

## Practical Lab Guide 3: Deploying a Local Application on EC2 Remote Server Using Ansible

In this exercise, you'll learn how to install and configure Ansible on a control node, manage EC2 instances via Ansible, and automate the deployment of a local Django-React application on a remote server.

### Step 1: Installing Ansible on the Control Node

Ansible runs from a control node and manages remote servers (target nodes). This step outlines how to install Ansible on your local machine (the control node). Please note, **Ansible cannot be installed on Windows**; it requires a Unix-like system such as Linux or macOS.

**Add Ansible PPA to your system:** This step ensures your system includes the Ansible repository.

```
sudo apt-add-repository ppa:ansible/ansible
```

**Update the package index:** Refresh the package list to include the newly added Ansible PPA.

```
sudo apt update
```

**Install Ansible:** Finally, install Ansible using the following command:

```
sudo apt install ansible
```

**Ansible is written in Python, so you'll notice that Python will be installed as a dependency.**

### For windows users :

Since Ansible is designed for Unix-like systems, Windows users can work with Ansible through Docker. This approach allows Windows users to set up a Docker container as the control node for Ansible tasks, without needing a Unix environment on their local machine

### **Pull the `cytopia/ansible` Docker Image**

The `cytopia/ansible` image contains Ansible pre-installed, which is ideal for running playbooks and managing remote nodes.

```
docker pull cytopia/ansible
```

**Note:** Work in a directory on your local machine as outlined in the following steps. Then, mount this directory as a volume to the Ansible container. From there, you'll be able to run your playbooks directly.

## Step 2: Setting Up the Inventory File

The inventory file holds information about the servers (hosts) you'll manage. For this exercise, you'll create an inventory file listing the EC2 instance(s) you want to manage.

Create an Ansible project directory:

```
mkdir ansible  
cd ansible
```

Download the **deployer\_key.pem**:

- This key was generated by Terraform and attached to your EC2 instance.
- After downloading, place the key inside the **ansible** directory.

**Set the Key Permissions:** ( only for linux users )

- The SSH key file should have strict permissions for security.  

```
chmod 600 deployer_key.pem
```

Create the inventory file: Create a file named **hosts** to define your managed hosts.

```
touch hosts
```

Organize your hosts into groups: An inventory file can group servers for better management. For example:

```
[servers]  
server1 ansible_host=203.0.113.111  
server2 ansible_host=203.0.113.112  
server3 ansible_host=203.0.113.113
```

Set global variables for Python interpreter: If your remote servers use Python 3 (which is common for Ubuntu), you'll need to specify it in the inventory file.

```
[servers:vars]
ansible_python_interpreter=/usr/bin/python3
```

### Step 3: Configure the EC2 Instance for Ansible

To connect Ansible to an EC2 instance, you need to define certain details such as the host's IP address, SSH key file, and user information.

Edit the hosts file to include your EC2 instance:

```
[servers]
54.159.37.2 # your ec2 instance public ip address
[servers:vars]
ansible_python_interpreter=/usr/bin/python3
ansible_ssh_private_key_file=deployer_key.pem
ansible_user=ubuntu
```

- **ansible\_python\_interpreter**: Specifies the path to Python 3 on the remote server.
- **ansible\_ssh\_private\_key\_file**: Refers to the SSH private key used to connect to the EC2 instance.

### Step 4: Check the Connection

After setting up the inventory file, check the connection between your control node (local machine) and the remote EC2 instance.

**Run the ping command:** This command verifies that Ansible can communicate with the remote instance.

```
ansible all -i hosts -m ping
or
ansible servers -i hosts -m ping
```

You should see a successful response (pong).

### for windows users : manipulation with ansible container

```
cd ansible
run the ansible container
```

```
docker run -d -v "full path of your directory ansible in your machine:/workspace" --name my_ansible_container cytopia/ansible sh -c "while true; do sleep 3600; done"
```

**for example :**

```
docker run -d -v "C:/ansible:/workspace" --name my_ansible_container cytopia/ansible sh -c "while true; do sleep 3600; done"
```

**Explanation of the command:**

- **-d**: Runs the container in detached mode.
- **-v "C:\ansible:/workspace"**: Mounts the local ansible directory to /workspace inside the container. This allows you to access playbooks and other files within the container.
- **--name my\_ansible\_container**: Assigns a name to the container so you can easily reference it.
- **cytopia/ansible**: Specifies the Docker image.
- **sh -c "while true; do sleep 3600; done"**: Keeps the container running by making it loop indefinitely.

then `cd /workspace` and `ls` there you can see your mounted files in the container

**Note:** When your container is running in detached mode with a mounted volume, any updates you make to files in the mounted directory on your local machine (e.g., `ansible`) are immediately reflected inside the container. You don't need to restart or reattach the volume; Docker dynamically reflects changes from the host into the container's mounted volume.

### **Install SSH Client Inside the Container**

Ansible requires SSH to connect to remote nodes. Since `cytopia/ansible` doesn't include SSH by default, install the `openssh-client` within the container.

Inside the container, run:

```
apk update && apk add openssh
```

(The `apk` package manager is used for Alpine-based images, which is likely the base for `cytopia/ansible`.)

## Verify SSH Installation

After installation, confirm that `ssh` is available by running:

```
ssh -V
```

This should display the version of the SSH client, confirming that it's now installed.

```
/workspace # apk update && apk add openssh
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/community/x86_64/APKINDEX.tar.gz
v3.16.9-124-g7ebe731c813 [https://dl-cdn.alpinelinux.org/alpine/v3.16/main]
v3.16.9-125-gec300a94000 [https://dl-cdn.alpinelinux.org/alpine/v3.16/community]
OK: 17044 distinct packages available
(1/8) Installing openssh-keygen (9.0_p1-r5)
(2/8) Installing libedit (20210910.3.1-r0)
(3/8) Installing openssh-client-common (9.0_p1-r5)
(4/8) Installing openssh-client-default (9.0_p1-r5)
(5/8) Installing openssh-sftp-server (9.0_p1-r5)
(6/8) Installing openssh-server-common (9.0_p1-r5)
(7/8) Installing openssh-server (9.0_p1-r5)
(8/8) Installing openssh (9.0_p1-r5)
Executing busybox-1.35.0-r17.trigger
OK: 83 MiB in 48 packages
/workspace # ssh -V
OpenSSH_9.0p1, OpenSSL 1.1.1w  11 Sep 2023
```

Now lets give the right permissions to the deployer key :

```
chmod 600 deployer_key.pem
```

**Run the ping command:** This command verifies that Ansible can communicate with the remote instance.

```
ansible all -i hosts -m ping
```

or

```
ansible servers -i hosts -m ping
```

You should see a successful response (`pong`).

```
/workspace # chmod 600 deployer_key.pem
/workspace # ansible all -i hosts -m ping
[WARNING]: Platform linux on host 3.208.34.116 is using the discovered Python interpreter at /usr/bin/python3.12, but future installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
3.208.34.116 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3.12"
  },
  "changed": false,
  "ping": "pong"
}
/workspace #
```

## Step 5: Handling Host Key Checks

When SSH'ing into a remote server, you might encounter a "host key check" issue.

```
[\\W]$ ssh -i ~/Downloads/ansible.pem ec2-user@ec2-35-180-204-130.eu-west-3.compute.amazonaws.com
The authenticity of host 'ec2-35-180-204-130.eu-west-3.compute.amazonaws.com (35.180.204.130)' can't be
established.
ECDSA key fingerprint is SHA256:r1afatu8D0g5zCQdrdDpFE8sRh+uJRNKcBudKq6zVS8.
Are you sure you want to continue connecting (yes/no)? █
```

There are two ways to handle this:

1. Long-lived servers: Use `ssh-keyscan` to add the server to `known_hosts`:

```
ssh-keyscan -H <your-ec2-ip-address> >> ~/.ssh/known_hosts
```

2. Ephemeral servers: For automation, disable the host key checking Ansible's configuration file.

Create the Ansible configuration file: Add a configuration file (`ansible.cfg`) with the following content:

```
touch ansible.cfg

[defaults]

host_key_checking = False

inventory = hosts
```

This setting ensures that the host key checking is bypassed for smooth automation.

```
imen@imen-ASUS-TUF-Dash-F15-FX517ZC-FX517ZC:~/tp enis/ansible$ ansible servers -i hosts -m ping
The authenticity of host '54.226.174.47 (54.226.174.47)' can't be established.
ED25519 key fingerprint is SHA256:uHpth49nz9aVWEcCXZntVdIM3csQG2L5fRAjoQBf5cE.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
54.226.174.47 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
imen@imen-ASUS-TUF-Dash-F15-FX517ZC-FX517ZC:~/tp enis/ansible$ touch ansible.cfg
imen@imen-ASUS-TUF-Dash-F15-FX517ZC-FX517ZC:~/tp enis/ansible$ ansible servers -i hosts -m ping
54.226.174.47 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

## Step 6: Basic Ansible Playbook Example

**Create Your First Playbook :** Inside your ansible project directory, create a file named `first_playbook.yml`:

```
touch first_playbook.yml
```

This playbook performs the following tasks on the target servers:

1. **Ensures SSH connectivity** is available before continuing.
2. **Updates the apt package cache** to make sure packages are up to date.
3. **Installs the Nginx web server** package to set up a basic web server.
4. **Starts the Nginx service** and ensures it will run on startup.
5. **Creates an index.html** file with simple content that will be served by Nginx.

```
1  ---
2  - name: Basic Web Server Setup
3    hosts: servers
4    gather_facts: False
5    become: yes
6
7    tasks:
8      - name: Ensure SSH connection is available
9        ansible.builtin.wait_for:
10          port: 22
11          delay: 10
12          timeout: 100
13          search_regex: OpenSSH
14          host: '{{ ansible_host | default(inventory_hostname) }}'
15        vars:
16          ansible_connection: local
17        become: no
18
19      - name: Update apt cache manually
20        apt:
21          update_cache: yes
22          cache_valid_time: 3600
23
24      - name: Install Nginx
25        apt:
26          name: nginx
27          state: present
28      - name: Start Nginx service
29        service:
30          name: nginx
31          state: started
32          enabled: yes
33
34      - name: Create index.html with "Hello, World!"
35        copy:
36          dest: /var/www/html/index.html
37          content: "Hello from ansible "
38          mode: '0644'
```

## Copy code :

```
---
- name: Basic Web Server Setup
  hosts: servers
  gather_facts: False
  become: yes

  tasks:
    - name: Ensure SSH connection is available
      ansible.builtin.wait_for:
        port: 22
        delay: 10
        timeout: 100
        search_regex: OpenSSH
        host: '{{ ansible_host | default(inventory_hostname) }}'
      vars:
        ansible_connection: local
      become: no

    - name: Update apt cache manually
      apt:
        update_cache: yes
        cache_valid_time: 3600

    - name: Install Nginx
      apt:
        name: nginx
        state: present

    - name: Start Nginx service
      service:
        name: nginx
        state: started
        enabled: yes

    - name: Create index.html with "Hello, World!"
      copy:
        dest: /var/www/html/index.html
        content: "Hello from ansible "
        mode: '0644'
```

## Explanation of Key Components

### Play:

- A play is a collection of tasks that will run on a set of hosts. The **name** of the play is a descriptive identifier.

### hosts:



- Specifies which hosts or group of hosts (from your inventory file) will be managed by the play.
- Example: `hosts: servers` means this play will run on all hosts in the `servers` group in the hosts file.

#### become:

- This is used to escalate privileges (like running tasks with `sudo`).
- Example: `become: yes` means the tasks will be run with elevated privileges.

#### tasks:

- The `tasks` section defines a list of actions that will be executed sequentially on the target hosts.
- Each item under `tasks` is called a **task**.

#### Task:

- A task is a single action that Ansible performs. Tasks are defined inside the `tasks` list.
- Each task has a `name` (a description of the task), followed by a module.

#### Example:

```
- name: Install NGINX
  apt:
```

#### Module:

- A module is a predefined command that Ansible uses to perform specific actions (like installing packages, copying files, or managing services).
- Modules are the building blocks of Ansible tasks.
- Examples of modules: `apt`, `service`, `copy`.
- Example of using the `apt` module to install NGINX:

#### Subtasks/Parameters:

- Parameters are the options or arguments that you pass to a module to define how the module should behave.
- For example, in the `apt` module, `name: nginx` specifies the package to install, and `state: present` ensures that the package is installed.
- Similarly, in the `copy` module, `content` defines the content of the file, and `dest` specifies the destination file path.

To **run the Ansible playbook**, you use the `ansible-playbook` command. Below, I'll explain how to execute your playbook step by step, including the key options:

`ansible-playbook -i hosts first_playbook.yml`

### same for ansible container for windows users

Command Breakdown:

1. `ansible-playbook`: Executes the playbook.
2. `-i hosts`: Specifies the inventory file (hosts) that lists target servers.
3. `first_playbook.yml`: The name of the playbook to run.

### for linux users

```
imen@imen-ASUS-TUF-Dash-F15-FX517ZC-FX517ZC:~/tp enis/ansible$ ansible-playbook -i hosts first_playbook.yml

PLAY [Basic Web Server Setup] *****

TASK [Ensure SSH connection is available] *****
ok: [34.229.247.74]

TASK [Update apt cache manually] *****
changed: [34.229.247.74]

TASK [Install Nginx] *****
changed: [34.229.247.74]

TASK [Start Nginx service] *****
ok: [34.229.247.74]

TASK [Create index.html with "Hello, World!"] *****
changed: [34.229.247.74]

PLAY RECAP *****
34.229.247.74      : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

### for windows users

```
/workspace # ansible-playbook -i hosts first_playbook.yml

PLAY [Basic Web Server Setup] *****

TASK [Ensure SSH connection is available] *****
ok: [3.208.34.116]

TASK [Update apt cache manually] *****
changed: [3.208.34.116]

TASK [Install Nginx] *****
changed: [3.208.34.116]

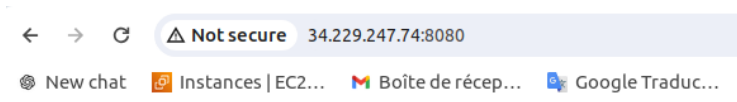
TASK [Start Nginx service] *****
ok: [3.208.34.116]

TASK [Create index.html with "Hello, World!"] *****
changed: [3.208.34.116]

PLAY RECAP *****
3.208.34.116      : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

/workspace #
```

You can access the web server from a browser using the public IP address of the server. Just open `http://<your-ec2-public-ip>` to see the message "Hello from ansible".



# Hello, World

## Step 7: Deploying Applications with Ansible

In this step, we will create a new Ansible playbook that will deploy our application using Docker. This playbook will help us automate the process of pulling a Docker image, running the container, and ensuring that our application is up and running on the server.

1. Create a new YAML file named `docker_deploy_playbook.yml` to deploy your application using Docker.
2. Install docker and docker compose:

To eliminate hardcoding of variables, create a file named `ansible-vars.yml` to store commonly used variables, such as Docker version, application image name, and ports. Here is an example of the variables file:

```
touch ansible-vars.yml
```

```
# ansible-vars.yml

docker_service_name: "docker"

cache_time: 3600

docker_package_name: "docker.io"
```

## Updated Playbook

Here's the `docker_deploy_playbook.yml` Ansible playbook that references these variables: ( we keep the first tasks that ssh into the instance then we add the bunch of tasks that install docker and docker compose )

## Copy code :

```
---

- name: Basic Web Server Setup

  hosts: servers

  gather_facts: False
```

```
become: yes
```

```
tasks:
```

- name: Ensure SSH connection is available

```
ansible.builtin.wait_for:
```

```
port: 22
```

```
delay: 10
```

```
timeout: 100
```

```
search_regex: OpenSSH
```

```
host: '{{ ansible_host | default(inventory_hostname) }}'
```

```
vars:
```

```
ansible_connection: local
```

```
become: no
```

- name: Install Docker and Docker-compose

```
hosts: servers
```

```
become: yes
```

```
vars_files:
```

- ansible-vars.yaml

```
tasks:
```

- name: Update apt cache manually

```
apt:
```

```
update_cache: yes
```

```
cache_valid_time: '{{ cache_time }}'
```

- name: Install Docker

```
apt:
```

```
name: '{{ docker_package_name }}'
```

```
update_cache: yes
```

```
state: present
```

- name: Install Docker-compose

```
apt:
```

```
name: docker-compose
```

```
state: present
```

- name: Ensure Docker service is started

```
systemd:
```

```
name: '{{ docker_service_name }}'
```

```

    state: started

    enabled: yes

- name: Check Docker version

    command: docker --version

    register: docker_version_output

- name: Display Docker version

    debug:

        msg: "Docker version: {{ docker_version_output.stdout }}"

- name: Check Docker Compose version

    command: docker-compose --version

    register: docker_compose_version_output

- name: Display Docker Compose version

    debug:

        msg: "Docker Compose version: {{ docker_compose_version_output.stdout }}"

```

**run the playbook: ansible-playbook -i hosts docker\_deploy\_playbook.yml**

### 3. Add ubuntu user to docker group:

```

- name: Add ubuntu user to docker group
  user:
    name: ubuntu
    groups: docker
    append: yes

- name: Reconnect to server session
  meta: reset_connection

```

### Copy code :

```

- name: Add ubuntu user to docker group

user:

    name: ubuntu

    groups: docker

    append: yes

```

```
- name: Reconnect to server session

meta: reset_connection
```

## Explanation

### Task 1: Add ubuntu user to docker group

This task adds the **ubuntu** user to the **docker** group on the target host .  
By adding the user to the **docker** group, the user will have permission to execute Docker commands without using **sudo**. The **append: yes** ensures that existing groups are not overwritten, and only the **docker** group is added.

### Task 2: Reconnect to server session

After modifying the user's group membership, Ansible resets the SSH connection to the server using **meta: reset\_connection**. This ensures that the updated group membership takes effect immediately for subsequent tasks.

run the playbook: **ansible-playbook -i hosts docker\_deploy\_playbook.yml**

**Before pushing images to the repo in AWS there is few changes need to be done :**

- **Creating the database using AWS Service RDS using FREE TIER**

The screenshot shows the Amazon RDS console interface. On the left is a navigation menu with options like Dashboard, Databases, Query Editor, Performance Insights, Snapshots, Exports In Amazon S3, Automated backups, Reserved Instances, Proxies, Subnet groups, Parameter groups, Option groups, Custom engine versions, Zero-ETL integrations, Events, Event subscriptions, Recommendations, and Certificate update. The main content area is titled 'RDS > Databases'. It features a red error message at the top stating that the user is not authorized to perform certain actions. Below this is a blue informational banner suggesting the creation of a Blue/Green Deployment. The 'Databases (0)' section includes a search bar and a table with columns for DB identifier, Status, Role, Engine, Region, Size, Recommendations, CPU, and Current... The table currently shows 'No instances found'.

## Create database [Info](#)

### Choose a database creation method

☒ **Standard create**

You set all of the configuration options, including ones for availability, security, backups, and maintenance.

☐ **Easy create**

Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

### Engine options

#### Engine type [Info](#)

☐ Aurora (MySQL Compatible)



☐ Aurora (PostgreSQL Compatible)



☒ **MySQL**



☐ MariaDB



☐ PostgreSQL



☐ Oracle

ORACLE®

☐ Microsoft SQL Server



☐ IBM Db2

IBM Db2

#### Edition

☒ **MySQL Community**

#### Engine version [Info](#)

View the engine versions that support the following database features.

▼ **Hide filters**

☐ **Show only versions that support the Multi-AZ DB cluster** [Info](#)

Create a Multi-AZ DB cluster with one primary DB instance and two readable standby DB instances. Multi-AZ DB clusters provide up to 2x faster transaction commit latency and automatic failover in typically under 35 seconds.

☐ **Show only versions that support the Amazon RDS Optimized Writes** [Info](#)

Amazon RDS Optimized Writes improves write throughput by up to 2x at no additional cost.

#### Engine version

MySQL 8.0.35 ▼

☐ **Enable RDS Extended Support** [Info](#)

Amazon RDS Extended Support is a [paid offering](#). By selecting this option, you consent to being charged for this offering if you are running your database major version past the RDS end of standard support date for that version. Check the end of standard support date for your major version in the [RDS for MySQL documentation](#).

### Templates

Choose a sample template to meet your use case.

☐ **Production**

Use defaults for high availability and fast, consistent performance.

☐ **Dev/Test**

This instance is intended for development use outside of a production environment.

☒ **Free tier**

Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS. [Info](#)

### Deployment options [Info](#)

The deployment options below are limited to those supported by the engine you selected above.

- ☐ Multi-AZ DB Cluster  
Creates a DB cluster with a primary DB instance and two readable standby DB instances, with each DB instance in a different Availability Zone (AZ). Provides high availability, data redundancy and increases capacity to serve read workloads.
- ☐ Multi-AZ DB Instance (not supported for Multi-AZ DB cluster snapshot)  
Creates a primary DB instance and a standby DB instance in a different AZ. Provides high availability and data redundancy, but the standby DB instance doesn't support connections for read workloads.
- ☐ Single DB Instance (not supported for Multi-AZ DB cluster snapshot)  
Creates a single DB instance with no standby DB instances.

## Settings

### DB Instance Identifier [Info](#)

Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

### ▼ Credentials Settings

#### Master username [Info](#)

Type a login ID for the master user of your DB instance.

1 to 16 alphanumeric characters. The first character must be a letter.

#### Credentials management

You can use AWS Secrets Manager or manage your master user credentials.

☐ **Managed in AWS Secrets Manager - *most secure***  
RDS generates a password for you and manages it throughout its lifecycle using AWS Secrets Manager.

☒ **Self managed**  
Create your own password or have RDS create a password that you manage.

☐ **Auto generate password**

Amazon RDS can generate a password for you, or you can specify your own password.

#### Master password [Info](#)

#### Password strength [Strong](#)

Minimum constraints: At least 8 printable ASCII characters. Can't contain any of the following symbols: / ' " @

#### Confirm master password [Info](#)

## Instance configuration

The DB instance configuration options below are limited to those supported by the engine that you selected above.

### DB Instance class [Info](#)

#### ▼ Hide filters

☒ **Show Instance classes that support Amazon RDS Optimized Writes [Info](#)**

Amazon RDS Optimized Writes improves write throughput by up to 2x at no additional cost.

☐ **Include previous generation classes**

- ☐ Standard classes (Includes m classes)
- ☐ Memory optimized classes (Includes r and x classes)
- ☒ **Burstable classes (Includes t classes)**

2 vCPUs 1 GiB RAM Network: Up to 2,085 Mbps

## Storage

### Storage type [Info](#)

Provisioned IOPS SSD (io2) storage volumes are now available.

Performance scales independently from storage

### Allocated storage [Info](#)

GIB

Minimum: 20 GiB. Maximum: 6,144 GiB

**ⓘ** After you modify the storage for a DB instance, the status of the DB instance will be in storage-optimization. Your instance will remain available as the storage-optimization operation completes. [Learn more](#)

#### ► Advanced settings

Baseline IOPS of 3,000 IOPS and storage throughput of 125 MiBps are included for allocated storage less than 400 GiB.

#### ► Storage autoscaling



► Storage autoscaling

## Connectivity [Info](#)



### Compute resource

Choose whether to set up a connection to a compute resource for this database. Setting up a connection will automatically change connectivity settings so that the compute resource can connect to this database.

- ☒ **Don't connect to an EC2 compute resource**  
Don't set up a connection to a compute resource for this database. You can manually set up a connection to a compute resource later.

- ☐ **Connect to an EC2 compute resource**  
Set up a connection to an EC2 compute resource for this database.

### Virtual private cloud (VPC) [Info](#)

Choose the VPC. The VPC defines the virtual networking environment for this DB instance.

tp\_cloud\_devops\_vpc (vpc-09dbb252bddeabdce)  
2 Subnets, 2 Availability Zones

Only VPCs with a corresponding DB subnet group are listed.

- After a database is created, you can't change its VPC.**

### DB subnet group [Info](#)

Choose the DB subnet group. The DB subnet group defines which subnets and IP ranges the DB instance can use in the VPC that you selected.

Create new DB Subnet Group

### Public access [Info](#)

- ☒ **Yes**  
RDS assigns a public IP address to the database. Amazon EC2 instances and other resources outside of the VPC can connect to your database. Resources inside the VPC can also connect to the database. Choose one or more VPC security groups that specify which resources can connect to the database.
- ☐ **No**  
RDS doesn't assign a public IP address to the database. Only Amazon EC2 instances and other resources inside the VPC can connect to your database. Choose one or more VPC security groups that specify which resources can connect to the database.

### VPC security group (firewall) [Info](#)

Choose one or more VPC security groups to allow access to your database. Make sure that the security group rules allow the appropriate incoming traffic.

- ☒ **Choose existing**  
Choose existing VPC security groups

- ☐ **Create new**  
Create new VPC security group

#### Existing VPC security groups

Choose one or more options

web-server-sg ✕

### Availability Zone [Info](#)

No preference

### RDS Proxy

RDS Proxy is a fully managed, highly available database proxy that improves application scalability, resiliency, and security.

- ☐ **Create an RDS Proxy [Info](#)**  
RDS automatically creates an IAM role and a Secrets Manager secret for the proxy. RDS Proxy has additional costs. For more information, see [Amazon RDS Proxy pricing](#).

### Certificate authority - optional [Info](#)

Using a server certificate provides an extra layer of security by validating that the connection is being made to an Amazon database. It does so by checking the server certificate that is automatically installed on all databases that you provision.

rds-ca-rsa2048-g1 (default)  
Expiry: May 26, 2061

If you don't select a certificate authority, RDS chooses one for you.

## ► Additional configuration

### Tags - optional

A tag consists of a case-sensitive key-value pair.

No tags associated with the resource.

Add new tag

You can add up to 50 more tags.

## Database authentication

### Database authentication options [Info](#)

- ☒ **Password authentication**  
Authenticates using database passwords.
- ☐ **Password and IAM database authentication**  
Authenticates using the database password and user credentials through AWS IAM users and roles.
- ☐ **Password and Kerberos authentication**  
Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.

### Monitoring

☐ **Enable Enhanced Monitoring**  
Enabling Enhanced Monitoring metrics are useful when you want to see how different processes or threads use the CPU.

► **Additional configuration**

Database options, encryption turned on, backup turned on, backtrack turned off, maintenance, CloudWatch Logs, delete protection turned off.

### Estimated monthly costs

The Amazon RDS Free Tier is available to you for 12 months. Each calendar month, the free tier will allow you to use the Amazon RDS resources listed below for free:

- 750 hrs of Amazon RDS in a Single-AZ db.t2.micro, db.t3.micro or db.t4g.micro Instance.
- 20 GB of General Purpose Storage (SSD).
- 20 GB for automated backup storage and any user-initiated DB Snapshots.

[Learn more about AWS Free Tier.](#)

When your free usage expires or if your application use exceeds the free usage tiers, you simply pay standard, pay-as-you-go service rates as described in the [Amazon RDS Pricing page](#).

**ⓘ** You are responsible for ensuring that you have all of the necessary rights for any third-party products or services that you use with AWS services.

Cancel

Create database

**Ensure you reach finally to that configuration with no additional cost**

RDS > Databases

**ⓘ** Consider creating a Blue/Green Deployment to minimize downtime during upgrades  
 You may want to consider using Amazon RDS Blue/Green Deployments and minimize your downtime during upgrades. A Blue/Green Deployment provides a staging environment for changes to production databases. [RDS User Guide](#) [Aurora User Guide](#)

Databases (1)

Group resources

Modify

Actions

Restore from S3

Create database

DB identifier	Status	Role	Engine	Region &...	Size	Recommendations	CPU	Current a...	Maintena...
mydb	Available	Instance	MySQL Com...	us-east-1a	db.t3.micro		-		none

After creating the database, please wait a few minutes for the database status to change to **Available**. Once the status is Available, click on the database name to open its details page. Navigate to the **Connectivity & security tab**, where you'll find the **Endpoint** and **Port** for accessing the database.

Connectivity & security

Monitoring

Logs & events

### Connectivity & security

#### Endpoint & port

Endpoint

mydb.cxuo68eem91h.us-east-1.rds.amazonaws.com

Port

3306

#### 4. Changes in backend and frontend before build:

go to settings.py in the django project and change the database section:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'enis_tp',
        'USER': 'dbuser',
        'PASSWORD': 'DBpassword2024',
        'HOST': 'endpoint of rds ',
        'PORT': 3306,
    }
}
```

**You need to open the port 3306 of the database in the associated security group , Go back to terraform project and add this portion of code that allows inbound traffic on port 3306**

##### **# Allow inbound RDS traffic (e.g., MySQL on port 3306)**

```
resource "aws_security_group_rule" "allow_rds_inbound" {
    type                = "ingress"
    security_group_id   = aws_security_group.web_sg.id

    from_port           = 3306                # Change this to match your
    database port
    to_port              = 3306                # Same as above
    protocol              = "tcp"

    cidr_blocks          = ["0.0.0.0/0"]      # For production, replace
    this with a specific IP or CIDR block
}
```

##### **# Allow inbound to backend on port 8000**

```
resource "aws_security_group_rule" "allow_backend_inbound" {
    type                = "ingress"
    security_group_id   = aws_security_group.web_sg.id

    from_port           = 8000
    to_port              = 8000                # Same as above
    protocol              = "tcp"
}
```

```

    cidr_blocks      = ["0.0.0.0/0"]          # For production, replace
this with a specific IP or CIDR block
}

```

## # Allow inbound HTTP traffic on port 81 to access the final application

```

resource "aws_security_group_rule" "allow_web_http_inbound-81" {

    type            = "ingress"

    security_group_id = aws_security_group.web_sg.id

    from_port      = 81

    to_port        = 81

    protocol       = "tcp"

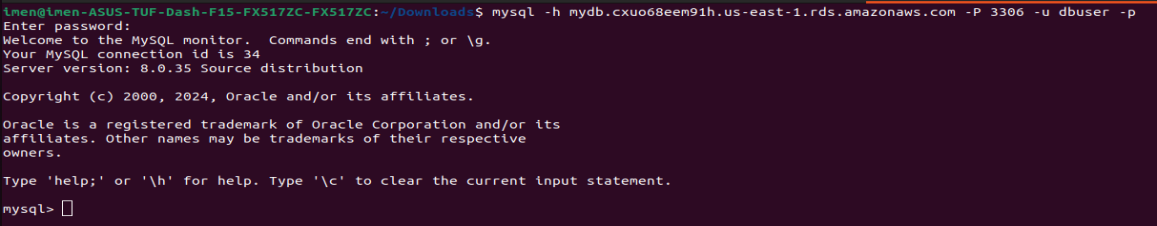
    cidr_blocks    = ["0.0.0.0/0"]
}

```

## Run terraform apply

## Now connect on the db to create the enis\_tp DB inside the rds

```
mysql -h mydb.cxuo68eem91h.us-east-1.rds.amazonaws.com -P 3306 -u dbuser -p
```



```

lmen@lmen-ASUS-TUF-Dash-F15-FX517ZC-FX517ZC:~/Downloads$ mysql -h mydb.cxuo68eem91h.us-east-1.rds.amazonaws.com -P 3306 -u dbuser -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 34
Server version: 8.0.35 Source distribution

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 

```

You'll be prompted for the password. Enter **DBpassword2024**

Create the Database: **CREATE DATABASE enis\_tp;**

Check if the Database Exists : **SHOW DATABASES;**

Exit the MySQL Shell: Once done, exit the MySQL shell with: **EXIT;**

```

lmen@lmen-ASUS-TUF-Dash-F15-FX517ZC-FX517ZC:~/Downloads$ mysql -h mydb.cxuo68eem91h.us-east-1.rds.amazonaws.com -P 3306 -u dbse
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 34
Server version: 8.0.35 Source distribution

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE enis_tp;
Query OK, 1 row affected (0.11 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| enis_tp   |
| information_schema |
| mysql     |
| performance_schema |
| sys       |
+-----+
5 rows in set (0.10 sec)

mysql> EXIT;
Bye

```

### Then you can rebuild the backend app

2. go to the react app and change this in api.js and change

```

const api :AxiosInstance = axios.create({
  baseURL: "http://localhost:8000"
});

```

by the ip address of the ec2 instance

```

const api :AxiosInstance = axios.create({
  baseURL: "http://34.229.247.74:8000"
});

```

### Then you can rebuild the frontend app

## 5. Manually Build and Push Backend and Frontend Docker Images to AWS ECR

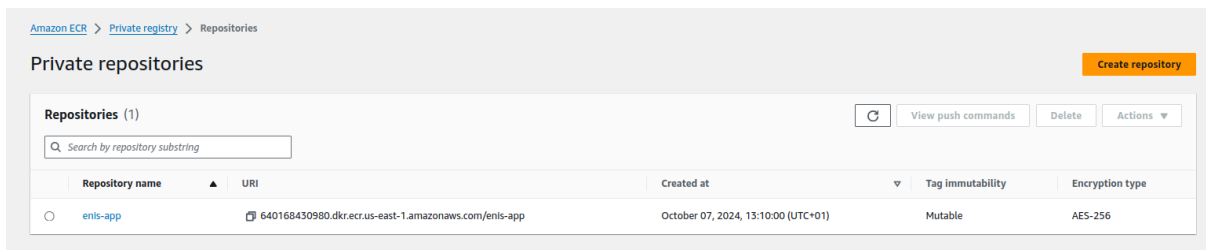
### Step 1: What is AWS ECR (Elastic Container Registry)?

Amazon Elastic Container Registry (ECR) is a fully managed Docker container registry provided by AWS that allows you to store, manage, and deploy container images. It integrates with Amazon ECS, EKS, and other AWS services, providing a secure and scalable storage for container images. You can push Docker images to ECR and pull them to AWS services or other environments when needed.

### Step 2: Create an ECR Repository in AWS

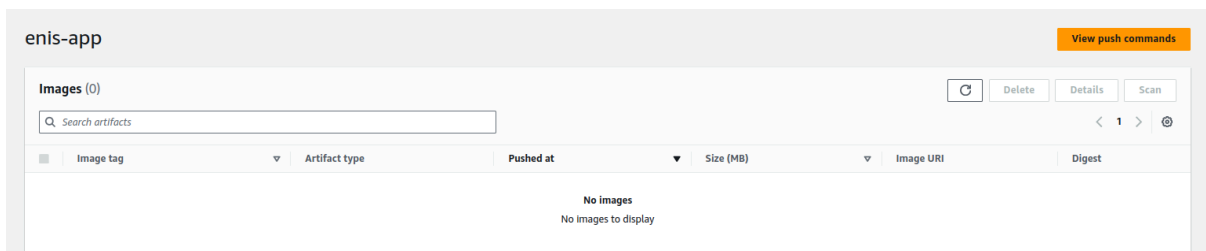
Before you can push Docker images, you must create a repository in AWS ECR to store your backend and frontend images.

1. **Log in to the AWS Management Console:**
  - Go to the [Amazon ECR Console](#).
2. **Create a Repository:**
  - Navigate to **Amazon ECR > Repositories**.
  - Click on **Create repository**.
  - Name the repository **enis-app** (you can use any name you prefer).
  - Choose **Visibility**: Private (by default).
  - Click **Create repository** to finalize the setup.
3. Now, you have an ECR repository ready to receive your Docker images.



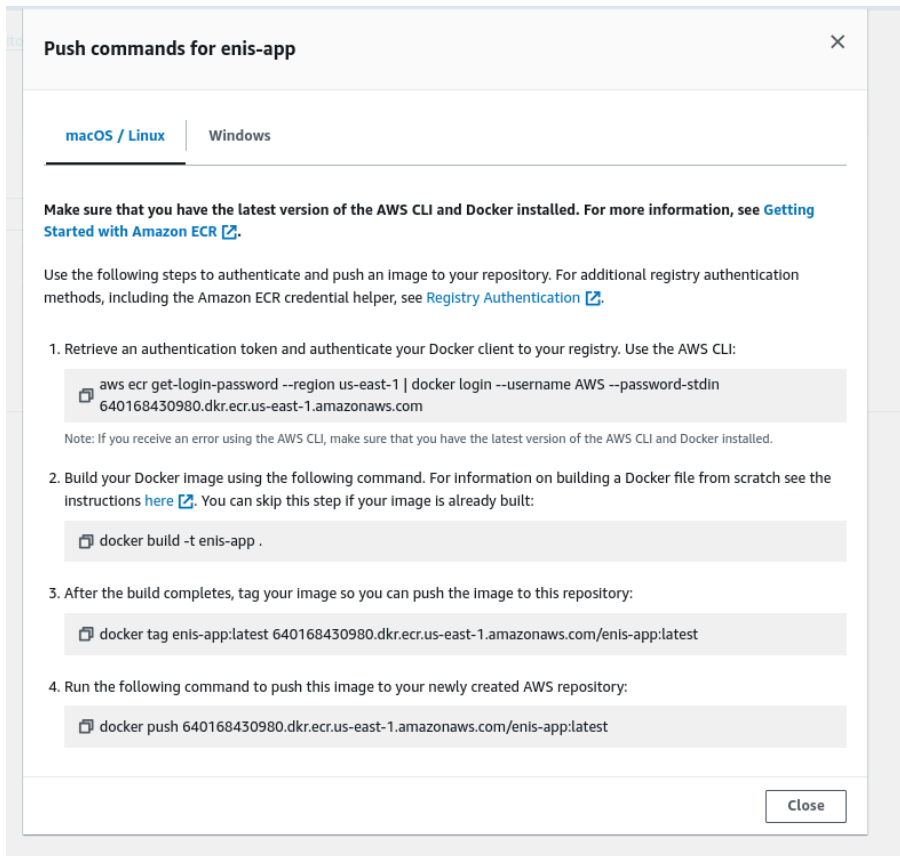
### Step 3: Build and Push Frontend Image to ECR

#### Commands to Build and Push the Frontend Docker Image



**Click on View push commands and follow**

**we will need just log in , tag and push the images since they are already built**



### 1. Give Docker Access to Your User:

```
sudo usermod -aG docker $USER
```

```
sudo chmod 666 /var/run/docker.sock
```

### 2. Login to AWS ECR:

```
aws ecr get-login-password --region us-east-1 | docker login  
--username AWS --password-stdin  
746200881003.dkr.ecr.us-east-1.amazonaws.com
```

This command logs you into your AWS ECR repository by using the AWS CLI to retrieve your login credentials.

Replace `746200881003.dkr.ecr.us-east-1.amazonaws.com` with your ECR URL.

### 3. Tag the Frontend Image:

```
docker tag frontend:latest 640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app:frontend-1.0
```

`docker tag` assigns the image to the ECR repository with a specific name and version.

`frontend:latest` is the local image you're tagging.

`640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app:frontend-app-1.0` is the full path of the repository in ECR

```
imen@imen-ASUS-TUF-Dash-F15-FX517ZC-FX517ZC:~/tp enis/enis-app-tp/frontend$ docker tag backend:latest 640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app:backend-1.0
imen@imen-ASUS-TUF-Dash-F15-FX517ZC-FX517ZC:~/tp enis/enis-app-tp/frontend$ docker images
REPOSITORY                                     TAG          IMAGE ID      CREATED        SIZE
640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app    frontend     02d3d9dbf1dd  16 minutes ago  43.4MB
640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app    latest       02d3d9dbf1dd  16 minutes ago  43.4MB
640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app    <none>       15646fe2696b  16 minutes ago  728MB
640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app    backend-1.0  e8a975c57f0e  21 minutes ago  295MB
```

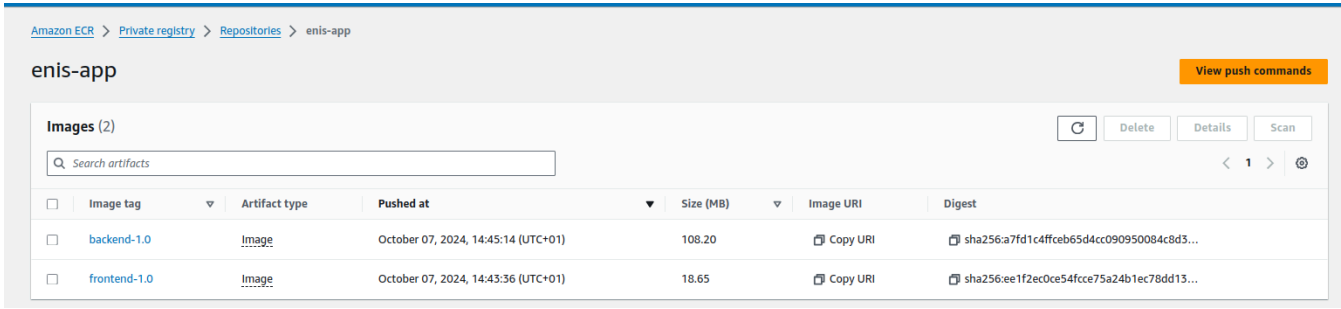
4. Push the Frontend Image to ECR:

`docker push 640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app:frontend-1.0`

`docker push 640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app:backend-1.0`

```
imen@imen-ASUS-TUF-Dash-F15-FX517ZC-FX517ZC:~/tp enis/enis-app-tp/frontend$ docker push 640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app:frontend-1.0
The push refers to repository [640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app]
e28aadd394cf: Pushed
b0f60355fd52: Pushed
027907faf592: Pushed
11134cc97d7f: Pushed
f7a5847cdca9: Pushed
aec1e8cf14f5: Pushed
717b3a077b07: Pushed
2ff96b2e5450: Pushed
63ca1fbb43ae: Pushed
frontend-1.0: digest: sha256:ee1f2ec0ce54fccc75a24b1ec78dd132477b96b94841b37f5aef401f207db7e8 size: 2198
imen@imen-ASUS-TUF-Dash-F15-FX517ZC-FX517ZC:~/tp enis/enis-app-tp/frontend$ docker push 640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app:backend-1.0
The push refers to repository [640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app]
4a662fda027d: Pushed
8c16490b08d9: Pushed
58783f9e10fd: Pushed
90034b501a11: Pushed
6f65de0d2cde: Pushed
7bbf9f88fb23: Pushed
c5321f7f53ff: Pushed
df6c1b185b95: Pushed
b23fedba7dbd: Pushed
ae2d55769c5e: Pushed
e2ef8a51359d: Pushed
backend-1.0: digest: sha256:a7fd1c4ffceb65d4cc090950084c8d3f845836ac802e8003fc29b0d44236f3fb size: 2625
imen@imen-ASUS-TUF-Dash-F15-FX517ZC-FX517ZC:~/tp enis/enis-app-tp/frontend$
```

Verify that the images exist in the ECR





## 6. Pull images from the AWS ECR to our remote server (ec2)

```
- name: Pull Docker image for the frontend from ECR to EC2 instance
hosts: servers
become: yes
vars_files:
  - ansible-vars.yaml

tasks:
  - name: Log in to Amazon ECR
    docker_login:
      registry_url: "{{ ecr_url }}"
      username: AWS
      password: "{{ lookup('pipe', 'aws ecr get-login-password --region ' + region) }}"
      reauthorize: yes

  - name: Pull frontend Docker image from ECR
    docker_image:
      name: "{{ ecr_url }}/{{ ecr_repository }}"
      tag: "{{ frontend_image_tag }}"
      source: pull

  - name: Pull backend Docker image from ECR
    docker_image:
      name: "{{ ecr_url }}/{{ ecr_repository }}"
      tag: "{{ backend_image_tag }}"
      source: pull
```

### Copy code:

```
- name: Pull Docker image for the frontend from ECR to EC2 instance

hosts: servers

become: yes

vars_files:
  - ansible-vars.yaml

tasks:
  - name: Log in to Amazon ECR

    docker_login:

      registry_url: "{{ ecr_url }}"

      username: AWS

      password: "{{ lookup('pipe', 'aws ecr get-login-password --region ' + region) }}"

      reauthorize: yes
```

```

- name: Pull frontend Docker image from ECR

  docker_image:

    name: "{{ ecr_url }}/{{ ecr_repository }}"

    tag: "{{ frontend_image_tag }}"

    source: pull


- name: Pull backend Docker image from ECR

  docker_image:

    name: "{{ ecr_url }}/{{ ecr_repository }}"

    tag: "{{ backend_image_tag }}"

    source: pull


- name: Display Docker images on the host

  command: docker images

  register: docker_images_output


- name: Show Docker images

  debug:

    msg: "{{ docker_images_output.stdout }}"

```

## Updated file ansible-vars.yaml

```

# ecr-vars.yaml

ecr_url: "ecr url"

ecr_repository: "enis-app"

region: "us-east-1"

frontend_image_tag: "frontend-1.0"

backend_image_tag: "backend-1.0"

```

## 7. Start the containers with docker compose

### Updated **docker-compose.yaml** for Server

Here's an updated version of the **docker-compose.yaml** file .

This new configuration is optimized to run on the server, with the **mysql** service removed and the backend's dependency on it eliminated. The updated file references the correct images in Amazon ECR and configures health checks for the **backend-app** and **frontend-app**.

Change the images names based on the images names inside the server pulled from ecr including the tag

```

version: '3.8'

services:

```

```

backend-app:

  image: 640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app:backend-1.0 # Use the already pulled image

  container_name: backend-app

  restart: always

  ports:

    - "8000:8000"

  networks:

    - my_bridge

  healthcheck:

    test: ["CMD-SHELL", "curl -f http://localhost:8000/admin/login/?next=/admin/ || exit 1"]

    interval: 30s

    timeout: 10s

    retries: 5


frontend-app:

  image: 640168430980.dkr.ecr.us-east-1.amazonaws.com/enis-app:frontend-1.0 # Use the already built image

  container_name: frontend-app

  restart: always

  ports:

    - "81:80"

  networks:

    - my_bridge

  healthcheck:

    test: ["CMD-SHELL", "curl -f http://localhost || exit 1"]

    interval: 30s

    timeout: 10s

    retries: 3


networks:

  my_bridge:

    external: true

```

## Explanation of Changes

- Removed MySQL Service: Since the server now connects directly to an external MySQL database, the `mysql` service, its environment variables, and its volume have been removed.

- Backend and Frontend Images: Both **backend-app** and **frontend-app** now use images pulled from ECR, specified with the complete ECR repository path.
- Dependencies:
  - **frontend-app** depends only on **backend-app** and will wait for the backend to be healthy before starting.
- Health Checks:
  - **backend-app**: Checks the Django admin login page to confirm the service is healthy.
  - **frontend-app**: won't check on backend because there are no migrations applied yet so it would report either way unhealthy

**Now we should put that file inside the ansible directory and copy it inside the ec2 instance to be able to execute it there and run the containers , but first create the network my\_bridge**

## Updated playbook : add this set of tasks

```
- name: Start Docker containers

hosts: servers

become: yes

vars_files:

- ansible-vars.yaml

tasks:

- name: Create Docker Network

  docker_network:

    name: "{{ docker_network_name }}"

    state: present

- name: Copy Docker Compose file to the EC2 instance

  copy:

    src: "{{ compose_local_path }}"

    dest: "{{ compose_remote_path }}"

- name: Start Docker containers from Docker Compose

  docker_compose:

    project_src: "{{ project_src_path }}"

    state: present # "present" is equivalent to "docker-compose up", "absent" would stop the containers.

- name: Run makemigrations inside backend container

  command: docker exec {{ backend_container_name }} python manage.py makemigrations

- name: Run migrate inside backend container
```

```
command: docker exec {{ backend_container_name }} python manage.py migrate
```

## Updated file ansible-vars.yaml ( for linux users )

```
# docker variables
```

```
docker_network_name: "my_bridge"
```

```
compose_local_path: "/home/imen/tp enis/ansible/docker-compose.yaml" #replace with your local path of the updated file
```

```
compose_remote_path: "/home/ubuntu/docker-compose.yaml"
```

```
project_src_path: "/home/ubuntu"
```

```
backend_container_name: "backend-app"
```

## For windows users who are using ansible container ;

the compose local path should be the path inside the container which is normally : /workspace/docker-compose.yaml

## Updated file ansible-vars.yaml ( for windows users )

```
# docker variables
```

```
docker_network_name: "my_bridge"
```

```
compose_local_path: "/workspace/docker-compose.yaml" #replace with your local path of the updated file
```

```
compose_remote_path: "/home/ubuntu/docker-compose.yaml"
```

```
project_src_path: "/home/ubuntu"
```

```
backend_container_name: "backend-app"
```

## Explanation:

- Name: Copy docker compose to my ec2 instance  
This task copies the `docker-compose.yaml` file from your local machine (where you're running Ansible) to the target EC2 instance. Make sure the local path to the file is correct in the variables file.
- `copy` Module:
  - `src`: Specifies the source path on your local machine where the `docker-compose.yaml` file is located.
  - `dest`: Specifies the destination path on the EC2 instance where the file will be copied. Here, it is copied to `/home/ubuntu`, which is a directory on the target EC2 instance.
  - The `docker-compose.yaml` file defines the services, networks, and volumes for your multi-container Docker application. It needs to be present on the EC2 instance to run `docker-compose` and start the containers.

- This step ensures that the configuration file is available on the EC2 instance before executing the `docker-compose up` command.
- Name: `start docker containers from compose`  
This task runs `docker-compose up` on the EC2 instance to start the Docker containers defined in the `docker-compose.yaml` file.
- `docker_compose` Module:
  - `project_src`: Specifies the directory where the `docker-compose.yaml` file is located on the EC2 instance. In this case, it's `/home/ubuntu`, where the file was copied in the previous step.
  - `state`:
    - `present`: This is equivalent to running `docker-compose up`. It starts the containers defined in the `docker-compose.yaml` file. If the containers are already running, they will be left as is.
    - `absent`: If used instead, it would stop and remove the containers (equivalent to `docker-compose down`).

### Run makemigrations and migrate inside backend container

Our backend container is set up to connect to the RDS instance, but the necessary database tables aren't automatically generated. After the backend container starts up via Docker Compose, we need to enter the container and run `makemigrations` and `migrate`, just as we did manually in the Docker session. This will create the tables in the RDS database. Once the migrations are applied, we can create a superuser, allowing us to log in to the hosted app with admin access.

To create a superuser in Django within the backend container, you can add a task that runs a Django management command for superuser creation. This command will use environment variables for the username, email, and password to automate the process.

Here's how you can add this task to your playbook:

1. Update `ansible-vars.yaml`: Define environment variables for the superuser's details in your variables file.

```
# app credentials

superuser_username: "admin"

superuser_email: "admin@example.com"

superuser_password: "SuperSecretPassword"
```

Add the Superuser Creation Task: In your playbook, add a task to run the Django command for superuser creation. This uses a one-time command to create the superuser if it doesn't already exist.

- name: Create superuser inside backend container

command: >

```
docker exec {{ backend_container_name }} python manage.py shell -c

"from django.contrib.auth import get_user_model;

User = get_user_model();

User.objects.filter(username='{{ superuser_username }}').exists() or

User.objects.create_superuser('{{ superuser_username }}', '{{ superuser_email }}', '{{ superuser_password
}}')"
```

**Finally run the playbook and try to access the app on <http://<your-ec2-instance-public-ip>:81> and try to connect with the credentials set in the ansible-vars file**

The image shows two screenshots of a web browser. The first screenshot displays a 'Login' page with a username field containing 'admin', a password field with masked characters, a loading spinner, and a blue 'Login' button. The browser's address bar shows '34.229.247.74/81/login'. The second screenshot shows a 'Notes' page. It lists a 'first note' with the content 'App is working just perfect' and a 'Delete' button. Below this is a 'Create a Note' form with fields for 'Title' (containing 'first note') and 'Content' (containing 'App is working just perfect'), and a blue 'Submit' button. The browser's address bar shows '34.229.247.74/81'.

After you've ensured that the application is working perfectly and all migrations are applied successfully, it's essential to clean up any resources you've provisioned to avoid unnecessary costs.

1. Destroy Terraform Resources: Run the following command to delete the infrastructure created by Terraform: `terraform destroy`
2. Terminate RDS Instances (if RDS wasn't managed by Terraform): If your RDS instance was created outside of Terraform, you'll need to terminate it separately:
  - Go to the AWS RDS Console.
  - Select your RDS instance and choose Actions > Delete.
  - be sure to **uncheck** the option to create a final snapshot before deletion. This avoids additional storage costs for the snapshot.

The screenshot shows a modal dialog titled "Delete mydb instance" with a close button (X). The dialog contains the following elements:

- A warning message: "Permanently delete **mydb** DB instance. You can't undo this action."
- A yellow warning box: "Proceeding with this action will delete the instance with all its content and can affect related resources. [Learn more](#)"
- Two unchecked checkboxes:
  - ☐ Create final snapshot: Determines whether a final DB Snapshot is created before the DB instance is deleted.
  - ☐ Retain automated backups: Determines whether retaining automated backups for 1 day after deletion
- A checked checkbox: ☒ I acknowledge that upon instance deletion, automated backups, including system snapshots and point-in-time recovery, will no longer be available.
- A text prompt: "To avoid accidental deletion provide additional written consent. To confirm deletion, type *delete me* into the field."
- A text input field containing the text "delete me".
- A yellow warning box: "We strongly recommend taking a final snapshot before instance deletion since after your instance is deleted, automated backups will no longer be available."
- At the bottom, there are "Cancel" and "Delete" buttons.

The screenshot shows the "Databases" page in the AWS RDS console. At the top, there is a notification banner about Blue/Green Deployment. Below the banner, there are tabs for "Group resources", "Modify", "Actions", "Restore from S3", and "Create database". A search bar labeled "Filter by databases" is present. Below the search bar is a table with the following columns: DB identifier, Status, Role, Engine, Region &..., Size, Recommendations, CPU, Current a..., and Maintena... The table contains one row for the instance "mydb", which is in a "Deleting" status. The instance details are: Instance, MySQL Com..., us-east-1a, db.t3.micro, -, and none.

DB identifier	Status	Role	Engine	Region &...	Size	Recommendations	CPU	Current a...	Maintena...
mydb	Deleting	Instance	MySQL Com...	us-east-1a	db.t3.micro	-	-	-	none