OpenPlatform - Plan Documents Access

# 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.

A diagram of a company

AI-generated content may be incorrect.

Figure 1: Solution Sketch – Simplified

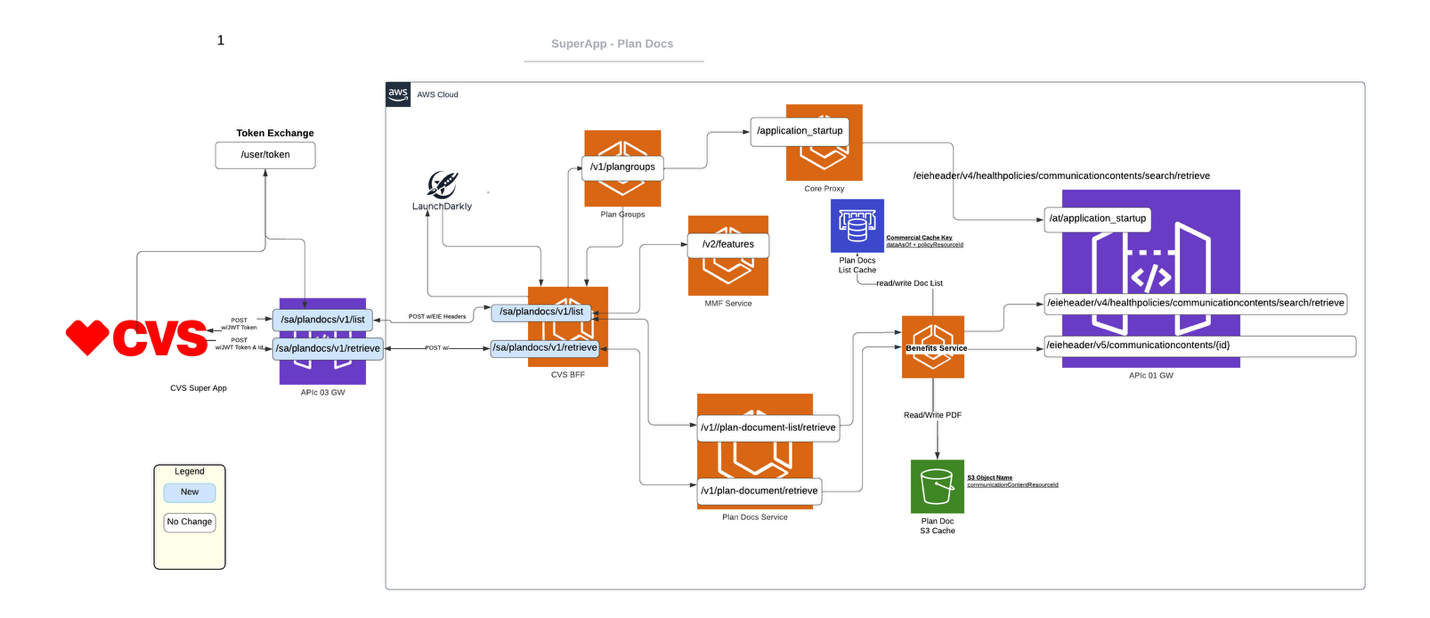


Figure 2: Solution Sketch – Extended (Features)

## 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.

A diagram of a plan

AI-generated content may be incorrect.

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

A diagram of a diagram

AI-generated content may be incorrect.

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**

A diagram of a software development process

AI-generated content may be incorrect.

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

## List Retrieval

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

**{**

**"membershipResourceId": "5~265106287+11+51+20190101+751133+BA+2",**

**"policyResourceId": "3~751133+BA+2",**

**"dateAsOf": "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

**{**

**"data": [**

**{**

**"planId": "751133+BA+2",**

**"documentName": "Plan Document (Spanish)",**

**"documentId": "70023~13910995",**

**"documentUrl": "/sa/plandocs/v1/retrieve/70023~13910995"**

**}**

**],**

**"readCommunicationsResponse": "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**

**{**

**"readHealthPoliciesCommunicationContentsResponse": {**

**"readCommunicationContents": {**

**"communicationContent": [**

**{**

**"webHyperlink": "https://qa3www.aetna.com/epublishing/getPlanDocument?serviceName=fetchPDF&nodeId=13910995",**

**"documentType": "Medical",**

**"description": "Benefit Plan Description",**

**"communicationContentIdentifier": [**

**{**

**"idSource": "70023",**

**"idValue": "13910995",**

**"idType": "communicationContents",**

**"resourceId": "70023~13910995"**

**}**

**]**

**},**

**{**

**"webHyperlink": "https://qa3www.aetna.com/epublishing/getPlanDocument?serviceName=fetchPDF&nodeId=13910996",**

**"documentType": "Medical",**

**"description": "Plan Update",**

**"communicationContentIdentifier": [**

**{**

**"idSource": "70023",**

**"idValue": "13910996",**

**"idType": "communicationContents",**

**"resourceId": "70023~13910996"**

**}**

**]**

**}**

**]**

**}**

**} ],**

**"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:

**{**

**"data": [**

**{**

**"planId": "751133+BA+2",**

**"documentName": "Medical-Benefit Plan Description",**

**"documentId": "70023~13910995",**

**"documentUrl": "/sa/plandocs/v1/retrieve/70023~13910995"**

**},**

**{**

**"planId": "751133+BA+2",**

**"documentName": "Medical-Plan Update”,**

**"documentId": "70023~13910996",**

**"documentUrl": "/sa/plandocs/v1/retrieve/70023~139109956”**

**},**

**],**

**"readCommunicationsResponse": "”**

**}**

HHL Not Signed Sample Response

**{**

**"data": [],**

**"readCommunicationsResponse": "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

The client requests a specific document via the BFF, providing the documentId (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/communicationContent[].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**](https://apiforge.cvshealth.com/apis/editor/Microservices/ah-openplatform-plandocs/1.0.4)

openapi: 3.0.3

info:

title: OpenPlatform - Plan Documents BFF

description: Provides plan document list and PDF retrieval for the OpenPlatform.

version: 1.0.4

components:

securitySchemes:

bearerAuth:

type: http

scheme: bearer

bearerFormat: JWT

schemas:

PlanDocListResponse:

type: object

properties:

data:

type: array

items:

type: object

properties:

planId:

type: string

description: Plan identifier

example: 751133+BA+2

documentName:

type: string

description: Name of the document

example: Plan Document (Spanish)

documentId:

type: string

description: Unique document identifier

example: 70023~13910995

documentUrl:

type: string

description: Internal URL to retrieve the PDF via BFF

example: /sa/plandocs/v1/retrieve/70023~13910995

readCommunicationsResponse:

type: string

nullable: true

description: >-

Special messaging for the response. Valid values: null (request

succeeded), "No docs available" (no PlanDocsMsgAvail but no docs

returned), "Your employer can provide these documents to you", "HHL

Not signed"

example: HHL Not signed

PlanDocRetrieveRequest:

type: object

required:

- documentId

properties:

documentId:

type: string

description: >-

Corresponds to communicationContentResourceId in the backend

(communicationContentIdentifier.resourceId)

example: 70023~13910995

Error:

type: object

properties:

httpCode:

type: string

example: '400'

httpMessage:

type: string

example: Bad Request

moreInformation:

type: string

example: planId is a required field.

IdTokenPayload:

type: object

description: JWT token payload structure for id\_token header

properties:

ae\_version:

type: string

description: Aetna Enterprise version

example: 1.1.0

iss:

type: string

description: Token issuer

example: https://openid.aetna.com/consumer

sub:

type: string

description: Subject identifier

example: 031RCS6NTYHFKX25DE89@aetnae.com

aud:

type: string

description: Audience

example: 89d435af-00b0-4cb6-9a35-f3189f5adc55

exp:

type: integer

description: Expiration time (Unix timestamp)

example: 1753561541

iat:

type: integer

description: Issued at time (Unix timestamp)

example: 1753557941

given\_name:

type: string

description: User given name

example: BAILEY

family\_name:

type: string

description: User family name

example: SCHEUERMAN

acr:

type: string

description: Authentication context class reference

example: http://consumer.aetna.com/assurance/loa-2

ae\_dgn:

type: string

description: Aetna Enterprise Distinguished Name

example: >-

CN=DMT-S-W265416171,OU=Members,OU=External,DC=aetheq,DC=aetnaeq,DC=com

ae\_hcr:

type: string

description: Aetna Enterprise Health Care Role

example: nextGenMember

ae\_accountId:

type: string

description: Aetna Enterprise Account ID

example: 1~DMT-S-W265416171

ae\_busIndID:

type: array

items:

type: string

description: Aetna Enterprise Business Individual IDs

example:

- globalIdentifier

- 60005~6803568937376433212

- preferredProxyId

- 15~QS3YXBBBHPXZ

ae\_impAUD:

type: string

description: Aetna Enterprise Impersonation Audience

example: ''

ae\_impHCR:

type: string

description: Aetna Enterprise Impersonation Health Care Role

example: ''

ae\_impACR:

type: string

description: >-

Aetna Enterprise Impersonation Authentication Context Class

Reference

example: ''

ae\_impDGN:

type: string

description: Aetna Enterprise Impersonation Distinguished Name

example: ''

ae\_impAccountId:

type: string

description: Aetna Enterprise Impersonation Account ID

example: ''

ae\_impBusIndID:

type: array

items:

type: string

description: Aetna Enterprise Impersonation Business Individual IDs

example: []

ae\_impGrantedLOA:

type: string

description: Aetna Enterprise Impersonation Granted Level of Assurance

example: ''

paths:

/sa/plandocs/v1/list:

post:

summary: Retrieves the list of plan documents for a user.

security:

- bearerAuth: []

parameters:

- name: id\_token

in: header

required: true

description: JWT token containing user context and authorization information

schema:

type: string

example: >-

eyJhbGciOiJSUzI1NiJ9.-UVMzWVhCQkJIUFhaIl0sImFlX2ltcEFVRCI6IiIsImFlX2ltcEhDUiI6IiIsImFlX2ltcEFDUiI6IiIsImFlX2ltcERHTiI6IiIsImFlX2ltcEFjY291bnRJZCI6IiIsImFlX2ltcEJ1c0luZElEIjpbXSwiYWVfaW1wR3JhbnRlZExPQSI6IiJ9.ADRmgfGQeRujWsf0-feL3IHQTaCDIJ18zFhOwCdW83G3iheH05-UwBf8AKj8xcFjer3IJh1eZrh9wBX6Pxl8lSmJ3sOb\_ZkHnXlo2WPS05n\_4wWCqgQPGINGwK6UcLYcHT20Yo1nmxyFbigOgnMZgiarqN9SNeIz1C1ZFXt38\_d03VbE3CEuyupLzqlqasmuTEiJlJOfhl4r\_buZGhiEmO3ZZyyf-fEak1n3TgFhlxKXKEPSVaH2ndSIjcUmUP6-lNsbPbOeJ7Cy3KL3vzDb25Z0lVR2bkQ34VYJjpQBzTNk3Dyehk51EknLAreZBtwEcoTLyPIlnCgqG78mbeK1UQ

responses:

'200':

description: Successful retrieval of document list.

content:

application/json:

schema:

$ref: '#/components/schemas/PlanDocListResponse'

'400':

description: Bad request

content:

application/json:

schema:

$ref: '#/components/schemas/Error'

'401':

description: Unauthorized - Invalid or missing authentication

content:

application/json:

schema:

$ref: '#/components/schemas/Error'

'403':

description: Forbidden - Insufficient permissions

content:

application/json:

schema:

$ref: '#/components/schemas/Error'

'500':

description: Internal Server Error

content:

application/json:

schema:

$ref: '#/components/schemas/Error'

/sa/plandocs/v1/retrieve:

post:

summary: Retrieves a specific plan document PDF.

requestBody:

required: true

content:

application/json:

schema:

$ref: '#/components/schemas/PlanDocRetrieveRequest'

responses:

'200':

description: Success - PDF file returned.

content:

application/pdf:

schema:

type: string

format: binary

'400':

description: Bad request

content:

application/json:

schema:

$ref: '#/components/schemas/Error'

'500':

description: Internal Server Error

content:

application/json:

schema:

$ref: '#/components/schemas/Error'

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.

# OpenPlatform Plan Documents BFF API

## 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",  
 "60005~6803568937376433212",  
 "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 app performance. Caching at the Benefits Service (TTL 1 hour) and S3 layer will be leveraged to ensure fast responses.
* **Rate Limiting**: Existing rate limits for the backend services will be used. The Plan Doc List endpoint is 950/min, and the PDF endpoint is 2333/min. These will be monitored for the new OpenPlatform traffic.
* **Future-proofing**: The architecture must not prevent the future implementation of search functionality.
* **Scalability**: The chosen architecture (Option 1) reuses existing, scalable AWS services (API Gateway, Lambda, S3, Redis), ensuring that the solution can handle anticipated growth in OpenPlatform user traffic.
* **Observability**: Comprehensive logging, monitoring, and alerting will be implemented for the new OpenPlatform BFF, providing visibility into its performance, errors, and traffic patterns. This will include integration with existing APM (Application Performance Monitoring) tools.

# Services

**Service Logic**

* **OpenPlatform BFF**:
  + Receives a request from the OpenPlatform client for the document list, including bearer token and id\_token.
  + Calls the backend Benefits Service's /plan-document-list/retrieve endpoint, passing these identifiers.
  + Receives the document list from the backend.
  + Transforms the backend response into the format required by the OpenPlatform client, as defined in the API section. This includes:
    - Creating the appropriate JSON structure for the zero, single, or multiple document scenarios.
    - Appending the language (from the Benefits Service response) to the documentName for display purposes (e.g., "Plan Document (Spanish)").
    - Ensuring that if a description is not available, the field is not shown.
    - **Forwarding reasonCode**: If the Benefits Service response includes a reasonCode (e.g., "HHL Not signed"), this is passed directly to the client.
* **Benefits Service (Existing)**:
  + **List**: Logic remains as-is. It expects policyResourceId, membershipResourceId, and dateAsOf in its request. It checks a Redis cache based on a complex key (considering Medicare vs. Commercial, ANOC, etc., potentially including dateAsOf + policyResourceId + externalPlanId + classCode + priorAdjacentContractPBP for Medicare Group or dateAsOf + policyResourceId + priorAdjacentContractPBP for Medicare Individual, and dateAsOf + policyResourceId for IFP/Commercial) before calling the core API. The cache has a TTL of one hour. It returns reasonCode: "HHL Not signed" if applicable for self-insured plans without the signed letter.
  + **PDF**: Logic remains as-is. It expects communicationContentResourceId in its request. It checks an S3 bucket for the PDF. On a miss, it fetches from the core API, streams it to S3 for caching, and returns the document. It will also check the byte size of the file from core and return an error if it is 0 to avoid caching empty files.

**1. Foundational & Strategic Stories**

* **Initiative**
  + **Story:** Enable OpenPlatform users to access their plan documents by creating a new BFF, integrating with existing backend services, and providing a resilient, performant user experience.
* **Theme**
  + **Story:** Enhance User Experience by providing users with seamless and fast access to their essential plan documents.
  + **Story:** Increase Development Velocity by reusing proven, existing architecture and services to reduce initial development time and risk.
* **Epic**
  + **Story:** Implement plan document access for OpenPlatform by building a dedicated Backend-for-Frontend (BFF) that routes list and PDF retrieval requests to existing AWS services.

**2. Architectural & Technical Enablement Stories**

* **Architectural Runway Story**
  + **Story:** Establish the new OpenPlatform BFF as a deployable microservice within the AWS platform, leveraging existing CI/CD pipelines and infrastructure-as-code practices.
* **Enabler Story (Exploration/Spike)**
  + **Story:** Spike: Analyze architectural options for fetching plan documents, including adapting the existing on-demand architecture, pre-fetching at app startup, and a BFF-centric hybrid model, to determine the optimal approach for balancing performance and complexity.
* **Enabler Story (Infrastructure)**
  + **Story:** Configure the necessary routing, permissions, and service integrations within the AWS ecosystem to allow the new OpenPlatform BFF to communicate with the existing backend Benefits Service and Features API.
* **Enabler Story (Compliance/Security)**
  + **Story:** Conduct a security review of the new OpenPlatform BFF, validating its authentication/authorization mechanisms, rate limiting, data encryption, and access controls to ensure it is a secure entry point into the AWS ecosystem.

**3. Migration & Integration Stories**

* **Integration Story**
  + **Story:** As the OpenPlatform BFF, I want to integrate with the existing Benefits Service by calling its /plan-document-list/retrieve endpoint to fetch a user's document list.
* **Strangler Fig Pattern Story**
  + **Story:** Implement a new route for Plan Document requests within the OpenPlatform ecosystem that directs all client traffic through the new OpenPlatform BFF instead of any existing path.

**4. User-Facing & Functional Stories**

* **Feature-Driven Story**
  + **Feature:** OpenPlatform - Plan Documents Access.
* **UX Stories (Frontend)**
  + **Story:** As an OpenPlatform user, when I navigate to the Plan Documents section, I want the system to first check if I'm eligible to view documents or if there's a special message for me, so I get the correct information immediately.
    - **Acceptance Criteria:** Client calls the Features API with membershipResourceId on page load.
    - **Acceptance Criteria:** If PlanDocsMsgAvail is enabled, or if no DenPlanDocs, MedPlanDocs, or VisionPlanDocs are enabled, the client directly displays a predefined message instead of attempting to fetch a list.
  + **Story:** As an OpenPlatform user, if my plan requires a signed Hold Harmless Letter and it's not signed, I want to see a clear message explaining this, instead of an empty document list.
    - **Acceptance Criteria:** The client displays a user-friendly message for the "HHL Not signed" scenario.
    - **Acceptance Criteria:** This message is triggered either by the PlanDocsMsgAvail feature flag or by the reasonCode: "HHL Not signed" from the BFF response.
    - **Acceptance Criteria:** The UI handles this gracefully, avoiding errors or blank screens.
  + **Story:** As an OpenPlatform user, when I navigate to the Plan Documents section and there is only one plan document available (and no special messages), I want the PDF to open directly upon my initial tap, so I can quickly view it.
    - **Acceptance Criteria:** The client displays a "Plan Document" link/button.
    - **Acceptance Criteria:** Upon tapping, the PDF viewer opens immediately with the single document.
    - **Acceptance Criteria:** The PDF viewer chrome (design and behavior, e.g., share, print options) is consistent with other document views in the app.
  + **Story:** As an OpenPlatform user, when I navigate to the Plan Documents section and there are multiple plan documents available (and no special messages), I want to see a list of documents with their names, so I can choose which one to view.
    - **Acceptance Criteria:** The client displays a list of documents.
    - **Acceptance Criteria:** Each item in the list shows the documentName (derived from description or title and possibly language).
    - **Acceptance Criteria:** Tapping on a document in the list opens its PDF in the viewer.
  + **Story:** As an OpenPlatform user, I want the plan document viewing experience to be consistent in design and behavior, so I have a seamless user experience across the app.
    - **Acceptance Criteria:** The UI/UX for plan document access (e.g., buttons, layout, PDF viewer) adheres to the established OpenPlatform design guidelines ("chrome").
    - **Acceptance Criteria:** The feature may be implemented as a full page instead of a sheet for optimal usability.
* **Server Stories (Backend)**
  + **Story:** As an OpenPlatform BFF, I want to retrieve a list of plan documents for a given policyResourceId, membershipResourceId, and dateAsOf, so that I can provide it to the OpenPlatform client.
    - **Acceptance Criteria:** The BFF successfully calls the Benefits Service's /plan-document-list/retrieve endpoint with the correct policyResourceId, membershipResourceId, and dateAsOf.
    - **Acceptance Criteria:** The BFF handles various response scenarios from the Benefits Service (e.g., success, no documents, errors, including reasonCode like "HHL Not signed").
    - **Acceptance Criteria:** The BFF correctly transforms the Benefits Service response into the OpenPlatform API contract format.
    - **Acceptance Criteria:** The BFF gracefully handles cases where description or title are missing from the upstream response.
  + **Story:** As an OpenPlatform BFF, I want to retrieve a specific plan document PDF by its documentId (mapping to communicationContentResourceId), so that I can deliver it to the OpenPlatform client.
    - **Acceptance Criteria:** The BFF successfully calls the Benefits Service's /plan-document/retrieve endpoint with the provided documentId.
    - **Acceptance Criteria:** The BFF correctly streams the PDF content received from the Benefits Service to the OpenPlatform client.
    - **Acceptance Criteria:** The BFF handles error responses (e.g., 400, 500) from the Benefits Service and propagates appropriate error codes to the client.
  + **Story:** As the Benefits Service, I want to use Redis as a cache for plan document lists based on relevant identifiers, so that I can reduce load on core APIs and improve response times.
    - **Acceptance Criteria:** The Benefits Service correctly derives and uses composite cache keys.
    - **Acceptance Criteria:** If a cache miss occurs, the core API is called, and the successful response is cached in Redis with a 1-hour TTL before being returned.
  + **Story:** As the Benefits Service, I want to use S3 as a cache for plan document PDFs, so that I can reduce load on core APIs and improve response times.
    - **Acceptance Criteria:** The Benefits Service checks the configured S3 bucket for the requested PDF using communicationContentResourceId as the key.
    - **Acceptance Criteria:** If the PDF is not in S3, it is retrieved from the core API, stored in S3 for future requests, and then streamed to the BFF.
    - **Acceptance Criteria:** The Benefits Service verifies the byte size of the file from core; if 0 bytes, it returns an error and does not cache the empty file.
  + **Story:** As the Benefits Service, I want to ensure my APIs are resilient and return appropriate errors and reasonCodes for various scenarios, particularly for HHL.
    - **Acceptance Criteria:** The /plan-document-list/retrieve and /plan-document/retrieve endpoints return standard HTTP error codes with consistent error schemas.
    - **Acceptance Criteria:** For self-insured plans where the HHL is not signed, the /plan-document-list/retrieve API returns an empty communicationContent array with a reasonCode of "HHL Not signed".

**5. Non-Functional Requirement (NFR) Stories**

* **Performance Story**
  + **Story:** As the Benefits Service, I want to use Redis to cache plan document lists for one hour so that I can reduce load on core APIs and improve response times for the OpenPlatform BFF.
  + **Story:** The new BFF endpoints must adhere to the existing backend rate limits of 950/min for the list endpoint and 2333/min for the PDF endpoint to ensure system stability.
* **Scalability Story**
  + **Story:** The Plan Documents feature must be horizontally scalable to handle anticipated growth in OpenPlatform user traffic by leveraging existing scalable AWS services (API Gateway, Lambda, S3).
* **Reliability/Availability Story**
  + **Story:** The OpenPlatform BFF must gracefully handle API error responses (e.g., 4xx, 5xx) from the downstream Benefits Service by propagating appropriate error codes and user-friendly messages to the client.
* **Usability Story**
  + **Story:** As an OpenPlatform user, I want the plan document viewing experience to be consistent in design and behavior with other document views in the app, so I have a seamless user experience.

**6. Operational & Maintenance Stories**

* **Tooling and Automation Story**
  + **Story:** Implement comprehensive logging, monitoring, and alerting for the new OpenPlatform BFF to provide visibility into its performance, errors, and traffic patterns via existing APM tools.
* **Documentation Story**
  + **Story:** Create and publish a formal OpenAPI (YAML) specification for the new OpenPlatform BFF endpoints, including detailed schemas and examples for the request and response models.
* **Technical Debt Story**
  + **Story:** To mitigate data gaps from upstream services, a limited data augmentation capability will be built into the BFF, while deferring the complexity of a full BFF-side caching solution to a future iteration

**Service Components**

* **New OpenPlatform BFF (Backend-for-Frontend)**: A new service specific to the OpenPlatform, responsible for handling client requests and communicating with the backend services.
* **Features API (Existing)**: An existing API consumed by the client to determine enabled features (e.g., DenPlanDocs, MedPlanDocs, VisionPlanDocs, PlanDocsMsgAvail) for a given membershipResourceId.
* **AWS Benefits Service (Existing)**: The existing microservice that contains the business logic for fetching plan documents.
* **AWS S3 (Existing)**: Used for caching PDF documents to improve performance and resilience.
* **Redis (Existing)**: Used by the Benefits Service for caching the plan document list (metadata).
* **Core APIs (Existing)**: The source of truth for plan documents when they are not in cache.

**Client Changes**

**Client Side Logic**

The client will integrate with the new OpenPlatform BFF endpoints.

* **Pre-check for Features and Messaging:**
  + When the user accesses the Plan Documents feature, the client will **first call a "Features API"** using the user's membershipResourceId (derived from application startup API data).
  + The client will check the Features API response for featureCode values such as DenPlanDocs, MedPlanDocs, VisionPlanDocs, and PlanDocsMsgAvail.
  + **Decision Point:**
    - **If PlanDocsMsgAvail is enabled OR if specific PlanDoc types (Medical, Dental, Vision) are NOT enabled for this membershipResourceId:** The client will **immediately display a "Special Messaging" UI** (e.g., an "HHL Not Signed" message or a "No documents available" message), bypassing the call to the BFF for document retrieval.
    - **Otherwise (if PlanDocsMsgAvail is not enabled AND relevant PlanDoc types ARE enabled):** The client proceeds to request the document list from the BFF.
* **Requesting Document List from BFF:**
  + The client will identify the appropriate policyResourceId and dateAsOf (today's date for actively covered, coverageDatetimeBegin for pending Medicare) for the selected membershipResourceId (based on the enabled DenPlanDocs, MedPlanDocs, or VisionPlanDocs features).
  + The client will call the new OpenPlatform BFF endpoint (POST /v1/cvs/plan-documents/list) with the policyResourceId, membershipResourceId, and dateAsOf in the request payload.
* **Processing BFF Response for Document List:**
  + Upon receiving the response from the BFF, the client will first check for a reasonCode (e.g., "HHL Not signed").
    - **If reasonCode: "HHL Not signed" is present:** The client will **display the "Special Messaging" UI** for the HHL scenario, even if some document data might have been theoretically returned (though typically data will be empty in this case).
    - **Otherwise:** The client processes the data array in the response:
      * If data.length is 0, the client must handle the "zero document" scenario gracefully.
      * If data.length is 1, the client should directly open the PDF view upon the user's initial tap.
      * If data.length is >1, the client will display a list of the available documents for the user to choose from.
* **PDF Viewing and Actions:**
  + The client will need to provide UI options for viewing, sharing, and printing the selected PDF document.
  + The UI/UX should follow the same design and behavior ("chrome") as other document views in the app.
  + The feature may need to be implemented as a full page rather than a sheet for better usability.

**UX Stories (Frontend)**

* **As an OpenPlatform user, when I navigate to the Plan Documents section, I want the system to first check if I'm eligible to view documents or if there's a special message for me, so I get the correct information immediately.**
  + **Acceptance Criteria:**
    - Client calls the Features API with membershipResourceId on page load.
    - If PlanDocsMsgAvail is enabled, or if no DenPlanDocs, MedPlanDocs, or VisionPlanDocs are enabled, the client directly displays a predefined message instead of attempting to fetch a list.
* **As an OpenPlatform user, if my plan requires a signed Hold Harmless Letter and it's not signed, I want to see a clear message explaining this, instead of an empty document list.**
  + **Acceptance Criteria:**
    - The client displays a user-friendly message for the "HHL Not signed" scenario.
    - This message is triggered either by PlanDocsMsgAvail feature flag or by the reasonCode: "HHL Not signed" from the BFF response.
    - The UI handles this gracefully, avoiding errors or blank screens.
* **As an OpenPlatform user, when I navigate to the Plan Documents section and there is only one plan document available (and no special messages), I want the PDF to open directly upon my initial tap, so I can quickly view it.**
  + **Acceptance Criteria:**
    - The client displays a "Plan Document" link/button.
    - Upon tapping, the PDF viewer opens immediately with the single document.
    - The PDF viewer chrome (design and behavior, e.g., share, print options) is consistent with other document views in the app.
* **As an OpenPlatform user, when I navigate to the Plan Documents section and there are multiple plan documents available (and no special messages), I want to see a list of documents with their names, so I can choose which one to view.**
  + **Acceptance Criteria:**
    - The client displays a list of documents.
    - Each item in the list shows the documentName (derived from description or title and possibly language).
    - If a description field is present in the API response, it is used for the document title; otherwise, it is handled gracefully (e.g., using title or a generic name).
    - Tapping on a document in the list opens its PDF in the viewer.
    - The PDF viewer offers options such as share and print.
    - All displayed documents are PDFs.
* **As an OpenPlatform user, when I navigate to the Plan Documents section and there are no documents available (and no special messages from PlanDocsMsgAvail), I want to see a clear message indicating that, so I understand the situation.**
  + **Acceptance Criteria:**
    - The client displays a user-friendly message indicating no plan documents are available.
    - The UI handles the "zero document" scenario gracefully, avoiding errors or blank screens.
* **As an OpenPlatform user, if I have documents in multiple languages, I want to see the language indicated (e.g., in parentheses) so I can easily identify the correct version.**
  + **Acceptance Criteria:**
    - The documentName displayed in the list includes the language in parentheses (e.g., "Plan Document (Spanish)").
    - The client includes attributes to allow for future filtering by language.
* **As an OpenPlatform user, I want the plan document viewing experience to be consistent in design and behavior, so I have a seamless user experience across the app.**
  + **Acceptance Criteria:**
    - The UI/UX for plan document access (e.g., buttons, layout, PDF viewer) adheres to the established OpenPlatform design guidelines ("chrome").
    - The feature may be implemented as a full page instead of a sheet for optimal usability.

**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:

1. **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".
2. **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).
3. **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.
4. **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**

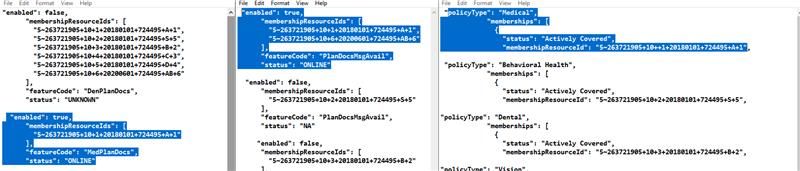
PlanDocsMsgAvail flag can be returned more than once if the memberships have different statuses. i.e:

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



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** | **{**  **"membershipResourceId": "5~265106287+11+51+20190101+751133+BA+2",**  **"policyResourceId": "3~751133+BA+2",**  **"dateAsOf": "2025-05-30T00:00:00.000Z"**  **}** |
| **Response** | {  "data":[{  "planId" : "",  "documentName":"" ,  "documentId":"",  "documentUrl" :"" //nice to have  }]  } | **{**  **"readHealthPoliciesCommunicationContentsResponse": {**  **"readCommunicationContents": {**  **"communicationContent": [**  **{**  **"webHyperlink": "string",**  **"documentType": "string",**  **"description": "string",**  **"title": "string",**  **"format": "string",**  **"documentSubType": "string",**  **"language": "string",**  **"contentSize": "string",**  **"communicationContentIdentifier": [**  **{**  **"idSource": "string",**  **"idValue": "string",**  **"idType": "string",**  **"resourceId": "string"**  **}**  **],**  **"communicationContentEffectivePeriod": {**  **"dateTimeAsOf": "string",**  **"dateTimeBegin": "string",**  **"dateTimeEnd": "string"**  **},**  **"viewerURL": "string"**  **}**  **]**  **}**  **}**  **}** |
| **Retrieve** | **Request** | {  "documentName":"" ,  "documentId":"",  "documentUrl" :"" //nice to have  }]  } | **{**  **"communicationContentResourceId": "70023~13910995"**  **}** |
| **Response** | application/pdf stream | application/pdf stream |

|  |  |  |  |
| --- | --- | --- | --- |
| **Service** | **Request Response** | **CVS** | **AETNA** |
| **List** | **Request** | HEADER: **JWT id\_token** | **{**  **"membershipResourceId": "5~265106287+11+51+20190101+751133+BA+2",**  **"policyResourceId": "3~751133+BA+2",**  **"dateAsOf": "2025-05-30T00:00:00.000Z"**  **}** |
| **Response** | {  "data":[{  "planId" : "",  "documentName":"" ,  "documentId":"",  "documentUrl" :"" //nice to have  }]  } | **{**  **"readHealthPoliciesCommunicationContentsResponse": {**  **"readCommunicationContents": {**  **"communicationContent": [**  **{**  **"webHyperlink": "string",**  **"documentType": "string",**  **"description": "string",**  **"title": "string",**  **"format": "string",**  **"documentSubType": "string",**  **"language": "string",**  **"contentSize": "string",**  **"communicationContentIdentifier": [**  **{**  **"idSource": "string",**  **"idValue": "string",**  **"idType": "string",**  **"resourceId": "string"**  **}**  **],**  **"communicationContentEffectivePeriod": {**  **"dateTimeAsOf": "string",**  **"dateTimeBegin": "string",**  **"dateTimeEnd": "string"**  **},**  **"viewerURL": "string"**  **}**  **]**  **}**  **}**  **}** |
| **Retrieve** | **Request** | {  "documentName":"" ,  "documentId":"",  "documentUrl" :"" //nice to have  }]  } | **{**  **"communicationContentResourceId": "70023~13910995"**  **}** |
| **Response** | application/pdf stream | application/pdf stream |