**Problem Statement 1:**

Install Jenkins on the Ubuntu machine and create a pipeline job with the following functionality. You have access to an Ubuntu terminal and will execute all tasks within the working directory: */home/ubuntu/code.*

**Tasks**

1. Jenkins Installation and Setup:
   * You have been provided with a script file, *jenkins\_setup.sh*, to install and configure Jenkins along with the required plugins. Execute this script to initiate the installation.
   * After the installation is complete, the **Jenkins admin password**will be stored in the file *jenkins\_admin\_password* within your current working directory (*/home/ubuntu/code*).
   * In case you encounter a GPG Key error during installation, refer to the *README.md* file for troubleshooting steps.
2. You are provided with a *pipeline-job.xml* file. Implement a pipeline with the following stages:
   * **Build**: Print "Building the application...".
   * **Test**: Retry the stage up to 3 times if it fails. Print "Testing the application..." before attempting to run the test. Introduce a simulated failure that triggers retries. Error message: 'Simulated test failure' should be printed before retrying.
   * **Deploy**: Print "Deploying the application..." only if the pipeline reaches this stage.
3. Post-Build Actions:
   * Always print "Pipeline Execution Completed." at the end.
   * If the pipeline succeeds, print "Pipeline Successful."
   * If the pipeline fails, print "Pipeline Failed."
4. Create and build the pipeline job. After that is done, send the console output to*pipeline-output.log.*

**Problem Statement 2:**

Install Jenkins on the Ubuntu machine and complete the pipeline job with the following functionality. You have access to an Ubuntu terminal and will execute all tasks within the working directory: /home/ubuntu/code.

**Tasks**

1. Jenkins Installation and Setup:
   * You have been provided with a script file, *jenkins\_setup.sh*, to install and configure Jenkins along with the required plugins. Execute this script to initiate the installation.
   * After the installation is complete, the **Jenkins admin password**will be stored in the file *jenkins\_admin\_password* within your current working directory (*/home/ubuntu/code*).
   * In case you encounter a GPG Key error during installation, refer to the *README.md* file for troubleshooting steps.
2. You are provided with a *logs/pipeline-job.xml* file. Complete the given steps in it.
   * Add a stage named "Collect Logs":
     + Create a directory named /home/ubuntu/code/logs/processing/
     + Copy all .log files from /home/ubuntu/code/logs/source/ to /home/ubuntu/code/logs/processing/ using the cp command.
     + Print a message: *"Logs collected to /home/ubuntu/code/logs/processing/."*
   * Add a stage named Process Logs:
     + Create a directory named /home/ubuntu/code/logs/processed/ using the mkdir -p command.
     + Use the grep command with the -E option to search for lines containing ERROR or WARN in the log files from /home/ubuntu/code/logs/processing/.
     + Save the filtered output to a file named filtered.log in /home/ubuntu/code/logs/processed/.
     + Print a message: *"Logs processed to /home/ubuntu/code/logs/processed/filtered.log."*
   * Add a stage named Analyze Logs:
     + Create a directory named /home/ubuntu/code/logs/reports/ using the mkdir -p command.
     + Count the number of occurrences of the word ERROR in filtered.log using the grep and wc -l commands. Save the count to a file named error\_count.txt in /home/ubuntu/code/logs/reports/.
     + Count the number of occurrences of the word WARN in filtered.log using the grep and wc -l commands. Save the count to a file named warn\_count.txt in /home/ubuntu/code/logs/reports/.
     + Print a message: *"Log analysis completed. Reports generated in /home/ubuntu/code/logs/reports/."*
   * Add a stage named Notify:
     + Print a message: *"Log processing pipeline completed successfully."*
3. Build the jenkins job and send console output to*pipeline-output.log*

**Testing instructions**

1. Redirect the output to a file using the syntax:*your\_command > test.log* to log the console output.
2. After building the job, wait for 10 seconds before redirecting the output to a file.
3. On clicking **Submit**, if the Jenkins installation fails, the first test case will show **JenkinsInstallationFailure**. Otherwise, it will be skipped, and statuses for test cases used in the actual evaluation will be displayed.

**Problem Statement 3:**

You are tasked with simulating a real-world DevOps scenario to deploy and configure web servers. Using docker containers and ansible automation, your goal is to set up web servers with nginx on two operating systems: ubuntu and alpine linux. Ensure that nginx is installed, running, and accessible from specific ports on the host machine.

**Tasks**

* Install python, pip, and ansible on the Host:
  + Ensure *python3* and *pip* are installed on the host machine.
  + Install *ansible* using *pip*.
  + Add the */.local/bin* directory to the system's PATH to ensure ansible is accessible.
* Pull Docker Images:
  + Pull the *ubuntu:20.04* and *alpine:3.20.3* images from Docker Hub.
* Create Docker Containers:
  + Create a container using the *ubuntu:20.04* image, named *ubuntu-container*, with port *8080* on the host forwarded to port *80* in the container.
  + Create a container using the *alpine:3.20.3* image, named *alpine-container*, with port *8081* on the host forwarded to port *80* in the container.
* Install Python in Containers:
  + For the *ubuntu-container*, install *python3* and *pip* using *apt.*
  + For the *alpine-container*, install *python3* and *pip* using *apk.*
* Configure Ansible Inventory:
  + Create an ansible inventory file named *inventory.ini* with the following format:

[all]

<container\_name\_1> ansible\_connection=docker ansible\_python\_interpreter=<path\_to\_python>

<container\_name\_2> ansible\_connection=docker ansible\_python\_interpreter=<path\_to\_python>

* Automate Nginx Deployment and Configuration with Ansible:
  + Create an ansible playbook named *deploy\_nginx.yml* that:
    - Installs nginx on both containers using the respective package managers:
      * *apt* for *ubuntu-container*.
      * *apk* for *alpine-container.*
    - Starts Nginx in both containers using the respective commands:
      * *service nginx start* for *ubuntu-container.*
      * *nginx* for *alpine-container.*
* Run the Ansible Playbook:
  + Use the created ansible inventory file and playbook to automate the deployment and configuration of nginx on both containers.
* Set Up Port Forwarding:
  + Configure port forwarding rules to make nginx accessible on the host:
    - *localhost:8080* for the *ubuntu-container.*
    - *localhost:8081* for the *alpine-container*.

**Problem Statement 4:**

You are tasked with using bash commands directly in the terminal to monitor system uptime and CPU load averages. The solution should fetch and display these metrics and conditionally provide warnings if the CPU load exceeds a certain threshold. Additionally, the solution must log the uptime and load averages as separate entries in a log file.

**Tasks**

1. Fetch and Display System Uptime:
   * Use a bash command to retrieve and display the system uptime in a human-readable format (e.g., 2 days, 5 hours, 30 minutes).
   * Log this uptime as a separate entry in the log file, system\_status.log, with a timestamp.
2. Fetch and Display CPU Load Averages:
   * Use a bash command to retrieve the CPU load averages for the last 1 minute, 5 minutes, and 15 minutes.
   * Log this as a separate entry in the log file, system\_status.log, with a timestamp, format and display them as:  
     Load Averages: 0.23 (1 min), 0.45 (5 min), 0.67 (15 min)
3. Check High CPU Load:
   * Compare the 1-minute CPU load average with a threshold of **1.0**.
   * If the load exceeds the threshold, display:  
     Warning: High CPU load detected!
   * If the load is below the threshold, display and log:  
     System is running within safe CPU load limits.
4. Log Data:
   * Ensure that all logs are appended to the file system\_status.log in the format:

                        YYYY-MM-DD HH:MM:SS | <Log Message>

**Problem Statement 5:**

As a developer, you need to manage a monorepo containing multiple services and a shared library. This exercise tests your skills in making updates to specific services, managing file and directory states, modifying the shared library, and tagging versions. You will be working in a pre-configured Git repository located at */home/ubuntu/code/monorepo*.

**Tasks**

* **Tag the Initial Commit**
  + Tag the initial commit of the repository with a version identifier (v1.0.0).
* **Update Service A**
  + Add a configuration file *(config.json)* to Service A, reflecting a version update. Commit the changes with the message "*Add config.json to service-a*".
* **Remove and Restore Service B**
  + Remove the *service-b/* directory from the repository (without deleting the files locally).
  + Commit the removal with the message "*Remove service-b from repository*".
  + Reintroduce the *service-b/* directory to the repository.
  + Commit the restoration with the message "*Reintroduce service-b*".
* **Update the Shared Library**
  + Add a *.gitignore* file in the *common-lib* directory to exclude .tmp files from being tracked by Git.
  + Commit the *.gitignore* and README file with the message "*Update common-lib with README and .gitignore*".
* **Tag the Final Update**
  + Tag the repository's final state with a new version identifier representing the monorepo's updated state (v1.0.1).

|  |
| --- |
| **LeetCode** |
| #192, #193 |
| #175, #176 |
| #43, #657 |
| #359, #362 |
| #194, #195 |
| #10, #68 |
| #196, #197 |
| #20, #22 |
| #125, #290 |
| #1789, #1795 |