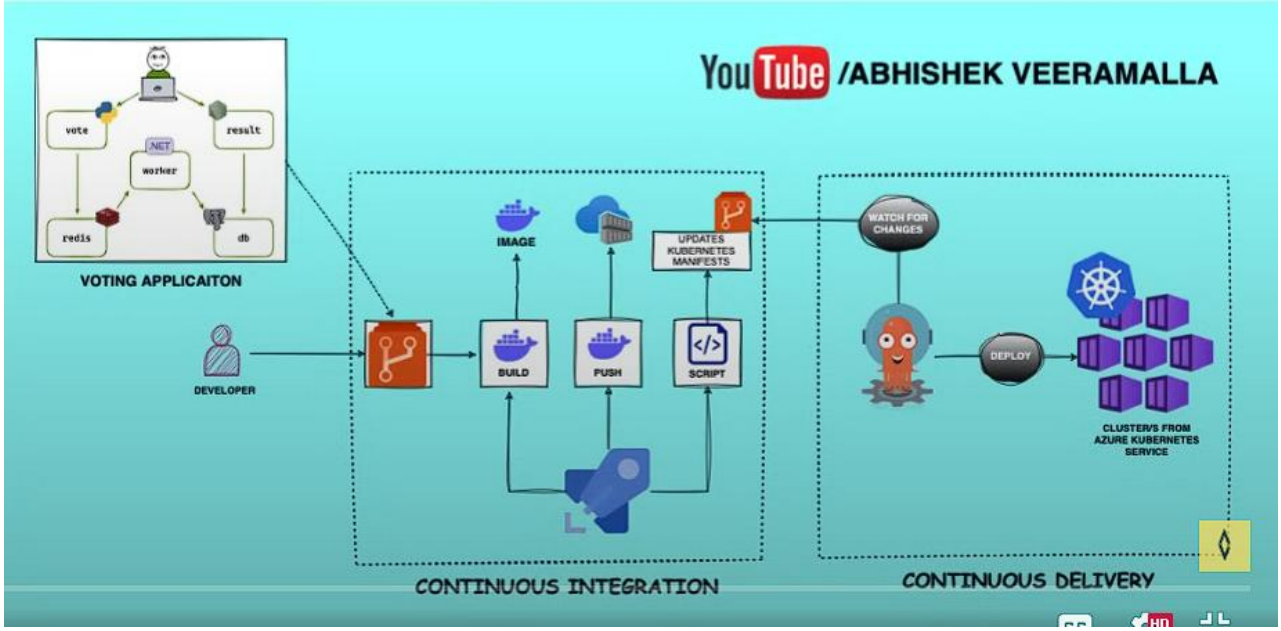
Week-15 | Ultimate Azure CI/CD Pipeline using Azure Pipelines & Argo CD | Part II | Multi Microservice Project



Index

---

1. Introduction

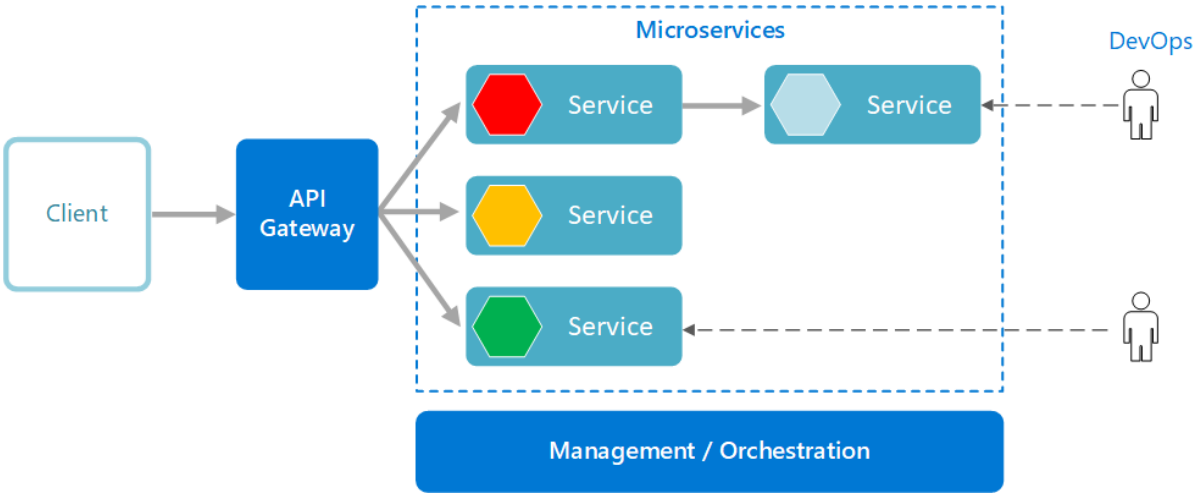
Modern applications are built using a **microservices architecture**, which enables modularity, scalability, and faster development cycles. However, managing a **multi-microservice** application comes with significant challenges in **deployment, monitoring, and automation**. To address these challenges, organizations leverage **Azure Pipelines** for CI/CD and **Argo CD** for continuous deployment.

This document serves as a detailed guide for setting up a **fully automated CI/CD pipeline** that enables the seamless deployment of microservices to **Azure Kubernetes Service (AKS)**. The goal is to ensure **automated builds, testing, containerization, and deployments** while following GitOps principles with Argo CD.

---

2. Understanding the Multi-Microservice CI/CD Pipeline

2.1 Overview of Multi-Microservice Architecture



A **multi-microservice** project consists of multiple independent services that communicate via APIs. Each microservice is built, tested, and deployed independently, allowing faster feature releases and scalability.

2.2 Key Components of the CI/CD Pipeline

- **Version Control System (GitHub or Azure Repos)** – Manages the source code and infrastructure configurations.
- **Azure Pipelines** – Automates code integration, testing, and containerization for each microservice.
- **Container Registry (Azure Container Registry or Docker Hub)** – Stores container images securely.
- **Kubernetes Cluster (Azure Kubernetes Service - AKS)** – Orchestrates microservices, ensuring load balancing and scalability.
- **Argo CD** – Synchronizes Kubernetes deployments with Git repositories using GitOps principles.
- **Monitoring & Logging Tools (Azure Monitor, Prometheus, and Grafana)** – Provides visibility into application health and logs.

---

3. Setting Up Azure Pipelines for CI/CD in a Multi-Microservice Project

3.1 Source Code Management and Version Control

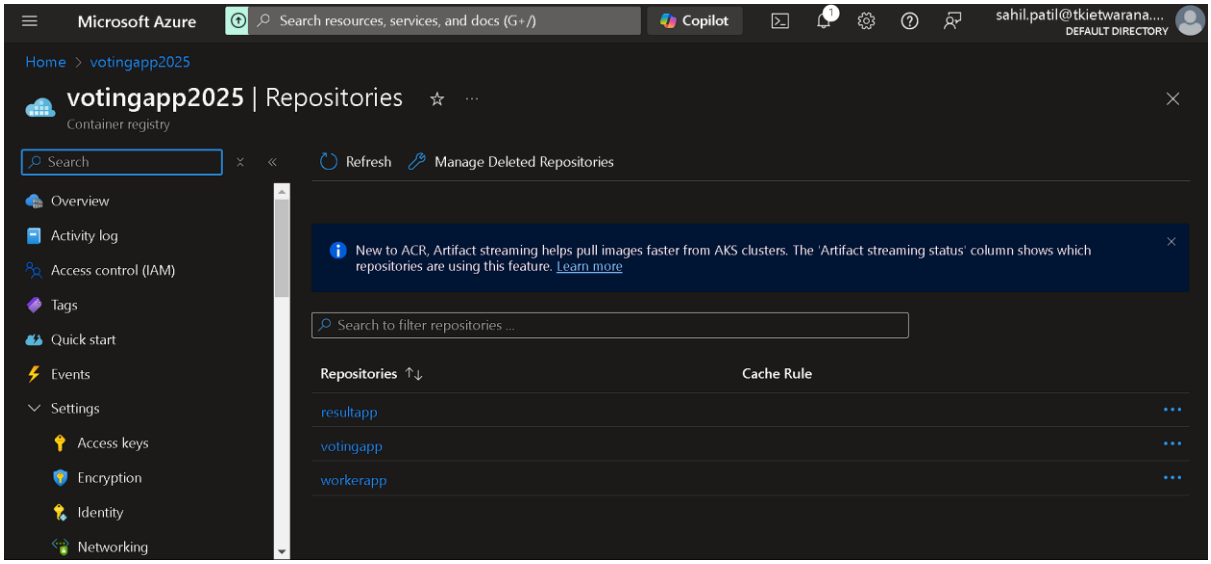
- The repository follows a structured format with separate directories for each microservice.
- Git branches are used for **feature development, staging, and production deployments**.
- Code changes trigger automatic builds through **Azure Pipelines**.

3.2 Build and Test Phase

- The pipeline validates code changes by compiling and running tests.
- Unit tests and integration tests are executed to catch issues early.

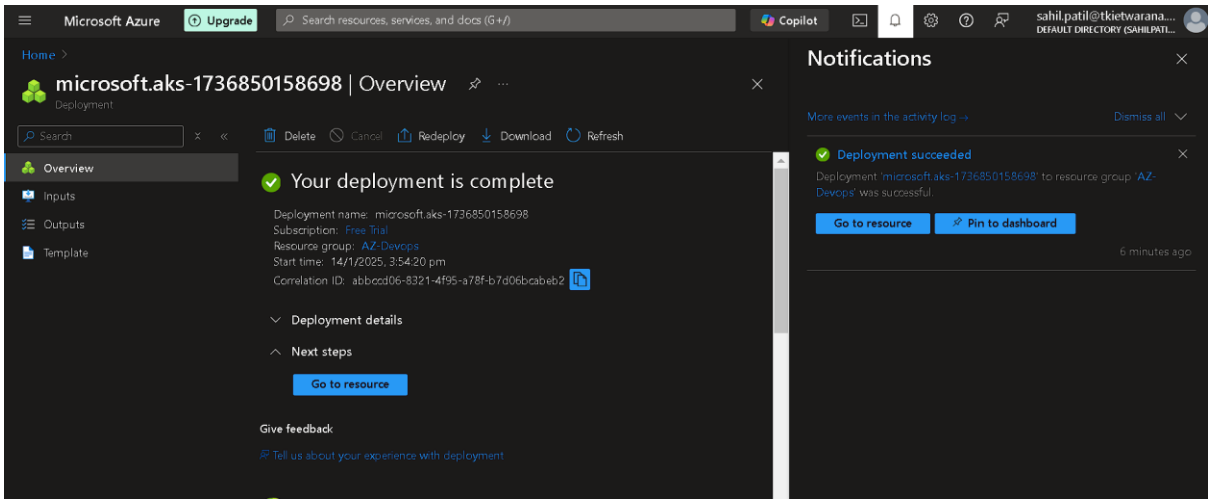
- If tests fail, the pipeline stops to prevent faulty deployments.

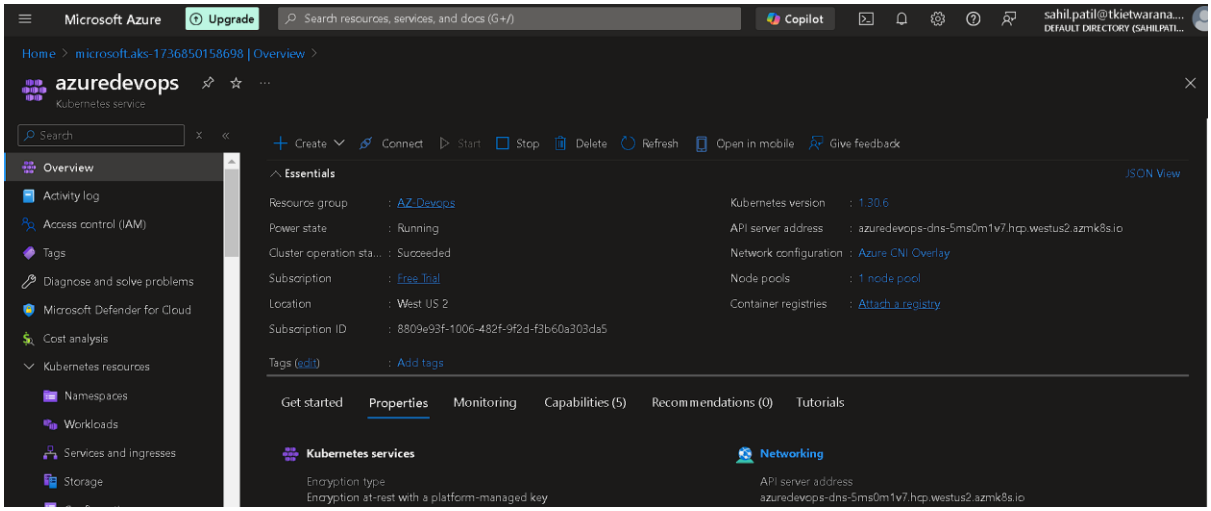
3.3 Containerization and Image Management



- Each microservice is packaged into a **Docker container** to ensure portability.
- Container images are tagged with unique versions and pushed to **Azure Container Registry (ACR)**.
- Version control helps rollback deployments in case of failures.

3.4 Deployment to Kubernetes Cluster



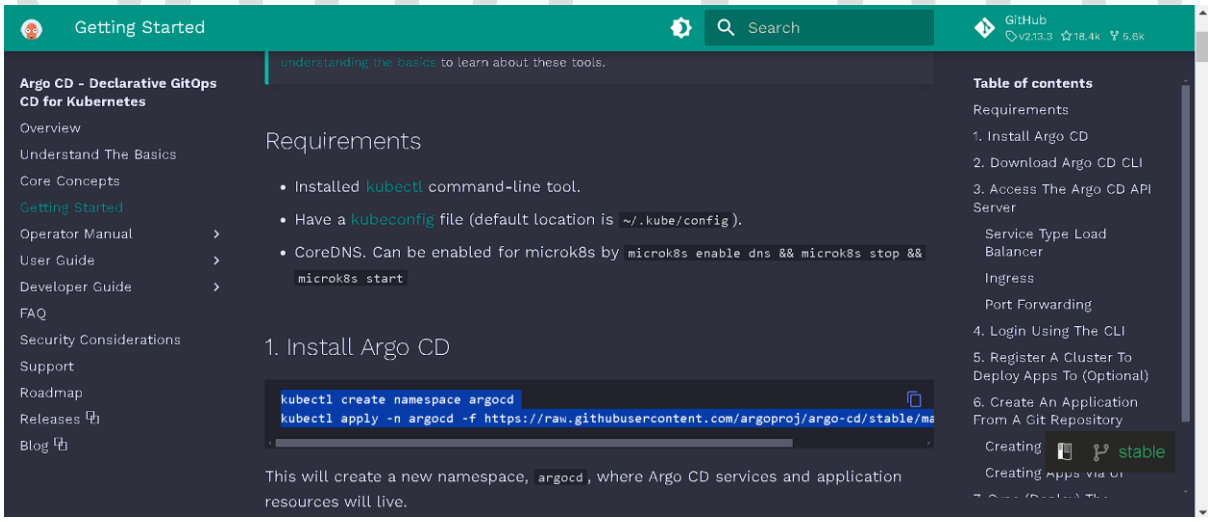


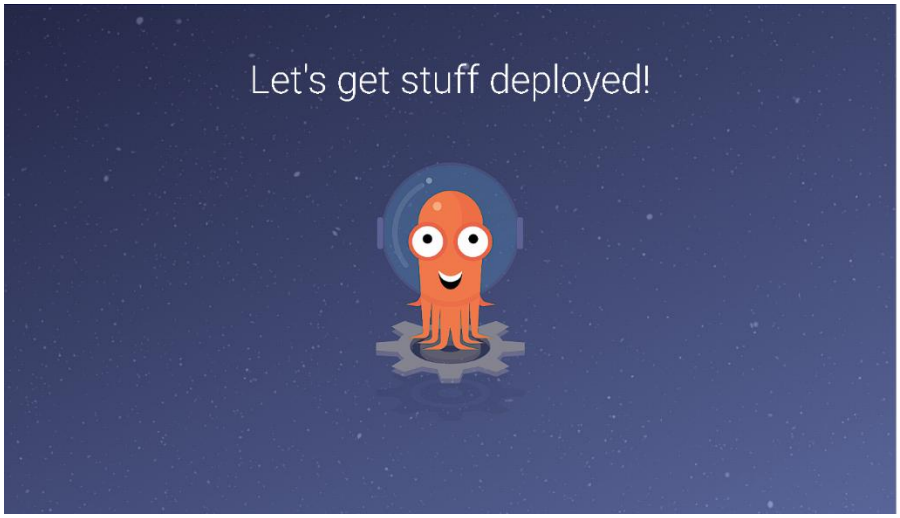
- Azure Pipelines deploys microservices to an **AKS cluster**.
- **Kubernetes manifests (YAML files)** define deployments, services, and ingress configurations.
- **Rolling updates and zero-downtime deployments** ensure smooth releases.

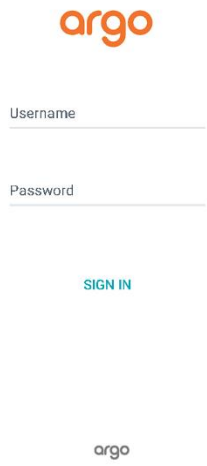
---

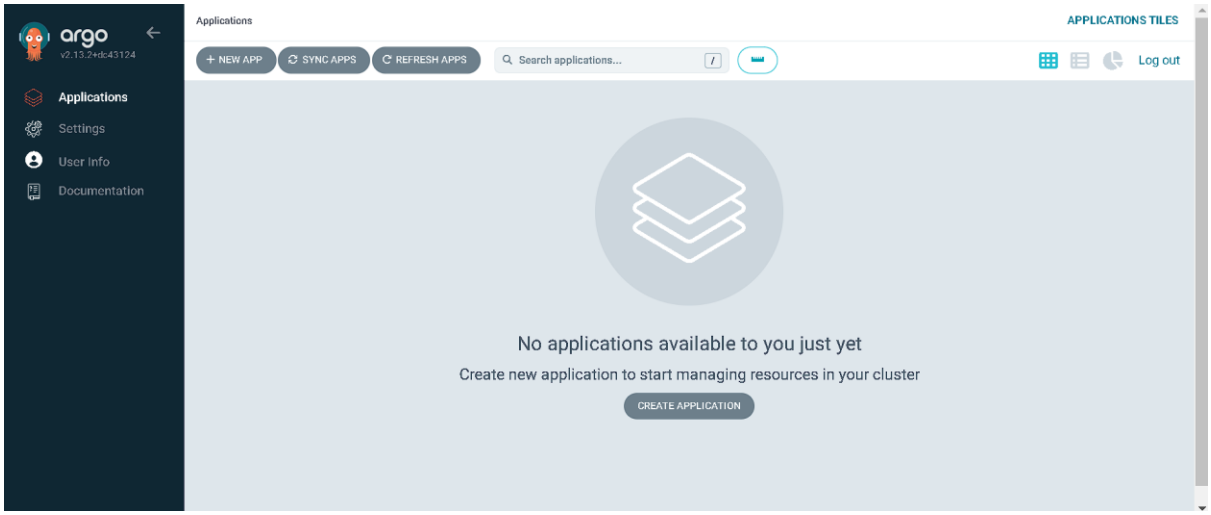
4. Integrating Argo CD for Continuous Deployment

4.1 Introduction to Argo CD









Argo CD is a **GitOps-based** continuous deployment tool that ensures **Kubernetes clusters** remain synchronized with a **Git repository**.

4.2 Installation and Configuration

- Argo CD is installed in a dedicated **namespace** within the **AKS cluster**.
- Secure authentication is configured using **RBAC (Role-Based Access Control)**.
- The **Argo CD UI** allows monitoring and manual deployment management.

4.3 Creating Argo CD Applications

- Each microservice has a dedicated Argo CD application.
- Configuration files define deployment strategies for each microservice.
- Argo CD continuously monitors changes in Git and applies updates automatically.

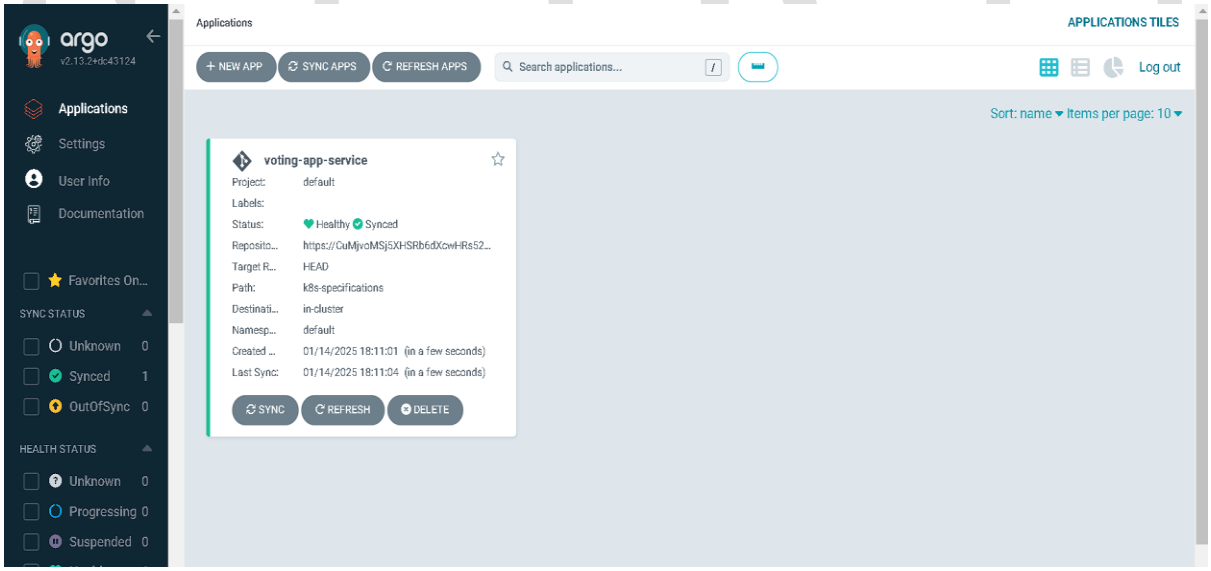
4.4 GitOps Workflow in Argo CD

- Developers push changes to Git, triggering a new deployment cycle.
- Azure Pipelines **builds, tests, and containerizes** the microservices.
- Argo CD detects changes and synchronizes the **AKS cluster** automatically.

```
sahil@SAHI-SNEH MINGW64 ~
$ kubectl get deployment -n argocd
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
argocd-applicationset-controller    1/1     1            1           100s
argocd-dex-server                   1/1     1            1           100s
argocd-notifications-controller     1/1     1            1           99s
argocd-redis                        1/1     1            1           98s
argocd-repo-server                  1/1     1            1           98s
argocd-server                       1/1     1            1           97s

sahil@SAHI-SNEH MINGW64 ~
$ kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP   10.0.188.177   <none>        7000/TCP,8080/TCP            115s
argocd-dex-server                         ClusterIP   10.0.200.174   <none>        5556/TCP,5557/TCP,5558/TCP   114s
argocd-metrics                            ClusterIP   10.0.243.65    <none>        8082/TCP                     113s
argocd-notifications-controller-metrics   ClusterIP   10.0.35.221    <none>        9001/TCP                     113s
argocd-redis                              ClusterIP   10.0.179.167   <none>        6379/TCP                     112s
argocd-repo-server                        ClusterIP   10.0.252.60    <none>        8081/TCP,8084/TCP            111s
argocd-server                             ClusterIP   10.0.203.235   <none>        80/TCP,443/TCP               111s
argocd-server-metrics                     ClusterIP   10.0.252.234   <none>        8083/TCP                     110s

sahil@SAHI-SNEH MINGW64 ~
$ kubectl get pods -n argocd
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     1/1     Running   0          115s
argocd-applicationset-controller-684cd5f5cc-v2zpj   1/1     Running   0          119s
argocd-dex-server-77c55fb54f-dprdm                  1/1     Running   0          119s
argocd-notifications-controller-69cd888b56-ttxcc    1/1     Running   0          118s
argocd-redis-55c76cb574-g6mq8                       1/1     Running   0          117s
argocd-repo-server-584d45d88f-gdmgs                 1/1     Running   0          117s
argocd-server-8667f8577-bc5ch                       1/1     Running   0          116s
```



NAME	vote-76fc6b86f-b7gkb
NAMESPACE	default
CREATED AT	01/14/2025 19:27:46 (5 minutes ago)
IMAGES	votingapp2025/votingapp:15
STATE	ImagePullBackOff
STATE DETAILS	Back-off pulling image "votingapp2025/votingapp:15"
CONTAINER STATE	vote Container is *waiting* because of *ImagePullBackOff*. It is *not started and not ready*.
HEALTH	Back-off pulling image "votingapp2025/votingapp:15"
LINKS	

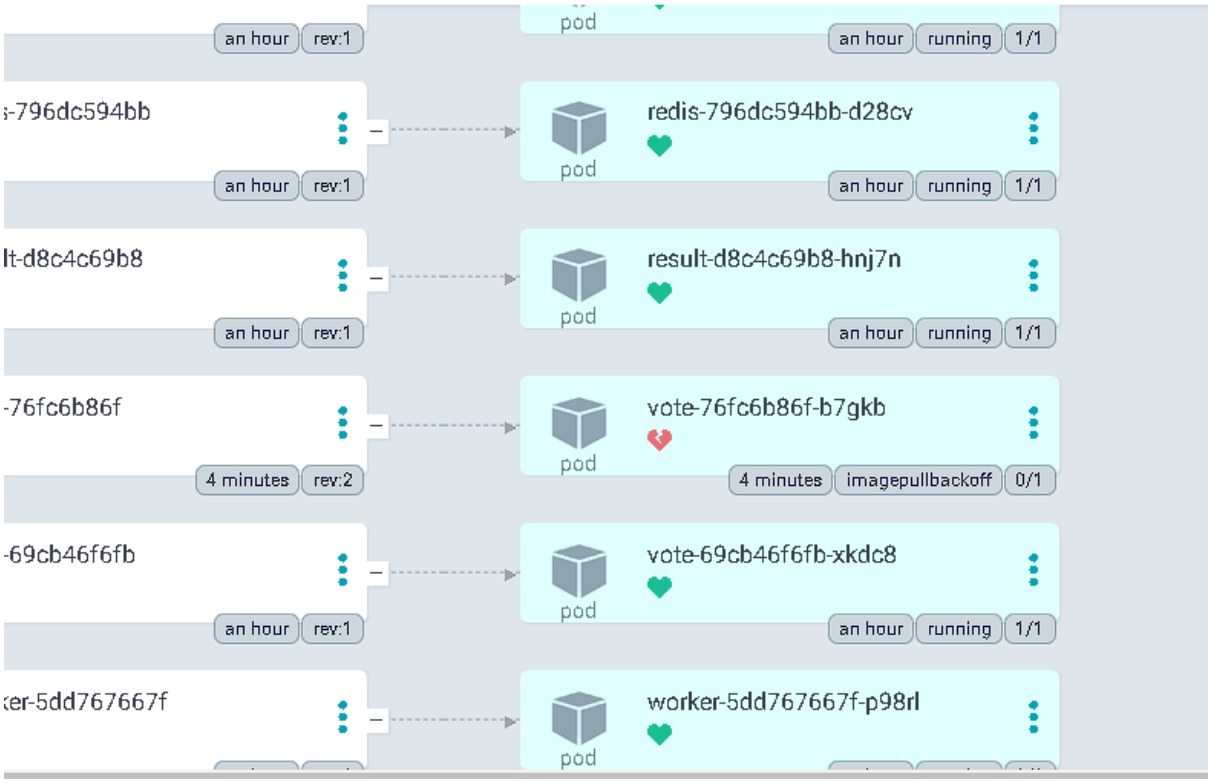
LAST SYNC

Sync OK to 2f8c0f2

Succeeded 4 minutes ago (Tue Jan 14 2025 19:27:46 GMT+0530)

Author: Ubuntu <azure@azureagent.svftgcj1whze1aoobml1...

Comment: Update Kubernetes manifest



5. Advanced CI/CD Configurations for Scalability and Security

5.1 Managing Secrets Securely

- **Azure Key Vault** securely stores API keys, database credentials, and secrets.
- Kubernetes **Secrets** ensure encrypted storage of sensitive information.

5.2 Implementing Auto-Scaling for Microservices

- **Horizontal Pod Autoscaler (HPA)** scales services based on CPU and memory usage.
- **Cluster Autoscaler** dynamically adjusts AKS node count.
- **Ingress Controllers** optimize traffic distribution across microservices.

5.3 Enabling Blue-Green and Canary Deployments

- **Blue-Green Deployment** – Releases new versions in parallel, reducing downtime.
- **Canary Deployment** – Gradually introduces updates to minimize impact.

---

6. Monitoring, Logging, and Troubleshooting

6.1 Real-Time Monitoring with Prometheus and Grafana

- **Prometheus** collects Kubernetes metrics.
- **Grafana dashboards** visualize system performance.
- **Alerts** notify teams about failures.

6.2 Centralized Logging with Azure Monitor and ELK Stack

- **Azure Monitor** tracks application performance logs.
- **ELK Stack (Elasticsearch, Logstash, and Kibana)** provides log analysis.

6.3 Automated Incident Management

- **Application Insights** detects performance anomalies.
- **Incident response automation** ensures quick rollback if needed.

```
azure@azureagent: ~/myagent
2025-01-14 13:08:52Z: Job Build completed with result: Succeeded
2025-01-14 13:08:56Z: Running job: Push
2025-01-14 13:09:25Z: Job Push completed with result: Succeeded
2025-01-14 13:09:29Z: Running job: Update
2025-01-14 13:09:45Z: Job Update completed with result: Succeeded
2025-01-14 13:13:37Z: Running job: Build
2025-01-14 13:14:06Z: Job Build completed with result: Succeeded
2025-01-14 13:14:11Z: Running job: Push
2025-01-14 13:14:32Z: Job Push completed with result: Succeeded
2025-01-14 13:14:36Z: Running job: Update
2025-01-14 13:14:51Z: Job Update completed with result: Succeeded
2025-01-14 13:30:07Z: Running job: Build
2025-01-14 13:30:36Z: Job Build completed with result: Succeeded
2025-01-14 13:30:41Z: Running job: Push
2025-01-14 13:30:58Z: Job Push completed with result: Succeeded
2025-01-14 13:31:02Z: Running job: Update
2025-01-14 13:31:17Z: Job Update completed with result: Succeeded
2025-01-14 13:39:36Z: Running job: Build
2025-01-14 13:40:03Z: Job Build completed with result: Succeeded
2025-01-14 13:40:07Z: Running job: Push
2025-01-14 13:40:27Z: Job Push completed with result: Succeeded
2025-01-14 13:40:31Z: Running job: Update
2025-01-14 13:49:33Z: Job Update completed with result: Canceled
2025-01-14 13:52:14Z: Running job: Build
2025-01-14 13:52:44Z: Job Build completed with result: Succeeded
2025-01-14 13:52:48Z: Running job: Push
2025-01-14 13:53:10Z: Job Push completed with result: Succeeded
2025-01-14 13:53:14Z: Running job: Update
2025-01-14 13:53:30Z: Job Update completed with result: Failed
2025-01-14 13:55:22Z: Running job: Build
2025-01-14 13:55:51Z: Job Build completed with result: Succeeded
2025-01-14 13:55:55Z: Running job: Push
2025-01-14 13:56:14Z: Job Push completed with result: Succeeded
2025-01-14 13:56:17Z: Running job: Update
2025-01-14 13:56:35Z: Job Update completed with result: Succeeded
2025-01-14 14:16:53Z: Running job: Build
2025-01-14 14:17:26Z: Job Build completed with result: Succeeded
2025-01-14 14:17:30Z: Running job: Push
2025-01-14 14:17:49Z: Job Push completed with result: Succeeded
2025-01-14 14:17:53Z: Running job: Update
2025-01-14 14:18:12Z: Job Update completed with result: Succeeded
2025-01-14 14:20:32Z: Running job: Build
2025-01-14 14:21:04Z: Job Build completed with result: Succeeded
2025-01-14 14:21:08Z: Running job: Push
2025-01-14 14:21:30Z: Job Push completed with result: Succeeded
2025-01-14 14:21:34Z: Running job: Update
2025-01-14 14:21:52Z: Job Update completed with result: Succeeded
```

---

7. Benefits of This CI/CD Pipeline Setup

7.1 Improved Deployment Speed and Reliability

- Automates the entire deployment process.
- Ensures **quick recovery** in case of failures.

7.2 Enhanced Security and Compliance

- Secure handling of **credentials and environment variables**.
- **Continuous vulnerability scanning** detects security threats.

7.3 Cost Optimization and Resource Efficiency

- **Auto-scaling** optimizes cloud resources.

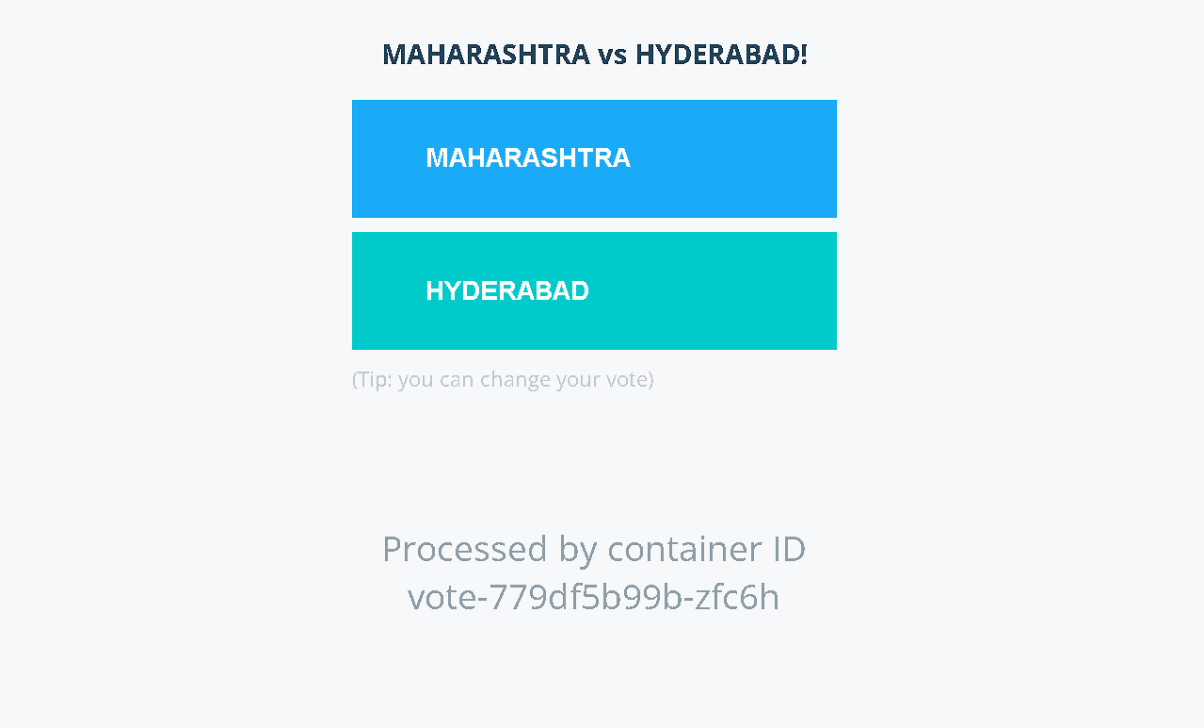
- **Monitoring tools** detect inefficient resource usage.

7.4 Seamless Collaboration and Developer Productivity

- Developers focus on coding rather than deployments.
- **Automated feedback loops** improve software quality.

---

9. Conclusion



Setting up a **multi-microservice CI/CD pipeline using Azure Pipelines and Argo CD** provides an efficient, automated, and scalable solution for modern application deployments.

By following **GitOps best practices**, leveraging **Azure Kubernetes Service (AKS)**, and integrating **monitoring tools**, organizations achieve **faster releases, better security, and reduced downtime**.

With this approach, teams can focus on innovation while ensuring that applications remain **reliable, resilient, and optimized** in production.
