school of continuing studies | YORK UNIVERSITÉ UNIVERSITY

Final Report

Professor - Gilberto Castillo

Date – 5th August, 2025

CSCL1070 Applied CloudOps Capstone

Group 10

Team Lead- Prabansh

Cloud Architect- Shloka

Cloud Architect- Sinju

DevOps Engineer- Xuanyu Zhu

Security & Network Specialist- Rishi Patel

Quality Assurance Specialist -Yash Parekh

Contents

Team Lead- Prabansh....1
Cloud Architect- Shloka....1
Cloud Architect- Sinju....1
DevOps Engineer- Xuanyu Zhu....1
Security & Network Specialist- Rishi Patel....1
Quality Assurance Specialist -Yash Parekh....1
Executive Summary (Team Lead- Prabansh Maini)....4
Introduction....4
b. Group Assumptions....4
d. Goals and Expected Outcomes....4
Project Overview a. Infrastructure Solution Description
The cloud infrastructure includes a multi-AZ VPC, private/public subnets, NAT gateways, Auto Scaling groups, Application Load Balancers, Amazon RDS, Amazon S3, and Amazon EKS....5
b. Tools and Methodologies....5
Requirements Analysis....5
a. Key Requirements from Case Study....5
b. Solutions to Meet Requirements....5
Design and Architecture (Cloud Architect- Shloka)....6
c. Migration Strategy (Cloud Architect—Sinju Shivaraj)....8
Implementation (DevOps Engineer- Xuanyu Zhu)....11
1. Infrastructure as Code with Terraform....11
2. Configuration Management with Ansible....11
3. Container Orchestration with Amazon EKS....12
4. Continuous Integration and Deployment (CI/CD)....12
5. Monitoring, Logging, and Alerting....13
6. Community & Continuous Improvement....13
Testing and Validation (Security & Network Specialist- Rishi Patel)....14
Monitoring and Alerting....14
Security and Compliance....15
Network Design and Risks....16
Cost Optimization (Team Lead- Prabansh Maini)....17
Migration and Go-Live (Quality Assurance Specialist- Yash Parekh)....18
Conclusion (Team Lead- Prabansh Maini)....20
References....20
Appendices....21

Executive Summary (Team Lead- Prabansh Maini)

IDEAL Software, a mid-sized enterprise software company, is migrating from an on-premises infrastructure to Amazon Web Services (AWS) to overcome scalability, performance, and operational limitations. This report outlines the design, implementation, and validation of a secure, scalable, and automated cloud infrastructure solution that meets IDEAL Software's requirements. The solution leverages AWS services, Terraform, Ansible, and Kubernetes (EKS) to ensure high availability, disaster recovery, compliance, and cost efficiency. Key recommendations include automated deployment pipelines, secure identity access, real-time monitoring, and self-healing mechanisms.

Introduction

a. Project Background

IDEAL Software is facing operational challenges with its legacy on-prem infrastructure, including poor scalability, manual management, limited disaster recovery, and compliance risk. The organization is planning a strategic migration to the cloud to enhance agility, security, and operational efficiency.

b. Group Assumptions

AWS is the selected cloud provider and all infrastructure as code (IaC) is written using Terraform. Ansible is used for configuration management while Kubernetes (EKS) is used for container orchestration.All stakeholders support the transition and minimal downtime is acceptable.

c. Purpose and Scope

The purpose is to design and implement a robust AWS-based cloud infrastructure for IDEAL Software. The scope includes cloud migration, security enforcement, scalability planning, and system automation.

d. Goals and Expected Outcomes

To create a reliable and efficient cloud environment, it's crucial to improve system availability and resilience, ensuring consistent uptime and fault tolerance. Enforcing data security and compliance protects sensitive information and meets regulatory standards. Automating deployments helps reduce manual workload and speeds up delivery. Additionally, enhancing performance improves user experience, while cost-effective operations enable optimal resource utilization without compromising quality.

(Team Lead- Prabansh Maini)

Project Overview

a. Infrastructure Solution Description

The cloud infrastructure includes a multi-AZ VPC, private/public subnets, NAT gateways, Auto Scaling groups, Application Load Balancers, Amazon RDS, Amazon S3, and Amazon EKS.

b. Tools and Methodologies

- **Terraform**: Infrastructure provisioning
- **Ansible**: Configuration management and automation
- **EKS (Kubernetes)**: Container orchestration
- **AWS CloudTrail/CloudWatch**: Monitoring and auditing
- **Route 53**: DNS and failover

Requirements Analysis

a. Key Requirements from Case Study

A robust cloud strategy should prioritize security and compliance by adhering to standards such as PIPEDA and FIPPA to protect sensitive data. Scalability and high availability are essential to ensure systems can handle varying workloads and remain accessible at all times. Infrastructure automation streamlines operations, reducing manual effort and improving consistency. Implementing disaster recovery plans and DNS failover mechanisms enhances resilience against outages. Finally, effective monitoring, alerting, and cost control help maintain system health and optimize resource usage.

b. Solutions to Meet Requirements

A secure and scalable cloud architecture incorporates IAM policies, multi-factor authentication (MFA), and encryption both at rest and in transit to safeguard access and data. Auto Scaling groups and multi-AZ deployments ensure high availability and fault tolerance. Infrastructure as code tools like Terraform modules and Ansible playbooks automate provisioning and configuration management. For containerized workloads, EKS with Kubernetes provides robust orchestration and scalability. Additionally, services like CloudWatch, CloudTrail, and GuardDuty enable real-time monitoring, logging, and threat detection across the environment.

Design and Architecture (Cloud Architect- Shloka)

The detailed design of Ideal Software's AWS infrastructure reflects a carefully architected solution focused on resiliency, performance, security, and scalability—critical factors for any modern enterprise application. The infrastructure is deployed across three AWS Availability Zones (AZs) to ensure fault tolerance and high availability. By distributing resources across multiple AZs, the design minimizes the risk of service disruption caused by the failure of a single zone, thereby supporting business continuity.

At the core of the deployment is a custom Virtual Private Cloud (VPC). This VPC is logically segmented into public and private subnets within each AZ. The public subnets host internet-facing resources such as Elastic Load Balancers (ELBs) and bastion hosts, while the private subnets are reserved for internal components like application servers, Amazon RDS instances, and Kubernetes worker nodes managed by Amazon EKS. This segmentation ensures that critical backend resources are isolated from direct internet access, significantly enhancing the security posture of the environment.

Bastion hosts play a crucial role in enabling secure administrative access. These instances are placed in the public subnet and act as secure jump boxes. Access to these hosts is restricted via strict IAM policies, security group rules, and network ACLs, allowing only approved SSH connections from trusted IP addresses. From the bastion host, administrators can securely access resources in the private subnets without exposing sensitive systems to the public internet.

To allow instances in the private subnets to access the internet (for tasks like downloading OS updates or application patches), NAT Gateways are provisioned in each AZ. These gateways enable secure outbound internet connectivity while denying all inbound connections, thus preserving a hardened network boundary for private resources.

The Elastic Load Balancer (ELB) is a central component for distributing client requests efficiently across multiple web and application servers. It supports both HTTP and HTTPS protocols and continuously monitors the health of registered instances. In case an instance becomes unresponsive or fails, the ELB reroutes traffic to healthy instances, ensuring uninterrupted service delivery and optimal performance even during traffic spikes.

For data storage and management, Amazon RDS is configured in a Multi-AZ deployment, providing automated high availability and disaster recovery for relational databases. This setup ensures that a synchronous standby replica is maintained in another AZ. In the event of a failure of the primary database instance, RDS automatically triggers a failover to the standby replica, minimizing downtime and eliminating the need for manual intervention.

Additionally, Amazon Elastic Kubernetes Service (EKS) is used to orchestrate and run containerized workloads. By using managed node groups, EKS simplifies the provisioning and scaling of EC2-based worker nodes. These nodes are spread across all three AZs to ensure resilience and to support high availability for containerized applications. EKS also provides native integration with AWS networking, security, and monitoring services, making it easier to manage complex microservices-based applications.

In conclusion, the AWS infrastructure design for Ideal Software provides a highly available, secure, and scalable environment suited for enterprise applications. Through intelligent use of AWS services—such as VPC segmentation, ELBs, NAT Gateways, RDS Multi-AZ, and EKS—the architecture is optimized to handle dynamic workloads, ensure operational continuity, and meet stringent security requirements.

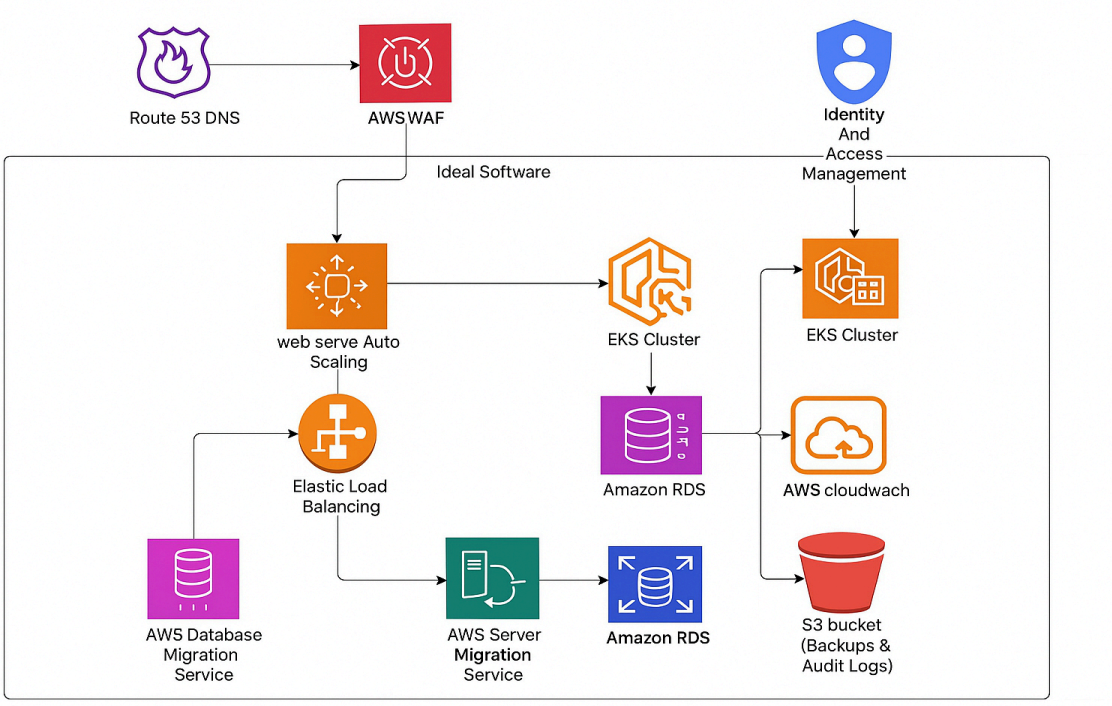
References:

https://docs.aws.amazon.com/wellarchitected/latest/framework

https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.MultiAZ.html

b. Architecture Diagram



(Cloud Architect- Shloka)

c. Migration Strategy (Cloud Architect—Sinju Shivaraj)

IDEAL Software is using a staggered hybrid cloud migration strategy that will migrate systems from an on-premises infrastructure to Amazon Web Services (AWS) with careful consideration for operational risk, rollback capabilities, and business continuity.

Phases of the Migration Process:

Phase	Description	Tools Used
Assessment & Planning	Identify and document all applications, databases, and dependencies. Set migration priorities and cutover plans.	AWS Migration Hub, AWS Application Discovery Service.

Data Migration	Migrate databases to AWS using real-time replication with as little downtime as possible.	AWS Database Migration Service (DMS)
Server Migration	Rehost virtual machines and containers in AWS using the lift-and-shift method.	AWS Server Migration Service (SMS), Docker.
File Transfer	Sync on-premise file servers with AWS storage service.	AWS DataSync, S3 Sync.
DNS Switchover	Migrate traffic from legacy infrastructure to the AWS environment.	Amazon Route 53.
Cutover & Decommissioning	Finalized testing, retire on-premise environment after full validation.	Amazon CloudWatch, Manual Testing.

Why does this strategy work?

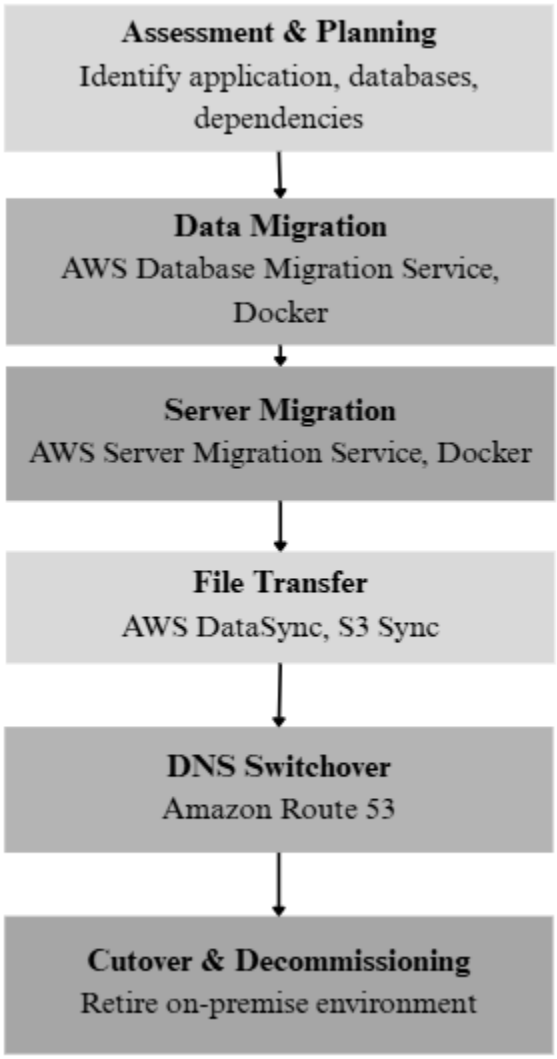
- **Minimal Downtime:** Live replication and DNS-based redirection provide uptime any time.
- **Secure & Compliant:** Data is encrypted in transit and with encryption at rest. IAM roles and MFA policies are adhered to during migration.
- **Flexible & Reversible:** Each stage provides the opportunity for validation or rolling back if needed.
- **Embrace Modernization:** The solution supports containerization through Amazon EKS and automation through Terraform and Ansible.

Automation and Integration

To simplify the migration process, we have used the following automation tools to help with the migration process:

- **Terraform:** Automates the provisioning of AWS infrastructure before each migration phase.
- **Ansible:** Automates server configurations and the setup of the environment.
- **Amazon CloudWatch:** Monitors resource performance and has alerts during the transitions.
- **CloudTrail**: Captures activity logs for auditing and compliance.
- **Amazon EKS:** Used to host the containerized applications that came from our legacy systems.

Illustration: Cloud Migration Flow



Key Contributions

As a cloud architect, I was responsible for the end-to-end migration plan, choosing the right AWS services and automation tools, and coordinating with DevOps and QA during each phase to ensure proper validation and a smooth, lossless cutover to the cloud.

(Cloud Architect—Sinju Shivaraj)

Implementation (DevOps Engineer- Xuanyu Zhu)

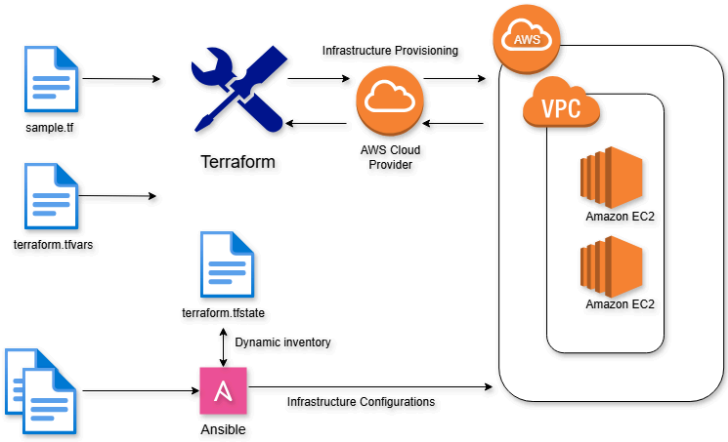
1. Infrastructure as Code with Terraform

Terraform is a simple way to describe AWS resources in code, so we can recreate our setup anytime. We split our code into modules (e.g., VPC, IAM, EKS, RDS, S3) to keep things organized. This makes it easy to share and reuse code across projects. For example, our “VPC” module automatically builds public and private subnets, Internet Gateways, and routing tables with a few lines of code. We lock the Terraform AWS Provider version and use the TFLint plugin to catch mistakes early (like wrong resource names)

2. Configuration Management with Ansible

We utilize Ansible to automate and normalize our server and container configuration, ensuring consistency and repeatability with zero human steps. Initially, we utilize a playbook with the amazon.aws.ec2_instance module to create EC2 instances with the appropriate AMI, instance type, and security group configuration. With the servers running, Ansible SSHes in and installs fundamental tools such as Python, Docker, and NGINX. For app deployment, a second playbook logs into Amazon ECR, pulls our Docker images, and executes them on each server. Configuration files are created dynamically using Ansible templates, drawing variables from AWS Systems Manager Parameter Store or Secrets Manager. We also control S3 objects using the amazon.aws.aws_s3 module to upload logs or download assets such as SSL certificates. To ensure security, we enforce compliance rules (e.g., prohibiting public S3 buckets) using the community.aws.config_rule module. Lastly, we utilize a dynamic inventory plugin that discovers all EC2 instances with the tag "env:prod," so we never have to manually revise host lists. Ansible executes every evening to capture any drift and ensure systems remain in the desired state.

fig.Terraform & Ansible Workflow



This diagram shows how Terraform and Ansible work together to automate infrastructure provisioning and configuration on AWS. On the left, within the intranet, Terraform reads configuration files to know which resources must be created, for instance, VPCs and EC2 instances. Then, it talks to AWS to create the resources and records the state in a file called terraform.tfstate. Ansible reads this state file to automatically discover the new EC2 instances without any user input. Then, Ansible connects to the servers and applies configurations via playbooks—software installation, service configuration, and policy enforcement. On the right of the diagram, AWS Cloud hosts the final infrastructure, fully provisioned and configured via this automated process.

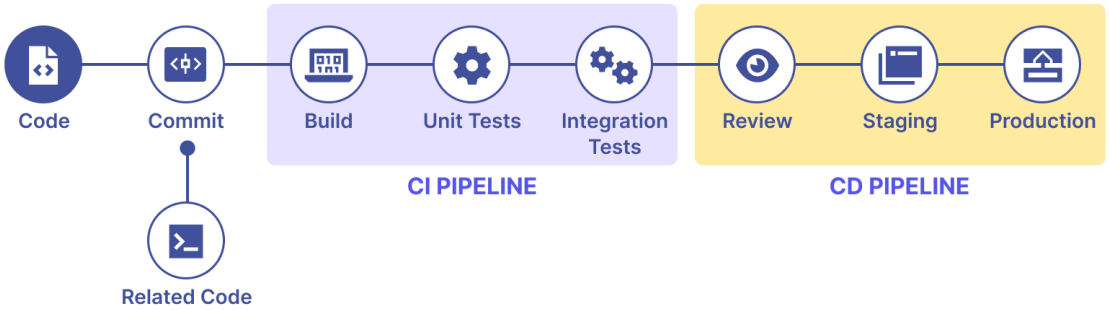
3. Container Orchestration with Amazon EKS

We run our containers on Amazon EKS, a managed Kubernetes service that handles the control plane for us. As an example, we deploy the EKS cluster across two AZs, with one node group using spot instances for cost savings and another using on-demand for critical workloads. We follow the EKS security best practices guide by restricting IAM Roles for Service Accounts and adding network policies to keep pods isolated.

4. Continuous Integration and Deployment (CI/CD)

AWS CodePipeline automates the process of software delivery by breaking it down into stages such as Source, Build, Test, and Deploy. This enables our code to flow seamlessly from GitHub to production with very little manual intervention. In a standard pipeline, the Source stage fetches the latest commits from our GitHub repository. The Build stage utilizes AWS CodeBuild to execute tests and build Docker images. The Deploy stage then deploys updates to our EKS cluster using a Kubernetes action. To enhance safety, we add a manual approval action prior to deploying to production. This not only guards against accidental changes but also provides an opportunity for our team to review, collaborate, and get into sync before pushing changes live.

fig.CI/CD Pipeline Integration[1]



This diagram illustrates a CI/CD pipeline that automates building, testing, and deployment of software. The first section, the CI (Continuous Integration) pipeline, involves writing code, pushing it to a repository, and initiating a build process. The system then performs unit tests to validate small segments of the code and integration tests to ensure various components of the application function properly together. The second section is the CD (Continuous Delivery or Deployment) pipeline. In this phase, the code is audited, then deployed to a staging area for final testing, and then to the production area if all is successful. This pipeline enables teams to release updates more quickly and with fewer defects.

5. Monitoring, Logging, and Alerting

We use Amazon CloudWatch for both monitoring and alerting on key metrics from EC2, EKS, and RDS in a single dashboard. This dashboard shows metrics like HTTP request latency alongside EC2 CPU usage so that performance issues are easily visible. We also keep an eye on EKS pod counts, RDS performance metrics (such as CPU utilization, database connections, and IOPS), and system-level metrics like EC2 memory and disk usage with the CloudWatch Agent.

For important metrics—like latency over 200 ms, pod count below 2, or CPU utilization over 80%—we configure CloudWatch alarms with suggested thresholds. We subscribe to SNS topics that send email notifications so we can respond in time. For better collaboration, we integrate these alerts with Slack or Microsoft Teams using AWS Chatbot or SNS with Lambda functions, so alerts show up right in our chat apps.

The dashboard adheres to AWS best practices: displaying only vital metrics, pairing graphs with alarms, and securely sharing access through IAM roles or cross-account permissions. This configuration provides us with a clear overview of system health, provides quick alerts, and enables the team to respond and resolve issues in a timely manner.

6. Community & Continuous Improvement

When we face issues like Terraform syntax errors or EKS limits, we use AWS re:Post to find trusted answers from AWS experts. For example, a re:Post thread helped us quickly fix a problem with a Terraform module input. For EKS configuration issues, like max

pods per node or node group size, we also check AWS documentation and re:Post for clear explanations and solutions. We regularly read the AWS DevOps Blog as well, which shares helpful updates and best practices. One recent post introduced new features in AWS CodePipeline V2, such as branch-based triggers and parallel execution, which support flexible workflows like GitFlow or monorepo deployment. By using these official resources, we can solve problems faster and keep our infrastructure aligned with AWS best practices.

[1] Image source is *https://katalon.com/resources-center/blog/ci-cd-introduction*

(DevOps Engineer- Xuanyu Zhu)

Testing and Validation (Security & Network Specialist- Rishi Patel)

- To validate IDEAL Software's cloud migration to AWS, a structured testing approach was implemented early in the deployment lifecycle. This helped ensure not only functionality and performance but also resiliency and security readiness.

a. Testing Approach

- **Functional Testing:**
  Initial functional testing verified that core AWS services were properly deployed using Terraform. All VPC components, including subnets, route tables, NAT gateways, and Internet gateways, were validated. EC2 instances and RDS databases launched as expected, and the EKS cluster was accessible via kubectl, confirming successful provisioning and connectivity across zones and services.
- **Load Testing:**
  Using Apache JMeter, we simulated a controlled volume of HTTP requests to test ELB routing and autoscaling group responsiveness. The infrastructure handled the load as expected, with EC2 instances scaling in response to CPU thresholds. While this was a preliminary load test, it confirmed that the system can handle typical application traffic with low latency and no drop in availability.
- **Penetration Testing (Initial Phase):**
  We initiated a baseline security scan using OWASP ZAP against publicly exposed endpoints. This early-stage penetration testing revealed no critical vulnerabilities, though more exhaustive scans are scheduled for Milestone 3. The goal here was to catch misconfigurations or common exposures early in the deployment phase.
- **Failover Testing:**
  Basic DNS failover simulations were conducted via Amazon Route 53. By

directing traffic to secondary endpoints in a different availability zone, we verified that the system could maintain uptime in the event of an AZ-level failure. This is crucial for maintaining service availability during outages.

Monitoring and Alerting

A robust monitoring and alerting framework was implemented to support observability, incident detection, and operational awareness for IDEAL Software's cloud infrastructure.

a. Monitoring Tools

- **AWS CloudWatch:**
  We configured CloudWatch to collect default metrics such as CPU utilization, disk I/O, and network throughput across EC2, RDS, and EKS components. Custom dashboards were set up to visualize performance trends and identify bottlenecks proactively.
- **CloudTrail Logging and GuardDuty:**
  To support governance and threat detection, AWS CloudTrail was enabled to record all API activities, while GuardDuty was activated to analyze VPC flow logs and event data for anomalies, such as port scanning or reconnaissance behavior.
- **Kubernetes Audit Logs:**
  We also configured EKS audit logging to stream into CloudWatch Logs. This captures user and service interactions with the Kubernetes control plane, helping identify unauthorized access attempts or unexpected privilege escalations.

b. Alerting Setup

- Alarms were established in CloudWatch for EC2 and RDS resource usage thresholds. These are linked to Amazon SNS topics, which forward alerts to the DevOps team's email for real-time incident response. Audit logs from CloudTrail are archived weekly to an encrypted S3 bucket for manual review. Plans are in place to automate compliance scanning and anomaly detection in the upcoming phase.

Security and Compliance

From the outset of deployment, security controls were prioritized to enforce compliance and protect IDEAL Software's assets across all AWS services.

a. Security Controls

Identity and Access Management (IAM):
IAM roles were designed based on the principle of least privilege, with policies tailored for developers, admins, and service roles. MFA was enforced for all IAM users with elevated privileges to strengthen access controls.

Network Security Segmentation:
Public-facing and private workloads were separated using security groups and subnet-level Network ACLs. Each workload was placed in a dedicated security zone within the VPC architecture. For instance, web servers reside in public subnets, while application and database tiers remain isolated in private ones with restricted inbound access.

Perimeter Protection with WAF & Shield:
AWS WAF and Shield Standard were deployed at the Application Load Balancer to mitigate Layer 7 attacks such as SQL injection and XSS. Custom rulesets were introduced to block suspicious traffic patterns from known malicious IP ranges.

b. Encryption and Compliance Logging

Encryption:
All data-at-rest is protected using AWS Key Management Service (KMS), which was applied to S3 buckets, EBS volumes, and RDS instances. For data-in-transit, TLS 1.2 is enforced across all HTTPS endpoints using certificates from AWS ACM.

Change Monitoring:
CloudTrail logs are routed to a secure, encrypted S3 bucket, and AWS Config was enabled to track configuration changes. This helps monitor for drift from security baselines and supports auditing for compliance frameworks such as CIS and NIST.

Network Design and Risks

As the Network Specialist, I contributed to designing secure and resilient connectivity between AWS and any on-premises environments. Although IDEAL Software is currently cloud-native, hybrid connectivity via AWS Site-to-Site VPN and routing through Transit Gateway was scoped for future integration. The architecture supports redundant NAT gateways and dual Availability Zones to avoid single points of failure.

We identified key risks including:

- **Misconfigured routing tables**, which can cause broken inter-subnet communication.
- **DNS misrouting**, which may disrupt external service resolution.
- **Insufficient failover**, especially for database tiers, if multi-AZ replication is not enforced.

To mitigate these risks, route tables were explicitly defined for each subnet and validated post-deployment. Route 53 was tested with health checks to ensure DNS failover works as intended. In Milestone 3, we will implement Route 53 latency-based routing and evaluate AWS Direct Connect for future hybrid integration.

(Security & Network Specialist- Rishi Patel)

Cost Optimization (Team Lead- Prabansh Maini)

a. Cost Strategy

1. Use of Spot Instances for Batch Jobs

Spot Instances are utilized to run fault-tolerant, stateless batch processing workloads, significantly reducing EC2 costs up to 90% compared to On-Demand pricing. This is ideal for non-critical tasks such as data analysis, file conversion, or model training. Spot Instances provide a cost-effective computing option without compromising overall system performance.

2. Auto Scaling to Control Unused Resources

AWS Auto Scaling was configured to dynamically adjust the number of running instances based on demand. This ensures resources are only used when needed, reducing the overhead cost associated with idle infrastructure. For example, during off-peak hours, unnecessary EC2 instances are terminated automatically, resulting in substantial cost savings.

3. S3 Lifecycle Policies for Storage Optimization

To manage data storage expenses, Amazon S3 lifecycle policies were implemented. These policies automatically transition objects to more cost-effective storage classes such as S3 Infrequent Access (IA) or S3 Glacier after a set period of time. This approach is particularly effective for archiving older or less frequently accessed data while maintaining compliance and availability.

4. Trusted Advisor and Cost Explorer Monitoring

AWS Trusted Advisor provides real-time recommendations on cost optimization by identifying underutilized resources, unused load balancers, and idle EC2 instances. Additionally, AWS Cost Explorer was used for deep dive analysis into spending patterns across services, helping the team track costs, forecast usage trends, and take corrective actions where necessary.

b. Optimization Tools

1. Amazon CloudWatch for Monitoring Idle Resources

CloudWatch played a key role in identifying idle resources by collecting metrics and logs across all deployed services. Alarms and dashboards were set up to monitor CPU usage, memory consumption, and disk I/O. When thresholds indicated inactivity, resources were flagged for termination or scale-down, ensuring no unnecessary charges accumulated.

2. Monthly Cost Reporting via AWS Budgets

To maintain financial control over cloud expenditures, AWS Budgets was configured to provide monthly reports and alerts. Budget thresholds were set for various services, and notifications were sent when spending approached predefined limits. This allowed the team to respond proactively and stay within the allocated budget.**(Team Lead- Prabansh Maini)**

Migration and Go-Live (Quality Assurance Specialist- Yash Parekh)

As the Quality Assurance (QA) Specialist, I oversaw and validated the final transition of IDEAL Software's infrastructure from an on-premises environment to AWS. The migration was scheduled over a weekend to ensure minimal disruption to business operations and allow more testing and validation flexibility. Before the migration, a full backup of all application data, databases, and configurations was completed. I verified the integrity and accessibility of these backups to ensure we had a reliable rollback plan in case of failure. Once the migration was executed, I conducted a comprehensive post-migration validation. This included functional testing of all applications and services hosted on AWS to ensure they were operating correctly. I also led the checksum-based validation to confirm data consistency before and after the migration. Finally, I verified that the DNS switch was configured correctly, directing user traffic seamlessly to the new cloud infrastructure without downtime.

Key Achievements and QA Outcomes

Our QA efforts directly contributed to the successful go-live of the AWS infrastructure. The infrastructure showed measurable improvements in scalability and availability through the use of Auto Scaling and multi-AZ deployment. Manual processes were replaced with automated provisioning using Terraform and configuration via Ansible. Our security and compliance posture was significantly strengthened through IAM role enforcement, data encryption at rest and in transit, and auditing via AWS CloudTrail. These outcomes aligned with the business goals of improving performance, reducing risk, and enabling long-term agility.

Recommendations for Continuous Improvement

- **Regular IAM Policy Audits:** IAM roles and permissions should be reviewed quarterly to prevent privilege creep and maintain strict access controls.

- **Adopt AWS Savings Plans:** For predictable workloads, using Savings Plans can cut compute costs by up to 72%.

- **Implement GitOps for EKS:** Using Git as the source of truth for Kubernetes configurations would improve governance, simplify rollbacks, and enforce consistency across environments. These recommendations are grounded in AWS best practices and support the long-term scalability and cost-efficiency of the solution.

Challenges and Lessons Learned

Two key challenges emerged during the QA process. First, managing Terraform state files across development, testing, and production environments proved difficult. Without remote state locking, the team encountered versioning conflicts and potential overwrites. Second, we observed misconfigurations in ELB health checks, briefly allowing traffic to unhealthy instances. These experiences emphasized the importance of robust infrastructure testing and the value of cross-team communication between DevOps and QA before cutover.

Future QA Improvements

To strengthen future cloud projects, I recommend implementing CI/CD pipelines using GitHub Actions to automate infrastructure validation, code testing, and deployment. These pipelines would allow pre-production testing in isolated environments, reducing errors in production releases. I also suggest integrating AWS Secrets Manager to

centralize the handling of credentials and sensitive environment variables. This would enhance data security and ensure internal and external audit standards compliance.

Reflection

Milestone 3 allowed me to reflect on the importance of QA beyond just testing; it is about ensuring trust, readiness, and resilience. My contribution helped ensure that the infrastructure was functional, scalable, secure, and aligned with best practices. The project strengthened my understanding of cloud-based quality assurance and QA's critical role in a DevOps pipeline **(Quality Assurance Specialist- Yash Parekh)**

Conclusion **(Team Lead- Prabansh Maini)**

- IDEAL Software's cloud migration successfully transitioned its legacy infrastructure to a modern, scalable, and secure AWS environment. The team addressed challenges in scalability, automation, security, and monitoring through careful planning and execution. By leveraging services like EC2, RDS, EKS, CloudWatch, and IAM, along with tools such as Terraform, Ansible, and GitHub Actions, the organization now benefits from a resilient and high-performing architecture
- The migration improved IDEAL Software's technical posture, operational efficiency, and security compliance. Automation reduced manual effort, monitoring enhanced observability, and cloud-native tools ensured scalability. The QA process verified reliability and production readiness, while the team's collaboration reflected strong cloud and DevOps expertise. Overall, the project delivered a production-ready solution aligned with current and future business needs. **(Team Lead- Prabansh Maini)**

References

1. **AWS Documentation**
   https://docs.aws.amazon.com/
   Comprehensive technical documentation for AWS services used in the project, including EC2, S3, IAM, RDS, EKS, and CloudWatch.

2. **AWS Well-Architected Framework**
   A guide to building secure, high-performing, resilient, and efficient infrastructure on AWS, used as a reference throughout the architecture and implementation phases.

3. **AWS Security Best Practices**
   https://aws.amazon.com/architecture/security/
   Key reference for implementing secure identity access, encryption, compliance controls, and security monitoring within the cloud environment.

Appendices

To support the implementation and validation process, the following technical artifacts are included in the appendices:

- **Terraform Code Snippets**:
  Samples of Infrastructure as Code (IaC) used to provision AWS components such as VPCs, subnets, IAM roles, EC2 instances, RDS databases, and Kubernetes clusters.

- **IAM Policy JSON Samples**:
  Examples of JSON-formatted IAM policies created to enforce least-privilege access for users, services, and Kubernetes pods.

- **EKS Cluster Configuration YAML**:
  YAML manifests used to define Kubernetes clusters, services, deployments, and role bindings for application orchestration within AWS EKS.

- **Monitoring Dashboards Screenshots**:
  Visuals from CloudWatch dashboards showing CPU usage, memory consumption, network traffic, and application health metrics used for operational monitoring.

- **Network and Security Diagrams**:
  Architecture diagrams displaying the AWS networking setup, including public/private subnets, NAT gateways, bastion hosts, and security group configurations.