Migration of Applications from Monolithic to Micro-services and DevOps Automation

Customer: India's largest trucking platform

Problem Statement

The customer’s environment on AWS was facing scalability challenge as it was maintained across a heterogeneous set of software solutions with many different types of programming languages and systems and there was no fault-tolerant mechanism implemented. The lead time to get a developer operational was high as the developer ended up waiting for a longer duration to access cloud resources like EC2, RDS, etc. Additionally, the deployment process was manual which increased the chances of unforced human errors and configuration discrepancies. Configuration management took longer time which further slowed down the deployment process. Furthermore, there was no centralized mechanism for user management, log management, and cron jobs monitoring.

Proposed Solution

For AWS cloud development the built-in choice for infrastructure as code (IAC) is AWS CloudFormation. However, before building the AWS Cloudformation (CF) templates, Powerup conducted a thorough assessment of customer’s existing infrastructure to identify the gaps and plan the template preparation phase accordingly. Below were a few key findings from their assessment:

- Termination Protection was not enabled to many EC2 instances
- IAM Password policy was not implemented
- Root Multi Factor Authentication (MFA) was not enabled
- IAM roles were not used to access the AWS services from EC2 instances
- CloudTrail was not integrated with Cloudwatch logs
- S3 Access logs for Cloudtrail S3 bucket was not enabled
- Log Metric was not enabled for Unauthorised API Calls; Using ROOT Account to access the AWS Console; IAM Policy changes; Changes to CloudTrail,
CloudConfig, S3 Bucket policy; Alarm for any security group changes, NACL, RouteTable, VPCs

- SSH ports of few security groups were open to Public
- VPC Flow logs were not enabled for few VPCs
Powerup migrated their monolithic service into smaller independent services which are self-deployable, sustainable, and scalable. They also set up CI/CD using Jenkins and Ansible. Centralized user management was implemented using FreeIPA, while ELK stack was used to implement centralized log management. Healthcheck.io was used to implement centralized cron jobs monitoring.

CloudFormation (CF) Templates were then used in the creation of the complete AWS environment. The template can be reused to create multiple environments in the future. 20 Microservices in the stage environment were deployed and handed over to the customer team for validation. Powerup also shared the Ansible playbook which helps in setting up the following components - Server Hardening / Jenkins / Metabase / FreeIPA / Repository.

The below illustrates the architecture:

- Different VPCs are provisioned for Stage, Production and Infra management. VPC peering is established from Infra VPC to Production / Stage VPC.
- VPN tunnel is established between customer office to AWS Infra VPC for the SSH access / Infra tool access.
- All layers except the elastic load balancer is configured in private subnet.
- Separate security group configured for each layer like DB / Cache / Queue / App / ELB / Infra security groups. Only required Inbound / Outbound rules.
- Amazon ECS is configured in Auto-scaling mode. So the ECS workers will scale horizontally based on the Load to the entire ECS cluster.
- Service level scaling is implemented for each service to scale the individual service automatically based on the load.
- Elasticache (Redis) is used to store the end user session
- Highly available RabbitMQ cluster is configured. RabbitMQ is used as messaging broker between the micro services.
- For MySQL and Postgresql RDS Multi-AZ is configured. MongoDB is configured in Master-slave mode.
- IAM roles are configured for accessing the AWS resources like S3 from EC2 instances.
• VPC flow logs / cloud trail / Cloud Config are enabled for logging purpose. The logs are streamed into AWS Elasticsearch services using AWS Lambda. Alerts are configured for critical events like instance termination, IAM user deletion, Security group updates etc.
• AWS system manager is used to manage collect the OS, Application, instance meta data of EC2 instances for inventory management.
• AMIs and backups are configured for business continuity.
• Jenkins is configured for CI / CD process.
• CloudFormation template is being used for provisioning / updating of the environment.
• Ansible is used as configuration management for all the server configurations like Jenkins / Bastion / FreeIPA etc.
• Sensu monitoring system is configured to monitor system performance
• New Relic is configured for application performance monitoring and deployment tracking

Cloud platform

AWS.

Technologies used

Amazon Redshift, freeIPA, Amazon RDS, Redis.

Benefit

IaC enabled customer to spin up an entire infrastructure architecture by running a script. This will allow the customer to not only deploy virtual servers, but also launch pre-configured databases, network infrastructure, storage systems, load balancers, and any other cloud service that is needed. IaC completely standardized the setup of infrastructure, thereby decreasing the chances of any incompatibility issues with infrastructure and applications can run more smoothly. IaC is helpful for risk mitigation because the code can be version-controlled, every change in the server configuration is documented, logged, and tracked. And these configurations can be
tested, just like code. So if there is an issue with the new setup configuration, it can be pinpointed and corrected much more easily, minimizing risk of issues or failure.

Developer productivity drastically increases with the use of IaC. Cloud architectures can be easily deployed in multiple stages to make the software development life cycle much more efficient.