

# Automated Deployment of a Containerized Web Application Using ArgoCD & GitHub Actions

*Ayomide Ajayi*

## Introduction

This Project implemented an automated deployment of a containerized web application using **ArgoCD** and **GitHub Actions**, executed via a self-hosted runner on a local Windows machine. The environment is powered by **Docker Desktop**, with **Kubernetes managing a three-node cluster** to ensure efficient orchestration and deployment.

By implementing a **GitOps-driven deployment pipeline**, this project leverages **ArgoCD** and **GitHub Actions** to achieve automation, security, and reliability. Infrastructure is maintained declaratively in a Git repository, ensuring continuous reconciliation with the desired state while minimizing manual intervention and configuration drift.

Despite its simplicity, this project is directly applicable to **production environments**, offering individuals and organizations the flexibility to customize and enhance the implementation to meet their specific needs while benefiting from its inherent advantages.

## Project Overview

- **Platform:** Windows 11 with **Docker Desktop Kubernetes (3-node cluster)**.
- **Deployment Tool:** **ArgoCD** (manually installed in the argocd namespace).
- **CI/CD:** **GitHub Actions** for the automating of deployments.
- **Application Exposure:** **NodePort services** for external access.
- **Access Method:** **Port Forwarding** for browser access.
- **Runner Type:** **Self-hosted GitHub Actions runner** for local execution.

## **Project Deliverables**

This project delivers a **fully automated, GitOps-driven deployment pipeline** for Kubernetes applications, ensuring robust automation, security, consistency, and scalability. The following are the key deliverables and tangible outcomes of this project implementation:

1. **Automated Deployment System:** A fully automated pipeline using **ArgoCD** and **GitHub Actions**, enabling application updates and infrastructure changes to be deployed seamlessly via Git commits.
2. **Continuous Synchronization Mechanism:** A GitOps-based reconciliation process that ensures alignment between the **Git repository** and the **Kubernetes cluster**, preventing configuration drift.
3. **Enhanced Security & Auditability Framework:** Enforced **version control** and infrastructure audit trails, allowing for secure deployment management and traceability of changes.
4. **Configuration Consistency & Rollback Capability:** A mechanism to maintain application stability, continuously reconciling with the desired state and enabling **rapid rollbacks** to prevent downtime in case of failures.
5. **Scalable Microservices Deployment Strategy:** A Kubernetes architecture supporting **dynamic scaling**, optimized for microservices-based applications, ensuring efficient resource utilization.

This approach delivers a **robust, efficient, and resilient Kubernetes deployment strategy**, providing a **structured framework** for managing containerized applications with **ArgoCD and GitHub Actions**. It enables **seamless automation**, ensures **infrastructure consistency**, and enhances **operational reliability**, optimizing deployment workflows for scalability and maintainability.

## Step-by-Step Implementation

### Prerequisites

1. **Install Docker Desktop**
2. **Install Kubectl**
3. **Set Up Kubernetes on Docker Desktop**
4. **Verify 3-node Cluster Availability**
5. **Install ArgoCD Manually in the argocd Namespace.**

- **Create the argocd namespace:** kubectl create namespace argocd.
- **Apply ArgoCD installation manifest:**

```
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

- **Verify ArgoCD pods are running:** *kubectl get pods -n argocd*
- **Expose ArgoCD UI using NodePort/Port-forwarding:**

**Apply the service:** *kubectl apply -f argoservice.yaml*

- **Access ArgoCD UI via browser:**  <http://localhost:8080>
- **Retrieve ArgoCD admin password:** *using powershell commands*

6. **Create the manifests for infrastructure, services and ArgoCD Application**
7. **Automate Deployment Using GitHub Actions (Self-Hosted Runner)**

**✓ GitHub Actions Workflow (.github/workflows/kube.yaml) Using a Self-Hosted Runner**

## Technical Overview

This project highlights the **interdependency between ArgoCD, GitHub Actions, and Git repositories**, demonstrating their combined role in deploying **scalable, secure, and automated infrastructure**. Efficient deployment of a **containerized web application** relies on a **seamlessly interconnected workflow**, where each component plays a distinct role in **automation, management, and maintaining deployment integrity**.

## Architecture & Workflow

### 1. The Git Repository: Source of Truth

The Git repository **acts as the single source of truth** for the containerized web application as it stores various artifacts including but not limited to **Application source code (containerized in this case)**, **Kubernetes deployment manifests**, **ArgoCD application configurations** and **GitHub Actions workflow definitions**. Hence, the **key responsibilities of the Git repository include**:

- ✓ **Version Control:** Git enables tracking changes, rollbacks, and collaborative development.
- ✓ **Declarative Infrastructure:** Kubernetes manifests stored in Git define the desired state of the application.
- ✓ **Integration with GitOps:** ArgoCD continuously monitors changes in the repo, ensuring deployments are still up to date.

### 2. ArgoCD: Kubernetes Continuous Deployment

ArgoCD as a **GitOps tool facilitates the automation of continuous deployment and management of the Kubernetes application** using CICD and **Git repositories** as the control mechanism. Hence, ArgoCD offers key responsibilities that are central to:

**ArgoCD ensures seamless GitOps-driven Kubernetes deployments through several key mechanisms:**

- **Application Syncing:** It continuously monitors the Git repository and automatically synchronises Kubernetes manifests, ensuring deployments stay up to date.
- **Declarative Deployment:** The live cluster state is consistently aligned with the desired state defined in the Git repository, minimizing manual intervention.

- **Self-Healing:** If unauthorized changes are made directly in the Kubernetes cluster, ArgoCD detects and reverts them to maintain consistency.
- **Automated Rollbacks:** In case of deployment failures, ArgoCD can roll back to the last known stable version, preventing downtime and errors.

## Interaction with Git in ArgoCD Deployment

ArgoCD utilizes **application manifests** to track designated branches and paths within a **Git repository**, continuously monitoring for changes pushed by **GitHub Actions**. When updates are detected, **automated synchronization** is triggered, ensuring that deployments remain secure, traceable, and fully aligned with the declared Kubernetes configuration. This approach guarantees a **self-healing, GitOps-driven workflow**, minimizing manual intervention while maintaining infrastructure integrity.

### 3. GitHub Actions: Automating CI/CD

GitHub Actions provided an automation layer for **building, testing, and updating application deployments**. These key responsibilities were achieved during this project implementation:

- ✓ **Trigger Deployments:** Runs workflows upon code changes (push, merge).
- ✓ **Manifest Updates:** Applies Kubernetes manifests from Git to trigger ArgoCD synchronization.
- ✓ **Port Forwarding and Service Exposure:** Automates Kubernetes service setup for accessibility.

## Interaction Between Git & ArgoCD

- **GitHub Actions triggers workflows on code changes**, updating deployment manifests in the **Git repository**.
- **ArgoCD detects updates in Git**, pulls the latest manifests, and synchronizes changes with the **Kubernetes cluster**.
- **CI/CD pipeline ensures deployment integrity**, automating tests before ArgoCD applies updates, maintaining a secure and reliable deployment process.

#### **4. Rollback & Disaster Recovery**

- a. Every deployment is version-controlled, allowing easy rollback to previous states if needed.
- b. ArgoCD ensures self-healing deployments by reapplying desired configurations if accidental changes occur.

### **Deployment Process**

#### **1. Containerization:**

- o The web application is packaged into a Docker container.
- o The container image is pushed to and stored in a local registry.

#### **2. Infrastructure as Code:**

- o Kubernetes manifests define application deployment, services, and ingress rules.
- o ArgoCD application configuration ensures automated synchronization.

#### **3. GitOps Workflow:**

- o Code pushed to GitHub, triggering GitHub Actions for deployment builds.
- o Updated manifests in the Git repository signal ArgoCD to deploy the new application version.

#### **4. Monitoring & Observability:**

- o ArgoCD UI provides real-time visibility into application deployment status.
- o [Logging tools \(such as Prometheus and Grafana\) can be integrated for performance monitoring albeit not implemented.](#)

## Project Artefacts

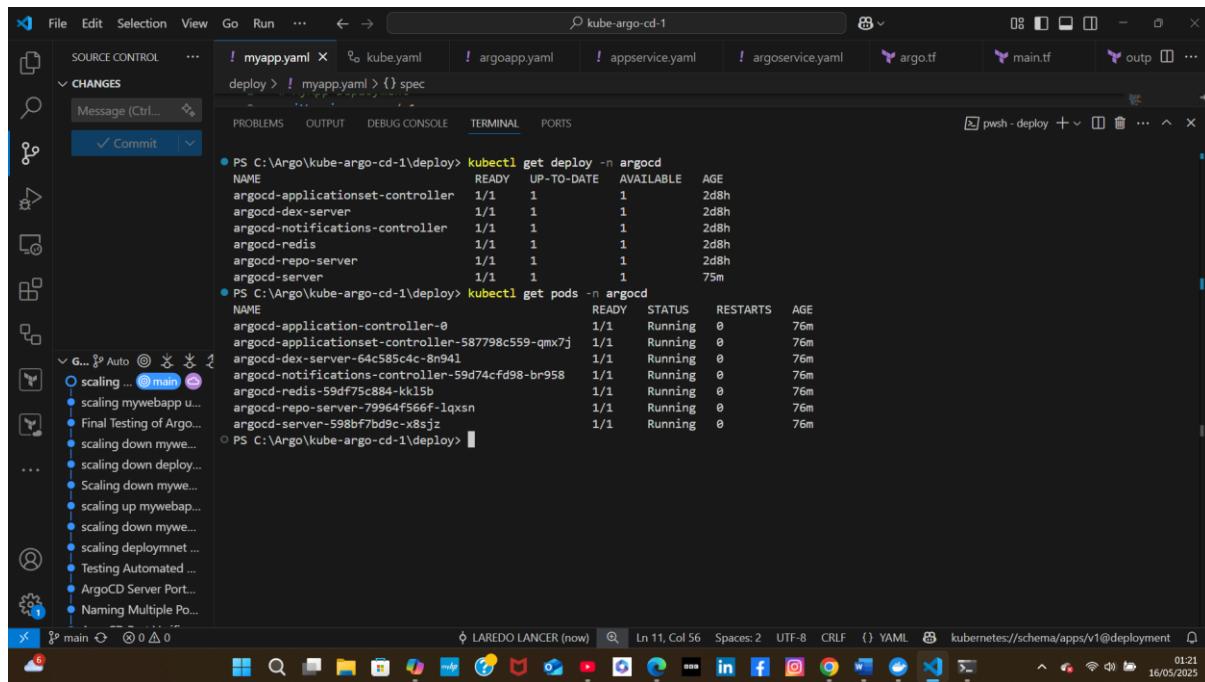
### Project Outcomes: Testing ArgoCD for Web Application Deployment

To evaluate the functionality of ArgoCD in deploying a web application on Kubernetes, I conducted a structured assessment focusing on deployment accuracy, associated services, scaling behaviour, security measures, high availability, and fault tolerance. The following systematic testing approach ensured comprehensive validation of these key aspects.

#### A. Initial Deployment Verification

##### Steps:

1. **Ensure ArgoCD is Installed & Running:** Confirm that all ArgoCD components (server, repo-server, controller, etc.) are running.



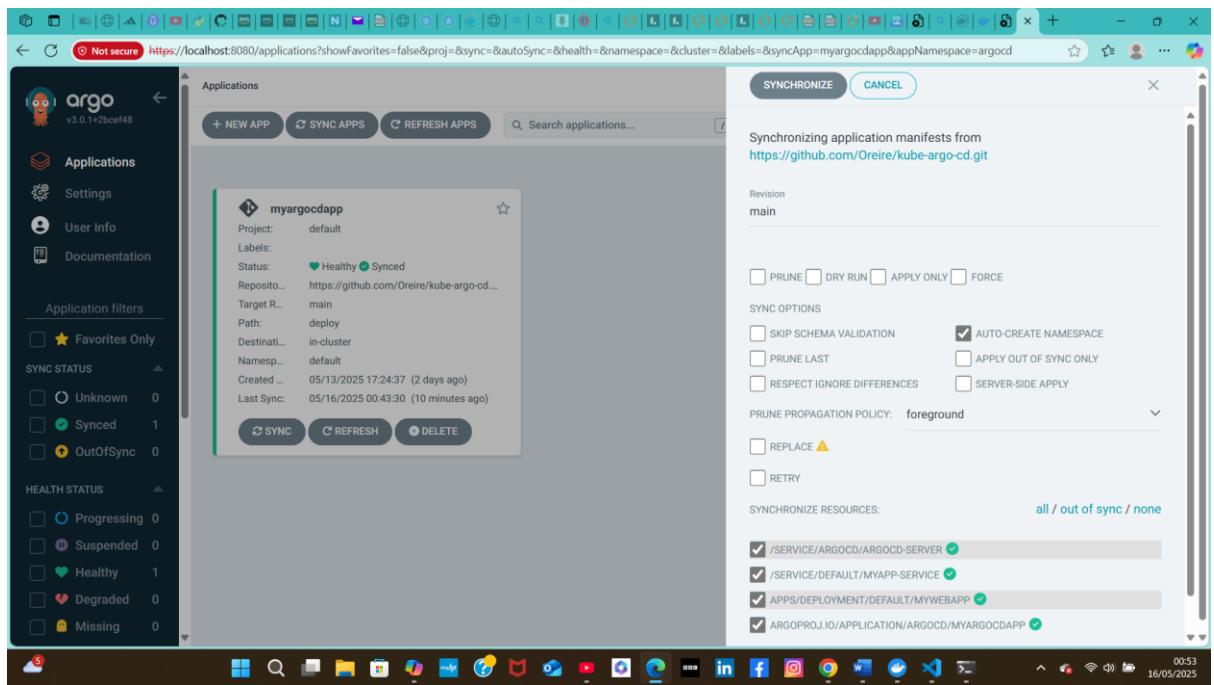
The screenshot shows the ArgoCD application interface. On the left, there's a sidebar with various project and deployment-related items. The main area has tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected), and 'PORTS'. The 'TERMINAL' tab displays command-line logs from a PowerShell session (pswsh - deploy). The logs show the execution of 'kubectl get deploy -n argocd' and 'kubectl get pods -n argocd', both of which return tables of running pods. The terminal also shows the command 'PS C:\Argo\kube-argo-cd-1\deploy>'.

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
argocd-applicationset-controller	1/1	1	1	2d8h
argocd-dex-server	1/1	1	1	2d8h
argocd-notifications-controller	1/1	1	1	2d8h
argocd-redis	1/1	1	1	2d8h
argocd-repo-server	1/1	1	1	2d8h
argocd-server	1/1	1	1	75m

NAME	READY	STATUS	RESTARTS	AGE
argocd-application-controller-0	1/1	Running	0	76m
argocd-applicationset-controller-587798c559-qmx7j	1/1	Running	0	76m
argocd-dex-server-64c585c4c-8n94l	1/1	Running	0	76m
argocd-notifications-controller-59d74cf98-br958	1/1	Running	0	76m
argocd-redis-59df75c884-kkl5b	1/1	Running	0	76m
argocd-repo-server-79964f566f-lqxsn	1/1	Running	0	76m
argocd-server-598bf7bd9c-x8sjz	1/1	Running	0	76m

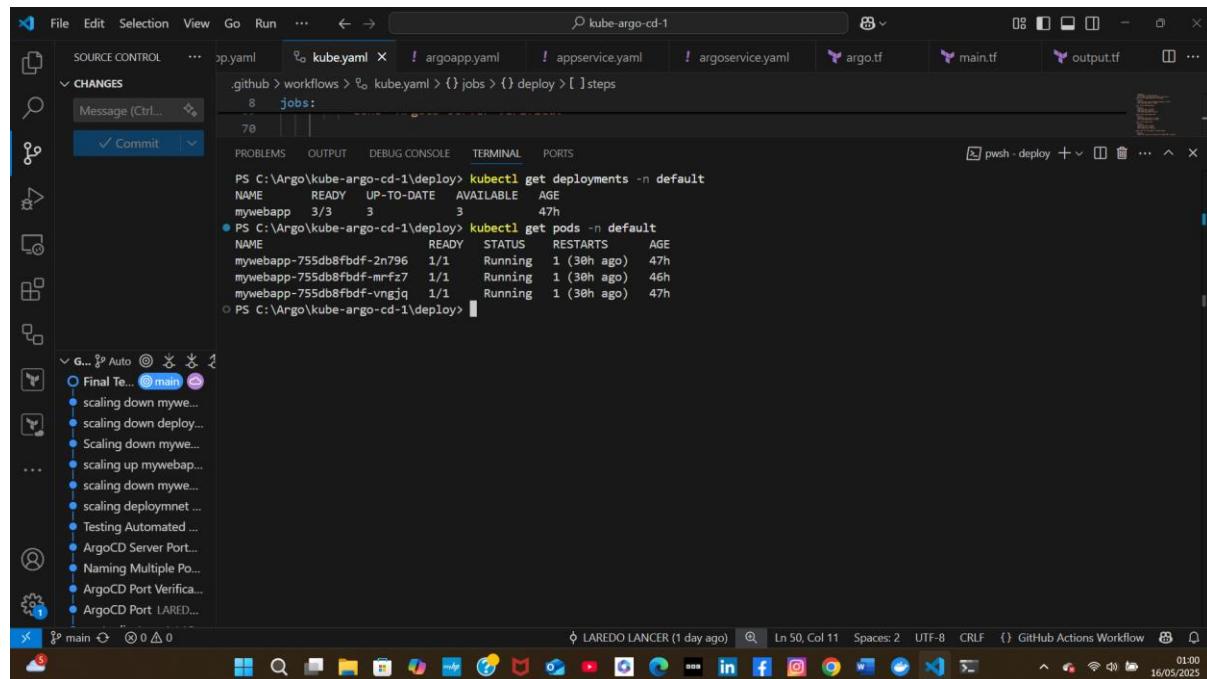
## 2. Check Application Syncing

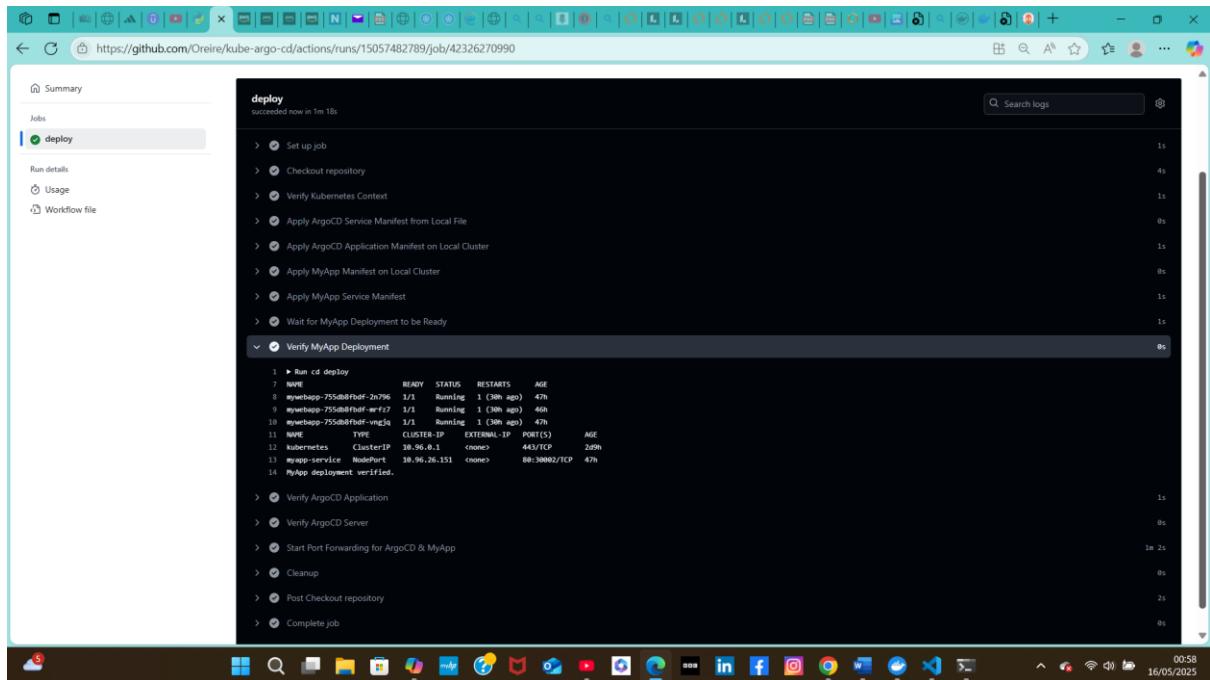
Apply an ArgoCD application manifest:



## 3. Confirm Deployment Creation

### Check web Deployment and Pods Status



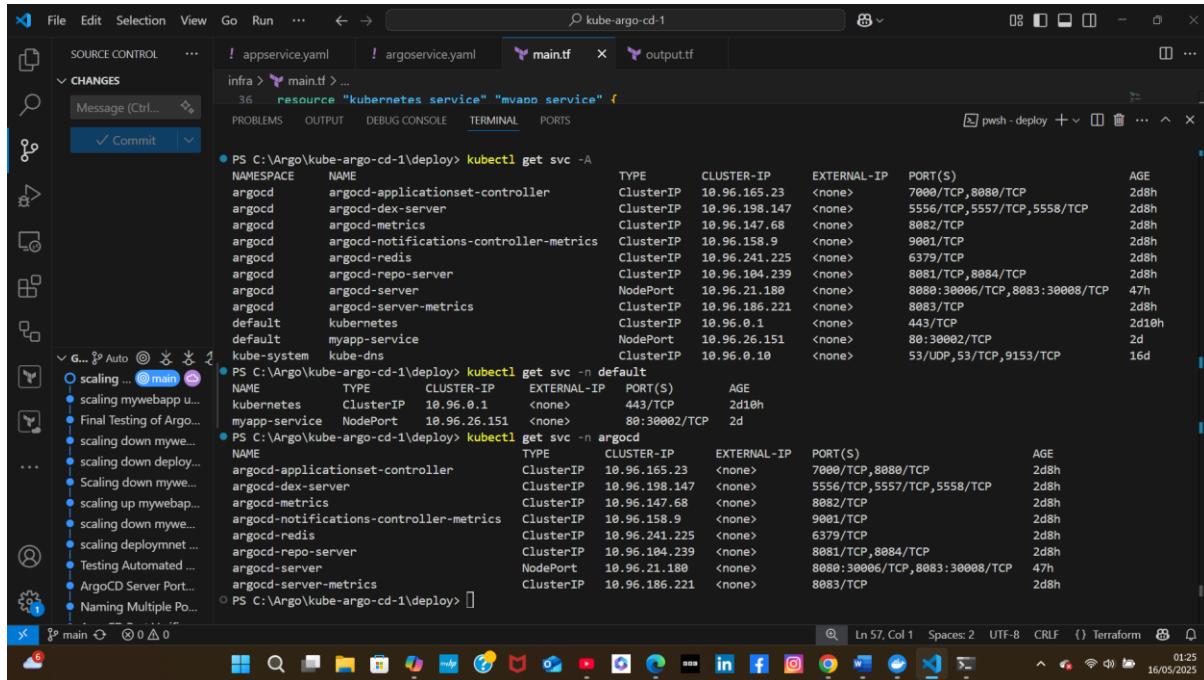


#### 4. Verify ArgoCD UI Sync

Access <https://localhost:8080>

## 2. Validating Associated Service

### Check If the Web Application Service Is Running



The screenshot shows a terminal window with several command-line outputs. The first output lists all services in the default namespace:

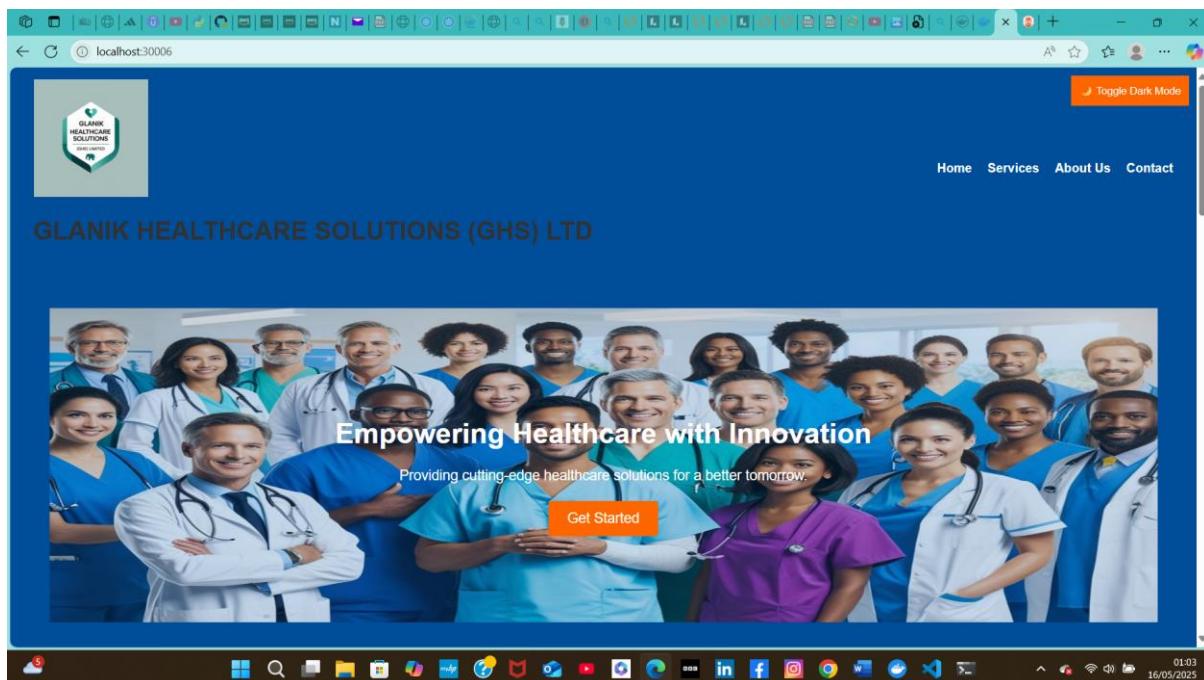
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
argocd-applicationset-controller	ClusterIP	10.96.165.23	<none>	7000/TCP, 8080/TCP	2d8h
argocd-dex-server	ClusterIP	10.96.198.147	<none>	5556/TCP, 5557/TCP, 5558/TCP	2d8h
argocd-metrics	ClusterIP	10.96.147.68	<none>	8082/TCP	2d8h
argocd-notifications-controller-metrics	ClusterIP	10.96.158.9	<none>	9001/TCP	2d8h
argocd-redis	ClusterIP	10.96.241.225	<none>	6379/TCP	2d8h
argocd-repo-server	ClusterIP	10.96.104.239	<none>	8081/TCP, 8884/TCP	2d8h
argocd-server	NodePort	10.96.21.180	<none>	8080:30006/TCP, 8083:30008/TCP	47h
argocd-server-metrics	ClusterIP	10.96.186.221	<none>	8083/TCP	2d8h
default-kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	2d10h
default-myapp-service	NodePort	10.96.26.151	<none>	80:30002/TCP	2d
kube-system-kube-dns	ClusterIP	10.96.0.10	<none>	53/UDP, 53/TCP, 9153/TCP	16d

The second output lists services in the argocd namespace:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
argocd-applicationset-controller	ClusterIP	10.96.165.23	<none>	7000/TCP, 8088/TCP	2d8h
argocd-dex-server	ClusterIP	10.96.198.147	<none>	5556/TCP, 5557/TCP, 5558/TCP	2d8h
argocd-metrics	ClusterIP	10.96.147.68	<none>	8082/TCP	2d8h
argocd-notifications-controller-metrics	ClusterIP	10.96.158.9	<none>	9001/TCP	2d8h
argocd-redis	ClusterIP	10.96.241.225	<none>	6379/TCP	2d8h
argocd-repo-server	ClusterIP	10.96.104.239	<none>	8081/TCP, 8884/TCP	2d8h
argocd-server	NodePort	10.96.21.180	<none>	8080:30006/TCP, 8083:30008/TCP	47h
argocd-server-metrics	ClusterIP	10.96.186.221	<none>	8083/TCP	2d8h

- Services correctly created and configured for external access.

### B. Manually Access the Web Application (mywebapp)



localhost:30006

**About Us**

**GHS was established in April 2022 as a consequence of the identified Healthcare workforce needs in the UK.**

The organisation prides itself as a state-of-the-art enterprise that has the knowledge, talent, and competence. We provide services, human resources, capacity building, and development across various spectrums of the UK Healthcare domain. Consequently, our underlying philosophy is enshrined in the need for the delivery of personalised services to all our clients, be they individuals or organisations. Also, we are a leading provider of healthcare technology and consultancy services, dedicated to improving patient outcomes and operational efficiency. Our team of experts combines deep industry knowledge with innovative technology solutions to address the unique challenges faced by healthcare providers. Therefore, we enable seamless interactions with healthcare providers and patients through technology, education and professional competencies.

**Our Services**



**Telemedicine Solutions**  
Connecting patients with healthcare



**Healthcare Data Analytics**  
AI-driven insights to optimize medical



**Medical Software Development**  
Customized applications for healthcare

Toggle Dark Mode

localhost:30006

"GHS transformed our healthcare operations with their innovative solutions!"  
- Healthcare Provider

**Contact Us**

Name:

Email:

Message:

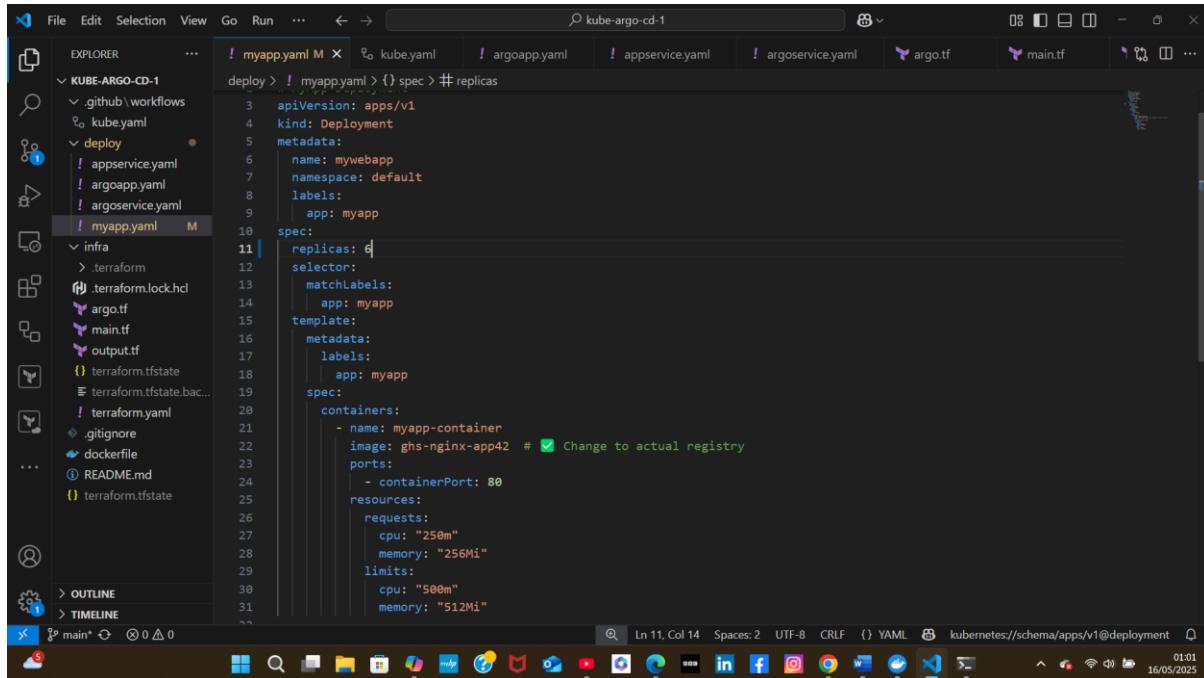
**Send Message**

© 2025 Glanik Healthcare Solutions. All Rights Reserved.

localhost:30006 (via port-forwarding)

## C. Scaling Tests for High Availability

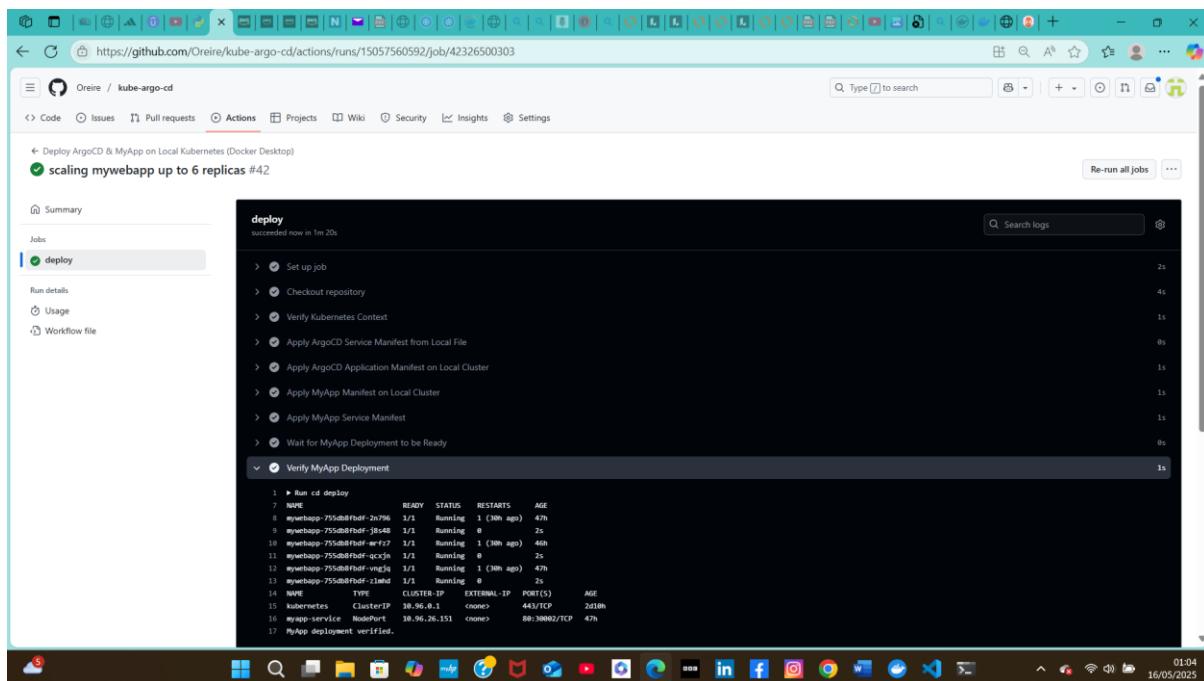
### Scaling Up Deployment



The screenshot shows a code editor interface with multiple tabs open. The left sidebar displays a file tree for a project named 'KUBE-ARGO-CD-1'. The main editor area contains a file named 'myapp.yaml' which defines a Kubernetes Deployment. The deployment has a single replica and specifies a container named 'myapp-container' with specific resource requirements (cpu: "250m", memory: "256Mi") and limits (cpu: "500m", memory: "512Mi"). Other files visible in the tree include 'kube.yaml', 'deploy', 'infra', 'main.tf', 'output.tf', and various Terraform state files.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mywebapp
  namespace: default
  labels:
    app: myapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: myapp-container
          image: ghs-nginx-app42 # Change to actual registry
          ports:
            - containerPort: 80
          resources:
            requests:
              cpu: "250m"
              memory: "256Mi"
            limits:
              cpu: "500m"
              memory: "512Mi"
```

### Scaling deployment up to 6 replicas: Pipeline



The screenshot shows a GitHub Actions pipeline log for a job named 'scaling mywebapp up to 6 replicas #42'. The pipeline consists of several steps: 'Setup job', 'Checkout repository', 'Verify Kubernetes Context', 'Apply ArgoCD Service Manifest from Local File', 'Apply ArgoCD Application Manifest on Local Cluster', 'Apply MyApp Manifest on Local Cluster', 'Apply MyApp Service Manifest', 'Wait for MyApp Deployment to be Ready', and 'Verify MyApp Deployment'. The final step shows the deployment logs, which list six replicas of the 'mywebapp' deployment, each in a 'Running' state with a short age. It also lists the 'myapp-service' NodePort service and the 'kubernetes' ClusterIP service.

NAME	READY	STATUS	RESTARTS	AGE
mywebapp-75d8fbef-2n796	1/1	Running	1 (30h ago)	47h
mywebapp-75d8fbef-18448	1/1	Running	0	2s
mywebapp-75d8fbef-wr7f7	1/1	Running	1 (30h ago)	46h
mywebapp-75d8fbef-1qjwz	1/1	Running	1 (30h ago)	47s
mywebapp-75d8fbef-wng3q	1/1	Running	1 (30h ago)	47h
mywebapp-75d8fbef-j1med	1/1	Running	0	2s

```
PS C:\Argo\kube-argo-cd-1\deploy> kubectl get pods -n default
NAME          READY   STATUS    RESTARTS   AGE
mywebapp-755db8fbdf-2n796  1/1    Running   1 (30h ago)  47h
mywebapp-755db8fbdf-j8s48  1/1    Running   0          2m17s
mywebapp-755db8fbdf-mrfz7  1/1    Running   1 (30h ago)  46h
mywebapp-755db8fbdf-qckjn  1/1    Running   0          2m17s
mywebapp-755db8fbdf-vngjq  1/1    Running   1 (30h ago)  47h
mywebapp-755db8fbdf-zlmhd  1/1    Running   0          2m17s
PS C:\Argo\kube-argo-cd-1\deploy> kubectl get deployments -n default
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
mywebapp   6/6     6           6           47h
```

The screenshot shows a terminal window within a development environment. It displays the output of two Kubernetes commands: `kubectl get pods -n default` and `kubectl get deployments -n default`. The terminal window is part of a larger interface with a sidebar containing file navigation and commit history.

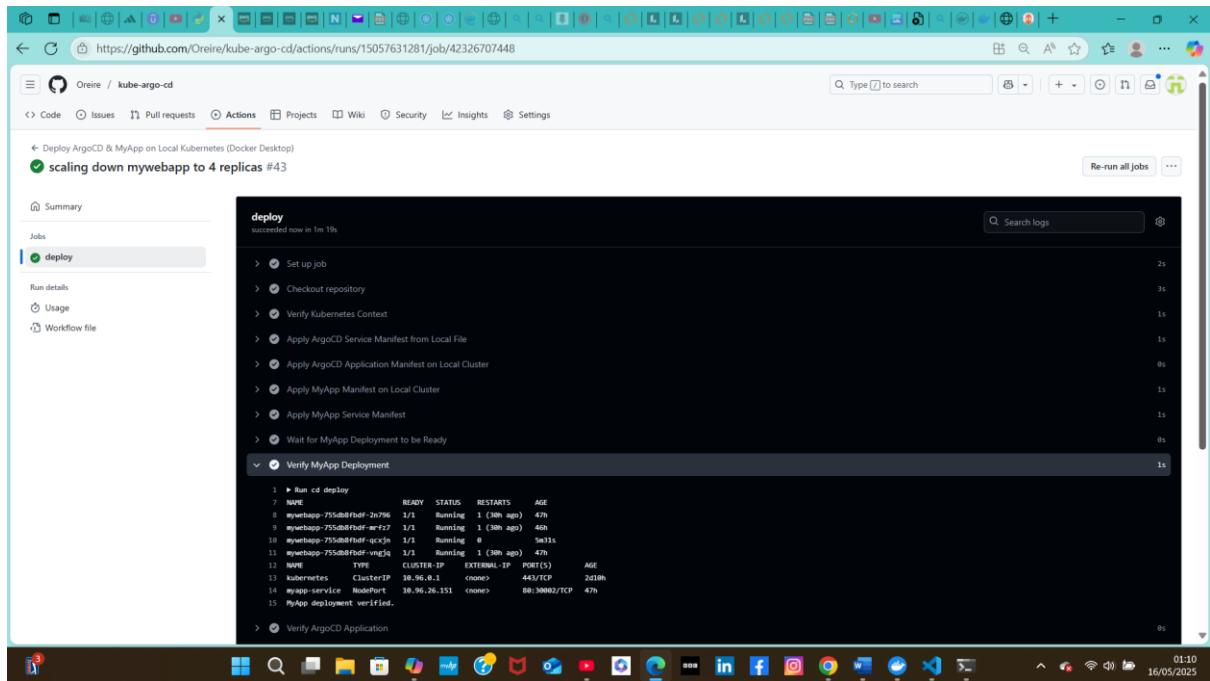
## Scaling Down Deployment

Scaling down Mywebapp deployment to 4 replicas

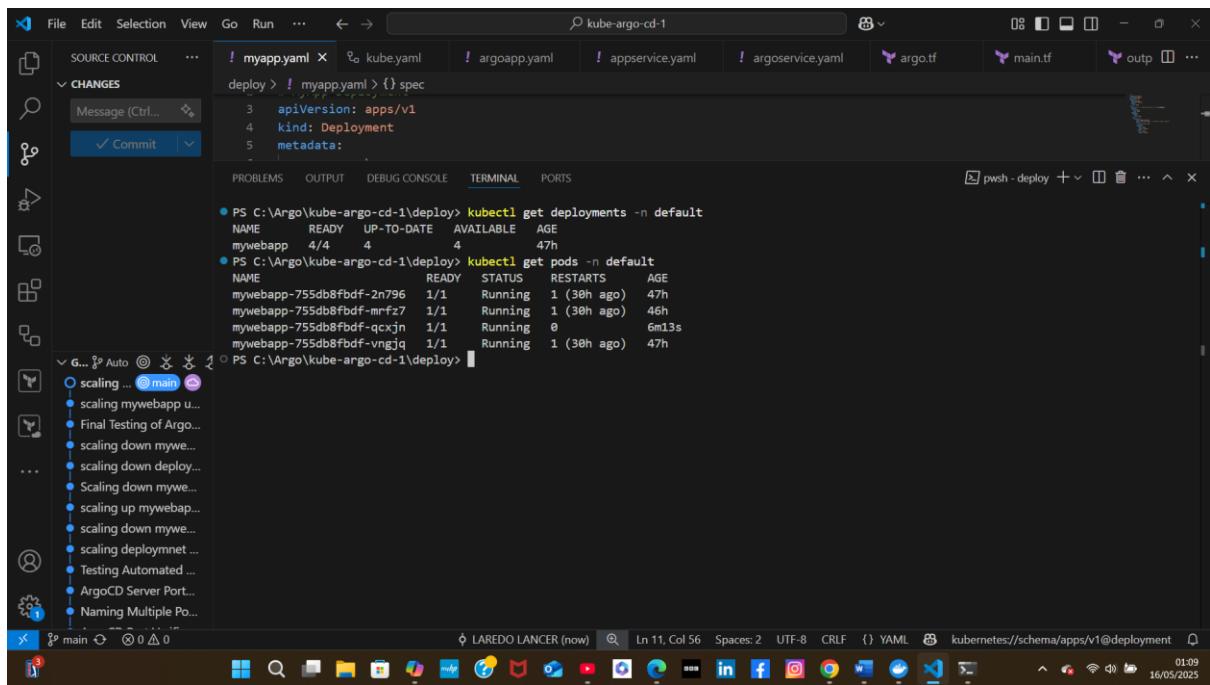
```
PS C:\Argo\kube-argo-cd-1\deploy> kubectl get deployment myapp -n default -o yaml | sed -i 's/replicas: 6/replicas: 4/' myapp.yaml
```

The screenshot shows a terminal window where a command is being run to modify a deployment manifest. The command uses `kubectl` to get the current deployment configuration for 'myapp' in the 'default' namespace, outputs it as YAML, and then uses `sed` to change the 'replicas' field from 6 to 4. This results in a new manifest where the deployment is set to have 4 replicas. The terminal window is part of a larger interface with a sidebar containing file navigation and commit history.

## Git push to deploy via ArgoCD (pipeline)



## Vscode verification



## D. Fault Tolerance Tests

### Forced Pod Failures to observe recovery:

Delete a pod and check if Kubernetes restarts it:

1. pod "mywebapp-755db8fbdf-2n796" deleted.
2. mywebapp-755db8fbdf-xbqkb (new pod created via self-healing and ArgoCD SYN)

```

resource "kubernetes_service" "myapp_service" {
  ...
}

PS C:\Argo\kube-argo-cd-1\deploy> kubectl get pods -n default
NAME           READY   STATUS    RESTARTS   AGE
mywebapp-755db8fbdf-2n796  1/1    Running   1 (32h ago)  2d1h
mywebapp-755db8fbdf-mrfz7  1/1    Running   1 (32h ago)  2d
mywebapp-755db8fbdf-qcxjn  1/1    Running   0          113m
mywebapp-755db8fbdf-vngjq  1/1    Running   1 (32h ago)  2d1h
● PS C:\Argo\kube-argo-cd-1\deploy> kubectl delete pod/mywebapp-755db8fbdf-2n796
pod "mywebapp-755db8fbdf-2n796" deleted
● PS C:\Argo\kube-argo-cd-1\deploy> kubectl get pods -n default
NAME           READY   STATUS    RESTARTS   AGE
mywebapp-755db8fbdf-mrfz7  1/1    Running   1 (32h ago)  2d
mywebapp-755db8fbdf-qcxjn  1/1    Running   0          116m
mywebapp-755db8fbdf-vngjq  1/1    Running   1 (32h ago)  2d1h
mywebapp-755db8fbdf-xbqkb  1/1    Running   0          61s
○ PS C:\Argo\kube-argo-cd-1\deploy>

```

**Delete pod in namespace:** kubectl delete pod <pod-name> -n default.

### Monitor pod recreation:

kubectl get pods -n default -w (recreated pod: mywebapp-755db8fbdf-vblng)

```

resource "kubernetes_service" "myapp_service" {
  ...
}

PS C:\Argo\kube-argo-cd-1\deploy> kubectl get pods -n default
NAME           READY   STATUS    RESTARTS   AGE
mywebapp-755db8fbdf-2n796  1/1    Running   1 (32h ago)  2d1h
mywebapp-755db8fbdf-mrfz7  1/1    Running   1 (32h ago)  2d
mywebapp-755db8fbdf-qcxjn  1/1    Running   0          113m
mywebapp-755db8fbdf-vngjq  1/1    Running   1 (32h ago)  2d1h
● PS C:\Argo\kube-argo-cd-1\deploy> kubectl delete pod/mywebapp-755db8fbdf-2n796
pod "mywebapp-755db8fbdf-2n796" deleted
● PS C:\Argo\kube-argo-cd-1\deploy> kubectl get pods -n default
NAME           READY   STATUS    RESTARTS   AGE
mywebapp-755db8fbdf-mrfz7  1/1    Running   1 (32h ago)  2d
mywebapp-755db8fbdf-qcxjn  1/1    Running   0          116m
mywebapp-755db8fbdf-vngjq  1/1    Running   1 (32h ago)  2d1h
mywebapp-755db8fbdf-xbqkb  1/1    Running   0          4m24s
● PS C:\Argo\kube-argo-cd-1\deploy> kubectl get pods -n default -w
NAME           READY   STATUS    RESTARTS   AGE
mywebapp-755db8fbdf-mrfz7  1/1    Running   1 (32h ago)  2d
mywebapp-755db8fbdf-qcxjn  1/1    Running   0          119m
mywebapp-755db8fbdf-vngjq  1/1    Running   1 (32h ago)  2d1h
mywebapp-755db8fbdf-xbqkb  1/1    Running   0          4m24s
● PS C:\Argo\kube-argo-cd-1\deploy> kubectl delete pod/mywebapp-755db8fbdf-xbqkb
pod "mywebapp-755db8fbdf-xbqkb" deleted
○ PS C:\Argo\kube-argo-cd-1\deploy> kubectl get pods -n default -w
NAME           READY   STATUS    RESTARTS   AGE
mywebapp-755db8fbdf-mrfz7  1/1    Running   1 (32h ago)  2d
mywebapp-755db8fbdf-qcxjn  1/1    Running   0          120m
mywebapp-755db8fbdf-vngjq  1/1    Running   1 (32h ago)  2d1h
mywebapp-755db8fbdf-xbqkb  1/1    Running   0          87s

```

## Network Disruptions Simulation

### Testing Node Tainting

To simulate a **network outage**, node disconnection was achieved using **tainting**. However, before applying the taint, it was crucial to determine which pods were already scheduled on the affected nodes. To achieve this, I **scaled the mywebapp deployment to 9 replicas** and verified the **distribution of pods across worker nodes** to assess the impact of tainting on pod placement and scheduling behavior.

### Replicas: 9

Summary

Jobs

deploy

Run details

Usage

Workflow file

deploy

successed 11 minutes ago in 1m 20s

Re-run all jobs

Search logs

NAME	READY	STATUS	RESTARTS	AGE
mywebapp-75d8bf0ef-7het2	1/1	Running	0	5s
mywebapp-75d8bf0ef-jzph2	1/1	Running	0	5s
mywebapp-75d8bf0ef-kchm3	1/1	Running	0	5s
mywebapp-75d8bf0ef-wfz72	1/1	Running	1	(33h ago)
mywebapp-75d8bf0ef-zxcm3	1/1	Running	0	5s
mywebapp-75d8bf0ef-p7k45	1/1	Running	0	5s
mywebapp-75d8bf0ef-dcx3n	1/1	Running	0	13m
mywebapp-75d8bf0ef-vb1q3	1/1	Running	0	13m
mywebapp-75d8bf0ef-vng3g	1/1	Running	1	(33h ago)
kubernetes	ClusterIP	10.96.0.1	none	443/TCP

## Pods Scheduled on Nodes

The screenshot shows the Argo CD application interface. On the left, there's a sidebar with icons for Source Control, Changes, and other project management tools. The main area has tabs for appservice.yaml, argoservice.yaml, main.tf, myapp.yaml (which is currently selected), and output.tf. Below the tabs, there are sections for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active, displaying PowerShell commands and their outputs:

```
PS C:\Argo\kube-argo-cd-1\deploy> kubectl get nodes -n default
NAME           STATUS   ROLES      AGE    VERSION
desktop-control-plane Ready   control-plane 16d   v1.31.1
desktop-worker   Ready   <none>     16d   v1.31.1
desktop-worker2  Ready   <none>     16d   v1.31.1

● PS C:\Argo\kube-argo-cd-1\deploy> kubectl get pods -n default
NAME          READY   STATUS    RESTARTS   AGE
mywebapp-755db8fbdf-7bmt2 1/1    Running   0          13m
mywebapp-755db8fbdf-jzp2q  1/1    Running   0          13m
mywebapp-755db8fbdf-kchhs  1/1    Running   0          13m
mywebapp-755db8fbdf-mrfz7  1/1    Running   1 (33h ago) 2d1h
mywebapp-755db8fbdf-p2gzh  1/1    Running   0          13m
mywebapp-755db8fbdf-p7145  1/1    Running   0          13m
mywebapp-755db8fbdf-qcxjn 1/1    Running   0          146m
mywebapp-755db8fbdf-vblng  1/1    Running   0          27m
mywebapp-755db8fbdf-vngjq  1/1    Running   1 (33h ago) 2d2h

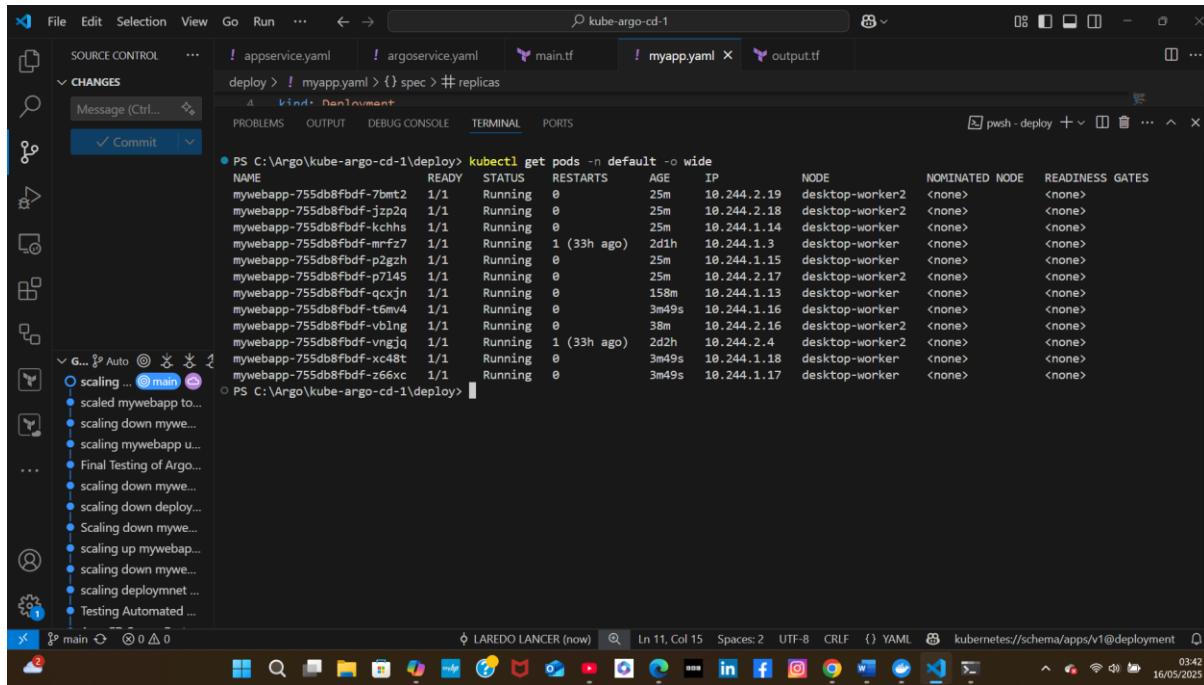
● PS C:\Argo\kube-argo-cd-1\deploy> kubectl get pods -n default --wide
NAME          READY   STATUS    RESTARTS   AGE   IP          NODE   NOMINATED NODE   READINESS GATES
mywebapp-755db8fbdf-7bmt2  1/1   Running   0          14m  10.244.2.19  desktop-worker2  <none>        <none>
mywebapp-755db8fbdf-jzp2q  1/1   Running   0          14m  10.244.2.18  desktop-worker2  <none>        <none>
mywebapp-755db8fbdf-kchhs  1/1   Running   0          14m  10.244.1.14  desktop-worker   <none>        <none>
mywebapp-755db8fbdf-mrfz7  1/1   Running   1 (33h ago) 2d1h  10.244.1.3   desktop-worker   <none>        <none>
mywebapp-755db8fbdf-p2gzh  1/1   Running   0          14m  10.244.1.15  desktop-worker   <none>        <none>
mywebapp-755db8fbdf-p7145  1/1   Running   0          14m  10.244.2.17  desktop-worker2  <none>        <none>
mywebapp-755db8fbdf-qcxjn 1/1   Running   0          147m 10.244.1.13  desktop-worker   <none>        <none>
mywebapp-755db8fbdf-vblng  1/1   Running   0          27m  10.244.2.16  desktop-worker2  <none>        <none>
mywebapp-755db8fbdf-vngjq  1/1   Running   1 (33h ago) 2d2h  10.244.2.4   desktop-worker2  <none>        <none>
```

At the bottom, the terminal shows the user is in LAREDO LANCER (now) and the command was run at 03:31 on 16/05/2025.

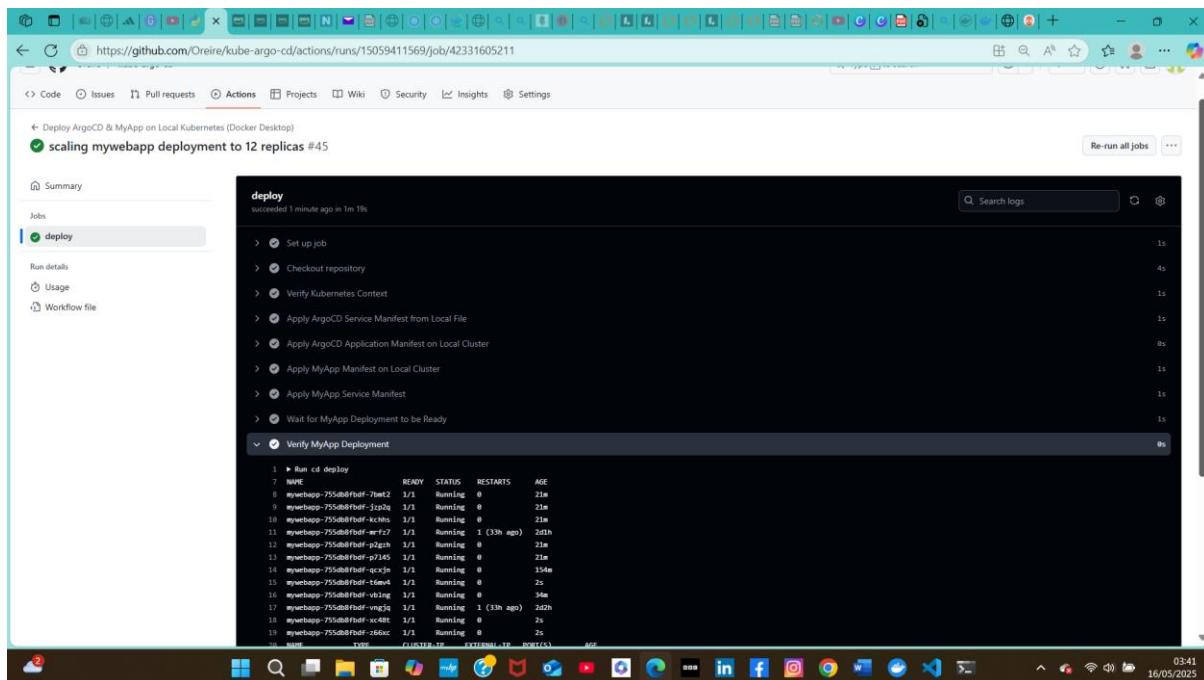
## Tainting of desktop-worker2

kubectl taint nodes desktop-worker2 key=value:NoSchedule

After applying the taint, I scaled my web application deployment to 12 replicas to accommodate an additional 3 pods. Based on the taint restrictions, these new pods should be scheduled exclusively on available nodes, ensuring they are provisioned only on desktop-worker while avoiding desktop-worker2. ArgoCD automatically reconciled the changes, enforcing deployment rules and rescheduling pods, accordingly, as shown in the results below.



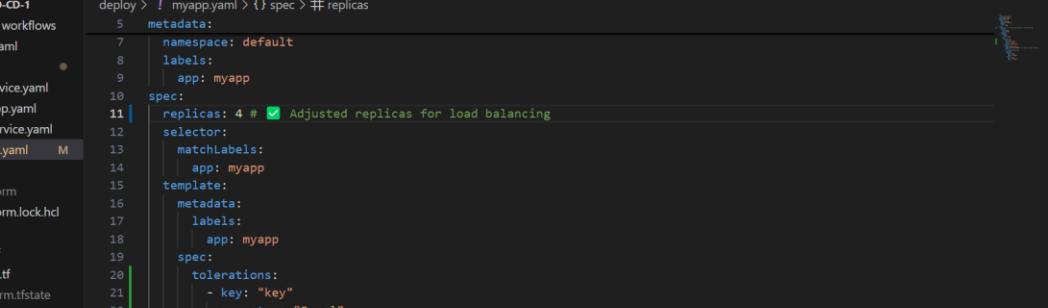
```
PS C:\Argo\kube-argo-cd-1\deploy> kubectl get pods -n default -o wide
NAME          READY   STATUS    RESTARTS   AGE      IP           NODE      NOMINATED NODE   READINESS GATES
mywebapp-755db8fbdf-7bmt2  1/1    Running   0          25m     10.244.2.19  desktop-worker2  <none>        <none>
mywebapp-755db8fbdf-jzp2q  1/1    Running   0          25m     10.244.2.18  desktop-worker2  <none>        <none>
mywebapp-755db8fbdf-kchhs  1/1    Running   0          25m     10.244.1.14  desktop-worker   <none>        <none>
mywebapp-755db8fbdf-mrfz7  1/1    Running   1 (33h ago) 2d1h   10.244.1.3   desktop-worker   <none>        <none>
mywebapp-755db8fbdf-p2gzh  1/1    Running   0          25m     10.244.1.15  desktop-worker   <none>        <none>
mywebapp-755db8fbdf-p7145  1/1    Running   0          25m     10.244.2.17  desktop-worker2  <none>        <none>
mywebapp-755db8fbdf-qcxjn  1/1    Running   0          158m   10.244.1.13  desktop-worker   <none>        <none>
mywebapp-755db8fbdf-t6mv4  1/1    Running   0          3m49s   10.244.1.16  desktop-worker   <none>        <none>
mywebapp-755db8fbdf-vblng  1/1    Running   0          38m     10.244.2.16  desktop-worker2  <none>        <none>
mywebapp-755db8fbdf-vngjq  1/1    Running   1 (33h ago) 2d2h   10.244.2.4   desktop-worker2  <none>        <none>
mywebapp-755db8fbdf-xc48t  1/1    Running   0          3m49s   10.244.1.18  desktop-worker   <none>        <none>
mywebapp-755db8fbdf-z66xc  1/1    Running   0          3m49s   10.244.1.17  desktop-worker   <none>        <none>
```



```
  1 Run cd deploy
  2
  3 NAME          READY   STATUS    RESTARTS   AGE
  4 mywebapp-755db8fbdf-7bmt2  1/1    Running   0          21m
  5 mywebapp-755db8fbdf-jzp2q  1/1    Running   0          21m
  6 mywebapp-755db8fbdf-kchhs  1/1    Running   0          21m
  7 mywebapp-755db8fbdf-mrfz7  1/1    Running   1 (33h ago) 2d1h
  8 mywebapp-755db8fbdf-p2gzh  1/1    Running   0          21m
  9 mywebapp-755db8fbdf-p7145  1/1    Running   0          154m
 10 mywebapp-755db8fbdf-qcxjn  1/1    Running   0          22
 11 mywebapp-755db8fbdf-t6mv4  1/1    Running   0          34m
 12 mywebapp-755db8fbdf-vblng  1/1    Running   0          2d2h
 13 mywebapp-755db8fbdf-vngjq  1/1    Running   1 (33h ago) 2d2h
 14 mywebapp-755db8fbdf-xc48t  1/1    Running   0          25
 15 mywebapp-755db8fbdf-z66xc  1/1    Running   0          25
```

## Testing Pod Tolerations

## Scaling down deployment to 4 replica and applying tolerations



The screenshot shows the Visual Studio Code interface with the title bar "kube-argo-cd-1". The left sidebar displays a file tree for a project named "KUBE-ARGO-CD-1". The main editor area contains a YAML file named "myapp.yaml" which defines a Kubernetes deployment for an application named "myapp". The deployment has 4 replicas and includes tolerations for nodes where the key is "key" and the value is "value". The container uses an image named "ghs-nginx-app42" and exposes port 80. Resource requests and limits are set to 250Mi CPU and 256Mi memory.

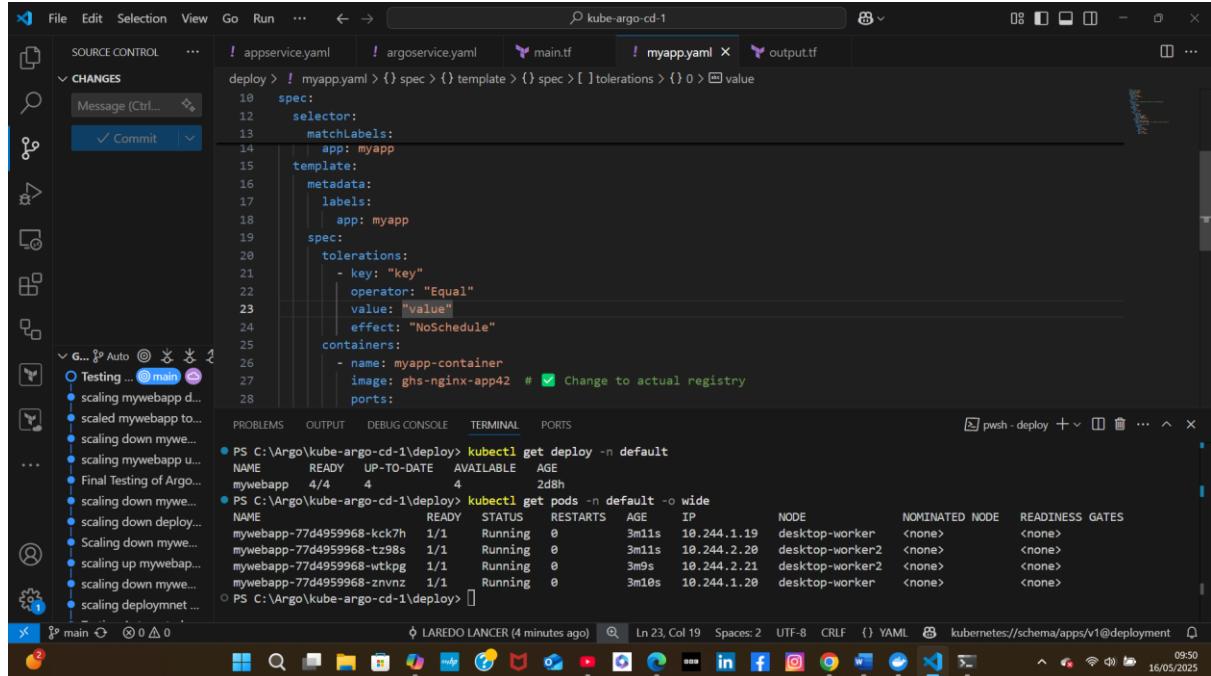
```
version: '1'
services:
  myapp:
    image: ghs-nginx-app42
    ports:
      - 80:80
    resources:
      requests:
        cpu: "250m"
        memory: "256Mi"
      limits:
        cpu: "500m"
```

## Pipeline Success

The screenshot shows a GitHub Actions job run for the repository 'Oreire / kube-argo-cd'. The job, named 'deploy', has succeeded. The job details are as follows:

- Run details:** Succeeded now in 1m 21s.
- Job Steps:**
  - Set up job
  - Checkout repository
  - Verify Kubernetes Context
  - Apply ArgoCD Service Manifest from Local File
  - Apply ArgoCD Application Manifest on Local Cluster
  - Apply MyApp Manifest on Local Cluster
  - Apply MyApp Service Manifest
  - Wait for MyApp Deployment to be Ready
  - Verify MyApp Deployment
- Logs:** A table showing log entries for each step. For example, the 'Verify MyApp Deployment' step shows the command '▶ Run cd deploy' and the output of the 'kubectl get pods' command, which lists 15 pods including 'mywebapp' and 'kubernetes'.

## Two (2) Mywebapp pod with tolerations deployed on the tainted Desktop-worker2 Node.



The screenshot shows the Visual Studio Code interface with the following details:

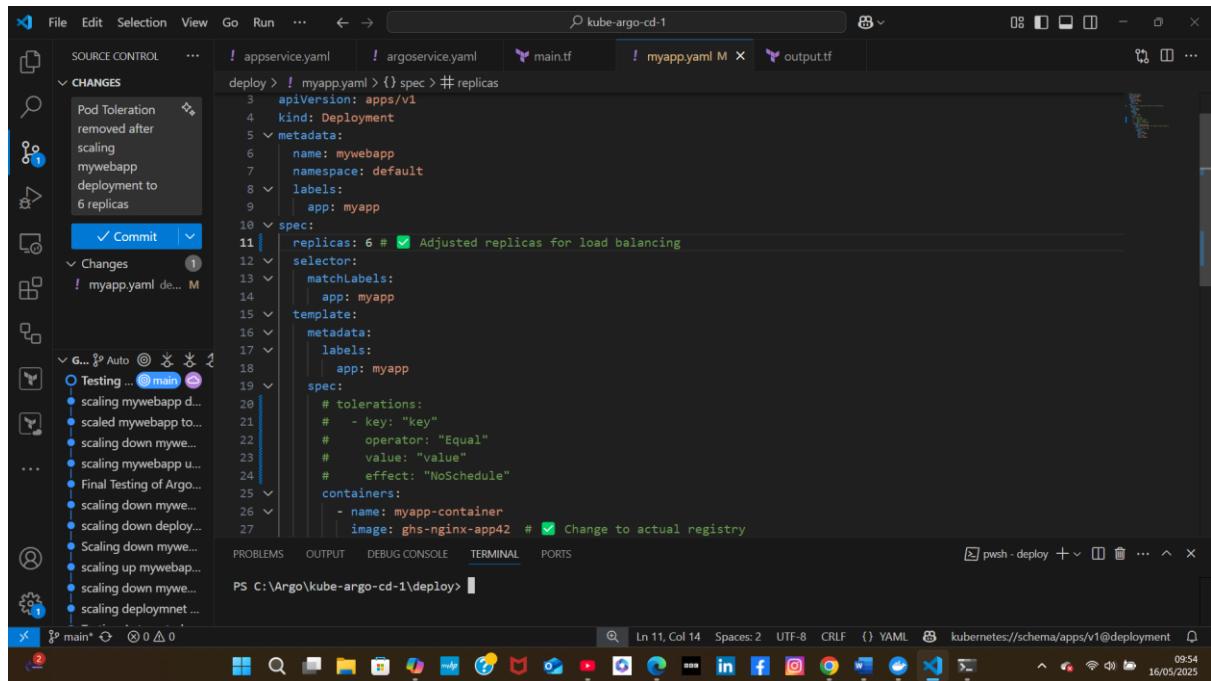
- SOURCE CONTROL:** Changes made to myapp.yaml, including the addition of tolerations.
- TERMINAL:** Command history showing deployment creation and pod listing.
- OUTPUT:** Shows deployment status and pod logs.
- PROBLEMS:** No issues found.
- DEBUG CONSOLE:** Not used.
- PORTS:** Not used.
- COMMANDS:** psh - deploy

```

10 spec:
11   selector:
12     matchLabels:
13       app: myapp
14   template:
15     metadata:
16       labels:
17         app: myapp
18     spec:
19       tolerations:
20         - key: "key"
21           operator: "Equal"
22           value: "value"
23           effect: "NoSchedule"
24     containers:
25       - name: myapp-container
26         image: ghs-nginx-app42 # Change to actual registry
27
28
  ● PS C:\Argo\kube-argo-cd-1\deploy> kubectl get deploy -n default
  NAME READY UP-TO-DATE AVAILABLE AGE
  mywebapp 4/4 4 4 2d8h
  ● PS C:\Argo\kube-argo-cd-1\deploy> kubectl get pods -n default -o wide
  NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
  mywebapp-77d4959968-kcl7h 1/1 Running 0 3m11s 10.244.1.19 desktop-worker <none> <none>
  mywebapp-77d4959968-tz98s 1/1 Running 0 3m11s 10.244.2.20 desktop-worker2 <none> <none>
  mywebapp-77d4959968-vtkpg 1/1 Running 0 3m9s 10.244.2.21 desktop-worker2 <none> <none>
  mywebapp-77d4959968-znvz 1/1 Running 0 3m10s 10.244.1.20 desktop-worker <none> <none>
  ○ PS C:\Argo\kube-argo-cd-1\deploy> []

```

## Removing Tolerations and scaling up deploy to 6 replicas.



The screenshot shows the Visual Studio Code interface with the following details:

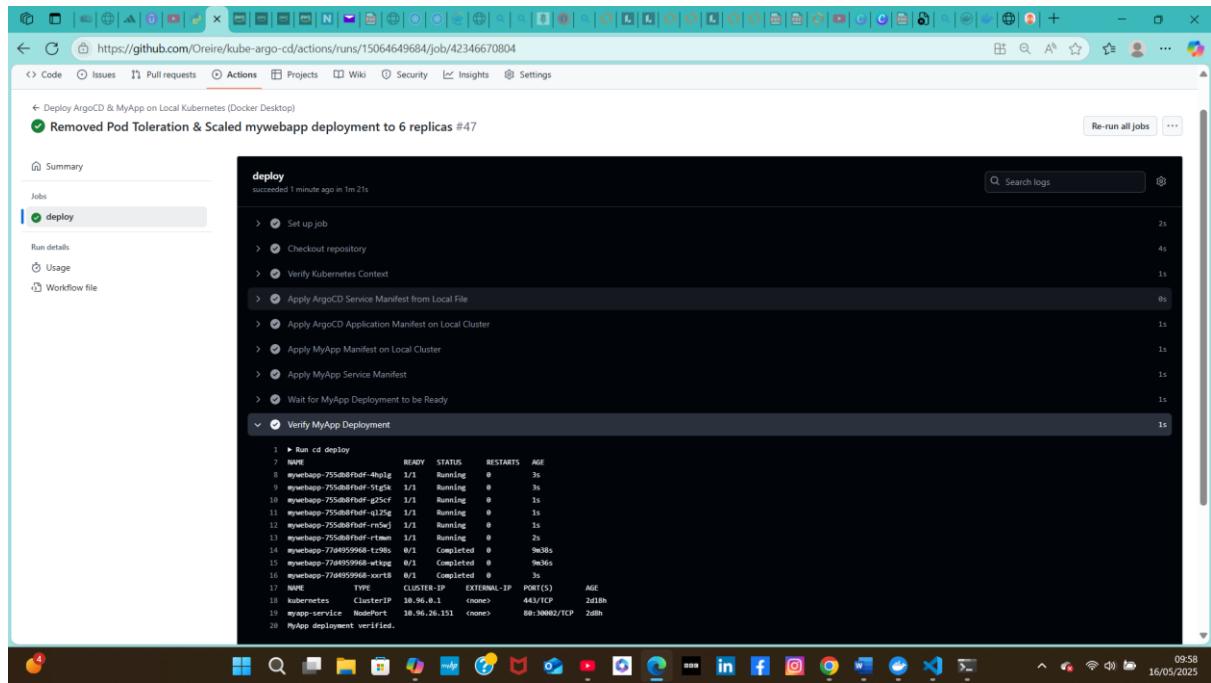
- SOURCE CONTROL:** Changes made to myapp.yaml, including the removal of tolerations and setting replicas to 6.
- TERMINAL:** Command history showing deployment creation.
- OUTPUT:** Shows deployment status and pod logs.
- PROBLEMS:** No issues found.
- DEBUG CONSOLE:** Not used.
- PORTS:** Not used.
- COMMANDS:** psh - deploy

```

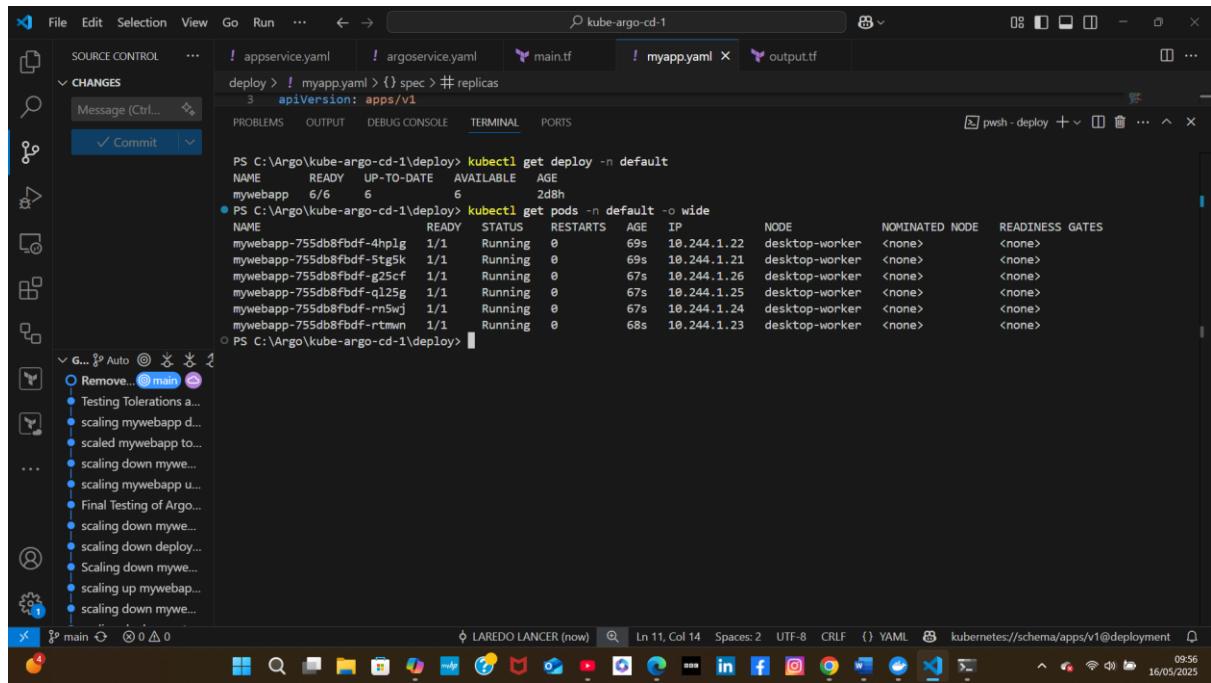
3 apiVersion: apps/v1
4 kind: Deployment
5   metadata:
6     name: mywebapp
7     namespace: default
8     labels:
9       app: myapp
10    spec:
11      replicas: 6 # Adjusted replicas for load balancing
12      selector:
13        matchLabels:
14          app: myapp
15      template:
16        metadata:
17          labels:
18            app: myapp
19        spec:
20          # tolerations:
21          #   - key: "key"
22          #     operator: "Equal"
23          #     value: "value"
24          #     effect: "NoSchedule"
25        containers:
26          - name: myapp-container
27            image: ghs-nginx-app42 # Change to actual registry
28
29
  ○ PS C:\Argo\kube-argo-cd-1\deploy> []

```

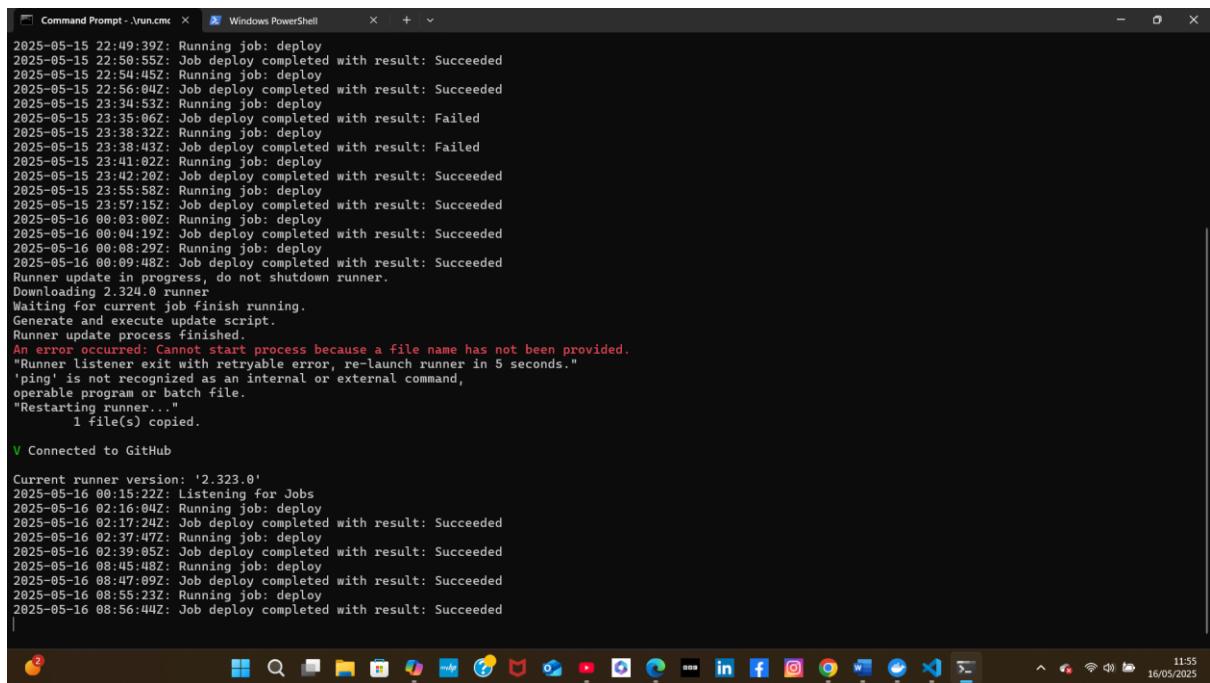
## Pipeline success



All six (6) Mywebapp pod with tolerations deployed on the tainted desktop-worker Node.



## The Local Self-Hosted Runner Portal



A screenshot of a Windows desktop environment. In the center, there is a Command Prompt window titled "Command Prompt - .vrun.cmd" which contains logs of a runner's activity. The logs show multiple job deployments starting at 2025-05-15 22:49:39Z and ending at 2025-05-16 08:56:44Z, with varying results (Succeeded or Failed). It also shows a runner update process starting at 2025-05-16 00:03:08Z, failing due to a command not found ("ping"), and then restarting. The taskbar at the bottom of the screen displays various pinned icons for applications like File Explorer, Edge, and social media platforms.

```
2025-05-15 22:49:39Z: Running job: deploy
2025-05-15 22:50:55Z: Job deploy completed with result: Succeeded
2025-05-15 22:54:45Z: Running job: deploy
2025-05-15 22:56:04Z: Job deploy completed with result: Succeeded
2025-05-15 23:34:53Z: Running job: deploy
2025-05-15 23:35:06Z: Job deploy completed with result: Failed
2025-05-15 23:38:32Z: Running job: deploy
2025-05-15 23:38:43Z: Job deploy completed with result: Failed
2025-05-15 23:41:02Z: Running job: deploy
2025-05-15 23:42:20Z: Job deploy completed with result: Succeeded
2025-05-15 23:55:58Z: Running job: deploy
2025-05-15 23:57:15Z: Job deploy completed with result: Succeeded
2025-05-16 00:03:08Z: Running job: deploy
2025-05-16 00:04:19Z: Job deploy completed with result: Succeeded
2025-05-16 00:08:29Z: Running job: deploy
2025-05-16 00:09:48Z: Job deploy completed with result: Succeeded
Runner update in progress, do not shutdown runner.
Downloading 2.324.0 runner
Waiting for current job finish running.
Generate and execute update script.
Runner update process finished.
An error occurred: Cannot start process because a file name has not been provided.
"Runner listener exit with retryable error, re-launch runner in 5 seconds."
'ping' is not recognized as an internal or external command,
operable program or batch file.
"Restarting runner...""
1 file(s) copied.

V Connected to GitHub

Current runner version: '2.323.0'
2025-05-16 00:15:22Z: Listening for Jobs
2025-05-16 02:16:04Z: Running job: deploy
2025-05-16 02:17:24Z: Job deploy completed with result: Succeeded
2025-05-16 02:37:47Z: Running job: deploy
2025-05-16 02:39:05Z: Job deploy completed with result: Succeeded
2025-05-16 08:45:48Z: Running job: deploy
2025-05-16 08:47:09Z: Job deploy completed with result: Succeeded
2025-05-16 08:55:23Z: Running job: deploy
2025-05-16 08:56:44Z: Job deploy completed with result: Succeeded
```

## Project Highlights

### Managing a Production-Grade Kubernetes Cluster Using Tainting & Tolerations

In a GitOps-driven deployment pipeline for Kubernetes applications—such as the one implemented in this project using ArgoCD and GitHub Actions—ensuring proper node management, workload distribution, and fault tolerance is critical. Therefore, the role of Tainting and Tolerations for was examined to enable a structured approach to node allocation and restrictions of pod scheduling based on predefined constraints to optimize security, high availability, and cluster resilience or fault tolerance thereby emphasizing the need for separation of concerns in microservice architectures.

#### Tainting Nodes

Taints define scheduling constraints, restricting certain pods from running on tainted nodes unless explicitly permitted. Hence, pods bypasses taint restrictions using tolerations. The format is: `kubectl taint nodes <node-name> key=value:<effect>`

The above allows the pod to run on nodes tainted with `key=value:NoSchedule` as tested in this project. The effect can be one of three scenarios as shown:

- **NoSchedule:** Prevents new pods without matching tolerations from scheduling.
- **PreferNoSchedule:** Avoids scheduling new pods but does not enforce strict restrictions.
- **NoExecute:** Evicts existing pods unless they tolerate the taint.

The scaling in or out of the **deployment** while **applying tolerations** to the new deployment achieved several key goals in **workload placement, scheduling control, and resource optimization**. Notably, the project showed how to effectively manage production-grade Kubernetes cluster deployment architectures in a few of many diverse ways such as:

#### 1. Controlled Workload Reduction

By applying **taints to nodes** and **tolerations to pods**, Kubernetes enforces controlled scheduling, ensuring that workloads are distributed efficiently while respecting predefined constraints. This mechanism enables **scaling deployments** to achieve a **balanced redistribution of workloads**, preventing pod overcrowding on nodes and optimizing **cluster resource utilization** for enhanced performance and stability. Therefore, the integrations of tolerations with tainting helps workload isolation from infrastructure services, optimizing resource allocation.

## 2. Validating Toleration-Based Scheduling

Applying tolerations to the new deployment enables Kubernetes to:

- **Reschedule pods exclusively onto tainted nodes (desktop-worker2)** when their tolerations match the applied taint.
- **Prevent pods from being scheduled on restricted nodes**, ensuring workloads adhere to defined toleration policies and avoiding conflicts with existing constraints.
- **Verify that ArgoCD correctly reconciles deployment updates**, maintaining alignment with the GitOps workflow and enforcing declarative infrastructure management.

## 3. Ensuring High Availability & Fault Tolerance

With tolerations correctly configured:

- **Pods remain operational on permitted nodes**, ensuring uninterrupted service availability.
- **New pods are scheduled only on designated nodes**, enforcing controlled workload distribution and preventing unintended placement.
- **The cluster maintains resilience**, allowing automatic failover mechanisms to mitigate node failures and ensure stability.

## 4. Kubernetes Scheduler Testing

This process ensures the **Kubernetes scheduler** effectively manages workload placement by validating:

- **Pod assignment under new toleration rules**, ensuring proper scheduling on tainted nodes.
- **Rescheduling behavior**, confirming that pods are deployed on available nodes while adhering to defined deployment constraints.
- **Reaction to GitOps-driven updates**, verifying that ArgoCD correctly reconciles changes, maintaining consistency between the Git repository and the Kubernetes cluster.

## 5. Infrastructure Optimization

Refining **toleration rules** has enabled:

- **Efficient resource utilization**, ensuring workloads are scheduled on targeted nodes for optimal performance.
- **Proper workload isolation**, preventing deployment drift and maintaining infrastructure consistency.
- **Seamless integration with GitOps workflows**, enforcing declarative control over infrastructure changes to enhance automation and reliability.

## Conclusion

This project established a **GitOps-driven deployment pipeline**, leveraging **Infrastructure-as-Code, ArgoCD, and GitHub Actions** to enhance **automation, security, and reliability**. By maintaining infrastructure declaratively within a **Git repository**, Kubernetes continuously reconciles deployments with the desired state, minimizing manual intervention and configuration drift.

Rigorous testing validated its relevance for **production environments**, focusing on the impact of **taints and tolerations** on workload placement. Scaling deployments with tolerations enabled effective testing of **controlled workload distribution, Kubernetes scheduler behavior, and GitOps-driven deployment integrity**, reinforcing **high availability and fault tolerance**.

Despite its simplicity, this project offers **flexibility for customization**, making it applicable in real-world **production environments**. By streamlining **Kubernetes management**, it optimizes **cluster efficiency, reliability, and automation**, delivering a scalable solution through **ArgoCD and GitHub Actions**.

## Next Steps for Optimizing Project

1. **Implement Role-Based Access Control (RBAC) in ArgoCD for Enhanced Security**  
Strengthen **access control** in ArgoCD by defining **RBAC policies**, restricting permissions based on roles such as **admin, developer, and viewer**. This enhances security by preventing unauthorized modifications and ensuring controlled access to deployed applications.
2. **Using ArgoCD ApplicationSets to Deploy Multiple Environments Dynamically**  
Use **ApplicationSets** to automate multi-environment deployments (**dev, staging, production**). Configure **template-based** application definitions to simplify GitOps workflows, enabling **dynamic application provisioning** across clusters with minimal manual intervention.