Automating Document Creation and Storage: Integrating Salesforce, Google Pub/Sub, and C# for PDF Generation

Chirag Amrutlal Pethad

PetSmart.com, LLC, Stores and Services, Phoenix, Arizona, USA
ChiragPethad@gmail.com, ChiragPethad@live.com, Cpethad@petsmart.com

Abstract

The document outlines a comprehensive guide on implementing prompt engineering in Salesforce, detailing steps for integrating AI services, creating custom Apex classes, testing and refining prompts, deploying and monitoring performance, and optimizing for continuous improvement. It emphasizes best practices such as understanding business context, clarity in prompts, iterative testing, and ethical considerations, while providing specific instructions for creating various types of prompt templates, including text completion, summarization, and classification.

Keywords: Attachments, Documents, Storage, Cloud Storage, Authentication, REST, Security, Efficient, Scalable, Compliance, CDN, Cloud Run, File Storage, Limits.

1. Introduction

Salesforce File Storage [12] is a cloud-based system used to store and manage files, documents, and other media within the Salesforce ecosystem. Files in Salesforce can be uploaded through various objects, used in Chatter posts, integrated with records, and shared with users for collaboration.

Salesforce provides several ways to store documents, each with its own set of limits and best practices. Understanding these limits is crucial for managing storage effectively and ensuring optimal performance of your Salesforce implementation. As businesses increasingly rely on Salesforce for customer relationship management, the need for efficient data storage solutions has grown.

Amazon S3, Azure Blob Storage, and Google Cloud Storage all offer a scalable, durable, and secure environment for storing large volumes of unstructured data, such as documents and media files. Integrating Salesforce with these external storage options allows organizations to leverage both platforms' strengths, providing a streamlined solution for managing documents.

This white paper outlines a comprehensive solution for integrating with Google Cloud to generate PDF document and uploading that document to Cloud Storage using event driven architecture there by decoupling the different components of the system. The approach leverages Salesforce's capabilities to publish an event to Google Pub Sub and Google cloud’s Compute and storage infrastructure, offering a seamless way to manage and store large volumes of documents outside the Salesforce ecosystem. This integration provides a cost-effective, scalable, and reliable solution for businesses to manage their document storage needs.

2. Limitations And Challenges of File Storage in Salesforce [12]

A. Storage Limits

Salesforce imposes storage limits on both data and files. Exceeding these limits can result in additional storage costs or a need for third-party storage solutions, especially for organizations with large volumes of files. File storage is specifically for storing Files, Content Documents, Chatter Files, Images, and attachments.

B. File Storage Allocations

Salesforce provides a default file storage allocation based on the edition and the number of user licenses per organization. The default allocations for Enterprise, Unlimited, Performance and Professional Salesforce edition is 10GB + 2GB per user license. Additional storage can be purchase in increments of 1GB.

C. Maximum File Size

The maximum size for files attached to records (like attachments and feed attachments) and chatter is 2 GB. The maximum size for files uploaded via the Documents tab is 5 MB.

D. Storage Usage Calculations

Each attachment, Files stored in Salesforce CRM Content and Files Shared in Chatter counts towards the file storage limit.

E. Complex Managememt

Managing file storage across large organizations with diverse user needs can become complex, especially with sharing settings, file expiration, and permission controls.

F. Performance Impacts

As file storage grows in Salesforce, managing large file repositories can impact performance, especially for file retrieval and sharing in record-heavy environments.

G. Archival and Retention

Salesforce doesn't provide built-in advanced archiving or file retention policies, requiring businesses to implement custom solutions to manage long-term storage and retention of files.

3. Benefits of Leveraging Cloud Storage for Salesforce

Google Cloud Storage (GCS) [5] offers a scalable, secure, and highly durable cloud storage solution for managing vast amounts of data. Integrating GCS with platforms like Salesforce or other business systems provides several advantages, including cost efficiency, scalability, and ease of management. Here are the key benefits:

H. Cost-Effective Scalability

- **Pay-as-you-go pricing:** Google Cloud Storage allows you to only pay for the storage you use, avoiding upfront costs. This flexibility is ideal for businesses that may have fluctuating storage needs.
- **Unlimited scalability**: GCS can scale seamlessly to accommodate growing data volumes, making it suitable for large enterprises or businesses experiencing rapid growth.

I. Enhanced File Storage Capacity

- **Offloading Salesforce Storage**: By integrating GCS with Salesforce, organizations can offload files from Salesforce's native storage limits, avoiding costly additional Salesforce storage purchases.
- **Large File Handling**: GCS can manage large files (up to terabytes), which may be challenging to store in platforms like Salesforce without integration.

J. High Availability and Durability

- **Global accessibility**: Google Cloud Storage is built on Google's global infrastructure, providing high availability across regions. Data stored in GCS is redundantly stored across multiple locations, ensuring durability (99.999999999% durability) and minimal downtime.
- **Automatic backups**: With built-in redundancy, GCS automatically replicates files, protecting against data loss or hardware failure.

K. Security and Compliance

- **Advanced Security features**: GCS provides built-in encryption for data at rest and in transit, role-based access controls (RBAC), and identity and access management (IAM) to ensure sensitive files are protected.
- **Compliance Certifications**: Google Cloud meets numerous compliance standards such as GDPR, HIPAA, ISO 27001, and SOC 2, making it a secure option for storing sensitive data.

L. Seamless Integration and Accessibility

- **Easy API Integration**: GCS offers APIs and SDKs that allow for seamless integration with Salesforce, enabling automatic file uploads, storage, and retrieval between systems.
- **Collaboration with Google Workspace**: GCS integrates natively with Google Workspace tools like Google Docs, Sheets, and Drive, allowing teams to access and edit files stored in the cloud directly from these applications.

M.Customizable Storage Classes

- GCS offers different storage classes (Standard, Nearline, Cold line, and Archive) tailored to different use cases based on access frequency. This allows organizations to store frequently accessed files at higher performance while optimizing costs for less frequently accessed data.

N. Improved File Management and Search

- **Centralized Storage**: With GCS, files can be managed centrally, which improves file searchability and retrieval times, especially when dealing with large data sets.
- **Versioning and Object Lifecycle Management**: GCS allows versioning of files and object lifecycle management policies to automatically transition files between storage classes or delete files after a set period, improving data retention and compliance.

O. Analytics and Machine Learning Integration

- **Big Query AI/ML Integration**: Files stored in GCS can be easily linked to Google Big Query for analytics or integrated into machine learning pipelines for AI-driven applications, offering deeper insights and automation possibilities.

P. Reliability and Speed

- Google Cloud's infrastructure ensures low-latency access to stored files, making it suitable for mission-critical applications that require high performance and reliability.

4. Step By Step Implementation

Here's a step-by-step guide for deploying an ASP.NET Core web API to **Google Cloud Run**, which is a fully managed service that allows you to run containerized applications. To implement the use case where Salesforce publishes a Pub/Sub message for invoice creation, a C# application subscribes to the message using the push model, generates a PDF invoice, and saves it to Cloud Storage, follow the detailed steps listed below.

Q. Prerequisites

- **Google Cloud Account**: If you don't have one, create an account on Google Cloud Platform.
- **Google Cloud SDK**: Install the Google Cloud SDK [7].
- **.NET Core SDK**: Install the .NET Core SDK [6].
- **Salesforce**: Salesforce Developer Sandbox or Developer Edition Org with Administrator profile/access.

R. Create Google Cloud Service Account

- Log in to GCP account.
- Go to IAM & Admin -> Service Accounts.
- Create a new Service Account with the required permissions (e.g., Pub/Sub Publisher for Pub/Sub and Storage Admin for Cloud Storage).
- Generate a **JSON key** for this Service Account by clicking on the service account and then **Keys** → **Add Key** → **Create New Key** → **JSON**.
- Save the JSON key file, as you will need it later to create a JWT (JSON Web Token).

S. Create Remote Site Settings

- Navigate to Setup.
- In the Quick Find box, enter "Remote Site Settings".
- Create a new remote site with the following details:
- Remote Site Name: GCSRemoteSite
- Remote Site URL: https://pubsub.googlapis.com/

T. Setup and Configure Named Credentials [3] in Salesforce

- **Go to Setup** → **Named Credentials**.
- **Click New Named Credential** and configure the following:
- **Label**: Choose a descriptive label (e.g., GooglePubSub).
- **Name**: This is used in your Apex code (e.g., GooglePubSub).
- **URL**: Set this to https://pubsub.googleapis.com/ (this is the base URL for Google Pub/Sub).
- **Identity Type**: Select **Named Principal** (for a single OAuth configuration).
- **Authentication Protocol**: Choose **OAuth 2.0**.
- **Scope**: This depends on the Google API you're using. For Pub/Sub, use https://www.googleapis.com/auth/pubsub.
- **Issuer**: Leave this blank (this field is used when integrating with platforms like Salesforce Communities).
- **Client ID**: This is found in the Google Cloud Service Account's JSON file under "client_id".
- **Client Secret**: This will be empty because we are using JWT-based authentication instead of client credentials.
- **Token Endpoint URL**: Use https://oauth2.googleapis.com/token.
- **Use Digital Signatures**: Check this box.
- **Certificate**: Upload the **Private Key** from the Google Service Account JSON file.
- **Principal Type**: Choose **JWT Bearer Token Flow** (this allows for seamless Google Service Account authentication).
- Save the Named Credential.

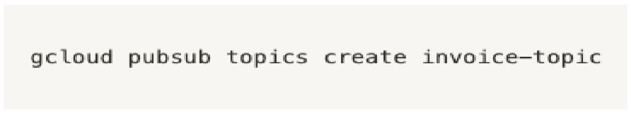
U. Create an Authentication Provider for JWT Bearer Token Flow in Salesforce

Since Salesforce does not natively support Google Service Account authentication via OAuth 2.0 [2] without some extra configuration, you'll need to set up a custom JWT bearer flow using an Authentication Provider.

- Go to Setup -> Auth Providers -> New
- **Provider Type**: Choose **OpenID Connect**.
- **Name**: Use something like GoogleServiceAccountAuth.
- **Consumer Key**: Use the "client_id" from the Service Account's JSON file.
- **Consumer Secret**: Leave this empty for JWT authentication.
- **Authorize Endpoint URL**: Leave blank (since we're using JWT flow).
- **Token Endpoint URL**: https://oauth2.googleapis.com/token.
- **Default Scopes**: https://www.googleapis.com/auth/pubsub.
- **Registration Handler**: Create a dummy handler if required (it won't be used for this flow).
- Save the Authentication Provider.

V. Configure Google Cloud Pub/Sub [9][10]

- Go to the Google Cloud Console.
- Navigate to **Pub/Sub -> Topics**.
- Click **Create Topic**.
- Name the topic, example "invoice-topic" and click **Create**.
- Alternatively, you can create the topic from Google Cloud CLI using below command.

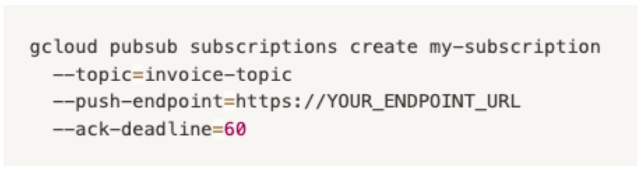


```
gcloud pubsub topics create invoice-topic
```

Figure 1: G-Cloud command to create Pub/Sub Topic

W.Create a Push Subscription

- In the same Pub/Sub section, click on **Subscriptions**.
- Click **Create Subscription**.
- Choose the topic created above (invoice-topic).
- Under **Delivery Type**, select **Push**.
- In the **Push Endpoint URL**, provide the URL where your C# app will listen for messages (e.g., https://yourapp.com/push-endpoint). (We will update this placeholder in a later step).
- Click **Create**.
- Alternatively, you can create the push subscription via the command line:



```
gcloud pubsub subscriptions create my-subscription
  --topic=invoice-topic
  --push-endpoint=https://YOUR_ENDPOINT_URL
  --ack-deadline=60
```

Figure 2: G-Cloud command to create a Push Subscription

X. Configure Platform Event [13] in Salesforce (Optional)

- In Salesforce Setup, search for **Platform Events**.

- Create a new Platform Event called InvoiceEvent.
- Define fields like InvoiceId, CustomerName, Amount, DueDate, etc.

Y. Create Apex Classes [1][4] to Publish the Event to Google Cloud Pub/Sub

- Use an Apex trigger or flow to publish an event when an invoice is created in Salesforce as follows.

```
InvoiceEvent__e newInvoice = new InvoiceEvent__e(
    InvoiceId__c = '123',
    CustomerName__c = 'John Doe',
    Amount__c = 1500,
    DueDate__c = Date.today()
);
EventBus.publish(newInvoice);
```

Figure 3: Apex Code [4] to Publish Platform Event in Salesforce

- In the trigger, integrate Salesforce with Pub/Sub by calling the Google Pub/Sub API, publishing the InvoiceEvent to the invoice-topic as follows.

```
String endpoint = '/v1/projects/' + projectId
        + '/topics/' + topicId + ':publish';
HttpRequest req = new HttpRequest();
req.setEndpoint('callout:GooglePubSub' + endpoint);
req.setMethod('POST');
req.setHeader('Content-Type', 'application/json');
String base64Message = EncodingUtil.base64Encode(
    Blob.valueOf(JSON.Serialize(newInovoiceEvent)));
String requestBody =
    '{"messages":[{"data":"' + base64Message + '"}]}';
req.setBody(requestBody);Http http = new Http();
HttpResponse res = http.send(req);
if (res.getStatusCode() == 200) {
  System.debug('Message published successfully: '
    + res.getBody());
} else {
    System.debug('Failed to publish message: '
    + res.getStatusCode() + ' ' + res.getStatus());
}
```

Figure 4: Apex Code to make Callout to Pub / Sub to Publish a message

Z. Create C# (C Sharp) Application

- Install .NET Core SDK [6]
- Create a new project as follows.

```
dotnet new webapi -n InvoiceProcessor
cd InvoiceProcessor
```

Figure 5: .NET Core Web API project creation command

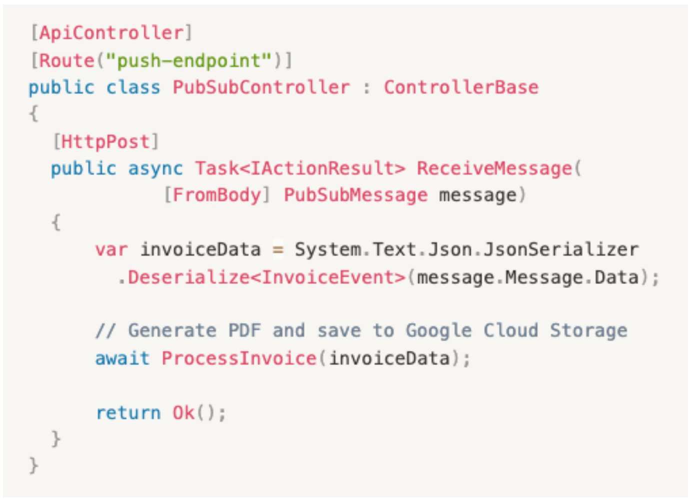
- Install Google Cloud Libraries for Pub/Sub, Cloud Storage and PDF generation as follows.

```
dotnet add package Google.Cloud.PubSub.V1
dotnet add package Google.Cloud.Storage.V1
dotnet add package DinkToPdf
```

Figure 6: .NET Core Command to Install libraries

AA. Create Pub/Sub Listener Endpoint in C# Application

- In your Startup.cs or Program.cs, create a controller to listen for Pub/Sub messages. This is the push endpoint that you provided earlier in the Pub/Sub subscription as follows.



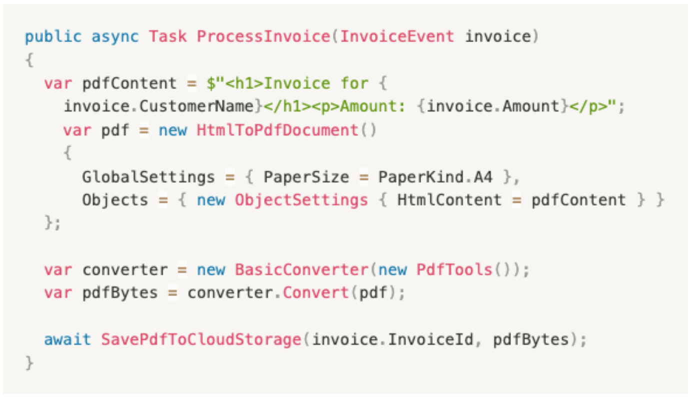
```
[ApiController]
[Route("push-endpoint")]
public class PubSubController : ControllerBase
{
  [HttpPost]
  public async Task<IActionResult> ReceiveMessage(
             [FromBody] PubSubMessage message)
  {
      var invoiceData = System.Text.Json.JsonSerializer
        .Deserialize<InvoiceEvent>(message.Message.Data);

      // Generate PDF and save to Google Cloud Storage
      await ProcessInvoice(invoiceData);

      return Ok();
  }
}
```

Figure 7: Controller / Endpoint to listen for Pub / Sub messages

- We extract the message from the request body, which contains the Pub/Sub message data.
- We deserialize the message into a variable.
- We process the message, and we return an HTTP 200 (OK) status to acknowledge that the message has been received.

BB. Create PDF Generator in C# Application

- Use the DinkToPdf library to generate a PDF as follows.

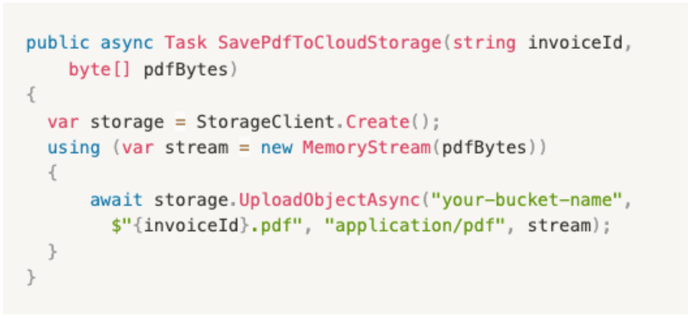


```
public async Task ProcessInvoice(InvoiceEvent invoice)
{
  var pdfContent = $"<h1>Invoice for {
    invoice.CustomerName}</h1><p>Amount: {invoice.Amount}</p>";
    var pdf = new HtmlToPdfDocument()
    {
      GlobalSettings = { PaperSize = PaperKind.A4 },
      Objects = { new ObjectSettings { HtmlContent = pdfContent } }
  };

  var converter = new BasicConverter(new PdfTools());
  var pdfBytes = converter.Convert(pdf);

  await SavePdfToCloudStorage(invoice.InvoiceId, pdfBytes);
}
```

Figure 8: A method / task to generate the PDF

CC. Create Cloud Storage [5] File Uploader in C# Application

- Use the Google Cloud Storage API to save the generated PDF as follows.



```
public async Task SavePdfToCloudStorage(string invoiceId,
    byte[] pdfBytes)
{
  var storage = StorageClient.Create();
  using (var stream = new MemoryStream(pdfBytes))
  {
      await storage.UploadObjectAsync("your-bucket-name",
        $"{invoiceId}.pdf", "application/pdf", stream);
  }
}
```

Figure 9: A method / task to upload the pdf to Cloud Storage

DD. Test the Application Locally

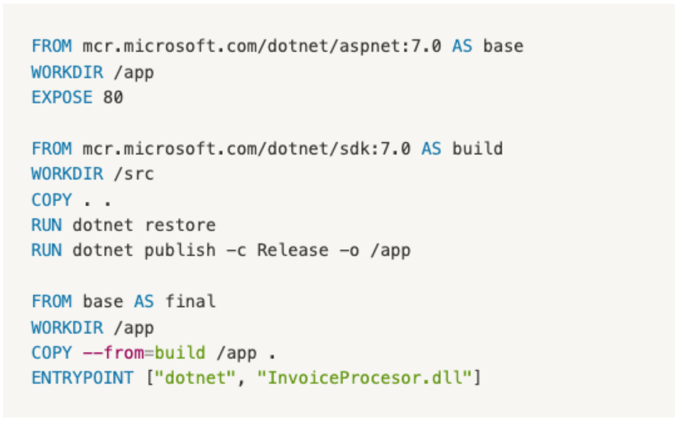
- You can run your application locally to verify its working as expected by executing the command "dotnet run".

- Since Pub/Sub requires a publicly accessible endpoint, testing push delivery locally might not be feasible. We will deploy it to Google Cloud Run in the next step.

EE. Deploy the application to Google Cloud Run [11]

To make the application publicly accessible, you can deploy it to Google Cloud Run.

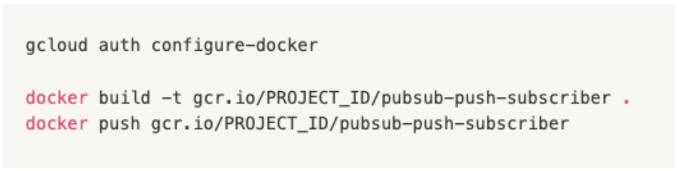
- Build a Dockerfile for the app in your project root folder as follows.



```
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /src
COPY . .
RUN dotnet restore
RUN dotnet publish -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=build /app .
ENTRYPOINT ["dotnet", "InvoiceProcesor.dll"]
```

Figure 10: Docker file

- Build and push the Docker [8] Image to Google Container Registry (GCR) as follows.

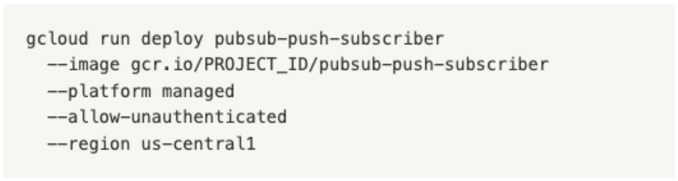


```
gcloud auth configure-docker

docker build -t gcr.io/PROJECT_ID/pubsub-push-subscriber .
docker push gcr.io/PROJECT_ID/pubsub-push-subscriber
```

Figure 11: Commands to authenticate, Build and push the image to GCR

FF. Deploy the application to Cloud Run

- Deploy to Cloud Run (replace PROJECT_ID with your actual project ID) as follows.

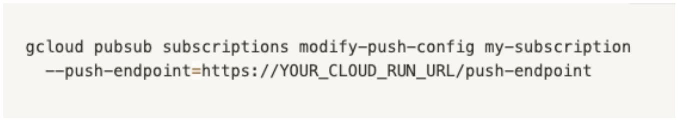


```
gcloud run deploy pubsub-push-subscriber
  --image gcr.io/PROJECT_ID/pubsub-push-subscriber
  --platform managed
  --allow-unauthenticated
  --region us-central1
```

Figure 12: Command to deploy the GCR Image to Cloud Run

GG. Update the Pub Sub Subscription

- Once your Cloud Run app is deployed, you will get a public URL. Update the Pub/Sub push subscription to use this URL from Console or via Command as follows.



```
gcloud pubsub subscriptions modify-push-config my-subscription
  --push-endpoint=https://YOUR_CLOUD_RUN_URL/push-endpoint
```

Figure 13: G-Cloud Command to update the Pub Sub Subscription

HH. Test the Application End to End

- Create an invoice record in Salesforce.
- Verify that the event is published to Google Pub/Sub.
- Check that the C# app receives the message, generates the PDF, and uploads it to Google Cloud Storage Bucket.
- If everything is set up correctly, you should see the message being logged in your C# app console, or you can view logs via **Cloud Logging** in the GCP console.

5. Challenges

II. Message Delivery and Reliability

- **Challenge**: Ensuring that the message from Salesforce is reliably delivered to the C# app via Google Pub/Sub can be tricky, especially if there are network issues, service downtimes, or misconfigurations. The Pub/Sub push model depends on the availability of the receiving endpoint.
- **Mitigation**: Messages may fail to deliver if the C# app is not available or if the endpoint is misconfigured. If the message is not acknowledged within a specified timeout period, it may be re-delivered multiple times, potentially leading to duplicates.

JJ. Handling Duplicate Messages

- **Challenge**: Google Pub/Sub guarantees at-least-once delivery, meaning that the C# app may receive the same message more than once. This could result in generating multiple PDFs for the same invoice.
- **Mitigation**: The system must be able to handle duplicate messages effectively, ensuring that duplicate invoice PDFs are not created.

KK. Security and Access Control

- **Challenge**: Ensuring secure communication between Salesforce, Pub/Sub, and the C# app is critical. Improper security configurations can expose sensitive invoice data or cloud storage to unauthorized users.
- **Mitigation**: Misconfiguring service accounts, permissions, or API access could result in security vulnerabilities.

LL. API Rate Limits and Quotas

- **Challenge**: Both Salesforce and Google Cloud APIs have rate limits and quotas. Exceeding these limits could result in delays or failures in processing invoices.
- **Mitigation**: If you have a high volume of invoices, you need to carefully monitor and handle API rate limits to avoid downtime or incomplete message processing.

MM. PDF Generation Performance

- **Challenge**: Generating PDFs from invoice data, especially in bulk or for complex invoice templates, could slow down the system. Poor performance in PDF generation can cause delays in the entire process, leading to bottlenecks.
- **Mitigation**: Ensuring that the C# app is performant and can handle large volumes of messages and PDF generations in a timely manner is essential.

NN. Error Handling and Retries

- **Challenge**: Errors can occur at multiple stages—message delivery, PDF generation, or storage upload. Without proper error handling and retry mechanisms, messages could be lost, and the system may fail silently.
- **Mitigation**: If a failure occurs during PDF generation or uploading, the system must retry safely without creating duplicates or errors.

OO. Storage Costs

- **Challenge**: Storing PDFs in Google Cloud Storage can incur costs, especially as the number of invoices grows over time. If file retention policies aren't properly managed, this could lead to escalating costs.
- **Mitigation**: Managing long-term storage costs by implementing lifecycle rules in Google Cloud Storage is important.

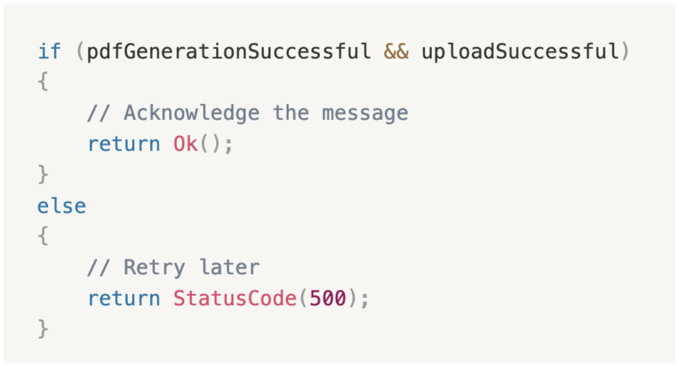
6. Best Practices

PP. Idempotency and Duplicate Handling

- Implement idempotency in your C# app to ensure that processing the same message multiple times does not result in duplicate PDFs or data errors. One way to do this is to check if the PDF for a specific invoice has already been generated before creating a new one.
- Store metadata in a database or Cloud Storage to track which invoices have been processed.

QQ. Message Acknowledgement

- Ensure that the C# app only acknowledges the message once the PDF is successfully generated and stored in Cloud Storage. If the message fails to process, do not acknowledge it so Pub/Sub can re-deliver it.



```
if (pdfGenerationSuccessful && uploadSuccessful)
{
    // Acknowledge the message
    return Ok();
}
else
{
    // Retry later
    return StatusCode(500);
}
```

Figure 1: Returns Http Response Status Code to the PUB SUB Subscription

RR. Implement Secure Communication

- Use secure, encrypted communication channels (HTTPS) between all components (Salesforce, Pub/Sub, C# app). Ensure that the C# app is running on a secure public endpoint with TLS/SSL and use OAuth or service account keys securely to authenticate with Google Cloud services.
- Enable TLS/SSL for the C# app.
- Store service account credentials securely and avoid hardcoding sensitive information.

SS. Use Service Accounts with Least Priviledge

- Follow the principle of least privilege by creating a Google Cloud Service Account that has only the necessary permissions to subscribe to Pub/Sub and write to Cloud Storage. Avoid granting excessive permissions.
- Assign "roles/pubsub.subscriber" for Pub/Sub and "roles/storage.objectCreator" for Cloud Storage to minimize the permissions granted to the service account.

TT. Error Handling and Retries

- Implement robust error handling and retry logic in the C# app to handle failures in PDF generation, message processing, or uploading to Cloud Storage. You can also use Google Cloud Pub/Sub dead-letter queues (DLQ) to manage failed messages and avoid message loss.
- Log errors when PDF generation or upload fails and retry processing a limited number of times.
- Use exponential backoff for retries to avoid overwhelming the system in case of failures.

UU. Monitor and Scale

- Monitor the performance of the C# app and scale it horizontally, if necessary, especially for high volumes of invoices. You can use Google Cloud's monitoring tools to track Pub/Sub message delivery and performance.
- If you notice delays or bottlenecks, you might need to increase the number of instances of the C# app

or optimize the PDF generation process.

VV. Implement Object Lifecycle Management in Cloud Storage

- Implement lifecycle policies for objects in Google Cloud Storage to automatically move or delete older PDFs after a certain period. This will help manage storage costs and prevent excessive accumulation of files.
- Define rules in Cloud Storage to transition PDFs to a lower-cost storage class (e.g., Nearline or Cold line) or delete them after a defined retention period.

WW. API Rate Limit Monitoring

- Monitor the API usage for Salesforce and Google Cloud to ensure you don't hit rate limits. If necessary, batch messages or use throttling mechanisms to avoid overwhelming the system.

XX. Test for Edge Cases

- Thoroughly test the system for edge cases, such as network failures, duplicate messages, large invoice sizes, and high-volume message bursts, to ensure stability under different scenarios.

7. Conclusion

In conclusion, Integrating Salesforce with Cloud Storage provides a powerful solution for managing and storing documents. By following the outlined steps, businesses can leverage the strengths of both platforms, ensuring secure, scalable, and efficient document management. This white paper provides a detailed roadmap for implementing this integration, enabling organizations to optimize their data storage strategy.

Managing document storage within Salesforce requires an understanding of the platform's limits and effective use of its tools and features. By leveraging best practices and integrating with external storage solutions, organizations can optimize their storage usage, ensure compliance, and maintain the performance and efficiency of their Salesforce environment.

References

1. Apex Developer Guide –
2. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dev_guide.htm
3. OAuth2.0 Documentation - https://oauth.net/2/
4. Named Credentials – https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_callouts_named_credentials.htm
5. Apex REST Callouts
6. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_callouts.htm
7. Google Cloud Storage - https://cloud.google.com/storage/docs
8. ASP.NET Core - https://dotnet.microsoft.com/en-us/apps/aspnet
9. Google Cloud SDK - https://cloud.google.com/sdk?hl=en
10. Docker Containers - https://www.docker.com/resources/what-container/
11. Push Subscription - https://cloud.google.com/pubsub/docs/push
12. Create Push Subscription - https://cloud.google.com/pubsub/docs/create-push-subscription
13. Cloud Run Deployment - https://cloud.google.com/run/docs/deploying
14. Salesforce File Storage - https://help.salesforce.com/s/articleView?id=sf.files_storage.htm&type=5
15. Salesforce Platform Events - https://developer.salesforce.com/docs/atlas.en-
16. us.platform_events.meta/platform_events/platform_events_intro.htm