**Custom Bridge**

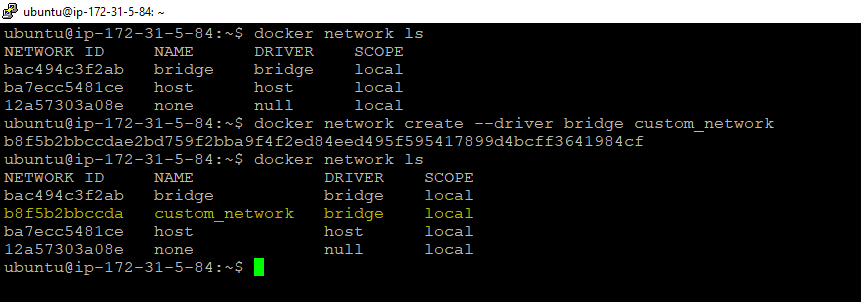
In Docker, a custom bridge network is a user-defined network that allows containers to communicate with each other while remaining isolated from external networks. By default, when you run a container, Docker places it on a bridge network, which is a private internal network that facilitates communication between containers on the same host. However, you can create custom bridge networks with specific configurations to suit your application's needs.

Here's how you can create a custom bridge network:

Command :- **$docker network create --driver bridge custom\_network**

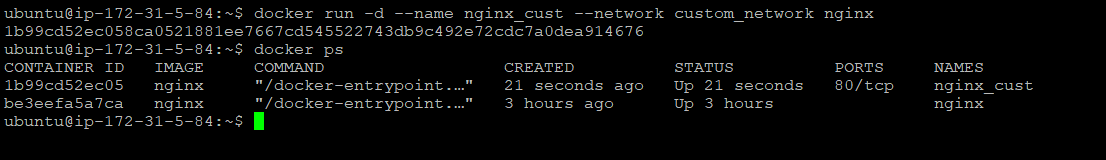
In this example, custom\_network is the name of the custom bridge network. You can replace it with any name you prefer.

Let's explore:-

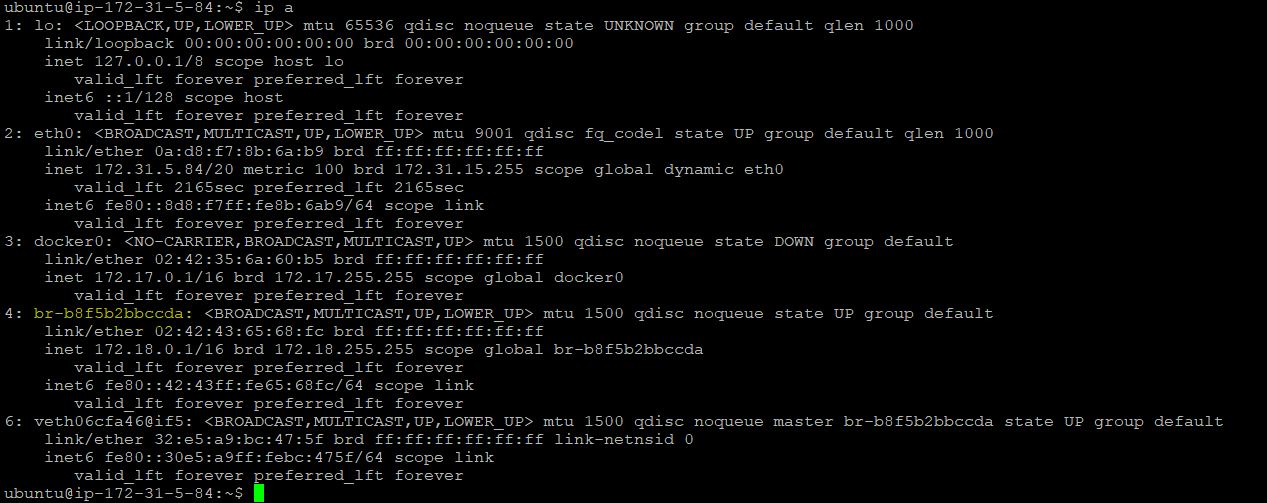


We have created a custom bridge network named **custom\_network**

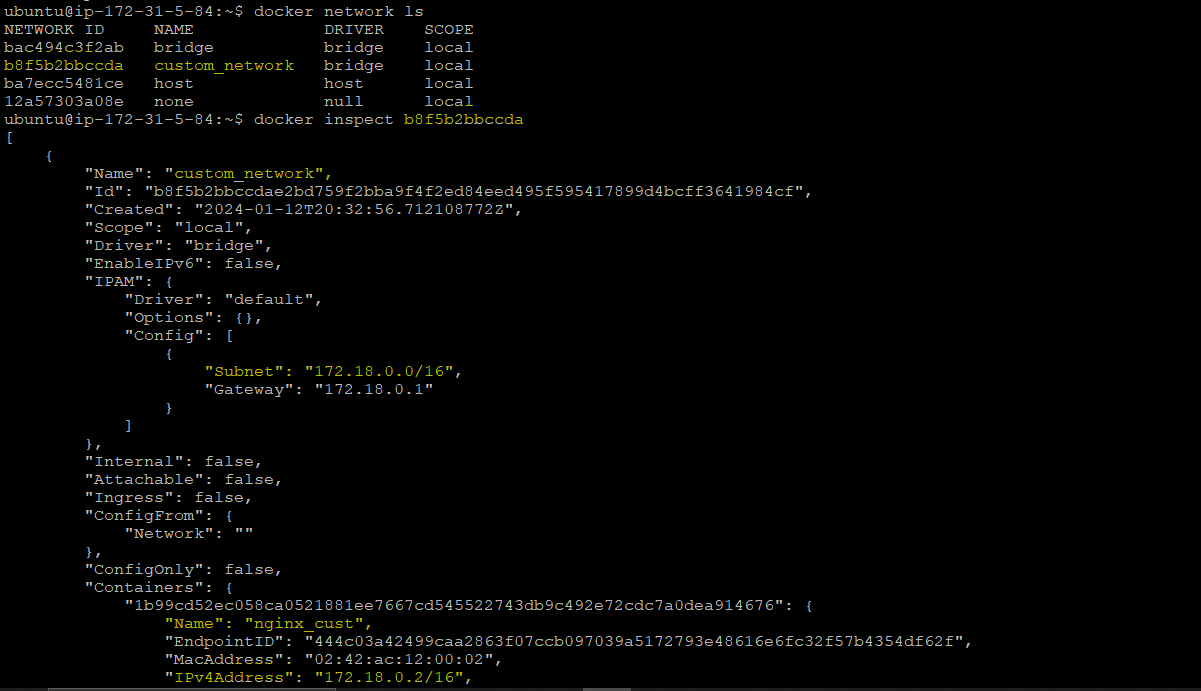
Let’s create a container using this custom bridge network and see.



We can see there are 2 containers running but both are using different network. They should not be able to communicate each other.



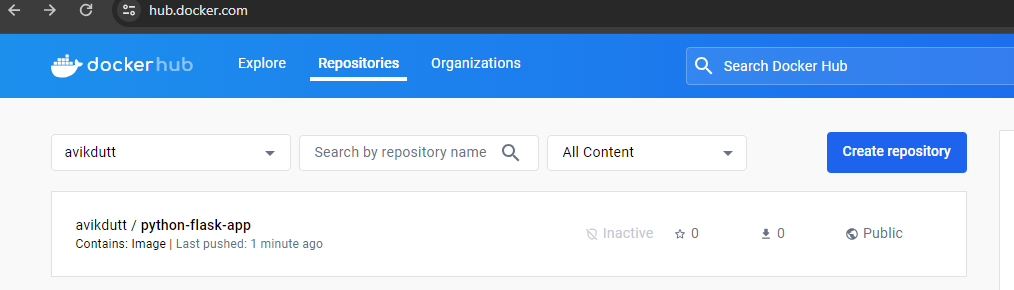
**4: br-b8f5b2bbccda** -> this is our custom bridge network. Lets inspect and see the ip of the nginx\_cust container to validate the same.



Now let’s create a project which has front end and back end db.

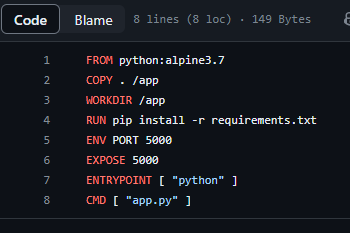
* First we will create the front end docker container

I will use one docker image which is already present in my dockerHub.



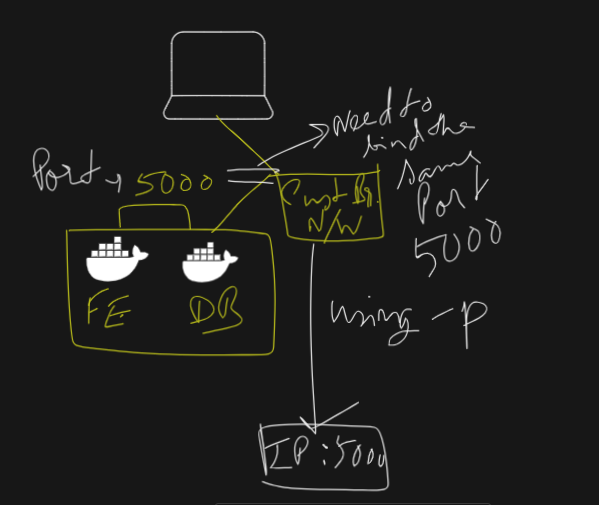
GitHub link for code : <https://github.com/avikdutt/microservices-k8s.git>

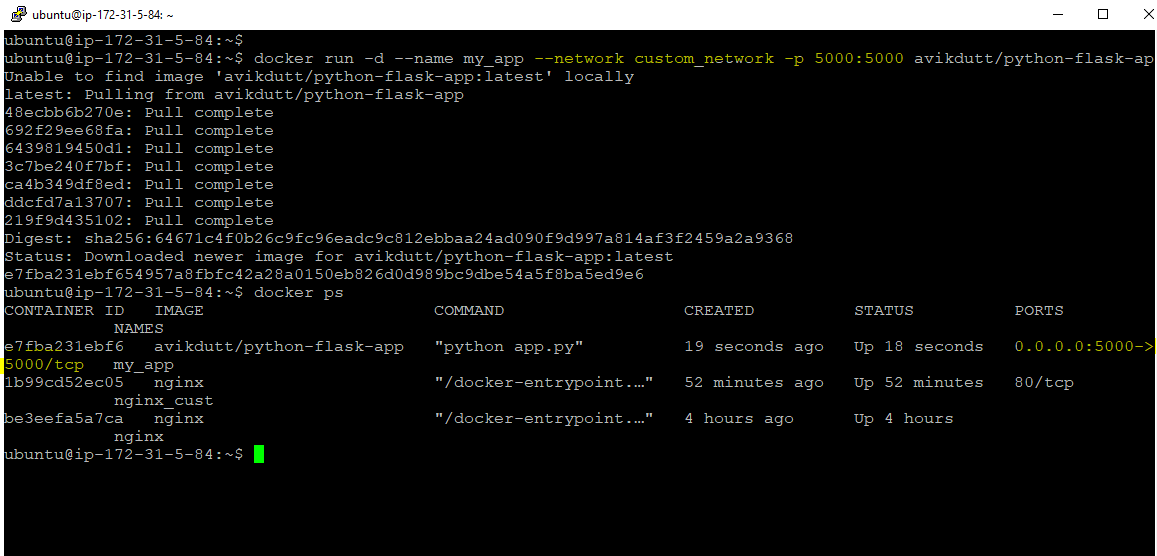
Let’s create the container remotely in custom bridge network. The app needs port 5000 to run. So we will have to bind the port 5000 while creating and running our front end container.



Binding port 5000 means:- our front end docker container’s exposed port is 5000, in order to run match this with our network bridge we have to make them bind in same port.

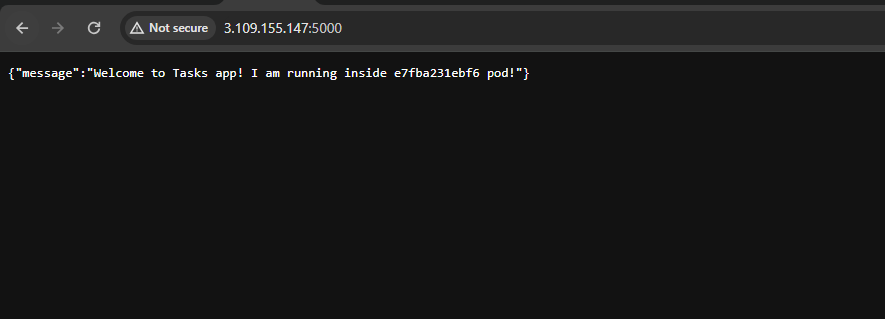
Diagram:-





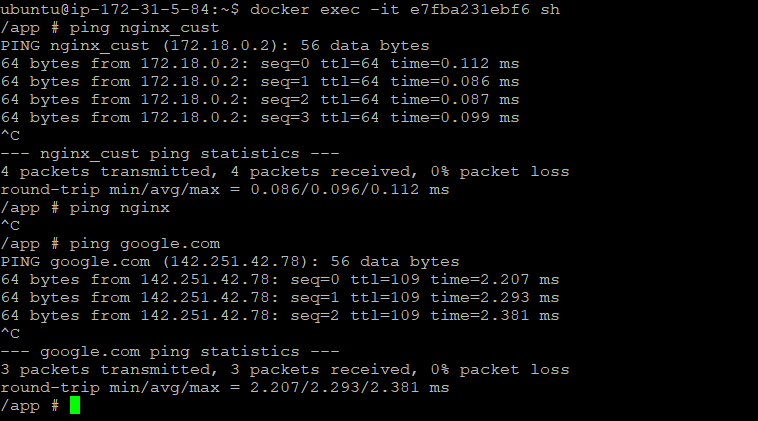
Command:- **$docker run -d --name my\_app --network custom\_network -p 5000:5000 avikdutt/python-flask-ap**

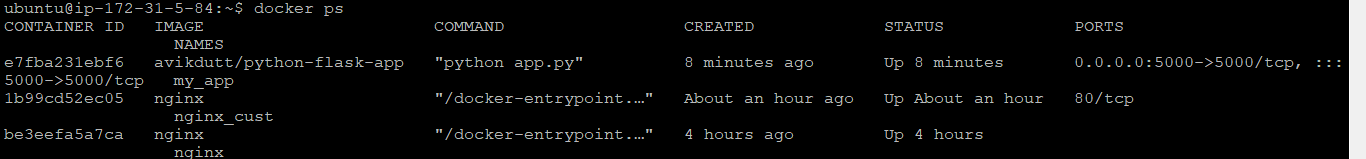
Our App is running in port 5000



Now we have total 3 containers. 2 nginx and one my\_app. 1 nginx container is in same network with my\_app which is in our custom bridge name custom\_network. Another nginx container is in host network.

As mentioned these 2 container (my\_app and nginx\_cust ) should communicate to each other but they won’t be able to communicate to another nginx container which is not in this custom bridge. Let’s see in practically:-





We can see my\_app container has access to internet as well. They can talk to each other but can’t talk to another nginx container.

As we mentioned our app is 2 tier, we need one db container to be running, in order to make is functional and talk to our front end it should be created inside this custom\_netwok.

You can try by yourself creating the db and let me know.

Hint:- the docker image name is mongo which you can get remotely, it is in public. No need any specific repo.

---------------------------------------------------------------------------------------------------------------

Let's explore some key aspects of custom bridge networks in Docker:

1. Isolation:

Custom bridge networks provide network isolation for containers. Containers connected to the same custom bridge network can communicate with each other, but they are isolated from containers on other networks and the host itself.

2. User-Defined Networks:

When you create a custom bridge network, you have the ability to specify certain parameters, such as the subnet and gateway. This allows you to have more control over the network configuration.

*docker network create --driver bridge --subnet 172.18.0.0/16 --gateway 172.18.0.1 custom\_network*

In this example, the --subnet and --gateway options are used to define a specific IP range for the network.

3. Container Connectivity:

Containers on the same custom bridge network can communicate with each other using their container names or IP addresses. This makes it easy to set up communication between containers in an application.

4. DNS Resolution:

Containers connected to a custom bridge network can resolve each other's hostnames using Docker's built-in DNS resolution. This simplifies communication between containers using service names rather than IP addresses.

5. Service Discovery:

Custom bridge networks provide a form of service discovery within the network. Containers can discover and communicate with other containers on the same network using the container name as a hostname.

Use Case Examples:

* Microservices Architecture:

Custom bridge networks are often used in microservices architectures, where different services are deployed in separate containers and need to communicate with each other.

* Application Stacks:

When deploying multi-container applications with interconnected components, custom bridge networks help organize communication and maintain isolation.

* Testing and Development:

Custom bridge networks are useful for creating isolated environments for testing and development, allowing developers to simulate network interactions between containers.

Remember to manage the lifecycle of your custom bridge networks appropriately, and consider the security and scalability requirements of your application when choosing the network configuration.