Running Adobe Experience Manager on AWS

July 2016

Notices

Contents

Abstract

Adobe Experience Manager (AEM) is a leader in digital experience delivery. It is a powerful, enterprise-ready solution that provides businesses with an easy way to deliver immersive web experiences, build a brand, drive demand, and extend reach to audiences across the globe. Amazon Web Services (AWS) is a leader in cloud services and infrastructure providing a flexible, cost effective, and easy-to-use computing platform. Combining AEM and AWS can be an effective method to deliver personalized digital experiences to your customers.

This whitepaper outlines some of the main benefits and describes some best practices that should be applied when deploying AEM on AWS. The content targets technical leaders responsible for deploying and managing AEM, allowing them to get started quickly on deploying AEM on AWS.

Introduction

Delivering a fast, secure, and seamless experience is essential in today's digital marketing environment. The need to reach a broader audience across all devices is essential. A shorter time to market can be a differentiator from competitors. Companies are turning to cloud-based solutions to boost business agility, harness new opportunities, and gain cost efficiencies. The first section of this whitepaper summarizes some of the key benefits of using AEM on AWS, and highlights how Advanced AWS Partner Network (APN) Consulting Partner Razorfish uses AEM on AWS. We then briefly describe some of the key components of AEM, providing the foundation for the discussion of AEM on AWS implementation best practices.

With any deployment on AWS, there are many different considerations and options, so your approach might be different from the approach we walk through in this paper.

Why Use AEM on AWS?

AEM, as a Web Experience Manager (WEM) platform, can take advantage of some the benefits of the AWS platform, including **global capacity, security, reliability**, **fault tolerance**, **programmability,** and **usability**. This section discusses several ways in which deploying AEM on AWS is different from deploying it to an on-premises infrastructure.

Flexible Capacity

One of the benefits of using the AWS Cloud is the ability to scale up and down as needed. When using AEM, you have full freedom to scale all your environments quickly and cost effectively, giving you opportunities to establish new development, quality assurance (QA), and performance testing environments.

AEM is frequently used in scenarios that have unknown or significant variations in traffic volumes. The on-demand nature of the AWS platform allows you to scale your workloads to support your unique traffic peaks during key events, such as Black Friday (the big shopping day after the US Thanksgiving holiday) or the US National Football League (NFL) Super Bowl.

Flexible capacity also streamlines upgrades. At this point, many AEM clients are upgrading from AEM 5.x to AEM 6.1. AWS makes it very easy to set up a parallel environment, so you can migrate and test your application and content in a production-like environment. Performing the actual production upgrade itself can then be as simple as the change of a domain name system (DNS) entry.

Broad Set of Capabilities

As a leading web content management system solution, customers often use AEM as the foundation of their digital marketing platform. Running AEM on AWS provides customers with the benefits of easily integrating third-party solutions for auxiliary experiences such as blogs, and provide additional tools for supporting mobile delivery, analytics, and big data management. You can integrate the open and extensible APIs of both AWS and AEM to create powerful new combinations for your firm. Razorfish uses AEM on AWS with their turnkey digital marketing platform, Fluent, to help organizations market across multiple channels in the areas of analytics, targeting and experience management.

With solutions like Amazon Simple Notification Service (SNS), Amazon Simple Queue Service (SQS), and AWS Lambda, AEM functionality can easily be integrated with other third-party functionality in a decoupled fashion. Examples of this can be seen for a Content Publishing Integration Scenario in Figure 1 and for a Workflow Integration Scenario in Figure 2.

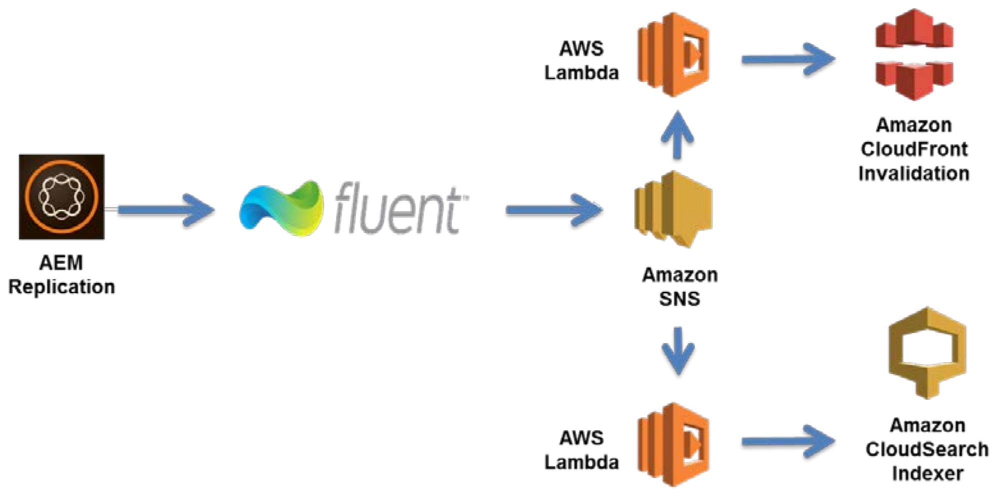

Figure 1: Content Publishing Integration Scenario

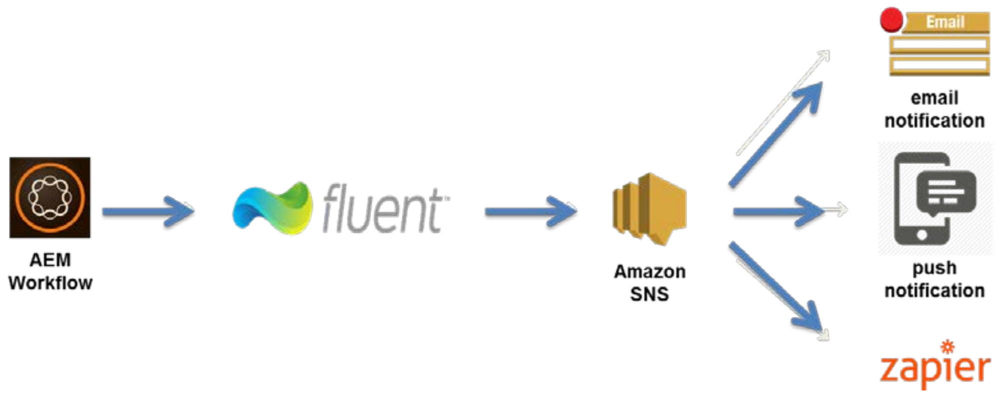

Figure 2: Workflow Integration Scenario

AWS can also provide a clean, manageable, and auditable approach to decoupled integration with backend systems such as customer relationship management (CRM) and commerce systems. Figure 3 shows an Enterprise Systems Integration scenario.

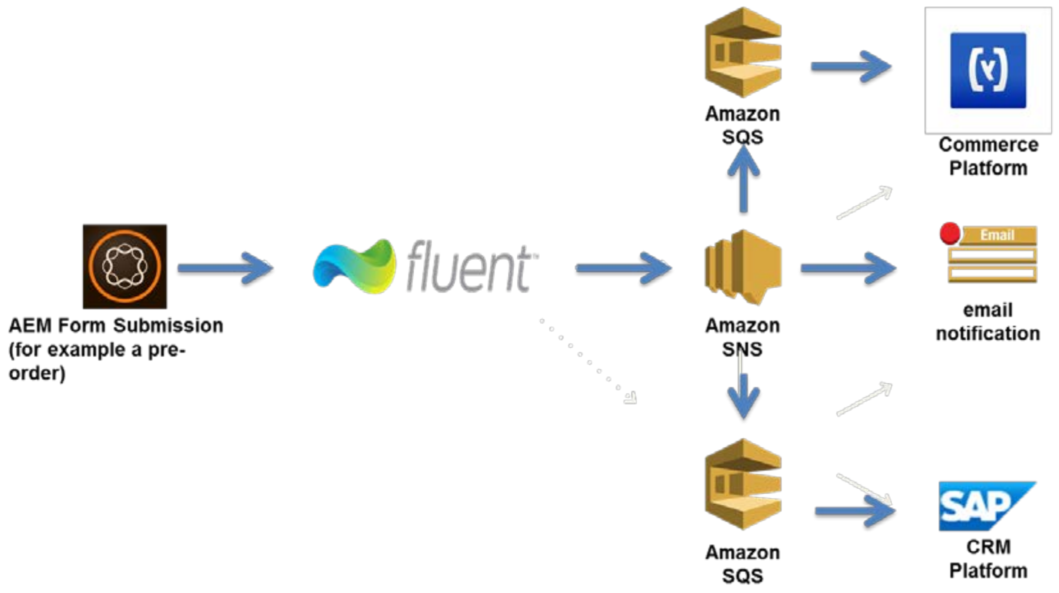

Figure 3: Enterprise Systems Integration Scenario

Leveraging AEM on AWS enables the creation of a digital marketing foundation that not only delivers the full set of capabilities, but also does so using a scalable, de-coupled, and micro services-based solution.

Adobe Experience Manager Overview

This section highlights some of the key technical elements for Adobe Experience Manager and offers some best practice recommendations.

Capabilities

Adobe Experience Manager (AEM) has a broad set of capabilities for digital experience delivery. Some of the key use cases for AEM are content management, experience management and personalization, digital asset management, communities, mobile applications, and in-store digital experiences. The majority of use cases for deployments are content, asset, and experience management. This whitepaper will focus on those deployment scenarios.

This whitepaper focuses on AEM 6.1 (released May 2015), but is applicable to the newly released AEM version 6.2 as well.

Architecture

A standard AEM architecture consists of three environments: author, publish, and dispatcher. Each of these environments consists of one or more instances.

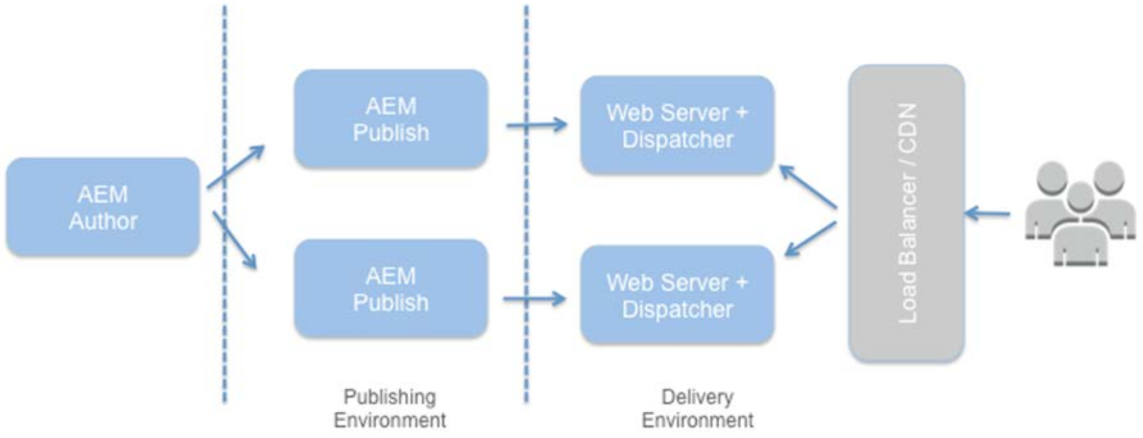

Figure 4: Sample AEM Architecture

The author environment is used for creating and managing the content and layout of an AEM experience. It provides functionality for reviewing and approving content updates, and publishing approved versions of content to the publish environment.

The publish environment delivers the experience to the intended audience. It renders the actual pages, with an ability to personalize the experience based on audience characteristics or targeted messaging.

The author and publish instances are Java web applications that have identical installed software. They are differentiated by configuration only.

The third component in an AEM architecture is the dispatcher. This a caching and/or load balancing tool that helps realize a fast and dynamic web authoring environment. For caching, the dispatcher works as part of an HTTP server, such as Apache, with the aim of storing (or *caching*) as much of the static website content as possible and accessing the website's publisher layout engine as infrequently as possible. For caching, the dispatcher module uses the web server's ability to serve static content. The dispatcher places the cached documents in the document root of the web server.

Repositories

Within AEM, everything is content and stored in the underlying repository. AEM's repository is called CRX, and it implements the Content Repository API for Java (JCR), and is based on Apache Jackrabbit Oak.

The Oak storage layer provides an abstraction layer for the actual storage of the content.

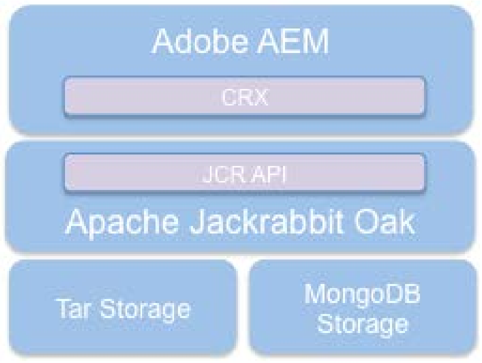

Figure 5: AEM Storage Options

Currently, there are two primary[1] storage implementations available in AEM6: **Tar Storage** and **MongoDB Storage**. The Tar storage uses tar files. It stores the content as various types of records within larger segments. Journals are used to track the latest state of the repository. The MongoDB storage leverages MongoDB for sharding and clustering. The repository tree is kept in one MongoDB database where each node is a separate document.

In AEM, binary data can be stored independently from the content nodes. The binary data is stored in a data store, whereas content nodes are stored in a node store.

Tar Storage is the most common approach and recommended by Adobe for the majority of the deployment scenarios, so this whitepaper will primarily focus on this option. MongoDB Storage is currently only recommended for author environments that require horizontal scaling[2].[3]

AEM Implementation on AWS

In this section, we will outline key design elements for deployment AEM on AWS.

Architecture Options

A first design decision for any implementation on AWS is to determine the Amazon Virtual Private Cloud (VPC) layout. For simplicity, we will assume a basic setup with one author and two publish AEM instances.

VPC with Public Subnets

In this architecture, all AEM components are deployed in public subnets. The publish instances are spread across two AWS Availability Zones. Each of the AEM components has its own security group (and uses Auto Scaling, if applicable). To ensure the right security for publish and author instances, it is critical to follow Adobe's security checklist for AEM[4].

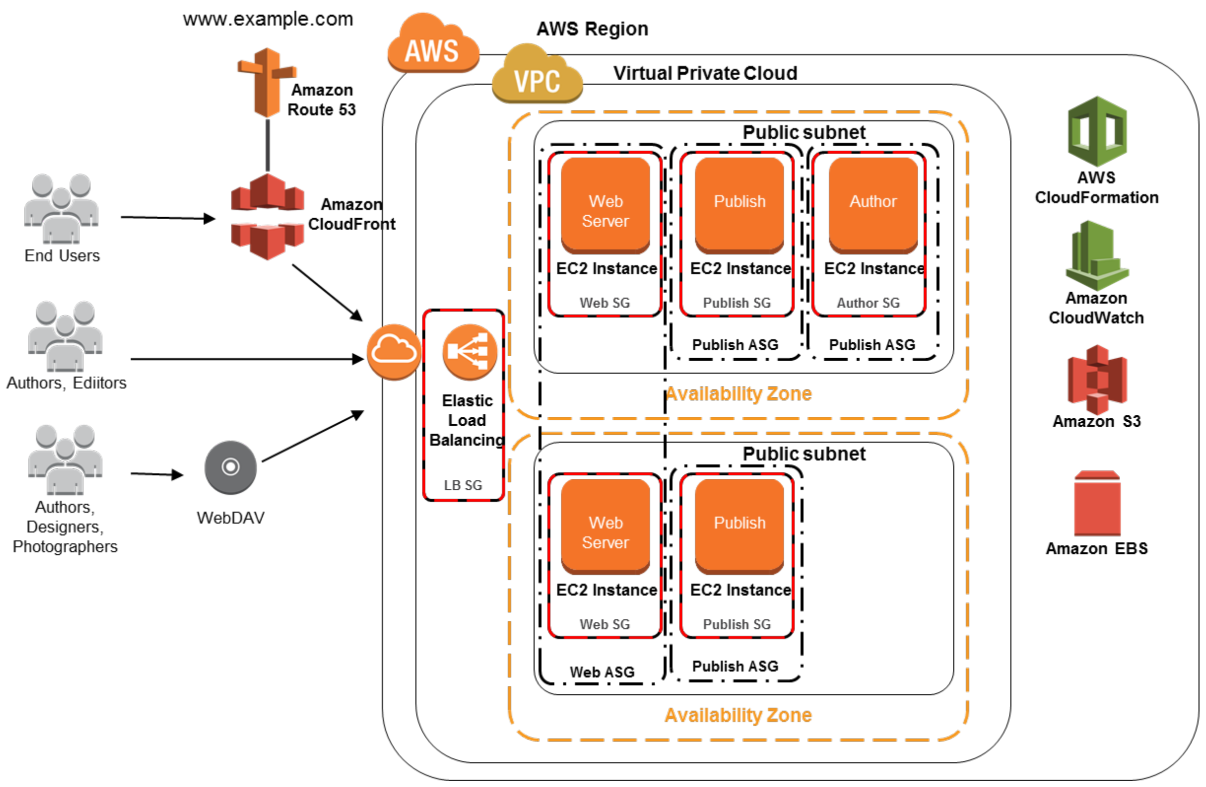

Figure 6: Solution with Public Subnets

VPC with Public and Private Subnets

In this architecture, the web server and the dispatcher are deployed in the public subnets. Author and publish instances are deployed in private subnets. Although AEM will still need to be configured according to Adobe's checklist, this will create an additional layer of security implemented through the VPC architecture. However, because the author instance is not reachable through a public subnet, a virtual private network (VPN) connection will be necessary to allow content authors to interact with the system.

It's necessary to use NAT instances or a VPC NAT gateway to provide outbound access for the AEM publish and author instances for any automated retrieval updates and security patches. In addition, AEM's Link Checker and certain cloud services need outbound access.

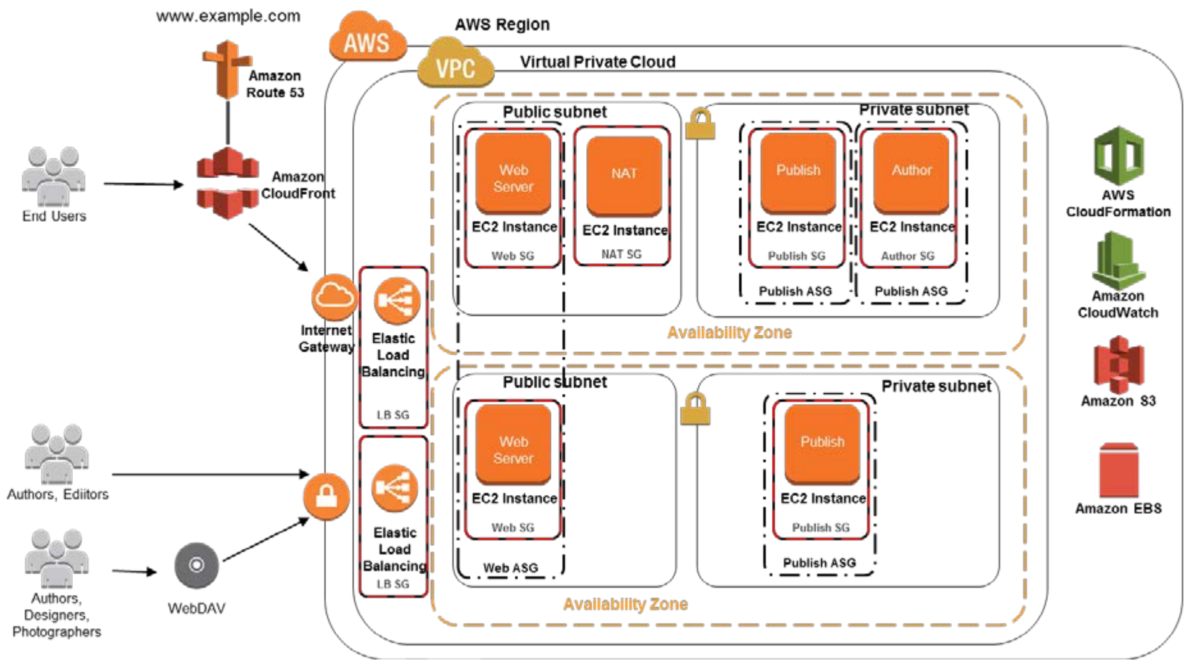

Figure 7: Architecture with Public and Private Subnets

Instead of a VPN connection for authoring, an alternative approach can be leveraging an author dispatcher server that is only accessible to content administrators.

Architecture Sizing

For AEM, the right instance type depends on the usage scenario. For author and publish instances in the most common publishing scenario, a solid mix of memory, CPU, and I/O performance is necessary. Therefore, Amazon Elastic Compute Cloud (Amazon EC2) General Purpose M3 or M4 instances are good targets for these environments, depending on sizing. M4 instances support enhanced networking and are Amazon Elastic Block Store (Amazon EBS) optimized by default.

The dispatcher is installed on a web server, and it is a key caching layer. Therefore, sizing memory and compute is important, but optimization for I/O is critical for this tier.

For all these instances, Amazon EBS optimization is important. Amazon EBS volumes on which AEM is installed should use either GP2 volumes or provisioned input/output operations per second (IOPS) volumes. This will guarantee a specific level of performance and lower latency for those operations.

The specific sizing for the number of servers you have depends on your AEM use case (e.g., experience management, digital asset management, etc.) and the level of caching that should be applied. Detailed calculations can be made with guidelines outlined on the Adobe support site[5].

Load Balancing

A key component in the architecture is Elastic Load Balancing. In an AEM setup, Elastic Load Balancing is configured to balance traffic to the dispatchers. By default, a load balancer distributes incoming requests evenly across its enabled Availability Zones (AZs). To ensure that a load balancer distributes incoming requests evenly across all back-end instances, regardless of the Availability Zone that they are in, enable cross-zone load balancing.

For authenticated AEM experiences, authentication is maintained by a login token. When a user logs in, the token information is stored under the **.tokens** node of the corresponding user node in the repository. The value of the token (i.e., the session ID) is also stored in the browser as a cookie named **login-token**. In that case, the load balancer should be configured for sticky sessions, routing requests with the login-token cookie to the same instance. Starting with version 6.1[6], AEM can be configured to recognize the authentication cookie across all publish instance. However, it also requires that all relevant user session information (e.g., a shopping cart) is available across all publish instances.

Elastic Load Balancing can be used in front of the dispatchers to provide a Single CNAME URL for the application. The load balancer, in conjunction with AWS Certificate Manager[7] can be used to provide an HTTPS access and to offload SSL. By using the load balancer you can further secure your website deployment by moving the publisher instances into a private subnet, allowing access from only the load balancer. The load balancer can also translate the port access from port 80 to the default publish port 4503.

High Availability

For a highly available AEM solution, the architecture should be set up to leverage AWS strengths. Each instance in the AEM cluster should be set up with the Amazon EC2 Auto Recovery[8] feature. Additionally, when the cluster is built in conjunction with a load balancer, Auto Scaling can be used to automatically provision nodes across multiple Availability Zones[9]. We recommend that you provision across multiple Availability Zones for high availability and use multiple AWS Regions to address global deployment considerations as needed. Amazon Route 53 can be set up to perform DNS failover based on health checks in a multi-region deployment.

Scaling

Different components of AEM require different approaches to scaling. The virtually unlimited capacity of AWS can be used to provide the necessary capacity for your application needs. A simple way to accomplish this is to create separate Amazon Machine Images (AMIs) for the publish instance and author instance. Two separate launch configurations can be created using these AMIs and included in separate Auto Scaling groups.

These launch configurations can be invoked programmatically or manually to add or delete instance for each of the pools—publish and author. For faster startup and synchronization, you can place the AEM installation on a separate Amazon EBS volume. By taking frequent snapshots of the volume and attaching the snapshots to the newly launched instances, the need to replicate large amounts of data from the author can be cut down. In their startup process, the publish instance can then trigger author—publish replication to fully ensure the latest content.

For the dispatcher, Auto Scaling is relatively straightforward because no state is managed on the dispatcher. However, on a content publish event, the dispatcher gets called from the author or publish instance to invalidate the cache. Newly launched instances will need to notify author or publish instances to receive future invalidation calls. One approach is to pair and co-locate dispatchers with a publish instance. In that case, the cache invalidation call becomes a localhost call from the publish instance.

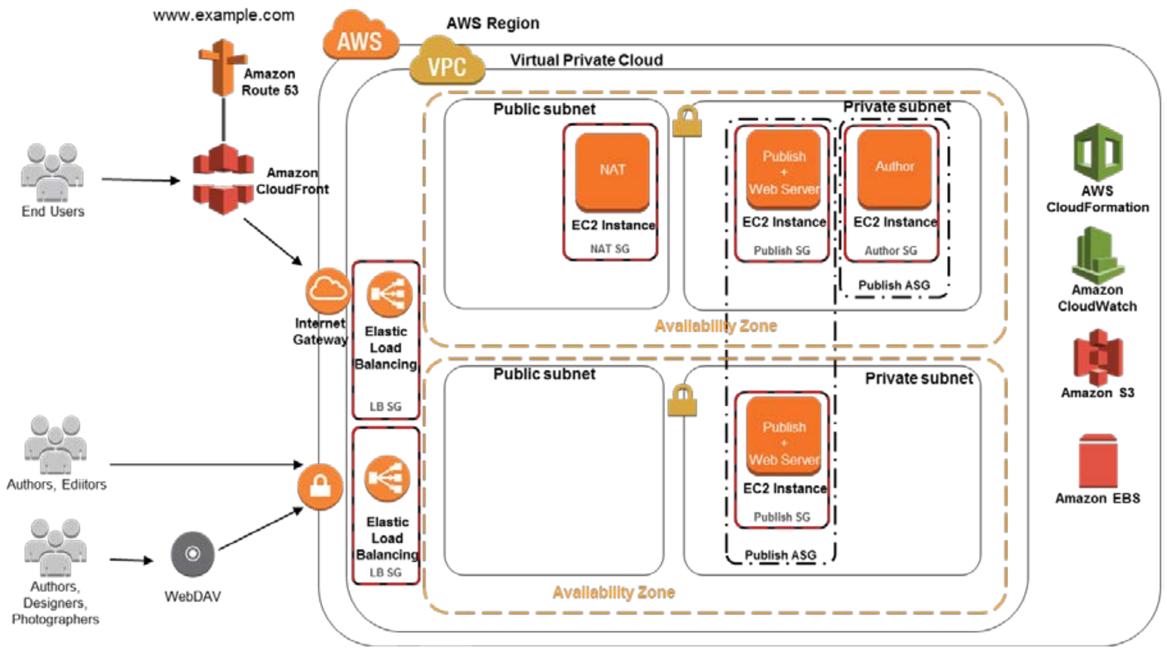

Figure 8: Co-located Publish and Web Server+Dispatcher Instances

If you're using MongoDB storage, Auto Scaling becomes more straightforward, because the repository is stored in the database.

Content Delivery

AEM on AWS can benefit from leveraging Amazon CloudFront, a content delivery network (CDN). Sites can utilize a CDN in addition to the standard AEM dispatcher caching layer. When you use a CDN, you need to consider how content is invalidated and refreshed in the CDN when content is updated.

One approach is to build a custom invalidation workflow step, an AEM replication event listener, or a dispatcher monitor service (using the

/invalidateHandler property) that communicates with the Amazon CloudFront API. When using this approach, take care to delay the cache invalidation until content has been replicated to all publish instances.

Using cache control headers in combination with times to live (TTLs) provides another method of updating cached assets. The common approach is to configure Amazon CloudFront to have a ***minTTL*** of 0 and set cache-control headers (i.e., s-max-age and max-age) in Apache web server for all content. Amazon CloudFront will check for changed content every ***s-maxage***, assuming a request is made to it; browsers will cache content for ***s-maxage***. As a result, CDN invalidations will be effective every ***s-maxage seconds***[10] **.**

In this scenario, special care should be taken with code deployments that invalidate all Amazon CloudFront assets.

Dynamic Content

The dispatcher is the caching layer with the AEM product. It allows for defining caching rules at the web server layer. To realize the full benefit of the dispatcher, pages should be fully cacheable. Any element that isn't cacheable will "break" the cache functionality.

To incorporate dynamic elements in a static page, the recommended approach is to use client-side JavaScript, Edge Side Includes (ESI), or web server level Server Side Includes (SSI). Within an AWS environment, Edge Side Includes can be configured using a solution such as Varnish, replacing the dispatcher.

Security

Security is an important first consideration in any web-hosting environment. The security of the AEM hosting environment can be broken down into two areas: application security and infrastructure security. A crucial first step for application security is to follow the security checklist for AEM and the dispatcher. You want to prevent Denial of Service (DoS) attacks, and for that the *mod_rewrite*[11] module in Apache Web Server is often used to prevent the request passing through to the dispatcher or publish instance. In addition, the Apache *mod_security* module[12] can provide additional security against XSS attacks.

From an infrastructure level, AWS provides many tools to lock down your environment. One of the core components of network security is Amazon VPC[13]. This service provides multiple layers of network security for your application such as public and private subnets, security groups, and network access control lists for subnets.

Amazon CloudFront can be used to provide some crucial security benefits: 1) you can offload direct access to your backend infrastructure, and 2) using the web application firewall (WAF) provided by the AWS WAF[14] service, you can apply rules to prevent the application from getting compromised by scripted attacks. The same rules that are encoded in Apache mod_security on the dispatcher can be moved or replicated in AWS WAF. Because AWS WAF integrates with Amazon CloudFront CDN, this enables earlier detection, minimizing overall traffic and impact. Additionally, AWS WAF provides centralized control, automated administration, and real-time metrics.

AWS also provides audit tools such as AWS Trusted Advisor[15]. AWS Trusted Advisor inspects your AWS environment and makes recommendations for saving money, improving system performance and reliability, and security. We also recommend that you consider tools such as Amazon Inspector. Amazon Inspector is an automated security assessment service that helps improve the security and compliance of applications deployed on AWS. Amazon Inspector automatically assesses applications for vulnerabilities or deviations from best practices. After performing an assessment, Amazon Inspector produces a detailed report with prioritized steps for remediation. This can support system management and gives security professionals the necessary visibility into vulnerabilities that need to be fixed. In addition to Amazon Inspector, other third-party products such as Burp Suite or Qualys SSL Test (for certificate validation) can be used.

Finally, having an audit log of all API actions and configuration changes can be useful in determining what changed and by whom. AWS CloudTrail and AWS Config provide you the capability to capture extensive audit logs. We recommend that you enable these services in your hosting environment.

Digital Asset Management

AEM includes a Digital Asset Management (DAM) solution. When planning for your AWS architecture, you should evaluate the potential use of the DAM solution as part of your planning. With DAM usage, the number of large assets usually increases and often involves resource intensive processes such as image transformations and renditions. Various architecture options should be considered depending on the scenario, and they are described in detail in Deploying Adobe Experience Manager DAM: Architecture blueprints and best practices.[16].

You can find details for sizing for a DAM solution in the Adobe AEM Manuals[17]. A common scenario to manage server utilization is to leverage a separate server for all resource-intensive digital asset management tasks. Many of the transformations utilize FFmpeg or ImageMagick. If necessary, you can create an Auto Scaling configuration to offload these processes.

Binary data can be stored independently from the content nodes in AEM. When deploying on AWS, the binary data store can be Amazon Simple Storage Service (Amazon S3)[18], simplifying management and backups[19]. Also, the binary data store can then be shared across author instances and even between author and publish instances, reducing overall storage and data transfer requirements.

Automated Deployment

AWS provides API access to all AWS services, and Adobe does this for AEM as well. Many of the various commands to deploy code or content, or to create backups, can be invoked through an HTTP service interface[20]. This allows for a very clean organization of the continuous integration and deployment process with the use of Jenkins as a central hub, invoking AEM functionality through CURL or similar commands.

Jenkins can support manual, scheduled, and triggered deployments, and can be the central point for your AEM on AWS deployment. If necessary, additional automation can be enabled using the Jenkins AWS CloudFormation plugin[21], enabling the creation of a complete environment from the Jenkins console.

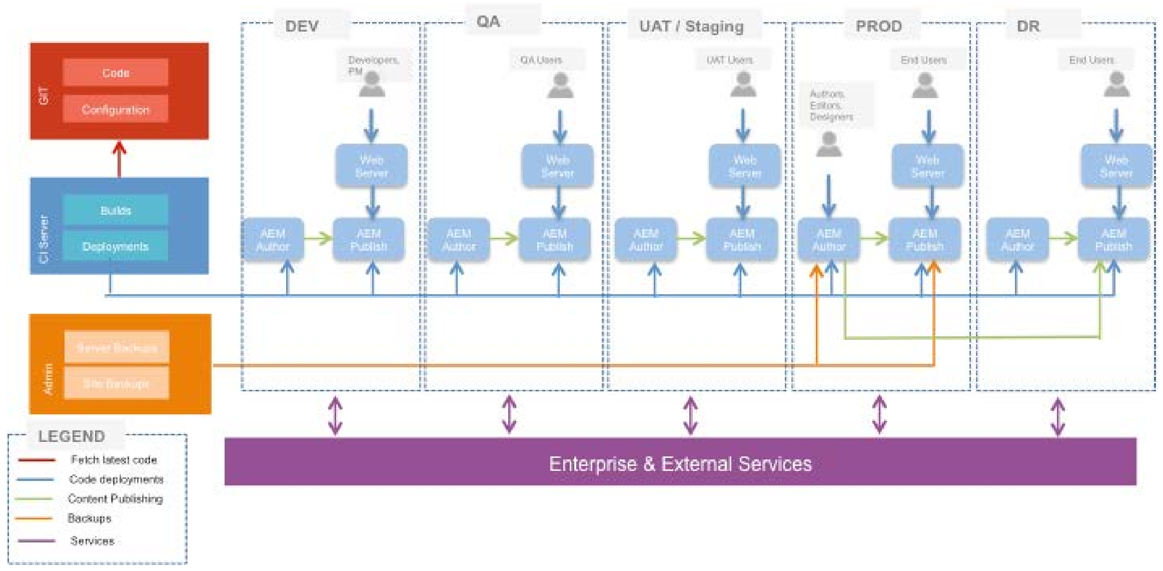

Figure 9: Example CI Setup for an AEM Jenkins Architecture

Jenkins can be installed on an Amazon EC2 instance, pulling code from AWS CodeCommit or an alternative code management solution.

Automated Operations

One of the key benefits of running AEM on AWS is the streamlined AEM Operations process.

To provision instances, AWS CloudFormation or AWS OpsWorks can be leveraged to fully automate the deployment process, from setting up the architecture to provisioning the necessary instances. Using the AWS CloudFormation embedded stacks functionality[22], scripts can be organized to support the different architectures outlined in the earlier sections.

When using AEM's Tar Storage, repository content is stored on the file system. To create an AEM backup, a file system snapshot must be made. This is done easily on AWS through Amazon EBS snapshots. To ensure a consistent backup, use tools such as fsfreeze[23] to suspend file system I/O. With MongoDB storage, backing up the MongoDB database will create the backup. If the data store is configured outside the main repository, this will need to be backed up separately.

When using Amazon S3 for data storage, it is important to set the appropriate policies to prevent data loss from accidental deletions or overwrites. Additionally, a cross-region bucket copy can be used to provide appropriate disaster recovery for your application. Because files in the file data store directory are immutable, they can be backed up incrementally (potentially using rsync or aws cli sync) or after running the online backup.

The Amazon CloudWatch dashboard can support the creation of a consolidated monitoring dashboard, allowing you to view the various performance statistics in one single place. Custom metrics can also be generated and incorporated in the CloudWatch dashboard.

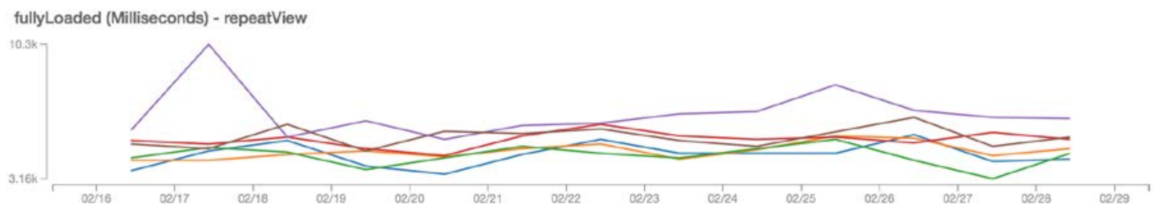

Figure 4: Sample custom metric view

Additional AWS Services

This section highlights a few additional services and capabilities you can leverage from both the AEM platform and AWS to add further value to your AEM deployment on AWS.

Amazon Elastic File System

Amazon Elastic File System (Amazon EFS) is a file storage service for Amazon EC2 instances. Multiple Amazon EC2 instances can access an Amazon EFS file system at the same time, providing a common data source for workloads and applications running on more than one instance. Some other considerations are as follows:

- Tar Storage combined with Amazon EFS can create new opportunities for streamlining the creation of highly available instances.
- Amazon EFS can be an alternative to Amazon S3 for data storage.

- Amazon EFS can provide new options for offloading AEM tasks, such as rendition generations.

Personalization and Targeting

AEM offers tools to manage targeting within experiences delivered through the solution. Adobe also has complementary products, which integrate well with AEM, that further personalize and target the experience for customers. Combined with AWS solutions such as Amazon Kinesis, Amazon Machine Learning, and AWS Lambda, a powerful targeting engine can be created to deliver 1:1 personalization.

Mobile

AEM provides capabilities to support mobile app and mobile web deployments. AWS has launched a number of offerings targeting mobile users, including AWS Mobile Hub, AWS Device Farm, AWS Mobile Analytics, and Amazon Simple Notification Service (SNS) mobile push notifications. Combining these can create a full turnkey solution to create rich mobile experiences. AEM 6.1 already includes a connector for Amazon SNS mobile push notifications.

Conclusion

Using Adobe Experience Manager on AWS can provide you with a great platform and foundation for delivering digital experiences. As you look to deploy AEM on AWS, we recommend that you consider the best practices and guidance outlined in this document, and consult the additional references outlined in the Further Reading section that follows.

Contributors

The following individuals and organizations contributed to this document:

- Pawan Agnihotri, Solutions Architect, Amazon Web Services
- Martin Jacobs, GVP, Technology, Razorfish

Further Reading

For additional help, please consult the following sources:

- Adobe Experience Manager 6.1 Documentation
- AWS Documentation

Notes

[1] IBM DB2 10.5, Oracle Database 12c and MySQL 5.5 are available in restricted support configuration.

[2] https://docs.adobe.com/content/docs/en/aem/6-1/deploy/best-practices/performance/_jcr_content/par/download/file.res/AEM_Scalability_White_Paper_FINAL - 06122015je.pdf

[3] MongoDB Storage is also an option for storing User Generated Content in the AEM Communities module.

[4] https://docs.adobe.com/docs/en/aem/6-1/administer/security/security-checklist.html

[5] https://docs.adobe.com/docs/en/aem/6-1/manage/hardware-sizing-guidelines.html

[6] https://docs.adobe.com/docs/en/aem/6-1/administer/security/encapsulated-token.html

[7] https://aws.amazon.com/certificate-manager/

[8] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-recover.html

[9] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html

[10] http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Expiration.html#ExpirationDownloadDist

[11] http://httpd.apache.org/docs/current/mod/mod_rewrite.html

[12] https://www.modsecurity.org

[13] https://aws.amazon.com/vpc/

[14] https://aws.amazon.com/waf/

[15] https://aws.amazon.com/premiumsupport/trustedadvisor/

[16] https://helpx.adobe.com/content/dam/help/attachments/white3.pdf

[17] https://docs.adobe.com/docs/en/aem/6-1/deploy/configuring/performance/assets-performance-sizing.html

[18] https://aws.amazon.com/s3/

[19] https://docs.adobe.com/docs/en/aem/6-1/deploy/platform/data-store-config.html#Amazon S3 Data Store

[20] https://docs.adobe.com/content/docs/en/crx/2-3/how_to/package_manager.html#Managing Packages on the Command Line

[21] https://wiki.jenkins-ci.org/display/JENKINS/AWS+Cloudformation+Plugin

[22] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-stack.html

[23] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/xfsfreeze.html