Problem Statements

1. Modernizing Legacy File-Based Deployments

Legacy Tomcat deployments rely on manual retrieval of WAR files from static locations, leading to inconsistent versions and "all-at-once" server updates. We need to automate the secure pull of artifacts from Object Stores/GCS, implement mandatory backup procedures, and enforce a gated 1-to-N server rollout to ensure stability across VM clusters.

2. Standardizing API and Container Orchestration

The lack of a unified deployment engine makes it difficult to maintain parity between Linux-based Docker runs and GKE cluster deployments. Our goal is to establish a single pipeline capable of executing sophisticated Canary and Blue-Green traffic shifts across both infrastructure types to eliminate deployment-related downtime.

3. Automated Database Lifecycle Management

Manual database schema updates and connection management for CloudSQL Postgres instances create bottlenecks and high security risks. We must integrate automated database provisioning and lifecycle management into the CI/CD flow to ensure that application logic and database states are always synchronized and version-controlled.

Success Criteria

1. Modernizing Legacy File-Based Deployments

Success Criteria	Target	Measure
All WAR and Docker artifacts should be versioned and stored in GCS with Github Actions **run_id-based path structure**	100% of builds should be tracked in GCS with immutable artifact URLs	Audit trail of all deployed versions should be available with timestamp and commit hash .
Need to Automate artifact download from GCS before deployment executes without manual intervention	Zero manual artifact retrieval steps in deployment workflow	Deployment job logs should confirm automated **gsutil cp** operations complete successfully
Create Version mismatch prevention through **image_tag** input parameter	All deployments need to be tagged with explicit version (**default: 1.0.0**)	Application response should include **APP_VERSION** environment variable matching deployment tag.

Create Rollback capability via **stable_run_id** parameter	Any deployment should be rolled back to previous stable version within **5 minutes**	Canary rollback and full rollback strategies should complete successfully in < 5 minutes
Pre-deployment backups of current Tomcat application state	Need to take automatic backup of **/opt/tomcat9/webapps/ROOT** before any new deployments .	Backup artifacts should be created and verified before deployment proceeds
Backup retention policy enforcement for Deployment Logs	**Retain last 10 successful** deployments; purge older backups after 30 days	Backup management script logs archive/purge operations
Canary strategy deploys to single server first with **10% traffic weight**	When **deployment_strategy = "canary"**, first server should receives **weight=10**	Application logs need to show 10% of requests routing to canary instance
Remaining servers weighted equally for **90% traffic distribution**	Servers 2 and 3 each should receive **weight=45** during canary deployment	Traffic distribution verified via access logs (90% split between stable servers)
Canary deployment **completes before stable servers** receive updates	Canary server **needs to be fully deployed and healthy** before any stable server deployment	Deployment logs show canary server deployment **timestamp < all other server timestamps**
Operator can **abort canary and rollback all servers** within decision window	Canary rollback decisions can be made to be enabled for the first 30 minutes post-deployment.	**canary_rollback** strategy execution should brings canary to "**down**" state with **weight=0 in < 5 minutes**
Zero downtime during phased rollout across all servers	Service availability should be maintained at **99.9% during deployment**	Load balancer health checks need to show no server marked as **unhealthy during update.**
Automatic rollback if any server deployment fails	**Failed server deployment should trigger** cluster-wide rollback to previous stable version	All servers should return to the previous version **within 10 minutes of failure detection .**

2. Standardizing API and Container Orchestration

Success Criteria	Target	Measure
Single workflow executes both Linux server and GKE deployments with strategy parity	Github workflow Pipeline should need to deploy between target choice **linux-servers, gke-cluster**	Both deployment paths should need to use identical deployment strategy options **(normal_deployment, canary, blue_green,**

		canary_rollback, full_rollback)
Docker image **built once**, deployed to both Docker hosts and GKE	A **single docker build step** should produce artifact consumed by both Linux Servers and Kubernetes environments.	Github Actions **jobs to build docker images** and to be consumed to deploy in both Linux Servers and K8s Clusters
Artifact registry provides **centralized image storage** for both targets	Docker **images tagged and pushed** to artifact-registry	Unique Image Tags will be utilized in the Deployment Stage for **reducing redundant deployments of same tags .**
Canary instances are isolated and can be monitored independently	**Linux**: canary server should be marked separately in NGINX; **GKE**: canary pods should need to be isolated via labels .	Logs and metrics can be filtered by **canary vs. stable instances for independent** analysis .
Instant traffic cutover from **blue to green (or vice versa)** with near-zero downtime	Traffic should be switched **within 5 seconds of operator approval**	Service selector **should be updated and verified** with kubectl; no request timeouts in logs

3. Automated Database Lifecycle Management

Success Criteria	Target	Measure
Deployment pipeline **provisions database connections** automatically	CloudSQL Postgres connection string should be **injected via deployment job**	Environment variable **CLOUDSQL_CONNECTION_NAME** available in both Linux containers and GKE pods
Database credentials managed securely via **GitHub Secrets**	No credentials **should be hardcoded in workflow** and all secrets should be rotated quarterly	Secret audit logs should show no plaintext credential exposure
Connection pool validation before traffic shift	Pre-deployment health checks should confirm **successful connection to CloudSQL**	**Health check SQL query** needs to be executed successfully; connection pool metrics logged
Automated backup of database before each deployment	CloudSQL on-demand backup should be triggered **before deployment begins**	Backup completion **confirmed in logs** before application deployment proceeds

---

POC :

Linux Server Deployment – https://ucd-demo.bootlabstech.com/
GKE Cluster Deployment – https://ucd-demo-gke.bootlabstech.com/

Github Actions Workflow