Automation of the CI/CD process for the Python application

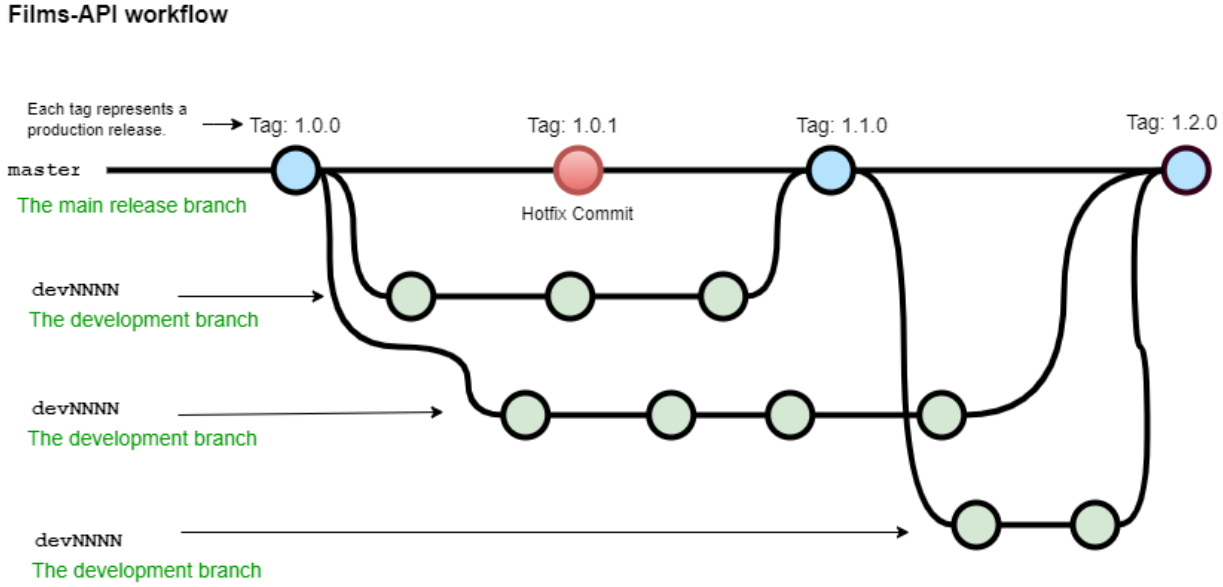
Growing demand for digital transformation requires constant professional development and continuing education of the IT specialists. That is the reason why I have enrolled in the course Epam DevOps Autumn 22 L1 that gives the opportunity to get acquainted and practically apply modern DevOps tools.

The project aims to automate the testing and deployment of my application. My application is a Python learning project: an implementation of a certain API on Flask for working with movie information (https://github.com/smokiei/films_api). The following business processes (simplified for learning purposes) have been described for the initial application:

1. The source code of the application on Python is in the repository on Github

2. There is the following branch strategy in Github:

- the master branch from which releases are made
- task branches dev-NNNN, where NNNN is the number of the task in Jira



Dev-NNNN branches are created from the master branch on demand. Each branch is linked to a certain task. Upon task completion, developers make a pull request to the master branch. The leading developer reviews the pull request to the master branch and performs a merge.

3. Tests of two types are created for the code base - the smoke test and module tests with stubs from the unittest library, using the “pytest” library.

4. Upon a release, the master branch must be compiled into a docker image and uploaded to Amazon Elastic Container Registry for further manual testing and deployment to the production server.

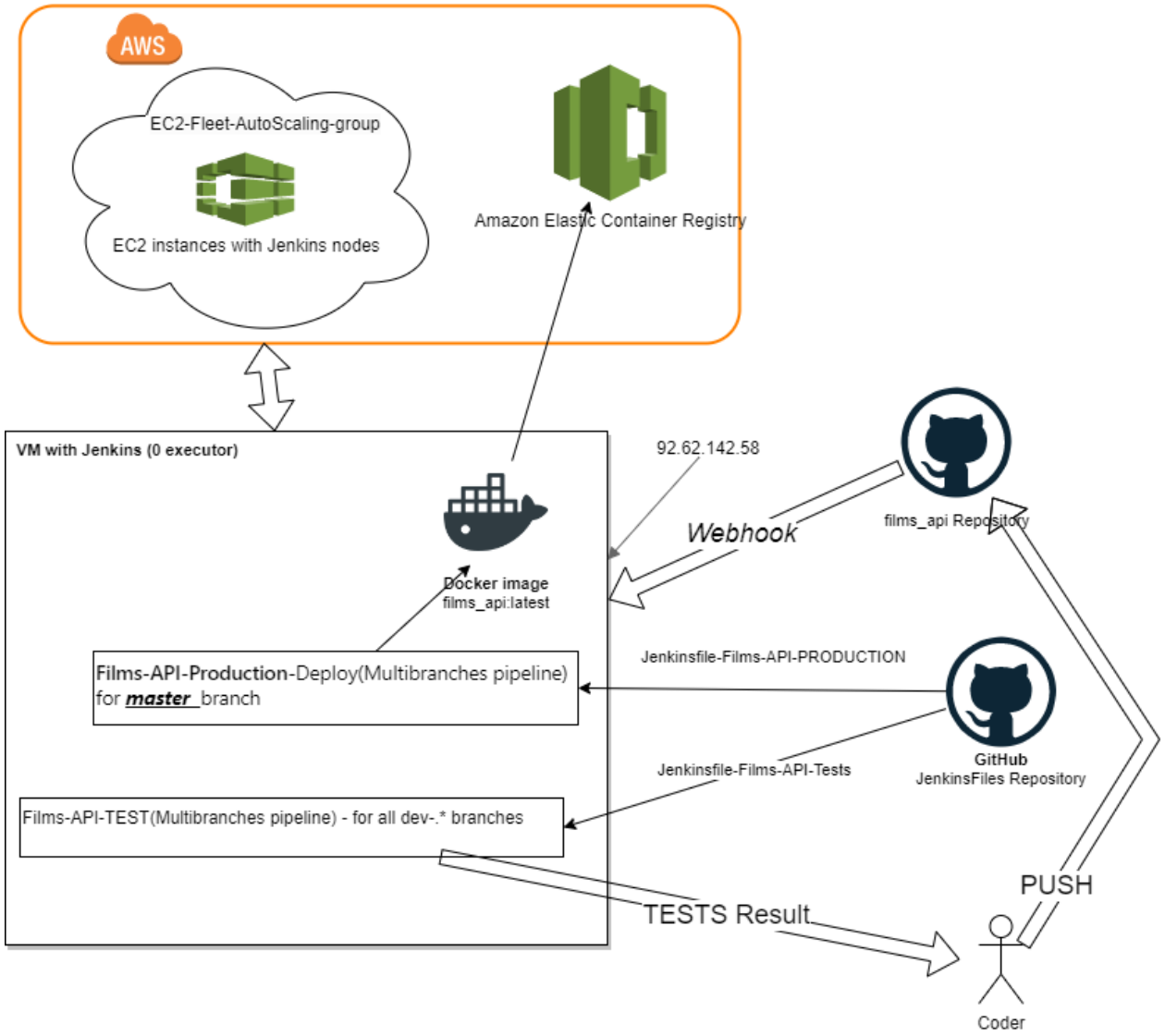
The project is considered being successful when stages 3 and 4 are automated and checked.

Project implementation

The project has been implemented using Jenkins as the main tool. The Jenkins server has been deployed on my rented server running Ubuntu. All Jenkins pipeline files have been stored in a separate private git repository to ensure code independence from the pipeline. To do this, the Remote Jenkins file provider plugin has been chosen.

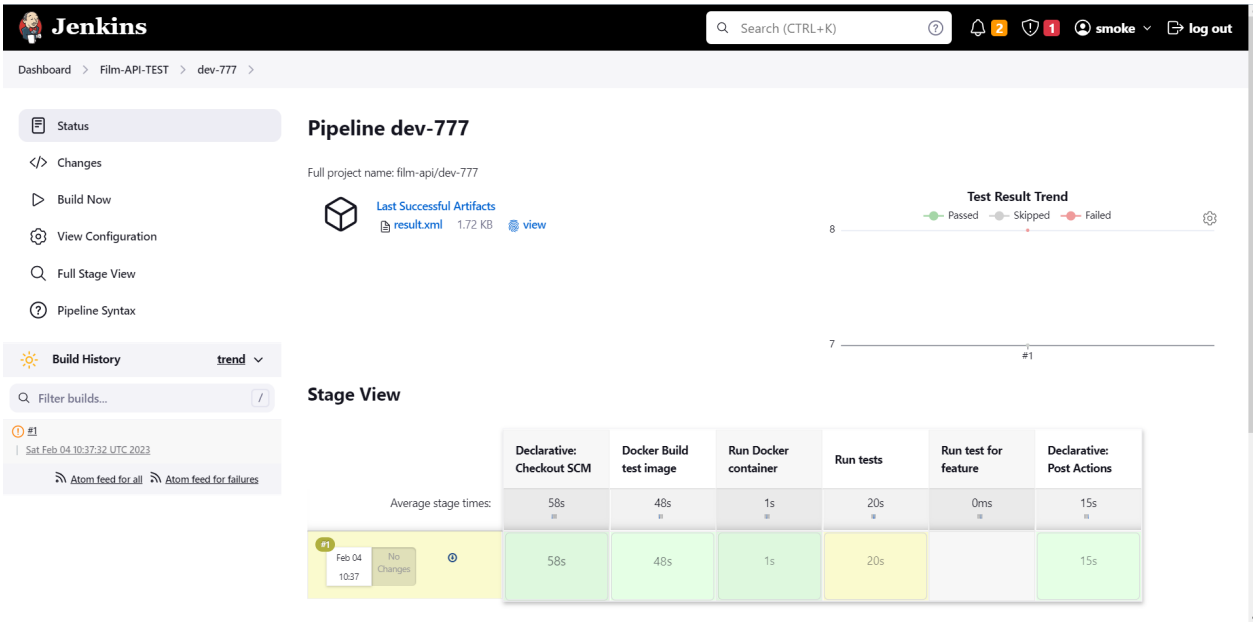
In accordance with best practices, Jenkins pipelines are executed on nodes. The nodes have been set up to be dynamically launched on Amazon EC2 using the Amazon EC2 Fleet plugin. The nodes have been configured as follows: on the Jenkins server # of executors = 0, and the Amazon EC2 Fleet Cloud was configured for a maximum of 1 and a minimum of 0 executors. This configuration corresponds to the load - several builds per day and does not consume resources during idle time. The speed of starting the build and the time it takes are not critical in this project.

Github is configured with webhooks on the Jenkins server so that it could automatically trigger pipelines when changes are made to the git repository.



Multibranch Pipeline has been created for the automation of testing in dev-NNNN branches. The source of branches is git with SSH key access, a filter by branch name (dev.*) is set. Docker has been chosen as an environment to run tests in, the obtained artifact is a report to be published or sent in any way.

Example of pipeline work on one of the branches:

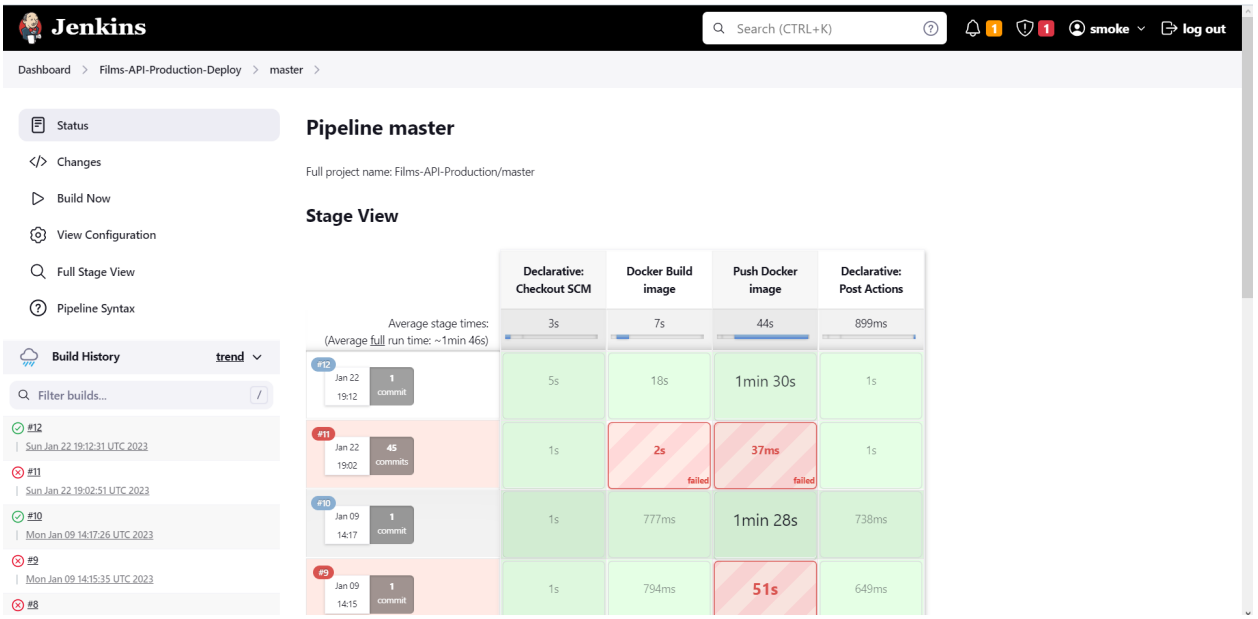


Jenkinsfile of that pipeline:

```
pipeline {
  agent any
  stages {
      stage('Docker Build test image') {
          steps {
              sh 'docker build -t film-api-tests-image -f Dockerfile-Tests .'
          }
      }
      stage('Run Docker container') {
          steps {
              sh 'docker run -d --tty --network=host --name film-api-tests-container film-api-tests-image'
          }
      }
      stage('Run tests') {
          steps {
              sh 'docker exec -d film-api-tests-container sh -c "pytest -v --junitxml=result.xml"'
              sh 'rm -f result.xml'
              sh 'docker cp film-api-tests-container:/films_api/result.xml .'
              junit 'result.xml'
              archiveArtifacts artifacts: 'result.xml'
          }
      }
      stage('Run test for feature') {
          when { branch 'dev-feature' }
          steps {
              sh 'docker exec -d film-api-tests-container sh -c "pytest -v --junitxml=result-feature.xml"'
              sh 'rm -f result-feature.xml'
              sh 'docker cp film-api-tests-container:/films_api/result-feature.xml .'
              junit 'result-feature.xml'
              archiveArtifacts artifacts: 'result-feature.xml'
          }
      }
  }
  post {
      always {
          script {
              sh 'docker stop film-api-tests-container'
              sh 'docker rm film-api-tests-container'
              sh 'docker rmi film-api-tests-image'
          }
      }
  }
}
```

To automate the creation of a release Docker Image from the master branch, a Multibranch Pipeline has been also created. The source of the branches is git with SSH key access. The filter is set to use only a master branch name. This pipeline was decided to be done using the Docker Pipeline plugin.

Example of the pipeline's work:



Jenkinsfile of that pipeline:

```
pipeline {
  environment {
    imagename = "films_api"
    ecrurl = "https://858631253727.dkr.ecr.eu-central-1.amazonaws.com/"
    ecrcredentials = "ecr:eu-central-1:XXXXXXXXXX"
    dockerImage = ''
  }
  agent any
  stages {
    stage('Docker Build image') {
      steps {
        script { dockerImage = docker.build imagename + ":latest" }
      }
    }
    stage('Push Docker image') {
      steps {
        script {
          docker.withRegistry(ecrurl, ecrcredentials) { dockerImage.push('latest') }
        }
      }
    }
  }
  post {
    always {
      script {
        sh "docker rmi $imagename:latest"
      }
    }
  }
}
```

Dockerfile for the creating final image

```
FROM python:3.8
MAINTAINER smoke
RUN useradd --create-home userapi
WORKDIR /films_api
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY ./ .
RUN chown -R userapi:userapi ./
USER userapi
EXPOSE 5001
CMD gunicorn --bind 0.0.0.0:5001 wsgi:app
```

Conclusions

Process of testing and building the Docker Image has been automated, that is the goal is achieved. Ideas for further development of the project:

- Setting up emailing of artifacts from testing
- Reducing the size of the Docker Image.