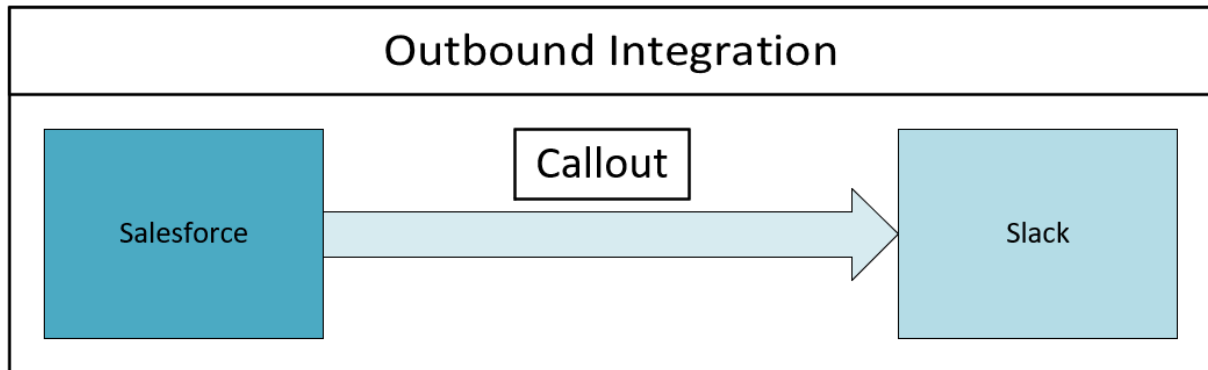


Salesforce Apex: Outbound Integration with Slack Webhook

Objective: Integrate Salesforce with Slack using webhooks and Lightning Web Components (LWC) for seamless message sending.

Overview

Outbound integration refers to Salesforce initiating communication with an external system. In this project, Salesforce makes an API callout to interact with Slack.



This diagram represents a one-way integration where Salesforce sends data or notifications to Slack, using a webhook.

In the context of Salesforce Apex and outbound integration, this suggests that Apex code in Salesforce is making an HTTP callout to a Slack webhook URL to post messages or updates directly into a Slack channel or to a Slack user. The term "Callout" refers to the outbound HTTP request made from Salesforce to an external system, in this case, Slack's API endpoint for webhooks.

This type of integration allows Salesforce to automatically notify Slack users or channels about important events, updates, or information stored in Salesforce, enhancing real-time communication and workflow automation between the two platforms.

Learning Objectives

1. Configuring a Slack webhook.
2. Setting up Remote Site Settings.
3. Writing an Apex class to make a call to the Slack webhook.

4. Creating a Lightning Web Component (LWC).
5. Using Apex to make webhook callouts in LWC.
6. Displaying toast notifications in LWC.
7. Deploying the LWC.

Project Motivation

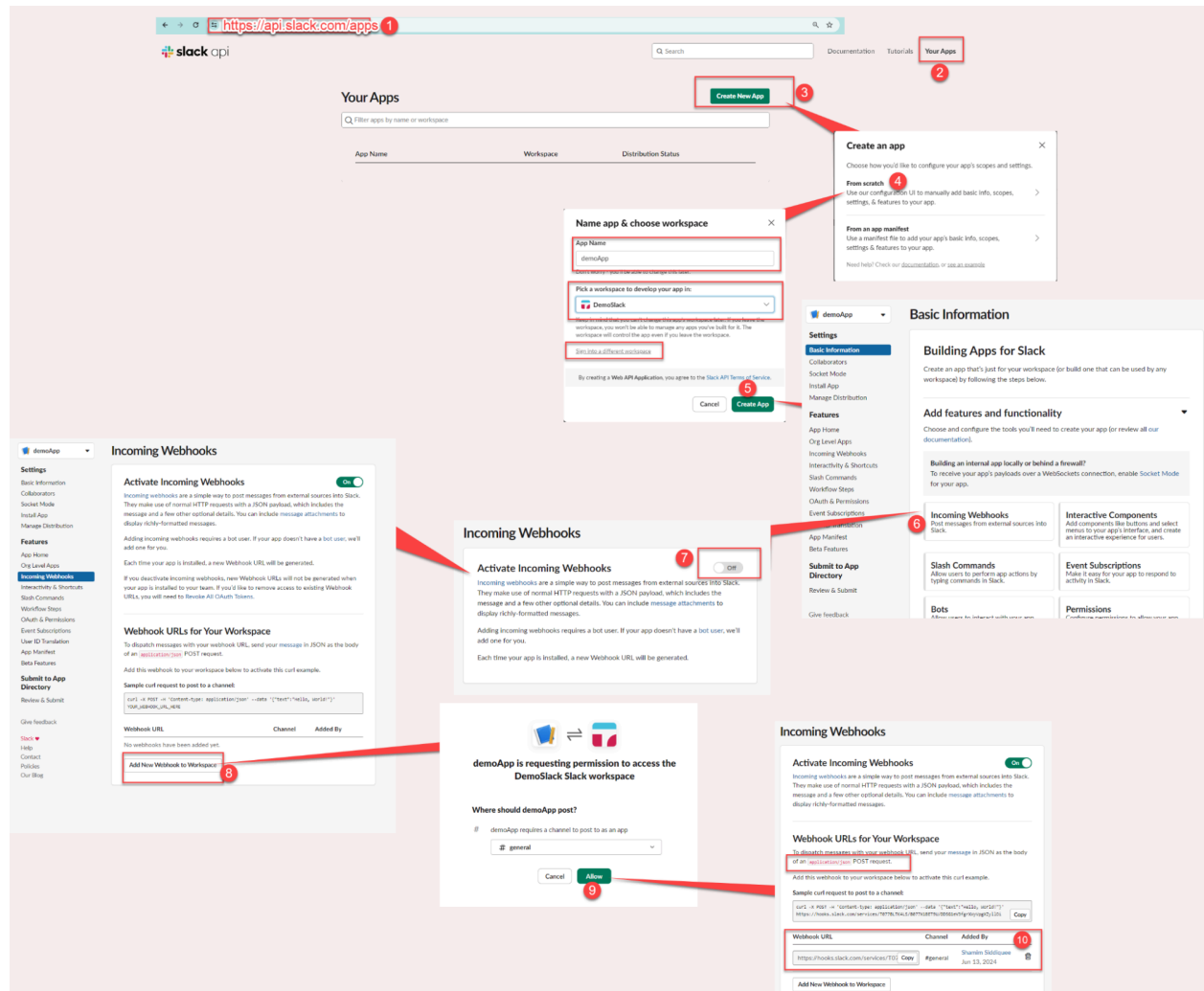
The goal is to connect Slack with Lightning Web Components (LWC) for sending messages. Although there are various methods using Apex code or JavaScript, a clear step-by-step guide on setting up a Slack app with webhooks and integrating it with LWC was needed.

Scenario

A business requirement necessitates sending messages from Salesforce to Slack channels. The developer aims to achieve this using LWC and webhooks.

Steps to Achieve the Integration

1. Configuring a Slack Webhook



This image shows the process of creating a new Slack app and setting up incoming webhooks. Here are the steps illustrated in the image:

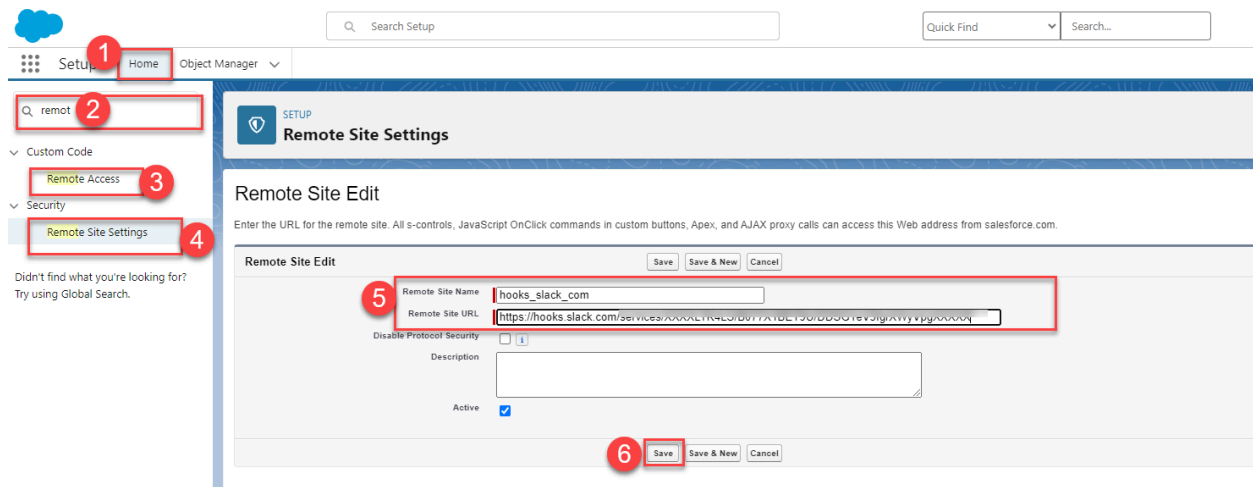
1. Navigate to [Slack API](https://api.slack.com/apps).
2. Click on "Your Apps" in the top right corner.
3. Click the "Create New App" button.
4. Choose "From scratch" to create a new app.
5. Enter an App Name (e.g., "demoApp") and select a workspace (e.g., "DemoSlack").
6. Click "Create App" to confirm.
7. In the app settings, navigate to "Incoming Webhooks" under the "Features" section.
8. Activate Incoming Webhooks by toggling the switch.
9. Click "Add New Webhook to Workspace".
10. Select a channel for the webhook to post to (e.g., "#general").
11. Click "Allow" to give the app permission.
12. Copy the generated Webhook URL for use in your integration.

The image also shows various sections of the Slack API dashboard, including:

- Basic Information about the app.
- Features list, including Incoming Webhooks, Slash Commands, Bots, and Permissions.
- A description of Incoming Webhooks and their purpose.
- Webhook URLs for the workspace.
- A sample curl request to post to a channel.

This process sets up a Slack app that can receive incoming messages from external sources (like Salesforce) via webhooks, allowing for the integration described in the previous diagram.

2. Setting up Remote Site Settings



This image shows the process of setting up a Remote Site in Salesforce to allow outbound calls to Slack. Here are the steps illustrated:

1. Navigate to Setup in Salesforce.
2. In the Quick Find box, type "remot" to search for remote site settings.
3. Under Custom Code, click on "Remote Access".
4. Select "Remote Site Settings".
5. In the Remote Site Edit section:
 - Enter a Remote Site Name (e.g., "hooks_slack_com").
 - Enter the Remote Site URL (the Slack webhook URL obtained earlier).
6. Click "Save" to add the new remote site.

This configuration allows Salesforce to make callouts to the specified Slack webhook URL, which is necessary for the outbound integration we discussed earlier. By adding this remote site, you're explicitly allowing Salesforce to send HTTP requests to Slack's domain, which is a security measure to prevent unauthorized external communications.

3. Writing an Apex Class to Call the Slack Webhook

Here is the Apex code to make a call to the Slack webhook:

```

private static final String SLACK_WEBHOOK =
'https://hooks.slack.com/services/T077BLTK4LS/B077X1BET9U/DDSG1eV3fgrXWyVp
gXZy1lOi';

//SlackFileUploadController2.webhookPublishMessageToSlack('Hello
Slack!');

@AuraEnabled
public static String webhookPublishMessageToSlack(String msg) {
    String json_string = prepareJSON(msg).getAsString();

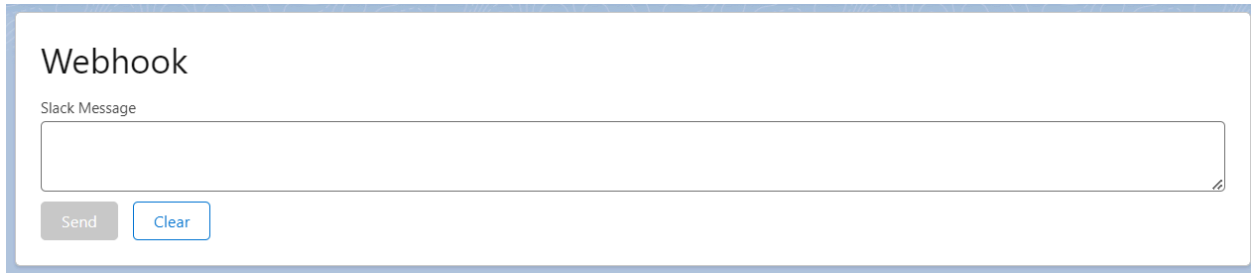
    // API callout
    HttpRequest request = new HttpRequest();
    request.setEndpoint(SLACK_WEBHOOK);
    request.setMethod('POST');
    request.setBody(json_string);

    Http http = new Http();
    try {
        HttpResponse response = http.send(request);

        return JSON.serialize(new Map<String, String>{'status' =>
response.getStatus(), 'message' => 'Message uploaded to Slack
successfully!'});
    } catch (Exception ex) {
        System.debug('Callout exception-->' + ex.getMessage());
        return JSON.serialize(new Map<String, String>{'status' =>
'400', 'message' => ex.getMessage()});
    }
}

```

4. Creating a Lightning Web Component (LWC)



Webhook

Slack Message

Send Clear

This image shows a simple user interface for sending a webhook message to Slack. Here are the key elements:

- The title "Webhook" at the top of the panel.
- A text area labeled "Slack Message" where the user can enter the message they want to send to Slack.
- Two buttons below the text area:
 - A "Send" button, which is likely used to submit the message to Slack via the webhook.
 - A "Clear" button, which probably clears the text in the message field.

This interface appears to be part of a larger application or system that integrates with Slack. It provides a straightforward way for users to compose and send messages to a Slack channel or user through a pre-configured webhook without needing to interact with Slack directly or know the technical details of the webhook implementation.

HTML Code for lwcSlackWebhook Component:

lwcSlackWebhook Html

```
<template>
  <template if:true={processing}>
    <lightning-spinner
      alternative-text="Loading"
      size="small"
    ></lightning-spinner>
  </template>

  <form class="slds-p-around_medium slds-card" onsubmit={submitHandler}>
    <lightning-layout multiple-rows>
      <lightning-layout-item size="12"
class="slds-p-around_xx-small">
        <h2 class="slds-text-heading_large
slds-text-align_left">Webhook</h2>
      </lightning-layout-item>
```

```

        <lightning-layout-item size="12"
class="slds-p-around_xx-small">
            <lightning-textarea label="Slack Message"
name="textarea_2" value={formData.textarea_2} variant="standard"
data-id="message" onchange={changeHandler}></lightning-textarea>
        </lightning-layout-item>
        <lightning-layout-item size="12"
class="slds-p-around_xx-small">
            <lightning-button variant="brand" label="Send"
title="Send" type="submit" disabled={uploadDisabled} ></lightning-button>
            <lightning-button variant="brand-outline" label="Clear"
title="Clear" type="button" class="slds-var-m-around_small"
onclick={handleClear}></lightning-button>
        </lightning-layout-item>
    </lightning-layout>
</form>
</template>

```

JavaScript Code for LwcSlackWebhook Component:

LwcSlackWebhook JS

```

import { LightningElement, track } from 'lwc';
import publishMessage from
'@salesforce/apex/SlackFileUploadController2.WebhookPublishMessageToSlack'
;
import { ShowToastEvent } from 'lightning/platformShowToastEvent';

export default class LwcSlackWebhook extends LightningElement {
    formData = {

    };

    @track processing=false;
    @track uploadDisabled =true;

```

```

changeHandler(event) {
    const { name, value, checked, type } = event.target;
    const isChecked = type === 'checkbox' || type ===
'checkbox-button' || type === 'toggle';
    this.formData = { ...this.formData, [name]: isChecked ? checked :
value };

    this.uploadDisabled = !this.formData.textarea_2;
}

async submitHandler(event) {
    event.preventDefault();
    console.log("Form Data", JSON.stringify(this.formData));
    this.processing=true;
    let publishInfo;
    try {
        const messageResponse = await publishMessage({
msg:this.formData.textarea_2});
        publishInfo = JSON.parse(messageResponse);
        this.handleClear();
    } catch (error) {
        console.log('error-->',error);
    }

    this.processing=false;
    // Notify the user of success
    this.dispatchEvent(
        new ShowToastEvent({
            title: publishInfo.status === 'OK'? 'Success': 'Error',
            message: publishInfo.message,
            variant: publishInfo.status === 'OK'? 'success': 'error'
        })
    );
}

handleClear(event)
{
    this.template.querySelector('[data-id="message"]').value='';
    this.uploadDisabled =true;
}

```



```
}  
}
```

XML Configuration for lwcSlackWebhook Component:

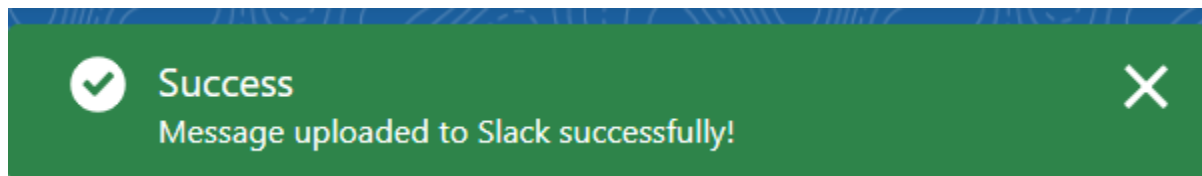
lwcSlackWebhook XML

```
<?xml version="1.0" encoding="UTF-8"?>  
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">  
  <apiVersion>59.0</apiVersion>  
  <isExposed>true</isExposed>  
  <targets>  
    <target>lightning__HomePage</target>  
  </targets>  
</LightningComponentBundle>
```

5. Using Apex to Make Webhook Callouts in LWC

The JavaScript code for the LWC handles the message submission to the Slack webhook by calling the `publishMessage` Apex method. The component also manages form data and provides feedback to the user through toast notifications.

6. Displaying Toast Notifications in LWC



Toast notifications are used to inform the user about the success or failure of their message submission. The `ShowToastEvent` is dispatched in the LWC JavaScript code based on the response from the Apex callout.

7. Deploying the LWC

The final step involves deploying the Lightning Web Component to the appropriate Salesforce environment and ensuring it functions as expected in production.

Deploy Apex and LWC to the salesforce org in the following order

Apex

SlackController
LWC
lwcSlackWebhook

By following these steps, you can successfully integrate Salesforce with Slack using webhooks and Lightning Web Components, enhancing communication and workflow automation between the two platforms.

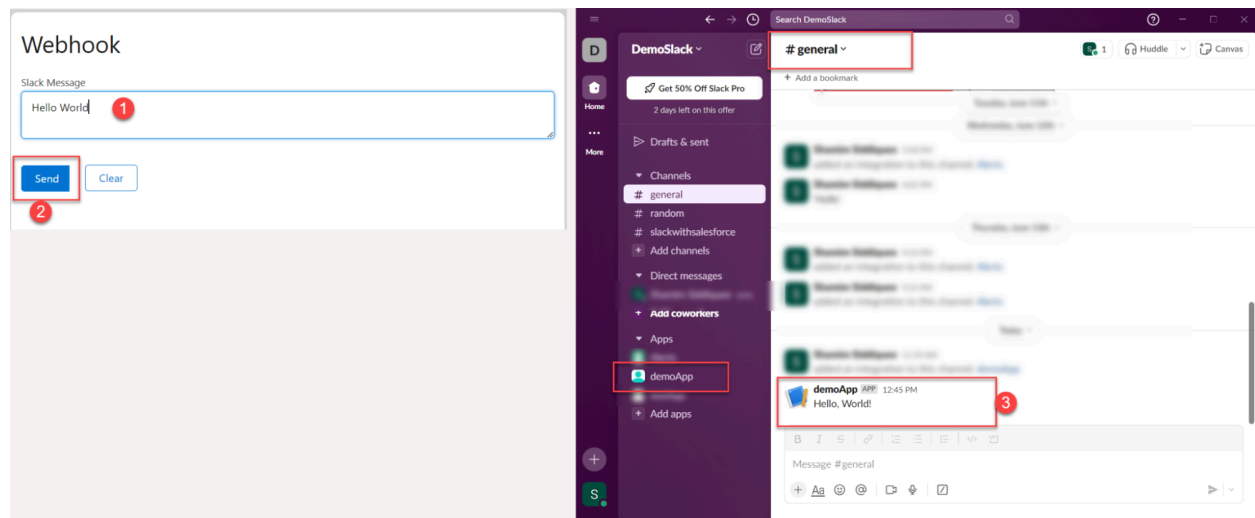
8. Final Output

To integrate the LWC component with Slack:

1. Place the LWC on your Salesforce home page.
2. In the "Webhook" section, enter your desired message in the text field labeled "Slack Message".
3. Click the "Send" button to transmit the message.

Upon successful transmission, your message will appear in the "#general" Slack channel of your workspace. The message will be posted by the "demoApp" integration, as shown in the image.

This setup allows for seamless communication between your Salesforce instance and Slack, enabling real-time notifications and updates to be shared with your team directly from the Salesforce platform.



This image demonstrates the process of sending a message from a webhook interface to Slack and its successful delivery. Here's what's happening:

- In the webhook interface on the left, "Hello World" is typed into the Slack Message text area.
- The "Send" button is highlighted, indicating it's about to be or has been clicked to send the message.

- On the right side, we see the Slack interface for a workspace called "DemoSlack". In the #general channel, we can see the message "Hello World!" has been received and posted by an app named "demoApp".

This showcases the full cycle of the integration:

- Message composition in the webhook interface (likely part of Salesforce or another external system).
- Sending the message through the configured webhook.
- Receipt and display of the message in the specified Slack channel.

The presence of "demoApp" in the Slack sidebar under "Apps" confirms that the integration has been set up correctly and is functioning as intended.