Multitenancy and Concurrent Development in AEM

Ian Reasor, Technical Architect, Adobe Partner Experience
Tim Donovan, Technical Architect, Adobe Partner Experience
Opkar Gill, Technical Architect, Adobe Consulting

Introduction

When multiple teams are deploying their code to the same AEM environments, there are practices they should follow to ensure that teams can work as independently as possible, without stepping on other teams' toes. Although they can never be fully eliminated, these techniques will minimize cross-team dependencies. For a concurrent development model to be successful, good communication between the development teams is critical.

Additionally, when multiple development teams work on the same AEM environment, there is likely some degree of multi-tenancy at play. Much has been written on the practical considerations of attempting to support multiple tenants in an AEM environment, especially around the challenges faced when managing governance, operations, and development. This paper explores some of the technical challenges around implementing AEM in a multitenant environment, but many of these recommendations will apply to any organization with multiple development teams.

It is important to note up front that while AEM can support multiple sites and even multiple brands being deployed on a single environment, it does not offer true multitenancy. Some environment configurations and systems resources will always be shared across all sites that are deployed on an environment. This paper provides guidance for minimizing the impacts of these shared resources and offers suggestions to streamline communication and collaboration in these areas.

Benefits and Challenges

There are many challenges to implementing a multi-tenant environment. These include:

- Additional technical complexity
- Increased development overhead
- Cross-organization dependencies on shared resources
- Increased operational complexity

Despite the challenges faced, running a multitenant application does have benefits, such as:

- Reduced hardware costs
- Reduced time to market for future sites
- Lower implementation costs for future tenants

- Standard architecture and development practices across the business
- A common codebase

If the business requires true multitenancy, with zero knowledge of other tenants and no shared code, content, or common authors, then separate author instances are the only viable option. The overall increase in development effort should be compared against the savings in infrastructure and license costs to determine if this approach is the best fit.

Development Techniques

Managing Dependencies

When managing Maven project dependencies, it is important that all teams use the same version of a given OSGi bundle on the server. To illustrate what can go wrong when Maven projects are mismanaged, we present an example:

Project A depends on version 1.0 of the library, *foo*; foo version 1.0 is embedded in their deployment to the server. Project B depends on version 1.1 of the library, *foo*; foo version 1.1 is embedded in their deployment.

Furthermore, let's assume that an API has changed in this library between versions 1.0 and 1.1. At this point, one of these two projects will no longer work properly.

To address this concern, we recommend making all Maven projects children of one parent *reactor* project. This reactor project serves two purposes: it allows for the building and deployment of all projects together if so desired, and it contains the dependency declarations for all the child projects. The parent project defines the dependencies and their versions while the child projects only declare the dependencies that they require, inheriting the version from the parent project.

In this scenario, if the team working on Project B requires functionality in version 1.1 of *foo*, it will quickly become apparent in the development environment that this change will break Project A. At this point the teams can discuss this change and either make Project A compatible with the new version or look for an alternative solution for Project B.

Note that this does not eliminate the need for these teams to share this dependency–it just highlights issues quickly and early so that teams can discuss any risks and agree on a solution.

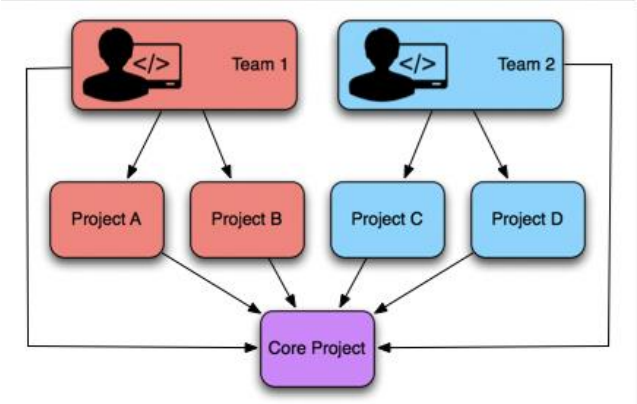
Preventing Code Duplication

When working on multiple projects, it is important to ensure that code is not duplicated. Code duplication increases the likelihood of defect occurrences, the cost of changes to the system, and overall rigidity in the code base. To prevent duplication, refactor common logic into reusable libraries that can be used across multiple projects.

To support this need, we recommend the development and maintenance of a *core* project that all teams can depend on and contribute to. When doing so, it is important to ensure that this core project does not, in turn, depend on any of the individual teams' projects; this allows for independent deployability while still promoting code reuse.

Some examples of code that is commonly in a core module include:

- System-wide configurations such as:
  - OSGi configurations
  - Servlet filters
  - ResourceResolver mappings
  - Sling Transformer pipelines
  - Error Handlers (or use the ACS AEM Commons Error Page Handler[1])
  - Authorization servlets for permission-sensitive caching
- Utility classes
- Core business logic
- Third-party integration logic
- Authoring UI overlays
- Other customizations required for authoring, such as custom widgets
- Workflow launchers
- Common design elements that are used across sites



Modular Project Architecture

This does not eliminate the need for multiple teams to depend on and potentially update the same set of code. By creating a core project, we have decreased the size of the codebase that is shared between teams–decreasing but not eliminating the need for shared resources.

To ensure that the changes made to this core package do not disrupt the functionality of the system, we recommend that a senior developer or team of developers maintain oversight. One option is to have a single team that manages all the changes to this package; another is to have teams submit pull requests that are reviewed and merged by these resources. It is important that a governance model is designed and agreed to by the teams and that developers follow it.

[1] https://adobe-consulting-services.github.io/acs-aem-commons/features/errorpagehandler.html

Managing Deployment Scope

As different teams deploy their code to the same repository, it is important that they do not overwrite each other's changes. AEM has a mechanism to control this when deploying content packages, the filter.xml file. It is important that there is no overlap between filter.xml files, otherwise one team's deployment could potentially erase another team's previous deployment. To illustrate this point, see the following examples of well-crafted vs. problematic filter files:

/apps/my-company	*vs.*	*/apps/my-company/my-site*
/etc/clientlibs/my-company	*vs.*	*/etc/clientlibs/my-company/my-site*
/etc/designs/my-company	*vs.*	*/etc/designs/my-company/my-site*

If each team explicitly configures their filter file down to the site(s) that they are working on, each team can deploy their components, client libraries, and site designs independently without erasing each other's changes.

Because it is a global system path and not specific to one site, the following servlet should be included in the core project, as changes made here could potentially impact any team:

/apps/sling/servlet/errorhandler

Overlays

Overlays are frequently used to extend or replace out of the box AEM functionality, but using an overlay affects the entire AEM application (that is, any functionality changes will be available for all tenants). This would be further complicated if tenants had different requirements for the overlay. Ideally, business groups should work together to agree on the functionality and appearance of AEM's administrative consoles.

If consensus cannot be reached amongst the various business units, a possible solution would be to simply not use overlays. Instead, create a custom copy of the functionality and expose it via a different path for each tenant. This allows each tenant to have a completely different user experience, but this approach increases the cost of implementation and subsequent upgrade efforts as well.

Workflow Launchers

AEM uses workflow launchers to automatically trigger workflow execution when specified changes are made in the repository. AEM provides several launchers out of the box, for example, to execute rendition generation and metadata extraction processes on new and updated assets. While it is possible to leave these launchers as-is, in a multitenant environment, if tenants have different launcher and/or workflow model requirements, then it is likely that individual launchers will need to be created and maintained for each tenant. These launchers will need to be configured to execute on their tenant's updates while leaving content from other tenants untouched. Easily accomplish this is by applying launchers to specified repository paths that are tenant-specific.

Vanity URLs

AEM provides vanity URL functionality that can be set on a per-page basis. The concern with this approach in a multi-tenant scenario is that AEM does not ensure uniqueness between the vanity URLs configured in this manner. If two different users configure the same vanity path for different pages, unexpected behavior can be encountered. For this reason, we recommend using mod_rewrite rules in the Apache dispatcher instances, which allow for a central point of configuration in concert with outbound-only Resource Resolver rules.

Component Groups

When developing components and templates for multiple authoring groups, it is important to make effective use of componentGroup and allowedPaths properties. By leveraging these effectively with site designs, we can ensure that authors from Brand A only see components and templates that have been created for their site while authors from Brand B only see theirs.

Testing

While a good architecture and open communication channels can help prevent the introduction of defects into unexpected areas of the site, these approaches are not foolproof. For this reason, it is important to fully test what is being deployed on the platform before releasing anything into production. This requires coordination between teams on their release cycles and reinforces the need for a suite of automated tests that cover as much functionality as possible. Additionally, because one system will be shared by multiple teams, performance, security, and load testing become more important than ever.

Operational Considerations

Shared Resources

AEM runs within a single JVM; any deployed AEM applications inherently share resources with each other, on top of resources already consumed in the normal running of AEM. Within the JVM space itself, there will be no logical separation of threads, and the finite resources available to AEM, such as memory, CPU, and disk i/o will also be shared. Any tenant consuming resources will inevitably affect other system tenants.

Performance

If not following AEM best practices, it is possible to develop applications that consume resources beyond what is considered normal. Examples of this are triggering many heavy workflow operations (such as DAM Update Asset), using MSM push-on-modify operations over many nodes, or using expensive JCR queries to render content in real-time. These will inevitably have an impact on the performance of other tenant applications.

Logging

AEM provides out of the box interfaces for robust logger configuration that can be used to our advantage in shared development scenarios. By specifying separate loggers for each brand, by package name, we can achieve some degree of log separation. While system-wide operations like replication and authentication will still be logged to a central location, non-shared custom code can be logged separately, easing monitoring and debugging efforts for each brand's technical team.

Backup and Restore

Due to the nature of the JCR repository, traditional backups work across the entire repository, rather than on an individual content path. It is therefore not possible to easily separate backups on a per-tenant basis. Conversely, restoring from a backup will rollback content and repository nodes for all tenants on the system. While it is possible to perform targeted content backups, using tools such as VLT, or to cherry pick content to restore by building a package in a separate environment, these approaches do not easily encompass configuration settings or application logic and can be cumbersome to manage.

Security Considerations

ACLs

It is, of course, possible to use Access Control Lists (ACLs) to control who has access to view, create, and delete content based on content paths, which requires the creation and management of user groups. The difficulty in maintaining the ACLs and groups depends on the emphasis on ensuring each tenant has zero knowledge of others and whether the deployed applications rely on shared resources. To ensure efficient ACL, user, and group administration, we recommend having a centralized group with the necessary oversight to ensure that these access controls and principals overlap (or do not overlap) in a way that promotes efficiency and security.

Administrative Sessions

Elevated access to the repository is often needed within the logic of a custom AEM application to perform complex repository operations. Best practice dictates that administrator sessions should *not* be used. Instead, service accounts should be created and given enough access rights to perform the intended operation. In practice however, developers often use a single service account that has administrator access. In a multi-tenant scenario, this session would be able to bypass any ACL security that has been setup, underscoring the need for implementation of best practices in session management.

Full-Text Indexing

When using an external, third-party indexing solution like Apache Solr, ACLs are not evaluated—therefore content from across different tenants may be visible in search results. Performing repository

queries should never be done as an administrator.

User Authentication

AEM supports the implementation of multiple authentication handlers, meaning multiple identity management systems can be used for authentication. Ideally, though, a single system should be used when possible to reduce implementation complexity. Multiple LDAP configurations are supported, allowing authentication against multiple LDAP servers. Alternately, SAML integration can be leveraged, which allows the use of separate authentication handlers for different AEM repository content paths, a perfect solution for separating concerns in a multitenant environment.

Servlets

When following best practices for servlet definition, servlets are defined by the resource type that they apply to. When using this approach to servlet registration, ACLs will be evaluated as part of the request, and no special security considerations are required. If, however, path-based servlet registration is employed, the servlet will exist outside the context of any resource in the repository; this prevents the application of ACLs to the servlet request and requires additional considerations from the developer.

Governance

Stakeholder Alignment

In multi-tenant scenarios, it is important that stakeholders from the various business units that leverage the system stay in contact and are aligned on deployment dates, maintenance windows, and platform patches and upgrades. Because system-wide configurations like replication agents, dispatcher configuration, and others can impact all teams that depend on the system, it is important to include representatives from each of these groups in the decision-making process.

Many organizations find it useful to have a lead product owner who works to ensure that the overarching vision of the project is met by the individual teams during implementations. Additionally, a lead developer or architect can ensure that each team is contributing to a sensible system and software architecture. This can help the overall implementation run as a cohesive whole rather than a sum of disparate parts.

Release Cycles

Releases often require downtime of both author and publish instances, especially when deploying system upgrades, and this will affect all users on an instance, even if the release is only applicable to a single tenant. Additionally, when testing changes made by a development team, it is important to test the impact of these changes across the entire environment. Therefore, a common release methodology and schedule should be adopted.

Support

With multiple tenants developing and deploying custom code, support becomes a challenge, as no one application team will want total ownership of the environment. Downtime or performance problems can become extremely political. Furthermore, if tenants do not follow a single model of development, there is a greater burden on any support team to understand the differing approaches to both deployment and operations. This underscores the need for a lead developer who can help diffuse these situations and diagnose the source of issues that may be cross-cutting.

Encouraging Reuse

Allowing each tenant of the system to create and maintain a diverse set of components and services can quickly lead to issues and an increased cost of ownership. In Adobe's experience, the most successful multitenant sites are deployed with a single codebase used across all sites, with simple style variations. While this might not be possible for all use cases, it is an ideal to strive for. Work across business units during the requirements definition phase and find opportunities to reuse existing templates and components. These efforts can lower the cost of development, and the system will be easier to support in the future.

Improving Visibility

While it is important to maintain team autonomy to ensure efficiency, disparate teams should at least understand what other teams are working on or have delivered at a high level. For this reason, we encourage members of other teams to be included in sprint demos.

While not required, it is often helpful for members of all teams to have at least read-level access to other teams' projects. This will improve discoverability and can reduce code duplication. A developer cannot know that another team has already written the logic that they need if they cannot see what the other team has developed. In addition to greater code reuse, providing visibility also allows developers to ensure that they are not stepping on other teams' toes when configuring globally applicable settings, such as servlet paths. Visibility saves developers from redundant work and improves efficiency across organizations.

Checklist

Use the following checklist to ensure that you have considered the important pieces of a multitenant AEM deployment:

- ☐ A strong architecture/central team in place
- ☐ High-level executive buy-in
- ☐ Business unit level buy-in
- ☐ Governance and processes are in place
- ☐ An emphasis on testing
- ☐ Agreement on release/upgrades and schedules
- ☐ Common infrastructure and operations (storage, backup/restore)

- ☐ Single place (e.g. wiki) to document the platform and each individual application
- ☐ Common development and deployment strategy
- ☐ Development of a core platform of services used by all sites
- ☐ Extensive product knowledge by development and operation teams

Conclusion

When planning, multi-team coordination, and collaboration are treated as critical aspects of a development model, good outcomes follow. Overall, from both a technical and a business standpoint, concurrent development practices that focus on communication and visibility benefit all our stakeholders, from the business unit to developers, and from partners to customers.

Additional Resources

Blog posts from Jörg Hoh on multitenancy:
https://cqdump.wordpress.com/2015/05/04/the-problems-of-multi-tenancy-governance/

https://cqdump.wordpress.com/2015/06/23/the-problems-of-multi-tenancy-the-development-model/

https://cqdump.wordpress.com/2015/10/16/the-problems-of-multi-tenancy-tenant-separation-and-friendly-tenants/