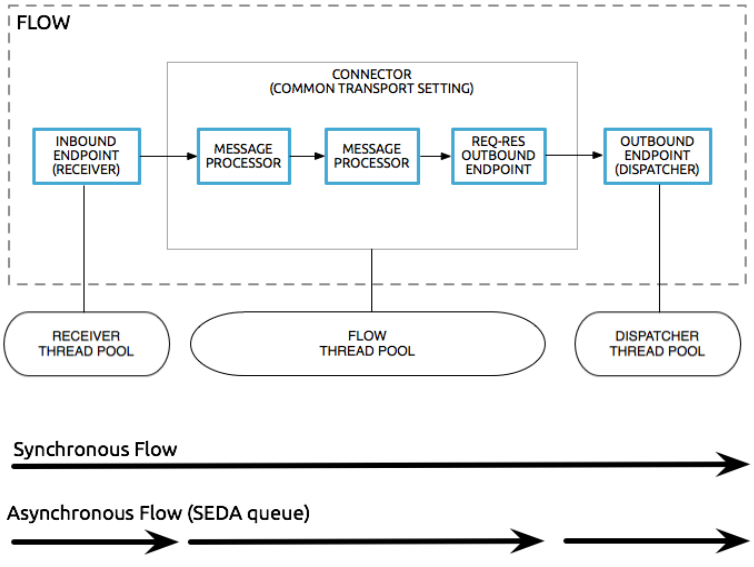
Flows contain 3 thread pools

MuleSoft

- Receiving
  - Message source's threads
- Flow processing
  - Message processor's threads
- Dispatching
  - Outbound endpoint's threads

Usage of pools depends on a flow's behavior

MuleSoft



Staged Event Driven Architecture (SEDA)

MuleSoft

- The architecture upon which Mule was built
- Decouples receiving, processing, and dispatching phases
- Supports higher levels of parallelism in specific stages of processing
- Allows for more-specific tuning of areas within a flow's architecture

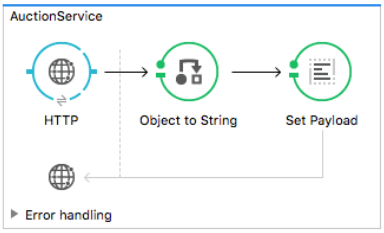
What determines a flow's processing strategy?

MuleSoft

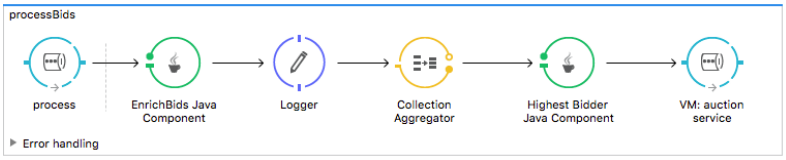
- Mule automatically sets a flow to be
  - Synchronous or queued-asynchronous
- A flow is set to synchronous if
  - The message source is request-response
    - The sender of the message is expecting a response
  - The flow partakes in a transaction
- Otherwise a flow is set to queued-asynchronous
  - The message source is not expecting a response

Synchronous flows

- When a flow receives a message, all processing, including the processing of the response, is done in the same thread
- Uses only the message source's thread pool
- The flow's thread pool is elastic and will have one idle thread that is never used
- Tuning for higher-throughput happens on the connector receiver's level



Queued-asynchronous flows

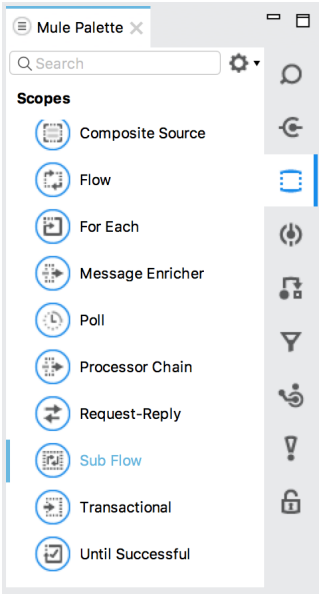


- Decouples and uses all 3 thread pools
- Uses queues, whose threads drop messages off for the subsequent pool's thread to pickup
- Pools, queues, and behaviors of this strategy are configurable
- By default, the flow thread pool has 16 threads

Creating flows and subflows

MuleSoft

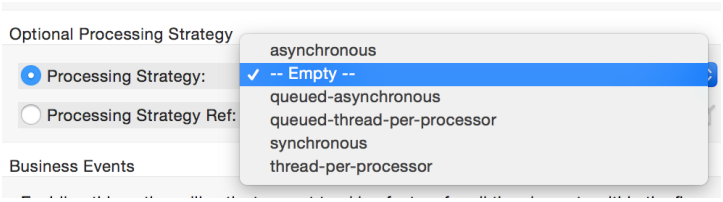
- Use Flow scope to create a new flow
  - Or drag any message processor to the canvas
- Use Sub flow scope to create subflows
- Use Flow Reference component to pass messages to other flows or subflows
- Flow variables persist through all flows unless the message crosses a transport boundary
  - We will explore this shortly



Setting the flow processing strategy

MuleSoft

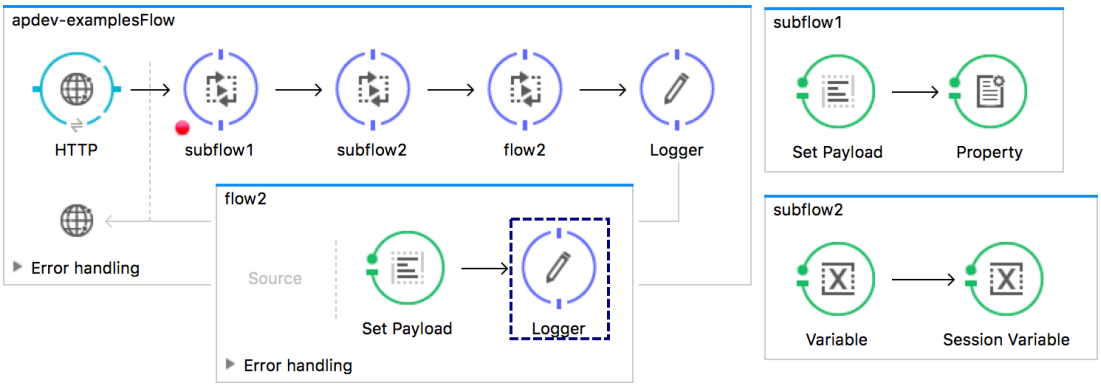
- The flow processing strategy is automatically set
- It can be changed in the flow's properties view
- This is usually done when a custom queued-asynchronous profile has been created for tuning application performance
  - This is covered in the *Anypoint Platform Development: Advanced* course



Walkthrough 7-1: Create and reference flows and subflows

MuleSoft

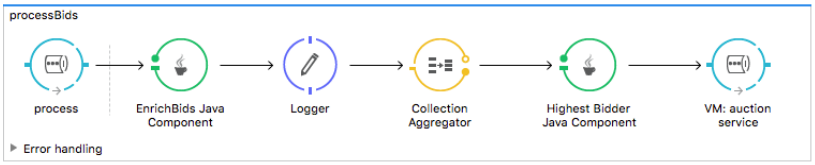
- Extract processors into separate subflows and flows
- Use the Flow Reference component to reference other flows
- Explore variable persistence through flows and subflows



18

Using VM endpoints in flows

The Java Virtual Machine (VM) transport



- The Java Virtual Machine (VM) transport can be used for intra-JVM communication between Mule flows
- Each app in a Mule instance has its own, unique set of VM endpoints
- The VM transport can only handle communications within an app or between apps in the same domain
- This transport by default uses in-memory queues but can optionally be configured to use persistent queues
  - VM file persistency does not work on clusters

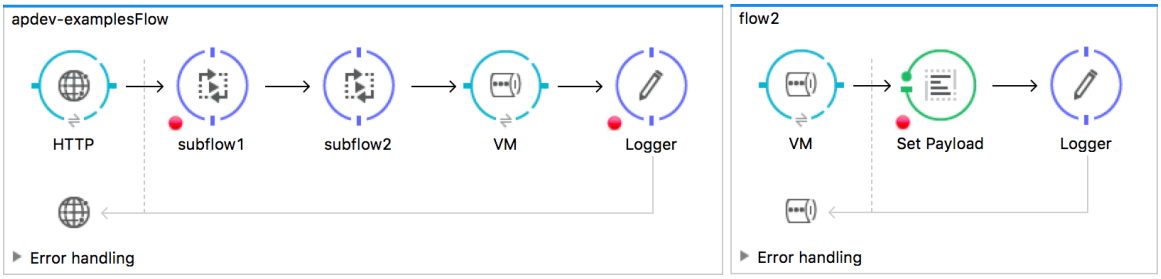
Using the VM transport

- Before Mule 3, the VM transport was needed to pass a message from one flow to another
- In Mule 3, Flow Reference was added to let flows directly reference one another without a transport in the middle
  - This obviates the need for VM in many cases
- VM transport is now mostly used to
  - Achieve higher levels of parallelism in specific stages of processing
  - Allow for more-specific tuning of areas within a flow's architecture
  - Call flows in other applications that are in the same domain
- Using VMs to tune application performance is covered in the *Anypoint Platform Development: Advanced* course

Walkthrough 7-2: Pass messages between flows using the Java Virtual Machine (VM) transport

MuleSoft

- Pass messages through an HTTP transport barrier
- Pass messages between flows using the VM transport
- Explore variable persistence across transport barriers



Organizing Mule application files

Separating apps into multiple configuration files

MuleSoft

- Just as we separated flows into multiple flows, we also want to separate configuration files into multiple configuration files
- Monolithic files are difficult to read and maintain
- Separating an application into multiple configuration files makes code
  - Easier to read
  - Easier to work with
  - Easier to test
  - More maintainable

Encapsulating global elements in a configuration file

MuleSoft

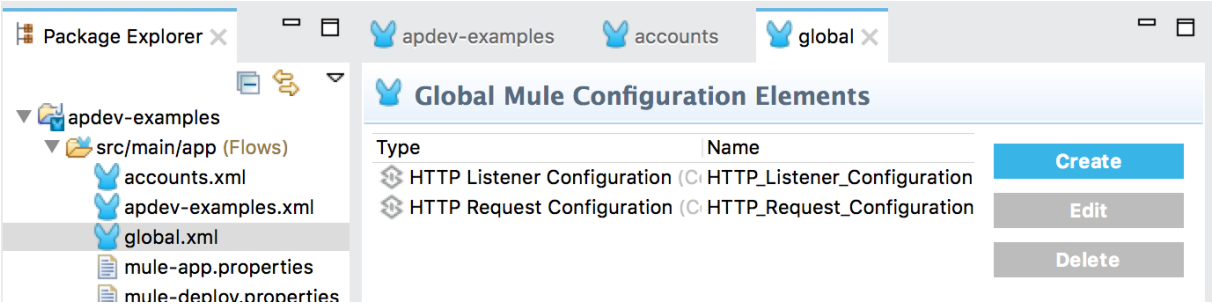
- It can be confusing if you reference global elements in one file that are defined in various, unrelated files
- It also makes it hard to find them
- A good solution is to put most global elements in one config file
  - All the rest of the files reference them
  - If a global element is specific to and only used in one file, it makes sense to keep it in that file

Creating multiple applications

- You are also not going to want all your flows in one application/project
- Separate functionality into multiple applications to
  - Allow managing and monitoring of them as separate entities
  - Use different, incompatible JAR files
- Run more than one application at a time in Anypoint Studio by creating a run configuration

Walkthrough 7-3: Encapsulate global elements in a separate configuration file

- Create a new configuration file with an endpoint that uses an existing global element
- Create a configuration file global.xml for just global elements
- Move the existing global elements to global.xml

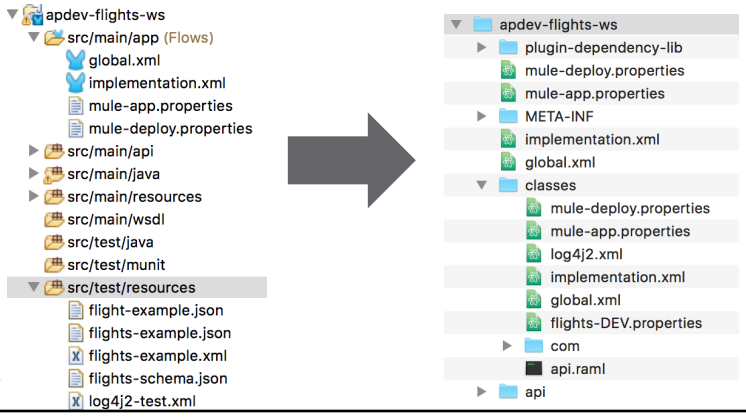


Organizing Mule project files

Examining the folder structure for a Mule project

MuleSoft

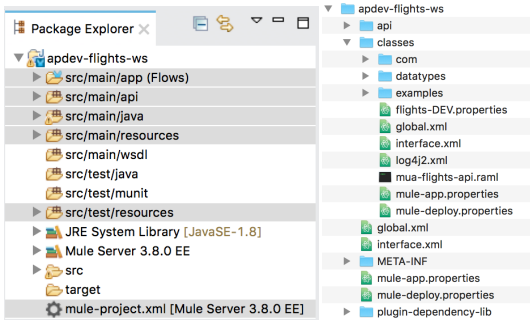
- The names of folders indicate what they should contain
- src/test folders should contain files only needed at development time
  - Like schema and example files for metadata types, sample data for transformations
  - They are not included in the application ZIP when it is packaged



Walkthrough 7-4: Create a well-organized Mule project

MuleSoft

- Create a project with a new RAML file that is added to Anypoint Platform
- Review the project's configuration and properties files
- Create an application properties file and a global configuration file
- Add Java files and test resource files to the project
- Create and examine the contents of a deployable archive for the project



Sharing resources between applications

Sharing resources between applications

- **Shared domains** can be used to share some connectors and endpoints between applications
- Enables multiple development teams to work in parallel using the same set of reusable connectors
- Defining these connectors as shared resources at the domain level allows the team to
  - Expose multiple services within the domain on same port
  - Share the connection to persistent storage
  - Ensure consistency between applications upon any changes, as the configuration is only set in one place

Using domains to share resources between apps

- Only available on customer-hosted Mule runtimes, not on CloudHub
- The general process is to
  - Create a Mule Domain Project and associate Mule applications with a domain
  - Define a set of resources (and the libraries required by those resources) for a domain to share between the apps
- Currently, only the following can be shared
  - HTTP/HTTPS (both endpoints and connectors), WMQ, Database, JMS, VM, JBoss and Bitronix Transaction Managers

Sharing code between applications

- You may also want to share code between applications
  - Connector configurations
  - Endpoint definitions
  - Flows and subflows
  - Transformers
  - DataWeave DWL files
- To share code between applications, use **Maven and its dependency management mechanism**
- Maven is a software project management tool that helps
  - Control the build lifecycle of software projects
  - Manage project dependencies

Using Maven to share code between applications

- Using Maven is covered in the *Anypoint Platform Development: Advanced* course
- The general process is to
  - Create a shared XML configuration file containing the code to share
  - Compile it into a JAR file
  - Use Maven's dependency mechanism to include it in your application
  - Import the configuration file for reference
  - https://docs.mulesoft.com/mule-user-guide/v/3.8/sharing-custom-configuration-fragments

Summary

Summary

MuleSoft

- Separate a flow into multiple flows for easier to read files and for reusability
- Extract duplicate code into reusable private flows or subflows
- **Private flows** are flows without message sources
- **Subflows** are executed exactly as if in the calling flow
  - Always run synchronously
  - Inherit processing and exception strategies of the calling flow
- A **flow processing strategy** determines how Mule implements message processing for a given flow
  - All Mule flows have an implicit processing strategy which Mule applies automatically
  - Endpoints with a request-response exchange pattern are set to synchronous
  - Endpoints with a one-way exchange pattern are set to queued-asynchronous

Summary

MuleSoft

- Use **Flow Reference** to let flows directly reference one another without a transport in the middle
- Use the **Java Virtual Machine (VM)** transport for intra-JVM communication between Mule flows
  - To achieve higher levels of parallelism in specific stages of processing
  - To allow for more-specific tuning of areas within a flow's architecture
  - To call flows in other applications that are in the same domain
- Flow variables persist through all flows unless the message crosses a transport boundary (like VM or HTTP)
- Session variables persist across some but not all transport barriers (VM yes, HTTP no)

Summary

MuleSoft

- Separate application functionality into **multiple configuration files** for easier development and maintenance
  - Encapsulate global elements that will be used in multiple configuration files into their own separate configuration file
- Put files only needed at development time in the **src/test** folders
  - They are not included in the application ZIP when it is packaged
- Separate functionality into **multiple applications** to allow managing and monitoring of them as separate entities
- Share resources between applications by creating a **shared domain**