SharePoint WebHooks

KOMMU, ANIL KUMAR

04-Sep-2017

What is SharePoint WebHooks?

A Webhook is "**a way to be notified of a done change**". SharePoint WebHooks enable developers to build applications that subscribe to receive notifications on specific events that occur in SharePoint. When an event is triggered, SharePoint sends an HTTP POST payload to the subscriber. Webhooks are easier to develop and consume than Windows Communication Foundation (WCF) services used by SharePoint add-in remote event receivers. This is because webhooks are regular HTTP services (web API).

Currently webhooks are only enabled for SharePoint list/library items. SharePoint list item webhooks cover the events corresponding to list item changes for a given SharePoint list or a document library. SharePoint webhooks provide a simple notification pipeline so your application can be aware of changes to a SharePoint list without polling the service.

Web hooks are not specific to Microsoft. They are a universal web standard that's also being adopted by other vendors (e.g., WordPress, GitHub, MailChimp, and others).

Why not Remote Event Receivers?

- WebHooks have a **retry mechanism** with an incremental back-off strategy
- WebHooks are **lightweight** for building service endpoints
  - The payload is small
  - Notifications are batched in the response to the GetChanges() call
- WebHooks are secure, no event information is passed in the notification
- WebHooks are easier to consume by non-SharePoint developers
  - Office Devs have an opportunity to learn a new standard
- No WCF endpoints; regular HTTPS services are sufficient

Registering / creating webhooks

To create a new SharePoint webhook, you add a new subscription to the specific SharePoint resource, such as a SharePoint list.
The following information is required for creating a new subscription:

- **Resource** - The resource endpoint URL you are creating the subscription for. For example a SharePoint List API URL.
- **Server notification URL** - Your service endpoint URL. SharePoint will send an HTTP POST to this endpoint when events occur in the specified resource.
- **Expiration date** - The expiration date for your subscription. You need to set an expiration date when creating the subscription. The expiration date should be less than 6 months. Webhook subscriptions are set to expire after 6 months by default if an expiration Date value is not specified. Your application is expected to handle the expiration date according to your application's needs by updating the subscription periodically.

You can also include the following additional information if needed:

- **Client State** - An opaque string passed back to the client on all notifications. You can use this for validating notifications, tagging different subscriptions, or other reasons.

Handling webhook validation requests

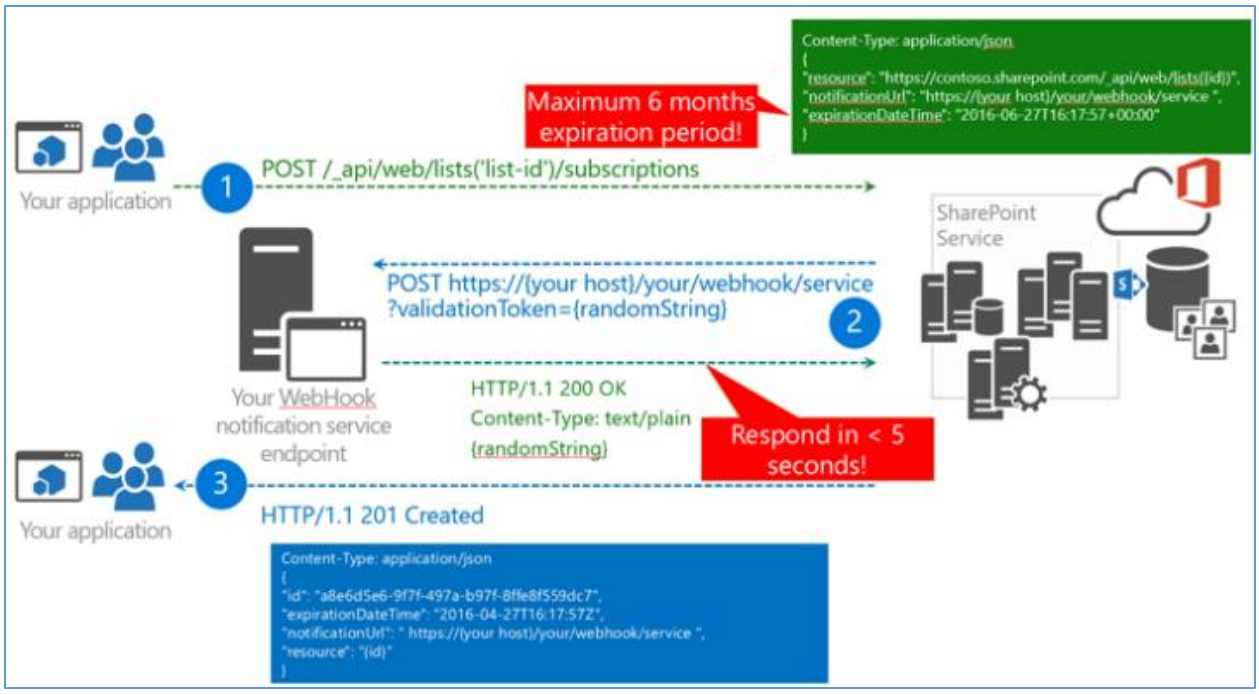
When a new subscription is created, SharePoint will validate whether the notification URL supports receiving webhook notifications. This validation is performed during the subscription creation request. The subscription will only be created if your service responds in a timely manner back with the validation token.

Example validation request

To add a webhook to a SharePoint list, your application first creates a webhook subscription by sending a POST /_api/web/lists('list-id')/subscriptions request. The request includes the following items:

- A payload that identifies the list which you're adding the webhook for.
- The location of your webhook service URL to send the notifications.
- The expiration date of the webhook.

After you've requested SharePoint to add your webhook, SharePoint will validate that your webhook service endpoint exists. It sends a validation string to your service endpoint. SharePoint expects that your service endpoint returns the validation string within 5 seconds. If this process fails then the webhook creation is canceled. If you've deployed your service then this will work and SharePoint returns an HTTP 201 message on the POST request the application initially sent. The payload in the response contains the ID of the new webhook subscription.



POST https://xxxxxx.azurewebsites.net/your/webhook/service?validationToken={randomString}
Content-Length: 0

Response

For the subscription to be created successfully, your service must respond to the request by returning the value of the **validationToken** query string parameter as a plain-text response.

```
HTTP/1.1 200 OK
Content-Type: text/plain
{randomString}
```

If your application returns a status code other than 200 or fails to respond with the value of the **validationToken**parameter, the request to create a new subscription will fail.

Receiving notifications

The notification payload will inform your application that an event occurred in a given resource for a given subscription. Multiple notifications to your application may be batched together into a single request, if multiple changes occurred in the resource within the same time period.
The notification payload body will also contain your client state if you used it when creating the subscription.

Webhook notification resource

The notification resource defines the shape of the data provided to your service when a SharePoint webhook notification request is submitted to your registered notification URL.

JSON representation

Each notification generated by the service is serialized into a **webhookNotifiation** instance:


```
{
  "subscriptionId":"91779246-afe9-4525-b122-6c199ae89211",
  "clientState":"00000000-0000-0000-0000-000000000000",
  "expirationDateTime":"2016-04-30T17:27:00.0000000Z",
  "resource":"b9f6f714-9df8-470b-b22e-653855e1c181",
  "tenantId":"00000000-0000-0000-0000-000000000000",
  "siteUrl":"/",
  "webId":"dbc5a806-e4d4-46e5-951c-6344d70b62fa"
}
```


Since multiple notifications may be submitted to your service in a single request, these are combined together in an object with a single array **value**:


```
{
  "value":[
    {
      "subscriptionId":"91779246-afe9-4525-b122-6c199ae89211",
      "clientState":"00000000-0000-0000-0000-000000000000",
      "expirationDateTime":"2016-04-30T17:27:00.0000000Z",
      "resource":"b9f6f714-9df8-470b-b22e-653855e1c181",
      "tenantId":"00000000-0000-0000-0000-000000000000",
      "siteUrl":"/",
```

```
      "webId":"dbc5a806-e4d4-46e5-951c-6344d70b62fa"
    }
  ]
}
```

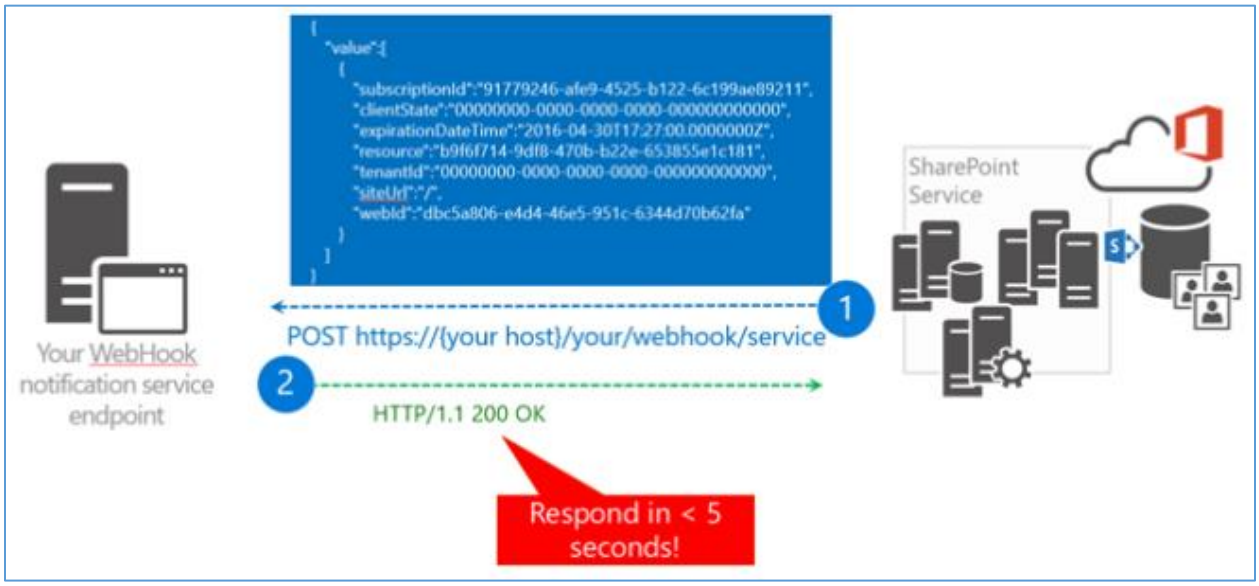
SharePoint calls out to your webhook service

When SharePoint detects a change in a list for which you've created a webhook subscription, your service endpoint will be called by SharePoint. When you look at the payload from SharePoint, notice that the following properties are important:

Property	Description
subscriptionId	The ID of the webhook subscription. If you want to update the webhook subscription, for example you prolong the webhook expiration, then you need this ID.
resource	The ID of the list for which the change happened.
siteUrl	The server relative URL of the site holding the resource for which the change happened.

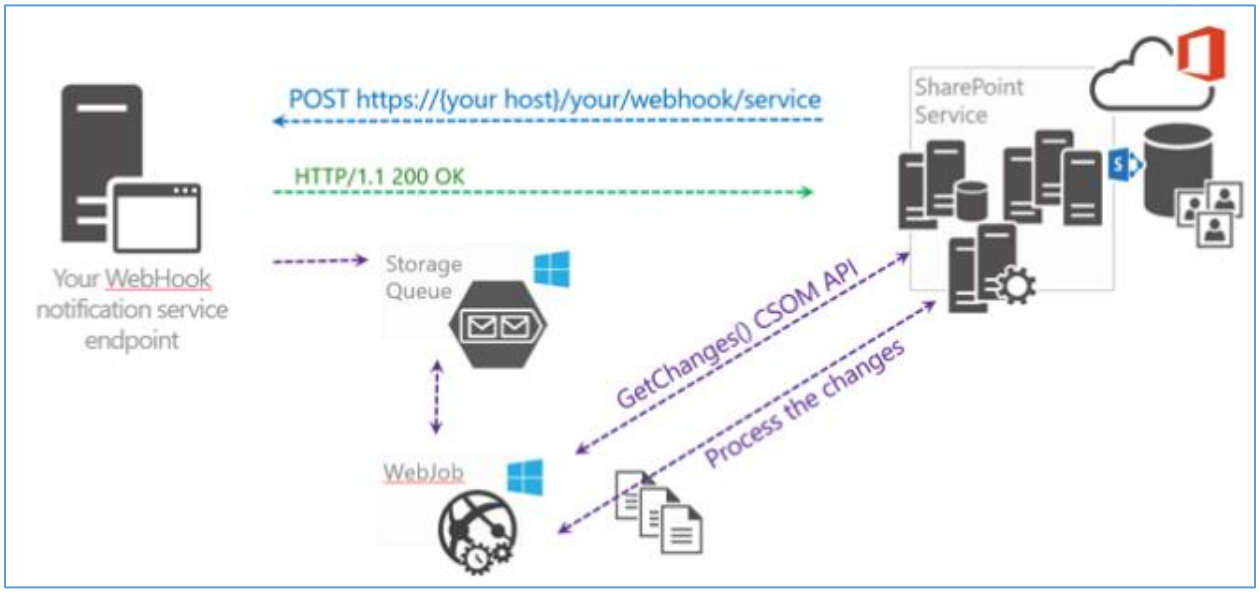
Note: SharePoint only sends a notification that a change happened, but the notification does not include what actually changed. Because you get information about the web and list that were changed, this means that you can use the same service endpoint to handle webhook events from multiple sites and lists.

When your service is called, it's important that your service replies with an HTTP 200 message in under 5 seconds. Later in this article you'll learn more about the response time, but essentially this requires that you **asynchronously** handle the notifications. In this reference implementation you'll do this by using Azure Web Jobs and Azure Storage Queues.



Processing a notification event

In the previous step your service endpoint was called but SharePoint only provided information about where the change happened, not what was actually changed. To understand what was changed, you'll need to use the SharePoint GetChanges() API, as shown in the following image.



To avoid getting the same change repeatedly, it's important that you inform SharePoint from which point you want the changes. This is done by passing a **changeToken**, which also implies that your service endpoint needs to persist the last used **changeToken** so that it can be used the next time the service endpoint is called.

The following are some key things to note about changes:

- SharePoint does not call your service in real-time: when a change happens on a list that has a webhook, SharePoint will queue a webhook call out. Once each minute this queue will be read and the appropriate service endpoints are called. This batching of requests is important. For example, if a bulk upload of 1000 records occurred at once, batching prevents SharePoint from calling your endpoint 1000 times. So your endpoint is only called once but when you call the GetChanges() method you'll get 1000 change events that you need to process.
- To guarantee an immediate response, regardless of the number of changes there, it's important that the workload of your service endpoint runs asynchronously. In the reference implementation we leveraged the power of Azure: the service will serialize the incoming payload and store it in an Azure Storage queue while there's an Azure web job that runs continuously and checks for messages in the queue. When there are messages in the queue the web job will process them and also execute your logic asynchronously.

Subscription Properties

Property Name	Type	Description
resource	String	Unique identifier of the list where the subscription is registered.
subscriptionId	String	The unique identifier for the subscription resource
clientState	String - optional	An optional string value that is passed back in the notification message for this subscription.
expirationDateTime	DateTime	The date and time when the subscription will expire if not updated or renewed.
tenantId	String	Unique identifier for the tenant which generated this notification.
siteUrl	String	Server relative URL of the site where the subscription is registered.
webId	String	Unique identifier of the web where the subscription is registered.

Example notification

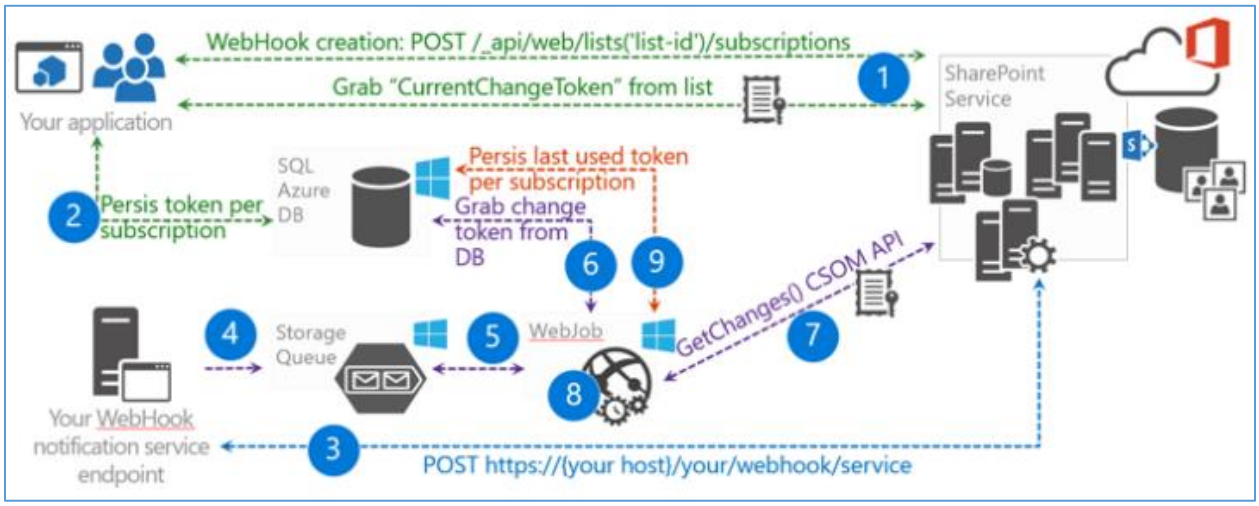
The body of the HTTP request to your service notification URL will contain a webhook notification. The following example shows a payload with one notification:


```
{
  "value":[
    {
      "subscriptionId":"91779246-afe9-4525-b122-6c199ae89211",
      "clientState":"00000000-0000-0000-0000-000000000000",
      "expirationDateTime":"2016-04-30T17:27:00.0000000Z",
      "resource":"b9f6f714-9df8-470b-b22e-653855e1c181",
      "tenantId":"00000000-0000-0000-0000-000000000000",
      "siteUrl":"/",
      "webId":"dbc5a806-e4d4-46e5-951c-6344d70b62fa"
    }
  ]
}
```


The notification doesn't include any information about the changes that triggered it. Your application is expected to use the GetChanges API on the list to query the collection of changes from the change log and store the change token value for any subsequent calls when the application is notified.

Complete end-to-end flow

The following diagram describes the complete end-to-end webhook flow:



1. Your application creates a webhook subscription. When it does it gets the current **changeToken** from the list it created the webhook for.
2. Your application persists the **changeToken** in a persistent storage, such as SQL Azure in this case.
3. A change in SharePoint occurs and SharePoint calls your service endpoint.
4. Your service endpoint serializes the notification request and stores it in a storage queue.
5. Your web job sees the message in the queue and starts your message processing logic.
6. Your message processing logic retrieves the last used change token from the persistent storage.
7. Your message processing logic uses the GetChanges() API to determine what changed.
8. The returned changes are processed and now your application performs what it needs to do based on the changes.
9. Finally the application persists the last retrieved **changeToken** so that next time it does not receive changes that were already processed.

Event types

SharePoint webhooks only support asynchronous events. This means that webhooks are only fired after a change happened (similar to **-ed** events), and whereas synchronous events (**-ing** events) are not supported in webhooks.

Notifications will be sent to your application for the following asynchronous list item events in SharePoint:

- ItemAdded
- ItemUpdated
- ItemDeleted
- ItemCheckedOut
- ItemCheckedIn
- ItemUncheckedOut

- ItemAttachmentAdded
- ItemAttachmentDeleted
- ItemFileMoved
- ItemVersionDeleted
- ItemFileConverted

Error handling

If an error occurs while sending the notification to your application, SharePoint will follow exponential back-off logic. Any response with an HTTP status code outside of the 200-299 range, or that times out, will be attempted again over the next several minutes. If the request is not successful after 15 minutes, the notification is dropped.
Future notifications will still be attempted to your application, although the service may remove the subscription if a sufficient number of failures are detected.

Retry mechanism

If an event occurs in SharePoint and your service endpoint is not reachable (e.g., due to maintenance) SharePoint will retry sending the notification. SharePoint performs a retry of **5 times with a 5 minute wait time** between the attempts. If for some reason your service is down for a longer time, the next time when it's up and gets called by SharePoint, the call to the **GetChanges**() will recover the changes that were missed when your service was not available.

How to work with webhook renewal

By default Webhook subscriptions are set to expire max by 6 months or at the specified date when they are created. Often you need the webhook to be available for a longer time. The patterns described below are good for increasing the lifetime of a webhook subscription. The first pattern is lightweight and the second one is slightly more complex and requires an additional web job to be hosted.

Basic model

When your service receives a notification it also gets information about the subscription lifetime. If the subscription is about to expire, inside your notification processing logic you simply extend the lifetime of the subscription. This model is implemented in this reference implementation and works fine for most cases. However, in a case where there's no change for 6 months on the list you've created a webhook subscription for, the webhook subscription is never prolonged and will be deleted.

Reliable but more complex model

Create a web job that on a weekly basis reads all the subscription IDs from the persistent storage. One-by-one extend the found subscriptions each time.

Required permissions to update WebHooks

Microsoft Azure Active Directory (AD) applications

- Set the following Azure AD application permissions:

Application	Permission
Office 365 SharePoint Online	Read and write items and lists in all site collections.

SharePoint add-in

- A subscription can only be updated by the SharePoint Add-in that created it.
- Set the following SharePoint Add-in permissions (or higher):

Scope	Permission Rights
List	Manage

Reference Links

Reference Implementation
https://dev.office.com/sharepoint/docs/apis/webhooks/webhooks-reference-implementation

Sample Code Location:
https://github.com/SharePoint/sp-dev-samples

Sample Code Deployment Guide
https://github.com/SharePoint/sp-dev-samples/blob/master/Samples/WebHooks.List/Deployment%20guide.md

Walkthrough Video
https://www.youtube.com/watch?v=P4a1_EWokwM