

SALESFORCE CONTACT INTEGRATION

Document Owner : Himansh Sood | Last Updated: 13 April 2025

Document Version	v1.0
Document Status	On track
Objective	Automate the synchronization of Zeotap customer profile data into Salesforce Contact
Target Audience	Customer service agents, support managers, Salesforce admins, IT teams
Authentication Method	OAuth 2.0
Testing Tools	Postman, Salesforce Workbench
Relevant Document	https://developer.salesforce.com/docs/apis

TABLE OF CONTENTS

1. Overview
2. Goals
3. Data flow overview
4. Field Mapping
5. Authentication method
6. Key endpoints
7. Sample requests and responses
8. Key Constraints
9. Common API errors and fixes
10. Rate limiting policies
11. Assumptions and edge cases
12. API testing

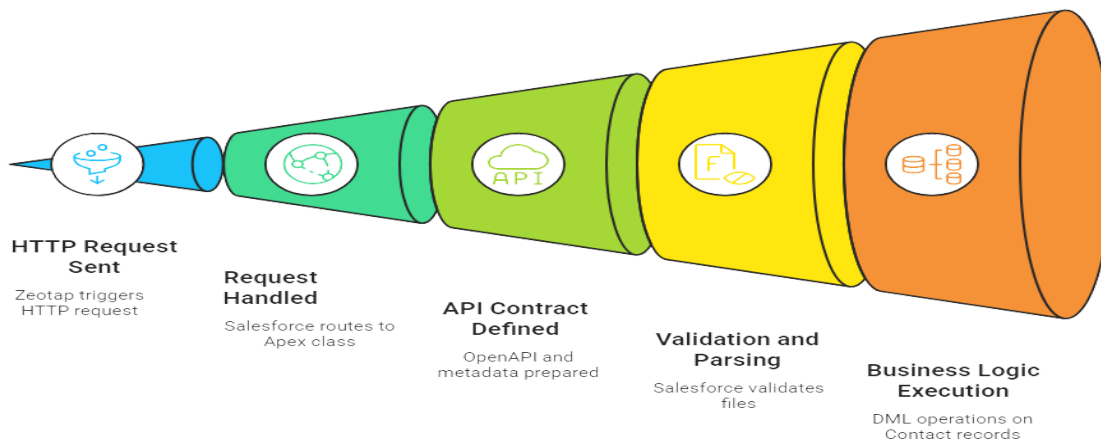
OVERVIEW

Salesforce Contact records do not display the rich customer data that Zeotap gathers from various sources. Agents lack important insights without this integration, which causes slow replies and missed opportunities to provide individualized service. Integration with Salesforce bridges the gap by raising operational effectiveness and improving customer satisfaction. The primary target audience for this integration is customer service agents and support managers who handle customer interactions using Salesforce.

GOALS

The purpose of this integration is to synthesize all the customer data from multiple sources into one, 360-degree view by directly integrating Zeotap's enriched customer data into Salesforce Contact records. Agents will have access to real-time insights during interactions, which will lead to drop in average handling time (AHT) by an estimated 10–15% by surfacing enriched data within 2 seconds of a query, faster response times, more customized service, and improved decision-making. Additionally, automating the data transfer eliminates manual entry errors and ensures that Salesforce always reflects the latest customer state within 5 minutes of an update on Zeotap.

DATA FLOW OVERVIEW



1. User updates his profile on Zeotap

The user performs an action such as creating a new profile, updating existing details (e.g., email, phone), or deleting their profile via a Zeotap-managed application or portal.

2. Zeotap sends an HTTP request to Salesforce

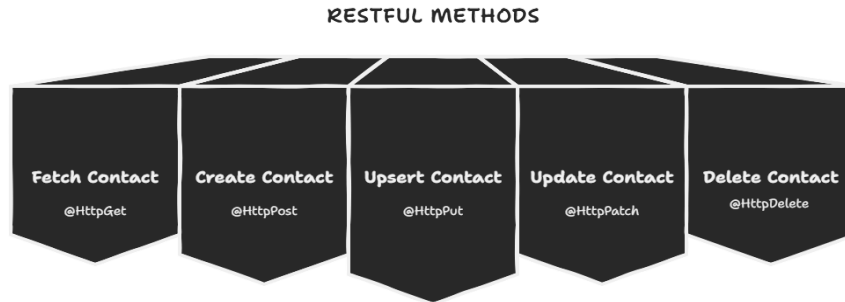
Zeotap detects the change and triggers the appropriate HTTP request to the Salesforce endpoint defined in the Apex REST class.

Example endpoint:

<https://<instance>.salesforce.com/services/apexrest/apex-rest-examples/v1/Contacts/<contactId>>

3. Apex REST class handles the request

Salesforce receives the HTTP request and routes it to the appropriate Apex class — in this case, `ContactManager.cls`. This class contains RESTful methods that interact with the **Contact** object by creating, updating, or deleting records based on the request type and payload.



4. OpenAPI and Metadata

The **OpenAPI document** (a `.yaml` file) defines what the API does — including endpoints, request/response formats, supported HTTP methods, and parameters.

The accompanying `.externalServiceRegistration-meta.xml` file provides the **metadata** Salesforce needs to internally register and expose this API for use (e.g., as agent actions).

Together, the `.yaml` and `.xml` files form an **API contract**, making the Apex REST API **discoverable and consumable** within Salesforce and by external systems.

5. Parsing and Validation

The YAML and XML files are parsed and validated by Salesforce. Each operation (GET, POST, PATCH, DELETE) is listed as a separate actionable endpoint.

Once validated, this API becomes **ready to be consumed**.

6. Salesforce Business Logic

The Apex method processes the request body or URL parameters received from Zeotap. It executes DML operations such as `insert`, `update`, or `delete` on Contact records, depending on the request type.

EXAMPLE

Use Case: A User wants to update their contact details on **Zeotap**-managed platform.

Expected data flow:

1. User updates his profile details (name, phone etc) in the Zeotap system.
2. Zeotap triggers a Data update event internally.
3. Zeotap identifies the matching user record using a unique identifier (e.g., CRM ID, email, or external ID).
4. Zeotap sends a RESTful API call to the Salesforce Apex REST endpoint.

5. Salesforce Apex Class (ContactManager.cls) receives and parses the request using the [@HttpPatch](#) or [@HttpPut](#) method.
6. **Data is updated** in the Contact record or a custom object.
7. Salesforce returns a response indicating success or failure.
8. Zeotap receives a success/failure response from the destination.



FIELD MAPPING

Field mapping is essential to ensure that data from Zeotap is accurately and reliably integrated into Salesforce Contact records. It aligns each data point with the correct Salesforce field, prevents sync errors, and ensures required fields are populated.

This section outlines the mapping between fields in Zeotap's customer profile schema and the corresponding fields in Salesforce's [Contact](#) object.

ZEOTAP FIELD	SALESFORCE CONTACT FIELD	REQUIRED
crm_id	External_Id__c	✓
first_name	FirstName	✗
last_name	LastName	✓
email	Email	✗
phone_number	Phone	✗
address.city	MailingCity	✗
address.zip	MailingPostalCode	✗
source_system	Source__c	✗

AUTHENTICATION METHODS

Salesforce uses the **OAuth 2.0** protocol to issue access tokens to clients. This is the most widely adopted and secure method of API authentication. It enables **secure, token-based communication** between Zeotap and Salesforce, ensuring that only authorized applications can access Salesforce data.

Here's how tokens can be obtained and accessed:

1. Register Zeotap as a Connected App

Register Zeotap as a Connected App in Salesforce and obtain **Client ID** and **Client Secret**.

2. Requesting for a token:

- Send a POST request to the Salesforce token endpoint:
<https://login.salesforce.com/services/oauth2/token>
- Include the following form fields in the request body:
 - grant_type: password
 - client_id
 - Client_secret
 - Username
 - Password

Note: GET does not work because of security reasons.

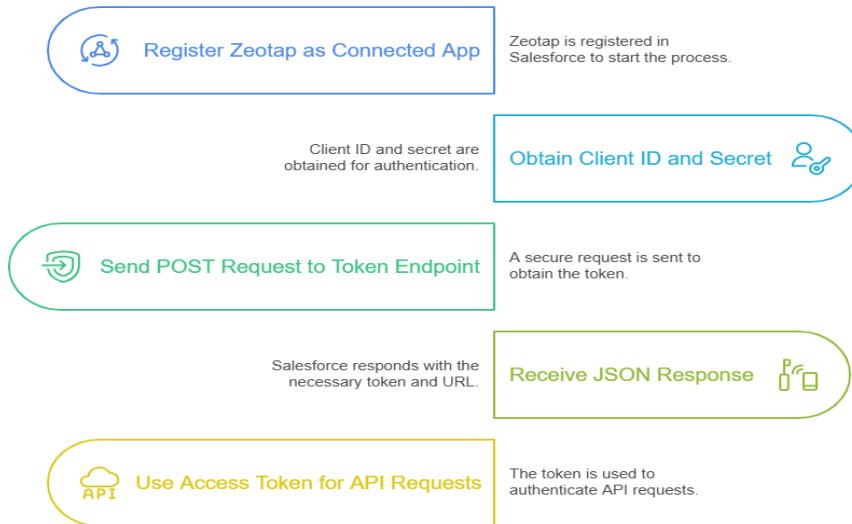
3. Salesforce Response

Upon successful validation, Salesforce returns a **JSON response** containing

- **access token** : A token used to authorize API requests.
- **instance_url** : The base URL for all API calls.

Note: Access token is time limited. Valid for 10-15 mins.

Salesforce OAuth 2.0 Token Acquisition



Common Edge Cases in Authentication:

1. “I’m trying to post my credentials, but I keep getting an “unauthorized” error.”

Where it fails: Token request to `/oauth2/token`

Status code: 401 Unauthorised

Fix: Double-check `client_id`, `client_secret`, username, and password.

2. “I used the correct URL, but nothing works — maybe it’s a network issue?”

Where it fails: Using HTTP instead of HTTPS

Status code: 403 or blocked

Fix: Make sure you’re using **HTTPS**, not HTTP.

3. “I’m logging in from a new IP and my password alone doesn’t work.”

Where it fails: Token request

Status code: 401 Unauthorized

Fix: Append your **security token** to the end of your password.

4. My API calls were working fine earlier, but now they all say “unauthorized.”

Where it fails: API request after token expiry

Status code: 401 Unauthorized

Fix: Get a **new access token** or implement token refresh logic.

5. “I posted the data as JSON, and now the request fails with a weird error.”

Where it fails: Token request

Status code: 400 Bad Request

Fix: Use `application/x-www-form-urlencoded` as the content type.

6. “I entered the wrong password a few times and now nothing works.”

Where it fails: Token request

Status code: 403 Forbidden

Fix: Wait for account unlock or reset password.

7. ***"I'm sending multiple login requests quickly and now they're blocked."***

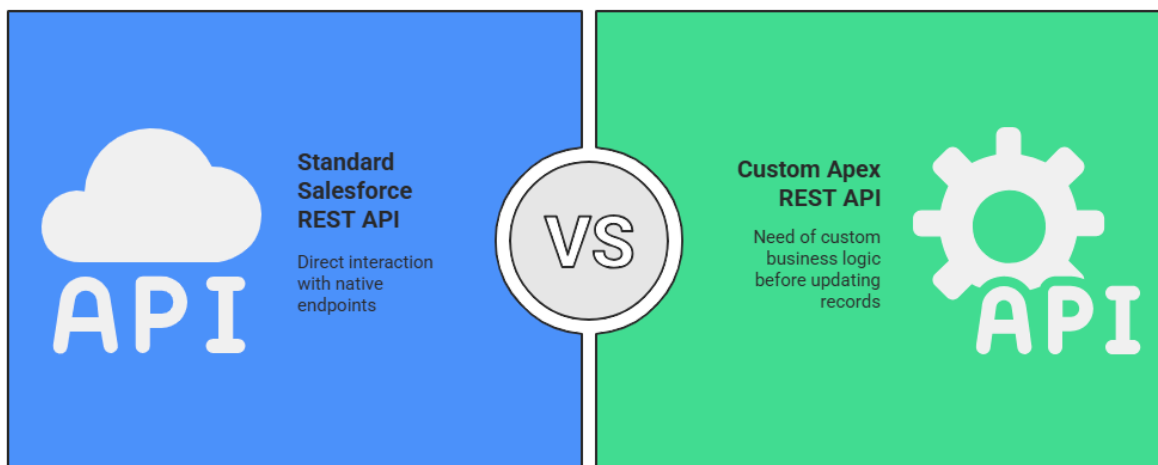
Where it fails: Token request

Status code: 429 Too Many Requests

Fix: Add a retry mechanism with **exponential backoff**.

KEY ENDPOINTS

SENDING DATA FROM ZEOTAP TO SALESFORCE CONTACT



Salesforce provides two principal methods of programmatic data access through its REST APIs: **Salesforce REST API** and **custom Apex REST APIs**.

Using the standard Salesforce REST API, one has access to predefined endpoints that allow for the execution of CRUD operations on standard and custom objects, including the creation or updating of Contact records via specific URL endpoints.

Conversely, Custom Apex REST APIs grant developers the capability of extending the capabilities of existing APIs by defining their own endpoints within Apex classes using the `@RestResource` annotation.

Both methods are secured with OAuth authentication and support common HTTP verbs like GET, POST, PATCH, and DELETE, but the selection between them lies in the use case customization and complexity needed.

1. Salesforce REST API Endpoint

a. AUTHENTICATION

Before making any API request, you must authenticate with Salesforce and obtain an access token.

Endpoint: **POST** <https://login.salesforce.com/services/oauth2/token>

Content-Type: [application/x-www-form-urlencoded](#)

b. CREATING A NEW CONTACT

Endpoint: **POST** </services/data/v60.0/subjects/Contact/>

REQUEST	RESPONSE
<pre>{ "FirstName": "John", "LastName": "Doe", "Email": "john.doe@example.com" }</pre>	<pre>{ "id": "0031t00000XXXXXAA2", "success": true, "errors": [] }</pre>

c. UPDATING AN EXISTING CONTACT

Endpoint: **POST** </services/data/v60.0/subjects/Contact/{ContactId}>

REQUEST	RESPONSE
<pre>{ "Phone": "9876543210", "Title": "Marketing Manager" }</pre>	No Content , Status code 204

COMMON ERRORS

ERROR	CAUSE	FIX
INVALID_FIELD	Misspelled or non-existent field in the request.	Use correct API field names from Salesforce Object Manager.
REQUIRED_FIELD_MISSING	A required field (e.g., LastName for Contact) was not included in the request.	Include all required fields for the object.
INVALID_TYPE_ON_FIELD_IN_RECORD	Field value has the wrong data type (e.g., string instead of date).	Ensure data types match the field definition.
DUPLICATES_DETECTED	Duplicate rules triggered due to similar existing records.	Provide unique values or modify duplicate rules in Salesforce.
INVALID_OR_NULL_FOR_RESTRICTED_PICKLIST	Invalid value sent for a restricted picklist field.	Use only values that exist in the picklist definition.
STRING_TOO_LONG	Field value exceeds the maximum allowed character length.	Shorten the string to comply with the field length limit.
INVALID_SESSION_ID / INVALID_TOKEN	OAuth token/session ID is invalid or expired.	Re-authenticate using OAuth 2.0 and obtain a valid access token.
INSUFFICIENT_ACCESS_OR_READONLY	User does not have permission to create or edit the record.	Ensure the user has proper object/field-level permissions via profiles or permission sets.
CANNOT_INSERT_UPDATE_ACTIVATE_ENTITY	Trigger, flow, or automation failed during insert/update.	Debug and fix the logic in the associated automation or Apex trigger.

2. Custom APEX REST API Endpoint

We generally use Apex classes when we need custom logic or validation not handled by standard Salesforce configuration

In a custom Apex REST class, the logic for each operation—such as creating, updating, or deleting a record—is defined using Apex methods. These methods are annotated with HTTP method annotations like `@HttpPost`, `@HttpPatch`, `@HttpGet`, etc., and are grouped under a `@RestResource` annotation, which defines the base URL of the API.

1. Creating a contact

Endpoint: **POST** `/services/apexrest/custom-api/v1/contacts/`

REQUEST	RESPONSE
<pre>{ "firstName": "Ravi", "lastName": "Verma", "email": "ravi@zeotap.com", "phone": "9876543210" }</pre>	<pre>"0038c00002xxxxxAAK"</pre>

2. Updating a contact

Endpoint:

PATCH /services/apexrest/custom-api/v1/contacts/0037F00000XXXXXX

REQUEST	RESPONSE
<pre>{ "Phone": "9876543210", "Title": "Marketing Manager" }</pre>	No Content , Status code 204

Understanding Response: It's a primitive string returned directly by the Apex method. It's just the ID of the record that was created or updated.

#	TITLE	USER STORY	ACCEPTANCE CRITERIA	PRIORITY	NOTES
1	Add Contact	As a user, I want to add a contact so that I can save their details	<ul style="list-style-type: none">- A form to enter name, phone, email- Save button stores the contact- Shows success message	High	Basic CRUD feature
2	Update Contact	As a user, I want to update an existing contact	<ul style="list-style-type: none">- Click edit on contact- Form pre-filled with contact details- Save updates info correctly	High	Editable inline or in separate form
3	Delete Contact	As a user, I want to delete a contact	<ul style="list-style-type: none">- Delete button available- Confirmation before delete- Contact is removed from the list	High	Soft delete optional
4	View Contact List	As a user, I want to view a list of all my contacts	<ul style="list-style-type: none">- List displays contact names & details- Pagination if too many- Search or filter available	High	Add sort/filter later
5	Search Contacts	As a user, I want to search for a contact by name or number	<ul style="list-style-type: none">- Search bar filters contact list- Case insensitive search	Medium	Real-time filtering preferred
6	View Contact Details Page	As a user, I want to view full contact details by clicking on a contact name	<ul style="list-style-type: none">- Contact card or detail page- Displays all saved fields	Low	Modal view or separate route

KEY CONSTRAINTS

1. Path parameters:

Caseid must be provided in path for get patch and delete

2. Authentication Constraints:

Authentication is handled using **OAuth 2.0 with a valid access token.**

3. Request Body Constraints:

Post: */Cases/*

Required: subject (String)

Optional: status, origin , priority

Put: */Cases/*

id is optional: If provided, the case will be updated. If not, a new case will be created.

Patch */Cases/{caseId}*

At least one updatable field must be included in the body. Only fields defined in the schema can be updated.

4. Data Format Constraints

1. All requests must have content type *application/json*.
2. All string values must be UTF-8 encoded and free from injection-pron characters (<, >, etc.).
3. Max length for subject: 255 characters.

5. General Constraints

1. API usage must respect Salesforce rate limits and daily quotas.

Required fields for Contacts

To ensure successful creation or update of a **Contact** in Salesforce, the following fields must be provided:

- *LastName* – Mandatory by default in Salesforce for Contact creation.
- Additional required fields – These may include *Email*, *Phone*, or other custom fields marked as required in your Salesforce org.

API Errors & Fixes

1. ***“Invalid credentials during token request”***
Where it fails: **POST** /services/oauth2/token
Status code: **401 Unauthorized**
Fix: Double-check credentials; ensure user has API access
2. ***“Using HTTP instead of HTTPS”***
Where it fails: Using HTTP instead of HTTPS
Status code: **403 Forbidden**
Fix: Always use **https://**
3. ***“Missing security token on new IP login”***
Where it fails: Token request
Status code: **401 Unauthorized**
Fix: Append your **security token** to the end of your password.
4. ***“Token expired”***
Where it fails: API request after token expiry
Status code: **401 Unauthorized**
Fix: Request a new token or implement refresh token flow
5. ***“JSON posted with wrong content type”***
Where it fails: Token request
Status code: **400 Bad Request**
Fix: Set proper **Content-Type** header in the request
6. ***“Too many failed login attempts”***
Where it fails: Token request
Status code: **403 Forbidden**
Fix: Wait for unlock or reset password
7. ***“Multiple rapid login attempts.”***
Where it fails: Token request
Status code: **429 Too Many Requests**
Fix: Add a retry mechanism with **exponential backoff**.
8. ***“Missing required fields in Contact creation”***
Where it fails: Token request
Status code: **400 Bad Request**
Fix: Ensure required fields (**LastName**, etc.) are provided

Rate Limiting Policies

Salesforce imposes limits on API usage based on the edition and user count of the org. If you exceed these, you may:

- Hit performance issues
- Be blocked from adding new ones

LIMIT TYPE	ENTERPRISE
Daily API Call Limit	100,000 per day
Concurrent Request Limit	25-50 concurrent calls
Bulk API Limit	10,000 records per batch

You can monitor usage via the [Limits API](#):

GET /services/data/vXX.X/limits

Measures to stay within limits:

1. Disable Specific Operations

To reduce the number of active operations in the API catalog:

Set `<active>false</active>` in the `ExternalServiceRegistration` metadata.

2. Delete an Entire API Registration

If an API is no longer required:

- Remove references (e.g., agent actions or flows).
- Generate a destructive manifest to delete the registration.

ASSUMPTIONS:

1. Developer has installed Agentforce for Developers via VS Code or is using Code Builder.
2. It's assumed that the developer already has access to a Salesforce org (e.g., an AI AppDev org) that can be used for Apex development and deployment.
3. developer has installed the MuleSoft for Agentforce API Design Extension, needed for validating OpenAPI documents.
4. The Salesforce Command Line Interface (sf CLI) is installed and properly configured
5. Apex class follows `@RestResource` and contains at least one method with an appropriate HTTP method annotation
6. The Apex REST class must include sharing, without sharing, or inherited sharing to be eligible for OpenAPI spec generation and registration.

7. Developers have successfully decomposed the External Service Registration (ESR) files using the required CLI command before making modifications in the YAML editor.
8. structure of the generated YAML and XML OpenAPI files is valid
9. The user understands how to use the Org Browser to retrieve and locate Apex REST classes stored at `/force-app/main/default/classes`.
10. All work is done inside a properly configured Salesforce DX project, with a valid `sfdx-project.json` file and connected to the default org.

EDGE CASES

1. If your Apex class lacks the `@RestResource` annotation, the system will not recognize it as a REST resource and it will be excluded from OpenAPI spec generation.
2. Classes without any method-level HTTP annotations (`@HttpGet`, `@HttpPost`, etc.) will be ignored during OpenAPI spec generation.
3. Using an unsupported or misspelled HTTP method annotation (e.g., `@HttpPatch` if not supported) can cause failures in spec generation or runtime errors.
4. Apex classes must explicitly declare sharing (`with sharing`, `without sharing`, or `inherited sharing`). If omitted, the class won't qualify for custom action generation.
5. If `urlMapping` contains invalid characters or dynamic path variables without proper formatting, the endpoint may not be callable or may not register correctly.
6. Even minor formatting errors in the OpenAPI YAML (like incorrect indentation or bad parameter types) will cause the MuleSoft extension or `sf generate spec` command to fail.
7. If the path in OpenAPI spec doesn't match the actual Apex class path (`urlMapping`), the action may be registered but will fail at runtime.
8. Trying to generate a spec or register the custom action before deploying the Apex class to the org will result in metadata not found or empty spec errors.
9. Two methods (even across different classes) with the same `operationId` can lead to conflicts when registering the OpenAPI spec.
10. If the parameter types in the OpenAPI spec don't match the Apex method's expected input types, the action might register but fail when called.
11. If you forget to re-deploy the modified YAML or XML ESR files after editing them, the custom action won't update in the org, even though your local files are correct.
12. Salesforce metadata is case-sensitive in many places. Having mismatched casing in paths, class names, or identifiers can lead to subtle bugs or failed deployments.

VALIDATING AND TESTING API

Before we send data from Zeotap to Salesforce using the API, we need to make sure everything works correctly. This means checking that the API is written properly, follows Salesforce's rules, and does what it's supposed to do.

The **MuleSoft for Agentforce Extension Pack (beta)** offers comprehensive tools to validate API specifications for correctness, governance compliance, and readiness for agent actions.

It's like a toolkit that helps us check and test our API. It has:

MuleSoft for Agentforce API Design Extension - A tool to design and review the API

MuleSoft for Agentforce Dependencies Extension - A tool that helps with Salesforce-specific needs

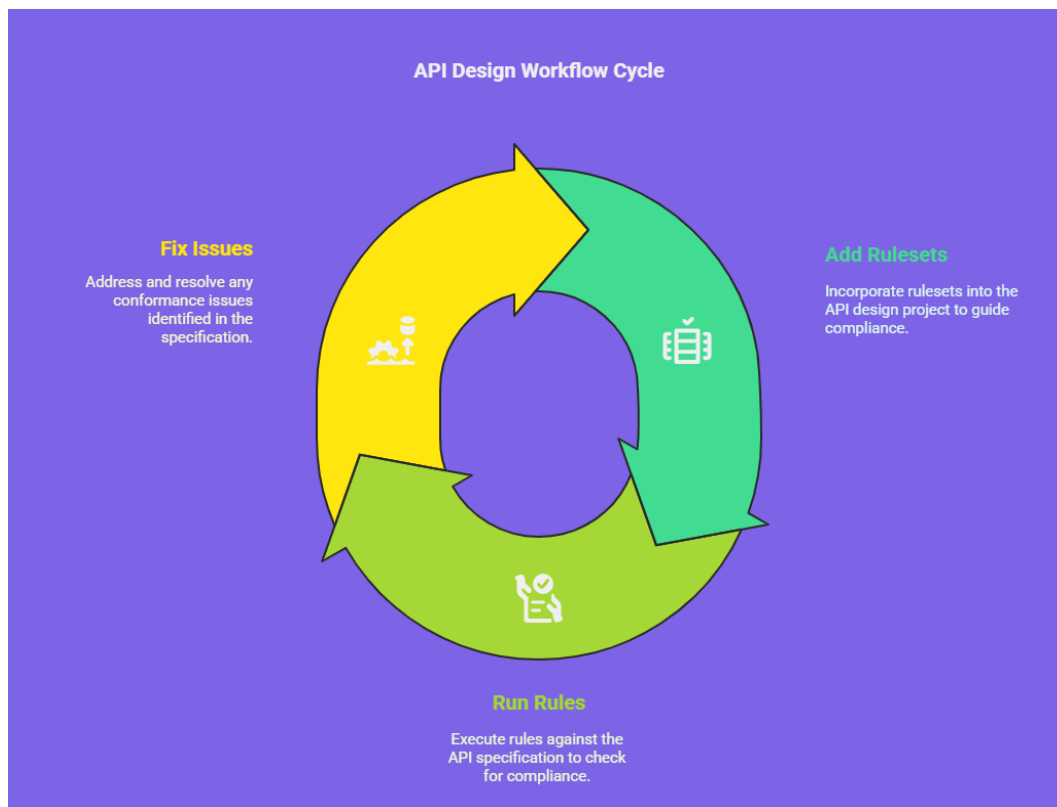
MuleSoft for Agentforce Platform Extension- A tool that checks the setup for the platform

Salesforce Set of rules:

1. **Salesforce API Topic and Action Enablement** - Rules to connect the API with Salesforce topics and actions
2. **Salesforce Apex REST Best Practices**-Best practices for using Apex REST (Salesforce's custom APIs)

Validating API against Rulesets

This document guides you through the process of running validations on your API specification to ensure it complies with organizational or platform-specific rules.



Workflow:

1. Add Rulesets to the API Design Project

- Rulesets (e.g., *Anypoint Best Practices*, *HTTPS Enforcement*) are added from Anypoint Exchange.
- Teams may create and publish custom rulesets.

2. Run Governance Validation

- Execute validation rules from the Governance Rulesets panel.
- Rules can be run at:
 - All rulesets level
 - Individual ruleset level
 - Single rule level
- Errors () and warnings () are highlighted in the Governance Results panel.

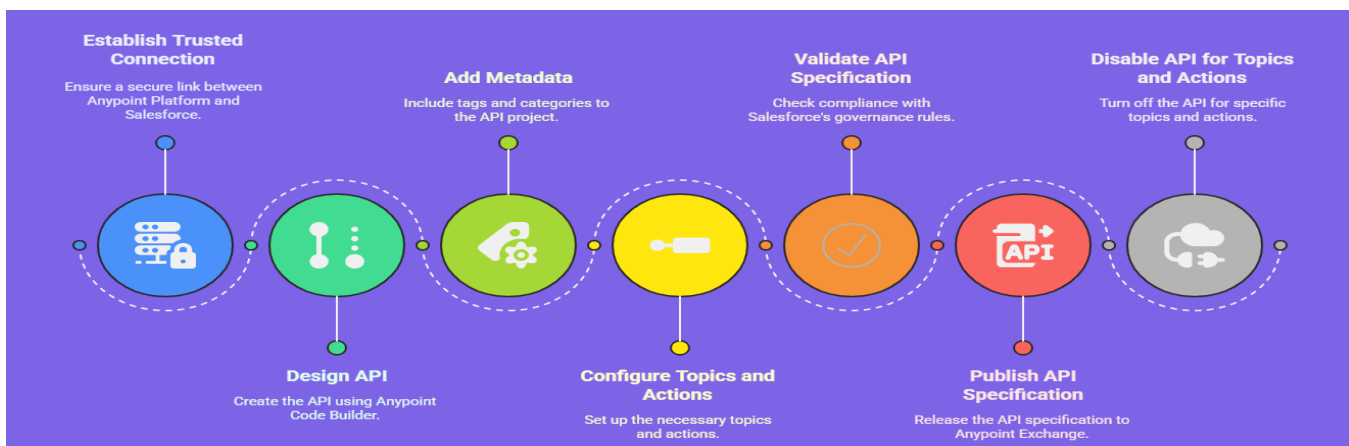
3. Fix Conformance Issues

- Developers modify the RAML/OpenAPI specification to address issues such as:
 - Using HTTPS protocol only.
 - Providing required API descriptions and documentation.
- Rules are re-executed post-fix to verify compliance.

Enabling an API Project for Topics and Agent Actions

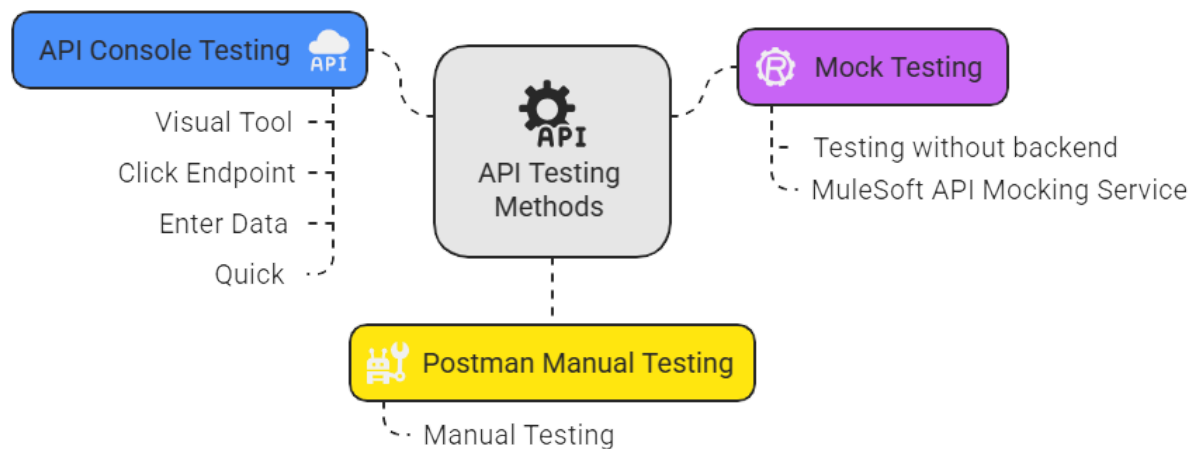
This doc provides instructions for enhancing your API definition so it can interact with Salesforce's real-time and event-driven systems, particularly through topics (like Pub/Sub) and agent actions (used by tools like Einstein or Agentforce).

It allows you to integrate MuleSoft APIs with Salesforce Agentforce, enabling automation tasks and real-time decision-making by Salesforce agents.



1. Ensure a trusted connection between Anypoint Platform and a Salesforce organization with agent entitlements.
2. Verify API Catalog is enabled in Salesforce.
3. Ensure the required permissions are in place for asset management, application deployment, and API design.
4. In Anypoint Code Builder, create a project and select Enable this API for Agent Topics and Actions.
5. Define Project Name, API Type (REST), API Specification Language (OAS 3.0), and Business Group.
6. Add metadata (tags, categories) to the API Project Properties tab, including sf-api-catalog and sf-api-topic tags.
7. Configure topics by defining labels, classification descriptions, scope, and instructions for agent actions.
8. Annotate API operations with x-sfdc/agent/action/publishAsAgentAction set to true for enabling actions.
9. Validate API governance rules using the SF API Topic and Action Enablement ruleset.
10. Publish the API specification to Anypoint Exchange to make it available in API Catalog.
11. To disable topics, deselect Enable this API for Agent Topics and Actions and apply changes.

API Testing Methods and Tools



API Console Testing (for PRD)

To validate the integration between Zeotap and Salesforce before deployment, we use API console testing tools such as **Postman** and **Salesforce Workbench**. These tools allow us to simulate authentication, record creation, updates, and error scenarios in a controlled environment.

Goals of Console Testing:

- Confirm endpoint accessibility
- Validate authentication and token usage
- Ensure payload formats match Salesforce expectations
- Test common success and error scenarios
- Document sample requests/responses

Test Examples:

- **POST** /services/oauth2/token: validate access token retrieval
- **POST** /subjects/Contact/: create new contact with required fields
- **PATCH** /subjects/Contact/{Id}: update existing contact details
- Simulate failures (e.g., missing **LastName**, invalid token)

Testing will be done using:

- **Postman** for request simulation and payload verification
- **Workbench** for native Salesforce testing
- **Manual token refresh handling** and error retry logic will be validated