SDAIA
الهيئة السعودية للبيانات والذكاء الاصطناعي
Saudi Data & AI Authority

PROJECT: End-to-End GitOps Pipeline Implementation

Automating Kubernetes Deployments with GitLab CI and ArgoCD

Talal Muhaysin Alharthi

Role: DevOps Engineer
Project Type: Infrastructure as Code (IaC) & Continuous Delivery

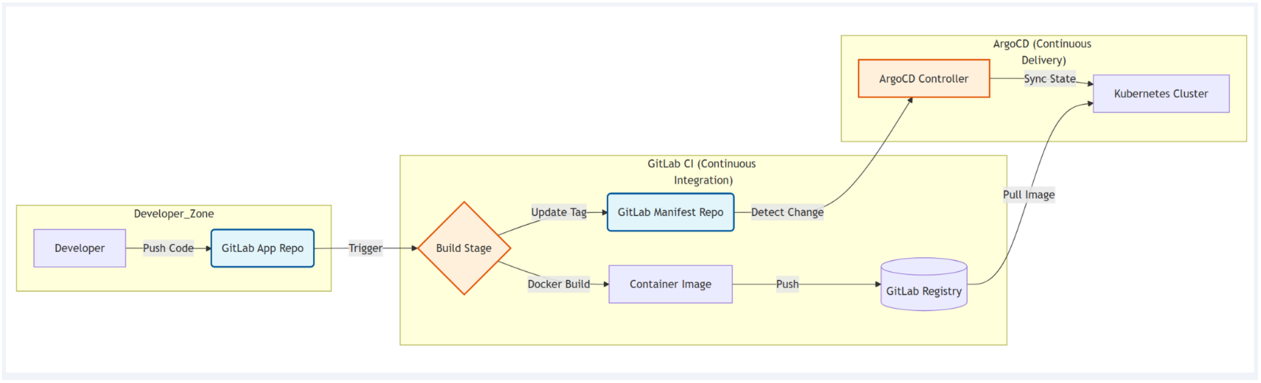
1. Executive Summary

This project demonstrates the engineering of a robust, "End-to-End" GitOps pipeline designed to automate the lifecycle of a cloud-native application. The primary objective was to eliminate imperative deployment operations (manual `kubectl` commands) in favor of a declarative, audit-compliant workflow.

The system integrates **GitLab CI** for Continuous Integration (artifact generation) and **ArgoCD** for Continuous Deployment (state reconciliation). This architecture ensures that the Git repository remains the "Single Source of Truth" for both application code and infrastructure state, adhering to industry-standard DevOps practices.

2. System Architecture & Workflow Graph

The following directed graph illustrates the data flow designed for this project, moving from local development to production deployment.



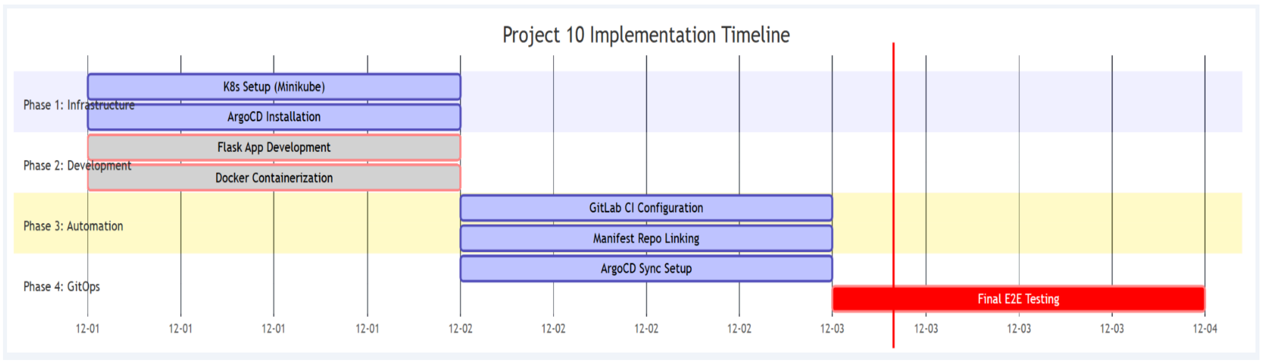
Workflow Description:

1. **Code Commit:** Code is pushed to the Application Repository.
2. **Artifact Generation:** GitLab CI builds a Docker image tagged with the specific Commit SHA to ensure traceability.
3. **Manifest Injection:** The CI pipeline bridges the gap by automatically cloning the Manifest Repository and updating the deployment YAML using sed.
4. **Reconciliation:** ArgoCD detects the configuration drift and synchronizes the cluster to the new state.

3 Implementation

3.1 Implementation Schedule

The project was executed over a structured timeline to ensuring component stability before integration.



3.2 Technical Implementation Details

3.2.1 Application Layer

The target application is a Python-based microservice using the Flask framework.

- Version Control: The application reads an environment variable APP_VERSION to display deployment metadata, verifying successful configuration injection.
- Containerization: A multi-stage Dockerfile is utilized based on python:3.11-slim to minimize the attack surface and image size. Dependencies are installed via requirements.txt.

3.2.2 Continuous Integration (GitLab CI)

The CI pipeline is defined in .gitlab-ci.yml and consists of two primary stages: build and deploy.

- Build Stage (build_image):
  - Utilizes the Docker-in-Docker (dind) service to build the image context.
  - Immutability Strategy: Images are tagged with the specific Git Commit SHA ($CI_COMMIT_SHORT_SHA) rather than just latest. This ensures that every deployment is traceable to a specific point in the codebase.
  - Artifact Storage: Images are authenticated and pushed to the GitLab Container Registry.
- Deploy Stage (update_manifests):
  - This stage acts as the automated bridge between Code and Infrastructure.
  - It executes a sed transformation command to inject the new image hash into the manifest.yaml file: sed -i "s|image:.*|image: $NEW_IMAGE|g" manifest.yaml.
  - Loop Prevention: The commit message includes [skip ci] to prevent the manifest update from triggering a recursive pipeline loop.

3.2.3 Infrastructure & GitOps (ArgoCD)

The infrastructure is managed declaratively via Kubernetes manifests.

- Deployment Controller: Configured with replicas: 2 for high availability. It utilizes imagePullSecrets to authenticate with the private GitLab registry.
- State Reconciliation: ArgoCD is configured to monitor the gitops-manifests repository. It automatically synchronizes the cluster state when the CI pipeline pushes updates to the manifest file.

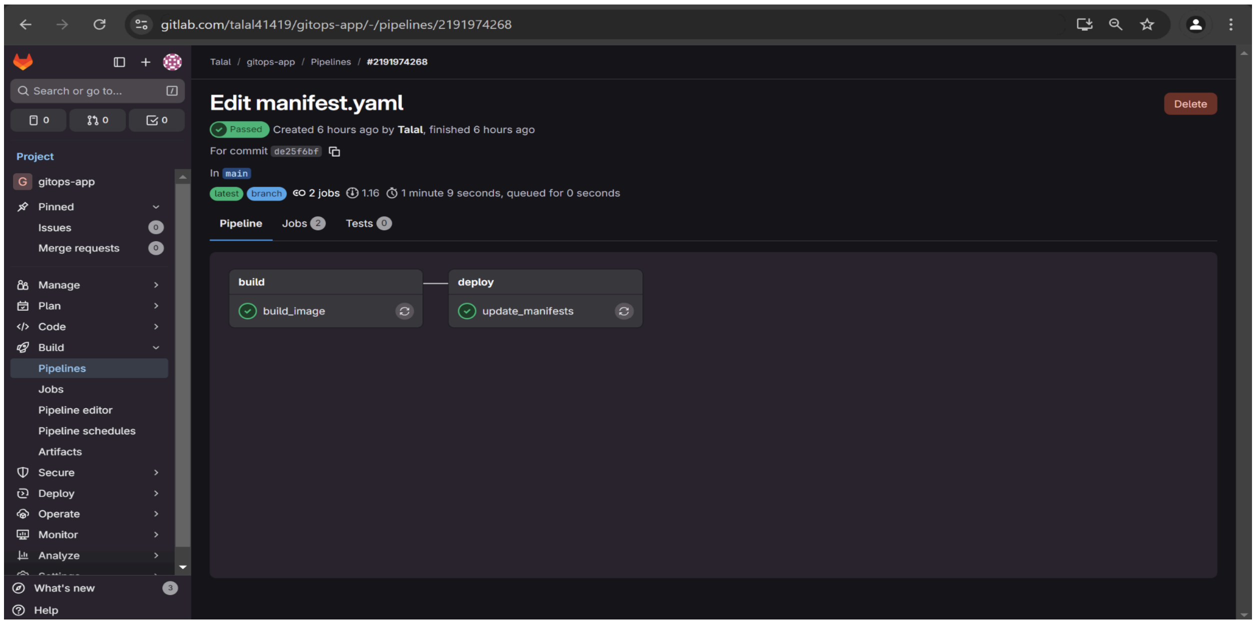
4. Technical Validation & Evidence

The following section validates the successful execution of the project using system screenshots and logs.

4.1 Continuous Integration (GitLab CI)

The pipeline successfully executed the build_image and update_manifests stages.

- **Pipeline ID:** #2191974268
- **Result:** The pipeline passed in approximately 1 minute, proving high-velocity feedback.


gitlab.com/talal41419/gitops-app/-/pipelines/2191974268

Talal / gitops-app / Pipelines / #2191974268

Edit manifest.yaml

Passed Created 6 hours ago by **Talal**, finished 6 hours ago

For commit `de25f6bf`

In `main`

latest branch 2 jobs 1.16 1 minute 9 seconds, queued for 0 seconds

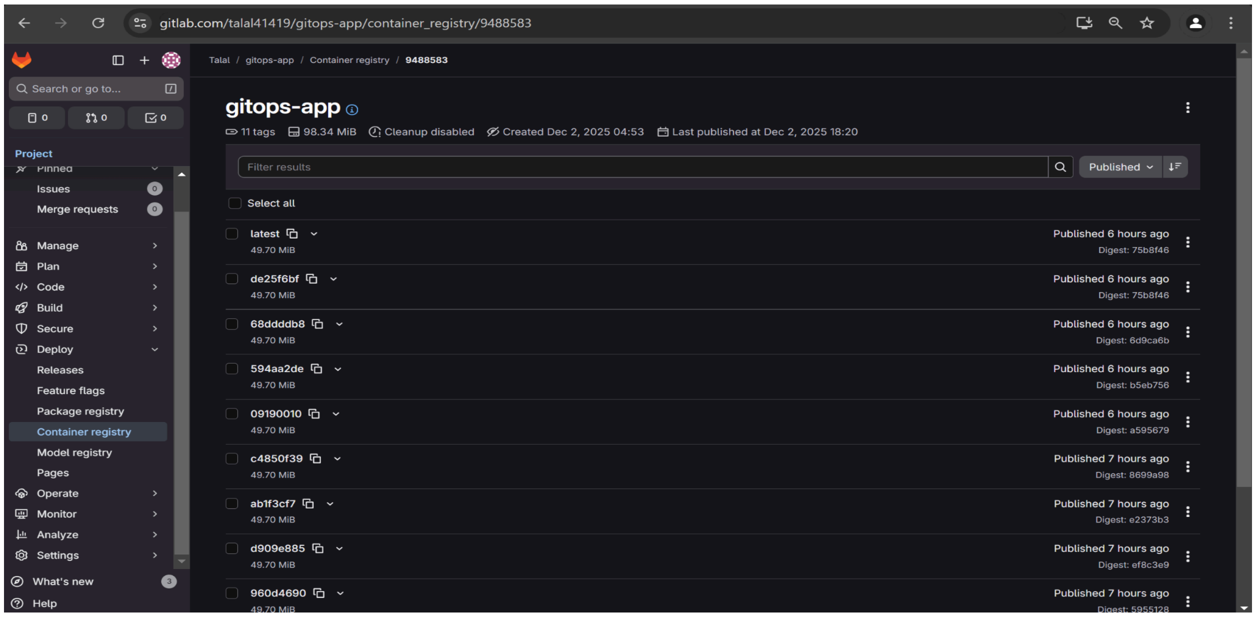
build: build_image

deploy: update_manifests

Figure 1: Pipeline Success

4.2 Artifact Management (Registry)

The immutability strategy was verified. Images are tagged with the short SHA (e.g., de25f6bf) rather than mutable tags like latest. This allows for precise rollbacks to any historical version.


gitlab.com/talal41419/gitops-app/container_registry/9488583

Talal / gitops-app / Container registry / 9488583

gitops-app

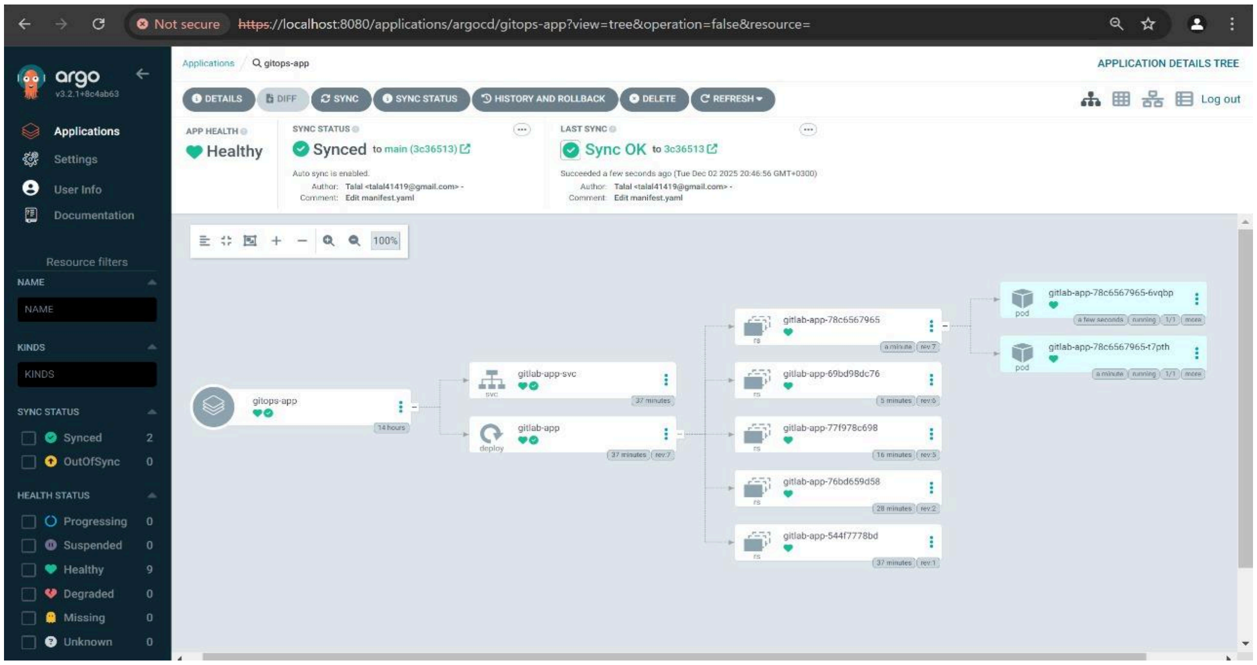
11 tags · 98.34 MiB · Cleanup disabled · Created Dec 2, 2025 04:53 · Last published at Dec 2, 2025 18:20

Tag	Size	Published	Digest
latest	49.70 MiB	Published 6 hours ago	Digest: 75b8f46
de25f6bf	49.70 MiB	Published 6 hours ago	Digest: 75b8f46
68ddddb8	49.70 MiB	Published 6 hours ago	Digest: 6d9ca6b
594aa2de	49.70 MiB	Published 6 hours ago	Digest: b5eb756
09190010	49.70 MiB	Published 6 hours ago	Digest: a595679
c4850f39	49.70 MiB	Published 7 hours ago	Digest: 8699a98
ab1f3cf7	49.70 MiB	Published 7 hours ago	Digest: e2373b3
d909e885	49.70 MiB	Published 7 hours ago	Digest: ef8c3e9
960d4690	49.70 MiB	Published 7 hours ago	Digest: 5955128

Figure 2: Container Registry

4.3 GitOps Synchronization (ArgoCD)

ArgoCD is confirmed to be in a **Healthy** and **Synced** state. The application topology shows the controller managing the replica sets and pods correctly. The "Last Sync" timestamp confirms automated triggering.


Not secure ~~https~~://localhost:8080/applications/argocd/gitops-app?view=tree&operation=false&resource=

APP HEALTH: Healthy

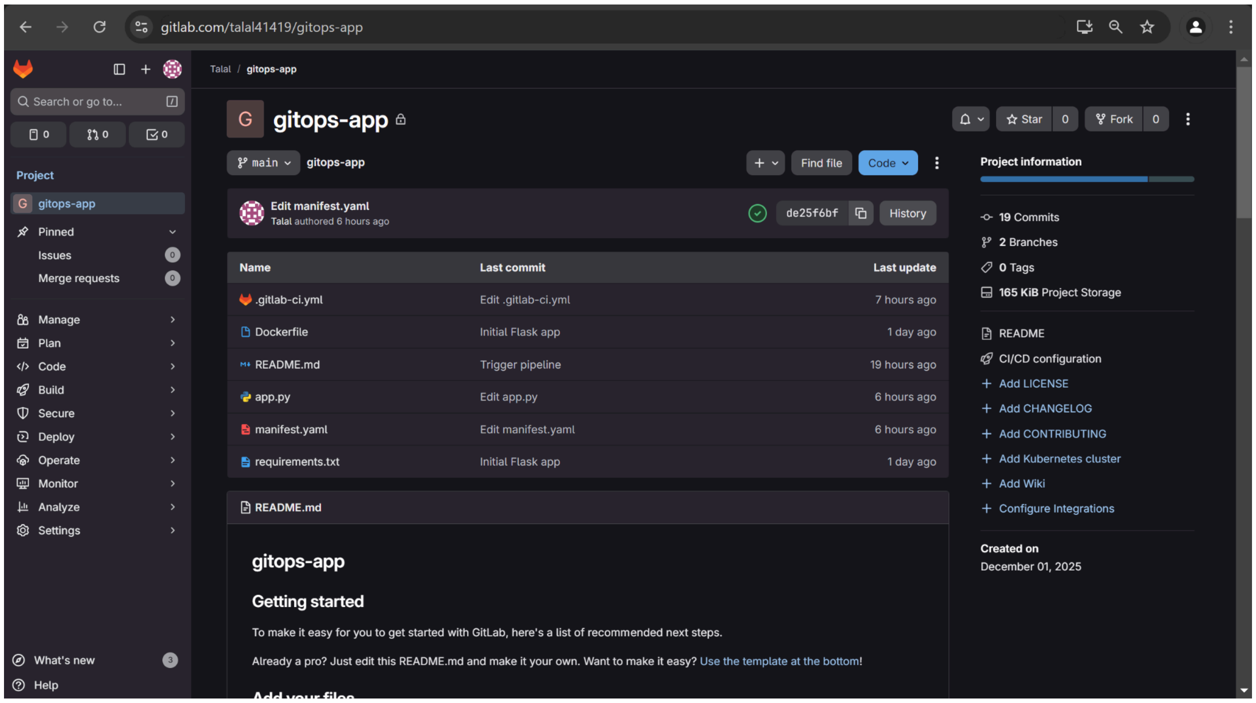
SYNC STATUS: Synced to main (3c36513)
Auto sync is enabled.
Author: Talal <talal41419@gmail.com> -
Comment: Edit manifest.yaml

LAST SYNC: Sync OK to 3c36513
Succeeded a few seconds ago (Tue Dec 02 2025 20:46:56 GMT+0300)
Author: Talal <talal41419@gmail.com> -
Comment: Edit manifest.yaml

Figure 3: ArgoCD Dashboard

4.4 Automated Version Control

The "Bridge" mechanism works as designed. The Git history of the gitops-manifests repository shows the automatic commit by the "GitLab CI" user, updating the manifest to match the build artifact.



gitops-app

Edit manifest.yaml
Talal authored 6 hours ago
de25f6bf

Name	Last commit	Last update
.gitlab-ci.yml	Edit .gitlab-ci.yml	7 hours ago
Dockerfile	Initial Flask app	1 day ago
README.md	Trigger pipeline	19 hours ago
app.py	Edit app.py	6 hours ago
manifest.yaml	Edit manifest.yaml	6 hours ago
requirements.txt	Initial Flask app	1 day ago

Project information
19 Commits
2 Branches
0 Tags
165 KiB Project Storage
README
CI/CD configuration
Add LICENSE
Add CHANGELOG
Add CONTRIBUTING
Add Kubernetes cluster
Add Wiki
Configure Integrations
Created on
December 01, 2025

README.md

gitops-app

Getting started

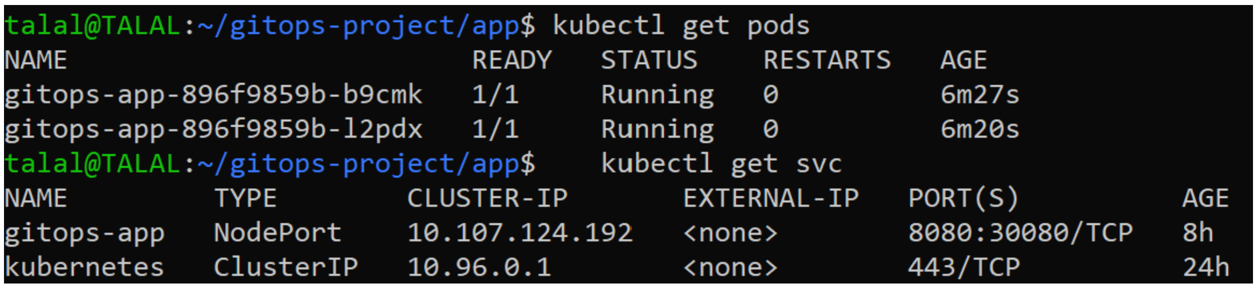
To make it easy for you to get started with GitLab, here's a list of recommended next steps.

Already a pro? Just edit this README.md and make it your own. Want to make it easy? Use the template at the bottom!

Figure 4: Automated Commit History

4.5 Kubernetes Pod Verification

Terminal verification confirms that the pods were scheduled and are in a Running state. The service gitops-app was successfully allocated a ClusterIP and NodePort.



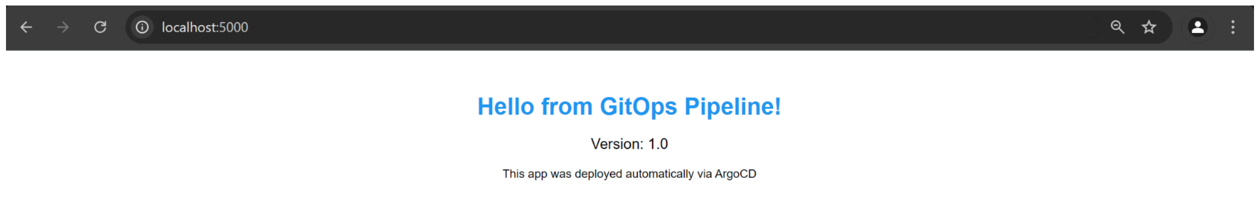
```
talal@TALAL:~/gitops-project/app$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
gitops-app-896f9859b-b9cmk    1/1     Running   0          6m27s
gitops-app-896f9859b-l2pdx    1/1     Running   0          6m20s
talal@TALAL:~/gitops-project/app$   kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
gitops-app   NodePort    10.107.124.192   <none>        8080:30080/TCP   8h
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP          24h
```

Figure 5: CLI Verification

4.6 Application Availability

The application is accessible via the browser at localhost:5000 (forwarded). The UI correctly renders the HTML content, confirming the Python code is executing correctly within the container.

Browser displaying "Hello from GitOps Pipeline!"



localhost:5000

Hello from GitOps Pipeline!

Version: 1.0

This app was deployed automatically via ArgoCD

Figure 6: Live Application

5. Engineering Challenges & Solutions

Challenge	Solution Implemented	Source Reference
Private Registry Auth	Implemented imagePullSecrets in Kubernetes manifest to authenticate with GitLab Registry.	manifest.yaml line 16
CI Infinite Loops	Added [skip ci] to the commit message in the manifest update script to prevent recursive pipeline triggering.	.gitlab-ci.yml line 44
Dynamic Versioning	Utilized sed stream editor to dynamically inject the $CI_COMMIT_SHORT_SHA into the deployment YAML.	.gitlab-ci.yml line 42

6. Conclusion

This project successfully establishes a "Zero-Touch" deployment pipeline. By leveraging the power of GitLab CI for integration and ArgoCD for delivery, the system achieves:

- **High Velocity:** Code moves from commit to production in minutes.
- **High Stability:** Automatic drift detection and self-healing via ArgoCD.
- **Security:** Separation of access concerns and immutable artifact tracking.

This architecture represents a production-ready baseline for scaling microservices on Kubernetes.