Project Skyline: Legacy-to-Cloud Migration

Executive Summary

Project Skyline is a comprehensive initiative to migrate UNH's legacy on-premises applications to AWS cloud infrastructure. This migration aims to improve scalability, reduce operational costs, and enhance overall system reliability. The project encompasses the migration of core business applications, databases, and supporting services to a modern, cloud-native architecture.

Project Timeline

- **Start Date**: June 1, 2023
- **End Date**: April 30, 2024
- **Duration**: 11 months

Project Scope

1. Migrate 5 core business applications to AWS
2. Transition from on-premises MySQL databases to Amazon RDS
3. Implement containerization using Docker and orchestration with Amazon EKS
4. Establish a robust CI/CD pipeline using Jenkins and AWS services
5. Set up comprehensive monitoring and logging solutions
6. Implement disaster recovery and high availability strategies
7. Optimize cloud costs and resource utilization

Architecture Overview

Show Image

Key Technologies Used

- **Cloud Platform**: Amazon Web Services (AWS)
- **Containerization**: Docker, Amazon ECR
- **Orchestration**: Amazon EKS (Kubernetes)
- **CI/CD**: Jenkins, AWS CodePipeline
- **Infrastructure as Code**: Terraform
- **Configuration Management**: Ansible
- **Monitoring and Logging**: AWS CloudWatch, DataDog, Splunk
- **Database**: Amazon RDS (MySQL)
- **Networking**: Amazon VPC, Route 53, ELB

Implementation Details

1. Application Containerization

We containerized the legacy applications using Docker to ensure consistency across environments and facilitate easier deployment to EKS.

Example Dockerfile for one of the applications:

dockerfile

Copy

```
FROM node:14-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

2. Kubernetes Deployment

We used Terraform to provision the EKS cluster and Kubernetes manifests for deploying the applications.

Example Terraform script for EKS cluster:

hcl

Copy

```
module "eks" {
  source          = "terraform-aws-modules/eks/aws"
  cluster_name    = "skyline-cluster"
  cluster_version = "1.21"
  subnets         = module.vpc.private_subnets

  node_groups = {
    eks_nodes = {
      desired_capacity = 3
      max_capacity     = 5
      min_capacity     = 1

      instance_type = "m5.large"
    }
  }
}
```

3. CI/CD Pipeline

We implemented a Jenkins pipeline for continuous integration and deployment to EKS.

Example Jenkinsfile:

groovy

Copy

```
pipeline {
    agent any
    environment {
        AWS_ACCOUNT_ID="12345678"
```

```
        AWS_DEFAULT_REGION="us-east-1"
        IMAGE_REPO_NAME="skyline-app"
        IMAGE_TAG="${BUILD_NUMBER}"
        REPOSITORY_URI = "${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_DEFAULT_REGION}.amazonaws.com/${IMAGE_REPO_NAME}"
    }
    stages {
        stage('Build and Push Docker Image') {
            steps {
                script {
                    sh "aws ecr get-login-password --region ${AWS_DEFAULT_REGION} | docker login --username AWS --password-stdin ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_DEFAULT_REGION}.amazonaws.com"
                    sh "docker build -t ${IMAGE_REPO_NAME}:${IMAGE_TAG} ."
                    sh "docker tag ${IMAGE_REPO_NAME}:${IMAGE_TAG} ${REPOSITORY_URI}:${IMAGE_TAG}"
                    sh "docker push ${REPOSITORY_URI}:${IMAGE_TAG}"
                }
            }
        }
        stage('Deploy to EKS') {
            steps {
                script {
                    sh "kubectl set image deployment/skyline-app skyline-app=${REPOSITORY_URI}:${IMAGE_TAG}"
                }
            }
        }
    }
}
```

4. Monitoring and Logging

We set up comprehensive monitoring using AWS CloudWatch, DataDog, and Splunk.

Example CloudWatch dashboard configuration:

json

Copy

```
{
    "widgets": [
        {
            "type": "metric",
            "x": 0,
            "y": 0,
            "width": 12,
            "height": 6,
            "properties": {
                "metrics": [
                    [ "AWS/EKS", "cluster_failed_node_count", "ClusterName", "skyline-cluster" ]
                ],
                "view": "timeSeries",
                "stacked": false,
```

```
            "region": "us-east-1",
            "title": "EKS Failed Nodes"
          }
        }
    ]
}
```

5. Database Migration

We migrated on-premises MySQL databases to Amazon RDS using AWS Database Migration Service (DMS).

Example DMS task configuration:

json

Copy

```
{
  "MigrationType": "full-load-and-cdc",
  "TableMappings": {
    "rules": [
      {
        "rule-type": "selection",
        "rule-id": "1",
        "rule-name": "1",
        "object-locator": {
          "schema-name": "%",
          "table-name": "%"
        },
        "rule-action": "include"
      }
    ]
  },
  "ReplicationTaskSettings": {
    "TargetMetadata": {
      "TargetSchema": "",
      "SupportLobs": true,
      "FullLobMode": false,
      "LobChunkSize": 64,
      "LimitedSizeLobMode": true,
      "LobMaxSize": 32
    },
    "FullLoadSettings": {
      "TargetTablePrepMode": "DO_NOTHING"
    }
  }
}
```

Challenges and Solutions

8. **Challenge**: Ensuring zero downtime during migration **Solution**: Implemented blue-green deployment strategy using EKS and Route 53 for traffic management

9. **Challenge**: Managing secrets and configurations across environments **Solution**: Utilized AWS Secrets Manager and Kubernetes ConfigMaps/Secrets for secure configuration management
10. **Challenge**: Optimizing cloud costs **Solution**: Implemented auto-scaling policies, used Spot Instances for non-critical workloads, and set up AWS Cost Explorer alerts

Results and Benefits

11. Reduced infrastructure costs by 30% through optimized resource utilization
12. Improved application performance with 99.99% uptime
13. Reduced deployment time from hours to minutes
14. Enhanced security posture with AWS security best practices
15. Improved scalability to handle 3x the previous traffic load

Lessons Learned

16. Early involvement of all stakeholders is crucial for smooth migration
17. Thorough testing of all components in a staging environment prevents issues in production
18. Continuous monitoring and optimization are essential for maintaining efficient cloud operations

Future Recommendations

19. Implement serverless architectures for suitable workloads to further reduce costs
20. Explore multi-region deployment for enhanced disaster recovery
21. Implement Infrastructure as Code (IaC) for the entire infrastructure to improve repeatability and version control