ATTACK DEFENSE

by PentesterAcademy

Name	DevSecOps Pipeline: Django WebApp Pipeline as Code
URL	https://www.attackdefense.com/challengedetails?cid=2069
Type	Pipeline Basics: Web Applications

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Challenge Description

DevOps practices are to combine software development (Dev) and IT operations (Ops) in order to improve the delivery process. DevOps pipelines are chained tasks and components that run in a sequence to cover different phases of software compilation, packaging, automated testing, and test deployment.

In this lab, we have a simple DevOps pipeline for a sample Java-based web application. The pipeline consists of the following components (and tasks):

- Kali machine (For pulling, modifying, and pushing the code)
- GitLab server (For hosting code)
- Jenkins server (For integrating all parts: building django project, deploying with Ansible, and dynamic testing with Selenium)
- Test server (For test deployment)

It is suggested to play the DevOps focused lab before playing this lab.

DevSecOps refer to introducing security in different stages of the DevOps process. This is done to catch the vulnerabilities/insecurities as soon as possible in the pipeline. In this lab, the pipeline consists of the following components (and tasks):

- Static Code Analysis: SonarQube
- Dynamic Application Security Testing: OWASP ZAP
- Automated Code Review: DevSkim

Objective: Run the pipeline and observe/understand the DevSecOps process!

Instructions:

- The GitLab server is reachable with the name 'gitlab'
- Gitlab credentials:

Username	Password
root	welcome123

- The Jenkins server is reachable with the name 'jenkins'
- Jenkins credentials:

Username	Password
admin	welcome123

- The test deployment server is reachable by the name "test-server"
- Test server SSH credentials:

Username	Password
tomcat	password1

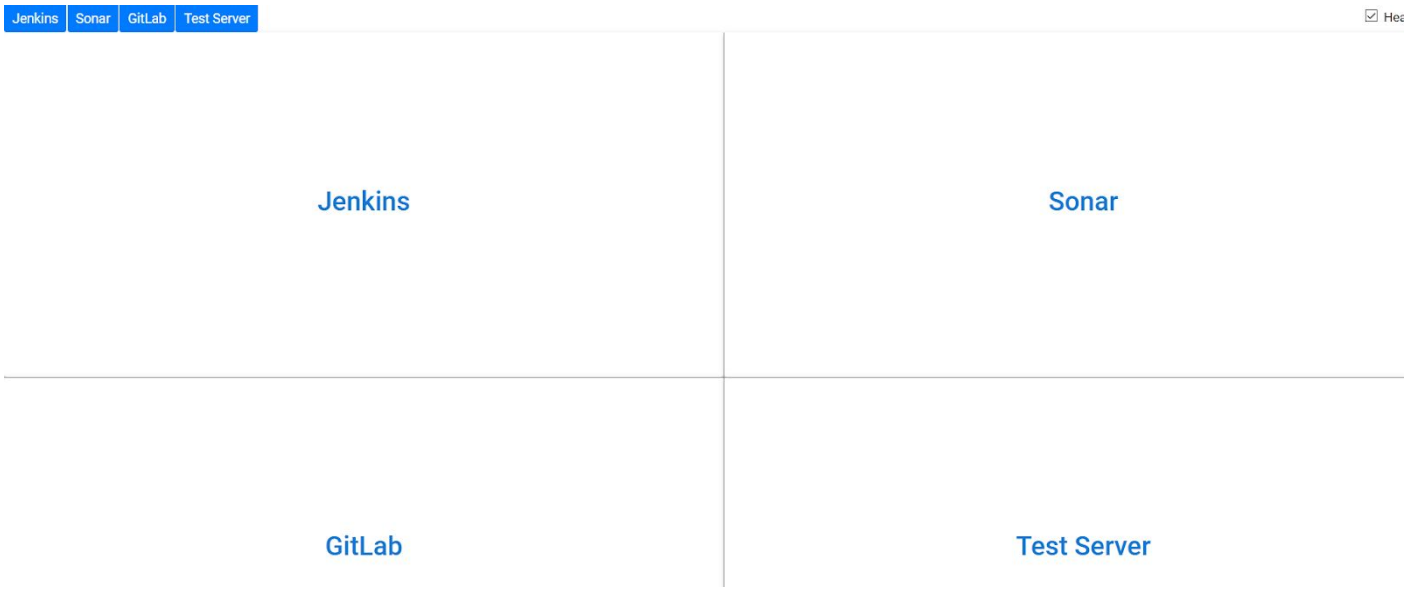
- The Sonar server is reachable by the name "sonar"
- Sonar server credentials:

Username	Password

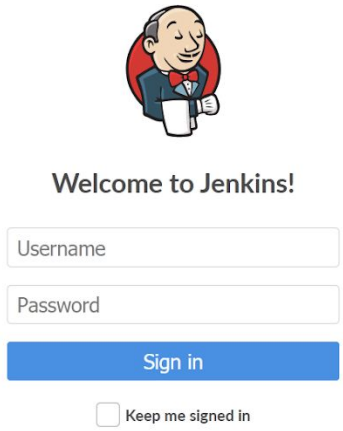
admin	admin

Lab Setup

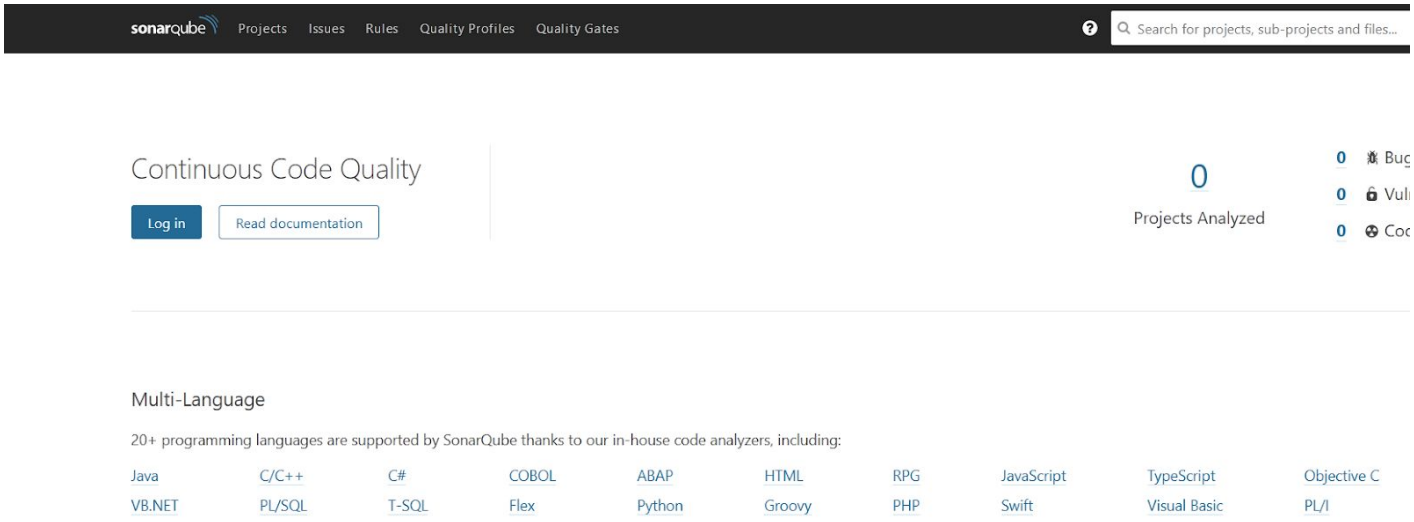
On starting the lab, the following interface will be accessible to the user.



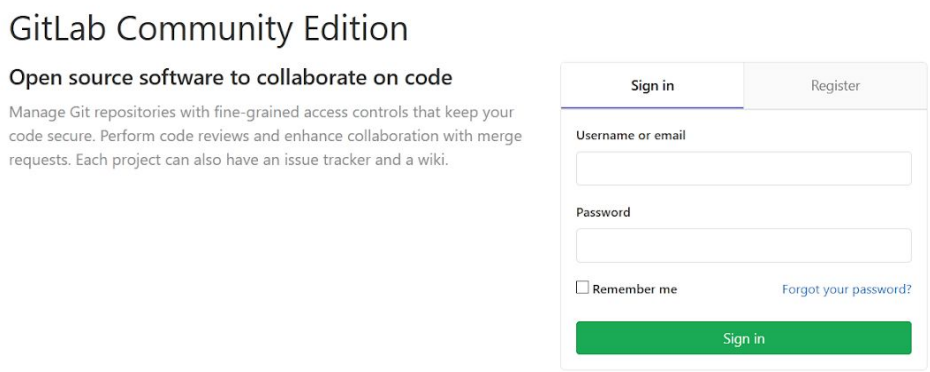
On choosing (clicking the text in the center) top left panel, **Jenkins** will open in a new tab



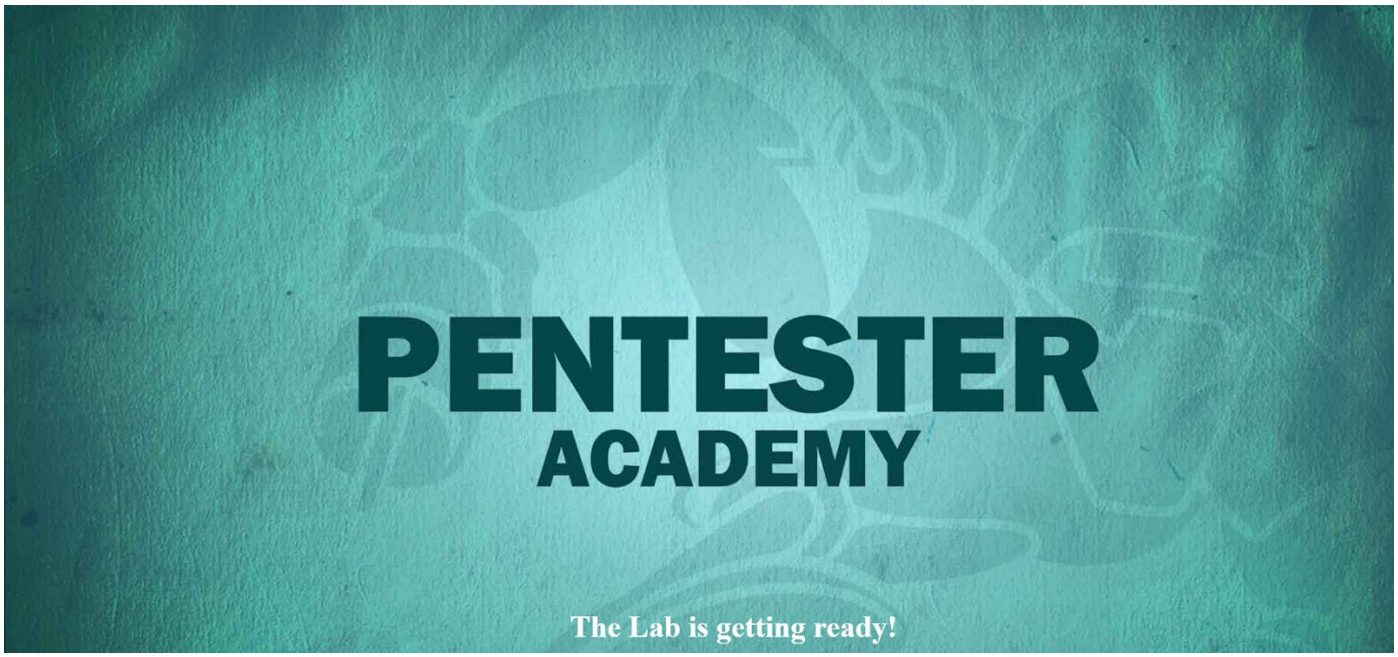
Similarly on selecting the top right panel, a web UI of **Sonar** will open in a new tab.



On selecting the bottom left panel, a web UI of **Gitlab** will open in a new tab.



And on selecting the bottom right panel, a web UI of **Test Server** will open in a new tab.



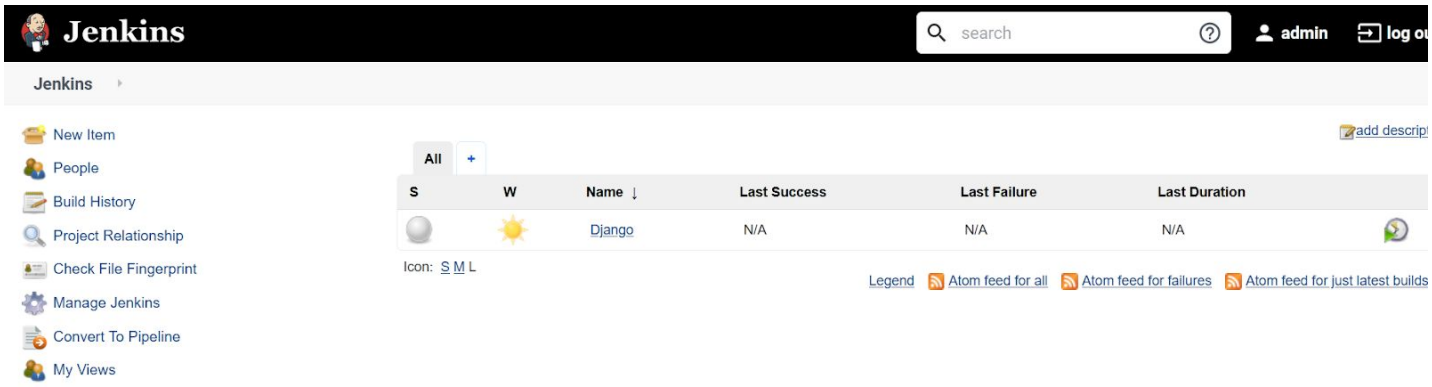
The page will reload until the test-server has started running the web service at port 8080

Solution

Step 1: Login into the Jenkins, The credentials are provided in the challenge description.

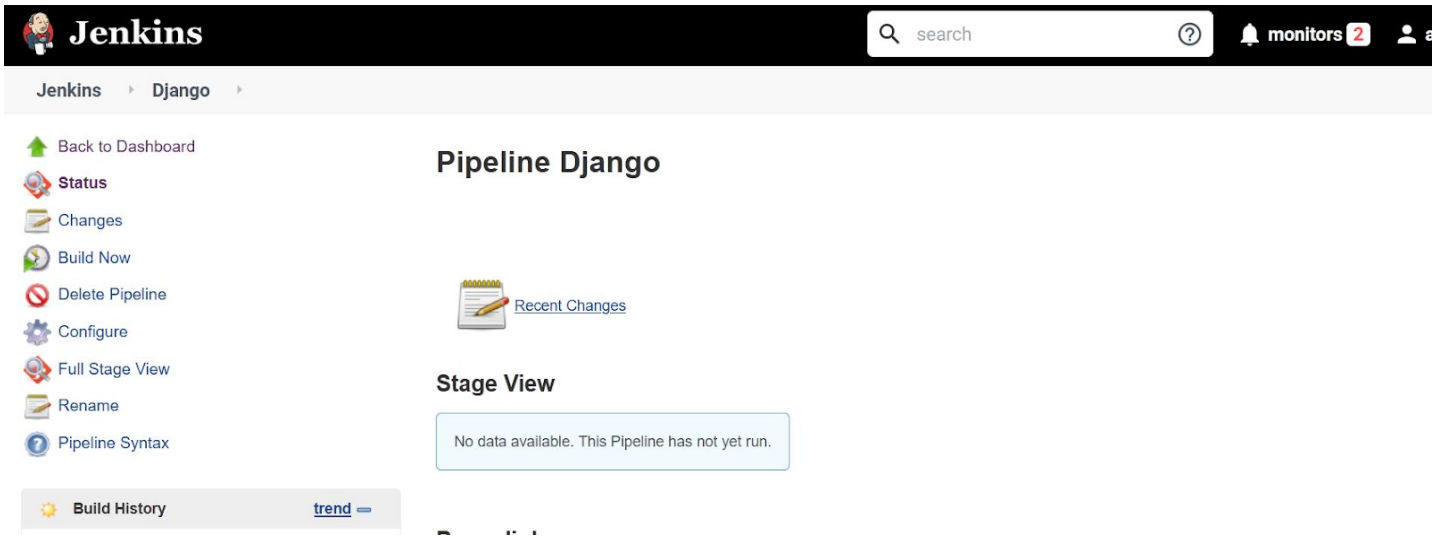
Credentials:

- **Username:** admin
- **Password:** welcome123



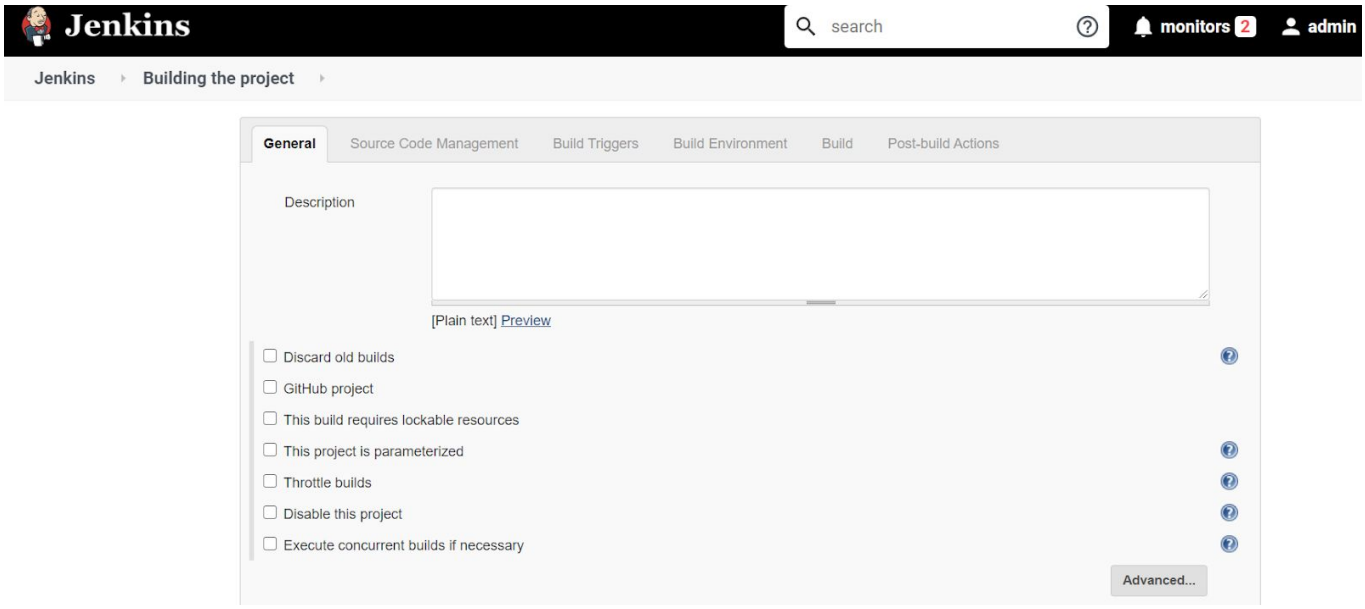
There is only one job present in the Jenkins Interface, We will take one job at a time to study.

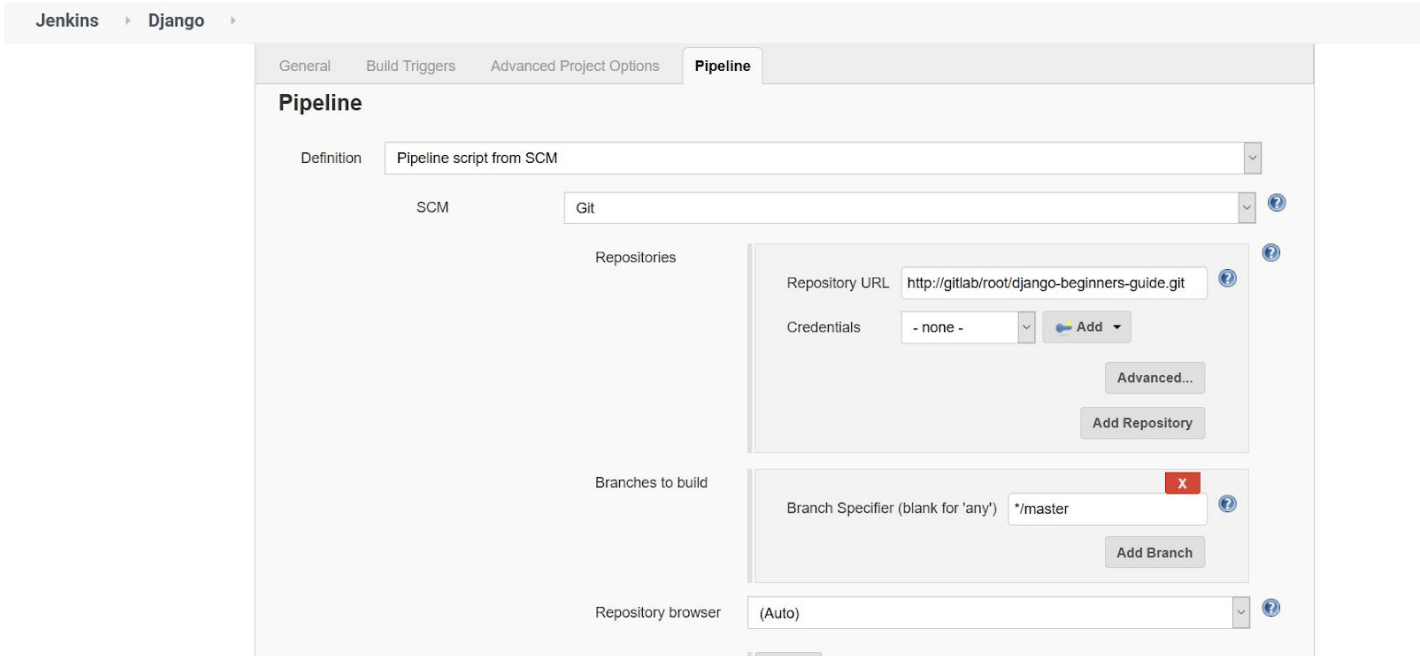
Step 2: Click on the “Django” job.



This page is for “Pipeline Django” job, The Pipeline is appended in front of the Job name because this is a “Pipeline” type job in which it accepts a ‘Jenkinsfile’ which has all the commands and configuration of the pipeline.

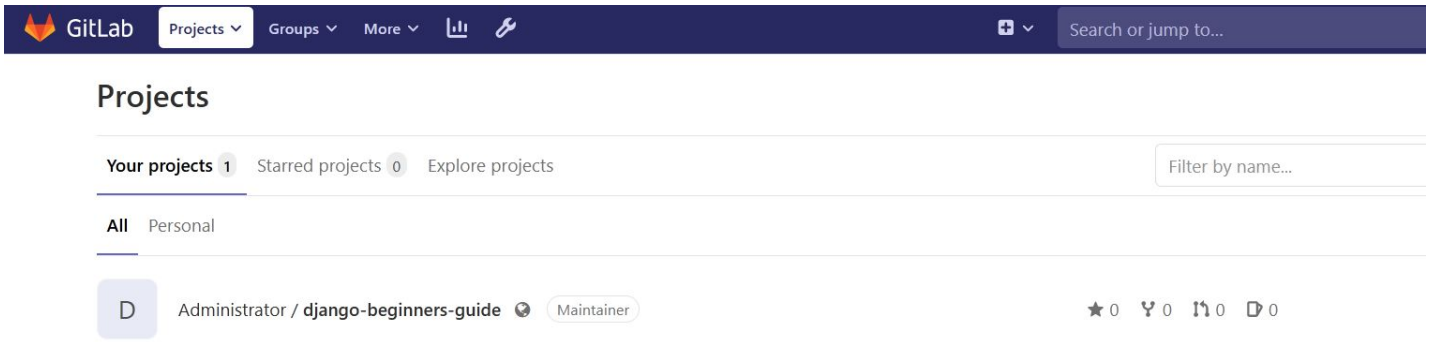
Step 2: Click on the “Configure” option to check the configuration of the Job.





The “Pipeline” sections accept Jenkinsfile directly or a source such as Gitlab where the code and Jenkinsfile are stored for the project.

The code is hosted on GitLab instance at this path “http://gitlab/root/django-beginners-guide.git”



Step 3: Open the project on Gitlab and check the Jenkinsfile to build the pipeline.

```
        stage ('Building the project - Checkout') {
         checkout([$class: 'GitSCM', branches: [[name: '*/master']], doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [], userRe
         sh """
         tar -zcvf /tmp/django.tar.gz .
         """
        }

stage ('SonarQube - Scan') {

    sh """
       sonar-scanner
      """
    }

stage ('DevSkim - Scan') {

  sh """
      devskim analyze . -s critical
     """

}
```

```
stage ('Django Installation - Build') {
                        // Shell build step


 ansiblePlaybook(
        inventory: '',
        playbook: 'django.yml',
        )

        }
```

```
stage ('Selenium Testing') {

  sh """
      pytest --capture=no selenium_checks.py
     """

}

stage ('OWASP ZAP Testing - Build') {

sh """
/opt/ZAP_2.9.0/zap.sh  -cmd -quickurl http://test-server:8080/ -quickprogress -silent -quickout /tmp/OWASP_ZAP_REPORT.xml; mv /tmp/OWASP_ZAP_REPORT.xml
  """
}

}
}
```

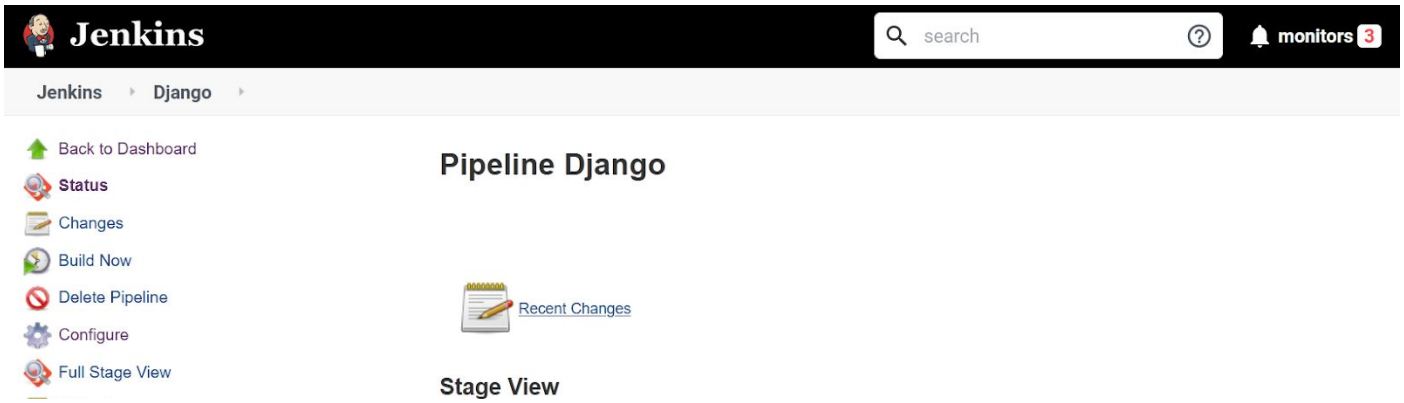
There are 6 stages in the Jenkinsfile, We will take one stage (DevSecOps only) at a time to study as DevOps stages are already covered in the DevOps pipeline lab. Please check that first if you haven't already.

Jenkinsfile Stages:

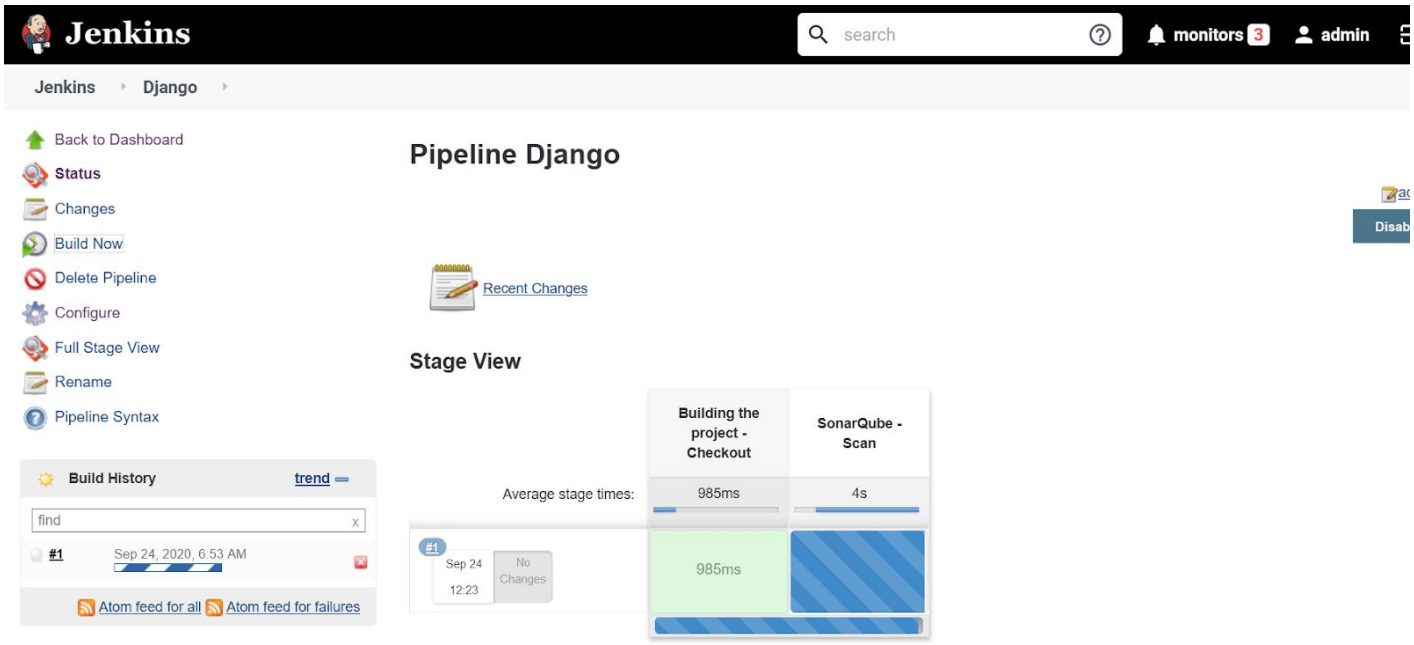
- **SonarQube Scan:** In this stage, sonarqube will perform static analysis on the source code to find bugs.
- **DevSkim:** In this stage, the devskim will perform inline security analysis on the source code to check for vulnerabilities.
- **OWASP ZAP Testing:** In this stage, The OWASP ZAP will perform dynamic analysis on the deployed web application.

Pipeline Execution

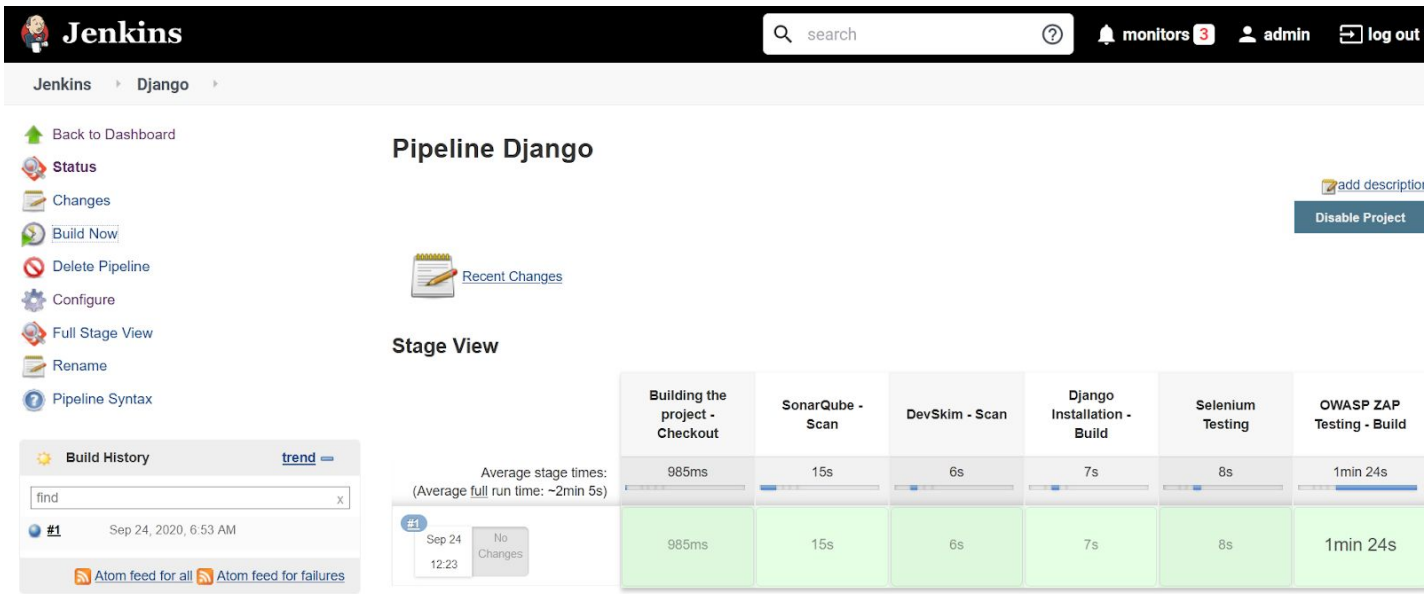
Step 1: Navigate to the Pipeline tab.



Step 2: Click on the “Build Now” button to start the Pipeline.

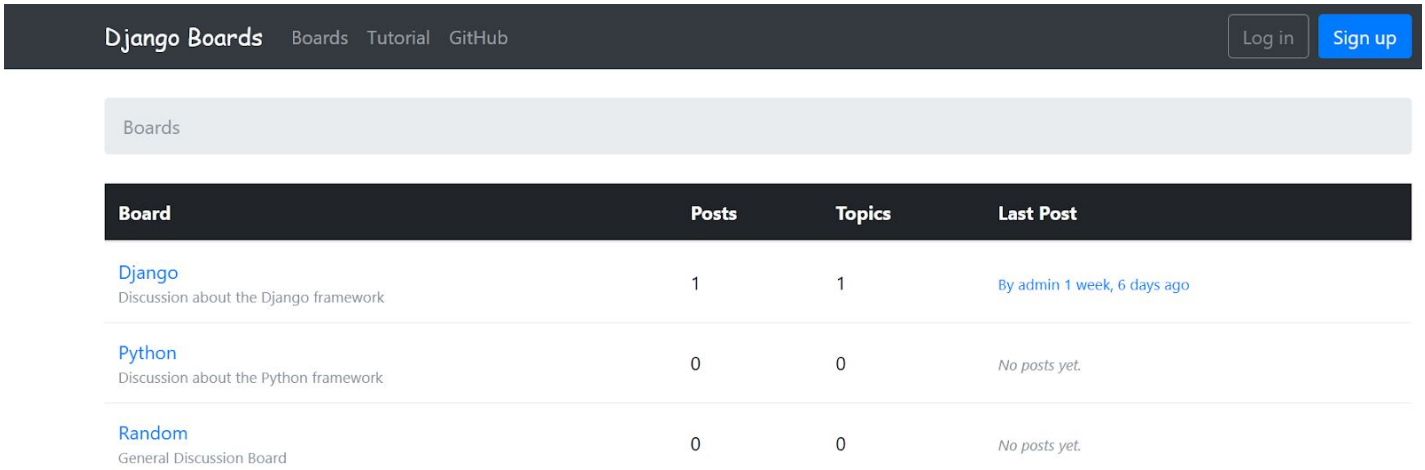


Reload the page to see the recent changes in the pipeline



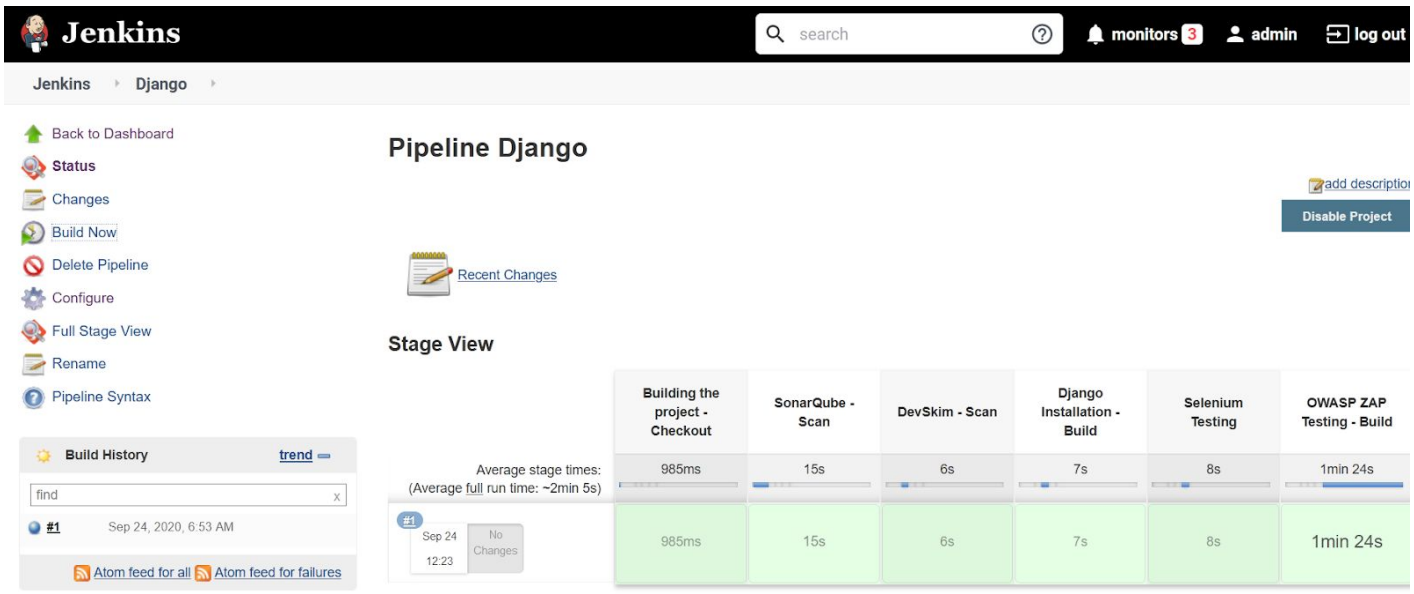
The pipeline completed the execution successfully.

Step 3: Navigate to the deployed website.

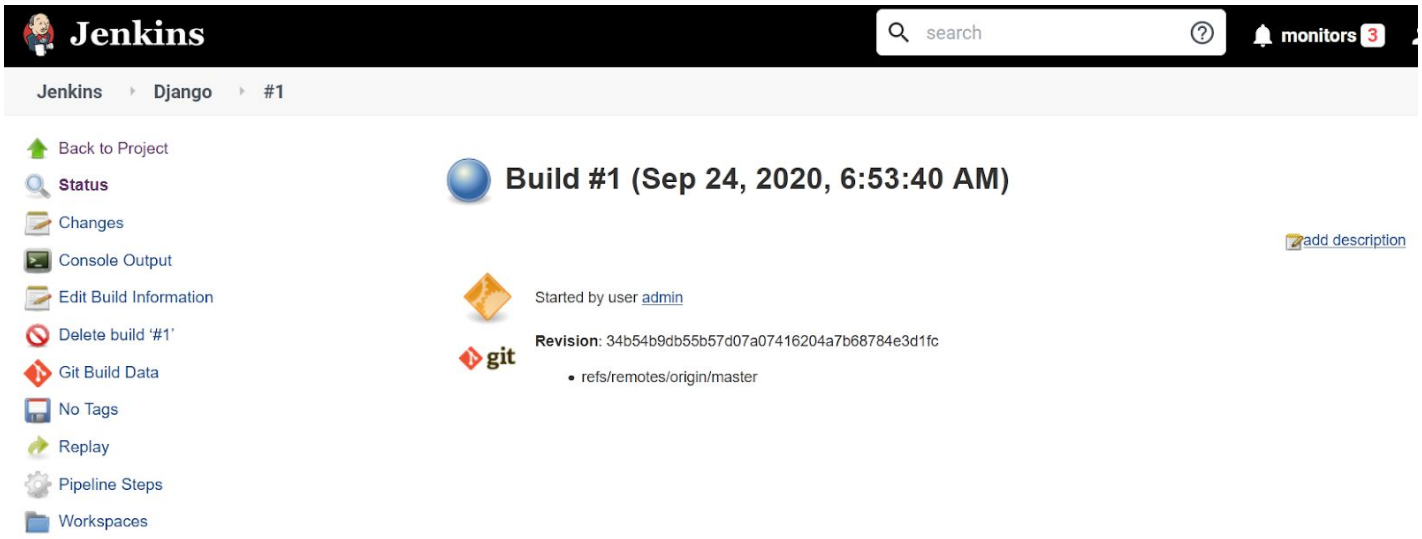


Log Review:

Step 1: Navigate to the Django Job Panel



Step 2: Click on the latest build which in this case is #1



Click on the "Console Output".

SonarQube Scanner:

```
06:53:43  + sonar-scanner
06:53:43  INFO: Scanner configuration file: /opt/sonar-scanner-4.4.0.2170/conf/sonar-scanner.properties
06:53:43  INFO: Project root configuration file: /var/lib/jenkins/workspace/Django/sonar-project.properties
06:53:43  INFO: SonarScanner 4.4.0.2170
06:53:43  INFO: Java 11.0.8 Ubuntu (64-bit)
06:53:43  INFO: Linux 4.15.0-72-generic amd64
06:53:44  INFO: User cache: /var/lib/jenkins/.sonar/cache
06:53:44  INFO: Scanner configuration file: /opt/sonar-scanner-4.4.0.2170/conf/sonar-scanner.properties
06:53:44  INFO: Project root configuration file: /var/lib/jenkins/workspace/Django/sonar-project.properties
06:53:44  INFO: Analyzing on SonarQube server 7.5.0
06:53:44  INFO: Default locale: "en_US", source code encoding: "US-ASCII" (analysis is platform dependent)
06:53:44  INFO: Publish mode
06:53:44  INFO: Load global settings
06:53:45  WARNING: An illegal reflective access operation has occurred
06:53:45  WARNING: Illegal reflective access by com.google.protobuf.UnsafeUtil (file:/var/lib/jenkins/.sonar/cache
/193d1645c91fbb07781506b7df9db0b9/sonar-scanner-engine-shaded-7.5-all.jar) to field java.nio.Buffer.address
06:53:45  WARNING: Please consider reporting this to the maintainers of com.google.protobuf.UnsafeUtil
06:53:45  WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
06:53:45  WARNING: All illegal access operations will be denied in a future release
06:53:45  INFO: Load global settings (done) | time=138ms
06:53:45  INFO: Server id: BF41A1F2-AXS-kayloAGMPm-WAT3F
06:53:45  INFO: User cache: /var/lib/jenkins/.sonar/cache
06:53:45  INFO: Load/download plugins
06:53:45  INFO: Load plugins index
06:53:45  INFO: Load plugins index (done) | time=64ms
06:53:45  INFO: Load/download plugins (done) | time=196ms
06:53:45  INFO: Loaded core extensions:
```

```
06:53:45  INFO: Loaded core extensions:
06:53:46  INFO: Process project properties
06:53:46  INFO: Execute project builders
06:53:46  INFO: Execute project builders (done) | time=4ms
06:53:46  INFO: Load project repositories
06:53:46  INFO: Load project repositories (done) | time=19ms
06:53:46  INFO: Load quality profiles
06:53:54  INFO: Sensor SonarCSS Rules [cssfamily]
06:53:58  INFO: Sensor SonarCSS Rules [cssfamily] (done) | time=3549ms
06:53:58  INFO: Sensor JaCoCo XML Report Importer [jacoco]
06:53:58  INFO: Sensor JaCoCo XML Report Importer [jacoco] (done) | time=4ms
06:53:58  INFO: Sensor SonarJavaXmlFileSensor [java]
06:53:58  INFO: Sensor SonarJavaXmlFileSensor [java] (done) | time=3ms
06:53:58  INFO: Sensor Zero Coverage Sensor
06:53:58  INFO: Sensor Zero Coverage Sensor (done) | time=79ms
06:53:58  INFO: SCM provider for this project is: git
06:53:58  INFO: 34 files to be analyzed
06:53:58  INFO: 34/34 files analyzed
06:53:58  INFO: 14 files had no CPD blocks
06:53:58  INFO: Calculating CPD for 26 files
06:53:58  INFO: CPD calculation finished
06:53:58  INFO: Analysis report generated in 291ms, dir size=173 KB
06:53:58  INFO: Analysis reports compressed in 131ms, zip size=86 KB
06:53:59  INFO: Analysis report uploaded in 350ms
06:53:59  INFO: ANALYSIS SUCCESSFUL, you can browse http://sonar:9000/dashboard?id=django-beginners-guide
06:53:59  INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis
report
06:53:59  INFO: More about the report processing at http://sonar:9000/api/ce/task?id=AXS-499HoAGMPm-WAV4J
06:53:59  INFO: Task total time: 13.418 s
06:53:59  INFO: ------------------------------------------------------------------------
06:53:59  INFO: EXECUTION SUCCESS
06:53:59  INFO: ------------------------------------------------------------------------
06:53:59  INFO: Total time: 15.461s
06:53:59  INFO: Final Memory: 13M/54M
06:53:59  INFO: ------------------------------------------------------------------------
```

The Scanner has finished scanning and sent the data to Sonar Server which can be accessed at ‘sonar’ hostname.

DevSkim:

```
[Pipeline] { (DevSkim - Scan)
[Pipeline] sh
06:53:59  + devskim analyze . -s critical
06:54:06  Issues found: 0 in 0 files
06:54:06  Files analyzed: 3381
06:54:06  Files skipped: 1688
[Pipeline] }
```

Devskim checked the source code for vulnerabilities but found none in the django web application.

OWASP ZAP:

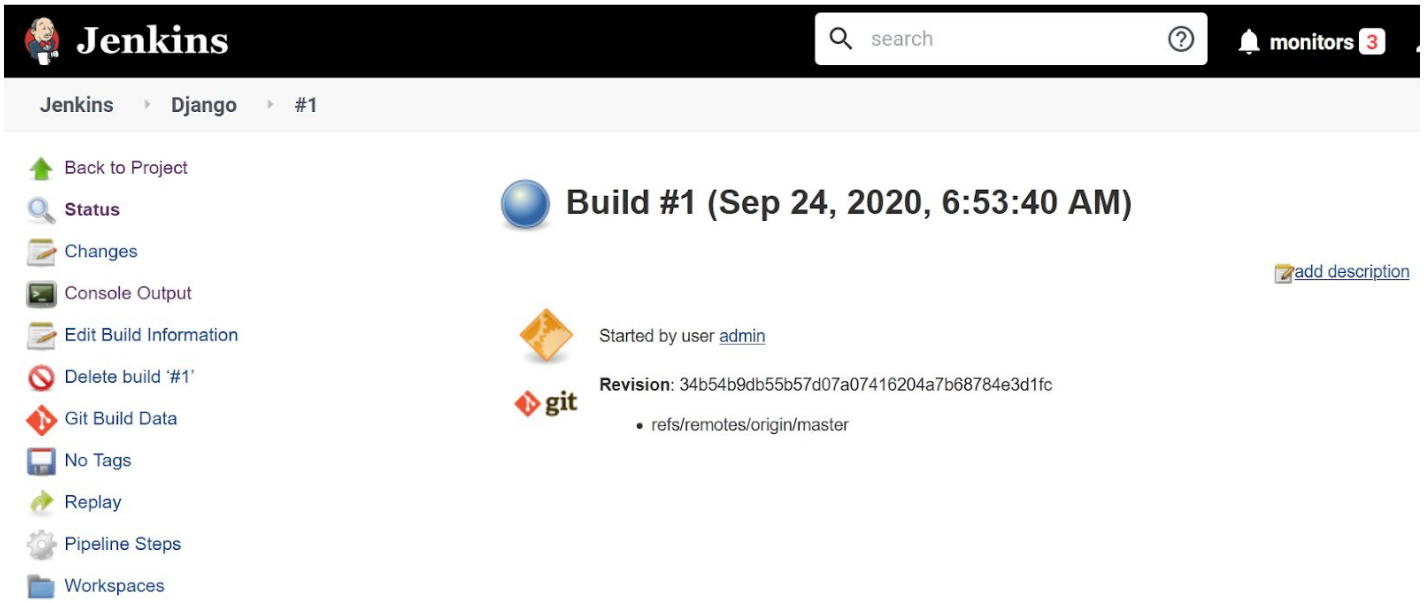
```
06:54:22  + /opt/ZAP_2.9.0/zap.sh -cmd -quickurl http://test-server:8080/ -quickprogress -silent -quickout
/tmp/OWASP_ZAP_REPORT.xml
06:54:22  Found Java version 11.0.8
06:54:22  Available memory: 32166 MB
06:54:22  Using JVM args: -Xmx8041m
06:54:27  No check for updates for over 3 month - add-ons may well be out of date
06:54:27  Accessing URL
06:54:27  Using traditional spider
06:54:30  [====================] 100%
06:54:31  Active scanning
06:55:43  [                    ] 1% |
[                    ] 2% /
[                    ] 2% -
[                    ] 2% \
[                    ] 2% |
[                    ] 2% /
[                    ] 2% -
[                    ] 2% \
[                    ] 2% |
[                    ] 3% /
[                    ] 3% -
[=================   ] 87% -
[==================  ] 90% \
[==================  ] 91% |
[==================  ] 91% /
[==================  ] 91% -
[==================  ] 91% \
[==================  ] 92% |
[==================  ] 93% /
[==================  ] 93% -
[====================] 100%
06:55:43  Attack complete
06:55:44  Writing results to /tmp/OWASP_ZAP_REPORT.xml
06:55:46  + mv /tmp/OWASP_ZAP_REPORT.xml .
```

OWASP ZAP performed dynamic analysis on the deployed application and generated a HTML report.

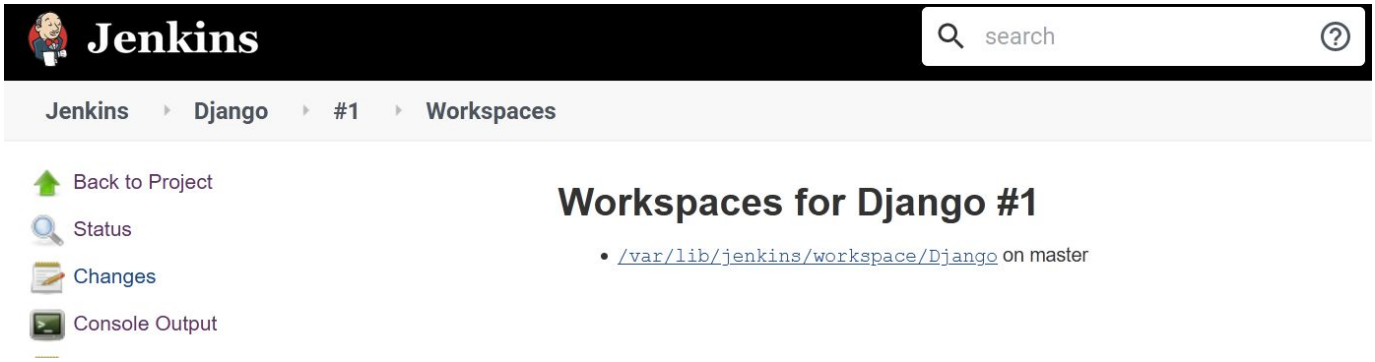
Scan Reports:

Report 1: OWASP ZAP

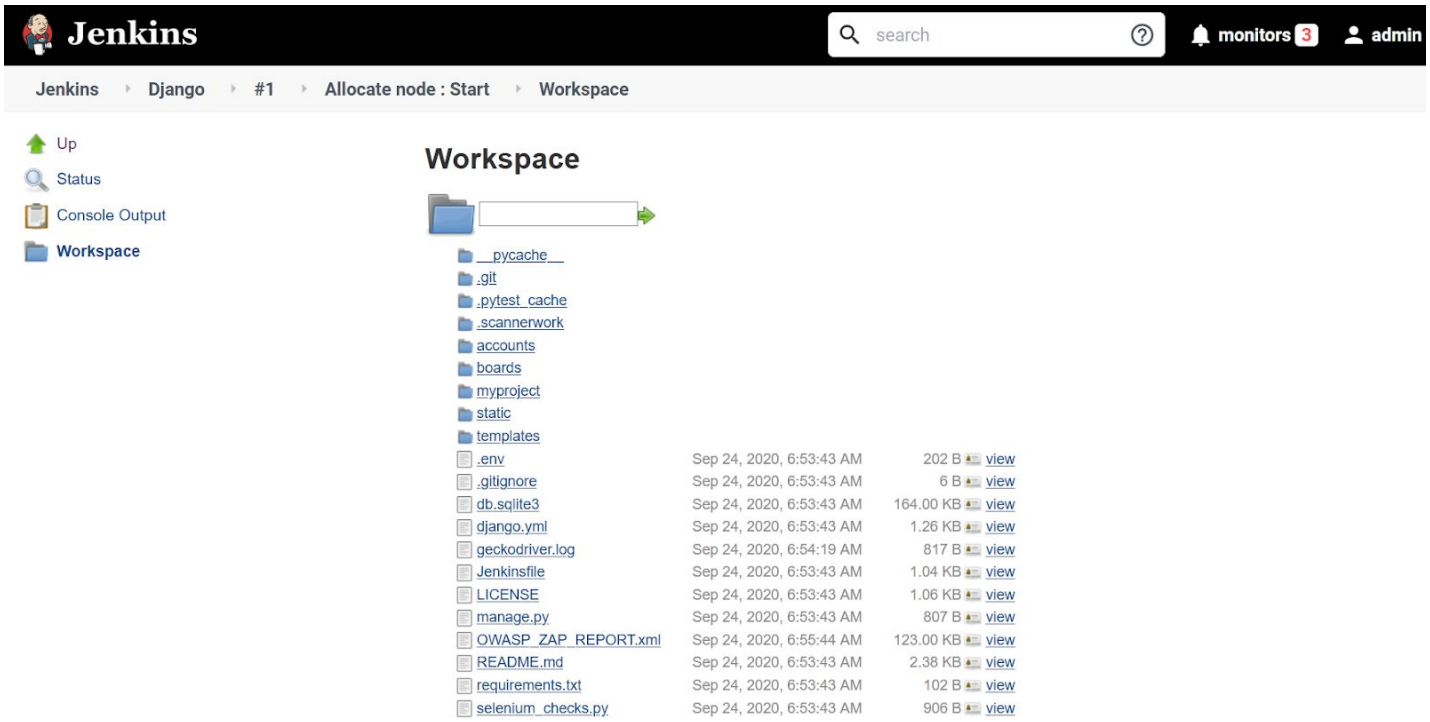
Step 1: Navigate to the Latest build of Pipeline Django which in this case is #1



Step 2: Click on the "Workspaces" section



Click on the Directory Link (/var/lib/jenkins/workspace/Django)



Step 3: Open the OWASP_ZAP_REPORT.xml. This is the report generated by OWASP ZAP after scanning the test-server.


```
<pluginid>10010</pluginid>
<alert>Cookie No HttpOnly Flag</alert>
<name>Cookie No HttpOnly Flag</name>
<riskcode>1</riskcode>
<confidence>2</confidence>
<riskdesc>Low (Medium)</riskdesc>
-<desc>
  <p>A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.</p>
</desc>
-<instances>
 -<instance>
    <uri>http://test-server:8080/login/</uri>
    <method>GET</method>
    <param>csrftoken</param>
    <evidence>Set-Cookie: csrftoken</evidence>
  </instance>
 -<instance>
    <uri>http://test-server:8080/login/?next=/boards/2/new/</uri>
    <method>POST</method>
    <param>csrftoken</param>
    <evidence>Set-Cookie: csrftoken</evidence>
  </instance>
 -<instance>
```

```
<pluginid>10054</pluginid>
<alert>Cookie Without SameSite Attribute</alert>
<name>Cookie Without SameSite Attribute</name>
<riskcode>1</riskcode>
<confidence>2</confidence>
<riskdesc>Low (Medium)</riskdesc>
–<desc>
  <p>A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks.</p>
</desc>
–<instances>
  –<instance>
    <uri>http://test-server:8080/signup/</uri>
    <method>GET</method>
    <param>csrftoken</param>
    <evidence>Set-Cookie: csrftoken</evidence>
  </instance>

<alert>Absence of Anti-CSRF Tokens</alert>
<name>Absence of Anti-CSRF Tokens</name>
<riskcode>1</riskcode>
<confidence>2</confidence>
<riskdesc>Low (Medium)</riskdesc>
–<desc>
  <p>No Anti-CSRF tokens were found in a HTML submission form.</p><p>A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.</p><p></p><p>CSRF attacks are effective in a number of situations, including:</p><p> * The victim has an active session on the target site.</p><p> * The victim is authenticated via HTTP auth on the target site.</p><p> * The victim is on the same local network as the target site.</p><p></p><p>CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.</p>
</desc>
–<instances>
  –<instance>
    <uri>http://test-server:8080/boards/2/</uri>
    <method>GET</method>
    <evidence><form class="form-inline ml-auto"></evidence>
  </instance>

<pluginid>90033</pluginid>
<alert>Loosely Scoped Cookie</alert>
<name>Loosely Scoped Cookie</name>
<riskcode>0</riskcode>
<confidence>1</confidence>
<riskdesc>Informational (Low)</riskdesc>
–<desc>
  <p>Cookies can be scoped by domain or path. This check is only concerned with domain scope.The domain scope applied to a cookie determines which domains can access it. For example, a cookie can be scoped strictly to a subdomain e.g. www.nottrusted.com, or loosely scoped to a parent domain e.g. nottrusted.com. In the latter case, any subdomain of nottrusted.com can access the cookie. Loosely scoped cookies are common in mega-applications like google.com and live.com. Cookies set from a subdomain like app.foo.bar are transmitted only to that domain by the browser. However, cookies scoped to a parent-level domain may be transmitted to the parent, or any subdomain of the parent.</p>
</desc>
–<instances>

<pluginid>10096</pluginid>
<alert>Timestamp Disclosure - Unix</alert>
<name>Timestamp Disclosure - Unix</name>
<riskcode>0</riskcode>
<confidence>1</confidence>
<riskdesc>Informational (Low)</riskdesc>
–<desc>
  <p>A timestamp was disclosed by the application/web server - Unix</p>
</desc>
–<instances>
  –<instance>
```

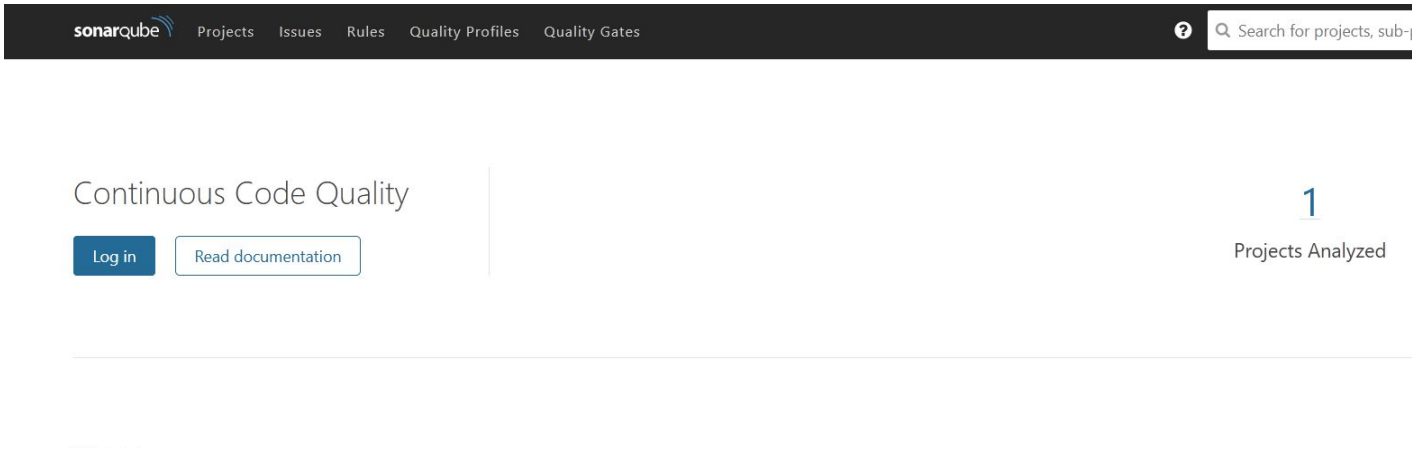
```
<pluginid>10021</pluginid>
<alert>X-Content-Type-Options Header Missing</alert>
<name>X-Content-Type-Options Header Missing</name>
<riskcode>1</riskcode>
<confidence>2</confidence>
<riskdesc>Low (Medium)</riskdesc>
-<desc>
  <p>The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.</p>
</desc>
-<instances>
```


Issues Detected:

- No HTTP Only flag found
- Cookie Without SameSite Attribute
- Absence of Anti-CSRF Tokens
- Loosely Scoped Cookie
- Timestamp Disclosure
- X-Content-Type-Options Header Missing

SonarQube Server

Step 1: Open the SonarQube server from the interface.

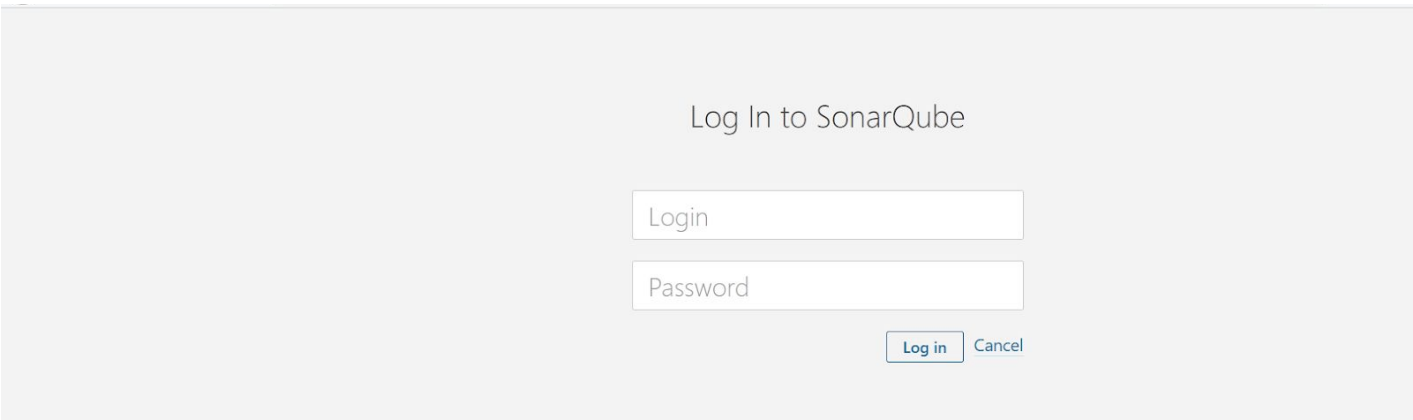


Step 2: Click on the Log in button. Login using the credentials provided in the challenge description

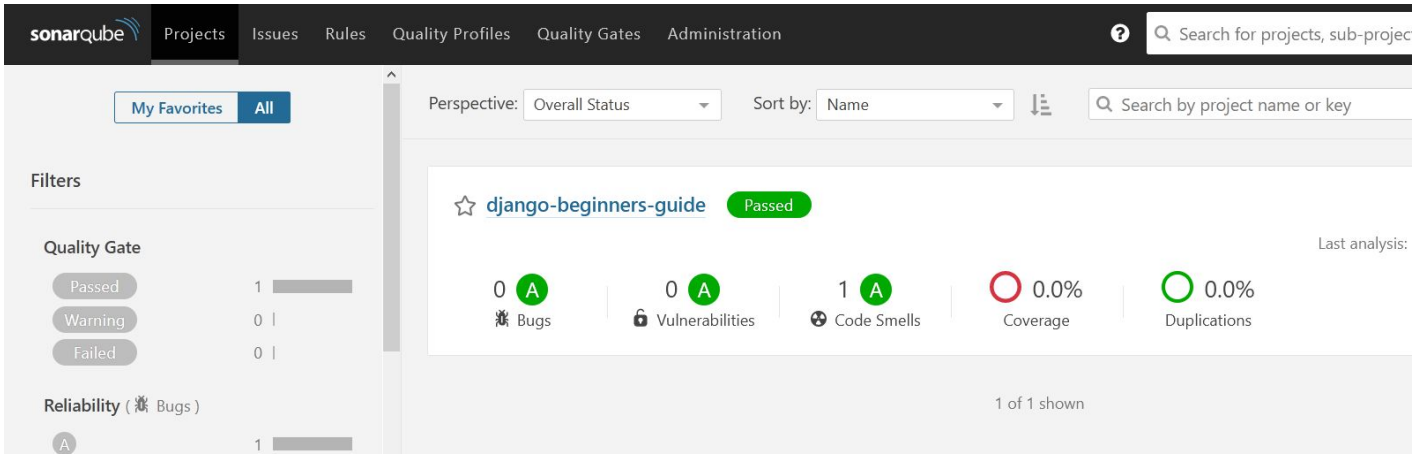
Credentials:

- **Username:** admin
- **Password:** admin

Login Page:

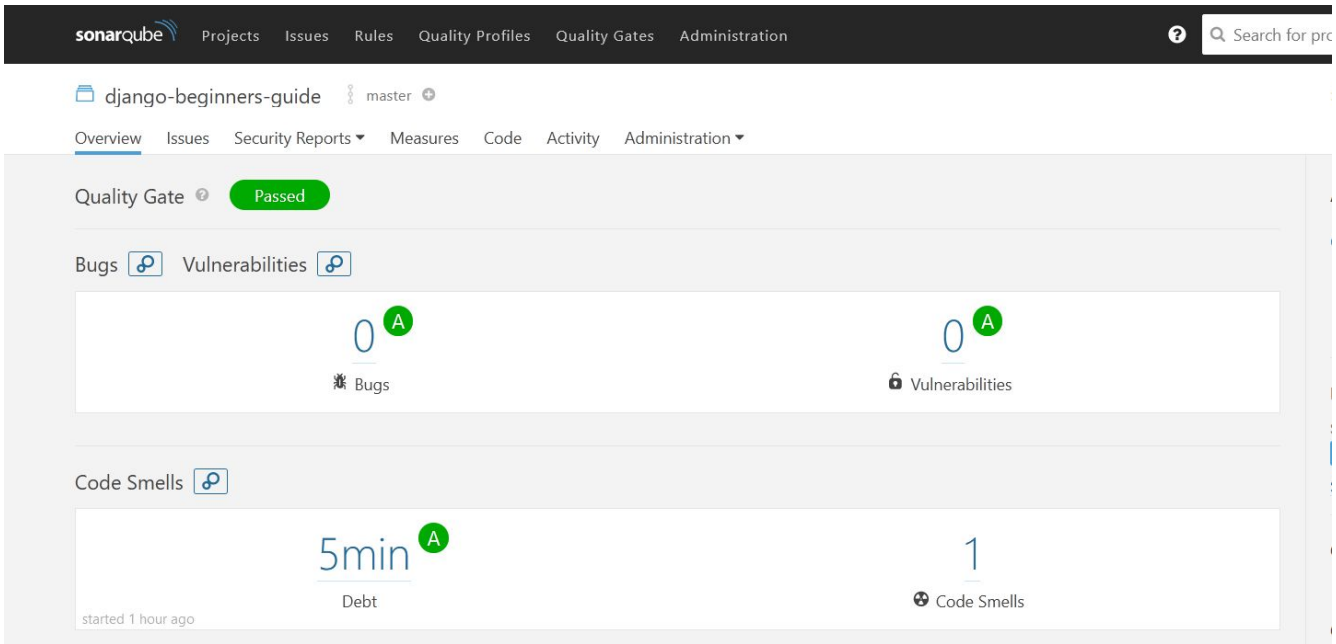


Login Dashboard:

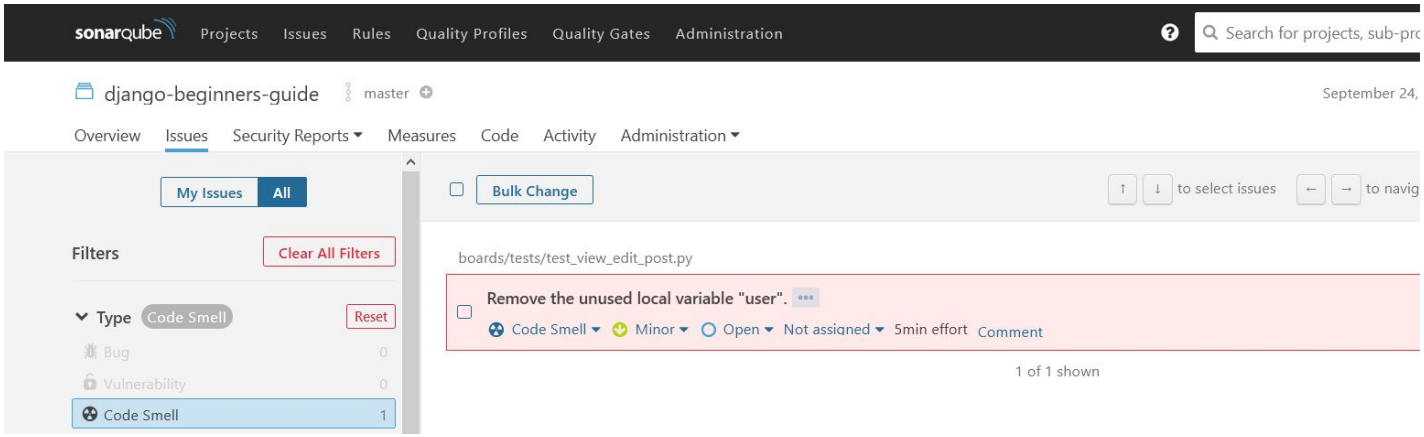


There are no bugs or vulnerabilities found in the source code except the “Code Smells” which means there’s a coding mistake which does not follow the Coding standards.

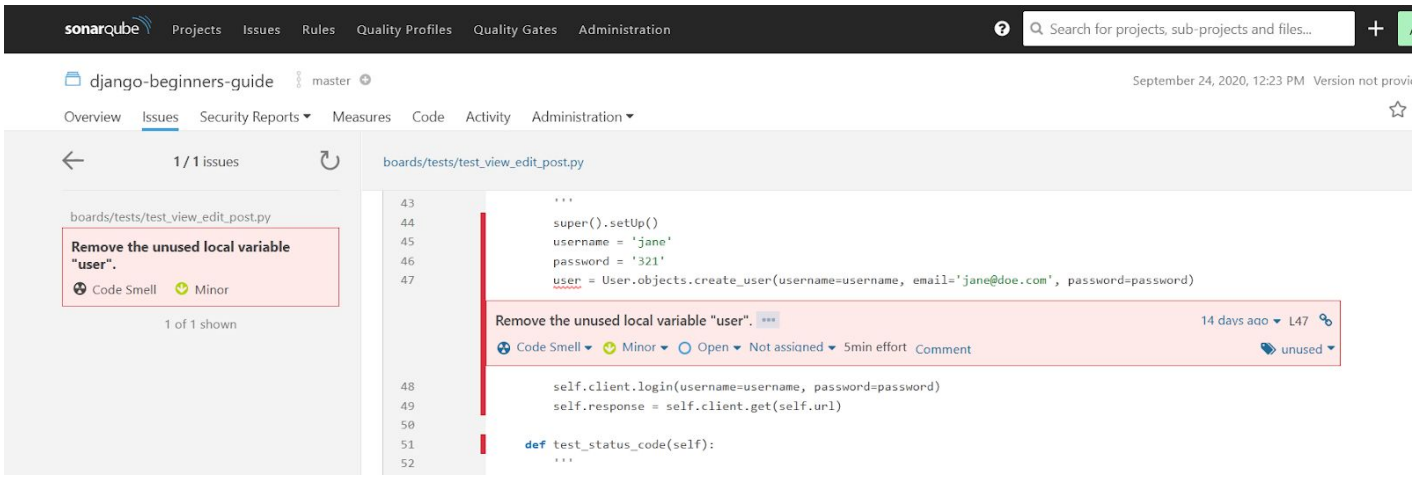
Step 2: Click on the “django-beginners-guide” to get more information.



Step 3: Click on the “Code Smells” option to check the mistake.



Click on the error to get more information.



Issues Detected:

- Unused variable “user” found in the test_view_edit_post.py file.

Learning

Working of a simple DevSecOps pipeline consisting of different components.