AWS Infrastructure Automation and Migration Project at Accenture

During my time at Accenture in Bangalore from September 2017 to October 2020, I was deeply involved in a major project to migrate our client's services from managed hosting to AWS. This project involved service design, network setup, data migration, automation, and deployment. Here's a detailed account of what I did and learned throughout this project.

Project Overview

We called this initiative "Project CloudShift." Our main goals were to:

1. Migrate existing services to AWS
2. Set up a robust infrastructure-as-code pipeline using Terraform and Jenkins
3. Implement container orchestration with Kubernetes
4. Establish a comprehensive CI/CD pipeline

AWS Migration

Network Setup

One of my first tasks was to set up the AWS network infrastructure. I used Terraform to define our VPC, subnets, and security groups. Here's a snippet of the Terraform code I wrote for this:

hcl

```hcl
provider "aws" {
  region = "ap-south-1"
}

resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"

  tags = {
    Name = "main-vpc"
    Project = "CloudShift"
  }
}

resource "aws_subnet" "public" {
  count             = 3
  vpc_id            = aws_vpc.main.id
  cidr_block        = "10.0.${count.index}.0/24"
  availability_zone = data.aws_availability_zones.available.names[count.index]

  tags = {
    Name = "Public Subnet ${count.index + 1}"
  }
}

resource "aws_security_group" "allow_web" {
  name        = "allow_web_traffic"
  description = "Allow inbound web traffic"
  vpc_id      = aws_vpc.main.id

  ingress {
    description = "HTTPS"
    from_port   = 443
```

```
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "allow_web"
  }
}
```

Data Migration

For data migration, I wrote Python scripts to transfer data from our old managed hosting to AWS S3 and RDS. Here's an example of a script I used to migrate data to S3:

python

```
import boto3
import os

s3 = boto3.client('s3')
bucket_name = 'my-migration-bucket'

def upload_to_s3(local_directory, bucket, s3_directory):
    for root, dirs, files in os.walk(local_directory):
        for filename in files:
            local_path = os.path.join(root, filename)
            relative_path = os.path.relpath(local_path, local_directory)
            s3_path = os.path.join(s3_directory, relative_path)

            print(f"Uploading {local_path} to {bucket}/{s3_path}")
            s3.upload_file(local_path, bucket, s3_path)

upload_to_s3('/path/to/local/data', bucket_name, 'migrated-data')
```

Infrastructure Automation

Terraform and Jenkins Integration

I set up a Jenkins job to automatically create AWS infrastructure from our GitHub repos containing Terraform code. Here's a simplified version of the Jenkinsfile I created:

groovy

```
pipeline {
    agent any

    environment {
        AWS_ACCESS_KEY_ID     = credentials('AWS_ACCESS_KEY_ID')
        AWS_SECRET_ACCESS_KEY = credentials('AWS_SECRET_ACCESS_KEY')
    }
```

```
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/accenture/cloudshift-infra.git'
            }
        }

        stage('Terraform Init') {
            steps {
                sh 'terraform init'
            }
        }

        stage('Terraform Plan') {
            steps {
                sh 'terraform plan -out=tfplan'
            }
        }

        stage('Approval') {
            steps {
                input message: 'Do you want to apply this plan?', ok: 'Apply'
            }
        }

        stage('Terraform Apply') {
            steps {
                sh 'terraform apply -auto-approve tfplan'
            }
        }
    }

    post {
        always {
            cleanWs()
        }
    }
}
```

Ansible for Configuration Management

I used Ansible for configuration management of our EC2 instances. Here's an example of an Ansible playbook I wrote to set up a web server:

yaml

```
---
- name: Configure Web Server
  hosts: web_servers
  become: yes

  tasks:
    - name: Install Nginx
      apt:
```

```
      name: nginx
      state: present

  - name: Start Nginx Service
    service:
      name: nginx
      state: started
      enabled: yes

  - name: Copy Web Content
    copy:
      src: /path/to/web/content
      dest: /var/www/html
      owner: www-data
      group: www-data
      mode: '0644'

  - name: Configure Nginx
    template:
      src: nginx.conf.j2
      dest: /etc/nginx/nginx.conf
    notify:
      - Restart Nginx

handlers:
  - name: Restart Nginx
    service:
      name: nginx
      state: restarted
```

Container Orchestration with Kubernetes

I set up and managed Kubernetes clusters on AWS EKS for efficient container deployments. Here's an example of a Kubernetes deployment YAML I created for one of our services:

yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: 12345.dkr.ecr.ap-south-1.amazonaws.com/myapp:latest
        ports:
```

```
    - containerPort: 80
    resources:
      limits:
        cpu: 500m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 256Mi
```

CI/CD Pipeline

I maintained Jenkins for our CI/CD pipelines, integrating it with tools like Nexus, SonarQube, and Ansible. Here's a high-level overview of our pipeline:

5. Code Checkout from Git
6. Build with Maven
7. Unit Tests
8. Static Code Analysis with SonarQube
9. Package and store artifacts in Nexus
10. Build Docker image and push to ECR
11. Deploy to EKS using Ansible

Challenges and Solutions

12. **Legacy Application Compatibility**: Some of our legacy applications weren't cloud-native. I worked with the development team to containerize these applications and refactor them to work well in a cloud environment.
13. **Data Migration Complexity**: Migrating large amounts of data without downtime was challenging. I implemented a staged migration approach, using AWS Database Migration Service for databases and custom scripts for file data.
14. **Security Concerns**: Moving from a managed hosting to the cloud raised security concerns. I worked closely with our security team to implement VPCs, security groups, and IAM roles to ensure our AWS environment was secure.

Results and Learnings

By the end of the project, we had successfully migrated all our services to AWS, set up a robust infrastructure-as-code pipeline, and implemented efficient container orchestration. Our deployment times decreased significantly, and we gained much more flexibility in scaling our services.

I learned a ton about AWS services, Kubernetes, and infrastructure automation. This project really deepened my understanding of cloud architecture and DevOps practices, and I'm proud of what we accomplished.