Medicare Cloud — Phase 7: Integration & External Access

Goal: Securely integrate Salesforce with external systems, enable real-time events and data replication, and manage API access and authentication for Medicare Cloud. This document provides step-by-step instructions, example code snippets, and best practices for Named Credentials, External Services, Web Services (REST/SOAP), Callouts, Platform Events, Change Data Capture, Salesforce Connect, API Limits, OAuth & Authentication, and Remote Site Settings.

1. Named Credentials (recommended for callouts)

Purpose: Named Credentials store endpoint URLs and authentication details centrally. Use them in Apex callouts via the "callout:" prefix so you don't hard-code credentials or endpoints.

Steps:

1. Setup → Quick Find → Named Credentials → New Named Credential.

2. Fill Label and Name (example: Label = Medicare_API, Name = Medicare_API).

3. URL: https://api.example.com (root endpoint).

4. Identity Type: Named Principal (or per User if needed).

5. Authentication Protocol: OAuth 2.0 (recommended) or Password Authentication for simple integrations.

6. For OAuth: create an Authentication Provider or a Connected App (see OAuth section). Select it here.

7. Allow Merge Fields if required, set Generate Authorization Header if using basic auth. Save.

Using in Apex:

Example callout using Named Credential (no hard-coded URL):

```
HttpRequest req = new HttpRequest();
req.setEndpoint('callout:Medicare_API/v1/patients/123');
req.setMethod('GET');
Http http = new Http();
HttpResponse res = http.send(req);
```

System.debug(res.getBody());

2. Remote Site Settings (legacy method)

Purpose: If you do not use Named Credentials, you must register external endpoints in Remote Site Settings to allow callouts.

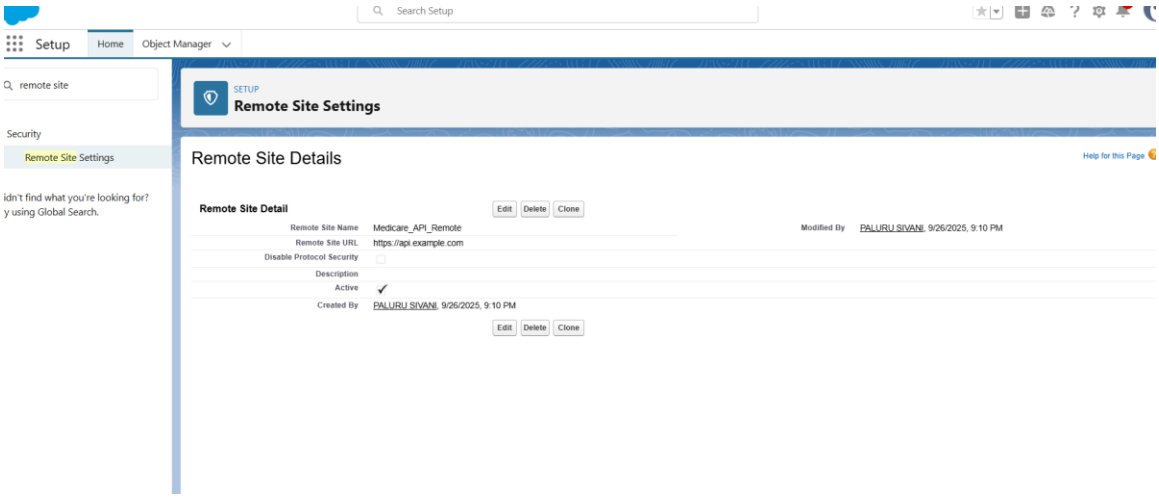
Steps:

1. Setup → Quick Find → Remote Site Settings → New Remote Site.

2. Remote Site Name: Medicare_API_Remote; Remote Site URL: https://api.example.com. Save.

3. Now Apex HTTP callouts to the domain are allowed (but credentials still need to be handled securely).

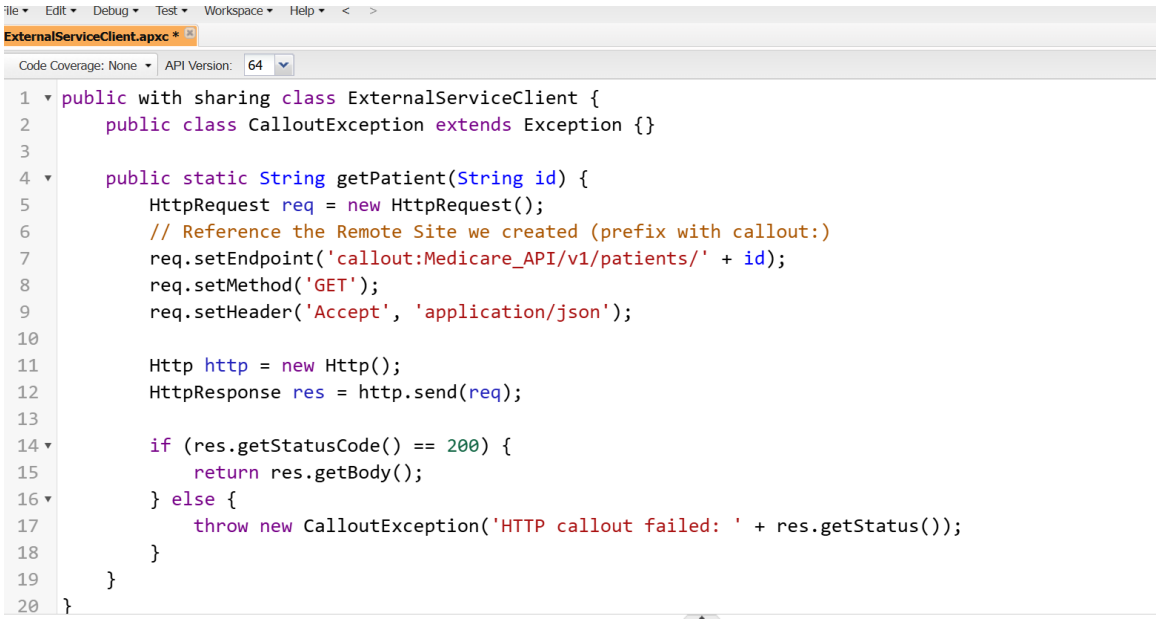
Note: Prefer Named Credentials for better security and simpler code.



3. Callouts (Apex HTTP callouts)

Purpose: Apex performs outbound HTTP requests to consume external REST/SOAP services.

Synchronous callout example (GET):




```apex
public with sharing class ExternalServiceClient {
    public class CalloutException extends Exception {}

    public static String getPatient(String id) {
        HttpRequest req = new HttpRequest();
        // Reference the Remote Site we created (prefix with callout:)
        req.setEndpoint('callout:Medicare_API/v1/patients/' + id);
        req.setMethod('GET');
        req.setHeader('Accept', 'application/json');

        Http http = new Http();
        HttpResponse res = http.send(req);

        if (res.getStatusCode() == 200) {
            return res.getBody();
        } else {
            throw new CalloutException('HTTP callout failed: ' + res.getStatus());
        }
    }
}
```

Notes:

- For long-running requests from UI, use Continuation or async patterns.

- To callouts from triggers, use @future(callout=true) or Queueable Apex to avoid transaction limits.

4. Web Services — Exposing Salesforce (Apex REST & SOAP)

Apex REST

1. Create an Apex class annotated with @RestResource(urlMapping='/Patients/*').

2. Implement @HttpGet, @HttpPost methods. Example:

```apex
@RestResource(urlMapping='/Patients/*')
global with sharing class PatientREST {
  @HttpGet
  global static Patient__c getPatient() {
    RestRequest req = RestContext.request;
    String id = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
    return [SELECT Id, Full_Name__c, Date_of_Birth__c FROM Patient__c WHERE Id = :id];
  }
}
```

Apex SOAP

1. Create a global Apex class and expose methods as webservice.

2. Generate WSDL by accessing the SOAP endpoint and share it with external systems. Example:

```
global class PatientSOAPService {
  webService static String ping(String msg) {
    return 'pong: ' + msg;
  }
}
```

3. The SOAP endpoint is /services/Soap/class/PatientSOAPService and you can download WSDL.

Security: Protect methods with with sharing and access checks; use OAuth or IP/restrictions.

5. External Services (OpenAPI & Named Credentials)

Purpose: Register an external REST API via an OpenAPI spec and call it as declarative actions (invocable) in Flows and Process Builder.

Steps:

1. Prepare OpenAPI / Swagger spec for the external API (JSON or YAML).

2. Setup → Quick Find → External Services → New External Service.

3. Give a name, and either upload the OpenAPI spec file or enter the spec URL.

4. Create or select a Named Credential for authentication.

5. Save — Salesforce will create Apex actions you can invoke from Flows.

Use case: call external eligibility check from a Flow without writing Apex.

6. Salesforce Connect (External Objects)

Purpose: Bring external data into Salesforce as External Objects (via OData or other connectors) without storing data in Salesforce.

Steps:

1. Setup → Quick Find → External Data Sources → New.

2. Choose Type (OData 2.0/4.0, Cross-Org, etc.), fill URL and authentication.

3. Validate and then click Sync to create External Objects.

4. Add External Object Tabs, create relationships (Indirect Lookup), and use in UI & Reports (some limits apply).

7. Platform Events (Pub/Sub)

Purpose: Asynchronous pub/sub model for event-driven integration (decoupled).

Steps to create & use:

1. Setup → Platform Events → New Platform Event. Define fields (e.g., PatientId__c, Status__c).

2. Publish events from Apex:

```
PatientEvent__e ev = new PatientEvent__e(PatientId__c = 'a012345', Status__c = 'UPDATED');
Database.SaveResult sr = EventBus.publish(ev);
```

3. Subscribe:

- Server-side: create an Apex trigger on the platform event:

```
trigger PatientEventTrigger on PatientEvent__e (after insert) {
  for (PatientEvent__e ev : Trigger.New) {
    // process event
  }
}
```

- Client-side: use CometD / Streaming API or Lightning empApi to subscribe in LWC/JS.

Best practices: use durable platform events for guaranteed delivery and design idempotent consumers.

8. Change Data Capture (CDC)

Purpose: Automatically receive events for create/update/delete/undelete on Salesforce records. Great for syncing external systems.

Steps:

1. Setup → Change Data Capture → Select objects to enable CDC (e.g., Patient__c, Appointment__c).

2. Subscribe using Streaming API (CometD) or Platform Events consumers.

3. The event payload contains changed fields and change type (CREATE/UPDATE/DELETE/UNDELETE).

Use case: keep an external data warehouse in sync or trigger downstream workflows.

9. API Limits & Monitoring

Purpose: Monitor and manage API usage to avoid hitting org limits.

Ways to view limits:

- Setup → System Overview shows API calls used in last 24 hours.

- Use REST endpoint: /services/data/vXX.0/limits to programmatically check limits.

Best practices:

- Use Bulk API for large data loads (batching).

- Use caching, reduce polling, prefer Platform Events or CDC for real-time sync.

- Implement retry with exponential backoff on transient failures.

10. OAuth & Authentication (Connected Apps)

Purpose: Configure authentication for external clients and Named Credentials.

Steps to create a Connected App (for OAuth):

1. Setup → App Manager → New Connected App.

2. Provide Name, Contact Email, and enable “Enable OAuth Settings”.

3. Set Callback URL (for web server flow) and select OAuth Scopes (e.g., full, api, refresh_token).

4. Save. Note the Consumer Key and Consumer Secret shown — use in OAuth flows.

Common OAuth flows:

- Web Server Flow (Authorization Code) for web apps.

- User-Agent Flow for single-page apps.

- JWT Bearer Flow for server-to-server integrations using certificates.

- Refresh tokens to obtain long-lived access.

Security: limit scopes and set IP restrictions / session policies for connected app.

11. External Authentication Providers

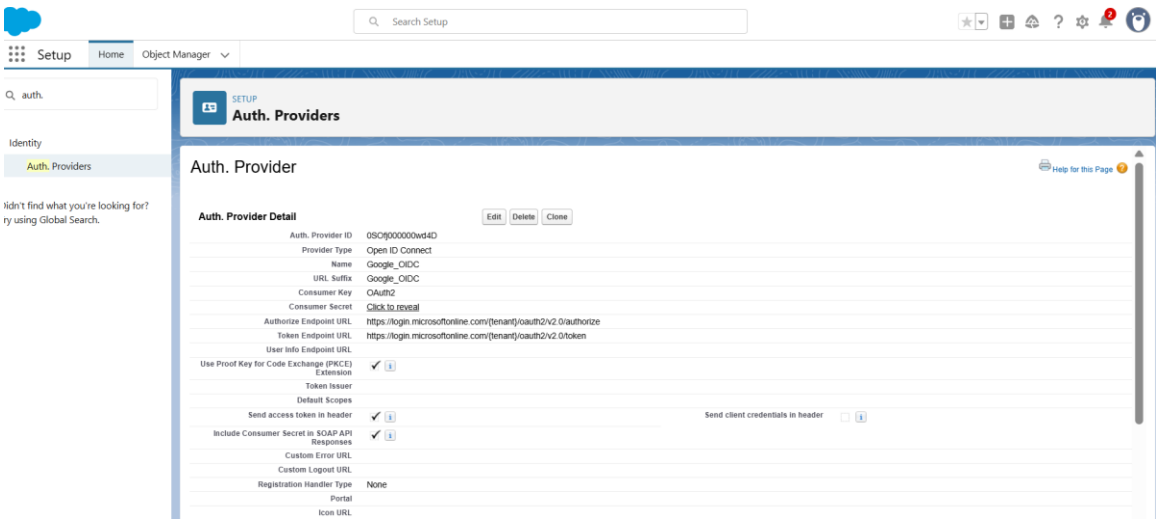
Purpose: Configure auth providers (Google, Azure AD, OpenID Connect) for Named Credentials or Communities.

Steps:

1. Setup → Auth. Providers → New.

2. Select provider (OpenID Connect / SAML) and fill client id/secret and endpoints.

3. Use this Auth Provider in Named Credentials or Connected Apps.



12. Security & Best Practices

Key recommendations:

- Prefer Named Credentials over Remote Site Settings.

- Use OAuth/JWT for secure server-to-server authentication and rotate keys/certificates periodically.

- Use TLS (HTTPS), validate certificates and avoid self-signed certs in production.

- Use per-user Named Principal if calls should run in context of the logged-in user.

- Log and monitor failed callouts and platform event delivery issues.

- Use retries and queueing for resilient integrations.

13. Troubleshooting Tips

Common issues & fixes:

- "Callout not allowed: endpoint not found in Remote Site Settings" → Add Remote Site or use Named Credential.

- "401 Unauthorized" → verify OAuth token, refresh token, or credentials.

- "403 Forbidden" → check connected app policies and permission sets.

- Streaming subscription disconnects → check replayId handling, authenticate, and increase client reconnect logic.

- Platform Event delivery failures → inspect Apex trigger logs and enable durable events if needed.

14. Example Integration Patterns

Patterns to consider:

- Synchronous Request/Response: Use Apex REST or external API callouts for immediate responses (small payloads).

- Asynchronous Processing: Use Platform Events, CDC, Queueable/Batch Apex for heavy processing.

- Data Virtualization: Use Salesforce Connect for read-only or near-real-time access to large external datasets.

- Bulk Data: Use Bulk API for ETL and large data migrations.

15. Deliverables & Checklist

• Named Credentials configured for each external system

• External Services registered (if using OpenAPI)

• Apex callouts implemented and tested (unit tests with mock callouts)

• Connected Apps (OAuth) configured for integrations requiring user consent

• Platform Events and CDC enabled and sample subscribers implemented

• Salesforce Connect external data sources configured and synced

• Monitoring & alerting for API usage and integration failures documented