Vendor: Salesforce

Exam Code: Plat-Arch-204

Exam Name: Salesforce Certified Platform Integration Architect

Version: DEMO

QUESTION 1
A customer imports data from an external system into Salesforce using Bulk API. These jobs have batch sizes of 2,000 and are run in parallel mode. The batches fail frequently with the error "Max CPU time exceeded". A smaller batch size will fix this error. What should be considered when using a smaller batch size?

A. Smaller batch size may increase time required to execute bulk jobs.
B. Smaller batch size can trigger "Too many concurrent batches" error.
C. Smaller batch size may exceed the concurrent API request limits.

Answer: A
Explanation:
The Bulk API is designed to process massive datasets by breaking them into smaller batches that Salesforce processes asynchronously. When a batch fails with the "Max CPU time exceeded" error, it typically indicates that the complexity of the operations triggered by the record--such as Apex triggers, Flows, or complex sharing calculations--exceeds the 10,000ms limit within a single transaction.

Reducing the batch size is the standard architectural remedy because it reduces the number of records processed in a single transaction, thereby lowering the total CPU time consumed by those records. However, the architect must consider the impact on the overall throughput and execution time.

When batch sizes are smaller, the total number of batches required to process the same dataset increases. For instance, moving from a batch size of 2,000 to 200 for a 1-million-record dataset increases the number of batches from 500 to 5,000. Each batch carries its own overhead for initialization and finalization within the Salesforce platform. Consequently, while the individual batches are more likely to succeed, the total time required to complete the entire job will increase.

The architect should also be aware of the daily limit on the total number of batches allowed (typically 15,000 in a 24-hour period). While Option C mentions API request limits, the Bulk API is governed more strictly by its own batch limits. Option B is less likely because "parallel mode" naturally manages concurrency. Thus, the primary trade-off the architect must present to the business is a gain in reliability (successful processing) at the cost of total duration (increased sync time).

QUESTION 2
An integration architect needs to build a solution that will use the Streaming API, but the data loss hould be minimized, even when the client re-connects every couple of days. Which two types of Streaming API events should be considered?

A. High Volume Platform and Generic Events
B. Change Data Capture and High Volume Platform Events
C. Push Topic and Change Data Capture Events

Answer: B
Explanation:
In a robust event-driven architecture, "durability" is the ability of a system to retain events so that a subscriber can retrieve them even after being offline. For an architect needing to minimize data loss over "every couple of days," the selection must focus on event types that support a high retention window.

Salesforce provides two modern event types specifically designed for high-scale, durable messaging:

Change Data Capture (CDC): This automatically broadcasts changes to Salesforce records (Create, Update, Delete, Undelete). CDC events are highly durable and are retained in the event bus for 72 hours (3 days).

High-Volume Platform Events: These are custom events defined by the architect. Like CDC, High-Volume Platform Events also provide a 72-hour retention window.

Earlier versions of the Streaming API, such as PushTopic Events and Generic Events (Option C and parts of A), typically offered a 24-hour retention window, which would not satisfy the requirement of a client re-connecting "every couple of days" (potentially 48-72 hours later).

By utilizing CDC and High-Volume Platform Events, the architect can leverage the Replay ID. When the client re-connects, it can send the Replay ID of the last event it successfully processed. Salesforce then "replays" all events that occurred during the client's downtime from that specific point, up to the 72-hour limit. This mechanism ensures zero data loss for planned or unplanned outages lasting up to three days, making it the most resilient choice for the specified requirements.

QUESTION 3
A company's security assessment noted vulnerabilities on the unmanaged packages in its Salesforce orgs; notably, secrets that are easily accessible and in plain text, such as usernames, passwords, and OAuth tokens used in callouts from Salesforce. Which persistence mechanisms should an integration architect require to be used to ensure that secrets are protected from deliberate or inadvertent exposure?

A. Protected Custom Metadata Types and Named Credentials
B. Encrypted Custom Fields and Protected Custom Settings
C. Named Credentials and Protected Custom Settings

Answer: A
Explanation:
Named Credentials is the primary Salesforce-recommended mechanism for securely storing authentication details (including passwords, tokens, and secrets) for HTTP callouts. Secrets are encrypted, not visible in debug logs, and Salesforce handles authentication without exposing them in Apex code.

However, in Named Credentials, admins with "Customize Application" permission can view/edit the secrets.

To further protect secrets (e.g., hide them completely from admins or in packaged scenarios), use Protected Custom Metadata Types (preferably in a managed package). These allow Apex code in the same namespace/package to access the secrets while hiding them from users, API queries, or subscriber orgs.

This combination addresses both standard callouts (via Named Credentials) and cases needing maximum obfuscation (via Protected Custom Metadata), directly mitigating plain-text exposure in unmanaged packages.

QUESTION 4
Universal Containers (UC) is a global financial company that sells financial products and services. There is a daily scheduled Batch Apex job that generates invoices from a given set of orders. UC requested building a resilient integration for this Batch Apex job in case the invoice generation

fails. What should an integration architect recommend to fulfill the requirement?

A. Build Batch Retry and Error Handling using BatchApexErrorEvent.
B. Build Batch Retry and Error Handling in the Batch Apex job itself.
C. Build Batch Retry and Error Handling in the middleware.

Answer: A
Explanation:
Resiliency in long-running Batch Apex processes is best achieved by utilizing modern, event-driven error handling frameworks provided by the Salesforce platform. The BatchApexErrorEvent is the architecturally recommended component for monitoring and responding to failures in Batch Apex jobs.

When a Batch Apex class implements the `Database.RaisesPlatformEvents` interface, the platform automatically publishes a BatchApexErrorEvent whenever an unhandled exception occurs during the execution of a batch. This event contains critical metadata, including the exception message, the stack trace, and the scope (the specific IDs of the records that were being processed when the failure occurred).

An Integration Architect should recommend building a Platform Event Trigger that subscribes to these error events. This trigger can perform sophisticated error handling logic, such as:

Logging the failure details into a custom "Integration Error" object for auditing.

Initiating a retry logic by re-enqueuing only the failed records into a new batch job.

Notifying administrators or external systems via an outbound call or email.

QUESTION 5
An integration architect has built a Salesforce application that integrates multiple systems and keeps them synchronized via Platform Events. What is taking place if events are only being published?

A. The platform events are published after the Apex transaction completes.
B. The platform events are published immediately before the Apex transaction completes.
C. The platform events are being published from Apex.

Answer: A
Explanation:
The timing of Platform Event publishing is a critical detail for an Integration Architect, as it affects data consistency and transaction integrity. In Salesforce, the default and most common behavior for publishing Platform Events from Apex is "Publish After Commit."

When an architect chooses the "Publish After Commit" setting (defined at the event level), the events are held in a buffer and are only released to the event bus after the Apex transaction completes successfully. This ensures that if the database transaction fails and rolls back, the event--which might trigger external actions--is never sent. This prevents "ghost" events where an external system is told to process data that was never actually saved to the Salesforce database.

QUESTION 6
Northern Trail Outfitters is planning to perform nightly batch loads into Salesforce from an external system with a custom Java application using the Bulk API. The CIO is curious about monitoring recommendations for the jobs from the technical architect. Which recommendation

should help meet the requirements?

A. Use the getBatchInfo method in the Java application to monitor the status of the jobs from the Java application.
B. Write the error response from the Bulk API status to a custom error logging object in Salesforce using an Apex trigger, and create reports on the object.
C. Set the Salesforce debug logs level to "finest", and add the user ID running the job to monitor in the "Debug Logs" in the setup menu.

Answer: A
Explanation:
For high-volume data loads using the Bulk API, monitoring should be performed programmatically by the orchestrating client--in this case, the custom Java application. The Bulk API is asynchronous, meaning that when you submit a job, Salesforce acknowledges the request and processes it in the background.

The Java application must actively track the state of its own jobs. Using the `getBatchInfo` (or `getJobInfo` in Bulk API 2.0) method allows the application to retrieve the real-time status of each batch. The application can check for statuses such as `Queued`, `InProgress`, `Completed`, or `Failed`. Once a batch is marked as `Completed`, the application can then call `getBatchResult` to retrieve a list of successes and failures for individual records.

Option B is architecturally unsound because Bulk API operations are designed to bypass most synchronous Apex logic to ensure performance; furthermore, creating custom records for every error in a "nightly batch load" would likely hit other platform limits (like storage or CPU) and defeat the purpose of using the Bulk API. Option C is ineffective for Bulk API monitoring, as debug logs do not capture the background processing of bulk batches and would quickly hit the log size limits.

By recommending Option A, the architect ensures that the Java application maintains full control over the integration lifecycle. The application can log errors locally, implement automated retries for transient failures, and provide the CIO with accurate, high-level reporting on the success rate of the nightly loads without placing unnecessary overhead on the Salesforce platform.

QUESTION 7
Northern Trail Outfitters needs to present shipping costs and estimated delivery times to its customers. Shipping services used vary by region and have similar but distinct service request parameters. Which integration component capability should be used?

A. Apex REST Service to implement routing logic to the various shipping service
B. Enterprise Service Bus to determine which shipping service to use and transform requests to the necessary format
C. Enterprise Service Bus user interface to collect shipper-specific form data

Answer: B
Explanation:
When dealing with multiple external service providers (like different regional shippers) that have varying API requirements, the most scalable architectural choice is an Enterprise Service Bus (ESB) or middleware solution. This scenario describes a classic "Orchestration" and "Transformation" use case.

An ESB excels at:

1. Routing/Mediation: The ESB can receive a single, standardized request from Salesforce ("Get

Shipping Rates") and use business logic to determine which regional carrier's API should be called.

2. Transformation: Each carrier likely has a distinct XML or JSON schema. The ESB can transform the common data model sent by Salesforce into the specific format required by the chosen shipper.

3. Protocol Bridging: If one shipper uses SOAP and another uses REST, the ESB handles these technical differences, allowing Salesforce to interact with a single, simplified interface.

QUESTION 8
Northern Trail Outfitters wants to use Salesforce as a front end for creating accounts using the lead-to-opportunity process.

An order is created in Salesforce when the opportunity is Closed/Won, but the back-end Enterprise Resource Planning (ERP) system is the data Master for order.

The customer wants to be able to see within Salesforce all the stages of order processing, like Order Created, Order Shipped, and Order Paid, that are within the retention window.

Which message durability consideration should an integration architect make when designing a solution to meet these business requirements?

A. When subscribing to Salesforce Event Bus, ReplayID is used with a value of -1 to be able to see new events.
B. High-volume event messages are stored for 24 hours (1 day).
C. When subscribing to Salesforce Event Bus, ReplayID is used with a value of -2 to be able to see old and new events.

Answer: C
Explanation:
When designing an event-driven architecture to track order processing stages (Created, Shipped, Paid), the Integration Architect must ensure the solution provides "durability"--the ability to recover messages that were sent while a subscriber was offline or during a system failure. In Salesforce, this is managed through the Event Bus and the ReplayID mechanism.

For High-Volume Platform Events and Change Data Capture, Salesforce provides a standard retention window of 72 hours (3 days). This means that events are stored in the bus for this duration, allowing clients to "replay" events that occurred in the past. To leverage this durability, the subscribing client must specify where in the event stream they wish to begin receiving messages.

There are two special values for the ReplayID:

ReplayID = -1 (Tip of the Stream): The subscriber receives only new events that are published after the subscription is established. Any events published while the client was disconnected are missed. This does not meet the requirement of seeing processing stages that occurred within the retention window if the connection was interrupted.

ReplayID = -2 (All Available Events): The subscriber receives all events that are currently stored in the event bus (up to 72 hours old) as well as all new events.

By recommending the use of ReplayID = -2, the architect ensures that even if the Salesforce frontend or the integration middleware experiences downtime, the system can "catch up" by

retrieving all order status updates (Shipped, Paid, etc.) that were published during that window. This provides a robust and resilient user experience, ensuring that the Opportunity and Order records in Salesforce accurately reflect the state of the ERP system without data gaps. This configuration is essential for maintaining data synchronization in a distributed landscape where Salesforce acts as the engagement layer for a back-1end ERP master.

QUESTION 9
A large consumer goods manufacturer operating in multiple countries is planning to implement Salesforce for its sales and support operations globally. The Manufacturer has the following security requirements:

Internal users from each country have to be authenticated with their local active directory.

Customers can create their own login or use Google login.

Partners have to be authenticated through a central system which is to be determined.

Internal users will have access to the central Enterprise Resource Planning (ERP) with their credentials maintained in the ERP system.

Additional internal systems will be integrated with Salesforce for sales and support business processes.

Which requirement should the integration architect evaluate while designing the integration needs of this project?

A. Evaluate Salesforce native authentication mechanism for all users including customers and partners.
B. Evaluate the build of a custom authentication mechanism for users in each country and support for customers and partners.
C. Consider a third-party single sign-on (SSO) solution supporting all user authentication including customer and partner.

Answer: C
Explanation:
Managing identity across a global enterprise with diverse user personas (Employees, Customers, Partners) requires a centralized Identity and Access Management (IAM) strategy. In a landscape involving multiple local Active Directories, social logins (Google), and a central ERP system, attempting to manage authentication natively within Salesforce or through custom-built local silos would result in high technical debt and security vulnerabilities.

The architect should recommend a third-party Single Sign-On (SSO) solution, acting as a central Identity Provider (IdP). This IdP serves as the orchestration layer for all authentication requests.

For Internal Users: The IdP can federate with the various local Active Directories, allowing users to log in with their existing corporate credentials.

For Customers: The IdP can handle "Social Sign-On" (OpenID Connect) with Google and manage self-registration.

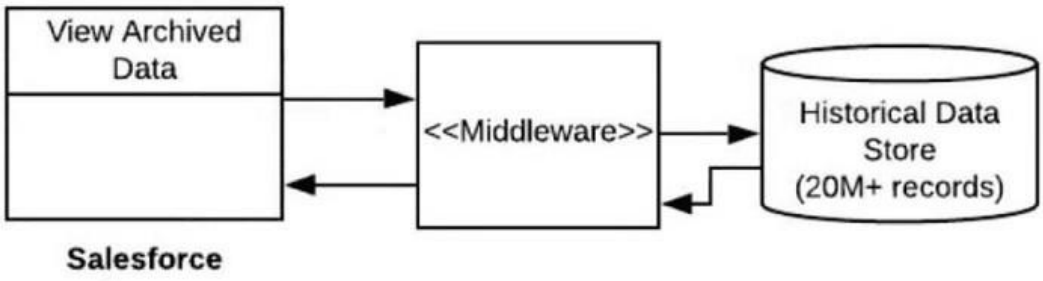
For Partners: It provides the "central system" required for their authentication.

By using a central SSO solution, Salesforce acts as a Service Provider (SP). When a user attempts to access Salesforce, the request is redirected to the IdP via the SAML 2.0 or OpenID

Connect protocol. Once the IdP validates the user against the appropriate backend (AD, Google, or its own directory), it sends a secure assertion back to Salesforce to grant access.

Furthermore, this central IdP can facilitate access to the ERP system and other internal systems. If these systems support SAML, the same SSO session used for Salesforce can be extended to them, providing a true single sign-on experience. This architecture centralizes security auditing, simplifies user de-provisioning (the "kill switch" effect), and ensures a consistent user experience across the global manufacturing landscape. Implementing a third-party IdP is the industry-standard approach for complex integrations where security, scalabi3lity, and multi-protocol support are primary requirements.

QUESTION 10
Given the diagram above, a Salesforce org, middleware, and Historical Data store exist with connectivity between them. Historical records are archived from Salesforce, moved to a Historical Data store (which houses 20 million records and growing), and fine-tuned to be performant with search queries. When reviewing occasional special cases, call center agents that use Salesforce have requested access to view the historical case items that relate to submitted cases.



Which mechanism and patterns are recommended to maximize declarative configuration?

A. Use an ESB tool with a Data Virtualization pattern, expose the OData endpoint, and then use Salesforce Connect to consume and display the External object alongside the Case object.
B. Use an ESB tool with a Request and Reply pattern, and then make a real-time Apex callout to the ESB endpoint to fetch and display historical Data in a custom Lightning compo4nent related to the Case object.
C. Use an ESB tool with a Fire and Forget pattern, and then publish a platform event for the requested historical data.

Answer: A
Explanation:
When designing a solution to view large volumes of archived data (over 20 million records) without physically storing them back in Salesforce, a Data Virtualization pattern is the architecturally preferred approach. This pattern allows users to view and interact with external data in real-time without the overhead of data replication, which would otherwise consume significant storage and impact platform performance.

To maximize declarative configuration, the Salesforce Platform Integration Architect should recommend Salesforce Connect. Salesforce Connect allows for the creation of External Objects, which behave much like standard objects but point to data residing outside of Salesforce. This is achieved by having the middleware (ESB) expose the Historical Data store via an OData (Open Data Protocol) endpoint. Once configured, call center agents can view historical case items directly on the Case record page using standard related lists or lookups, all configured through

the point-and-click interface rather than custom code.

The provided landscape diagram illustrates a clear path from Salesforce through middleware to the Historical Data Store. Option A leverages this by using the ESB to bridge the protocol gap. Because the data store is already "fine-tuned to be performant with search queries," Salesforce Connect can efficiently query only the specific historical records needed for the current case view.

QUESTION 11
A customer's enterprise architect has identified requirements around caching, queuing, error handling, alerts, retries, event handling, etc. The company has asked the integration architect to help fulfill such aspects with its Salesforce program. Which recommendation should the integration architect make?

A. Message transformation and protocol translation should be done within Salesforce.
B. Transform a Fire and Forget mechanism to Request and Reply, which should be handled by middleware tools.
C. Provide true message queueing for integration scenarios given that a middleware solution is required.

Answer: C
Explanation:
Salesforce is a highly capable CRM platform, but it is not a dedicated messaging or orchestration engine. When requirements include complex message queuing, process choreography, and guaranteed quality of service (QoS), the Integration Architect must recommend a middleware solution (ESB or iPaaS).

"True message queuing" involves holding messages in a persistent state until the target system is ready to receive them, handling sophisticated retry logic (such as exponential backoff), and providing dead-letter queues for failed messages. While Salesforce has basic asynchronous tools like Outbound Messaging or Platform Events, they lack the granular control over queuing and orchestration that enterprise middleware provides.

QUESTION 12
Northern Trail Outfitters (NTO) wants to improve the quality of callouts from Salesforce to its REST APIs by adhering to RAML (REST API Markup Language) specifications. The RAML specs serve as interface contracts. Which design specification should the integration architect include to ensure that Apex REST API Clients' unit tests confirm adherence to the RAML specs?

A. Require the Apex REST API Clients to implement the HttpCalloutMock.
B. Call the HttpCalloutMock implementation from the Apex REST API Clients.
C. Call the Apex REST API Clients in a test context to get the mock response.

Answer: A
Explanation:
In Salesforce, you cannot perform real HTTP callouts during unit tests. To test integration logic, developers must use the HttpCalloutMock interface to simulate the API's response. To ensure that the Apex code adheres to the RAML contract, the architect should require that the test mock implementation strictly follows the RAML specifications.

By requiring the Apex REST API Clients to implement the HttpCalloutMock (or more specifically, creating a mock class that implements it), the developer creates a controlled testing environment. The mock class should be coded to return a payload that matches the RAML-defined structure (fields, data types, and status codes). When the test runs, the Apex client receives this "contract-

compliant" response. The unit test then uses assertions to verify that the Apex code correctly parses and handles this specific data structure.

QUESTION 13
Universal Containers (UC) is currently managing a custom monolithic web service that runs on an on- premise server. This monolithic web service is responsible for Point-to-Point (P2P) integrations between Salesforce and a legacy billing application, a cloud-based ERP, and a data lake. UC has found that the tight interdependencies are causing failures. What should an integration architect recommend to decouple the systems and improve performance?

A. Use the Salesforce Bulk API when integrating back into Salesforce.
B. Move the custom monolithic web service from on-premise to a cloud provider.
C. Leverage modular design by breaking up the web service into smaller pieces for a microservice architecture.

Answer: C
Explanation:
The primary architectural flaw in UC's current landscape is the reliance on a monolithic P2P integration layer. In such designs, any failure in one integration thread or a surge in volume for one system can monopolize resources (CPU, memory, threads), causing the entire service--and thus all other integrations--to fail. This lack of isolation leads to the "tight interdependencies" described.

To effectively decouple these systems, the architect should recommend a Microservices Architecture. By breaking the monolithic service into smaller, independent, and modular components, each integration (Billing, ERP, Data Lake) becomes its own isolated service. This approach provides several key architectural benefits:

Isolation of Failure: If the connection to the legacy billing application fails or times out, it no longer impacts the ERP or Data Lake integrations.

Independent Scalability: If the Data Lake integration requires high throughput, that specific microservice can be scaled horizontally without wasting resources on the others.

Technology Agility: Each microservice can be updated or patched independently, allowing for faster maintenance cycles.

QUESTION 14
Northern Trail Outfitters is creating a distributable Salesforce package. The package needs to call into a Custom Apex REST endpoint in the central org. The security team wants to ensure a specific integration account is used in the central org that they will authorize after installation. Which item should an architect recommend?

A. Contact Salesforce Support and create a case to temporarily enable API access for managed packages.
B. Use an encrypted field to store the password that the security team enters.
C. Create an authentication provider in the package and set the consumer key and consumer secret of the connected app in the central org.

Answer: C
Explanation:
When building a distributable package (likely a Managed Package) that must securely communicate back to a central "Hub" org, the architect must use a framework that supports

OAuth 2.0 flows. Storing plain-text or even encrypted passwords (Option B) is a security violation and is brittle across different environments.

The architecturally sound solution is to leverage the Authentication Provider and Named Credentials framework. In the central org, a Connected App is created to act as the OAuth endpoint. In the package, an Authentication Provider is configured using the Consumer Key and Consumer Secret from that Connected App. This setup allows the administrator in the "Subscriber" org (the org where the package is installed) to initiate an OAuth flow.

When the security team "authorizes" the integration after installation, they are essentially completing the OAuth handshake. This grants the subscriber org an Access Token and a Refresh Token associated with the specific integration user in the central org. This mechanism ensures:

Credential Security: No passwords are ever stored in the code or metadata.

Centralized Control: The security team in the central org can revoke the Refresh Token at any time to kill the integration.

Scalability: The same package can be distributed to hundreds of orgs, each with its own unique, secure connection to the central Hub.

By using an Authentication Provider combined with a Named Credential, the Apex code in the package can simply call the endpoint by its developer name, and Salesforce handles the entire authentication header injection automatically, ensuring a robust and secure cross-org integration.

QUESTION 15
When a user clicks "Check Preferences" as part of a Lightning flow, preferences from an externally hosted RESTful service are to be checked in real time. The service has OpenAPI 2.0 definitions. Which integration pattern and mechanism should be selected?

A. Data Virtualization: Salesforce Connect maps external REST data in external objects.
B. Request and Reply: Enhanced External Services invokes a REST API.
C. Remote Call-In: Salesforce REST API with REST Composite Resources.

Answer: B
Explanation:
This scenario describes a classic Request and Reply pattern where a user action in the UI requires an immediate, synchronous response from an external system to determine the next step in a business process (the Flow).

The requirement specifies that an OpenAPI 2.0 (Swagger) definition is available. For an Integration Architect, this is a prime use case for External Services. External Services allow you to import an OpenAPI schema and automatically generate "Invocable Actions" that can be used directly in Flow Builder without writing a single line of Apex code.

Why this is the best fit:

Low Code: It fulfills the requirement purely through declarative configuration, which reduces maintenance and development costs.

Real-Time: It performs a synchronous HTTP callout and waits for the Boolean/String values to be returned to the Flow variables.

Type Safety: Because it uses the OpenAPI definition, Salesforce understands the data types

(Boolean/String) natively.

QUESTION 16
A company needs to send data from Salesforce to a homegrown system behind a corporate firewall. The data is pushed one way, doesn't need to be real-time, and averages 2 million records per day. What should an integration architect consider?

A. Due to high volume of records, number of concurrent requests can hit the limit for the REST API.
B. Due to high volume of records, the external system will need to use a BULK API Rest endpoint to connect to Salesforce.
C. Due to high volume of records, a third-party integration tool is required to stage records off platform.

Answer: C
Explanation:
With a volume of 2 million records per day, this integration exceeds the practical limits of standard near-real-time patterns like Outbound Messaging or synchronous Apex Callouts. Sending 2 million individual REST requests would likely exhaust the daily API limit and could cause significant performance degradation in Salesforce due to transaction overhead.

An Integration Architect must recommend an Asynchronous Batch Data Synchronization pattern, typically facilitated by a third-party ETL/Middleware tool (e.g., MuleSoft, Informatica, or Boomi). Staging the records off-platform is essential for several reasons:

Throttling: The homegrown system behind a firewall may not be able to handle a massive, sudden burst of 2 million records. A middleware tool can ingest the data from Salesforce and "drip-feed" it into the target system at an acceptable rate.

Error Handling and Retries: Middleware provides sophisticated persistence and "Dead Letter Queues" to ensure that if the homegrown system goes offline, no data is lost.

API Efficiency: The middleware can use the Salesforce Bulk API 2.0 to extract the data in large chunks, which is significantly more efficient than individual REST calls and consumes far fewer API limits.

Thank You for Trying Our Product

Lead2pass Certification Exam Features:

★ More than **99,900** Satisfied Customers Worldwide.

★ Average **99.9%** Success Rate.

★ **Free Update** to match latest and real exam scenarios.

★ **Instant Download** Access! No Setup required.

★ Questions & Answers are downloadable in **PDF** format and **VCE** test engine format.

★ Multi-Platform capabilities - **Windows, Laptop, Mac, Android, iPhone, iPod, iPad**.

★ **100%** Guaranteed Success or **100%** Money Back Guarantee.

★ **Fast**, helpful support **24x7**.

View list of all certification exams: http://www.lead2pass.com/all-products.html

10% Discount Coupon Code: **ASTR14**