AEM Code Promotion and Content Synchronization Best Practices

Ian Reasor, Technical Architect, Adobe Partner Experience

Introduction

When considering the movement of content through environments in an AEM topology, we need to look at it from two perspectives. The code that is running a website will start on a developer's local sandbox and be promoted through the integration and test environments, eventually ending up in production. Conversely, the content that is being authored for the website will be created in production and be synchronized to these same staging and test environments. This same code/content flow applies to implementations of AEM Assets and AEM Forms.

This document outlines some best practices for promoting code from developers’ workstations up to the production environment as well as keeping content in sync throughout your AEM environments. There are many options for synchronizing content between environments, so we will look into each of them and weigh their pros and cons.
**Recommended Environment Topology**

In an ideal scenario, Adobe recommends the use of five distinct environments for code promotion and testing. These environments are listed below along with their purposes.

- **Developer Local Sandbox** – these environments run locally on a developer's workstation and allow for testing and debugging as part of the development process.
- **Development Integration** – an environment for automated build deployment and execution of integration tests.
- **QA** – an environment where code can be continuously tested. This is where the QA team will perform the bulk of their validation testing.
- **Staging/UAT** – an environment where a build candidate can be deployed for user acceptance testing as well as security and performance testing. The hardware on this environment should match production as closely as possible.
- **Production** – the live environment where the website is authored and served from.

**Promoting Code Through Environments**

As code changes are made, the build will be promoted through each of these environments, encountering checkpoints along the way. These checkpoints allow us to ensure that the staging environment is stable enough for the end users while enabling the developers and QEs to “move fast and break things.” The process can be visualized as such:
Code Branches

To support concurrent activities such as applying hotfixes to the production environment, fixing bugs discovered during user acceptance testing, and active development on a future release, we recommend using a Git-based repository along with the Gitflow system for branching and merging. Read more about Gitflow at http://nvie.com/posts/a-successful-git-branching-model/.

In general, code that lives on developers' local sandboxes is checked into feature branches. When code is ready for deployment to the development integration and QA environments, it is first merged into the develop branch, often by using a pull request. When code is to be deployed to staging, it will be branched into a release branch, and the version will be set to a non-SNAPSHOT release version before deployment to the environment. When code is ready to be deployed to production, it will be merged into master. This allows us to always have a branch of the code that is currently live on the site, no matter where we are in the development process.

Continuous Integration

We recommend the use of a continuous integration (CI) solution when building and deploying your AEM project. Not only does using a system such as Jenkins or TravisCI make your build process easily repeatable and testable, but with AEM's Maven-centric approach, it is easy to implement as well. The small amount of time invested up front to configure continuous integration will pay for itself in saved time doing deployments over the course of a project. For most projects, we recommend the following jobs:

- **Push to Dev** – This job polls the source code repository, and whenever a new check-in is detected, it will download the latest code, compile it, run unit tests, run static code analysis, deploy to the dev integration environment, and run the automated integration tests on it.
- **Push to QA** – This job can be executed by the QE team when they desire a new build to perform testing on. It will take the last successful build from the push to dev job and deploy it to the QA environment.
- **Push to Staging** – This job is executed by an administrator when a build is ready for User Acceptance Testing. The job will pull down the release branch specified by the administrator at run time, build it, run unit tests, and deploy it to the Staging environment.
- **Content Refresh** – Depending on the content synchronization method chosen, jobs are sometimes implemented to run scripts that enable content synchronization from production to lower environments.

It is possible to create a CI job that can perform production deployments, but this job would be more complex than these other examples. This is due to the need to deploy to some of the servers in the environment and validate the deployment before deploying to the rest of the environment. That being said, scripting out deployments, especially production ones, is widely considered to be a best practice as it eliminates the risk of user error during the process.

---


May 7, 2018
For teams that have mature CI adoption practices, they may consider using Jenkins Pipelines to fully automate a Continuous Delivery process. For more information on Jenkins Pipelines, see https://jenkins.io/doc/book/pipeline/.

**Static Code Analysis**

We recommend implementing a static code analysis tool to ensure that code quality remains consistent over time and potential errors can be caught early. SonarQube ([https://www.sonarqube.org/](https://www.sonarqube.org/)) is a popular option that offers a free version. Cognifide offers AEM Rules for SonarQube ([https://github.com/Cognifide/AEM-Rules-for-SonarQube](https://github.com/Cognifide/AEM-Rules-for-SonarQube)) that can extend SonarQube’s default scanning rules for some common AEM patterns.

**Repository Management**

In instances where build artifacts need to be reused across multiple projects or have different versions maintained for later use, a repository manager can be helpful. These can be integrated directly into the build process so that release artifacts and even snapshots can be stored and used in future Maven builds and CI jobs. The most popular repositories on the market are Artifactory\(^2\) and Nexus\(^3\), but there are many options available.

**Rolling Production Deployments**

Before an implementation has been brought live, deploying to production is the same as deploying to any other environment. Since end users are not yet accessing the servers, we can push out code and author content at whatever pace is convenient and start sending traffic to the servers when we think that the environment is ready. However, when the environment is already a “live” environment, there are additional factors that must be considered to ensure continued operation of the website during the deployment.

For the sake of simplicity in this discussion, we will illustrate these concepts using a common deployment scenario of one author server with two publish servers and two dispatchers, behind a load balancer.

\(^2\) [https://www.jfrog.com/artifactory/](https://www.jfrog.com/artifactory/)

\(^3\) [https://www.sonatype.com/nexus-repository-sonatype](https://www.sonatype.com/nexus-repository-sonatype)
**Phase 1 – Normal Operating Conditions**

In our normal running configuration, the load balancer is splitting traffic between dispatchers 1 and 2. Each of these dispatchers communicates with a single publish instance. The author is pushing content to both publish servers.

**Phase 2 – Internal Deployment**

During the first phase of the deployment, Dispatcher 1 is removed from the load balancer pool and the replication agent for Publisher 2 is disabled on the author instance. At this point, users of the website will only be served content from Dispatcher 2 and Publisher 2. We then deploy our code changes to the author instance and Publisher 1. Users can then test the website via an internal-only URL to Dispatcher 1 and validate functionality.
Content Synchronization

Many approaches can be taken to synchronize content from production environments to lower environments. One thing is clear, though: for accurate testing to be performed on code changes, it is imperative that tests are run on production-like content. For this reason, we highly recommend that a content synchronization process is defined and adhered to on a regular schedule. The methods below are several that we have seen our clients be successful with in the past. The best approach for any given client will vary based on how often their content is updated and the size of their repository.

In all of the cases listed below, we are assuming data synchronization from a production author to a lower author environment. Once the content has been copied into the author environment, replication should take place in the lower environment for use on the lower environment's publish instance.

Backup/Restore

The simplest approach to synchronizing content between instances is to take a backup of the production author instance and restore it to the lower environment. Once the restore is complete, update the lower instance's configurations to ensure that it does not interact with other production environments such as publish instances, translation vendors, or other Adobe Experience Cloud solutions. Common areas for configuration include replication agents and web service endpoints. As an added layer of protection, it is best to set up the network topology such that the lower authoring environment cannot replicate to the production publish servers.

Pros:
- Relatively straightforward
- Allows for constant testing of the backup/restore processes
- Ensures that all content and configurations are included

Phase 3 – Deployment Completion

At this point, we can switch the load balancer over to serve live traffic from Dispatcher 1 and remove Dispatcher 2 from the pool while we upgrade Publisher 2. The replication agent on the author can also be re-enabled. Once the testers have validated that Dispatcher 2 is working properly, we add it back to the pool and our deployment is complete.
Cons:

- Because the entire repository is being moved, the size of the content being moved is larger than necessary.
- Updating configurations after restoring the backup is a manual step and thus error-prone.
- The lower environment must be brought offline during the restore operation.

Content Packages

For customers who have a small amount of data that will need to be synchronized, content packages can sometimes be employed. We first create a content package on the production author environment with the filter configured for the authored content that we wish to synchronize. After building and downloading the package, it can then be uploaded and installed in any lower environments.

Pros:

- Straightforward approach
- Package definitions allow us to pick and choose which content to refresh
- Only the needed content is copied over

Cons:

- Only useful for situations in which there is a small amount of content to synchronize as content packages can be unwieldy when they are more than a couple GB in size.
VLT-RCP
Many developers are familiar with FileVault (VLT) as a method of synchronizing changes between the JCR and their local filesystem. VLT also includes the option to synchronize changes over RCP. Either through the vlt command line or through the vlt-rcp UI tool, content can be extracted from the production instance and copied into a lower environment. Since this does put some load on the resources of the servers involved, when using this approach, it is recommended to copy from production to staging and then from staging to QA, etc. This will prevent an unnecessary load from being placed on the production author environment.

The VLT-RCP UI is included in the ACS AEM Tools project. Instructions on running the tool from the command line are available in the Jackrabbit documentation.

Pros:
• Works well for large amounts of content
• Only specified content paths are copied over

Cons:
• Can be time consuming

Replication
The final recommended approach involves adding a replication agent to the production author instance to replicate content to the staging author. This setup can be repeated in lower environments, with the staging author replicating to the QA author, etc. All content replicated on the production author will automatically be pushed to the staging author. Pass-through replication can be configured to automatically activate this content to the staging publish instances and lower environments. Note that when using this method, only activated content will be synchronized to lower environments.

Pros:
• Only updated content is copied over.
• Content is copied as it is activated in production rather than in batches.

Cons:
• Adding a replication agent on the production author increases the overhead required to publish content.

---

5 http://jackrabbit.apache.org/filevault/rcp.html
Content Sync Method Comparison

These various synchronization methods are laid out in the following table to allow for easier comparison of the various benefits that they each offer:

<table>
<thead>
<tr>
<th>Method</th>
<th>Easy to implement</th>
<th>Allows testing existing processes</th>
<th>Ensures all content is included</th>
<th>Allows for selective content synchronization</th>
<th>Appropriate for large amounts of content</th>
<th>Content activated as it is updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backup/Restore</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Content Packages</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>VLT-RCP</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Replication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

You can also reference the following decision tree when selecting the right content synchronization method for your use cases:

![Decision Tree Diagram]