**Slack–Salesforce Integration Documentation**

**Overview**

This integration enables Salesforce to send real-time notifications to Slack channels whenever a Case is created or updated.

We use a **Queueable Apex class** with **callouts** to Slack's **Incoming Webhook** URL, making it **asynchronous and governor-limit friendly**.

1. Create Slack App Incoming Webhook.
2. Configure External Credential and Named Principal (Slack\_Cred, SlackPrincipal).
3. Deploy SlackNotificationQueueable.
4. Implement Trigger or Flow on Case.
5. Deploy Slack\_Message\_Log\_\_c and add to page layouts or reports.

**1. Slack Setup:**

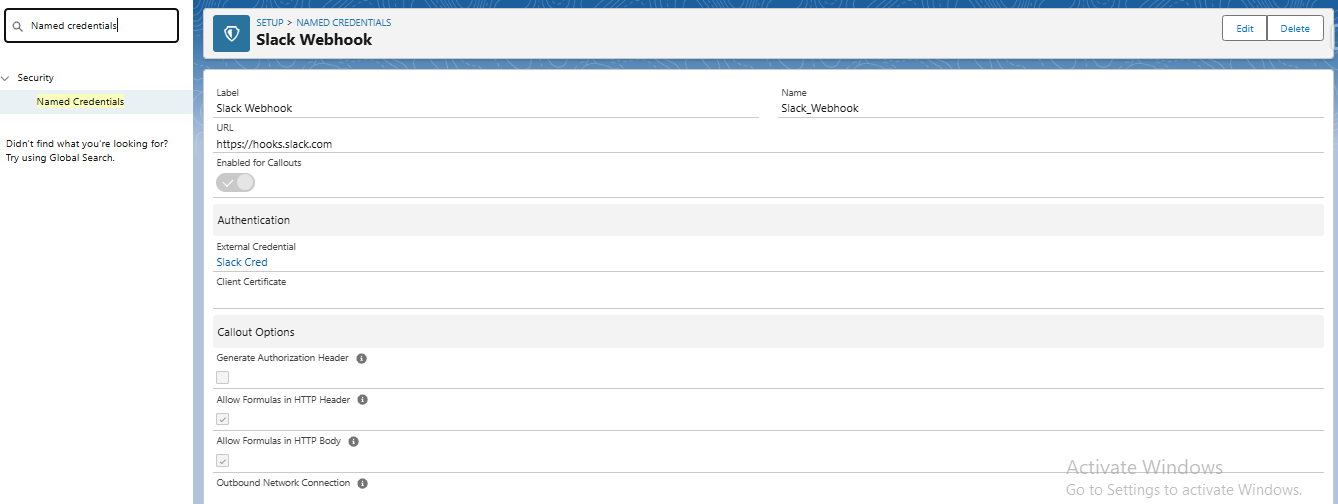
### Set up Slack App & Webhook

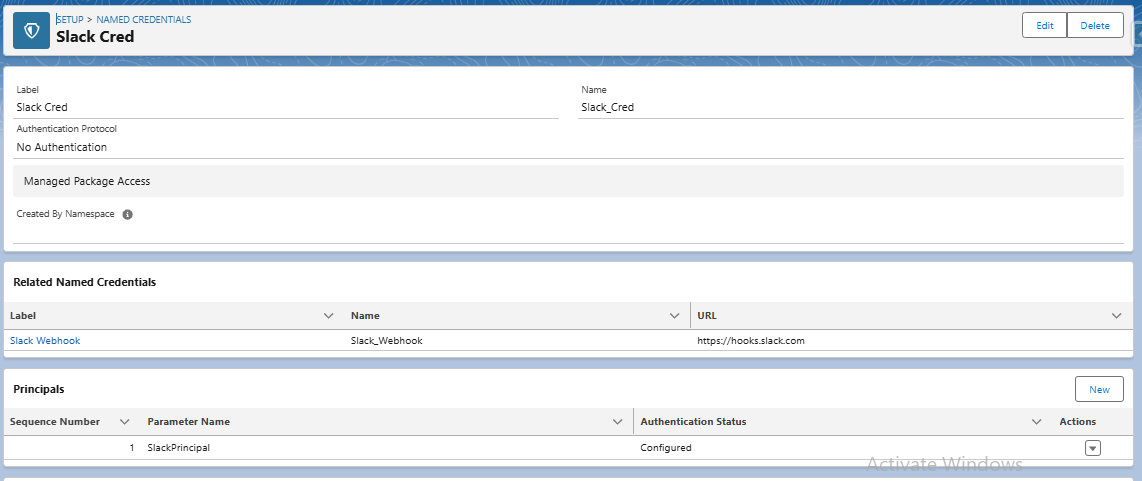
* Go to: https://api.slack.com/apps  
  Click **“Create New App”**
  + Choose **From Scratch**
  + Give it a name (e.g., "Salesforce Case Alerts")
  + Select your **Workspace**
  + Create new channel (#support-alerts)
* Under Channel features, go to **Integrations** option
  + Click **Add an App** in App section
  + Install “**Incoming Webhooks**” app
  + Select a channel to send messages to (e.g., #support-alerts)
  + Copy the generated **Webhook URL** for JSON payloads (e.g., <https://hooks.slack.com/services/TXXXXX/BXXXXX/XXXXXXXX>)

**2. Salesforce Setup:**

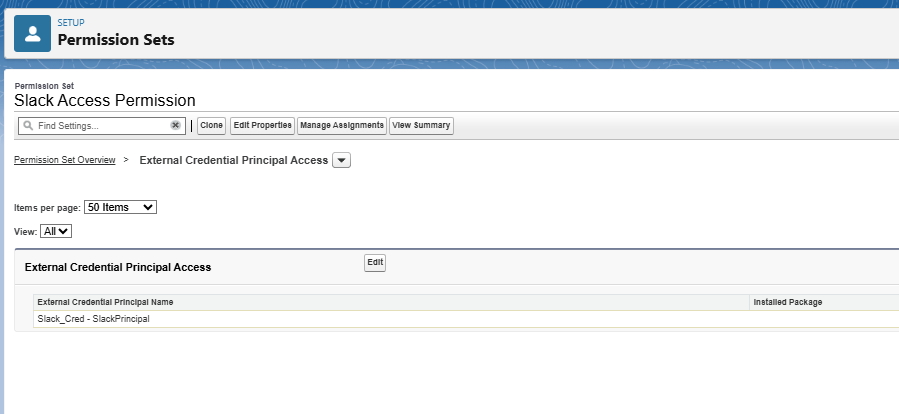
* Avoids hardcoding credentials
* Allows Salesforce admins to control access with a **Permission Set**.

Go to setup in Salesforce and follow steps as below,

* Named Credentials (Slack\_Webhook)  
  
* External Credential (Slack\_Cred)  
  Principals (SlackPrincipal)



* **Permission Set**  
  Create new ‘Slack Access Permission’  
  go to ‘External Credential Principal Access’ and select the ‘External Credential Principals’



**3. Queueable Apex Class:**

This class:

* Implements Queueable & Database.AllowsCallouts.
* Accepts Case details (CaseNumber, Priority, Case Id, Status).
* Constructs a Slack JSON payload.
* Sends the callout to the Slack Webhook.
* Handles errors gracefully.

**Apex Integration Code:** SlackNotificationQueueable

public class SlackNotificationQueueable implements Queueable, Database.AllowsCallouts {

private String caseNumber;

private String priority;

private Id caseId;

private String status;

public SlackNotificationQueueable(String caseNumber, String priority, Id caseId, String status) {

this.caseNumber = caseNumber;

this.priority = priority;

this.caseId = caseId;

this.status = status;

}

public void execute(QueueableContext context) {

if (caseId == null) {

System.debug('Error: caseId is null.');

return;

}

Http http = new Http();

HttpRequest request = new HttpRequest();

request.setEndpoint('callout:Slack\_Webhook/services/T091U7KGGDT/B091UA8BDRT/WfbzX8hm7rQiWblZtOVn6mOJ');

request.setMethod('POST');

request.setHeader('Content-Type', 'application/json');

String instanceUrl = System.Url.getOrgDomainUrl().toExternalForm();

String caseUrl = instanceUrl + '/lightning/r/Case/' + caseId + '/view';

String subjectText = (this.status == 'New') ? 'New Case Created' : 'New High Priority Case';

String plainTextMessage = subjectText + '\n' + 'Case Number: ' + caseNumber + '\n' + 'Priority: ' + priority + '\n' + 'Link: ' + caseUrl;

String messageBody = JSON.serialize(

new Map<String, Object>{

'blocks' => new List<Object>{

new Map<String,Object>{

'type' => 'header',

'text' => new Map<String,Object>{

'type' => 'plain\_text',

'text' => subjectText

}

},

new Map<String, Object>{

'type' => 'section',

'text' => new Map<String, Object>{

'type' => 'mrkdwn',

'text' => '\*Case Number:\* ' + caseNumber + '\n' +

'\*Priority:\* ' + priority + '\n' +

'<' + caseUrl + '|View Case>'

}

}

}

}

);

request.setBody(messageBody);

try {

HttpResponse response = http.send(request);

if(response.getStatusCode() == 200){

System.debug('Success! Slack response: ' + response.getBody());

String resStatus = 'Success';

insertLog(caseId, plainTextMessage, resStatus);

}else{

System.debug('Failed');

}

} catch (Exception ex) {

System.debug('Slack callout failed: ' + ex.getMessage());

String resStatus = 'Failure';

insertLog(caseId, plainTextMessage, resStatus);

}

}

private static void insertLog(Id caseId, String plainTextMessage, String status) {

insert new Slack\_Message\_Log\_\_c(

Case\_\_c = caseId,

Message\_Sent\_\_c = plainTextMessage,

Status\_\_c = status,

Timestamp\_\_c = System.now()

);

}

}

**4. Trigger / Entry Point**

You can invoke this Queueable from:

* Case Triggers (after insert or after update)

**Apex Trigger code:**

trigger CaseCreatedSendSlackNotification on Case (after insert, after update) {

for (Case c : Trigger.new) {

Case oldCase = Trigger.isUpdate ? Trigger.oldMap.get(c.Id) : null;

if (Trigger.isInsert) {

System.enqueueJob(new SlackNotificationQueueable(c.CaseNumber, c.Priority, c.Id, c.Status));

}

if (Trigger.isUpdate && c.Priority == 'High' && oldCase.Priority != 'High') {

System.enqueueJob(new SlackNotificationQueueable(c.CaseNumber, c.Priority, c.Id, c.Status));

}

}

}

**5. Logging (Audit Trail)**

We created a **Custom Object** Slack\_Message\_Log\_\_c to record each notification for traceability:

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| Case\_\_c | Lookup(Case) | Related Case |
| Message\_Sent\_\_c | Long Text | JSON payload sent to Slack |
| Status\_\_c | Picklist | Success / Failure |
| Timestamp\_\_c | DateTime | Time message was sent |

Insert a log record after each callout:

insert new Slack\_Message\_Log\_\_c(

Case\_\_c = caseId,

Message\_Sent\_\_c = messageBody.abbreviate(32768),

Status\_\_c = status,

Timestamp\_\_c = System.now()

);

**Error Handling & Debugging**

* Errors are caught in a try/catch block.
* Logged in Debug Logs (System.debug)
* Failures also logged in Slack\_Message\_Log\_\_c object

**Testing & Validation**

* Test with a new Case (New status).
* Test with updated Case (High Priority).
* Check Slack channel for messages.
* Review Slack\_Message\_Log\_\_c records.