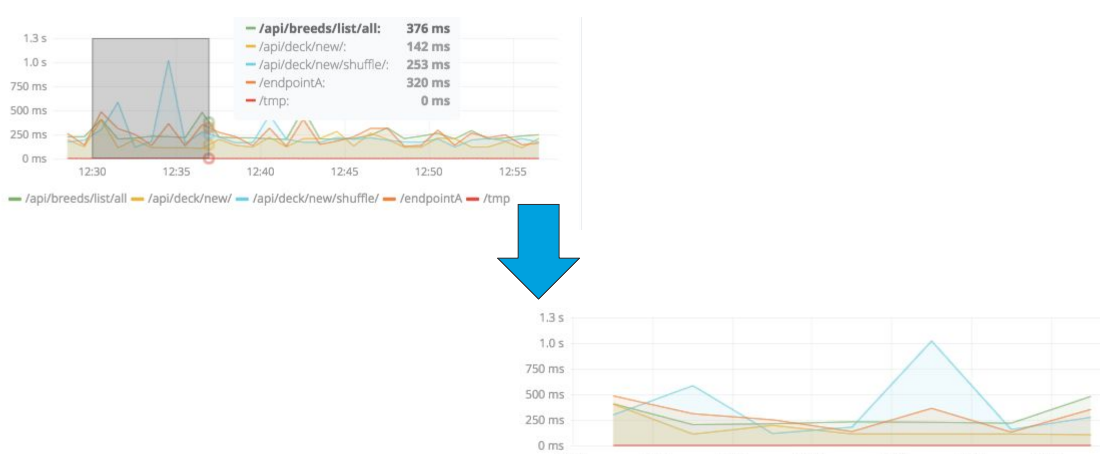
Module 14: Optimizing the Performance of Deployed Mule Applications

Goal



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

- Clarify performance goals for Mule applications
- Balance performance goals with reliability and HA goals
- Identify the need for performance optimization and associated trade-offs
- Identify ways to search for and locate performance bottlenecks
- Plan, architect, design, and implement for performance
- Identify ways to measure performance
- Identify methods and best practices to performance tune Mule applications and Mule runtimes

Identify why and when to optimize Mule application performance

Why and when performance matters in a Mule application

- Some Mule applications may experience performance issues
- Performance goals are often at least partially dictated by service level agreements (SLAs)
- To optimize performance, key performance indicators must be identified and agreed upon by key stakeholders

Iterating to define and achieve performance goals

- It is important to not over-optimize performance goals
  - Performance optimization goals require additional time, money, and resources
  - Goals need to be clearly stated and agreed upon, as well as service level agreements
- It is typically recommended to tune performance in logical stages, driven by real data, tests, and tools
  - To meet, but not necessarily overly excede performance goals

Exercise 14-1: The need for performance optimization

- Identify performance goals that may be important to particular types of Mule applications
- Identify how performance goals balance with other reliability goals
- Identify key performance indicators to measure Mule application performance objectives
- Decide which types of performance optimizations are required for a particular type of Mule application and its associated business use cases

Exercise steps

- Discuss and answer these questions:
  - Why do Mule applications need to optimize performance?
  - What are the key performance indicators for Mule application?
  - Is optimizing performance mandatory for all Mule applications?

Exercise solution

- A Mule application may experience performance issues when
  - Processing large numbers of messages
  - Processing large messages (payload size > 1MB)
  - Transforming messages extensively
  - Suffering from network latency or other performance bottlenecks

Exercise solution

- Key performance indicators for integration Mule applications
  - Throughput
  - Response time/latency
  - Capacity (number of concurrently processed messages)

Exercise solution

MuleSoft

- Performance optimization is not always required for all Mule applications
  - Not unless experiencing performance issue
  - Use Mule runtime defaults as long as possible

"Premature optimization is the root of all evil" - Donald Knuth

How to identify performance bottlenecks

MuleSoft

- Log analysis
  - Application log
    - Includes all log events log from the application
    - Log level govern by log setting in application or Mule runtime
  - System log
    - Includes all log events from Mule runtime
    - Log level can not govern however additional logging can enable

How to identify performance bottlenecks

- Application performance monitoring tools, including
  - The Anypoint Platform dashboards, VisualVM/JConsole, or similar
  - Commercial tools like AppDynamics, New Relic, etc.
- Monitor and manage performance and availability of Mule apps
- Detect and diagnose complex Mule application performance problems to meet SLAs
- Measure key performance indicators
- Monitoring is possible at different levels depending on each tool's capability

Profiling Mule applications to identify performance bottlenecks

- Application profiling in Mule applications is often performed for two reasons
  - Memory issues
    - To detect memory leaks or excessive load situations
    - A Java heap memory dump can help to analyze these types of issues
  - Application unresponsiveness
    - To detect blocked threads, long-running threads, and waiting threads related to the Mule application
    - Thread dumps can help analyze the issue

Planning for performance tuning

- Before starting performance tuning, verify the Mule application already functions as expected
- Guarantee a stable performance testing environment
  - Guarantee all external systems work within predefined SLAs
  - Or mock expected data and simulate external input SLAs, and external response SLAs accordingly
  - Guarantee the testing tools and environment do not introduce additional load that will skew or bias collected data and observations
- Select appropriate tools for performance measurements
- Make sure **performance KPIs** are **clearly defined** and **agreed** by stakeholders

Example: KPIs to be measured to performance tuning Mule applications

- Average and worst case payload size
- Processing latency and CPU load at each Mule component
- Time spent between each network hop
- Time spent between Mule components
- Throughput
  - Requests or transactions per unit of time, e.g., 100 requests/second
- Latency or response time
  - Unit of time at a given load, e.g., 100 milliseconds mean +/- 20 milliseconds standard deviation at 100 requests/second

Example: Tools to measure performance

- Advanced Rest Client, SoapUI, JMeter to generate load
- Profiling tool such as YourKit or VisualVM to identify symptoms
- Other larger load testers to test performance at expected production scales
- Anypoint Monitoring
- MUnit to test individual components or parts of a flow

Example: Monitoring traffic and KPIs

- Common tools include
  - Anypoint Monitoring, Anypoint Visualizer, and other Anypoint dashboards
  - AppDynamics
  - New Relic
  - Nagios
  - Spunk or ELK
  - Zipkin or Jaeger

Architecting for Performance

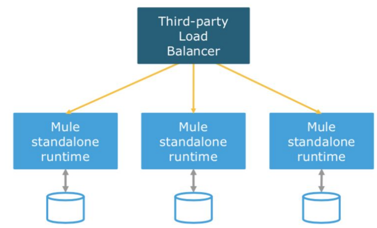
Architecting for performance

MuleSoft

- Particular performance goals may be achieved using Mule runtime and Mule application **scaling options**
  - Vertical scaling
    - Scale Up (for example, increase the vCore size on CloudHub)
    - Provide more resources (CPU cores, RAM) on each machine
    - Main use case: performance
  - Horizontal scaling
    - Scale out
    - Process on multiple machines
    - Improves both performance and availability
    - Load balancing and/or clustering need to be decided

HTTP request load balancing for performance

- Multiple Mule runtimes execute the same Mule application(s)
- When not clustered, there is no data sharing/synchronization between Mule runtimes and Mule applications
  - Could potentially lead to duplicate message processing or lost messages
  - Using a shared datastore is possible
- Messages must be distributes/load balanced
  - Requires third party products (on-prem only)
  - Deployed apps on multiple workers are load balanced in CloudHub



Clustering for performance

- A performance profile can be configured inside the Mule application or Mule runtime cluster for high performance
  - Disabled by default
- Configure a Mule runtime cluster for performance by setting the storeprofile value in the mule-cluster.properties or wrapper.conf file

```
mule.cluster.storeprofile=performance
```

- Or override per Mule application in a configuration global element

```
<mule>
    <configuration>
        <cluster:cluster-config>
            <cluster:performance-store-profile/>
        </cluster:cluster-config>
    </configuration>
</mule>
```

Effects of the performance profile

- Setting the performance profile has two effects
  - Disables distributed VM queues, using local VM queues instead to prevent data serialization/deserialization and distribution in the shared data grid
  - Implements node local in-memory object stores instead of shared memory grid architecture (Hazelcast) to avoid replication
- It is a common misconception that applications always perform better in a cluster!

Reviewing features of clustering and load balancing

- Both clustering and load balancing in active-active scenarios have pros and cons

	Clustering	Load Balancing
Pros	• Shared, distributed memory • Ideal for HA scenarios • Built-in load balancing for VM queues • Built into Mule	• Easy to set up • No performance overhead due to latency or data replication • Configurable load balancing algorithms (round-robin, IP sticky, load-based, etc.)
Cons	• Performance overhead due to latency and data replication • Not supported by CloudHub • Requires 3rd-party product to achieve HTTP load balancing	• Requires third-party product • No data synchronization • Manage idempotency programmatically

Protocols

- HTTP vs HTTPS
  - The latency difference between HTTP and HTTPS requests are only about 0.2 to 0.4ms more than HTTP requests (Mule 3.* stats), after the initial TLS handshake
- TLS
  - Use latest version of TLS 1.2.
    - Older version of TLS has poorer performance and vulnerabilities
- VM vs JMS
  - VM in memory protocol performs better than JMS
  - Transient queue performs better than persistent in VM and JMS

Sizing CloudHub workers to meet performance goals

How Mule applications are deployed to multiple CloudHub workers

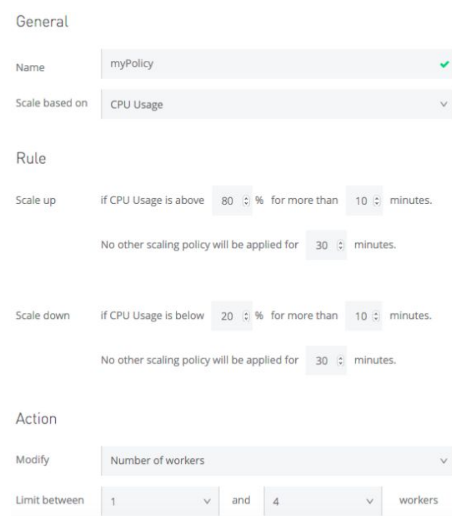
- CloudHub automatically distributes multiple workers for the same Mule application across two or more availability zones for maximum reliability
- When a Mule application is deployed to two or more workers
  - The HTTP load balancing service automatically distributes requests across all the CloudHub workers in an approximately **round-robin** manner
- A Mule application can be scaled out to a maximum of 8 workers or 16 vCores

Configuring autoscaling in CloudHub

Autoscaling in CloudHub

MuleSoft

- Runtime Manager provides an **autoscaling** feature
  - Only certain enterprise licenses support autoscaling
  - Must ask MuleSoft support to enable this feature
- Allows Mule runtimes to **scale** in response to CPU or Memory usage thresholds being exceeded
- Each autoscaling policy can decide to either
  - Increase or decrease the current Mule runtime vCore size (vertical scaling)
  - Increase or decrease the current number of Mule runtimes (horizontal scaling)



Restrictions of the Anypoint Platform autoscaling feature

MuleSoft

- Each autoscaling policy is only triggered every 30 minutes
- Trigger conditions are limited to memory or CPU usage

Other ways to design autoscaling policies

- Write your own scripts to deploy or undeploy Mule runtimes or CloudHub workers based on various triggering conditions
  - Perhaps using the Mule Maven plugin
  - Combine with other Anypoint Platform REST APIs
- Triggering conditions could be
  - Alerts (CPU, memory, ...)
  - Monitoring results (Anypoint Monitoring)
  - Behavior of requests/transactions submitted specifically for autoscaling purposes

Designing for Performance

Designing for performance

- Different Mule components have different performance behaviors
  - Synchronously
    - When a client produces a request, the client waits for a response from the consumer
    - Usually has better performance for moderate throughput and small payloads
    - Examples: HTTP Listener, Web Service Consumer, JMS Publish consume operation
  - Asynchronously
    - Request and response are separate interactions, and the response is optional
    - Producer does not waits for response from consumer
    - High throughput and large payload
    - Examples: JMS Publish operation, Async scope

Performance features of common Mule components

- Batch
  - Processes messages in batches
  - Useful for streaming input or synchronizing/processing collections of data in batches of records at a time
  - High Throughput
- Scheduler
  - Scheduler runs periodically for new data
  - Useful for batch processing
    - Use a short scheduling interval to keep the source and target in sync
- Event/messaging queues
  - Events/messages decouple data producers from data consumers
  - Useful for real time data integration

Design for performance

- Streaming
  - Data can be read multiple times or accessed randomly from stream using the DataWeave expression language
  - Data can be sent to multiple places without the need to cache that data in memory first.
  - Can transparently process larger-than-memory data
  - SaaS providers often have restrictions on accepting streaming input. Rather, use streaming batch processing when writing to a file such as CSV, JSON, or XML.
  - Slows the pace at which it processes transactions

Exercise 14-2: Identify the best processing model for optimizing performance

- Identify ways to sync data between an enterprise database system and Salesforce
- Make optimal design decisions for three different data synchronization scenarios
  - Daily periodic data synchronization
  - High volume data synchronization
  - Real time lower latency data synchronization of smaller volumes of data

Exercise scenarios: Identify the best processing model for optimizing performance

- The application has to sync data between a MySQL source database and a target Salesforce system
- Design a flow for 3 different scenarios and explain merit for your decision
- Three differents scenarios
  - Scenario 1 - Sync 500 records a day with periodic sync
  - Scenario 2 - Sync more than 10,000 records within 5 minutes
  - Scenario 3 - Real time sync between MySQL and salesforce (less than 10 records a minute)

Exercise step: Design flows to meet the Scenario 1 requirements

- Scenario 1: Sync 500 records a day with periodic synchronization
  - Identify expected and required workload for the scenario
  - Does the scenario need real time processing or can periodic processing suffice?
    - Identify how much latency is allowed before new database records must be synchronized to Salesforce
  - Identify message source(s) for flow(s) that best meet workload and processing requirement (real time, periodic, etc.)
  - Evaluate various processing options for scenarios
    - Async message queues, batch, schedulers, etc.
  - Evaluate and analyze if the expected and required workloads justify the use of the proposed processing model

Exercise solution: Design flows to synchronize 500 records a day with periodic synchronization

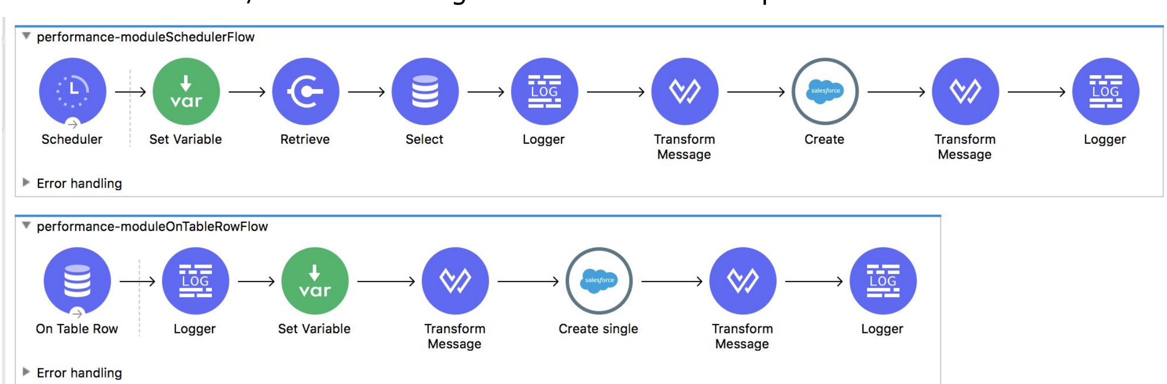
MuleSoft

- This is low throughput data sync scenario and data sync is not required in real time
- An appropriate processing model is
  - Scheduled polling of the source database
    - Works for periodic sync of data
    - Low to high throughput of data
    - Smaller number of records should not require streaming data
      - All 500 records can be transformed and processed at once in the Mule app's memory
  - Event triggered by source database changes
    - Allows real time data sync with moderate throughput of data

Exercise solution: Sync 500 records a day with periodic sync

MuleSoft

- Periodically select from a database table, then transform and sync (write) each retrieved row/record to Salesforce in a bulk operation, and process the results
- Also poll the same database table, then transform and sync (write) each retrieved row/record in a single Salesforce Create operation



Exercise step: Design flows to meet the Scenario 2 requirements

MuleSoft

- Scenario 2 - Sync more than 10,000 records within 5 minutes
  - Identify expected and required workload for the scenario
  - Does the scenario need real time processing or can periodic processing suffice?
    - Identify how much latency is allowed before new database records must be synchronized to Salesforce
  - Identify message source(s) for flow(s) that best meet workload and processing requirement (real time, periodic, etc.)
  - Evaluate various processing options for scenarios
    - Async message queues, batch, schedulers, etc.
  - Evaluate and analyze if the expected and required workloads justify the use of the proposed processing model

Exercise solution: Sync more than 10,000 records within 5 minutes

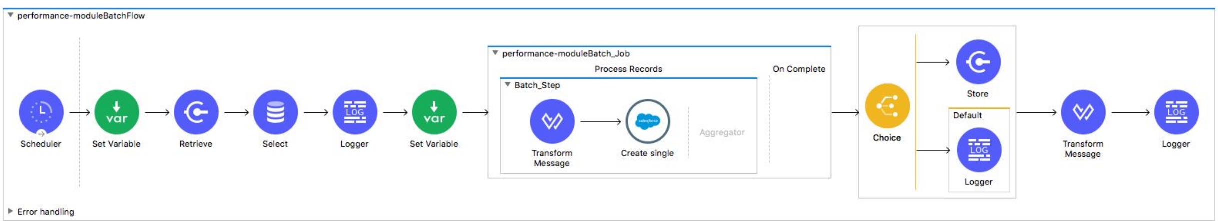
MuleSoft

- This is a high throughput data synchronization scenario and data synchronization is required in batches
- The likely processing model is to combine
  - Scheduler
    - Works for periodic high frequency sync (every 30 secs)
    - Can handle high data throughput (can handle large streaming data)
  - Batch
    - Can handle high data throughput (can handle large streaming data)
    - Records can be processed in parallel in multiple threads
    - Useful when data synchronization is required to complete as fast as possible

Scenario 2 - Sync more than 10,000 records within 5 minutes

MuleSoft

- Retrieve the last processed record id from an object store
- Periodically select from a database table
- Transform and sync (write) batches of database records in a batch job
- The object store is used to only create new records in Salesforce, to avoid duplicate work
  - New Salesforce record IDs are then stored in the object store



Exercise step: Design flows to meet the Scenario 3 requirements

MuleSoft

- Real time sync between a source database and Salesforce
  - Identify expected and required workload for the scenario
  - Does the scenario need real time processing or can periodic processing suffice?
    - Identify how much latency is allowed before new database records must be synchronized to Salesforce
  - Identify message source(s) for flow(s) that best meet workload and processing requirement (real time, periodic, etc.)
  - Evaluate various processing options for scenarios
    - Async message queues, batch, schedulers, etc.
  - Evaluate and analyze if the expected and required workloads justify the use of the proposed processing model

Exercise solution: Design flows for real time sync between a source database and Salesforce

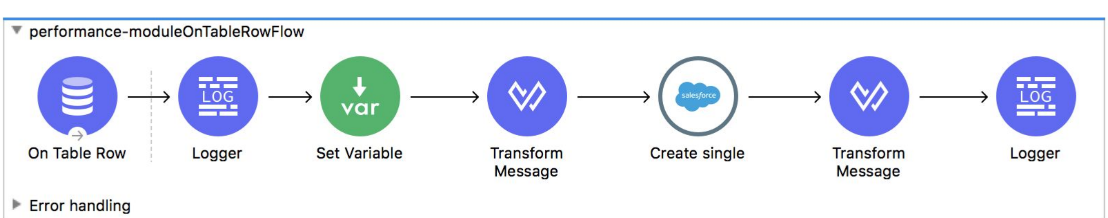
MuleSoft

- This is low throughput data sync scenario and data sync is required in real time
- An appropriate processing model is
  - Let the database trigger new events
    - Low throughput of data
    - When data sync is required in real time

Scenario 3 - Real time sync between salesforce and MySQL

MuleSoft

- Use the On Table Row event source to poll for new rows in the source database table
  - This is not true real time, but the polling interval can be set to a low value, and a watermark and avoid duplicate processing of existing rows
  - Only new rows are sent over the network to this Mule flow
  - Individual rows are processed one at a time then created in the target Salesforce system



Implementing Mule applications to meet specific performance goals

Implementing for performance

MuleSoft

- Transformation
  - Avoid unnecessary conversions to Java data types, and work with data directly
  - Let the Mule handle streaming for you
  - Do not convert binary data types to String, Byte[] or InputStream unless you are performing a direct integration with custom Java code.
  - DataWeave uses default memory buffer of 1572864 bytes, if transformation need excess memory then it uses system's hard disk as a buffer
  - DataWeave buffer size can changed by setting system property `com.mulesoft.dw.buffersize` and assign it the number (in bytes)

Implementing for performance

- Scatter-Gather
  - Parallel execution of routes greatly increases the efficiency of your application
  - Max Concurrency sets concurrency for parallel execution of scope

Implementing for performance

- Logging
  - Defaults to async for performance
  - Can log errors synchronously and the info,debug level asynchronously
  - GC log can be captured

```
<!-- log4j2.xml in MULE_HOME/conf for Mule classes -->
    <AsyncLogger name="org.mule" level="INFO"/>
    <AsyncLogger name="com.mulesoft" level="INFO"/>
<!-- wrapper.conf in MULE_HOME/conf →
   wrapper.java.additional.<n>=-Xloggc:%MULE_HOME%/logs/gc.log
```

Implementing for performance

- Network latency
  - Large payload
    - Compress large payload (gzip)
    - Can use the Compression module
  - Caching
    - Cache responses

Measuring Mule application performance

How to measure Mule application performance

MuleSoft

- First identify related KPIs
- Identify tools that can effectively and efficiently measure these KPIs
- Measure CPU and memory utilization of a Mule application
- Decide if performance optimization is required and the associated trade-offs
- Profile the Mule application then tune the Mule application to optimize resource utilization
- Continue measuring to validate performance goals are met

Exercise 14-3: Measure Mule application performance

MuleSoft

- Tune a Mule flow and a batch job to optimize performance
- Identify tools that can effectively and efficiently measure agreed upon KPIs
- Profile and tune a Mule application to optimize resource utilization
- Validate performance goals are met

Exercise overview

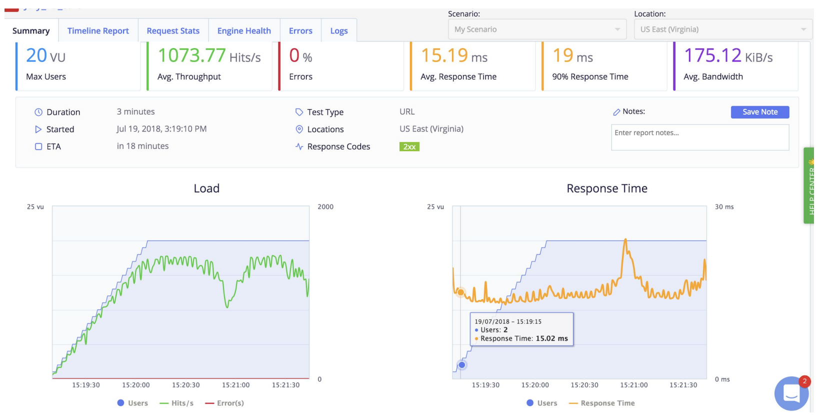
- Measure the key performance indicators identified in exercise 14-1
  - Identify tools useful for measuring KPI for application
  - Decide whether performance improvement is required for application
  - Identify CPU and memory utilization for demo app
  - Optimize resource utilization

Exercise steps: Measure performance

- Have you used any tools in past to measure the performance of system?
- Check whether a tool is able to measure the KPIs for the application
- Analyse KPI against SLA requirement
- Do you need performance improvement in application
- Does any monitoring tool on Anypoint platform will help to measure CPU and memory utilization for application
- Can we optimize resource utilization based on CPU and memory optimization
- Can we form a rule for resource optimization

Exercise solution: Measure performance

- Identify tools useful for measuring KPI for application
  - JMeter or BlazeMeter or any open source performance testing too



Exercise solution: Decide whether performance improvement is required

- Compare your KPI against your SLA
- If SLA is not met, then performance improvement is required

Exercise solution: Measure performance

MuleSoft

- Identify CPU and memory utilization for demo app
  - Anypoint Monitoring provides built-in dashboards
    - Overview, Inbound, Outbound, Performance, Failure, JVM and Infrastructure
  - Overview Dashboard provides below details for application -
    - Total Inbound Requests
    - Average Response Time Inbound
    - Total Outbound Requests
    - Average Response Time Inbound
    - CPU Utilization
    - Memory Utilization
    - Thread Count - Server
- Anypoint Monitoring docs
  - https://docs.mulesoft.com/monitoring/dashboards

Exercise solution: Measure performance

MuleSoft

- Optimize resource utilization
  - Identify CPU and memory utilization on 0.1 vCore(smallest vCore for CloudHub)
  - Deploy more than one instance of application for high availability
  - If CPU and memory utilization is above 70% then scale horizontally/vertically
  - Optimize vCores for application based on resource utilization
  - Auto scaling of CloudHub workers is option(only for special licensed customers)

Performance tuning Mule Applications

Performance tuning Mule applications

- Auto-tuning of thread pools in Mule runtime

Pool	Min	When?	Max	Increment
CPU_LITE	#cores	Mule startup	2 * #cores	1
CPU_INTENSIVE	#cores	Mule startup	2 * #cores	1
BLOCKING_IO	#cores	Mule startup	#cores + ((mem - 245760)/5129)	1
GRIZZLY (shared)	#cores	Deployment of first app using HTTP Listener	#cores + 1	1
GRIZZLY (dedicated)	#cores	Deployment of first app using HTTP Requester	#cores + 1	1

Performance tuning features and considerations using HTTP/S connectors

- HTTP connector
  - Uses HTTP persistent connections by default
  - Set keep-alive header true with HTTP 1.0 client
- HTTPS connector
  - Use latest version of TLS 1.2.
  - Older version of TLS has poorer performance and vulnerabilities

Performance tuning features and considerations using JMS connectors

- JMS Connector
  - Caching is on by default
    - Caches JMS sessions/consumers and producers
  - Use sessionCacheSize to adjust size of JMS cached session
  - Disable JMS message persistent at JMS server
  - Configure `numberOfConsumers` on JMS listener for high throughput or else sequential message processing
  - Configure ACK mode to meet your SLA (discuss in detailed transaction module)
  - Avoid durable subscriber and message filtering

Performance tuning features and considerations using JMS connectors in clusters

- Set the `primaryNodeOnly` to false for queue listeners so consumers on all cluster nodes can receive message **from** the **queue**
- Set `primaryNodeOnly` to false for topic listeners so all consumers can receive message from **topic with shared subscription**
- A shared subscription to a JMS topic ensures that the different replicas of the Mule application on different cluster nodes receive different JMS messages from the topic

Performance tuning features and considerations using a database connector

- Database connector
  - Caching is on by default
    - Caches db connections
  - Use bulk operation based DB connector for batch processing
  - Streaming
    - Enable to start processing large result sets
  - Max rows and fetch size
    - Example: max rows = 1000, fetch size = 200 **limits** the response to max 1000 rows, and the database will try to stream at most 200 rows at a time to the database connector (so in this case would require 5 separate network round trips)

Performance tuning features and considerations using a VM connector

- VM connector
  - Use for HA
  - For performance, prefer flow references within the same Mule app instead of VM endpoints

Performance tuning features and considerations using Batch jobs

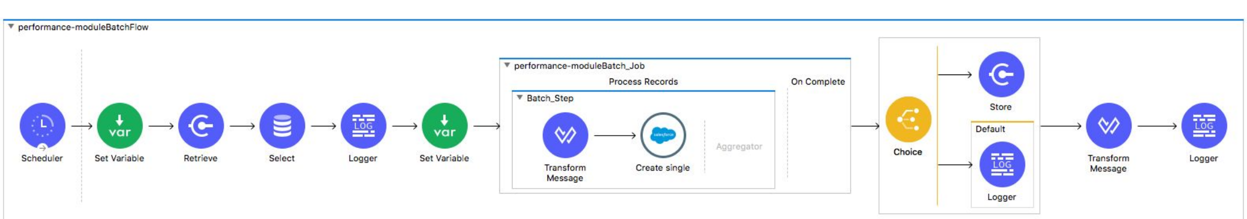
- Batch jobs
  - Default block size is 100
  - Max Concurrency sets concurrency for batch jobs (default is twice the available cores in CPU)
  - Run comparative batch sizes to find optimum concurrency for the use case. If you are processing 200 records with default block size of 100, to process 2 records in parallel, you need concurrency of 2.
  - Use Fixed Size Batch Aggregator to do bulk operation for supported connectors(db, salesforce)

Performance tuning features and considerations using a streaming Batch Aggregator scope

- Streaming Batch Aggregator
  - Receives all the records in the job instance without running out of memory
  - Random access of processing records is **not supported** for streaming aggregators
  - SaaS providers often have **restrictions** on accepting streaming input
    - Rather, use streaming batch processing when writing to a file such as CSV, JSON, or XML
  - Slows the pace at which it processes transactions

Exercise 14-4: Performance tune a Mule application

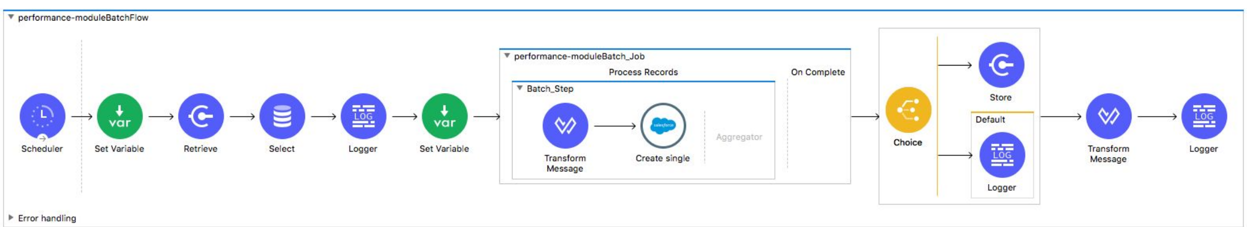
- Tune a Mule flow and a batch job to optimize performance



Exercise steps

MuleSoft

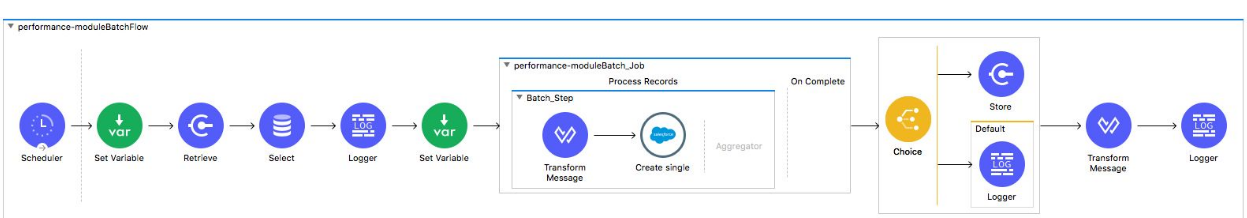
- In Scenario 2 of Exercise 14-2, you designed a flow to sync data between an enterprise database system MySQL and Salesforce
- Import project from student files exercise-14-4.jar



Exercise steps

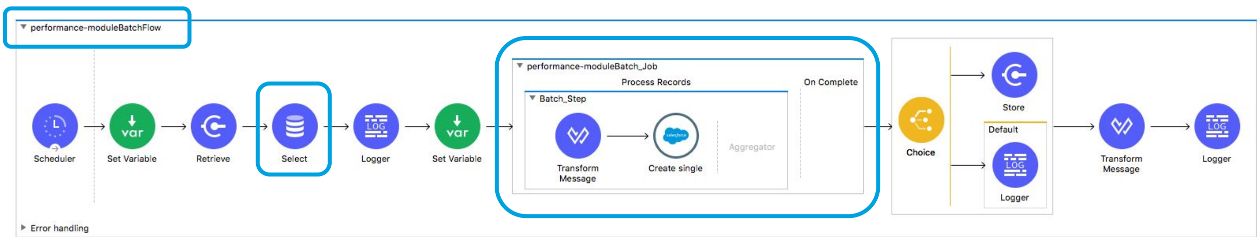
MuleSoft

- Tune the flow and its components for optimum performance
  - Select Mule components that may benefit from performance tuning
  - Identify tuning parameters for the selected Mule components
  - Analyze the trade-offs of each tuning parameter within the context of the entire Mule application



Exercise solution: Select Mule components that may benefit from performance tuning

- Flow
- DB Connector
- Batch Job
- Batch Aggregator



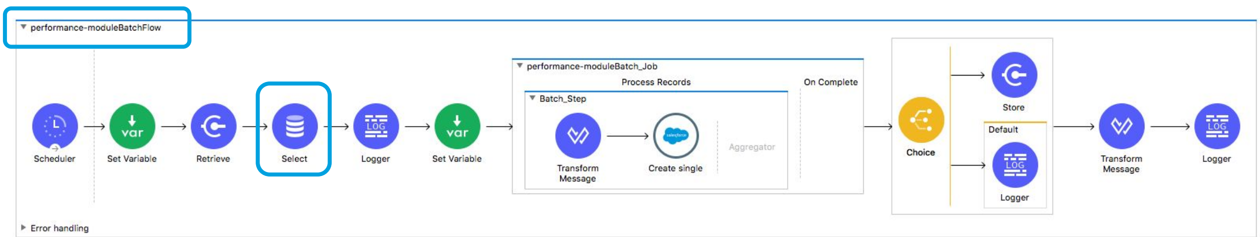
Exercise solution: Identify tuning parameters for the selected Mule components

- Flow or components
  - max concurrency
    - The max number of simultaneous invocations that component can receive an any given time
- DB Connector
  - streaming, max rows, fetch size
    - Limit total results from a DB SELECT, then fetch/stream them in smaller batches
- Batch Job
  - max concurrency, block size
- Batch Aggregator
  - streaming, fixed size batch

Exercise solution: Analyze tuning parameter trade-offs for flows and connectors

MuleSoft

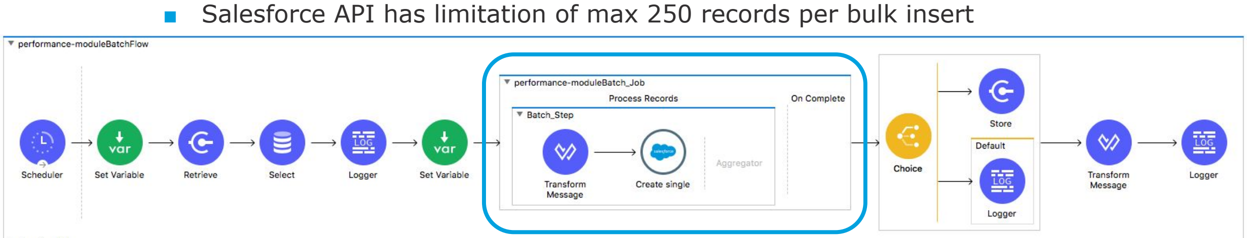
- Analyze the trade-offs of each tuning parameter within the context of the entire Mule application
  - Flow
    - Flow concurrency is not required for the batch job use case
  - DB Connector
    - Streaming is option, but not required, for this use case
    - Max rows with fetch size can create incremental batches



Exercise solution: Analyze tuning parameter trade-offs for batch jobs

MuleSoft

- Analyze the trade-offs of each tuning parameter within the context of the entire Mule application
  - Batch Job
    - Run comparative batch block sizes to find optimum concurrency for the use case
  - Batch Aggregator
    - Fixed size batch aggregator suits for use case
    - Streaming with SaaS provider has restriction
    - Salesforce API has limitation of max 250 records per bulk insert



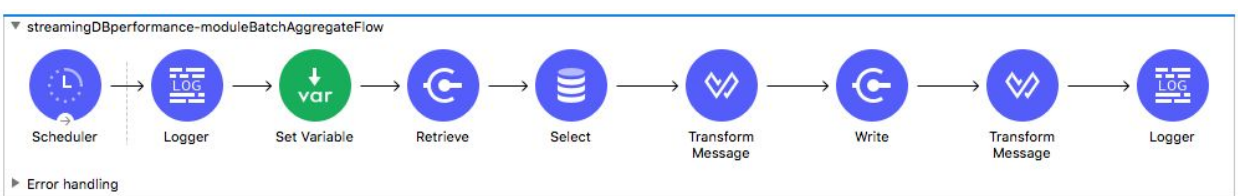
Exercise step: Performance tune a high volume Mule application

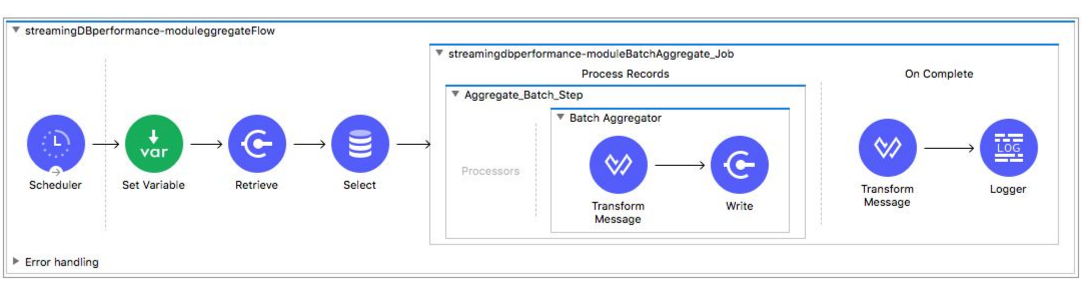
MuleSoft

- Design flow for write 1M records from enterprise database system MySQL into JSON file. Tune the flow/batch for optimum performance
  - Identify component to performance tuning
  - Identify tuning parameter for component
  - Understand trade-off of each tuning parameter

Exercise solution: Performance Tuning of Mule application

MuleSoft





Exercise solution: Performance Tuning of Mule application

MuleSoft

- Identify Mule application components that should be tuned for performance
  - Flow
  - DB Connector
  - Batch Job
  - Batch Aggregator
  - File

Exercise solution: Performance Tuning of Mule application

MuleSoft

- Identify tuning parameter for the Mule application's components
  - Flow
    - max concurrency
  - DB Connector
    - streaming, max rows, fetch size
  - Batch Job
    - max concurrency, block size
  - Batch Aggregator
    - streaming, fixed size batch
  - File
    - Lock

Exercise solution: Performance Tuning of Mule application

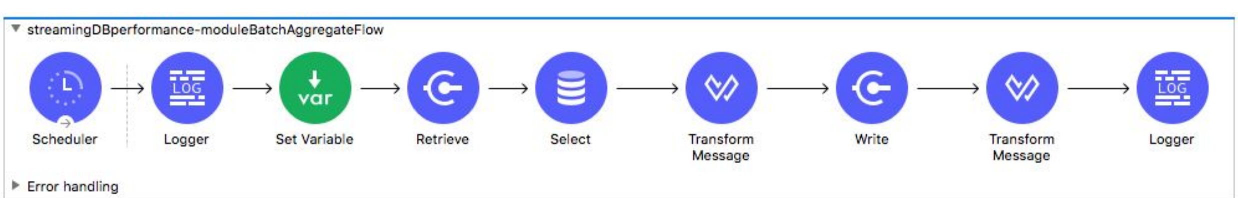
MuleSoft

- Understand the trade-offs of each tuning parameter
  - Flow
    - Flow concurrency is not required otherwise file write need synchronization
  - DB Connector
    - Streaming is required for use case
    - Max rows with fetch size can create stream, fetch size is mandatory for streaming
  - Batch Job
    - Run comparative batch block sizes to find optimum concurrency for the use case
  - Batch Aggregator
    - Streaming batch aggregator suits for use case as writing to file
  - File
    - File lock is disabled by default however writing to file using multiple thread may create deadlock on file.

Exercise solution: Performance Tuning of Mule application

MuleSoft

- Flow with DB streaming works best for the use case
  - Flow
    - Flow concurrency is not required otherwise file write need synchronization
  - DB Connector
    - Streaming is required for use case
    - Max rows with fetch size can create stream, fetch size is mandatory for streaming
    - Max rows = 1M and Fetch Size = 500 work best



Best practices for Batch processing

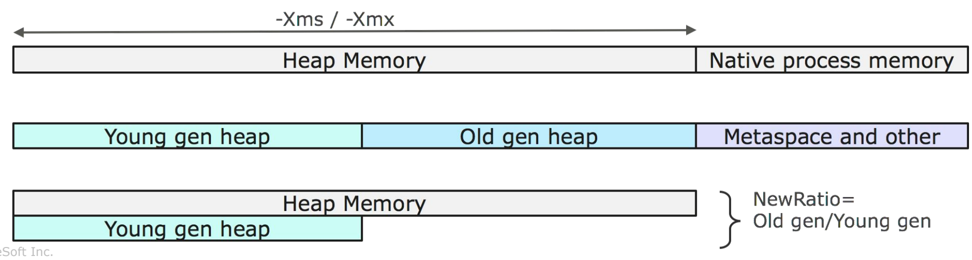
- Batch Aggregator streaming slows pace of transaction compare to non streaming aggregator however reduces memory footprints
- It advisable to have steaming on for Batch Aggregator if prior processing using streaming however it is not mandatory
- Batch processing improves performance with non repeatable streaming

Performance tuning a Mule Runtime

Performance tuning for customer-hosted Mule runtimes

MuleSoft

- Mule 4 uses JDK 1.8
  - Java Objects reside in **heap memory**
    - New objects are created in **young generation heap memory**
    - **Garbage collectors** move objects to **old generation heap memory**
  - **Metadata memory** resides natively outside the JVM heap memory
    - Used for class metadata such as class descriptors, methods, bytecodes, classes and classloaders
    - By default, uses all available memory on the host



Performance tuning for Mule(customer hosted)

MuleSoft

- The JVM that starts a Mule runtime can be tuned with standard JVM system properties
  - Can be set in the Mule runtime's wrapper.conf file, located in $MULE_HOME/conf
  - CloudHub uses HotSpot, the standard Oracle JVM
  - Set the initial and maximum heap size to the same value (default 1024 MB)
  - Set the `NewRatio` to set the ratio of the old and young generation heaps (default 1)

```
wrapper.java.additional.N=-XX:MetaspaceSize=256m
```

  - Set the **MaxMetaspaceSize** and **MetaspaceSize** to limit the amount of native memory used for class metadata (default 256 MB)

```
wrapper.java.additional.N=-XX:MetaspaceSize=256m

wrapper.java.additional.N=-XX:MaxMetaspaceSize=256m
```

Performance tuning for Mule(customer hosted)

- The JVM garbage collector (GC) is selected and tuned using standard JVM system properties
  - Can be set in the Mule runtime's wrapper.conf in $MULE_HOME/conf
- Parallel GC
  - Default GC for Oracle HotSpot JVM
  - Enough memory and large number of CPU
  - Optimized for throughput
- Concurrent-Mark-Sweep (CMS) GC
  - Uses more memory and CPU
  - Response time for application is crucial

Thread pools used by Mule runtimes

- Global thread pools are used to execute all flows for all Mule applications
- Three thread pools are used to execute individual event processors, based on the event processors execution type
  - CPU-intensive, CPU-light, or IO-intensive
- Some event processors internally flag their execution type
- Otherwise the Mule runtime automatically selects an execution type based on all the other event processors in the flow

Performance tuning Mule runtimes

- Threads and concurrency parameters can be tuned in the scheduler-pools.conf located in $MULE_HOME/conf
  - Only accessible in customer-hosted Mule runtimes
  - The **cpuLight** and **cpuIntensive** thread pool size is twice the number of vCores
  - The **IO** thread pool core size is the same as the number of cores
  - The **IO** thread pool max size is governed by the number of cores and maxMemory
    - `cores + ((mem - 245760) / 5120)` for 4 vCores worker with 1024 MB max memory ~ 152
- Except for rare edge cases, MuleSoft makes a strong recommendation to **not touch this file** and **stay with the default values**

Performance tuning individual flows and components

Component	Tuning Parameter
Flow	Max Concurrency
Batch Job	Max Concurrency, Batch Block Size
Batch Aggregator	Streaming, Aggregator Size(both are mutually exclusive)
Scatter-Gather	Max Concurrency

Performance tuning individual Mule connectors

MuleSoft

- Reconnection strategy applies to all connectors

Connector	Tuning Parameter
HTTP Connector	Use persistent connections
JMS Connector	Session cache size, cache consumer, cache producers, ack mode, max redelivery
JMS Listener	No of consumers, ack mode, max redelivery
DB Connector	Max pool size, Min pool size, acquire Increment, prepare statement cache size, transaction Isolation
DB Operation	Streaming, timeout, max rows, fetch size (required for streaming)

Summary

Summary

- Performance optimization can be performed at the architecture, design, development, runtime, and OS/VM levels
- Load balancing and clustering may help to improve performance, but not always
- The JVM instance can be tuned for Mule runtimes installed in customer-hosted infrastructure
- JVM tuning options include heap and metaspace sizes, memory tuning parameter, garbage collection choices
- Streaming adds latency to flow executions, but also reduces memory footprints
- Strong Recommendation : Go with DEFAULTS

Mule runtime tuning guidelines and resources

- Java tuning guides
  - https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/
  - https://docs.oracle.com/cd/E21764_01/web.1111/e13814/jvm_tuning.htm#PERFM151
  - https://www.slideshare.net/mulesoft/mule-runtime-performance-tuning