OKLAHOMA STATE
UNIVERSITY

A report on

AWS Prescriptive Guidance Library – Slack Notifications

Achyutha Santhoshi

Cloud Acceleration Team, Amazon

Date: 6th December 2022

Introduction

Cloud acceleration team (CAT) is an AWS based product team. It consists of 3 different products; AWS prescriptive guidance library[1] is one among them. The APG library consists of different patterns, strategies and guides for their product and for their partners.

AWS Prescriptive Guidance Library[1] is a product of Amazon Web Services. APG Library provides patterns, guides, strategies which help customers to migrate[2] from one cloud platform to another cloud platform. For an example, if a customer wants to migrate from AZURE web services to AMAZON WEB SERVICES

At present, APG Library[1] workflow notifications are sent through E-mail. Though E-mail is the primary contact for APG users, there is a possibility that they miss important time bound notifications due to multiple emails waiting for their attention in their inbox. We want to introduce a new feature which pushes notifications through slack messages to make the process faster.

Goals and Requirements

The main goal of the project is to integrate with slack using a new slack client lambda. New Slack Sandbox[1] will be created for testing these scenarios, New Slack app "APGL-Notifications" will be created within the sandbox. The new "MPC Slack Client" lambda will be created as part of MPC Service[5] which handles all the slack notifications. Existing "MPC Service Request Handler", "MPC Event Handler", "MPC Content Bounty Processor" methods will be updated to call Slack Client. The new slack client uses secrets manager to retrieve the token used for calling the slack application. Publish informational messages, errors to the CloudWatch logs. New "MPC Slack Client" publishes metrics related to channels creation, deletion and number of notifications sent. User notification preferences are obtained from Dynamo DB which are manually inserted. Different API scopes are tested and tried out to find the required ones to use in the project.

Project Approach

To create a new “MPC Slack Client”, I have evaluated the following approaches

1. Library class
    a. Using library class from existing MPC request handler, etc lambdas.
    b. Pros:
        i. It is simpler to implement.
    c. Cons:
        i. Since the slack client is invoked within the existing request handler lambda, it will affect request processing times
        ii. Any kind of error thrown in the slack client will result in failure of the request
2. Synchronous lambda function
    a. Request handler lambda invokes slack client lambda synchronously and waits for response
    b. Pros:
        i. Manage and deploy changes independently
        ii. Easier to scale by having a separate lambda
    c. Cons:
        i. Since the request processing lambda is waiting for this lambda to complete, it impacts request processing time
        ii. It's not cost effective, both lambda functions will be running for the duration of notification processing
        iii. If the lambda function fails, retries are the responsibility of calling lambda
3. Asynchronous lambda function
    a. We decided to go with Asynchronous lambda function as it does not impact request processing
    b. The lambda that invokes the slack client lambda does not wait for the lambda to finish and immediately returns.
    c. Pros:
        i. It is cost effective as only one lambda is in process at a time

    ii. It does not impact request processing
    iii. If the lambda function fails, retries are the responsibility of the lambda framework
    iv. Manage and deploy changes independently
    v. Easier to scale by having a separate lambda

d. Cons:
    i. It can be unstable and unpredictable because some asynchronous invocations may get canceled and some might even run multiple times (double runs).
    ii. Some function executions would get delayed, which is acceptable in our case

<u>Approaches for Slack API</u>

1. Webhooks: A webhook needs to be setup in slack workspace manually and then the hook URL is used to post request to slack. Limitation: In webhooks, we need to provide the channel name to post messages. We can provide channel name for the public channel. Since the private channels are intended to be created when a context is created, we can't use this approach for private channels
2. Slack Bot: Implemented with Node.js and uses events api to post messages. Events API is used for receiving notifications from slack
3. WebAPI and Conversational API[4]: Like REST APIs, The Web API is a collection of HTTP RPC-style methods. We can pass the bearer token for authentication in the request. Web API and Conversational API ideal method to implement our project. Web API consists of different scopes which can be used in method implementation

   The below API methods will be used from slack

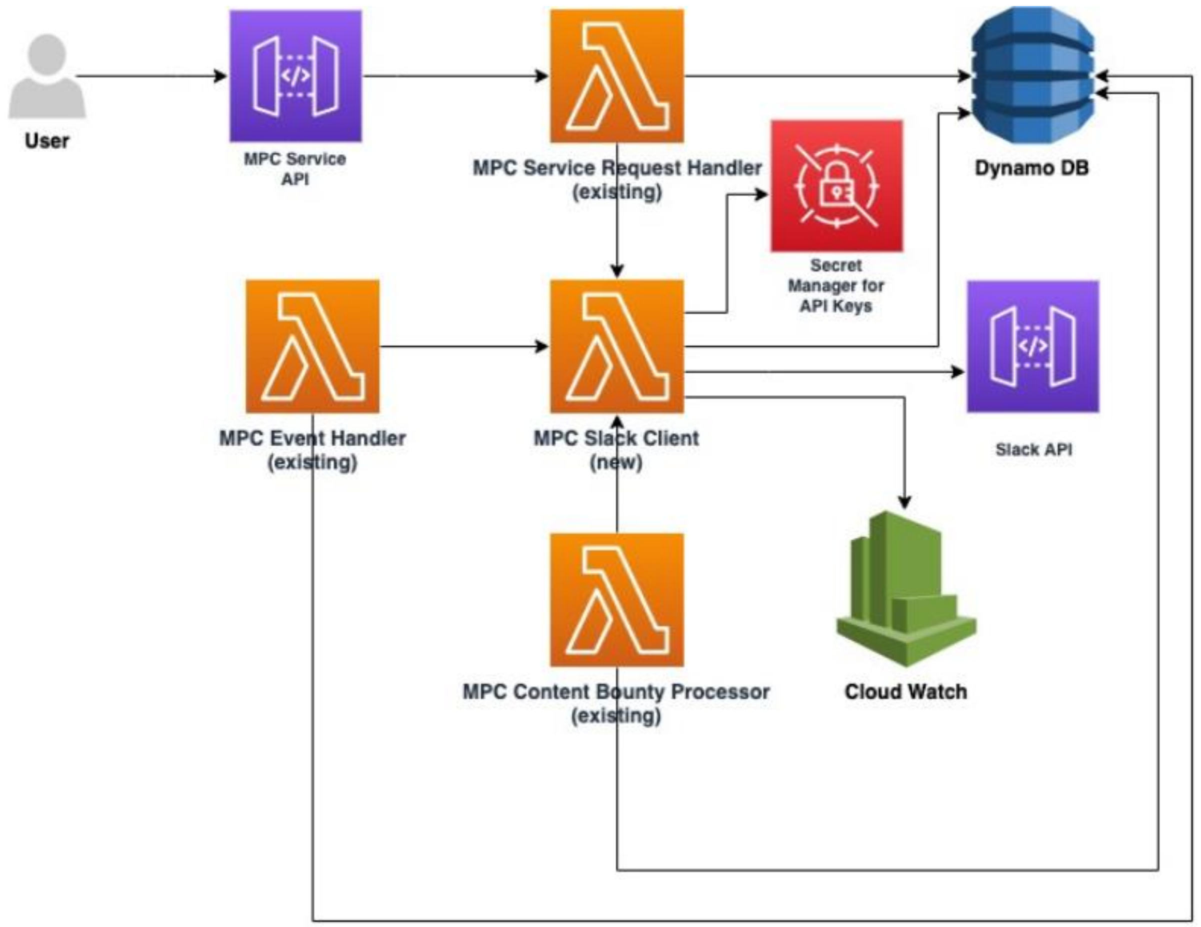
   - conversations.create - Initiates a public or private channel-based conversation
   - conversations.invite - Invites users to a channel.
   - conversations.kick - Removes a user from a conversation.
   - conversations..archive - Archives a conversation
   - chat.postMessage - Post messages in approved channels and conversations
   - user.lookupByEmail - Find a user with an email address.

<u>MPC Slack Client Lambda (asynchronous)</u>

The new “MPC Slack Client” lambda includes these methods

- A method is created to create a channel and invite members by passing the channel name. This method allows us to create a channel and invite the required members to join the channel. If channel exists, it checks and invites members if not already on the channel.
    - createChannel (channelName, members?)
- A method is created to post to a public channel by passing channel name, message.
    - notifyPublicChannel(channelName, message)
- A method is created to post to a private channel by passing channel name, message. This method creates a channel if it doesn’t exist and optionally adds members.
    - notifyPrivateChannel (channelName, message, members?)
- A method is created to add members to existing private channel: This method allows us to add members to the existing channel
    - changeChannelMembers(channelName, members)
- A method is created to delete members from existing private channel: This method allows to delete members from the existing channel
    - changeChannelMembers(channelName, members)
- A method is created to check whether the channel exists
    - checkChannelExists(channelName)
- A method is created to delete the channel
    - DeleteChannel(channelName)
- Before adding members to slack channel, user preference is checked using UserResourceDao→getUserProfile() method

High level design architecture



User
MPC Service API
MPC Service Request Handler (existing)
Dynamo DB
Secret Manager for API Keys
MPC Event Handler (existing)
MPC Slack Client (new)
Slack API
MPC Content Bounty Processor (existing)
Cloud Watch

Fig 1: High Level Architecture for Slack Client Integration

The above architecture illustrates the integration with Slack API, and the new slack client lambda function to implement the feature. We will reuse the existing table by adding an additional record type for user slack notification preference. The existing lambda functions with the newly created lambda functions posts messages to slack. API authentication is confidential. It is not discussed in this report

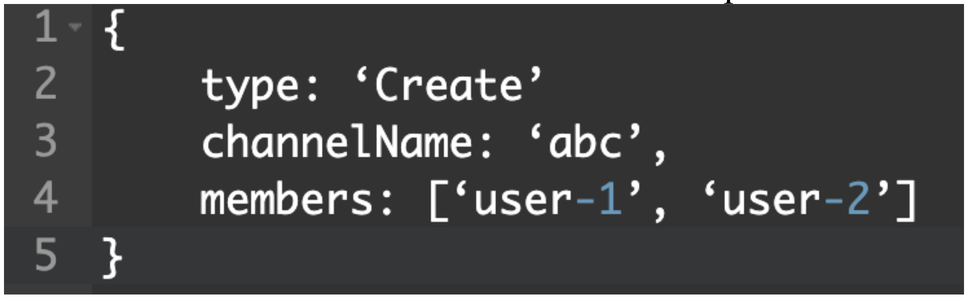
MPC service changes

The following changes will be implemented within the existing MPC Service code

- DraftContentService → createDraftContent
    - create channel
    - add author to channel
- WorkflowActionManager → content 'publish'
    - send notification to public channel
- ContentBountyService → createBounty
    - check channel exists
    - create channel and add author if it doesn't exist
    - send notification to private channel
- WorkflowActionManager → change owner
    - check channel exists
    - create channel if it doesn't exist
    - remove previous owner from channel
    - add new owner to channel
- UserResourceService → shareContents
    - check channel exists
    - create channel and add author if it doesn't exist
    - add shared user to channel
- UserProfileAttributes → add new value 'SlackNotificationPreference' to enum
- All methods log informational and error messages to cloudwatch logs
- Throws error back from lambda in case of errors so that lambda framework retries requests.

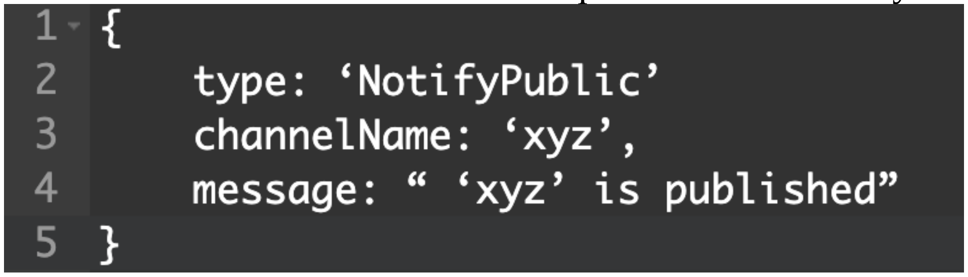
API Changes

“MPC Service Slack Client” sample create channel event



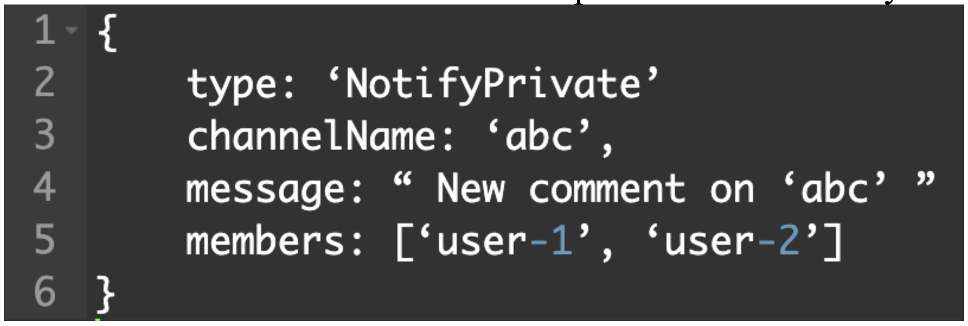
```
{
    type: ‘Create’
    channelName: ‘abc’,
    members: [‘user-1’, ‘user-2’]
}
```

“MPC Service Slack Client” sample event to notify the public channel



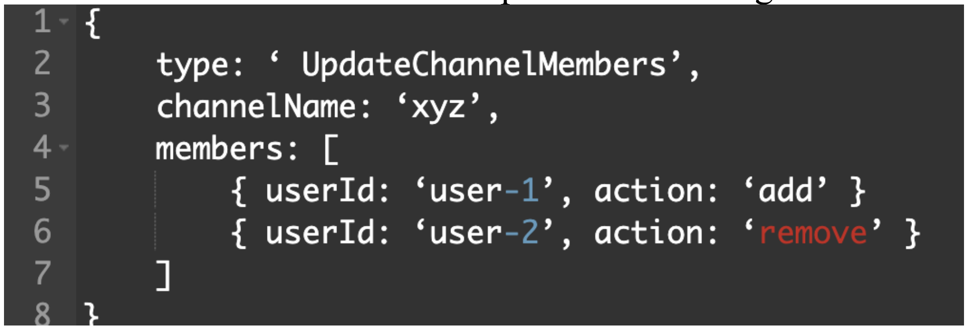
```
{
    type: ‘NotifyPublic’
    channelName: ‘xyz’,
    message: “ ‘xyz’ is published”
}
```

“MPC Service Slack Client” sample event to notify the private channel



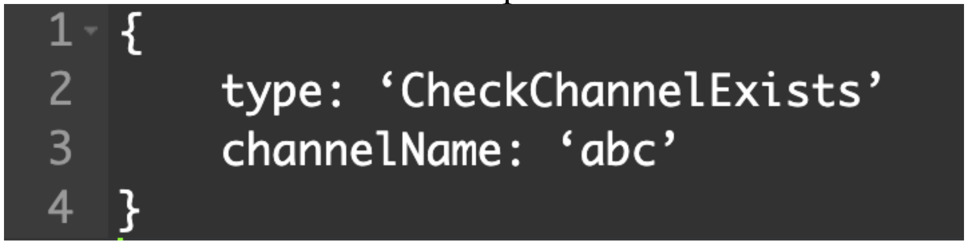
```
{
    type: ‘NotifyPrivate’
    channelName: ‘abc’,
    message: “ New comment on ‘abc’ ”
    members: [‘user-1’, ‘user-2’]
}
```

“MPC Service Slack Client” sample event to change the channel users membership



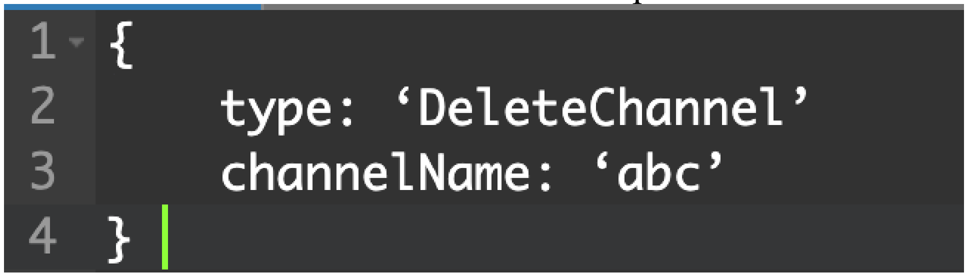
```
{
    type: ‘ UpdateChannelMembers’,
    channelName: ‘xyz’,
    members: [
        { userId: ‘user-1’, action: ‘add’ }
        { userId: ‘user-2’, action: ‘remove’ }
    ]
}
```

“MPC Service Slack Client” sample event to check whether the channel is existing



```
{
    type: ‘CheckChannelExists’
    channelName: ‘abc’
}
```

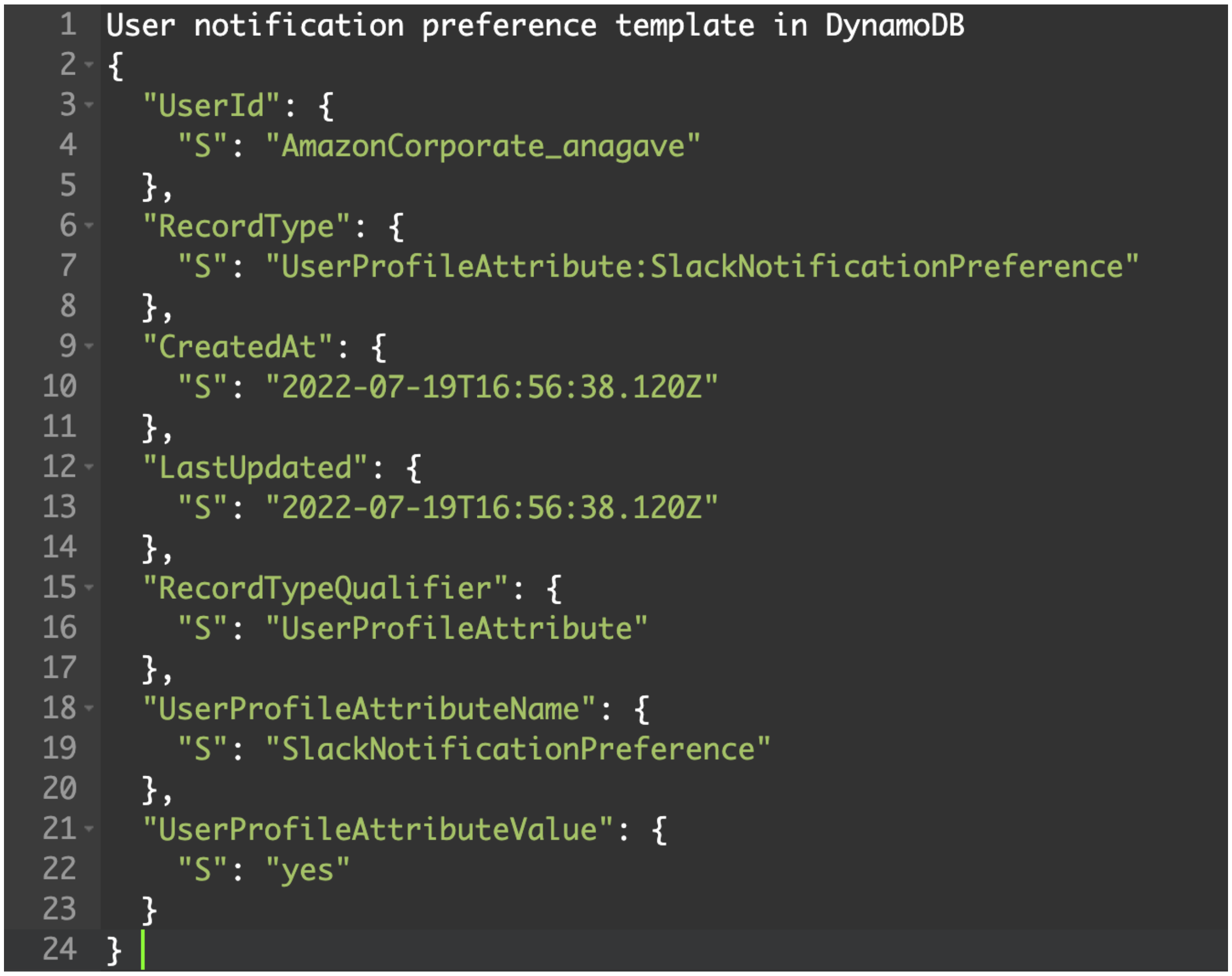
“MPC Service Slack Client” sample event to delete the channel



```
{
    type: ‘DeleteChannel’
    channelName: ‘abc’
}
```

<u>Data Model</u>

Table name: mpc-user-resources
We will reuse existing table by adding an additional record type for user slack notification preference



```
User notification preference template in DynamoDB
{
  "UserId": {
    "S": "AmazonCorporate_anagave"
  },
  "RecordType": {
    "S": "UserProfileAttribute:SlackNotificationPreference"
  },
  "CreatedAt": {
    "S": "2022-07-19T16:56:38.120Z"
  },
  "LastUpdated": {
    "S": "2022-07-19T16:56:38.120Z"
  },
  "RecordTypeQualifier": {
    "S": "UserProfileAttribute"
  },
  "UserProfileAttributeName": {
    "S": "SlackNotificationPreference"
  },
  "UserProfileAttributeValue": {
    "S": "yes"
  }
}
```

Testing

Unit Testing

- createChannel (channelName, members?)
    - If channel does not exist, it should create channel
    - If channel does not exist, it should log error message
    - Add users to channel, if user exists
    - If the member exists, it adds the
    - If the user not found, it log error message
- notifyPrivateChannel (channelName, message, members?)
    - If the channel is existing, it should the post message
    - If the channel is not existing, it should call create channel method to create one and should post message
- notifyPublicChannel(channelName, message)
    - If a content is published, It should post a message in public channel “apgl-notifications”
    - If channel does not exist, it should throw an error
- changeChannelMembers(memberWithAction)
    - If a member is added to the APG content, it should add the member to the private channel
    - If a member is updated their membership in APG content, it should update the member in the corresponding channel
    - if the user not found, it should log error message
- checkChannelExists(channelName)
    - If the function is called, it should check whether the channel is existing or not
- DeleteChannel(channelName)
    - If the APG content is deleted, it should delete the corresponding channel
    - If the channelName does not exist, it should log error message

API Integration Tests

- Create new APG content, verify that a private channel created with the author as a member
- Publish a APG content, see whether the message is posted in public channel
- Trigger the content bounty process for a content that needs update, and verify the notification that is sent in the private channel for the author

- When the author is changed, it has to update the channel membership
- Delete a content to check whether the channel is archived

Telemetry

- The errors are logged to CloudWatch logs
- The following metrics will be posted to CloudWatch
    - Number of channels created
    - Number of members being added
    - Number of members being deleted
    - Number of notifications sent

Summary

To introduce a new feature to the Amazon's AWS product which allows to send notifications through the workflow of the APG library, I have tried and tested out various approaches for slack client lambda implementation and for integrating with Slack API. Team approved to use the asynchronous lambda function to implement slack client. For the integration with slack API, Web API and Conversational API is used and as its scopes are useful for required methods. This allows us to check messages instantly throughout the workflow of APG library. Logs are logged into CloudWatch and checked on the number of channels created, and number of members added. Which keeps track of all the activity of the client.

References

[1] https://aws.amazon.com/prescriptive-guidance

[2] https://docs.aws.amazon.com/prescriptive-guidance/latest/migration-ocm/migration-ocm.pdf

[3] https://api.slack.com/

[4] **https://api.slack.com/methods**

[5] https://docs.aws.amazon.com/prescriptive-guidance/latest/migration-retiring-applications/apg-gloss.html