🖹 MediConnect — Phase 7: Integration & External Access Documentation

Author: Neha Doddi
Org Alias: MediConnectOrg
Date: 2025-10-03

Purpose

In this phase, the MediConnect application was extended to communicate with an external system using Salesforce's integration features. The goal was to simulate how MediConnect could connect with third-party healthcare services such as lab systems, telemedicine APIs, or SMS gateways for sending alerts. For the purpose of demonstration, a public REST API (https://jsonplaceholder.typicode.com) was used to represent an external service. The integration was implemented through **Named Credentials** and an **Apex callout class**, ensuring security, scalability, and reusability.

Steps Implemented

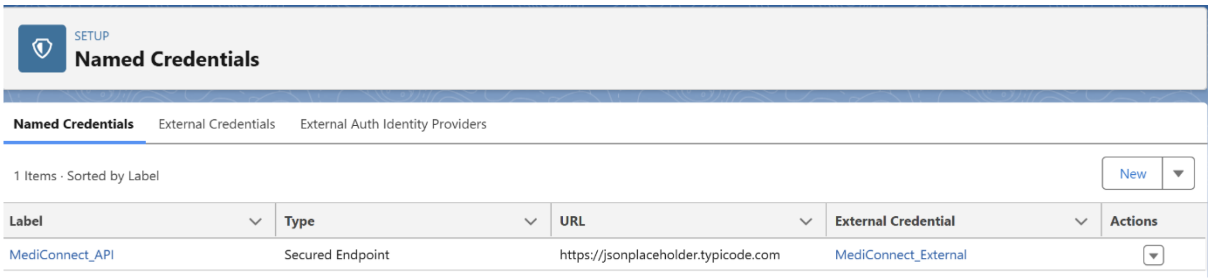
Step 1: Create Named Credential

- **Navigation:**
  Setup → Security → Named Credentials → New

The first step was to configure a **Named Credential** in Salesforce. A Named Credential provides a secure way to store endpoint URLs and authentication details, so developers do not need to hardcode them in Apex classes. In this project, a Named Credential named *MediConnect_API* was created and configured to allow callouts.

- **Details:**
  - Label: MediConnect_API
  - Name: MediConnect_API
  - URL: https://jsonplaceholder.typicode.com
  - Enabled for Callouts: ✅
  - Authentication: None (for test API)

The URL was set to https://jsonplaceholder.typicode.com, which acts as a mock API provider. In real-world healthcare use cases, this URL would point to actual lab systems or patient monitoring services, and authentication would be required.



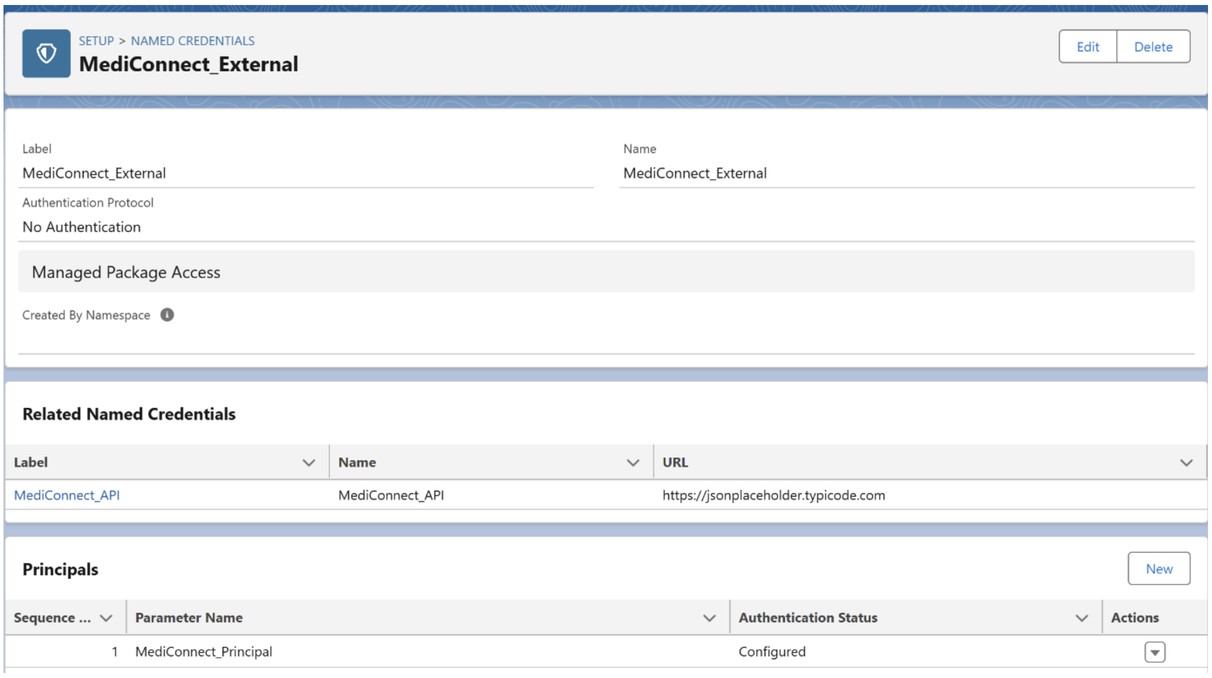
1 Items · Sorted by Label

Label	Type	URL	External Credential	Actions
MediConnect_API	Secured Endpoint	https://jsonplaceholder.typicode.com	MediConnect_External	▾

Step 2: Create External Credential & Principal

External Credential and Principal were created to define how Salesforce should authenticate with the external service. Although no authentication was needed for the mock API, this step establishes the structure for future scalability when secure authentication is required.

- **Navigation:**
  Setup → Security → External Credentials → New
- Created a **Principal** (MediConnect_Principal)
- Assigned Sequence Number = 1



MediConnect_External

Label	MediConnect_External
Name	MediConnect_External
Authentication Protocol	No Authentication

Managed Package Access

Created By Namespace

Related Named Credentials

Label	Name	URL
MediConnect_API	MediConnect_API	https://jsonplaceholder.typicode.com

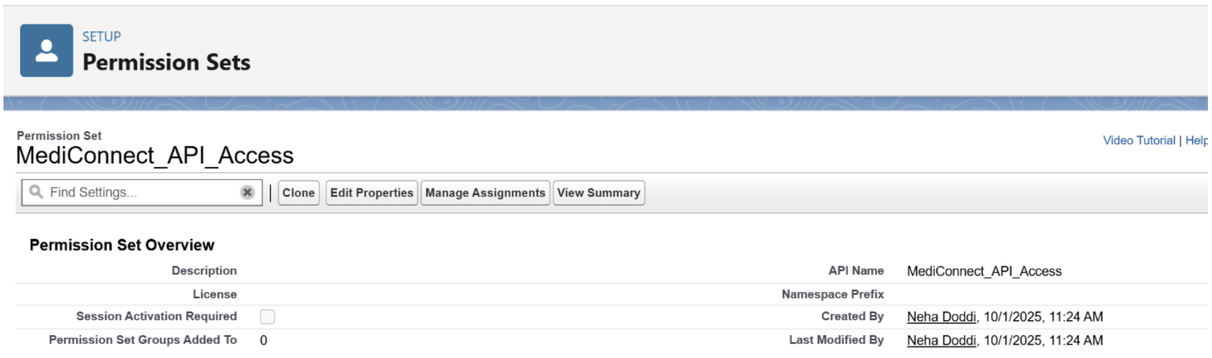
Principals

Sequence ...	Parameter Name	Authentication Status	Actions
1	MediConnect_Principal	Configured	

The principal was assigned a sequence number, and permissions were granted by assigning a **Permission Set** to the user, ensuring that only authorized profiles can execute the callout.

Step 3: Assign Permission Set

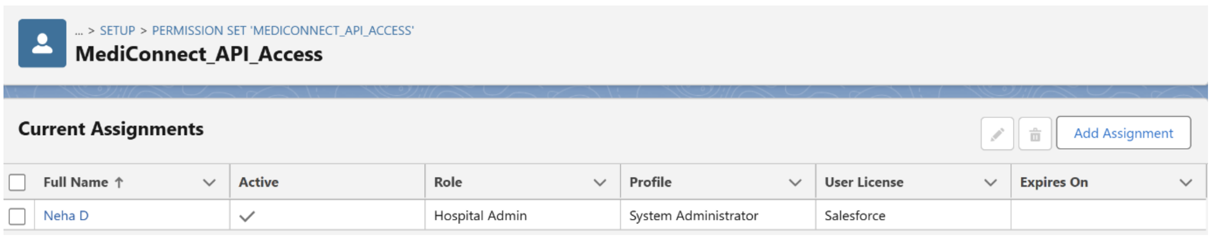
- Created a Permission Set (MediConnect_Integration_Access)
- Added External Credential permission
- Assigned to Self User


SETUP
Permission Sets

Permission Set
MediConnect_API_Access

Permission Set Overview

Description		API Name	MediConnect_API_Access
License		Namespace Prefix	
Session Activation Required	☐	Created By	Neha Doddi, 10/1/2025, 11:24 AM
Permission Set Groups Added To	0	Last Modified By	Neha Doddi, 10/1/2025, 11:24 AM


... > SETUP > PERMISSION SET 'MEDICONNECT_API_ACCESS'
MediConnect_API_Access

Current Assignments

Full Name ↑	Active	Role	Profile	User License	Expires On
Neha D	✓	Hospital Admin	System Administrator	Salesforce	

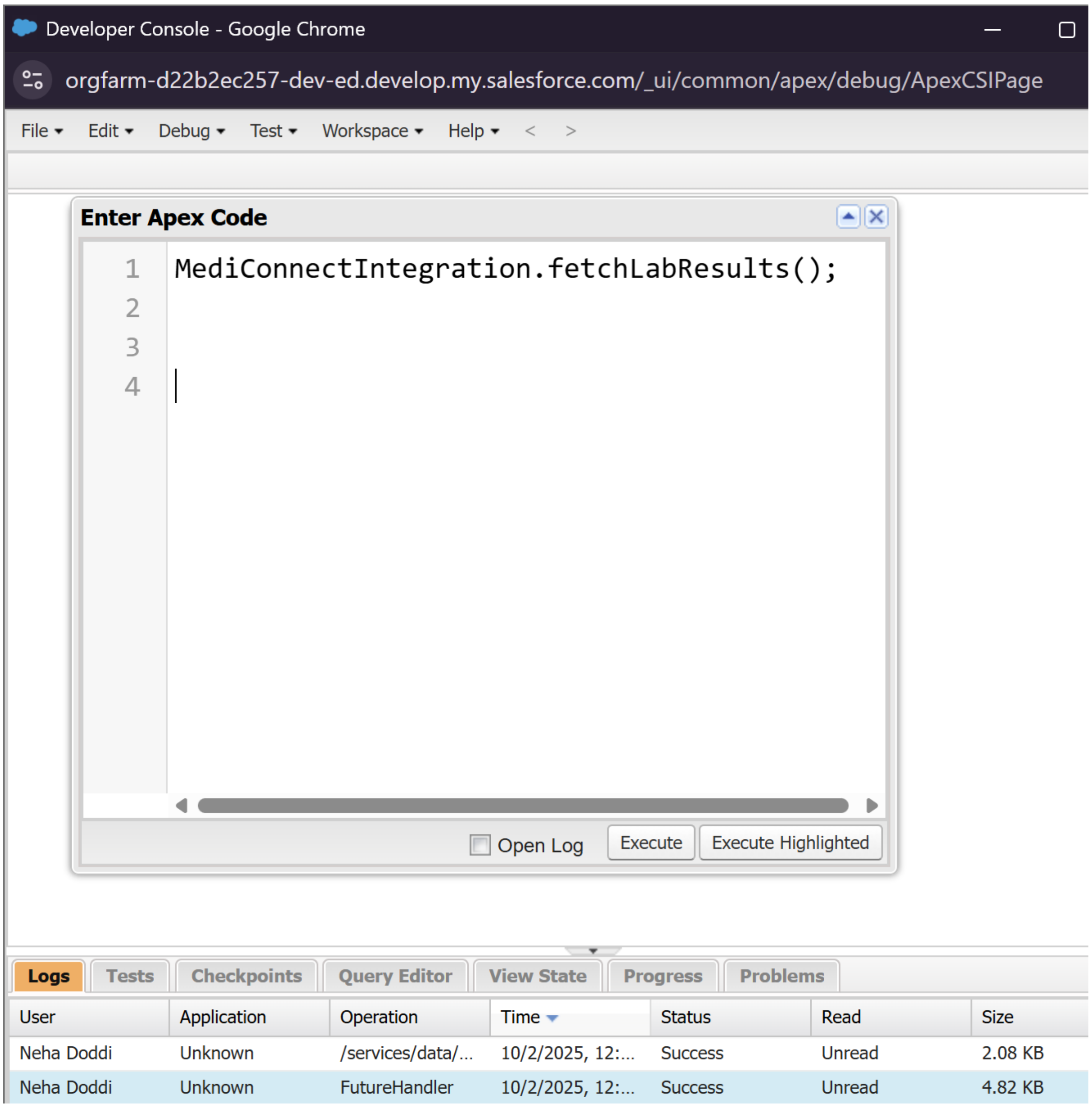
Step 4: Create Apex Class for Callout

```
public with sharing class MediConnectIntegration {
    @AuraEnabled
    public static void fetchLabResults() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('callout:MediConnect_API/posts');
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        System.debug('Response Status: ' + response.getStatus());
        System.debug('Response Body: ' + response.getBody());
    }
}
```

An **Apex integration class** was developed. The class MediConnectIntegration was written to perform a REST callout using the endpoint defined in the Named Credential. The method fetchLabResults() sends a GET request to the mock API and processes the response. This simulates retrieving patient lab results from an external system. The key advantage here is that the integration code remains clean and secure, as sensitive details like endpoint URLs and credentials are not hardcoded.

Step 5: Test Callout in Developer Console

- Opened Developer Console → Execute Anonymous → MediConnectIntegration.fetchLabResults();
- Status: **Success**


Developer Console - Google Chrome
orgfarm-d22b2ec257-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File Edit Debug Test Workspace Help
Enter Apex Code
Open Log
Execute
Execute Highlighted
Logs Tests Checkpoints Query Editor View State Progress Problems

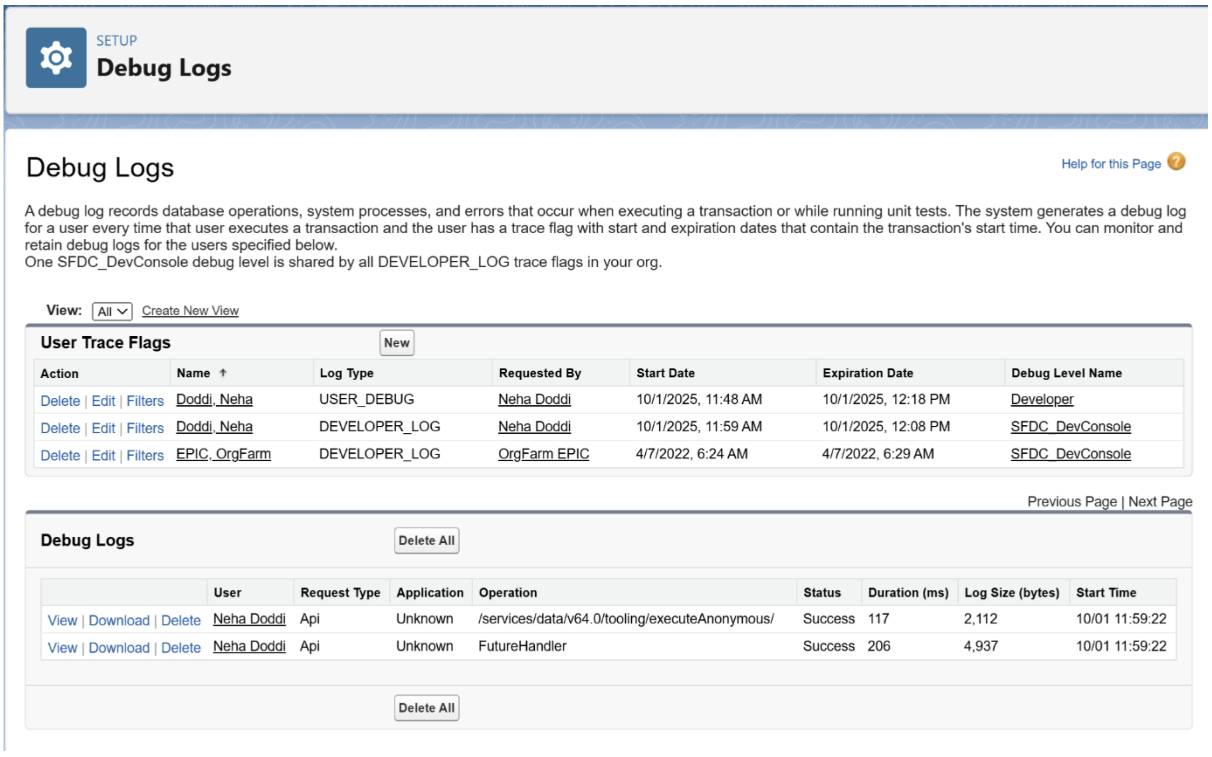
```
MediConnectIntegration.fetchLabResults();
```

User	Application	Operation	Time	Status	Read	Size
Neha Doddi	Unknown	/services/data/...	10/2/2025, 12:...	Success	Unread	2.08 KB
Neha Doddi	Unknown	FutureHandler	10/2/2025, 12:...	Success	Unread	4.82 KB

Step 6: View Debug Logs

- Navigation: Setup → Debug Logs → Add User Trace → Select Self → Run Apex → Refresh Logs
- Confirmed API Response displayed in debug logs.

Once the Apex class was implemented, the integration was tested using the **Developer Console**. By executing the line MediConnectIntegration.fetchLabResults(); in the console's Execute Anonymous Window, the callout was triggered, and the response was received successfully. To confirm and analyze the integration, **Debug Logs** were generated. Debug Logs provided detailed insights into the request and response, confirming that Salesforce successfully communicated with the external REST API.



SETUP

Debug Logs

Debug Logs

Help for this Page

A debug log records database operations, system processes, and errors that occur when executing a transaction or while running unit tests. The system generates a debug log for a user every time that user executes a transaction and the user has a trace flag with start and expiration dates that contain the transaction's start time. You can monitor and retain debug logs for the users specified below.
One SFDC_DevConsole debug level is shared by all DEVELOPER_LOG trace flags in your org.

View: All | Create New View

User Trace Flags New

Action	Name ↑	Log Type	Requested By	Start Date	Expiration Date	Debug Level Name
Delete | Edit | Filters	Doddi, Neha	USER_DEBUG	Neha Doddi	10/1/2025, 11:48 AM	10/1/2025, 12:18 PM	Developer
Delete | Edit | Filters	Doddi, Neha	DEVELOPER_LOG	Neha Doddi	10/1/2025, 11:59 AM	10/1/2025, 12:08 PM	SFDC_DevConsole
Delete | Edit | Filters	EPIC, OrgFarm	DEVELOPER_LOG	OrgFarm EPIC	4/7/2022, 6:24 AM	4/7/2022, 6:29 AM	SFDC_DevConsole

Previous Page | Next Page

Debug Logs Delete All

	User	Request Type	Application	Operation	Status	Duration (ms)	Log Size (bytes)	Start Time
View | Download | Delete	Neha Doddi	Api	Unknown	/services/data/v64.0/tooling/executeAnonymous/	Success	117	2,112	10/01 11:59:22
View | Download | Delete	Neha Doddi	Api	Unknown	FutureHandler	Success	206	4,937	10/01 11:59:22

Delete All