DevOps Task 01

1- Terraform Configuration:

Terraform Cloud

- Create an account
- Create an organization called "pwc_project"
- Create workspace called "project_one"
- Generate token to connect with

Terraform configuration:

Terraform configuration sets up an AWS infrastructure that includes the following components:

1. **Terraform Configuration**:
   - Specifies the AWS provider.
   - Uses a remote backend on Terraform Cloud.
2. **AWS Provider**:
   - Configures the AWS provider for the us-east-1 region.
3. **VPC**:
   - Creates a Virtual Private Cloud (VPC) with a specified CIDR block.
4. **Subnet**:
   - Defines a public subnet within the VPC, specifying CIDR block and availability zone.
5. **Internet Gateway**:
   - Creates an internet gateway and attaches it to the VPC for internet access.
6. **Route Table**:
   - Creates a route table with a route to the internet via the internet gateway.
   - Associates the route table with the public subnet.
7. **Security Group**:
   - Defines a security group with rules allowing SSH (port 22) and HTTP (port 80) access from any IP address.
8. **Elastic IP**:
   - Allocates an Elastic IP address for public internet access.
9. **EC2 Instance**:
   - Launches an EC2 instance in the public subnet with the specified AMI, instance type, and security group.
   - Associates the EC2 instance with the Elastic IP.
10. **Elastic IP Association**:
    - Associates the allocated Elastic IP with the launched EC2 instance.

2- SonarQube Deployment with Docker

Docker-compose:

Services

SonarQube

- **Build Context**: The Dockerfile for SonarQube is used to build the service image.
- **Ports**: The service is accessible on port 9000.
- **Environment Variables**: The JDBC URL, username, and password are set to connect to the PostgreSQL database.
- **Volumes**: Data and extensions are persisted using Docker volumes to ensure data is retained across container restarts.

PostgreSQL

- **Image**: The latest PostgreSQL image from Docker Hub is used.
- **Environment Variables**: The database user, password, and database name are set to support SonarQube.
- **Volumes**: PostgreSQL data is persisted using a Docker volume to ensure data is retained across container restarts.

Dockerfile

The Dockerfile for SonarQube is extended to include custom configuration settings. A `sonar.properties` file is copied to the appropriate directory within the SonarQube container.

sonar.properties

The SonarQube web context is set to `/sonar` to customize the application context path.

Volumes

Three Docker volumes are defined to persist data:

- `sonarqube_data`: Stores SonarQube data.
- `sonarqube_extensions`: Stores SonarQube extensions.
- `postgres_data`: Stores PostgreSQL data.

3- Ansible Playbook

Variables:

- **ports_conf_src:** Path to the ports.conf file used for Apache configuration.
- **reverse_proxy_conf_src:** Path to the reverse-proxy.conf file used for Nginx reverse proxy configuration.

Tasks:

1. **Update apt cache:**
   - Ensures the package manager cache is up to date (apt: update_cache: yes).
2. **Install Apache:**
   - Installs the Apache web server (apt: name: apache2 state: present).
3. **Configure Apache:**
   - Copies ports.conf to /etc/apache2/ports.conf, ensuring Apache listens on specified ports.
4. **Restart Apache:**
   - Restarts Apache to apply configuration changes (service: name: apache2 state: restarted).
5. **Install Nginx:**
   - Installs Nginx web server for reverse proxy functionality (apt: name: nginx state: present).
6. **Configure Nginx Reverse Proxy:**
   - Copies reverse-proxy.conf to /etc/nginx/sites-available/reverse-proxy.conf and creates a symbolic link to enable the configuration (copy, file tasks).
7. **Restart Nginx:**
   - Restarts Nginx to activate the reverse proxy configuration (service: name: nginx state: restarted).
8. **Install Docker Dependencies:**
   - Installs necessary dependencies for Docker (apt: name: ... state: present).
9. **Add Docker GPG Key and Repository:**
   - Adds Docker's GPG key and repository to the system (apt_key, apt_repository tasks).
10. **Install Docker:**
    - Installs Docker CE (apt: name: docker-ce state: present).
11. **Upgrade Docker CE:**
    - Ensures Docker CE is upgraded to the latest version (apt: name: docker-ce state: latest).
12. **Ensure Docker Service is Running:**
    - Starts Docker service and enables it to start on boot (service: name: docker state: started enabled: yes).

13. **Download and Install Docker Compose:**
    - Downloads Docker Compose binary and installs it to /usr/local/bin/docker-compose (shell, file tasks).
14. **Create Symlink for Docker Compose:**
    - Creates a symlink for docker-compose in /usr/bin for easier access (file: src: ... dest: ... link tasks).
15. **Copy Docker Configuration Files:**
    - Copies necessary Docker configuration files (docker-compose.yaml, Dockerfile, sonar.properties) to the remote host (ansible.builtin.copy).
16. **Run Docker Compose:**
    - Executes docker-compose up -d to deploy and start Docker containers defined in docker-compose.yaml.

Configuration Files

`ports.conf`

- Configures Apache to listen on port 8000.

`reverse-proxy.conf`

- Configures Nginx to act as a reverse proxy, directing traffic for specific paths to different back-end services:
    - `/corstat/` is proxied to `http://localhost:8000/`.
    - `/sonar` is proxied to `http://localhost:9000/sonar`.

4- GitHub Actions Workflow to automate the whole process

Workflow Trigger

The workflow is triggered by a push event to the main branch. This ensures that any changes pushed to the main branch will automatically initiate the deployment process.

Workflow Steps:

1. **Checkout Code:**
    - Retrieves the latest code from the repository using GitHub Actions' checkout action.
2. **Setup Terraform:**
    - Configures Terraform with the provided API token (TF_API_TOKEN) stored in GitHub Secrets for authentication.

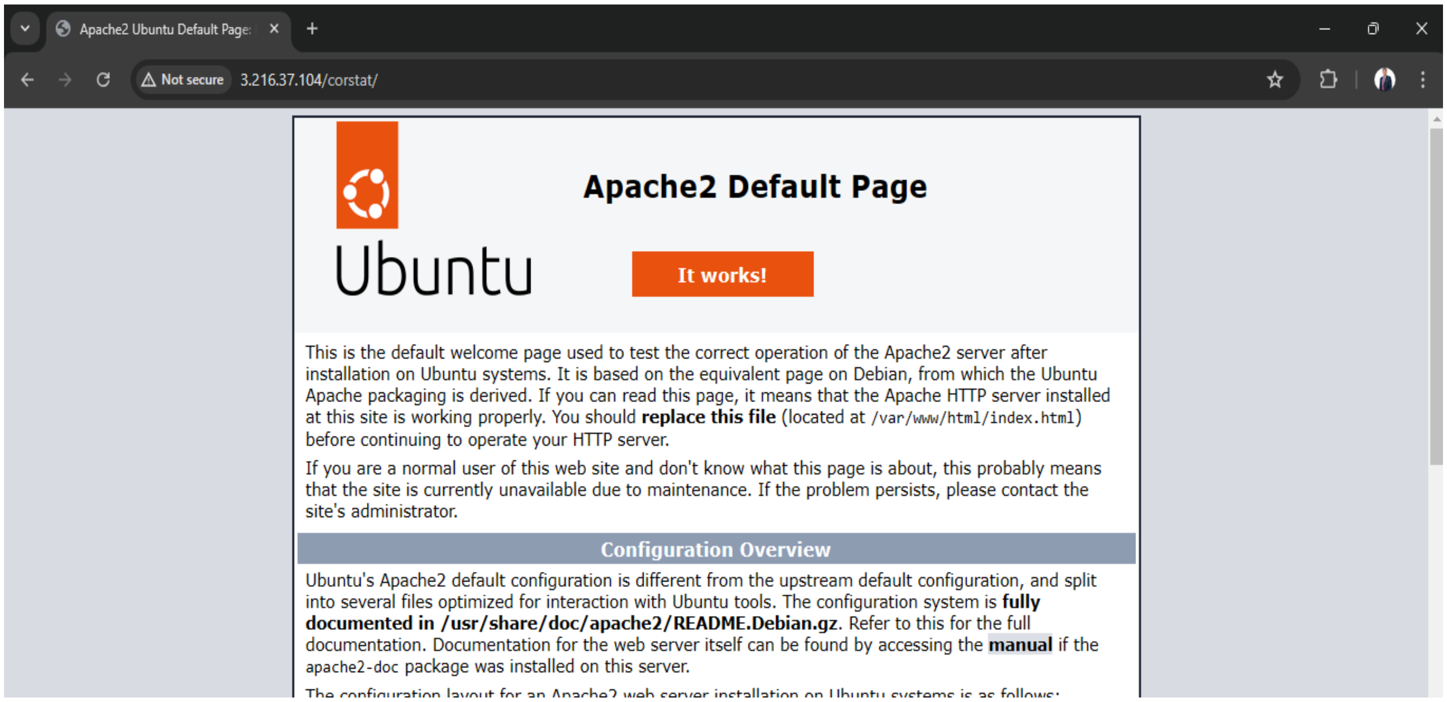
3. **Terraform Init:**
   - Initializes Terraform in the working directory (terraform init), preparing it for deployment.
4. **Terraform Apply:**
   - Applies the Terraform configuration (terraform apply -auto-approve), automatically deploying the defined infrastructure on AWS.
5. **Create Inventory:**
   - Extracts the Elastic IP address from Terraform output and stores it in an Ansible inventory file (ansible/inventory).
6. **Show Inventory:**
   - Displays the contents of the Ansible inventory file to verify the generated configuration.
7. **Set up SSH key:**
   - Sets up SSH authentication by creating an SSH private key (id_rsa) from the PRIVATE_KEY stored in GitHub Secrets. This key is used for secure communication with deployed instances.
8. **Install Ansible:**
   - Installs Ansible on the GitHub Actions runner machine to enable configuration management
9. **Run Ansible Playbook:**
   - Changes directory to ansible/ and executes the Ansible playbook (ansible-playbook playbook.yaml) against the infrastructure defined in the inventory file. The playbook likely contains tasks to configure applications, services, or perform additional setup steps post-deployment.

Secrets Management

The workflow leverages GitHub Secrets to securely manage sensitive information such as **AWS credentials, Terraform API token, and SSH private key**. These secrets are referenced in the workflow to authenticate and access required resources without exposing sensitive data.

5- Final Result:

http://3.216.37.104/corstat/



Apache2 Default Page

It works!

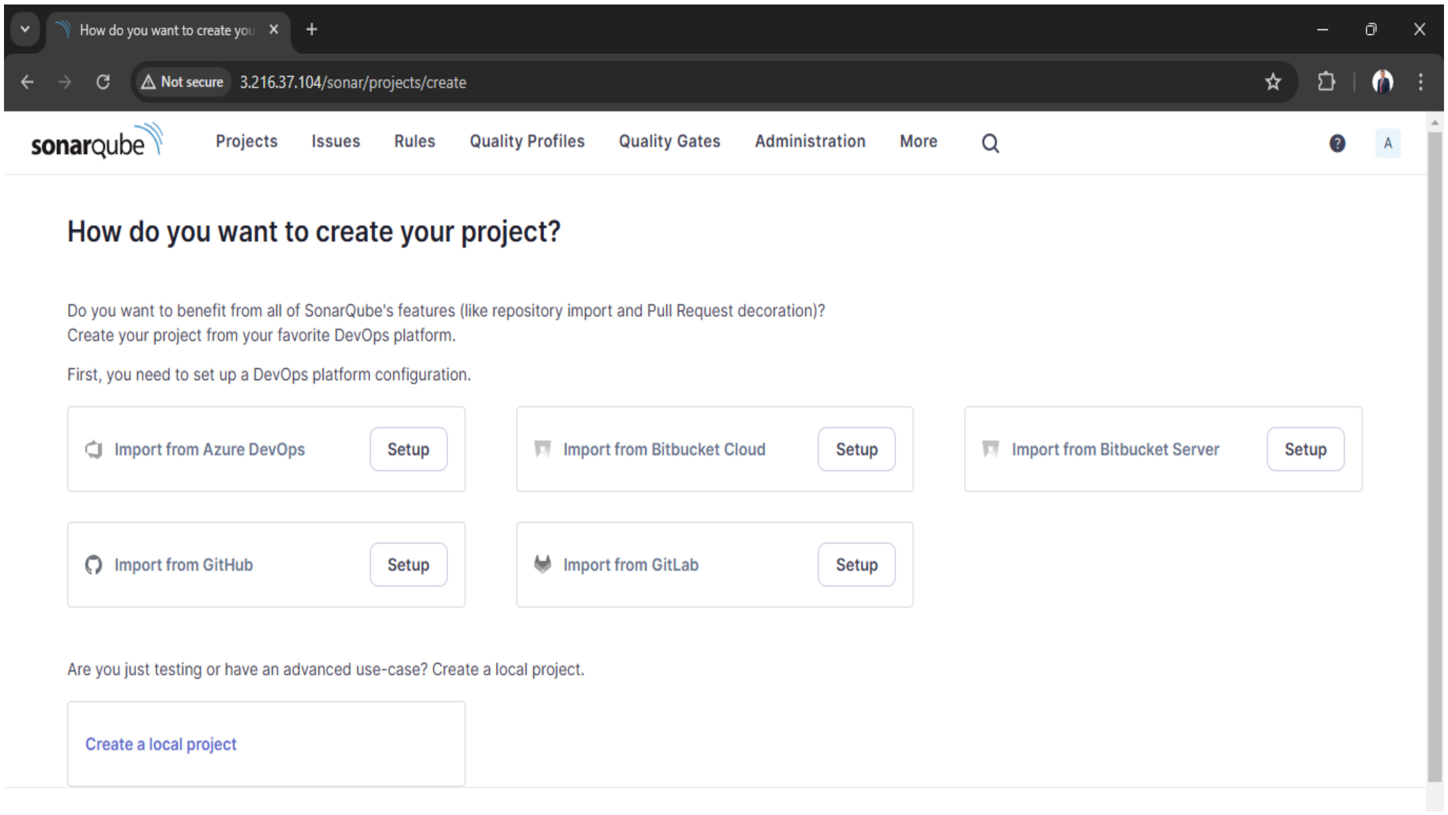
This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

http://3.216.37.104/sonar/



How do you want to create your project?

Do you want to benefit from all of SonarQube's features (like repository import and Pull Request decoration)? Create your project from your favorite DevOps platform.

First, you need to set up a DevOps platform configuration.

Import from Azure DevOps — Setup

Import from Bitbucket Cloud — Setup

Import from Bitbucket Server — Setup

Import from GitHub — Setup

Import from GitLab — Setup

Are you just testing or have an advanced use-case? Create a local project.

Create a local project