OpenPlatform (SuperApp) Plan Documents Arch

OpenPlatform - Plan Documents Architecture

Contents

Feature Summary

This feature is proposed as an OpenPlatform aggregation designed to server users their plan documents. The API provides a new BFF (Backend-for-Frontend) for the OpenPlatform, integrated with the existing Aetna Health's AWS- Benefits Service currently used by Aetna Health. The solution will handle scenarios for displaying single, multiple, or zero available documents, **ensuring resource availability through feature flag checks, and integrating specific availability messaging in lieu of or in addition to documents.** It will also include support for language preferences.

The primary goal is to route the plan document list and PDF retrieval through a new dedicated OpenPlatform BFF to the existing AWS services. This approach benefits from existing capabilities like caching, which reduces the load on core services and provides a more resilient and faster user experience. The solution must also account for potential gaps in upstream data, such as missing plan names or document descriptions, and handle them gracefully. Search functionality is not in scope for the initial release but is on the roadmap, and the architecture should not degrade performance for future search implementations.

Discovery

Several options were considered for fetching the plan document list and retrieving the PDFs. The primary challenge is balancing app startup performance, on-page performance, and implementation complexity, especially considering the known data inconsistencies.

Architectural Options Analysis

Here's a breakdown of the considered architectural options, their descriptions, and a "meatball" chart summarizing their pros and cons.

Option 1: Adapt Existing On-Demand Architecture (Chosen Direction)

Description: This approach mirrors the existing Aetna Health architecture. A new route will be created in a dedicated OpenPlatform BFF. When a user navigates to the Plan Documents page, the client first performs **feature flag checks** to determine eligibility and message availability. If eligible for documents, the client calls the BFF, which in turn calls the backend Benefits Service to get the document list for that user using specific membershipResourceId and policyResourceId parameters. PDF retrieval follows the established pattern: the client requests a PDF via the BFF, which first checks an S3 cache and, if not present, retrieves it from the core endpoint and caches it for future requests.

Option 2: App Startup Pre-fetch

Description: This option proposes fetching the plan document list when the application starts up, based on the observation that plan metadata is sometimes missing. The data would be sourced from a plan doc index or API and cached on the client or in the BFF for the duration of the user's session. This would make the Plan Documents page load instantly.

Option 3: BFF-Centric Caching & Augmentation (Hybrid)

Description: In this model, the OpenPlatform BFF takes on more responsibility. It follows the on-demand pattern of Option 1 but introduces an additional layer of caching within the BFF itself, tailored to the OpenPlatform's traffic patterns. The BFF would be responsible for calling the backend Benefits Service and then augmenting the response with metadata sourced from another service or a static cache (e.g., "sourcing from app startup" data loaded into the BFF) before returning the final payload to the client.

Meatball Chart: Pros and Cons of Architectural Options

Feature	Option 1: Adapt Existing On-Demand Architecture	Option 2: App Startup Pre-fetch	Option 3: BFF-Centric Caching & Augmentation
Pros	- Reuses proven architecture	- Extremely fast user experience on page load	- Fast app startup
	- Minimal app startup impact	- Augments missing metadata centrally at startup	- Robust data handling/augmentation
	- Leverages existing caching		- Optimized caching for OpenPlatform
	- Lower complexity		- Simplified client-side logic

Cons	- On-page performance dependent on real-time API call	- Increases app startup time	- Increased BFF complexity
	- Requires BFF/client to handle data gaps	- Data can become stale	- Potential for slight latency
		- Increased complexity for app startup data fetching	

Chosen Direction

Option 1 is the chosen direction due to its simplicity and direct reuse of existing, proven infrastructure. This aligns with best practices by prioritizing a phased approach, starting with a lean, functional solution that leverages existing, well-tested components. It reduces initial development time and risk by building upon a stable foundation.

To mitigate the risk of missing data, the OpenPlatform BFF will incorporate a **limited augmentation capability** (as described in Option 3) to normalize the data before it reaches the client. This hybrid approach ensures that critical display fields are present, without adding the complexity of full BFF-side caching in the initial phase. This approach aligns with the principle of "start small, iterate fast" while still addressing known data quality concerns.

Solution Sketches

OpenPlatform (SuperApp) Plan Docs Solution Sketches

This solution sketch illustrates the proposed architecture for accessing plan documents within the OpenPlatform (SuperApp) ecosystem. The diagram shows the end-to-end flow, starting with the CVS Super App initiating requests for a document list or a specific PDF. These requests are routed through a new, dedicated Backend-for-Frontend (BFF) which orchestrates calls to various existing

downstream services, including plan groups and the plan documents service, to ultimately retrieve the necessary data or PDF document from underlying AWS services and caches.

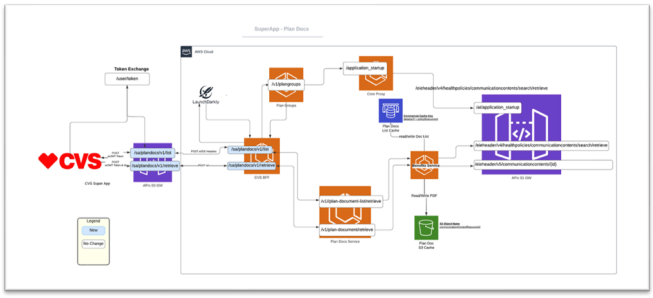


Figure 1: Solution Sketch – Simplified

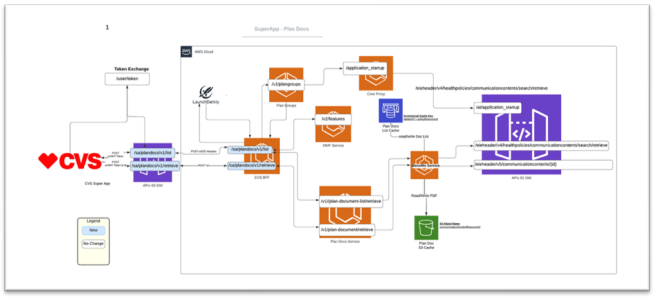


Figure 2: Solution Sketch – Extended (Features)

Plan Docs via AWS Solution Sketches

These diagrams depict the existing Aetna Health (AH) architecture that the new OpenPlatform solution is built upon. They detail the current, proven process AH clients use to retrieve plan document lists and PDFs. This existing model includes crucial components like the Benefits Service, caching layers (Redis for lists and S3 for PDFs), and interactions with core data sources like FileNet. The diagrams also show integrations with other services, such as the Features API (MMF), which are leveraged by the new solution but not re-diagrammed in the simplified OpenPlatform sketches for brevity.

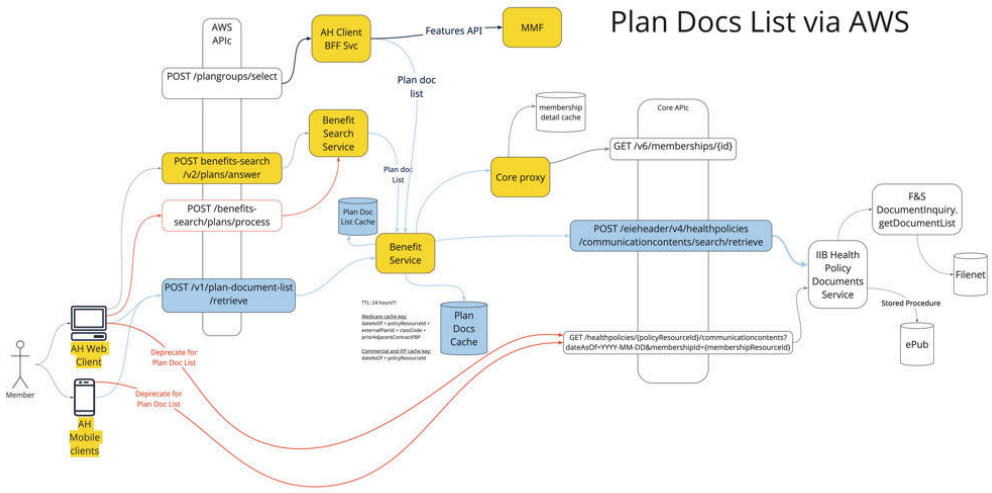


Figure 3: Plan Docs List via AWS (existing solution)

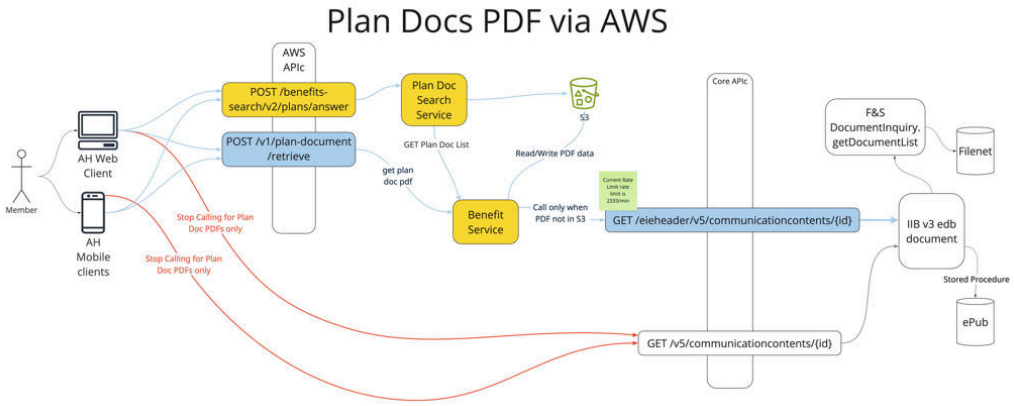


Figure 4: Plan Docs PDF via AWS (existing solution)

The high-level flow involves the OpenPlatform client communicating with a new OpenPlatform BFF. This BFF will act as a proxy to the existing AWS-based backend infrastructure.

Convert OpenPlatform id_token to Application Startup Headers and Content

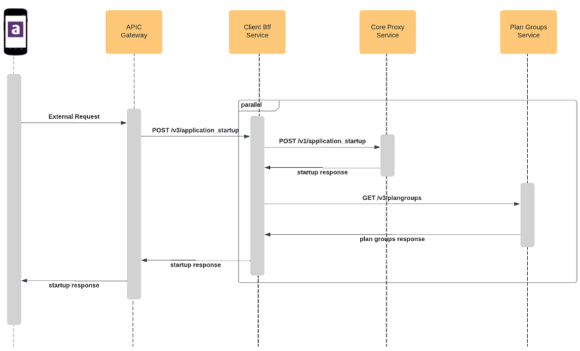


Figure 5: Open Platform Service Application Startup Sequence

To access backend services, the OpenPlatform solution first requires an application startup sequence to be performed. The client initiates this by passing its bearer token and

id_token, which contains the necessary user context. As shown in the sequence diagram, this external request flows through the APIC Gateway to a Client BFF Service. This service then orchestrates parallel calls to the Core Proxy Service (to get the main startup response) and the Plan Groups Service. The aggregated startup response prepares the user's session with the required context, including EIE headers and membership data, which are essential for all subsequent service calls like retrieving the plan document list.

Service Logic

Sequence Diagrams

The following diagrams illustrate the detailed, step-by-step interactions between the various services involved in retrieving plan documents.

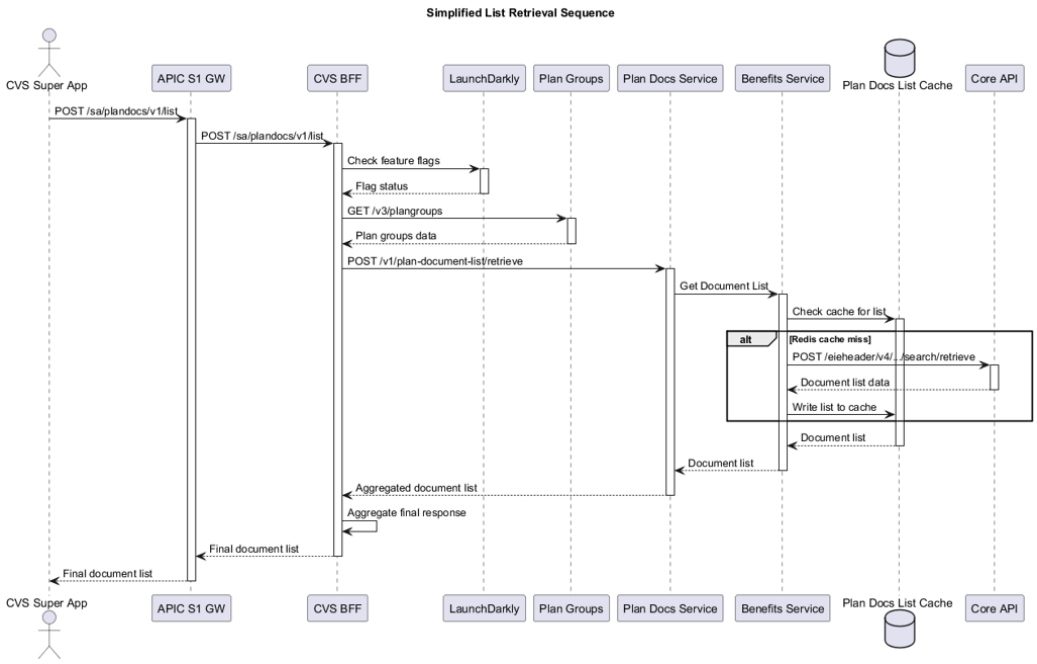
Simplified List Retrieval Sequence

This sequence illustrates the primary path for fetching the list of available plan documents. The user's request from the CVS Super App hits the /sa/plandocs/v1/list endpoint on the APIC Gateway, which forwards it to the CVS BFF.

The BFF then orchestrates a series of preparatory calls:

1. It queries **LaunchDarkly** to check for relevant feature flags.
2. It calls the **Plan Groups** service to gather necessary plan metadata.

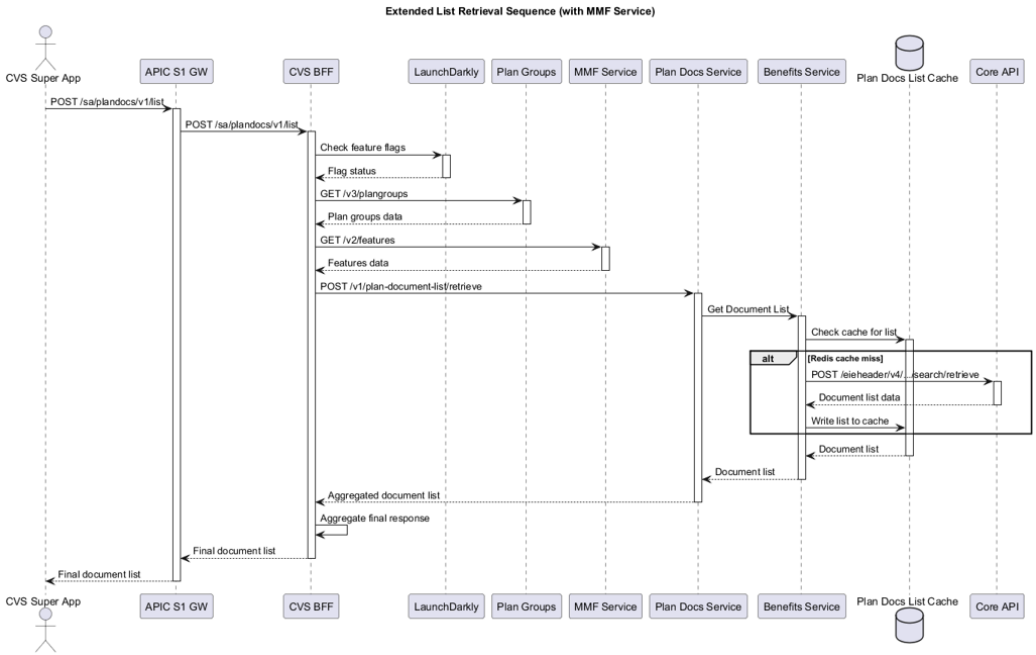
With this information, the BFF calls the Plan Docs Service, which hands off to the Benefits Service. The Benefits Service checks the Redis-based **Plan Docs List Cache**. On a cache miss, it calls the Core API to get the authoritative list, which it then caches in Redis before returning the data. Finally, the BFF aggregates this information into the final response for the client.



Extended List Retrieval Sequence (with MMF Service)

This sequence builds upon the simplified flow by adding an interaction with the MMF (Member & Member Features) Service. This is required for more complex scenarios where additional feature checks are necessary to determine document eligibility.

The flow is identical to the simplified sequence until after the call to the Plan Groups service. At this point, the BFF makes an additional call to the **MMF Service** to get /v2/features data. This data is then used in conjunction with the other information to make a more informed request to the downstream Plan Docs Service. The rest of the flow, including the caching logic with Redis and the Core API, remains the same.



PDF Retrieval Sequence

This sequence details the process of fetching a specific PDF document.

1. **Client to BFF**: The flow begins with the CVS Super App making a POST request to the /sa/plandocs/v1/retrieve endpoint, containing the documentId in the payload.
2. **BFF to Backend**: The request is proxied through the APIC Gateway to the new CVS BFF. The BFF then calls the existing Plan Docs Service, which in turn orchestrates the retrieval by querying the Benefits Service.
3. **Backend to S3 Cache**: The Benefits Service first checks the configured Plan Doc S3 Cache for the requested PDF using the communicationContentResourceId as the key.

On a cache hit, the document is found and streamed back immediately through the service chain to the client.

Backend to Core API (Cache Miss):

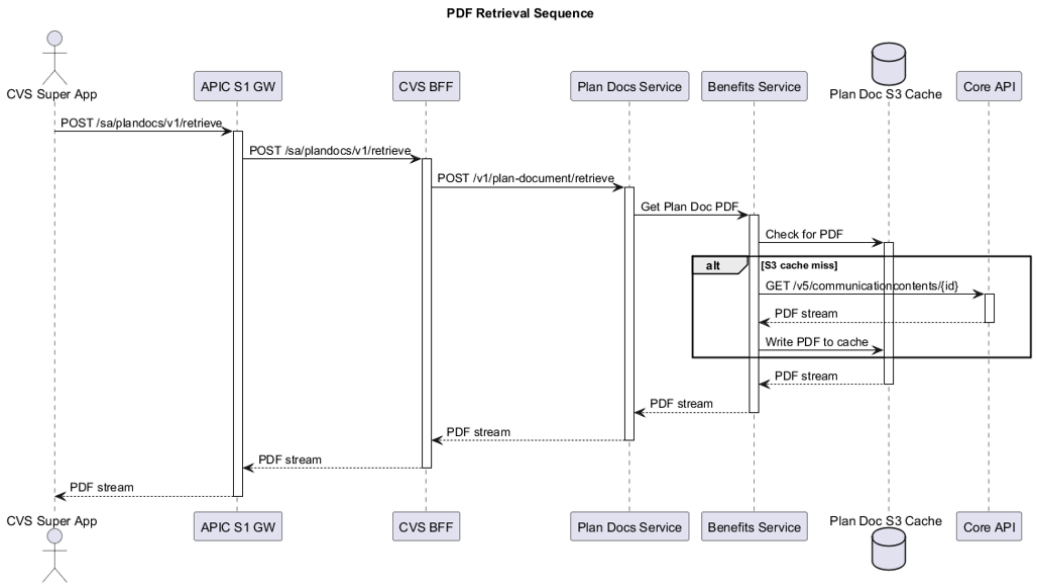
On a cache miss, the Benefits Service calls the Core API (

/eieheader/v5/communicationcontents/{id}) to fetch the document.

The service performs a

0-byte file check; if the retrieved file is empty, it returns an error and does not cache it.

If valid, the document is written to the S3 cache for future requests and then streamed back to the client.



List Retrieval Narrative

The OpenPlatform Plan Documents BFF service acts as an intermediary layer that processes data from the application startup response to generate multiple requests to the underlying Commercial/Medicare Plan Doc List service. This document describes the data extraction and transformation process.

- Service will call plangroups API
  - Service will call launch darkly to check on PSUIDs to ensure member is eligible to see Super App ID Cards Feature
  - Find portalGroups with LOB of either commercial or medicare
  - Filter plans who have digital access revoked (plan sponsor wave number 955 or 988)
    - within these plans - find policies with primary policy type as either Dental or Medical
      - Check memberships that have a status of Active or Dependant Active Coverage
      - The remaining portal groups are valid, so we loop through each portal group, returning all appropriate unique memberIds
- Service will loop through plangroups response
  - collect valid /plan-document-list/retrieve requests

"5~265106287+11+51+20190101+751133+BA+2"

"3~751133+BA+2"

"2025-05-30T00:00:00.000Z"

- dateAsOf will always be <now>
- Service will call /v1/plan-document-list/retrieve on Plan Docs Service
- Service will aggregate responses into reduced list for SA

"751133+BA+2"

"Plan Document (Spanish)"

"70023~13910995"

"/sa/plandocs/v1/retrieve/70023~13910995"

"HHL Not signed"

If the /plan-document-list/retrieve indicates a PlanDocsMsgAvail status the status will be returned to the caller unchanged in content in the readCommunicationResponse field.

Aggregated response returned to SA caller

Example Return for existing plan-document-list/retrieve

"documentId = resourceId

“documentName” = documentType + “-“ + description

“documentUrl” = preloaded call to /sa/plandocs/v1/retrieve with document Id loaded

“planId” = policyResourceId (passed in), stripped of cache prefix (example: policyResourceId = 3~751133+BA+2 / planId = 751133+BA+2)

Sample Integrated Response:

"751133+BA+2"

"Medical-Benefit Plan Description"

"70023~13910995"

"/sa/plandocs/v1/retrieve/70023~13910995"

"751133+BA+2"

"Medical-Plan Update”,

"70023~13910996"

"/sa/plandocs/v1/retrieve/70023~139109956”

""”

HHL Not Signed Sample Response

"HHL Not signed"

List Retrieval - Source Data Structure

The source data comes from the application startup response (`/aetnahealth/applicationstartup`) with the following structure:

```
readApplicationStartup
├── proxyResourceId
└── portalGroups[]
    └── policies[]
        ├── primaryPolicyType
        └── memberships[]
            ├── membershipResourceId
            ├── policyResourceId
            ├── status
            ├── relationshipToSubscriber
            ├── effectiveDatetimeBegin
            ├── effectiveDatetimeEnd
            └── policyTypes[]
```

List Retrieval - Data Extraction Process

The BFF service performs the following steps:

1. **Parse Application Startup Response**
   - Receives the complete application startup response
   - Validates the response structure
   - Extracts the readApplicationStartup object
2. **Iterate Through Portal Groups**
   - Loops through each portal group in the response

- Each portal group represents a different plan sponsor or employer

3. **Process Policies Within Each Portal Group**
   - For each portal group, iterates through all policies
   - Policies can include: Medical, Dental, Vision, Hearing, Pharmacy
   - Each policy has a primaryPolicyType indicating the coverage type
4. **Extract Memberships from Each Policy**
   - For each policy, processes all memberships
   - Each membership represents an active coverage relationship
   - Only "Actively Covered" memberships are processed
5. **Generate Underlying Service Requests**
   - For each valid membership, creates a request to the underlying service
   - Target endpoint: POST `/plan-document-list/retrieve`
   - One request per membership

List Retrieval - Data Mapping

The following fields are extracted and mapped:

Source Field (Application Startup)	Target Field (Underlying Service)
membership.membershipResourceId	membershipResourceId
membership.policyResourceId	policyResourceId
Current date (generated)	dateAsOf

List Retrieval - Example Data Transformation

Input (from application startup response):

```
{
  "readApplicationStartup": {
    "portalGroups": [
```

```
{
  "policies": [
    {
      "primaryPolicyType": "Medical",
      "memberships": [
        {
          "membershipResourceId":
"5~265416171+10+1+20230101+805233+A+1",
          "policyResourceId": "3~805233+A+1",
          "status": "Actively Covered",
          "relationshipToSubscriber": "Self"
        }
      ]
    },
    {
      "primaryPolicyType": "Dental",
      "memberships": [
        {
          "membershipResourceId":
"5~265416171+10+2+20230101+805233+B+2",
          "policyResourceId": "3~805233+B+2",
          "status": "Actively Covered",
          "relationshipToSubscriber": "Self"
        }
      ]
    }
  ]
}
```

```
  ]
 }
}
```

Generated Underlying Service Requests:

```
// Request 1 - Medical Policy
{
 "membershipResourceId":
"5~265416171+10+1+20230101+805233+A+1",
 "policyResourceId": "3~805233+A+1",
 "dateAsOf": "2025-05-30T00:00:00.000Z"
}

// Request 2 - Dental Policy
{
 "membershipResourceId":
"5~265416171+10+2+20230101+805233+B+2",
 "policyResourceId": "3~805233+B+2",
 "dateAsOf": "2025-05-30T00:00:00.000Z"
}
```

List Retrieval - Processing Logic

1. **Filtering Criteria**
   - Only process memberships with status = "Actively Covered"
   - Skip memberships with future effective dates
   - Skip expired memberships

2. **Date Handling**
   - dateAsOf is set to the current date in ISO format
   - Format: YYYY-MM-DDTHH:mm:ss.sssZ
   - Example: "2025-05-30T00:00:00.000Z"

3. **Error Handling**
   - If application startup response is invalid, return 400 error
   - If no valid memberships found, return empty data array
   - If underlying service calls fail, aggregate errors and return 500

4. **Response Aggregation**
   - Collect responses from all underlying service calls
   - Merge document lists from multiple memberships
   - Remove duplicates based on documentId
   - Sort by documentName for consistent ordering

List Retrieval - Business Rules

1. **Coverage Types Processed**
   - Medical policies
   - Dental policies
   - Vision policies
   - Hearing policies
   - Pharmacy policies

2. **Membership Status Requirements**
   - Must be "Actively Covered"
   - Must have valid effective date range

- Must not be expired

3. **Document Deduplication**
   - Documents with same documentId are considered duplicates
   - First occurrence is kept, subsequent duplicates are removed
   - Maintains document metadata from the first occurrence

4. **Error Propagation**
   - Individual underlying service failures don't fail the entire request
   - Partial results are returned with error information
   - Special messaging indicates partial failures

List Retrieval - Performance Considerations

1. **Parallel Processing**
   - Multiple underlying service calls can be made in parallel
   - Improves response time for users with multiple memberships
   - Configurable concurrency limits

2. **Caching Strategy**
   - Application startup data can be cached for short periods
   - Reduces load on the application startup service
   - Cache invalidation on membership changes

3. **Timeout Handling**
   - Individual underlying service calls have timeouts
   - Overall request timeout is longer than individual timeouts

- Graceful degradation when some calls timeout

List Retrieval - Security Considerations

1. **Data Validation**
   - All extracted data is validated before use
   - Prevents injection attacks through malformed data
   - Sanitizes all output data

2. **Access Control**
   - User context from id_token determines access permissions
   - Only memberships belonging to the authenticated user are processed
   - Audit logging for all data access

3. **Token Propagation**
   - Bearer token is propagated to underlying services
   - Maintains security context throughout the call chain
   - Proper error handling for authentication failures

List Retrieval - Monitoring and Observability

1. **Metrics Collected**
   - Number of memberships processed
   - Number of underlying service calls made
   - Response times for individual calls
   - Error rates and types

2. **Logging**

   - Request correlation IDs for tracing

   - Membership processing details

   - Underlying service call results

   - Error details for debugging

3. **Health Checks**

   - Underlying service availability

   - Application startup service health

   - Overall BFF service health

This processing approach allows the BFF to provide a unified interface for plan document retrieval while handling the complexity of multiple memberships and underlying service orchestration using the existing AWS framework.

PDF Retrieval

PDF Retrieval

The client requests a specific document via the BFF, providing the ocumentId (which maps to communicationContentResourceId). The BFF calls the Benefits Service

/plan-document/retrieve endpoint. This service first attempts to get the PDF from an S3 bucket. If it's a cache miss, the service retrieves the document from the core API, stores it in S3 for subsequent requests, and returns it.

.

Field Mapping

Response Key	Mapping
data[].documentId	readHealthPoliciesCommunicationContentsResponse.readCommunicationContents/communicationCont

	ent[].communicationContentIdentifier"[].resourceId
data[].documentName	readHealthPoliciesCommunicationContentsResponse.readCommunicationContents/communicationContent[].communicationContentIdentifier"[].documentType + "-" + readHealthPoliciesCommunicationContentsResponse.readCommunicationContents/communicationContent[].communicationContentIdentifier"[].description
data[].documentUrl	"/sa/plandocs/v1/retrieve/" + readHealthPoliciesCommunicationContentsResponse.readCommunicationContents/communicationContent[].communicationContentIdentifier"[].resourceId
data[].planId	<request>.policyResourceId (stripped of cache prefix)

Orchestration Diagram (Conceptual)

(For IA work/EDB Service Updates Only) This work involves creating a new BFF but leverages existing EDB service updates. The orchestration will be:

OpenPlatform → OpenPlatform BFF OpenPlatform BFF → AWS APIc → Benefits Service Benefits Service → Redis Cache (for list) Benefits Service → S3 Bucket (for PDF) Benefits Service → Core Plan Doc API (on cache miss)

APIs/Swagger

[ah-openplatform-plandocs]

A new API contract will be defined for the OpenPlatform BFF based on the specific needs of the OpenPlatform, aligning with the data required from the existing native client APIs.

Overview

The **OpenPlatform Plan Documents BFF (Backend for Frontend)** service provides a streamlined interface for retrieving plan documents and their associated metadata within the Aetna OpenPlatform ecosystem. This service acts as an intermediary layer that simplifies access to plan documents by abstracting the complexity of underlying backend systems.

Service Details

- **Title**: OpenPlatform - Plan Documents BFF
- **Version**: 1.0.4
- **Description**: Provides plan document list and PDF retrieval for the OpenPlatform
- **Base Path**: /sa/plandocs/v1

Authentication & Security

The service implements a dual authentication mechanism to ensure secure access to plan documents:

1. Bearer Token Authentication

- **Type**: HTTP Bearer authentication
- **Format**: JWT (JSON Web Token)
- **Header**: Authorization: Bearer <jwt_token>
- **Purpose**: Primary authentication mechanism for API access

2. ID Token Header Authentication

- **Type**: Custom header parameter
- **Header Name**: id_token
- **Format**: JWT containing user context and authorization information
- **Required**: Yes
- **Purpose**: Provides detailed user context and authorization information

ID Token Payload Structure

The id_token contains comprehensive user information including:

```
{
  "ae_version": "1.1.0",
  "iss": "https://openid.aetna.com/consumer",
  "sub": "031RCS6NTYHFKX25DE89@aetnae.com",
  "aud": "89d435af-00b0-4cb6-9a35-f3189f5adc55",
  "exp": 1753561541,
  "iat": 1753557941,
  "given_name": "BAILEY",
  "family_name": "SCHEUERMAN",
  "acr": "http://consumer.aetna.com/assurance/loa-2",
  "ae_dgn": "CN=DMT-S-
W265416171,OU=Members,OU=External,DC=aetheq,DC=aetnaeq,DC=com",
  "ae_hcr": "nextGenMember",
  "ae_accountId": "1~DMT-S-W265416171",
  "ae_busIndID": [
    "globalIdentifier",
    "60005~68035689373764332l2",
    "preferredProxyId",
    "15~QS3YXBBBHPXZ"
  ],
  "ae_impAUD": "",
  "ae_impHCR": "",
  "ae_impACR": "",
  "ae_impDGN": "",
  "ae_impAccountId": "",
  "ae_impBusIndID": [],
  "ae_impGrantedLOA": ""
}
```


Key Token Fields:

- ae_version: Aetna Enterprise version
- sub: Subject identifier (user ID)
- ae_hcr: Health Care Role (e.g., “nextGenMember”)
- ae_accountId: Account identifier
- ae_busIndID: Business individual identifiers

- exp: Token expiration time
- iat: Token issued at time

API Endpoints

1. Plan Document List Retrieval

Endpoint: POST /sa/plandocs/v1/list

Purpose: Retrieves a comprehensive list of available plan documents for a specific user and membership.

Authentication Requirements

- **Bearer Token**: Required in Authorization header
- **ID Token**: Required in id_token header

Request Headers

Authorization: Bearer <jwt_token>
id_token: <jwt_token_with_user_context>
Content-Type: application/json

Response Structure

Success Response (200):


```
{
  "data": [
    {
      "planId": "751133+BA+2",
      "documentName": "Plan Document (Spanish)",
      "documentId": "70023~13910995",
      "documentUrl": "/sa/plandocs/v1/retrieve/70023~13910995"
    },
    {
      "planId": "751133+BA+2",
      "documentName": "Summary of Benefits and Coverage",
      "documentId": "70023~13910996",
      "documentUrl": "/sa/plandocs/v1/retrieve/70023~13910996"
    }
  ],
```

```
  "readCommunicationsResponse": "HHL Not signed"
}
```


Response Fields:

- data: Array of simplified document objects
- readCommunicationsResponse: Special messaging for the response

Document Object Fields:

- planId: Plan identifier
- documentName: Name of the document
- documentId: Unique document identifier
- documentUrl: Internal URL to retrieve the PDF via BFF (includes documentId)

readCommunicationsResponse Valid Values:

- null: Request succeeded (normal case)
- "No docs available": No PlanDocsMsgAvail but no docs returned
- "Your employer can provide these documents to you": Special messaging for employer-provided documents
- "HHL Not signed": Special messaging for HHL not signed scenario

Error Responses

Bad Request (400):


```
{
  "httpCode": "400",
  "httpMessage": "Bad Request",
  "moreInformation": "Invalid request parameters."
}
```


Unauthorized (401):


```
{
  "httpCode": "401",
  "httpMessage": "Unauthorized",
  "moreInformation": "Invalid or missing authentication."
}
```


Forbidden (403):

```
{
  "httpCode": "403",
  "httpMessage": "Forbidden",
  "moreInformation": "Insufficient permissions to access plan documents."
}
```


Internal Server Error (500):


```
{
  "httpCode": "500",
  "httpMessage": "Internal Server Error",
  "moreInformation": "An unexpected error occurred while processing the request."
}
```


2. Plan Document PDF Retrieval

Endpoint: POST /sa/plandocs/v1/retrieve

Purpose: Retrieves a specific plan document PDF file based on the document identifier.

Request Structure


```
{
  "documentId": "70023~13910995"
}
```


Required Fields:

documentId: Corresponds to communicationContentResourceId in the backend (communicationContentIdentifier.resourceId)

Response Structure

Success Response (200):

- **Content-Type**: application/pdf
- **Body**: Binary PDF file content

Error Responses

Bad Request (400):


```
{
  "httpCode": "400",
  "httpMessage": "Bad Request",
```

```
  "moreInformation": "Invalid documentId provided."
}
```


Internal Server Error (500):


```
{
  "httpCode": "500",
  "httpMessage": "Internal Server Error",
  "moreInformation": "Failed to retrieve the requested document."
}
```


Data Models

PlanDocListResponse

Simplified response containing an array of document objects and optional special messaging.

PlanDocRetrieveRequest

Simple request containing the document identifier for PDF retrieval.

Error

Standard error response structure with HTTP code, message, and additional information.

IdTokenPayload

Comprehensive JWT payload structure containing user context, authentication information, and authorization details.

Use Cases

1. Member Portal Integration

Members can view available plan documents in their portal, with the ability to download PDFs directly. The simplified response structure makes it easy to display document lists.

2. Mobile Application Support

Mobile apps can display document lists and provide PDF viewing capabilities with minimal data transfer.

3. Customer Service Tools

Customer service representatives can access and share plan documents with members using the dual authentication system.

4. Compliance and Regulatory Requirements

Ensures proper access to required plan documents for regulatory compliance with comprehensive audit trails.

Integration Patterns

Frontend Integration with Authentication

```
// Example: Fetching plan document list with dual authentication
const response = await fetch('/sa/plandocs/v1/list', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer ' + bearerToken,
    'id_token': idToken
  }
});

const documents = await response.json();

// Handle special messaging
if (documents.readCommunicationsResponse) {
  console.log('Special message:', documents.readCommunicationsResponse);
}

// Display documents
documents.data.forEach(doc => {
  console.log(`${doc.documentName} (${doc.documentId})`);
});
```

PDF Download

```
// Example: Downloading a specific document
const response = await fetch('/sa/plandocs/v1/retrieve', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer ' + bearerToken
```

```
  },
  body: JSON.stringify({
    documentId: '70023~13910995'
  })
});

if (response.ok) {
  const pdfBlob = await response.blob();
  const url = URL.createObjectURL(pdfBlob);
  window.open(url);
} else {
  const error = await response.json();
  console.error('Download failed:', error.moreInformation);
}
```

Error Handling

```
// Comprehensive error handling
async function fetchPlanDocuments() {
  try {
    const response = await fetch('/sa/plandocs/v1/list', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': 'Bearer ' + bearerToken,
        'id_token': idToken
      }
    });

    if (response.status === 401) {
      // Handle authentication failure
      console.error('Authentication failed - please log in again');
      return;
    }

    if (response.status === 403) {
```

```
      // Handle authorization failure
      console.error('Insufficient permissions to access plan documents');
      return;
    }

    if (!response.ok) {
      const error = await response.json();
      console.error('Request failed:', error.moreInformation);
      return;
    }

    const documents = await response.json();
    return documents;
  } catch (error) {
    console.error('Network error:', error);
  }
}
```

Security Considerations

Authentication Layers

- **Bearer Token**: Primary authentication for API access
- **ID Token**: Provides detailed user context and authorization information
- **Token Validation**: Both tokens are validated on every request

Authorization

- Document access is restricted based on user membership and roles
- Health Care Role (ae_hcr) determines access permissions
- Account ID (ae_accountId) ensures proper data isolation

Data Protection

- Sensitive plan information is protected through proper authorization
- Audit trails are maintained for document access
- Token expiration ensures secure session management

Performance Considerations

Simplified Response Structure

- Reduced payload size compared to previous versions
- Faster parsing and rendering in frontend applications
- Optimized for mobile and web applications

Caching Strategy

- Document lists can be cached based on user context
- PDF retrieval is optimized for large file handling
- Token caching reduces authentication overhead

Error Handling Strategy

The service provides comprehensive error handling with specific error codes:

- **400 Bad Request**: Invalid input parameters or missing required fields
- **401 Unauthorized**: Invalid or missing authentication tokens
- **403 Forbidden**: Insufficient permissions to access requested resources
- **500 Internal Server Error**: Unexpected server-side errors

Future Enhancements

- Support for document search and filtering
- Pagination for large document lists
- Document preview capabilities
- Multi-language document support
- Document versioning and history tracking
- Enhanced caching strategies
- Real-time document availability updates

NFR (Non-Functional Requirements)

Performance: The solution must not degrade application performance. Caching at the Benefits Service (TTL 1 hour for lists) and S3 layer (for PDFs) will be leveraged to ensure fast responses that meet the targets defined in the

Response Time and Latency Requirements section.

Rate Limiting: The new BFF endpoints must adhere to the existing backend service rate limits to ensure system stability.

- **Plan Document List:** 950 requests per minute.
- **Plan Document PDF:** 4100 requests per minute.

Scalability: The architecture reuses existing, horizontally scalable AWS services (e.g., API Gateway, Lambda, S3) to handle anticipated growth in OpenPlatform user traffic.

Observability: The new OpenPlatform BFF will have comprehensive logging, monitoring, and alerting integrated with existing APM tools to provide visibility into performance, errors, and traffic patterns.

Query successful

Of course. Here are the requested updates to the NFR, Services, and User Stories sections of the document.

Updated NFR (Non-Functional Requirements) Section

This updated NFR section is more concise and directly references the performance metrics defined in the new *Performance Requirements and Testing* section.

NFR (Non-Functional Requirements)

Performance: The solution must not degrade application performance. Caching at the Benefits Service (TTL 1 hour for lists) and S3 layer (for PDFs) will be leveraged to ensure fast responses that meet the targets defined in the

Response Time and Latency Requirements section.

- **Rate Limiting:** The new BFF endpoints must adhere to the existing backend service rate limits to ensure system stability.
  - **Plan Document List:** 950 requests per minute.
  - **Plan Document PDF:** 2333 requests per minute.
- **Scalability:** The architecture reuses existing, horizontally scalable AWS services (e.g., API Gateway, Lambda, S3) to handle anticipated growth in OpenPlatform user traffic.
- **Observability:** The new OpenPlatform BFF will have comprehensive logging, monitoring, and alerting integrated with existing APM tools to provide visibility into performance, errors, and traffic patterns.

Commentary on the "Services" Section

The short "Services" section immediately following the NFR section is **not necessary** and can be removed.

Reasoning: All the components listed in that section (OpenPlatform BFF, Features API, Benefits Service, etc.) are already defined and explained in much greater detail throughout the document, particularly in the *Solution Sketches*, *Service Logic*, and *Appendices* sections. Removing it will reduce redundancy and improve the document's flow.

Key User and Service Stories

User-Facing Stories (Frontend)

- **Story:** As an OpenPlatform user, when I navigate to the Plan Documents section, I want the system to first check my eligibility and for any special messages so I get the correct information immediately.
  - **Acceptance Criteria:** The client calls the Features API with the membershipResourceId. If PlanDocsMsgAvail is enabled or no relevant document type flags (MedPlanDocs, etc.) are enabled, the client displays a message and does not call the BFF.
- **Story:** As an OpenPlatform user, I want to see a clear message if my documents aren't available due to a missing Hold Harmless Letter (HHL) instead of an empty list.
  - **Acceptance Criteria:** A user-friendly message for the "HHL Not signed" scenario is displayed, triggered either by the Features API pre-check or the readCommunicationsResponse field from the BFF.
- **Story:** As an OpenPlatform user, if I have multiple plan documents, I want to see them presented in a clear list so I can choose which one to view.
  - **Acceptance Criteria:** The client displays a list showing the documentName for each item. If a document is in a language other than English, the language is indicated (e.g., "Plan Document (Spanish)"). Tapping an item opens its PDF.
- **Story:** As an OpenPlatform user, if I have only one document available, I want it to open directly when I tap the link to save a step.
  - **Acceptance Criteria:** When the BFF returns a list with a single document, the client UI presents a direct link that opens the PDF viewer immediately upon being tapped.

- **Story:** As an OpenPlatform user, if no documents are available for me, I want to see a clear message stating that so I am not left with a blank screen.
  - **Acceptance Criteria:** The client gracefully handles a response with an empty data array by displaying a user-friendly message.

Service Stories (Backend)

- **Story:** As an OpenPlatform BFF, I want to accept a POST request to /sa/plandocs/v1/list that includes a **Bearer token** and an **id_token** to securely fetch a user's document list.
  - **Acceptance Criteria:** The BFF validates both tokens before processing. It uses the user context from the

    id_token to process the application startup response and generate requests for the backend Benefits Service.

Acceptance Criteria: The BFF correctly transforms the detailed backend response into the simplified OpenPlatform format, creating fields like documentName and documentUrl. It also forwards any

    readCommunicationsResponse message, such as "HHL Not signed".

- **Story:** As an OpenPlatform BFF, I want to accept a POST request to /sa/plandocs/v1/retrieve to securely stream a specific PDF document to the client.
  - **Acceptance Criteria:** The BFF calls the backend Benefits Service with the documentId (mapping to communicationContentResourceId).
  - **Acceptance Criteria:** The BFF correctly streams the PDF content from the backend to the client with an application/pdf content type.
- **Story:** As the existing Benefits Service, I want to use Redis to cache plan document lists and S3 to cache PDFs to reduce load on core systems and improve response times.
  - **Acceptance Criteria:** Plan document lists are cached in Redis with a 1-hour TTL using a composite key that accounts for different lines of business.
  - **Acceptance Criteria:** PDFs are cached in S3 using the communicationContentResourceId as the key. A check is performed to prevent 0-byte files from being cached

Field Mapping

- **Client Request to BFF (POST /v1/cvs/plan-documents/list):**
  - policyResourceId: From application startup API response, corresponding to the selected plan's primary policy type (Medical, Dental, or Vision).

    - membershipResourceId: From application startup API response for the selected membership, associated with enabled feature codes.
    - dateAsOf: YYYY-MM-DD format. Today's date for Actively Covered. For Pending Coverage (Medicare only), use coverageDatetimeBegin from app startup.
- **BFF Response (PlanDocListResponse) to Client:**
    - data[].documentName: Derived by BFF. If description is available, use it. If language is available and not 'ENG', append (language) to the name. If description is missing, try to use title.
    - data[].documentId: Maps to communicationContentIdentifier.resourceId from Benefits Service response.
    - data[].documentUrl: Generated by BFF to point to its /v1/cvs/plan-documents/retrieve endpoint.
    - data[].documentType, data[].description, data[].title, data[].format, data[].language, data[].contentSize: Directly mapped from Benefits Service response if available.
    - data[].planId: Maps from policyResourceId related to the document.
    - reasonCode: Directly mapped from Benefits Service response if present.
- **Client Request to BFF for PDF (POST /v1/cvs/plan-documents/retrieve):**
    - documentId: Value obtained from data[].documentId in the Plan Doc List response.

Any differences between platforms

The logic and documentation must be consistent across Aetna and CVS implementations, and by extension, across Web, iOS, and Android platforms.

Scenarios

- **User navigates to Plan Docs:**
    - **Scenario 1: No PlanDoc features enabled for any membership:** Client displays "No documents available" message immediately (triggered by Features API).
    - **Scenario 2: PlanDocsMsgAvail feature is enabled (e.g., HHL not signed externally):** Client displays "Special Messaging" UI immediately (triggered by Features API), bypassing BFF call.
    - **Scenario 3: PlanDocsMsgAvail is NOT enabled, but HHL Not signed reason code is returned by Benefits Service:** Client displays "Special Messaging" UI for HHL (triggered by BFF response).
- **Single Document**: User has exactly one plan document available (and no special messages). Tapping the link opens the PDF directly.

- **Multiple Documents**: User has more than one document available (and no special messages). A list is displayed showing the title and description (if available).
  - **Sub-scenario: Multi-language documents:** List displays language in parentheses.
- **Zero Documents**: User has no documents available (and no special messages) after BFF call returns an empty list. The UI must handle this case clearly.
- **Missing Metadata**: A document is returned from the service without a description or title for the document name field. The UI should use available data or hide these fields gracefully.
- **Plan Type/Coverage Specifics (handled by client selecting correct policyResourceId/membershipResourceId):**
  - **Medicare Active, Pending different membershipIds:** Client makes separate calls for active and pending coverage using appropriate dateAsOf and membershipResourceId values.
  - **Medicare Active and Pending same membershipId.**
  - **IFP Active.**
  - **Commercial Active Medical, Dental, Vision.**
  - **Commercial Dependents Only Active Medical, Dental, Vision.**
  - **Commercial with both Active and Dependents Only Active for Dental** (Adult Dental/Pediatric Dental scenario).
  - **No plan docs enabled for any membership.**
- **PDF Retrieval:**
  - Successful retrieval and display of PDF.
  - Error during PDF retrieval (e.g., 0-byte PDF, network error) gracefully handled.

Deployment Strategy (Brief)

The new OpenPlatform BFF will be deployed as a microservice on the AWS platform, leveraging existing CI/CD pipelines and infrastructure-as-code practices. Deployments will be phased, starting with development and QA environments, followed by production.

Rollback Plan (Brief)

In the event of critical issues post-deployment, a rollback to the previous stable version of the OpenPlatform BFF will be initiated using automated deployment tools. The stateless nature of the BFF (with caching externalized) facilitates quicker rollbacks.

Security Review

A security review should be conducted, particularly as a new BFF is being introduced as an entry point into the AWS ecosystem. While the backend service patterns are being reused, the BFF's exposure, rate limiting, and authentication/authorization mechanisms need to be validated by the security team. This includes reviewing API security best practices, data encryption in transit and at rest, and access controls for the new BFF endpoints.

Testing

The solution should be tested to ensure all scenarios (zero, single, multiple documents) are handled correctly on the client and in the BFF. Testing should verify that missing metadata is handled gracefully and does not cause UI errors.

Detailed Testing Plan

To ensure robust functionality and a seamless user experience, the following detailed testing plan will be executed:

Test Data Preparation:

- **User Profiles:** Create a diverse set of test users covering various scenarios:
  - **Zero Documents:** Users with no plan documents available (verify both Features API and BFF response paths).
  - **Single Document:** Users with exactly one plan document.
  - **Multiple Documents:** Users with several plan documents.
  - **Multi-language Users:** Users with documents available in different languages (e.g., English, Spanish), ensuring language field is correct.
  - **Users with Missing Metadata:** Users whose documents are known to have missing description or title fields from the upstream service.
- **Plan Types & Coverage:**
  - **Medical-only:** Users covered only under a medical plan.
  - **Dental-only:** Users covered only under a dental plan.
  - **Vision-only:** Users covered only under a vision plan.
  - **Combined Plans:** Users with combinations (e.g., Medical + Dental, Medical + Vision, Medical + Dental + Vision), verifying correct policyResourceId and membershipResourceId selection by client.
  - **Varying Family Coverages:** Create scenarios where different members under the same plan have varied coverages:

        - **Example 1:** Plan holder has Medical, Dental, and Vision. Dependent children only have Medical and Dental (no Vision).
        - **Example 2:** Plan holder has Medical. Spouse has Medical and Dental.
        - **Example 3:** Different enrollment dates or policy periods impacting document availability for various family members.
        - Verify correct membershipResourceId and policyResourceId are used for each family member.
    - **HHL Not Signed Scenarios:** Test users associated with self-insured plans where the Hold Harmless Letter (HHL) has not been signed.
        - Verify client displays message when PlanDocsMsgAvail is enabled.
        - Verify client displays message when Benefits Service returns reasonCode: "HHL Not signed".

Functional Testing:

- **End-to-End Flow:**
    - Verify the complete user journey from navigating to the Plan Documents section to viewing/downloading a PDF.
    - Confirm correct routing from the OpenPlatform client **through the Features API pre-check**, then through the new BFF to the existing backend services.
- **Scenario Validation:**
    - **Features API Pre-check:**
        - Verify client correctly evaluates PlanDocsMsgAvail.
        - Verify client correctly evaluates DenPlanDocs, MedPlanDocs, VisionPlanDocs to determine if BFF call should proceed.
        - Confirm "Special Messaging" UI is displayed immediately when PlanDocsMsgAvail is enabled or no PlanDoc features are enabled.
    - **Zero Documents:** Verify the UI clearly communicates that no documents are available (after BFF call returns empty list).
    - **Single Document:** Confirm direct PDF opening behavior and proper PDF viewer display.
    - **Multiple Documents:** Validate accurate display of the document list, including names (derived from description/title and language) and descriptions (if present).
    - **Language Display:** Verify that language (e.g., "(Spanish)") is correctly appended to the document name in the list, based on the language field in the Benefits Service response.
    - **Missing Metadata Handling:** Confirm that description and title fields are handled gracefully (e.g., description is hidden if not provided; title used as fallback for documentName).

    - **PDF Actions:** Test "Share" and "Print" functionalities for PDF documents.
    - **reasonCode Handling:** Verify that if reasonCode: "HHL Not signed" is returned by the Benefits Service (via BFF), the client displays the appropriate special messaging, overriding any document display.
- **Error Handling:**
    - Test various API error responses (e.g., 400 Bad Request, 403 Forbidden, 404 Not Found, 500 Internal Server Error, 503 Service Unavailable, 504 Timeout) from the Benefits Service and ensure the BFF translates and the client displays user-friendly messages.
    - Verify that 0-byte PDFs from core APIs are correctly handled (not cached, appropriate error returned).

Performance Testing:

- **Latency Impact:** Measure response times for both list and PDF retrieval through the new BFF.
- **Scalability:** Conduct load tests to ensure the BFF and backend services can handle anticipated OpenPlatform traffic volumes without degradation.
- **Caching Effectiveness:** Monitor cache hit/miss ratios for Redis and S3 to confirm caching mechanisms are reducing load on core services as expected.

Security Testing:

- Validate authentication and authorization mechanisms for the new BFF endpoints.
- Conduct penetration testing to identify vulnerabilities.
- Verify data encryption in transit (HTTPS) and at rest (for cached data in S3).

Contacts and Impacted Teams/People

- **CVS Contact**: Rohit Puri
- **Architecture**: Jenn Tang
- **Plan Doc Search**: Jesse Jackman
- **Product Management**: Darlene Scarola
- **AEI Team**: Wizards
- **Front-end Teams**: Wolf and Moonlight
- **Facets Contact**: Alan Chamberlain
- **Other Contacts**: Harshal Patel

PlanDocsMsgAvail

PlanDocsMsgAvail flag can be returned more than once if the memberships have different statuses. i.e:

PlanDocsMsgAvail: { status: "ENABLED", membershipResourceIds: ["A+2"] }
PlanDocsMsgAvail: { status: "N/A", membershipResourceIds: ["B+2"] }
PlanDocsMsgAvail: { status: "OFFLINE", membershipResourceIds: ["D+2"] }

So, on the Coverage page, we check if any of them are ENABLED. If they are, we check if the current membershipResourceId (for the selected policy) is listed in the membershipResourceIds array. If it is, the selfInsured content will be rendered.

The downside to this feature flag being returned like this from the features API is that it doesn't appear to be supported on the features page so we can't toggle the flag to test both scenarios – we now have to login with users that reflect each scenario as desired.

This can be seen with the qa3 user: QA3-S-SEBAGOALLCLAIMS

Open b80f6a7f-19c4-44b8-ae1d-8e023e2a6cf0.jpg

UNKNOWN ATTACHMENT

The screenshot on the right shows the Active/Medical membershipResourceId. Middle screenshot shows that the PlanDocsMsgAvail flag is ENABLED/ON with the medical membershipResourceId listed in the membershipResourceIds array. Left screenshot shows that the user has MedPlanDocs ON.

Service	Request Response	CVS	AETNA
List	**Request**	HEADER: **JWT id_token**	"5~265106287+11+51+20190101+751133+BA+2"

			"3~751133+BA+2" "2025-05-30T00:00:00.000Z"

	Response	{
		"data":[{
		"planId" : "",
		
		"documentName":"" ,
		
		"documentId":"",
		
		"documentUrl":"" //nice to have
		}]
		}
			"string"
			
			"string"
			
			"string"

"string"

"string"

"string"

"string"

"string"

"string"

"string"

"string"

"string"

"string"

			"string" "string" "string" "string"
Retrieve	**Request**	{ "documentNam e":"" , "documentId":"" , 	 "70023~13910 995"

		"documentUrl" :"" //nice to have }] }	
	Response	application/pdf stream	application/pdf stream

Service	Request Response	CVS	AETNA
List	**Request**	HEADER: **JWT id_token**	**"5~265106287+11+51+20190101+751133+BA+2"** **"3~751133+BA+2"** **"2025-05-30T00:00:00.000Z"**

	Response	{
		"data":[{
		"planId" : "",
		"documentNam e":"" ,
		"documentId":"" ,
		"documentUrl" :"" //nice to have
		}]
		}
			"string"
			"string"
			"string"
			"string"
			"string"
			"string"

"string"

"string"

"string"

"string"

"string"

"string"

"string"

			"string" "string" "string"

Retrieve	**Request**	{
		
		"documentNam e":"" ,
		
		"documentId":"" ,
		
		"documentUrl" :"" //nice to have
		}]
		}
			"70023~13910 995"

	Response	application/pdf stream	application/pdf stream

Performance Requirements and Testing

Service Call Chains

To create an accurate test plan, the cascading service calls for each user function must be understood.

List Retrieval Call Chain

1. **Client Pre-Check:** The client first calls the **Features API** to check for flags like PlanDocsMsgAvail to determine if a special message should be displayed immediately, bypassing the main document list call.
2. **Client to BFF:** The client sends a POST request to the new **OpenPlatform BFF** at /sa/plandocs/v1/list, including the JWT id_token for user context .
3. **BFF to Backend:** The BFF calls the existing **Plan Docs Service** via POST /v1/plan-document-list/retrieve to fetch the document list.
4. **Backend to Cache/Core:** The backend Benefits Service checks a **Redis cache** for the list. On a cache miss, it calls the

   Core API (/eie/header/v4/healthpolicies/communicationcontents/search/retrieve) and stores the result in Redis for one hour before returning it.

Document (PDF) Retrieval Call Chain

1. **Client to BFF:** The client sends a POST request to the **OpenPlatform BFF** at /sa/plandocs/v1/retrieve with a specific documentId .
2. **BFF to Backend:** The BFF calls the existing **Benefits Service** at its /plan-document/retrieve endpoint.
3. **Backend to Cache/Core:** The Benefits Service first checks an **S3 cache** for the PDF. On a cache miss, it retrieves the file from the

   Core API (/eie/header/v5/communicationcontents/{id}), stores it in S3, and then streams it back to the client.

Required Performance Metrics

This section outlines the performance targets the solution must meet. These metrics serve as the acceptance criteria for the test plan.

Volume and Throughput Requirements

The new BFF endpoints must support the existing backend service rate limits to ensure system stability.

- **Plan Document List (/sa/plandocs/v1/list):** 950 requests per minute.
- **Plan Document PDF (/sa/plandocs/v1/retrieve):** 2333 requests per minute.

Response Time and Latency Requirements

The solution must provide "fast responses" and not degrade application performance. Specific latency targets (e.g., 95th percentile response time) should be defined and filled in below.

- **List Retrieval (Warm Cache):** _____ ms
- **List Retrieval (Cold Cache):** _____ ms
- **PDF Retrieval (Warm Cache):** _____ ms
- **PDF Retrieval (Cold Cache):** _____ ms

Performance Test Plan and Results Grid

Narrative and Methodology

The objective of this performance test plan is to validate that the end-to-end plan documents solution is responsive, stable, and scalable against the defined NFRs. The methodology focuses on simulating real-world user traffic against the OpenPlatform BFF endpoints to measure the cumulative latency of all downstream calls.

To ensure comprehensive coverage, tests will be executed under two distinct cache conditions:

- **Cold Cache:** All caches (Redis, S3) are cleared before the test to measure the worst-case performance scenario.
- **Warm Cache:** Caches are fully populated before the test to measure the best-case performance and validate the caching strategy.

Testing will use a variety of user profiles to cover scenarios such as users with zero, one, or multiple documents.

Test Execution Grid

Test ID	User Action	API Endpoint	Test Type	Cache State	Scenario Description	Metric / Result

LT-LST-01	List Retrieval	POST /sa/plan docs/v1/ list	Latency	Cold	Single user requests a list of 10 documents. Cache is empty.	95th Percentile Latency (ms):
LT-LST-02	List Retrieval	POST /sa/plan docs/v1/ list	Latency	Warm	Single user requests a list of 10 documents. Cache is populated.	95th Percentile Latency (ms):
LD-LST-01	List Retrieval	POST /sa/plan docs/v1/ list	Load	Warm	Sustained load test with a mix of users (0, 1, and multiple docs).	Throughput (RPM): Error Rate (%):
LD-LST-02	List Retrieval	POST /sa/plan docs/v1/ list	Load	Warm	Stress test by increasing load beyond target RPM to	Max RPM @ <1% Error Rate:

					find the breaking point.	
LT-PDF-01	PDF Retrieval	POST /sa/plan docs/v1/ retrieve	Latency	Cold	Single user requests a 1MB PDF docume nt. Cache is empty.	95th Percentil e Latency (ms):
LT-PDF-02	PDF Retrieval	POST /sa/plan docs/v1/ retrieve	Latency	Warm	Single user requests a 1MB PDF docume nt. Cache is populat ed.	95th Percentil e Latency (ms):
LD-PDF-01	PDF Retrieval	POST /sa/plan docs/v1/ retrieve	Load	Warm	Sustaine d load test with requests for various cached PDF docume nts.	Through put (RPM): Error Rate (%):
LD-PDF-02	PDF Retrieval	POST /sa/plan	Load	Warm	Stress test by	Max RPM @

		docs/v1/retrieve			increasing load beyond target RPM to find the breaking point.	<1% Error

Appendices

The following appendices are extracted from the source architecture document for the existing AWS Benefits Service. Since the new OpenPlatform BFF service acts as a proxy that leverages this proven backend infrastructure, these sections are included here for completeness. This provides engineers and reviewers with a single, comprehensive view of the entire end-to-end service flow without needing to cross-reference multiple documents.

Source Documents

- **Plan Documents via AWS - Architecture:** <INSERT HYPERLINK HERE>
- **OpenPlatform Plan Docs BFF - OpenAPI Specification:** <INSERT HYPERLINK HERE>

Appendix A: AWS Plan Document List Service Logic

This section details the existing logic within the AWS Benefits Service when it receives a request for a plan document list.

- **Incoming Request:** The service expects a request payload containing:
  - policyResourceId
  - membershipResourceId
  - dateAsOf
- **Cache Existence Check:**
  - The service first determines the correct cache key based on the Line of Business (LOB).
  - **Commercial Group (5~) and IFP (59~ with IDT38):** The cache key is reusable across members and is based on dateAsOf + policyResourceId. The

membershipResourceId is not used for the cache key in this case.

Medicare (not IFP): The cache key is more complex to handle scenarios like ANOC. The service calls the Core-Proxy

/v1/memberships/{id} API to gather additional identifiers. The cache key is composed of:

- dateAsOf + policyResourceId + externalPlanId + classCode + priorAdjacentContractPBP (for Medicare Group)
- dateAsOf + policyResourceId + priorAdjacentContractPBP (for Medicare Individual)
- **Cache Miss Logic:**
  - If a list is not found in the Redis cache, the service calls the core API (

    /eieheader/v4/healthpolicies/communicationcontents/search/retrieve) to get the document list.

The successful response from the core API is then stored in the Redis cache with a

1-hour TTL before being returned to the caller.

- **HHL (Hold Harmless Letter) Scenario:**
  - If a self-insured plan sponsor has not signed an HHL, members cannot be shown their plan documents.
  - In this case, the service returns a

    reasonCode of "HHL Not signed" with an empty list of documents, and this specific response is cached.

Appendix B: AWS Plan Document PDF Retrieval Service Logic

This section details the existing logic for retrieving a specific PDF document.

- **Incoming Request:** The service expects the communicationContentResourceId in the request payload.
- **S3 Cache Check:**
  - The service first checks the configured plan documents S3 bucket to see if the PDF already exists, using the

    communicationContentResourceId as the key.

If the object exists, it is retrieved from S3 and streamed back to the caller.

- **Cache Miss Logic:**

    - If the PDF is not found in S3, the service calls the core API (

      /eieheader/v5/communicationcontents/{id}) to download the document.
- **0-Byte File Check:** After downloading, the service checks the byte size of the file. If it is 0 bytes, it returns an error and does not cache the empty file to prevent issues.
- If the file is valid, it is stored in the S3 bucket and then streamed back to the caller.

Appendix C: Feature Flag Logic and Client-Side Restrictions

The client application contains important pre-check logic that determines whether a call to the BFF is necessary.

- **Initial Features API Call:**
    - When a user navigates to the Plan Documents section, the client first calls the

      Features API using the user's membershipResourceId.
- **Decision Logic:**
    - The client evaluates the response for several feature codes, including

      DenPlanDocs, MedPlanDocs, VisionPlanDocs, and PlanDocsMsgAvail.
- **Bypass Scenario:** If the PlanDocsMsgAvail flag is ENABLED for that member, or if none of the specific PlanDoc type flags are enabled, the client **bypasses** the BFF call entirely. It immediately displays a special message to the user (e.g., "HHL Not signed" or "No documents available").
- **Proceed Scenario:** If PlanDocsMsgAvail is not enabled AND at least one of the relevant PlanDoc type flags is enabled, the client proceeds to call the OpenPlatform BFF to retrieve the document list.

Appendix D: Known Limitations and Architectural Considerations

This section outlines key architectural complexities and business rules from the underlying AWS services that the OpenPlatform solution inherits.

Medicare Cache Key Complexity: The derivation of the cache key for Medicare members is complex due to business rules around ANOC and requires a call to another core service to get additional identifiers (e.g., priorAdjacentContractPBP, externalPlanId). There is a known tactical alternative to use the

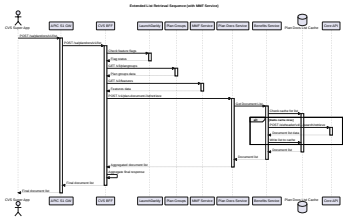
membershipResourceId in the key, but this reduces cache reusability across members and is not the preferred strategic approach.

- **Future Medicare Changes:** The Medicare cache key requirements are expected to change in the future with the potential introduction of a "Master Plan ID" that will need to be incorporated.
- **Hold Harmless Letter (HHL):** The availability of documents for members of self-insured plans is strictly dependent on whether the plan sponsor has signed an HHL. The backend services correctly handle this by returning a specific

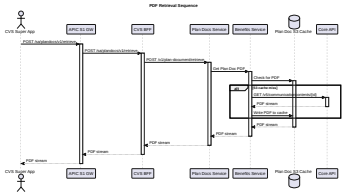
  reasonCode instead of a document list.

0-Byte File Handling: The PDF retrieval service includes a specific check to prevent the caching of empty (0-byte) files retrieved from core systems, which improves the resilience of the solution.

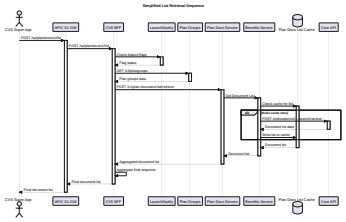
extended-list-r... ce.puml
30 Jul 2025, 06:46 PM

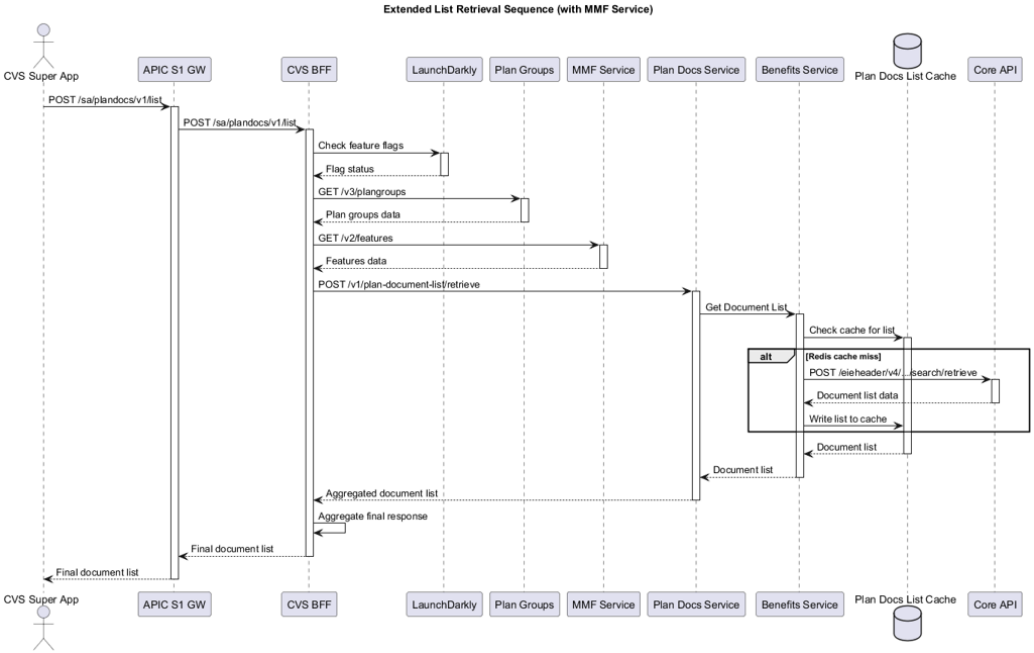


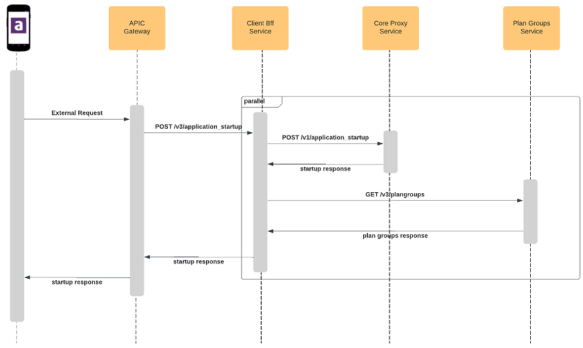
pdf-retrieval-s... ce.puml
30 Jul 2025, 06:46 PM

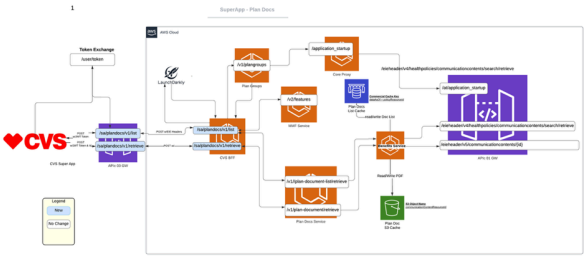


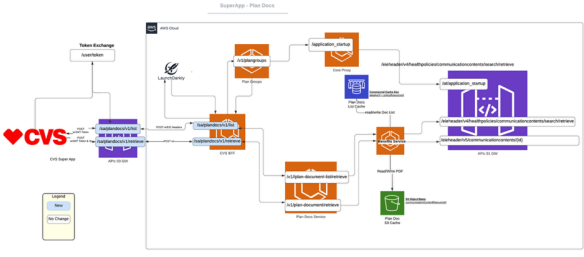
simplified-list-r...ne.puml
30 Jul 2025, 06:46 PM

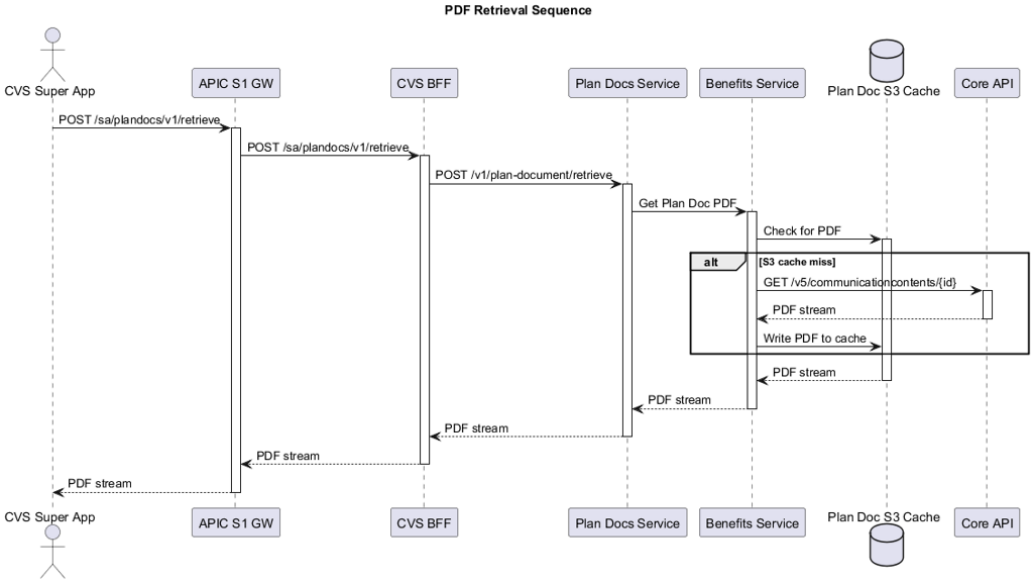


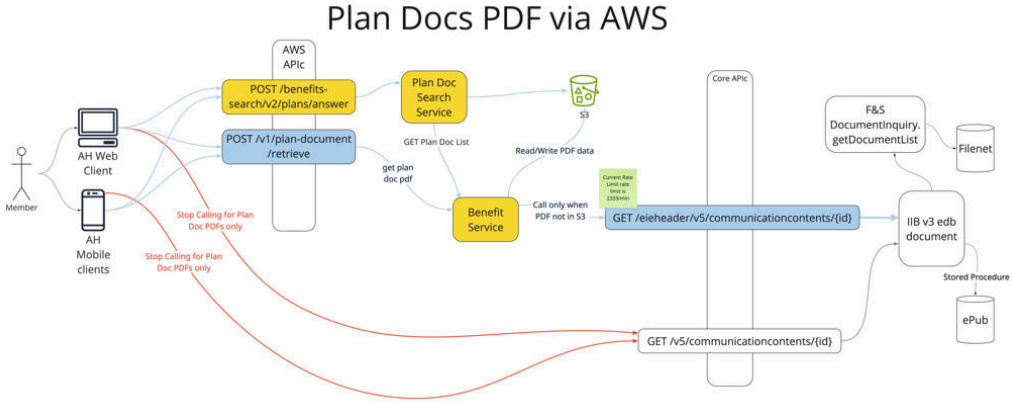

Extended List Retrieval Sequence (with MMF Service)
CVS Super App
APIC S1 GW
CVS BFF
LaunchDarkly
Plan Groups
MMF Service
Plan Docs Service
Benefits Service
Plan Docs List Cache
Core API
POST /sa/plandocs/v1/list
POST /sa/plandocs/v1/list
Check feature flags
Flag status
GET /v3/plangroups
Plan groups data
GET /v2/features
Features data
POST /v1/plan-document-list/retrieve
Get Document List
Check cache for list
alt
[Redis cache miss]
POST /eie/header/v4/.../search/retrieve
Document list data
Write list to cache
Document list
Document list
Aggregated document list
Aggregate final response
Final document list
Final document list

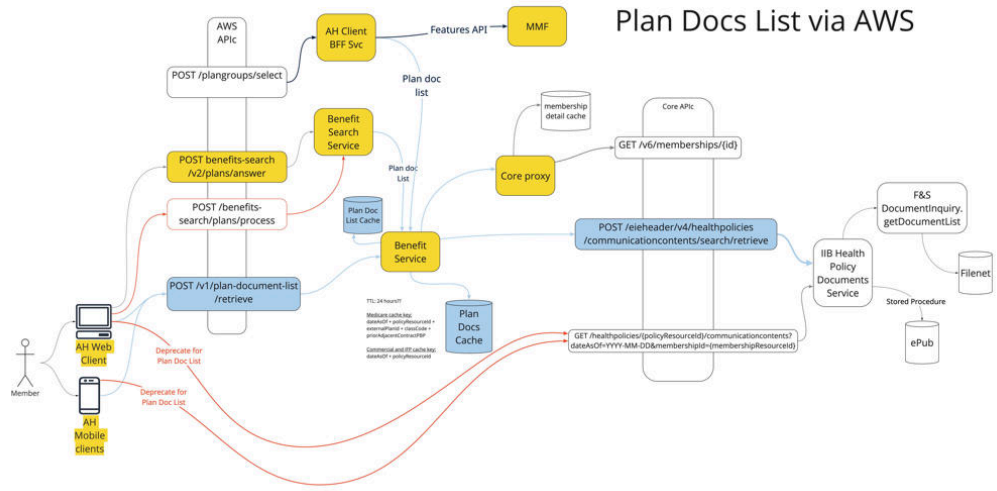

APIC Gateway
Client Bff Service
Core Proxy Service
Plan Groups Service
parallel
External Request
POST /v3/application_startup
POST /v3/application_startup
startup response
GET /v3/plangroups
plan groups response
startup response
startup response

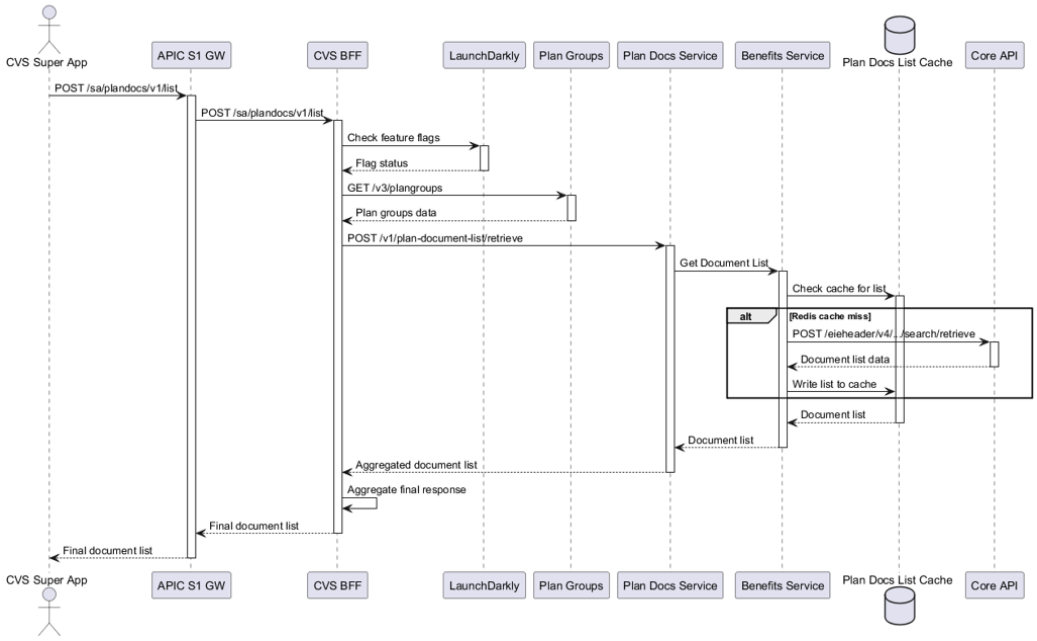





PDF Retrieval Sequence
CVS Super App
APIC S1 GW
CVS BFF
Plan Docs Service
Benefits Service
Plan Doc S3 Cache
Core API
POST /sa/plandocs/v1/retrieve
POST /sa/plandocs/v1/retrieve
POST /v1/plan-document/retrieve
Get Plan Doc PDF
Check for PDF
alt
[S3 cache miss]
GET /v5/communicationcontents/{id}
PDF stream
Write PDF to cache
PDF stream
PDF stream
PDF stream
PDF stream
PDF stream
CVS Super App
APIC S1 GW
CVS BFF
Plan Docs Service
Benefits Service
Plan Doc S3 Cache
Core API


Plan Docs PDF via AWS
AWS APIc
POST /benefits-search/v2/plans/answer
POST /v1/plan-document/retrieve
Plan Doc Search Service
S3
GET Plan Doc List
Read/Write PDF data
get plan doc pdf
Benefit Service
Call only when PDF not in S3
Core APIc
GET /eieheader/v5/communicationcontents/{id}
F&S DocumentInquiry. getDocumentList
Filenet
IIB v3 edb document
Stored Procedure
ePub
GET /v5/communicationcontents/{id}
Member
AH Web Client
AH Mobile clients
Stop Calling for Plan Doc PDFs only
Stop Calling for Plan Doc PDFs only


Plan Docs List via AWS
AWS APIc
AH Client BFF Svc
Features API
MMF
POST /plangroups/select
Plan doc list
membership detail cache
Core APIc
Benefit Search Service
GET /v6/memberships/{id}
POST benefits-search /v2/plans/answer
Plan doc List
Core proxy
POST /benefits-search/plans/process
Plan Doc List Cache
POST /eieheader/v4/healthpolicies /communicationcontents/search/retrieve
F&S DocumentInquiry. getDocumentList
Benefit Service
IIB Health Policy Documents Service
Filenet
POST /v1/plan-document-list /retrieve
Stored Procedure
Plan Docs Cache
ePub
GET /healthpolicies/{policyResourceId}/communicationcontents?dateAsOf=YYYY-MM-DD&membershipId={membershipResourceId}
Member
AH Web Client
AH Mobile clients
Deprecate for Plan Doc List
Deprecate for Plan Doc List


Simplified List Retrieval Sequence
CVS Super App
APIC S1 GW
CVS BFF
LaunchDarkly
Plan Groups
Plan Docs Service
Benefits Service
Plan Docs List Cache
Core API
POST /sa/plandocs/v1/list
POST /sa/plandocs/v1/list
Check feature flags
Flag status
GET /v3/plangroups
Plan groups data
POST /v1/plan-document-list/retrieve
Get Document List
Check cache for list
alt
[Redis cache miss]
POST /eieheader/v4/.../search/retrieve
Document list data
Write list to cache
Document list
Document list
Aggregated document list
Aggregate final response
Final document list
Final document list