



PROJECT REPORT

Scalable PHP and Flask Web Application Deployment on AWS EKS with
Multi-IaC Strategy and Full DevOps Lifecycle



JUNE 26, 2025

UST GLOBAL
India

Project Overview

This project delivers a comprehensive deployment of a highly available and disaster-resilient multi-tier web application on Amazon Web Services (AWS). It comprises a PHP-based frontend, a Flask/Python backend, and a MySQL database, all containerized and deployed within Amazon Elastic Kubernetes Service (EKS).

The application architecture is distributed across two AWS regions: **us-east-1** (primary) and **us-west-1** (disaster recovery). Infrastructure provisioning is automated through Terraform (for the primary region) and AWS CloudFormation (for the DR region). A complete CI/CD pipeline enables consistent builds, testing, and deployments, while integrated monitoring and automated DNS failover ensure high availability and rapid recovery in case of regional failure.

Application Stack

The application is built using a multi-tier architecture and utilizes modern, cloud-native technologies. The table below summarizes the core components of the stack:

Tier	Technology
Frontend	PHP (HTML/CSS rendered)
Backend	Flask (Python REST APIs)
Database	Amazon RDS MySQL (Multi-AZ)
Containerization	Docker
Orchestration	Amazon EKS (Kubernetes)
Load Balancing	Application Load Balancer (ALB)
Networking	VPC with public/private subnets, NAT Gateways, Route Tables

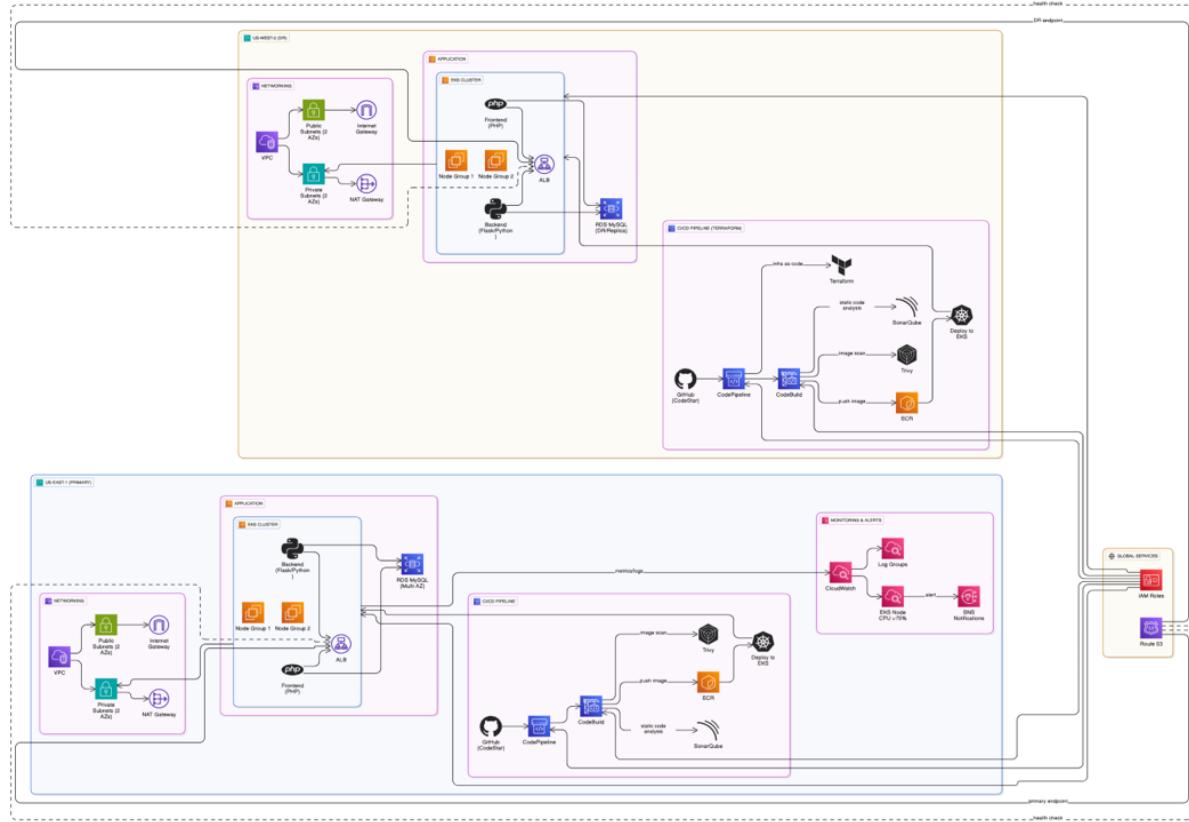
Project Overview & Architecture

Core Concepts

- **Multi-Tier Application:** A classic three-tier architecture separates the user interface (Frontend), business logic (Backend), and data storage (Database) for scalability and maintainability.
- **Infrastructure as Code (IaC):** The entire infrastructure is defined as code. The primary region (us-east-1) is deployed using **Terraform**, and the disaster recovery region (us-west-1) is deployed using **AWS CloudFormation**. This ensures consistent, repeatable, and version-controlled environments.
- **CI/CD Automation:** AWS CodePipeline automates the entire release process. Code changes pushed to GitHub trigger automated builds, security scans, and deployments to the EKS cluster.
- **Disaster Recovery (DR):** An active-passive DR strategy is implemented using Amazon Route 53's DNS failover. If the primary region becomes unhealthy, traffic is automatically rerouted to the standby region with minimal downtime.
- **Comprehensive Monitoring:** Amazon CloudWatch, including Container Insights, provides deep visibility into the application and infrastructure performance. Alarms are configured with Amazon SNS to notify the operations team of issues like high CPU usage or pipeline failures.

Architecture Diagram

The architecture consists of two mirrored regional deployments managed by a global Route 53 layer.



Common Architecture and AWS Resources Across Regions

To support high availability and fault tolerance, the application infrastructure is mirrored across two AWS regions: us-east-1 (primary) and us-west-1 (disaster recovery). The following AWS services and resources are consistently provisioned in both regions to ensure functional parity and seamless failover support.

Networking Layer

Resource	Description
VPC (Virtual Private Cloud)	Custom VPC created for the application in each region with a /16 CIDR block (e.g., 10.0.0.0/16), providing private IP addressing for all AWS resources.
Subnets total (4)	Each VPC is configured with subnets across 2 Availability Zones for high availability: 2 Public Subnets for the ALB and NAT Gateways, and 2 Private Subnets for the EKS worker nodes and RDS database.
Internet Gateway (IGW)	Attached to the VPC to allow internet access for resources in the public subnets, such as the Application Load Balancer.
NAT Gateways	Placed in each AZ's public subnet, allowing resources in the private subnets (like EKS nodes) to securely access the internet for updates and pulling container images.
Route Tables	Custom route tables direct traffic within the VPC; the public route table forwards traffic to the IGW, while private route tables forward outbound internet traffic through the NAT gateways.

Compute Layer: Amazon EKS

Resource	Description
Amazon EKS Cluster	A fully managed Kubernetes control plane named three-tier-cluster is deployed in each region to orchestrate the application's containers and deployments.
EKS Node Group (Managed)	An auto-scaled group of t3.medium EC2 instances is managed by EKS to serve as the worker nodes for the Kubernetes cluster, running across private subnets for high availability.
Security Groups for EKS	Granular security groups control traffic to and from the EKS nodes, allowing inbound traffic from the ALB, outbound traffic to the RDS instance on port 3306, and necessary internal cluster communication.
Amazon ECR	Private Elastic Container Registry repositories are created in each region to store the frontend and backend Docker images securely.

IAM (Identity & Access Management) Roles

Role	Usage & Permissions
EKS Cluster Role	Grants permissions for EKS control plane to interact with other AWS services (e.g., ELB, Auto Scaling, etc.). - AmazonEKSClusterPolicy, AmazonEKSServicePolicy
EKS Node IAM Role	Attached to the EC2 worker nodes to allow access to Amazon ECR and other AWS resources. - AmazonEC2ContainerRegistryReadOnly, AmazonEKSWorkerNodePolicy, AmazonEKS_CNI_Policy, CloudWatchAgentServerPolicy
CodeBuild Role	Executes pipeline build jobs. Has permissions to: - Build/push Docker images - Pull Secrets - Update EKS (via kubectl) - Needs inline policy for eks:DescribeCluster, eks:UpdateConfigMap - Needs AmazonEC2ContainerRegistryPowerUser
CodePipeline Role	Coordinates actions between GitHub (source), CodeBuild (build), and EKS (deploy). Includes permissions for: - codecommit:*, codebuild:*, eks:*, s3:*
CloudFormation Execution Role	Created during stack deployment. Allows CFN to create/modify AWS resources.

Prerequisites

Before you begin, ensure you have the following:

- An **AWS Account** with administrative privileges.
- **AWS CLI** installed and configured on your local machine.
- **Terraform** installed.
- **kubectl** (Kubernetes command-line tool) installed.
- **Docker Desktop** installed and running.
- A **GitHub Account** to host the application and IaC repositories.
- A registered domain name in **Amazon Route 53**.

Step-by-Step Implementation

This guide will walk you through setting up the primary region, building the application CI/CD pipeline, provisioning the DR region, and configuring failover and monitoring.

Part 1: Setting Up the GitHub Repositories

First, structure your code in GitHub. Based on the project, you will need at least three repositories:

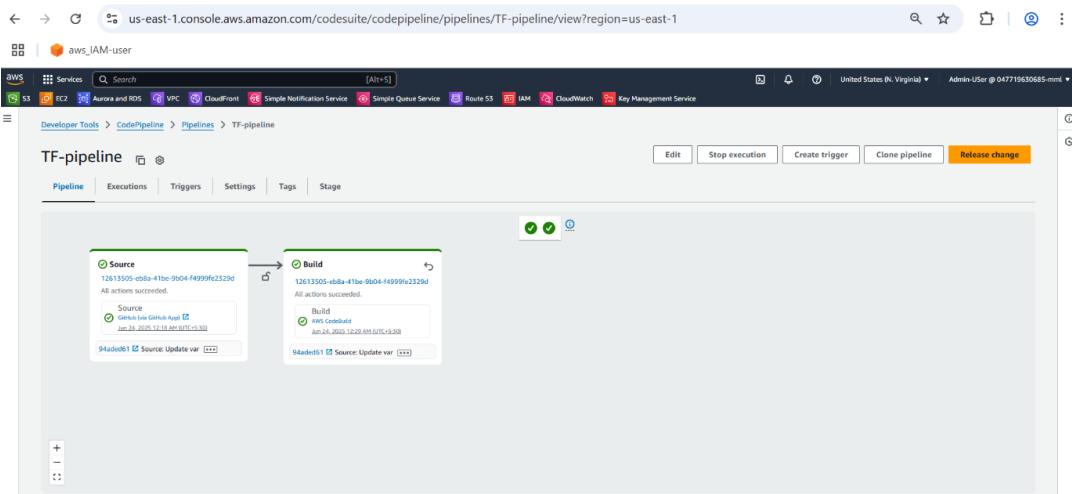
1. **Terraform-Configuration-Files**: Contains the Terraform code responsible for provisioning the **primary infrastructure in us-east-1**. This includes VPCs, subnets, compute resources, and other foundational AWS infrastructure.
2. **PHP-Flask-application**: Stores the application codebase, including:
 - PHP-based frontend
 - Flask-powered Python backend
 - Kubernetes manifests for deployment
 - buildspec.yml for the CI/CD pipeline This repository is linked to the infrastructure created via Terraform.
3. **CloudFormation-Template**: Hosts CloudFormation templates for setting up a **disaster recovery environment in us-west-1**. This includes infrastructure similar to the primary region and has **additional logic to automatically set up an application pipeline** (e.g., CodePipeline, CodeBuild).
4. **PHP-Flask-application-CF**: Similar to the PHP-Flask-application repo, this holds the app code and deployment configuration, but tailored for the **CloudFormation-managed DR region**. It mirrors the code structure with minor tweaks to align with the CF-based infrastructure setup.

Part 2: Provisioning the Primary Region (us-east-1) with Terraform

We will use a CI/CD pipeline to deploy our Terraform code, ensuring an automated and repeatable process.

Step 1: Terraform Infrastructure Pipeline

- **CI/CD Tool:** AWS CodePipeline (Region: us-east-1)
- **Pipeline Name:** TF-pipeline
- **Source Stage:**
 - Source Provider: GitHub (Version 2)
 - Repository: [Terraform-Configuration-Files](#)
 - Branch: main
- **Build Stage:**
 - Provider: AWS CodeBuild
 - This CodeBuild project is responsible for reading the Terraform configuration files and applying them.
 - It uses a [buildspec.yml](#) file



Once this pipeline runs, it provisions the following infrastructure in the us-east-1 region:

- A Virtual Private Cloud (VPC) with two public and two private subnets across two Availability Zones

The screenshot shows the AWS VPC dashboard with the 'Resource map' tab selected. It displays the following infrastructure components:

- VPC dashboard:** Shows the Main network ACL (ad-050fb5686d5555cfa), Default VPC (No), IPv4 CIDR (10.0.0.0/16), and IPv6 pool (None).
- Subnets:** Subnets within this VPC (4 total):
 - us-east-1a:** PublicSubnet1, PrivateSubnet1
 - us-east-1b:** PublicSubnet2, PrivateSubnet2
- Route tables:** Route tables (3 total):
 - rtb-0d5ee131bdaebc878
 - rtb-0a801ac77e1969a5f
 - rtb-0bee74d08b5e9d2a
- Network connections:** Connections to other networks (2 total):
 - igw-0defdb14f2f1f5448
 - nat-029f56adffef6f5575

- An Amazon EKS cluster named three-tier-cluster

The screenshot shows the Amazon Elastic Kubernetes Service (EKS) console with the 'Clusters' tab selected. The cluster 'three-tier-cluster' is displayed with the following details:

- Cluster info:**
 - Status: Active
 - Kubernetes version: 1.32
 - Support period: Standard support until March 21, 2026
 - Provider: EKS
- Overview:** Shows Cluster health (green), Upgrade insights (green), and Node health issues (green).
- Details:** Includes API server endpoint (<https://998f762ef058af80cf03574fee1551.gr7.us-east-1.eks.amazonaws.com>), Certificate authority (certificate chain), OpenID Connect provider URL (<https://oidc.eks.us-east-1.amazonaws.com/id/B98F762EF4058AF80CF03574FE1551>), Cluster IAM role ARN ([arn:aws:iam:047719630685:role/EKSClusterRoleTerraform](#)), and Platform version (eks.12).
- EKS Auto Mode:** Describes EKS automating routine cluster tasks for compute, storage, and networking to meet application compute needs.

- An EKS node group to host application pods

The screenshot shows the AWS EKS Cluster Compute Tab. On the left, there's a sidebar for 'Amazon Elastic Kubernetes Service' with sections for Dashboard, Clusters, Settings, Amazon EKS Anywhere, Related services (Amazon ECR, AWS Batch), and Documentation. The main area has tabs for Overview, Resources, Compute (which is selected), Networking, Add-ons, Access, Observability, Update history, and Tags. A message at the top says 'New AMI release versions are available for 1 node group. Learn more'. Below it, the 'Nodes (2)' section lists two nodes: 'ip-10-0-2-200.ec2.internal' and 'ip-10-0-3-254.ec2.internal', both t3.medium instances managed by 'three-tier-nodegroup'. A note at the bottom states: 'Amazon EKS will no longer publish EKS-optimized Amazon Linux 2 (AL2) AMIs after November 26th, 2025. Additionally, Kubernetes version 1.32 is the last version for which Amazon EKS will release AL2 AMIs. From version 1.33 onwards, Amazon EKS will continue to release AL2023 and Bottlerocket based AMIs.' Below the nodes is the 'Node groups (1)' section, showing one group named 'three-tier-nodegroup' with a size of 2 and AMI release version 1.32.3-20250610.

- An ECR Repositories for the frontend and backend images

The screenshot shows the AWS ECR Private Registry. The sidebar includes sections for Private registry (Repositories, Images, Permissions, Lifecycle Policy, Repository tags, Features & Settings) and Public registry (Repositories, Settings). Under 'Private registry', there's an 'ECR public gallery' section with links to Amazon ECS and Amazon EKS. The main area shows the 'Images (41)' for the 'backend' repository. It lists 41 image tags from v1.9 to v1.19, each with details like Pushed at, Size (MB), Image URI, Digest, and Last recorded pull time. Buttons for Delete, Details, Scan, and View push commands are at the top right.

The screenshot shows the AWS ECR Private Registry. The sidebar is identical to the previous screenshot. The main area shows the 'Images (40)' for the 'frontend' repository. It lists 40 image tags from v1.9 to v1.19, with similar columns and buttons as the 'backend' repository. This indicates that both the frontend and backend applications are being deployed using Docker images stored in ECR.

- An Amazon RDS MySQL database instance named three-tier-mysql

Step 2: Securely Store the Database Password

Hardcoding secrets is a security risk. Use AWS Systems Manager Parameter Store.

- Navigate to **Systems Manager > Parameter Store** in us-east-1.
- Create a parameter with the name `/my/RDS/DBPassword` and select the **SecureString** type.
- Store your RDS master password as the value. Your application will fetch this parameter at runtime.

Step 3: Create the Main Application Pipeline

1. In **CodePipeline**, create a new pipeline (e.g., three-tier-eks-app-pipeline).
2. **Source Stage:** Connect to your php-flask-application-CF GitHub repository.
3. **Build Stage:**
 - o Create a new **CodeBuild** project. This is the core of your CI/CD process.
 - o The buildspec.yml file in your application repository should perform the following actions:
 1. **Code & Image Scanning:** Integrate static code analysis with **SonarQube** and vulnerability scanning with **Trivy**. The build should fail if critical vulnerabilities are found.
 2. **Build Docker Images:** Build the frontend and backend Docker images.
 3. **Push to ECR:** Tag the images and push them to the ECR repositories created in the previous step.
 4. **Deploy to EKS:** Use kubectl commands to apply your Kubernetes manifests (deployment.yaml, service.yaml) to the three-tier-cluster. This will trigger a rolling update of your application pods.
4. **Deploy Stage (Optional - Direct EKS Deployment):**
 - o Alternatively, CodePipeline offers a direct "Deploy to EKS" action which can be used after the build stage.

The screenshot shows the AWS CodePipeline console with the URL <https://us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipelines?pipelines-meta=eyJmIjp7InRleHQiOiIifSwicyI6eyJwcm9wZXJ0eSI6InVwZGF0...>. The user is signed in as 'aws_IAM-user'. The top navigation bar includes links for S3, EC2, Aurora and RDS, VPC, CloudFront, Simple Notification Service, Simple Queue Service, Route 53, IAM, CloudWatch, and Key Management Service. The top right shows 'United States (N. Virginia)' and 'Admin-User @ 047719G10685-mnd'. The main content area displays the 'Pipelines' list under 'Developer Tools > CodePipeline > Pipelines'. The table lists four pipelines:

Name	Latest execution status	Latest source revisions	Latest execution started	Most recent executions
tesing_pipeline	Failed	Source - 608c6f66 Update buildspec.yaml	1 day ago	✖️✖️✖️✖️✖️ View details
TF-application-pipeline	Succeeded	Source - ba2815b6 Update index.php	2 days ago	✅✅✅✅✅ View details
TF-pipeline	Succeeded	Source - 94aded61 Update variables.tf	2 days ago	✅✅✅✅✅ View details
3-tier-eks-pipeline	Succeeded	Source - 693155b3 Update buildspec.yaml	8 days ago	✅✖️✖️✖️✖️ View details

Step 4: Verify Deployment in EKS

1. Navigate to your **EKS cluster > Resources > Services**.
2. You should see the services defined in your manifests, such as `php-service` (which creates a public-facing Load Balancer) and `app-service` for backend communication.
3. Access the Load Balancer's DNS to see your live application.

The screenshot shows the AWS EKS Services page for a cluster named "three-tier-cluster". The service "php-service" is selected. The "Details" section shows the service was created on June 24, 2025, at 10:44 UTC. It is in the "default" namespace, has a selector of "app:php", and is of type "LoadBalancer". The "Ports" section shows a single port mapping from port 80 to node port 51078. The "Endpoints" section shows two endpoints. The URL for the load balancer is a815bf85f135f474bb32022adeba9765-2050385105.us-east-1.elb.amazonaws.com.

The screenshot shows a web browser displaying the application running on the EKS load balancer. The URL is a815bf85f135f474bb32022adeba9765-2050385105.us-east-1.elb.amazonaws.com. The page title is "Add a Name". It features a text input field labeled "Enter a name" and a "Submit" button. Below the input field is a "User List" section containing three entries: "ram", "harish", and "raj".

Part 3: Provisioning the DR Region (us-west-1) with CloudFormation

For disaster recovery, we will provision a mirrored infrastructure stack in us-west-1 using CloudFormation.

Step 1: Create the CloudFormation Infrastructure Pipeline

1. In the us-west-1 region, create a new **CodePipeline** named CFT-pipeline.

The screenshot shows the AWS CodePipeline console with a pipeline execution ID of 52149da5. The pipeline consists of two stages: Source and Deploy. The Source stage is connected to a GitHub repository and has succeeded. The Deploy stage is connected to AWS CloudFormation and has also succeeded. The timeline shows the execution details for both stages, including the start and completion times.

2. **Source Stage:** Connect it to your CloudFormation-Template GitHub repository.

3. Deploy Stage:

- Choose **AWS CloudFormation** as the action provider.
- Select the "Create or update a stack" action mode.
- Provide a stack name, like CFT-infra.
- Specify your template file (CFT.yml) and parameter file (parameters.json).

The screenshot shows the AWS CloudFormation console with the Stacks page. A new stack named 'CFT-infra' is listed under the stacks table, with its status shown as 'CREATE_COMPLETE'. The Events tab shows the creation history of the stack, including events for MyNodeGroup, MyRDSInstance, and PrivateRoute resources.

- 4. Run the pipeline:** This will create a parallel set of resources (VPC, EKS, RDS) in the us-west-1 region.

VPC dashboard < [Alt+S]

Main network ACL: `act-056eab21d405d14` | Default VPC: No | IPv4 CIDR: `10.0.0.0/16` | IPv6 pool: -

IPv6 CIDR: - | Network Address Usage metrics: Disabled | Route 53 Resolver DNS Firewall rule groups: -

Owner ID: `047719630685`

Resource map Info

Subnets (4): Subnets within this VPC

- us-west-1b**: `PublicSubnet1`, `PrivateSubnet1`
- us-west-1c**: `PublicSubnet2`, `PrivateSubnet2`

Route tables (3): Route network traffic to resources

- `rtb-05e0d83d919500307`
- `rtb-0c6654c7701260431`
- `rtb-0559b0c31aeff9230`

Network connections (2): Connections to other networks

- `igw-024ea7c84521ac14b`
- `nat-0136d189f3b452bff`

5. ECR Repositories

Amazon Elastic Container Registry < [Alt+S]

Private registry: `Images`

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	Last recorded pull time
v1.10	Image	June 26, 2025, 12:45:05 (UTC+05.5)	213.43	Copy URI	<code>sha256:21a2aac1954cd1075bd463356c5a...</code>	June 26, 2025, 12:45:39 (UTC+05.5)
v1.9	Image	June 26, 2025, 12:41:56 (UTC+05.5)	213.43	Copy URI	<code>sha256:aa935069e100dd193a90cb930e645...</code>	June 26, 2025, 12:42:39 (UTC+05.5)
v1.8	Image	June 24, 2025, 12:30:39 (UTC+05.5)	213.43	Copy URI	<code>sha256:05a826ded288848bf4e43533161c7c...</code>	June 24, 2025, 12:31:15 (UTC+05.5)
v1.7	Image	June 23, 2025, 19:12:49 (UTC+05.5)	177.31	Copy URI	<code>sha256:a75101f7326fciae5e30db16192dbda...</code>	June 23, 2025, 19:14:15 (UTC+05.5)

Amazon Elastic Container Registry < [Alt+S]

Private registry: `Images`

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	Last recorded pull time
v1.10	Image	June 26, 2025, 12:44:47 (UTC+05.5)	177.32	Copy URI	<code>sha256:06d4175f44d0cb1ea50d19e4ad856...</code>	June 26, 2025, 12:45:39 (UTC+05.5)
v1.9	Image	June 26, 2025, 12:41:28 (UTC+05.5)	177.32	Copy URI	<code>sha256:2bd1a17b2a27d8a52f00e015fb962...</code>	June 26, 2025, 12:42:39 (UTC+05.5)
v1.8	Image	June 24, 2025, 12:30:21 (UTC+05.5)	177.31	Copy URI	<code>sha256:5580307fea94e29f72e10c9cefa02...</code>	June 24, 2025, 12:31:15 (UTC+05.5)
v1.7	Image	June 23, 2025, 19:12:49 (UTC+05.5)	177.31	Copy URI	<code>sha256:a75101f7326fciae5e30db16192dbda...</code>	June 23, 2025, 19:14:15 (UTC+05.5)

- An Amazon RDS MySQL database instance named three-tier-mysql

The screenshot shows the AWS RDS console for the 'three-tier-mysql' database instance. The 'Summary' tab is selected, displaying basic information like DB identifier, status, role, engine, and region. The 'Connectivity & security' tab is also visible, showing endpoint details, port, and security groups.

- Application pipeline

The screenshot shows the AWS CodePipeline console for a pipeline named 'CFT-infra-ThreeTierPipeline-BRs0BpnjWDLd'. The execution status is 'Succeeded'. The pipeline consists of three stages: Source, Build, and Deploy. Each stage has one action that succeeded.

Action name	Stage name	Status	Action provider	Started	Completed	Duration
GithubSource	Source	Succeeded	GitHub (via GitHub App)	1 hour ago	1 hour ago	3 seconds
CodeBuildBuild	Build	Succeeded	AWS CodeBuild	56 minutes ago	55 minutes ago	3 minutes 5 seconds
DeployToEKS	Deploy	Succeeded	Amazon EKS	53 minutes ago	52 minutes ago	1 minute 4 seconds

- EKS Cluster

The screenshot shows the AWS EKS console for the 'three-tier-cluster'. The 'Cluster info' section displays the Kubernetes version (1.32), support period (Standard support until March 21, 2026), and provider (EKS). The 'Resources' tab is selected, showing service and networking details. A table lists services under the 'default' namespace, including app-service, kubernetes, and php-service.

- Load balancer service of PHP(frontend).

The screenshot shows the AWS EKS Cluster Services page for a 'three-tier-cluster' cluster. The 'php-service' is selected. The 'Details' section shows the following configuration:

Setting	Value
Created	June 23, 2025, 19:14 (UTC+05:30)
Namespace	default
Finalizers	service.kubernetes.io/load-balancer-clean-up
Type	LoadBalancer
IP family policy	SingleStack
Cluster IPs	172.20.205.197
Selector	app:php
Session affinity	None
IP families	IPv4

The 'Ports' section shows one port mapping:

Protocol	Port	Target port	Node port
TCP	80	80	30525

The 'Endpoints' section shows two endpoints:

Endpoint	Port
a1296c5c3e9bba50b067748cd59e021-1271285800.us-west-1.lettamazonaws.com	80

Integrating Security Scanning (SonarQube & Trivy)

The project utilizes both SonarQube and Trivy for security scanning as part of its automated CI/CD pipeline. These tools are integrated into the AWS CodeBuild stage to ensure that code quality and container security are checked automatically before any deployment occurs.

SonarQube: Static Code Analysis

SonarQube is used as a static code analysis tool to inspect the PHP and Flask source code for quality issues, bugs, and security vulnerabilities before the application is even built.

- **Purpose:** It helps identify potential issues like code smells, security hotspots, and maintainability problems directly in the source code.
- **Integration:** It is run as a step during the CodeBuild process within the CI/CD pipeline.
- **Project Results:** The provided SonarQube dashboard screenshot for the "main" project shows that the code **Passed** the Quality Gate check, indicating it meets the configured quality standards. The scan reported **0 New Bugs** and **0 New Vulnerabilities**.

The screenshot shows the SonarQube dashboard for the 'main' project. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. A search bar is present at the top right. The main content area displays a large green box with the word 'Passed' and the message 'All conditions passed.' Below this, there are two tabs: 'New Code' (which is active) and 'Overall Code'. The 'New Code' tab shows statistics from June 23, 2025, with 0 New Bugs and 0 New Vulnerabilities. The 'Overall Code' tab shows a reliability rating of 'A'. At the bottom, there is a section for 'New Security Hotspots' with 0 results, a 'Reviewed' status, and a security review rating of 'A'.

Trivy: Container Vulnerability Scanning

Trivy is used to scan the Docker images after they are built to detect known vulnerabilities within their layers and packages.

- **Purpose:** It ensures that the container images being pushed to the Amazon Elastic Container Registry (ECR) are free from known security threats.
- **Integration:** This scan is performed within the CodeBuild step after the docker build command and before the docker push command.
- **Project Results:** The scan reports provided in the screenshots show the following findings for the version v1.15 images:
 - **Frontend Image (frontend:v1.15):** A total of 151 vulnerabilities were found, with 3 rated as CRITICAL.

The screenshot shows the Trivy report for the frontend image. At the top is a summary table:

Target	Type	Vulnerabilities	Secrets
047719630685.dkr.ecr.us-east-1.amazonaws.com/frontend:v1.15 (debian 12.11)	debian	151	-

Below the summary is a legend:

- : Not scanned
- 0 : Clean (no security findings detected)

Then there is a detailed table of vulnerabilities:

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
libc-bin	CVE-2025-4882	HIGH	affected	2.36-9+deb12u10		libc: static <code>setuid</code> binary <code>dlopen</code> may incorrectly search <code>LD_LIBRARY_PATH</code>
libc-dev-bin						
libc6						
libc6-dev						
libexpat1	CVE-2023-52425			2.5.0-1+deb12u1		expat: parsing large tokens can trigger a denial of service
	CVE-2024-8176		will_not_fix			libexpat: expat: Improper Restriction of XML Entity Expansion Depth in libexpat
libcu72	CVE-2025-5222		affected	72.1-3		libc: Stack buffer overflow in the <code>SBRRoot::addTag</code> function

At the bottom of the report are status indicators: Ln 1, Col 1, 94,908 characters, 70%, Unix (LF), and UTF-8.

- **Backend Image (backend:v1.15):** A total of 148 vulnerabilities were found, with 3 rated as CRITICAL.

The screenshot shows the Trivy report for the backend image. At the top is a summary table:

Target	Type	Vulnerabilities	Secrets
047719630685.dkr.ecr.us-east-1.amazonaws.com/backend:v1.15 (debian 12.11)	debian	148	-

Below the summary is a legend:

- : Not scanned
- 0 : Clean (no security findings detected)

Then there is a detailed table of vulnerabilities:

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
usr/local/lib/python3.11/site-packages/MarkupSafe-3.0.2.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/PyMySQL-1.1.0.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/blinker-1.9.0.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/cffi-1.17.1.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/click-8.2.1.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/cryptography-42.0.5.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/flask-3.0.2.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/itsdangerous-2.2.0.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/jinja2-3.1.6.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/pip-24.0.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/pyparser-2.22.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/setuptools-65.5.1.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/werkzeug-3.1.3.dist-info/METADATA						
usr/local/lib/python3.11/site-packages/wheel-0.45.1.dist-info/METADATA						

At the bottom of the report are status indicators: Ln 1, Col 1, 96,383 characters, 70%, Unix (LF), and UTF-8.

These reports provide a detailed list of vulnerabilities, allowing developers to address critical security issues by updating base images or libraries.

Part 5: Configuring Global Failover & Monitoring

Step 1: Configure Route 53 DNS Failover

1. Navigate to **Route 53** in the AWS Console (it is a global service).

The screenshot shows the AWS Route 53 console with the domain `nimbusnames.click` selected. The left sidebar shows navigation options like Dashboard, Hosted zones, Health checks, Profiles, IP-based routing, Traffic flow, Domains, Registered domains, Requests, and Resolver. The main panel displays domain details: Registration date (June 24, 2025), Auto-renew (Off), Domain status code (addPeriod ok), Name servers (ns-1048.awsdns-03.org, ns-1011.awsdns-62.net, ns-404.awsdns-50.com, ns-1910.awsdns-46.co.uk), Expiration date (June 24, 2026), Transfer lock (Off), and DNSSEC status (Not configured). Below this, there are tabs for Contact information, DNSSEC keys, and Tags. Under Contact information, it lists Registrant contact (Thumati Pavan Venkata Narendra Kumar, pavanthumati999@gmail.com) and Admin contact (Thumati Pavan Venkata Narendra Kumar, pavanthumati999@gmail.com). An 'Edit' button is also present.

2. Health Checks:

- o Create two health checks, one for the public endpoint (Load Balancer URL) of the application in us-east-1 and another for the endpoint in us-west-1.

The screenshot shows the AWS Route 53 console on the Health checks page. The left sidebar includes Health checks under the Route 53 section. The main area displays a table of existing health checks:

ID	Name	Details	Status in last 24 hours	Current s...	Alarm	State	Actions
3c6cdd75-fa9-4f49-9a4...	us-west-1-ALB-health-ch...	http://a1286c3c3e9db45...	Green	Healthy	None, Create alarm	Enabled	
7ca5f18f-4979-419b-9b...	us-east-1-ALB-health-ch...	http://a815bf85f135f47...	Green	Healthy	None, Create alarm	Enabled	

A 'Create health check' button is located at the top right of the table.

3. Hosted Zone Records:

- Go to your domain's Hosted Zone.
- Create two A records for your domain (e.g., nimbusnames.click).
- **Primary Record (us-east-1):**
 - Set **Routing policy** to **Failover**.
 - Set **Failover record type** to **Primary**.
 - Point the value to the us-east-1 Load Balancer.
 - Associate it with the us-east-1 health check.
- **Secondary Record (us-west-1):**
 - Set **Routing policy** to **Failover**.
 - Set **Failover record type** to **Secondary**.
 - Point the value to the us-west-1 Load Balancer.
 - Associate it with the us-west-1 health check.

Hosted zone details:

Record name	Type	Routing policy	Alias	Value/Route traffic to	TTL (s...)	Health ...	Evaluat...	Rec...
nimbusnames.click	A	Failover	Primary	dualstack.a15hf8f135f474...	-	7ca5f1bf...	Yes	Us-ea...
nimbusnames.click	A	Failover	Secondary	dualstack.a1286c3c5ebfb45...	-	3c6add75...	Yes	us-wes...
nimbusnames.click	NS	Simple	-	No	ns-1048.awsdns-03.org. ns-1011.awsdns-62.net. ns-404.awsdns-50.com. ns-1910.awsdns-46.co.uk.	172800	-	-
nimbusnames.click	SOA	Simple	-	No	ns-1048.awsdns-03.org.awv...	900	-	-

Add a Name

Enter a name

User List

- ram
- harish
- raj

Step 2: Configure CloudWatch Monitoring and Alarms

- Enable Container Insights:** For both EKS clusters, enable Container Insights to get detailed performance metrics on pods, nodes, and containers.

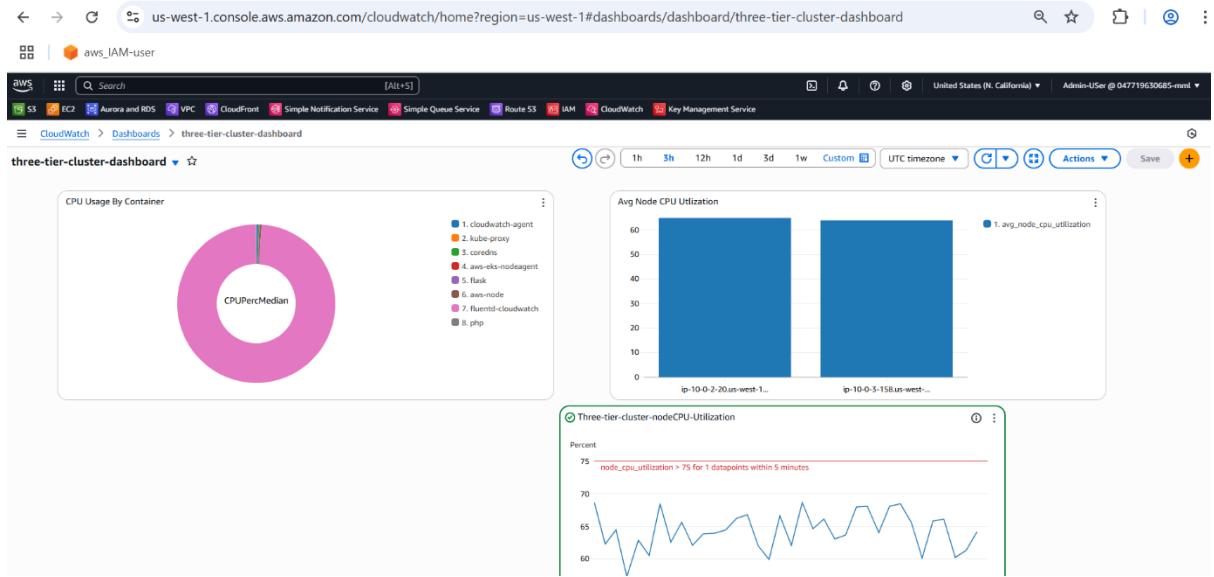
The following screenshots show the AWS CloudWatch Metrics and Container Insights interface for monitoring a three-tier cluster EKS cluster.

Screenshot 1: CloudWatch Metrics - Performance monitoring dashboard

Screenshot 2: CloudWatch Metrics - Container map

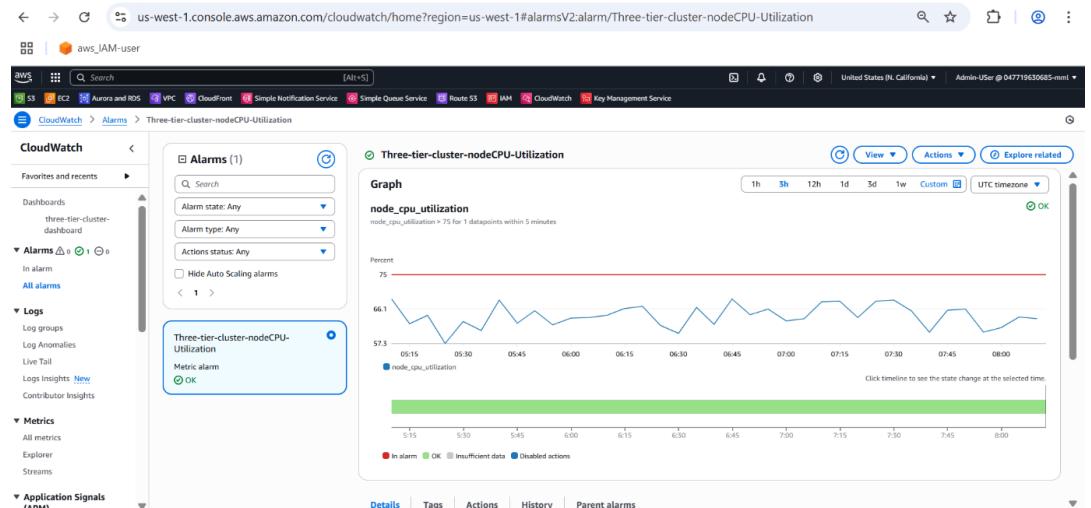
Screenshot 3: CloudWatch Metrics - Container map (Detailed view)

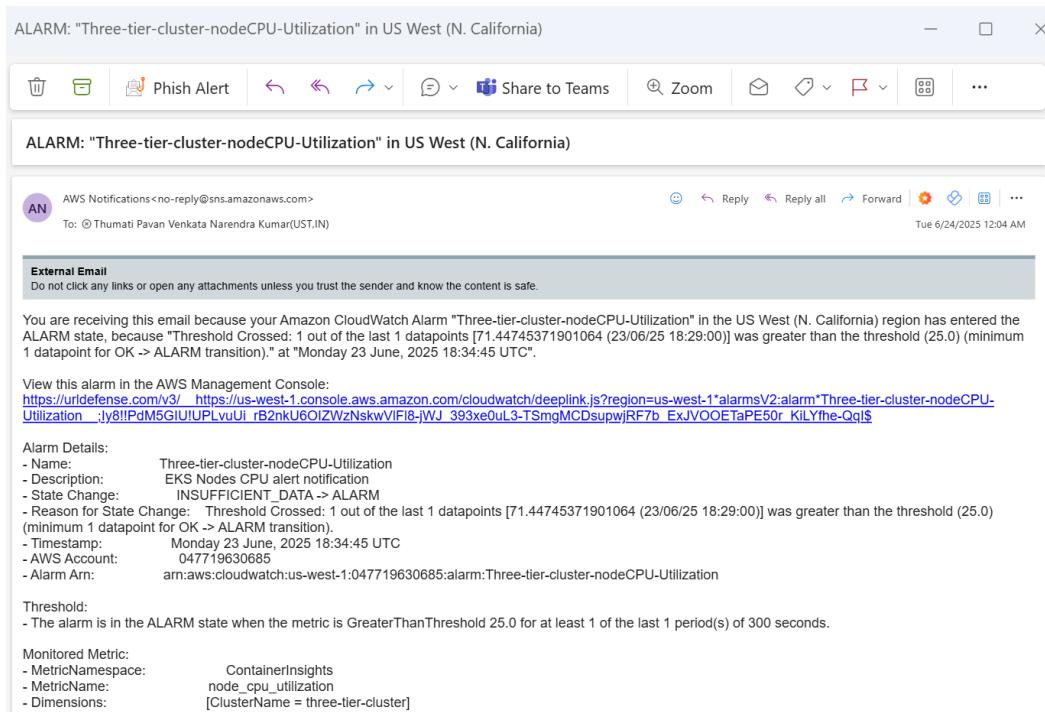
2. **Create a CloudWatch Dashboard:** Create a centralized dashboard to monitor key metrics like Node CPU Utilization, Memory Utilization, and Pod performance.



3. **Set Up Alarms:**

- In **CloudWatch > Alarms** (in the us-west-1 region for this example), create an alarm.
- Select the `node_cpu_utilization` metric from the `ContainerInsights` namespace for your cluster.
- Set a threshold, for example, $> 75\%$ for 5 minutes.





4. Configure SNS Notifications:

- Create an **SNS Topic** (e.g., three-tier-cluster-CloudWatch_Alarms_Topic).
- Create an email subscription to this topic and confirm it.
- Configure the CloudWatch alarm's action to publish a notification to this SNS topic whenever it enters the ALARM state.

us-west-1.console.aws.amazon.com/sns/v3/home?region=us-west-1#/topic/arn:aws:sns:us-west-1:047719630685:three-tier-cluster_CloudWatch_Alarms_Topic

three-tier-cluster_CloudWatch_Alarms_Topic

Details

Name: three-tier-cluster_CloudWatch_Alarms_Topic
ARN: arn:aws:sns:us-west-1:047719630685:three-tier-cluster_CloudWatch_Alarms_Topic
Type: Standard

Subscriptions (1)

ID	Endpoint	Status	Protocol
c00187ce-d742-40ae-89ff-c7e85a3c09b1	289219@outlook.com	Confirmed	EMAIL

Step 3: Set Up Pipeline Failure Notifications

1. Navigate to **Amazon EventBridge** and create a new rule.
2. Define an event pattern to listen for

The screenshot shows the 'failed-pipeline-eventRule' configuration in the Amazon EventBridge Rules section. The rule details are as follows:

- Rule name:** failed-pipeline-eventRule
- Status:** Enabled
- Event bus name:** default
- Type:** Standard
- Rule ARN:** arn:aws:events:us-east-1:047719630685:rule/failed-pipeline-eventRule
- Event bus ARN:** arn:aws:events:us-east-1:047719630685:event-bus/default

The **Event pattern** tab is selected, displaying the following JSON code:

```
1 { "source": ["aws.codepipeline"],  
2   "detail-type": ["CodePipeline Pipeline Execution State Change"],  
3   "detail": {  
4     "state": ["FAILED"]  
5   }  
6 }  
7 }
```

Below the event pattern, there is a **Copy** button.

CodePipeline Pipeline Execution State Change events where the state is FAILED.

3. Set the target as a new **Lambda function**. This function will parse the event and send a formatted message to a dedicated SNS topic (e.g., PipelineNotifications).

The screenshot shows the 'failed-pipeline-eventRule' configuration in the Amazon EventBridge Rules section. The rule details are the same as before. The **Targets** tab is selected, showing the following target configuration:

Details	Target Name	Type	ARN	Input	Role
▼	failed-pipeline-lambda-function	Lambda function	arn:aws:lambda:us-east-1:047719630685:function:failed-pipeline-lambda-function	Matched event	Amazon_EventBridge_Invoke_Lambda_1569190320

The input to the target is set to "Matched event".

4. Subscribe your email to the PipelineNotifications SNS topic to receive alerts when a pipeline fails.

[FAILED] Pipeline testing_pipeline

AWS Notifications<no-reply@sns.amazonaws.com>
To: Thumati Pavan Venkata Narendra Kumar(UST.IN)

Tue 6/24/2025 7:28 PM

External Email
Do not click any links or open any attachments unless you trust the sender and know the content is safe.

✖ Pipeline 'testing_pipeline' has FAILED.
Execution ID: 738581cf-c1d0-4403-8cf8-dd5670b28a44
Failed Stage: Error fetching stage: 'stageStates'
State: FAILED

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://urldfense.com/v3/_https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:047719630685:PipelineNotifications:937110d3-f37a-4646-a98f

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at [https://urldfense.com/v3/_https://aws.amazon.com/support_.!PdM5GIUITI130Q!9LC3ov8_8BQtlnlpcv94FZyfnVHkip_-EN7XCth-Ua2O4Q0unnQ2lsz_eYgEQ1H0ICRfMsUwh-J974E\\$](https://urldfense.com/v3/_https://aws.amazon.com/support_.!PdM5GIUITI130Q!9LC3ov8_8BQtlnlpcv94FZyfnVHkip_-EN7XCth-Ua2O4Q0unnQ2lsz_eYgEQ1H0ICRfMsUwh-J974E$)

Reply Forward

Conclusion and Next Steps

You have successfully deployed a robust, multi-region, three-tier application on AWS. This project showcases best practices in IaC, CI/CD automation, container orchestration, and disaster recovery.

Future Enhancements

- **Implement Auto-Scaling:** Configure Horizontal Pod Autoscaler (HPA) in EKS and EC2 Auto Scaling for the node groups based on the metrics you are now monitoring.
- **Automate Database Failover:** Enhance the DR strategy by automating the promotion of the RDS read replica in the DR region upon failover.
- **Cost Optimization:** Implement cost-saving measures such as using EC2 Spot Instances for worker nodes or Reserved Instances for RDS.
- **Expand Monitoring:** Expand the detailed monitoring coverage from the DR region (us-west-1) to the primary region (us-east-1) for a complete operational view.

Final Summary

This project successfully demonstrates the design and implementation of an enterprise-grade, highly available, and resilient web application on the AWS cloud. By leveraging a multi-region architecture with automated failover, the application ensures maximum uptime and business continuity. The use of a dual Infrastructure as Code strategy, with Terraform for the primary region and CloudFormation for the disaster recovery region, showcases flexibility in tooling while maintaining consistency and automation.

Furthermore, the integration of a full DevOps lifecycle via AWS developer tools, including a CI/CD pipeline with embedded security scanning (SonarQube, Trivy), ensures that deployments are rapid, reliable, and secure. The project is rounded out by a comprehensive monitoring and alerting system, providing the necessary observability to maintain operational excellence. Ultimately, this capstone serves as a practical blueprint for building and managing modern, scalable cloud-native applications.

References

- [AWS EKS Documentation](#)
- [Terraform AWS Provider](#)
- [Amazon RDS Cross-Region Replication](#)
- [AWS CI/CD Best Practices](#)