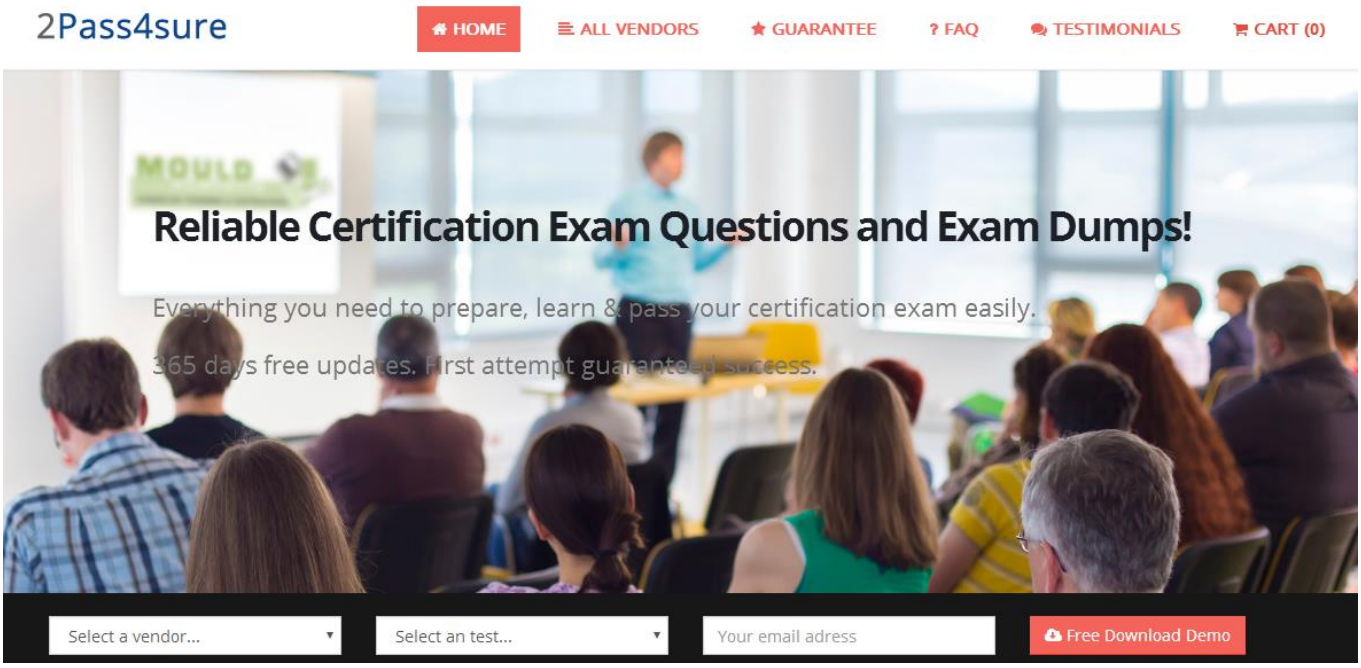
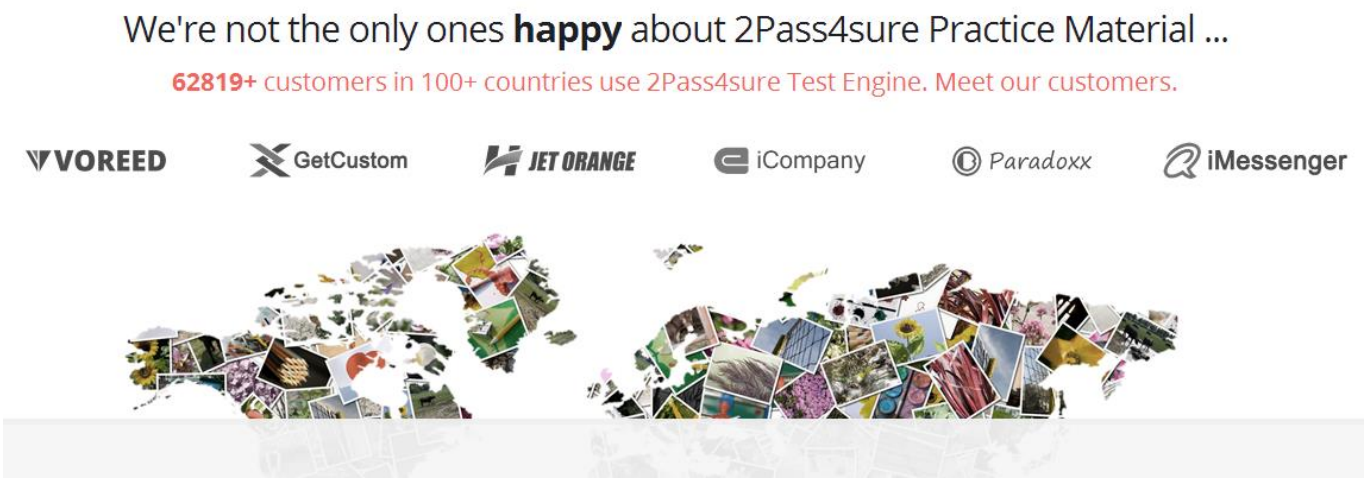
2Pass4sure





http://www.2pass4sure.com/

Reliable Certification Exam Questions and Exam Dumps - 2Pass4sure

Exam : **Plat-Arch-204**

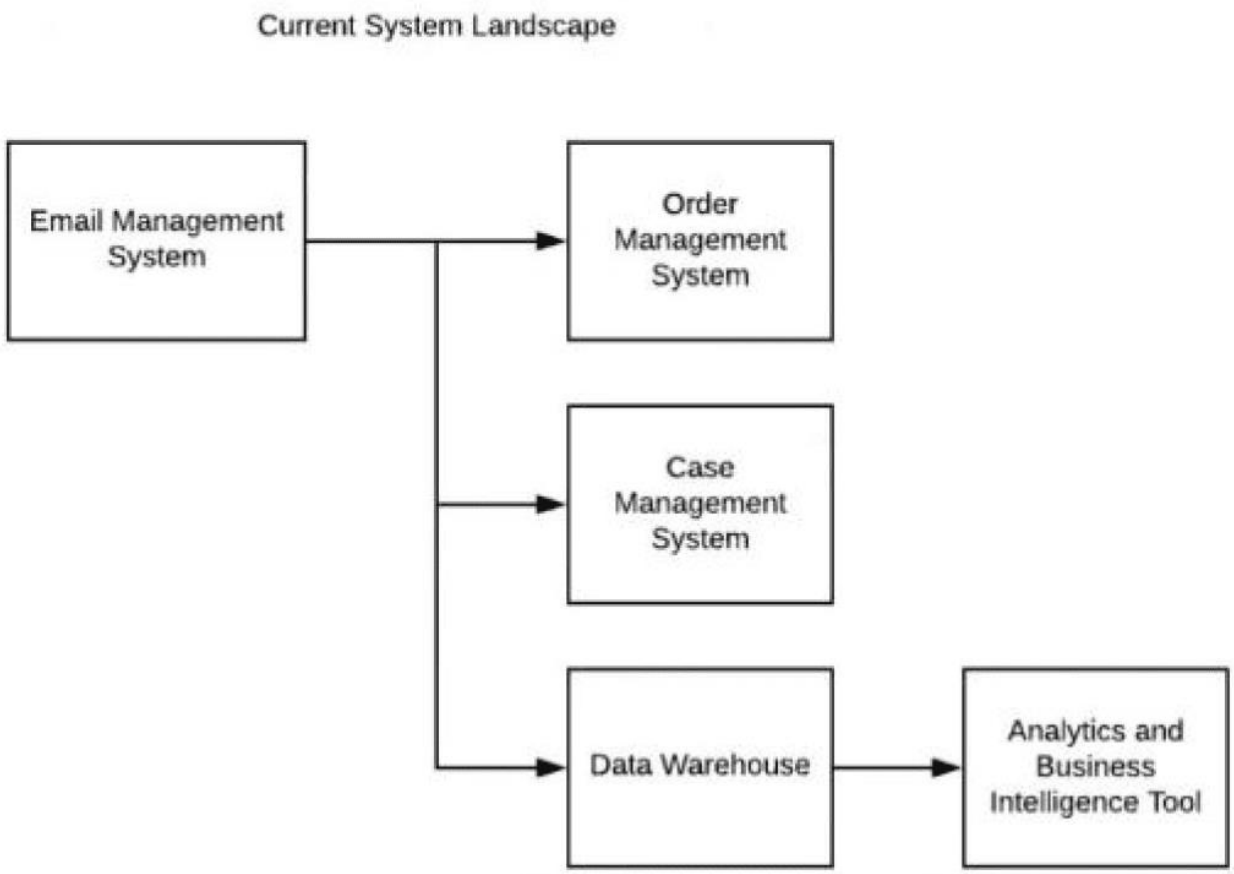
Title : Salesforce Certified Platform Integration Architect

Vendor : Salesforce

Version : DEMO

NO.1 Northern Trail Outfitters (NTO) has a requirement to encrypt a few widely-used standard fields. NTO also wants to be able to use these fields in record-triggered flows.
Which security solution should an integration architect recommend to fulfill the business use case?
A. Shield Platform Encryption
B. Classic Encryption
C. Data Masking
Answer: A
Explanation:
To satisfy the requirement of encrypting standard fields while maintaining their functionality within record-triggered flows, Shield Platform Encryption is the recommended architectural solution.1
Shield Platform Encryption is a modern security layer that allows for encryption at rest while 2preserving critical platform features. Unlike Classic Encryption (Option B)-which is limited to a specific "Encrypted Text" custom field type and often breaks platform features like search and automation-Shield is designed to work with standard fields such as Name, Email, and Phone.
Key architectural considerations for Shield include:
Compatibility with Automation: Shield fields can be used in Flows, Apex triggers, and validation rules. This allows NTO to implement the required record-triggered business logic without needing to decrypt the data manually in code.
Search and Filtering: By using Deterministic Encryption, Shield allows users to filter and search for records based on encrypted fields, which is often a requirement for "widely-used" standard fields.
Compliance and Governance: Shield provides advanced key management (Bring Your Own Key - BYOK) and auditing, ensuring that NTO meets corporate security guidelines while data is being processed by the platform.
Data Masking (Option C) is primarily used for sandboxes to obfuscate PII during testing and is not a production encryption-at-rest solution. By recommending Shield, the architect provides a transparent security model that protects sensitive data without sacrificing the declarative power of Flow Builder.

NO.2 An enterprise customer is planning to implement Salesforce to support case management.



Below is their current system landscape diagram. Considering Salesforce capabilities, what should the integration architect evaluate when integrating Salesforce with the current system landscape?

A. Integrate Salesforce with Data Warehouse, Order Management and Email Management System.

B. Integrate Salesforce with Order Management System, Data Warehouse and Case Management System.

C. Integrate Salesforce with Email Management System, Order Management System and Case Management System.

Answer: A

Explanation:

An Integration Architect's primary responsibility when evaluating a landscape for a new Salesforce implementation is to identify the system of record for each business process and determine which legacy systems will be replaced by Salesforce. In this scenario, the customer is implementing Salesforce specifically to support case management.

According to the provided landscape diagram, the Case Management System currently exists as a standalone entity. Since Salesforce Service Cloud provides native, best-in-class case management capabilities, this legacy system is the primary candidate for retirement. Retiring the legacy Case Management system avoids data fragmentation and ensures that Salesforce serves as the single source of truth for support interactions.

However, for Salesforce to function effectively as a new case management hub, it must integrate with the remaining surrounding systems:

Email Management System: This system likely handles inbound customer communications. An architect must evaluate integrating this with Salesforce (via Email-to-Case or a specialized connector) so that incoming emails automatically generate or update cases.

Order Management System (OMS): Support agents often need to view order history or status to resolve customer inquiries. Integrating Salesforce with the OMS allows for a 360-degree view, enabling agents to see relevant order data directly within the Salesforce case console.
Data Warehouse: For long-term reporting, trend analysis, and a unified customer profile, case data from Salesforce needs to be pushed to the Data Warehouse. This ensures that the Analytics and Business Intelligence Tool downstream can report on support metrics alongside other enterprise data.
Therefore, the architect should evaluate integrations with the Data Warehouse, Order Management, and Email Management System. Option B and C are incorrect because they suggest integrating with the "Case Management System," which is the very system being superseded by Salesforce's native capabilities. By focusing on the integration of these three supporting systems, the architect ensures a seamless transition where Salesforce is fully enriched with the necessary external data to drive support excellence.

NO.3 A company captures orders and needs to send them to the Order fulfillment system. The user is not required to have confirmation from the Order fulfillment system. Which system constraint question should be considered when designing an integration to send orders from Salesforce to a fulfillment system?

A. What latency is acceptable for orders to reach the fulfillment system?

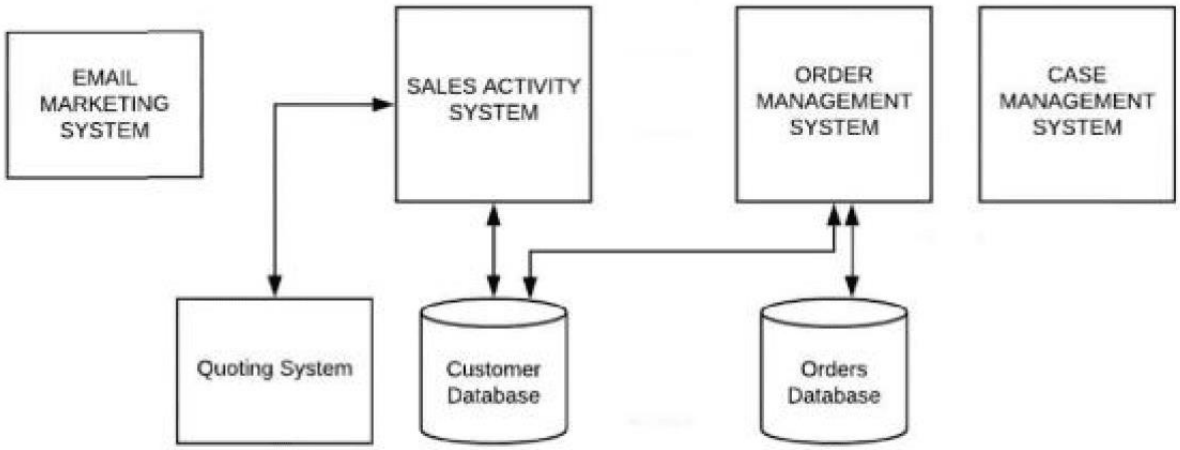
B. Can the fulfillment system implement a contract-first Outbound Messaging interface?

C. Which system will validate order shipping addresses?

Answer: A

Explanation:
When designing an integration where the user does not require immediate confirmation, the architect is moving away from a synchronous "Request-Reply" pattern toward an asynchronous "Fire-and-Forget"16 or "Batch Processing" pattern. In such scenarios, the most critical architectural constraint is defining the latency requirements.
Latency dictates the technical choice of the integration tool. If the fulfillment system needs the order within seconds of creation to begin a high-speed picking process, the architect might choose Salesforce Outbound Messaging or an Apex Callout triggered by a Platform Event. If the system only needs to process orders once an hour or overnight, a Batch ETL process is more appropriate. Understanding the acceptable delay (latency) ensures that the solution meets business expectations without over-engineering for real-time performance where it isn't required.
While Option B (Outbound Messaging) is a valid technical capability, it is a specific solution rather than a high-level "system constraint question" that drives the initial design phase. Option C (Address Validation) is a functional requirement regarding data integrity, but it does not define the architectural framework of the integration as effectively as latency does. By identifying the latency threshold, the architect can determine if the integration should be near real-time, hourly, or daily, which in turn influences how the system handles error recovery, retries, and transaction volumes.

NO.4 A large business-to-consumer (B2C) customer is planning to implement Salesforce CRM to become a customer-centric enterprise. Below is the B2C customer's current system landscape diagram.



The goals for implementing Salesforce include:
Develop a 360-degree view of the customer.
Leverage Salesforce capabilities for marketing, sales, and service processes.
Reuse Enterprise capabilities built for quoting and order management processes.
Which three systems from the current system landscape can be retired with the implementation of Salesforce?

A. Email Marketing, Sales Activity, and Case Management

B. Sales Activity, Order Management, and Case Management

C. Order Management, Case Management, and Email Marketing

Answer: A

Explanation:
In the framework of a Salesforce Platform Integration Architect's landscape evaluation, the primary goal is to determine the "system of record" for each business function and identify redundancies between legacy systems and the proposed Salesforce architecture. This process is driven by the alignment of Salesforce's native "Customer 360" capabilities with the specific goals defined by the enterprise stakeholders.
According to Goal 2, the customer intends to leverage Salesforce specifically for marketing, sales, and service processes. Within the standard Salesforce ecosystem, these domains are addressed by the three core cloud products:
Marketing Cloud provides the capabilities found in the legacy Email Marketing System.
Sales Cloud replaces the functions of the Sales Activity System.
Service Cloud is the native replacement for the Case Management System.
By migrating these three domains to a single platform, the organization directly fulfills Goal 1-developing a 360-degree view of the customer. Consolidating these interactions onto the Salesforce platform allows for a unified data model where customer behaviors in marketing, sales, and support are visible in one place, eliminating the silos inherent in the previous landscape.
However, a critical constraint is presented in Goal 3, which explicitly mandates the reuse of existing enterprise capabilities for quoting and order management. In an integration architecture, this signals that the Quoting System and Order Management System (OMS) are designated as external systems of record that must remain active. These systems often contain complex logic, tax calculations, or supply chain integrations (such as with an SAP Business Suite) that the business is not currently ready to migrate.
Therefore, since the Quoting and Order Management systems must be retained, they are excluded from the retirement list. The remaining three systems-Email Marketing, Sales Activity, and Case

Management-overlap with Salesforce's native strengths and are not protected by the "reuse" requirement. Retiring them streamlines the technology stack and allows the architect to focus on building robust integration patterns (such as REST or SOAP callouts) to connect Salesforce to the retained Quoting and Order Management systems.

NO.5 Northern Trail Outfitters (NTO) has recently changed its Corporate Security Guidelines requiring all cloud applications to pass through a secure firewall before accessing on-premise resources. NTO is evaluating middleware solutions. Which consideration should an integration architect evaluate before choosing a middleware solution?

A. The middleware solution enforces the OAuth security protocol.

B. The middleware solution is able to interface directly with databases via an Open Database Connectivity (ODBC) connection string.

C. The middleware solution is capable of establishing a secure API Gateway between cloud applications and on-premise resources.

Answer: C

Explanation:

When corporate guidelines mandate a firewall-protected entry point for cloud traffic, the middleware architecture must include a component capable of residing in a Demilitarized Zone (DMZ) or perimeter network. The architect must evaluate the solution's API Gateway capabilities.

A secure API Gateway acts as the intermediary that terminates external (cloud) TLS connections and inspects incoming traffic before proxying it to internal systems. It allows the security team to implement:

IP Whitelisting: Ensuring only Salesforce's IP ranges can access the gateway.

Mutual Authentication: Using certificates to verify that the request is genuinely coming from the Salesforce org.

Rate Limiting: Protecting on-premise resources from being overwhelmed by cloud requests.

Option A (OAuth) is an authorization framework and does not satisfy the network-level firewall requirement on its own. Option B (ODBC) is an internal database protocol that should generally never be exposed to a cloud-facing firewall due to security risks. By prioritizing a solution with a hardened API Gateway, the architect ensures that NTO meets its new security mandates while providing a scalable and secure bridge for Salesforce to access back-office services.

NO.6 Northern Trail Outfitters needs a low-code, near real-time REST integration to an ERP when an Order is created. They have limited development resources. Which option should fulfill the use case requirements?

A. Use Lightning Flow to create a platform event, selecting the record type as the platform event name on insert of record.

B. Implement Change Data Capture on the Order object and leverage the replay ID in the middleware solution.

C. Implement a Workflow Rule with Outbound Messaging to send SOAP messages to the designated endpoint.

Answer: A

Explanation:

For a team with limited development resources and a low-code requirement, Lightning Flow (Flow Builder) is the primary declarative engine for modern integrations.

In this scenario, creating a Platform Event via a record-triggered flow provides a highly scalable, near real-time "Fire-and-Forget" signal to the middleware. This is considered low-code because it can be built entirely using point-and-click tools within Salesforce. Once the Order is created, the flow publishes the event to the bus. The middleware, acting as a subscriber, receives the event and orchestrates the REST call to the ERP.
Option C (Outbound Messaging) is also a low-code tool, but it natively sends messages in SOAP format. Since the ERP requires a REST endpoint, using Outbound Messaging would necessitate additional transformation logic in the middleware, making it a less efficient architectural choice than using a Flow to trigger an event or an External Service. Option B (Change Data Capture) is a powerful tool but often requires more advanced "pro-code" configuration on the subscriber/middleware side to handle Replay IDs and Bayeux protocols, which might exceed the "limited resources" constraint of the Salesforce team.

NO.7 Northern Trail Outfitters uses a custom Java application to display code coverage and test results for all of its enterprise applications and plans to include Salesforce as well. Which Salesforce API should an integration architect use to meet the requirement?

A. Analytics REST API

B. Metadata API

C. Tooling API

Answer: C

Explanation:
For developer-centric tools that need to access fine-grained technical data like code coverage and test results, the Tooling API is the correct architectural choice.
While the Metadata API (Option B) is used to deploy or retrieve code, it does not provide real-time query access to the underlying metrics of a test run. The Tooling API, however, exposes specialized objects such as ApexCodeCoverage, ApexCodeCoverageAggregate, and ApexTestResult. These objects allow the Java application to query exactly which lines of code were executed during a test and the overall percentage of coverage for the organization.
The Analytics REST API (Option A) is designed for querying and interacting with Einstein Analytics (CRM Analytics) datasets and dashboards, which is irrelevant to software development lifecycle (SDLC) metrics. By using the Tooling API, the Java application can perform RESTful queries to gather comprehensive data on test successes, failures, and coverage gaps. This allows NTO to integrate Salesforce into its existing enterprise-wide quality dashboard, ensuring a unified view of code health across all platforms.

NO.8 A customer is migrating from an old legacy system to Salesforce and wants to integrate all existing systems currently working with the legacy application. Which constraint/pain-point should an integration architect consider when choosing the integration pattern/mechanism?

A. Data volume and processing volume

B. Multi-language and multi-currency requirement

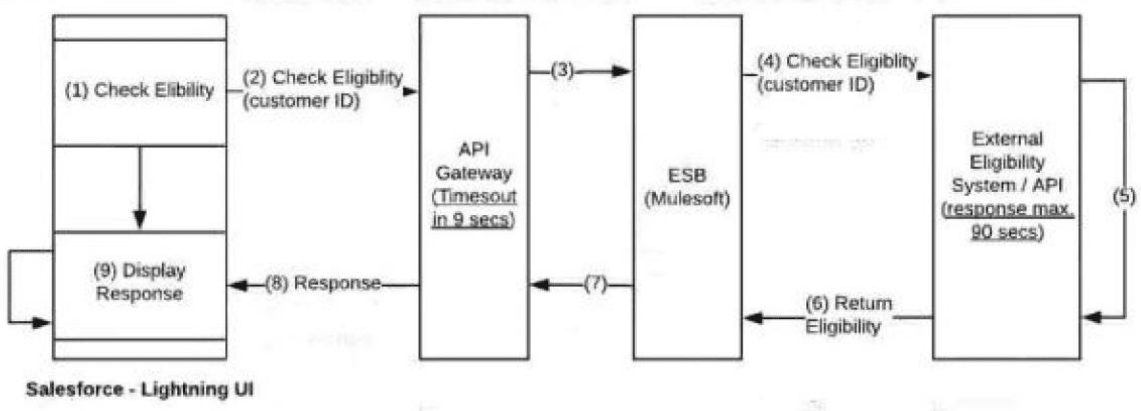
C. Reporting and usability requirements

Answer: A

Explanation:
When designing an integration architecture for a legacy migration, Data volume and processing volume are the primary technical constraints that dictate the choice of integration pattern.

Salesforce is a multi-tenant environment with strict governor limits. High data volumes can quickly exhaust synchronous request limits, API quotas, and storage allocations. An architect must evaluate:
Synchronous vs. Asynchronous: High-volume processing often requires asynchronous patterns (like Batch or Fire-and-Forget) to avoid blocking user actions and hitting concurrent request limits.
Bulk vs. REST: If millions of records need to be migrated or synchronized daily, the Bulk API is the only scalable mechanism, as standard REST or SOAP APIs are not optimized for massive datasets.
Data Persistence: Large volumes of read-only data might be better served through Data Virtualization (Salesforce Connect) to avoid consuming expensive Salesforce storage.
While reporting (Option C) and multi-currency (Option B) are important business requirements, they are functional configurations within Salesforce and do not drive the technical "plumbing" of the integration as heavily as volume does. By prioritizing the evaluation of volume and processing needs, the architect ensures that the integration is stable, performant, and capable of scaling as the business grows.

NO.9 An enterprise architect has requested the Salesforce integration architect to review the following (see diagram and description) and provide recommendations after carefully considering all constraints of the enterprise systems and Salesforce Platform limits.
About 3,000 phone sales agents use a Salesforce Lightning user interface (UI) concurrently to check eligibility of a customer for a qualifying offer.
There are multiple eligibility systems that provide this service and are hosted externally.
Their current response times could take up to 90 seconds to process and return.
These eligibility systems are accessed through APIs orchestrated via ESB (MuleSoft).
All requests from Salesforce traverse the customer's API Gateway layer, which imposes a timeout constraint of 9 seconds.



Which recommendation should the integration architect make?
A. Recommend synchronous Apex callouts from Lightning UI to External Systems via Mule and implement polling on an API Gat8eway timeout.
B. Implement a "Check Update" button that passes a requestID received from ESB (user action needed).
C. Create a platform event in Salesforce via Remote Call-In and use the empAPI in the Lightning UI to serve 3,000 concurrent users when responses are received by Mule.
Answer: C
Explanation:
In this architectural scenario, the Integration Architect must navigate two critical technical

"bottlenecks": the 9-second API Gateway timeout and the 90-second backend processing time. Since the backend takes significantly longer than the gateway allows for a synchronous connection, a standard Request-Reply pattern will fail. Furthermore, having 3,000 concurrent agents perform synchronous callouts would risk hitting Salesforce's concurrent long-running request limits.
The most scalable and user-friendly solution is to implement an Asynchronous Request-Reply pattern using Platform Events and the empAPI.
When an agent clicks "Check Eligibility," Salesforce sends an initial asynchronous request to the ESB (MuleSoft). The ESB immediately acknowledges receipt with a 202 Accepted status, freeing up the Salesforce UI thread and avoiding the API Gateway's 9-second timeout. Once the backend eligibility system completes its 90-second process, MuleSoft acts as a client to Salesforce, performing a Remote Call-In to publish a specific Platform Event containing the result and the original Request ID.
On the frontend, the Lightning UI uses the empAPI (Enterprise Messaging Platform API) to subscribe to the streaming channel for that Platform Event. Because the empAPI uses CometD technology to maintain a single long-lived connection, it can efficiently push the response to the agent's screen the moment it arrives, without requiring the agent to manually refresh or click a "Check Update" button (as suggested in Option B). This provides a "real-time" feel despite the long backend latency.
Option A is non-viable because synchronous polling would exacerbate the load on the API Gateway and likely lead to governance limit issues within Salesforce. By using Platform Events and empAPI, the architect ensures the solution remains within Salesforce's execution limits while providing a seamless, automated experience for a high-volume call center environment.

NO.10 A company needs to integrate a legacy on-premise application that can only support SOAP API. The integration architect determines that the Fire and Forget integration pattern is most appropriate for sending data from Salesforce to the external application and getting a response back in a strongly-typed format. Which integration capabilities should be used?

A. Platform Events for Salesforce to Legacy System direction and SOAP API using Enterprise WSDL for the communication back from legacy system to Salesforce

B. Outbound Messaging for Salesforce to Legacy System direction and SOAP API using Partner Web Services Description Language (WSDL) for the communication back from legacy system to Salesforce

C. Outbound Messaging for Salesforce to Legacy System direction and SOAP API using Enterprise WSDL for the communication back from legacy system to Salesforce

Answer: C

Explanation:
For an outbound, declarative, Fire-and-Forget integration to a legacy SOAP-based system, Salesforce Outbound Messaging is the native tool of choice. Outbound Messaging sends an XML message to a designated endpoint when specific criteria are met. It is highly reliable as Salesforce will automatically retry the delivery for up to 24 hours if the target system is unavailable.
For the communication back from the legacy system to Salesforce, a strongly-typed SOAP API approach is required. The Enterprise WSDL is the correct recommendation here because it is a strongly-typed WSDL that is specific to the organization's unique data model (including custom objects and fields). Using the Enterprise WSDL allows the legacy system to communicate with Salesforce using specific data types, providing compile-time safety and reducing errors during the mapping process.
Option A is less efficient because Platform Events would likely require middleware to translate the event into the legacy system's SOAP format. Option B suggests the Partner WSDL, which is loosely-typed and designed for developers building tools that must work across many different Salesforce

orgs. Since this is an internal integration for a specific company, the Enterprise WSDL provides a much more streamlined development experience with better data integrity. By combining Outbound Messaging (for fire-and-forget delivery) and the Enterprise WSDL (for the strongly-typed callback), the architect fulfills the technical requirements while minimizing custom code.

NO.11 Universal Containers (UC) support agents would like to open bank accounts on the spot. During the process, agents execute credit checks through external agencies. At any given time, up to 30 concurrent reps will be using the service. Which error handling mechanisms should be built to display an error to the agent when the credit verification process has failed?

A. Handle Integration errors in the middleware in case the verification process is down, then the middleware should retry processing the request multiple times.

B. In case the verification process is down, use fire and forget mechanism instead of Request and Reply to allow the agent to get the response back when the service is back online.

C. Handle the error in the synchronous callout and display a message to the agent. (Note: While not explicitly in the user's snippet, A and B are provided options; the standard architect answer for "displaying an error to the agent" in a synchronous flow is handling the exception in the UI layer).

Answer: A

Explanation:

In a synchronous Request-Reply scenario where a bank agent is waiting "on the spot" for a credit check, the error-handling strategy must balance immediate feedback with system resilience.

Option A is the recommended architectural approach for enterprise resiliency. By placing a Middleware layer (like MuleSoft) between Salesforce and the credit agencies, the architect can implement sophisticated error-handling patterns that are invisible to the user but critical for success. If a credit agency's API is momentarily unreachable, the middleware can perform automated retries (e.g., three attempts with 500ms intervals). If the retries still fail, the middleware sends a clean, structured error response back to Salesforce.

Option B (Fire and Forget) is fundamentally unsuitable for this use case because the agent needs the result immediately to open the account; they cannot wait for a callback that might arrive hours later.

Option C (Mock service) is only a testing tool and provides no value in a production environment where real financial data is required. By delegating the retry logic to the middleware, the architect protects Salesforce's concurrent request limits (since the agent only occupies a thread for the duration of the final response) and ensures that transient network issues do not result in a "failed" bank account application for the customer.

NO.12 Universal Containers (UC) is decommissioning its legacy CRM system and migrating data to Salesforce. The data migration team asked for a recommendation to optimize the performance of the data load. Which approach should be used to meet the requirement?

A. Contact Salesforce Support to schedule performance load.

B. Use Bulk API to process jobs in serial mode.

C. Use Bulk API to process jobs in parallel mode.

Answer: C

Explanation:

For large-scale data migrations, the Bulk API is the primary architectural tool for high-performance loading. To maximize throughput and "optimize performance," the architect should recommend processing jobs in parallel mode.

In parallel mode, Salesforce processes multiple batches of a job simultaneously, taking advantage of the multi-tenant platform's concurrent processing capabilities. This significantly reduces the total time required for a massive data migration compared to serial mode (Option B), which processes batches one by one.
However, the architect must warn the team about potential lock contention. If multiple parallel batches attempt to update the same parent record or participate in complex sharing calculations at the same time, "Unable to lock row" errors may occur. To mitigate this while maintaining parallel speed, the data should be sorted by Parent ID to ensure that batches do not overlap on the same records. Option A is rarely necessary for standard migrations unless the volume exceeds extreme thresholds. Parallel Bulk API is the standard "best practice" for ensuring the migration completes within the allotted cutover window.

NO.13 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).