Jenkins practical task

1. Install Jenkins according to documentation (Java based or docker based).
2. Make a base setup of Jenkins (User configuration, plugin installation).
3. Add a Dockerfile to your repository with spring-petclinic.
4. Create 2 docker repositories on your own Nexus Repository (Instruction) or https://hub.docker.com/ called “main” and “mr”.
5. Add Jenkinsfile and describe the following behavior there:
    a. The pipeline for a merge request should include the following jobs:
        i. Checkstyle: Use Maven or Gradle checkstyle plugin to generate a code style report. It should be available as a job artifact.
        ii. Test (with Maven or Gradle)
        iii. Build: Build without tests (with Maven or Gradle).
        iv. Create a docker image: Using your Dockerfile in the spring-petclinic repo, create a docker image with spring-petclinic application, tag it using GIT_COMMIT (short) and push it to the “mr” repository.

    Note: Pipelines should be executed in Jenkins agents

    b. The pipeline for the main branch should include the following job:
        i. Create a docker image: Build a docker image and push it to the “main” repository.

    Note: Pipelines should be executed in Jenkins agents

1. Install Jenkins according to documentation (Java based or docker based).

I have decided to use Docker based Jenkins distribution, so first we need to run and configure Docker image for Jenkins master and agent node.

Run container with Jenkins master:

```
05:14:15 adrwal@olek-desktop-pc jenkins → docker run \
  --name jenkins-blueocean-task \
  --restart=on-failure \
  --detach \
  --network jenkins \
  --publish 8080:8080 \
  --publish 50000:50000 \
  --volume jenkins-data:/var/jenkins_home \
  jenkins/jenkins
Unable to find image 'jenkins/jenkins:latest' locally
latest: Pulling from jenkins/jenkins
Digest: sha256:4bc56852fb6e174a634c1e0615ffbb60441bea74d91f2684f2f1668ffdd107dd
Status: Downloaded newer image for jenkins/jenkins:latest
a08828fb26862924c3f884bb5f4da87d4bb4de1526866acd7aa3924f5271933b
05:14:55 adrwal@olek-desktop-pc jenkins →
```

Run container with Docker daemon, because our agent will need to use Docker to build spring-petclinic app:

```
05:20:16 adrwal@olek-desktop-pc jenkins → docker run \
  --name jenkins-docker \
  --rm \
  --detach \
  --privileged \
  --network jenkins \
  --network-alias docker \
  --env DOCKER_TLS_CERTDIR=/certs \
  --volume jenkins-docker-certs:/certs/client \
  --volume jenkins-data:/var/jenkins_home \
  --publish 2376:2376 \
  docker:dind \
  --storage-driver overlay2
d235a1686b68ce2c83e14f35141fecafbba76aacc3f24190864fdc26885427a6
05:21:08 adrwal@olek-desktop-pc jenkins →
```

Run agent:

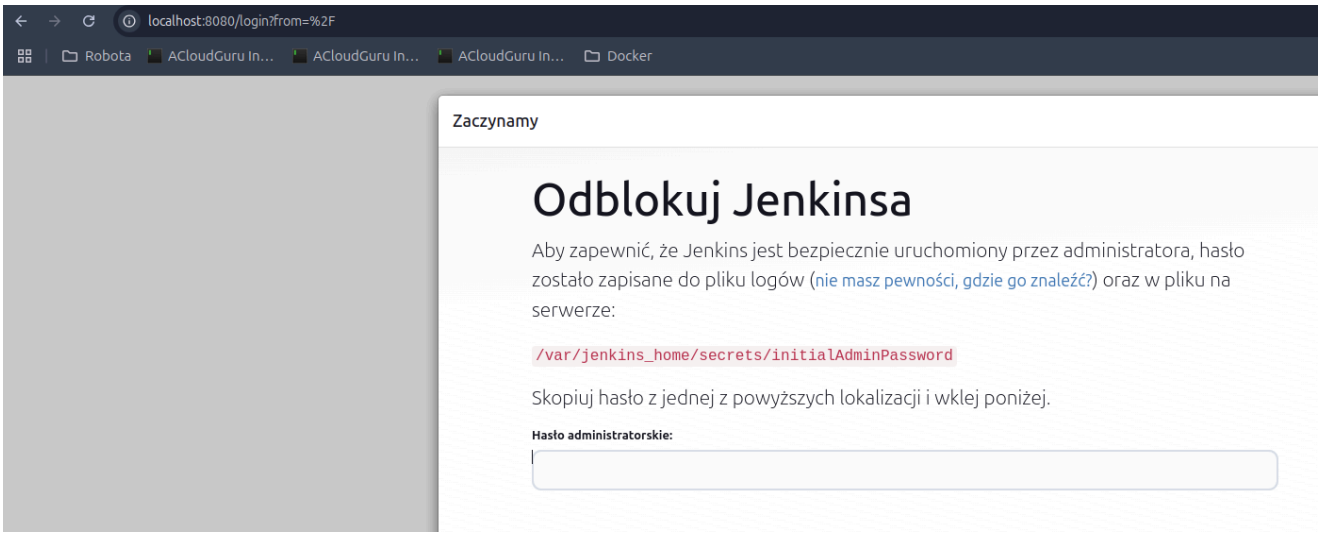
```
05:19:12 adrwal@olek-desktop-pc jenkins → docker run -it --rm --name agent \
  --network jenkins \
  --env DOCKER_HOST=tcp://docker:2376 \
  --env DOCKER_CERT_PATH=/certs/client \
  --env DOCKER_TLS_VERIFY=1 \
  --volume jenkins-docker-certs:/certs/client:ro \
jenkins/jnlp-agent-docker
WARNING: Are you running agent from an interactive console?
If so, you are probably using it incorrectly.
See https://wiki.jenkins.io/display/JENKINS/Launching+agent+from+console
<===[JENKINS REMOTING CAPACITY]===>rO0ABXNyABpodWRzb24ucmVtb3RpbmcuQ2FwYWJpbGl0eQAAAAAAAAABAgABSgAEbWFza3hwAAAAAAAAAf4=
```

At the end we should have 3 running containers.

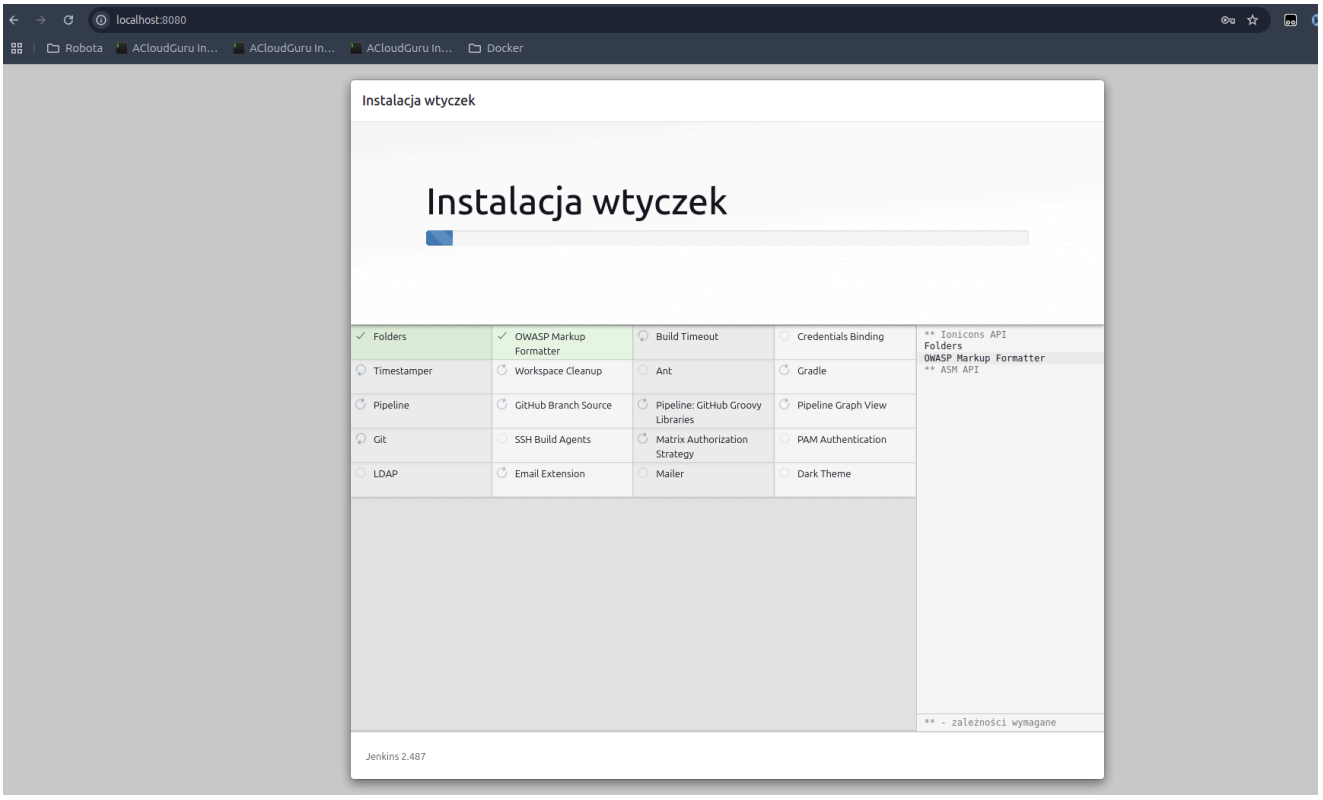
```
05:21:08 adrwal@olek-desktop-pc jenkins → docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED              STATUS              PORTS                                                  NAMES
d235a1686b68   docker:dind               "dockerd-entrypoint.…"   18 seconds ago       Up 17 seconds       2375/tcp, 0.0.0.0:2376->2376/tcp                       jenkins-docker
9c82abd52b77   jenkins/jnlp-agent-docker "/usr/local/bin/jenk…"  About a minute ago   Up About a minute                                                          agent
a08828fb2686   jenkins/jenkins           "/usr/bin/tini -- /u…"   6 minutes ago        Up 6 minutes        0.0.0.0:8080->8080/tcp, 0.0.0.0:50000->50000/tcp   jenkins-blueocean-task
05:21:26 adrwal@olek-desktop-pc jenkins →
```

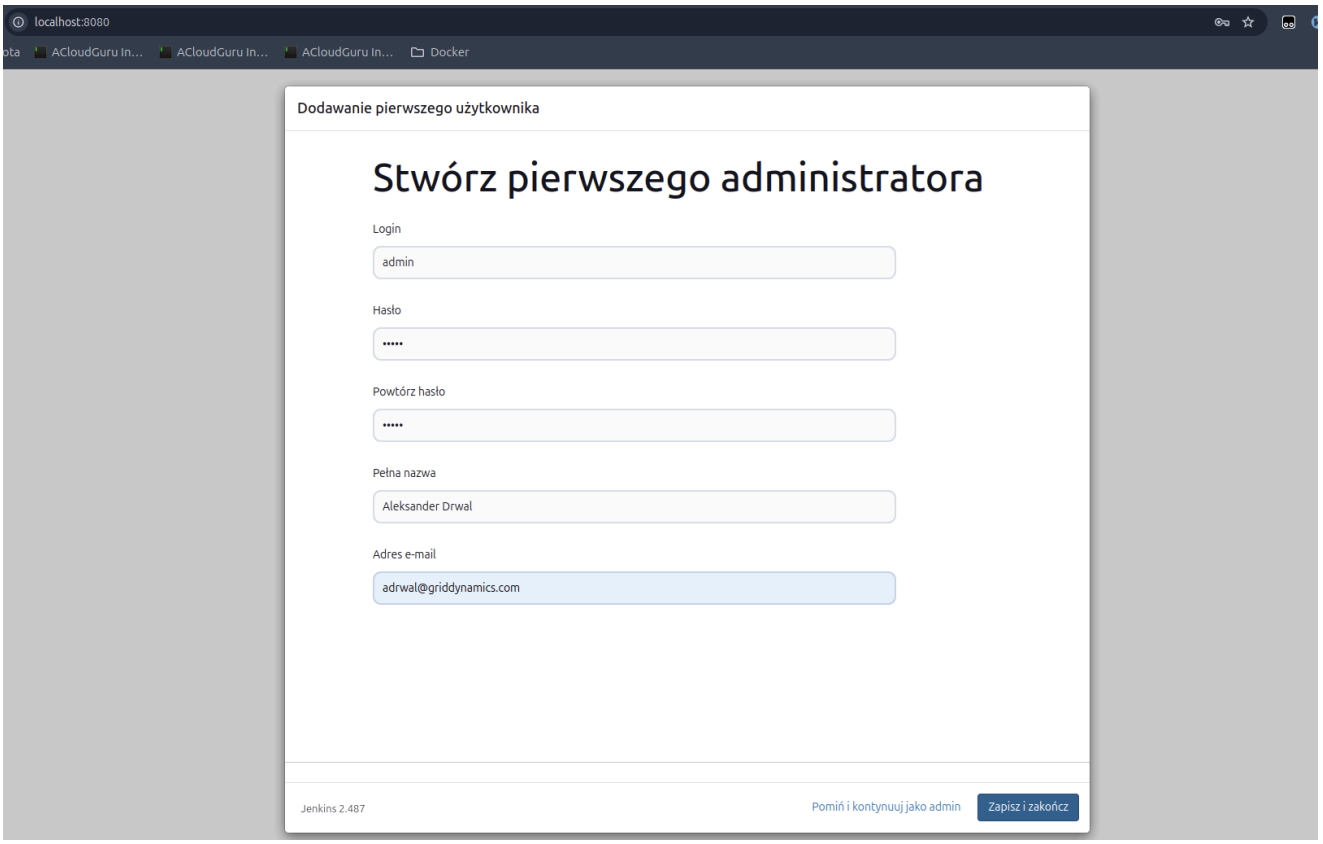
2. Make a base setup of Jenkins (User configuration, plugin installation).

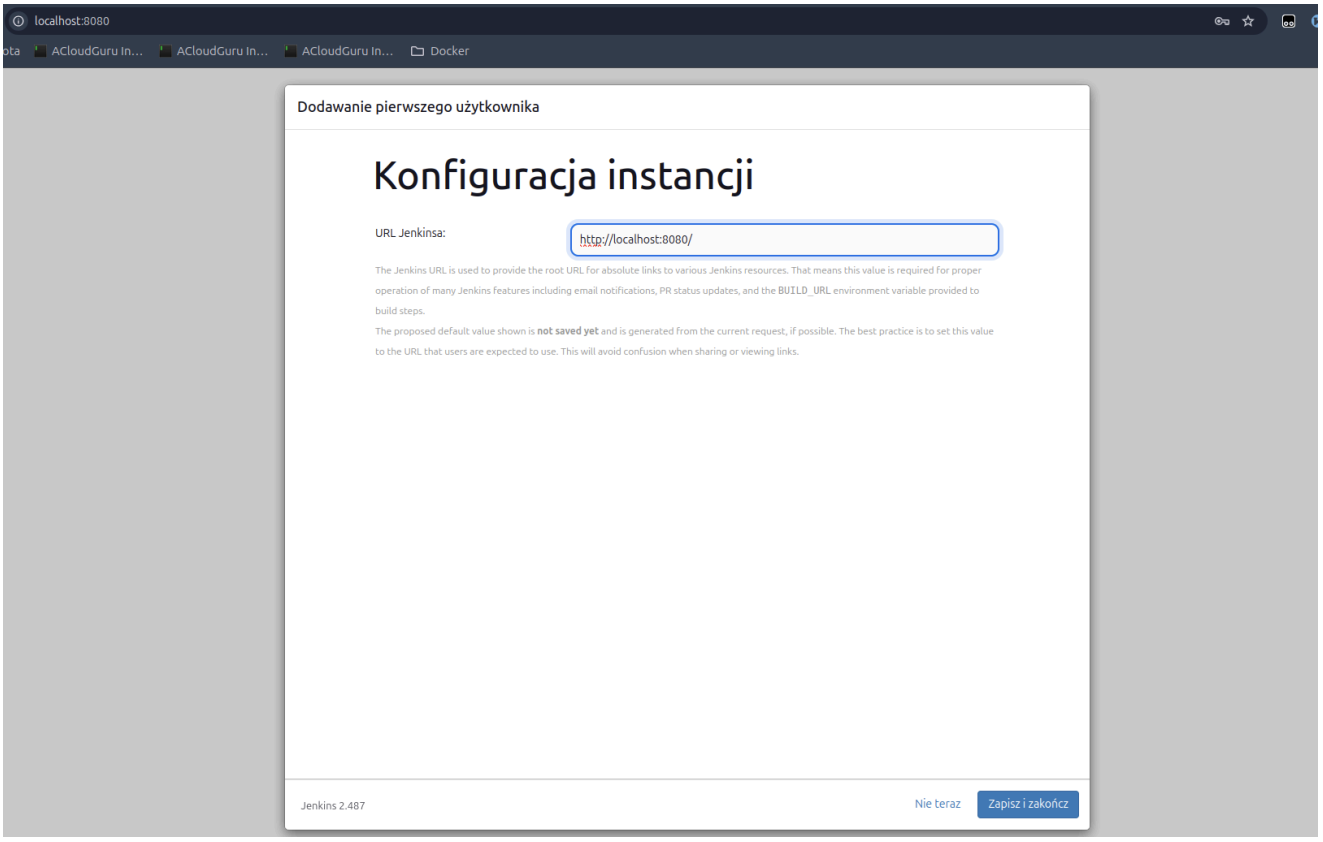
We can now access jenkins on `localhost:8080`

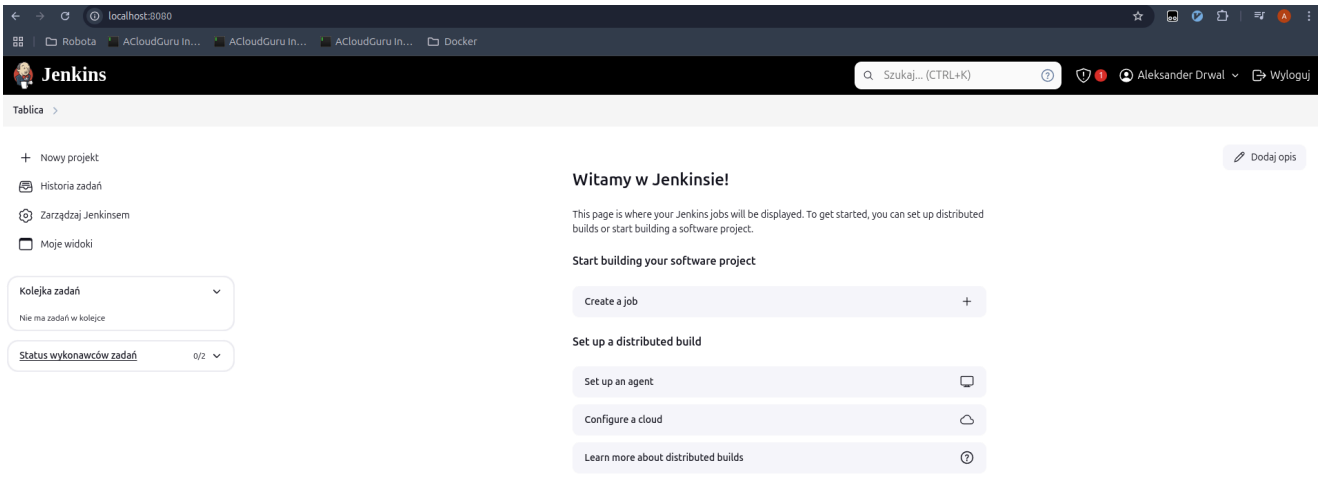



```
05:27:36 adrwal@olek-desktop-pc ~ → docker exec -it jenkins-blueocean-task cat /var/jenkins_home/secrets/initialAdminPassword
cd392107cd794b0b8cf1274a64a853ea
05:29:19 adrwal@olek-desktop-pc ~ →
```

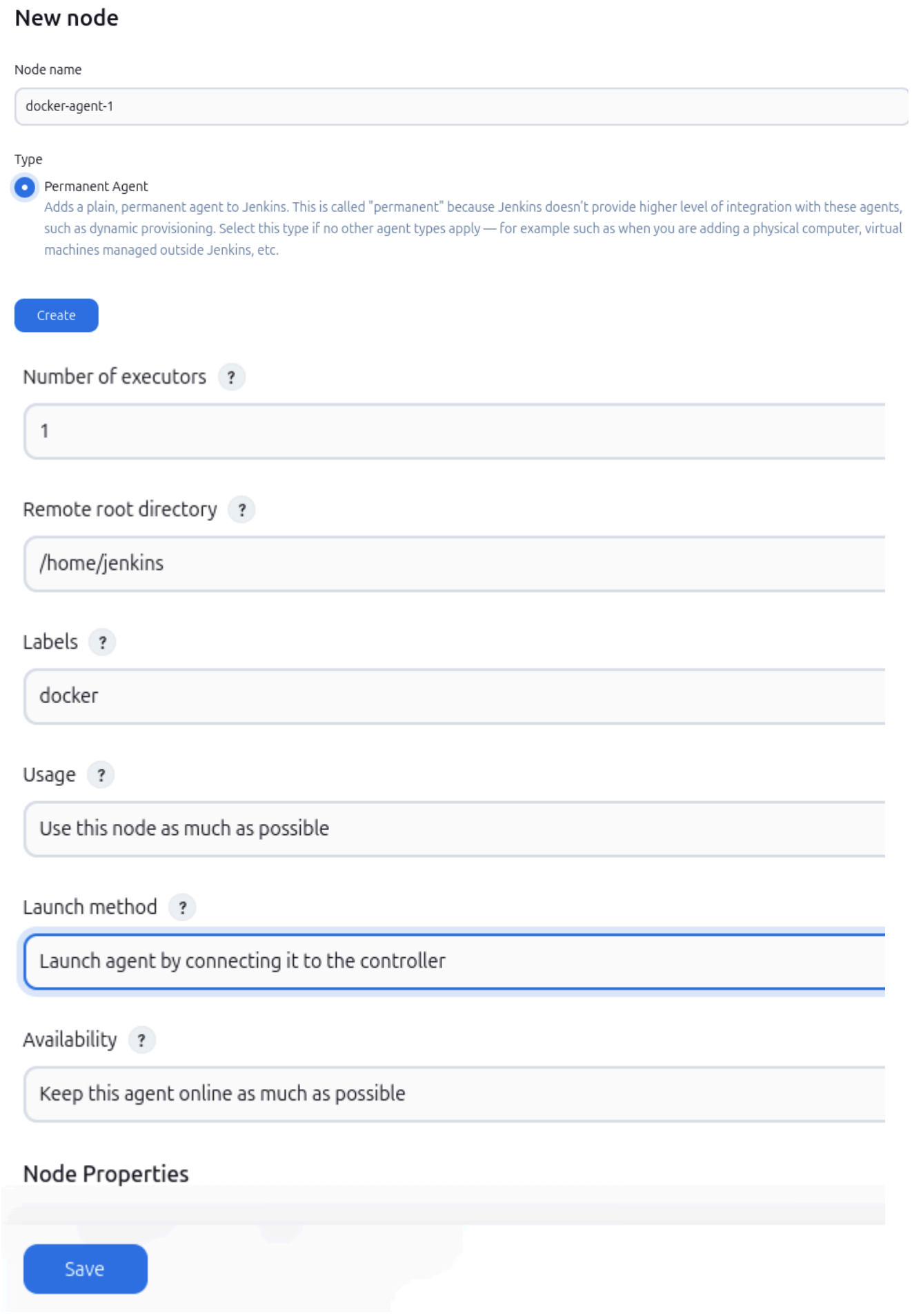



localhost:8080
ACloudGuru In...
ACloudGuru In...
ACloudGuru In...
Docker
Dodawanie pierwszego użytkownika
Stwórz pierwszego administratora
Login
admin
Hasło
Powtórz hasło
Pełna nazwa
Aleksander Drwal
Adres e-mail
adrwal@griddynamics.com
Jenkins 2.487
Pomiń i kontynuuj jako admin
Zapisz i zakończ

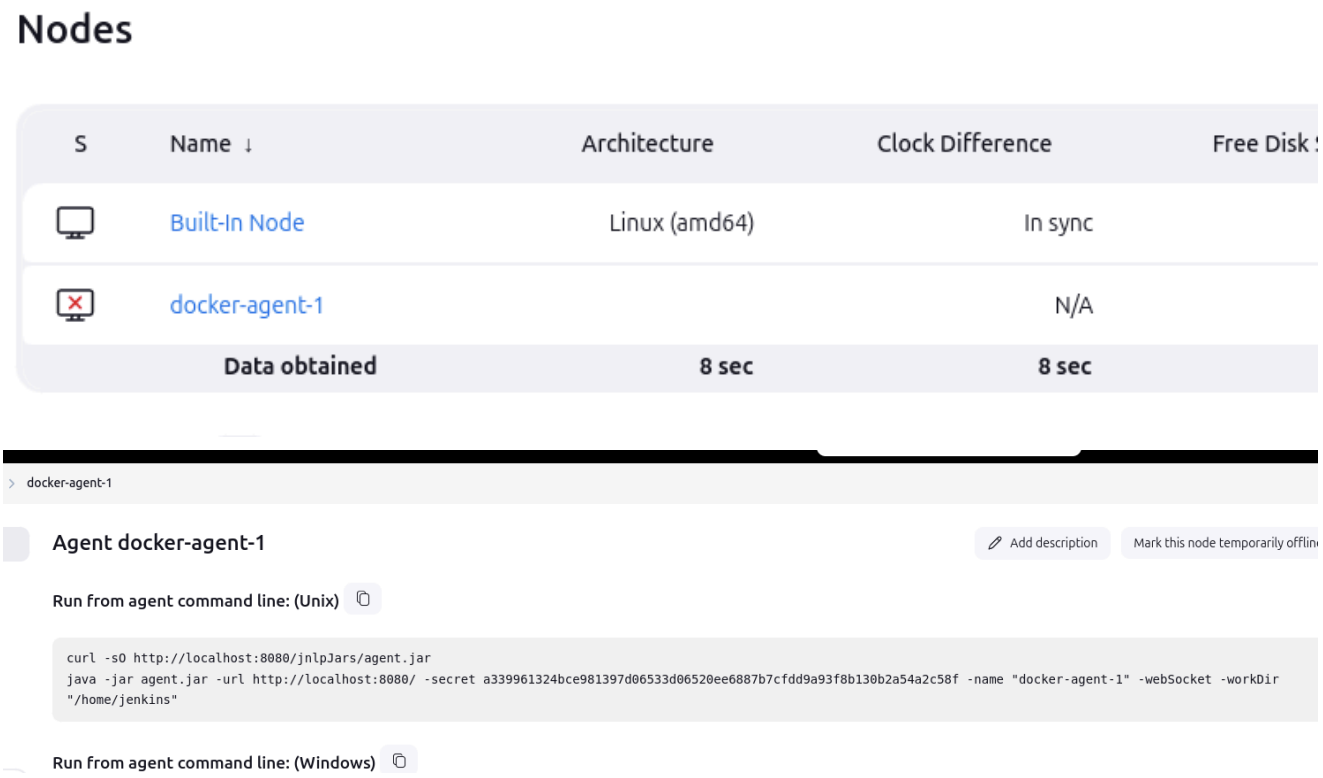

localhost:8080
ACloudGuru In...
ACloudGuru In...
ACloudGuru In...
Docker
Dodawanie pierwszego użytkownika
Konfiguracja instancji
URL Jenkinsa:
http://localhost:8080/
The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.
The proposed default value shown is not saved yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.
Jenkins 2.487
Nie teraz
Zapisz i zakończ



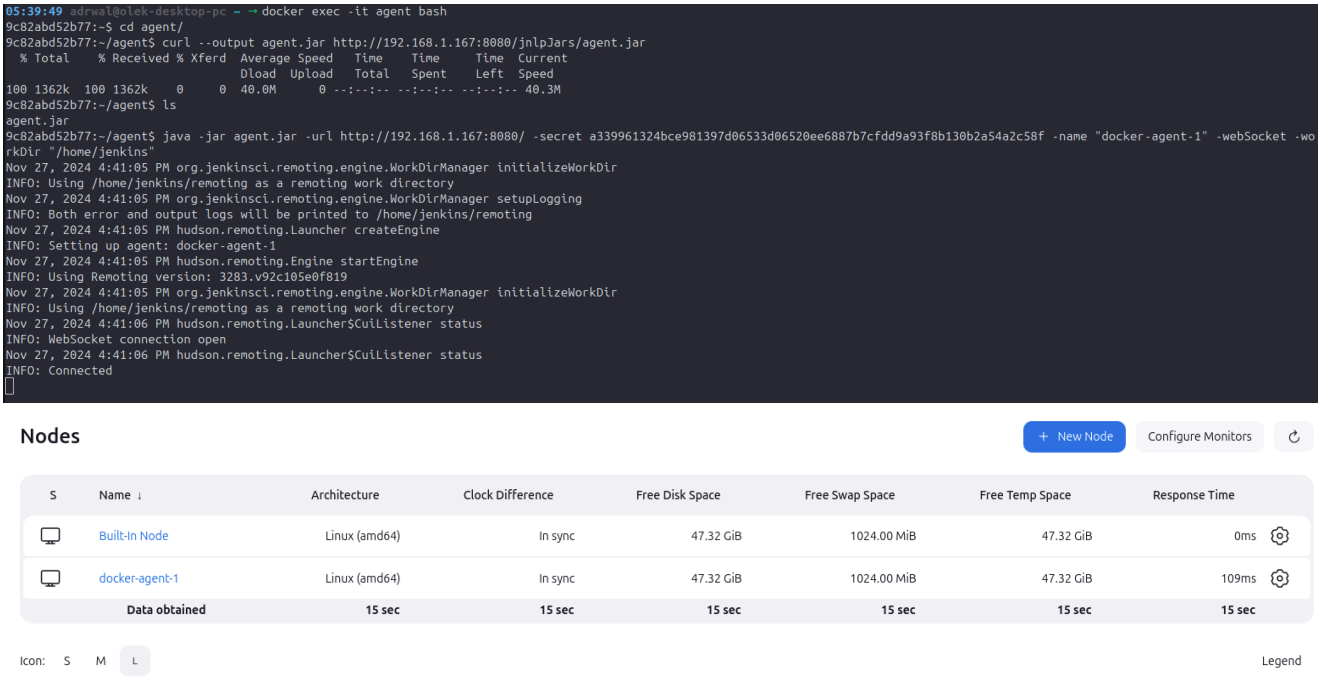
After the initial setup is complete, we need to add our Docker based agent. `Manage Jenkins > Nodes > + New Node`



After we click Save, our agent still isn't available. We need to launch `agent.jar` on it to make it available to Jenkins master.

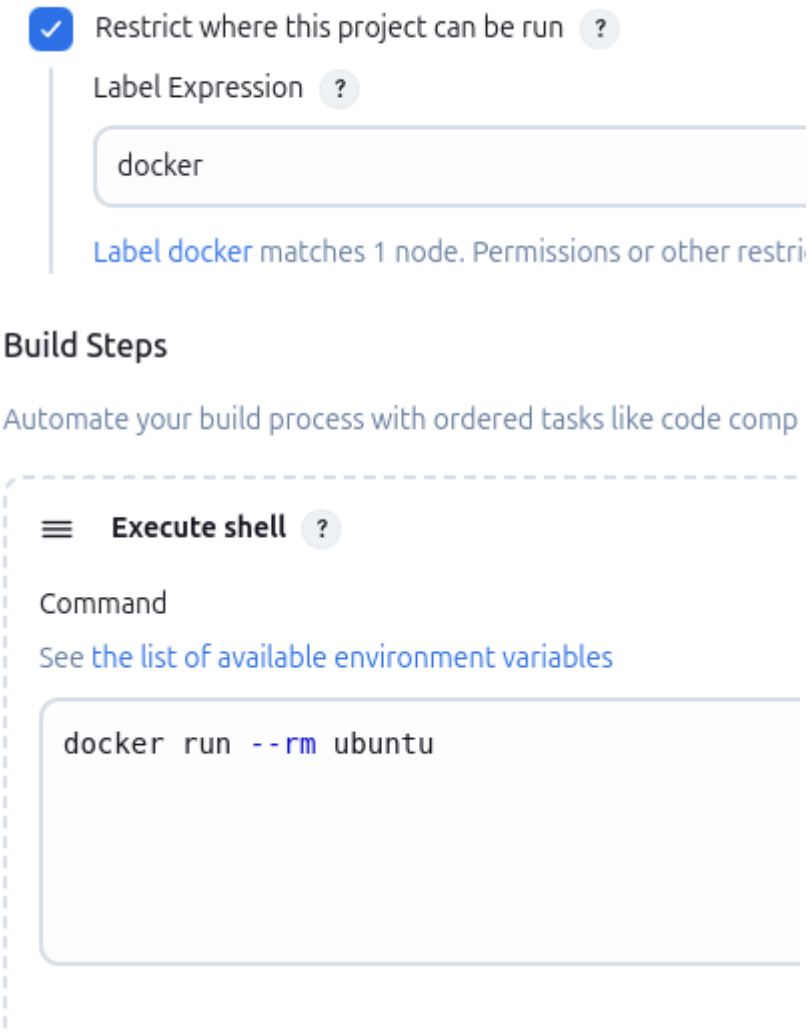


I changed localhost from above snippet to host's IP to access Jenkins master from inside of the agent's container.

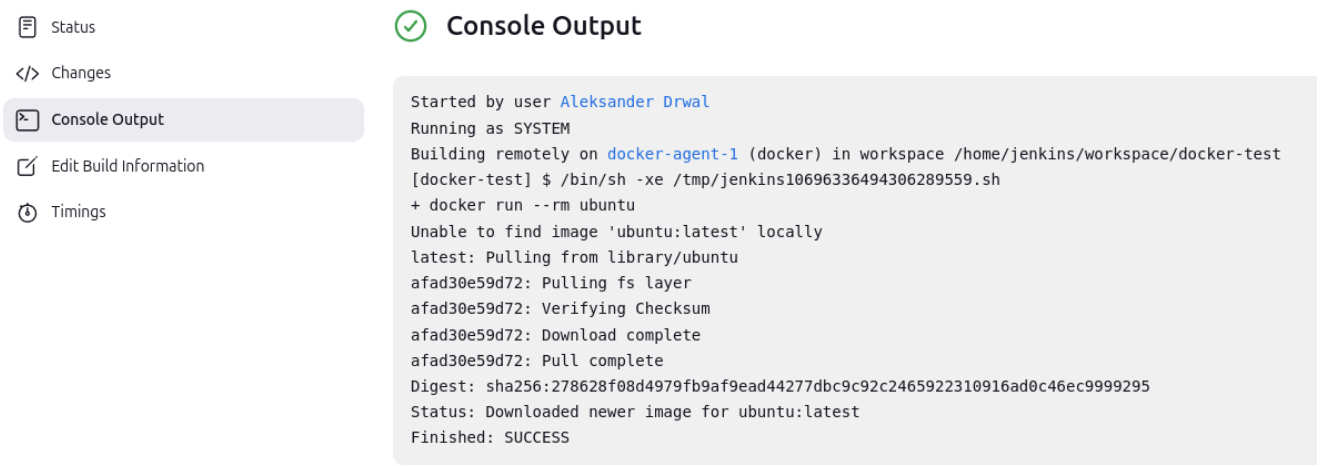


We can validate that this agent works as expected by running simple job on it which runs docker command `Create a job > Freestyle project`.

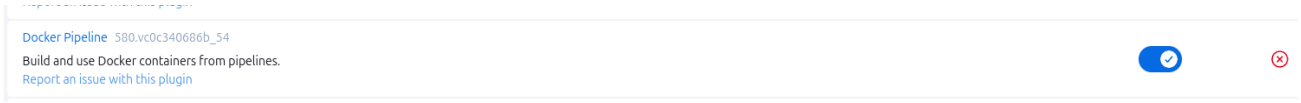
We leave everything default and just set 2 below fields:



After running the build, we can see that everything is working correctly.



We can also install Docker pipeline plugin for easier use of docker in pipelines.

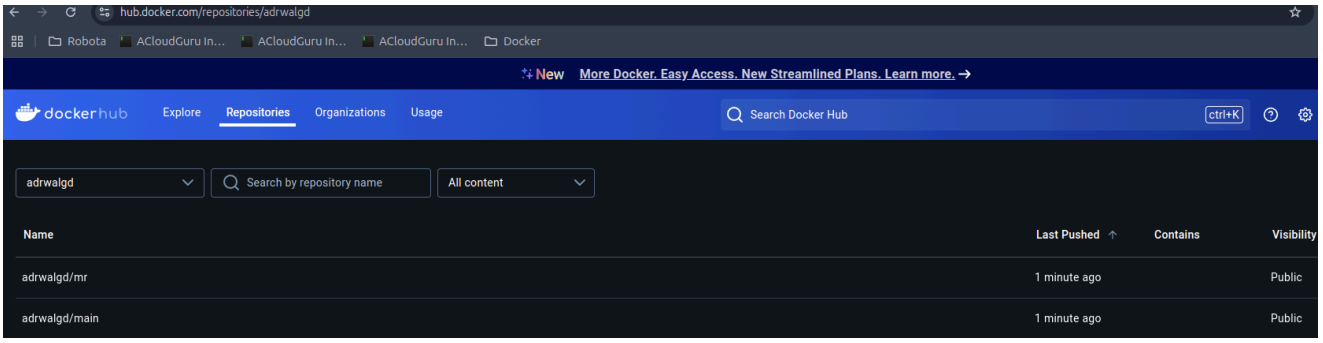


3. Add a Dockerfile to your repository with spring-petclinic.

Dockerfile contents:

```
FROM maven:3.8.7-openjdk-18-slim AS build

RUN mkdir /app
COPY . /app
WORKDIR /app
RUN mvn package

# Minimal rintime image - only JRE
FROM gcr.io/distroless/java21-debian12 AS runtime
COPY --from=build /app/target/*.jar /app.jar
ENTRYPOINT [ "java" ]
CMD [ "-jar", "/app.jar" ]
```

4. Create 2 docker repositories on your own Nexus Repository ([Instruction](#)) or https://hub.docker.com/ called “main” and “mr”.



5. Pipelines

a. merge request pipeline

First we need to add credentials for Github and Docker hub.
Create Github access token for Jenkins to setup webhooks/access private repository and add it to jenkins (`Dashboard > Manage Jenkins > Credentials > System > Global credentials > + Add Credentials`).

Note

jenkins-lab

What's this token for?

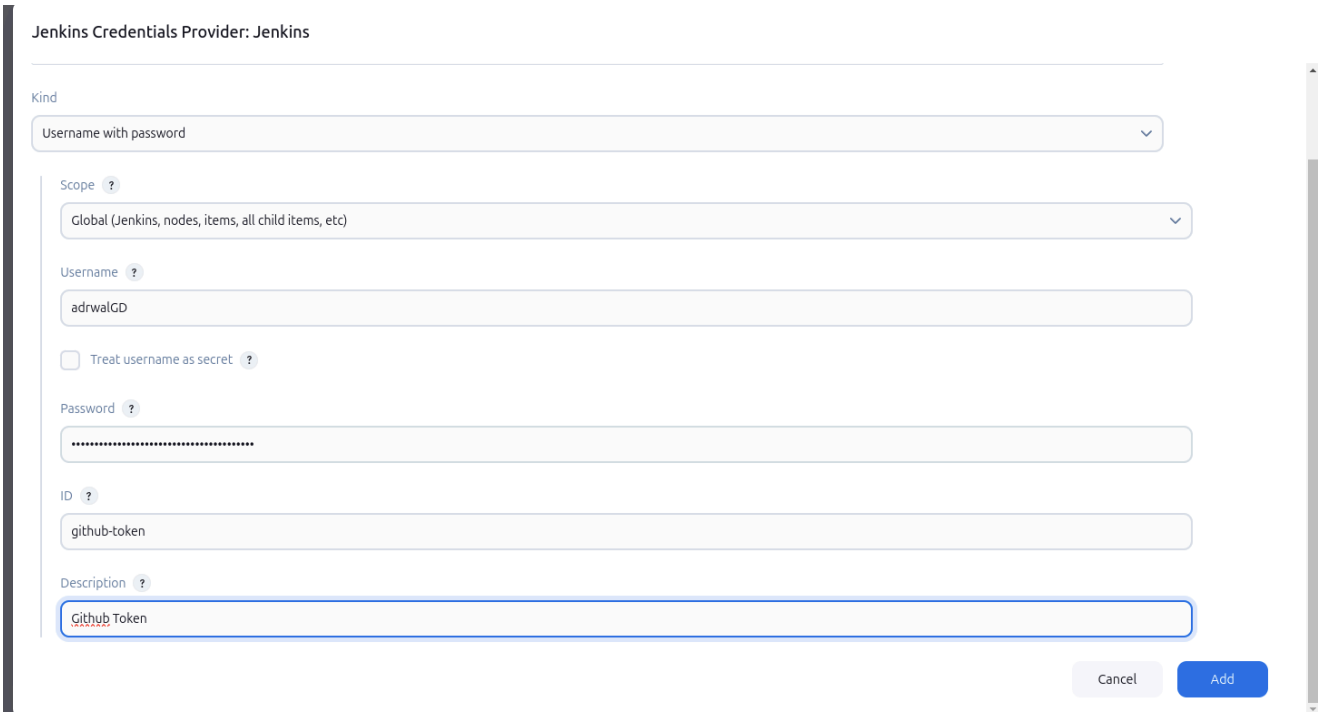
Expiration

This token expires *on Tue, Dec 3 2024*. To set a new expiration date, you must regenerate the token.

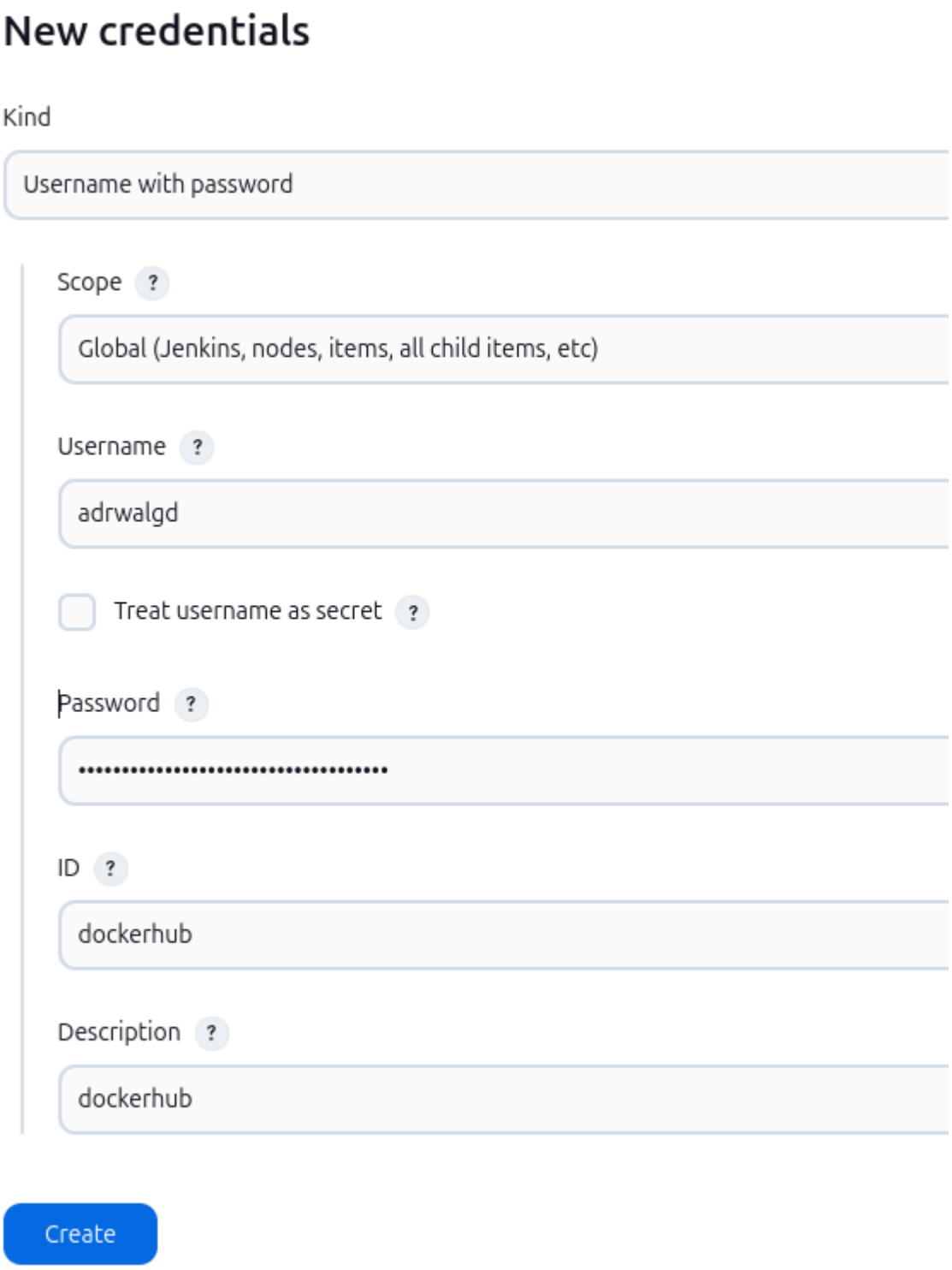
Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

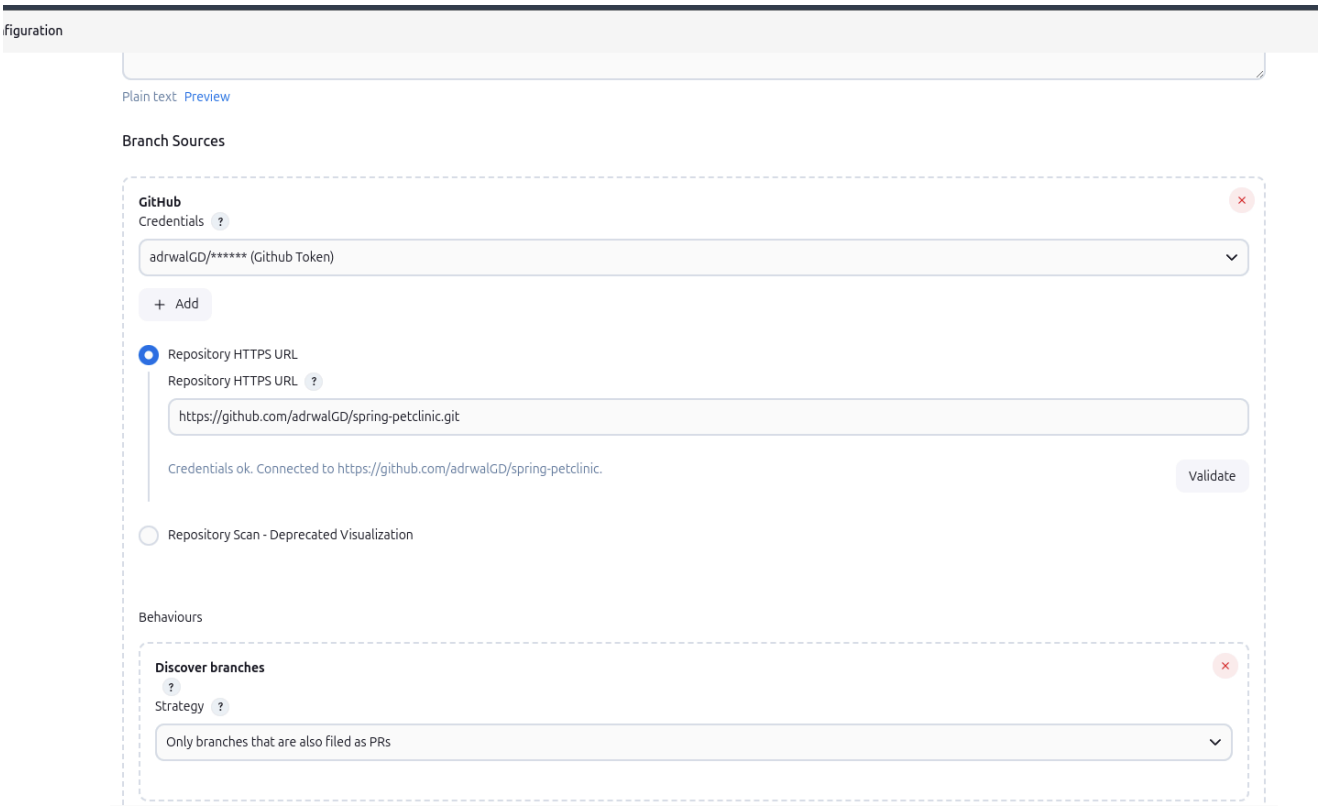
☐ **repo**	Full control of private repositories
☐ repo:status	Access commit status
☐ repo_deployment	Access deployment status
☐ public_repo	Access public repositories
☐ repo:invite	Access repository invitations
☐ security_events	Read and write security events
☐ **workflow**	Update GitHub Action workflows
☐ **write:packages**	Upload packages to GitHub Package Registry
☐ read:packages	Download packages from GitHub Package Registry
☐ **delete:packages**	Delete packages from GitHub Package Registry
☐ **admin:org**	Full control of orgs and teams, read and write org projects
☐ write:org	Read and write org and team membership, read and write org projects
☐ read:org	Read org and team membership, read org projects
☐ manage_runners:org	Manage org runners and runner groups
☐ **admin:public_key**	Full control of user public keys
☐ write:public_key	Write user public keys
☐ read:public_key	Read user public keys
☑ **admin:repo_hook**	Full control of repository hooks
☑ write:repo_hook	Write repository hooks
☑ read:repo_hook	Read repository hooks



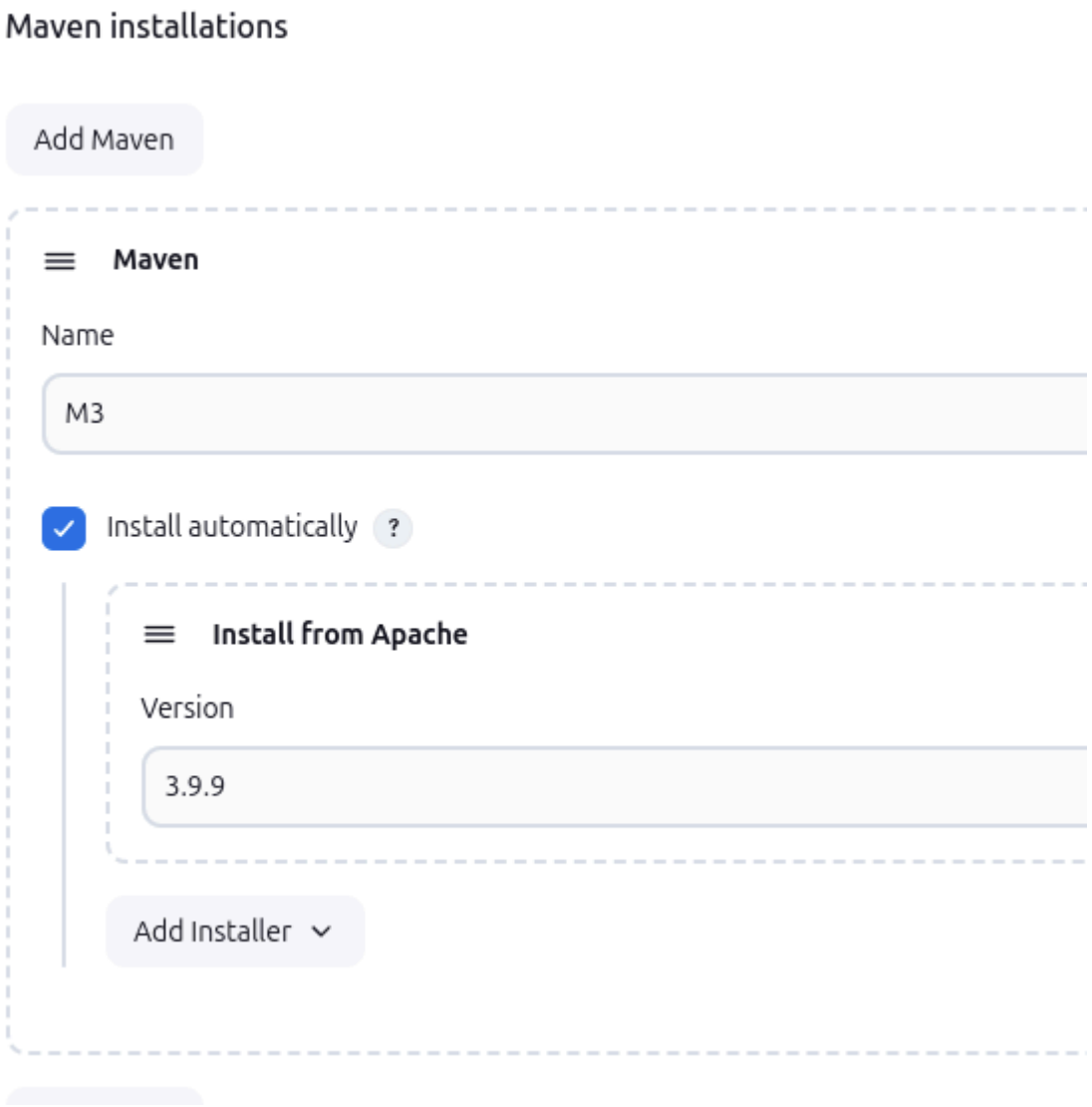
Add credentials for docker hub `Dashboard > Manage Jenkins > Credentials > System > Global credentials > + Add Credentials`. In password place access token generated on docker hub.



Now we create new project (`Dashboard > All > New item > Multibranch Pipeline`) and configure source as Github repository and set it to discover only PRs.



Because we will be running maven commands, we need to add maven to Jenkins `Dashboard > Manage Jenkins > Tools > Add Maven`:

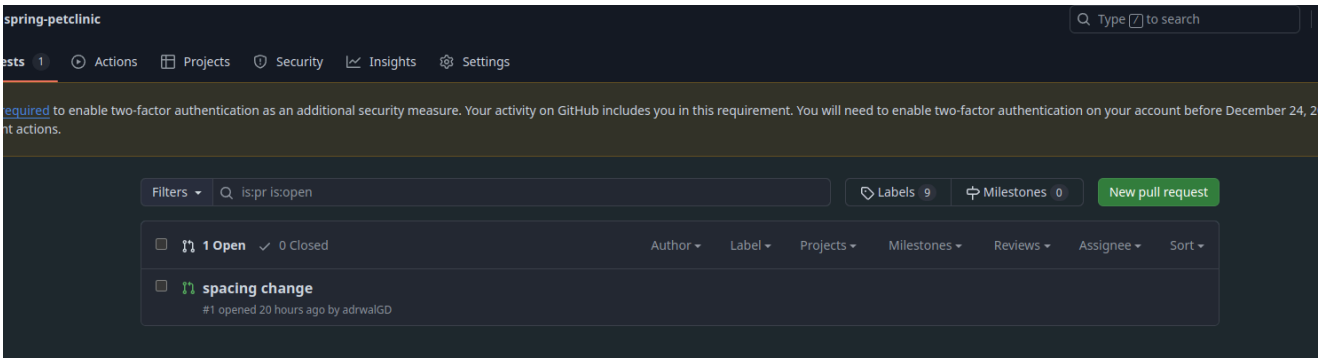


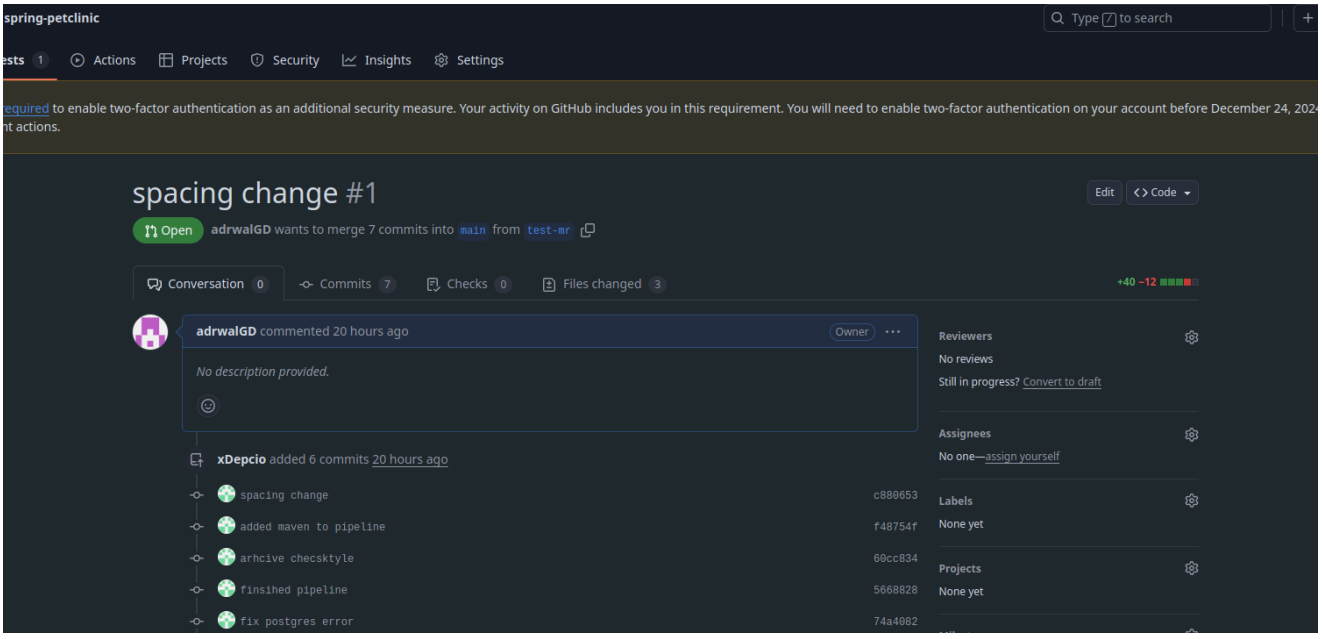
Creating actual pipeline logic

Below is pipeline to handle merge requests:

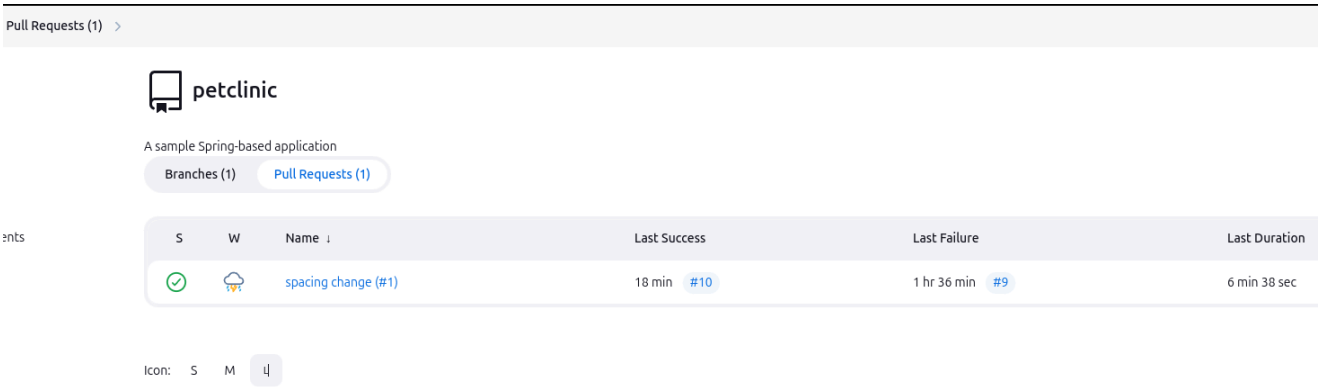
```
pipeline {
    agent {
        label 'docker'
    }
    tools {
        maven 'M3'
    }

    stages {
        stage('Checkstyle') {
            steps {
                sh 'mvn checkstyle:checkstyle'
                sh 'tar -czf checkstyle-result.tar.gz target/reports'
                archiveArtifacts artifacts: 'checkstyle-result.tar.gz', onlyIfSuccessful: true
            }
        }
        stage('Tests') {
            steps {
                sh 'mvn test'
            }
        }
        stage('Build without tests') {
            steps {
                sh 'mvn package -DskipTests'
            }
        }
        stage('Build docker image') {
            steps {
                script {
                    docker.build("adrwalgd/mr:$GIT_COMMIT")
                    docker.withRegistry('https://registry.hub.docker.com', 'dockerhub') {
                        docker.image("adrwalgd/mr:$GIT_COMMIT").push()
                    }
                }
            }
        }
    }
}
```

To test that everything is working correctly I created new pull request on Github with this pipeline already on the main branch (as well as on PR branch).





Back on Jenkins after scanning the repository we can see that branch containing the pull request has been detected and ran.



Checkstyle artifact has been archived:

spacing change (#1)

Full project name: petclinic/PR-1

Last Successful Artifacts

checkstyle-result.tar.gz 330.67 KiB view

Permalinks

- Last build (#10), 20 min ago
- Last stable build (#10), 20 min ago
- Last successful build (#10), 20 min ago
- Last failed build (#9), 1 hr 37 min ago
- Last unsuccessful build (#9), 1 hr 37 min ago
- Last completed build (#10), 20 min ago

And job has finished with success and built image has been pushed to docker hub:

Dashboard > petclinic > Pull Requests (1) > spacing change (#1) > #10

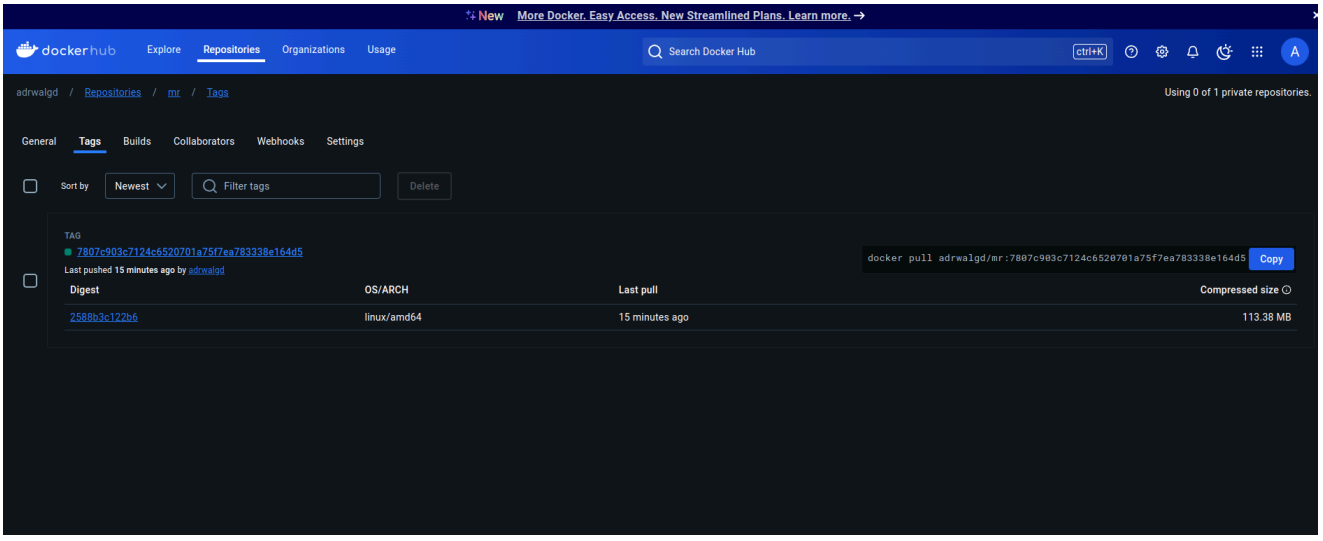
- Status
- Changes
- Console Output
- Edit Build Information
- Delete build '#10'
- Timings
- Git Build Data
- Pipeline Overview
- Pipeline Console
- Restart from Stage
- Replay
- Pipeline Steps
- Workspaces
- Previous Build

Console Output

Skipping 1,741 KB.. Full Log

```
#11 235.9 Progress (1): 262/334 kB
Progress (1): 266/334 kB
Progress (1): 270/334 kB
Progress (1): 274/334 kB
Progress (1): 278/334 kB
Progress (1): 282/334 kB
Progress (1): 286/334 kB
Progress (1): 290/334 kB
Progress (1): 294/334 kB
Progress (1): 298/334 kB
Progress (1): 303/334 kB
Progress (1): 307/334 kB
Progress (1): 311/334 kB
Progress (1): 315/334 kB
Progress (1): 319/334 kB
Progress (1): 323/334 kB
Progress (1): 327/334 kB
Progress (1): 331/334 kB
Progress (1): 334 kB

Downloaded from central: https://repo.maven.apache.org/maven2/com/
#11 236.1 [INFO] Recompiling the module because of changed depende
#11 236.1 [INFO] Compiling 24 source files with javac [debug param
#11 236.8 [INFO]
#11 236.8 [INFO] --- maven-resources-plugin:3.3.1:testResources (d
```

```
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withDockerRegistry
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline

Could not update commit status, please check if your scan
selected


GitHub has been notified of this commit's build result

Finished: SUCCESS
```



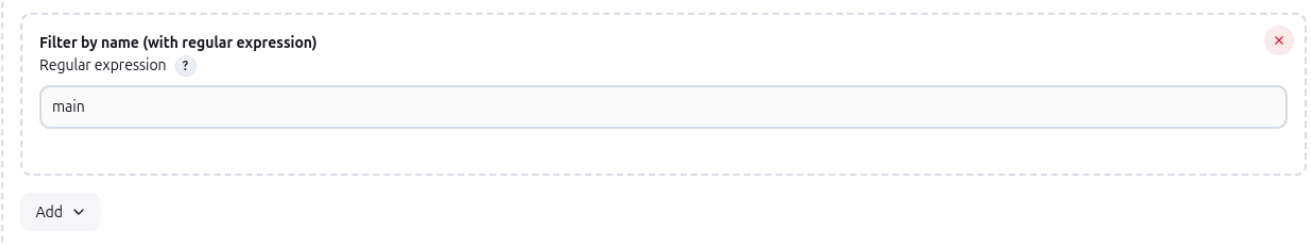
b. pipeline for main branch

To handle updates for the main branch correctly we need to:

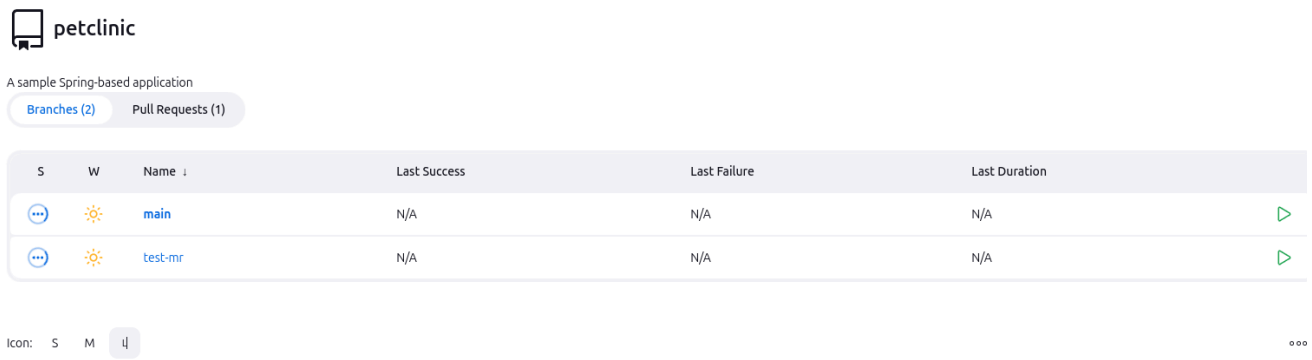
1. Update project configuration in Jenkins to also detect main branch.
2. Update pipeline to run only Docker build and push stage on main branch.

1. Update project configuration in Jenkins

In `Dashboard > [project name] > Configuration` we add new source with same credentials and repo URL, but different branch discovery strategy (only branch named main).



After saving we can see that now also branch main has been detected:



2. Update pipeline

I updated Jenkinsfile on main branch and rebased branch with PR on top of main so that now both of them have the same Jenkinsfile, but we can still see the differences between build steps on PR branch and main.

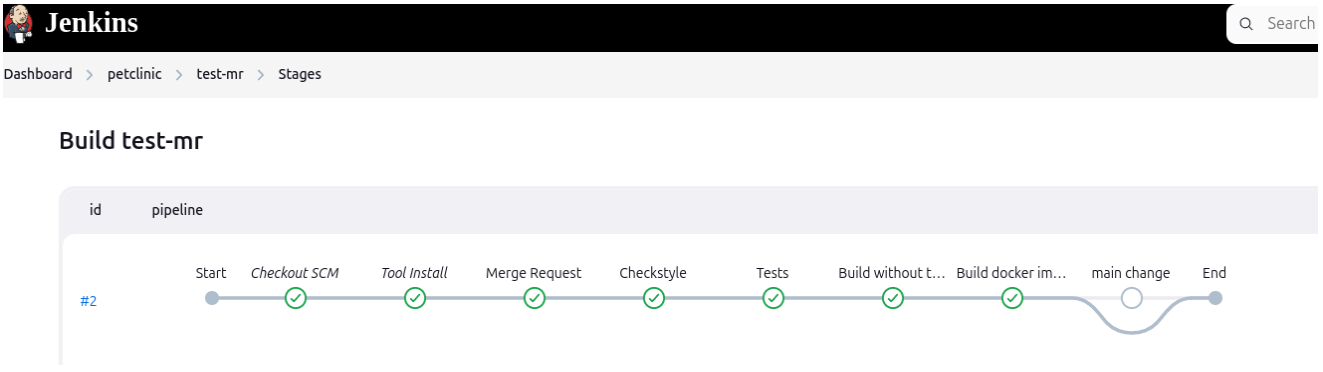
In updated Jenkisfile we run one of two stages based on the branch name:

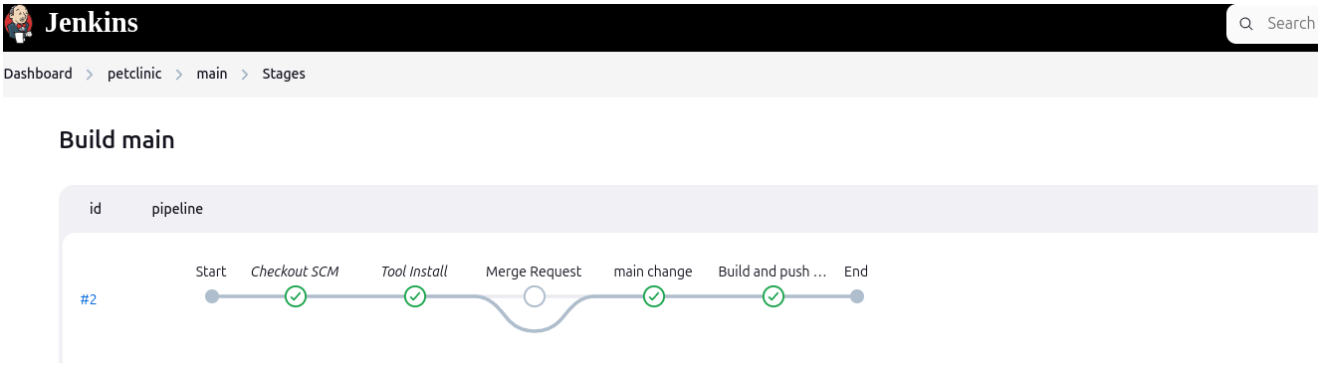
```
pipeline {
    agent {
        label 'docker'
    }
    tools {
        maven 'M3'
    }

    stages {
        stage('Merge Request') {
            when {
                expression {
                    env.BRANCH_NAME != 'main'
```

```
                }
            }
            stages {
                stage('Checkstyle') {
                    steps {
                        sh 'mvn checkstyle:checkstyle'
                        sh 'tar -czf checkstyle-result.tar.gz
target/reports'
                        archiveArtifacts artifacts: 'checkstyle-
result.tar.gz', onlyIfSuccessful: true
                    }
                }
                stage('Tests') {
                    steps {
                        sh 'mvn test'
                    }
                }
                stage('Build without tests') {
                    steps {
                        sh 'mvn package -DskipTests'
                    }
                }
                stage('Build docker image') {
                    steps {
                        script {
                            docker.build("adrwalgd/mr:$GIT_COMMIT")

docker.withRegistry('https://registry.hub.docker.com', 'dockerhub') {

docker.image("adrwalgd/mr:$GIT_COMMIT").push()
                            }
                        }
                    }
                }
            }
        }

        stage('main change') {
            when {
                branch 'main'
            }
            stages {
                stage('Build and push Docker Image') {
                    steps {
                        script {
                            docker.build("adrwalgd/main:$GIT_COMMIT")

docker.withRegistry('https://registry.hub.docker.com', 'dockerhub') {

docker.image("adrwalgd/main:$GIT_COMMIT").push()
```

```
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
```

After running the build in Jenkins we can validate that correct stages were run for specific branches:





And that Docker images has been pushed to mr and main repository:

adrwalgd / Repositories / mr / Tags

General | Tags | Builds | Collaborators | Webhooks | Settings

Sort by Newest | Filter tags | Delete

TAG
e0063074c8a2b0449179ae17262eec08cd6411f5
Last pushed **19 hours ago** by adrwalgd

Digest	OS/ARCH	Last pull
c5cc72e6858d	linux/amd64	---

TAG
7807c903c7124c6520701a75f7ea783338e164d5
Last pushed **19 hours ago** by adrwalgd

Digest	OS/ARCH	Last pull
2588b3c122b6	linux/amd64	20 hours ago

adrwalgd / Repositories / main / Tags

General | Tags | Builds | Collaborators | Webhooks | Settings

Sort by Newest | Filter tags | Delete

TAG
ffc3df21d58d78b892515062c9d207facee357ae
Last pushed **19 hours ago** by adrwalgd

Digest	OS/ARCH	Last pull
f103ea66e5b3	linux/amd64	19 hours ago