AEM Assets Migration Blueprint

Part II – Migration and Tooling

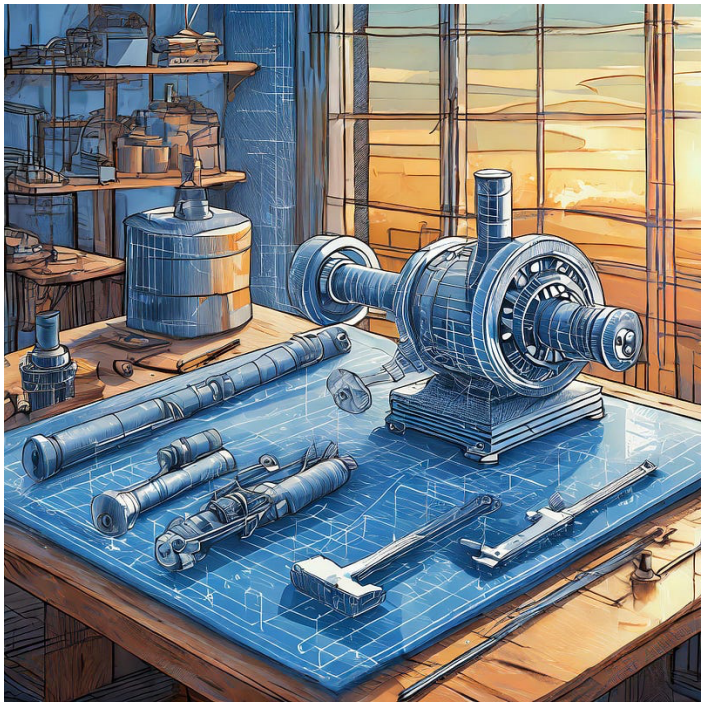


Table of contents

1 Series Overview

In the past, I have done a couple of migrations from a few Digital Asset Management Systems (DAM) to AEM Assets. Here want to share what I have learned.

This is Part 2 of a three parts series on migrating to AEM Assets:

- Part 1 — Planning
- **Part 2 — Transforming and tools**
- Part 3 — How to not Migrate Taxonomies to AEM Assets

In the first part I covered planning, stakeholder management, scheduling, and infrastructure.

This part is dedicated to transforming — how to map artifacts from the legacy system to AEM Assets.

2 Artifacts to Migrate

First, we must be aware of *what* we are going to migrate. "The images of course!" you say? Yes — but there is more:

- Asset binary files
- Metadata
- Metadata Schema
- Folder Structure
- Taxonomy

The **asset's binary files** are the most obvious part: These are the actual assets, the images, videos, and documents we want to migrate.

Each asset is associated with metadata. **Metadata** *describes* the content of the asset, provides information on where and how it can be used and provides attributes that allow assets to be retrieved via search.

The **Metadata *Schema*** describes the available attributes, allowed values and structure. In AEM the schema is defined by defining an input dialog for asset metadata (tools/assets/meta data schema). The legacy system you migrate from might have a different means to define a schema. This can be a database schema, an XML schema or something proprietary like we have in AEM. A metadata schema is more than just a loose collection of input fields. Fields often have inter-dependencies or sub-categories. Think of cases like "field B is only available if value of field A is x". Mapping a schema from a legacy DAM to AEM is not always straight forward and requires a good understanding of the schema — and often some creativity as the range of features the legacy system offers will differ from what AEM supports.

AEM requires assets to be physically store in a **Folder Structure**. Migrating offers a good opportunity to re-organize and clean up the folder structure. So, there will be some mapping required here as well. This especially is the case, when the legacy system does *not* have a physical folder structure and you must create one out of thin air — or by using a taxonomy from the legacy DAM (see below).

Assets are often classified in one or more category systems. This is what we call a **Taxonomy**. Taxonomies and folders are similar concepts and easily confused. In AEM, a file is store in one (and only one) physical folder, but it can be categories by any number of tags from multiple categories.

There is a challenge, when the legacy system does not provide a folder structure but exclusively uses taxonomies or "virtual folder". I describe the inherent problems in detail in "How to not migrate Taxonomies" [1].

3 Transforming Data

Except for the binary data, which is stored as-is the storage and structure of the artifacts are system dependent and will require some transformation.

Moreover, the transformations of the artifacts are inter-dependent. I.e. the concrete metadata of an assets must conform to the target metadata schema. Or when you transform the taxonomy of categories, the transformations need to be applied on each asset's metadata.

4 ETL — Extract Transform Load

The classic approach for migration is ETL:

- Extract data from the legacy system into an intermediary format.
- Transform into the target structure and format.
- Load the data to the new platform.

This is the approach we'll use for all artifacts.

5 CSV is your friend

I learned to love CSV files for migration. True — it is not as sophisticated, structured, and as expressive as JSON or XML. But it's damn good readable for a human. And the extracted a data file will be read a lot for reverse engineering and quality assurance.

A human can't read and comprehend tens of thousands of JSON items in an array. But you your eye can easily detect structural anomalies in a CSV table. Of course, I assume that you are reviewing the CSV files in a spreadsheet application like *MS Excel*, *macos Numbers* or Libre *Office Calc* that aligns columns vertically.

Note, you can't use a text editor to review the files. Also, you will share the data files with your business stakeholders for confirmation. Businesspeople love spreadsheets and hate reading XML files — just like everyone else ;-) .

6 Transformation Rule Engine

No two migrations are the same. Metadata rarely is 100% clean and perfect. During transformation there always will be several exceptions, edge cases and quirks.

To transform we need a highly expressive rule language. Also, transformation is a one-off exercise: Once you have migrated all assets, you will throw away the transformation code and never use it again.

Do not(!) create a custom, purpose-built rule engine up front before conducting the migration. Instead, you should use the scripting language you are most familiar with to write the transforming code. This can be for example *JavaScript/NodeJS* or *Python*. Any scripting language gives you full expressiveness to do any kind of transformation you might ever need.

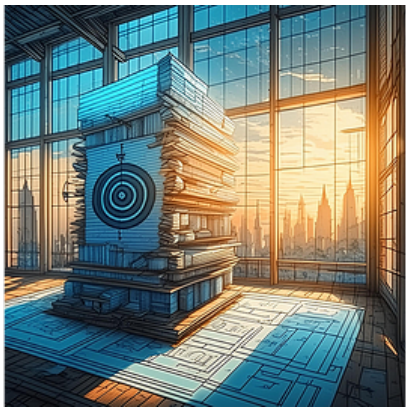
To select the scripting language for the task, consider a few criteria:

- The language should be dynamically typed,
- has good support for CSV and JSON or XML,
- supports the database the legacy data is stored on (if the system does not natively support metadata export)
- has a library to support AWS S3 or Azure Blob Storage.

You don't have to be fluent in the language. Such one-off scripts are a good opportunity to learn a new language. The task is simple enough for first steps. The code runs only once — performance doesn't matter that much. Also, the code never makes it into production, so it does not have to be ultra secure. Finally: You'll throw it away afterwards, so best practice, clean code and re-usability are of no concern.

I am using NodeJS because — being a backend developer - it is a good opportunity to refresh my JS skills. Also, NodeJS becomes more relevant on AEM as a Cloud Service, where you often use AppBuilder to augment the platform.

7 One Shot, only (And a Safety Net)



The quality of the transformed data is of utmost importance. You have only one shot.

Transformation code is throw-away code. Non-functional qualities like maintainability or re-usability are negligible.

However: The quality of the transformed *data* is of utmost importance. You only have one shot. Data that was not migrated or not migrated properly cannot be reconstructed later in the target system.

This is different to page rendering code. If a page or component does not render properly, you just fix that bug, and all is good. You cannot simply repeat the migration after you have discovered a bug two months in production…

Keep exported CSV and legacyIDs as safety net

To have a bit of a safety net, it is advised to keep the exported metadata and references to the assets.

In my last project, I kept the exported CSV files with the metadata in a local archive and stored the legacy asset IDs as a hidden field in the AEM metadata. In case there was some "data missing" after the migration, I could easily proof, there already was a bug on the legacy system… or write some patch script. Luckily, this was not necessary.

Note, we also considered storing the original metadata as a hidden JSON string in the metadata of AEM. But we decided not to, because this metadata would have polluted the results of the full text search.

8 Transformation "in Place" vs. "in Flight"

Data transformation theoretically can happen

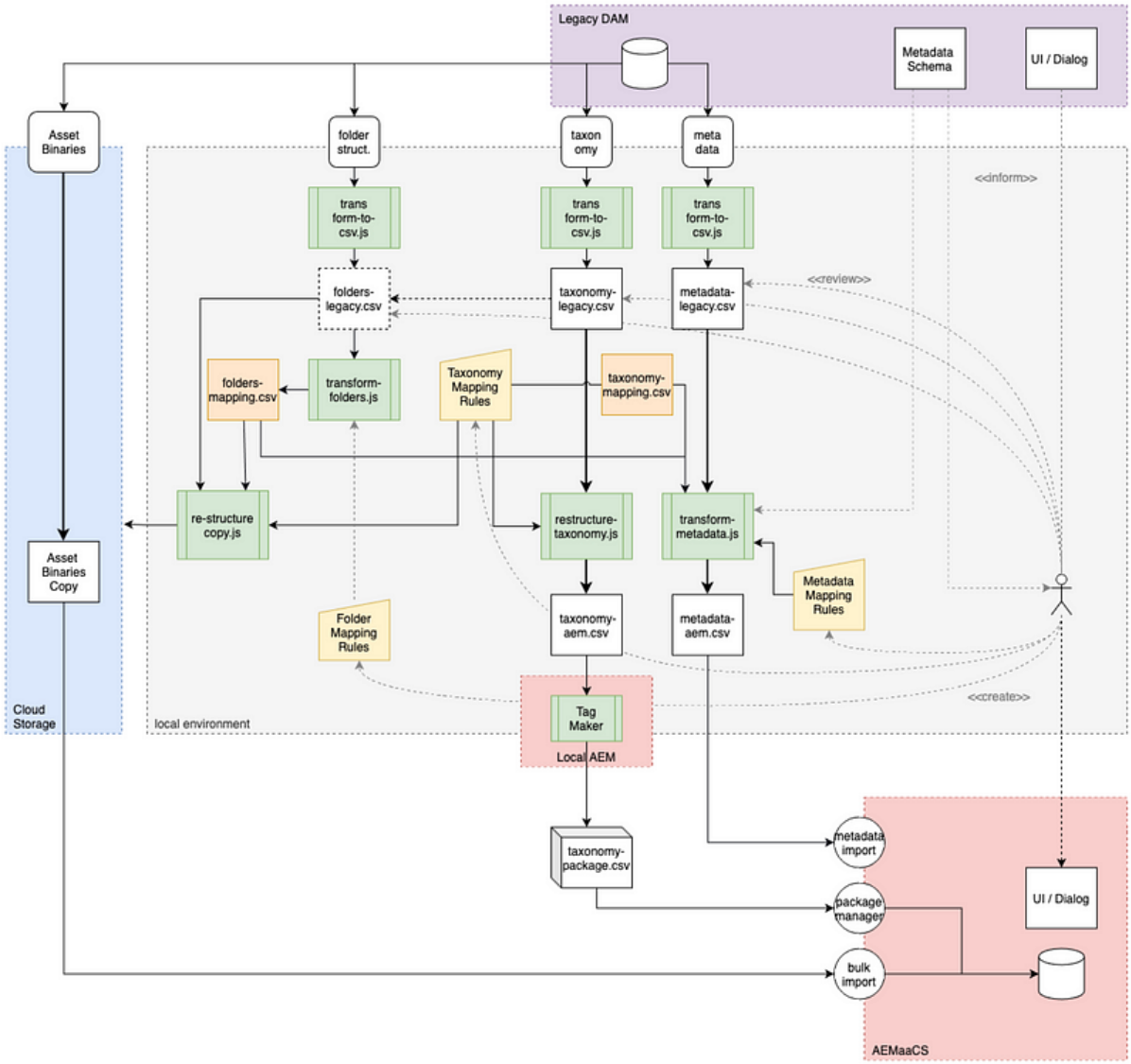
- In place on the AEM system after the raw data was transferred from the legacy system. (This is also referred to as *in-situ*).
- In flight while the data is being transferred from the legacy system.

Best practice is to do an "in flight" transformation!

The best practice is to do an "in flight" transformation: You load the data from legacy, transform on a local machine and upload the transformed data to AEM. Not only does that give you the chance to assess the data on the local machine before uploading, the process also can be repeated as often as required to get it right.

"In place" on the target AEM should **not** be considered: It is a destructive process: If something goes wrong, you must repeat the whole migration of the asset binaries and re-run the transformation. This takes much longer, as it requires re-uploading the data. Also, AEM is not particularly fast when writing bulk data.

9 The Migration Blueprint



Asset Migration Blueprint

With the basic considerations out of the way let me explain the blueprint. I am making a few assumptions about the environments. You might have to adapt in a different context.

My approach is to do the transformation outside of AEM and use existing interfaces for data ingestion. I.e., I don't have to deploy any code to AEM.

(1) Export metadata from the legacy system

- Legacy folder structure (if available)
- Legacy taxonomy (hierarchy of categories)
- Legacy asset metadata

(Do not export the binary data yet — we'll cover this later.)

(2) Transform export to CSV

- If the data is in a proprietary, structured format (e.g., JSON or XML) transform to CSV.

Note, in theory, you could transform the original files without transforming to CSV first, but migration often requires a bit of reverse-engineering of the data thus it is important to transform the data into a form that is easy to comprehend by a human. This also gives the business stakeholders the opportunity to assess the exported raw data.

CSV files can be imported into *Excel*, *Numbers* or *Calc* for better readability. Though you should not permanently transform into the respective native formats. Script processing should be done on the CSV files. *Visual Studio Code* has a good syntax highlighting for CSV files — if you need to reason about the raw CSV files (not how they are interpreted in Excel).

The output will be called

- folders-legacy.csv
- taxonomy-legacy.csv
- metadata-legacy.csv

(3) Cleanse and transform the taxonomy

A migration is a good opportunity to cleanse data. Provide the taxonomy export to the business stakeholder and ask if the need any transformation. Business may produce a couple of transformation rules in the form of:

/products/appliances -> /products/non-food/electronics/appliances
/products/convenience-food -> /products/food/convenience
/assets/usage/print -> REMOVE
…

Simple rules that can be expressed in a simple mapping file like the one above. More complex ones require some custom code.

(4) Transform Taxonomy

The transformation script generates the target *taxonomy-aem.csv* file.

The file should be in a format that AEM can process. There are two Common alternatives:

a) *ACS Commons — MCP Tag Creator* [2]

b) *ACS Commons — Tag Maker* [3]

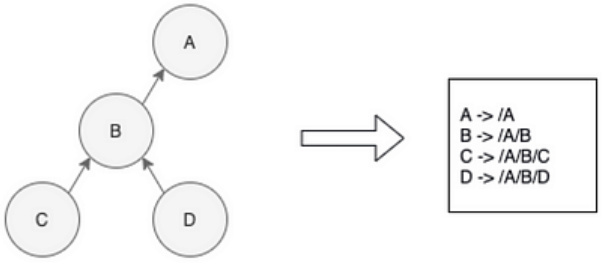
I prefer *Tag Maker* — even if it is marked as "deprecated". *Tag Creator* can handle CSV files — exactly what we are producing in this blueprint. *MCP Tag Creator,* in contrast, requires Excel Files. This is great when you are creating a taxonomy manually from scratch — but not so great, when you are migrating. If you prefer working directly with XLS files, you can use a library like *exceljs* [4] . Note, that's another library you'll have to learn. I keep things simple and use the deprecated *Tag Maker*.

Both packages require that you install ACS Commons. If you can't or if you do not want to install ACS commons on the target platform, you can install it on a local machine, generate the tags there and export the results as an AEM package *taxonomy-package.zip*

(5) Extract or build the folder structure.

Create a CSV file that describes the legacy folder structure. We'll call this file *folders-legacy.csv*. This sounds trivial. But it can be a challenge when the legacy system is not folder-based but pure search based — see [1] for details.

The last systems I migrated from had some hierarchies that resembled a folder structure — but the structure required some interpretation. I.e. folders were stored in database records, where each record had a unique ID and a field denoting the ID of its parent.



Flattening the hierarchical folder structure

Such a structure needs to be "flattened" into a table, first before you can reason about this. A simple method is to

- Load the entries into a lookup table first (aka "hash table", aka "associative array") in memory
- Recursively add all parent folders for all items
- Sort the array by paths

Sorting is important to be able to scan the result visually and spot anomalies.

(6) Transform folder structure

As was for the taxonomy, your business stakeholders might want to take the opportunity to cleanse and restructure the folders. They might provide you a set of transformation rules that you can use to script the transformation of the folder structure. In my experience, rules from business users are a mix of Excel tables with some explanation in plain English. I transform the tables into CSVs (of course) and translate the more general explanations into regular expressions or custom JS code. Both will be formalized and scripted in a transformation script (*transform-folders.js*).

This script then processes the file *folders-legacy.csv*. The result will be 1:1 mapping file with two or three rows:

- Legacy ID
- Legacy Path
- AEM Path

We'll call this file *folders-mapping.csv*.

(7) Extract the binaries

Note: I am assuming that we are migrating to AEM as a Cloud Service (AEMaaCS). This is the preferred option if you deal with hundreds of gigabytes of assets. Ping me for tips on how to extract/upload to AEM on premise or AMS.

A good practice is to extract the asset binaries to an intermediate cloud storage that both the legacy system and AEM can understand. This is for practical reasons:

- **You can quality-assure intermediate results** before uploading.
- **You can use existing tools.** You *could* migrate the binaries directly from legacy to AEM. But you would have to write the code yourself.
- **Driving force:** In an AEM migration, it is usually the AEM team driving the migration. It is unusual that the maintainers of the old system are invested in writing custom scripts to migrate off of "their" system. And the AEM team wouldn't know how to write custom scripts for the legacy system

- Standard interface: The intermediate cloud storage can act as a common interface between the two systems. Neither the legacy staff needs to learn AEM nor does the AEM team need to know the intricacies of the legacy system. Both only need to know the "industry standards".
- Separation of concerns: The AEM team might not be allowed to access the internal interfaces of the legacy system: The legacy system is still in production, and the legacy team is committed to provide certain SLAs, which could be endangered by letting a third party on "their" system.
- Cloud Storage can easily be provisioned and is cheap.

As intermediate storage, either *AWS S3* or *Azure Blob Storage* can be used as they are natively supported by AEM Assets as a Cloud Service for uploading. Extracting from legacy can be done by either

- Using native cloud export (if provided by the legacy system)
- Export to SFTP — both *Azure* and *S3* provide SFTP adapter interfaces for the storage systems.
- Use a serverless function to pull data from the legacy system's native interface and stream to the cloud storage. You can use *Adobe App Builder* as serverless runtime, *AWS Lambda*, or *Azure Functions*.

(8) Transform binary folders

The assets will be extracted in the legacy system's native structure. Within that structure, the assets must be uniquely identifiable. Either by the Legacy ID or by their path.

We then use the *folders-mapping.csv* to copy the files *within* the cloud storage from the old legacy path into to mapped path we want to use in AEM.

Copy (not move). Remember we wanted to only use non-destructive methods, so that we can adjust and repeat in case we encounter issues while developing the migration script. *Moving* would mean you'd have to physically extract the binaries again when the mapping fails.

The trick here is to use the native copy method the cloud storage provides. *Copying* means, that the cloud platform creates a new reference to an already uploaded file. The platform will not create a physical copy. Plus, there is no need for you to download or re-upload the asset to a new location. Copying tens of thousands of assets from one S3 location to another takes only a few minutes (while extracting takes hours.).

Note, that if the legacy system used the *legacyID* as filename, you might also apply a human readable physical file name while mapping. This helps AEM to derive the filetype and helps users later to identify the contents. "board-of-directors.png" is a better name than "7363542".

(9) Upload binaries

Now the assets are in the folder structure that is expected by AEM Assets. Importing is as simple as pointing the AEM Bulk Importer [5] to the cloud storage and start the upload process. AEM then directly transfers the assets from the cloud into its own storage — cloud to cloud. This usually is quite fast. Transfer speed will differ depending on your location and the service you are using. For planning, ingest a smaller subset of the assets for interpolation.

(10) Metadata schema mapping

This is a manual process. You need to understand the structure of the meta data on the legacy system and re-build something similar as AEM metadata dialog. Here also, you must involve the business stakeholders. The old schema will contain fields that have never been used and do not need to be migrated.

Fields with a shallow hierarchy and few items can be implemented as interdependent selection fields or as tag fields.

When you implement tag fields you need to decide if you want on tag field with and separate categories by name space or if want to have dedicated name spaces for each category.

There are much more decisions to make. To help the decision-making process, I usually reverse-engineer the old schema (with help of the power users and maintainers), document everything in a excel table and have a few sessions with the business owner how to map each field. I then create the meta data dialog(s) in AEM and do a couple of transformations "manually" to prove the schema can store the necessary data.

I also do a quantitative analysis of the metadata fields used on the legacy system to facilitate the discussion with the business owners: E.g., If there are 100.000 assets, and field X is used only in 20 cases, it is safe to assume, this field is not actively used and should not be migrated.

Make sure to reserve enough time for these types of workshops with the business owners. This is the most difficult part of the project.

Eventually there will be a set of transformation requirements in plain English that can be translated into script code. Let's call the script *transform-metadata.js*.

(11) Metadata transformation

We extract the metadata from legacy and transform it into CSV if necessary. The input file is called *metadata-legacy.csv*.

transform-metadata.js processes this CSV and produces another CSV *metadata.csv* that then can be ingested by AEM directly via the Metadata Import feature [6].

Note, that we also need the *folder-mapping.csv* file to map the metadata to the Assets in their new, transformed location.

10 Conclusion

I could have created a huge all-in-one script that runs the whole migration end-to-end in one swoop.

Instead, I am creating many intermediate files and sub-divided the migration into smaller scripts. This is on purpose:

a) It gives me the opportunity to do sanity checks at large scale in each step. (Using a debugger would only allow to x-check but a few assets).

b) It gives me the opportunity to partially re-run the parts of the migration.

This approach is best suited for a big-bang one-off migration. For a continuous migration, a more integrated approach would be necessary.

I hope you find this article useful. Please consider giving me a thumbs-up or follow me here. And if you haven't read already, don't forget

11 Tools and References

[1] How to not migrate Taxonomies

[2] ACS Commons — MCP Tag Creator

[3] ACS Commons — Tag Maker

[4] ExcelJS

[5] Bulk Import

[6] Metadata Import