Instructions: There are 3 assignments you must complete. More weight is given on having completed each assignment over extra credit. In other words, if you find yourself bound for time, make sure you complete each assignment before moving on to extra credit.

Take-Home Assignment 1: Terraform and AWS
Task: You are tasked with using Terraform to provision and configure an Nginx server in AWS. Your goal is to automate the infrastructure setup process and ensure that the server is properly configured. Complete the following steps:

1. Create a Terraform configuration to provision an EC2 instance running Ubuntu 20.04 in AWS.
    - Use an appropriate AWS provider configuration.
    - Define the necessary resources, including the EC2 instance, security groups, and any other required components.
2. Configure the EC2 instance:
    - Install Nginx on the EC2 instance using a provisioner or a user_data script.
3. Secure the server:
    - Create security groups to allow incoming HTTP (port 80) traffic to the EC2 instance.
    - Implement any additional security measures that you deem necessary.
4. Document the configuration:
    - Provide a README.md file that explains how to deploy and test the infrastructure using Terraform.
    - Include any necessary configuration variables or files.
    - Document any assumptions or considerations you made during the setup.
    - Provide instructions on how to access the Nginx server once it is deployed.

Extra credit #1:
Implement Terraform state management. You don't have to write code for this part, just explain why we need it and your logic behind your proposed solution:

- Choose a suitable method for managing Terraform state (e.g., local, remote backend).
- Configure the backend to store the state in a secure and accessible location.
- Document the chosen method and any configuration changes required and include your reason for choosing this option. Discuss any pros, cons, and tradeoffs.

Take-Home Assignment 2: Linux, Ansible, and FluentD
Task: You are given a virtual machine running Ubuntu 20.04. Your task is to automate the installation and configuration of a web server using Ansible. You should write an Ansible playbook that accomplishes the following:

1. Ensures that Nginx is installed
2. Configures Nginx to serve a simple "Hello, World!" HTML page.
3. Started the Nginx service.

4. Secures the server by disabling unnecessary services and configuring a basic firewall using UFW.

Extra credit #1:
1. Install Fluentd on the server using Ansible
2. Configure Fluentd to collect and process logs from the Nginx access and error logs
3. Implement Fluentd filtering to exclude logs from IP addresses contained in the deny list of IP addresses. The file is called denylist.txt.
4. Route the filtered logs to a separate file called "denylist_audit.log"

Extra credit #2: The access log file is growing at a fast rate. We need a solution for managing log size. The team has determined that we only need 5 days worth of log data. Explain or implement a solution.

Take-Home Assignment 3: Docker
Task: You are provided with a Dockerfile that contains several mistakes. Your task is to identify and fix these mistakes to make the Dockerfile functional. Follow the steps below:
1. Dockerfile Assessment:
    - Review the provided Dockerfile for any syntax errors, missing or incorrect instructions, or potential issues.
    - Identify the mistakes in the Dockerfile.
2. Dockerfile Fixes:
    - Modify the Dockerfile to address the identified mistakes.
    - Correct any syntax errors, missing or incorrect instructions, or potential issues.
3. Build and Test the Docker Image:
    - Build the Docker image using the fixed Dockerfile.
    - Run a container based on the built image to ensure that the container functions as expected.
    - Test the containerized application or service to verify its functionality.
4. Documentation:
    - Create a README.md file that explains the mistakes in the original Dockerfile and describes the fixes you implemented.
    - Provide clear instructions on how to build and run the Docker image.
    - Showing your output of the working version would be greatly appreciated

```
FROM ubuntu:latest

# Update the package repository and install PostgreSQL
RUN apt-get update && \
    apt-get install -y postgresql
```

```
# Create a new PostgreSQL user and database
RUN service postgresql start && \
    su - postgres -c "psql -c 'CREATE USER myuser WITH PASSWORD
mypassword;'" && \
    su - postgres -c "createdb -O myuser mydatabase"

# Configure PostgreSQL to allow connections from all IP addresses
RUN sed -i 's/#listen_addresses = 'localhost'/listen_addresses = '*'/'
/etc/postgresql/13/main/postgresql.conf && \
    echo "host    all             all             0.0.0.0/0
trust" >> /etc/postgresql/13/main/pg_hba.conf

# Expose the PostgreSQL default port
EXPOSE 5432

# Start the PostgreSQL service
CMD service postgresql start && tail -f /dev/null
```

Extra Credit #1: Implement a secret encryption solution in order to prevent exposure of the postgres db credentials.

Extra Credit #2: Explain the steps you can take to troubleshoot issues with the running container