Automated Security Scans

Updated: 5/5/2022

Overview

In this lab, students will learn how to scan Docker images stored in Harbor for security vulnerabilities.

Estimated Time: 30 min

Setup needed:

- Introduction to Harbor

Learning Objectives

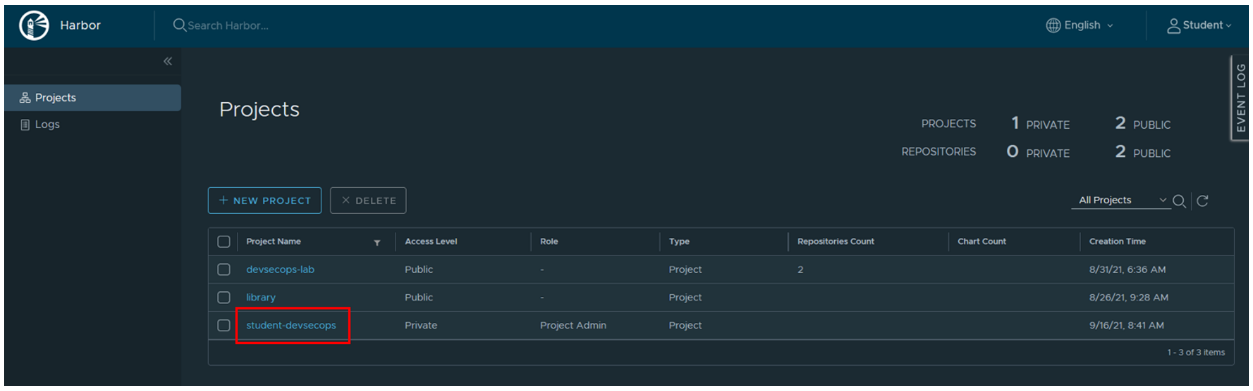
1. Manually Run a Harbor Scan
2. Automate Harbor Scanning
3. Understanding How to Fix Vulnerabilities

Instructions

Manually Run a Harbor Scan

1. In a browser, navigate to Harbor https://registry.dev.afsmtddso.com/ and log in to your account

2. Click on your project



Projects

PROJECTS 1 PRIVATE 2 PUBLIC
REPOSITORIES 0 PRIVATE 2 PUBLIC

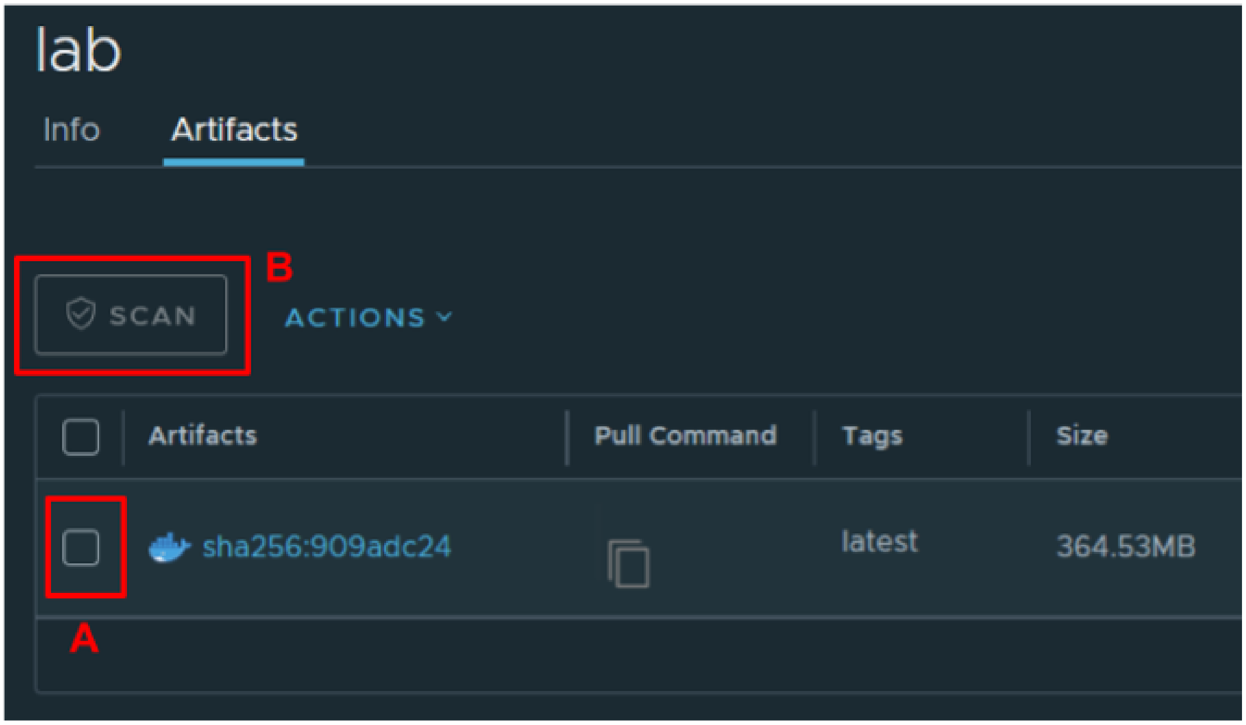
Project Name	Access Level	Role	Type	Repositories Count	Chart Count	Creation Time
devsecops-lab	Public	-	Project	2		8/31/21, 6:36 AM
library	Public	-	Project			8/26/21, 9:28 AM
student-devsecops	Private	Project Admin	Project			9/16/21, 8:41 AM

1 - 3 of 3 items

3. Select the repository with your application in it (it should be `<your harbor project name>/app`)

   a. Once in the repository, click the checkmark next to your image; if you have multiple, check the most recent one at the top

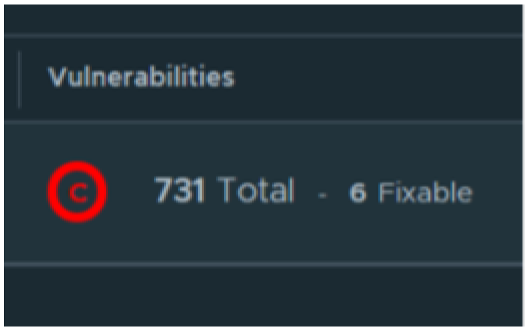
   b. Click the “Scan” button



Artifacts	Pull Command	Tags	Size
sha256:909adc24		latest	364.53MB

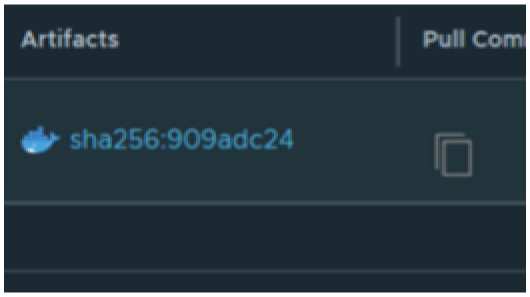
4. You will see your scan be queued, then the scan will occur

5. Once the scan is complete, you should now see information in the “Vulnerabilities” tab

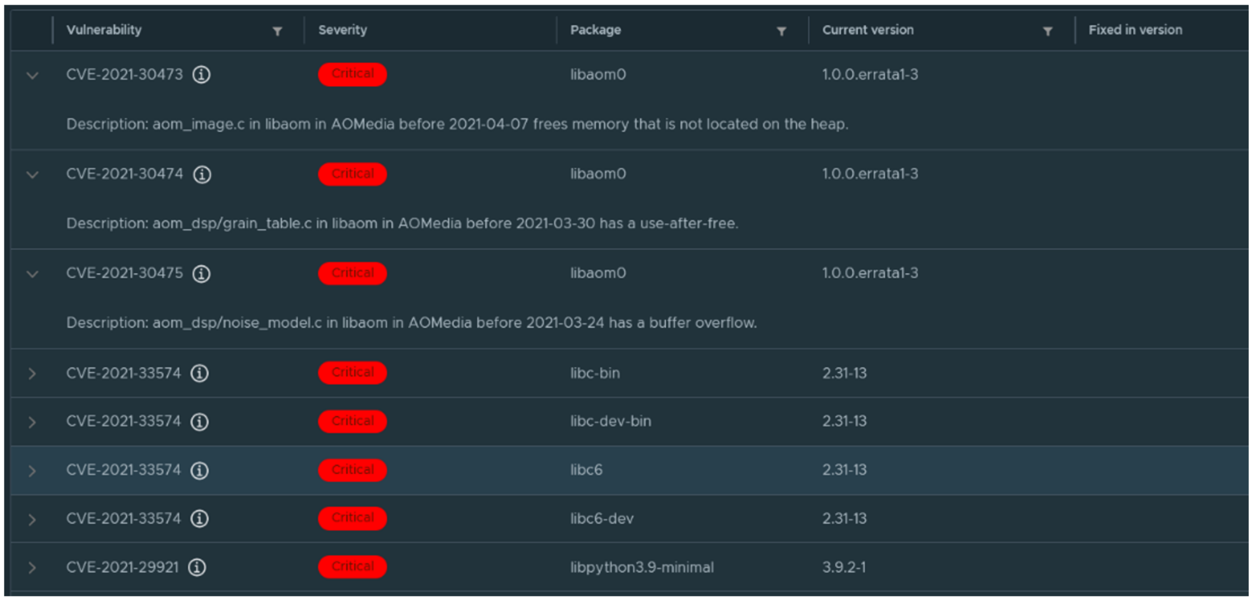

Vulnerabilities

731 Total - 6 Fixable

6. Click on your image's SHA link and scroll down to see a more detailed report of vulnerabilities


Artifacts	Pull Com
sha256:909adc24	

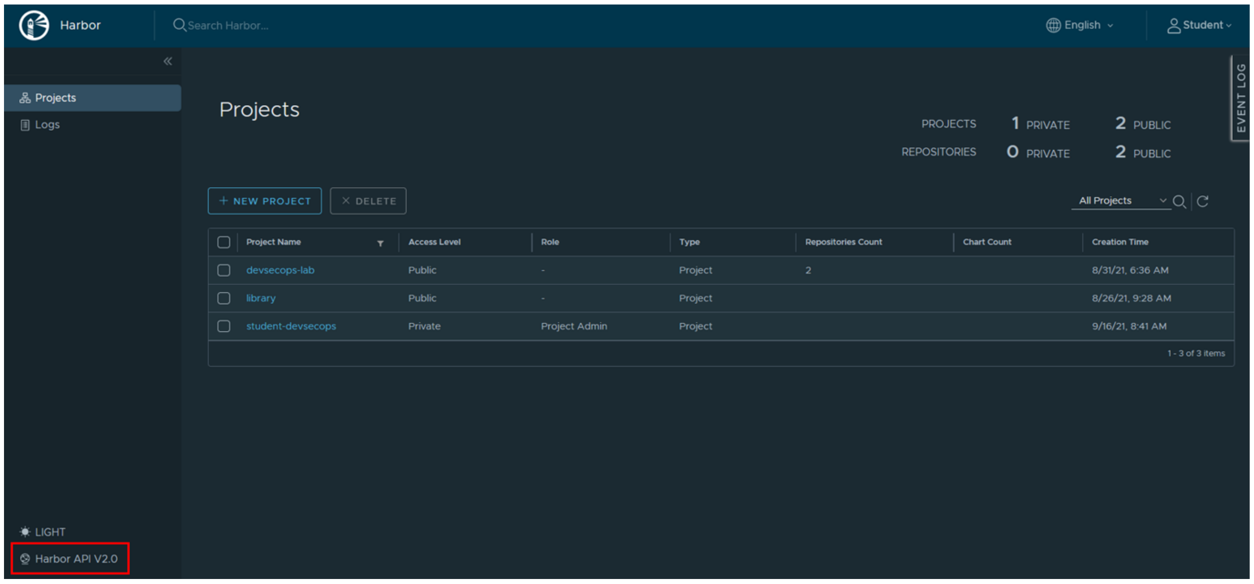
7. Expand some of the vulnerabilities from the list to see a description of what each issue is


Vulnerability	Severity	Package	Current version	Fixed in version
CVE-2021-30473	Critical	libaom0	1.0.0.errata1-3	
Description: aom_image.c in libaom in AOMedia before 2021-04-07 frees memory that is not located on the heap.				
CVE-2021-30474	Critical	libaom0	1.0.0.errata1-3	
Description: aom_dsp/grain_table.c in libaom in AOMedia before 2021-03-30 has a use-after-free.				
CVE-2021-30475	Critical	libaom0	1.0.0.errata1-3	
Description: aom_dsp/noise_model.c in libaom in AOMedia before 2021-03-24 has a buffer overflow.				
CVE-2021-33574	Critical	libc-bin	2.31-13	
CVE-2021-33574	Critical	libc-dev-bin	2.31-13	
CVE-2021-33574	Critical	libc6	2.31-13	
CVE-2021-33574	Critical	libc6-dev	2.31-13	
CVE-2021-29921	Critical	libpython3.9-minimal	3.9.2-1	

Automate Harbor Scanning

1. To automate the scanning process, your pipeline script will run a Python program. The Python program will utilize Harbor API endpoints to scan the Docker images via HTTP protocol.

a. On the Harbor website, click on “Harbor API v2.0” on the bottom left to access the Swagger documentation



Projects

PROJECTS 1 PRIVATE 2 PUBLIC
REPOSITORIES 0 PRIVATE 2 PUBLIC

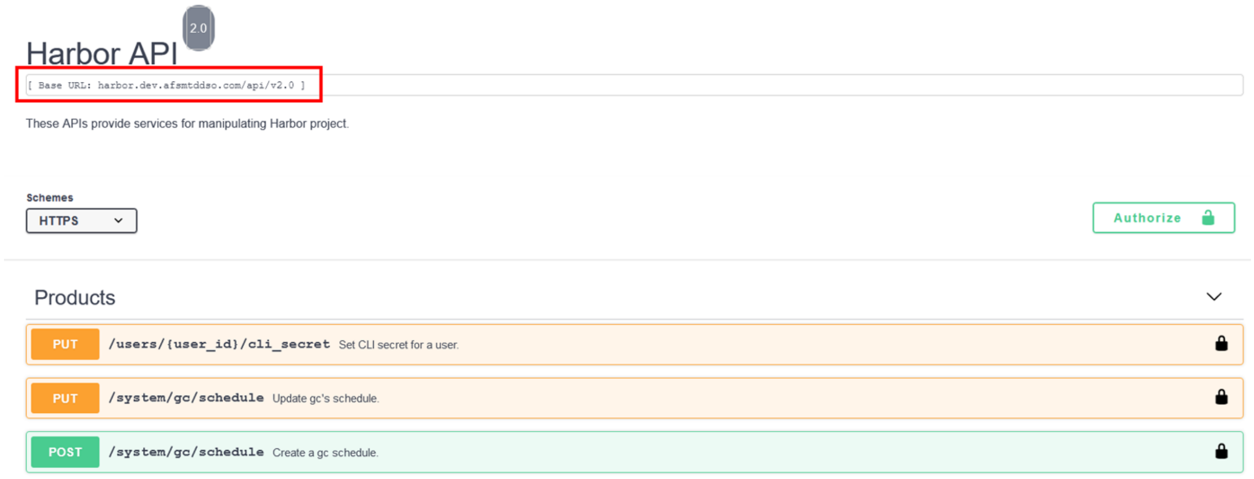
Project Name	Access Level	Role	Type	Repositories Count	Chart Count	Creation Time
devsecops-lab	Public	-	Project	2		8/31/21, 6:36 AM
library	Public	-	Project			8/26/21, 9:28 AM
student-devsecops	Private	Project Admin	Project			9/16/21, 8:41 AM

1 - 3 of 3 items

2. The Swagger documentation will provide you the base URL and a list of available URI endpoints to use for automation.

> **Note:** The following steps 2a to 2c for the Swagger endpoints are for informational purposes.

a. The base URL will be used in your Python program



[Base URL: harbor.dev.afsmtddso.com/api/v2.0]

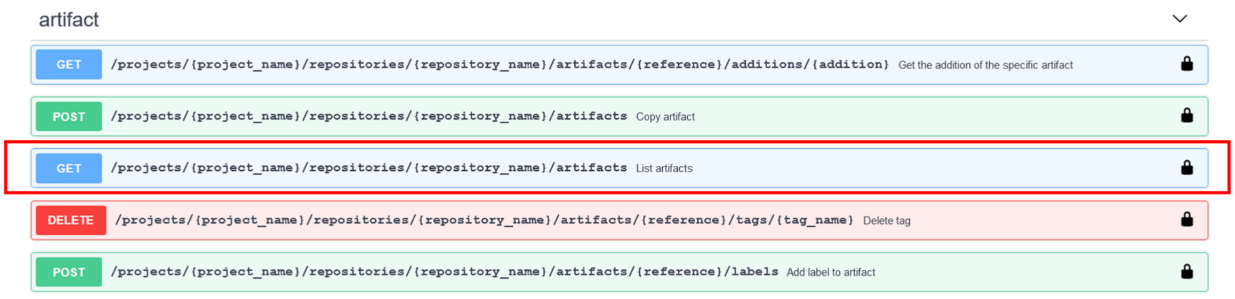
These APIs provide services for manipulating Harbor project.

PUT /users/{user_id}/cli_secret Set CLI secret for a user.
PUT /system/gc/schedule Update gc's schedule.
POST /system/gc/schedule Create a gc schedule.

b. Scroll down to the “artifact” section

i. Locate the URI to list artifacts

ii. This endpoint will list the images in your Harbor repository


artifact

GET /projects/{project_name}/repositories/{repository_name}/artifacts/{reference}/additions/{addition} Get the addition of the specific artifact

POST /projects/{project_name}/repositories/{repository_name}/artifacts Copy artifact

GET /projects/{project_name}/repositories/{repository_name}/artifacts List artifacts

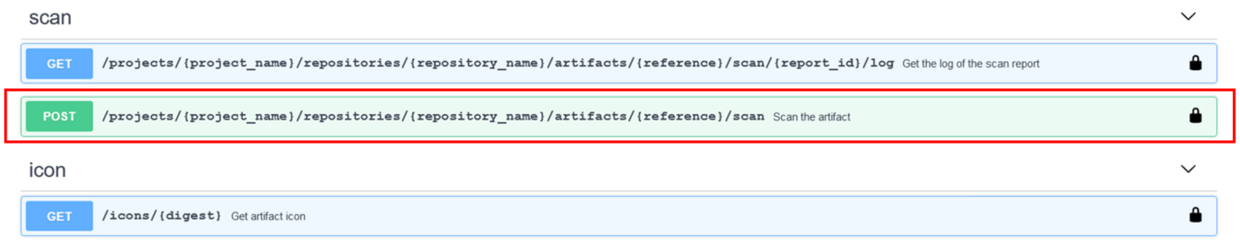
DELETE /projects/{project_name}/repositories/{repository_name}/artifacts/{reference}/tags/{tag_name} Delete tag

POST /projects/{project_name}/repositories/{repository_name}/artifacts/{reference}/labels Add label to artifact

c. Scroll down to the “scan” section

i. Locate the URI to scan an artifact

ii. This endpoint will start an image scan


scan

GET /projects/{project_name}/repositories/{repository_name}/artifacts/{reference}/scan/{report_id}/log Get the log of the scan report

POST /projects/{project_name}/repositories/{repository_name}/artifacts/{reference}/scan Scan the artifact

icon

GET /icons/{digest} Get artifact icon

> **Note:** Clicking on the URI endpoint will show useful information such as query parameters

3. Download the Python script that will be used to scan your images by running the following command in your VSCode terminal, within your `devops-camp-pipeline` folder:

```
curl -O 'https://raw.githubusercontent.com/khaledAFS/sample-files/main/post_security_scan_l
```

4. Please read the following explanations so that you can understand how the script automates the process of scanning your application and database images.

> ***IMPORTANT:*** The following code snippets are in the script you just downloaded, so you don't have to copy and paste them again.

a. We want to configure the script to accept arguments and flags (in this case, your Harbor project name, docker image, registry hostname, and credentials). Later on, we'll add a command to our Jenkinsfile that calls/runs this script along with the aforementioned arguments.

```
import requests, time, sys, json, getopt

## Arguments needed from user ##
argList = sys.argv[1:]
options = 'c:i:p:r:'

arguments, values = getopt.getopt(argList, options)
for currentArgument, currentValue in arguments:
  if currentArgument in ("-c"):
    username, password = currentValue.split(':')
  elif currentArgument in ("-i"):
    imageName = currentValue
  elif currentArgument in ("-p"):
    projectName = currentValue
  elif currentArgument in ("-r"):
    registry = currentValue
```

b. In the following section of the code, we are using the base URL and artifact URI to create a GET request to the Harbor scanner. The request will grab the artifact reference associated with your repository.

```
## Grab sha256 digest from Harbor project repository ##
urlArtifact = 'https://' + registry + '/api/v2.0/projects/' + projectName + '/repositories/
digestResp = requests.get(urlArtifact, auth=(username, password))
artifactReference = digestResp.json()[0]['digest']
```

> **Note:** The artifact reference is a URI parameter to target a specific image resource. In the code snippet above, the SHA256 digest will be used as the artifact reference.

c. Next, we are making a POST request to scan the docker image and checking to ensure the scan completes successfully

```
## Initialize image scanner ##
urlScanInit = urlArtifact + artifactReference + '/scan'
scanInitResp = requests.post(urlScanInit, data={}, auth=(username, password))
if scanInitResp.status_code != 202:
  print('Failed to scan image')
  print('Server response code:', scanInitResp.status_code)
  sys.exit(-1)
```

> **Note:** Use exit codes to control your pipeline when something goes wrong. An exit code of -1 will fail the scanning stage of your pipeline.

d. Then, we are parsing the JSON results from the scan request to display relevant information about project vulnerabilities

```
## Checks scanner status ##
urlScanOverview = urlArtifact + artifactReference + '?with_scan_overview=true'
scanStatus = 'Pending'
maxApiCall = 5

while scanStatus != 'Success':
  scanOverviewResp = requests.get(urlScanOverview, auth=(username, password))
  scanOverviewResult = scanOverviewResp.json()['scan_overview']['application/vnd.security.v
  scanStatus = scanOverviewResult['scan_status']
  print(scanStatus)
  if scanStatus == 'Success':
    break
  elif maxApiCall <= 0:
    print('Reached maximum API calls')
    sys.exit(-1)
  else:
    maxApiCall -= 1
    time.sleep(4)

print(json.dumps(scanOverviewResult['summary'], indent=4))
```

> **Note:** The `?with_scan_overview=true` query parameter will provide information on the status and summary of the scan.

5. After adding your automation script to the `devops-camp-pipeline` directory, we need to add a "Security scanning" stage in `devops-camp-jenkinsfile` *after* the `stage('Application docker build') {}` block to scan the Docker images.

```
    stage('Security scanning') {
            steps {
                withCredentials([usernamePassword(credentialsId: '<YOUR HARBOR CREDENTIAL
                    echo "Scanning $APP_IMAGE_NAME image"
                    sh 'python harbor_scanner.py -i $APP_IMAGE_NAME -r $HARBOR_REGISTRY -
                }
            }
        }
```

> **Note:** Notice how the lines that begin with `sh` use the format explained in step 4a.

> **Tip:** *Replace* `<YOUR HARBOR CREDENTIAL ID>` with your your credentials for Harbor (should be `<first initial + last name>-harbor-auth`) so that the pipeline has authorization to scan your Docker images.

a. Add the same code snippet to your `devops-camp-db-jenkinsfile`, *after* the `stage('Database docker build') {}` block. *Replace* all instances of `$APP_IMAGE_NAME` to `$DB_IMAGE_NAME`

6. Make sure you're in the right directory in the terminal before adding, committing and pushing your changes to GitHub

   a. `git add .`

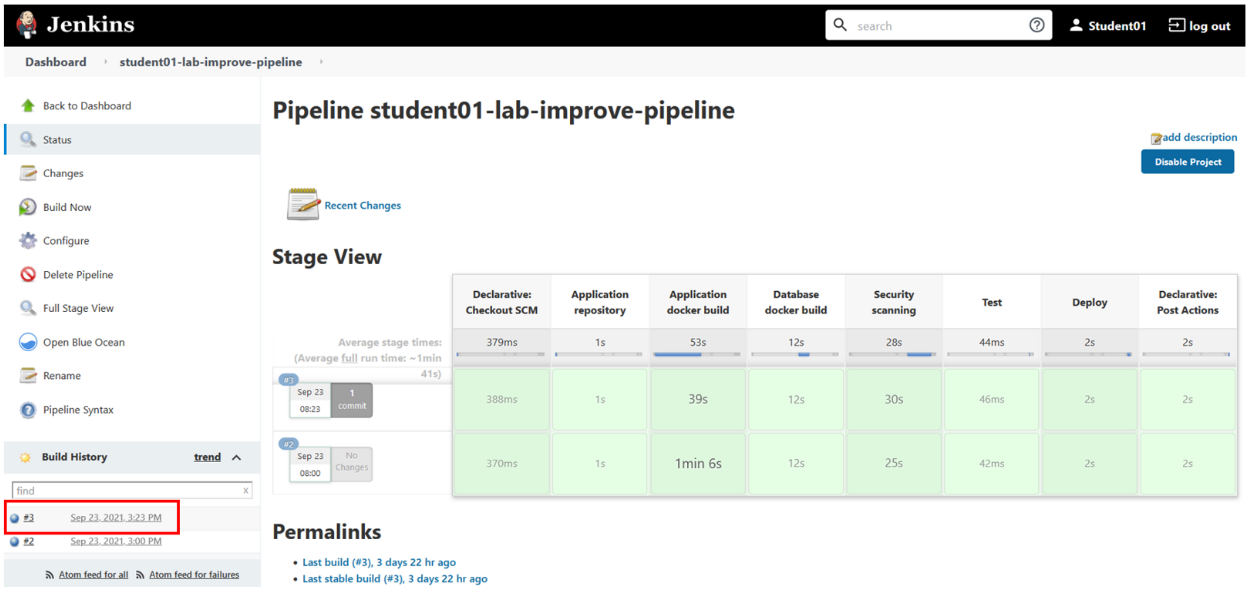
   b. `git commit -m "adding harbor scan file"`

   c. `git push`

You're done if...

1. Your application pipeline builds successfully; i.e., all the stages are green
2. Verify that your Python script completed the scan

   a. Check the status of your application pipeline by clicking on the most recent build in the “Build History” section on the bottom left



Pipeline student01-lab-improve-pipeline

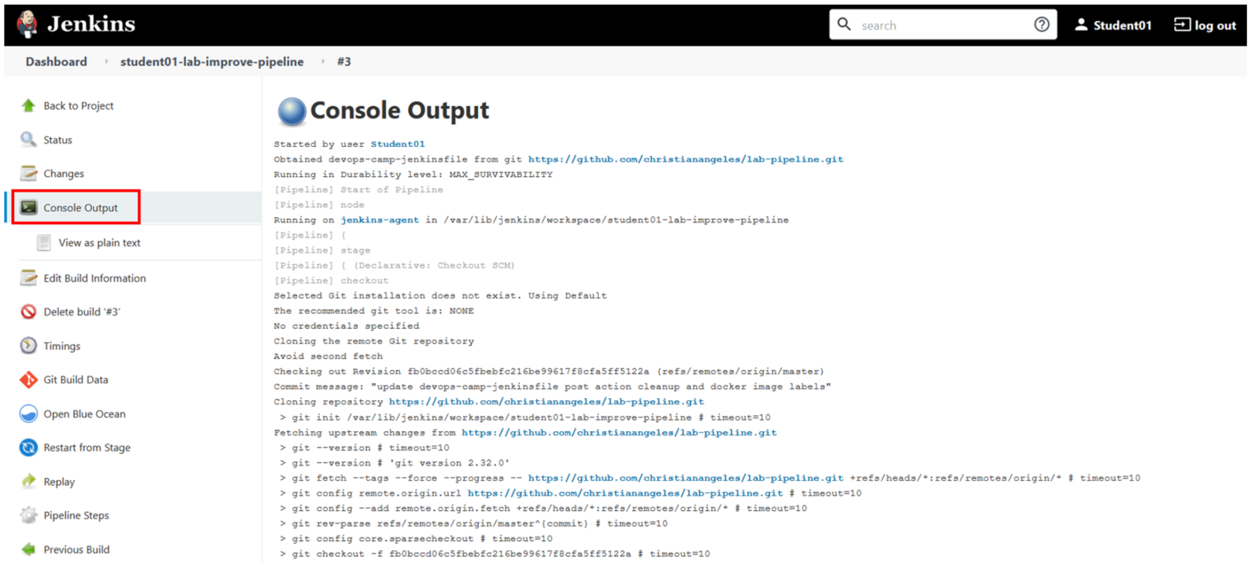
Stage View

	Declarative: Checkout SCM	Application repository	Application docker build	Database docker build	Security scanning	Test	Deploy	Declarative: Post Actions
Average stage times: (Average full run time: ~1min 41s)	379ms	1s	53s	12s	28s	44ms	2s	2s
#3 Sep 23 08:23 1 commit	388ms	1s	39s	12s	30s	46ms	2s	2s
#2 Sep 23 08:00 No Changes	370ms	1s	1min 6s	12s	25s	42ms	2s	2s

Permalinks

- Last build (#3), 3 days 22 hr ago
- Last stable build (#3), 3 days 22 hr ago

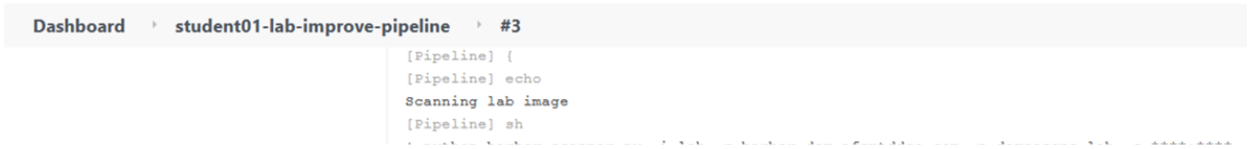
b. Click on “Console Output” to view the output



Console Output

```
Started by user Student01
Obtained devops-camp-jenkinsfile from git https://github.com/christianangeles/lab-pipeline.git
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on jenkins-agent in /var/lib/jenkins/workspace/student01-lab-improve-pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Avoid second fetch
Checking out Revision fb0bccd06c5fbebfc216be99617f8cfa5ff5122a (refs/remotes/origin/master)
Commit message: "update devops-camp-jenkinsfile post action cleanup and docker image labels"
Cloning repository https://github.com/christianangeles/lab-pipeline.git
 > git init /var/lib/jenkins/workspace/student01-lab-improve-pipeline # timeout=10
Fetching upstream changes from https://github.com/christianangeles/lab-pipeline.git
 > git --version # timeout=10
 > git --version # 'git version 2.32.0'
 > git fetch --tags --force --progress -- https://github.com/christianangeles/lab-pipeline.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url https://github.com/christianangeles/lab-pipeline.git # timeout=10
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git config core.sparsecheckout # timeout=10
 > git checkout -f fb0bccd06c5fbebfc216be99617f8cfa5ff5122a # timeout=10
```

c. Verify the summary of your image scan in the “Console Output” by holding down Ctrl+F and searching for "harbor_scanner.py":



```
[Pipeline] {
[Pipeline] echo
Scanning lab image
[Pipeline] sh
```

> **Note:** You'll have to make a change to [`/afs-labs-student/database/database.sql`]() to see the scan's output in your DB pipeline.

```
        "High": 216,
```

Understanding How to Fix Vulnerabilities

```
[Pipeline] echo
```

1. Choose any of your images in Harbor and look through the list of vulnerabilities produced by a scan

   > **Note:** You can find the steps for this in Section 1 of this handout

2. Do some research to determine what some of the issues are and how they would be addressed.
   - Pay attention to the dependencies that are a part of your image—why are vulnerabilities coming from there rather than from the code you wrote?

Appendix