**Orchestration Challenge Documentation**

**Overview**

**Make sure you have installed all the dependencies before running the script file. Some of them are Ansible, Python 3, Python Libraries: FastAPI, uvicorn, pydantic, Make, curl**

This documentation provides a comprehensive guide to the orchestration challenge, outlining the setup, implementation, and usage of the provided solution. The orchestration challenge involves automating the deployment and management of a Python codebase, including setting up a FastAPI server and testing its functionality.

Solution Components

The solution consists of the following components:

**Ansible Playbook (orchestrate.yml)**: An Ansible playbook responsible for automating the deployment process, including creating a new user, installing Python and required dependencies, copying Python code to the server, and starting the FastAPI server.

**Python Codebase:** The codebase includes two main files: main.py and APIs.py. main.py serves as the entry point for the FastAPI server, while APIs.py contains the FastAPI endpoints and business logic.

**Makefile:** The Makefile provides convenient commands for building, running, and verifying the code. It simplifies executing tasks such as deploying the code and testing the server.

Let's start by explaining the code

**orchestrate.yml**

This Ansible playbook is designed to orchestrate the deployment of a Python codebase with a FastAPI server. Let's break down the tasks and explain their purpose for documentation:

1. Prompt User for Username: This task uses the pause module to prompt the user to enter a username for the new user account. The entered username is stored in the username\_input variable.

A screen shot of a computer

Description automatically generated

2. Create a New User: The user module creates a new user account on the host system using the username provided by the user. It ensures that the user exists and creates a home directory for the user.

A black screen with pink text

Description automatically generated

3. Install Python and pip: The apt module installs Python 3 and pip (Python package manager) on the host system. This step ensures that the required Python environment is set up for running the FastAPI server.

A black screen with green text

Description automatically generated

4. Check if Remote\_tmp Directory Exists: The stat module checks if the. ansible/tmp directory exists in the home directory of the newly created user. Ansible uses this directory for temporary files during playbook execution.

A screen shot of a computer

Description automatically generated

5. Create Remote\_tmp Directory: If the .ansible/tmp directory does not exist, this task creates it using the file module. The directory is set with appropriate permissions (0700) for security purposes.

A computer screen with colorful text

Description automatically generated

6. Install FastAPI and Its Dependencies: The pip module installs FastAPI and its dependencies (uvicorn) within the context of the newly created user. This ensures that the FastAPI server can be executed with the required packages.

A computer screen with text

Description automatically generated

7. Limit Ports to Allow SSH and Python Code API: The ufw module configures the firewall to allow incoming connections on ports 22 (SSH) and 8000 (Python code API port). This step ensures that SSH access and API requests can reach the server.

A computer screen with text on it

Description automatically generated

8. Copy Python Code to Host: The copy module copies the Python code files (APIs.py and main.py) from the Ansible control machine to the home directory of the newly created user on the host system.

A computer screen with text on it

Description automatically generated

9. Run Python Script: This task executes the APIs.py script within the context of the newly created user. It ensures that the Python code is executed with the correct permissions and environment.

A screenshot of a computer program

Description automatically generated

10. Stop FastAPI Server if Running: This task stops the FastAPI server if it's already running on port 8000. It uses the kill command to terminate the process associated with the specified port. The ignore errors: yes, the directive prevents the task from failing if no process is found running on the port.

A computer screen with text

Description automatically generated

11. Start FastAPI Server: Finally, this task starts the FastAPI server using the uvicorn command with the specified host (0.0.0.0) and port (8000). The nohup command ensures that the server continues running even after the SSH session is terminated, and the output is redirected to /dev/null to discard any output. The server is started within the context of the newly created user.

A computer screen with text

Description automatically generated

This playbook automates the deployment process and ensures the FastAPI server is correctly set up and running on the host system. Users can follow these documented steps to efficiently deploy the Python codebase with the FastAPI server.

**Makefile**

A black screen with a black background

Description automatically generated

Explanation:

Command: make build

Description: Executes the Ansible playbook orchestrate.yml to build and deploy the Python environment. It automates the deployment process by setting up the necessary environment, installing dependencies, and configuring the server. You will be prompted to enter the sudo password to run privileged tasks.

Command: make server

Description: This function starts the FastAPI server after the code is built and deployed. It uses uvicorn to run the FastAPI application (main:app) on the specified host (0.0.0.0) and port (8000). The server will begin listening for incoming requests on port 8000.

Command: make verify

Description: Verifies that the server is running correctly by making a GET request to the /hello endpoint of the FastAPI server using curl. It sends a GET request to the server running on localhost at port 8000 and expects a response from the /hello endpoint. If the server functions properly, it should respond with "Hello!".