Jenkins CI/CD + Git

Hướng dẫn CI/CD với Jenkins trên Kubernetes - Từ đầu đến khi chạy thành công

Tổng quan hệ thống

- **Project**: Hospital Management System (.NET 9.0 + PostgreSQL)
- **Môi trường**: Minikube trên Mac M1/M2 (Apple Silicon)
- **CI/CD Tool**: Jenkins chạy trong Kubernetes
- **Container Registry**: DockerHub
- **Architecture**: Microservices với GitOps

Bước 1: Setup Minikube

1.1. Khởi tạo Minikube với resources đủ lớn

```
bash # Xóa cluster cũ nếu có
minikube delete

# Tạo cluster mới với resources phù hợp
minikube start --cpus=6 --memory=10240 --disk-size=50g --driver=docker

# Verify
minikube status
kubectl get nodes
```

Lý do: Docker-in-Docker build .NET images cần nhiều CPU/RAM.

1.2. Start Minikube Tunnel (Terminal riêng)

```
bashminikube tunnel
```

Lưu ý: Giữ terminal này mở suốt quá trình làm việc. Tunnel cho phép truy cập LoadBalancer services qua `127.0.0.1`.

Bước 2: Deploy Jenkins lên Kubernetes

2.1. Tạo file `jenkins-deployment.yaml`

```
 textapiVersion: v1
kind: Namespace
metadata:
  name: jenkins
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins-admin
  namespace: jenkins
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: jenkins-admin-role
rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: jenkins-admin-binding
subjects:
  - kind: ServiceAccount
    name: jenkins-admin
    namespace: jenkins
roleRef:
  kind: ClusterRole
  name: jenkins-admin-role
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: jenkins-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data/jenkins-home"
  storageClassName: jenkins-storage
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: jenkins-pvc
  namespace: jenkins
spec:
  accessModes:
```

```
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: jenkins-storage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins
  namespace: jenkins
spec:
  replicas: 1
  selector:
    matchLabels:
      app: jenkins
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      serviceAccountName: jenkins-admin
      containers:
        - name: jenkins
          image: jenkins/jenkins:lts-jdk17
          ports:
            - containerPort: 8080
            - containerPort: 50000
          volumeMounts:
            - name: jenkins-home
              mountPath: /var/jenkins_home
      volumes:
        - name: jenkins-home
          persistentVolumeClaim:
            claimName: jenkins-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: jenkins-service
  namespace: jenkins
spec:
  type: LoadBalancer
  selector:
    app: jenkins
  ports:
    - name: http
      port: 8080
      targetPort: 8080
    - name: agent
      port: 50000
      targetPort: 50000
```

2.2. Deploy Jenkins

```bash
kubectl apply -f jenkins-deployment.yaml

# Đợi pod ready
kubectl get pods -n jenkins -w

# Lấy initial admin password
kubectl exec -it <jenkins-pod-name> -n jenkins -- cat /var/jenkins_home/secrets/initialAdminPassword
```

2.3. Truy cập Jenkins UI

```
http://127.0.0.1:8080
```

- Nhập initial admin password
  - Lấy pod name và password trong 1 lệnh
    - `kubectl exec -n jenkins $(kubectl get pods -n jenkins -l app=jenkins -o jsonpath='{.items[0].metadata.name}') -- cat /var/jenkins_home/secrets/initialAdminPassword`
- Install suggested plugins
- Tạo admin user
- Configure Jenkins URL: `http://jenkins-service.jenkins.svc.cluster.local:8080`

Bước 3: Cấu hình Kubernetes Cloud trong Jenkins

3.1. Install Kubernetes Plugin

Manage Jenkins → **Plugins** → **Available plugins** → Tìm "**Kubernetes**" → Install

3.2. Configure Kubernetes Cloud

Manage Jenkins → **Clouds** → **New cloud** → Chọn **Kubernetes**

Cấu hình:

- **Name**: `kubernetes`
- **Kubernetes URL**: `https://kubernetes.default.svc.cluster.local`
- **Kubernetes Namespace**: `jenkins`
- **Jenkins URL**: `http://jenkins-service.jenkins.svc.cluster.local:8080`
- **Jenkins tunnel**: `jenkins-service.jenkins.svc.cluster.local:50000`
- **Credentials**: None (dùng ServiceAccount)

Click **Test Connection** → Phải thấy "Connected to Kubernetes..."

Click **Save**

Bước 4: Tạo Credentials trong Jenkins

4.1. DockerHub Credentials

Manage Jenkins → Credentials → System → Global credentials → Add Credentials

- **Kind**: Username with password
- **ID**: `dockerhub-credentials`
- **Username**: `sangrk`
- **Password**: `<your-dockerhub-password>`

4.2. Environment File Credential

Add Credentials → Kind: Secret file

- **ID**: `hms-env-file`
- **File**: Upload file `.env` chứa config của ứng dụng

Bước 5: Tạo Jenkinsfile

5.1. Jenkinsfile hoàn chỉnh

```
groovypipeline {
  agent {
    kubernetes {
      yaml """
apiVersion: v1
kind: Pod
metadata:
 labels:
  jenkins: agent
spec:
 serviceAccountName: jenkins-admin
 securityContext:
  fsGroup: 1000
 containers:
 - name: jnlp
  image: jenkins/inbound-agent:latest
  args: ['\$(JENKINS_SECRET)', '\$(JENKINS_NAME)']
  resources:
   requests:
```

```
      cpu: "500m"
      memory: "512Mi"
    limits:
      cpu: "1"
      memory: "1Gi"

- name: docker
  image: docker:24.0-dind
  securityContext:
    privileged: true
  env:
  - name: DOCKER_TLS_CERTDIR
    value: ""
  - name: DOCKER_DRIVER
    value: overlay2
  command:
  - dockerd
  - --host=unix:///var/run/docker.sock
  - --host=tcp://0.0.0.0:2375
  - --tls=false
  volumeMounts:
  - name: docker-storage
    mountPath: /var/lib/docker
  - name: docker-sock
    mountPath: /var/run
  resources:
    requests:
      cpu: "2"
      memory: "4Gi"
    limits:
      cpu: "4"
      memory: "8Gi"
  readinessProbe:
    exec:
      command: ["docker", "info"]
    initialDelaySeconds: 20
    periodSeconds: 5
    timeoutSeconds: 10

- name: kubectl
  image: alpine/k8s:1.28.3
  command: ['/bin/sh']
  args: ['-c', 'while true; do sleep 30; done']
  tty: true
  resources:
    requests:
      cpu: "250m"
      memory: "256Mi"
    limits:
      cpu: "500m"
      memory: "512Mi"

volumes:
- name: docker-storage
  emptyDir:
    sizeLimit: 30Gi
```

```
  - name: docker-sock
    emptyDir: {}
"""
        }
    }

    options {
        buildDiscarder(logRotator(numToKeepStr: '10'))
        timeout(time: 60, unit: 'MINUTES')
        timestamps()
        disableConcurrentBuilds()
    }

    environment {
        DOCKER_REGISTRY = 'sangrk'
        BACKEND_IMAGE_NAME = "${env.DOCKER_REGISTRY}/hms-api"
        FRONTEND_IMAGE_NAME = "${env.DOCKER_REGISTRY}/hms-frontend"
        BACKEND_TAG = "v20"
        FRONTEND_TAG = "latest"
        DOCKER_HOST = "tcp://localhost:2375"
        DOCKER_BUILDKIT = "1"
    }

    stages {
        stage('Checkout') {
            steps {
                container('jnlp') {
                    echo '🔄 Checking out source code...'
                    checkout scm
                }
            }
        }

        stage('Wait for Docker') {
            steps {
                container('docker') {
                    script {
                        echo '⏳ Waiting for Docker daemon to be ready...'
                        sh '''
                        for i in {1..30}; do
                            if docker info > /dev/null 2>&1; then
                                echo "✅ Docker daemon is ready!"
                                docker info
                                break
                            fi
                            echo "Waiting for Docker daemon... ($i/30)"
                            sleep 3
                        done
                        '''
                    }
                }
            }
        }

        stage('Setup Buildx') {
            steps {
```

```
            container('docker') {
                script {
                    echo '🔧 Setting up Docker Buildx...'
                    sh '''
                    docker buildx rm mybuilder || true

                    docker buildx create \\
                        --name mybuilder \\
                        --driver docker-container \\
                        --driver-opt network=host \\
                        --use

                    docker buildx inspect --bootstrap
                    docker buildx ls
                    '''
                }
            }
        }
    }

    stage('Setup Configuration') {
        steps {
            container('kubectl') {
                script {
                    withCredentials([file(credentialsId: 'hms-env-file', variable: 'ENV_FILE_PATH')]) {
                        echo '⚙️ Applying Kubernetes configurations...'
                        sh """
                        kubectl delete configmap hms-api-config -n default || true
                        kubectl create configmap hms-api-config --from-env-file=\${ENV_FILE_PATH} -n default
                        """
                    }
                }
            }
        }
    }

    stage('Build & Push Backend') {
        when {
            anyOf {
                expression { env.BUILD_NUMBER == '1' }
                changeset "HospitalManagementSystem.API/**"
            }
        }
        steps {
            container('docker') {
                script {
                    echo "🏗️ Building Backend: ${env.BACKEND_IMAGE_NAME}:${env.BACKEND_TAG}"

                    withCredentials([usernamePassword(
                        credentialsId: 'dockerhub-credentials',
                        usernameVariable: 'DOCKER_USER',
                        passwordVariable: 'DOCKER_PASS'
                    )]) {
                        sh "echo \${DOCKER_PASS} | docker login -u \${DOCKER_USER} --password-stdin"
                    }
```

```
                sh """
                docker buildx build \\
                  --platform linux/arm64 \\
                  --tag ${env.BACKEND_IMAGE_NAME}:${env.BACKEND_TAG} \\
                  --file HospitalManagementSystem.API/Dockerfile \\
                  --progress=plain \\
                  --pull \\
                  --push \\
                  .
                """

                echo "✅ Backend image pushed: ${env.BACKEND_IMAGE_NAME}:${env.BACKEND_TAG}"
            }
        }
    }
}

stage('Deploy Backend') {
    when {
        anyOf {
            expression { env.BUILD_NUMBER == '1' }
            changeset "HospitalManagementSystem.API/**"
        }
    }
    steps {
        container('kubectl') {
            script {
                echo "🚀 Deploying Backend with tag ${env.BACKEND_TAG}..."
                sh """
                kubectl rollout restart deployment/hms-api -n default
                kubectl rollout status deployment/hms-api -n default --timeout=10m
                """
                echo '✅ Backend deployed successfully!'
            }
        }
    }
}

stage('Build & Push Frontend') {
    when {
        anyOf {
            expression { env.BUILD_NUMBER == '1' }
            changeset "frontend/**"
        }
    }
    steps {
        container('docker') {
            script {
                echo "🏗️ Building Frontend: ${env.FRONTEND_IMAGE_NAME}:${env.FRONTEND_TAG}"

                withCredentials([usernamePassword(
                    credentialsId: 'dockerhub-credentials',
                    usernameVariable: 'DOCKER_USER',
                    passwordVariable: 'DOCKER_PASS'
                )]) {
                    sh "echo \${DOCKER_PASS} | docker login -u \${DOCKER_USER} --password-stdin"
```

```
                }

                dir('frontend') {
                    sh """
                    docker buildx build \\
                        --platform linux/arm64 \\
                        --tag ${env.FRONTEND_IMAGE_NAME}:${env.FRONTEND_TAG} \\
                        --progress=plain \\
                        --pull \\
                        --push \\
                        .
                    """
                }

                echo "✅ Frontend image pushed: ${env.FRONTEND_IMAGE_NAME}:${env.FRONTEND_TAG}"
            }
        }
    }
}

stage('Deploy Frontend') {
    when {
        anyOf {
            expression { env.BUILD_NUMBER == '1' }
            changeset "frontend/**"
        }
    }
    steps {
        container('kubectl') {
            script {
                echo "🚀 Deploying Frontend with tag ${env.FRONTEND_TAG}..."
                sh """
                kubectl rollout restart deployment/hms-frontend -n default
                kubectl rollout status deployment/hms-frontend -n default --timeout=10m
                """
                echo '✅ Frontend deployed successfully!'
            }
        }
    }
}

stage('Cleanup') {
    steps {
        container('docker') {
            script {
                echo '🧹 Cleaning up Docker resources...'
                sh '''
                docker image prune -af --filter "until=24h" || true
                docker buildx rm mybuilder || true
                docker system df
                '''
            }
        }
    }
}
}
```

```
    post {
        always {
            container('docker') {
                script {
                    echo '🔒 Logging out from Docker Hub...'
                    sh 'docker logout || true'
                }
            }
        }
        success {
            echo '✅ Pipeline completed successfully!'
        }
        failure {
            echo '❌ Pipeline failed! Check logs above.'
        }
        cleanup {
            echo '🧹 Cleaning up workspace...'
        }
    }
}
```

5.2. Commit Jenkinsfile vào GitHub

```
bashcd HospitalManagementSystem
git add Jenkinsfile
git commit -m "Add Jenkins CI/CD pipeline"
git push origin main
```

Bước 6: Tạo Jenkins Pipeline Job

6.1. Create Pipeline

Dashboard → New Item

- **Name**: `hms-project-pipeline`
- **Type**: **Pipeline**
- Click **OK**

6.2. Configure Pipeline

Pipeline section:

- **Definition**: Pipeline script from SCM
- **SCM**: Git
- **Repository URL**: `https://github.com/ComBiCha/HospitalManagementSystem`

- **Branch:** `/main` (hoặc `/refactor/ddd-merge`)
- **Script Path:** `Jenkinsfile`
- Nếu muốn tự động build khi Push code lên github thì trong **Trigger** phải bật **GitHub hook trigger for GITScm polling**

Click **Save**

Bước 7: Chạy Pipeline

7.1. Trigger Build

Click **Build Now**

7.2. Monitor Build

- Xem real-time logs
- Pods sẽ được tạo tự động trong namespace `jenkins`
- Monitor pods: `kubectl get pods -n jenkins -w`

Các lỗi gặp phải và giải pháp

Lỗi 1: Jenkins mất credentials sau restart

Nguyên nhân: Minikube pod bị restart mà không có persistent volume

Giải pháp: Đã thêm PersistentVolume trong `jenkins-deployment.yaml`

```
textapiVersion: v1
kind: PersistentVolume
metadata:
  name: jenkins-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data/jenkins-home"
  storageClassName: jenkins-storage
```

Lỗi 2: Minikube tunnel bị TLS handshake timeout

Nguyên nhân: Minikube cluster quá tải do Docker-in-Docker build

Giải pháp:

- Tăng resources cho Minikube: `-cpus=6 --memory=10240`
- Không bắt buộc phải chạy tunnel khi build (chỉ cần khi truy cập UI)

Lỗi 3: Docker buildx không có `-platform` flag

Nguyên nhân: Image `docker:20.10.7` không có buildx plugin

Giải pháp: Đổi sang `docker:24.0-dind` có buildx sẵn

```
text- name: docker
 image: docker:24.0-dind
```

Lỗi 4: kubectl container không execute được shell commands

Nguyên nhân: Image `bitnami/kubectl:latest` với `command: ['cat']` không tương thích

Giải pháp: Đổi sang `alpine/k8s:1.28.3` với proper shell

```
text- name: kubectl
 image: alpine/k8s:1.28.3
 command: ['/bin/sh']
 args: ['-c', 'while true; do sleep 30; done']
 tty: true
```

Lỗi 5: ImagePullBackOff - no matching manifest for linux/arm64

Nguyên nhân: Build image cho `linux/amd64` nhưng Mac M1/M2 là `linux/arm64`

Giải pháp: Build cho ARM64 trên Mac

```
groovydocker buildx build \\
  --platform linux/arm64 \\  # Cho Mac M1/M2
  --tag ${env.BACKEND_IMAGE_NAME}:${env.BACKEND_TAG} \\
  --push \\
  .
```

Hoặc build multi-platform cho production:

```
groovy--platform linux/amd64,linux/arm64 \\
```

Lỗi 6: Pod bị stuck ở ContainerCreating

Nguyên nhân: Resources không đủ hoặc image pull chậm

Giải pháp:

- Tăng resources Minikube

- Giảm resources requests trong pod spec
- Đợi image pull xong (check với `kubectl describe pod`)

Lỗi 7: Permission Denied - /var/jenkins_home

Triệu chứng:

```
texttouch: cannot touch '/var/jenkins_home/copy_reference_file.log': Permission denied
```

Nguyên nhân: Jenkins user (UID 1000) không có quyền ghi vào hostPath volume

Giải pháp: Thêm init container vào deployment:

```
textinitContainers:
- name: fix-permissions
  image: busybox
  command:
  - sh
  - -c
  - |
    chown -R 1000:1000 /var/jenkins_home
    chmod -R 755 /var/jenkins_home
  volumeMounts:
  - name: jenkins-home
    mountPath: /var/jenkins_home
  securityContext:
    runAsUser: 0
```

Hoặc fix trực tiếp trong Minikube:

```
bashminikube ssh "sudo chown -R 1000:1000 /data/jenkins-home"
```

Lỗi 8: Lệnh docker báo lỗi "Cannot connect to the Docker daemon"

Triệu chứng:
Log trong pipeline hiển thị lỗi: `Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?`

Nguyên nhân:
Pipeline step (chạy trong container jnlp) cố gắng thực thi lệnh docker. Nó tìm Docker daemon ở địa chỉ mặc định (unix:///var/run/docker.sock), nhưng Docker daemon lại đang chạy trong một container docker hoàn toàn riêng biệt và không chia sẻ socket đó.

Giải pháp:

1. Trong Jenkinsfile, cấu hình container docker để khởi động Docker daemon và lắng nghe trên một cổng TCP: `command: ['dockerd', '--host=tcp://0.0.0.0:2375', ...]`.

2. Khai báo biến môi trường `DOCKER_HOST = "tcp://localhost:2375"` trong pipeline. Điều này hướng dẫn tất cả các lệnh docker từ bất kỳ container nào trong pod kết nối đến Docker daemon thông qua localhost trên cổng 2375.

Lỗi 9: Lỗi xác thực "Forbidden" khi kubectl chạy trong pipeline

Triệu chứng:
Pipeline thất bại ở các stage Deploy với lỗi: `Error from server (Forbidden): deployments.apps "hms-api" is forbidden: User "system:serviceaccount:jenkins:jenkins-admin" cannot get resource "deployments" ...`

Nguyên nhân:
ServiceAccount mặc định mà pod agent sử dụng không có quyền để tương tác (get, list, create, patch) với các tài nguyên Kubernetes như Deployment, ConfigMap trong namespace default.

Giải pháp:

1. Tạo một **ServiceAccount** riêng (jenkins-admin).
2. Tạo một **ClusterRole** (jenkins-admin-role) định nghĩa các quyền hạn trên toàn cluster (sử dụng * cho resources và verbs).
3. Gắn **ClusterRole** này vào ServiceAccount thông qua một ClusterRoleBinding.
4. Trong **Jenkinsfile**, chỉ định pod agent phải sử dụng Service Account này: serviceAccountName: jenkins-admin.

Lỗi 10: Docker-in-Docker không khởi động được

Triệu chứng:
Container docker trong pod agent liên tục bị crash hoặc không bao giờ sẵn sàng. kubectl describe pod có thể báo lỗi liên quan đến permissions.

Nguyên nhân:
Docker daemon cần các quyền rất cao ở mức hệ thống (kernel capabilities) để có thể tạo và quản lý các container khác. Mặc định, một container không có các quyền này.

Giải pháp:
Thêm securityContext: { privileged: true } vào định nghĩa của container docker trong Jenkinsfile. Điều này cấp cho nó toàn quyền cần thiết để hoạt động.

Lỗi 11: Docker buildx "multiple platforms feature is currently not supported"

Triệu chứng:

```
textERROR: multiple platforms feature is currently not supported for docker driver
```

Nguyên nhân: Default docker driver không hỗ trợ multi-platform

Giải pháp: Đã fix trong pipeline:

```
groovydocker buildx create --driver docker-container --use
```

Lỗi 11: file .env không tồn tại trên github

Nguyên nhân: File .env bị gitignore

Giải pháp: Tạo Credential trong Jenkins bằng secret file và ID: hms-env-file

Lỗi 12: Không dùng accessToken trên docker

Triệu chứng:

```
textunauthorized: incorrect username or password
```

Hoặc:

```
textError response from daemon: login attempt to https://registry-1.docker.io/v2/ failed with status: 401 Unauthorized
```

Nguyên nhân: DockerHub đã deprecate password authentication từ 2021. Phải dùng **Access Token**.

Giải pháp:

Bước 1: TẠO DOCKERHUB ACCESS TOKEN

1. Truy cập: https://hub.docker.com/settings/security
2. Click **New Access Token**
3. Cấu hình:

   ```
   textAccess Token Description: Jenkins
   Access permissions: Read, Write, Delete
   ```

4. Click **Generate**
5. **COPY TOKEN NGAY** (chỉ hiện 1 lần)

Bước 2: CẬP NHẬT JENKINS CREDENTIAL

Lỗi 13: chưa setup cloud kubernetes

Triệu chứng:

```
textERROR: Could not create Kubernetes client: java.io.IOException: Failed to connect to https://kubernetes.default.svc.cluster.local
```

Hoặc Pipeline stuck:

```
textAgent jenkins-agent-xxxxx is offline
Waiting for agent to connect (1/100)
```

Hoặc trong Jenkins logs:

```
textWARNING: Kubernetes cloud 'kubernetes' is not configured properly
```

Nguyên nhân:

1. Kubernetes Plugin chưa được cài

2. Kubernetes Cloud chưa được configure

3. URLs sai hoặc không reachable

Giải pháp:

Bước 1: CÀI KUBERNETES PLUGIN

1. Manage Jenkins → Plugins
2. Tab **Available plugins**
3. Search: "Kubernetes"
4. Check ☑️ **Kubernetes plugin**
5. Click **Install without restart**
6. Wait for installation complete

Bước 2: CONFIGURE KUBERNETES CLOUD

1. Manage Jenkins → **Clouds**
2. Click **New cloud**
3. Name: `kubernetes`
4. Type: **Kubernetes**
5. Click **Create**

SETUP GITHUB WEBHOOK CHO JENKINS

OPTION 1: GITHUB WEBHOOK

Bước 1: Cài GitHub Plugin

```bash
# Check plugin đã cài chưa
kubectl exec -n jenkins $(kubectl get pods -n jenkins -l app=jenkins -o jsonpath='{.items[0].metadata.name}') -- jenkins-plugin-cli --list | grep github
```

Nếu chưa có:

1. Manage Jenkins → Plugins → Available plugins
2. Search: "GitHub plugin"
3. Install: **GitHub plugin** + **GitHub Branch Source Plugin**
4. Restart Jenkins

Bước 2: Tạo GitHub Personal Access Token

1. Truy cập: https://github.com/settings/tokens
2. Click **Generate new token** → **Generate new token (classic)**
3. Cấu hình:

```
textNote: Jenkins Webhook Token
Expiration: No expiration (hoặc 90 days)

Select scopes:
repo (Full control of private repositories)
admin:repo_hook (Write:repo_hook, Read:repo_hook)
```

4. Click **Generate token**
5. **Copy token ngay:** `ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`

Bước 3: Add Token vào Jenkins Credentials

1. Jenkins → Manage Jenkins → Credentials
2. System → Global credentials → Add Credentials
3. Cấu hình:

```
textKind: Secret text
Secret: ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ID: github-webhook-token
Description: GitHub Personal Access Token for Webhooks
```

4. Click **Create**

Bước 4: Configure GitHub Server trong Jenkins

1. Manage Jenkins → **System**
2. Scroll xuống **GitHub** section
3. Click **Add GitHub Server**
4. Cấu hình:

   ```
   textName: github.com
   API URL: https://api.github.com
   Credentials: [Select] github-webhook-token
   Manage hooks
   ```

5. Click **Test connection** → **Credentials verified for user ComBiCha**
6. Click **Save**

Bước 5: Expose Jenkins Service (Webhook cần public URL)

Problem: Jenkins đang chạy trong Minikube, GitHub không thể gọi đến `http://127.0.0.1:8080`

Solution A: ngrok (Nhanh nhất cho testing)

```bash
# Install ngrok
brew install ngrok

# Authenticate
ngrok config add-authtoken YOUR_NGROK_TOKEN

# Expose Jenkins
ngrok http 8080
```

Output:

```
textForwarding  https://abc123.ngrok-free.app → http://localhost:8080
```

Copy URL: `https://abc123.ngrok-free.app`

Bước 6: Configure Jenkins URL

1. Manage Jenkins → **System**
2. **Jenkins Location** section:

   ```
   textJenkins URL: https://abc123.ngrok-free.app
   ```

   (Hoặc URL public của bạn)
3. Click **Save**

Bước 7: Setup Webhook trong GitHub Repository

1. Truy cập: https://github.com/ComBiCha/HospitalManagementSystem
2. Settings → **Webhooks** → **Add webhook**
3. Cấu hình:

   `textPayload URL: https://abc123.ngrok-free.app/github-webhook/`

   `Content type: application/json`

   `Secret: [Leave empty hoặc generate secret]`

   `Which events would you like to trigger this webhook?`
   `Just the push event`

   `Active`
4. Click **Add webhook**

Bước 8: Verify Webhook

1. Trong GitHub Webhooks page, click vào webhook vừa tạo
2. Tab **Recent Deliveries**
3. Phải thấy: Response code 200 (hoặc trigger test delivery)

Bước 9: Configure Pipeline để nhận webhook

Update Pipeline Job:

1. Dashboard → hms-project-pipeline → **Configure**
2. **Build Triggers** section:

   `GitHub hook trigger for GITScm polling`
3. Click **Save**

Bước 10: Test Webhook

```bash
# Push commit mới lên GitHub
cd HospitalManagementSystem
echo "# Test webhook" >> README.md
git add README.md
git commit -m "Test Jenkins webhook trigger"
git push origin main
```

Expected result:

- Jenkins tự động trigger build mới
- Logs hiện: `Started by GitHub push by ComBiCha`

OPTION 2: POLLING (Fallback - Không cần public URL)

Nếu không thể expose Jenkins ra public:

Configure Polling

1. Pipeline job → **Configure**
2. **Build Triggers** section:

   `Poll SCM`

   `Schedule: H/5 * * * *`

   (Check GitHub mỗi 5 phút)
3. Click **Save**

Schedule syntax:

```
textH/5 * * * *    # Mỗi 5 phút
H/15 * * * *   # Mỗi 15 phút
H * * * *      # Mỗi giờ
H H * * *      # Mỗi ngày
```

Nhược điểm:

- Delay 5 phút trước khi build
- Tốn resources (Jenkins liên tục poll GitHub)