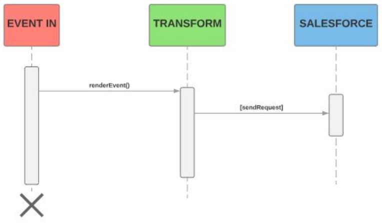
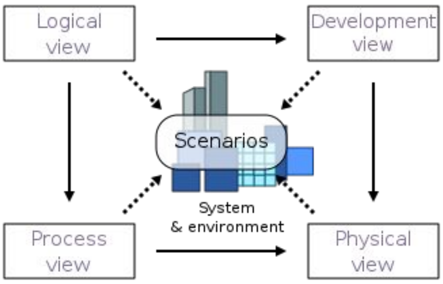
MuleSoft

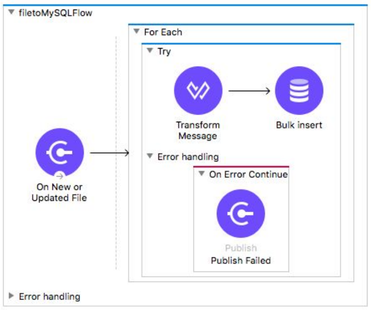
Module 17: Reviewing Documenting Integration Solutions Architectures

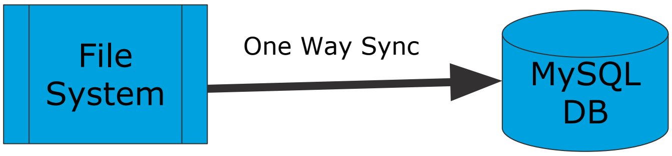
Goal

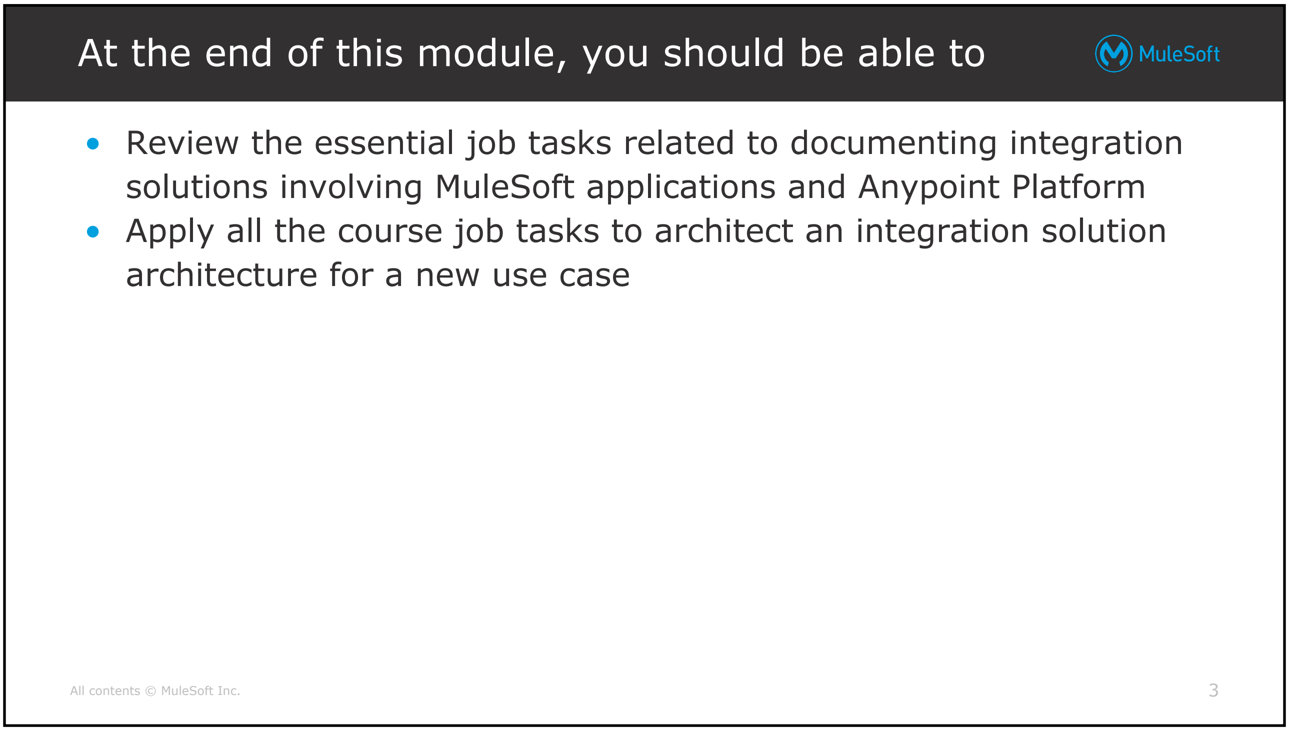
MuleSoft





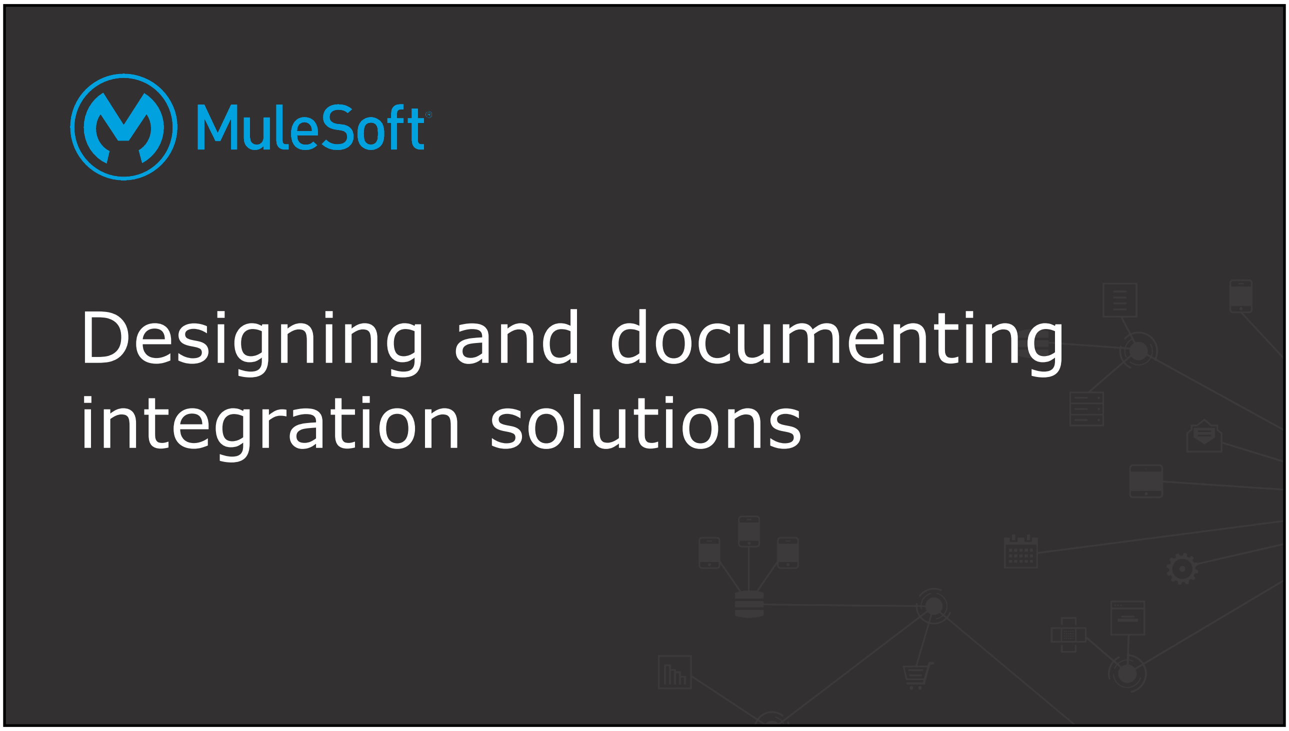






At the end of this module, you should be able to

- Review the essential job tasks related to documenting integration solutions involving MuleSoft applications and Anypoint Platform
- Apply all the course job tasks to architect an integration solution architecture for a new use case



Designing and documenting integration solutions

Introducing the course wrap-up exercises

- The goal is to apply the essential job tasks discussed in this course to a new use case
- You will work independently to design and propose a solution, then the group will discuss the options and agree on a solution
  - You can mock flows in Anypoint Studio, or use any other architecture diagramming tool
- Each exercise iterates on your design to add additional details and complexity
  - Exercise 17-1: Appropriately design Mule event processing for a file transfer use case
  - Exercise 17-2: Respond to changing performance requirements
  - Exercise 17-3: Respond to changing reliability requirements

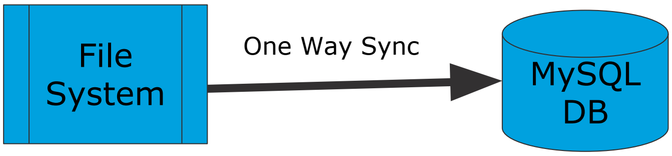
Essential job tasks related to documenting integration solutions

- In this class you have learned how to apply these essential job tasks to build a complete integration solutions architecture
  - Design integration solutions with Mule application flows and components
  - Apply appropriate event processing strategies in a Mule application to meet project requirements
  - Choose appropriate Mule event transformation and routing patterns
  - Choose appropriate state preservation and management options
  - Design secure Mule applications and network communications
  - Design testing strategies
  - Design effective logging and monitoring options
  - Decide and develop appropriate manual and automated deployment strategies
  - Design integration solutions architectures to balance high availability, reliability, transactionality, and performance goals

Exercise 17-1: Appropriately design Mule event processing for a file transfer use case

MuleSoft

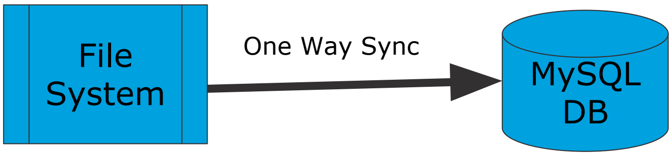
- Identify and recommend the Mule event processing models for a file transfer use case
- Identify and explain every factor that helps evaluate the best processing model
- Justify each Mule event processing model decision based on the identified factors
- Document flows that can implement the selected Mule event processing models

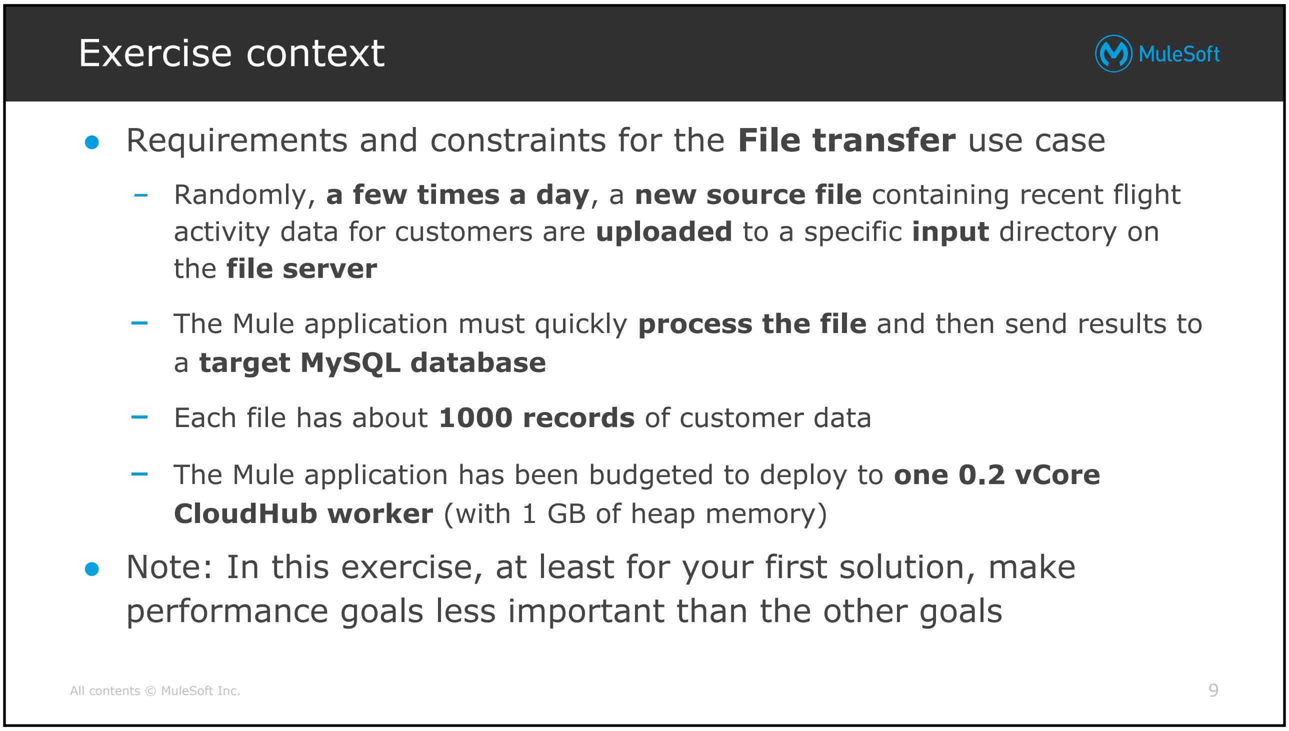


Exercise 17-1: Appropriately design Mule event processing for a file transfer use case

MuleSoft

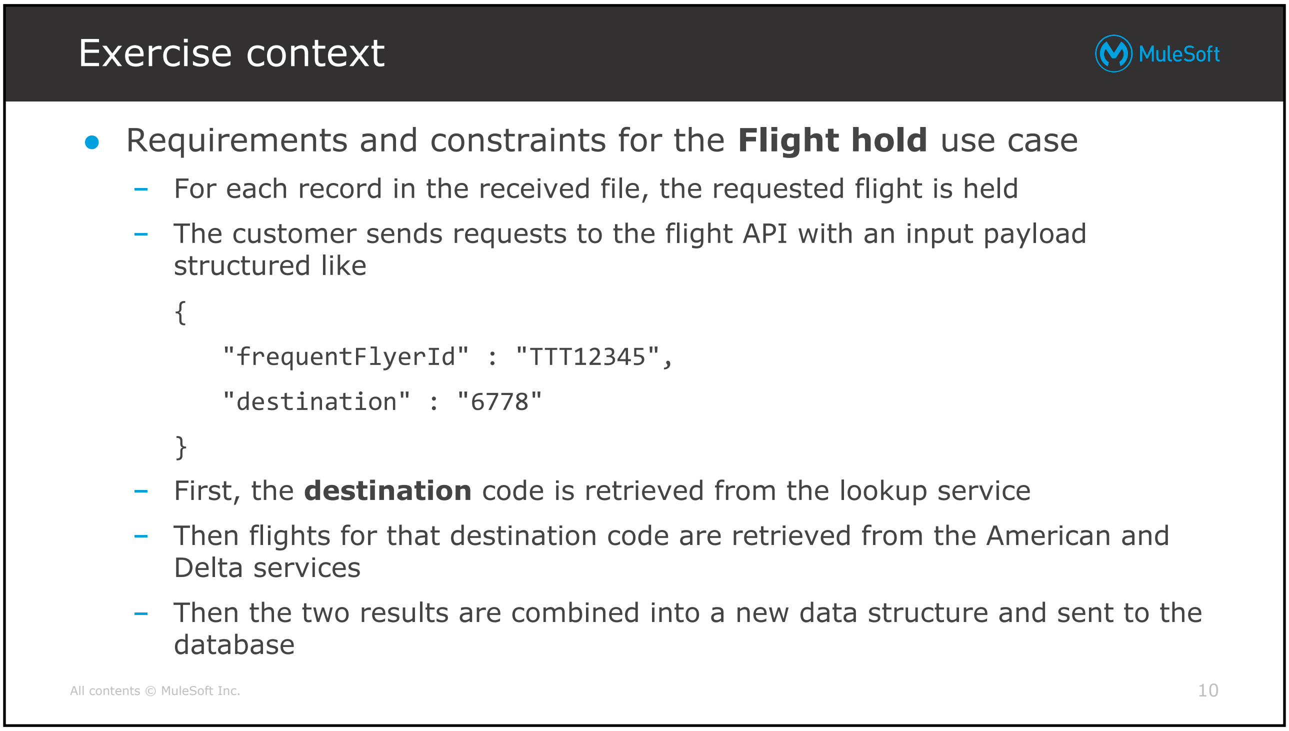
- Design for error handling and defensive programming
- Design for security requirements
- Design any needed state management and pick the best option
- Iterate on an integration solutions architecture to balance security, high availability, reliability, transactionality, and performance goals





Exercise context

- Requirements and constraints for the **File transfer** use case
  - Randomly, **a few times a day**, a **new source file** containing recent flight activity data for customers are **uploaded** to a specific **input** directory on the **file server**
  - The Mule application must quickly **process the file** and then send results to a **target MySQL database**
  - Each file has about **1000 records** of customer data
  - The Mule application has been budgeted to deploy to **one 0.2 vCore CloudHub worker** (with 1 GB of heap memory)
- Note: In this exercise, at least for your first solution, make performance goals less important than the other goals



Exercise context

- Requirements and constraints for the **Flight hold** use case
  - For each record in the received file, the requested flight is held
  - The customer sends requests to the flight API with an input payload structured like

```
{
    "frequentFlyerId" : "TTT12345",
    "destination" : "6778"
}
```

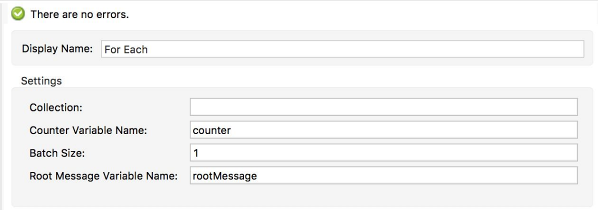
  - First, the **destination** code is retrieved from the lookup service
  - Then flights for that destination code are retrieved from the American and Delta services
  - Then the two results are combined into a new data structure and sent to the database

Exercise steps

- Load the starter project from the Module 17 exercise_starter folder into Anypoint Studio
- Define options to process the file
  - What are the options to read in a file?
  - After reading in a file, how are records in files processed?
  - Should records be processed sequentially, parallelly, or in batch?
  - How can the memory footprint be reduced while processing?
  - How are failed records managed?

Exercise solution

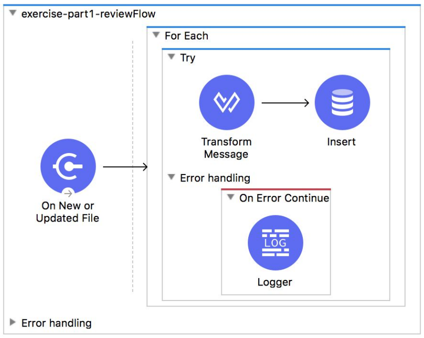
- A proposed processing model using a For Each scope
  - Processing is sequential, even though records are published to a JMS topic
  - For Each does not support buffering (Mule 3) so the entire file is loaded into memory
  - Throughput of the process is determined and limited by the For Each scope
  - A separate insert into the DB is performed for each record


There are no errors.

Display Name: For Each

Settings

Collection:	
Counter Variable Name:	counter
Batch Size:	1
Root Message Variable Name:	rootMessage



Exercise solution: Processing tradeoffs when using a For Each scope

MuleSoft

Pros

- Auditing and tracing on records is easier
- Easier and isolated error handling of failed inserts to the target DB

Cons

- Throughput limited by the For Each scope

Exercise 17-2: Respond to changing performance requirements

MuleSoft

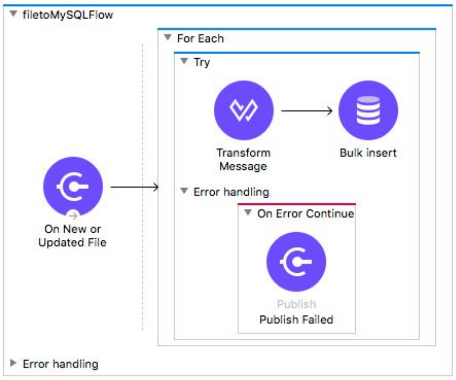
- Modify an integration solution architecture to respond to new performance goals and other requirement changes

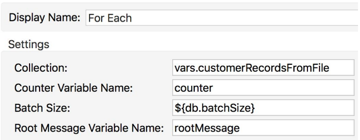
Exercise context: 6 months later… the file transfer use case requirements and constraints change

- The new source files containing recent flight activities for customers will still be uploaded at the same rate, **a few times a day**, to a specific **input** directory on the **file server**
- The size of each file has now **grown 100,000x** from 1000 records to **1M records** of customer data
- The Mule application must still quickly **process the file** and then send results to a **target MySQL database**
- The Mule application has been budgeted to deploy to **one 0.2 vCore CloudHub worker** (with 1 GB of heap memory)
- **Auditing** and **traceability** of each record is now deemed critical

Exercise solution

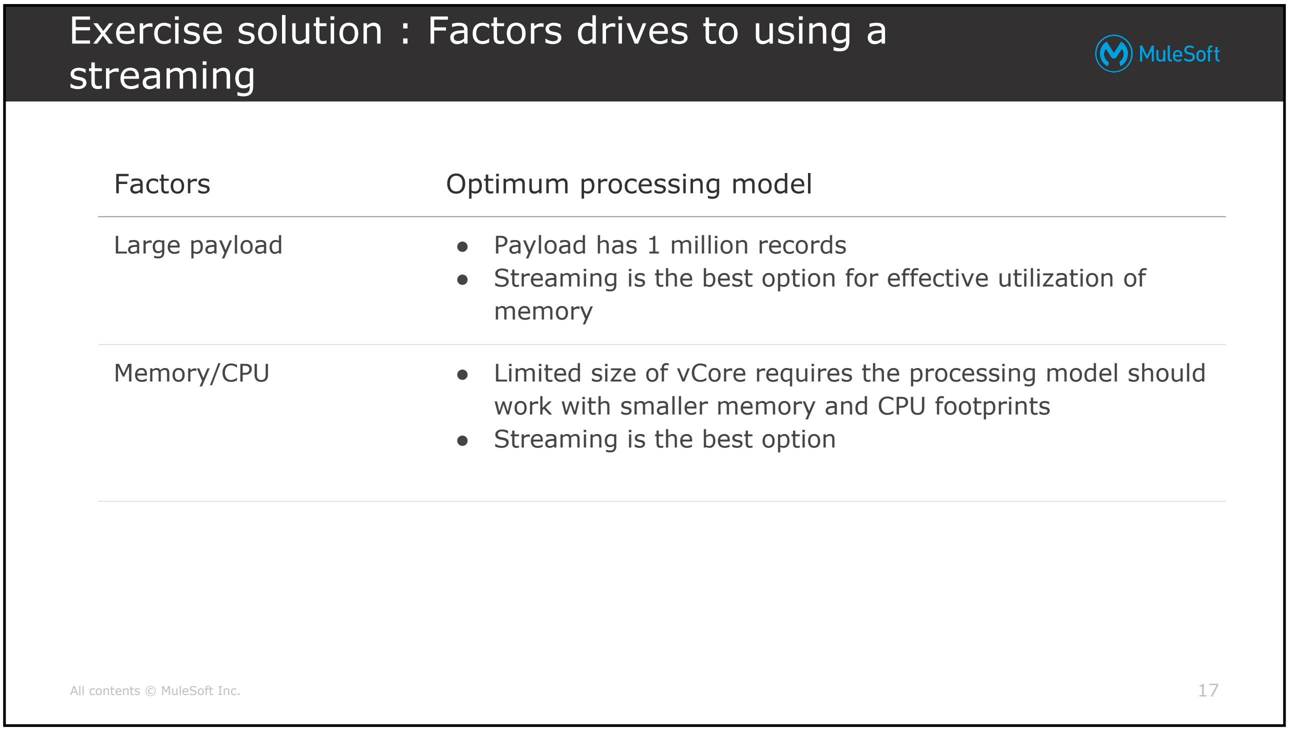
- A proposed processing model using a For Each scope configured to process the source file as streaming data
  - Each file is read by the flow as a **stream**
  - The For Each scope handles **batches** of records from the file
    - The batch size can be tuned by Ops as a property placeholder
  - In a Try Catch block, the stream is transformed using DataWeave
  - Records are then inserted into the target DB in a bulk insert using the saem batch size set by the For Each scope
  - In the On Error scope, failed batches of records are sent to the DLQ




Display Name: For Each

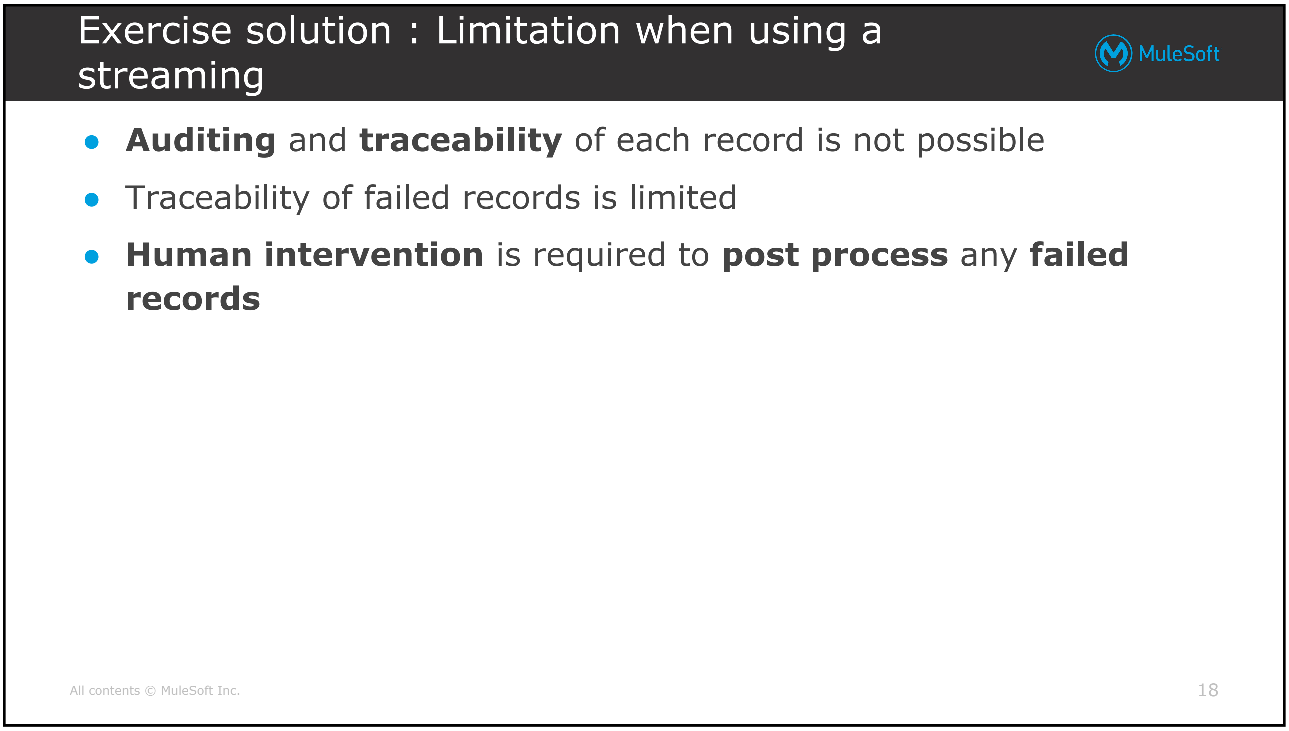
Settings

Collection:	vars.customerRecordsFromFile
Counter Variable Name:	counter
Batch Size:	${db.batchSize}
Root Message Variable Name:	rootMessage



Exercise solution : Factors drives to using a streaming

Factors	Optimum processing model
Large payload	● Payload has 1 million records ● Streaming is the best option for effective utilization of memory
Memory/CPU	● Limited size of vCore requires the processing model should work with smaller memory and CPU footprints ● Streaming is the best option

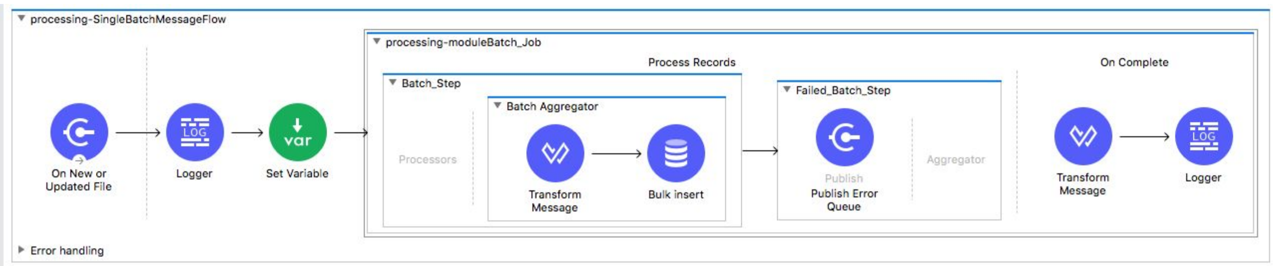


Exercise solution : Limitation when using a streaming

- **Auditing** and **traceability** of each record is not possible
- Traceability of failed records is limited
- **Human intervention** is required to **post process** any **failed records**

Exercise solution

- A different proposed processing model using a Batch Job scope
  - File is read as a stream in a Batch Job scope
  - The stream is transformed in a Batch Aggregator and then multiple records are inserted into the target DB in a single Bulk Insert operation
  - Failed records are sent to a JMS server to a dead letter queue (DLQ)



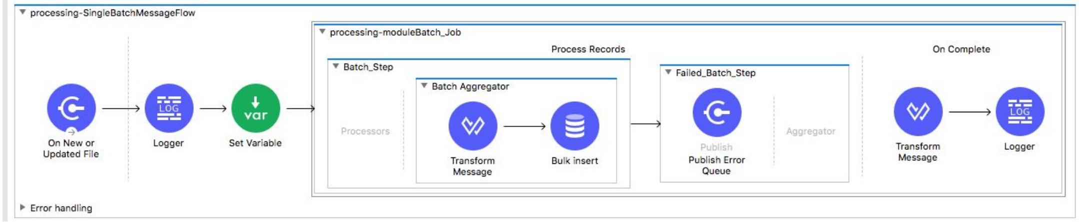
Exercise solution: Processing tradeoffs when using a Batch Job scope

Pros

- Audit and tracing per record
- Auditing and tracing is easier with JMS
- Easier and isolated error handling of failed records

Cons

- Uses internal queues
  - May cause out of memory errors with large payloads and high throughput



Exercise 17-3: Respond to changing reliability requirements

- Modify an integration solution to add real time data enrichment
- Modify an integration solution architecture to respond to new reliability SLA goals and other requirement changes

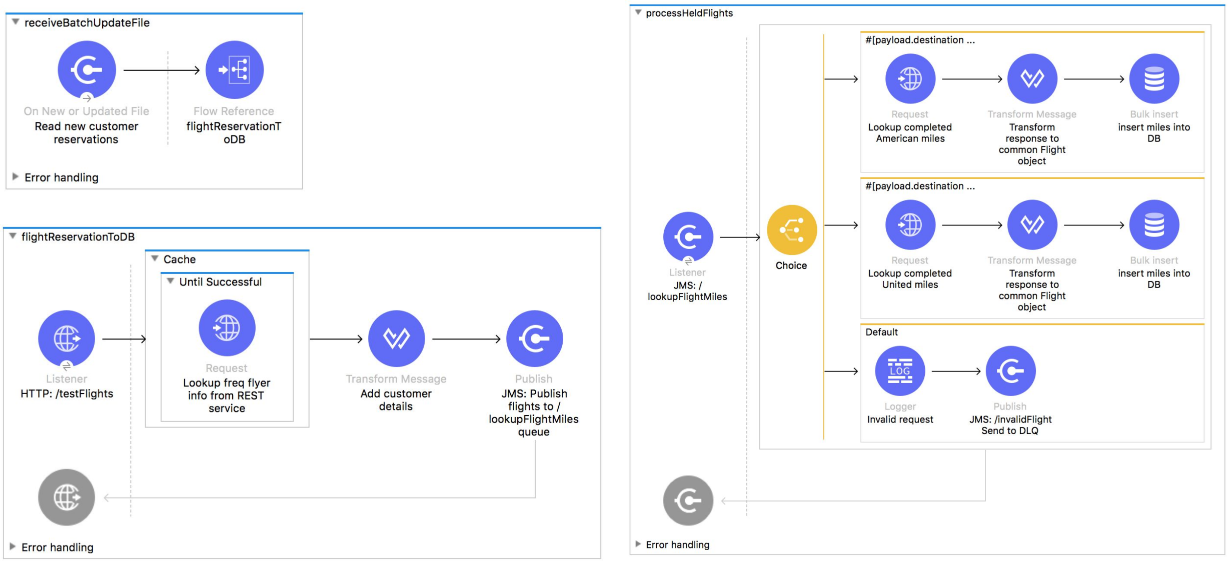
Exercise context: 6 months later... file transfer use case requirements and constraints change again

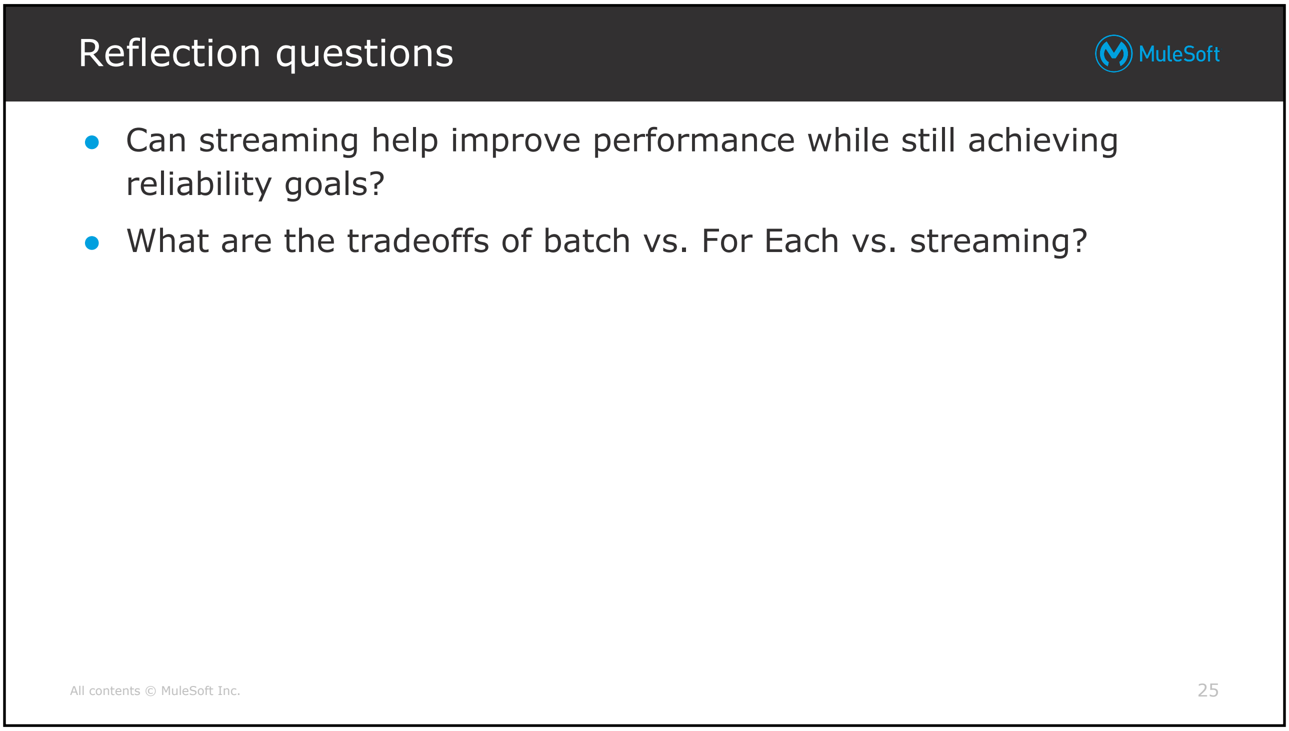
- A new requirement has been added to add additional flight details about a customers frequent flyer program to the target database records
- It has been observed that at random times the **server** to which the Mule application is deployed **shuts downs while processing files**
  - This results in service outages and lost data
  - Some records have also been processed twice when the previous file is uploaded again
- You need to decide the best way to add additional **reliability** to the current implementation

Exercise steps

- Decide how to call out to existing web services to enrich the flight reservations received in the uploaded file
- Decide how to handle server crashes
  - When should processed files be deleted from the input directory of the file server?
  - Should processed files be moved to a different directory, and if so, when?
  - How can duplicate processing of the same data be avoided?

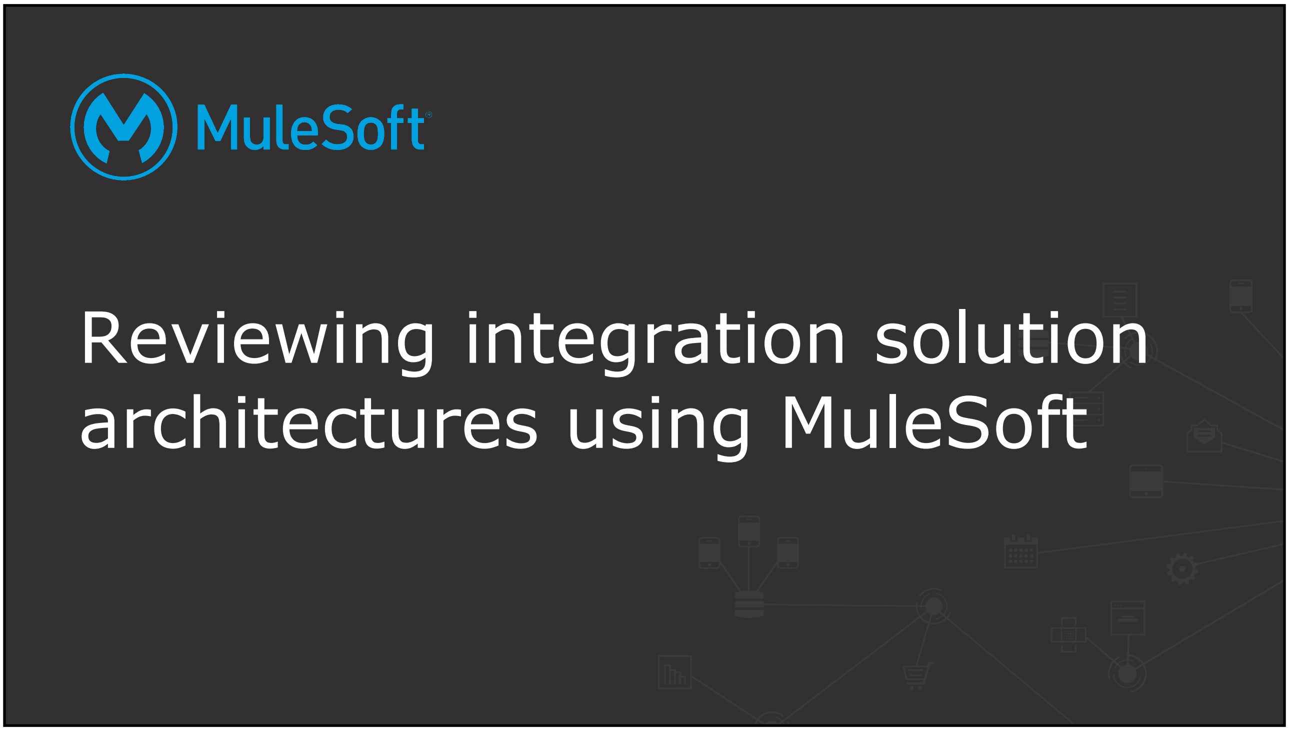
Exercise solution





Reflection questions

- Can streaming help improve performance while still achieving reliability goals?
- What are the tradeoffs of batch vs. For Each vs. streaming?



Reviewing integration solution architectures using MuleSoft

Apply the course goals to a new use case

- Work with technical and non-technical stakeholders to translate **functional and non-functional requirements** into integration interfaces and implementations
- Create the **high-level design of an integration solution** and guide implementation teams on the choice of Mule Components and patterns to use in the detailed design and implementation
- Design Mule applications for any of the **available deployment options of the Anypoint Platform runtime plane**
- Apply standard development methods covering the full development lifecycle (project preparation, analysis, design, development, testing, deployment, and support) to ensure solution quality

Apply the course goals to a new use case

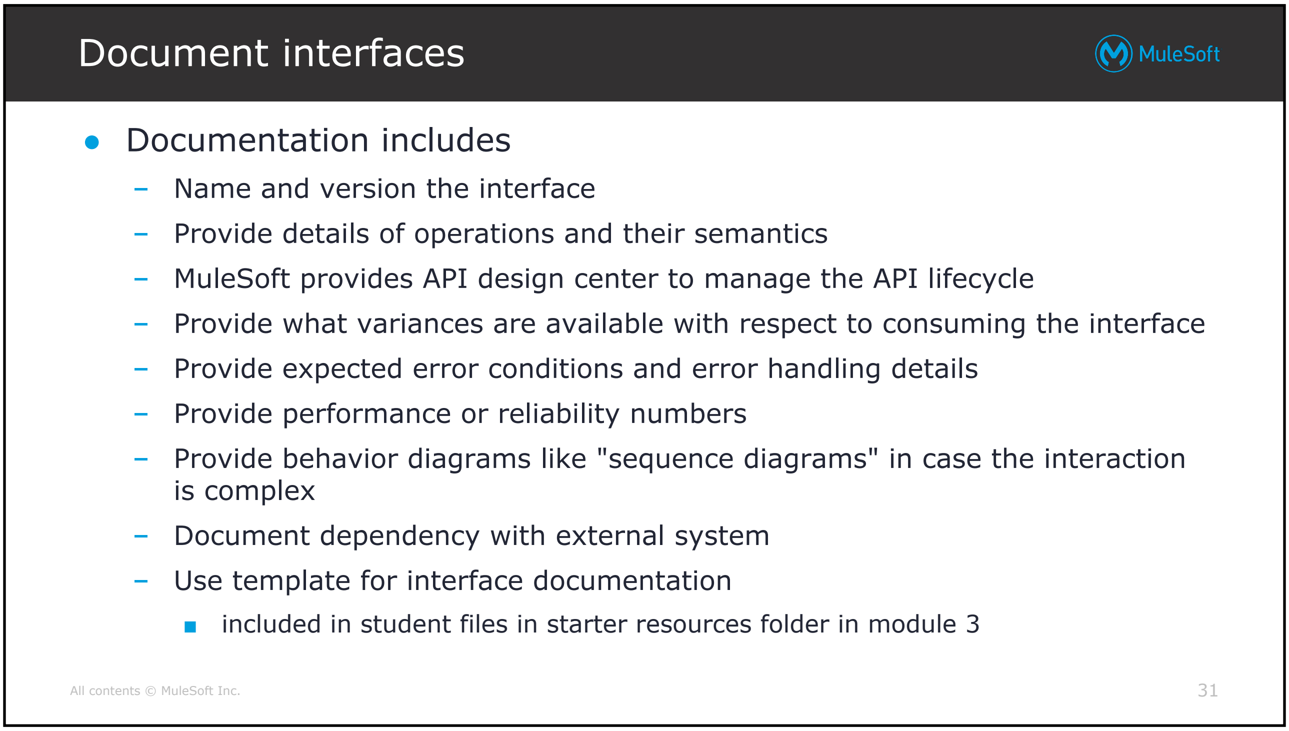
- Design reusable **assets, components, standards, frameworks**, and processes to support and facilitate API and integration projects
- Select the **deployment approach and configuration** of the Anypoint Platform with any of the available deployment options (MuleSoft-hosted or customer-hosted control plane and runtime plane)
- Design and be responsible for the **technical quality**, **governance** (ensuring compliance), and **operationalization** of the integration solution
- Advise technical teams on **performance, scalability, reliability, monitoring and other operational concerns** of the integration solution on Anypoint Platform

Key job tasks

- Refine architecture views in integration architectures
- Document interfaces in integration architectures
- Document key assumptions, decisions, and tradeoffs in integration architectures
- Document NFRs and SLAs in integration architectures
- Document best practices in integration architectures

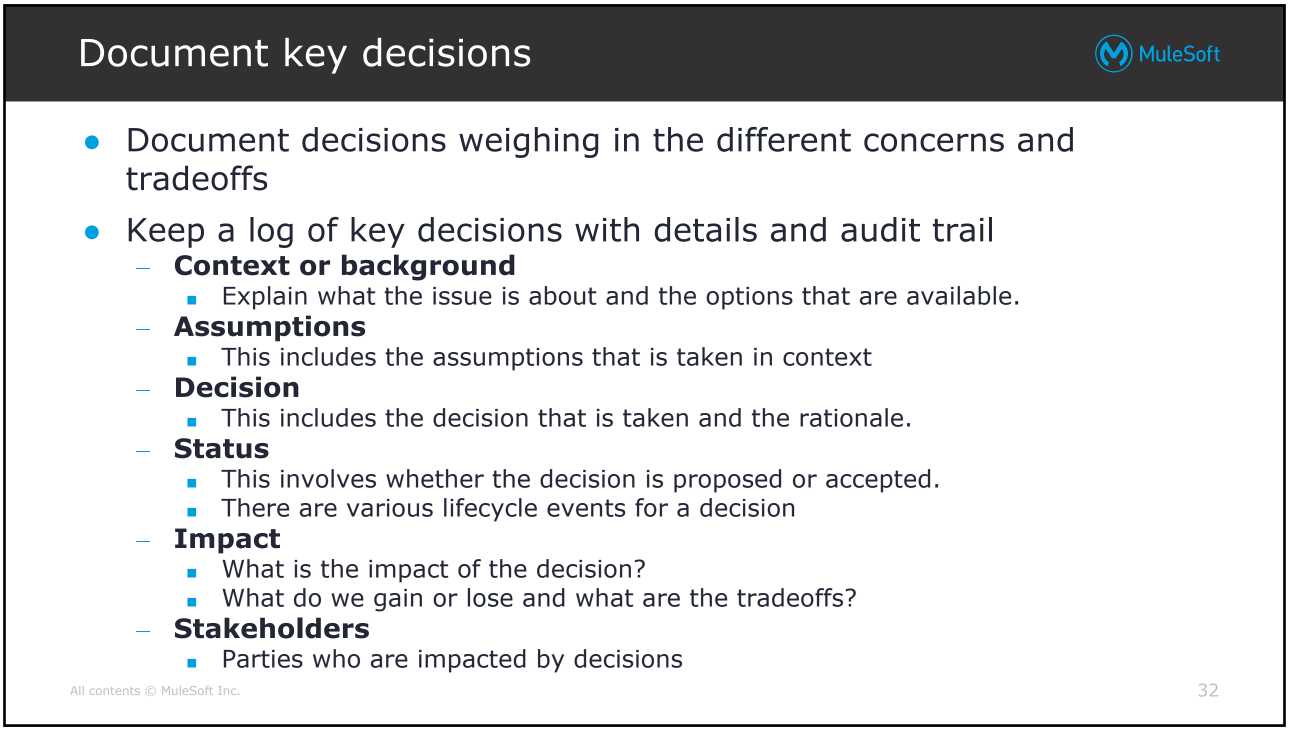
Refining architecture viewpoints for Mule applications

- **Deployment diagrams** are typically associated with use case realizations in the **Physical View** of the system
  - They document service and deployment models of the integration solution, including where and how MuleSoft tools and Anypoint Platform are used
- Sequence diagrams are typically associated with use case realizations of the Logical View of the system
  - They document the exchange of information across different systems and stakeholders
- Data transformation models describe how to connect external systems together
  - May involve common data models (CDMs) to promote reuse and decouple systems



Document interfaces

- Documentation includes
  - Name and version the interface
  - Provide details of operations and their semantics
  - MuleSoft provides API design center to manage the API lifecycle
  - Provide what variances are available with respect to consuming the interface
  - Provide expected error conditions and error handling details
  - Provide performance or reliability numbers
  - Provide behavior diagrams like "sequence diagrams" in case the interaction is complex
  - Document dependency with external system
  - Use template for interface documentation
    - included in student files in starter resources folder in module 3



Document key decisions

- Document decisions weighing in the different concerns and tradeoffs
- Keep a log of key decisions with details and audit trail
  - **Context or background**
    - Explain what the issue is about and the options that are available.
  - **Assumptions**
    - This includes the assumptions that is taken in context
  - **Decision**
    - This includes the decision that is taken and the rationale.
  - **Status**
    - This involves whether the decision is proposed or accepted.
    - There are various lifecycle events for a decision
  - **Impact**
    - What is the impact of the decision?
    - What do we gain or lose and what are the tradeoffs?
  - **Stakeholders**
    - Parties who are impacted by decisions

Document NFRs and SLA

- The design approach should consider the nonfunctional requirements and related cost
- Various factors like like **performance compliance, PCI and governance requirements**, etc. that impact the design are visible to the different stakeholders
- Different architectural components providing a service have SLA defined in their interface documentation
- The NFRs should show how different nonfunctional requirements are satisfied
- It helps different stakeholders like quality engineers and operational engineers to plan in advance various tasks like load testing, operational alerts, etc.

Document best practices

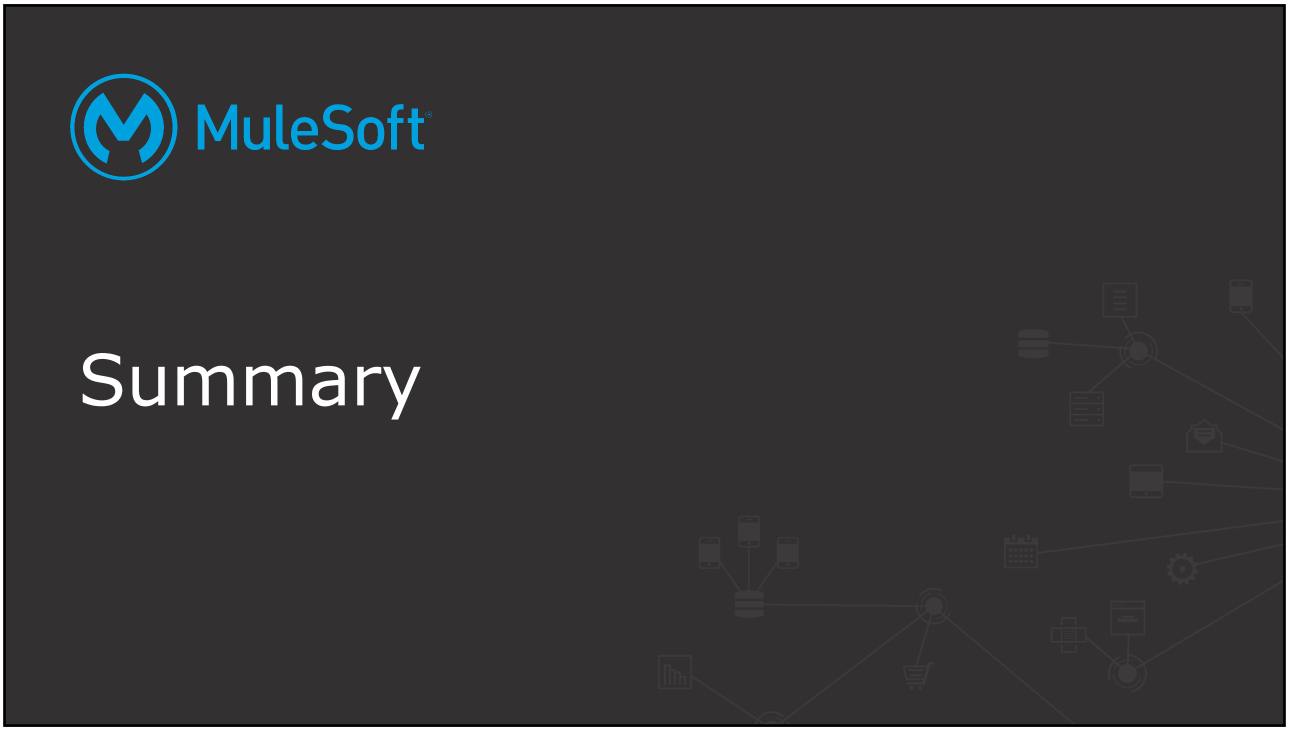
- Use some sort of standard templating around how the architecture document, interface design document should be produced
- Promote reusability of assets from Anypoint Exchange
- Document library and shared resources
- Standardized CI/CD for enterprise integration applications

Exercise 17-4: Document the architecture for the new use case

- Document the architecture for a new use case

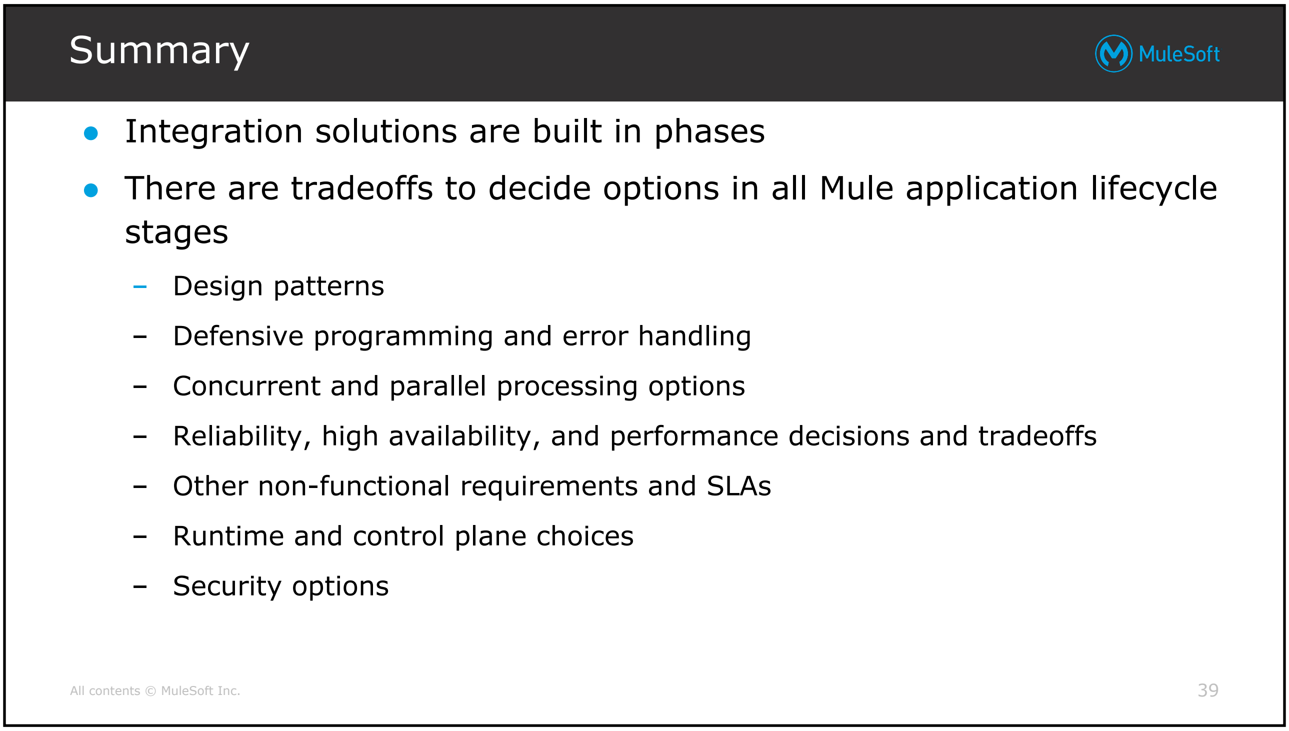
Exercise steps

- Use your mocked design from the previous exercises to fill in an architecture document
- Use the starter template doc in the Module 17 starter folder
- Begin filling in sections of the architecture document
- Review a completed architecture solution with the instructor and discuss the decisions with the group



MuleSoft

Summary



Summary

MuleSoft

- Integration solutions are built in phases
- There are tradeoffs to decide options in all Mule application lifecycle stages
  - Design patterns
  - Defensive programming and error handling
  - Concurrent and parallel processing options
  - Reliability, high availability, and performance decisions and tradeoffs
  - Other non-functional requirements and SLAs
  - Runtime and control plane choices
  - Security options