

ARCHITECTURE DESIGN DOCUMENT

WCS to AEM Content Migration

Target architecture, components, data flow and execution approach for migrating content from Oracle WebCenter Sites (WCS) to Adobe Experience Manager (AEM 6.5 and AEM as a Cloud Service).

5

pipeline stages

2

AEM target platforms

7

execution phases

5

open decisions

PREPARED BY

Onwardpath (Ganesh Narasimhan)

DATE

29 October 2026

TARGET PLATFORMS

AEM 6.5 and AEM as a Cloud Service

About this document

This document describes the target architecture, components, data flow and execution approach for migrating content from Oracle WebCenter Sites (WCS) to Adobe Experience Manager (AEM 6.5 and AEM as a Cloud Service).

Document control

Document	WCS to AEM Content Migration: Architecture Design
Prepared by	Onwardpath (Ganesh Narasimhan)
Date	29 October 2026
Target platforms	AEM 6.5 (on-premise / AMS) and AEM as a Cloud Service

Contents

1. Introduction	3	5.3 Transformation engine	8
1.1 Purpose	3	5.4 Rich text and link rewriting	8
1.2 Audience	3	5.5 Package Builder	9
1.3 Scope	3	5.6 DAM ingestion strategy	9
1.4 Assumptions	3	6. URL management and SEO	10
2. Architecture overview	4	7. Data quality, validation and reconciliation	10
2.1 Architecture principles	4	7.1 Reconciliation after installation	10
2.2 Component summary	5	8. Security	11
3. Source and target content models	5	9. Non-functional requirements	11
3.1 WebCenter Sites (source)	5	10. Migration execution approach	12
3.2 Adobe Experience Manager (target)	5	11. Risks and mitigations	13
4. Content model mapping	6	12. Open decisions	13
5. Component design	7	13. Future phases	14
5.1 Extraction	7	Appendix A. Example mapping rule	14
5.2 Normalization and the canonical asset model	7	Appendix B. Glossary	15

Introduction

1.1 Purpose

This document defines the architecture for migrating website content from Oracle WebCenter Sites to Adobe Experience Manager. It describes the migration platform's components, the data flow between them, how the WCS content model maps to AEM, and how the migration is validated and executed. It is intended as the basis for design review, estimation and implementation planning.

1.2 Audience

- Solution and AEM architects who will review and approve the design.
- Developers who will implement the migration components and the AEM components and templates they depend on.
- Content owners and business analysts who will sign off the content mapping.
- Project managers and QA leads who will plan trial runs, UAT and cutover.

1.3 Scope

✓ In scope

- Page assets and the site plan, which become the AEM page hierarchy
- Flex and basic content assets, which become Content Fragments or page components
- Image and document blobs, which become DAM assets with metadata
- All attribute types: string, number, date, list, asset reference, blob
- Named associations (for example related content)
- Rich text: internal links, embedded assets and legacy Satellite URLs
- Vanity URLs and legacy URLs, which become redirect rules
- Traceability from every AEM node back to its WCS source asset

– Out of scope (separate workstream or manual)

- Templates, CSElements and SiteEntries. These are code and are rebuilt as AEM components and editable templates.
- Multilingual content (Dimensions and translations). This is a later phase; see section 13.
- WCS Engage segments, recommendations and personalisation
- Workflow, approval and publish state
- ACLs, users and groups
- Front-end design, dispatcher and CDN configuration (apart from the redirect rules this design produces)

1.4 Assumptions

- The WCS REST API is enabled and reachable from the migration host, with a read-only service account.
- Read access to the WCS database is available for the site plan if the navigation REST resource is not exposed.
- AEM components, editable templates and Content Fragment models are built and deployed before the first migration run.
- A content freeze window is agreed with the business for the final migration run.

Architecture overview

The migration is a batch pipeline of independent stages. Each stage reads the previous stage's output from a staging store and writes its own output back, so any stage can be inspected, re-run or replaced without repeating the others. Site-specific decisions are kept out of code and held in a declarative mapping configuration.

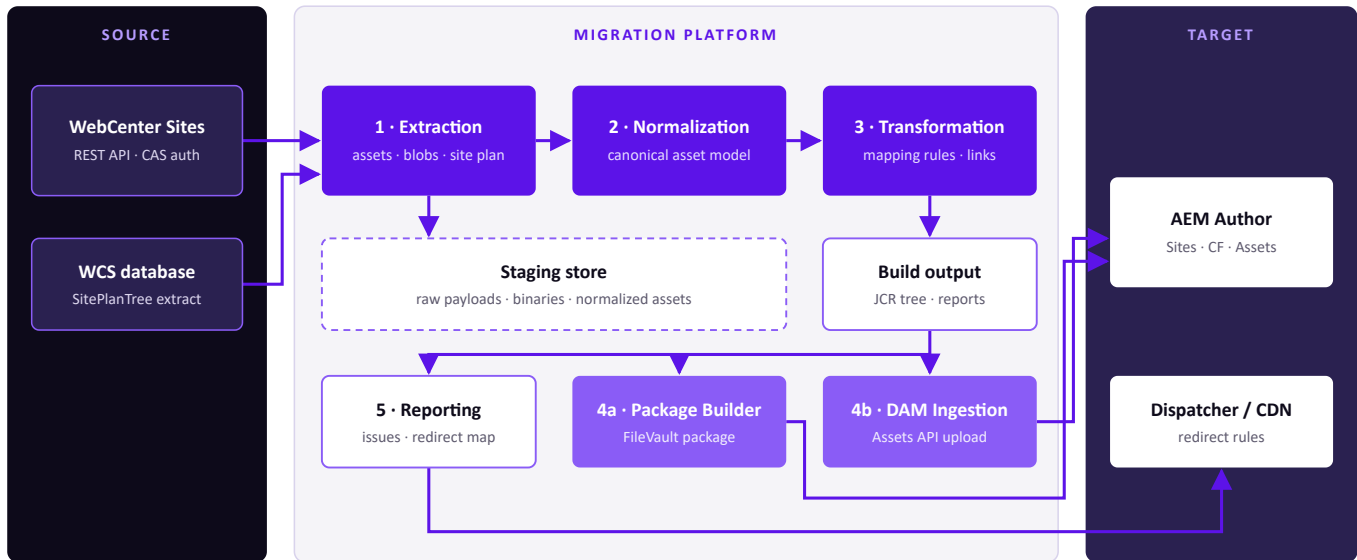


Figure 1. Content migration architecture

2.1 Architecture principles

1 Declarative mapping

Every decision about how a WCS type becomes AEM content lives in a version-controlled mapping configuration, not in code. Business analysts can review it, and changing it never needs a code release.

2 Source isolation

Only the normalization layer knows the WCS payload formats. Other sources, such as CSDT exports or database extracts, can feed the pipeline by producing the canonical asset model.

3 Two-pass transformation

Every asset's AEM path is planned before any content is generated, so references and links resolve whatever order assets are processed in.

4 Idempotent, repeatable runs

Runs can be repeated safely. Extraction resumes where it stopped, and load uses update semantics. Trial runs, delta runs and the final run all use the same process.

5 Nothing silent

Every gap is reported and nothing is dropped quietly. This covers unmapped types, missing references, links that cannot be resolved, orphan pages and missing binaries.

6 Standard AEM interfaces only

Content is loaded only through FileVault content packages and the AEM Assets APIs. Nothing writes directly to the repository.

7 Secure by default

Credentials are injected at runtime and never stored in configuration. Accounts are read-only on WCS and least-privilege on AEM.

Components and content models

2.2 Component summary

Component	Responsibility	Output
1. Extraction	Reads assets, binaries and the site plan from WCS through the REST API (or database extracts).	Raw payloads, binaries, site plan
2. Normalization	Converts version-specific WCS payloads into the canonical asset model.	Normalized assets
3. Transformation	Applies the mapping rules. Plans AEM paths, builds the content tree and rewrites rich text and references.	JCR content tree, path map
4a. Package Builder	Assembles the content tree into a FileVault content package with filters and package metadata.	Content package (.zip)
4b. DAM Ingestion	Uploads binaries to AEM Assets at the planned paths, then applies their metadata.	Assets in AEM DAM
5. Reporting	Writes the migration report, reconciliation counts, issue log and redirect map.	Report, redirect map
Staging store	File-based store holding every stage's output, keeping runs auditable and repeatable.	Versioned run folders

3.1 WebCenter Sites (source)

- **Asset types and subtypes.** Content lives in asset types such as Page, flex families (for example Article or Doctor), basic assets and media assets. Subtypes (flex definitions) set which attributes an asset has.
- **Attributes.** Typed values: string, number, date, large text (rich text), lists, asset references and blobs (binaries).
- **Site plan.** The page hierarchy lives in the SitePlanTree table. It is exposed through the REST navigation resource.
- **Associations.** Named links between assets, used for related content and page composition.
- **Webreferences.** Vanity URLs for pages and assets. Legacy URLs follow the `/cs/Satellite?c=<type>&cid=<id>` pattern.
- **Presentation.** Templates, CSElements and SiteEntries are JSP code. They are not migrated as content.

3.2 Adobe Experience Manager (target)

Area	Target structure
Sites	<code>/content/<site>/<country>/<language></code> holds the page tree (cq:Page). Pages are built from editable templates and Core Component proxies.
Content Fragments	<code>/content/dam/<site>/content-fragments/<model></code> holds structured fragments based on models under <code>/conf/<site>/settings/dam/cfm/models</code> .
Assets	<code>/content/dam/<site>/<images documents ...></code> holds dam:Asset nodes with metadata under <code>jcr:content/metadata</code> .
Configuration	<code>/conf/<site></code> holds templates, policies and Content Fragment models. These are deployed as code, not migrated.
Redirects	Dispatcher or CDN rewrite maps, or ACS Commons Redirect Manager, built from the generated redirect map.

Content model mapping

The table shows the standard mapping patterns. The concrete mapping for each asset type and subtype is set in the mapping configuration and signed off with content owners during the mapping phase.

WebCenter Sites	AEM	Notes
Site	Site root page (<code>aem.contentRoot</code>)	The WCS home page becomes the site root page.
Site plan (<code>SitePlanTree</code>)	Page hierarchy and sibling order	From the REST navigation resource, or a database extract.
Page asset (by subtype)	<code>cq:Page</code> built from an editable template	The subtype sets the template, the page resource type and the component layout.
Page attributes	Page properties and components	Title, description and SEO fields go to page properties. Body content goes to Text, Image, Teaser or List components.
Flex/basic content asset	Structured Content Fragment	Elements are mapped field by field. Rich text elements keep their HTML content type.
Content shown only on one page	Components on that page	One component is created for each referenced item (the <code>foreach</code> pattern).
Media asset (blob attribute)	<code>dam:Asset</code>	Metadata is mapped to Dublin Core and custom fields.
Asset reference attribute	Path property	Examples are <code>fileReference</code> , <code>linkURL</code> and fragment references. The value is resolved from the planned path map.
Named association	List component, fragment multi-reference or repeated components	Mapped per association name.
Webreference / Satellite URL	Redirect rule	Old URL mapped to the new public URL.
Template, <code>CSElement</code> , <code>SiteEntry</code>	AEM components and templates (rebuilt)	Code, not content. Excluded from migration.
Dimensions / translations	Language copies and MSM	Future phase.

Component design

5.1 Extraction

- **Interfaces.** The classic REST resources (/REST/sites/{site}/types/{type}/..., WCS 11g and 12c) and the 12c resources/v1 API. Which one is used is a configuration choice.
- **Authentication.** A CAS multiticket (the WCS default) or HTTP Basic, using a read-only service account. Expired tickets are renewed automatically.
- **Traversal.** Each configured asset type is paged through with the search resource, and the full asset is then fetched. Binaries are downloaded and stored next to the raw payload.
- **Site plan.** Taken from the navigation resource. If that resource is unavailable, a SitePlanTree database extract (id, parent, order) is used instead.
- **Resilience.** The process can resume: assets already extracted are skipped unless a forced refresh is requested. Failed requests are retried with backoff, and permanent failures are logged per asset.
- **Source protection.** Configurable request throttling keeps load low on a production WCS.

5.2 Normalization and the canonical asset model

Raw payloads differ between WCS versions and patch levels. The normalization layer converts them into one canonical model, which is the only contract the transformation stage depends on. References and binaries use explicit markers, so they are never confused with plain strings.

```
{ "id": "Page:1330000000004", "type": "Page", "subtype": "ServicePage", "name": "Cardiology",
  "attributes": { "title": "Cardiology",
    "heroImage": { "$asset": "Image:13300000000102" },
    "brochure": { "$blob": { "file": "...", "filename": "x.pdf",
      "mimetype": "application/pdf" } } },
  "associations": { "related": [ { "$asset": "Page:1330000000005" } ] },
  "webreferences": [ "/services/cardiology" ] }
```

Why a canonical model?

It is the only contract the transformation stage depends on. When WCS payload formats change between versions or patch levels, only the normalization layer needs adjusting.

Transformation and link rewriting

5.3 Transformation engine

The engine works in two passes.

1 Plan

It resolves a mapping rule for every asset, matching on type and subtype. It assigns each mapped asset its AEM path: pages follow the site plan, names become collision-safe slugs, and pages not in the site plan go to a holding folder. The result is a path map from WCS id to AEM path.

2 Build

It generates the JCR nodes (pages, components, Content Fragments, assets) by evaluating the rule expressions. Asset references become AEM paths through the path map, rich text is rewritten, and every node is stamped with its WCS source id.

A mapping rule declares a target (page, contentFragment, damAsset, inline or skip) together with its settings. A value is an expression of the form `source|filter`:

Expression part	Options
Sources	attr:<name>, field:<name>, assoc:<name>, const:<value>, self, webref, page, <source>
Reference filters	ref (gives the AEM path), link (gives the URL, adding .html for pages), title
Content filters	richtext (rewrites WCS markup), text (strips tags)
Type filters	date, bool, long, double
String and list filters	slug, lower, upper, strip, truncate, prefix, suffix, first, join, split, default

Appendix A shows a complete page rule.

5.4 Rich text and link rewriting

WCS markup	Result in AEM
Embedded link: <code>_CSEMBEDTYPE_=internal&_cid=...&_c=...</code> (as an href or as attributes)	A link to the target page's public URL, or to the DAM path for assets. WCS marker attributes are removed.
Embedded inclusion: <code></code>	An <code></code> for images. A link for other asset types.
Satellite URL: <code>/cs/Satellite?c=...&cid=...</code>	The new URL of the target.
BlobServer image URL in markup	Left unchanged and reported, because the blob id cannot be traced back to an asset.
Link to content that is not migrated	Left unchanged and reported for editorial follow-up.

Packaging and DAM ingestion

5.5 Package Builder

1 Vault serialization

Writes the content tree in FileVault (vault) serialization format, with correct JCR property typing and escaping.

2 Filters from real roots

Writes `filter.xml` from the roots actually produced (content root, DAM root, fragment root). The default mode is update, which overwrites the migrated nodes and never deletes content that is not in the package.

3 Typed parent nodes

Includes placeholder definitions for parent nodes above the filter roots, so the install never creates untyped folders under `/content`.

4 Versioned, splittable packages

Packages are versioned for each run and can be split by site section or content type to keep their size manageable.

5.6 DAM ingestion strategy

Option	How it works	Recommended for
Binaries in package	Binaries are serialized into the content package as each asset's original rendition.	AEM 6.5 with small to medium asset volumes. One artifact to install.
API upload	The DAM Ingestion component uploads binaries through the Assets HTTP API (AEM 6.5) or direct binary upload (AEM as a Cloud Service). It then applies metadata.	AEM as a Cloud Service (assets get processed by Asset Compute) and large asset volumes.

Recommendation

Use API upload for AEM as a Cloud Service. Upload the binaries before installing the content package, so that every image and document reference resolves as soon as the content is installed.

URLs, SEO and data quality

6. URL management and SEO

- A redirect map (old URL to new URL) is generated on every run from webreferences, legacy Satellite URL patterns and the new public URLs.
- New public URLs are derived from the AEM path, using the agreed Sling mapping and dispatcher rules (for example `/content/<site>/us/en → /`).
- The map is delivered as CSV for review and as a rewrite map for the dispatcher or CDN. Satellite URLs carry their identity in the query string, so they need a query-string-aware rule or a small redirect handler.
- Page titles, descriptions and SEO attributes are mapped to page properties, so search metadata is kept.

7. Data quality, validation and reconciliation

Every transformation run produces a migration report. The pipeline can also run in strict mode, where any unmapped type or open issue fails the run. This lets trial runs be gated until the report is clean.

Report item	Meaning	Resolution
Unmapped type	An asset type or subtype has no mapping rule.	Add a rule, or exclude the type explicitly.
Missing reference	A referenced asset was not extracted or not mapped.	Extract the type, or accept and document the gap.
Unresolved link or inclusion	Rich text points at content that is not migrated.	Fix it in WCS before the final run, or in AEM afterwards.
BlobServer reference	A binary URL is embedded directly in markup.	Replace it with a proper asset embed.
Orphan page	A page is not placed in the site plan.	Place it in WCS, or confirm it should be excluded.
Missing binary	A blob failed to download or is empty.	Re-extract the asset.

7.1 Reconciliation after installation

- Compare page, fragment and asset counts in AEM with the report.
- Sample-check pages on author and publish against the WCS originals.
- Run a link check on the publish tier.
- Replay a sample of the redirect map and confirm 301 responses.
- Use the path map and the source-id property on every node to audit any item back to WCS.

Security and non-functional requirements

8. Security

1 Least-privilege accounts

Use a read-only service account on WCS, and a dedicated AEM service user with rights limited to the migration paths.

2 Runtime credentials

Inject credentials at runtime (environment variables or a secrets manager). Never write them to configuration files, logs or the staging store.

3 Network and transport

Run the migration host inside the corporate network or a VPN. Use TLS for all WCS and AEM traffic.

4 Staging store protection

Protect the staging store with access controls, because it holds a full copy of the site content and binaries. Delete it, or archive it under retention policy, after decommissioning.

5 Controlled installs

Content packages are installed through Package Manager by authorized administrators only. Every install is logged by AEM.

9. Non-functional requirements

Area	Design response
Volume and performance	Extraction can be throttled and resumed. Transformation runs offline against the staging store. Packages and asset uploads can be split into batches.
Repeatability	Runs are idempotent with update semantics, so trial, delta and final runs follow the same process.
Delta migration	Changed assets are re-extracted (by modified date, or forced refresh), then re-transformed and installed as a new package version.
Auditability	Each run keeps its raw payloads, mapping version, build output, report and path map. Every AEM node carries its WCS source id.
Maintainability	Site-specific logic lives in configuration. Changes to WCS payload formats affect only the normalization layer.
Portability	AEM 6.5 and AEM as a Cloud Service are both supported through standard package and Assets API interfaces.

Migration execution approach

Seven phases take the migration from discovery to WCS retirement. Each phase has an exit criterion that gates the next.

Phase	Key activities	Exit criteria
1. Discovery	<ul style="list-style-type: none"> Inventory the asset types, subtypes and volumes. Confirm REST access and the payload formats. Extract the site plan and the URL patterns. 	Inventory signed off; access confirmed
2. AEM foundation	<ul style="list-style-type: none"> Build the components, editable templates and Content Fragment models. Agree the content and DAM structure. 	Foundation deployed to the target author
3. Mapping	<ul style="list-style-type: none"> Define the mapping configuration type by type. Review it with content owners. 	Every type is mapped or explicitly excluded
4. Trial runs	<ul style="list-style-type: none"> Run a full migration to a QA author. Work through the report until it is clean. Run UAT on the migrated content. 	Clean report; UAT sign-off
5. Final run	<ul style="list-style-type: none"> Freeze WCS content. Run a final extraction and transformation. Do the DAM upload, install the package and publish. 	Reconciliation passed; smoke tests green
6. Cutover	<ul style="list-style-type: none"> Deploy the redirect rules. Switch DNS. Monitor 404s and fix any gaps. 	Traffic on AEM; 404 rate within threshold
7. Decommission	<ul style="list-style-type: none"> Archive the staging store and the path map. Retire WCS. 	Archive verified

Risks and open decisions

11. Risks and mitigations

Risk	Impact	Mitigation
WCS REST payload formats differ between versions and patch levels	High	Validate against the client's WCS during discovery. Only the normalization layer needs adjusting.
Rich text holds many links to content that is not migrated, or BlobServer URLs	Medium	The report lists each one. Fix them in the source before the final run, or editorially after it.
Large asset volumes	Medium	Use API upload and split packages. Schedule the uploads outside peak hours.
Content changes during the project	Medium	Use delta runs and an agreed content freeze for the final run.
AEM components or models not ready for trial runs	High	Make the foundation build (phase 2) a gate for the first trial run.
Multilingual requirements found late	Medium	Confirm language scope during discovery and plan the language-copy design early.

12. Open decisions

#	Decision	Options	Owner
D1	AEM target platform	AEM 6.5 or AEM as a Cloud Service	Client architecture
D2	DAM ingestion method	Binaries in package, or API upload	Migration architect
D3	Content Fragment or component for each content type	Decided per type during mapping	Content owners
D4	Handling of orphan pages	Migrate to a holding folder, or exclude	Content owners
D5	Redirect implementation	Dispatcher rewrite map, CDN, or ACS Commons Redirect Manager	Platform team

Future phases and mapping example

13. Future phases

1 Multilingual sites

Map WCS Dimensions and DimensionSets to AEM language roots, with language copies and MSM live copies.

2 Personalisation

Re-implement WCS Engage segments and recommendations with Adobe Target or AEM ContextHub.

3 Publish state

Carry WCS approval state across and publish selectively after migration.

Appendix A. Example mapping rule

A page rule for a service-page subtype. It sets the page template and properties, and builds an image component (only when a hero image is present) and a rich text component.

```
"Page/ServicePage": {
  "target": "page",
  "template": "/conf/site/settings/wcm/templates/content-page",
  "resourceType": "site/components/page",
  "properties": { "jcr:title": "attr:title", "jcr:description": "attr:summary" },
  "components": [
    { "name": "image", "resourceType": "site/components/image", "when": "attr:heroImage",
      "properties": { "fileReference": "attr:heroImage|ref" } },
    { "name": "text", "resourceType": "site/components/text",
      "properties": { "text": "attr:body|richtext", "textIsRich": "const:true|bool" } } ] }
```

A Content Fragment rule for a flex asset type:

```
"Doctor": {
  "target": "contentFragment",
  "model": "/conf/site/settings/dam/cfm/models/doctor",
  "folder": "doctors",
  "title": "attr:fullName",
  "elements": { "fullName": "attr:fullName", "specialty": "attr:specialty",
    "bio": "attr:bio|richtext", "photo": "attr:photo|ref", "startDate": "attr:startDate|date"
  } }
```

Glossary

Term	Definition
WCS	Oracle WebCenter Sites, the source content management system.
Asset / subtype	The WCS unit of content and its definition (flex definition), which sets its attributes.
Site plan	The WCS page hierarchy, stored in the SitePlanTree table.
Webreference	A WCS vanity URL attached to an asset.
Satellite URL	The legacy WCS dynamic URL (/cs/Satellite?c=...&cid=...).
AEM	Adobe Experience Manager, the target platform (6.5 or as a Cloud Service).
Content Fragment (CF)	Structured, presentation-independent content in AEM, based on a CF model.
DAM	AEM Digital Asset Management (AEM Assets).
FileVault / content package	AEM's packaging format for repository content, installed through Package Manager.
JCR	Java Content Repository, the AEM content store.
Canonical asset model	The version-independent representation of WCS assets used between normalization and transformation.

Questions on this design?

Send comments and decisions on the open items (D1–D5) to the Onwardpath migration team.

