AWS Assignment 4

Below is an assignment that combines Jenkins, Docker, GitHub, and shell scripting. This assignment is designed to help freshers understand how to create a CI/CD pipeline that builds and deploys a containerized application.

Assignment: CI/CD Pipeline with Jenkins, Docker, GitHub, and Shell Scripting

Objective

Create an automated CI/CD pipeline that:

- Hosts a simple web application in a GitHub repository.
- Uses a Dockerfile to containerize the application.
- Leverages Jenkins (running on an EC2 instance or your local machine) to trigger builds when changes are pushed to GitHub.
- Uses Jenkins to build the Docker image, run a container, and execute shell scripts that verify the deployment.

Assignment Tasks

1. GitHub Repository Setup

- **Create a Repository:**
  - Create a new GitHub repository (e.g., `simple-web-app`).
- **Develop a Simple Web Application:**
  - Create a minimal web application (e.g., using Node.js, Python Flask, or a static HTML page served by Nginx).
- **Dockerfile:**
  - Write a Dockerfile that:
    - Uses an appropriate base image.
    - Copies the web application code.
    - Installs necessary dependencies.
    - Exposes the appropriate port (e.g., 80 or 8080).
    - Specifies the command to run the application.

- **Push Code:**
    - Ensure the repository contains all the necessary code files, the Dockerfile, and a README with instructions.

---

2. Jenkins Setup

- **Jenkins Installation:**
    - Install Jenkins on an EC2 instance or your local machine.
- **Configure GitHub Integration:**
    - Set up a GitHub webhook so that Jenkins triggers a build when code is pushed.
- **Create a Pipeline Job:**
    - Configure a Jenkins Pipeline (or freestyle job) that performs the following steps:
        - **Clone Repository:** Pull code from your GitHub repository.
        - **Build Docker Image:** Run `docker build` to create an image from your Dockerfile.
        - **Run Docker Container:** Use `docker run` to start a container from the built image.
        - **Execute Verification Script:** Run a shell script (see Task 3) to verify that the container is running and serving the web application.

---

3. Docker & Shell Scripting

- **Docker Integration in Jenkins Pipeline:**
    - Ensure the Jenkins environment has Docker installed and the Jenkins user has permission to run Docker commands.
- **Create a Shell Script (e.g., `verify.sh`):**
    - **Purpose:** Verify that the Docker container is up and the web application is responding.
    - **Tasks:**
        - Check if the Docker container is running (e.g., using `docker ps`).
        - Use `curl` to fetch the web page served by the container.
        - Print a success or error message based on the HTTP response.
- **Integrate Shell Script:**
    - Include this shell script as a post-build step in your Jenkins Pipeline to confirm the deployment was successful.

---

4. (Optional Bonus) Docker Hub Integration

- **Push Docker Image:**

- Extend your pipeline to tag the Docker image and push it to Docker Hub.
- Ensure that your Docker Hub credentials are securely stored in Jenkins (e.g., using credentials plugins).

Submission Requirements

1. **Documentation:**
   - A README file in your GitHub repository that explains:
     - The web application functionality.
     - How to build and run the Docker container locally.
     - Instructions on how the Jenkins pipeline is configured.
   - A simple architecture diagram showing:
     - GitHub → Jenkins Pipeline → Docker (Build & Run) → Shell Script Verification.
2. **Screenshots:**
   - GitHub repository structure.
   - Jenkins pipeline configuration and build logs.
   - Output of the shell script (e.g., successful curl response and container status).
   - (Optional) Docker Hub repository view after pushing the image.
3. **Source Code:**
   - Provide links to your GitHub repository containing:
     - Web application code.
     - Dockerfile.
     - Shell script (`verify.sh`).
     - Jenkins Pipeline script (Jenkinsfile) or configuration details.
4. **Deployment Instructions:**
   - A step-by-step guide on how to set up the Jenkins job, configure the GitHub webhook, and run the pipeline.

Evaluation Criteria

- **Integration:** Proper integration of GitHub, Jenkins, Docker, and shell scripting.
- **Functionality:** The pipeline should successfully clone the repository, build the Docker image, run the container, and execute the verification script.
- **Clarity & Documentation:** Clear instructions, comments, and diagrams that explain the process.
- **Automation:** Effective use of Jenkins to automate the build and deployment process.
- **Bonus Enhancements:** If Docker Hub integration is implemented, proper handling of image tagging and pushing.

This assignment will help you gain practical experience in setting up CI/CD pipelines with modern DevOps tools. Happy building!

AWS Assignment 5

Below is a **small assignment** that combines Terraform, Linux, and Bash scripting. This task will have you provision infrastructure with Terraform, configure a Linux EC2 instance via a Bash user-data script, and then use an additional Bash script to verify and interact with the deployed instance.

Assignment: Automated Web Server Deployment with Terraform & Bash Scripting

Objective

Provision an AWS EC2 instance running Linux using Terraform. Automatically configure the instance to install and run a web server (Apache or Nginx) via a Bash user-data script, and then create a separate Bash script to SSH into the instance, verify system information, and check the web server's status.

Assignment Tasks

1. Terraform – Provision an EC2 Instance

- **Create Terraform Configuration Files:**
  - **main.tf:**
    - Configure the AWS provider.
    - Define an AWS EC2 instance resource with a suitable Linux AMI.
    - Create a security group that allows:
      - Inbound SSH (port 22)
      - Inbound HTTP (port 80)
  - **variables.tf:**
    - Define variables for instance type, region, key pair name, etc.
  - **outputs.tf:**
    - Output the public IP address of the EC2 instance.
- **Use a Bash User-Data Script:**
  - Embed a Bash script in the EC2 resource's `user_data` that:
    - Updates the system package index.
    - Installs a web server (choose either Apache or Nginx).

        - Creates or modifies the default index.html to display a custom welcome message.
        - Starts and enables the web server service.

---

2. Bash Scripting – Verification Script

- **Create a Bash Script (e.g., `verify.sh`):**
    - Use SSH (with the proper key) to connect to the newly provisioned EC2 instance.
    - Run commands on the instance to:
        - Display system information (e.g., hostname, uptime).
        - Check that the web server process is running.
        - Optionally, fetch the contents of the index.html (using `curl` or similar) to verify the welcome message.
- **Make sure the script is executable** and document any prerequisites (such as SSH key configuration).

---

3. Testing and Documentation

- **Deploy the Infrastructure:**
    - Initialize Terraform.
    - Run `terraform plan` to review changes.
    - Execute `terraform apply` to provision the EC2 instance.
- **Run the Verification Script:**
    - Once the EC2 instance is up, run your `verify.sh` script to check the system information and web server status.
- **Documentation & Diagram:**
    - Write a brief README that explains:
        - How to set up and run the Terraform configuration.
        - How the Bash user-data script configures the web server.
        - How to execute the verification Bash script.
    - Optionally, include a simple diagram showing:
        - Terraform provisioning → EC2 instance with user-data configuration → SSH verification.

Submission Requirements

1. **Terraform Code:**

    - Include all Terraform configuration files (`main.tf`, `variables.tf`, `outputs.tf`, etc.).
2. **Bash Scripts:**
    - Provide the user-data script (embedded in Terraform) and the separate `verify.sh` script.
3. **Documentation:**
    - A README file with deployment instructions and an explanation of your setup.
    - A diagram (can be a simple image or ASCII diagram) showing the architecture flow.
4. **Screenshots/Outputs:**
    - Screenshot of a successful `terraform apply` output.
    - Terminal output from running `verify.sh` showing system details and web server status.

Evaluation Criteria

- **Correct Terraform Configuration:**
    - Provisioning of an EC2 instance with the required security group and outputs.
- **Effective Bash Scripting:**
    - Proper installation and configuration of the web server via the user-data script.
    - A working verification script that successfully connects via SSH and checks the instance.
- **Documentation & Clarity:**
    - Clear instructions and a logical explanation of each component.
    - Inclusion of a diagram to illustrate the workflow.
- **End-to-End Functionality:**
    - The deployed instance should be accessible (via its public IP) with the custom web page, and the verification script should confirm the server's status.

This assignment will help you gain practical experience with infrastructure as code using Terraform, automate instance configuration with Bash scripting, and interact with Linux systems over SSH. Happy building!