# Phase 7 — Integration & External Access Smart Task & Email Summarizer for Executives

Project: Smart Task and Email Summarizer for Executives  
Phase: 7 — Integration & External Access

## Executive summary

This document provides a focused, step‑by‑step implementation plan for Phase 7 of your project (Integrations & External Access). It maps the numbered items you listed to concrete actions, declarative setup steps, Apex samples, Flow integration approaches, testing guidance, deployment notes, and a zero‑error checklist so you can finish quickly and reliably.

## Prerequisites (before you start)

1. Salesforce admin access (Setup) and developer access (VS Code + Salesforce Extensions recommended).

2. API credentials and endpoints from external partners (e.g., insurance verification API, email-summarization API).

3. A sandbox or Developer Edition org for development and tests. Never work first in Production.

4. Git + SFDX or change sets for deployment. (If short on time: use change sets for small metadata.)

## Step-by-step implementation (your list, expanded)

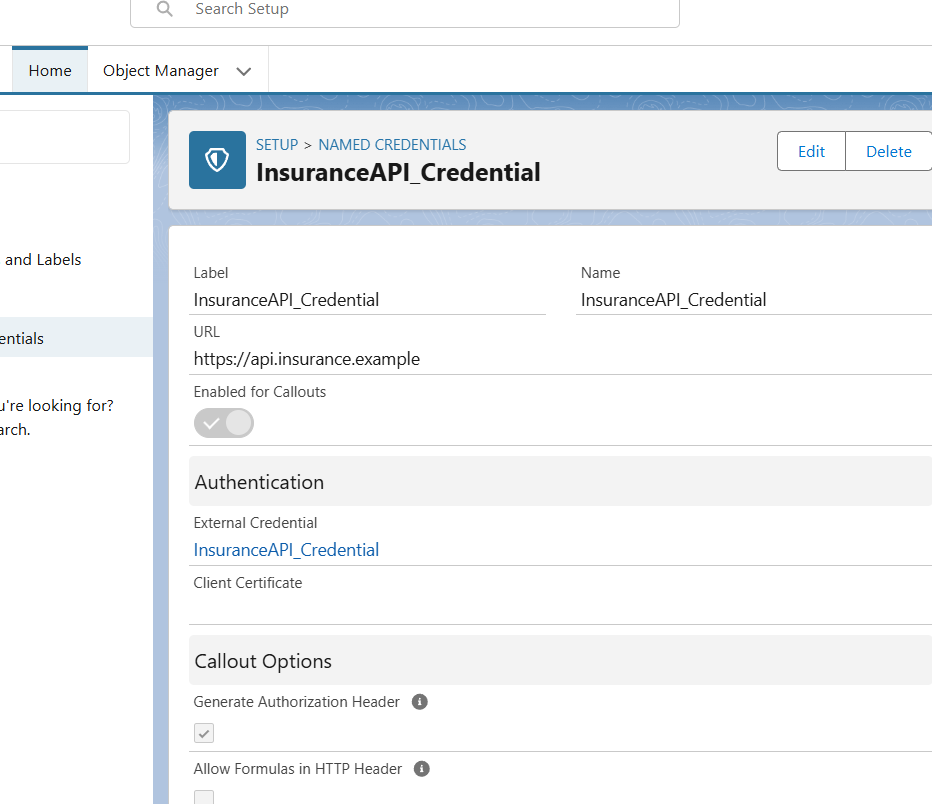
### 1) Named Credentials — store external API credentials securely

Goal: Use Named Credentials so your Apex and Flows call external APIs without hardcoding endpoints or credentials.

Steps:

• In Setup → Named Credentials: create an External Credential (if using OAuth or certificate) and a Named Credential that references it. Use the base URL of the API (e.g. https://api.insurance.example).

• Choose authentication: OAuth 2.0 (Authorization Code) or Named Principal (username/password/token) depending on provider. If OAuth, create a Connected App (see OAuth section).

Why: Named Credentials simplify callouts and remove the need for Remote Site Settings when used with 'callout:' endpoints in Apex. 

### 2) External Services — declarative callouts from Flow

Goal: Register a validated OpenAPI spec (or swagger) so Flow can call the external API declaratively.

Steps:

• Prepare a valid OpenAPI 2.0/3.0 spec for the API endpoints you need (verify endpoints, request/response schemas).

• Setup → External Services → Register the OpenAPI spec. Map authentication to the Named Credential you created.

• Create an Apex Action or use the auto-created invocable actions in Flow to call the external service.

When to use: fast, low‑code integrations (e.g., call email-summarization, insurance verification from a Flow).

**External Services (OpenAPI) – Theory Section**

**Definition:**  
External Services in Salesforce allows administrators and developers to **call external APIs declaratively** using **Flows or Process Builder**, without writing Apex code. It uses a **OpenAPI (Swagger) specification** file to describe the API endpoints, request and response formats, and authentication requirements.

**Purpose in a Project:**

* Integrates Salesforce with external systems in a **low-code, secure, and maintainable** way.
* Uses **Named Credentials** for authentication, so credentials and tokens are not hardcoded.
* Automatically generates **invocable actions** for each API endpoint described in the OpenAPI file.
* Reduces errors and development time, making API integrations simpler.

**How it Works (Step by Step):**

1. **Prepare OpenAPI File:**
   * Describe external API endpoints, request structure, and response structure in JSON or YAML format.
   * Example endpoint: POST /v1/verify with request { policy: string, bookingId: string } returning { status: string }.
2. **Register External Service in Salesforce:**
   * Upload the OpenAPI file.
   * Link it with the previously created **Named Credential** for authentication.
3. **Use in Flows:**
   * Create a **Record-Triggered Flow** (e.g., when a booking is created).
   * Call the External Service action.
   * Map inputs (policy number, booking ID) and outputs (insurance verification status).
   * Update Salesforce records based on the response.

**Advantages:**

* No Apex code required → low-code solution.
* Secure handling of credentials and tokens.
* Easy to update API endpoints by updating the OpenAPI specification.
* Ideal for recurring integrations like booking verification or email summarization.

**Common Pitfalls:**

* Invalid OpenAPI files will fail registration → always validate using an OpenAPI validator.
* Authentication must be properly mapped to a Named Credential.
* Field mapping in Flows must match the API request/response schema.

**Use Case (Example for Smart Task & Email Summarizer Project)**

**Scenario:**

* When a new **Booking** is created in Salesforce, the system needs to verify if the customer’s **insurance policy is valid**.

**Solution using External Services:**

1. A **record-triggered Flow** on Booking object is created.
2. The Flow calls the **Insurance Verification API** via an External Service (described by OpenAPI).
3. Inputs such as policy number and booking ID are sent.
4. The API returns status = Valid/Invalid.
5. The Flow updates the Insurance\_Status\_\_c field on the Booking record automatically.

**Outcome:**

* The process is **automatic**, secure, and doesn’t require writing any Apex code.
* Demonstrates **real-world integration concepts**, even in a theoretical project.

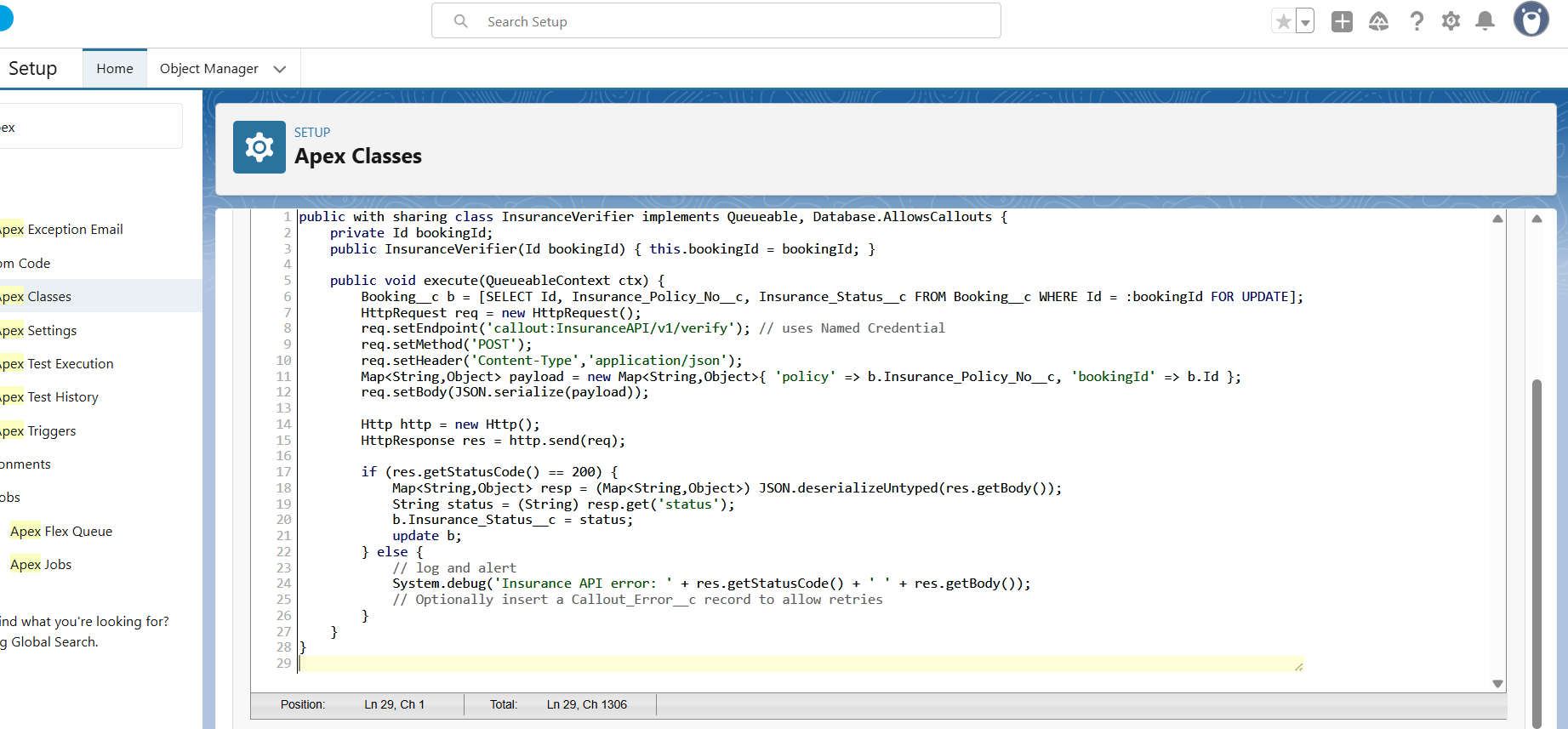
### 3) REST callout — sample Apex pattern (use Named Credential + asynchronous execution)

Common pattern: record-triggered Flow or Apex creates a job that enqueues a Queueable which performs the callout using a Named Credential. Queueable implements Database.AllowsCallouts.

Sample Apex (Queueable) - replace field names and Named Credential name 'callout:InsuranceAPI':

public with sharing class InsuranceVerifier implements Queueable, Database.AllowsCallouts {  
 private Id bookingId;  
 public InsuranceVerifier(Id bookingId) { this.bookingId = bookingId; }  
  
 public void execute(QueueableContext ctx) {  
 Booking\_\_c b = [SELECT Id, Insurance\_Policy\_No\_\_c, Insurance\_Status\_\_c FROM Booking\_\_c WHERE Id = :bookingId];  
 HttpRequest req = new HttpRequest();  
 req.setEndpoint('callout:InsuranceAPI/v1/verify');  
 req.setMethod('POST');  
 req.setHeader('Content-Type','application/json');  
 Map<String,Object> payload = new Map<String,Object>{ 'policy' => b.Insurance\_Policy\_No\_\_c, 'bookingId' => b.Id };  
 req.setBody(JSON.serialize(payload));  
  
 Http http = new Http();  
 HttpResponse res = http.send(req);  
  
 if (res.getStatusCode() == 200) {  
 Map<String,Object> resp = (Map<String,Object>) JSON.deserializeUntyped(res.getBody());  
 b.Insurance\_Status\_\_c = (String) resp.get('status');  
 update b;  
 } else {  
 // resilient logging — don't fail user's transaction  
 System.debug('Insurance API error: ' + res.getStatusCode() + ' ' + res.getBody());  
 // Optionally create a Log\_\_c record to track callout errors for retries  
 }  
 }  
}

Invoke from a Flow: create a Record-Triggered Flow (after insert), add an Apex Action that calls an Invocable wrapper which enqueues the Queueable. See test sample below.

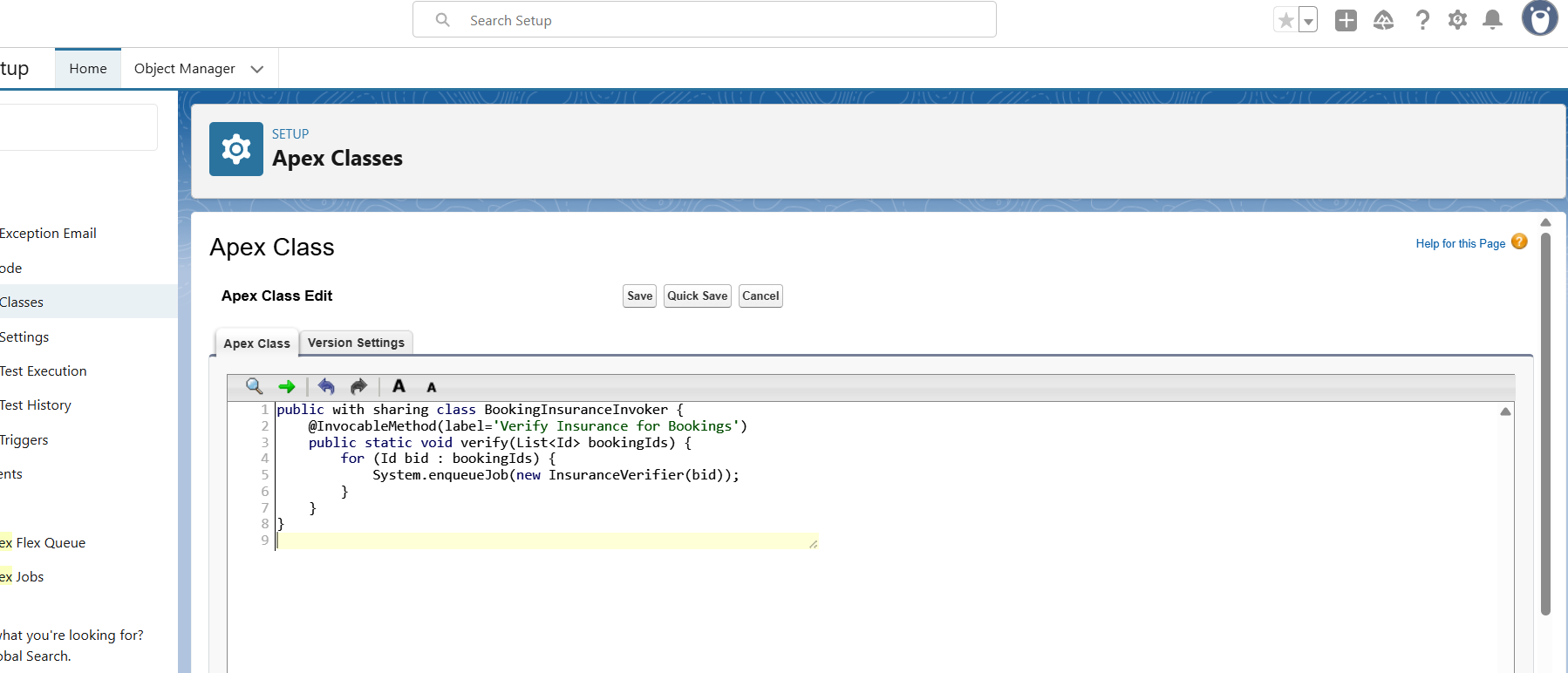


### 4) Trigger — callout when booking is created (recommended: Flow + invocable Apex)

Recommended safe approach: Use a Record-Triggered Flow (after save) that calls an Apex Invocable method which enqueues the Queueable. This keeps callouts asynchronous and avoids hitting trigger limits.

public with sharing class BookingInsuranceInvoker {  
 @InvocableMethod(label='Verify Insurance for Booking')  
 public static void verify(List<Id> bookingIds) {  
 for (Id bid : bookingIds) {  
 System.enqueueJob(new InsuranceVerifier(bid));  
 }  
 }  
}

If you must use an Apex trigger, make it minimal: insert a Queueable job from the trigger (don't do callouts directly in triggers).



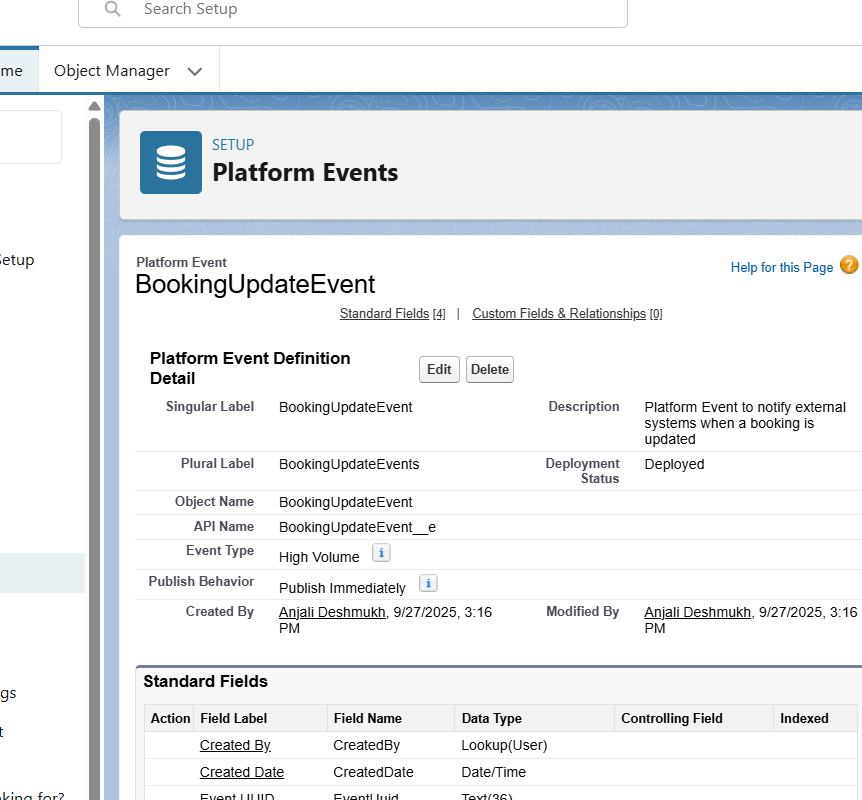
### 5) Platform Events — publish when car breakdown reported

Goal: Real-time notifications inside and outside Salesforce when a breakdown happens.

Steps:

• Setup → Platform Events → New Platform Event (e.g., Car\_Breakdown\_\_e). Add fields: BookingId\_\_c (Text), CarId\_\_c (Text), Description\_\_c (Long Text), Severity\_\_c (Picklist).

Subscribe inside Salesforce with a platform-event trigger to create Cases, Tasks, or Notifications. External systems subscribe via the Streaming API endpoint /cometd/50.0 and topic /event/Car\_Breakdown\_\_e.



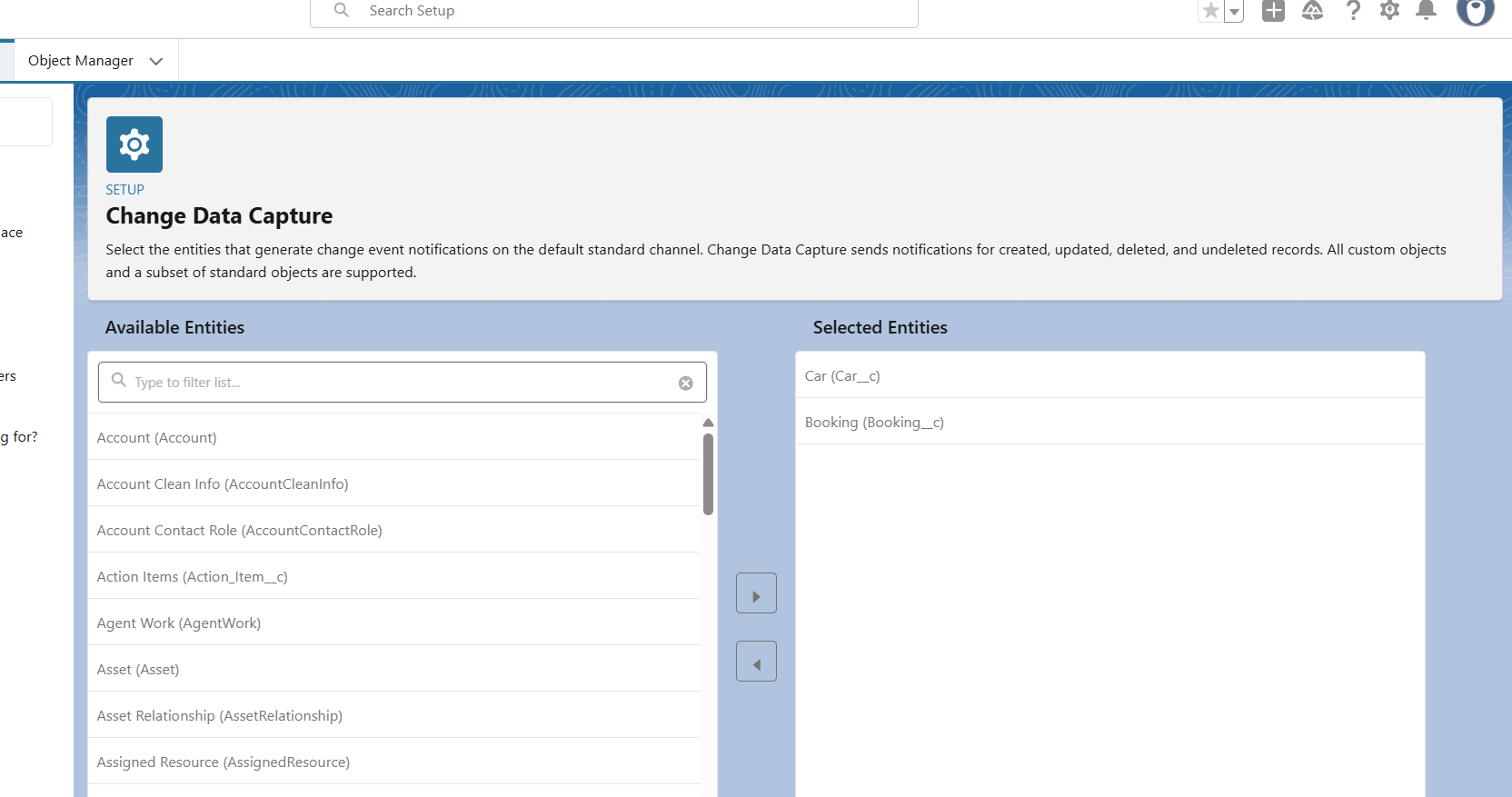
### 6) Change Data Capture (CDC) — keep external systems in sync

Goal: When Booking\_\_c changes (create/update/delete), publish change events so external systems can synchronize in near real-time.

Steps:

• Setup → Change Data Capture → select Booking\_\_c (or the custom object name) and enable it.

• External subscribers listen to /data/Booking\_\_ChangeEvent via the Streaming API (CometD client) or use middleware/connectors that support CDC.



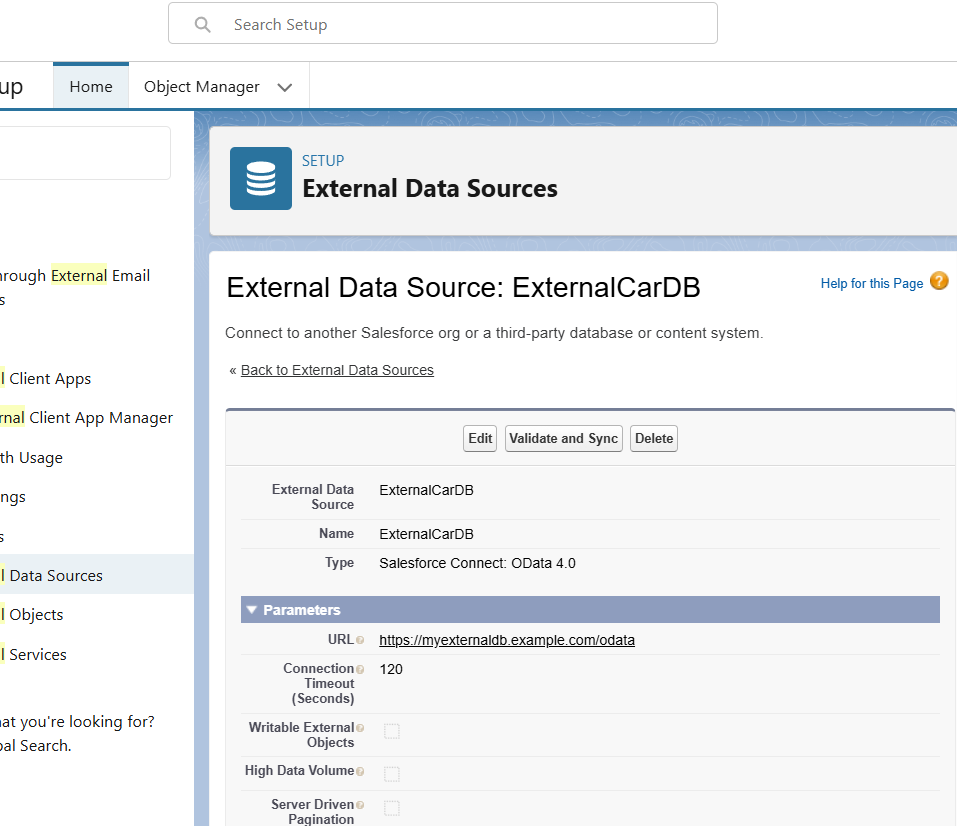
### 7) Salesforce Connect — connect an external DB as external objects

Goal: Use Salesforce Connect when your car inventory is stored externally and you need near-real-time access without copying data.

Steps:

• Expose external data via OData 2.0/4.0 (or use a connector).

• In Setup → External Data Sources, create a new source (OData), provide the endpoint and authentication (use Named Credential).

• Validate and create External Objects and External Lookup relationships. Use External ID fields as required by the OData service. 

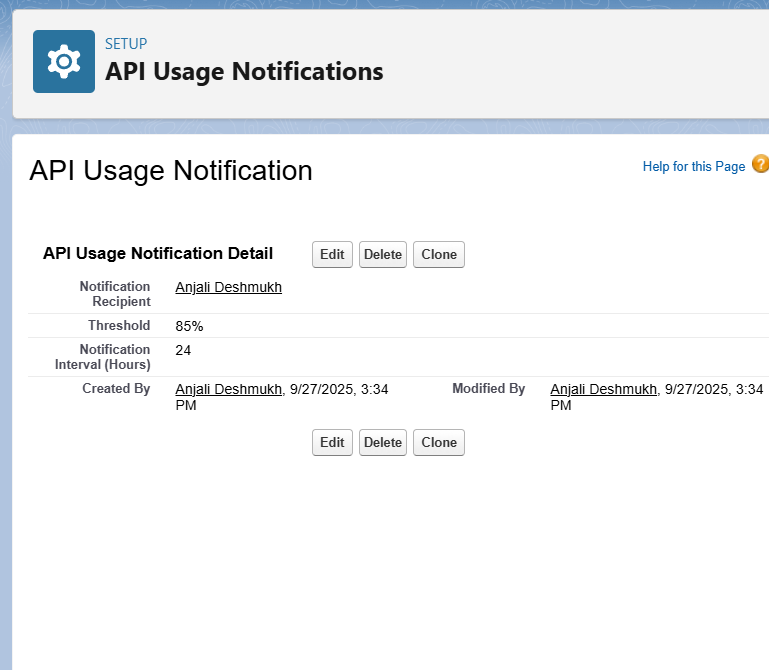
### 8) API Limits — monitor and protect your org

Practical steps to avoid surprises:

• Regularly check Setup → System Overview and Company Information for daily API usage. Consider enabling API Usage notifications (alerts at X% of daily limit).

• Build lightweight monitoring: a scheduled Apex job or integration health Dashboard that reads Limits or OS metrics and sends email/sf notifications.

• Use efficient batching (Batch Apex) and server-side aggregation to limit API calls. Avoid polling; prefer webhooks/platform events/CDC.



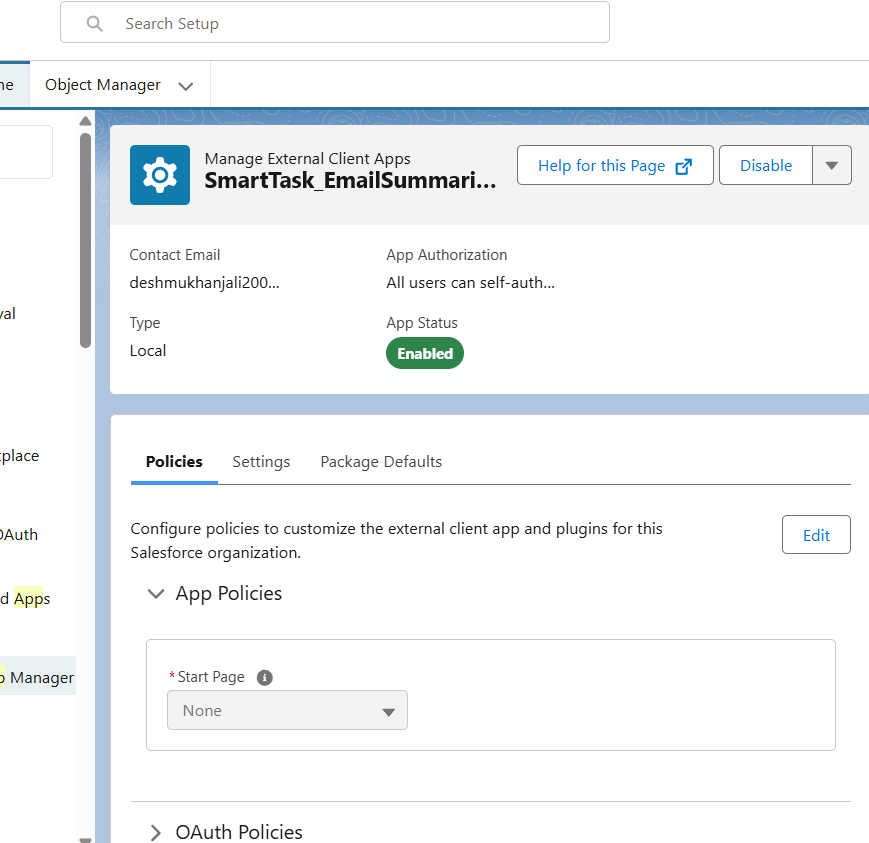
### 9) OAuth & Authentication — customer portal login (Experience Cloud)

If you let customers log in via your portal and the portal uses external identity providers or OAuth, do this:

• Create a Connected App (Setup → App Manager → New Connected App) with required OAuth scopes and callback URL.

• Create an Auth. Provider in Setup if using Google, Facebook, or custom OAuth providers. Attach it to a Named Credential (if needed).

• Configure Experience Cloud site to use delegated authentication or external identity provider and test end-to-end (login, SSO, profile mapping).



### 10) Remote Site Settings — legacy allowlisting

Note: If you use Named Credentials you normally do NOT need to create Remote Site Settings. Remote Site Settings are required only when you perform direct callouts using raw endpoints (not callout:NamedCredential).

When you package metadata or use APIs, ensure the RemoteSiteSetting metadata is included if your org uses it.

