DevOps Onboarding

Table of Contents

1. Our Mission

At CoffeeBeans, our mission is to empower businesses by solving complex challenges and streamlining operations through innovation and technology. Our DevOps infrastructure plays a crucial role in this mission, enabling us to

create, optimize, and scale solutions that address our clients' unique needs efficiently and effectively.

Purpose-Driven DevOps at CoffeeBeans

Our infrastructure serves as the foundation for agile, resilient, and scalable solutions. Guided by three core pillars, our approach ensures that we maximize impact at every stage:

1. **Discover**
   Our first step is to fully understand our clients' goals, challenges, and objectives. We dive deep to uncover valuable insights, which inform the creation of tailored DevOps solutions. From identifying bottlenecks to forecasting needs, we're committed to building value into every aspect of our projects.

2. **Optimize**
   Complexity doesn't deter us; it motivates us. Our infrastructure is designed to simplify and streamline processes, enhancing performance and security while maintaining adaptability. This agile approach allows us to iterate quickly, delivering optimized solutions that support our clients' evolving requirements.

3. **Scale**
   We ensure our clients are set up for sustainable growth by building scalable solutions that evolve with their business. Our infrastructure supports seamless expansion, enabling clients to achieve high availability, reliability, and efficiency as they grow.

A DevOps Ecosystem Built for Results

Our DevOps infrastructure brings together best-in-class tools, practices, and methodologies to deliver rapid, reliable, and secure solutions. From CI/CD pipelines and automated testing to monitoring and feedback loops, every component is designed to enhance speed and accuracy while reducing risks. At CoffeeBeans, we're dedicated to fostering a culture of continuous improvement, ensuring our clients always stay ahead of the curve.

Welcome to CoffeeBeans, where DevOps innovation meets business transformation. Together, let's build solutions that don't just meet today's needs but anticipate tomorrow's opportunities.

---

2. Infrastructure Overview

QA Environment

The QA environment is designed to replicate Production, allowing for testing under real-world conditions. It includes:

- VPC with dedicated subnets for isolation and security.

- RDS for database needs, ensuring consistency with production schemas.
- EKS configured with zero-scale auto-scaling for cost efficiency during low usage periods.
- S3 buckets for application data storage, test data, and backups.

Production Environment

The Production environment mirrors the QA setup and includes:

- Robust security configurations for all resources.
- Continuous monitoring and scaling capabilities.

Development Environment

Similar to QA, the Development environment replicates the same configurations to ensure consistency across all deployment environments.

3. Infrastructure Components

- **Overview of VPCs**
  - **cb-apps-qa-vpc**: The primary VPC for QA applications, designed with subnets across availability zones in the **ap-south-1** region to enable redundancy and support for staging/testing applications.

  - **cb-apps-prod-vpc**: The primary VPC for production applications, similarly structured with subnets in the **ap-south-1** region, ensuring high availability and security for production workloads.

Detailed Subnet Information

QA Environment: `cb-apps-qa-vpc`

Name	Subnet ID	CIDR	Type	Availability Zone	Route Table
cb-apps-qa-public-subnet-ap-south-1a	subnet-08dde4860718fd0bc	10.0.0.0/20	Public	ap-south-1a	rtb-0ffaba7a0a218e13f
cb-apps-qa-public-subnet-ap-south-1b	subnet-0b78c29a290dcab9f	10.0.32.0/20	Public	ap-south-1b	rtb-0ffaba7a0a218e13f

cb-apps-qa-public-subnet-ap-south-1c	subnet-00fba51091f80511f	10.0.64.0/20	Public	ap-south-1c	rtb-0ffaba7a0a218e13f	
cb-apps-qa-private-subnet-ap-south-1a	subnet-070c8b211322a8440	10.0.16.0/20	Private	ap-south-1a	rtb-026ed541a68ce7f18	
cb-apps-qa-private-subnet-ap-south-1b	subnet-0048d2b8073932508	10.0.48.0/20	Private	ap-south-1b	rtb-026ed541a68ce7f18	
cb-apps-qa-private-subnet-ap-south-1c	subnet-05ac5099e0f2e7976	10.0.80.0/20	Private	ap-south-1c	rtb-026ed541a68ce7f18	

Production Environment: `cb-apps-prod-vpc`

Name	Subnet ID	CIDR	Type	Availability Zone	Route Table	N A
cb-apps-prod-public-subnet-ap-south-1a	subnet-073695d5e60417e21	11.0.0.0/20	Public	ap-south-1a	rtb-0ed0be2570f1c998b	a 0 5
cb-apps-prod-public-subnet-ap-south-1b	subnet-02ede94bc0aa2cd18	11.0.32.0/20	Public	ap-south-1b	rtb-0ed0be2570f1c998b	a 0 5
cb-apps-prod-public-subnet-ap-south-1c	subnet-0296ef3d833cba943	11.0.64.0/20	Public	ap-south-1c	rtb-0ed0be2570f1c998b	a 0 5
cb-apps-prod-private-	subnet-0fde6fbc1e28383e8	11.0.16.0/20	Private	ap-south-1a	rtb-0d9bc2a701d82bac0	a 0 5

subnet-ap-south-1a						
cb-apps-prod-private-subnet-ap-south-1b	subnet-061e2270eaf7d6292	11.0.48.0/20	Private	ap-south-1b	rtb-0d9bc2a701d82bac0	a 0 5
cb-apps-prod-private-subnet-ap-south-1c	subnet-07c9f90a78cb7d9b6	11.0.80.0/20	Private	ap-south-1c	rtb-0d9bc2a701d82bac0	a 0 5

Key Configuration Details

- **Auto-Assign Public IP**: Public subnets across both environments have the "Auto-Assign Public IP" option enabled, allowing instances to receive public IPs automatically for direct internet access if needed.
- **Availability Zones**: Subnets are distributed across multiple availability zones (`ap-south-1a` , `ap-south-1b` , and `ap-south-1c`), enhancing fault tolerance and reliability.
- **Route Tables**:
  - **Public Route Table**: `cb-apps-qa-public-rt` and `cb-apps-prod-public-rt` route tables handle traffic for the public subnets, providing access to the internet.
  - **Private Route Table**: `cb-apps-qa-private-rt` and `cb-apps-prod-private-rt` manage routing for private subnets, typically restricted to internal network access only.

RDS (Relational Database Service)

- **RDS Databases Overview**
  Your infrastructure includes two primary RDS PostgreSQL databases set up for different environments, each provisioned to meet specific availability and performance requirements.
  1. Production RDS Database
  - **DB Identifier**: `cb-apps-prod-rds-postgres-instancec10d699`
  - **Status**: Available
  - **Role**: Instance
  - **Engine**: PostgreSQL
  - **Region & Availability Zone**: `ap-south-1a`

- **Instance Type:** `db.m5.2xlarge`
- **CPU Utilization**: Typically low (example 1.46%)
- **Connections**: Currently, around 31 connections
- **Maintenance Status**: Available
- **VPC:** `vpc-0994e52cac44dd0a1`
- **Multi-AZ Configuration**: Enabled for high availability and failover support

2. QA RDS Database

- **DB Identifier:** `cb-apps-qa-rds-postgres-instance77310c9`
- **Status**: Available
- **Role**: Instance
- **Engine**: PostgreSQL
- **Region & Availability Zone:** `ap-south-1b`
- **Instance Type:** `db.t3.micro` (optimized for lower-cost testing)
- **CPU Utilization**: Low (example 5.13%)
- **Connections**: Around 3 active connections
- **Maintenance Status**: Available
- **VPC:** `vpc-05891489432055f36`
- **Multi-AZ Configuration**: Not enabled

RDS Snapshots

Manual snapshots are taken periodically to provide data recovery options. These snapshots contain all relevant data for each environment but exclude time-sensitive details.

- **Snapshot Name:** `cb-apps-qa-final-snapshot-rds-postgres`
  - **Engine Version**: PostgreSQL `13.15`
  - **DB Instance:** `cb-apps-qa-rds-postgres-instance`
  - **Size**: 20 GiB
  - **Storage Type**: gp3
  - **Region & Zone:** `ap-south-1c`
  - **Encryption**: Not enabled
  - **Port**: 5432
- **Snapshot Name:** `prod-latest`
  - **Engine Version**: PostgreSQL `13.15`
  - **DB Instance:** `cb-apps-prod-rds-postgres-instance`
  - **Size**: 20 GiB
  - **Storage Type**: gp3
  - **Region & Zone:** `ap-south-1a`
  - **Encryption**: Not enabled
  - **Port**: 5432

RDS Automated Backups

Automated backups are retained for data recovery:

- Production Backup (`cb-apps-prod-rds-postgres-instance`)
  - **Earliest Restorable Time**: Data available from recent retention period
  - **Latest Restorable Time**: Latest update within retention window
  - **Encryption**: Not enabled
- QA Backup (`cb-apps-qa-rds-postgres-instance`)
  - **Earliest Restorable Time**: Data available from recent retention period
  - **Latest Restorable Time**: Latest update within retention window
  - **Encryption**: Not enabled

RDS Subnet Groups

Custom subnet groups are assigned to each RDS environment for enhanced network configuration.

- **Production Subnet Group**
  - **Name**: `cb-apps-prod-db-subnet-group`
  - **Description**: Managed by Pulumi
  - **Status**: Complete
  - **VPC**: `vpc-0994e52cac44dd0a1`
- **QA Subnet Group**
  - **Name**: `cb-apps-qa-db-subnet-group`
  - **Description**: Managed by Pulumi
  - **Status**: Complete
  - **VPC**: `vpc-058914894320 55f36`

S3 Buckets : (Simple Storage Service)

- Here we have a breakdown of the S3 buckets used across ‘Coffee-Beans’ AWS infrastructure. Each bucket serves a unique purpose within the development, staging, QA, and production environments, covering data storage, backups, configurations, and application state management.

Bucket Overview

Bucket Name	Purpose
artifacts.cb-apps-403913.appspot.com	Stores artifacts and deployment packages for Coffee Beans applications.
cb-apps-403913-daisy-bkt-asia-south1	Regional bucket for data specific to Asia (South), supporting redundancy and latency.

cb-apps-403913-daisy-bkt-us-east1	Regional bucket for US East data, providing regional failover and latency optimization.
cb-faros-db-backup	Backup repository for Faros database instances, ensuring data recovery capability.
cb-infra	Infrastructure bucket for storing configurations, IaC state files, and setup artifacts.
cb-sql-dumps	Contains SQL dumps used for database migrations and restores.
cloud-ai-platform-6cf97148-6e4a-46a7-8eec-4336e43c9896	Bucket related to machine learning models and AI data in the cloud AI platform.
cloud-ai-platform-8df0c3e7-b129-4ffe-8dbb-c696ff84b602	Similar to above, used for AI model storage and dataset hosting for the AI platform.
coffee-book-data	Stores general application data for Coffee Book applications, including user content.
coffee-book-state	Maintains state files for Coffee Book, allowing stateful application persistence.
coffee-flow-data	Holds application-specific data related to Coffee Flow, primarily for operational data.
coffee-presentator-data	Stores data files and user content for Coffee Presentator applications.
coffee-presentator-state	State management bucket for Coffee Presentator, tracking application and user state.
coffeeflow-state	Maintains stateful information for Coffee Flow applications, ensuring session continuity.

cru-dev-dump	Contains development dumps for CRU applications, typically used for QA and testing.
databricks-2548301193643265	General data bucket for Databricks environment, storing operational and training data.
databricks-2548301193643265-system	Dedicated to system files and configurations for the Databricks environment.
databricks-2548301193643265-unitycatalog	Manages Unity Catalog metadata and asset data for Databricks governance.
nocodb-state	Stores state information for the NocoDB database management tool.
runapps_default-olp4jd	General-purpose storage for default applications within Runapps infrastructure.
staffing-tool-dev	Used in the development environment for Staffing Tool application data.
staffing-tool-prod	Holds production data for the Staffing Tool, critical for daily operations.
staffing-tool-qa	Used for QA data related to Staffing Tool, allowing isolated testing and validation.
staffing-tool-staging	Stores staging data for Staffing Tool, serving as a pre-production testing environment.
tf-state-cb-apps	Manages Terraform state files for Coffee Beans applications, ensuring IaC state persistence.
vault-creds	Secure bucket for storing credentials and secrets managed by HashiCorp Vault.

Detailed Descriptions

1. Artifact and Deployment Buckets

- artifacts.cb-apps-403913.appspot.com: This bucket is dedicated to storing deployment packages, artifacts, and binaries required by Coffee Beans applications. It acts as a central repository for artifacts needed during the CI/CD pipeline processes.

2. Regional Buckets

- **cb-apps-403913-daisy-bkt-asia-south1** and **cb-apps-403913-daisy-bkt-us-east1**: These buckets are configured in specific regions (Asia South and US East, respectively) to optimize latency for users in these areas. They store region-specific data for improved access and redundancy.

3. Database and Backup Buckets

- **cb-faros-db-backup**: This bucket is used for regular backups of the Faros database, providing disaster recovery options and data retention.
- **cb-sql-dumps**: Stores SQL dump files used for database migration, restoration, and data recovery processes.

4. Machine Learning and AI Platform Buckets

- **cloud-ai-platform-6cf97148-6e4a-46a7-8eec-4336e43c9896** and **cloud-ai-platform-8df0c3e7-b129-4ffe-8dbb-c696ff84b602**: These buckets are dedicated to hosting datasets, models, and related files for machine learning workflows and AI projects on the cloud AI platform.

5. Application-Specific Buckets

- **coffee-book-data**: Holds application data for Coffee Book, including user-generated content and app-specific files.
- **coffee-book-state**: Stores state data for Coffee Book, preserving user sessions, app status, and data continuity.

6. State Management Buckets

- **coffee-presentator-state** and **coffeeflow-state**: These state buckets allow stateful tracking for Coffee Presentator and Coffee Flow, helping applications maintain session continuity and user data.

7. DevOps and Infrastructure Buckets

- **cb-infra**: Contains infrastructure-related files, configuration files, and scripts used for setting up and maintaining environments.
- **tf-state-cb-apps**: Stores Terraform state files, enabling Infrastructure as Code (IaC) practices and tracking infrastructure states.

8. Version Control and Development Buckets

- **nocodb-state**: Holds state information for NocoDB, assisting in database and data table management.
- **staffing-tool-dev, staffing-tool-prod, staffing-tool-qa, staffing-tool-staging**: These buckets correspond to different environments for the Staffing Tool, providing environment-specific data segregation for development, testing, staging, and production.

9. Databricks Environment Buckets

- **databricks-2548301193643265**: General-purpose storage for Databricks, hosting datasets, and files needed for data analysis workflows.
- **databricks-2548301193643265-system**: Stores system configuration files essential for the Databricks setup and maintenance.
- **databricks-2548301193643265-unitycatalog**: Dedicated to Unity Catalog data, supporting asset management and data governance in Databricks.

10. Runapps and Miscellaneous Buckets

- **runapps_default-olp4jd**: General-purpose bucket for storing default application data within Runapps infrastructure.

11. Security and Credential Buckets

- **vault-credentials**: This bucket is used to securely store credentials and sensitive configuration details managed by HASHICORP Vault.

EKS (Elastic Kubernetes Service) + HELM CHARTS

- The Coffee Beans infrastructure utilizes Helm charts to deploy a range of applications within its Kubernetes clusters. Each application is hosted in either the **QA** or **Production** cluster using **AWS EKS**, serving specific business needs and development stages. This setup ensures clear separation between environments, promotes secure application management, and enables efficient scalability.

Applications by Environment

Application Name	Purpose	Environment
ai-interviewer	AI-based interview automation for evaluating candidate responses and gathering insights.	Production
argocd	GitOps-based continuous delivery management, automating Kubernetes resource deployments.	Production
birthday-bot	Automated bot for sending birthday messages to team members, enhancing internal engagement.	Production

cb-faros	Provides data visualization and insights into application and infrastructure performance.	Production
cert-manager	Automates SSL certificate issuance and renewal, ensuring secure application communication.	QA, Production
coffeebaba	Send motivational quotes on daily basis to slack to every CoffeeBeans personal to keep our employees moral high.	Production
feedback-tool	Tool for collecting user feedback to aid in product improvement and team decision-making.	Production
hasura	GraphQL API engine that generates instant APIs from PostgreSQL databases for rapid prototyping.	Production
istio-system	Service mesh for managing and securing service-to-service communication within clusters.	QA, Production
listmonk	Email marketing and newsletter management tool for customer outreach and engagement.	Production
lunch-bot	Coordinates team lunch polling and planning to facilitate team-building activities.	Production
proposal-gpt	AI-driven tool for generating business proposals tailored to specific client needs.	Production
rewards	Rewards and recognition system to encourage and acknowledge employee achievements.	Production
sonarqube	Tool for continuous code quality inspection, identifying issues and potential vulnerabilities.	Production
staffing-tool	Manages staffing information and HR processes related to employee allocation and planning.	Production
tsa-bot	Tracks time and attendance, automating HR data collection for internal reporting.	Production
vault	Manages and secures secrets and sensitive information across applications.	QA, Production
tabby-qa	QA environment deployment of the Tabby application for pre-release testing.	QA
taiga-qa	QA deployment of Taiga project management tool for task tracking and collaboration.	QA

coffee-presentator-dev	Development environment for Coffee Presentator, testing new features and updates safely.	Development
coffee-table-test	Test environment for the Coffee Table application, ensuring new versions are production-ready.	Test
coffee-book-test	Test environment for Coffee Book application, validating changes before deployment.	Test

---

Application Descriptions

Core Production Applications

1. **ai-interviewer**: Deployed to facilitate automated interview processes by using AI to assess candidate answers and generate insights, streamlining the recruitment process.
2. **argocd**: Manages GitOps for continuous delivery, automatically syncing changes from Git repositories to Kubernetes, providing full visibility and easy rollbacks for deployments.
3. **birthday-bot**: Enhances team morale by sending birthday messages to team members, creating a personalized touch within the workplace.
4. **cb-faros**: Displays key application and infrastructure performance metrics, assisting with monitoring and optimization efforts.
5. **cert-manager**: Automates the process of issuing and renewing SSL/TLS certificates, critical for secure application communications across both QA and Production clusters.
6. **coffeebaba**: Send motivational quotes on daily basis to slack to every CoffeeBeans personal to keep our employees moral high.
7. **feedback-tool**: Collects feedback from users to aid in identifying and implementing necessary product enhancements.
8. **hasura**: Generates GraphQL-based APIs from PostgreSQL databases, allowing for rapid prototyping and easier API management.
9. **istio-system**: Provides traffic management, security, and observability across microservices, supporting inter-service communication with advanced monitoring and security policies.
10. **listmonk**: Manages email marketing campaigns, newsletters, and customer engagement for marketing efforts.
11. **lunch-bot**: Organizes team lunch polls and voting, facilitating team cohesion through scheduled activities.
12. **proposal-gpt**: Uses AI to generate proposals tailored to specific client and project requirements, helping automate business development processes.

13. **rewards**: Encourages a positive workplace environment through rewards and recognition features.
14. **sonarqube**: Analyzes code for bugs and vulnerabilities, ensuring code quality and security compliance for Coffee Beans applications.
15. **staffing-tool**: Centralizes staffing and HR data to support employee allocation and planning.
16. **tsa-bot**: Automates time tracking and attendance, simplifying HR data collection and reporting.
17. **vault**: Manages secrets and sensitive configuration data, with support for encrypted storage and controlled access.

QA and Development Applications

1. **tabby-qa**: QA deployment for testing new features and ensuring stability before updates reach production.
2. **taiga-qa**: QA version of the Taiga project management tool, helping teams test project tracking functionality in a non-production setting.
3. **coffee-presentator-dev**: Dedicated development instance of the Coffee Presentator app, facilitating safe testing of new functionality and updates.
4. **coffee-table-test**: Testing deployment of the Coffee Table application, used to verify stability and performance before production.
5. **coffee-book-test**: Test environment for the Coffee Book application, ensuring quality and consistency in the codebase.

Deployment and Management

Helm charts are used to maintain consistency across environments, enabling versioned deployments, rollbacks, and automated updates. Applications can be managed with standardized configurations, simplifying deployment and scaling as needed.

Security and Access Control

- **Role-Based Access Control (RBAC)** policies restrict access to applications based on roles, ensuring only authorized personnel have necessary access.
- **Istio** provides added layers of security, handling authentication, authorization, and encryption for service-to-service communication.
- **Vault** manages sensitive information, protecting credentials, and managing access through a secure mechanism.

Backup and Recovery

Automated backup strategies are in place for critical applications and databases, with recovery protocols documented to ensure resilience and quick restoration if necessary.

EC2 (Elastic Compute Cloud)

- Here’s a detailed overview of the EC2 instances in the Coffee Beans infrastructure, specifically focusing on bastion hosts and Gitea runners.

EC2 Instance Details

Instance Name	Purpose
prod-bastion	Bastion host for secure access to production resources.
qa-bastion	Bastion host for secure access to QA environment resources.
gitea	EC2 instance hosting the Gitea service, managing source control repositories.
gitea-act-runner-1	Gitea Actions runner instance for handling CI/CD jobs triggered within Gitea.

Instance Descriptions

1. prod-bastion

- **Purpose**: Acts as a secure entry point into the production environment, allowing controlled SSH access to production resources.
- **Access Control**: Limited to specific IP addresses and users, following strict security protocols.
- **Usage**: Enables DevOps and other authorized team members to securely access production instances and services, ensuring restricted and monitored access.

2. qa-bastion

- **Purpose**: Provides a secure entry point for the QA environment, allowing team members to access QA resources in a controlled manner.
- **Access Control**: Configured similarly to the production bastion with IP whitelisting and user access management.
- **Usage**: Supports testing and validation efforts by granting secure access to QA instances, helping teams maintain an isolated yet accessible QA environment.

3. gitea

- **Purpose**: Hosts the Gitea service, which functions as a self-hosted version control platform, managing source code repositories for the Coffee Beans projects.
- **Usage**: Centralized version control for development projects, enabling efficient code management, pull requests, and issue tracking.
- **Additional Features**: Supports integration with CI/CD workflows and provides repository insights for development and deployment activities.

4. gitea-act-runner-1

- **Purpose**: Dedicated runner instance for Gitea Actions, facilitating continuous integration (CI) and continuous deployment (CD) tasks triggered within the Gitea platform.
- **Usage**: Executes workflows, builds, tests, and deployments as specified in Gitea repositories, enabling automated processes to improve deployment efficiency.
- **Integration**: Configured to work seamlessly with the Gitea instance, this runner handles multiple jobs and scales as required to meet CI/CD demands.

Elastic Container Registry (ECR)

- The Coffee Beans ECR repositories are organized by application and environment, providing distinct containers for development (`dev`), quality assurance (`qa`), and production (`prod`). Each container image is tagged appropriately for version control and environment segmentation, simplifying deployment and rollback operations within Kubernetes.

Repository List

Repository Name	Description
coffeebeans	Primary repository holding container images for a range of applications. These images are deployed across various environments and updated regularly.

Container Images

Each image below corresponds to a specific application or microservice within the Coffee Beans infrastructure. Images follow a consistent naming convention to indicate their role and environment, ensuring clarity and ease of management.

Image Name	Purpose	Environment
ai-interview-tool	AI tool for conducting and evaluating candidate interviews.	Production
ai-interviewer	Backend for the AI Interviewer, used for processing candidate responses.	Production
birthday-bot	Sends birthday notifications within Coffee Beans, promoting team engagement.	Production
cb-attendance-backend	Backend for the attendance tracking system, part of internal HR functions.	Production

cb-attendance-web	Web frontend for attendance tracking, accessible to employees.	Production
cb-faros-db-backup	Manages backups for the database of the Faros data visualization tool.	Production
cb-feedback-frontend	Frontend for the feedback tool, gathering user input for product improvements.	Production
cb-feedback-tool-backend	Backend for the feedback tool, processing and storing user feedback.	Production
cb-mail	Manages internal email services and notifications within the organization.	Production
cb-presentator	Application used for presenting internal metrics and reports.	Production
cb-staffing-tool-backend	Backend for staffing tool used by HR for employee allocation and planning.	Production
cb-website	Legacy version of the main Coffee Beans website.	Production
cb-website-cms	Content management system backend for updating the Coffee Beans website.	Production
cb-website-new	New version of the Coffee Beans website with updated content and features.	Production
cbmailer	Internal mailing service, used for sending automated notifications.	Production
cbrewards-slackbot	Slack bot used for rewards and recognition within the team.	Production
coffeebaba	Analytics tool for tracking user behavior and application usage.	Production
coffeebaba-slackbot	Slack bot extension for CoffeeBaba, offering insights and notifications.	Production
coffeebeans-website	Main Coffee Beans website container, holding front-end assets and backend logic.	Production
cru-nginx-proxy	NGINX reverse proxy configuration, handling web traffic routing.	Production
dev-rewards-be	Backend for rewards system in the development environment.	Development
dev-rewards-fe	Frontend for rewards system in the development environment.	Development

dev/cb-mailer	Development environment for the internal email service.	Development
dev/cb-staffing-tool-frontend	Development frontend for staffing tool, tested before production deployment.	Development
dev/cb-website-new	Development version of the new Coffee Beans website, used for pre-release testing.	Development
feedback-tool-backend	Backend for the feedback collection tool.	Production
feedback-tool-frontend	Frontend for the feedback collection tool, accessible by employees and customers.	Production
genai-ocr	OCR tool leveraging Generative AI, used for document processing and data extraction.	Production
hello-chart	Example application for testing deployments and Helm configurations.	QA
lunch-bot	Bot organizing team lunches, polling, and planning activities within Coffee Beans.	Production
n8n	Workflow automation tool used for internal data processing and task automation.	Production
nocodb	No-code database application, providing data visualization and easy manipulation.	Production
outline	Knowledge management tool, centralizing documentation for the team.	Production
prod-rewards-be	Production backend for the rewards and recognition system.	Production
prod-rewards-fe	Production frontend for the rewards and recognition system.	Production
prod/cb-staffing-tool-frontend	Production version of the staffing tool's frontend.	Production
prod/cb-website-new	Production version of the updated Coffee Beans website.	Production
proposal-gpt	AI-powered tool for generating tailored client proposals.	Production
qa/slackbot	Slack bot for the QA environment, used to test internal communications and automation.	QA

redis	Redis image used for in-memory data caching across various applications.	Production
rewards-backend	Backend microservice for handling rewards functionality.	Production
rewards-frontend	Frontend for the rewards application, used by employees to track and redeem rewards.	Production
staffing-tool-be	Backend for staffing tool, providing APIs for employee and project data.	Production
staffing-tool-fe	Frontend for staffing tool, offering an interface for HR management.	Production
tabby	Application related to the internal team, supporting task and project tracking.	Production
tsa-bot	Time and attendance bot used for tracking team member attendance and productivity.	Production

Deployment and Management

Each container image within the **coffeebeans** repository is tagged by version and environment to facilitate proper deployment across Kubernetes clusters. Images are updated as part of the CI/CD pipeline, with strict version control and rollback capabilities to ensure minimal downtime and efficient update processes.

CI/CD Workflow

1. **Build and Push**: Containers are built and tagged through automated CI/CD pipelines, pushed directly to the ECR repository.
2. **Versioning**: Semantic versioning (`vX.Y.Z`) is applied to images, allowing precise control over updates and rollbacks.
3. **Environment Segmentation**: Separate images exist for development, QA, and production, with environments isolated to prevent cross-environment conflicts.
4. **Monitoring and Logging**: Continuous monitoring of ECR usage and image pull requests helps track deployment health and resource usage.

4. Configuration Management

HashiCorp Vault

- **Purpose**: Securely manages application secrets, configurations, and environment variables.

- **Scope**: Ensures that all environments can access secrets securely and that updates are consistent.

---

5. Migration of Data and Resource

Migration from GCP to AWS

Recently we moved from GCP to AWS, The migration involved transferring resources and workloads from Google Cloud Platform (GCP) to Amazon Web Services (AWS). This transition was meticulously planned and executed using automated scripts, Pulumi for Infrastructure as Code (IaC), and various migration techniques. Below are the details of the migration, categorized by resource type.

Resource Migration Details

1. Buckets

- **GCP**: Google Cloud Storage (Buckets)
- **AWS**: Amazon Simple Storage Service (S3)
- **Migration Method:**
  - **Script**: `bucket_backup_restore.py`
    - This script backs up data from GCP buckets and restores it to S3 buckets in AWS.
  - The script ensures all data, including metadata, is accurately migrated.
- **Outcome**: S3 buckets now host all the data originally stored in GCP buckets.

2. Databases

- **GCP**: Google Cloud SQL (Databases)
- **AWS**: Amazon Relational Database Service (RDS)
- **Migration Method**:
  - **Scripts**:
    - `database_backup.py`: Creates a backup of GCP databases.
    - `database_restore.py`: Restores the backup into AWS RDS.
  - This approach ensures data integrity during the migration process.
- **Outcome**: RDS instances are operational, hosting all data previously in GCP SQL.

3. Persistent Volumes

- **GCP**: GKE Persistent Volumes (GKE PVCs)
- **AWS**: EKS Persistent Volumes (EKS PVCs)
- **Migration Method**:
  - **Script**: `migratepvc.sh`

        - This script handles the migration of Persistent Volume Claims (PVCs) from GKE to EKS.
    - Ensures smooth transfer of stateful data to EKS.
- **Outcome**: Stateful workloads are successfully utilizing EKS PVCs.

4. Virtual Machines

- **GCP**: Compute Engine VMs
- **AWS**: EC2 Instances
- **Migration Method**:
    - **Pulumi**: EC2 instances and their configurations are created as part of IaC.
    - Additional data and disks are manually copied to the new EC2 instances as needed.
- **Outcome**: EC2 instances are fully functional with data and configurations migrated from GCP VMs.

5. Artifact Repository

- **GCP**: Artifact Registry
- **AWS**: Elastic Container Registry (ECR)
- **Migration Method**:
    - **Pulumi**: Creates ECR repositories.
    - Container images are copied to the new ECR repositories after creation.
- **Outcome**: ECR hosts all container images, maintaining consistency with the artifact repository in GCP.

6. Serverless Applications

- **GCP**: Cloud Run
- **AWS**: Elastic Kubernetes Service (EKS)
- **Migration Method**:
    - **Helm Charts**: Used to deploy applications on EKS.
- **Outcome**: Applications previously hosted on Cloud Run are now deployed and running on EKS.

7. Disks

- **GCP**: Persistent Disks
- **AWS**: Elastic Block Store (EBS)
- **Migration Method**:
    - Direct commands are used to copy data from GCP disks to EBS volumes in AWS.
- **Outcome**: All required disk data is now hosted on EBS volumes.

8. Networking

- **GCP**: VPC (Virtual Private Cloud)
- **AWS**: VPC

- **Migration Method**:
  - **Pulumi**: VPCs are recreated in AWS with equivalent configurations.
- **Outcome**: Networking infrastructure has been replicated on AWS.

Migration Overview

The migration process leveraged the following tools and technologies:

- **Pulumi**: For automating infrastructure creation, including VPCs, EC2 instances, and ECR repositories.
- **Custom Scripts**:
  - `bucket_backup_restore.py`: For bucket migrations.
  - `database_backup.py` and `database_restore.py`: For database migrations.
  - `migratepvc.sh`: For PVC migrations.
- **Commands**: For manual disk data migration.
- **Helm Charts**: For deploying applications on EKS.

This combination of automation, scripting, and manual processes ensured data consistency, minimized downtime, and streamlined the migration effort.

Post-Migration Benefits

- Consolidation of resources into AWS for unified cloud management.
- Leveraging AWS's ecosystem for better scalability and cost optimization.
- Improved resilience and availability with AWS's robust infrastructure services.

Migration Scripts and Methods

Our migration strategy includes custom scripts and tools to ensure smooth transitions and backups between environments and cloud providers.

AWS Service	Migration Method
S3	`bucket_backup_restore.py`
RDS	`database_backup.py`, `database_restore.py`
EKS PVC	`migratepvc.sh`
EC2	Created via Pulumi, with manual data transfer as needed
ECR	Pulumi-managed, images are copied after repository creation
EKS Applications	Deployed via Helm charts

EBS (Elastic Block Store)	Data migration handled via specific commands
VPC	Managed via Pulumi

GCP to AWS Mappings

As we migrate data from GCP, here's the mapping of GCP components to AWS services and methods:

GCP Component	AWS Component	Migration Method
Buckets	S3	`bucket_backup_restore.py`
Databases	RDS	`database_backup.py`, `database_restore.py`
GKE PVCs	EKS PVCs	`migratepvc.sh`
VMs	EC2	Pulumi, with data copied manually
Artifact Repository	ECR	Pulumi, images copied post-creation
Cloud Run	EKS	Helm charts

6. Orchestration and Automation

Pulumi for Infrastructure as Code (IaC)

Pulumi is the primary tool we use for managing and deploying our infrastructure as code (IaC). By leveraging Pulumi, we ensure a consistent, version-controlled approach to provisioning resources, minimizing manual errors and improving reproducibility across environments.

Automated Pulumi Workflow with Python Script

To streamline our Pulumi operations, we have created a custom Python script named `__main__.py`. This script automates Pulumi actions such as refreshing, creating, or deleting infrastructure in our **Production** or **QA** environments, based on the specified changes.

How It Works

1. **Purpose:**
   - The script acts as a wrapper around Pulumi commands, simplifying infrastructure management.
   - It ensures that actions are executed consistently across environments.

2. **Interactive Workflow**:
    - When the script is executed, it prompts the user to specify the desired action:
        - **Refresh**: Syncs the state of resources with the actual infrastructure.
        - **Create**: Applies changes to provision new resources or update existing ones based on the Pulumi program.
        - **Delete**: Destroys specified resources as per the script's configuration.
    - The user is also prompted to choose the target environment (**Production** or **QA**).
3. **Environment Awareness**:
    - The script dynamically applies changes only to the selected environment, ensuring that **Production** and **QA** infrastructure remain independent.
    - It utilizes Pulumi's state files and configurations to maintain accurate records of deployed resources.
4. **Version Control**:
    - Any modifications to the Pulumi program or script are tracked in version control (e.g., Git), ensuring that changes are auditable and reversible.

Key Benefits

- **Consistency**: Ensures all infrastructure deployments follow the same process, reducing errors.
- **Automation**: Eliminates manual execution of Pulumi commands, saving time and effort.
- **Environment-Specific Operations**: Isolates changes to the selected environment, ensuring no unintended impacts.
- **Interactive Interface**: Simplifies the workflow with clear prompts for user input.
- **Version Control Integration**: Facilitates tracking of infrastructure changes for easy rollback or updates.

Usage Instructions

1. **Run the Script**: Execute the Python script using the following command:

```
cd kubernetes
python3 __main__.py
```

2. **Follow the Prompts**:
    - Choose the desired action (Refresh, Create, or Delete).
    - Select the environment (Production or QA).
3. **Script Execution**:
    - The script runs the corresponding Pulumi commands based on user input.
    - It interacts with the Pulumi backend to perform actions safely and efficiently.

---

7. Capacity Management and Auto-Scaling

- **Zero-Scale Auto-Scaling**: Configured in EKS to minimize costs by scaling down to zero when not in use.
- **Scaling Policies**: Defined for all environments to ensure resources are available as demand increases.

---

8. Communication and Notifications

Monitoring and Downtime Notifications Overview

Our infrastructure incorporates a robust monitoring system using the Loki Stack and an automated Slack notification setup to ensure timely communication and proactive handling of downtime or performance issues.

Downtime Notifications

We use Slack as the primary channel for real-time communication with the team regarding downtime and maintenance.

- **Slack Channel**: `#monitoring-cb-infra`
- **Purpose**:
  - To notify teams of scheduled maintenance windows or unexpected downtime.
  - To provide immediate updates during service outages or performance issues.
  - To enable swift troubleshooting and resolution by notifying relevant personnel.

Notification Features:

- **Automated Alerts**: Sent directly to the Slack channel using monitoring tools like Prometheus and Grafana.
- **Transparent Communication**: Ensures that all stakeholders are informed of infrastructure status changes to minimize disruptions.
- **Customizable Messages**: Alerts include details such as the affected services, estimated resolution times, and recommended actions.

Loki Stack for Monitoring and Alerting

The **Loki Stack** (comprising **Prometheus**, **Grafana**, **Loki**, and **Promtail**) is our primary monitoring and alerting system. It provides comprehensive visibility into the state of our Kubernetes clusters, services, and applications.

Components:

1. **Prometheus**:
   - Collects metrics from Kubernetes clusters and external services.

- Configured with alerting rules to detect anomalies or failures.
- Sends alert triggers to connected notification systems like Slack.

2. **Grafana**:
   - Provides rich visualizations for monitoring data.
   - Displays custom dashboards with detailed metrics and insights.
   - Used for real-time monitoring of Kubernetes resources and application performance.
3. **Loki**:
   - Centralized log aggregation for all Kubernetes services, pods, and system components.
   - Enables log-based alerting and troubleshooting.
4. **Promtail**:
   - Collects logs from Kubernetes pods and forwards them to Loki for processing and storage.

Monitoring Features:

- **Custom Dashboards**:
  - Track cluster-level and namespace-level metrics.
  - Monitor pods, services, and workloads.
  - Provide visualizations of CPU, memory, disk utilization, and network activity.
  - Display detailed logs, error rates, and system health.
  - Custom dashboards cater to specific needs like application performance and Kubernetes resource utilization.
- **Log and Metric Correlation**:
  - Logs and metrics are integrated into a single system for easier debugging and root cause analysis.
- **Alert Rules**:
  - Prometheus alerting rules are configured for critical scenarios such as:
    - High resource usage (CPU, memory, disk).
    - Pod restarts or crashes.
    - Service unavailability or performance degradation.
  - Alerts are pushed to Slack for immediate visibility.

Workflow for Monitoring and Notifications

1. **Data Collection**:
   - Promtail collects logs from Kubernetes nodes and sends them to Loki.
   - Prometheus scrapes metrics from Kubernetes components and applications.
2. **Data Visualization**:
   - Grafana dashboards provide detailed insights into the health and performance of the infrastructure.
   - Logs and metrics are displayed side-by-side for quick troubleshooting.

3. **Alert Generation**:
    - Prometheus evaluates configured alerting rules.
    - When a condition is met, Prometheus triggers an alert.
4. **Notification Delivery**:
    - Alerts are sent to the Slack channel `#monitoring-cb-infra`.
    - Messages include details about the issue, severity, and recommended next steps.

---

9. Version Control Management

Our infrastructure includes **Gitea** as our internal version control system and **Argo CD** for deploying containerized applications from Amazon Elastic Container Registry (ECR) using Helm charts. This setup ensures efficient management of code changes and automated deployment of applications, aligning with best practices for DevOps.

---

1. Gitea Instance

Gitea is an open-source, self-hosted version control system providing Git services similar to GitHub, but hosted on our internal infrastructure. Gitea serves as the foundation for managing source code, enabling collaborative development, and preserving historical data.

- **Purpose**: The Gitea instance is used to manage all codebases and repositories for our projects. It facilitates a controlled environment for versioning, code reviews, and branching strategies.
- **Key Benefits**:
    - **Data Integrity**: Maintains a reliable and consistent history of code changes.
    - **Collaboration**: Enables team members to work collaboratively on code, track issues, and manage project progress.
    - **Internal Control**: Ensures that code remains within our secured environment, providing better control over access and permissions.

This setup helps developers maintain version control across projects, track changes, and ensure all code is up-to-date, which is especially critical for deployment processes.

---

2. Argo CD

Argo CD is a declarative, GitOps-based continuous delivery tool specifically designed for Kubernetes applications. By linking Argo CD to our Gitea repositories, we can streamline the deployment process directly from our Git-managed infrastructure.

- **Functionality**:
    - **Source Connection**: Argo CD connects to Gitea repositories, monitors changes in specified branches, and pulls the latest configurations or code updates.

  - **Application Deployment**: Deploys applications to Kubernetes clusters by pulling container images from ECR and using Helm charts for templated, scalable deployment.
  - **GitOps Workflow**: Automatically synchronizes desired state as defined in Git (our Gitea instance) with actual deployed state in Kubernetes, ensuring consistency.
- **Deployment Flow**:
  1. **Code Update**: When a new commit is pushed to the Gitea repository, containing updates or fixes.
  2. **Image Update in ECR**: The updated code is built, and the latest image is pushed to the Elastic Container Registry.
  3. **Argo CD Trigger**: Argo CD detects the updated image tag or configuration and initiates a deployment using Helm charts.
  4. **Application Rollout**: Deploys or updates the application on the Kubernetes cluster, making the latest version available.
- **Benefits of Argo CD**:
  - **Automated Deployments**: Reduces manual deployment processes by automating container rollout.
  - **Declarative Configuration**: Uses Kubernetes-native manifest files or Helm charts, making configuration clear and consistent.
  - **Self-Healing**: Automatically reverts applications to their desired state if configurations drift, adding stability and reliability.

Detailed Workflow and Integration

Gitea and Argo CD Integration:

1. **Repository Management in Gitea**:
   - Our Gitea instance holds the Helm chart configurations and application code. Each repository corresponds to a specific project or application, providing version control, access control, and historical tracking.
2. **Continuous Delivery with Argo CD**:
   - Argo CD monitors the Gitea repository for any changes to the configurations or application code.
   - When changes are detected, Argo CD automatically pulls the latest ECR image of the application and deploys it onto our Kubernetes cluster using the Helm chart defined in the repository.
3. **Helm Charts for Deployment**:
   - Each application is defined by a Helm chart stored within the Gitea repository.
   - Helm charts simplify the deployment by parameterizing configurations (like image tags, resource limits, and service definitions), making deployments consistent and reproducible.

---

10. Appendix

Common Commands and Scripts

- **S3 Backup and Restore:** `bucket_backup_restore.py`
- **Database Backup and Restore:** `database_backup.py`, `database_restore.py`
- **PVC Migration:** `migratepvc.sh`
- **VM Management**: Pulumi scripts and manual data copy commands.

Glossary of Terms

- **IaC (Infrastructure as Code)**: Managing and provisioning computing infrastructure through machine-readable configuration files.
- **EKS**: Managed Kubernetes service provided by AWS.
- **Pulumi**: Tool for IaC using familiar programming languages.
