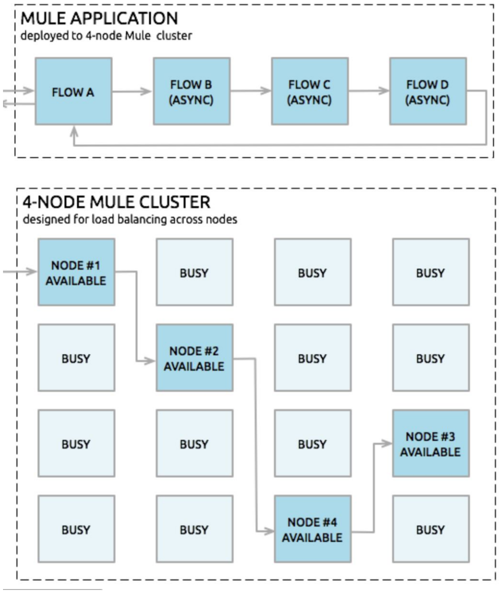
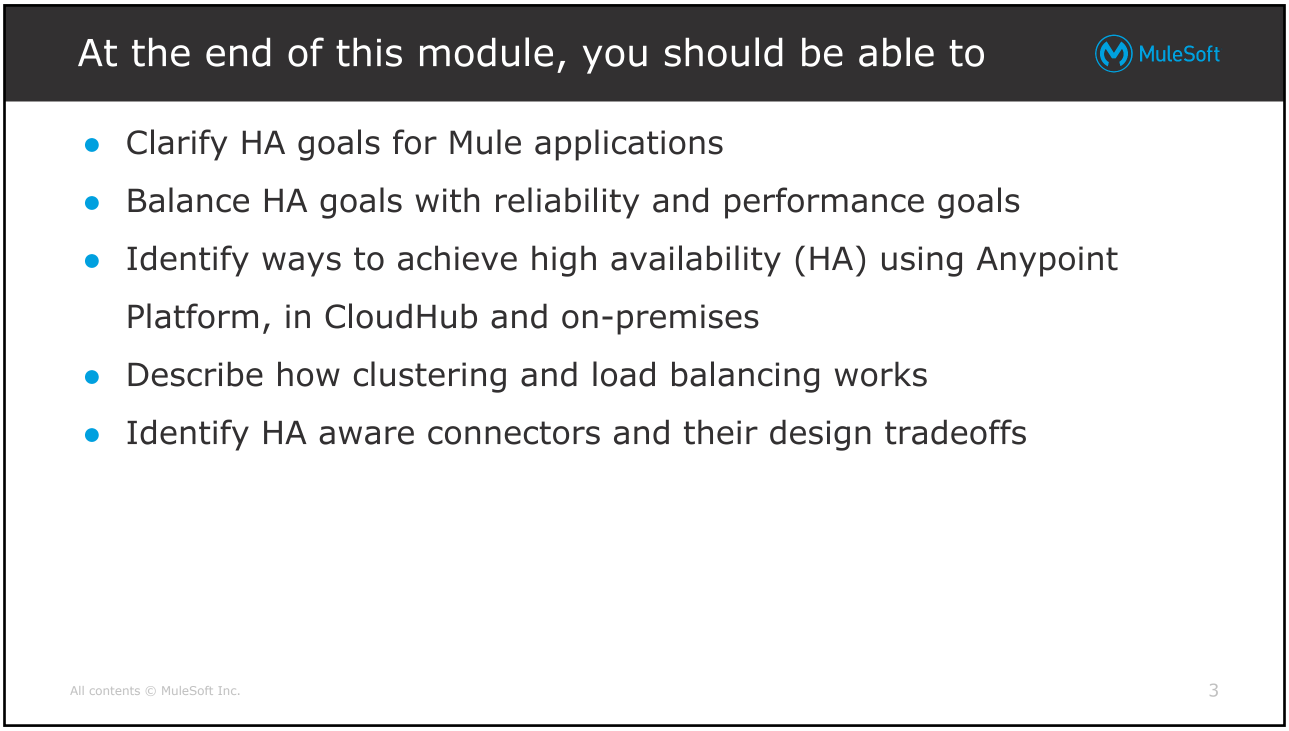
Module 13: Designing for High Availability Goals

Goal





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

- Clarify HA goals for Mule applications
- Balance HA goals with reliability and performance goals
- Identify ways to achieve high availability (HA) using Anypoint Platform, in CloudHub and on-premises
- Describe how clustering and load balancing works
- Identify HA aware connectors and their design tradeoffs

Achieving high availability (HA) goals using multiple Mule runtimes

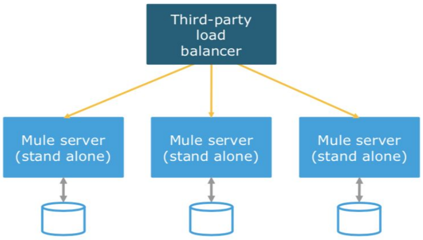
Distinguishing between high availability (HA) vs. disaster recovery (DR) goals

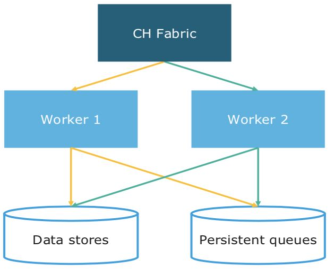
- High availability (HA)
  - How to keep the overall system operational when a system component fails
    - Usually achieved with multiple levels of fault tolerance and/or load balancing
  - In CloudHub, you can deploy a Mule application to multiple CloudHub workers
- Disaster recovery (DR)
  - How to restore a system to a previous acceptable state after a natural or man-made disaster
  - Must compensate for failure of entire sub-systems/regions/zones/data centers, and hence there are a large number of system components that failed
- Read details in the Mule User Guide

  https://docs.mulesoft.com/mule-user-guide/v/3.9/hadr-guide

Making Mule applications highly available using Mule runtimes

- High availability can be achieved by **horizontally scaling** to multiple Mule runtimes
  - Process on multiple concurrent physical machines/VMs, often distributed across networks
- HA goals can be met via **load balancing** and/or **clustering**
  - Clustering of Mule runtimes uses an active-active model of node
  - Load distributes across the active nodes


Third-party load balancer
Mule server (stand alone)
Mule server (stand alone)
Mule server (stand alone)



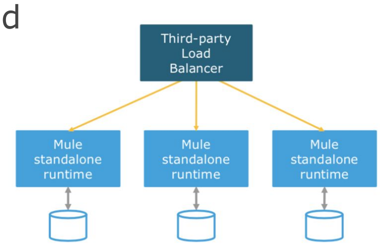
Comparing Mule runtime server groups vs. clusters

- Runtime Manager can define server groups or clusters
- A server group is just an administrative grouping of isolated Mule runtimes
- A cluster provides additional guarantees to prevent contention between the Mule runtimes
  - File access by File based transports
  - All JMS topic subscribers connect to the same topic, resulting in duplicate processing
  - JMS request/response queues might send the response to a different Mule runtime, causing uncorrelated response processing or other failures
  - Salesforce streaming API will fail because the API only supports a single consumer

https://docs.mulesoft.com/mule-runtime/4.1/mule-high-availability-ha-clusters

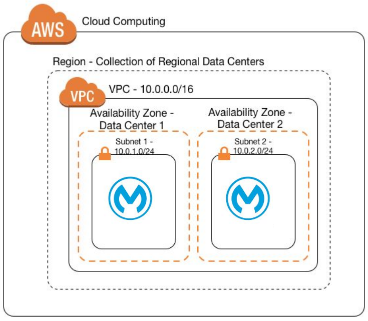
Unclustered load balancing for HA and performance in customer-hosted runtime planes

- Multiple Mule runtimes are configured to run the same Mule application(s)
- Does not share or synchronize data between Mule runtimes
  - Could potentially lead to processing duplicate or lost messages
  - But using a shared database, message broker, or other system is possible
- Messages must be distributed or load balanced
  - Requires third party products



Achieving HA in CloudHub using load balancing between multiple CloudHub workers

- HA can be achieved using multiple CloudHub workers (>1)
  - Easy to configure
  - Workers do not share any memory
  - Workers can use an external system for state management
- Each worker is created in a **different availability zone** in the **same AWS region**


AWS
Cloud Computing
Region - Collection of Regional Data Centers
VPC
VPC - 10.0.0.0/16
Availability Zone - Data Center 1
Availability Zone - Data Center 2
Subnet 1 - 10.0.1.0/24
Subnet 2 - 10.0.2.0/24

- Mule application data can be stored and shared between CloudHub workers in the Anypoint Object Store (OSv2)
- VM queues can be changed to persistent or non-persistent in Runtime Manager
  - Without changing any configuration or properties of the deployed Mule app

Achieving HA using customer-hosted clusters

- A cluster is a set of customer-hosted Mule runtimes that act as a unit
  - Servers in a cluster communicate and share information through a distributed shared memory grid
  - Selected data is replicated across memory in different physical machines
  - Cluster nodes can be configured to be more reliable so they also copy data to disk or external storage
- Nodes are aware of each other
- All cluster nodes work in active-active mode and there is a primary node where Schedulers, JMS listeners, etc. run
- To manage peak loads, nodes can be added to / subtracted from a cluster
- Only available for customer-hosted Mule runtimes

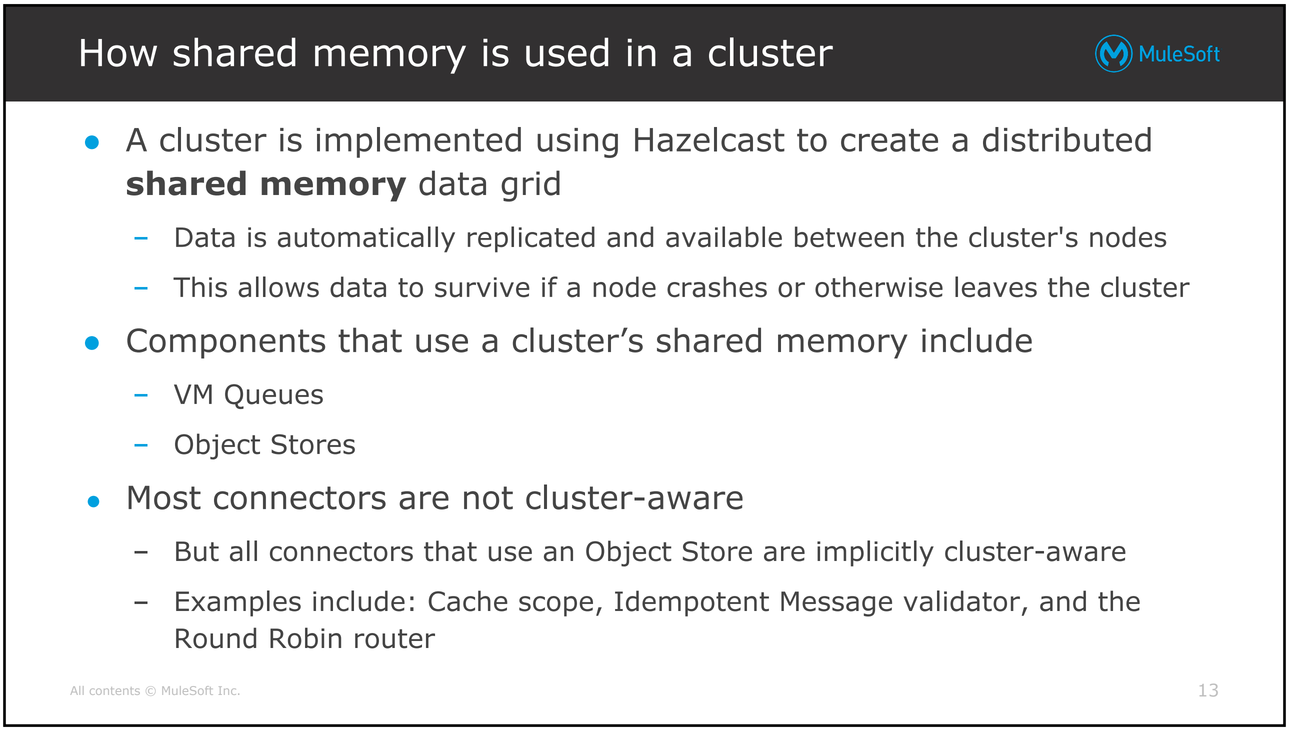
How clusters survive failure of a Mule runtime

- If one Mule runtime (node) fails, outstanding tasks transfer automatically to surviving nodes in the cluster so are still **available**
- If the failed node was the primary node than one of the remaining nodes is elected as the new primary node to continue managing HA
- Nodes can be added to a cluster to manage peak load, then later subtracted, without needing to redeploy Mule applications
- A cluster can be tuned to be either more performant or more reliable
  - The cluster can be configured with a quorum count so the cluster is not available until a minimum number of nodes are active
  - The cluster can be configured to decide how many nodes replicate the data, so data can survive if multiple nodes go offline

How customer-hosted Mule runtimes (nodes) join a cluster

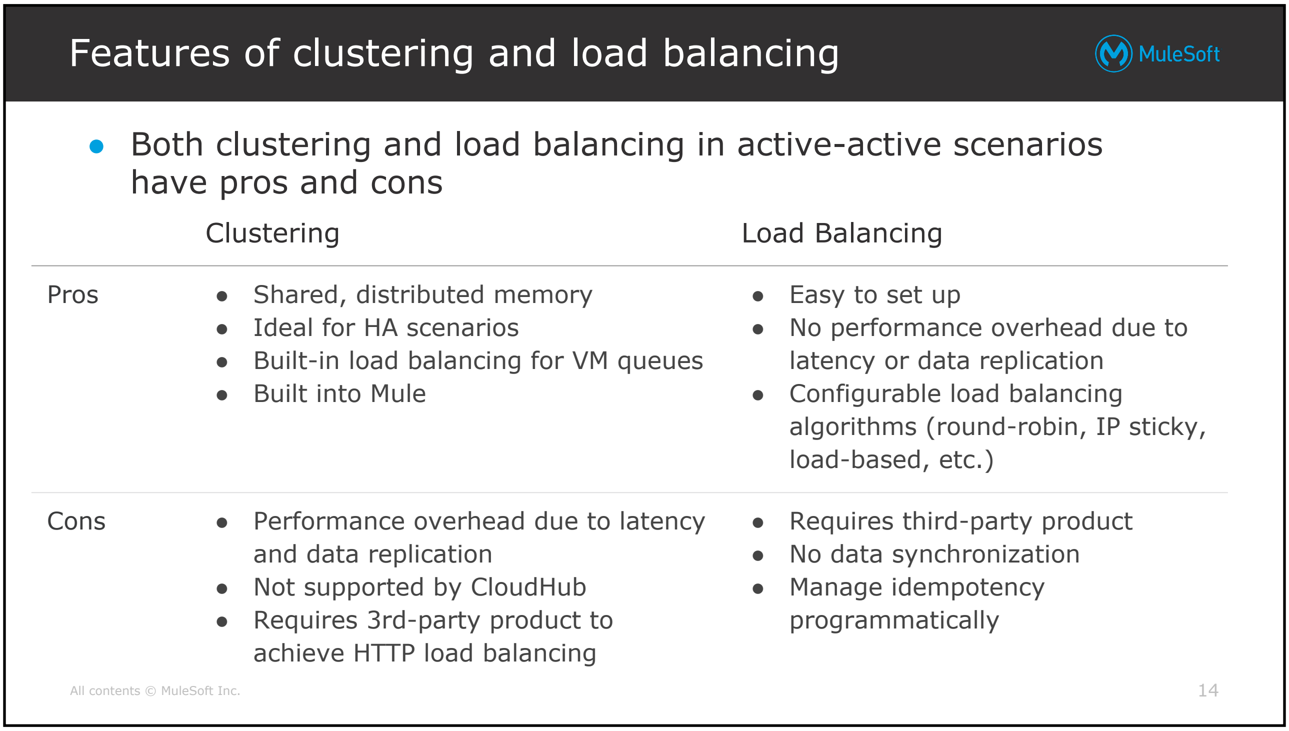
- Nodes can discover and join a cluster using multicast or unicast

	Unicast	Multicast
Characteristics	• Cluster uses IP address for identifying server	• Cluster group servers automatically detect each other
Pros	• No special network configuration other than IP of server	• Nodes dynamically join the cluster when the node is started
Cons	• IP of at least one other node must be known and configured in each node's cluster configuration	• Only permitted in network where multicast is allowed



How shared memory is used in a cluster

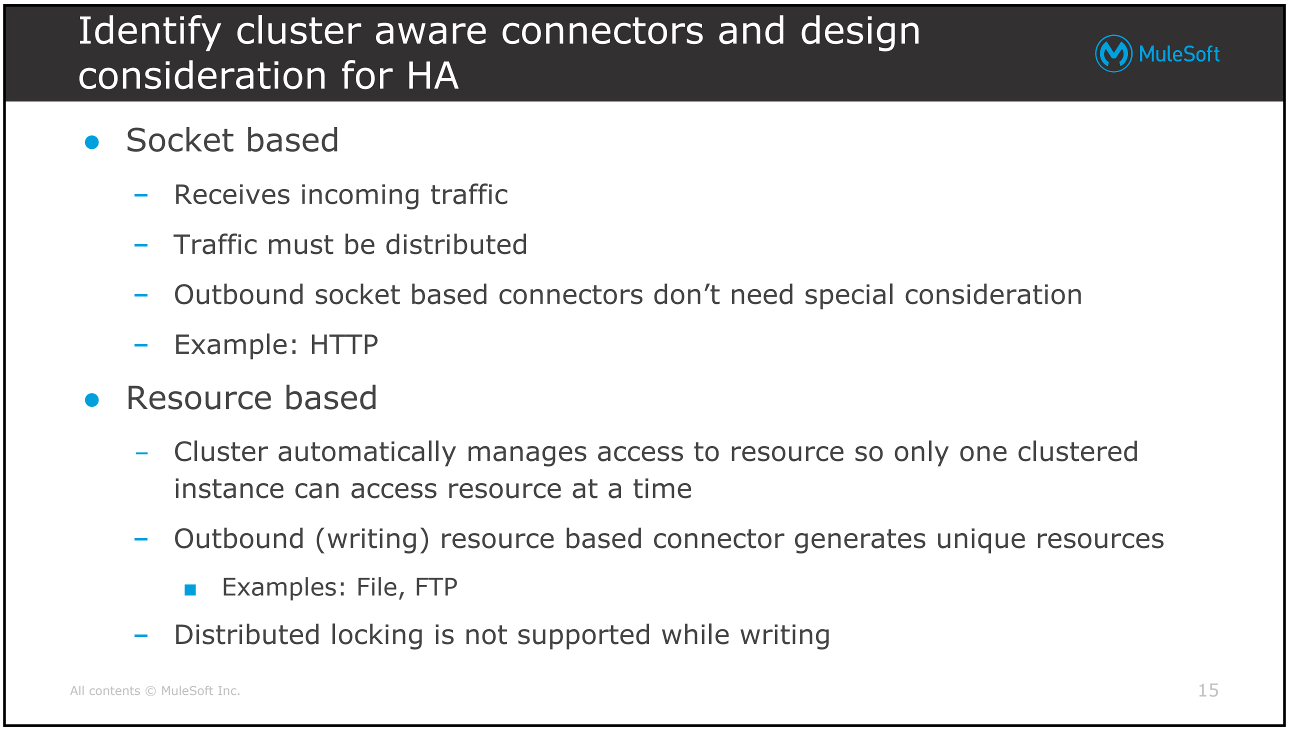
- A cluster is implemented using Hazelcast to create a distributed **shared memory** data grid
  - Data is automatically replicated and available between the cluster's nodes
  - This allows data to survive if a node crashes or otherwise leaves the cluster
- Components that use a cluster's shared memory include
  - VM Queues
  - Object Stores
- Most connectors are not cluster-aware
  - But all connectors that use an Object Store are implicitly cluster-aware
  - Examples include: Cache scope, Idempotent Message validator, and the Round Robin router



Features of clustering and load balancing

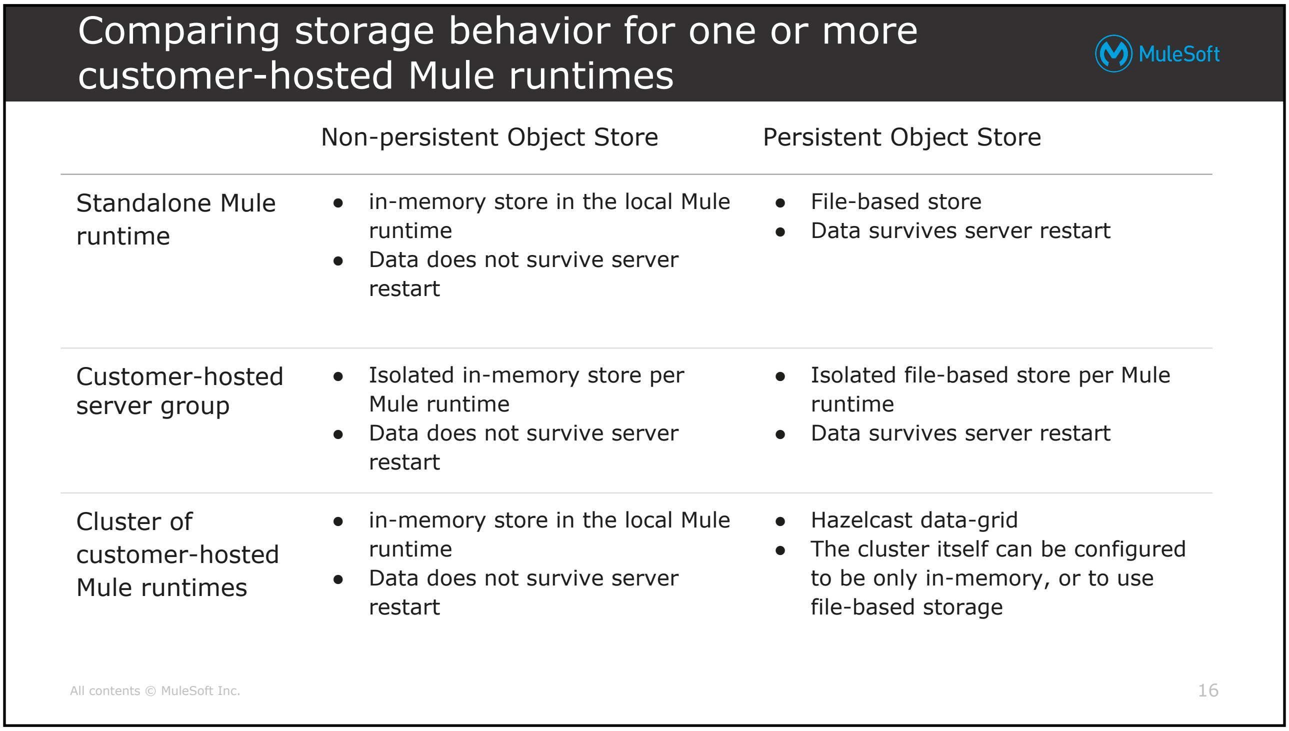
- Both clustering and load balancing in active-active scenarios have pros and cons

	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



Identify cluster aware connectors and design consideration for HA

- Socket based
  - Receives incoming traffic
  - Traffic must be distributed
  - Outbound socket based connectors don’t need special consideration
  - Example: HTTP
- Resource based
  - Cluster automatically manages access to resource so only one clustered instance can access resource at a time
  - Outbound (writing) resource based connector generates unique resources
    - Examples: File, FTP
  - Distributed locking is not supported while writing

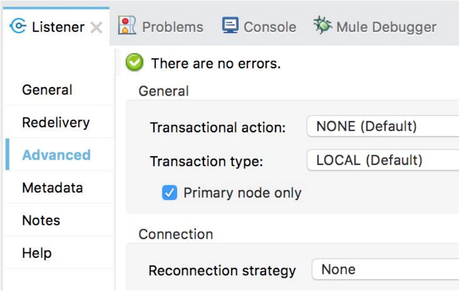


Comparing storage behavior for one or more customer-hosted Mule runtimes

	Non-persistent Object Store	Persistent Object Store
Standalone Mule runtime	• in-memory store in the local Mule runtime • Data does not survive server restart	• File-based store • Data survives server restart
Customer-hosted server group	• Isolated in-memory store per Mule runtime • Data does not survive server restart	• Isolated file-based store per Mule runtime • Data survives server restart
Cluster of customer-hosted Mule runtimes	• in-memory store in the local Mule runtime • Data does not survive server restart	• Hazelcast data-grid • The cluster itself can be configured to be only in-memory, or to use file-based storage

Identify cluster aware connectors and design consideration for HA

- Listener based
  - Traffic is distributed automatically
  - Must decide if the listener should only fire on the primary node or on all nodes
  - Examples: VM, JMS



There are no errors.

General

Transactional action: NONE (Default)

Transaction type: LOCAL (Default)

☑ Primary node only

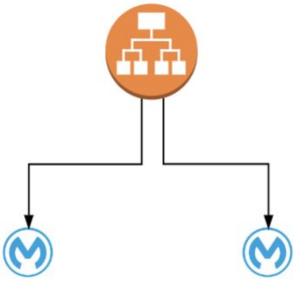
Connection

Reconnection strategy: None

- Schedulers
  - The scheduler only fires on the primary node

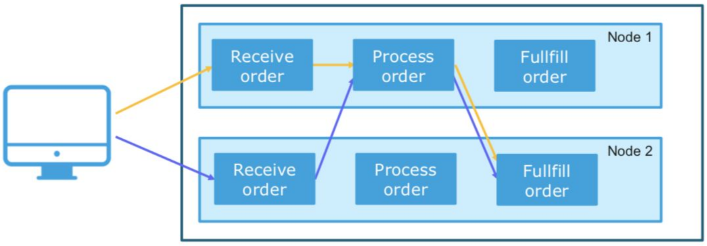
Load balancing for HTTP/S connector

- For customer-hosted Mule runtimes, HTTP requests need to be load balanced through a 3rd-party product
  - Traffic must be distributed
  - Load balancers is required (Nginx, Apache web server)
  - Outbound socket based connector don’t need special consideration
- For CloudHub workers, HTTP requests are automatically load balanced through shared or dedicated CloudHub load balancers



Clustering for VM Connector

- Messages published to a VM queue in a cluster are automatically load balanced to receiving flows
  - No additional servers or infrastructure are required
  - Every node in the cluster can execute flows of deployed Mule apps
  - The cluster manager automatically determines what node to use based on load
    - Not a deterministic round-robin algorithm

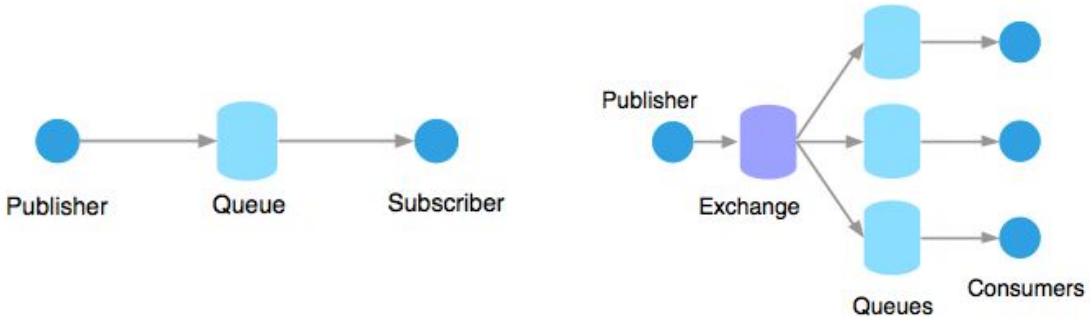


Understanding Anypoint MQ

- **Anypoint MQ** is a multi-tenant, cloud messaging service that enables customers to perform advanced asynchronous messaging scenarios between their applications
  - Is fully integrated with Anypoint Platform
    - Offers role based access control
    - Client application management
    - Connectors
  - Optionally, can provide strict first in, first out (FIFO) processing to enable ordering of messages
  - REST API allows access by non-Mule applications
  - Displays usage statistics on the number of messages

Anypoint MQ message processing pattern

- An Anypoint MQ **queue** delivers a single message to a single consumer
- An Anypoint MQ **exchange** provides a way to broadcast a single message to many queues



How Anypoint MQ distributes messages

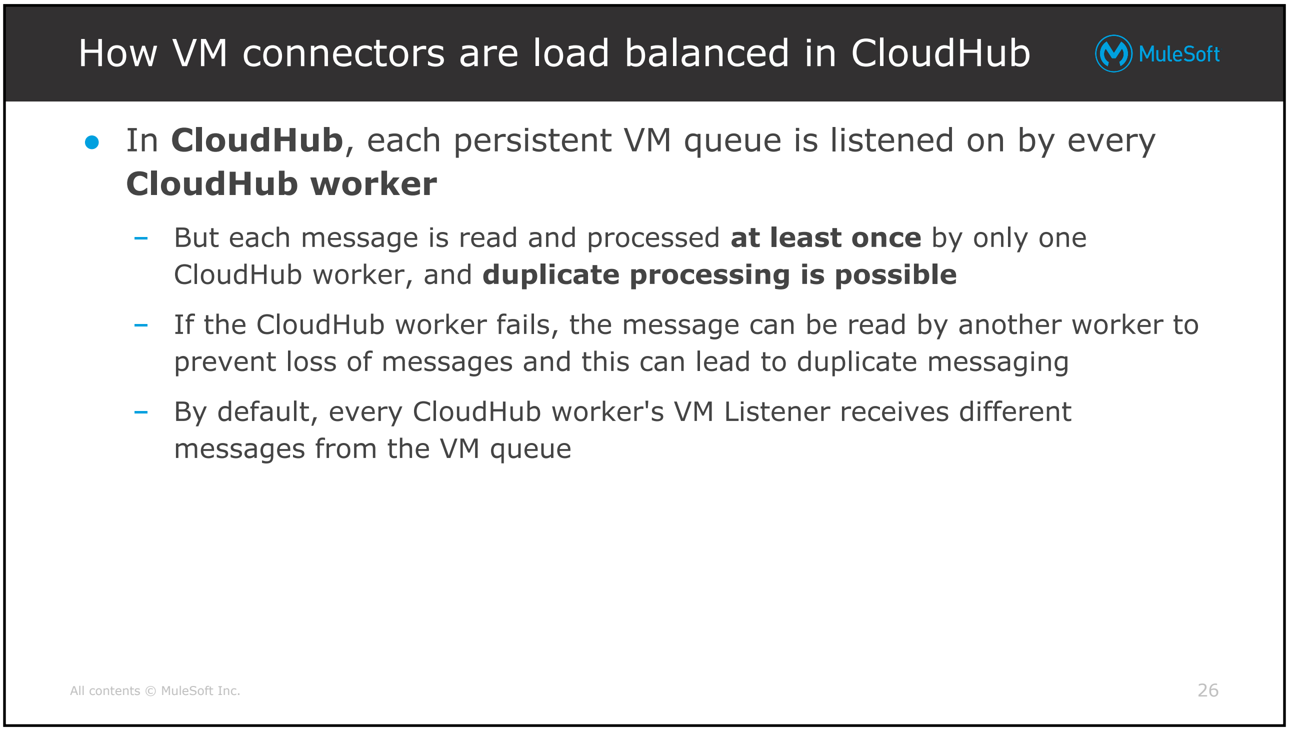
- In Anypoint MQ, reliability is provided via queues through a **lock and ack mode**
- A **consumer** retrieves messages from a **queue**
  - This locks the messages for a finite period of time, and makes the messages invisible to all other consumers
- If the message is not processed within the lock period, the lock times out, and consequently the message is made visible to other consumers to be processed
- This ensures that if there was a failure of some sort, such as the consumer node crashed, a message can be processed by another node

Messages in a VM queue are not load balanced for Mule apps deployed to a **standalone Mule runtime**

- Messages passed on a VM queue will NOT be automatically load balanced to receiving flows across Mule runtimes
  - Every standalone Mule runtime node will execute flow instances independently
  - The processing happens on a single node
  - No distributed processing

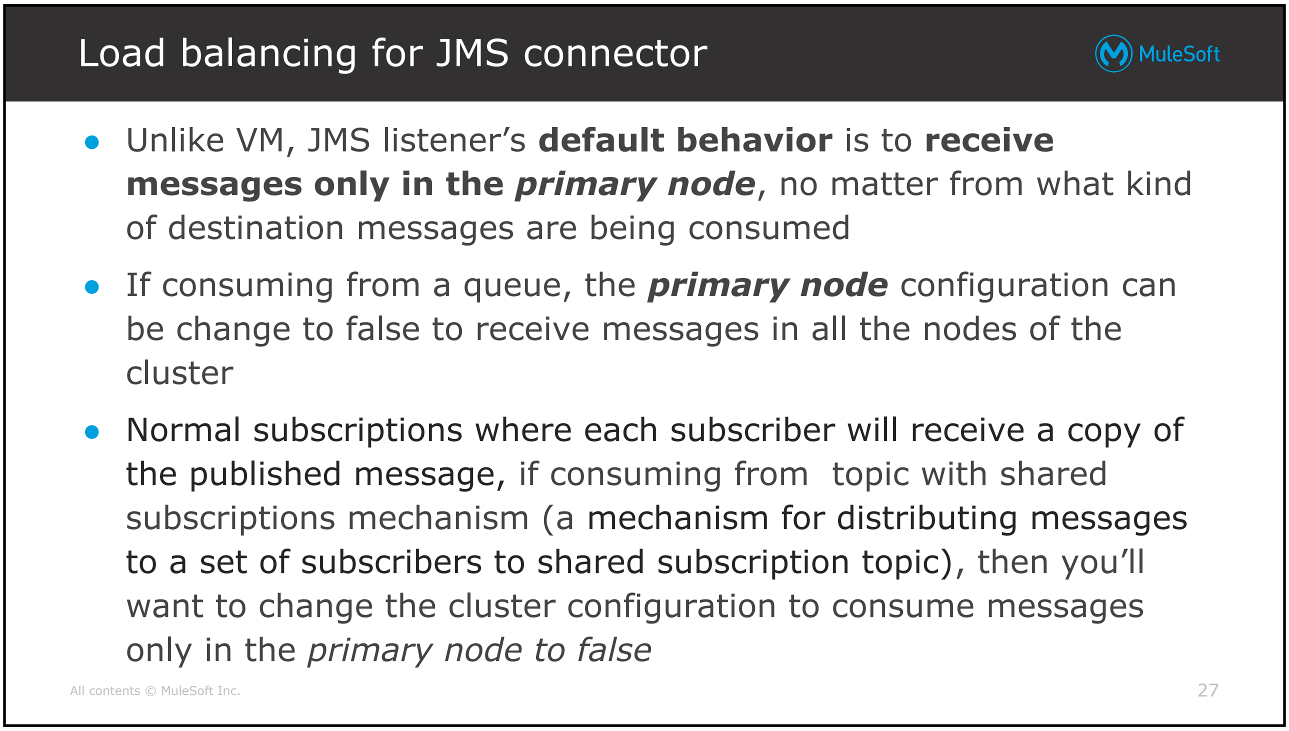
How VM connectors are load balanced in customer-hosted Mule runtimes

- When the Mule runtimes are combined into a cluster, persistent queues are backed by the memory grid
- Then multiple consumers share the VM queue, and messages are automatically load balanced between consumers
- The load balancing is as fast as possible, to whichever consumer is first (not round-robin)



How VM connectors are load balanced in CloudHub

- In **CloudHub**, each persistent VM queue is listened on by every **CloudHub worker**
  - But each message is read and processed **at least once** by only one CloudHub worker, and **duplicate processing is possible**
  - If the CloudHub worker fails, the message can be read by another worker to prevent loss of messages and this can lead to duplicate messaging
  - By default, every CloudHub worker's VM Listener receives different messages from the VM queue



Load balancing for JMS connector

- Unlike VM, JMS listener’s **default behavior** is to **receive messages only in the *primary node***, no matter from what kind of destination messages are being consumed
- If consuming from a queue, the ***primary node*** configuration can be change to false to receive messages in all the nodes of the cluster
- Normal subscriptions where each subscriber will receive a copy of the published message, if consuming from topic with shared subscriptions mechanism (a mechanism for distributing messages to a set of subscribers to shared subscription topic), then you’ll want to change the cluster configuration to consume messages only in the *primary node to false*

Load balancing for FTP, File connectors

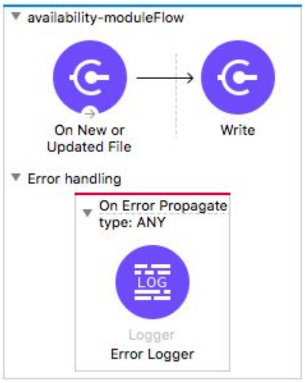
- A cluster automatically manages access to a resource so only one clustered node can access the resource at a time
- An outbound resource based connector generates unique resources
- Distributed locking is not supported

Failures of node in cluster / load balancer

- Failure of a node in a cluster or load balancer causes failures of HTTP requests processed by that node
  - However a typical enterprise level load balancer (such as AWS ELB) automatically recognizes failures of a node and stops sending subsequent requests to the failed node
- JMS, File, VM, FTP listeners, and Schedulers continue to perform processing on the elected primary node in cluster
  - The behavior is mostly consistent with the behavior in CloudHub
  - The primaryNodeOnly configuration is only used in a cluster, not by CloudHub
- Failures of a load balancer is a single point of failure
  - CloudHub load balancers are highly available and deployed in more than one availability zone

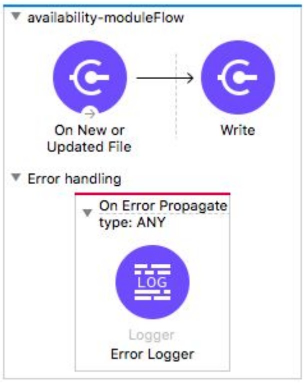
Exercise 13-1: Design an HA and reliable application for file transfer

- Design a file transfer use case that meets agreed HA goals, including minimum uptime and the minimum allowed number of Mule application failures



Exercise context

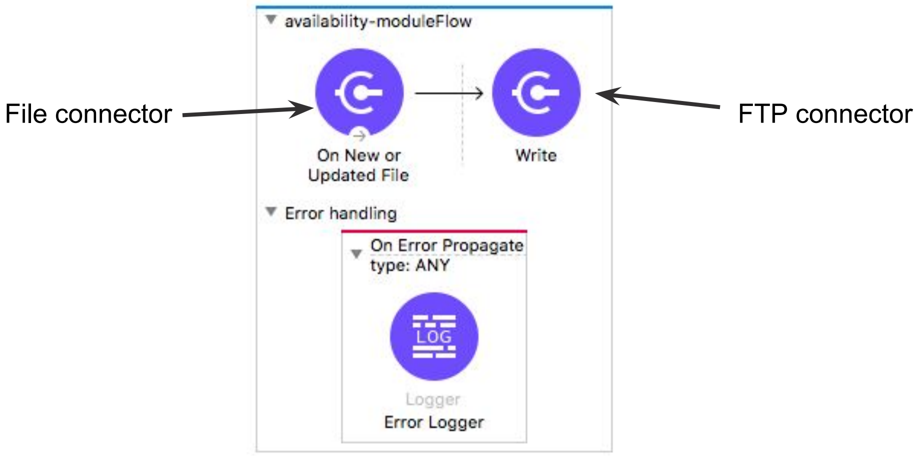
- The Mule application has to transfer a file from a local directory to an online FTP server
- The Mule application has to meet an HA availability SLA of 99.99% per year and must be highly reliable against Mule application failures



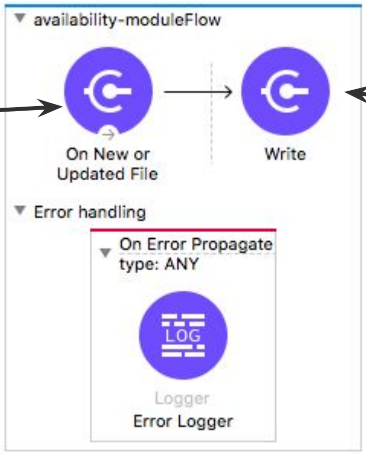
Exercise overview: Design an HA and reliable Mule application for file transfer

- Design flow/s for the Mule application
- Identify scaling options for the Mule application
- Identify the number of workers for the Mule application
- Decide Mule application persistence options to meet the Mule application's reliability goals
- Design the Mule application to meet reliability goals regarding Mule application and system crashes
- How would you design one flow to transfer a file to the FTP server
  - This is less reliable, but easier and more direct to implement

Exercise step: Design the file transfer flow



Exercise steps: What happens if the FTP write fails and the file is already deleted?



File connector

FTP connector

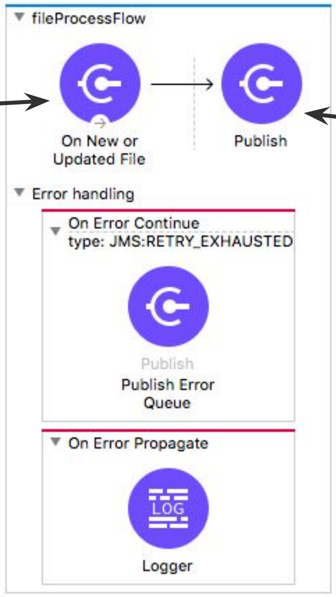
Exercise step: Answer these questions

- What failures points can reduce reliability of the system?
- If an error happens after reading the file, but before the file is completely processed, how can you assure the file is not deleted?
- What should be done if file reading fails a few times?
- What should be done if the FTP write process fails a few times?
- Can persistence help to enhance reliability?
- What are the options to maintain persistence?
- When should the file be deleted from the inbound file server?
- What horizontal or vertical scaling options can help?

Exercise solution options: Design an HA and reliable Mule application for file transfer

- Vertically scaling
  - A single Mule runtime does not make sense to use one node for HA (99.99%)
  - A better option is to horizontally scale the Mule application across multiple Mule runtimes
- Horizontal scaling options
  - Deploy to multiple Mule runtimes
    - Multiple nodes would access the same file
    - VM transport will not load balance between nodes
  - Clustering
    - Cluster manages resource access so only one node can access file
    - VM transport is selected for application persistence to enhance reliability in cluster

Exercise solution: How to orchestrate file transfer



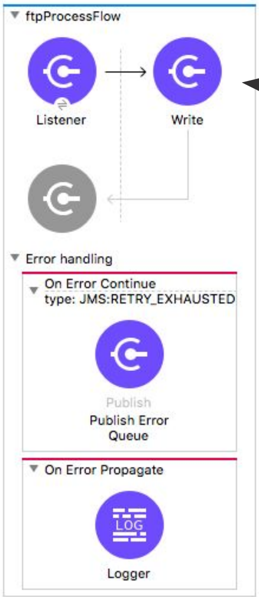
File connector

JMS or VM connector

- Send the file async to a processing queue
- Queue can be made HA and reliable
- Queue is configured with a reconnection strategy
- Error queue used if retries are exhausted

Exercise solution: How to orchestrate ftp process

JMS or VM connector



FTP connector

- Receive files from the processing queue
- Queue can be made HA and reliable
- Queue is configured with a reconnection strategy
- For JMS, the Listener can also configure a redelivery policy
- Error queue used if JMS redeliveries are exhausted

- Write the processed file using an FTP connector
- Set a reconnection strategy
- If the Mule application is deployed to multiple Mule runtimes (or CloudHub workers), then several files can be processed concurrently

Summary

Summary

- HA goals must be clearly defined for applications
  - Performance and HA are often opposing goals, which involve trade-offs
- HA is achieved by scaling the Mule application with clustering and load balancing
- Load balancing and clustering for HTTP/S requests require an external product
- Automatic load balancing in a cluster only works for flows that start with a VM connector