CI/CD java application

To consolidate knowledge, in the Final Project I attempted to utilize all DevOps tools which was studied in the course, such as Terraform, Ansible, Jenkins, and Docker and build infastructure on AWS.

In this project, I attempted to describe the entire software development process (in a simplified form) as I understand it after listening to the course.

The goal is to provide automatic delivery of the Java application to the end-user, eliminate downtime, and ensure failure tolerance.

Terraform: https://github.com/edmitrenko/final_project.git
Ansible: https://github.com/edmitrenko/ansible-roles.git
Application: https://github.com/edmitrenko/jenkins-java-app2.git

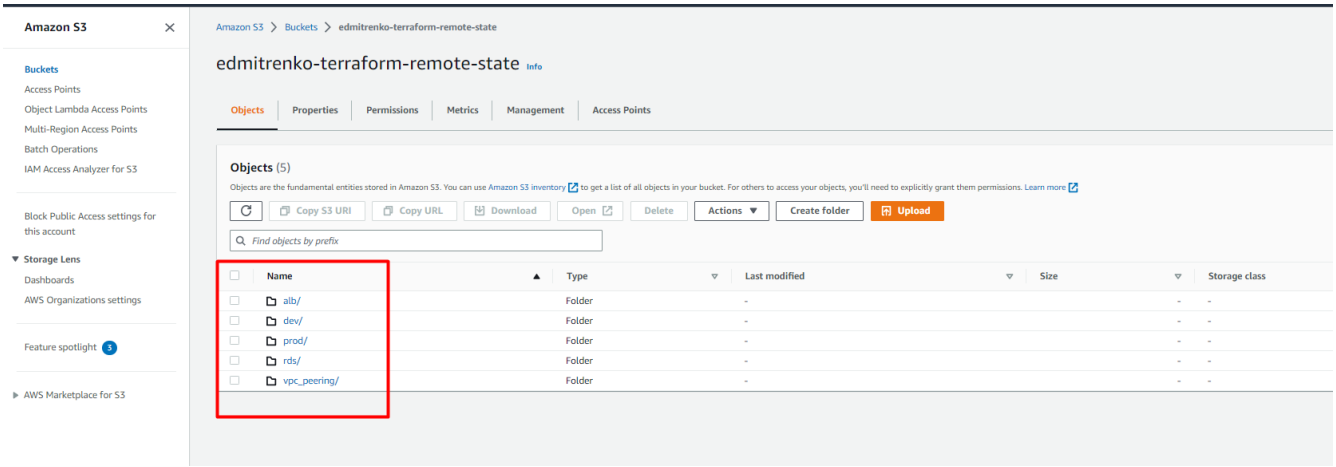
I have used **Terraform** to build an infrastructure on AWS.
Ansible for configuring Jenkins and Tomcat on infrastructure servers.
Jenkins - for CI/CD Java applications to Tomcat.

Build infrastructure for the final project on AWS using Terraform.

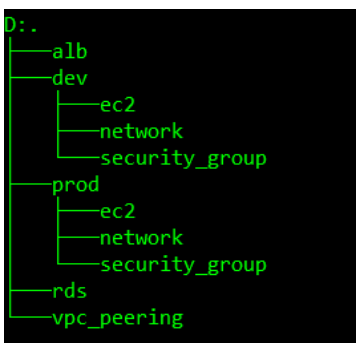
I use Terraform modules that are saved on GitHub.

https://github.com/edmitrenko/final_project.git

Files of Terraform state will be saved in an AWS S3 Bucket. The S3 Bucket was specially created on AWS with this structure:



Structure of Terraform folders on my laptop:

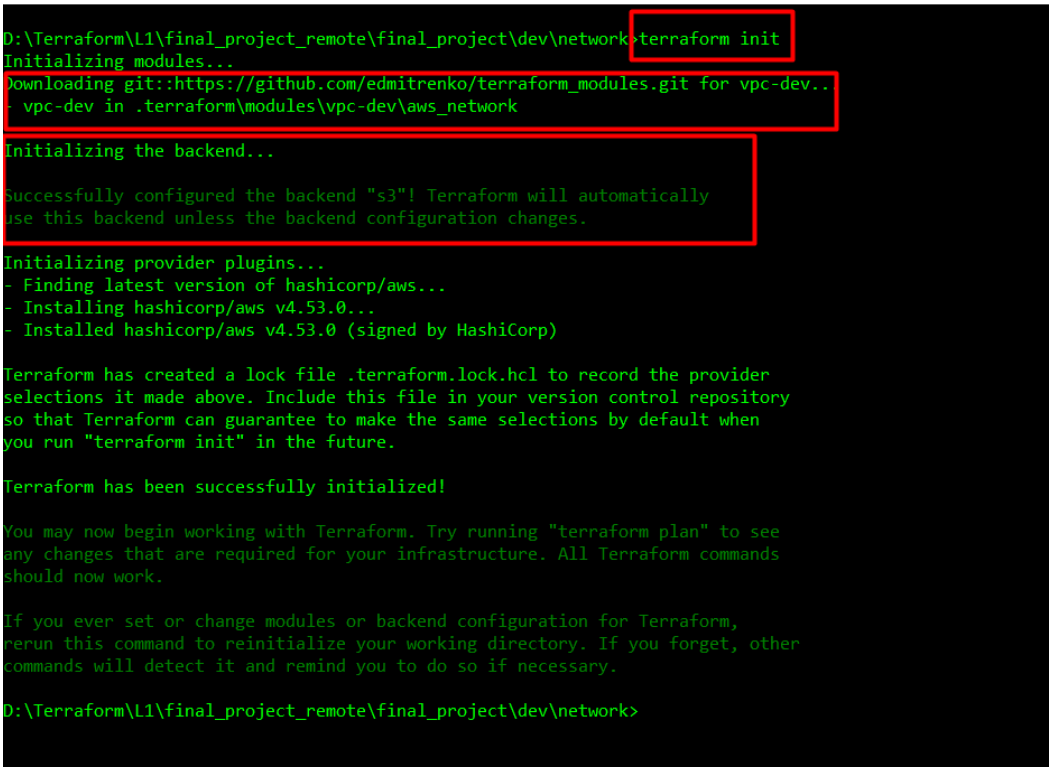


Build infrastructure for development environment.

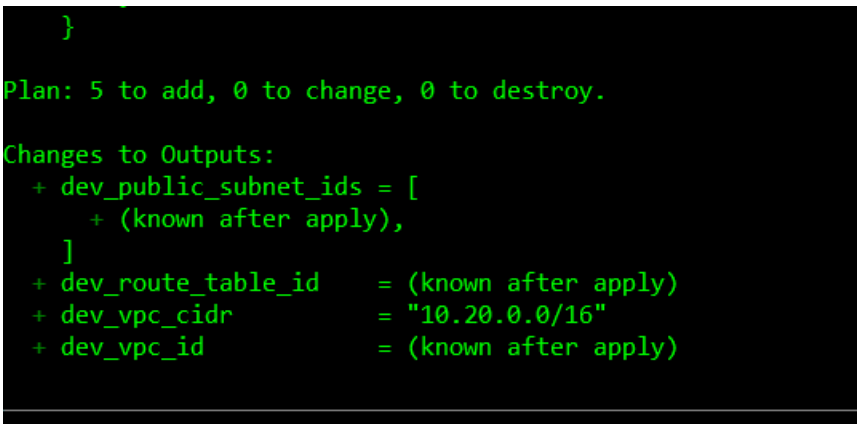
Configure network infrastructure for development environment:

Go to the "dev/network" folder containing Terraform configuration files to create the network. Then, run the following command:

‘terraform init’ to initialize the modules and set up the backend for saving the state, and ensure that the necessary Terraform provider is loaded:



‘terraform plan’ – we can see what will be done



‘terraform apply’ - to apply future changes. To confirm, enter 'YES' and press Enter (you can use the **‘terraform apply -auto-approve’** command for automation). The output of the terraform apply command will be displayed. The most important information is stored in the outputs.tf file:

```
Outputs:

dev_public_subnet_ids = [
  "subnet-08fa4d2d1b34fb4a5",
]
dev_route_table_id = "rtb-04d17de2173e77a0b"
dev_vpc_cidr = "10.20.0.0/16"
dev_vpc_id = "vpc-0c1439b78911868dd"
```

As a result we have network for development environment.

Configure security group for development environment

Go to the dev/security_group folder with Terraform configuration files to create the security groups in the dev environment and launch the commands:

terraform init
terraform apply

As a result security groups were created for development environment.

Configure servers for development environment:

Go to the dev/ec2 folder with Terraform config files for creating EC2 servers in the dev environment and launch the commands:

terraform init
terraform apply

As a result, servers were created for the dev environment. In the output, we get the result of the terraform apply command. The most important information is entered in the outputs.tf file:

```
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

dev-ansible_arn = "arn:aws:ec2:eu-west-2:860300666219:instance/i-089066eb144f44844"
dev-ansible_id = "i-089066eb144f44844"
dev-ansible_private_ip = "10.20.1.242"
dev-ansible_public_ip = "18.130.175.71"
dev-jenkins_arn = "arn:aws:ec2:eu-west-2:860300666219:instance/i-040f47875614843b4"
dev-jenkins_id = "i-040f47875614843b4"
dev-jenkins_private_ip = "10.20.1.178"
dev-jenkins_public_ip = "35.177.60.28"
dev-tomcat_arn = "arn:aws:ec2:eu-west-2:860300666219:instance/i-05a0ba901f1a91490"
dev-tomcat_id = "i-05a0ba901f1a91490"
dev-tomcat_private_ip = "10.20.1.10"
dev-tomcat_public_ip = "18.135.96.70"
```

As a result, we have three servers:

dev-jenkins - CI/CD Jenkins server from which Pipelines will be launched

dev-ansible - the server on which the Configuration Management Tool (Ansible) will be installed and used for the initial configuration of the prod environment. This server will also be added as the second node of Jenkins, and Docker will be installed on it to run future pipelines in Docker. The required software (Java, Docker, Ansible) will be automatically installed using the user_data script during the server configuration in AWS EC2.
user_data script:

```
locals {
user_data_amazon = <<-EOT
  #!/bin/bash
  amazon-linux-extras install -y ansible2 java-openjdk11
  yum update -y
  yum install -y git docker
  usermod -aG docker ec2-user
  EOT
}

locals {
user_data_ubuntu = <<-EOT
  #!/bin/bash
  apt install -y apt-transport-https ca-certificates curl software-properties-common
  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
  add-apt-repository -y "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
  add-apt-repository --yes --update ppa:ansible/ansible
  apt update
  apt install -y software-properties-common
  apt install -y ansible default-jdk git docker-ce
  usermod -aG docker ubuntu
  EOT
}
```

dev-tomcat – machine for the Tomcat server in dev environment

I have created the SSH key in advance in the appropriate AWS region

Build infrastructure for the Production environment.

Configure network infrastructure for prod environment

Go to the prod/network folder with Terraform configuration files to create the network and launch the commands:

terraform init
terraform apply

As a result, the network was created for the prod environment.

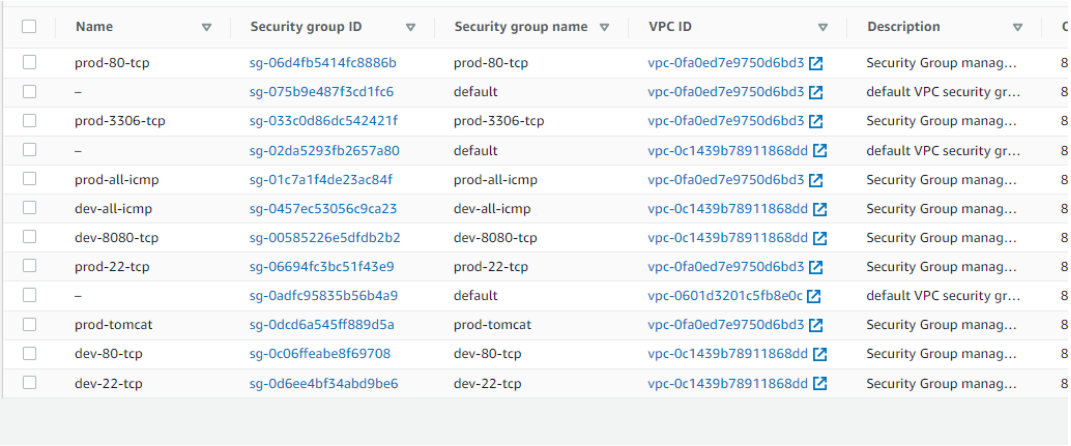
Configure security group for the production environment

Go to the prod/security_group folder with Terraform config files for creating security groups in the prod environment and launch the commands:

terraform init

terraform apply

As a result, security groups were created for the prod environment:



Name	Security group ID	Security group name	VPC ID	Description
prod-80-tcp	sg-06d4fb5414fc8886b	prod-80-tcp	vpc-0fa0ed7e9750d6bd3	Security Group manag...
–	sg-075b9e487f3cd1fc6	default	vpc-0fa0ed7e9750d6bd3	default VPC security gr...
prod-3306-tcp	sg-033c0d86dc542421f	prod-3306-tcp	vpc-0fa0ed7e9750d6bd3	Security Group manag...
–	sg-02da5293fb2657a80	default	vpc-0c1439b78911868dd	default VPC security gr...
prod-all-icmp	sg-01c7a1f4de23ac84f	prod-all-icmp	vpc-0fa0ed7e9750d6bd3	Security Group manag...
dev-all-icmp	sg-0457ec53056c9ca23	dev-all-icmp	vpc-0c1439b78911868dd	Security Group manag...
dev-8080-tcp	sg-00585226e5dfdb2b2	dev-8080-tcp	vpc-0c1439b78911868dd	Security Group manag...
prod-22-tcp	sg-06694fc3bc51f43e9	prod-22-tcp	vpc-0fa0ed7e9750d6bd3	Security Group manag...
–	sg-0adfc95835b56b4a9	default	vpc-0601d3201c5fb8e0c	default VPC security gr...
prod-tomcat	sg-0dcd6a545ff889d5a	prod-tomcat	vpc-0fa0ed7e9750d6bd3	Security Group manag...
dev-80-tcp	sg-0c06ffeabe8f69708	dev-80-tcp	vpc-0c1439b78911868dd	Security Group manag...
dev-22-tcp	sg-0d6ee4bf34abd9be6	dev-22-tcp	vpc-0c1439b78911868dd	Security Group manag...

Configure servers for prod environment.

Go to the prod/ec2 folder with Terraform config files for creating EC2 servers in the prod environment and launch the commands:

terraform init

terraform apply

As a result, a server for the prod environment was created. In the output, we get the result of the terraform apply command. The most important information is entered in the outputs.tf file:

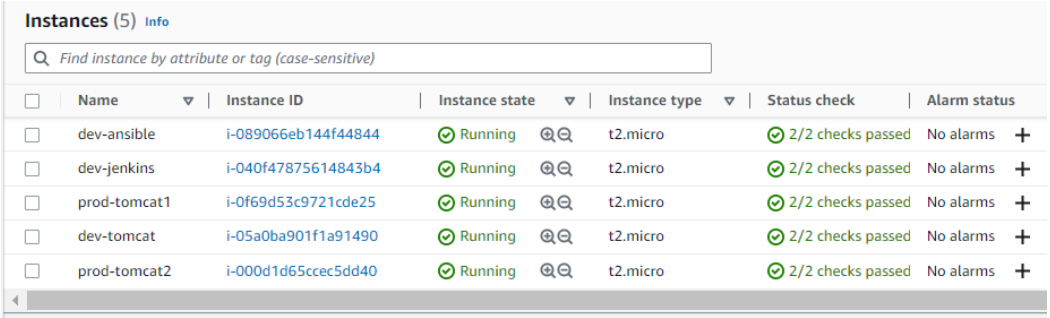
```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

prod-tomcat1_arn = "arn:aws:ec2:eu-west-2:860300666219:instance/i-0f69d53c9721cde25"
prod-tomcat1_id = "i-0f69d53c9721cde25"
prod-tomcat1_private_ip = "10.30.1.228"
prod-tomcat1_public_ip = "13.40.73.162"
prod-tomcat2_arn = "arn:aws:ec2:eu-west-2:860300666219:instance/i-000d1d65ccec5dd40"
prod-tomcat2_id = "i-000d1d65ccec5dd40"
prod-tomcat2_private_ip = "10.30.1.25"
prod-tomcat2_public_ip = "3.8.77.49"
```

As a result, two servers were created:

prod-tomca1
prod-tomca2

A cluster of two Tomcat servers will be configured to ensure failure tolerance (using Ansible). The servers will work using an AWS Application Load Balancer (which we will configure later with Terraform).

As a result, in the AWS console, we have five servers:



Configuring VPC Peering between the Dev and Prod Environment networks

Go to the vpc_peering folder with Terraform configuration files to configure VPC peering and launch the commands:

terraform init
terraform apply

As a result, a connection will be established between the networks in the Dev and Prod environments.

Configuring an AWS RDS MySQL database

The base will be necessary for the future application.

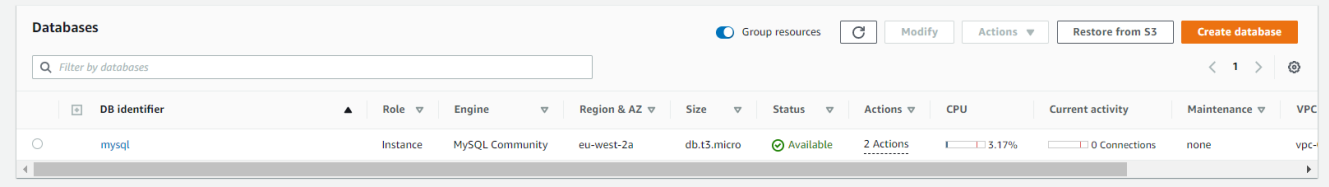
Go to the RDS folder that contains Terraform configuration files to configure the RDS database and launch the necessary commands:

terraform init
terraform apply

As a result, a database for future applications was created. The output of the '**terraform apply**' command provides the result. The most important information is entered in the 'outputs.tf' file:

```
Outputs:

db_instance_address = "mysql.cruawvpngh1q.eu-west-2.rds.amazonaws.com"
db_instance_arn = "arn:aws:rds:eu-west-2:860300666219:db:mysql"
db_instance_endpoint = "mysql.cruawvpngh1q.eu-west-2.rds.amazonaws.com:3306"
db_instance_name = "UserDB"
db_instance_password = "x89hysVVZ1q4"
db_instance_port = 3306
db_instance_username = "admin"
```

RDS Database in AWS Console:



By using Terraform, I created a table in the RDS database using the 'rds/initial.sql' script.

Configuring the Application Load Balancer for Prod Environment

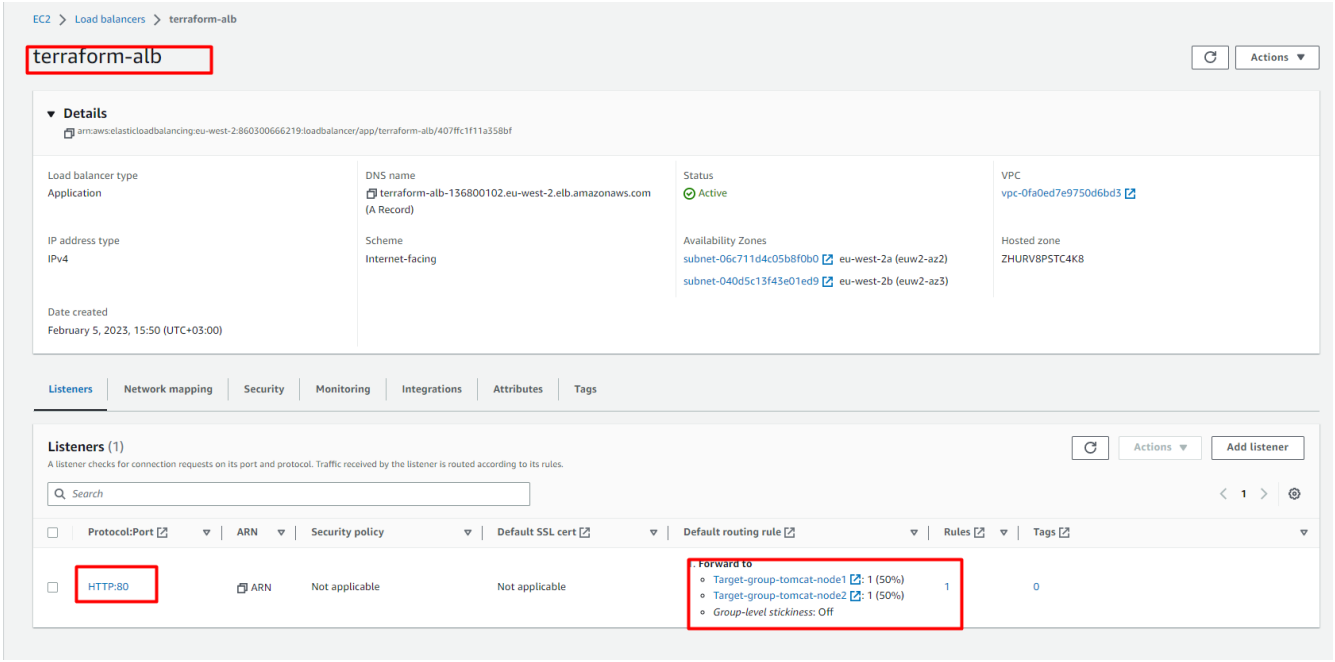
Go to the ALB folder that contains Terraform configuration files to configure the application load balancer and launch the necessary commands:

terraform init
terraform apply

The data of the command's output is noted. It will be required to access the application in the future.

```
Outputs:

dns_name_alb = "tomcat1.dmitrenko.pp.ua"
```

As a result, the application load balancer for the production environment was created. The application load balancer is necessary to balance the load between the nodes of Tomcat servers in the production environment. The application load balancer connects to the **tomcat1.dmitrenko.pp.ua** DNS name. The DNS zone 'dmitrenko.pp.ua' was connected to AWS Route 53 during the completion of the AWS homework.



Configure Jenkins using Ansible

Access the **dev-ansible** server via the SSH protocol using the MobaXterm program (using the address obtained from the Terraform outputs during the configuration of servers in the Development Environment, as well as the SSH key that was pre-generated via the AWS Console for the region in which the infrastructure was configured).

I have configured Ansible on that server through the 'user_data' script for AWS EC2 during the configuration of servers in the development environment using Terraform:

```
ubuntu@ip-10-20-1-242:~$ ansible --version
ansible [core 2.14.2]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/ubuntu/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] (/usr/bin/python3)
  jinja version = 3.0.3
  libyaml = True
```

Clone the Ansible roles for Jenkins and Tomcat from GitHub:

git clone https://github.com/edmitrenko/ansible-roles.git

Add the SSH key for the appropriate region to the "dev-ansible" server. I just copied the contents of the key to a file and added the necessary permissions to the key:

touch key.pem
nano key.pem (insert key content)
chmod 400 key.pem

Go to the **ansible-jenkins** folder and edit the **hosts** file - add the IP address of the 'dev-jenkins' server (data from the Terraform output from one of the previous steps), add the username and path to the SSH key:

[jenkins]

10.20.1.178 ansible_user=ubuntu ansible_ssh_private_key_file=/home/ubuntu/key.pem

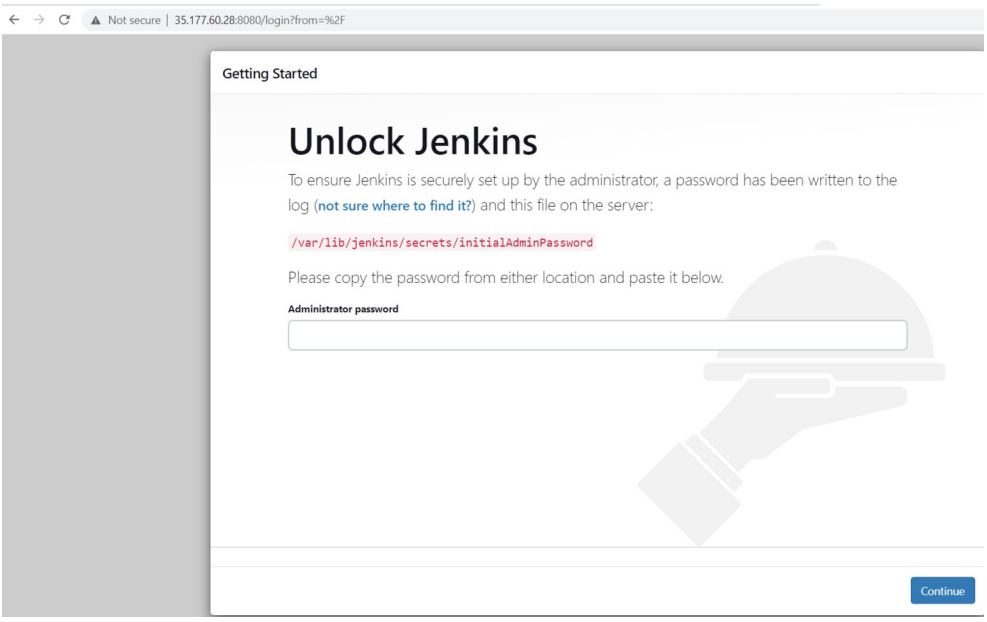
Run playbook:

ansible-playbook -i hosts playbook-jenkins.yml

As a result we obtain the password for further Jenkins configuration:

```
TASK [jenkins : print init password jenkins] ***********************************************************************************
ok: [10.20.1.178] => {
    "result.stdout": "ec55dd45f7dc4a18948688b0efb322ec"
}

PLAY RECAP *********************************************************************************************************************
10.20.1.178                : ok=16   changed=5    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0

ubuntu@ip-10-20-1-242:~/ansible-jenkins$
```

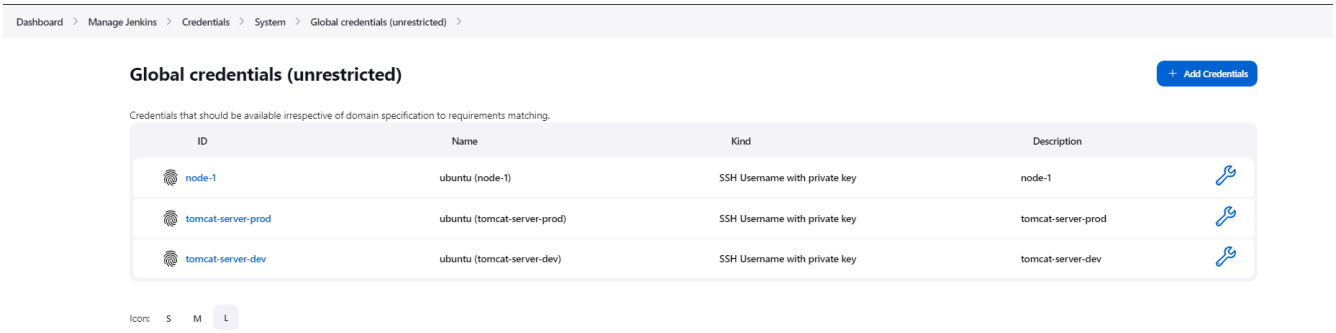
Go to the Jenkins web interface for further configuration (the IP address of Jenkins can be obtained from the Terraform outputs from one of the previous steps):



Enter the Administrator password which we obtained in the previous step. Select the necessary plugins (in my case: SSH Agent, SSH Build Agents, Copy Artifact, Deploy to Container, pipeline, git, docker, docker pipeline) and press the 'Continue' button. Add a new user and password, and then we can access Jenkins.

Add credentials for connecting to the second node in Jenkins and for connecting to the Tomcat servers:

Manage jenkins→ Credentials → System → Global Credential → Add Credentials

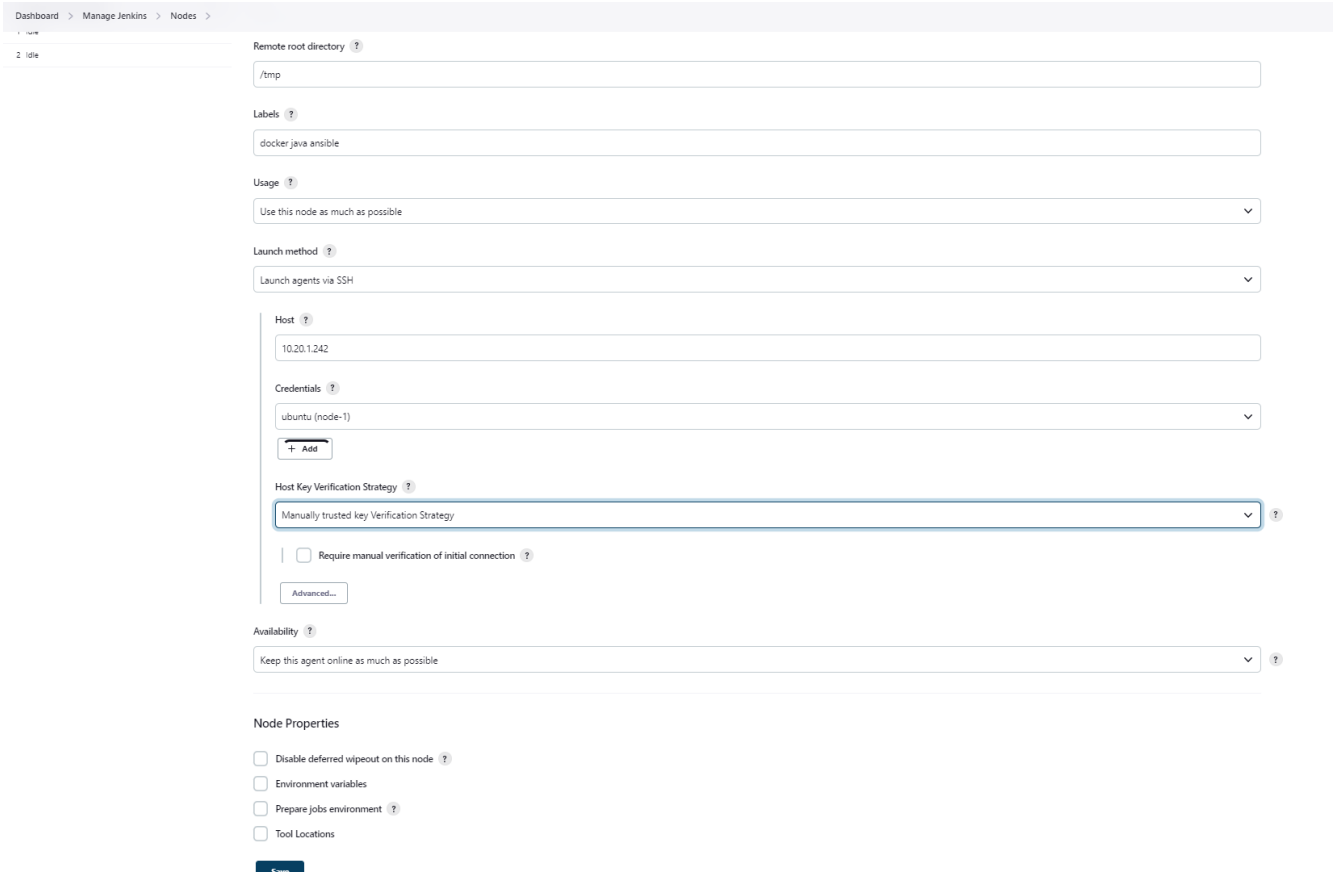


Add second node to Jenkins server:

Manage jenkins→Mange Nodes and Clouds → New Node → set name Node-1

You need to specify the following:

- **Remote root directory**: /tmp
- **Labels**: it will be used in the Jenkinsfile to specify on which node the Jenkins task will be performed (if you want to specify multiple labels, separate them with a space)
- **Launch method**: Launch Agent via SSH.



Now Jenkins is ready to work.

Configuring Tomcat via Ansible

Enter the 'dev-ansible' server via the SSH protocol using the MobaXterm program (using the address that was obtained from Terraform outputs during configuring servers in the Dev Environments, as well as the SSH key that was pre-generated via the AWS console for the region in which the infrastructure was configured) or return to the ssh console that was open in the previous step.

Go to the ansible-tomcat folder and edit the hosts file - add the IP address of the 'dev-tomcat' server and the addresses of the 'prod-tomcat' servers (data from Terraform output in one of the previous steps), add the username and path to the ssh key:

[dev]

10.20.1.10 ansible_user=ubuntu ansible_ssh_private_key_file=/home/ubuntu/key.pem

[prod1]
10.30.1.228 ansible_user=ubuntu ansible_ssh_private_key_file=/home/ubuntu/key.pem

[prod2]
10.30.1.25 ansible_user=ubuntu ansible_ssh_private_key_file=/home/ubuntu/key.pem

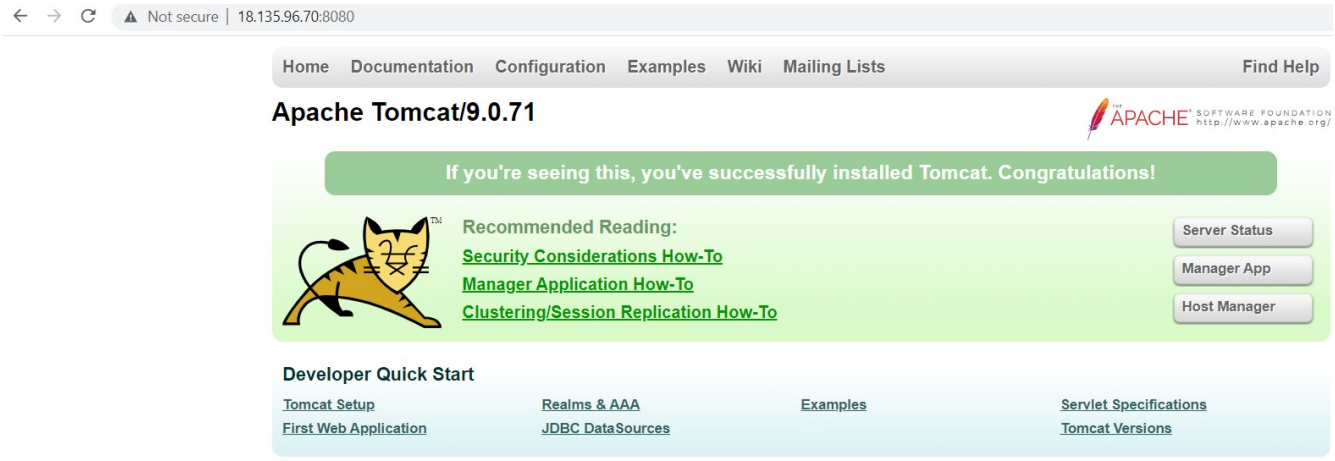
Also, you need to configure the playbook-tomcat.yml file. To configure the Tomcat server in the Prod environment, it is necessary to specify the port numbers on which the Tomcat server will operate. To set up the Tomcat server cluster, the port numbers should differ on the first and second node.

Install the Tomcat server on dev-tomcat by using the correct playbook-tomcat.yml:

hosts: dev

env: dev
and start:
ansible-playbook -i hosts playbook-tomcat.yml

As a result, we have a tomcat server running on dev-tomcat, whose IP address can be obtained from the Terraform outputs taken in an earlier step:



Install tomcat server on prod-tomcat1. Configure playbook-tomcat.yml:

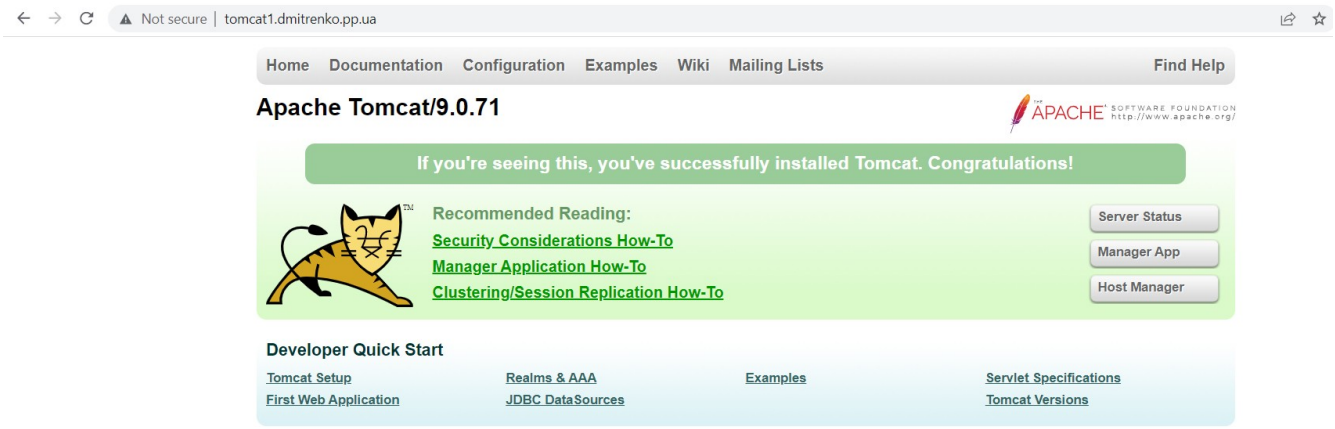
hosts: prod1
env: prod
server_port: 8006
http_port: 8081
arj_port: 8009
and start:
ansible-playbook -i hosts playbook-tomcat.yml

Install tomcat server on 'prod-tomcat'2. Configure playbook-tomcat.yml:

hosts: prod2
env: prod
server_port: 8007
http_port: 8082
arj_port: 8010

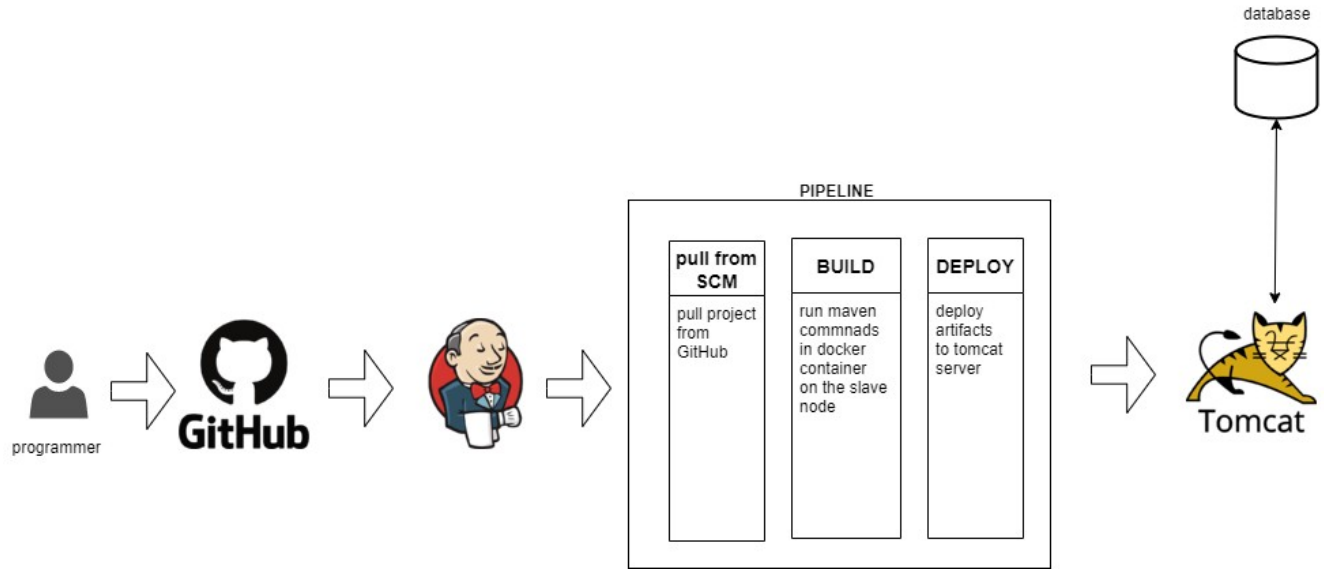
and start:
ansible-playbook -i hosts playbook-tomcat.yml

Now we can use the DNS name of the application load balancer to connect to the Tomcat cluster - http://tomcat1.dmitrenko.pp.ua/:



Configure Pipeline in Jenkins

How the architecture looks like:



For the final project, I have selected a small Java web application that has a database for storing user accounts.

This application will be built with Maven on a Jenkins node in a Docker container (the latest Docker Image Maven will be used). As a result, we will have a .war file.

Delivery of artifacts will occur using the Jenkins-SSH Agent plugin (for storing credentials) and by starting a shell scp command to copy the .war file. The .war file will be in the format of **name##version.**

The old version of the application will be deleted from the Tomcat servers.

server.xml:

```
<Host name="localhost"  appBase="webapps"
        unpackWARs="true" autoDeploy="true"
        undeployOldVersions="true" >

      <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
           prefix="localhost_access_log" suffix=".txt"
           pattern="%h %l %u %t &quot;%r&quot; %s %b" />
 </Host>
```

I also reduced the session timeout to 5 minutes in the web.xml file:

```
<session-config>
      <session-timeout>5</session-timeout>
</session-config>
```

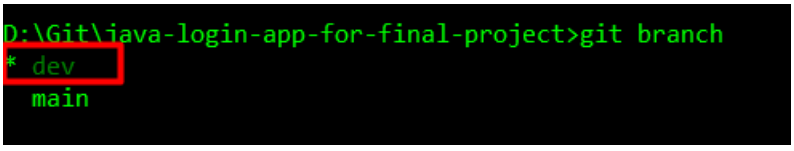
The Java application is located on GitHub:

https://github.com/edmitrenko/jenkins-java-app2.git

The Jenkins file is also located in this repository.

Before configuring the pipeline in Jenkins, we need to edit the Jenkinsfile and specify the addresses of the Tomcat servers. We also need to point to the path to the database that was created previously. The information about the database can be obtained from the Terraform outputs from previous steps. This information can be found in **src\main\resources\application.properties**.

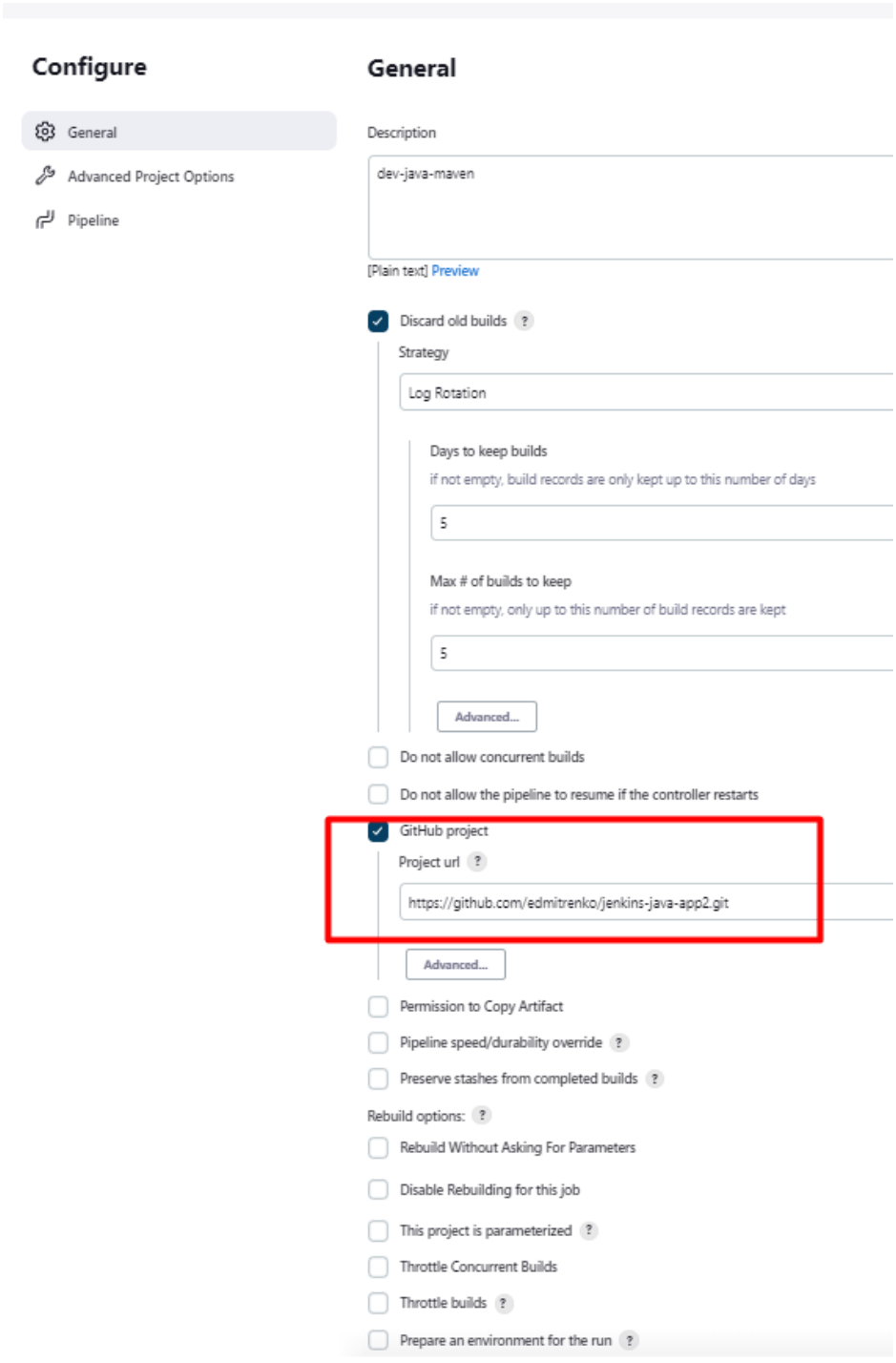
A programmer should clone the Git repository to their local computer and create a new branch, such as **dev**, for development. All changes should be pushed to the central repository, and after they have been checked, the **dev** branch should be merged with the main branch:

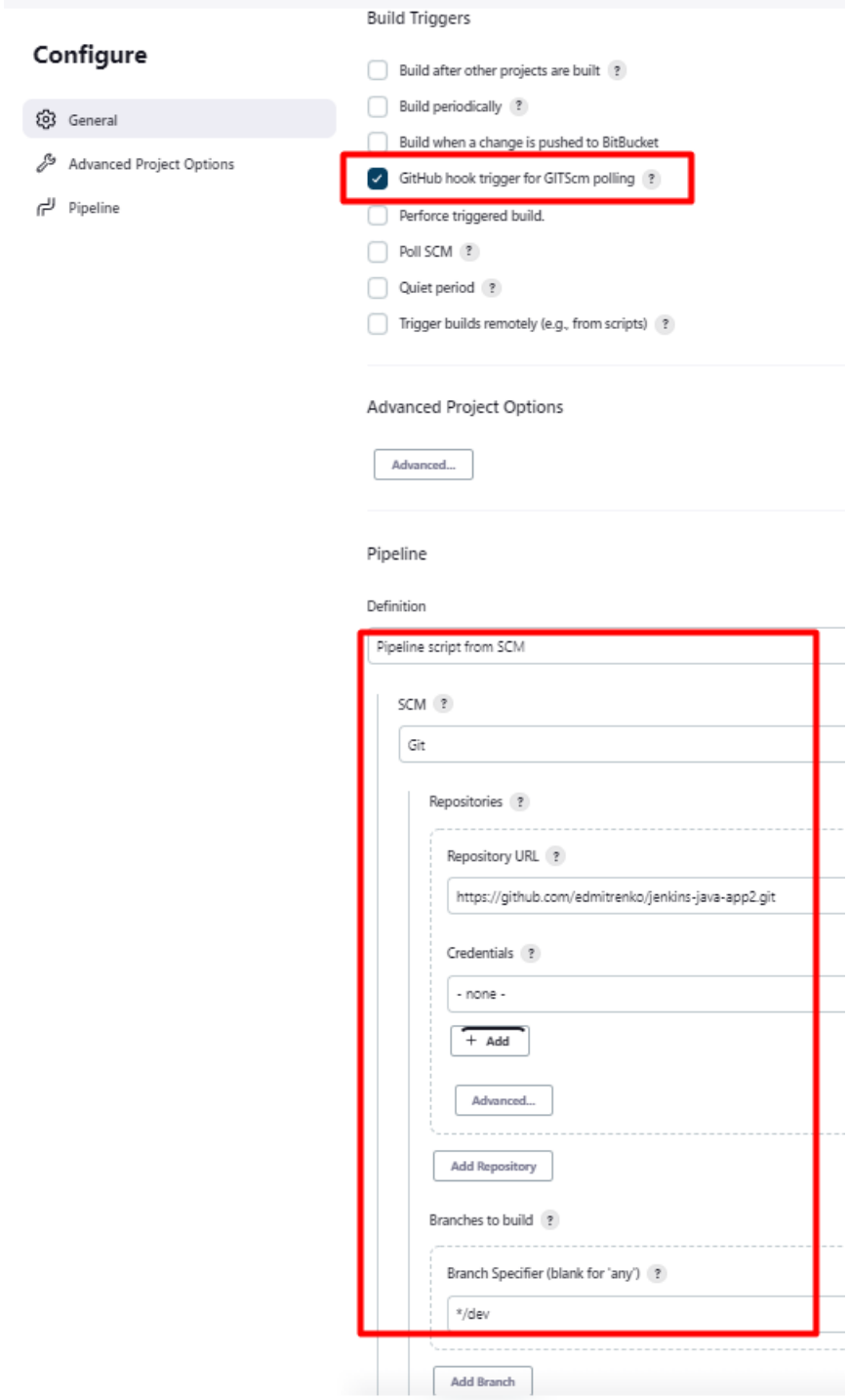


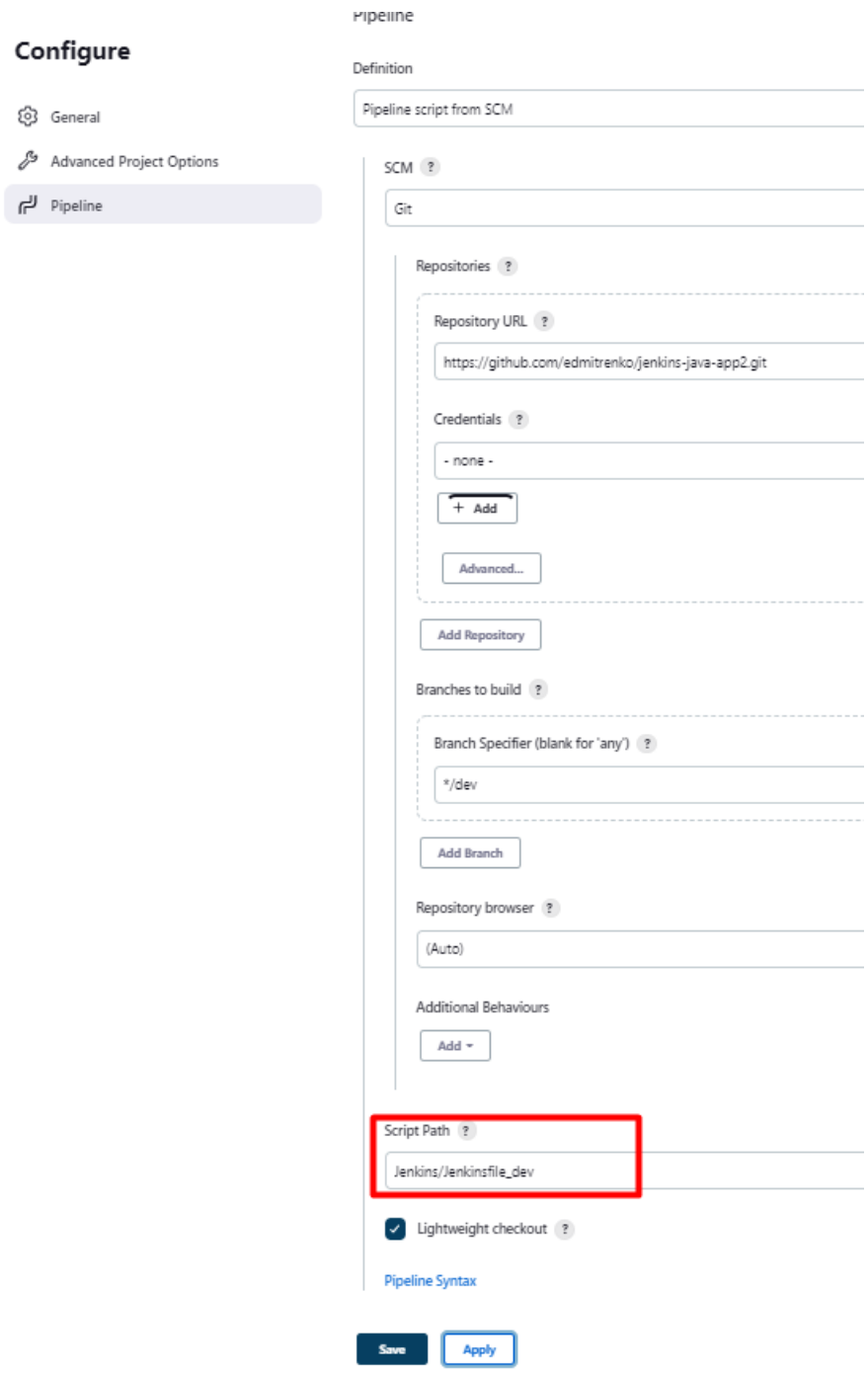
A programmers pushes changes to the central repository and creates a pull request.
Then, a GitHub webhook is triggered, which starts the pipeline in Jenkins. All changes from the dev branch are deployed to the dev-tomcat server.

After the branches are merged (dev with main), the GitHub webhook is triggered again, and the Jenkins pipeline starts to build and deploy the application to the prod servers.

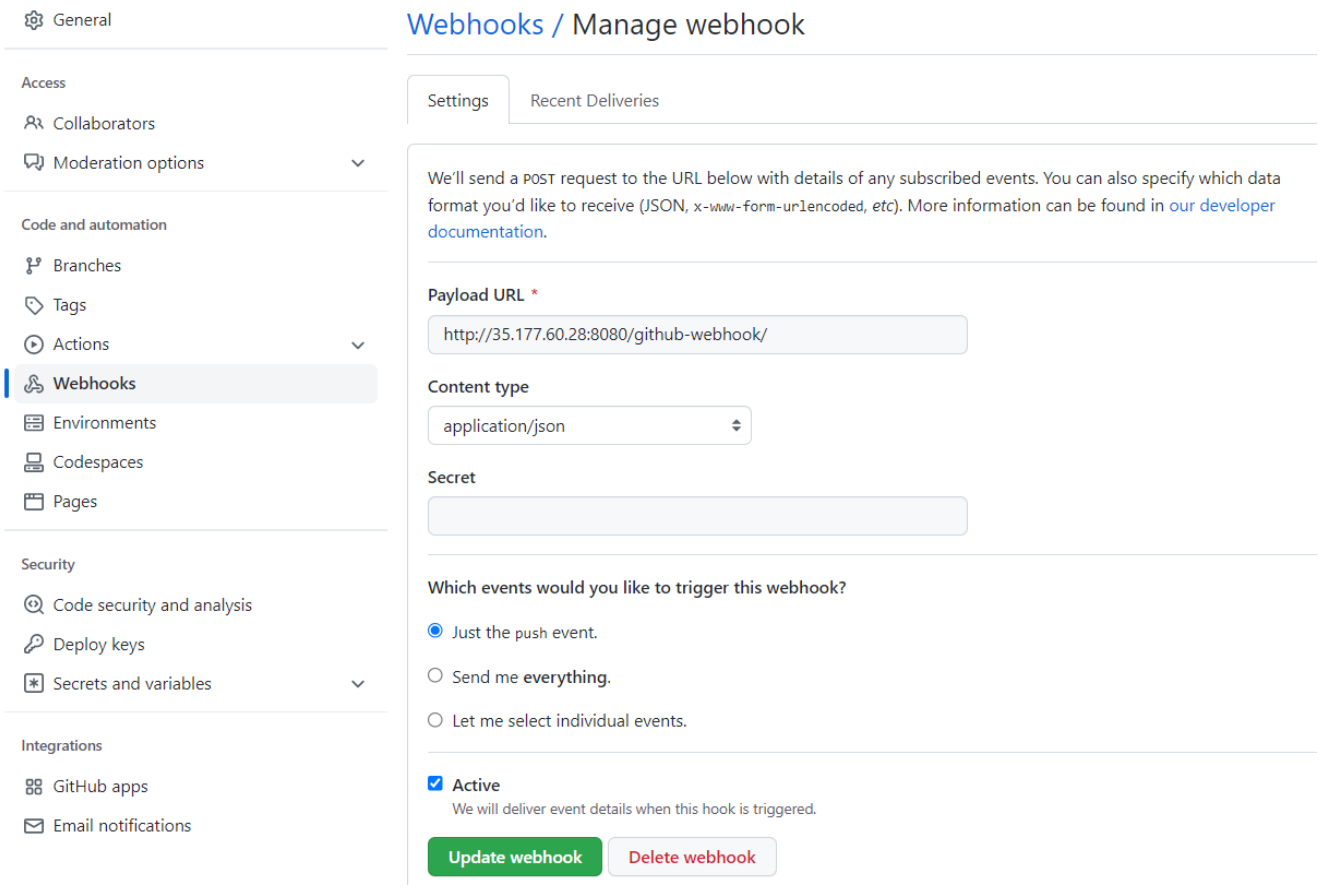
Create Jenkins pipeline for dev server




Configure
General
Advanced Project Options
Pipeline
Build Triggers
Build after other projects are built
Build periodically
Build when a change is pushed to BitBucket
GitHub hook trigger for GITScm polling
Perforce triggered build.
Poll SCM
Quiet period
Trigger builds remotely (e.g., from scripts)
Advanced Project Options
Advanced...
Pipeline
Definition
Pipeline script from SCM
SCM
Git
Repositories
Repository URL
https://github.com/edmitrenko/jenkins-java-app2.git
Credentials
- none -
+ Add
Advanced...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/dev
Add Branch


Pipeline
Configure
General
Advanced Project Options
Pipeline
Definition
Pipeline script from SCM
SCM
Git
Repositories
Repository URL
https://github.com/edmitrenko/jenkins-java-app2.git
Credentials
- none -
Add
Advanced...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/dev
Add Branch
Repository browser
(Auto)
Additional Behaviours
Add
Script Path
Jenkins/Jenkinsfile_dev
Lightweight checkout
Pipeline Syntax
Save
Apply

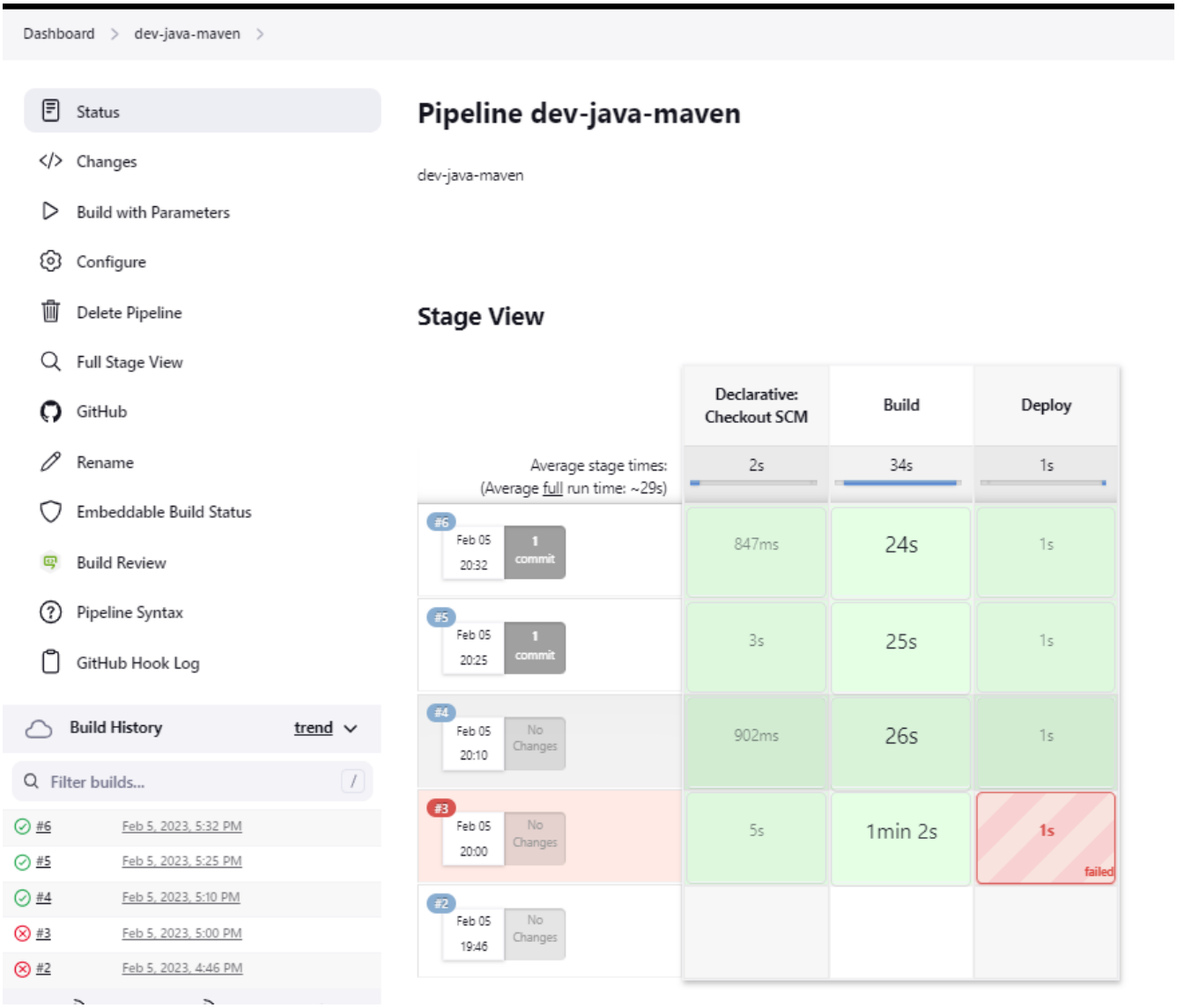
Add a github webhook:



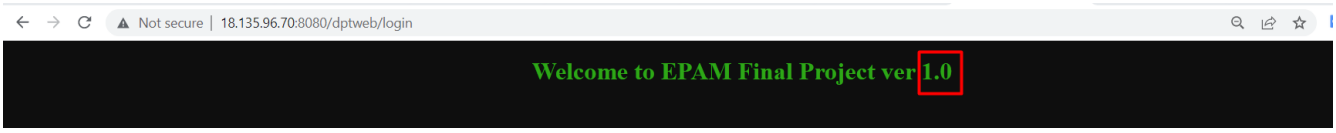
Push changes to Github:

```
D:\Git\java-login-app-for-final-project>git add .

D:\Git\java-login-app-for-final-project>git commit -m "final_v1.0"
[dev 1125efb] final_v1.0
 2 files changed, 2 insertions(+), 2 deletions(-)

D:\Git\java-login-app-for-final-project>git push --set-upstream origin dev
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 425 bytes | 425.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/edmitrenko/jenkins-java-app2.git
   0ee700d..1125efb  dev -> dev
branch 'dev' set up to track 'origin/dev'.
```

The Jenkins pipeline has started:

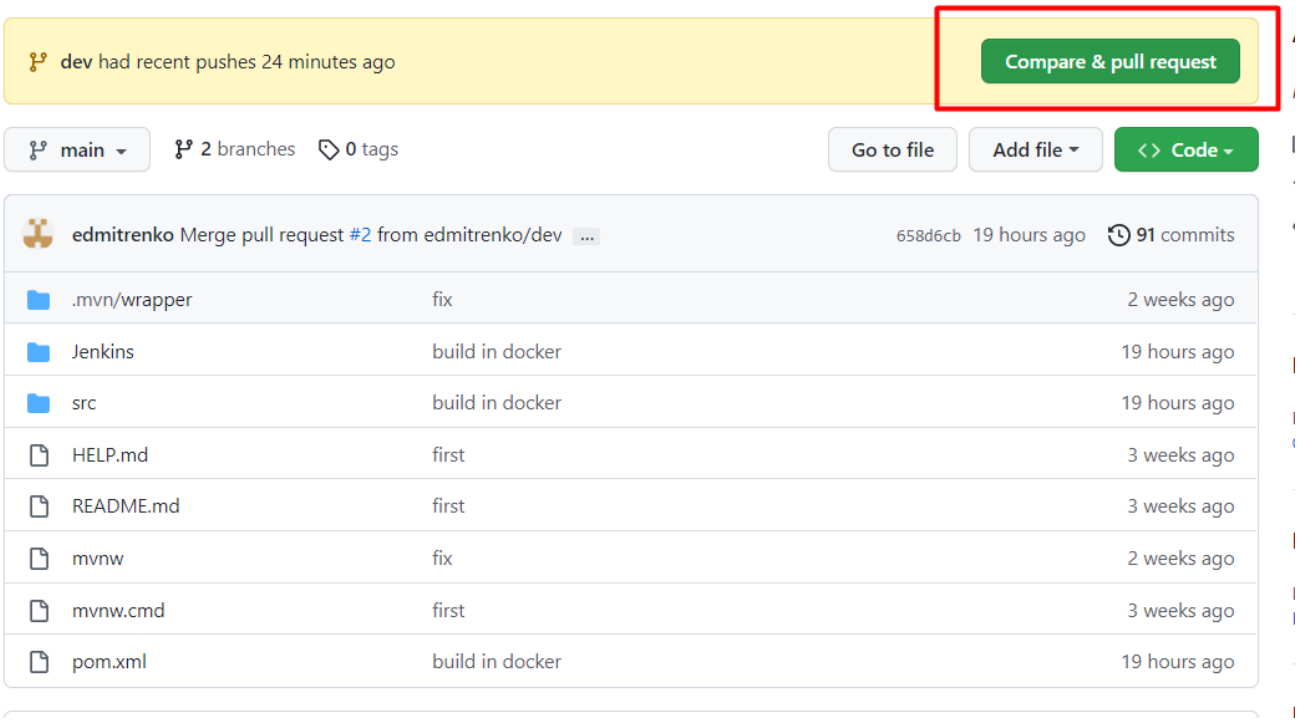


As a result, the application version 1.0 was deployed to the dev-tomcat server:

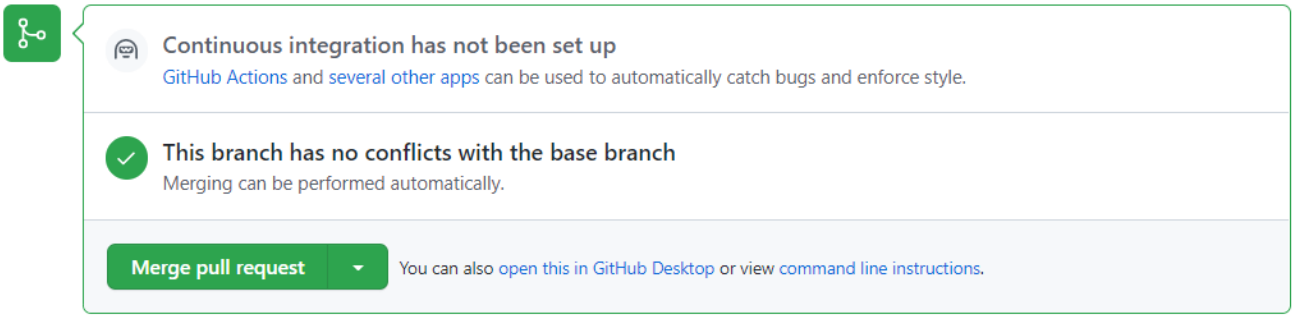


Configure the Jenkins pipeline for the prod environment similarly to the dev environment, but change the path to the Jenkinsfile and the GitHub branch.

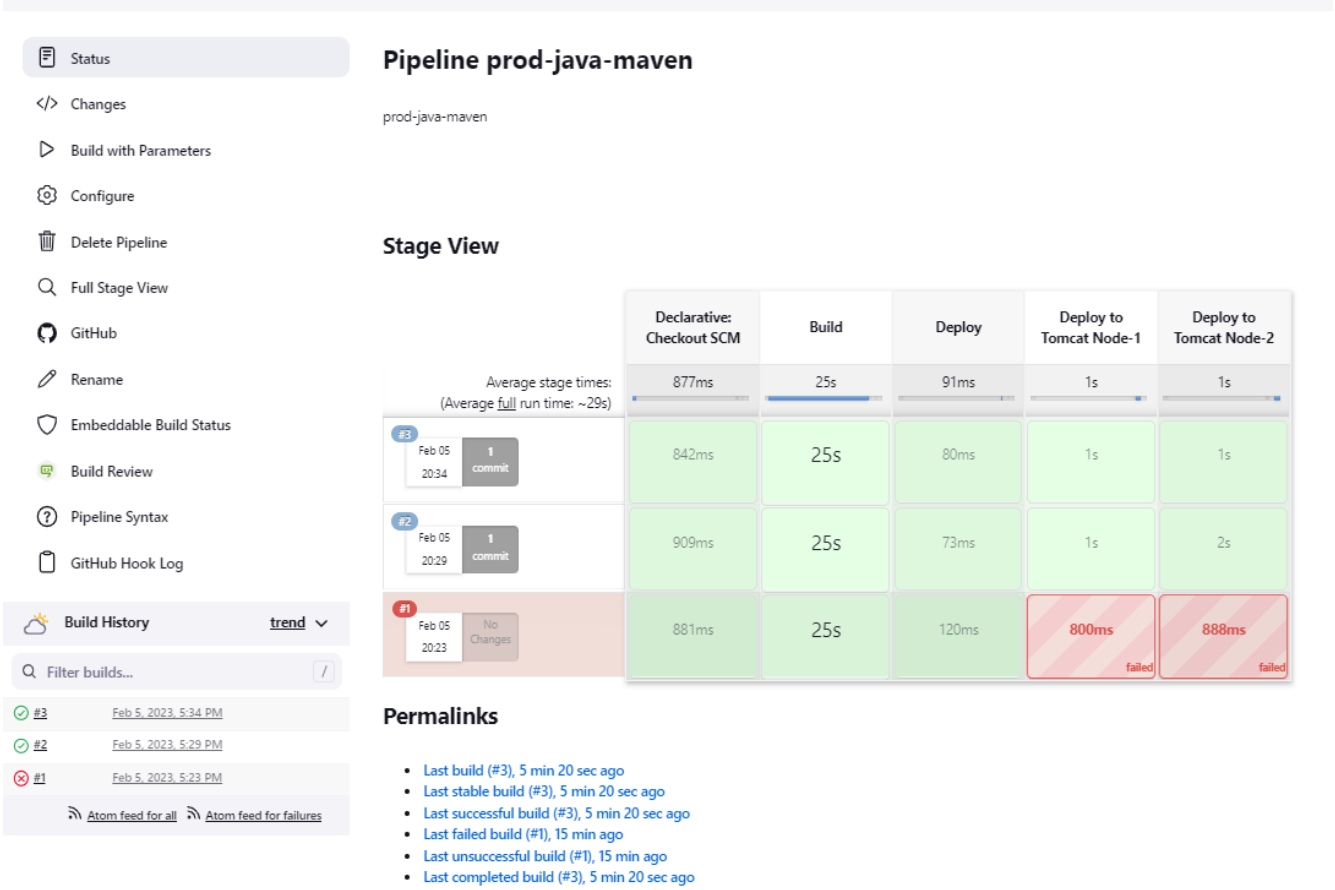
Create a pull request:



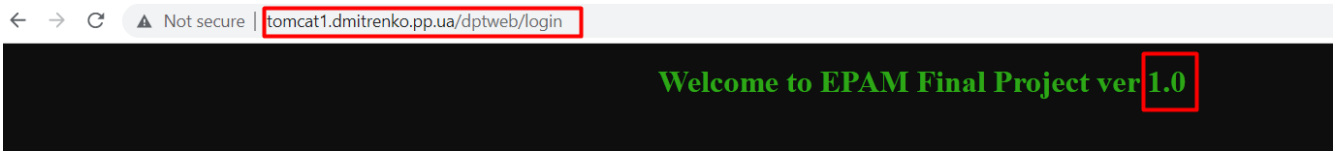
Merge the pull request:



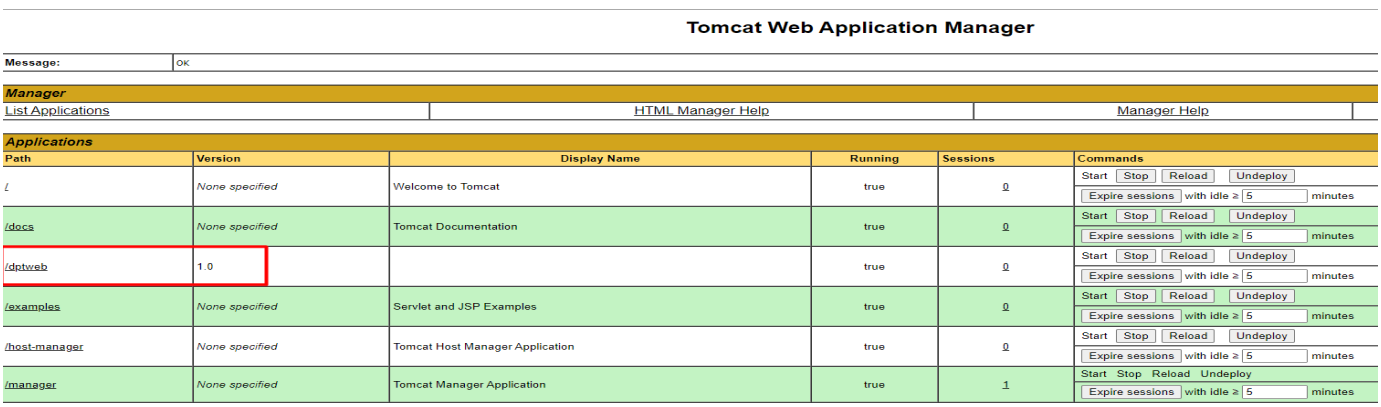
The pipeline for the prod environment was started:



As a result, the application version 1.0 was deployed to the prod-Tomcat servers:

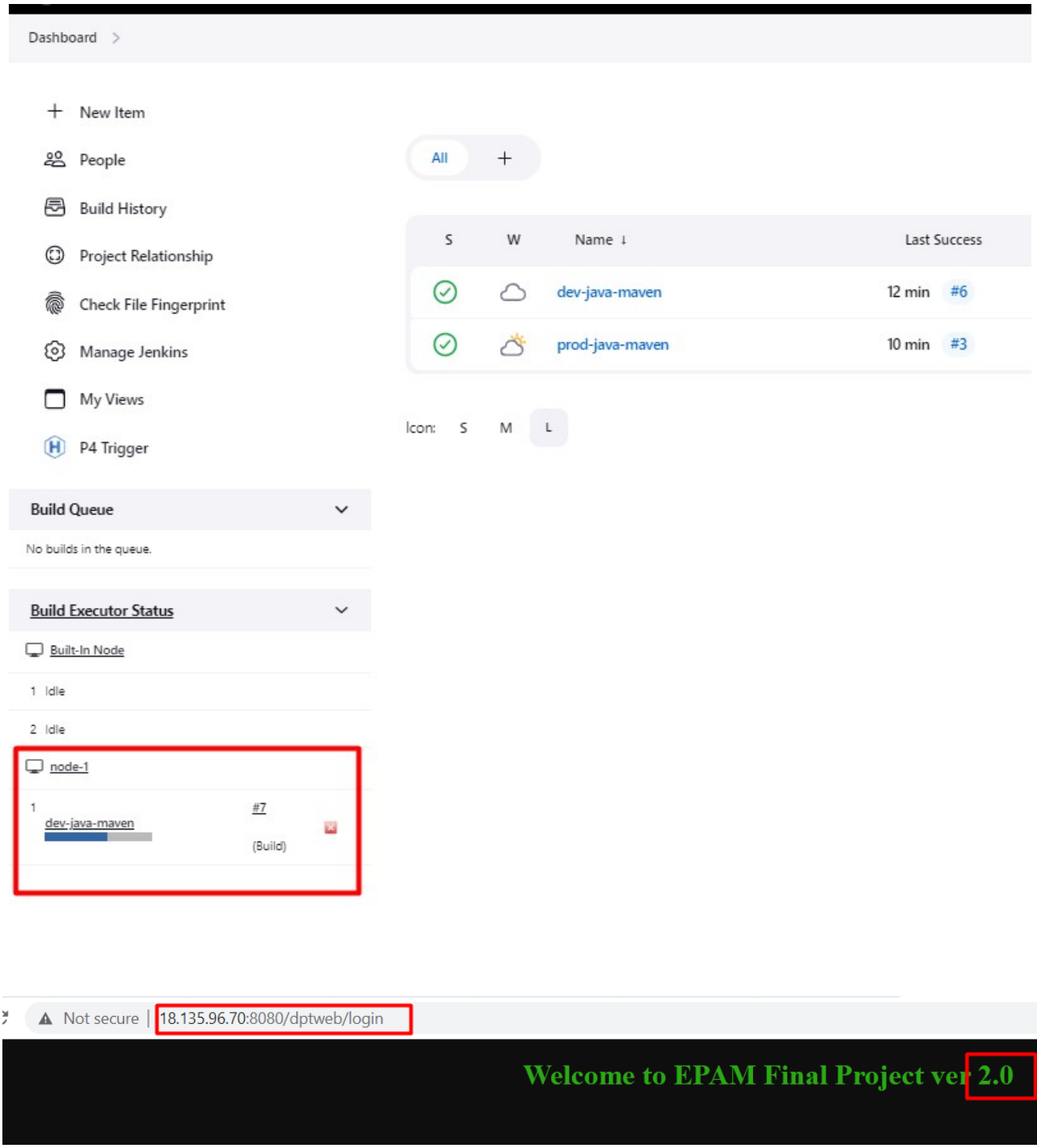


We can see version 1.0 of the application in the Tomcat manager:

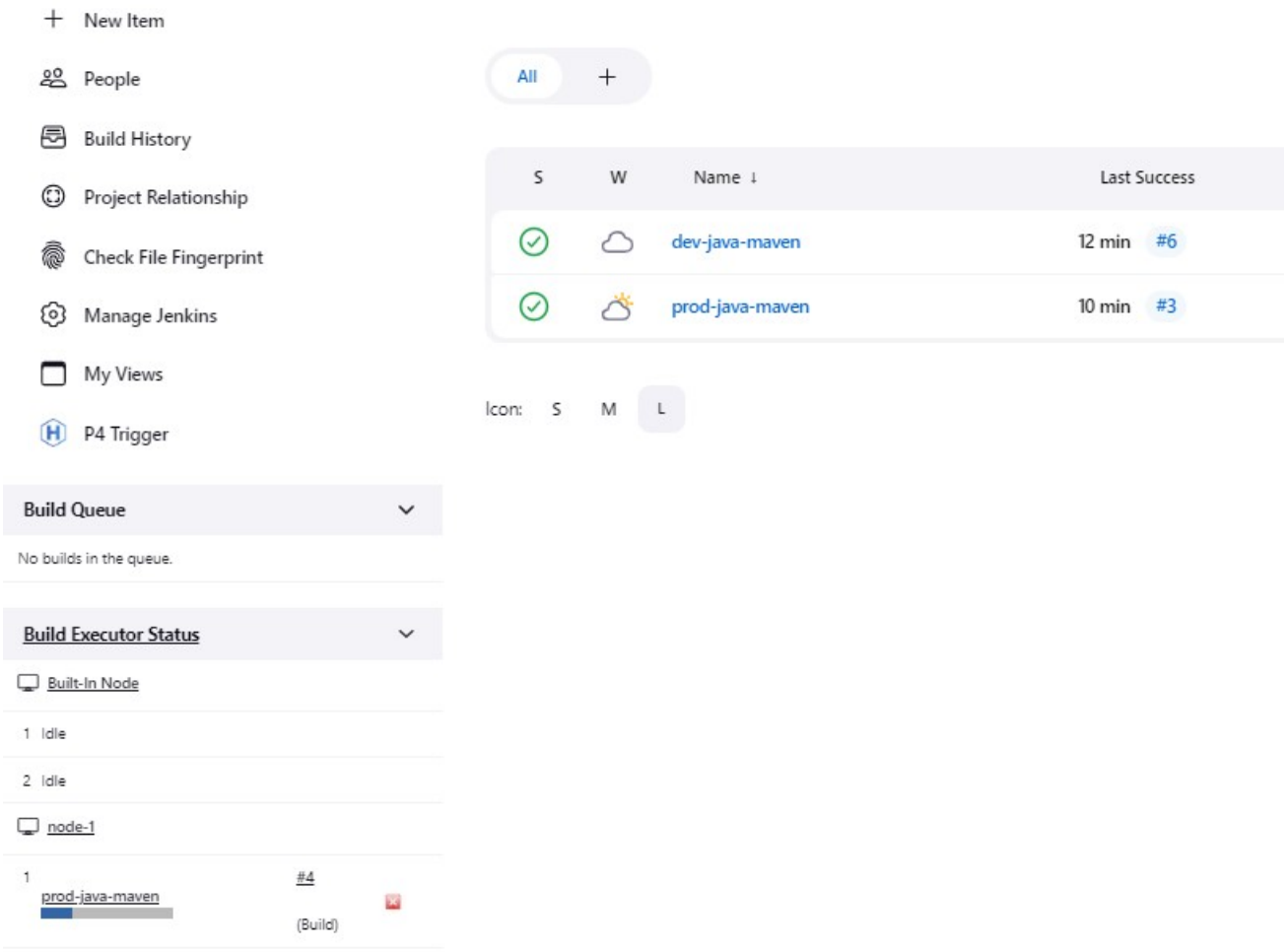


We make changes in the code, such as changing the version on the web page and the version in the Pom.xml file, and send the changes to GitHub in the dev branch.

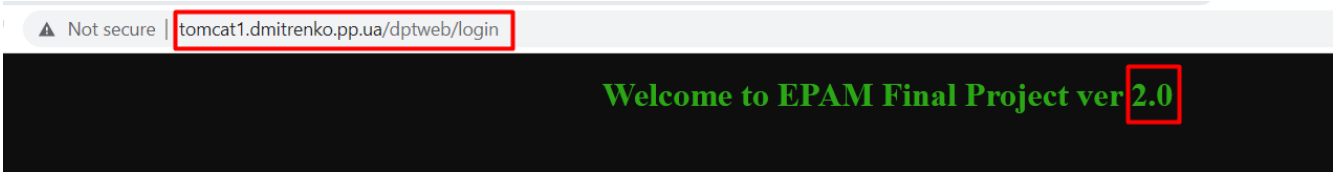
As a result, the Jenkins pipeline **dev-java-maven** should automatically start, build, and deploy the new version of the application to the dev-tomcat server



On GitHub, create a pull request → merge request → confirm merge. The GitHub webhook is triggered, and the Jenkins pipeline **prod-java-maven** for the prod environment should start automatically:



And the new version of the application was built and deployed to the prod servers:



And we can see the version of the application on the Tomcat manager page:

Not secure | tomcat1.dmitrenko.pp.ua/manager/html

Tomcat Web Application Manager

Message:	OK

Manager		
List Applications	HTML Manager Help	Manager Help

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	*None specified*	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 5 minutes
/docs	*None specified*	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 5 minutes
/dptweb	2.0		true	2	Start Stop Reload Undeploy Expire sessions with idle ≥ 5 minutes
/examples	*None specified*	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 5 minutes
/host-manager	*None specified*	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 5 minutes
/manager	*None specified*	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 5 minutes

Deploy
Deploy directory or WAR file located on server