Migration Scenario: Migrating Web Applications to the AWS Cloud

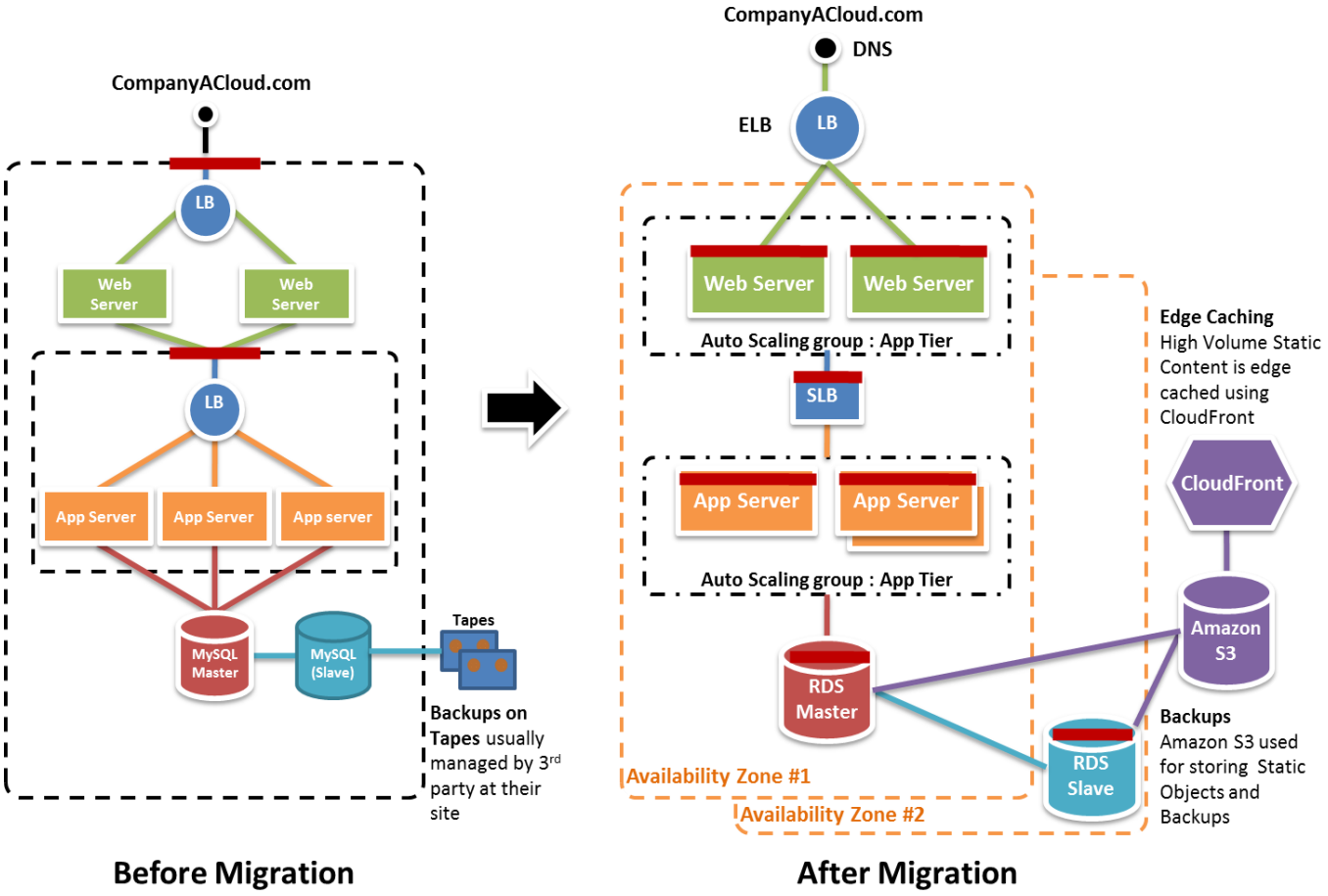


Figure 1: CompanyACloud.com Architecture (Before and After Migration)

Use case

CompanyACloud.com is a customer-facing web application of company A**,** which serves as a marketing portal and a customer management system. Customers, partners and employees use the web application to collaborate with each other using a rich web interface that can be viewed in a standard internet browser. CompanyACloud.com lists a complete catalog of products and their details. As new product announcements are made, marketing campaigns generate substantial amounts of traffic to the site resulting in periodic spikes. Outside of the timeframes caused by these spikes, CompanyACloud.com experiences a fairly steady and predictable traffic load, which is characteristically high on weekdays and low on weekends. The website is currently hosted on dedicated infrastructure at the company's headquarters.

Application Architecture

Using a standard 3-tier application architecture, the company deploys a frontend hardware-based load balancer, which manages traffic across two Apache web servers each running on a separate physical box. The application is running

behind the company firewall (DMZ) and uses standard SSL encryption. The backend business logic is implemented in Java, and leverages Tomcat as application container and application server, and three Tomcat servers power the website. The application also had a database layer with consists of one master MySQL server and two slave servers for greater performance. A simple illustration of the architecture is provided in Figure 1 on the left hand side.

Motivation for Migration

Company A would like to move the web app to the cloud environment for three main reasons.

1. The company wishes to scale out the web application, and address the growing traffic, without investing in new hardware.
2. They would like to cut down administration costs by automating deployments.
3. Finally, the company would like to expand and provision extra capacity only when it's needed, for example, when running marketing campaigns.

Migration Plan, Strategy, & Execution Steps

Cloud Assessment

On financial assessment, the technical program manager from the website development team was able to map the hardware configuration of physical servers to equivalent EC2 instance types and estimate the combined storage and bandwidth requirements. The team realized that they could free up the current infrastructure for other internal projects, discontinue a tape backup maintenance contract and reduce their operating expenditures by 30%.

During the technical assessment, they discovered that the entire CompanyACloud.com technology stack was compatible with AWS and could run on Amazon EC2 Instances with Linux. **They also discovered that the web app can be configured to run at peak capacity (7 Servers) during promotional campaigns, medium capacity (4 Servers) on weekdays and low capacity (2 Servers) on weekends.**

The IT Security team was able to get a complete SAS 70 Type II audit report from AWS and they were able to review security best practices[1].

Proof of Concept

The web development team was skeptical about the relational database migration. To test, they decided to build a proof of concept application. During the proof of concept, the **team learned the following techniques: starting, terminating and configuring Amazon EC2 instances and Amazon RDS DB Instances**, storing and retrieving Amazon S3 objects, and setting up elastic load balancers **using the AWS Management Console[2]**. They learned a ton about AWS and saw that they have full control over the environment, and felt a lot more confident about moving to the next step. The relational database files (binary and transaction logs) were moved to Amazon RDS instances using the standard "mysqlimport" utility. For the test environment, they deployed a DB Instance within a single Availability Zone, and for the production environment, they set up a DB Instance with the Multi-AZ deployment to increase availability. The team was able to successfully test and migrate all data to a DB instance, get performance metrics using Amazon CloudWatch, and set

[1] AWS Security Center – http://aws.amazon.com/security
[2] AWS Management Console – http://console.aws.amazon.com

retention policies for backups. They **built migration scripts** to automate the process and **created awareness within the organization** by organizing a "brownbag" session and successfully demonstrated their work to their peers.

Data Migration

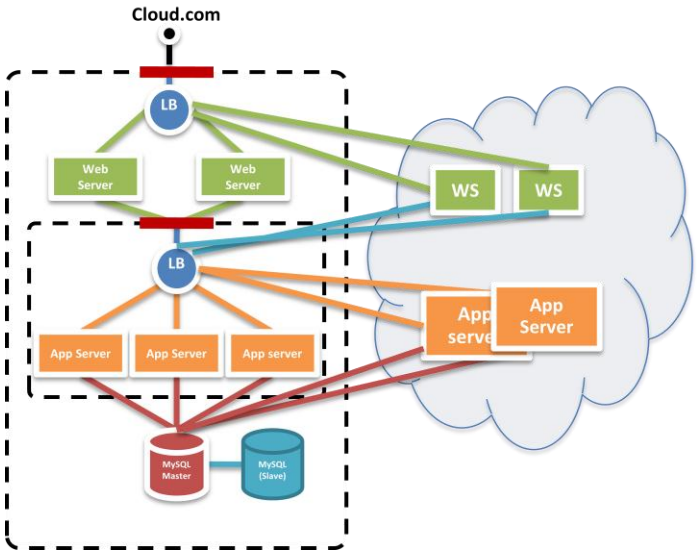
Once the proof of concept was complete, the team decided to **move all of the application's static files** (Images, JS, CSS, video, audio, and static HTML content) into an Amazon S3 bucket, created a CloudFront distribution of that Amazon S3 bucket, and modified the references in web pages so that end-users get the content directly from Amazon S3 and Amazon CloudFront. With a few scripts and the AWS SDK for Java library, they were able to **transfer all data from tape drives** and upload it to Amazon S3.

Application Migration

During the application migration phase, the development team **launched both small and large instances**[3] for their web and tomcat servers. They **created AMIs** (Amazon Machine Images, basically "golden" system images) for each server type. AMIs were designed to boot directly from an EBS volume and fetch the latest WAR file binaries during launch from the source code repository. They modified their build and deployment scripts to use the cloud as an endpoint. **Security Groups** were defined to isolate web servers from the applications and database servers. Testing (functional, load, performance etc.) was performed to ensure that the systems were performing at expected levels, and that exit criteria for each component were met.

Co-existence Phase

During the migration phase, the collocation infrastructure was not deprecated immediately. Company A employed a **hybrid migration strategy** during the migration of all web and application servers. The configuration of the on-premise hardware load balancer was modified to send requests to the new instances in the cloud. For a short duration, the load balancer was routing traffic to the servers in the cloud in as well as to the physical servers. After verifying that the servers in the cloud were performing at required levels, the physical servers were dismissed one by one, the load balancers were updated, and all of the web traffic was being served up by the EC2 instances running in the cloud.



After testing was completed, the DNS was switched to point to the cloud-based web servers and the application was fully migrated to the AWS cloud.

Leveraging the Cloud

Once the production site was launched, Company A was looking forward to the time when they could use some of the advanced features of AWS. The team automated some processes so that once the server is started it could be easily "attached" to the topology. They **created an Auto Scaling group** of web servers and were able to provision more capacity automatically when specific resources reach a certain threshold (Apache web servers CPU utilization above 80% for 10 min). The team invested some time and resources in **streamlining their development and testing processes** to

[3] Amazon EC2 Instance Types – http://aws.amazon.com/ec2/instance-types

make it is easy to clone testing environments. They gained lot of experience using AWS resources and also invested time in **leveraging multiple Availability Zones** for even higher availability.

Optimization

During the optimization phase, the development team analyzed their utilization patterns and realized that they could save 30% if **they switched to Reserved Instances**[4]. They purchased four Reserved Instances (2 for web servers and the other 2 for tomcat servers). They built additional scripts to run their web application in 3 different "modes": weekend, weekday and promotion mode. These modes defined the minimum number of servers to run. The team also integrated Amazon CloudWatch into their existing dashboards so that they can monitor the system metrics of every instance in their cloud fleet.

Mode	Web Servers	App Servers
Weekend	1	2
Weekday	2	3
Promotion	5	7

Conclusion

The company was able to successfully migrate an existing web application to the AWS cloud. With minimal effort, the team was not only able to free up the physical infrastructure for other projects but also reduce the operating expenditure by 30%. Using the phased-driven approach, the development team was able to resolve all the financial, technical and social-political concerns. Deciding to invest in a proof of concept proved extremely valuable. The resulting architecture was not only elastic and scalable but also flexible and easier to maintain.

[4] Amazon Reserved Instances – http://aws.amazon.com/ec2/reserved-instances