Integration Patterns and Practices

Version 64.0, Summer '25

Last updated: July 11, 2025

CONTENTS

INTRODUCTION

CHAPTER 1 Integration Patterns Overview

When you implement Salesforce, you frequently need to integrate it with other applications. Although each integration scenario is unique, there are common requirements and issues that developers must resolve.

This document describes strategies (in the form of patterns) for these common integration scenarios. Each pattern describes the design and approach for a particular scenario rather than a specific implementation. In this document you'll find:

- A number of patterns that address key "archetype" integration scenarios
- A selection matrix to help you determine which pattern best fits your scenario
- Integration tips and best practices

Purpose and Scope

This document is for designers and architects who need to integrate the Lightning Platform with other applications in their enterprise. This content is a distillation of many successful implementations by Salesforce architects and partners.

Read the pattern summary and selection matrix if you're considering large-scale adoption of Salesforce-based applications (or the Lightning Platform) to become familiar with the integration capabilities and options available. Architects and developers should consider these pattern details and best practices during the design and implementation phase of a Salesforce integration project.

If implemented properly, these patterns enable you to get to production as fast as possible and have the most stable, scalable, and maintenance-free set of applications possible. Salesforce's own consulting architects use these patterns as reference points during architectural reviews and are actively engaged in maintaining and improving them.

As with all patterns, this content covers most integration scenarios, but not all. While Salesforce allows for user interface (UI) integration—mashups, for example—such integration is outside the scope of this document. If you feel that your requirements are outside the bounds of what these patterns describe, speak with your Salesforce representative.

Pattern Template

Each integration pattern follows a consistent structure. This provides consistency in the information provided in each pattern and also makes it easier to compare patterns.

Name

The pattern identifier that also indicates the type of integration contained in the pattern.

Context

The overall integration scenario that the pattern addresses. Context provides information about what users are trying to accomplish and how the application will behave to support the requirements.

Problem

The scenario or problem (expressed as a question) that the pattern is designed to solve. When reviewing the patterns, read this section to quickly understand if the pattern is appropriate for your integration scenario.

Forces

The constraints and circumstances that make the stated scenario difficult to solve.

Solution

The recommended way to solve the integration scenario.

Sketch

A UML sequence diagram that shows you how the solution addresses the scenario.

Results

Explains the details of how to apply the solution to your integration scenario and how it resolves the forces associated with that scenario. This section also contains new challenges that can arise as a result of applying the pattern.

Sidebars

Additional sections related to the pattern that contain key technical issues, variations of the pattern, pattern-specific concerns, and so on.

Example

An end-to-end scenario that describes how the design pattern is used in a real-world Salesforce scenario. The example explains the integration goals and how to implement the pattern to achieve those goals.

Pattern Summary

The following table lists the integration patterns contained in this document.

List of Patterns

Pattern	Scenario
Remote Process Invocation—Request and Reply	Salesforce invokes a process on a remote system, waits for completion of that process, and then tracks state based on the response from the remote system.
Remote Process Invocation—Fire and Forget	Salesforce invokes a process in a remote system but doesn't wait for completion of the process. Instead, the remote process receives and acknowledges the request and then hands off control back to Salesforce.

Pattern	Scenario
Batch Data Synchronization	Data stored in Lightning Platform is created or refreshed to reflect updates from an external system, and when changes from Lightning Platform are sent to an external system. Updates in either direction are done in a batch manner.
Remote Call-In	Data stored in Lightning Platform is created, retrieved, updated, or deleted by a remote system.
UI Update Based on Data Changes	The Salesforce user interface must be automatically updated as a result of changes to Salesforce data.
Data Virtualization	Salesforce accesses external data in real time. This removes the need to persist data in Salesforce and then reconcile the data between Salesforce and the external system.

Pattern Approach

The integration patterns in this document are classified into three categories:

- Data Integration—These patterns address the requirement to synchronize data that resides in two or more systems so that both systems always contain timely and meaningful data. Data integration is often the simplest type of integration to implement, but requires proper information management techniques to make the solution sustainable and cost effective. Such techniques often include aspects of master data management (MDM), data governance, mastering, de-duplication, data flow design, and others.
- Process Integration—The patterns in this category address the need for a business process to leverage two or more applications to complete its task. When you implement a solution for this type of integration, the triggering application has to call across process boundaries to other applications. Usually, these patterns also include both orchestration (where one application is the central "controller") and choreography (where applications are multi-participants and there is no central "controller"). These types of integrations can often require complex design, testing, and exception handling requirements. Also, such composite applications are typically more demanding on the underlying systems because they often support long-running transactions, and the ability to report on and/or manage process state.
- Virtual Integration—The patterns in this category address the need for a user to view, search, and modify data that's stored in an external system. When you implement a solution for this type of integration, the triggering application has to call out to other applications and interact with their data in real time. This type of integration removes the need for data replication across systems, and means that users always interact with the most current data.

Choosing the best integration strategy for your system isn't trivial. There are many aspects to consider and many tools that can be used, with some tools being more appropriate than others for certain tasks. Each pattern addresses specific critical areas including the capabilities of each of the systems, volume of data, failure handling, and transactionality.

Pattern Selection Guide

The selection matrix tables list the patterns and their key aspects to help you determine which pattern best fits your integration requirements. The patterns are categorized using these dimensions.

Aspect	Description
Type	Specifies the style of integration: Process, Data, or Virtual. • Process—Process-based integrations are ways to integrate the processing of functional flow across two or more applications. These integrations typically involve a higher level of abstraction and complexity, especially for transactionality and rollback.

Aspect	Description
	• Data—Data integrations are the integration of the information used by applications. These integrations can range from a simple table insert or upsert to complex data updates requiring referential integrity and complex translations. • Virtual—Virtual integrations are where Salesforce interacts with data that resides in an external system without the need to replicate the data that's within Salesforce. This type of integration is always triggered via an event from the Salesforce platform such as a user action, workflow, search, or record update, resulting in data integration with an external source in real time.
Timing	Specifies the blocking (or non-blocking) nature of the integration. • Synchronous—Blocking or near-real-time requests are request/response operations. The result is returned to the caller immediately via this operation.[1] • Asynchronous—Nonblocking, queue, or message-based requests are invoked by a one-way operation. The results and any faults are returned by invoking other one-way operations.[2] The caller therefore makes the request and continues without waiting for a response.

Note: An integration can require an external middleware or integration solution (for example, Enterprise Service Bus) depending on which aspects apply to your integration scenario.

Integrating Salesforce with Another System

This table lists the patterns and their key aspects to help you determine which pattern best fits your requirements when your integration scenario goes from Salesforce to another system.

Type	Timing	Key Pattern to Consider
Process Integration	Synchronous	Remote Process Invocation—Request and Reply
Process Integration	Asynchronous	Remote Process Invocation—Fire and Forget
Data Integration	Synchronous	Remote Process Invocation—Request and Reply
Data Integration	Asynchronous	UI Update Based on Data Changes
Virtual Integration	Synchronous	Data Virtualization

Integrating Another System with Salesforce

This table lists the patterns and their key aspects to help you determine the pattern that best fits your requirements when your integration scenario goes from another system to Salesforce.

[1] "Synchronous vs. Asynchronous Communication in Applications Integration," MuleSoft, last accessed October 13, 2021, *https://www.mulesoft.com/resources/esb/applications-integration*.
[2] Ibid.

Type	Timing	Key Pattern to Consider
Process Integration	Synchronous	Remote Call-In
Process Integration	Asynchronous	Remote Call-In
Data Integration	Synchronous	Remote Call-In
Data Integration	Asynchronous	Batch Data Synchronization

Middleware Terms and Definitions

This table lists some key terms related to middleware and their definitions with regards to these patterns.

Term	Definition
Event handling	Event handling is the receipt of an identifiable occurrence at a designated receiver (“handler”). The key processes involved in event handling include: • Identifying where to forward an event • Executing that forwarding action • Receiving a forwarded event • Taking some kind of appropriate action in response, such as writing to a log, sending an error or recovery process, or sending an extra message Keep in mind that the event handler can ultimately forward the event to an event consumer. Common uses of this feature with middleware can be extended to include the popular “publish/subscribe” or “pub/sub” capability. In a publish/subscribe scenario, the middleware routes requests or messages to active data-event subscribers from active data-event publishers. These consumers with active listeners can then retrieve the events as they’re published. In Salesforce integrations using middleware, the middleware layer assumes control of event handling. It collects all relevant events, synchronous or asynchronous, and manages distribution to all endpoints, including Salesforce. Alternatively, this capability can be achieved with the Salesforce enterprise messaging platform by using the event bus with platform events. See *http://searchsoa.techtarget.com/definition/event-handler*.

Term	Definition
Protocol conversion	Protocol conversion "is typically a software application that converts the standard or proprietary protocol of one device to the protocol suitable for another device to achieve interoperability. In the context of middleware, connectivity to a particular target system may be constrained by protocol. In such cases, the message format needs to be converted to or encapsulated within the format of the target system, where the payload can be extracted. This is also known as tunneling. "[3] Salesforce doesn't support native protocol conversion, so it's assumed that any such requirements are met by the middleware layer or the endpoint. See *https://www.techopedia.com/definition/30653/protocol-conversion*.
Translation and transformation	Transformation is the ability to map one data format to another to ensure interoperability between the various systems being integrated. Typically, the process entails reformatting messages en route to match the requirements of the sender or recipient. In more complex cases, one application can send a message in its own native format, and two or more other applications can each receive a copy of the message in their own native format. Middleware translation and transformation tools often include the ability to create service facades for legacy or other non-standard endpoints. These service facades allows those endpoints to appear to be service-addressable. With Salesforce integrations, it's assumed that any such requirements are met by the middleware layer or the endpoint. Transformation of data can be coded in Apex, but we don't recommend it due to maintenance and performance considerations. See *http://en.wikipedia.org/wiki/Message-oriented_middleware*.
Queuing and buffering	Queuing and buffering generally rely on asynchronous message passing, as opposed to a request-response architecture. In asynchronous systems, message queues provide temporary storage when the destination program is busy or connectivity is compromised. In addition, most asynchronous middleware systems provide persistent storage to back up the message queue. The key benefit of an asynchronous message process is that if the receiver application fails for any reason, the senders can continue unaffected. The sent messages simply accumulate in the message queue for later processing when the receiver restarts. Salesforce provides only explicit queuing capability in the form of workflow-based outbound messaging. To provide true message queueing for other integration scenarios, including orchestration, process choreography, and quality of service, a middleware solution is required. See *http://en.wikipedia.org/wiki/Message-oriented_middleware*.
Synchronous transport protocols	Synchronous transport protocols refer to protocols that support activities wherein a "single thread in the caller sends the request message, blocks to wait for the reply message, and then processes the reply....The request thread awaiting the response implies that there is only one outstanding request or that the reply channel for this request is private for this thread."[4]

[3] Gregor Hohpe, and Bobby Woolf, *Enterprise Integration Patterns* (Boston: Addison-Wesley Professional, 2003).
[4] Gregor Hohpe, and Bobby Woolf, *Enterprise Integration Patterns* (Boston: Addison-Wesley Professional, 2003).

Term	Definition
Asynchronous transport protocols	Asynchronous transport protocols refer to protocols supporting activities wherein "one thread in the caller sends the request message and sets up a callback for the reply. A separate thread listens for reply messages. When a reply message arrives, the reply thread invokes the appropriate callback, which reestablishes the caller's context and processes the reply. This approach enables multiple outstanding requests to share a single reply thread."[5]
Mediation routing	Mediation routing is the specification of a complex "flow" of messages from component to component. For example, many middleware-based solutions depend on a message queue system. While some implementations permit routing logic to be provided by the messaging layer itself, others depend on client applications to provide routing information or allow for a mix of both paradigms. In such complex cases, mediation on the part of middleware simplifies development, integration, and validation. "Specifically, Mediators coordinate a group of objects so that they [do not] need to know how to coordinate with each other….Then, each consumer could focus on processing a particular kind of message, and the coordinator [Mediator] could make sure that the right message gets to the right consumer."[6]
Process choreography and service orchestration	Process choreography and service orchestration are each forms of "service composition" where any number of endpoints and capabilities are being coordinated. The difference between choreography and service orchestration is: • Choreography can be defined as an "asynchronous approach that enables processes to work autonomously, removing any issues caused by dependencies,"[7] which is behavior resulting from a group of interacting individual entities with no central authority. • Orchestration can be defined as a "synchronous approach that enables the execution of a query or process by allowing each microservice to implement tasks assigned by the orchestrator,"[8] which is behavior resulting from a central conductor coordinating the behaviors of individual entities performing tasks independent of each other. In addition, "an orchestration shows the complete behavior of each service whereas the choreography combines the interface behavior descriptions of each service."[9] Portions of business process choreographies can be built in Salesforce workflows or using Apex. We recommend that all complex orchestrations be implemented in the middleware layer because of Salesforce timeout values and governor limits, especially in solutions requiring true transaction handling.
Transactionality (encryption, signing, reliable delivery, transaction management)	Transactionality can be defined as the ability to support global transactions that encompass all necessary operations against each required resource. Transactionality implies the support of all four ACID properties, atomicity, consistency, isolation, durabilit, where atomicity guarantees all-or-nothing outcomes for the unit of work (transaction).

[5] Ibid.
[6] Ibid.
[7] "Microservice orchestration or choreography?" architect.io, last accessed Sep 20, 2023, *https://www.architect.io/blog/2022-06-30/microservices-orchestration-primer/*.
[8] Ibid.
[9] "Orchestration vs. Choreography," Stack Overflow, last accessed Sep 27, 2023, *http://stackoverflow.com/questions/4127241/orchestration-vs-choreography*.

Term	Definition
	While Salesforce is transactional within itself, it's not able to participate in distributed transactions or transactions initiated outside of Salesforce. Therefore, it's assumed that for solutions requiring complex, multi-system transactions, transactionality and associated roll-back/compensation mechanisms are implemented at the middleware layer. See *http://en.wikipedia.org/wiki/Distributed_transaction*.
Routing	Routing can be defined as specifying the complex flow of messages from component to component. In modern services-based solutions, such message flows can be based on a number of criteria, including header, content type, rule, and priority. With Salesforce integrations, it's assumed that any such requirements are met by the middleware layer or the endpoint. Message routing can be coded in Apex, but we don't recommend it due to maintenance and performance considerations.
Extract, transform, and load	Extract, transform, and load (ETL) refers to a process that involves: • Extracting data from the source systems. This process typically involves data from several source systems, and both relational and non-relational structures. • Transforming the data to fit operational needs, which can include data quality levels. The transform stage usually applies a series of rules or functions to the extracted data from the source to derive the data for loading into the end target(s). • Loading the data into the target system. The target system can vary widely from database, operational data store, data mart, data warehouse, or other operational systems. While not strictly necessary, most mature ETL tools provide a change data capture capability. This capability is where the tool identifies records in the source system that have changed since the last extract, which reduces the amount of record processing. Salesforce now also supports Change Data Capture, which is the publishing of change events that represent changes to Salesforce records. With Change Data Capture, the client or external system receives near-real-time changes of Salesforce records. With this information, the client or external system can synchronize corresponding records in an external data store. See *http://en.wikipedia.org/wiki/Extract,_transform,_load* and *Change Data Capture Developer Guide*.
Long polling	Long polling, also called Comet programming, emulates an information push from a server to a client. Similar to a normal poll, the client connects and requests information from the server. However, instead of sending an empty response if information isn't available, the server holds the request and waits until information is available (an event occurs). The server then sends a complete response to the client. The client then immediately re-requests information. The client continually maintains a connection to the server, so it's always waiting to receive a response. If the server times out, the client connects again and starts over. The Salesforce Streaming API uses the Bayeux protocol and CometD for long polling. • Bayeux is a protocol for transporting asynchronous messages, primarily over HTTP. • CometD is a scalable HTTP-based event routing bus that uses an AJAX push technology pattern known as Comet. It implements the Bayeux protocol.

CHAPTER 2 Remote Process Invocation—Request and Reply

Context

You use Salesforce to track leads, manage your pipeline, create opportunities, and capture order details that convert leads to customers. But, the Salesforce system doesn't contain or process orders. After the order details are captured in Salesforce, the order is created in the remote system, which manages the order to conclusion.

When you implement this pattern, Salesforce calls the remote system to create the order and then waits for successful completion. If successful, the remote system synchronously replies with the order status and order number. As part of the same transaction, Salesforce updates the order number and status internally. The order number is used as a foreign key for subsequent updates to the remote system.

Problem

When an event occurs in Salesforce, how do you initiate a process in a remote system, pass the required information to that process, receive a response from the remote system, and then use that response data to make updates within Salesforce?

Forces

Consider the following forces when applying solutions based on this pattern.

- Does the call to the remote system require Salesforce to wait for a response before continuing processing? Is the call to the remote system a synchronous request-reply or an asynchronous request?
- If the call to the remote system is synchronous, does Salesforce have to process the response as part of the same transaction as the initial call?
- Is the message size small or large?
- Is the integration based on the occurrence of a specific event, such as a button click in the Salesforce user interface, or DML-based events?
- Is the remote endpoint able to respond to the request with low latency? How many users are likely to be executing this transaction during a peak period?

Solution

This table contains solutions to this integration problem.

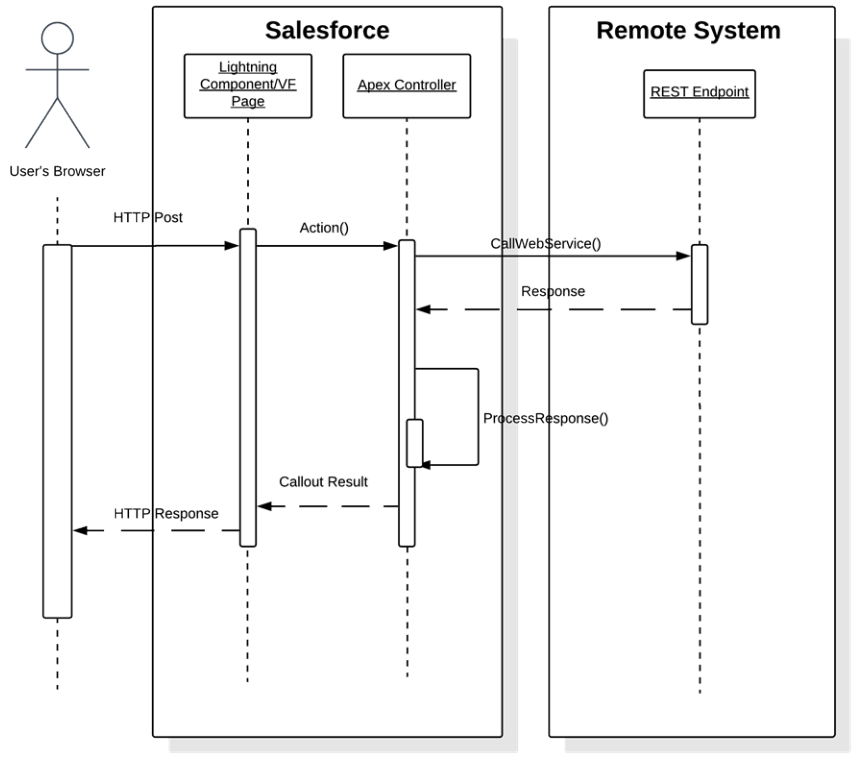
Solution	Fit	Comments
Enhanced External Services invokes a REST API call	Best	Enhanced External Services allows you to invoke an externally hosted service in a declarative manner (no code required). This feature is best used when the following conditions are met: • The externally hosted service is a RESTful service and the definitions are available in an OpenAPI 2.0 JSON schema format. • The request and response definitions contain primitive data types such as boolean, datetime, double, integer, string, or an array of primitive data types. Nested object types, and send parameters such as headers within the HTTP requests are supported. • The transaction can be invoked from a flow.
Salesforce Lightning—Lightning component or page initiates a synchronous Apex SOAP or REST callout. Salesforce Classic—A custom Visualforce page or button initiates a synchronous Apex SOAP callout. If the remote endpoint poses a risk of high latency response (refer to latest limits documentation for the applicable limits here), then an asynchronous callout, also called a continuation, is recommended to avoid hitting synchronous Apex transaction governor limits.	Best	Salesforce enables you to consume a WSDL and generate a resulting proxy Apex class. This class provides the necessary logic to call the remote service. Salesforce also enables you to invoke HTTP (REST) services using standard GET, POST, PUT, and DELETE methods. A user-initiated action on a Visualforce page or Lightning page then calls an Apex controller action that then executes this proxy Apex class to perform the remote call. Visualforce pages and Lightning pages require customization of the Salesforce application.
A custom Visualforce page or button initiates a synchronous Apex HTTP callout.	Best	Salesforce enables you to invoke HTTP services using standard GET, POST, PUT, and DELETE methods. You can use several HTTP classes to integrate with RESTful services. It's also possible to integrate to SOAP-based services by manually constructing the SOAP message. The latter isn't recommended because it's possible for Salesforce to consume WSDLs to generate proxy classes. A user-initiated action on a Visualforce page then calls an Apex controller action that then executes this proxy Apex class to perform the remote call. Visualforce pages require customization of the Salesforce application.
A synchronous trigger that's invoked from Salesforce data changes performs an asynchronous Apex SOAP or HTTP callout.	Suboptimal	You can use Apex triggers to perform automation based on record data changes. An Apex proxy class can be executed as the result of a DML operation by using an Apex trigger. However, all calls made from within the trigger context must execute asynchronously from the initiating event. Therefore, this solution isn't recommended for this integration problem. This solution is

Solution	Fit	Comments
		better suited for the Remote Process Invocation—Fire and Forget pattern.
A batch Apex job performs a synchronous Apex SOAP or HTTP callout.	Suboptimal	You can make calls to a remote system from a batch job. This solution allows batch remote process execution and processing of the response from the remote system in Salesforce. However, a given batch has limits to the number of calls. For more information, see Governor Limits. A given batch run can execute multiple transaction contexts (usually in intervals of 200 records). The governor limits are reset per transaction context.

Sketch

This diagram illustrates a synchronous remote process invocation using Apex calls.

Salesforce Calling Out to a Remote System



In this scenario:

1. An action is initiated on the Visualforce or Lightning page (for example, a button click).
2. The browser (via a client-side controller in the case of a Lightning component) performs an HTTP POST that in turn performs an action on the corresponding Apex controller.

3. The controller invokes the actual call to the remote web service.
4. The response from the remote system is returned to the Apex controller. The controller processes the response, updates data in Salesforce as required, and re-renders the page.

In cases where the subsequent state must be tracked, the remote system returns a unique identifier that's stored in the Salesforce record.

Results

The application of the solutions related to this pattern allows for event-initiated remote process invocations in which Salesforce handles the processing.

Calling Mechanisms

The calling mechanism depends on the solution chosen to implement this pattern.

Calling Mechanism	Description
Enhanced External service embedded in a flow or Lightning component or Visualforce and Apex controllers	Used when the remote process is triggered as part of an end-to-end process involving the user interface, and the result must be displayed or updated in a Salesforce record. For example, the submission of a credit card payment to an external payment gateway and the payment results are immediately returned and displayed to the user.
Apex triggers	Used primarily for invocation of remote processes using Apex callouts from DML-initiated events. For more information about this calling mechanism, see pattern Remote Process Invocation—Fire and Forget.
Apex batch classes	Used for invocation of remote processes in batch. For more information about this calling mechanism, see pattern Remote Process Invocation—Fire and Forget.

Error Handling and Recovery

It's important to include an error handling and recovery strategy as part of the overall solution.

- Error handling—When an error occurs (exceptions or error codes are returned to the caller), the caller manages error handling. For example, an error message displayed on the end user's page or logged to a table requiring further action.
- Recovery—Changes aren't committed to Salesforce until the caller receives a successful response. For example, the order status isn't updated in the database until a response that indicates success is received. If necessary, the caller can retry the operation.

Idempotent Design Considerations

Idempotent capabilities guarantee that repeated invocations are safe. If idempotency isn't implemented, repeated invocations of the same message can have different results, potentially resulting in data integrity issues. Potential issues include the creation of duplicate records or duplicate processing of transactions.

It's important to ensure that the remote procedure being called is idempotent. It's almost impossible to guarantee that Salesforce only calls one time, especially if the call is triggered from a user interface event. Even if Salesforce makes a single call, there's no guarantee that other processes (for example, middleware) do the same.

The most typical method of building an idempotent receiver is for it to track duplicates based on unique message identifiers sent by the consumer. Apex web service or REST calls must be customized to send a unique message ID.

In addition, operations that create records in the remote system must check for duplicates before inserting. Check by passing a unique record ID from Salesforce. If the record exists in the remote system, update the record. In most systems, this operation is termed as an upsert operation.

Security Considerations

Any call to a remote system must maintain the confidentiality, integrity, and availability of the request. The following security considerations are specific to using Apex SOAP and HTTP calls in this pattern.

- One-way SSL is enabled by default, but two-way SSL is supported with both self-signed and CA-signed certificates to maintain authenticity of both the client and server.
- Salesforce currently doesn't support WS-Security.
- Where necessary, consider using one-way hashes or digital signatures using the Apex Crypto class methods to ensure request integrity.
- The remote system must be protected by implementing the appropriate firewall mechanisms.

See Security Considerations.

Sidebars

Timeliness

Timeliness is of significant importance in this pattern. Usually:

- The request is typically invoked from the user interface, so the process must not keep the user waiting.
- Salesforce has a configurable timeout of up to 120 seconds for calls from Apex.
- Completion of the remote process is executed in a timely manner to conclude within the Salesforce timeout limit and within user expectations.
- External calls are subject to Apex synchronous transaction governor limits, so make sure to mitigate the risk of instantiating more than 10 transactions that run for more than five seconds each. In addition to ensuring the external endpoint is performant, options to mitigate the risk of a timeout include:
  - Setting the timeout of the external callout to five seconds
  - Using a continuation in Visualforce or Lightning Components to handle long-running transactions

Data Volumes

This pattern is used primarily for small volume, real-time activities, due to the small timeout values and maximum size of the request or response for the Apex call solution. Don't use this pattern in batch processing activities in which the data payload is contained in the message.

Endpoint Capability and Standards Support

The capability and standards support for the endpoint depends on the solution that you choose.

Solution	Endpoint Considerations
Apex SOAP callouts	The endpoint must be able to receive a web service call via HTTP. Salesforce must be able to access the endpoint over the public Internet. This solution requires that the remote system is compatible with the standards supported by Salesforce. At the time of writing, the web service standards supported by Salesforce for Apex SOAP callouts are: • WSDL 1.1 • SOAP 1.1 • WSI-Basic Profile 1.1

Solution	Endpoint Considerations
	• HTTP
Apex HTTP callouts	The endpoint must be able to receive HTTP calls. Salesforce must be able to access the endpoint over the public Internet. You can use Apex HTTP callouts to call REST services using the standard GET, POST, PUT, and DELETE methods.

State Management

When integrating systems, keys are important for ongoing state tracking. There are two options.

- Salesforce stores the remote system's primary or unique surrogate key for the remote record.
- The remote system stores the Salesforce unique record ID or some other unique surrogate key.

There are specific considerations for handling integration keys, depending on which system contains the master record, as shown in the following table.

Master System	Description
Salesforce	The remote system stores either the Salesforce RecordId or some other unique surrogate key from the record.
Remote system	The call to the remote process returns the unique key from the application, and Salesforce stores that key value in a unique record field.

Complex Integration Scenarios

In certain cases, the solution prescribed by this pattern can require the implementation of several complex integration scenarios. This is best served by using middleware or having Salesforce call a composite service. These scenarios include:

- Orchestration of business processes and rules involving complex flow logic
- Aggregation of calls and their results across calls to multiple systems
- Transformation of both inbound and outbound messages
- Maintaining transactional integrity across calls to multiple systems

Governor Limits

For information about Apex limits, see *Execution Governors and Limits* in the Apex Developer Guide.

Middleware Capabilities

The following table highlights the desirable properties of a middleware system that participates in this pattern.

Property	Mandatory	Desirable	Not Required
Event handling		X	
Protocol conversion		X	
Translation and transformation		X	
Queuing and buffering		X	

Property	Mandatory	Desirable	Not Required
Synchronous transport protocols	X		
Asynchronous transport protocols			X
Mediation routing		X	
Process choreography and service orchestration		X	
Transactionality (encryption, signing, reliable delivery, transaction management)		X	
Routing			X
Extract, transform, and load			X
Long polling			X

Example

A utility company uses Salesforce and has a separate system that contains customer billing information. They want to display the billing history for a customer account without storing that data in Salesforce. They have an existing web service that returns a list of bills and the details for a given account, but can't display this data in a browser.

This requirement can be accomplished with the following approach.

1. Salesforce consumes the billing history service WSDL from an Apex proxy class.
2. Execute the Apex proxy class with the account number as the unique identifier by creating a Lightning component and a custom controller or a Visualforce page and custom controller.
3. The custom controller then parses the return values from the Apex callout and the Lightning component or Visualforce page, and then renders the customer billing data to the user.

This example demonstrates that:

- The state of the customer is tracked with an account number stored on the Salesforce account object
- Subsequent processing of the reply message by the caller

CHAPTER 3 Remote Process Invocation—Fire and Forget

Context

You use Salesforce to track leads, manage your pipeline, create opportunities, and capture order details that convert leads to customers. However, Salesforce isn't the system that holds or processes orders. After the order details are captured in Salesforce, an order must be created in the remote system, which manages the order through to its conclusion.

When you implement this pattern, Salesforce calls the remote system to create the order, but doesn't wait for the call's successful completion. The remote system can optionally update Salesforce with the new order number and status in a separate transaction.

Problem

When an event occurs in Salesforce, how do you initiate a process in a remote system and pass the required information to that process without waiting for a response from the remote system?

Forces

Consider the following forces when applying solutions based on this pattern.

- Does the call to the remote system require Salesforce to wait for a response before continuing processing? Is the call to the remote system synchronous or asynchronous?
- If the call to the remote system is synchronous, does the response need to be processed by Salesforce as part of the same transaction as the call?
- Is the message size small?
- Is the integration based on the occurrence of a specific event, such as a button click in the Salesforce user interface, or DML-based events?
- Is guaranteed message delivery from Salesforce to the remote system a requirement?
- Is the remote system able to participate in a contract-first integration in which Salesforce specifies the contract? In some solution variants (for example, outbound messaging), Salesforce specifies a contract that the remote system endpoint implements.
- Does the endpoint or the Enterprise Service Bus (ESB) support long polling?
- Are declarative configuration methods preferred over custom Apex development? In this case, solutions such as platform events are preferred over Apex callouts.

Solution

The following table contains solutions to this integration problem.

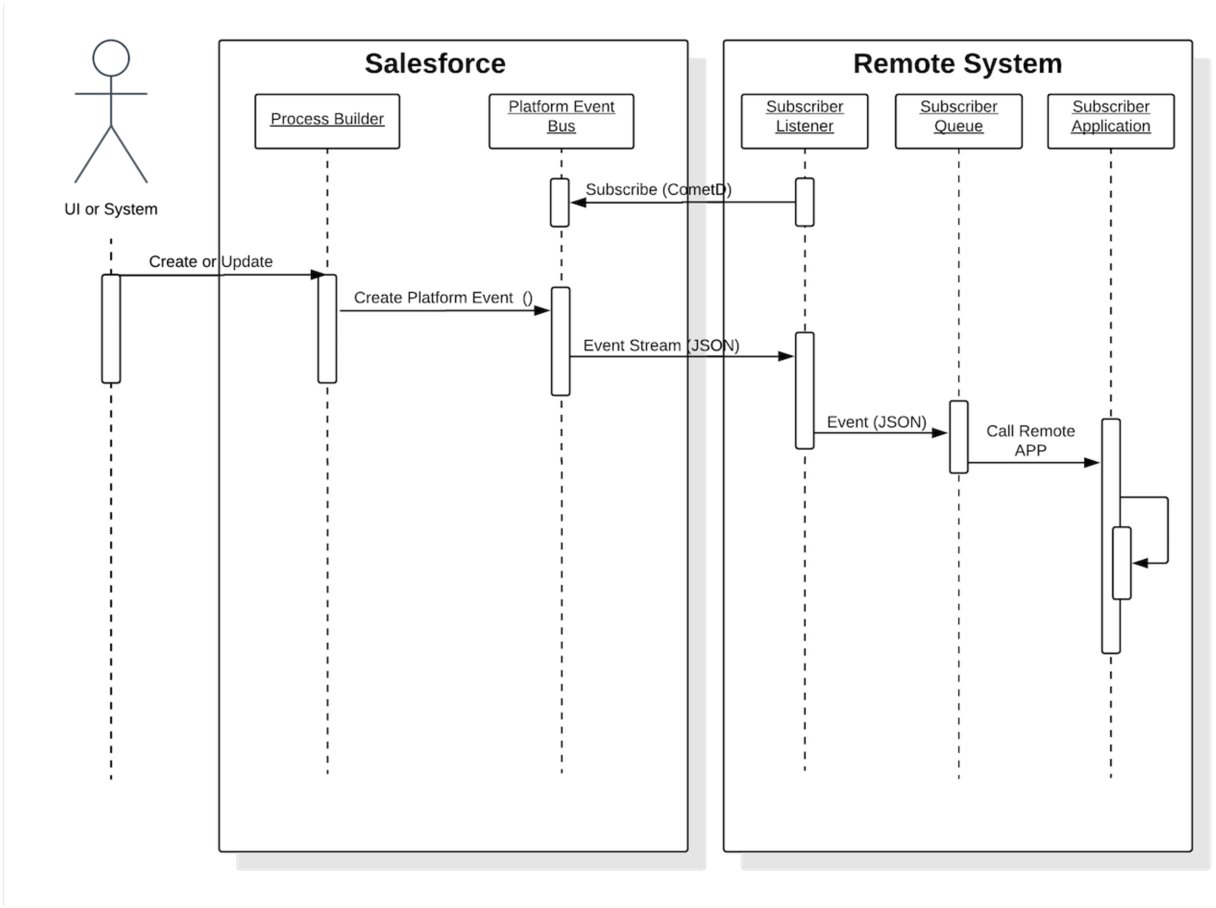
Solution	Fit	Comments
Process-driven platform events	Best	No customization is required in Salesforce to implement platform events. The recommended solution is when the remote process is invoked from an insert or update event. Platform events are event messages (or notifications) that your apps send and receive to take further action. Platform events simplify the process of communicating changes and responding to them without writing complex logic. One or more subscribers can listen to the same event and carry out actions. For example, a software system can send events containing information about printer ink cartridges. Subscribers can subscribe to the events to monitor printer ink levels and place orders to replace cartridges with low ink levels. External apps can listen to event messages by subscribing to a channel through CometD. Platform apps, such as Visualforce pages and Lightning components, can subscribe to event messages with CometD as well.
Customization-driven platform events	Good	Similar to process-driven platform events, but the events are created by Apex triggers or classes. You can publish and consume platform events by using Apex or an API. Platform events integrate with the Salesforce platform through Apex triggers. Triggers are the event consumers on the Salesforce platform that listen to event messages. When an external app uses the API or a native Salesforce app uses Apex to publish the event message, a trigger on that event is fired. Triggers run the actions in response to the event notifications.
Workflow-driven outbound messaging	Good	No customization is required in Salesforce to implement outbound messaging. The recommended solution for this type of integration is when the remote process is invoked from an insert or update event. Salesforce provides a workflow-driven outbound messaging capability that allows sending SOAP messages to remote systems triggered by an insert or update operation in Salesforce. These messages are sent asynchronously and are independent of the Salesforce user interface. The outbound message is sent to a specific remote endpoint. The remote service must be able to participate in a contract-first integration where Salesforce provides the contract. On receipt of the message, if the remote service doesn't respond with a positive acknowledgment, Salesforce retries sending the message, providing a form of guaranteed delivery.

Solution	Fit	Comments
		When using middleware, this solution becomes a “first-mile” guarantee of delivery.
Outbound messaging and callbacks	Good	Callbacks provide a way to mitigate the impacts of out-of-sequence messaging. In addition, they handle these scenarios. • Idempotency— If an acknowledgment isn’t received in a timely fashion, outbound messaging performs retries. Multiple messages can be sent to the target system. Using a callback ensures that the data retrieved is at a specific point in time rather than when the message was sent. • Retrieving more data—A single outbound message can send data only for a single object. A callback can be used to retrieve data from other related records, such as related lists associated with the parent object. The outbound message provides a unique SessionId that you can use as an authentication token to authenticate and authorize a callback with either the SOAP API or the REST API. The system performing the callback isn’t required to separately authenticate to Salesforce. The standard methods of either API can then be used to perform the desired business functions. A typical use of this variant is the scenario in which Salesforce sends an outbound message to a remote system to create a record. The callback updates the original Salesforce record with the unique key of the record created in the remote system.
Custom Lightning component or Visualforce page that initiates an Apex SOAP or HTTP asynchronous callout	Suboptimal	This solution is typically used in user interface-based scenarios, but does require customization. In addition, the solution must handle guaranteed delivery of the message in the code. Similar to the solution for the Remote Process Invocation—Request and Reply pattern solution that specifies using a Visualforce page or Lightning component, together with an Apex callout. The difference is that in this pattern, Salesforce doesn’t wait for the request to complete before handing off control to the user. After receiving the message, the remote system responds and indicates receipt of the message, then asynchronously processes the message. The remote system hands control back to Salesforce before it begins to process the message; therefore, Salesforce doesn’t have to wait for processing to complete.
Trigger that’s invoked from Salesforce data changes performs an Apex SOAP or HTTP asynchronous callout	Suboptimal	You can use Apex triggers to perform automation based on record data changes. An Apex proxy class can be executed as the result of a DML operation by using an Apex trigger. However, all calls made

Solution	Fit	Comments
		from within the trigger context must be executed asynchronously.
Batch Apex job that performs an Apex SOAP or HTTP asynchronous callout	Suboptimal	Calls to a remote system can be performed from a batch job. This solution allows for batch remote process execution and for processing of the response from the remote system in Salesforce. However, there are limits to the number of calls for a given batch context. For more information, see the *Salesforce Developer Limits and Allocations Quick Reference*.

Sketch

The following diagram illustrates a call from Salesforce to a remote system in which create or update operations on a record trigger the call.



In this scenario:

1. A remote system subscribes to the platform event.
2. A update or insert occurs on a given set of records in Salesforce.
3. A Salesforce Process triggers when a set of conditions is met.

4. This process creates a platform event.
5. The remote listener receives the event message, and places the message on a local queue.
6. The queuing application forwards the message to the remote application for processing.

In the case where the remote system must perform operations against Salesforce, you can implement an optional callback operation.

Results

The application of the solutions related to this pattern allows for:

- User interface–initiated remote process invocations in which the result of the transaction can be displayed to the end user
- DML event-initiated remote process invocations in which the result of the transaction can be processed by the calling process

Calling Mechanisms

The calling mechanism depends on the solution chosen to implement this pattern.

Calling Mechanism	Description
Process Builder	Used by both the process-driven and customization-driven solutions. Events trigger the Salesforce process, which can then publish a platform event for subscription by a remote system.
Lightning component or Visualforce and Apex controllers	Used to invoke a remote process asynchronously using an Apex callout.
Workflow rules	Used only for the outbound messaging solution. Create and update DML events trigger the Salesforce workflow rules, which can then send a message to a remote system.
Apex triggers	Used for trigger-driven platform events and invocation of remote processes, using Apex callouts from DML-initiated events.
Apex batch classes	Used for invocation of remote processes in batch mode.

Error Handling and Recovery

An error handling and recovery strategy must be considered as part of the overall solution. The best method depends on the solution you choose.

Solution	Error Handling and Recovery Strategy
Apex callouts	• *Error handling*—The remote system hands off invocation of the end process, so the callout only handles exceptions in the initial invocation of the remote service. For example, a timeout event is triggered if no positive acknowledgment is received from the remote callout. The remote system must handle subsequent errors when the initial invocation is handed off for asynchronous processing. • *Recovery*—Recovery is more complex in this scenario. A custom retry mechanism must be created if quality-of-service requirements dictate it.
Outbound messaging	• *Error handling*—Because this pattern is asynchronous, the remote system handles error handling. For outbound messaging, Salesforce initiates a retry operation if no positive acknowledgment is received within the timeout period, for up to 24 hours.

Solution	Error Handling and Recovery Strategy
	Error handling must be performed in the remote service because the message is effectively handed off to the remote system in a "fire-and-forget" manner. • *Recovery*—Because this pattern is asynchronous, the system must initiate retries based on the service's quality-of-service requirements. For outbound messaging, Salesforce initiates retries if no positive acknowledgment is received from the outbound listener within the timeout period, up to 24 hours. The retry interval increases exponentially over time, starting with 15-second intervals and ending with 60-minute intervals. The timeout period can be extended to seven days by request to Salesforce support, but automatic retries are limited to 24 hours. All failed messages after 24 hours are placed in a queue and administrators must monitor this queue for any messages exceeding the 24-hour delivery period and retry manually, if necessary.
Platform Events	• *Error handling*—Error handling must be performed by the remote service because the event is effectively handed off to the remote system for further processing. Because this pattern is asynchronous, the remote system handles message queuing, processing, and error handling. Additionally, platform events aren't processed within database transactions. As a result, published platform events can't be rolled back within a transaction. • *Recovery*—Because this pattern is asynchronous, the remote system must initiate retries based on the service's quality-of-service requirements. The replay ID associated with each event is atomic and increases with every published event. This ID can be used replay the stream from a specific event (for example, based upon the last successfully captured event). High-volume platform event messages are stored for 72 hours (three days). You can retrieve past event messages when using CometD clients to subscribe to a channel.

Idempotent Design Considerations

Platform events are only published to the bus once. There is no retry on the Salesforce side. It is up to the ESB to request that the events be replayed. In a replay, the platform event's replay ID remains the same and the ESB can try duplicate messages based on the replay ID.

Idempotency is important for outbound messaging because it's asynchronous and retries are initiated when no positive acknowledgment is received. Therefore, the remote service must be able to handle messages from Salesforce in an idempotent fashion.

Outbound messaging sends a unique ID per message and this ID remains the same for any retries. The remote system can track duplicate messages based on this unique ID. The unique record ID for each record being updated is also sent, and can be used to prevent duplicate record creation.

The idempotent design considerations in the Remote Process Invocation—Request and Reply pattern also apply to this pattern.

Security Considerations

Any call to a remote system must maintain the confidentiality, integrity, and availability of the request. Different security considerations apply, depending on the solution you choose.

Solution	Security Considerations
Apex callouts	A call to a remote system must maintain the confidentiality, integrity, and availability of the request. The following are security considerations specific to using Apex SOAP and HTTP calls in this pattern. • One-way SSL is enabled by default, but two-way SSL is supported with both self-signed and CA-signed certificates to maintain authenticity of both the client and server.

Solution	Security Considerations
	• Salesforce does not support WS-Security when generating the Apex proxy class. • Where necessary, consider using one-way hashes or digital signatures using the Apex Crypto class methods to ensure the integrity of the request. • The remote system must be protected by implementing the appropriate firewall mechanisms.
Outbound Messaging	For outbound messaging, one-way SSL is enabled by default. However, two-way SSL can be used together with the Salesforce outbound messaging certificate. The following are some additional security considerations. • Whitelist Salesforce server IP ranges for remote integration servers. • Protect the remote system by implementing the appropriate firewall mechanisms.
Platform Events	For platform events the subscribing external system must be able to authenticate to the Salesforce Streaming API. Platform events conform to the existing security model configured in the Salesforce org. To subscribe to an event, the user needs read access to the event entity. To publish an event, the user needs create permission on the event entity.

See Security Considerations.

Sidebars

Timeliness

Timeliness is less of a factor with the fire-and-forget pattern. Control is handed back to the client either immediately or after positive acknowledgment of a successful hand-off to the remote system. With Salesforce outbound messaging, the acknowledgment must occur within 24 hours (this can be extended to seven days); otherwise, the message expires. For platform events, Salesforce sends the events to the event bus and doesn't wait for a confirmation or acknowledgment from the subscriber. If the subscriber doesn't pick up the message, the subscriber can request to replay the event using the event reply ID. High-volume event messages are stored for 72 hours (three days). To retrieve past event messages, use CometD clients to subscribe to a channel.

Data Volumes

Data volume considerations depend on which solution you choose. For the limits of each solution, see the *Salesforce Limits Quick Reference Guide*.

Endpoint Capability and Standards Support

The capability and standards support for the endpoint depends on the solution that you choose.

Solution	Endpoint Considerations
Apex SOAP callouts	The endpoint must be able to process a web service call via HTTP. Salesforce must be able to access the endpoint over the public Internet. This solution requires that the remote system is compatible with the standards supported by Salesforce. At the time of writing, the web service standards supported by Salesforce for Apex SOAP callouts are: • WSDL 1.1 • SOAP 1.1 • WSI-Basic Profile 1.1 • HTTP
Apex HTTP callouts	The endpoint must be able to receive HTTP calls and be accessible over the public internet by Salesforce. Apex HTTP callouts can be used to call RESTful services using the standard GET, POST, PUT, and DELETE methods.
Outbound message	• The endpoint must be able to implement a listener that can receive SOAP messages in predefined format sent from Salesforce. • The remote listener must participate in a contract-first implementation, where the contract is supplied by Salesforce. • Each outbound message has its own predefined WSDL.
Platform Events	• Triggers, processes and flows can subscribe to events. You can receive event notifications regardless of how they were published. • Use CometD to subscribe to platform events from an external client. Implement your own CometD client or use EMP Connector, an open-source, community-supported tool that implements all the details of connecting to CometD and listening on a channel. Salesforce sends platform events to CometD clients sequentially in the order they're received. The order of event notifications is based on the replay ID of events.

State Management

When integrating systems, unique record identifiers are important for ongoing state tracking. For example, if a record is created in the remote system, you have two options.

- Salesforce stores the remote system's primary or unique surrogate key for the remote record.
- The remote system stores the Salesforce unique record ID or some other unique surrogate key.

The following table lists considerations for state management in this pattern.

Master System	Description
Salesforce	The remote system must store either the Salesforce RecordId or some other unique surrogate key in the Salesforce record.

Master System	Description
Remote system	Salesforce must store a reference to the unique identifier in the remote system. Because the process is asynchronous, storing this unique identifier can't be part of the original transaction. Salesforce must provide a unique ID in the call to the remote process. The remote system must then call back to Salesforce to update the record in Salesforce with the remote system's unique identifier, using the Salesforce unique ID. The callback implies specific state handling in the remote application to store the Salesforce unique identifier for that transaction to use for the callback when processing is complete, or the Salesforce unique identifier is stored on the remote system's record.

Complex Integration Scenarios

Each solution in this pattern has different considerations for complex integration scenarios such as transformation and process orchestration.

Solution	Considerations
Apex callouts	In certain cases, solutions prescribed by this pattern require implementing several complex integration scenarios best served using middleware or having Salesforce call a composite service. These scenarios include: • Orchestration of business processes and rules involving complex flow logic • Aggregation of calls and their results across calls to multiple systems • Transformation of both inbound and outbound messages • Maintaining transactional integrity across calls to multiple systems
Outbound messaging	Given the static, declarative nature of the outbound message, no complex integration scenarios, such as aggregation, orchestration, or transformation, can be performed in Salesforce. The remote system or middleware must handle these types of operations .
Platform Events	Given the static, declarative nature of events, no complex integration scenarios, such as aggregation, orchestration, or transformation can be performed in Salesforce. The remote system or middleware must handle these types of operations .

Governor Limits

Due to the multitenant nature of the Salesforce platform, there are limits to outbound callouts. Limits depend on the type of outbound call and the timing of the call.

In case of platform events, different limits and allocations apply. See the *Platforms Events Developer Guide*.

There are no governor limits for outbound messaging. See the *Salesforce Limits Quick Reference Guide*.

Reliable Messaging

Reliable messaging attempts to resolve the issue of guaranteeing the delivery of a message to a remote system in which the individual components are unreliable. The method of ensuring receipt of a message by the remote system depends on the solution you choose.

Solution	Reliable Messaging Considerations
Apex callouts	Salesforce doesn't provide explicit support for reliable messaging protocols (for example, WS-ReliableMessaging). We recommend that the remote endpoint receiving the Salesforce message implement a reliable messaging system, like JMS or MQ. This system ensures full end-to-end guaranteed delivery to the remote system that ultimately processes the message. However, this system doesn't ensure guaranteed delivery from Salesforce to the remote endpoint that it calls. Guaranteed delivery must be handled through customizations to Salesforce. Specific techniques, such as processing a positive acknowledgment from the remote endpoint in addition to custom retry logic, must be implemented.
Outbound messaging	Outbound messaging provides a form of reliable messaging. If no positive acknowledgment is received from the remote system, the process retries for up to 24 hours. This process guarantees delivery only to the point of the remote listener. The retry interval increases exponentially over time, starting with 15-second intervals and ending with 60-minute intervals. The overall retry period can be extended to seven days by request to Salesforce support, but automatic retries are limited to 24 hours. All failed messages after 24 hours are placed in a queue and administrators must monitor this queue for any messages exceeding the 24-hour delivery period and retry manually, if necessary. In most implementations, the remote listener calls another remote service. Ideally, the invocation of this remote service through a reliable messaging system ensures full end-to-end guaranteed delivery. The positive acknowledgment to the Salesforce outbound message occurs after the remote listener has successfully placed its own message on its local queue. Once the positive acknowledgment is received by Salesforce, automatic retries are stopped.
Platform Events	Platform Events attempts to provide reliable messaging by temporarily persisting event messages in the event bus. Subscribers can catch up on missed event messages by replaying messages from the event bus using the Replay ID of event messages. The event bus is a distributed system and doesn't have the same guarantees as a transactional database. As a result, Salesforce can't provide a synchronous response for an event publish request. Events are queued and buffered, and Salesforce attempts to publish the events asynchronously. In rare cases, the event message might not be persisted in the distributed system during the initial or subsequent attempts. This means that the events aren't delivered to subscribers, and they aren't recoverable.

Middleware Capabilities

The following table highlights the desirable properties of a middleware system that participates in this pattern.

Property	Mandatory	Desirable	Not Required
Event handling		X	
Protocol conversion		X	
Translation and transformation		X	
Queuing and buffering	X		
Synchronous transport protocols			X

Property	Mandatory	Desirable	Not Required
Asynchronous transport protocols	X		
Mediation routing		X	
Process choreography and service orchestration		X	
Transactionality (encryption, signing, reliable delivery, transaction management)	X		
Routing			X
Extract, transform, and load			X
Long Polling	X (required for platform events)		

Solution Variant—Platform Events: Publishing Behavior and Transactions

When platform event messages are published immediately, event publishing doesn't respect transaction boundaries of the publishing process. Event messages can be published before the transaction completes or even if a transaction fails. This behavior can lead to issues when a subscriber expects to find data that the publishing transaction commits. The data might not be present when the subscriber receives the event message. To solve this issue, set the platform event publish behavior to `Publish After Commit` in the event definition. The publish behaviors you can set in a platform event definition are:

- **Publish After Commit** to have the event message published only after a transaction commits successfully. Select this option if subscribers rely on data that the publishing transaction commits. For example, a process publishes an event message and creates a task record. A second process that is subscribed to the event is fired and expects to find the task record. Another reason for choosing this behavior is when you don't want the event message to be published if the transaction fails.
- **Publish Immediately** to have the event message published when the publish call executes. Select this option if you want the event message to be published regardless of whether the transaction succeeds. Also choose this option if the publisher and subscribers are independent, and subscribers don't rely on data committed by the publisher. For example, the immediate publishing behavior is suitable for an event used for logging purposes. With this option, a subscriber might receive the event message before data is committed by the publisher transaction.

Solution Variant—Outbound Messaging and Message Sequencing

Salesforce outbound messaging can't guarantee the sequence of delivery for its messages because a single message can be retried over a 24-hour period. Multiple methods for handling message sequencing in the remote system are available.

- Salesforce sends a unique message ID for each instance of an outbound message. The remote system discards messages that have a duplicate message ID.
- Salesforce sends only the RecordId. The remote system makes a callback to Salesforce to obtain the necessary data to process the request.

Solution Variant—Outbound Messaging and Deletes

Salesforce workflow rules can't track deletion of a record. The rules can track only the insert or update of a record. Therefore, you can't directly initiate an outbound message from the deletion of a record. You can initiate a message indirectly with the following process.

1. Create a custom object to store key information from the deleted records.
2. Create an Apex trigger, fired by the deletion of the base record, to store information, such as the unique identifier in the custom object.
3. Implement a workflow rule to initiate an outbound message based on the creation of the custom object record.

It's important that state tracking is enabled by storing the remote system's unique identifier in Salesforce or Salesforce's unique identifier in the remote system.

Example

A telecommunications company 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 and won, but the back-end ERP system is the data master. The order must be saved to the Salesforce opportunity record, and the opportunity status changed to indicate that the order was created.

The following constraints apply.

- No custom development in Salesforce.
- You don't require immediate notification of the order number after the opportunity converts to an order.
- The organization has an ESB that supports the CometD protocol and is able to subscribe to platform events.

This example is best implemented using Salesforce platform events, but does require that the ESB subscribes to the platform event..

On the Salesforce side:

- Create a Process Builder process to initiate the platform event (for example, when the opportunity status changes to Close-Won).
- Create a new platform event which publishes the opportunity details.

On the remote system side:

- The ESB subscribes to the Salesforce platform event using the CometD protocol.
- The ESB receives one or more notifications indicating that the opportunity is to be converted to an order.
- The ESB forwards the message to the back-end ERP system so that the order can be created.
- After the order is created in the ERP system, a separate thread calls back to Salesforce using the SessionId as the authentication token. The callback updates the opportunity with the order number and status. You can do this callback using documented pattern solutions, such as Salesforce platform events, Salesforce SOAP API, REST API, or an Apex web service.

This example demonstrates the following.

- Implementation of a remote process invoked asynchronously
- End-to-end guaranteed delivery
- Subsequent callback to Salesforce to update the state of the record

CHAPTER 4 Batch Data Synchronization

Context

You're moving your CRM implementation to Salesforce and want to:

- Extract and transform accounts, contacts, and opportunities from the current CRM system and load the data into Salesforce (initial data import).
- Extract, transform, and load customer billing data into Salesforce from a remote system on a weekly basis (ongoing).
- Extract customer activity information from Salesforce and import it into an on-premises data warehouse on a weekly basis (ongoing).

Problem

How do you import data into Salesforce and export data out of Salesforce, taking into consideration that these imports and exports can interfere with end-user operations during business hours, and involve large amounts of data?

Forces

There are various forces to consider when applying solutions based on this pattern:

- Should the data be stored in Salesforce? If not, there are other integration options an architect can and should consider (mashups, for example).
- If the data should be stored in Salesforce, should the data be refreshed in response to an event in the remote system?
- Should the data be refreshed on a scheduled basis?
- Does the data support primary business processes?
- Are there analytics (reporting) requirements that are impacted by the availability of this data in Salesforce?

Solution

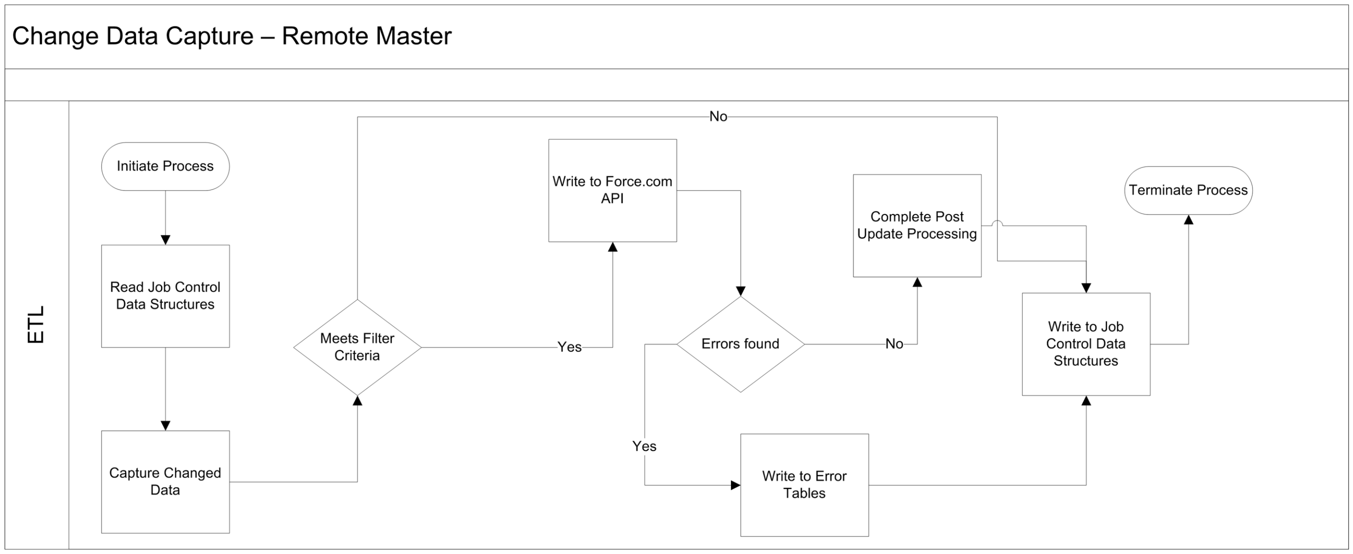
The following table contains various solutions to this integration problem.

Solution	Fit	Data master	Comments
Salesforce Change Data Capture	Best	Salesforce	Salesforce Change Data Capture publishes change events, which represent changes to Salesforce records. Changes include creation of a new record, updates to an existing record, deletion of a record, and undeletion of a record.

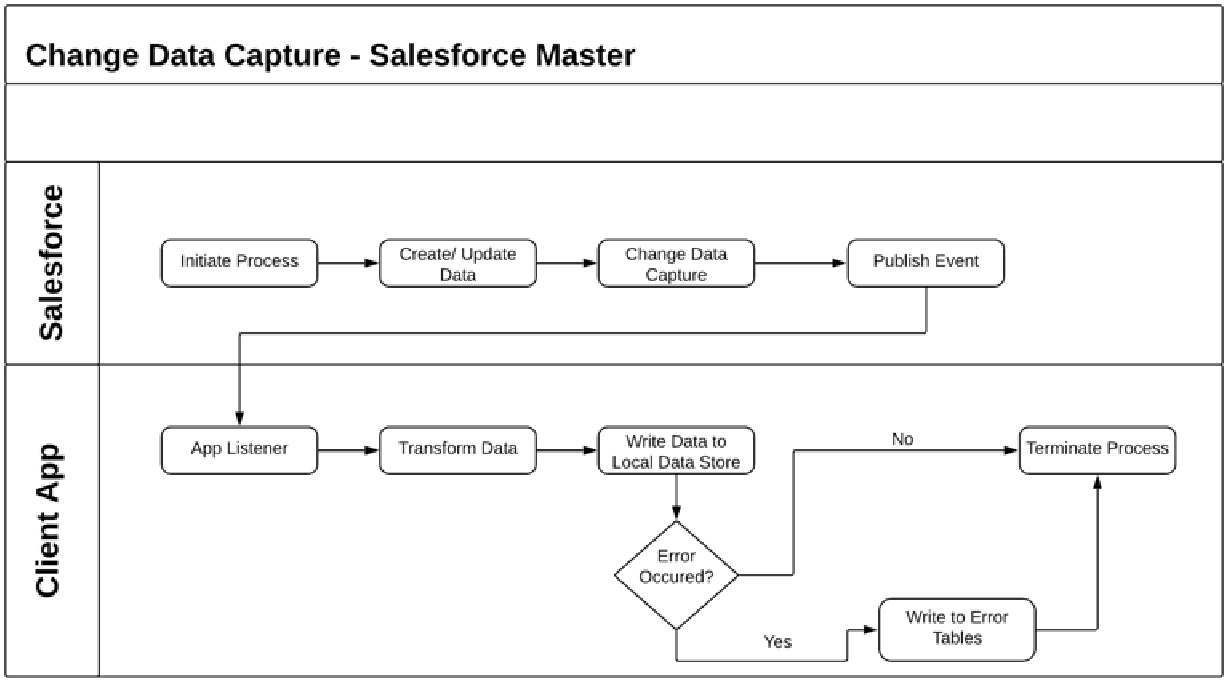
Solution	Fit	Data master	Comments
			With Change Data Capture, you can receive near-real-time changes of Salesforce records, and synchronize corresponding records in an external data store. Change Data Capture takes care of the continuous synchronization part of replication. It publishes the deltas of Salesforce data for new and changed records. Change Data Capture requires an integration app for receiving events and performing updates in the external system.
Replication via third-party ETL tool	Best	Remote system	Leverage a third-party ETL tool that allows you to run change data capture against source data. The tool reacts to changes in the source data set, transforms the data, and then calls Salesforce Bulk API to issue DML statements. This can also be implemented using the Salesforce SOAP API.
Replication via third-party ETL tool	Good	Salesforce	Leverage a third-party ETL tool that allows you to run change data capture against ERP and Salesforce data sets. In this solution, Salesforce is the data source, and you can use time/status information on individual rows to query the data and filter the target result set. This can be implemented by using SOQL together with SOAP API and the `query()` method, or by using SOAP API and the `getUpdated()` method.
Remote call-in	Suboptimal	Remote system	It's possible for a remote system to call into Salesforce by using one of the APIs and perform updates to data as they occur. However, this causes considerable on-going traffic between the two systems. Greater emphasis should be placed on error handling and locking. This pattern has the potential for causing continual updates, which has the potential to impact performance for end users.
Remote process invocation	Suboptimal	Salesforce	It's possible for Salesforce to call into a remote system and perform updates to data as they occur. However, this causes considerable on-going traffic between the two systems. Greater emphasis should be placed on error handling and locking. This pattern has the potential for causing continual updates, which has the potential to impact performance for end users.

Sketch

The following diagram illustrates the sequence of events in this pattern, where the remote system is the data master.



The following diagram illustrates the sequence of events in this pattern, where Salesforce is the data master.



Results

You can integrate data that's sourced externally with Salesforce under the following scenarios:

- External system is the data master—Salesforce is a consumer of data provided by a single source system or multiple systems. In this scenario, it's common to have a data warehouse or data mart that aggregates the data before the data is imported into Salesforce.
- Salesforce is the data master—Salesforce is the system of record for certain entities and Salesforce Change Data Capture client applications can be informed of changes to Salesforce data.

In a typical Salesforce integration scenario, the implementation team does one of the following:

- Implement change data capture on the source data set.
- Implement a set of supporting database structures, known as control tables, in an intermediate, on-premises database.

The ETL tool is then used to create programs that will:

1. Read a control table to determine the last run time of the job and extract any other control values needed.
2. Use the above control values as filters and query the source data set.
3. Apply predefined processing rules, including validation, enrichment, and so on.
4. Use available connectors/transformation capabilities of the ETL tool to create the destination data set.
5. Write the data set to Salesforce objects.
6. If processing is successful, update the control values in the control table.
7. If processing fails, update the control tables with values that enable a restart and exit.

Note: We recommend that you create the control tables and associated data structures in an environment that the ETL tool has access to even if access to Salesforce isn't available. This provides adequate levels of resilience. Salesforce should be treated as a spoke in this process and the ETL infrastructure is the hub.

For an ETL tool to gain maximum benefit from data synchronization capabilities, consider the following:

- Chain and sequence the ETL jobs to provide a cohesive process.
- Use primary keys from both systems to match incoming data.
- Use specific API methods to extract only updated data.
- If importing child records in a master-detail or lookup relationship, group the imported data using its parent key at the source to avoid locking. For example, if you're importing contact data, be sure to group the contact data by the parent account key so the maximum number of contacts for a single account can be loaded in one API call. Failure to group the imported data usually results in the first contact record being loaded and subsequent contact records for that account to fail in the context of the API call.
- Any post-import processing, such as triggers, should only process data selectively.
- If your scenario involves large data volumes, follow the best practices in the white paper *Best Practices for Deployments with Large Data Volumes*.

Error Handling and Recovery

An error handling and recovery strategy must be considered as part of the overall solution. The best method depends on the solution you choose.

Error location	Error handling and recovery strategy
Read from Salesforce using Change Data Capture	• *Error handling*—Error handling must be performed in the remote service because the event is effectively handed off to the remote system for further processing. Because this pattern is asynchronous, the remote system handles message queuing, processing, and error handling. Additionally, Change Data Capture events aren't processed within database transactions. As a result, published events can't be rolled back within a transaction. • *Recovery*—Because this pattern is asynchronous, the remote system must initiate retries based on the service's quality of service requirements. The replay ID associated with each Change Data Capture event is atomic and increases with every event published. This ID can be used replay the stream from a specific event (for example, based upon the last successfully captured event). High-volume platform event messages are stored for 72 hours (3 days). You can retrieve past event messages when using CometD clients to subscribe to a channel.

Error location	Error handling and recovery strategy
Read from Salesforce using a 3rd party ETL system	• *Error handling*—If an error occurs during a read operation, implement a retry for errors that aren't infrastructure-related. In the event of repeated failure, standard processing using control tables/error tables should be implemented in the context of an ETL operation to: – Log the error – Retry the read operation – Terminate if unsuccessful – Send a notification • *Recovery*—Restart the ETL process to recover from a failed read operation. If the operation succeeds but there are failed records, an immediate restart or subsequent execution of the job should address the issue. In this case, a delayed restart might be a better solution because it allows time to triage and correct data that might be causing the errors.
Write to Salesforce	• *Error handling*—Errors that occur during a write operation can result from a combination of factors in the application. The API calls return a result set that consists of the information listed below. This information should be used to retry the write operation (if necessary). – Record identifying information – Success/failure notification – A collection of errors for each record • *Recovery*—Restart the ETL process to recover from a failed read operation. If the operation succeeds but there are failed records, an immediate restart or subsequent execution of the job should address the issue. In this case, a delayed restart might be a better solution because it allows time to triage and correct data that might be causing the errors.
External master system	Errors should be handled in accordance with the best practices of the master system.

Security Considerations

Any call to a remote system must maintain the confidentiality, integrity, and availability of the request. Different security considerations apply, depending on the solution you choose.

- A Lightning Platform license with at least "API Only" user permissions is required to allow authenticated API access to the Salesforce API.
- We recommend that standard encryption be used to keep password access secure.
- Use the HTTPS protocol when making calls to the Salesforce APIs. You can also proxy traffic to the Salesforce APIs through an on-premises security solution, if necessary.

See Security Considerations.

Sidebars

Timeliness

Timeliness isn't of significant importance in this pattern. However, care must be taken to design the interfaces so that all of the batch processes complete in a designated batch window.

As with all batch-oriented operations, we strongly recommend that you take care to insulate the source and target systems during batch processing windows. Loading batches during business hours might result in some contention, resulting in either a user's update failing, or more significantly, a batch load (or partial batch load) failing.

For organizations that have global operations, it might not be feasible to run all batch processes at the same time because the system might continually be in use. Data segmentation techniques using record types and other filtering criteria can be used to avoid data contention in these cases.

State Management

You can implement state management by using surrogate keys between the two systems. If you need any type of transaction management across Salesforce entities, we recommend that you use the Remote Call-In pattern using Apex.

Standard optimistic record locking occurs on the platform, and any updates made using the API require the user, who is editing the record, to refresh the record and initiate their transaction. In the context of the Salesforce API, optimistic locking refers to a process where:

- Salesforce doesn't maintain the state of a record being edited by a specific user.
- Upon read, it records the time when the data was extracted.
- If the user updates the record and saves it, Salesforce checks to see if another user has updated the record in the interim.
- If the record has been updated, the system notifies the user that an update was made and the user should retrieve the latest version of the record before proceeding with their updates.

Middleware Capabilities

The most effective external technologies used to implement this pattern are traditional ETL tools. It's important that the middleware tools chosen support the Salesforce Bulk API.

It's helpful, but not critical, that the middleware tools support the `getUpdated()` function. This function provides the closest implementation to standard change data capture capability on the Salesforce platform.

The following table highlights the desirable properties of a middleware system that participates in this pattern.

Property	Mandatory	Desirable	Not required
Event handling		X	
Protocol conversion		X	
Translation and transformation	X	X	
Queuing and buffering	X		
Synchronous transport protocols			X
Asynchronous transport protocols	X		
Mediation routing		X	
Process choreography and service orchestration	X		
Transactionality (encryption, signing, reliable delivery, transaction management)		X	
Routing		X	
Extract, transform, and load	X		

Property	Mandatory	Desirable	Not required
Long Polling	X (required for Salesforce Change Data Capture)		

Example

A utility company uses a mainframe-based batch process that assigns prospects to individual sales reps and teams. This information needs to be imported into Salesforce on a nightly basis.

The customer has decided to implement change data capture on the source tables using a commercially available ETL tool.

The solution works as follows:

- A cron-like scheduler executes a batch job that assigns prospects to users and teams.
- After the batch job runs and updates the data, the ETL tool recognizes these changes using change data capture. The ETL tool collates the changes from the data store.
- The ETL connector uses the Salesforce SOAP API to load the changes into Salesforce.

CHAPTER 5 Remote Call-In

Context

You use Salesforce to track leads, manage your pipeline, create opportunities, and capture order details that convert leads to customers. But, Salesforce isn't the system that contains or processes orders. Orders are managed by an external (remote) system. That remote system needs to update the order status in Salesforce as the order passes through its processing stages.

Problem

How does a remote system connect and authenticate with Salesforce to notify Salesforce about external events, create records, and update existing records?

Forces

There are various forces to consider when applying solutions based on this pattern:

- Is the purpose of the remote call to Salesforce to notify Salesforce of an event that occurred externally using an event-driven architecture? Or is the purpose to perform CRUD operations on specific records? If you use an event-driven architecture, the event producer (the remote process) is decoupled from the Salesforce event consumer.
- Does the call to Salesforce require the remote process to wait for a response before continuing processing? Remote calls to Salesforce are always synchronous request-reply, although the remote process can discard the response if it isn't needed to simulate an asynchronous call.
- Does each transaction operate against a single Salesforce object or multiple, related objects?
- What is the format of the message (for example, SOAP or REST, or both over HTTP)?
- Is the message size relatively small or large?
- If the remote system is SOAP-capable, is the remote system able to participate in a contract-first approach, where Salesforce dictates the contract? This is required where our SOAP API is used, for which a predefined WSDL is supplied.
- Is transaction processing required?
- What is the extent to which you're tolerant of customization in Salesforce?

Solution

This table contains various solutions to this integration problem.

Solution	Fit	Comments
SOAP API	Best	• *Accessibility*—Salesforce provides a SOAP API that remote systems can use to: – Publish events to notify your Salesforce org – Query data in your org – Create, update, and delete data – Obtain metadata about your org – Run utilities to perform administrative tasks • *Synchronous API*—After the API call is made, the remote client application waits until it receives a response from the service. Asynchronous calls to Salesforce aren't supported. • *Generated WSDL*—Salesforce provides two WSDLs for remote systems: – Enterprise WSDL—Provides a strongly-typed WSDL that's specific to a Salesforce org. – Partner WSDL—Contains a loosely typed WSDL that's not specific to a Salesforce org. • *Security*—The client executing SOAP API must have a valid login and obtain a session to perform any API calls. The API respects object-level and field-level security configured in Salesforce based on the logged in user's profile. • *Transaction/Commit Behavior*—By default, each API call allows for partial success if some records are marked with errors. This can be changed to an "all or nothing" behavior where all the results are rolled back if any error occurs. It's not possible to span a transaction across multiple API calls. To overcome this limitation, it's possible for a single API call to affect multiple objects. • *Bulk Data*—Any data operation that includes more than 2,000 records is a good candidate for Bulk API 2.0 to successfully prepare, execute, and manage an *asynchronous* workflow that uses the Bulk framework. Jobs with fewer than 2,000 records should involve "bulkified" *synchronous* calls in REST (for example, Composite) or SOAP. • *Event-Driven Architecture*—Platform events are defined the same way you define Salesforce objects. Publishing an event via SOAP API is the same as creating a Salesforce record.
REST API	Best	• *Accessibility*—Salesforce provides a REST API that remote systems can use to: – Publish events to notify your Salesforce org

Solution	Fit	Comments
		– Query data in your org – Create, update, and delete data – Obtain metadata about your org – Run utilities to perform administrative tasks • *Synchronous API*—After the API call is made, the remote client application waits until it receives a response from the service. Asynchronous calls to Salesforce aren't supported. • *REST API vs. SOAP API*—REST exposes resources (entities/objects) as URIs and uses HTTP verbs to define CRUD operations on these resources. Unlike SOAP, REST API requires no predefined contract, utilizes XML and JSON for responses, and has loose typing. REST API is lightweight and provides a simple method for interacting with Salesforce. Its advantages include ease of integration and development, and it's an excellent choice for use with mobile apps and web apps. • *Security*—The client executing REST API must have a valid login and obtain a session to perform any API calls. The API respects object-level and field-level security configured in Salesforce based on the logged in user's profile. • *Transaction/Commit Behavior*—By default, each record is treated as a separate transaction and committed separately. Failure of one record change doesn't cause rollback of other record changes. This behavior can be altered to an "all or nothing" behavior. Use the REST API composite resources to make a series of updates in one API call. • *REST Composite Resources*—Use these REST API resources to perform multiple operations in a single API call. It's also possible to use the output of one call as the input to the next call. All of the response bodies and HTTP statuses for the requests are returned in a single response body. The entire request counts as a single call toward your API limits. • *Bulk Data*—Any data operation that includes more than 2,000 records is a good candidate for Bulk API 2.0 to successfully prepare, execute, and manage an *asynchronous* workflow that uses the Bulk framework. Jobs with fewer than 2,000 records should involve "bulkified" *synchronous* calls in REST (for example, Composite) or SOAP. • *Event-Driven Architecture*—Platform events are defined the same way you define Salesforce objects. Publishing an event via REST API is the same as creating a Salesforce record.

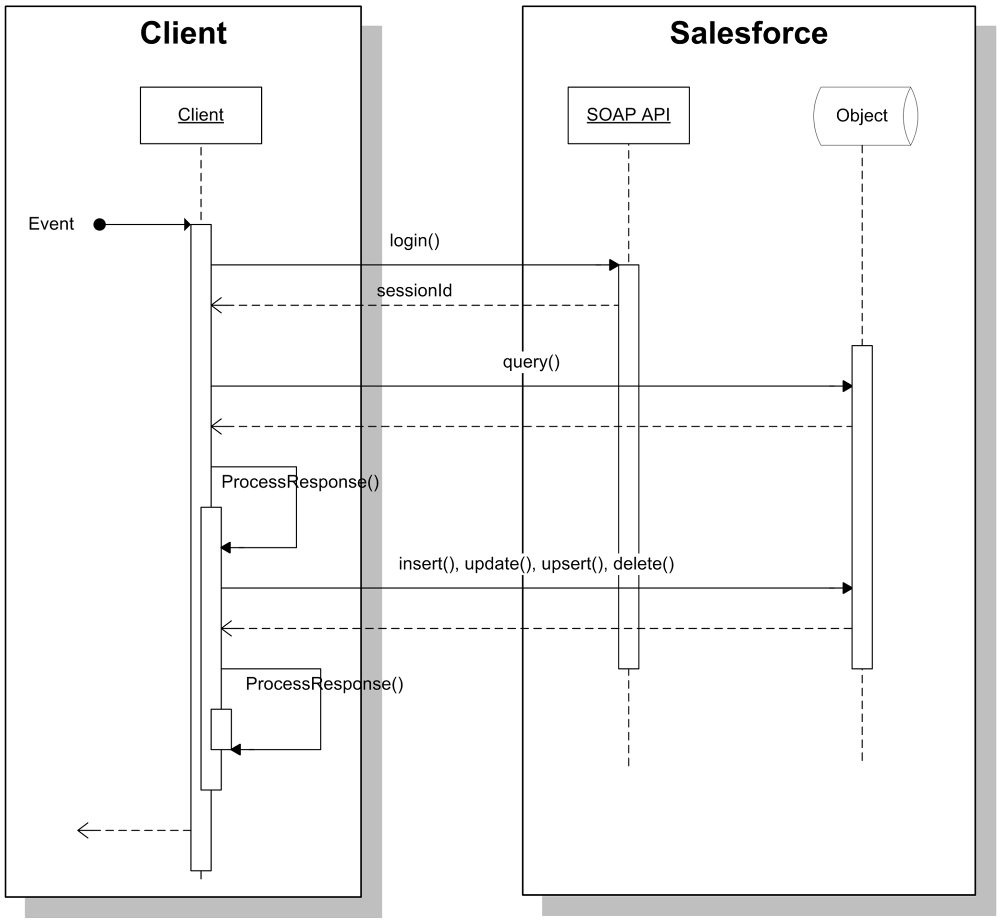
Solution	Fit	Comments
Apex web services	Suboptimal	Apex class methods can be exposed as web service methods to external applications. This method is an alternative to SOAP API, and is typically used only where the following additional requirements must be met. • Full transactional support is required (for example, create an account, contact, and opportunity all in one transaction). • Custom logic must be applied on the Salesforce side before committing. The benefit of using an Apex web service must be weighed against the additional code that needs to be maintained in Salesforce. Not applicable for platform events because transaction pre-insert logic at the consumer doesn't apply in an event-driven architecture. To notify a Salesforce org that an event has occurred, use SOAP API, REST API, or Bulk API 2.0.
Apex REST services	Suboptimal	An Apex class can be exposed as REST resources mapped to specific URIs with an HTTP verb defined against it (for example, POST or GET). You can use REST API composite resources to perform multiple updates in a single transaction. Unlike SOAP, there's no need for the client to consume a service definition/contract (WSDL) and generate client stubs. The remote system requires only the ability to form an HTTP request and process the returned results (XML or JSON). Not applicable for platform events because transaction pre-insert logic at the consumer doesn't apply in an event-driven architecture. To notify a Salesforce org that an event has occurred, use SOAP API, REST API, or Bulk API 2.0.
Bulk API 2.0	Optimal for bulk operations	Bulk API 2.0 is based on REST principles, and is optimized for loading or deleting large sets of data. It has the same accessibility and security behavior as REST API. Any data operation that includes more than 2,000 records is a good candidate for Bulk API 2.0 to successfully prepare, execute, and manage an *asynchronous* workflow that uses the Bulk framework. Jobs with fewer than 2,000 records should involve "bulkified" *synchronous* calls in REST (for example, Composite) or SOAP. Bulk API 2.0 allows the client application to query, insert, update, upsert, or delete a large number of records asynchronously by submitting a number of batches, which are processed in the background by Salesforce. In contrast, SOAP

Solution	Fit	Comments
		API is optimized for real-time client applications that update small numbers of records at a time. Although SOAP API can also be used for processing large numbers of records, when the data sets contain hundreds of thousands to millions of records, it becomes less practical. This is due to its relatively high overhead and lower performance characteristics. • *Event-Driven Architecture*—Platform events are defined the same way you define Salesforce objects. Publishing an event via Bulk API 2.0 is the same as creating a Salesforce record. Only the create and insert operations are supported. Events within a batch are published to the Salesforce event bus asynchronously as the batch job is processed.

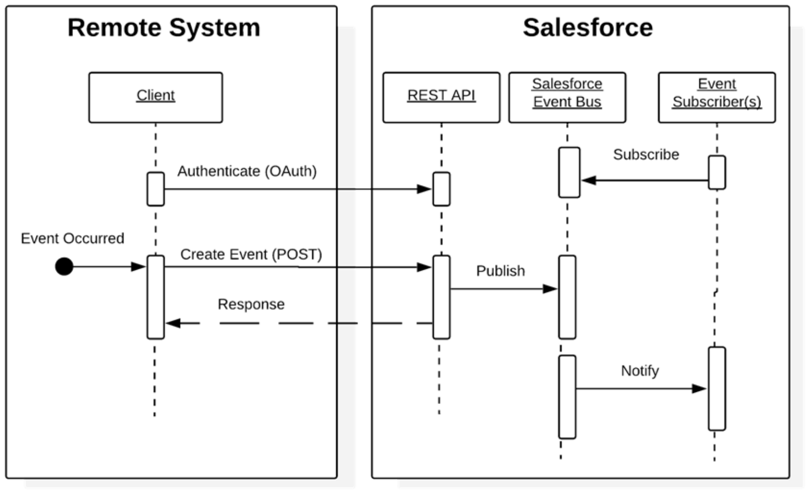
Sketch

The following diagrams illustrate the sequence of events when you implement this pattern using either REST API for notifications from external events or SOAP API to query a Salesforce object. The sequence of events is the same when using REST API.

Remote System Querying Salesforce Via SOAP API



Remote System Notifying Salesforce with Events Via REST API



Results

In an event-driven architecture, the remote system calls into Salesforce using SOAP API, REST API, or Bulk API 2.0 to publish an event to the Salesforce event bus. Publishing an event notifies all subscribers. Event subscribers can be on the Salesforce Platform such as Process Builder Processes, Flows, or Lightning Components, Apex triggers. Event subscribers can also be external to the Salesforce Platform such as CometD subscribers.

When working directly with Salesforce objects, the solutions related to this pattern allow for:

- The remote system to call the Salesforce APIs to query the database and execute single-object operations (create, update, delete, and so on).
- The remote system to call the Salesforce REST API composite resources to perform a series of object operations.
- Remote system to call custom-built Salesforce APIs (services) that can support multi-object transactional operations and custom pre/post processing logic.

Calling Mechanisms

The calling mechanism depends on the solution chosen to implement this pattern.

Calling mechanism	Description
SOAP API	The remote system uses the Salesforce Enterprise or Partner WSDL to generate client stubs that are in turn used to invoke the standard SOAP API.
REST API	The remote system has to authenticate before accessing any Apex REST service. The remote system can use OAuth 2.0 or username and password authentication. In either case, the client must set the authorization HTTP header with the appropriate value (an OAuth access token or a session ID can be acquired via a login call to SOAP API). The remote system then generates REST invocations (HTTP requests) with the appropriate verbs and processes the results returned (JSON and XML data formats are supported).
Apex web service	The remote system consumes the custom Apex web service WSDL to generate client stubs that are in turn used to invoke the custom Apex web service.
Apex REST service	As per REST API, the resource URI and applicable verbs are defined using the `@RestResource`, `@HttpGet`, and `@HttpPost` annotations.
Bulk API 2.0	Bulk API 2.0 is a REST-based API, so the same calling mechanisms as REST API apply.
REST API to invoke Flow	Use REST API to call a custom invocable actions endpoint to invoke an auto-launched flow.

Error Handling and Recovery

An error handling and recovery strategy must be considered as part of the overall solution.

- *Error handling*—All the remote call-in methods, standard or custom APIs, require the remote system to handle any subsequent errors, such as timeouts and the management of retries. Middleware can be used to provide the logic for error handling and recovery.
- *Recovery*—A custom retry mechanism needs to be created if quality-of-service requirements dictate it. In this case, it's important to ensure idempotent design characteristics. Platform event enables subscribers to use the replay ID to fetch messages within a certain time period after those messages were published.

Idempotent Design Considerations

Idempotent capabilities guarantee that repeated invocations are safe and won't have a negative effect. If idempotency isn't implemented, then repeated invocations of the same message can have different results, potentially resulting in data integrity issues, for example, creation of duplicate records, duplicate processing of transactions, and so on.

The remote system must manage multiple (duplicate) calls, in the case of errors or timeouts, to avoid duplicate inserts and redundant updates (especially if downstream triggers and workflow rules fire). While it's possible to manage some of these situations within Salesforce (particularly in the case of custom SOAP and REST services), we recommend that the remote system (or middleware) manages error handling and idempotent design.

Security Considerations

Different security considerations apply, depending on the pattern solution chosen. In all cases, the platform uses the logged-in user's access rights (for example, profile settings, sharing rules, permission sets, and so on). Additionally, profile IP restrictions can be used to restrict access to the API for a specific IP address range.

Solution	Security considerations
SOAP API	Salesforce supports Secure Sockets Layer v3 (SSL) and the Transport Layer Security (TLS) protocols. Ciphers must have a key length of at least 128 bits. The remote system must log in using valid credentials to obtain a session ID. If the remote system already has a valid session ID, then it can call the API without an explicit login. This is used in call-back patterns covered earlier in this document, where a preceding Salesforce outbound message contained a session ID and record ID for subsequent use. We recommend that clients that call SOAP API cache and reuse the session ID to maximize performance, rather than obtaining a new session ID for each call.
REST API	We recommend that the remote system establish an OAuth trust for authorization. REST calls can then be made on specific resources using HTTP verbs. It's also possible to make REST calls with a valid session ID that might have been obtained by other means (for example, retrieved by calling SOAP API or provided via an outbound message). We recommend that clients that call the REST API cache and reuse the session ID to maximize performance, rather than obtaining a new session ID for each call.
Apex web service	We recommend that the remote system establish an OAuth trust for authorization.
Apex REST service	We recommend that the remote system establish an OAuth trust for authorization.
Bulk API 2.0	We recommend that the remote system establish an OAuth trust for authorization.

See Security Considerations.

Sidebars

Timeliness

SOAP API and Apex web service API are synchronous. The following timeouts apply:

- Session timeout — The session times out if there's no activity based on the Salesforce org's session timeout setting.
- Query timeout — Each SOQL query has an individual timeout limit of 120 seconds.

Data Volumes

Data volume considerations depend on which solution and communication type you choose.

Solution	Communication type	Limits
SOAP API or REST API	Synchronous	• *SOAP Login*—The SOAP login request size is limited to 10 KB or less. You can make a maximum of 3,600 calls to the `login()` function per user per hour. If you exceed this limit, the API returns an error. • *Create, Update, Delete*—The remote system can create, update, or delete up to 200 records at a time. Multiple calls can be made to process more than a total of 200 records, but each request is limited to 200 records in size. • *BLOB Data*—You can use SObject Basic Information, SObject Rows, or SObject Collections REST resources to insert or update BLOB data in Salesforce standard objects. For the SObject Basic Information or SObject Rows resources, the maximum file size for uploads is 2 GB for ContentVersion objects and 500 MB for all other eligible standard objects. Using the SObject Collections resources, the maximum total size of all files in a single request is 500 MB. • *Query Results Size* — By default, the number of rows returned in the query result object (batch size), returned in a `query()` or `queryMore()` call is set to 500. The maximum number of rows returned is 2,000. You can explicitly set the batch size, but there's no guarantee that the requested batch size will be the actual batch size. This is done to maximize performance. Where the number of rows to be returned exceeds the batch size, use the `queryMore()` API call to iterate through multiple batches. Additional rules might apply, so for more information, see *Salesforce Developer Limits and Allocations Quick Reference*. • *Event Message*—The maximum event message size is 1 MB. Publishing an event using the Salesforce APIs counts against your standard API limits.
Bulk API 2.0	Synchronous	Bulk API 2.0 is optimized for importing and exporting large sets of data asynchronously. Any data operation that includes more than 2,000 records is a good candidate for Bulk API 2.0 to successfully prepare, execute, and manage an *asynchronous* workflow that uses the Bulk framework. Jobs with fewer than 2,000 records should involve “bulkified” *synchronous* calls in REST (for example, Composite) or SOAP. Bulk API 2.0 is synchronous when submitting the batch request and associated data. The actual processing of the data occurs asynchronously. For more information about API and batch processing limits, see *Limits*.

Endpoint Capability and Standards Support

The capability and standards support for the endpoint depends on the solution that you choose.

Solution	Endpoint considerations
SOAP API	The remote system must be capable of implementing a client that can call the Salesforce SOAP API, based on a message format predefined by Salesforce. The remote system (client) must participate in a contract-first implementation where the contract is supplied by Salesforce (for example, Enterprise or Partner WSDL).
REST API	The remote system must be capable of implementing a REST client that invokes Salesforce—defined REST services, and processes the XML or JSON results.
Apex web service	The remote system must be capable of implementing a client that can invoke SOAP messages of a predefined format, as defined by Salesforce. The remote system must participate in a code-first implementation, where the contract is supplied by Salesforce after the Apex web service is implemented. Each Apex web service has its own WSDL.
Apex REST service	The same endpoint considerations as REST API apply.

State Management

When integrating systems, keys are important for on-going state tracking, for example, if a record gets created in the remote system, to support ongoing updates to that record. There are two options:

- Salesforce stores the remote system's primary or unique surrogate key for the remote record.
- The remote system stores the Salesforce unique record ID or some other unique surrogate key.

There are specific considerations for handling integration keys in this synchronous pattern.

Master system	Description
Salesforce	In this scenario, the remote system stores either the Salesforce RecordId or some other unique surrogate key from the record.
Remote system	In this scenario, Salesforce stores a reference to the unique identifier in the remote system. Because the process is synchronous, the key can be provided as part of the same transaction using external ID fields.

Complex Integration Scenarios

Each solution in this pattern has different considerations when handling complex integration scenarios such as transformation and process orchestration.

Solution	Considerations
SOAP API or REST API	SOAP API and REST API provide for simple transactions on objects. Complex integration scenarios, such as aggregation, orchestration, and transformation, can't be performed in Salesforce. These scenarios must be handled by the remote system or middleware, with middleware as the preferred method.
Apex web service or Apex REST service	Custom web services can provide for cross-object functionality, custom logic, and more complex transaction support. This solution should be used with care, and you should always consider the suitability of middleware for any transformation, orchestration, and error handling logic.

Governor Limits

Due to the multitenant nature of the Salesforce platform, there are limits when using the APIs.

Solution	Limits
SOAP API, REST API, and custom Apex APIs	• *API request limits*—Salesforce applies a limit on the number of API calls per 24–hour period. The limit is based on the Salesforce edition type and number of licenses. For example, Unlimited Edition provides 5,000 API requests per Salesforce or Lightning Platform license per 24 hours. For more information, see *Salesforce Developer Limits and Allocations Quick Reference*. • *API query cursor limits*—A user can have up to 10 query cursors open at a time. Otherwise, the oldest of the 10 cursors is released. If the remote application attempts to open the released query cursor, an error results. For example, if sharing integration user credentials, the maximum query cursors need to be considered. Whenever possible, the middleware should complete the full query before executing another query (in a serial fashion) or each application should use a designated integration user. Alternatively, the middleware may need to execute requests across multiple users in a "round robin" fashion. • *Call limits*—See Data Volumes sidebar for create, update, and query limits.
Bulk API 2.0	See Data Volumes sidebar for more information.
Platform Events	• *Event notification limits*—A maximum of 100,000 events can be published per hour for standard volume platform events. A maximum of 250,000 events can be published per hour for high-volume usage-based platform events. To monitor high-volume event usage, use REST API limits resource. • *Event message size limits*—The maximum event message size is 1 MB. Publishing an event using the Salesforce APIs counts against your standard API limits.

Reliable Messaging

Reliable messaging attempts to resolve the issue of guaranteeing the delivery of a message to a remote system where the individual components themselves might be unreliable. The Salesforce SOAP API and REST API are synchronous and don't provide explicit support for any reliable messaging protocols, per se (for example, WS-ReliableMessaging).

We recommend that the remote system implement a reliable messaging system to ensure that error and timeout scenarios are successfully managed. Publishing of platform events from external systems relies on Salesforce APIs, so the same considerations for SOAP API and REST API apply.

Middleware Capabilities

This table highlights the desirable properties of a middleware system that participates in this pattern:

Property	Mandatory	Desirable	Not required
Event handling		X	
Protocol conversion		X	
Translation and transformation		X	
Queuing and buffering	X		
Synchronous transport protocols	X		

Property	Mandatory	Desirable	Not required
Asynchronous transport protocols			X
Mediation routing		X	
Process choreography and service orchestration		X	
Transactionality (encryption, signing, reliable delivery, and transaction management)	X		
Routing			X
Extract, transform, and load		X (for bulk/batches)	

Example

A printing supplies and services company uses Salesforce as a front end to create and manage printer supplies and orders. Salesforce asset records representing printers are periodically updated with printing usage statistics (ink status, paper level) from the on-premises Printer Management System (PMS), which regularly monitors printers on client sites. Upon reaching a set threshold value (for example, low ink status or low/empty paper level of less than 30%), multiple apps/processes (variable) interested in the event are notified, email or Chatter alerts are sent, and an Order record is created. The PMS stores the Salesforce ID (Salesforce is the asset record master).

The following constraints apply:

- The PMS can participate in a contract-first integration, where Salesforce provides the contract and the PMS acts as a client (consumer) of the Salesforce service (defined via the Enterprise or Partner WSDL).
- There should be no custom development in Salesforce.

This example is best implemented using the Salesforce SOAP API or REST API to publish events, and declarative automation (Process Builder) in Salesforce. The primary reason to use platform events is the variable and non-finite number of subscribers; however, for simple updates to a finite list of records such as orders, then use SOAP or REST API to update the records.

In Salesforce:

- Define a platform event in Salesforce to contain the notification data coming from the PMS.
- Create a Process Builder process that's triggered by the printer event notification. The process updates the printer asset and creates an order (using an auto-launched Flow).
- Download the Enterprise or Partner WSDL and provide it to the remote system.

In the remote system:

- Create a client stub from the Enterprise or Partner WSDL.
- Authenticate to Salesforce (via OAuth web server or bearer token flow) using the integration user's credentials.
- Upon printer status event, the PMS calls the API to create the printer status platform event (with printer usage statistics). The Salesforce event bus notifies Process Builder subscriber and all other subscribers.

When you use platform events, the event bus lets you replay events for 72 hours for high-volume platform events. Publishing those events using a middleware solution can help incorporate error handling on the publishing side. However, you can implement error handling on the subscribing side if you need higher reliability.

This example demonstrates the following:

- Implementation of a Salesforce synchronous API client (consumer).

- A callback to Salesforce to publish a platform event (aligned with previously covered request/reply platform event patterns).

CHAPTER 6 UI Update Based on Data Changes

Context

You use Salesforce to manage customer cases. A customer service rep is on the phone with a customer working on a case. The customer makes a payment, and the customer service rep needs to see a real-time update in Salesforce from the payment processing application, indicating that the customer has successfully paid the order's outstanding amount.

Problem

When an event occurs in Salesforce, how can the user be notified in the Salesforce user interface without having to refresh their screen and potentially losing work?

Forces

There are various forces to consider when applying solutions based on this pattern:

- Does the data being acted on need to be stored in Salesforce?
- Can a custom user interface layer be built for viewing this data?
- Will the user have access for invoking the custom user interface?

Solution

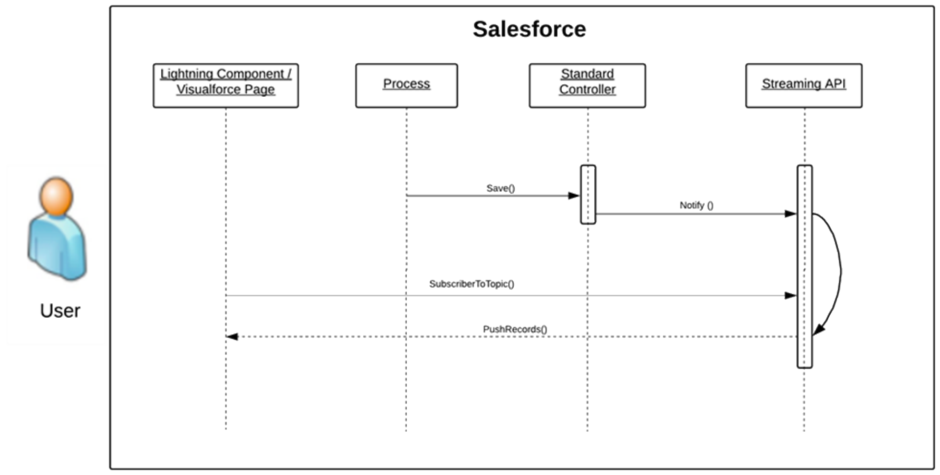
The recommended solution to this integration problem is to use the Salesforce Streaming API. This solution is composed of the following components:

- A PushTopic with a query definition that allows you to:
  - Specify what events trigger an update
  - Select what data to include in the notification
- A JavaScript-based implementation of the *Bayeux* protocol (currently *CometD*) that can be used by the user interface
- A Visualforce page or Lightning component
- A JavaScript library included as a static resource

Sketch

The following diagram illustrates how Streaming API can be implemented to stream notifications to the Salesforce user interface. These notifications are triggered by record changes in Salesforce.

UI Update in Salesforce Triggered by a Data Change



Results

Benefits

The application of the solution related to this pattern has the following benefits:

- Eliminates the need for writing custom polling mechanisms
- Eliminates the need for a user-initiated feedback loop

Unsupported Requirements

The solution has the following limitations:

- Delivery of notifications isn't guaranteed.
- Order of notifications isn't guaranteed.
- Notifications aren't generated from record changes made by Bulk API.

Security Considerations

Standard Salesforce org-level security is adhered to. It's recommended that you use the HTTPS protocol to connect to Streaming API. See Security Considerations.

Sidebars

The optimal solution involves creating a custom user interface in Salesforce. It's imperative that you account for an appropriate user interface container that can be used for rendering the custom user interface. Supported browsers are listed in the *Streaming API Developer Guide*.

Example

A telecommunications company uses Salesforce to manage customer cases. The customer service managers want to be notified when a case is successfully closed by one of their customer service reps.

Implementing the solution prescribed by this pattern, the customer should:

- Create a PushTopic that sends a notification when a case is saved with a Status of “Closed” and Resolution of “Successful.”
- Create a custom user interface available to customer service managers. This user interface subscribes to the PushTopic channel.
- Implement logic in the custom user interface that shows alerts generated by that manager’s customer service reps.

CHAPTER 7 Data Virtualization

Context

You use Salesforce to track leads, manage your pipeline, create opportunities, and capture order details that convert leads to customers. However, Salesforce isn't the system that contains or processes orders. Orders are managed by an external (remote) system. But sales reps want to view and update real-time order information in Salesforce without having to learn or use the external system.

Problem

In Salesforce, how do you view, search, and modify data that's stored outside of Salesforce, without moving the data from the external system into Salesforce?

Forces

There are various forces to consider when applying solutions based on this pattern:

- Do you want to build a declarative/point-and-click outbound integration or UI mashup in Salesforce?
- Do you have a large amount of data that you don't want to copy into your Salesforce org?
- Do you need to access small amounts of remote system data at any one time?
- Do you need real-time access to the latest data?
- Do you store your data in the cloud or in a back-office system, but want to display or process that data in your Salesforce org?
- Do you have data residency concerns for storing certain types of data in Salesforce?

Solution

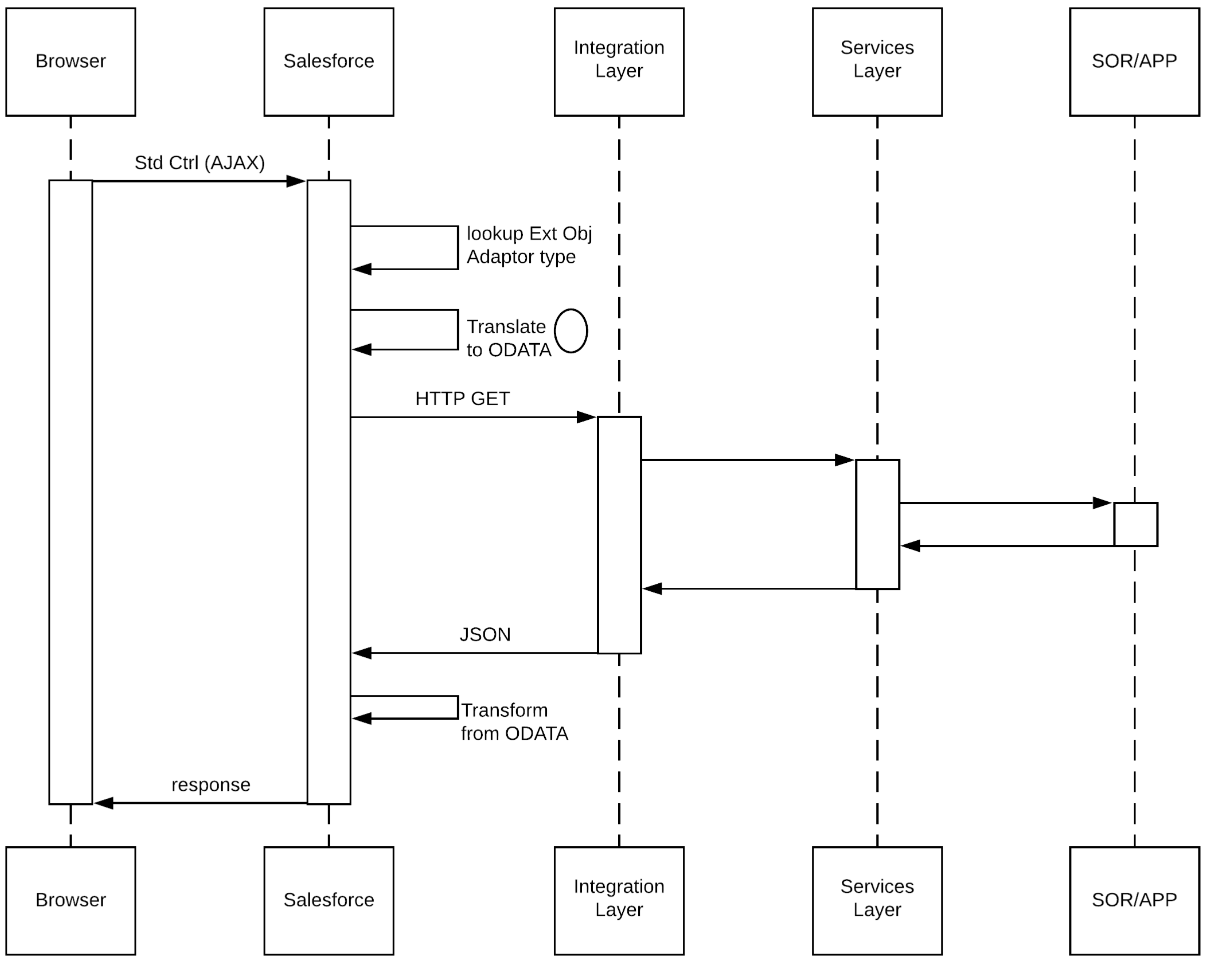
The following table contains various solutions to this integration problem.

Solution	Fit	Comments
Salesforce Connect	Best	Use Salesforce Connect to access data from external sources, along with your Salesforce data. Pull data from legacy systems such as SAP, Microsoft, and Oracle in real time without making a copy of the data in Salesforce. Salesforce Connect maps data tables in external systems to external objects in your org. External objects are similar to custom objects, except that they map to data located outside your Salesforce org. Salesforce Connect uses a live connection to external data to always keep external objects up-to-date.

Solution	Fit	Comments
		Accessing an external object fetches the data from the external system in real time. Salesforce Connect lets you: • Query data in an external system. • Create, update, and delete data in an external system. • Access external objects via list views, detail pages, record feeds, custom tabs, and page layouts. • Define relationships between external objects and standard or custom objects to integrate data from different sources. • Enable Chatter feeds on external object pages for collaboration. • Run reports (limited) on external data. • View the data on the Salesforce mobile app. To access data stored on an external system using Salesforce Connect, you can use one of the following adapters: • OData 2.0 adapter or OData 4.0 adapter — connects to data exposed by any OData 2.0 or 4.0 producer. • Cross-org adapter — connects to data that's stored in another Salesforce org. The cross-org adapter uses the standard Lightning Platform REST API. Unlike OData, the cross-org adapter directly connects to another org without needing an intermediary web service. • Custom adapter created via Apex — if the OData and cross-org adapters aren't suitable for your needs, develop your own adapter with the Apex Connector Framework.
Request and Reply	Suboptimal	Use Salesforce web service APIs to make ad-hoc data requests to access and update external system data. This solution includes the following approaches: Use Salesforce SOAP API. A custom Visualforce page or button initiates an Apex SOAP callout in a synchronous manner. In Salesforce, you can consume a WSDL and generate a resulting proxy Apex class. This class provides the necessary logic to call the remote service. A user-initiated action on a Visualforce page then calls an Apex controller action that executes this proxy Apex class to perform the remote call. Visualforce pages require customization of the Salesforce app. Use Salesforce REST API. A custom Visualforce page or button initiates an Apex HTTP callout (REST service) in a synchronous manner. In Salesforce, you can invoke HTTP services using standard GET, POST, PUT, and DELETE methods. You can use several HTTP classes to integrate with RESTful services. A user-initiated action on a Visualforce page then calls an Apex controller action that executes these proxy Apex classes to perform the remote call. Visualforce pages require customization of the Salesforce app. For more information on this solution, see Remote Process Invocation—Request and Reply.

Sketch

The following diagram illustrates how you can use Salesforce Connect to pull data from an external system using an OData adapter.



In this scenario:

1. The browser performs an AJAX call that in turn performs an action on the corresponding external object adapter.
2. The adapter translates the action into an OData request and makes an HTTP GET request to the remote system via the Integration and Services layers.
3. The remote system returns a JSON response to Salesforce via the Integration and Services layers.
4. The response is translated from OData into an external object and presented back to the browser.

Results

The application of the solutions related to this pattern allows for user-interface initiated invocations in which the result of the transaction can be displayed to the end user.

Calling Mechanisms

The calling mechanism depends on the solution chosen to implement this pattern.

Calling Mechanism	Description
External Objects	Salesforce Connect maps Salesforce external objects to data tables in external systems. Instead of copying the data into your org, Salesforce Connect accesses the data on demand and in real time. Even though the data is stored outside your org, Salesforce Connect provides seamless integration with the Lightning Platform. External objects are available to Salesforce tools, such as global search, lookup relationships, record feeds, and the Salesforce mobile app. External objects are also available to Apex, SOSL, SOQL queries, Salesforce APIs, and deployment via the Metadata API, change sets, and packages.
Lighting Components or Visualforce Pages	Used when the remote process is triggered as part of an end-to-end process involving the user interface, and the result must be displayed or updated in a Salesforce record. For example, a process that submits credit card payments to an external payment gateway and immediately returns payment results that are displayed to the user. Integration triggered from user interface events usually requires the creation of custom Lightning components or Visualforce pages.

Error Handling

It's important to include error handling as part of the overall solution. When an error occurs (exceptions or error codes are returned to the caller), the caller manages the error handling. The Salesforce Connect Validator is a free tool to run some common queries and notice error types and failure causes.

Benefits

Some of the benefits of using a Salesforce Connect solution are:

- This solution doesn't consume data storage in Salesforce.
- Users don't have to worry about regularly synchronizing data between the external system and Salesforce.
- A declarative setup that can be achieved quickly with OData, or a cross-org adapter, or using minimal code with a custom Apex adapter.
- Users can access external data with much of the same functionality as custom objects in the form of external objects.
- Ability to do a federated search in the connected external system using global search.
- Ability to run reports that access external data from cloud and on-premises sources. Refer to reporting considerations below.

Salesforce Connect Considerations

The Salesforce Connect solution has the following considerations:

- External objects behave like custom objects, but some features aren't available for external objects. For more information, see Salesforce Compatibility Considerations for Salesforce Connect.
- External objects can impact report performance. For more information, see Report Considerations for Salesforce Connect.
- For additional considerations for using Salesforce Connect adapters see Considerations for Salesforce Connect—All Adapters.
- If you're considering using a cross-org adapter, see Considerations for Salesforce Connect—Cross-Org Adapter.
- If you're considering using a OData adapter, see Considerations for Salesforce Connect—OData 2.0 and 4.0 Adapters.
- If you're considering using a custom Apex adapter, see Considerations for Salesforce Connect—Custom Adapter.

Security Considerations

Solutions for this pattern should adhere to standard Salesforce org-level security. It's recommended you use the HTTPS protocol to connect to any remote system. For more details, see Security Considerations.

If you're using an OData connector, make sure that you understand the special behaviors, limitations, and recommendations for Cross-Site Request Forgery (CSRF) on OData external data sources. For more information, see CSRF Considerations for Salesforce Connect — OData 2.0 and 4.0 Adapters.

Sidebars

Timeliness

Timeliness is of significant importance in this pattern. Keep the following points in mind:

- The request is typically invoked from the user interface, so the process must not keep the user waiting.
- Depending on the availability of and the connection to the external system, it can take a long time to retrieve external data. Salesforce has a configurable 120-second maximum timeout value to wait for a response from the external system.
- Completion of the remote process should execute in a timely manner and complete within the Salesforce timeout limit and within user expectations.

Data Volumes

This pattern is used primarily for small volume, real-time activities, due to the small timeout values and maximum size of the request or response for the Apex call solution. Don't use this pattern in batch processing activities in which the data payload is contained in the message.

Endpoint Capability and Standards Support

The capability and standards support for the endpoint depends on the solution that you choose.

Solution	Endpoint Considerations
Salesforce Connect	*OData APIs*—Uses the Open Data Protocol to access data that's stored outside Salesforce. The external data must be exposed via OData producers. *Other APIs*—Use the Apex Connector Framework to develop your own custom adapter when the other available adapters aren't suitable for your needs. A custom adapter can obtain data from any source. For example, some data can be retrieved from the internet via callouts, while other data can be manipulated or even generated programmatically. *Connect to Salesforce*—Uses the Lightning Platform REST API to access data that's stored in other Salesforce orgs. **Connect via Middleware** *Connect via Middleware*—The Salesforce Connect partner ecosystem has worked closely with Salesforce to make sure that their middleware gateways expose OData endpoints from their service so Salesforce can connect with them without writing additional code.
Request & Reply	**Apex SOAP Callouts** The endpoint must be able to receive a web service call via HTTP. Salesforce must be able to access the endpoint over the public Internet.

Solution	Endpoint Considerations
	Apex HTTP Callouts The endpoint must be able to receive HTTP calls. Salesforce must be able to access the endpoint over the public Internet. You can use Apex HTTP callouts to call RESTful services using the standard GET, POST, PUT, and DELETE methods.

State Management

When integrating systems, keys are important for on-going state tracking. For example, if a record gets created in the remote system, typically the record needs some sort of identifying key to support ongoing updates. There are two options:

- Salesforce stores the primary or unique surrogate key for the remote record.
- The remote system stores the Salesforce unique record ID or some other unique surrogate key. There are specific considerations for handling integration keys in this synchronous pattern.

Master System	Description
Salesforce	The remote system stores either the Salesforce record ID or some other unique surrogate key from the record.
Remote System	The call to the remote process returns the unique key from the application, and Salesforce stores that key value in a unique record field.

Complex Integrations

In certain cases, the solution prescribed by this pattern can require the implementation of a complex integration scenario. These scenarios are often solved using middleware. The scenarios include:

- Aggregation of calls and their results across calls to multiple systems
- Transformation of both inbound and outbound messages
- Maintaining transactional integrity across calls to multiple systems
- Other process orchestration activities between Salesforce and the external system

Governing Limits

Different limits apply for different adapters. For more details, see General Limits for Salesforce Connect.

Middleware Capabilities

The following table highlights the desirable properties of a middleware system that participates in this pattern.

Property	Mandatory	Desirable	Not required
Event handling		X	
Protocol conversion	X		
Translation and transformation	X		
Queuing and buffering	X		

Property	Mandatory	Desirable	Not required
Synchronous transport protocols	X		
Asynchronous transport protocols		X	
Mediation routing		X	
Process choreography and service orchestration		X	
Transactionality (encryption, signing, reliable delivery, transaction management)	X		
Routing			X
Extract, transform, and load			X
Long polling			X

External Object Relationships

External objects support standard lookup relationships, which use the 18-character Salesforce record ID to associate related records. However, data that's stored outside your Salesforce org often doesn't contain those record IDs. Therefore, two special types of lookup relationships are available for external objects: external lookups and indirect lookups.

This table summarizes the types of relationships that are available to external objects.

Relationship	Allowed Child Objects	Allowed Parent Objects	Parent Field for Matching Records
Lookup	Standard, Custom, External	Standard, Custom	The 18-character Salesforce record ID
External lookup	Standard, Custom, External	External	The External ID standard field
Indirect lookup	External	Standard, Custom	Select a custom field with the External ID and Unique attributes

High Data Volume Considerations for Salesforce Connect—OData 2.0 and 4.0 Adapters

If your org hits rate limits when accessing external objects, consider selecting the High Data Volume option on the associated external data sources. Doing so bypasses most rate limits, but some special behaviors and limitations apply. For more information, see High Data Volume Considerations for Salesforce Connect.

Client-Driven and Server-Driven Paging for Salesforce Connect—OData 2.0 and 4.0 Adapters

It's common for Salesforce Connect queries of external data to have a large result set that's broken into smaller batches or pages. You decide whether to have the paging behavior controlled by the external system (server-driven) or by the OData 2.0 or 4.0 adapter for Salesforce Connect (client-driven). The Server Driven Pagination field on the external data source specifies whether to use client-driven or server-driven paging. If you enable server-driven paging on an external data source, Salesforce ignores the requested page sizes, including the default `queryMore()` batch size of 500 rows. The pages returned by the external system determine the batches, but each page can't exceed 2,000 rows. However, the limits for the OData adapters for Salesforce Connect still apply.

Example

A manufacturing company uses Salesforce to manage customer cases. The customer service agents want to access the real-time order information from the back-office ERP system to get a 360 view of the customer without having to learn and manually run reports in ERP.

Implementing the solution prescribed by this pattern, you should:

- Configure your external data source with an OData endpoint. Your remote application may include native support for OData. For other applications, major integration vendors such as Dell Boomi, Informatica, Jitterbit, MuleSoft, and Progress Software have partnered with Salesforce on Salesforce Connect to build adapters.
- Point Salesforce Connect at the OData endpoint, either directly, or through a middleware solution.
- Sync your external database tables with external objects in Salesforce. When a user accesses a page with data from these external objects, Salesforce Connect makes real-time callouts to your back-end applications.

APPENDICES

APPENDIX A Resources—External

1. Hohpe, Gregor, and Bobby Woolf. *Enterprise Integration Patterns*. Boston: Addison-Wesley Professional, 2003.
2. Microsoft Corporation. *Integration Patterns (Patterns & Practices)*. Redmond: Microsoft Press, 2004.
3. "Synchronous vs. Asynchronous Communication in Applications Integration," MuleSoft, last accessed March 18, 2019, *https://www.mulesoft.com/resources/esb/applications-integration*.
4. "Hub and Spoke [or] Zen and the Art of Message Broker Maintenance," Enterprise Integration Patterns, last accessed March 18, 2019, *http://www.eaipatterns.com/ramblings/03_hubandspoke.html*.

APPENDIX B Resources—Salesforce

Developer Documentation

- *Salesforce Help: Give Integration Users API Only Access*
- *SOAP API Developer Guide*
- *REST API Developer Guide*
- *Streaming API Developer Guide*
- *Bulk API 2.0 and Bulk API Developer Guide*
- *Apex Developer Guide*
- *SOQL and SOSL Reference*
- *Salesforce Developer Limits and Allocations Quick Reference*
- *Platform Events Developer Guide*
- *Apex Developer Guide: Salesforce Connect*

Trailhead

- *Large Data Volumes*
- *Platform Events Basics*
- *Change Data Capture Basics*
- *Salesforce Connect*
- *Quick Start: Salesforce Connect*

APPENDIX C Security Considerations

To be effective members of the enterprise portfolio, all applications must be created and integrated with relevant security mechanisms. Modern IT strategies employ a combination of on-premises and cloud-based services.

While integrating cloud-to-cloud services typically focuses on web services and associated authorization, connecting on-premises and cloud services often introduces increased complexity. This section describes security tools, techniques, and Salesforce-specific considerations.

Reverse Proxy Server

"A reverse proxy is a server that sits in front of web servers and forwards client (e.g. web browser) requests to those web servers. Reverse proxies are typically implemented to help increase security, performance, and reliability."[10]

It's "a type of proxy server that retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client as though they originated from the proxy server itself. Unlike a forward proxy, which is an intermediary for its associated clients to contact any server, a reverse proxy is an intermediary for its associated servers to be contacted by any client."[11]

In Salesforce implementations, such a service is typically provided via an external gateway product. For example, open source options such as Apache HTTP, lighttpd, and nginix can be used. Commercial products include IBM WebSeal and Computer Associates SiteMinder. These products can be easily configured to proxy and manage all outbound Salesforce requests on behalf of the internal requester.

Encryption

Some enterprises require selected transactions or data fields to be encrypted between a combination of on-premises and cloud-based applications. If your organization must adhere to additional compliance requirements, you can implement alternatives, including:

- On-premises commercial encryption gateway services, including Salesforce's own, CipherCloud, IBM DataPower, Computer Associates. For each solution, the encryption engine or gateway is invoked at the transaction boundary by sending and receiving an encrypted payload or when encrypting or decrypting specific data fields before the HTTP(S) request executes.
- Cloud-based options, such as Salesforce Shield Platform Encryption. Shield Platform Encryption gives your data a whole new layer of security while preserving critical platform functionality. The data you select is encrypted at rest using an advanced key derivation system. You can protect your data more securely than ever before. Refer to the Salesforce online help for more information.

Specialized WS-* Protocol Support

To address the requirements of security protocols (such as WS-*), we recommend these alternatives.

[10] "What Is a Reverse Proxy?," Cloudflare, last accessed April 11, 2019, https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/.
[11] "Reverse proxy," Wikipedia, last accessed April 11, 2019, http://en.wikipedia.org/wiki/Reverse_proxy.

- Security/XML gateway—Inject WS-Security credentials (IBM WebSeal or Datapower, Layer7, TIBCO, and so on) into the transaction stream itself. This approach requires no changes to application-level web services or web service invocations from Salesforce. You can also reuse this approach across the Salesforce installation. However, it requires additional design, configuration, testing, and maintenance to manage the appropriate WS-Security injection into the existing security gateway approach.
- Transport-level encryption—Encrypt the communication channel using two-way SSL and IP restrictions. While this approach doesn't directly implement the WS-* protocol by itself, it secures the communication channel between the on-premises applications and Salesforce without passing a username and password. It also doesn't require changes to Salesforce-generated classes. However, some on-premises web services modifications might be required (at either the application itself or at the middleware/ESB layer).
- Salesforce custom development—Add WS-Security headers to the outbound SOAP request via the WSDL2Apex utility. This generates a Java-like Apex class from the WSDL file used to invoke the internal service. While this approach requires no changes to back-end web services or additional components in the DMZ, it does require:
  - an increased build and test effort
  - a relatively complex and manual process to hand-code the WS-Security attributes (including XML serialization within the Apex code)
  - a higher long-term maintenance effort

  Note: The last option isn't recommended due to its complexity and the risk that such integrations need periodic reviews based on regular updates to Salesforce.

APPENDIX D Event Driven Architecture

The Salesforce Enterprise Messaging Platform (EMP) with Platform Events, Streaming API, and Change Data Capture (CDC) enables enterprises to create event driven style architectures (EDAs).

An EDA decouples event message consumers (subscribers) from event message producers (publishers), allowing for greater scale and flexibility. Specific patterns are covered in this document as part of the existing pattern structure that compare and contrast related alternatives or options. However, getting a holistic understanding of the event driven architecture, and how patterns interplay, can help you select the right pattern for your needs.