**Cyberbit DevOps Exercise**

התרגיל משלב ידע בפיתוח ואינטגרציה בטכנולוגיות הבאות:

1. Git
2. Docker
3. Jenkins
4. Jfrog Artifactory
5. Python

לטובת ביצוע התרגיל, יש להתקין סביבה הכוללת:

1. Docker Host
2. Jenkins
3. Jfrog Artifactory OSS

דגשים לביצוע התרגיל:

* יש להשתמש בDocker Images להתקנת מערכות.
* את קבצי הDocker, Python, Jenkinsfile יש לשים בשרת SourceControl (GIT) כלשהו ולהביא אותו כחלק מהpipeline
* ביצירת Jenkins pipeline, יש להשתמש כמה שיותר בפלגינים של Jenkins, לדוגמא Docker,Artifactory,etc ופחות בcommand line
* בסיום התרגיל, יש לשלוח את התוצרים הבאים במייל חוזר:

1. Dockerfile
2. Jenkinsfile\_scripted
3. Jenkinsfile\_declarative
4. zip\_job.py
5. תיאור קצר בנקודות עיקריות של השלבים בהם בחרת לביצוע של כל תת משימה בתרגיל.
6. צילומי מסך של Jenkins jobs ו Artifactory.
7. Create Dockerfile for Docker image build as follows:

a.      Based on Ubuntu latest image

b.      Define environment variable VERSION=1.2.0

c.      Install python

d.      Install vim

e.      Install zip

f.        Install unzip

g.      Copy zip\_job.py into the image's /tmp folder

h.      Once docker container is up, run a command which will print OS type and architecture + verify /tmp/[zip\_job.py](http://zip_job.pl/) exists

2.       Create [zip\_job.py](http://zip_job.pl/) python script as follows:

a.       Create an array of a,b,c,d,e,f….z

b.       Based on this array create txt files (a.txt,b.txt….)

c.       Make sure all txt files created and if not - fail the script

d.       Create zip files with names based on array + VERSION environment variable, that each one will have one txt file inside (a\_1.2.0.zip should include a.txt, b\_1.2.0.zip should include b.txt and so on)

e.       Make sure all zip files created and if not - fail the script

f. all logs should be logged into console (stdout)

3.      Create 2 Jenkinsfiles pipeline jobs with the same logic, one Declarative and one Scripted:

1. Agent should be based on the Dockerfile you created in step 1

        i.            it should run in a privileged mode with label zip-job-docker

b.       Build stage should execute the [zip\_job.py](http://zip_job.pl/) you've created in step 2

c.       Publish stage should upload all the zip files created (only in case build stage succeeded) to Artifactory you installed using the following properties:

      i.         Artifactory server "https://artifactory-xx" (dns name defined by you)

    ii.        Artifactory user "superman"

    iii.        Artifactory password "P@ssw0rd123$"

    iv.       Artifactory repository to upload to "store-artifacts/{VERSION env variable}"

d.       Report stage - send email with job status in the subject to requestor address

e.       Cleanup stage - delete the workspace

1. Feel free to add some more stages that can be more efficient\intresting
2. Let us know if you have any additional questions.

Create Dockerfile for Docker image build as follows:

a. Based on Ubuntu latest image

b. Define environment variable VERSION=1.2.0

c. Install python

d. Install vim

e. Install zip

f. Install unzip

g. Copy zip\_job.py into the image's /tmp folder

h. Once docker container is up, run a command which will print OS type and architecture + verify /tmp/zip\_job.py exists

Here is an example Dockerfile that meets the requirements you've specified:

Copy code

FROM ubuntu:latest ENV VERSION=1.2.0 RUN apt-get update && apt-get install -y python vim zip unzip COPY zip\_job.py /tmp/ CMD ["/bin/bash", "-c", "echo $(uname -o) $(uname -m) && ls /tmp/zip\_job.py"]

* FROM ubuntu:latest specifies that the base image for the container is the latest version of Ubuntu.
* ENV VERSION=1.2.0 sets the environment variable "VERSION" to "1.2.0".
* RUN apt-get update && apt-get install -y python vim zip unzip runs the command to install python, vim, zip and unzip
* COPY zip\_job.py /tmp/ copies the "zip\_job.py" file from the host machine to the "/tmp" directory within the container.
* CMD ["/bin/bash", "-c", "echo $(uname -o) $(uname -m) && ls /tmp/zip\_job.py"] specifies that the container should run the command echo $(uname -o) $(uname -m) && ls /tmp/zip\_job.py when the container is started, which will print the OS type and architecture and also list the zip\_job.py if it exists.

Create zip\_job.py python script as follows:

a. Create an array of a,b,c,d,e,f….z

b. Based on this array create txt files (a.txt,b.txt….)

c. Make sure all txt files created and if not - fail the script

d. Create zip files with names based on array + VERSION environment variable, that each one will have one txt file inside (a\_1.2.0.zip should include a.txt, b\_1.2.0.zip should include b.txt and so on)

e. Make sure all zip files created and if not - fail the script

f. all logs should be logged into console (stdout)

Here is an example python script that meets the requirements you've specified:

Copy code

import os import zipfile # create an array of lowercase letters letters = [chr(i) for i in range(ord('a'), ord('z')+1)] # create txt files for each letter for letter in letters: with open(f"{letter}.txt", "w") as f: f.write(letter) # check if all txt files were created for letter in letters: if not os.path.exists(f"{letter}.txt"): print(f"Error: {letter}.txt not found.") exit(1) # get the VERSION environment variable version = os.environ.get("VERSION", "1.0.0") # create zip files for each letter for letter in letters: with zipfile.ZipFile(f"{letter}\_{version}.zip", mode='w') as z: z.write(f"{letter}.txt") # check if all zip files were created for letter in letters: if not os.path.exists(f"{letter}\_{version}.zip"): print(f"Error: {letter}\_{version}.zip not found.") exit(1) print("All txt and zip files were created successfully.")

* letters = [chr(i) for i in range(ord('a'), ord('z')+1)] it creates an array of lowercase letters from 'a' to 'z'
* with open(f"{letter}.txt", "w") as f: it opens each txt file with name letter.txt in write mode and write letter in it
* if not os.path.exists(f"{letter}.txt"): it checks if all txt files are created or not
* version = os.environ.get("VERSION", "1.0.0") it fetches the VERSION environment variable
* with zipfile.ZipFile(f"{letter}\_{version}.zip", mode='w') as z: it creates zip files with names letter\_version.zip and add txt files inside it
* if not os.path.exists(f"{letter}\_{version}.zip"): it checks if all zip files are created or not
* print("All txt and zip files were created successfully.") it prints the success message after successful completion of script

Please note that the script assumes that the VERSION environment variable is already set before the script is run.

Continuing with preview tasks, Create 2 Jenkinsfiles pipeline jobs with the same logic, one Declarative and one Scripted:

Agent should be based on the Dockerfile you created in step 1

i. it should run in a privileged mode with label zip-job-docker

b. Build stage should execute the zip\_job.py you've created in step 2

c. Publish stage should upload all the zip files created (only in case build stage succeeded) to Artifactory you installed using the following properties:

i. Artifactory server "https://artifactory-xx" (dns name defined by you)

ii. Artifactory user "superman"

iii. Artifactory password "P@ssw0rd123$"

iv. Artifactory repository to upload to "store-artifacts/{VERSION env variable}"

d. Report stage - send email with job status in the subject to requestor address

e. Cleanup stage - delete the workspace

Here is an example of a Declarative Jenkinsfile that meets the requirements you've specified:

Copy code

pipeline { agent { label 'zip-job-docker' dockerfile { filename 'Dockerfile' args '-u root:root' } } stages { stage('Build') { steps { sh 'python zip\_job.py' } } stage('Publish') { when { success } steps { withArtifacts([includePatterns: '\*.zip']) { artifactoryPublish ( serverId: 'artifactory-xx', repoKey: "store-artifacts/${env.VERSION}", user: 'superman', password: 'P@ssw0rd123$', useSpecs: true, specs: [ [file: 'artifactory.json'] ] ) } } } stage('Report') { when { success } steps { emailext ( subject: "Job ${env.JOB\_NAME} (${env.BUILD\_NUMBER}) status: SUCCESS", to: "requestor@email.com", ) } } stage('Cleanup') { steps { deleteDir() } } } }

* agent {label 'zip-job-docker'} specifies that the agent should use the Docker image built from the Dockerfile labeled 'zip-job-docker'
* sh 'python zip\_job.py' runs the python script to create txt and zip files
* when {success} block will run only when build stage is successful
* withArtifacts([includePatterns: '\*.zip']) it specifies the zip files need to be uploaded
* artifactoryPublish (serverId: 'artifactory-xx', repoKey: "store-artifacts/${env.VERSION}",user: 'superman',password: 'P@ssw0rd123$',useSpecs: true) it uploads the files to Artifactory server with the specified repository
* emailext (subject: "Job ${env.JOB\_NAME} (${env.BUILD\_NUMBER}) status: SUCCESS",to: "requestor@email.com",) sends an email with the job status
* deleteDir() deletes the workspace

And here is an example of a Scripted Jenkinsfile that meets the requirements you've specified:

Copy code

node { label 'zip-job-docker' stage('Build') { sh 'python zip\_job.py' } stage('Publish') { if (currentBuild.currentResult == 'SUCCESS') { withArtifacts([includePatterns: '\*.zip']) { def server = Artifactory.server 'artifactory-xx' server.upload spec: "${workspace}/artifactory.json", target: "store-artifacts/${env.VERSION}" server.publishBuildInfo buildName: env.JOB\_NAME, buildNumber: env.BUILD\_NUMBER }

[ChatGPT Jan 9 Version](https://help.openai.com/en/articles/6825453-chatgpt-release-notes). Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.