Complete Guide: Slack as an S3 File Interface

This document provides a comprehensive, step-by-step guide to building a serverless application that allows you to use a Slack channel as an interface for an Amazon S3 bucket. You will be able to upload, fetch, and list files stored in S3 directly from Slack.

This solution uses **AWS Lambda Function URLs**, which is a cost-effective and direct way to connect Slack to your AWS backend without needing API Gateway.

Part 1: AWS Setup

We will begin by creating the necessary infrastructure in your AWS account.

Step 1: Create the S3 Bucket

This is the secure storage location for your files.

1. Navigate to the **S3** service in the AWS Management Console.
2. Click **Create bucket**.
3. **Bucket name**: Enter a globally unique name (e.g., yourname-slack-file-storage).
4. **AWS Region**: Select your preferred region (e.g., eu-north-1). **Remember this region for later**.
5. **Block Public Access settings**: Ensure the **Block all public access** checkbox is **checked**. This is critical for security.
6. Click **Create bucket**.

Step 2: Create the IAM Role

This role grants your Lambda function the necessary permissions.

1. Navigate to the **IAM** service in the AWS Console.
2. Select **Roles** from the left-hand menu and click **Create role**.
3. **Trusted entity type**: Select **AWS service**.
4. **Use case**: Choose **Lambda** and click **Next**.
5. **Add permissions**: Search for and add the following two AWS managed policies:
   - AWSLambdaBasicExecutionRole (Allows writing logs to CloudWatch).
   - AmazonS3FullAccess (Allows the function to read, write, and list objects in S3).
6. Click **Next**.
7. **Role name**: Enter a descriptive name like SlackS3LambdaRole.
8. Click **Create role**.

Step 3: Create the Lambda Function

This is the serverless function that will contain all our logic.

1. Navigate to the **Lambda** service.

2. Click **Create function**.
3. Select **Author from scratch**.
4. **Function name**: Enter a name like SlackS3FileProcessor.
5. **Runtime**: Select **Python 3.11** (or a later version).
6. **Permissions**: Expand **Change default execution role**, select **Use an existing role**, and choose the SlackS3LambdaRole you just created.
7. Click **Create function**.

Part 2: Slack App Setup

Now, we will configure the Slack application that will interface with your users.

Step 4: Create the Slack App

1. Go to the Slack API website and click **Create New App**.
2. Select **From scratch**.
3. **App Name**: Enter a name like S3 File Manager.
4. **Workspace**: Select the Slack workspace where you want to use the app.
5. Click **Create App**.

Step 5: Configure Permissions (OAuth Scopes)

1. From your app's settings page, click on **OAuth & Permissions** in the sidebar.
2. Scroll to the **Scopes** section. Under **Bot Token Scopes**, click **Add an OAuth Scope** and add the following three scopes:
   - chat:write: To post confirmation and error messages.
   - files:read: To get information about shared files.
   - commands: To enable and use slash commands.

Step 6: Install the App and Get the Bot Token

1. Scroll back to the top of the **OAuth & Permissions** page.
2. Click **Install to Workspace**.
3. Follow the on-screen prompts to authorize the app.
4. After authorization, the page will display a **Bot User OAuth Token** (it starts with xoxb-). **Click the Copy button to copy this token.** You will need it in the next part.

Part 3: The Code & Connection

This is the core of the setup where we add the logic and connect the two services.

Step 7: Add the requests Library via Lambda Layer

The Python code uses the requests library, which is not included in Lambda by default.

1. On your local machine (in a terminal), create a directory structure for the layer:

mkdir -p requests-layer/python

2. Install the requests library into that specific directory:
   pip install requests -t requests-layer/python/

   pip3 install requests botocore -t requests-layer/python/
3. Navigate into the requests-layer directory and zip its contents:
   cd requests-layer && zip -r ../requests-layer.zip .

4. In the AWS Lambda console, go to **Layers** in the sidebar, click **Create layer**.
5. **Name**: requests-layer.
6. **Upload**: Upload the requests-layer.zip file you just created.
7. **Compatible runtimes**: Select **Python 3.11** (or your chosen version).
8. Click **Create**.
9. Go back to your SlackS3FileProcessor function, scroll to the **Layers** section at the bottom, click **Add a layer**, select **Custom layers**, and add the requests-layer you just made.

Step 8: Paste the Final Code

Go to your SlackS3FileProcessor function in the AWS Lambda console. In the **Code source** editor, replace all the default code in the lambda_function.py file with the final, working code below.

```
import json
import os
import logging
import boto3
import requests
import base64
from urllib.parse import parse_qs, quote_plus
from botocore.client import Config

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

# --- Environment Variables ---
SLACK_BOT_TOKEN = os.environ.get('SLACK_BOT_TOKEN')
S3_BUCKET_NAME = os.environ.get('S3_BUCKET_NAME')
S3_BUCKET_REGION = os.environ.get('S3_BUCKET_REGION')
```

```
# --- AWS S3 Client ---
s3_client = boto3.client('s3', region_name=S3_BUCKET_REGION,
config=Config(signature_version='s3v4'))


def lambda_handler(event, context):
    """
    Main handler for AWS Lambda.
    Handles Slack events and slash commands.
    """
    logger.info(f"Received event: {json.dumps(event)}")
    logger.info(f"S3 Bucket: {S3_BUCKET_NAME} | Region: {S3_BUCKET_REGION}")

    raw_body = event.get('body', '')
    if event.get('isBase64Encoded', False):
        try:
            raw_body = base64.b64decode(raw_body).decode('utf-8')
        except Exception as e:
            logger.error(f"Base64 decoding failed: {e}")
            return {'statusCode': 400, 'body': 'Bad request body.'}

    # Parse the body
    try:
        body = json.loads(raw_body)
    except (json.JSONDecodeError, TypeError):
        body = parse_qs(raw_body)

    # 1. Slack Challenge Verification
    if body.get('type') == 'url_verification':
        return {'statusCode': 200, 'body': body['challenge']}

    # 2. Handle Slash Commands
    if 'command' in body:
        command = body.get('command')[0]
        if command == '/s3-fetch':
            return handle_s3_fetch(body)
        elif command == '/s3-list':
            return handle_s3_list(body)
```

```
    # 3. Handle file upload events
    event_data = body.get('event', {})
    if event_data.get('type') == 'file_shared':
        return handle_file_upload(event_data)

    return {'statusCode': 200, 'body': json.dumps({'message': 'Event received.'})}


def handle_file_upload(event_data):
    """
    Uploads files shared in Slack to S3.
    """
    file_id = event_data.get('file_id')
    channel_id = event_data.get('channel_id')

    try:
        file_info = get_file_info(file_id)
        file_url = file_info['file']['url_private_download']
        file_name = file_info['file']['name']

        headers = {'Authorization': f'Bearer {SLACK_BOT_TOKEN}'}
        file_response = requests.get(file_url, headers=headers)
        file_response.raise_for_status()

        s3_client.put_object(
            Bucket=S3_BUCKET_NAME,
            Key=file_name,
            Body=file_response.content
        )
        post_slack_message(channel_id, f"✅ File `{file_name}` uploaded to S3.")

    except Exception as e:
        logger.error(f"Error uploading file to S3: {e}")
        post_slack_message(channel_id, f"❌ Error: {e}")
        return {'statusCode': 500}

    return {'statusCode': 200}
```

```
def handle_s3_fetch(body):
    """
    Fetches files from S3 using a presigned URL and posts to Slack.
    """
    try:
        file_name = body['text'][0].strip()
        channel_id = body['channel_id'][0]

        if not file_name:
            return {'statusCode': 200, 'body': 'Please provide a filename. Usage: `/s3-fetch
<filename>`'}

        presigned_url = s3_client.generate_presigned_url(
            'get_object',
            Params={
                'Bucket': S3_BUCKET_NAME,
                'Key': file_name,
                'ResponseContentDisposition': f'attachment;
filename="{quote_plus(file_name)}"'
            },
            ExpiresIn=3600
        )

        message = f"🔗 Here is your download link for `{file_name}` (valid for 1
hour):\n{presigned_url}"
        post_slack_message(channel_id, message)
        return {'statusCode': 200, 'body': f"Fetching `{file_name}`..."}

    except s3_client.exceptions.ClientError as e:
        if e.response['Error']['Code'] == 'NoSuchKey':
            return {'statusCode': 200, 'body': f"❌ File `{file_name}` not found in S3."}
        logger.error(f"S3 Error: {e}")
        return {'statusCode': 200, 'body': "❌ Error fetching the file."}

    except Exception as e:
        logger.error(f"Unexpected error: {e}")
        return {'statusCode': 500, 'body': 'Internal server error.'}
```

```python
def handle_s3_list(body):
    """
    Lists all files in the S3 bucket and posts them to Slack.
    """
    try:
        channel_id = body['channel_id'][0]

        # List objects in the bucket
        response = s3_client.list_objects_v2(Bucket=S3_BUCKET_NAME)
        if 'Contents' not in response:
            post_slack_message(channel_id, "📂 The S3 bucket is empty.")
            return {'statusCode': 200, 'body': "No files found."}

        files = [obj['Key'] for obj in response['Contents']]
        file_list_text = "\n".join([f"- {file}" for file in files])

        post_slack_message(channel_id, f"📂 **Files in S3:**\n{file_list_text}")
        return {'statusCode': 200, 'body': "Files listed."}

    except Exception as e:
        logger.error(f"Error listing S3 files: {e}")
        return {'statusCode': 500, 'body': 'Error listing files.'}


def get_file_info(file_id):
    """
    Fetches Slack file metadata.
    """
    url = 'https://slack.com/api/files.info'
    headers = {'Authorization': f'Bearer {SLACK_BOT_TOKEN}'}
    response = requests.get(url, headers=headers, params={'file': file_id})
    response.raise_for_status()
    file_info = response.json()
    if not file_info.get('ok'):
        raise Exception(f"Slack API error: {file_info.get('error')}")
    return file_info
```

```
def post_slack_message(channel_id, text):
    """
    Sends a message to a Slack channel.
    """
    try:
        response = requests.post(
            'https://slack.com/api/chat.postMessage',
            headers={
                'Authorization': f'Bearer {SLACK_BOT_TOKEN}',
                'Content-Type': 'application/json'
            },
            json={'channel': channel_id, 'text': text}
        )
        response.raise_for_status()
    except requests.exceptions.RequestException as e:
        logger.error(f"Failed to post to Slack: {e}")
```

Step 9: Configure Lambda and Connect to Slack

1. **Deploy the Code**: In the Lambda code editor, click the **Deploy** button to save your changes.
2. **Create Function URL**:
   - In your Lambda function's console, select the **Configuration** tab and then click on **Function URL**.
   - Click **Create function URL**.
   - For **Auth type**, select **NONE**.
   - Click **Save**. A new **Function URL** will be displayed. **Copy this URL**.
3. **Set Environment Variables**:
   - In your Lambda function's **Configuration** tab, click **Environment variables**, then **Edit**.
   - Add the following three variables:
     - **Key**: SLACK_BOT_TOKEN, **Value**: Paste the xoxb- token you copied from Slack in Step 6.
     - **Key**: S3_BUCKET_NAME, **Value**: Enter the name of the S3 bucket you created in Step 1.
     - **Key**: S3_BUCKET_REGION, **Value**: Enter the region of your S3 bucket from Step 1 (e.g., eu-north-1).
   - Click **Save**.

Part 4: Final Configuration and Testing

Step 10: Configure Slack Request URLs

1. Go back to your Slack App configuration page.
2. **Event Subscriptions**:
   - Click **Event Subscriptions** in the sidebar and turn the feature **On**.
   - In **Request URL**, paste your **Lambda Function URL** from Step 9. Slack should show a green "Verified" checkmark.
   - Expand **Subscribe to bot events** and add the file_shared event.
   - Click **Save Changes**.
3. **Slash Commands**:
   - Click **Slash Commands** in the sidebar, then **Create New Command**.
   - **Command**: /s3-fetch
   - **Request URL**: Paste your **Lambda Function URL** again.
   - **Short Description**: Fetches a file from S3.
   - Click **Save**.
   - Click **Create New Command** again.
   - **Command**: /s3-list
   - **Request URL**: Paste your **Lambda Function URL** again.
   - **Short Description**: Lists all files in the S3 bucket.
   - Click **Save**.
4. If Slack prompts you, **reinstall the app** to your workspace to apply the new command permissions.

Step 11: Test Your Application

1. **Invite the Bot**: In the Slack channel you want to use, type @S3 File Manager (or whatever you named your app) and send the message to invite it to the channel.
2. **Upload a File**: Drag and drop a file into the channel. The bot should respond: ✅ File <your-file-name> uploaded to S3.
3. **List Files**: Type /s3-list. The bot should respond with a list of all files in the S3 bucket.
4. **Fetch a File**: Type /s3-fetch <your-file-name>. The bot should respond with a secure, temporary download link for that file.

Your Slack-as-S3-interface is now complete and fully functional!