TATA COMMUNICATIONS

CASE STUDY

IMPLEMENTING INFRASTRUCTURE AS A CODE USING TERRAFORM

JANUARY 2020

IMPLEMENTING INFRASTRUCTURE AS A CODE USING TERRAFORM

Our customer is a non-banking, asset finance company registered with the Reserve Bank of India. Their key businesses include vehicle loans and leasing, housing loans and insurance.

The IT services catering to these businesses were primarily hosted on premises. They deployed their applications from Amazon Web Services (AWS) due to the increased operational costs of managing a data centre, the high-lead time for augmenting IT capacity, the rate of staff turnover affecting overall run and maintain functions, and the cost of securing the data.

The customer had no prior experience in using automation tools, so TCL expertise was deployed in AWS, with an automation approach for provisioning, DevOps and management.

CUSTOMER REQUIREMENTS

- Cloud vendor independence and portability when needed, with the least effort
- Stable, easily customised deployment
- Secure infrastructure and data
- Code build and release management tasks automated
- All infrastructure deployment tasks automated to minimise the impact of staff turnover

TCL APPROACH

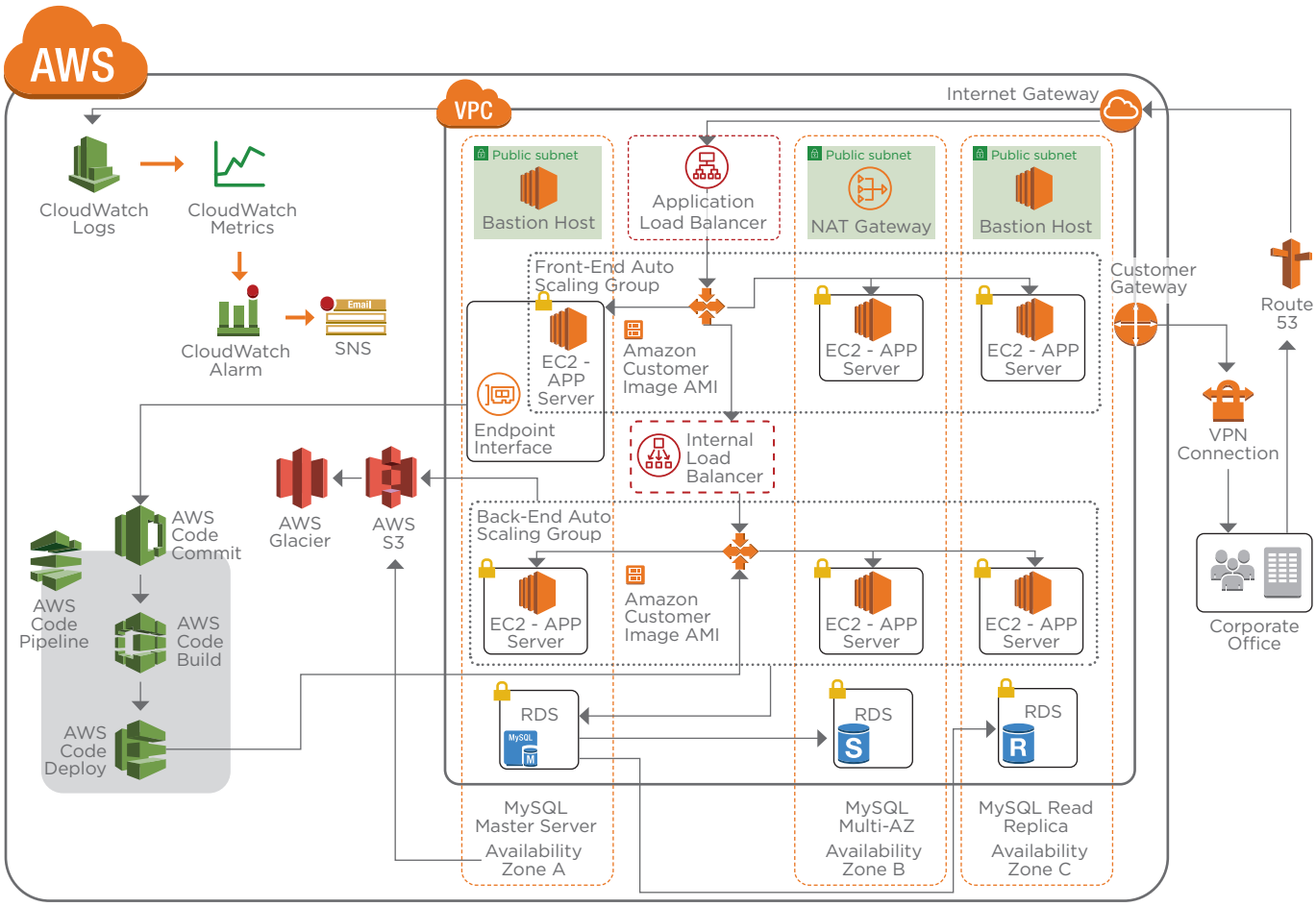
INFRASTRUCTURE AS A CODE (IAC)

We decided to use the Terraform tool to code the infrastructure components. Terraform is the ideal tool for building, changing, and versioning infrastructure safely and efficiently. Configuration files describe to Terraform the components needed to run a single application or your entire datacentre. Terraform generates an execution plan with a description of what it will do to reach the desired state, and then executes it. As the configuration changes, Terraform can determine what changed, and create incremental execution plans which can then be applied.

The infrastructure where Terraform can be used includes low-level components such as compute instances, storage, and networking; and high-level components such as DNS entries, SaaS features, etc. Terraform uses multiple **provider plugins** to create, modify, and destroy resources on infrastructure providers like AWS, GCP, Microsoft Azure, OpenStack as well as PaaS or SaaS services. This feature provides code portability across multiple cloud providers and helps to provision standardised infrastructure on any cloud.

DEVELOPMENT AUTOMATION FOR CODE BUILD AND RELEASE MANAGEMENT

We used AWS CodeCommit, CodeBuild, CodeDeploy and CodePipeline as the DevOps services. Once the developer commits the code in GIT (CodeCommit), the CI/CD automation will help the customer to perform a hands-free cascading of the application code across multiple environments (development, QA, pre-production and production). The CICD pipeline across the environments is tracked and an AWS SNS feature sends a notification to the DevOps team.



DEPLOYMENT FEATURES

VPC

We created a single production Virtual Private Cloud (VPC). It is a virtual network where you create and manage your AWS resources in a more secure and scalable manner. We automated VPC creation using Terraform, which also created end-to-end setup.

Subnet

We separated the web, application and database subnets into three subnets with smaller CIDR values, and added a fourth for load balancers. The design of the subnets was dependent on the application requirements. The code used for deployment is flexible, to create additional or lesser numbers with varied IP ranges. We have created four subnets under three availability zones for a 1+2 redundant zones implementation. While this was a customer requirement for three availability zones, having at least two availability zones is recommended for any three-tier application architecture.

Route and internet gateway

We deployed an internet gateway (IGW) as TCL-TT-PROD VPC to be used for downloading custom patches and connecting external email servers. The IGW was associated with all public subnets. Route tables were also configured to associate the inbound and outbound traffic to specific subnets.

NAT and NACL

We configured a NAT gateway to a NAT private IP and on to a public IP. This helped to secure the servers in the private subnets and get the outbound internet access to patch the OS, update the antivirus and send email to the external mail servers.

Using NACL (Network Access Control list) as a firewall for controlling traffic in and out of one or more subnets, we blocked and restricted traffic flow across the three subnets.

EC2 – web and app server under AWS Auto Scaling

We provisioned EC2 instances for web and app servers. The Terraform code was scripted to deploy 'c5. Large' type instances under auto scaling groups for the web and app servers. We chose three availability zones in US-EAST for deploying EC2 instances. Each zone has one instance of web and app deployed at any given time. Based on the load characteristics monitored by CloudWatch, the auto scaling group was configured to create additional EC2 instances using pre-defined custom templates.

Load Balancer

We configured Application Load Balancer instances separately for web server and app servers. Each load balancer is configured to the respective auto scaling targets with security groups.

S3 and Glacier

We used S3 for the static file store requirements of our customer. The web, app and database servers push the regular BI data, backups, and csv file for analytics into the S3 buckets. We have scripted the life cycle management policy configuration to move the infrequently accessed data to AWS Glacier.

CloudWatch and SNS

We enabled AWS CloudWatch monitoring to measure CPU, RAM, and disk utilisation, and to perform custom application monitoring. If the metrics reach the configured threshold, an alert is triggered from the CloudWatch alarm and a notification is sent to the customer's email.

CodePipeline

We automated the customer's developer activities, like source code build and release management. Once the developer commits their source code into AWS CodeCommit via private endpoint using a VPN connection, CodePipeline will trigger the CI/CD process. It gets the code from AWS CodeCommit and then starts AWS Code Build, using buildspec.yml and buildspec_test.yml.

Once the build and respective tools output are successful, they are moved to AWS Code Deploy. Here, Code Deploy checks the appspec.yml file, which provides the information on script folders for deployment configuration and strategy, and the application is then deployed in the target servers.

VPN and Customer Gateway

In compliance with the AWS Well Architected Framework, we have enabled and configured the VPN and Customer Gateway to use AWS services from the corporate office.