Tutorial: Setting Up an OpenSSH Server in Docker

Running an OpenSSH server inside a Docker container is a lightweight, secure way to manage SSH access without requiring a full server installation. This approach is highly flexible, allowing you to spin up SSH access only when needed, manage multiple servers from a single host, and configure SSH securely using Docker's environment variables.

In this tutorial, we will walk through a bash script that automates the creation of an OpenSSH server inside a Docker container. By setting environment variables and mapping key files, you can quickly deploy an OpenSSH server with minimal configuration effort.

Why Run OpenSSH in a Docker Container?

Docker offers several advantages for managing services like OpenSSH: - **Portability**: Easily deploy or remove the OpenSSH server without affecting the host system. - **Configuration Flexibility**: Customize the server with environment variables and Docker volumes, making it highly configurable without modifying host files. - **Isolation**: Running SSH in Docker isolates it from the rest of the system, adding a layer of security. - **Restart Capabilities**: Docker containers can be set to automatically restart if they fail or are stopped, ensuring constant availability.

Using Docker, you can deploy multiple OpenSSH servers on the same machine, each with different configurations, isolated from one another.

How the `openssh_docker.sh` Script Works

The script uses Docker's `docker run` command to pull and run an OpenSSH server image from LinuxServer.io's repository. Environment variables are used to define user credentials, SSH keys, and other settings. The script ensures the container runs in detached mode (`-d`), and it maps the necessary files from the host system to the container.

Let's break down the key elements of the script.

Step 1: Environment Variables

Before running the Docker container, the script defines several key environment variables that control how the OpenSSH server behaves:

```
docker run -d \
  --name=openssh-server \
  --hostname=paulgrey-omen \
  -e PUID=1000 \
  -e PGID=1000 \
  -e TZ=Etc/UTC \
  -e PUBLIC_KEY_FILE=/home/paulgrey/.sshk/id_rsa.pub \
  -e PUBLIC_KEY_DIR=/home/paulgrey/.sshk \
  -e SUDO_ACCESS=false \
  -e PASSWORD_ACCESS=false \
  -e USER_PASSWORD=Notnewtothegame89! \
  -e USER_NAME=paulgrey \
  -p 2222:2222 \
  -v /home/paulgrey/.sshc/sshd_config \
  --restart unless-stopped \
  lscr.io/linuxserver/openssh-server:latest
```

Key Environment Variables:

- **`PUID` and `PGID`**: These environment variables set the user and group IDs for the container. `PUID=1000` and `PGID=1000` ensure that the user inside the container has the same IDs as the user on the host system.

- **`TZ`**: Sets the timezone for the container (`Etc/UTC` in this case). Adjust this based on your local timezone.

- **`PUBLIC_KEY_FILE` and `PUBLIC_KEY_DIR`**: These variables point to the host system's SSH public key file and directory. Docker maps these files into the container to allow passwordless SSH authentication using your host's SSH keys.

- **`SUDO_ACCESS`**: Setting this to `false` disables sudo access inside the container for added security.

- **`PASSWORD_ACCESS`**: Disabling password-based authentication (`false`) ensures that only SSH key authentication is allowed, making the server more secure.

- **`USER_PASSWORD` and `USER_NAME`**: These environment variables define the SSH username and password. Note that in this setup, the password is required to create the user but isn't used for authentication (since password-based access is disabled).

By setting these environment variables, the OpenSSH server is securely configured inside the container without having to modify any files on the host system.

Step 2: Port Mapping

The script uses the `-p` flag to map the container's SSH port (2222) to the host's port:

```
-p 2222:2222
```

This exposes the container's SSH port on the host system. When connecting to this OpenSSH server, you will need to specify port `2222`. You can modify this to map to a different host port if necessary.

Step 3: Volume Mounting

The script mounts important configuration files from the host to the container using Docker's `-v` option:

```
-v /home/paulgrey/.sshc/sshd_config
```

- **`/home/paulgrey/.sshc/sshd_config`**: This maps the `sshd_config` file from the host to the container. This file controls the OpenSSH server's configuration, such as which authentication methods are allowed, logging levels, and more.

You can adjust the path of the configuration file based on your setup.

Step 4: Container Restart Policy

To ensure the OpenSSH server is always running, the script uses the `--restart` flag to set the container's restart policy:

```
--restart unless-stopped
```

This policy ensures that the container restarts automatically if it crashes or the system reboots. The container will continue running unless explicitly stopped.

Step 5: Run the Script

The script finishes by running the Docker command and creating the OpenSSH server container:

```
echo "Server created"
```

This final message confirms that the container has been successfully created and is running in detached mode.

Why This Script is Effective

1. **Minimal Setup**: By using Docker, you avoid the need to manually install and configure an OpenSSH server on the host. The entire configuration is handled by environment variables and Docker volumes.

2. **Secure Authentication**: With password authentication disabled and public key authentication enabled, the OpenSSH server is secure by default. Sudo access is also disabled for additional security.

3. **Portable and Isolated**: Running SSH in Docker isolates it from the host system, making it easier to manage and more secure. You can move or replicate the container on different machines without impacting the underlying system configuration.

4. **Automatic Restart**: The container's restart policy ensures that the server remains available even after crashes or reboots.

How to Use the Script

1. **Create the Script**: Copy the following code and save it as `openssh_docker.sh`.

```bash
#!/bin/bash

echo "Creating docker openssh server"

docker run -d \
  --name=openssh-server \
  --hostname=paulgrey-omen \
  -e PUID=1000 \
  -e PGID=1000 \
  -e TZ=Etc/UTC \
  -e PUBLIC_KEY_FILE=/home/paulgrey/.sshk/id_rsa.pub \
  -e PUBLIC_KEY_DIR=/home/paulgrey/.sshk \
  -e SUDO_ACCESS=false \
  -e PASSWORD_ACCESS=false \
  -e USER_PASSWORD=Notnewtothegame89! \
  -e USER_NAME=paulgrey \
  -p 2222:2222 \
  -v /home/paulgrey/.sshc/sshd_config \
  --restart unless-stopped \
  lscr.io/linuxserver/openssh-server:latest

echo "Server created"
```

2. **Make the Script Executable**: Run the following command to make the script executable:

   ```
   chmod +x openssh_docker.sh
   ```

3. **Run the Script**: Execute the script to create the OpenSSH server container:

   ```
   ./openssh_docker.sh
   ```

Once the script runs, the OpenSSH server will be available, and you can connect to it using an SSH client on port 2222.

Connecting to the OpenSSH Server

After running the script, you can connect to the OpenSSH server using an SSH client. Since the container maps port 2222, you will need to specify the port when connecting:

```
ssh -p 2222 paulgrey@your-server-ip
```

If you are using SSH key authentication, ensure that your public key is properly set in the `~/.sshk/` directory on the host.

Conclusion

Running OpenSSH in a Docker container provides a lightweight, isolated, and secure way to manage SSH access. This script automates the process of setting up the server, configuring user credentials, and managing SSH keys. With this setup, you can quickly deploy an SSH server on any machine, knowing that it will always restart if needed and provide secure access via SSH keys.

Feel free to customize the script for your specific use case, whether that means changing the port mapping, adjusting the container’s environment variables, or using a different public key. Let me know if you need further assistance or clarification!