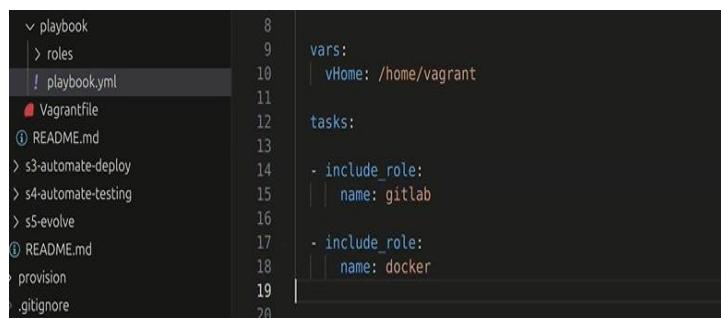


EXERCISE 1

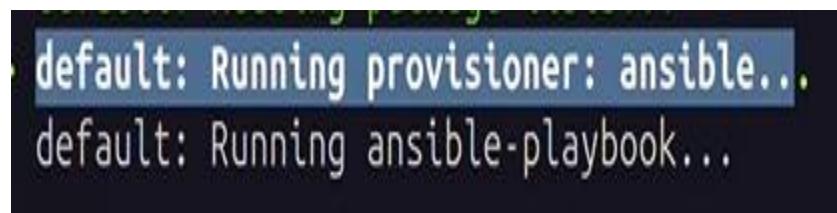
1.a Analyse how the provisioning is done.

Answer:

- When vagrant up is run, it executes an Ansible **playbook** (playbook.yml).
- Provisioning is performed automatically by **Ansible**, an automation tool, which is triggered by Vagrant.
- This playbook contains a series of **tasks** that run on the virtual machine, such as installing dependencies, adding software repositories, and installing the GitLab and Docker packages without any manual intervention.



```
v playbook
| > roles
|   ! playbook.yml
| Vagrantfile
| README.md
> s3-automate-deploy
> s4-automate-testing
> s5-evolve
| README.md
> provision
| .gitignore
```



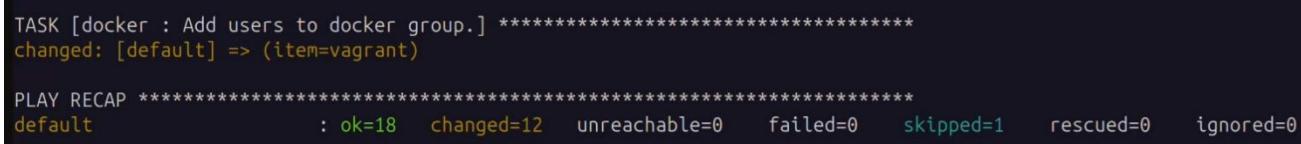
```
default: Running provisioner: ansible...
default: Running ansible-playbook...
```

1.b Is there any "qualities" being addressed by the way the provisioning is done? Justify.

Answer:

Yes, several key software qualities are addressed:

- **Automation:** The entire server setup is scripted from start to finish, which eliminates the need for manual commands and reduces the chance of human error.
- **Repeatability & Consistency:** The Ansible playbook guarantees that the server is set up identically every single time it is created. This ensures every developer has a consistent environment.
- **Maintainability:** The server's configuration is defined as code in the playbook file. This makes it easy to read, track changes in version control (like Git), and modify the server setup in a documented way.



```
TASK [docker : Add users to docker group.] ****
changed: [default] => (item=vagrant)

PLAY RECAP ****
default                  : ok=18   changed=12   unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
```

1.c Add package "nmap" to the playbook.

Answer:

This was achieved by:

1. Editing the Ansible playbook file (e.g., roles/gitlab/tasks/main.yml).
2. Adding nmap to the list of packages to be installed by the apt task (name: "Install GitLab dependencies." then Add - nmap to the list).
3. Re-running the provisioning with the command vagrant provision.
4. Verifying the installation by connecting to the VM with vagrant ssh and running nmap --version to confirm it was installed correctly.

The screenshot shows a terminal window with two parts. On the left, the terminal navigation shows the directory structure: > rgloader, √ playbook, √ roles, > docker, √ gitlab, > defaults, √ tasks, > scripts, ! main.yml, ! playbook.yml. The file main.yml is currently selected. On the right, the content of main.yml is displayed, showing an Ansible task to install GitLab dependencies, including nmap. Below this, the terminal shows the output of the command 'vagrant@integration-server:~\$ nmap --version', which displays the Nmap version 7.80 and platform information.

```
> rgloader
√ playbook
√ roles
  > docker
  √ gitlab
    > defaults
    √ tasks
      > scripts
        ! main.yml
      ! playbook.yml
  Vagrantfile

27   - name: "Install GitLab dependencies."
28     package: name={{ item }} state=present
29     with_items:
30       - openssh-server
31       - ca-certificates
32       - curl
33       - openssl
34       - tzdata
35       - nmap
36
37
38
```

```
vagrant@integration-server:~$ nmap --version
Nmap version 7.80 ( https://nmap.org )
Platform: x86_64-pc-linux-gnu
```

EXERCISE 2

Exercise 2.a: Collect the manual steps you have done up to now.

Answer:

This lists the server configuration steps I did *after* the initial vagrant up command.

1. GitLab Configuration:

- Connected to the VM using vagrant ssh.
- Edited the /etc/gitlab/gitlab.rb file to set the external_url (first to 192.168.33.9, then troubleshooting and correcting it to 192.168.56.9).
- Also in /etc/gitlab/gitlab.rb, uncommented and changed puma['port'] to 8088.

- Ran sudo gitlab-ctl reconfigure and sudo gitlab-ctl restart to apply the changes.

2. Docker Permissions:

- Added the vagrant user to the docker group using sudo usermod -aG docker vagrant.
- Exited and re-connected with vagrant ssh to make the permission change take effect.

3. GitLab Runner Installation and Registration:

- Installed the GitLab Runner software using curl and sudo apt-get install gitlab-runner.
- Manually registered the runner by running sudo gitlab-runner register and interactively providing:
 - The GitLab instance URL.
 - The project-specific registration token.
 - A description, tags, the docker executor, and a default image.
- Restarted the runner with sudo gitlab-runner restart.
- Manually edited the runner in the GitLab web UI to enable "**Run untagged jobs**".

Exercise 2.b: List those that could be added to a playbook.

Answer:

Almost every manual server configuration step can be automated with Ansible.

- **GitLab Configuration:** The changes to /etc/gitlab/gitlab.rb are perfect for automation.
- **Running Reconfigure:** Ansible can be configured to automatically run gitlab-ctl reconfigure only when the gitlab.rb file has actually changed. This is done using "handlers".
- **Docker Permissions:** Adding a user to a group is a standard task that Ansible's user module can handle easily.
- **GitLab Runner Installation:** Installing packages is one of Ansible's most basic and powerful features using the apt module.
- **GitLab Runner Registration:** This can also be automated. The gitlab-runner register command can be run in "non-interactive" mode with all the answers provided as command-line arguments. Ansible's command module can execute this.

The only step that is difficult to automate with a simple Ansible playbook is clicking the checkbox in the web UI, as that would typically require using the GitLab API.

Exercise 2.c: Add (at least) half of them to a playbook.

Answer:

First, it configures the gitlab.rb file with the correct URL and port. Second, it ensures the vagrant user has the right permissions for Docker. Third, and most importantly, it automates the runner registration by running the command in non-interactive mode. This playbook even handles running gitlab-ctl reconfigure automatically if the configuration file changes. This approach is far more robust and repeatable than performing the steps manually.

Here is a code I added to my playbook.yml file to automate the GitLab configuration and Docker permissions.

```
jabin@Jabin: ~/Desktop/3rd_sem/software_engineering/devops/pipeline/s2-automate-build/integration-server
GNU nano 7.2
playbook-exercise-2c.yml *

---
- hosts: all
  become: yes
  tasks:
    - name: Configure GitLab external_url and Puma port
      lineinfile:
        path: /etc/gitlab/gitlab.rb
        regexp: "{{ item.regex }}"
        line: "{{ item.line }}"
      loop:
        - { regexp: "external_url", line: "external_url 'http://192.168.56.9/gitlab'" }
        - { regexp: "# puma['port']=8080", line: "puma['port']=8080" }
      notify: Reconfigure GitLab

    - name: Ensure vagrant user is in the docker group
      user:
        name: vagrant
        groups: docker
        append: yes

    - name: Register the GitLab Runner (non-interactive)
      command: >
        gitlab-runner register
        --non-interactive
        --url "http://192.168.56.9/gitlab/"
        --registration-token "YOUR_REGISTRATION_TOKEN"
        --executor "docker"
        --docker-image alpine:latest
        --description "[integration-server] docker"
        --tag-list "integration"
        --run-untagged="true"
        --locked="false"
      args:
        creates: /etc/gitlab-runner/config.toml

    - name: Ensure GitLab Runner service is started and enabled
      service:
        name: gitlab-runner
        state: started
        enabled: yes

  handlers:
    - name: Reconfigure GitLab
      command: gitlab-ctl reconfigure
```

EXERCISE 3

3.a) Use GitLab to analyse what has just happened?

Answer:

- When I pushed the .gitlab-ci.yml file to the repository, GitLab automatically detected it and triggered a new CI/CD pipeline.
- The pipeline followed the instructions in the file: it created a temporary Docker container using the maven:latest image.
- It then executed the three stages in order: build (running mvn compile), test (running mvn test), and run (running mvn package and mvn exec:java).

The screenshot shows the GitLab Pipelines interface for the 'HelloWorldMaven' project. There is one pipeline listed under the 'All' tab. The pipeline has a status of 'Passed' and was created by 'Jabin Urmy' on '00:03:59' ago. The pipeline consists of three stages: build, test, and run, all of which are marked as successful (green checkmarks). The pipeline ID is '#2' and it is associated with the 'master' branch.

3.b) Try to run again the CI and inspect the intermediate results of each stage.

Answer:

- To run the pipeline again, I went to the **Build -> Pipelines** page and click the blue "Run pipeline" button.
- To inspect the results, I checked on the pipeline's status icon (e.g., the green checkmark). This shows the individual jobs (build_app, test_app, run_app).

The screenshot shows the GitLab Pipelines interface for the 'HelloWorldMaven' project. There are two pipeline runs listed under the 'All' tab. Both runs have a status of 'Passed' and were created by 'Jabin Urmy'. The first run was completed '00:00:56' ago, and the second run was completed '00:03:59' ago. Both runs are associated with the 'master' branch. Each run consists of three stages: build, test, and run, all of which are marked as successful (green checkmarks).

Build app:

- The log shows Maven successfully downloading all the necessary dependencies from the central repository.

- The stage concludes with the message **[INFO] BUILD SUCCESS**, which is the intermediate result proving that the Java source code compiled without any errors.

Jabin Urmy / HelloWorldMaven / Jobs / #19

```

167 Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-compiler-javac/2.15.0/plexus-compiler-javac-2.15.0.jar
168 Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-compiler-javac/2.15.0/plexus-compiler-javac-2.15.0.pom (1.3 kB at 39 kB/s)
169 Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-compilers/2.15.0/plexus-compilers-2.15.0.pom
170 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/3.4.2/maven-shared-utils-3.4.2.jar
171 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/3.4.2/maven-shared-utils-3.4.2.jar
172 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/3.4.2/maven-shared-utils-3.4.2.jar (151 kB at 3.2 MB/s)
173 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-incremental/1.1/maven-shared-incremental-1.1.jar
174 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-incremental/1.1/maven-shared-incremental-1.1.jar
175 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-incremental/1.1/maven-shared-incremental-1.1.pom
176 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-incremental/1.1/maven-shared-incremental-1.1.pom
177 Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-compiler-anl/2.15.0/plexus-compiler-anl-2.15.0.jar
178 Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-compiler-anl/2.15.0/plexus-compiler-anl-2.15.0.pom (16.9 kB at 32 kB/s)
179 Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-compiler-manager/2.15.0/plexus-compiler-manager-2.15.0.pom
180 Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-compiler-manager/2.15.0/plexus-compiler-manager-2.15.0.jar (59 kB at 1.2 MB/s)
181 Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-xnl/3.0.0/plexus-xnl-3.0.0.pom
182 Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-xnl/3.0.0/plexus-xnl-3.0.0.jar (59 kB at 549 kB/s)
183 Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-xnl/3.0.0/plexus-xnl-3.0.0.pom
184 Downloaded from central: https://repo.maven.apache.org/maven2/org/easybeans/asm/9.6/asm-9.6.jar (124 kB at 1.6 MB/s)
185 Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/4.0.0/plexus-utils-4.0.0.jar
186 Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/4.0.0/plexus-utils-4.0.0.pom
187 Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-xm/3.0.0/plexus-xm-3.0.0.jar (93 kB at 949 kB/s)
188 Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-compiler-manager/2.15.0/plexus-compiler-manager-2.15.0.pom (5.2 kB at 53 kB/s)
189 Downloaded from central: https://repo.maven.apache.org/maven2/com/thoughtworks/dox/dox/2.0.3/dox-2.0.3.jar (334 kB at 3.0 MB/s)
190 Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/4.0.0/plexus-utils-4.0.0.jar (192 kB at 1.3 MB/s)
191 [INFO] Nothing to compile - all classes are up to date.
192 [INFO] -----
193 [INFO] [INFO] BUILD SUCCESS
194 [INFO] -----
195 [INFO] Total time: 4.099 s
196 [INFO] Finished at: 2025-10-20T10:30:55Z
197 [INFO] -----
198 Saving cache for successful job
199 Creating cache default-protected...
200 target found 31 matching artifact files and directories
201 No URL provided, cache will not be uploaded to shared cache server. Cache will be stored only locally.
202 Created cache
203 Cleaning up project directory and file based variables
204 Job succeeded
  
```

Duration: 12 seconds
Finished: 1 day ago
Queued: 2 seconds
Timeout: 1h (from project)
Runner: #1 (lx04GDD) [integration-server docker]
Source: Web
Commit: 76f9e6d5
Add .gitlab-ci.yml to configure the CI pipeline
Pipeline #5 Passed for master build
Related jobs → build_app

Test app:

- The log shows the Maven Surefire Plugin executing the unit tests.
- The summary at the end, **[INFO] Results: Tests run: 1, Failures: 0, Errors: 0, Skipped: 0**, is the intermediate result. It confirms that all automated tests passed, ensuring the code quality meets the requirements.

Jabin Urmy / HelloWorldMaven / Jobs / #20

```

356 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-providers/3.2.5/surefire-providers-3.2.5.pom
357 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-providers/3.2.5/surefire-providers-3.2.5.pom (2.6 kB at 54 kB/s)
358 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-testng/3.2.5/surefire-testng-3.2.5.pom
359 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-testng/3.2.5/surefire-testng-3.2.5.pom (3.1 kB at 78 kB/s)
360 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-common/3.2.5/common-junit-3.2.5/common-junit-3.2.5.pom
361 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-common/3.2.5/common-junit-3.2.5/common-junit-3.2.5.com (2.0 kB at 54 kB/s)
362 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-common/3.2.5/common-junit-3.2.5/common-junit-3.2.5.pom
363 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit/3.2.5/junit-jupiter/5.9.1/junit-jupiter-5.9.1.pom
364 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit/3.2.5/junit-jupiter/5.9.1/junit-jupiter-5.9.1.pom (2.8 kB at 51 kB/s)
365 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit/3.2.5/junit-jupiter/5.9.1/junit-jupiter-5.9.1.pom
366 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit/3.2.5/junit-jupiter/5.9.1/junit-jupiter-5.9.1.pom (18 kB at 379 kB/s)
367 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-common/3.2.5/common-junit-3.2.5/common-junit-3.2.5.pom
368 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-common/3.2.5/common-junit-3.2.5.common-junit-3.2.5.pom
369 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-common/3.2.5/common-junit-3.2.5.common-junit-3.2.5.pom (18 kB at 428 kB/s)
370 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-common/3.2.5/common-junit-3.2.5.common-junit-3.2.5.pom
371 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-common/3.2.5/common-junit-3.2.5.common-junit-3.2.5.pom (18 kB at 409 kB/s)
372 [INFO] -----
373 [INFO] T E S T S
374 [INFO] -----
375 [INFO] Running com.jog.maven.AppTest
376 [INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0. Time elapsed: 0.032 s -- in com.jog.maven.AppTest
377 [INFO] -----
378 [INFO] Results:
379 [INFO] -----
380 [INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
381 [INFO] -----
382 [INFO] BUILD SUCCESS
383 [INFO] -----
384 [INFO] BUILD SUCCESS
385 [INFO] -----
386 [INFO] Total time: 04:42 min
387 [INFO] Finished at: 2025-10-20T10:41:48Z
388 [INFO] -----
389 Saving cache for successful job
390 Creating cache default-protected...
391 target found 31 matching artifact files and directories
392 No URL provided, cache will not be uploaded to shared cache server. Cache will be stored only locally.
393 Created cache
394 Cleaning up project directory and file based variables
395 Job succeeded
  
```

Duration: 4 minutes 50 seconds
Finished: 1 day ago
Queued: 2 seconds
Timeout: 1h (from project)
Runner: #1 (lx04GDD) [integration-server docker]
Source: Web
Commit: 76f9e6d5
Add .gitlab-ci.yml to configure the CI pipeline
Pipeline #5 Passed for master test
Related jobs → test_app

Run app:

- This log shows two key results. First, Maven packages the compiled code into a distributable file, indicated by the message **[INFO] Building jar:**
- Second, the application is executed directly on the runner, and its output, **Hello World!**, is printed to the console. This proves that the packaged application is executable and

The screenshot shows the GitLab CI interface for a project named 'HelloWorldMaven'. The 'Jobs' tab is selected. The log output shows the following sequence of events:

- Maven downloading dependencies from central repository.
- Maven building a jar file: `[INFO] Building jar: /tmp/196.k/24/maven-resolver-util-1.9.24.jar`
- The application is run, and it outputs "Hello World!".
- Maven executing the application: `[INFO] BUILD SUCCESS`
- The total build time is listed as 1.887 s.
- The job is marked as finished at 2023-10-26T18:42:15Z.
- Cleaning up the project directory and file based variables.
- The job is marked as succeeded.

Additional pipeline details shown on the right side of the interface include:

- Duration: 25 seconds
- Finished: 1 day ago
- Queued: 0 seconds
- Timeout: 1h (from project)
- Runner: #1 (lx04G4D0) (integration-server docker)
- Source: Web
- Commit: 0995ab5
- Add .gitlab-ci.yml to configure the CI pipeline
- Pipeline #5: **Passed** for master
- run
- Related jobs: run_app (green checkmark)

3.c) Where is the .jar file generated as result of the build?

Answer:

The .jar file is generated in the target/ directory during the run_app job.

Because the .gitlab-ci.yml file includes a cache: directive for the target/ path, the GitLab Runner saves this directory's contents after the job completes.

This cache is stored on the server inside a Docker volume managed by the GitLab Runner, located at a path similar to /var/lib/docker/volumes/.../_data/.

Crucially, while the file is saved, it is not a "build artifact." Its purpose is to speed up subsequent pipeline runs, not for users to download. The file is not easily accessible from the GitLab UI at this stage.

EXERCISE 4

Exercise 4.a: Find out what is the result of this modification.

Answer:

A deploy Stage and artifacts Keyword were Added: The `.gitlab-ci.yml` was modified to include a new deploy stage. This stage uses the artifacts keyword, which is specifically designed to capture and store the final products of a job.

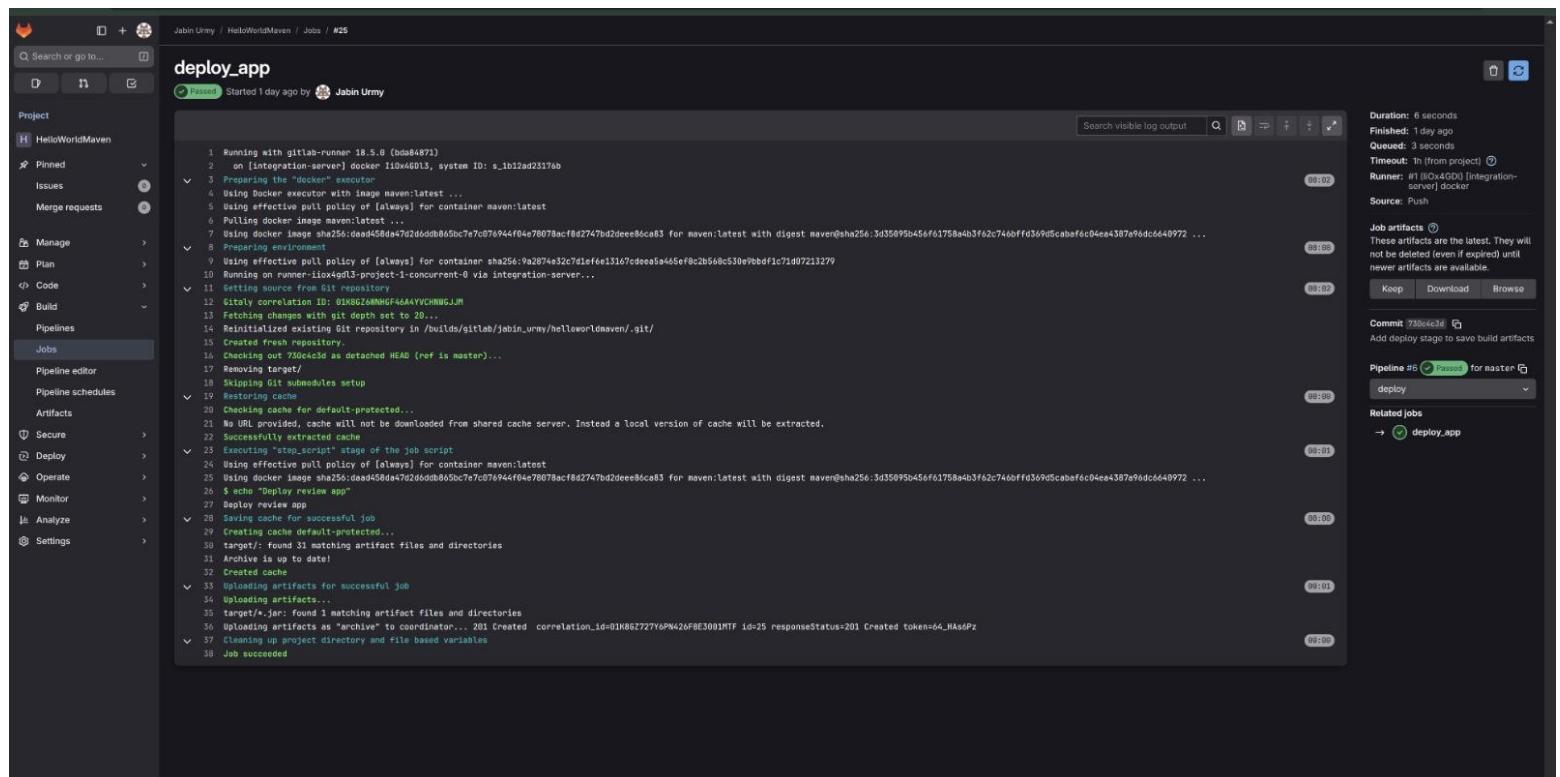
It Solves the Problem from Exercise 3: This is fundamentally different from the cache. The cache is for temporary storage to speed up future jobs, while artifacts are for permanently preserving the valuable output of a pipeline for users.

The Result is a Downloadable, Working Application:

The most visible result is the download button now available on the completed pipeline page in GitLab.

I verified the entire end-to-end process by downloading the artifact, extracting the .jar file, and executing it.

The successful execution (cowsay Hello World) provides definitive proof that the CI pipeline is correctly building, testing, and packaging a valid, distributable application.



The screenshot shows a GitLab pipeline log for a job named "deploy.app". The log details the execution of a CI pipeline, starting with "Running with gitlab-runner 18.5.0 (bd8a871)" and ending with "Job succeeded". The log includes steps for preparing the Docker executor, pulling the maven image, building the project, and deploying the artifact. It also shows the creation of a fresh repository and the execution of a "staging_script" stage. The pipeline is marked as "Passed" and has a duration of 6 seconds. A "Pipeline #2" is listed as related, and there is a "Download" button available for the artifact.

```
1 Running with gitlab-runner 18.5.0 (bd8a871)
2 on [integration-server] docker 110x46013, system ID: s_1b12ad23176b
3 Preparing the "docker" executor
4 Using Docker executor with image maven:latest ...
5 Using effective pull policy of [always] for container maven:latest
6 Pulling docker image maven:latest ...
7 Using docker image sha256:dead458da47d2d6db8b65bc7e7c876944f04e70878acf8d2747bd2deeb86ce83 for maven:latest with digest maven@sha256:3d35095b456f61758a4b3f62c746bffd369d5cabef6c04e4387e76dc6640972 ...
8 Preparing environment
9 Using effective pull policy of [always] for container sha256:9e2074e32c7d1fe13167cd0ea5a45ef0c2b560c530e9ebdf1c71d07213279
10 Running on runner-110x46013-project-1-concurrent-0 via integration-server...
11
12 GitLab correlation ID: 03K8Z6MMNF46AAVWCHNGJHM
13 Fetching changes with sit depth set to 20...
14 Reinitializing existing Git repository in /var/lib/gitlab/jabin_urmy/helloworldmaven/.git/
15 Created fresh repository
16 Checking out 750ca4c3d as detached HEAD (ref is master)...
17 Removing target/
18 Skipping Git submodules setup
19 Restoring cache
20 Checking cache for default-protected...
21 No URL provided, cache will not be downloaded from shared cache server. Instead a local version of cache will be extracted.
22 Successfully extracted cache
23 Executing "staging_script" stage of the job script
24 Using effective pull policy of [always] for container maven:latest
25 Using docker image sha256:dead458da47d2d6db8b65bc7e7c876944f04e70878acf8d2747bd2deeb86ce83 for maven:latest with digest maven@sha256:3d35095b456f61758a4b3f62c746bffd369d5cabef6c04e4387e76dc6640972 ...
26 $ echo "Deploy review app"
27 Deploy review app
28 Saving cache for successful job
29 Creating cache default-protected...
30 target/: found 31 matching artifact files and directories
31 Archive is up to date!
32 Created cache
33 Uploading artifacts for successful job
34 Uploading artifacts...
35 target/:jar: found 1 matching artifact files and directories
36 uploading artifacts as 'archive' to coordinator... 201 Created correlation_id=01N08Z227Y6Pw425F8E3001HTF id=25 responseStatus=201 Created token=64_HsdPz
37 Cleaning up project directory and file based variables
38 Job succeeded
```