Refactoring Web Services on AWS cloud (PaaS & SaaS)

Asst. Prof Pramod Salunkhe[1], Pem Tshering[2], Yashwant Chavan[3]

[1](Dept. Of Computer Engineering, Bharati Vidyapeeth College of Engineering, Lavale, India)
[2-3](B.E, Dept. Of Computer Engineering, Bharati Vidyapeeth College of Engineering, Lavale, India)

---***---

Abstract – *Refactoring or re-architecting in its simplest can be defined as ever-changing application code for the better. Typically, the goal is to boost performance, quality, and maintainability. Re-architecting for AWS expands those advantages and allows extra ones such as better availability, scalability, and reliability. Another major advantage that refactoring provides is the ability to dump massive amounts of undifferentiated work to AWS. Tasks such as hardware maintenance, software patch management, information updates, and more are all managed for you when using services like Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), or AWS Fargate, permitting your team to focus in what differentiates your business [1]. In this paper, we will refactor our services using a refactoring strategy to improve agility and business continuity. We can include new features, and skills effectively and easily, and have the best performance for our application workload.*

Key Words: **Refactoring, Maintainability, AWS, Continuity, Workload, Container Service, Kubernetes Service**

1. INTRODUCTION

We may assume, doing a project where the services are running on physical machines/virtual machines/cloud machines, it could be EC2 instances and we are dealing with various services for our application workload, we could be having databases, application servers, web servers, network services like DNS, DHCP, etc. To manage all these multiple teams are required, like the cloud computing team if we use a cloud computing platform, the Virtualization team if we are doing virtualization on our data center, the data center's operation team, monitoring team, system admin team, and few other teams will get involved managing this application workload.

1.2 Why choose to refactor?

We pick out to refactor those applications that are very critical, contribute revenue and are strategically crucial to the business development. While less strategic business applications are likely to be the candidate for re-hosting or re-platforming. Some compelling reasons to pick out to refactor an application include:

1. Create new revenue streams and/or optimize existing ones.

2. Create further improvements that directly impact future revenue opportunities.

3. Enable quick time-to-market with new features.

4. Support a changing/new business model.

5. Easily scale both up and down to meet planned and unplanned changes in traffic [2].

1.3 Key components of AWS web hosting Architecture

Here we describe some of the key components of the web hosting architecture implemented on AWS and explain how they differ from a traditional web hosting architecture.

- **Content Delivery**

  Edge caching remains relevant in Amazon Web Service's cloud computing infrastructure. Any existing solution in web application infrastructure should work just fine in the AWS Cloud. However, when using AWS, an additional option is available to use the Amazon CloudFront service for edge caching your website. Amazon CloudFront can be used to deliver dynamic, static, and streaming content, over a global network of edge locations to your website. Requests for the content are automatically routed to the closest edge location, delivering content with the best possible performance. CloudFront is optimized to work with other Amazon Web services such as Amazon S3 and Amazon EC2. CloudFront also works seamlessly with origin servers other than AWS that store the original, final versions of your files. Like other AWS, there are no monthly contracts or commitments to use CloudFront—you only pay for it much or as little content as you actually deliver through the Service.

- **Managing public DNS**

  Migrating a web application to the Amazon Web Service Cloud requires some DNS changes to take advantage of the multiple availability zones provided by AWS. For one to help manage DNS routing, AWS provides Amazon Route 53, a highly available and scalable DNS web service. Queries for your domain are automatically routed to the nearest DNS server and are therefore answered with the best possible performance. Route 53 resolves requests for your domain name (e.g. www.example.com) to your Elastic Load Balancer as well as your zone apex record (example.com).

- **Load balancing across clusters**

  Hardware load balancers are common networking devices used in traditional web application architectures. AWS provides this capability through the Elastic Load Balancing service, a configurable load balancing solution that supports host health checks, balancing traffic on EC2 instances across multiple availability zones, and dynamically adding and removing Amazon EC2 hosts from the Server supports rotation. Elastic Load Balancing can dynamically increase or decrease load balancing capacity to meet traffic demands while providing a predictable entry point through the use of a persistent CNAME. The Elastic Load Balancing service also supports persistent sessions to meet advanced routing requirements. If your application requires advanced load balancing capabilities, you can purchase a software load balancing package (e.g. HAProxy, or Nginx) on EC2 instances. You could then assign Elastic IP addresses to these load-balanced EC2 instances to minimize DNS changes [3].

2. PROBLEM STATEMENT

There is an excessive amount of operational overhead where our team is struggling with the uptime and regular scaling requirement, upfront capital expenditure, and regular operational expenditure if you are using your own local data center and processes will be manual and will be tough to automate even with virtualization, all these processes will be time-consuming and really expensive.

3. OBJECTIVE

- We need a flexible infrastructure.
- No upfront cost (Pay-as-we-go model).
- We need IaaC
- We need PaaS
- SaaS for ease of managing out infrastructure so to have low latency operational overhead.

4. SOLUTION

We are using a cloud platform, but instead of using IaaS, we will be mostly using PaaS and SaaS services. We are not going with regular Ec2 instances but will be using some cloud-managed services from AWS. Cloud here means we can code our infrastructure, so we can have IaaS.

PaaS and SaaS services are easy to manage, flexible, and elastic in nature. Scaling will be mostly taken care of by the cloud vendors and it will be a pay-as-you-go model with more automation, so refactoring the application gives an easy infrastructure to manage, increased performance, and is convenient to scale. There will be no need of huge teams to manage all this.

5. PROPOSED WORK

Aws services we will be using in this project include:

5.1 Front-end

- Instead of using regular Ec2 instances we will be using the Beanstalk service (VM for Tomcat app server) and this service will in turn create an Ec2 instance and host our application on it. We don't need to manage this Ec2 instance manually.
- Beanstalk service will also have a load balancer.
- It will also have auto-scaling and S3/EFS bucket for storing the artifacts or we can use our own S3 bucket.

5.2 Back-end

- In the back-end of the database we will be using RDS instances. It's really like a PaaS, so you will get a database platform to choose from, just need to fill in the requirement and the database is up and running in no time.
- Scaling will be very easy.
- Regular backup will be taken automatically.
- We will be using the ElastiCache service instead of Memcached.
- ActiveMQ instead of RabbitMQ.
- Route 53 for DNS.
- CloudFront for content delivery network, so if we have a global audience then using CloudFront for the content delivery network will be very easy and convenient.

5.3 System architecture

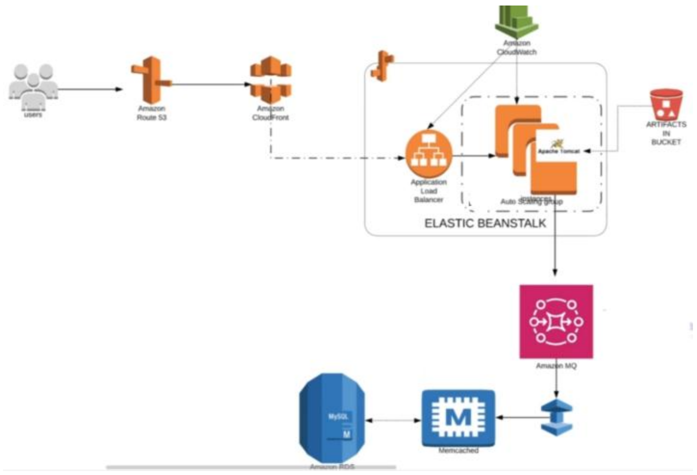

Fig -1: System architecture

Users will access our URL which will be resolved to an endpoint from Amazon Route 53. The endpoint will be of Amazon CloudFront content delivery network which will cache a wide variety of things to serve the global audience.

From there the request will be redirected to an application load balancer which is part of Elastic Beanstalk. The application load balancer will forward the request to an Ec2 instance which is in an auto-scaling group. Here our Tomcat application service will be running and all this will be a part of Elastic Beanstalk.

There will be also Amazon CloudWatch alarms that will be monitoring the auto-scaling group and will scale out and scale in based on the requirement.

There will be an S3 bucket where artifacts will be stored and we can deploy our latest artifact by just clicking a button.

So our entire front end will be managed by the beanstalk.

For the back-end in Memcached instead of RabbitMQ, we are using AmazonMQ. Instead of using Memcache on the Ec2 instance, we're using Elastic cache service.

Instead of using a database running on the Ec2 instance, we're going to use Amazon RDS.

5.4. Flow of execution

- Login to AWS account
- Create key pair for beanstalk instance login.
- Create a security group for (backend service) ElastiCache, RDS, and ActiveMQ.
- Then create:
    - ➢ RDS
    - ➢ Amazon ElastiCache
    - ➢ Amazon ActiveMQ
- Create an Elastic Beanstalk environment.
- Update backend security group to allow traffic from Beanstalk security group.
- Update backend security group to allow internal traffic.

So, by now our backend services will be also up and running, and RDS will be also up and running

- Launch Ec2-Instance for Database initialization.
- Login to the instance and Initialize the RDS database.
- Change healthcheck on Beanstalk to /login.
- Add 443 HTTPS listener to the elastic load balancer
- Build artifact with backend information.

So, by now we should have the endpoint of RDS, the endpoint of AmazonMQ, and the endpoint of ElastiCache. We will feed this information into our application properties file and we'll build the artifact.

- Then we'll deploy the artifact to the beanstalk environment.
- Create a content delivery network using Amazon CloudFront with an SSL certificate for the HTTPS connection.

Once we have this ready we can.

- Update our load balancer endpoint on GoDaddy DNS zones or we can also do this on Amazon Route 53 public DNS zone.

Once this is all ready,

- We will test it from the URL.

6. CONCLUSIONS

Our Website is really getting surfed from CloudFront distribution. Now we have our entire stack and we are using PaaS and SaaS services, we are not directly using Ec2 instances or so. Our entire setup is going to give us very low operational overhead so we don't need many engineers, and admins to manage our entire setup.

REFERENCES

[1] Modernizing with AWS, online available at: https://www.newrelic.com/resources/white-papers/modernizing-aws-refactoring-your-applications/

[2] Why is refactoring your code important in Agile. Mario Fernandez April 19, 2021 Available at: https://www.coscreen.co/blog/refactoring-your-code-in-agile/ [12/4/2022]

[3] Matt Tavis, Philip Fitzsimons. Web Application Hosting in AWS, September 2012.