

CI CD PROJECT

OBJECTIVE:

The primary objective of this project is to establish a robust CI/CD pipeline that automates the process of building, testing, and deploying applications by integrating Git, Jenkins, Maven, SonarQube, Docker Hub, ArgoCD and Kubernetes (EKS) , the project aims to achieve the following goals:

1. Continuous Integration (CI):

- Automate the process of code integration, ensuring that new code changes are seamlessly merged with the existing codebase.
- Perform automated code quality checks and static analysis using SonarQube.

2. Continuous Deployment (CD):

- Automate the deployment of applications to a Kubernetes cluster.
- Utilize Docker for creating and managing containerized applications, facilitating consistent deployment environments across different stages (development, testing, and production).
- Implement a streamlined workflow for updating deployment configurations and pushing them to a Git repository for version control.

SUMMARY:

This end-to-end Jenkins pipeline will automate the entire CI/CD process for a Java application, from code checkout to production deployment, using popular devops tools like Git, Jenkins, Maven, SonarQube, Docker Hub, ArgoCD and Kubernetes

This project covers setting up a CI/CD pipeline using.

- Installing and configuring Jenkins on a Linux server.
- Setting up Docker, configuring Jenkins to use Docker, and handling permissions.
- Configuring AWS CLI, installing kubectl and eksctl.
- Creating an EKS cluster with eksctl.
- Installing necessary Jenkins plugins and setting up credentials.
- Creating a Jenkins pipeline to:
 - Checkout code from GitHub.
 - Perform a SonarQube scan.
 - Build the project using Maven.
 - Build Docker Image using Docker file and push Docker images to Docker Hub.

- Update deployment configurations and push them to a Git repository.
- Deploy the application to Kubernetes using ArgoCD.

Pre-requisites

1. Java application code hosted on a Git repository

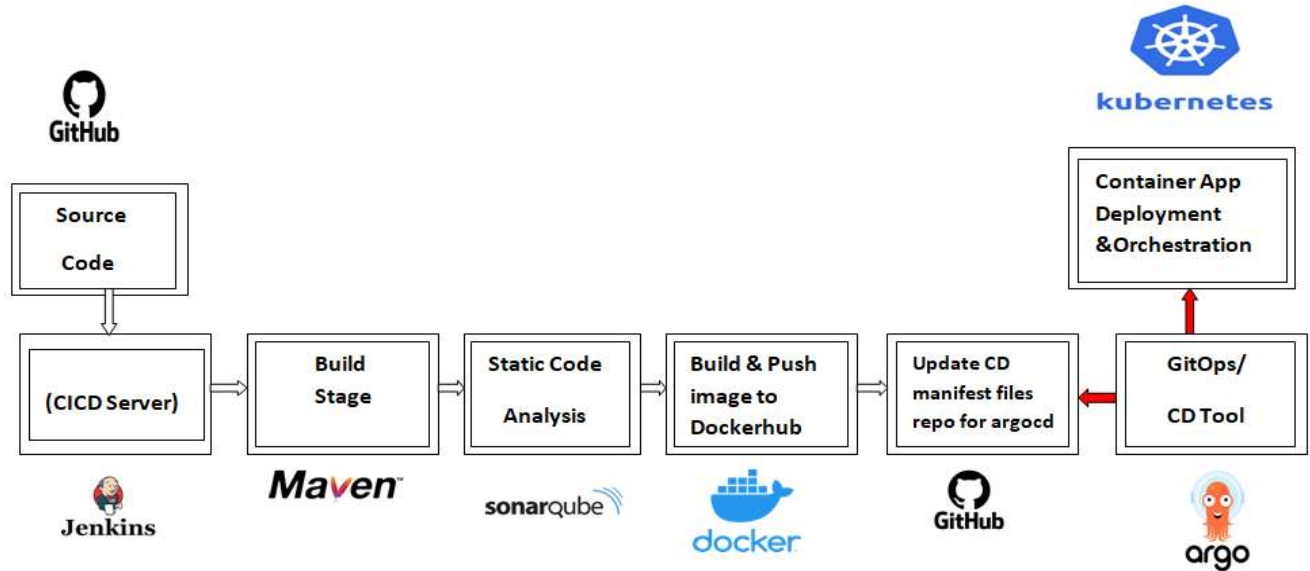
2. Server Requirements:

- A Linux server for Jenkins , Docker, Sonarqube setup with an instance type “t2.large” and a EKS Cluster to deploy container application.

3. Software Requirements:

- Jenkins
- Docker
- Maven
- SonarQube
- Kubernetes CLI (kubectl) , eksctl for managing EKS clusters
- EKS Kubernetes cluster
- Argo CD

FLOW CHART:



1. Jenkins Installation

First, install Jenkins on your Linux system:

```

sudo yum update -y

sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo

sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

sudo yum upgrade -y

sudo dnf install java-17-amazon-corretto -y

sudo yum install jenkins -y

sudo systemctl enable jenkins

sudo systemctl start jenkins

```

2. Docker Installation and Configuration

Install Docker and configure it to work with Jenkins:

```

sudo yum install docker -y

sudo systemctl start docker

```

```
sudo usermod -aG docker jenkins  
  
sudo usermod -aG docker ec2-user  
  
sudo systemctl restart docker  
  
sudo chmod 666 /var/run/docker.sock
```

3. AWS CLI Configuration

Configure AWS CLI to interact with AWS services:

```
aws configure  
  
# Enter your AWS  
  
Access Key ID, Secret Access Key, region, and output format when prompted
```

4. Kubectl and Eksctl Installation

Install **kubectl** and **eksctl** to manage Kubernetes clusters:

Kubectl doc: <https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>

Eksctl doc: <https://docs.aws.amazon.com/emr/latest/EMR-on-EKS-DevelopmentGuide/setting-up-eksctl.html>

```
# Install kubectl  
  
curl -LO "https://dl.k8s.io/release/$(curl -L -s  
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"  
  
curl -LO "https://dl.k8s.io/release/$(curl -L -s  
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"  
  
echo "$(cat kubectl.sha256) kubectl" | sha256sum --check  
  
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl  
  
# Install eksctl  
  
curl --silent --location  
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -  
s)_amd64.tar.gz" | tar xz -C /tmp
```

```
sudo mv /tmp/eksctl /usr/local/bin
```

5. Create EKS Cluster

Create an EKS cluster using **eksctl**:

```
eksctl create cluster --name mcappcluster --nodegroup-name mcng --node-type t3.micro --nodes 8 --managed
```

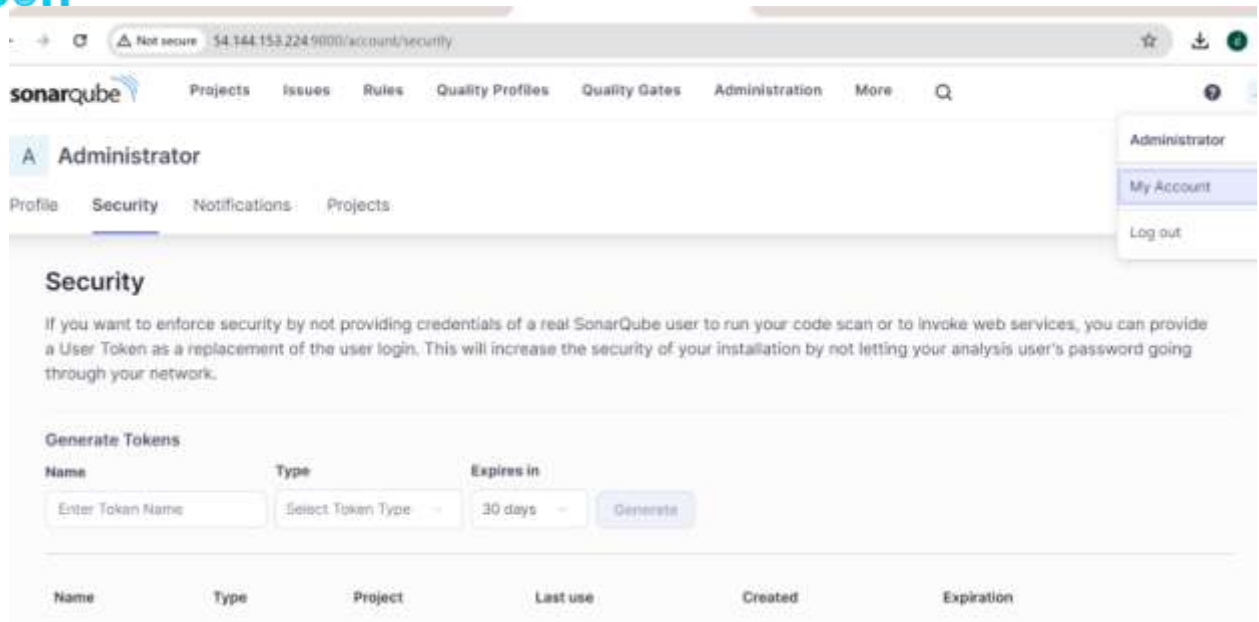
6. Install Required Jenkins Plugins

Install the Docker plugin in Jenkins:

1. Go to Jenkins Dashboard > Manage Jenkins > Manage Plugins > Available.
2. Search for "Docker" and install the Docker plugin.

7. Install and configure Sonarqube as a Docker container

1. Docker run `--itd --name sonar -p 9000:9000 sonarqube`
2. Check if the SonarQube container is running:
docker ps
3. Access the SonarQube web interface:
 - Open a web browser and go to `http://<your-server-ip>:9000`
 - The default login credentials are:
 1. **Username: admin**
 2. **Password: admin**
4. Log in to SonarQube using the default credentials.
 - Change the default password
5. Create Sonar token for Jenkins:
Sonar Dashboard -> Administration -> MyAccount -> Security -> Create token



8. Generate GitHub Token

Generate a GitHub token for Jenkins to access your GIT repositories:

1. Go to GitHub > Settings > Developer settings > Personal access tokens.
2. Generate a new token with the necessary scopes (e.g., **repo**, **admin:repo_hook**).

9. Create Jenkinsfile for CI/CD Pipeline

Create a **Jenkinsfile** in your repository to define the CI/CD pipeline:

```
pipeline {
  agent any
  // Specify the maven installation to use for this pipeline.
  // 'maven3' refers to a Maven installation configured in Jenkins.

  tools {
    maven 'maven3'
  }

  stages {

    stage('Checkout') {
      steps {
        echo 'Cloning GIT HUB Repo'
```

```
// Clone the specified branch from the GitHub repository
git branch: 'main', url: 'https://github.com/devopstraininghub/mindcircuit13.git'
}
}

stage('SonarQube Scan') {
  steps {
    echo 'Scanning project'
    // List directory contents for debugging purposes
    sh 'ls -ltr'
    // Run SonarQube scan with specified SonarQube server and login token
    sh ''' mvn sonar:sonar \\\
      -Dsonar.host.url=http://100.26.227.191:9000 \\\
      -Dsonar.login=squ_19733ad4e43d54992ef61923b91447e2d17a3062'''
  }
}

stage('Build Artifact') {
  steps {
    echo 'Build Artifact'
    // Clean and package the project using Maven
    sh 'mvn clean package'
  }
}

stage('Build Docker Image') {
  steps {
    echo 'Build Docker Image'
    // Build the Docker image using the Dockerfile in the project
    // Tag the image with the current build number
    sh 'docker build -t devopshubg333/batch13:${BUILD_NUMBER} .'
  }
}

stage('Push to Docker Hub') {
  steps {
    script {
      // Use Dockerhub credentials to access Docker Hub
      withCredentials([string(credentialsId: 'dockerhub', variable: 'dockerhub')]) {
        sh 'docker login -u devopshubg333 -p ${dockerhub}'
      }
    }
  }
}
```

```

    }
    // Push the Docker image to Docker Hub
    sh 'docker push devopshubg333/batch13:${BUILD_NUMBER}'
    echo 'Pushed to Docker Hub'
  }
}
}

stage('Update Deployment File') {
  environment {
    GIT_REPO_NAME = "mindcircuit13"
    GIT_USER_NAME = "devopstraininghub"
  }
  steps {
    echo 'Update Deployment File'
    // Use Github token to access GitHub
    withCredentials([string(credentialsId: 'githubtoken', variable: 'githubtoken')]) {
      sh '''
        # Configure git user
        git config user.email "madhuxxxx123@gmail.com"
        git config user.name "Madhu"
        # Replace the tag in the deployment YAML file with the current build
        number
        sed -i "s/batch13:*/batch13:${BUILD_NUMBER}/g"
        deploymentfiles/deployment.yml
        #Stage all changes
        git add .
        # Commit changes with a message containing the build number
        git commit -m "Update deployment image to version ${BUILD_NUMBER}"
        #Push changes to the main branch of the GitHub repository
        git push
        https://${githubtoken}@github.com/${GIT_USER_NAME}/${GIT_REPO_NAME}
        HEAD:main
        '''
      }
    }
  }
}
}

```


Explanation of Each Stage

1. Checkout

- Clones the **main** branch of the specified GitHub repository to the Jenkins workspace.

2. SonarQube Scan

- Lists the directory contents to ensure files are in place.
- Runs a SonarQube scan using Maven to analyze the code for bugs, vulnerabilities, and code smells. The scan results are sent to the specified SonarQube server.

3. Build Artifact

- Cleans the workspace and packages the Maven project, creating a build artifact (typically a JAR or WAR file).

4. Build Docker Image

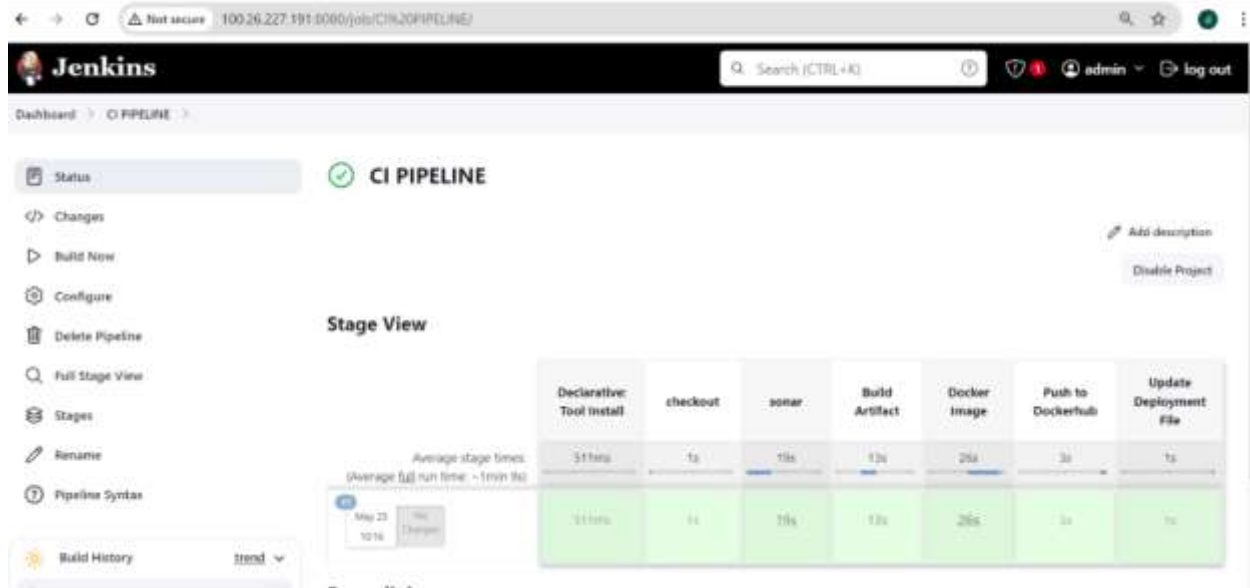
- Builds a Docker image from the Dockerfile in the project directory.
- Tags the Docker image with the Jenkins build number for versioning.

5. Push to Docker Hub

- Logs into Docker Hub using credentials stored in Jenkins.
- Pushes the Docker image to the Docker Hub repository.

6. Update Deployment File

- Configures git user details for committing changes.
- Updates the deployment YAML file to use the newly created Docker image with the current build number.
- Stages, commits, and pushes the updated deployment file back to the GitHub repository, ensuring the Kubernetes cluster can pull the latest image.



10. Kubernetes Deployment and Service Files

1. Installation of argocd:

```
kubectl create namespace argocd
```

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

2. Edit the Argo CD server service to type **LoadBalancer**:

```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
```

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

```
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d ; echo
```

Create **deployment.yaml** and **service.yaml** in your repository to define the Kubernetes resources:

deployment.yaml:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mc-app
  labels:
```

```
app: mc-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: mc-app
  template:
    metadata:
      labels:
        app: mc-app
    spec:
      containers:
        - name: mc-app
          image: devopshubg333/batch13:tag
          ports:
            - containerPort: 8080
```

service.yml:

```
apiVersion: v1
kind: Service
metadata:
  name: mc-app-service
spec:
  type: LoadBalancer
  ports:
    - name: http
      port: 80
      targetPort: 8080
      protocol: TCP
  selector:
    app: mc-app
```

11. ArgoCD Application Configuration

1. Open the Argo CD UI:

- Access the Argo CD UI using the **EXTERNAL-IP** (for LoadBalancer)
- Log in with the username **admin** and the password retrieved in the previous step.

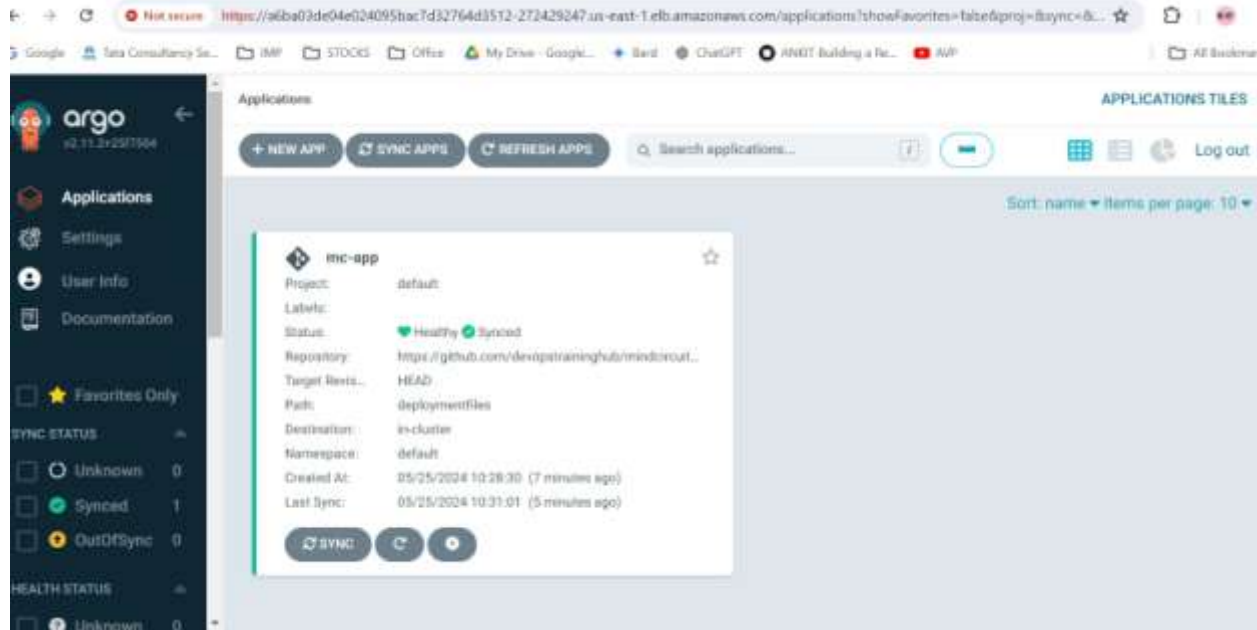
2. Create and Sync an Application



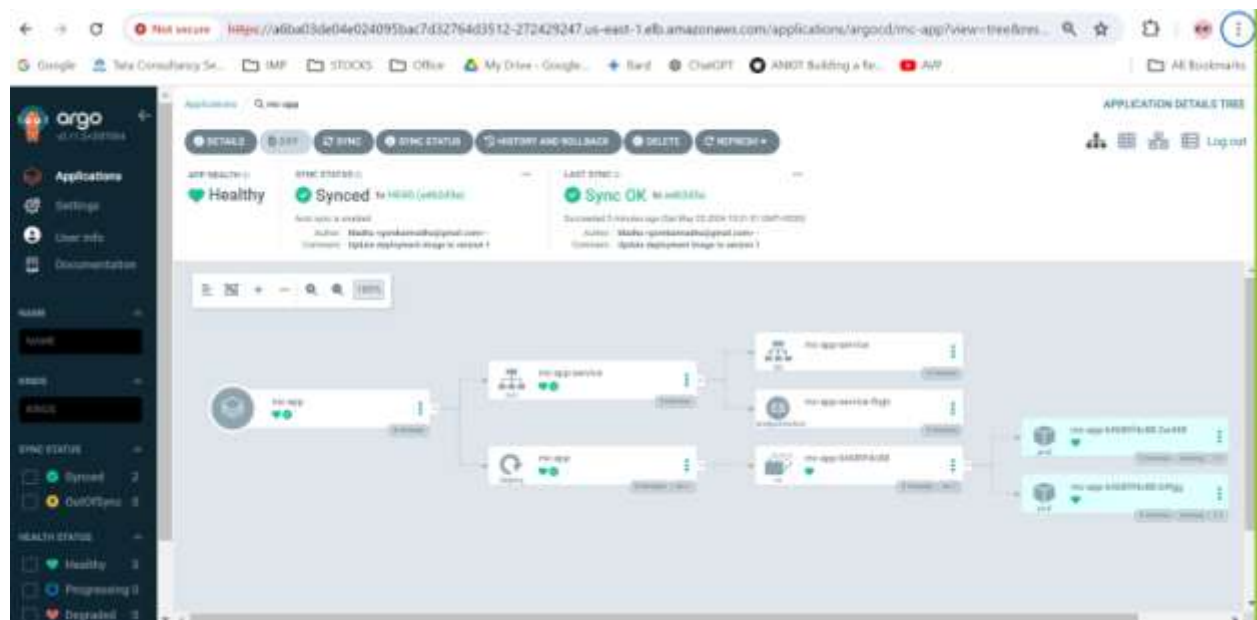
- Once logged in, click the + **New App** button at the top of the Argo CD dashboard.
- Fill out the application details in the "General" section:
- **Application Name:** **my-app** (or your preferred name)
- **Project:** **default**
- **Sync Policy:** Leave as Manual for now (you can change it to Automatic later if desired)
- **Fill out the "Source" section:**
 - **Repository URL:** Enter the URL of the Git repository containing your application manifests. For example, **<https://github.com/devopstraininghub/mindcircuit13.git>**.
 - **Revision:** Leave as **HEAD** or specify a branch/tag if needed.
 - **Path:** Enter the path within the repository where the Kubernetes manifests are located. For example, **deploymentfiles**.
- **Fill out the "Destination" section:**
 - **Cluster URL:** Leave as **<https://kubernetes.default.svc>** (this targets the current cluster).
 - **Namespace:** Specify the namespace where you want the application to be deployed. For example, **default**.
 - Click the **Create** button at the bottom of the page.

5. Sync the Application

- After the application is created, it will appear in the Argo CD dashboard with a status of **OutOfSync**.
- To sync the application, click on the application name (**my-app**).
- Click the **Sync** button at the top right of the application details page.
- In the sync dialog, review the resources to be synchronized and click **Synchronize** to start the sync process.
- The status will change to **Healthy** once the sync is complete and the application is successfully deployed.



The screenshot shows the Argo CD web interface. On the left is a sidebar with navigation links: Applications, Settings, User Info, and Documentation. Below these are sections for 'Favorites Only', 'SYNC STATUS' (Unknowns: 0, Synced: 1, OutOfSync: 0), and 'HEALTH STATUS' (Unknown: 0). The main panel displays the 'mc-app' application details. It includes a 'Project: default' and a 'Labels' section. The 'Status' is 'Healthy' and 'Synced'. The 'Repository' is 'https://github.com/devopstraininghub/mindcircuit...'. The 'Target Revis...' is 'HEAD'. The 'Path' is 'deploymentfiles'. The 'Destination' is 'in-cluster'. The 'Namespace' is 'default'. The 'Created At' is '05/25/2024 10:28:30 (7 minutes ago)' and the 'Last Sync' is '05/25/2024 10:31:01 (5 minutes ago)'. There are buttons for 'SYNC', 'REFRESH APPS', and 'Log out'.



The screenshot shows the Argo CD web interface with the 'mc-app' application details. The 'DETAILS' tab is selected, showing 'APP HEALTH: Healthy', 'SYNC STATUS: Synced', and 'LAST SYNC: Sync OK'. Below this is a 'APPLICATION DETAILS TREE' showing a hierarchical view of the application's resources. The tree includes 'mc-app', 'mc-app-service', 'mc-app-service-nginx', 'mc-app-service-nginx-nginx', 'mc-app-service-nginx-nginx-nginx', 'mc-app-service-nginx-nginx-nginx-nginx', and 'mc-app-service-nginx-nginx-nginx-nginx-nginx'. The 'mc-app' resource is highlighted, showing its 'HEALTHY' status and 'Synced' state. The 'mc-app-service' resource is also highlighted, showing its 'HEALTHY' status and 'Synced' state. The 'mc-app-service-nginx' resource is highlighted, showing its 'HEALTHY' status and 'Synced' state. The 'mc-app-service-nginx-nginx' resource is highlighted, showing its 'HEALTHY' status and 'Synced' state. The 'mc-app-service-nginx-nginx-nginx' resource is highlighted, showing its 'HEALTHY' status and 'Synced' state. The 'mc-app-service-nginx-nginx-nginx-nginx' resource is highlighted, showing its 'HEALTHY' status and 'Synced' state. The 'mc-app-service-nginx-nginx-nginx-nginx-nginx' resource is highlighted, showing its 'HEALTHY' status and 'Synced' state.