William Thayer
Senior Consultant and MBA
Avalon Consulting, LLC

Joshua S. Simon
Senior Systems Administrator
University of Michigan

Web Content Management Development and Redundancy

The University of Michigan,
College of Literature, Science, and the Arts

Avalon Consulting, LLC

January 31, 2013

Contents

Introduction

As information systems (IS) grow, complexity also increases. High levels of visibility and impact become moving targets of risk and failure. There are many aspects of failure such as budgets or schedules or worse failures where the system is never used. This document explores risk and tradeoffs with enterprise level IS development.

This document discusses a web content management system (CMS) at the University of Michigan's College of Literature, Science, and Arts (LSA). This high priority system enables a world class web presence for the College.

College of Literature, Science, and the Arts

LSA is the heart of the University of Michigan (U-M). It is the largest of U-M's 19 schools and colleges with an endowment over $716M and 19,755 enrolled students as of the Fall 2012 term. In 2012, U-M was ranked 18th among the best universities in the world by The Times of London and 6th in the nation for undergraduate teaching by U.S. News & World Report. In terms of program rankings, LSA has 5 number one programs, 20 more ranked in the top five, and another 18 ranked in the top 10. It is consistently ranked in the top 10 for undergraduate education with over 100 degree programs offered in 75 departments and programs.

The College's website assets are vital. It is the first opportunity to make an impression. Even though the reputation of the College often precedes itself, it is the web where most students collect information to support their decision on where to go for school. Additionally, the web is where students collect program information, schedules, special events and news. It is frequently used for existing and potential students, researchers, and faculty. For this reason, the websites require a consistent, professional, and logical format to represent the College as the world leader it is.

History

In March of 2002, LSA determined it was necessary to evaluate a solution to consolidate disparate website systems into a central repository. A consulting firm was commissioned to perform an in-depth analysis of LSA's Internet infrastructure and recommend possible solutions for deployment in the summer of 2002. Based on their review, the College selected Vignette Content Management now owned by OpenText Corporation.

The College had several platforms where each department maintained static HTML pages along with Microsoft ASP or .NET pages. The environments were inconsistent and each department had different interpretations on styles, navigation, logos, and page structure. Additionally, each department had different code bases obstructing maintenance and delaying web page updates. Since the systems were spread among departments, content redundancy was an issue. For instance, many departments used similar logos but stored them in different folders. It quickly became apparent that change was needed.

LSA initiated the CMS project with the goal of hosting all departments on a single platform. This would standardize the look and feel of each department while increasing efficiencies by using a single system for web content management. On the surface, one might not perceive the complexity of such a project, but that sentiment fails to realize the scale. There are 75 departments at LSA with over 1,300 faculty and 2,200 administration staff. Each department has a unique message and perspective on how to best represent their programs and research. As a result, many departments historically had their own unique process for managing web content.

Initially, the College tried to provide each department their own custom site structure while using the consolidated content repository of a CMS. It rapidly became apparent that a consistent and standard code base was necessary to bring all of the departments together under a single sustainable platform.

Three years after enforcing a common code base, the CMS repository hosts over 100 live websites as of January 2013. The number of sites coming online is rapidly growing. While this success is admirable, remember that it took several years of planning and strategy with pilots and intense training.

There are three aspects of importance for this rapid expansion:

1. Enforce a common code base and standardization.
2. Expand a tier's clustered environment with growth.
3. Keep the code, configuration, and content consistent among three environments of Development, Quality Assurance, and Production. This point is explored further in the "Synchronization" section on page 5.

Some statistics of interest about the LSA web environment:

- 133,097 pieces of content
- Over 100 sites
- Over 8.7 million requests for over 47 GB of traffic per month

Architecture

The macro goal of a new CMS system was migration of all departmental micro sites. Initially, the Gayle Morris Sweetland Center for Writing was targeted as the first site in the most recent CMS version. This site provided an initial success and a common code base for jumpstarting other sites. The Center's web requirements were aligned with most other departments in the College. Their website was limited in scope and they agreed to work with the developers while understanding the risk of being the first mover. Risks included site disruptions or bugs while the team finalized the environment and code functions. Additionally, the Center was in the process of rebranding their communication materials and a new website was necessary. For these reasons, the Center agreed to be the first site in the new version of the CMS.

State-of-the-art hardware was acquired and set up after a thorough analysis. The architecture had built-in redundancy for planned and unplanned downtime of web application servers. Web traffic would continue to operate in situations where one server is offline. The hardware includes a redundant load balancer per data center, and at least five blade servers per tier. The Production and disaster recovery sites are in different geographically separate data centers. The LSA tier structure includes:

- Production for serving the live public sites
- Quality Assurance (QA) for testing and disaster recovery
- Development for developing new code or changing configurations

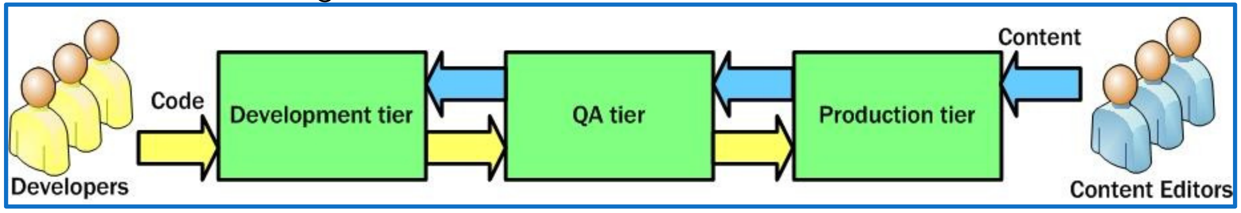
The QA environment's architecture is identical to and just as robust as Production. The QA environment is hosted in a separate data center and includes multiple servers with redundancy, load balancing, and clustering. This provides an opportunity for a fallback in the event of unforeseen issues on Production. It also provides confidence to developers and system administrators (SA) in the event of unexpected changes on the Production tier.

A lot of the difficulty with enterprise systems is the scope of impact if the system goes down. A system with high visibility and use sometimes obstructs project teams from reacting nimbly. Code releases can be delayed and configuration changes slowed. If there is a history of mishaps which result in unexpected outages, then a level of anxiety may surround the release process. This difficulty might delay changes necessary for the future and frustrate the project team.

Workflow

LSA has a workflow for managing both code and content. Code progresses downstream from Development to QA and after approval to Production. Conversely, content is entered by department representatives into the Production CMS. This content is then migrated back upstream to QA and Development. Figure 1 illustrates these concepts.

Figure 1
Code and content migration



Developers work on the Development tier while consistently unit testing new functionality or configuration changes. Once ready, code migrates to the QA tier. Then, the test team approves unit changes while conducting system tests on a holistic perspective. The release process is repeated on the Production environment.

Synchronization

When a work environment has a high level of anxiety, the above workflow becomes difficult to complete. In some workplaces it could take months to complete. The QA team may try to cover all test scenarios and try their best to approve a release as 100% accurate. Unfortunately, this is an impossible task where all scenarios can never be realized. Mistakes happen with any system and some have large consequences.

One way to improve confidence is to make sure all tiers are closely aligned with content, code, and configurations. LSA has a content synchronization workflow that runs automatically every night. Each morning, content is aligned on the QA and Production tiers. By synchronizing content, LSA essentially has two identical environments. For the QA test team, this proves advantageous since they are testing with Production content. Their test scripts are more authentic using Production content. System tests are also improved with remote areas of the websites accessible. Whereas most test systems have only test data and limited areas accessible, the QA system is always an almost-identical copy of Production. The Development tier is also synchronized with Production content on demand. For the developers, this content sync helps validate code changes.

The QA environment additionally provides a disaster recovery site for Production. This has benefited LSA by recovering from content corruption in the past. Even though database and file system backups are gathered, the QA environment provides another point of disaster recovery and failover. If for some reason the Production tier suffers from hardware outages, the QA tier servers can be brought into service without suffering any outages. For example, if the network fiber to the Production data center is severed, the QA data center can serve content as if it were Production.

Finally, the QA tier provides the project team an avenue for easily isolating issues reported on Production. One of the first steps while troubleshooting issues on Production is to recreate the problem on a neutral environment in order to find a solution. The QA provides this ability automatically.

Beyond the technical advantages of synchronizing each tier, there is also a subjective advantage: It provides a level of confidence enabling the team to react quickly when issues arise. There is always a fallback with the QA tier. A high level of confidence enhances the team's performance and they are able to make changes faster and potentially recover from any issue rapidly.

Technical Details

The content synchronization process is a collection of an outermost shell script wrapper, two Perl scripts, three shell script wrappers for three Java APIs, eleven SQL queries, and three SQL stored procedures, as well as the associated detailed documentation. The files are distributed across three hosts with Production acting as the content source and Development or QA as the content target. The master script runs on the Production side and calls the scripts on the target side. The process runs automatically nightly and runs on-demand whenever desired. There are options to increase or decrease the logging levels, skip certain steps, and not delete the working directory for troubleshooting.

Outermost Wrapper

The outermost wrapper script checks for the presence of a "do not run" file and exits with an error message if it finds one. Otherwise it calls the source side Perl script.

Source Side Perl Script, Pass 1

The source (Production) side Perl script first checks to make sure it's the only copy running since running more than one copy is too memory-intensive. It defines a unique job ID so all log messages are associated to the script. Next a working directory is created and nine SQL queries executed. The queries identify changes in the CMS system compared to the target or destination tier including:

- Content project hierarchy
- Sites and content type definitions (CTD)
- Channels and content items deleted
- New and updated channels and content items
- Published content in Production either stale or unpublished in the target tier

If the query returns data, the result set is inserted into a formatted manifest file for export. The publish and delete items are inserted as a simple list into a text file. Next, using the CMS vendor's export tool, export any new and updated channels and content items from Production using the formatted manifests.

Once the source side has exported the archives, the script calls to a SQL stored procedure to synchronize taxonomies between the two tiers. It then calls out to the target tier to create the working directory and copies the manifests for new and updated content. A GUID list for deleted content and content to publish is also copied. The source side Perl script then calls its counterpart Perl script on the target tier.

Target Side Perl Script, Pass 1

The target side Perl script first checks for and lists any pre-existing configuration changes not yet committed (which are overwritten later in the process). It disables custom listeners then deletes any channels and content items no longer in the Production tier via a shell wrapper for the delete Java API. The vendor's import tool is used to insert the Production data in the required order of project changes, new channels and content items, new sites and content type definitions, and finally updated channels and content items.

During the import process, the target side Perl script attempts to remediate several common errors and warns the user, often with a recommendation as to how to resolve the issue manually when it cannot do so automatically. Common errors include:

- Communications timeouts on import
- Importing expired events
- Import failures due to CTD changes
- Missing or null project IDs
- Objects deleted then recreated in Production resulting in different GUIDs between the two environments
- Pre-existing objects or projects that could not be overwritten
- Project paths already in use
- Shell scripts returning unexpected values
- Taxonomies out of sync between tiers

Once the imports are complete the script re-enables the listeners by reverting all uncommitted configuration changes. The script then publishes content that is stale or unpublished in the target tier via a shell wrapper around the publish Java API. It clears the Dynamic Site Management (DSM) caches for all stages via another shell wrapper around an HTTP Java API. Finally, the script deletes its working directory (option available to keep working directory for debugging) before returning control back to the source side Perl script.

Source Side Perl Script, Pass 2

The source side Perl script runs a second pass of queries to identify channel/file associations present in Production but missing in the target tier, channel order differences between Production and the target tier, and newly-imported content needed to be published. Any missing channel/file associations are updated via a SQL stored procedure. Any channels out of order are reordered via another SQL stored procedure. The target side working directory is recreated and the GUID lists for channel order and content to publish are copied to the target side. The source side Perl script then calls its target side counterpart again.

Target Side Perl Script, Pass 2

The target side Perl script publishes any channels that were out of order and which were previously updated by the stored procedure. It also publishes any of the newly-imported content, both with the same shell wrapper around the publish Java API. Once again it clears the DSM caches via the shell wrapper around the HTTP Java API. Finally it deletes its working directory (option available to keep working directory for debugging) before returning control back to the source side Perl script.

Source Side Perl Script, Pass 3

The source side Perl script performs a final set of the queries and warns the user if the two tiers are not actually in sync. Finally it deletes the working directory (option available to keep working directory for debugging).

Conclusion

Almost all organizations today have some level of enterprise application development projects. In our experience, these projects include multiple environments with at least a Production and Development tier. Code changes step through these environments while testing and approving is conducted.

It is valuable to make sure that each environment is aligned which enhances the team's ability to test and develop changes to the system. Synchronizing code, configuration files, and content improves confidence and the team's ability to quickly make corrections when mistakes are made.

Most organizations already have multiple tiers and a downstream code release process, but few spend the extra effort to migrate content back upstream. Usually, a Development or QA environment has test content, which does not reflect the actual content that users create in Production.

There is no substitution for Production content. It represents how the ultimate users interact with the system. In LSA's situation, synchronizing the Production content with the other tiers has proven critical with testing and recovery. The effort to migrate content from Production to QA has especially enhanced the team's ability to troubleshoot issues and quickly make corrections.

Authors

William Thayer, MBA

Senior Consultant, Avalon Consulting, LLC. (http://www.avalonconsult.com/)

Will Thayer is a senior business IT consultant with 15 years' experience in planning, strategy, development, and training. He has built numerous Internet portal applications for companies across the nation. His expertise includes Internet application development, management information systems, and system development life cycle practices. As an Adjunct Professor at the University of Denver, Will taught graduate and undergraduate students for 5 years. He currently works for Avalon Consulting, LLC and lives in Evergreen, Colorado.

Joshua S. Simon

Senior System Administrator, The University of Michigan (http://www.umich.edu/), College of Literature, Science, and the Arts (http://www.lsa.umich.edu/)

Josh Simon is a senior systems administrator with over 20 years of industry experience, employed at The University of Michigan's College of Literature, Science, and the Arts and living near Ann Arbor, Michigan. His areas of expertise include systems administration, project management, and technical writing. He was the Desk Editor of SAGEwire during its existence and is a long-time USENIX contributor, having served on five LISA program committees (1997, 1999, 2000, 2002, and 2003) and reviewed that conferences' refereed papers eleven times (those five plus 2005, 2007, 2008, 2009, 2011, and 2012). He's written and published conference and workshop summaries in *;login:* for since 1999.