How Allianz Insurance built a scalable document processing pipeline using Amazon Bedrock

Allianz Versicherungs AG is a cornerstone of Allianz's product business in Germany. With more than 120 years of experience behind us, we are number one in property and casualty insurance in Germany.

## Use case

Our insurance team manages contracts of over 20k small to medium sized corporate (‘midcorp’) customers (check) for a variety of insurance products (like: business interruption, property insurance …etc). The contracts’ details were stored in a legacy information management system, which we were looking to migrate to a newer system. The new system allows storing a larger set of contract attributes compared to those stored in the legacy system.

The extraction of the custom entities in scope (+150 attributes), is a lengthy process that could take an insurance clerk - on average - 2-3 hours per contract to extract the required info from relevant documents.The attribute values could be either numbers, percentages, booleans or strings, e.g. an attribute could be “Coverage amount for buildings in case of water damages” with attribute value 1.000.000,00 Euro.

## Our solution to this was to build a performant GenAI-enabled entity extraction pipeline, which is tasked to extract contract attributes. The extracted attributes can be verified by the end users in a dashboard. Once approved, the attributes are automatically ingested into the new target system. This shortens the average extraction time to 15 minutes.

To accelerate our initiative, we worked closely with our AWS peers to design and build this intelligent document processing (IDP) pipeline. In this post, we share how we built a scalable and performant pipeline to unblock the migration process and our clerks.

## Data description

Our input text corpus consists of scanned PDF documents (mostly contracts, updates and email correspondences). A policy number (identfier of a corporate customer) has multiple documents attached, which might be outdated or might not contain any relevant information for our task at hand (e.g. email correspondences between broker and customer).

Therefore the first challenge, before extracting the actual values of our +150 contract attributes, was to identify a subset of documents which likely contains the relevant paragraphs with the attribute values (e.g. Yes/No answer to the question “Are buildings insured against fire?”). This pre-selection is obviously very crucial for the quality of all subsequent retrieval tasks and therefore was conducted by our (internal) users, i.e. for a given insurance policy we provided all attached documents to the clerks in a UI so that they could select it for processing for the IDP.

In addition to the described initial document classification we faced multiple data quality challenges: The selected single scanned PDFs were composed of multiple different documents, e.g. with email correspondences at the beginning followed by the actual contract document with a large appendix and then sometimes again email correspondences. The distribution of the number of pages over all (how many??) documents in our corpus showed a ‘power law’ shape (i.e. right-skewed, heavy tailed), with a median of 12 and some documents exceeding 120 pages. Long documents introduced additional noise to our data set thus making the retrieval more challenging. Also varying policy layouts within a single document also created difficulties, e.g. some insurance brokers used different formats over time. And finally, as in many industrial applications, our domain vocabulary is quite specific and posed a challenge in the prompt engineering part. In our case the documents were all in German with many insurance specific terms and phrases, sometimes even for native speakers not so commonly used.

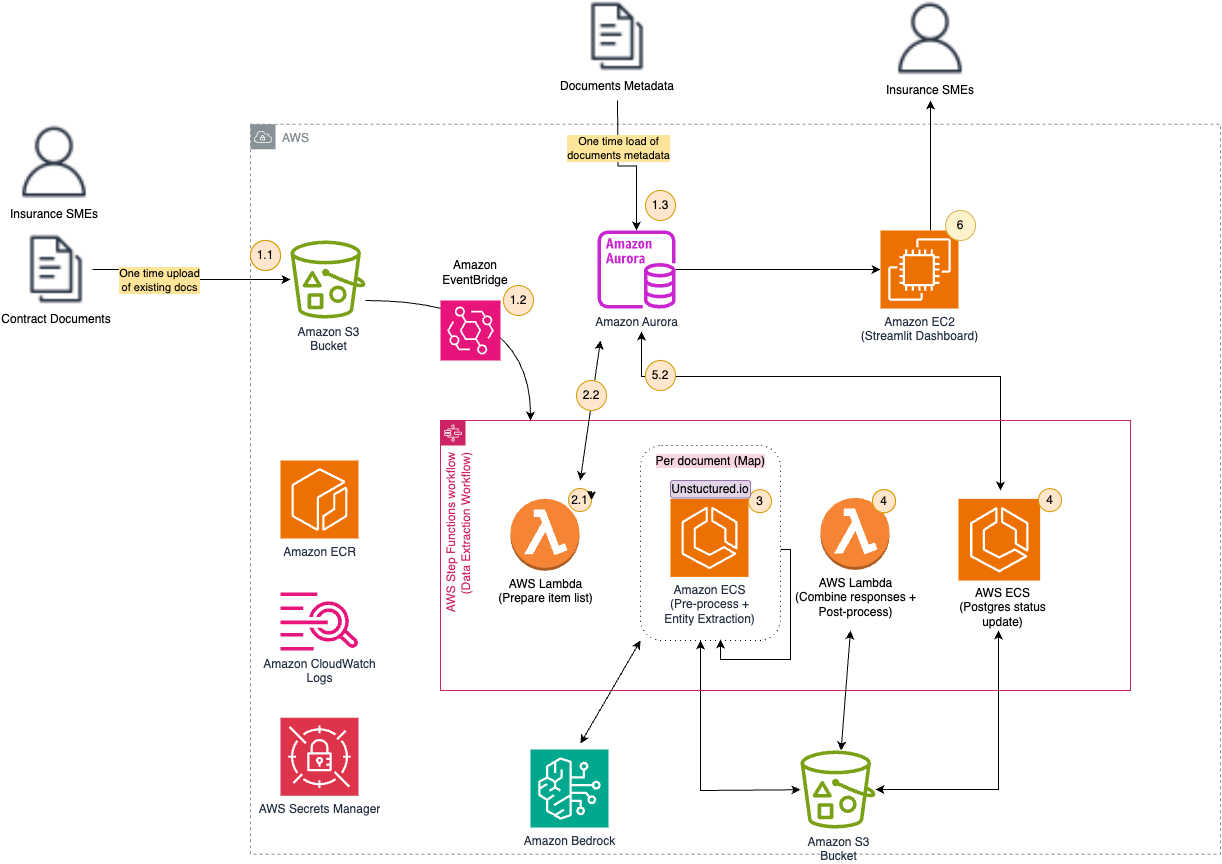
## Requirements (for solution)

One of the core requirements was to complete the migration process latest by Q3-2024 (check). As mentioned above the business goal of the midcorp migration project was to support our internal users by automating the attribute retrieval process and thus reducing time and costs overall. With regards to the later requirement we therefore opted for using an open source OCR library (‘Unstructured-io’) instead of using AWS Textract for the raw text extraction part to save costs.

With traceability as a top priority, we had to make sure that the entity values being extracted are referenced – with source documents and page numbers – to ease verification by clerks. Also to increase trust in our IDP, we additionally provided the end user with conscise reasoning for each extracted value.

## Solution

While working with the AWS team, we designed and built the solution with scalability, performance, and cost optimized in mind. The following solution architecture depicts the high-level components we built to fulfil our use case.



1. User uploads a JSON file lookup with policy numbers + associated document IDs and the corresponding raw PDF documents to an S3 bucket. By uploading the JSON file to S3 an Event Bridge rule will trigger the IDP, an AWS Step function. In the initial run, write document meta information to AWS Aurora
2. Lambda function reads document list from S3 and prepares item list for subsequent MAP function. Reads from AWS Aurora Postgres database to select previous results and combines it with current item list in case previous runs had failed.
3. Preprocess/Retrieve: MAP function which processes each document asynchronously in the item list in an ECS Fargate task. The ECS task runs a Docker image which contains the main code of the application, our Python package “midcorp\_migration”. There we first preprocess the raw documents using OCR and the apply document chunking. We build a large battery of prompt templates to capture all of the required entities using LangChain and then invoke AWS Bedrock to extract the values for us.
4. Postprocess: Combine Bedrock responses for each document chunk to a per-document response and finally combine per-doc Bedrock responses based on documents’ creation dates to final attribute values per contract. Push combined pipeline output also to S3
5. ECS Fargate task inserts results and pipeline run status to Aurora Postgres

In the following sections, we dive into some of the architecture decisions we’ve made and learnings we came across while building.

## Transient Vector Store and enhanced RAG

As far as our use case omits, we didn’t need to persist the processed text into a typical vector store, however we still needed to perform semantic search and probe our documents. For this we used a ChromaDB in-memory vector store to store Cohere multilingual vector embeddings which we used in a Retrieval Augmented Generation (RAG) step.

We also managed to enhance the retrieved document citations by performing hybrid search by combining a vector search with a BM25 keyword search. (More details on the RAG part?)

## Improve the LLM outputs

In the initial project phase we used Anthropic’s Claude 3.1 via Bedrock for the LLM part without in-context learning and without RAG to keep things simple. However the results were relatively poor given the above mentioned data quality issues. So to be able to experiment faster with new approaches we centralized the core components in our “midcorp\_migration” Python package and also created a small ground truth data set with train/test splits for prompt engineering. We switched to few shot learning with semantic grouping of queries/attributes. This reduced costs by reducing the input tokens per request and also helped the LLM by reducing the topics per query it had to focus on. Also by switching from Claude 3.1 to Claude 3.5 Sonnet helped a lot in combination with the other enhancements. (More details on the prompt engineering part?)

## Results evaluation on scale

With retrieval quality in mind, we continuously applied our pipeline to our user annotated data set during the development phase. Since each entity retrieval part can be viewed as a binary classification task, we calculate basic statistical metrics like precision, recall and F1 score for each retrieved attribute per document and also aggregated over all +150 attributes and documents for each contract. (did we do this? ;))

## Conclusion and key points to take away

In this post, we outlined the approach we followed, in conjunction with our AWS SMEs, to create a high-performing and scalable document processing pipeline. By using Amazon Bedrock to accelerate our innovations, we built a pipeline that processes a contract, and associated documents, in XX sec/min with an average accuracy of XX% of the extracted entities. This solution has improved our insurance clerks’ performance by XX% and saved them XX of working days/month.

As a next step, we plan (to extend this use case to ||| OR apply the learnings to other Gen AI solutions in that…..

### About the Authors

..



……..



……..



……..



Ahmed Ewis is a Senior Solutions Architect at AWS. He helps customers build solutions to solve business problems. When not collaborating with customers, he enjoys playing with his kids and cooking.