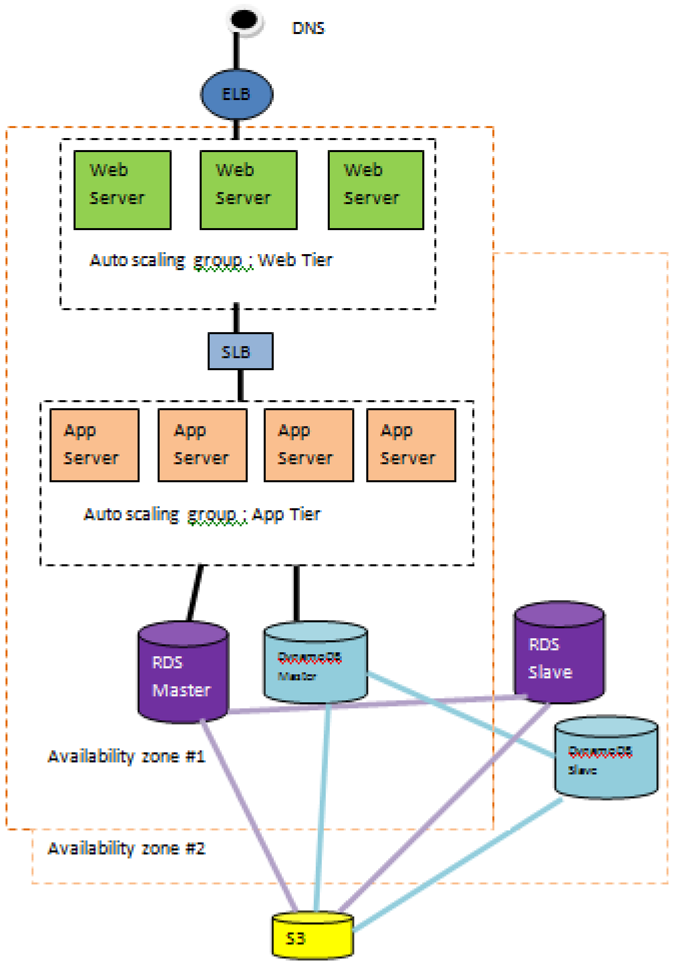
Migrating the Web Application to AWS Cloud

Problem Statement

We have a monolithic java application stack that includes an Apache Web Server, Apache Tomcat application server with Active MQ and Oracle and MongoDB backend.

The goal with this document is to propose a solution to migrate this application stack to AWS, which all the AWS services to use to maintain HA and Load Balancing (consider app to be stateless)

Migration Plan, Strategy and Execution Steps



Cloud Assessment

The objective of this step is to weigh the financial, security and technical considerations of owning and operating a data center or co-located facilities versus employing a cloud-based infrastructure.

On financial assessment, 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, discontinue a tape backup maintenance contract by using Gateway-Virtual Tape Library (Gateway-VTL) feature of AWS and for long term archival storage, Virtual Tape Libraries to be integrated with a Virtual Tape Shelf (VTS). Let us assume that this would reduce the operating expenditures by 25%.

During the technical assessment, it is discovered that the entire technology stack is compatible with the AWS cloud using AMIs, Amazon EC2 instances, DynamoDB, Amazon S3 bucket and Amazon RDS. Also, it will be beneficial to use a Utility Pricing Model with a Support Package provided by Amazon DevPay service. **Assuming the incoming traffic, let us consider that the web app can be configured to run at peak capacity (5 Servers) during promotional campaigns, medium capacity (3 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.

Proof of Concept

The goal of this phase is to learn AWS and ensure that the assumptions regarding suitability for migration to the cloud are accurate. Team will get familiar with **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.** Team will gain confidence by building pilots by deploying miniature pieces of architecture (Oracle, MongoDB, Load balancer, Tomcat). In the process, team will learn how to build a Web Server AMI, how to set the security group so that only the web server can talk to the app server, how to store all the static files on Amazon, how to move the database files to Amazon RDS, how to manage/monitor your application using Amazon and how to use IAM to restrict access to only the services and resources required for your application to function. For the test environment, DB Instance can be deployed within a single Availability Zone, and for the production environment, DB Instance can be setup with the Multi-AZ deployment to increase availability.

The team should be able to successfully test and migrate all the data to a DB instance, get performance metrics using Amazon CloudWatch, set the retention policies for backups and **build migration scripts** to automate the process.

Data Migration Phase

During data migration, major things to consider are the storage options to use and estimate the efforts required to migrate the data to the cloud.

As per the analysis, the best option for storing data from MongoDB and Oracle is DynamoDB and Amazon RDS respectively. Amazon RDS makes it easy to set up, operate, and scale Oracle Database deployments in the cloud. With Amazon RDS, one can deploy multiple editions of Oracle Database in minutes with cost-efficient and re-sizable hardware capacity. Amazon RDS allows the team to focus on application development and frees up from managing time-consuming database administration tasks including provisioning, backups, software patching, monitoring, and hardware scaling. In addition, Amazon RDS for Oracle makes it easy to use replication to enhance availability and reliability for production workloads. By using the Multi-AZ deployment option, you can run mission-critical workloads with high availability and built-in automated failover from your primary database to a synchronously replicated secondary database. As with all AWS services, no upfront investments are required, and you pay only for the resources you use. Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. All data items are stored on solid-state drives (SSDs) and are automatically replicated across the Availability Zones in an AWS region to provide built-in high availability and data durability. With Amazon DynamoDB, one can offload the administrative burden of operating and scaling a highly available distributed database cluster while paying a low variable price for only the resources you consume.

Migrating Oracle DB

The strategy for migrating Oracle DB is a two-step approach because it requires only minimal downtime and can be used for databases of any size.

1. The data is extracted from the source database at a point in time (preferably during non-peak usage) and migrated while the database is still up and running. Because there is no downtime at this point, the migration window can be sufficiently large. After you complete the data migration, you can validate the data in the destination database for consistency with the source and test the destination database on AWS for performance, for connectivity to the applications, and for any other criteria as needed.

2. Data changed in the source database after the initial data migration is propagated to the destination before switch over. This step synchronizes the source and destination databases. This should be scheduled for a time when the database can be shut down (usually non-business hours on a weekend). During this process, there won't be any more changes to the source database because it will be unavailable to the applications. Normally, the amount of data that is changed after the first step is small compared to the total size of the database, so this step will be quick and thus requires only minimal downtime. Once all the changed data is migrated, you can validate the data in the destination database, perform necessary tests, and, if all tests are passed, switch over to the database in AWS.

The tool to be used for this migration will be Oracle Golden gate since it can work for databases with any size and requires zero downtime.

MongoDB Migration

The four step strategy of migrating MongoDB is as follows:

1. Design and finalize the table schema for DynamoDB using AWS SDK for Java. The SDK helps take complexity out of coding by providing Java APIs for DynamoDB. The single, downloadable package includes the AWS Java library, code samples, and documentation. AWS SDk for java offers a feature called Amazon DynamoDB Object Mapper. The DynamoDBMapper eliminates the need for application-level data conversions and custom middleware solutions by using Plain Old Java Objects (POJOs) to store and retrieve Amazon DynamoDB data.
2. Produce a JSON or CSV export of data stored in the MongoDB instance using mongoexport.
3. Upload the exported file in the S3 bucket.
4. Import this data from S3 to DynamoDB using AWS Data Pipeline. AWS Data Pipeline is a web service that you can use to automate the movement and transformation of data. With AWS Data Pipeline, you can define data-driven workflows, so that tasks can be dependent on the successful completion of previous tasks. You define the parameters of your data transformations and AWS Data Pipeline enforces the logic that you've set up.

Application Migration Phase

The strategy to be used for application migration is Forklift Migration strategy. Stateless applications are better served using this approach. Rather than moving pieces of the system over time, forklift or “pick it all up at once” and move it to the cloud. During the application migration phase to migrate the applications on tomcat servers to EC2 instances, the team will be **launching both small and large instances** for the web and tomcat servers. They will be **creating AMIs** (Amazon Machine Images, basically “golden” system images) for each server type. AMIs will be designed to boot directly from an EBS volume and fetch the latest WAR file binaries during launch from the source code repository. ELBs will be configured above Web and application servers for load balancing. The build will be modified and deployment scripts will be prepared to use the cloud as an endpoint. **Security Groups** to be defined to isolate web servers from the applications and database servers. Once the entire setup is ready, DNS to be switched to point to the ELB above the web servers. Testing (functional, load, performance etc.) to be performed to ensure that the systems are performing at expected levels, and that exit criteria for each component are met. If any issue is encountered then the LB above web servers in the cloud will be immediately configured to point to the collocation infrastructure as it will not be deprecated immediately.

Leveraging the cloud

Once the production site is launched, team will be looking forward to the time when they could use some of the advanced features of AWS. The team to automate some processes so that once the server is started it could be easily “attached” to the topology. They will be **creating an Auto Scaling group** of web servers and to be 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 to invest some time and resources in **streamlining their development and testing processes** to make it is easy to clone testing environments. Also, it is necessary to harden the security by

- Safeguarding AWS credentials
- Restricting users to AWS resources
- Data encryption
- Adopting recovery strategy

Optimization Phase

In this phase, one should focus on how the cloud-based application can be optimized in order to increase cost savings. Following are some of the ways to do it :

Understand the usage pattern

Team will understand, monitor, examine and observe the load patterns. One can be more proactive if the traffic patterns are well understood. Once the team is able to closely align the traffic to the resources consumed, the cost savings will be high.

Terminate the Under-Utilized Instances

Team to be focusing on inspection of the system logs and access logs periodically to understand the usage and lifecycle patterns of each Amazon EC2 instance and terminate the idle instances. Also, check if the under-utilized instances can be eliminated to increase utilization of the overall system.

Improve Efficiency

Optimizing the system will lead to cost savings as AWS cloud provides utility-style pricing, i.e., you are billed only for the infrastructure that has been used.

Conclusion

The AWS cloud brings scalability, elasticity, agility and reliability to the enterprise. Following the above plan, team will be able to successfully migrate the existing application to AWS cloud very smoothly.