

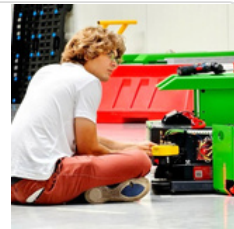
# Kubernetes GitOps: Multi-cluster deployment with ArgoCD

Check GitHub for helpful DevOps tools:

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

 <https://github.com/MichaelRobotics>



Ask Personal AI Document assistant to learn interactively (FASTER)!

1

Download PDF

2

Go to website

3

Browse file

4

Chat with Document

1

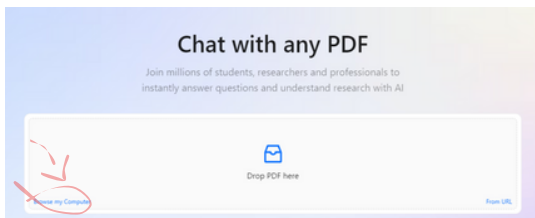
<https://github.com/MichaelRobotics/DevOpsTools/blob/main/KubernetesGitOps.pdf>

2

🔗 | Click there to go to ChatPdf website

⬆

3

The image shows the ChatPdf website interface. It has a light blue header with the text 'Chat with any PDF' and a sub-header 'Join millions of students, researchers and professionals to instantly answer questions and understand research with AI'. Below this is a large white box with a red dashed line and a red circle around the text 'Upload your PDF file'. In the center of the white box is a blue icon of a document with a checkmark and the text 'Drop PDF here'. At the bottom right of the white box is a small link 'From URL'.

4

Ask questions about document!

# Completly new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

 <https://www.hackthebox.com/>



## What is Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of nodes running containers, ensuring efficient and reliable operation.

## How Kubernetes clusters are made?

Kubernetes clusters consist of a control plane and multiple worker nodes. The control plane manages cluster operations, while worker nodes run the actual container workloads.

# Why and When use Kubernetes

Kubernetes is ideal for deploying scalable, resilient, and automated containerized applications. It is used when managing multiple containers across different environments is necessary.

Example: Running a microservices-based e-commerce platform that scales up during peak hours.

## System Requirements

- RAM: 2 GB per node (1 GB can work for testing but may lead to limited performance)
- 10 GB free storage
- Ubuntu

## Kubernetes: Main components & packages

- **kube-apiserver:** Central management component that exposes the Kubernetes API; acts as the front-end for the cluster.
- **etcd:** Distributed key-value store for storing all cluster data, ensuring data consistency across nodes.
- **kube-scheduler:** Assigns pods to available nodes based on resource requirements and policies.
- **kube-controller-manager:** Manages core controllers that handle various functions like node status, replication, and endpoints.
- **kubelet:** Agent that runs on each node, responsible for managing pods and their containers.
- **kube-proxy:** Manages networking on each node, ensuring communication between pods and services within the cluster.

# Kubernetes GitOps: Multi-cluster deployment with ArgoCD

## 1) Intro

Kubernetes GitOps with ArgoCD simplifies multi-cluster deployments by using Git as the single source of truth for infrastructure. This approach ensures consistent, automated, and scalable management of Kubernetes environments, aligning with modern DevOps practices.

To embrace GitOps in company, your configuration must follow those 4 pillars:

### 1. Declarative

Kubernetes GitOps with ArgoCD enables multi-cluster deployments by defining infrastructure and application configurations declaratively, ensuring clear and predictable desired states.

### 2. Versioned and Immutable

Infrastructure states are stored in Git repositories, ensuring immutability, versioning, and a complete history for reliable rollbacks and audits.

### 3. Pulled Automatically

ArgoCD automatically syncs the desired state from Git repositories to Kubernetes clusters, reducing manual intervention.

### 4. Continuously Reconciled

ArgoCD continuously monitors and reconciles the actual cluster state with the desired state, maintaining consistency across environments.

### 3) Argo Configuration models

#### Hub-Spoke Model - Used in this demo project

The hub-spoke model is centralized, best for ensuring consistency and enforcing global policies across multiple clusters. It is ideal for organizations needing centralized governance, simplified management, and shared configurations.

#### Standalone Model

The standalone model is decentralized, best for providing flexibility and autonomy to individual teams or regions. It works well in isolated environments or when clusters need to operate independently without relying on a central hub.

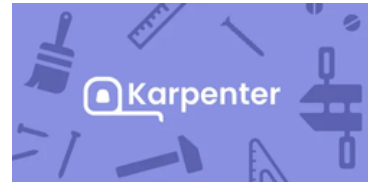
### 2) Cluster and Argo Setup

Install eksctl, aws and other tools, specified in this Karpenter tutorial.

Getting Started with Karpenter

Set up a cluster and add Karpenter

 <https://karpenter.sh/docs/>



You can install as many clusters as needed. I'll set up three: the first with ArgoCD, and the second and third for other purposes. If you want a more robust solution, it's recommended to deploy clusters using an Infrastructure-as-Code (IaC) tool like Terraform. However, for the sake of efficiency, the infrastructure will be defined manually in this case.

```
eksctl create cluster --name hub-cluster --region us-west-1
```

```
eksctl create cluster --name spoke-cluster-1 --region us-west-1
```

```
eksctl create cluster --name spoke-cluster-2 --region us-west-1
```

install argo components on a hub-cluster

```
kubectl create namespace argocd
```

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

For the sake of simplifying configuration management, we will disable TLS. However, this should not be done in production environments.

```
kubectl get cm -n argocd
```

```
laptopdev@laptopdev2:~$ kubectl get cm -n argocd
NAME                                DATA  AGE
argocd-cm                          0      63s
argocd-cmd-params-cm               0      62s
argocd-gpg-keys-cm                 0      61s
argocd-notifications-cm           0      61s
argocd-rbac-cm                     0      60s
argocd-ssh-known-hosts-cm         1      59s
argocd-tls-certs-cm               0      58s
kube-root-ca.crt                   1      88s
```

For sake of speed, we will not create ingress, but just change service type of ArgoCD to nodeport

```
kubectl edit configmap argocd-cmd-params-cm -n argocd
```

```
name: argocd-cmd-params-cm
namespace: argocd
resourceVersion: "24117"
uid: d5beff2b-0015-4485-8ff1-414d556d7dc7
data:
  server.insecure: "true"
```

In AWS, we will make this address accessible from our IP. However, remember to use TLS, ingress, and other security features in real-world implementations to ensure proper security.

## Change ArgoCD server service to NodePort

```
kubectl get svc -n argocd
```

```
laptopdev@laptopdev2:~$ kubectl get svc -n argocd
NAME                                TYPE               CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
argocd-applicationset-controller    ClusterIP          10.100.30.34    <none>           7000/TCP,8080/TCP 131m
argocd-dex-server                   ClusterIP          10.100.101.83   <none>           5556/TCP,5557/TCP,5558/TCP 131m
argocd-metrics                      ClusterIP          10.100.98.178   <none>           8082/TCP          131m
argocd-notifications-controller-metrics ClusterIP          10.100.88.182   <none>           9001/TCP          131m
argocd-redis                        ClusterIP          10.100.187.8    <none>           6379/TCP          131m
argocd-repo-server                  ClusterIP          10.100.94.40    <none>           8081/TCP,8084/TCP 131m
argocd-server                       NodePort           10.100.164.228   <none>           80:32467/TCP,443:30499/TCP 131m
argocd-server-metrics               ClusterIP          10.100.27.71    <none>           8083/TCP          131m
```

```
kubectl edit svc argocd-server -n argocd
```

```
protocol: TCP
targetPort: 8080
selector:
  app.kubernetes.io/name: argocd-server
sessionAffinity: None
type: NodePort
```

got to AWS, EC2, modify security group, so it will be reachable from our ip address

**Edit inbound rules** [Info](#)

Inbound rules control the incoming traffic that's allowed to reach the instance.

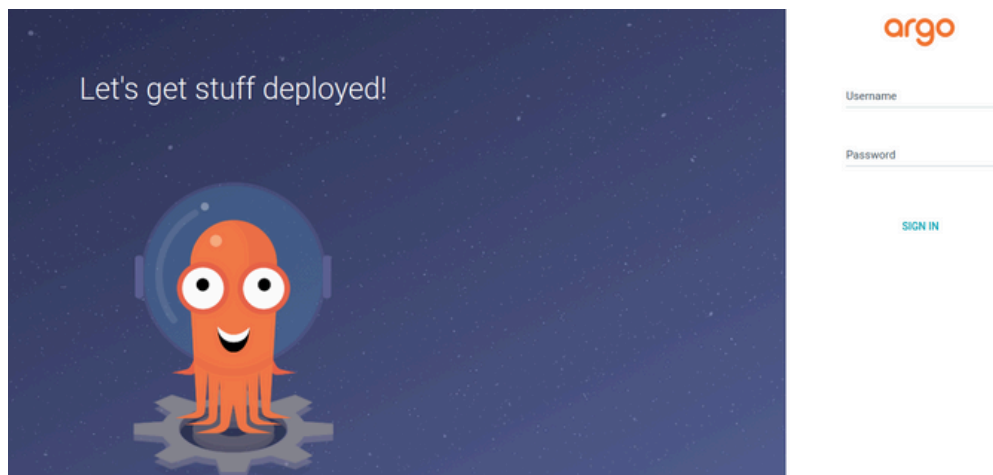
Security group rule ID	Type	Protocol	Port range	Source	Description - optional	
sg-05f2c846ba3a0b160	All traffic	All	All	Custom	Allow unmanaged nodes to comm	Delete
sg-06a526afeaa760fa0	All traffic	All	All	Custom	Allow unmanaged nodes to communicate with control plane (all ports)	Delete
-	All traffic	All	All	Custom		Delete

[Add rule](#)

Custom ☒ Anywhere-IPv4 Anywhere-IPv6 My IP

[Cancel](#) [Preview changes](#) [Save rules](#)

Get ip address of any node and access it with web browser: `http://<node_ip>:<argocd-server port>`



### 3) Configure hub-cluster model

ArgoCD have great UI. Get password from secret stored on cluster.

```
kubectl get secrets -n argocd
```

```
kubectl describe secret argocd-initial-admin-secret -n argocd
```

```
laptopdev@laptopdev2:~$ kubectl get secrets -n argocd
NAME                                TYPE      DATA      AGE
argocd-initial-admin-secret         Opaque    1           142m
argocd-notifications-secret         Opaque    0           143m
argocd-redis                        Opaque    1           142m
argocd-secret                       Opaque    5           143m
laptopdev@laptopdev2:~$ kubectl describe secret argocd-initial-admin-secret -n argocd
Name:         argocd-initial-admin-secret
Namespace:    argocd
Labels:       <none>
Annotations:  <none>

Type: Opaque

Data
====
password: 16 bytes
```

Now copy key and decode it

```
kubectl edit secret argocd-initial-admin-secret -n argocd
```

```
#
apiVersion: v1
data:
  password: aXlQTjFUak9yc2ZKdkJ2WA==
kind: Secret
metadata:
  creationTimestamp: "2025-01-14T19:17:48Z"
  name: argocd-initial-admin-secret
  namespace: argocd
  resourceVersion: "24380"
  uid: eeaabe42-9be0-4e6e-b3ef-8e0a26c628c1
type: Opaque
```

```
echo aXlQTjFUak9yc2ZKdkJ2WA== | base64 --decode
```

```
laptopdev@laptopdev2:~$ kubectl edit secret argocd-initial-admin-secret -n argocd
Edit cancelled, no changes made.
laptopdev@laptopdev2:~$ echo aXlQTjFUak9yc2ZKdkJ2WA== | base64 --decode
iyPN1Tj0rsfJvBvXlaptopdev@laptopdev2:~$
```

Login to ArgoCD using Username:admin Password:<your password>



At the end, we need to add our clusters using CLI. First, install it:

```
curl -sSL -o argocd-linux-amd64 https://github.com/argoproj/argo-  
cd/releases/latest/download/argocd-linux-amd64  
sudo install -m 555 argocd-linux-amd64 /usr/local/bin/argocd  
rm argocd-linux-amd64
```

login in CLI to ArgoCD

```
laptopdev@laptopdev2:~$ argocd login 13.57.254.131:32467  
WARNING: server certificate had error: tls: failed to verify certificate  
Proceed insecurely (y/n)? y  
Username: admin  
Password:  
'admin:login' logged in successfully  
Context '13.57.254.131:32467' updated  
laptopdev@laptopdev2:~$
```

kubectl config get-contexts

```
laptopdev@laptopdev2:~$ kubectl config get-contexts  
CURRENT  NAME                                     CLUSTER  
NAMESPACE  
Admin@hub-cluster.us-west-1.eksctl.io     hub-cluster.us-west-1.eksctl.io  
Admin@spoke-cluster-1.us-west-1.eksctl.io spoke-cluster-1.us-west-1.eksctl.io  
* Admin@spoke-cluster-2.us-west-1.eksctl.io spoke-cluster-2.us-west-1.eksctl.io  
arn:aws:eks:us-east-1:533267337200:cluster/demo arn:aws:eks:us-east-1:533267337200:cluster/demo  
arn:aws:eks:us-east-2:533267337200:cluster/demo2 arn:aws:eks:us-east-2:533267337200:cluster/demo2
```

Then get contexts of your clusters and add them to ArgoCD

```
laptopdev@laptopdev2:~$ argocd cluster add Admin@hub-cluster.us-west-1.eksctl.io --server 13.57.254.131:32467  
WARNING: This will create a service account 'argocd-manager' on the cluster referenced by context 'Admin@hub-cluster.us-west-1.eksctl.io'.  
Do you want to continue [y/N]? y  
INFO[0002] ServiceAccount "argocd-manager" already exists in namespace "kube-system"  
INFO[0003] ClusterRole "argocd-manager-role" updated  
INFO[0003] ClusterRoleBinding "argocd-manager-role-binding" updated  
Cluster 'https://A2A3F7E8E6F7E57937D1FB16747D905B.gr7.us-west-1.eks.amazonaws.com' added  
laptopdev@laptopdev2:~$ argocd cluster add Admin@spoke-cluster-1.us-west-1.eksctl.io --server 13.57.254.131:32467  
WARNING: This will create a service account 'argocd-manager' on the cluster referenced by context 'Admin@spoke-cluster-1.us-west-1.eksctl.io'.  
Do you want to continue [y/N]? y  
INFO[0002] ServiceAccount "argocd-manager" already exists in namespace "kube-system"  
INFO[0003] ClusterRole "argocd-manager-role" updated  
INFO[0003] ClusterRoleBinding "argocd-manager-role-binding" updated  
Cluster 'https://AB8E455E69FAFC744A3799AB55D2D4CC.sk1.us-west-1.eks.amazonaws.com' added
```

in my case:

```
argocd cluster add Admin@spoke-cluster-1.us-west-1.eksctl.io --server 13.57.254.131:32467  
argocd cluster add Admin@hub-cluster.us-west-1.eksctl.io --server 13.57.254.131:32467
```

Create git repo, our single source of truth. Remember to make it public

```
git clone https://github.com/MichaelRobotics/Kubernetes.git
cd Kubernetes/ArgoCD
```

Then connect each cluster with git repo through ArgoCD. Go to NEW APP -> CREATE. Setup Argo application name and sync policy. Repeat for each cluster

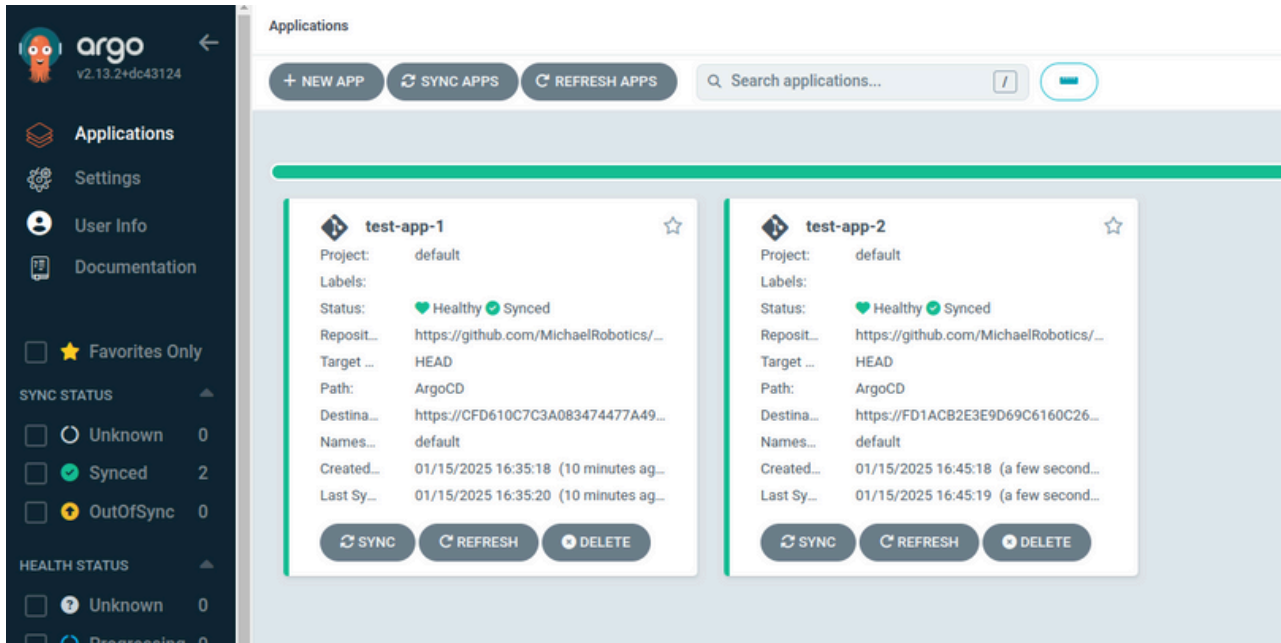
The screenshot shows the 'CREATE' form for a new ArgoCD application. The 'GENERAL' section is active, showing 'Application Name' as 'test-app-2' and 'Project Name' as 'default'. The 'SYNC POLICY' section is also visible, with 'Automatic' selected. There are checkboxes for 'PRUNE RESOURCES' and 'SELF HEAL', both of which are currently unchecked. On the left side of the form, there is a sidebar with a list of application details: Project, Labels, Status, Repository, Target, Path, Destination, Namespaces, Created, and Last Sync. A 'SYNC' button is located at the bottom of this sidebar.

Then Github folder path and path to folder with your manifests, chose Cluster URL from bar with desired k8s namespace

The screenshot shows the 'SOURCE' and 'DESTINATION' sections of the ArgoCD application form. The 'SOURCE' section includes fields for 'Repository URL' (https://github.com/MichaelRobotics/Kubernetes/), 'Revision' (HEAD), and 'Path' (ArgoCD). The 'DESTINATION' section includes fields for 'Cluster URL' (https://FD1ACB2E3E9D69C6160C26131A35B625.gr7.us-west-1.eks.amazonaws.com) and 'Namespace' (default). Both sections have dropdown menus for selecting the repository type (GIT) and the cluster URL (URL).

## 4) Test GitOps implementation

The First and Second pillars of GitOps have been implemented. Our Kubernetes application is defined in YAML files stored in a remote repository, with the application's state managed declaratively.



Now, let's verify if a change in our Git repository affects the cluster, confirming the implementation of the Third Pillar of GitOps:

Switch to the context of one of your spoke clusters. If Argo is functioning correctly, it should automatically deploy all files from the GitHub repository to these clusters.

```
kubectl config get-contexts
kubectl config use-context Admin@spoke-cluster-1.us-west-1.eksctl.io
kubectl get configmap
```

```
laptopdev@laptopdev2:~$ kubectl config use-context Admin@spoke-cluster-1.us-west-1.eksctl.io
Switched to context "Admin@spoke-cluster-1.us-west-1.eksctl.io".
laptopdev@laptopdev2:~$ kubectl get configmap
NAME          DATA   AGE
guest-book    1       14m
kube-root-ca.crt 1       48m
laptopdev@laptopdev2:~$
```

The ConfigMap is present, indicating that the manifests from the Git repository were successfully deployed. Now, verify its values, review its YAML in the Git repository, and confirm that changes in the repository are applied to our clusters.

```
apiVersion: v1
items:
- apiVersion: v1
  data:
    ui_properties_file_name: abhishek-interface.properties
  kind: ConfigMap
  metadata:
```

Change parameter `ui_properties_file_name`: to “`user_interface.properties`”

```
Code Blame 6 lines (6 loc) · 121 Bytes Code 55% faster with GitHub Copilot

1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: guest-book
5  data:
6    ui_properties_file_name: "user_interface.properties"
```

Wait a few minutes, then check the ConfigMap again.

```
apiVersion: v1
items:
- apiVersion: v1
  data:
    ui_properties_file_name: user_interface.properties
  kind: ConfigMap
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
```

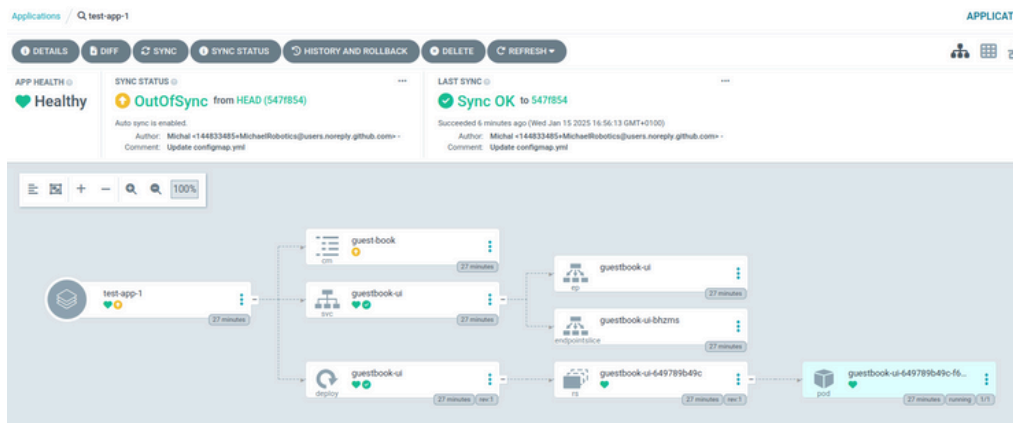
There it is! The change has been successfully applied to the Kubernetes cluster.

Now, test the Fourth Rule: ArgoCD should automatically monitor changes in our cluster. If there is any drift from the Git repository configuration, it should automatically delete or prevent unauthorized changes and notify us of the desynchronization with the Git repository.

edit configmap:

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
items:
- apiVersion: v1
  data:
    ui_properties_file_name: abhishek-interface.properties
  kind: ConfigMap
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
```

Check ArgoCD UI:

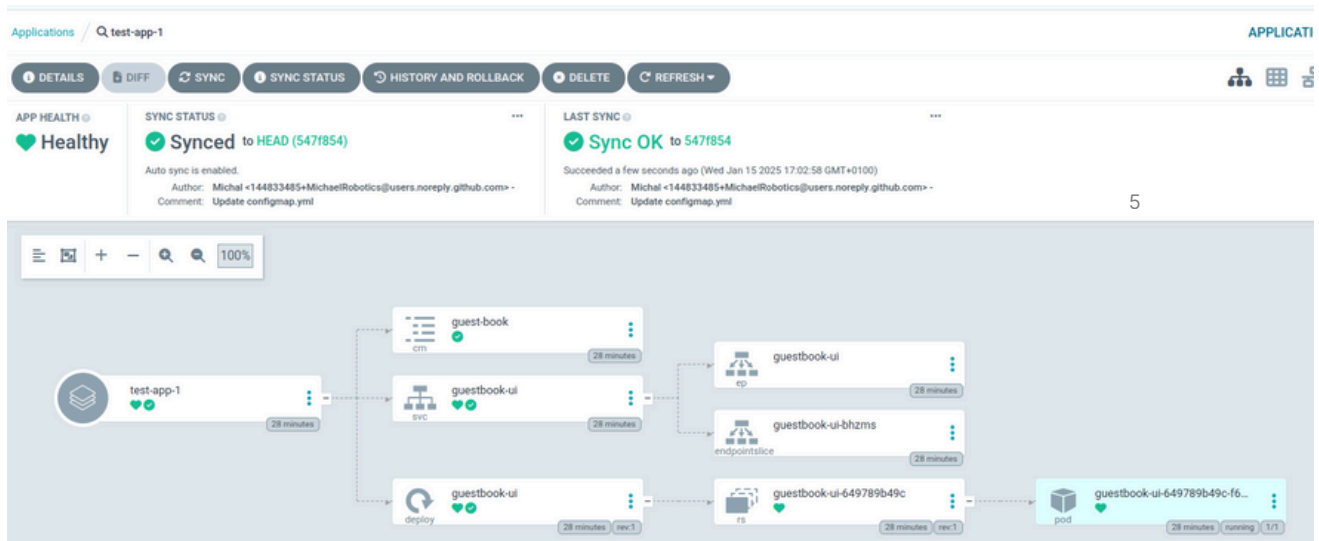


ArgoCD app which points to cluster where change was applied, communicates about OutOfSync issue.

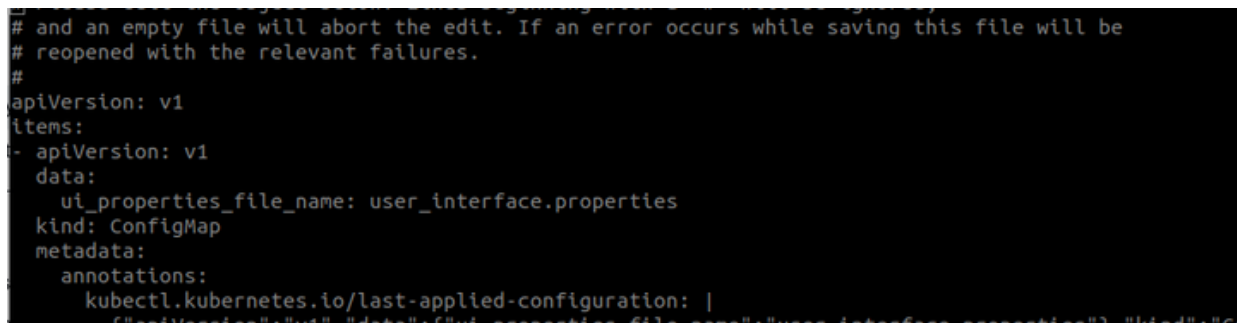
Go to SYNC->SYNCHRONIZE



Wait a moment, ArgoCD should revoke all changes applied to cluster:



It works! Cluster configmap got backed-up to previous state!



There's a lot to configure! You can set up automatic synchronization, ArgoCD application generation, and much more. Dive in and start experimenting!

# common troubleshooting

## 1) Application Deployment Fails Across Clusters

**Cause:** Incorrect configuration in the Git repository or missing permissions for cluster contexts.

**Solution:** Verify ArgoCD's Application manifests and cluster configurations in the Git repository. Ensure the cluster roles and permissions are correctly set using `kubectl describe clusterrolebinding`.

## 2) Synchronization Errors with ArgoCD

**Cause:** ArgoCD fails to sync due to manifest errors or network issues.

**Solution:** Check ArgoCD logs for sync errors using `kubectl logs -n argocd`. Validate the YAML files in the Git repository for proper formatting and completeness.

## 3) Cluster Drift Detected

**Cause:** Manual changes in cluster configurations not aligned with the Git repository.

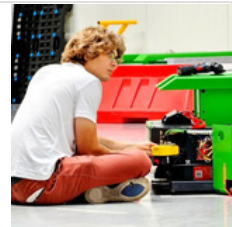
**Solution:** Review ArgoCD's drift detection logs. Revert unauthorized changes by triggering a manual sync from the ArgoCD dashboard or CLI.

## 4) Check my Kubernetes Troubleshooting series:

### Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

 <https://github.com/MichaelRobotics>




## Learn more about Kubernetes

**Check Kubernetes and piyushsachdeva - great docs!**

Setup a Multi Node Kubernetes Cluster

kubeadm is a tool to bootstrap the Kubernetes cluster

 <https://github.com/piyushsachdeva/CKA-2024/tree/main/Resources/Day27>



Kubernetes Documentation

This section lists the different ways to set up and run Kubernetes

 <https://kubernetes.io/docs/setup/>



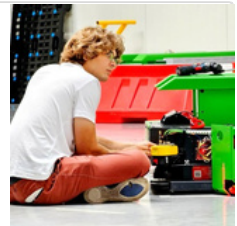
**Share, comment, DM and check GitHub for scripts & playbooks created to automate process.**

**Check my GitHub**

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

<https://github.com/MichaelRobotics>



*PS.*

*If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!*