Summary - Muhamad Affan

CRUD Application

For the CRUD application, i'm choosing Python with Flask framework for the backend and postgresql as the database.

The data that will be used is an `employee` table with the following schema:

```
id           : UUID
name         : String
age          : Integer
email        : String
created_at   : Timestamp
updated_at   : Timestamp
```

Documentation

Create

- URL : `/employees/add`
- Description : Create a new employee entry
- Method : `POST`
- Request Body :

  ```
  {
    "name": String [Required],
    "age": Integer [Required],
    "email": String [Required]
  }
  ```

  Example:

  ```
  {
    "name": "affan",
    "age": 25,
    "email": "foo+1@bar.com"
  }
  ```

- Response

  ```
  {
    "employee": {
  ```

```
        "age": Integer,
        "created_at": Timestamp,
        "email": String,
        "id": UUID,
        "name": String,
        "updated_at": Timestamp
    },
    "message": String
  }
```

Example:

```
  {
    "employee": {
        "age": "25",
        "created_at": "2025-05-20 22:22:46.243212",
        "email": "foo+1@bar.com",
        "id": "14538b70-78ec-4d82-9cda-41deed583f60",
        "name": "affan",
        "updated_at": "2025-05-20 22:22:46.243212"
    },
    "message": "success"
  }
```

Read

- URL : `/employees/get`
- Description : Get all employee data
- Method : `GET`
- Response

```
  {
    "employees": [
        {
          "age": Integer,
          "created_at": Timestamp,
          "email": String,
          "id": UUID,
          "name": String,
          "updated_at": Timestamp
        },
        {
          "age": Integer,
          "created_at": Timestamp,
          "email": String,
          "id": UUID,
          "name": String,
          "updated_at": Timestamp
        },
        .
        .
        .
    ],
    "message": String
  }
```

Example :

```
{
  "employees": [
      {
          "age": "25",
          "created_at": "2025-05-15 01:41:49.399918",
          "email": "foo+2@bar.com",
          "id": "a7ed1626-046c-4e4e-b88b-e675a076a224",
          "name": "affan2",
          "updated_at": "2025-05-15 01:41:49.399918"
      },
      {
          "age": "25",
          "created_at": "2025-05-16 02:49:54.586036",
          "email": "foo+3@bar.com",
          "id": "5e796c6c-f96c-4a93-9582-b63a96fbb2d1",
          "name": "affan3",
          "updated_at": "2025-05-16 02:49:54.586036"
      }
  ]
  "message": "success"
}
```

Update

- URL : `/employees/update/<email>`
- Description : Update employee data based on `<email>` that is given
- Method : `PATCH`
- Parameters:
    - `email` : String
- Request Body:

```
```

```
{
  "name": String [Optional],
  "age": Integer [Optional],
  "email": String [Optional]
}
```

> Note : despite all of it being optionals, at least one of the keys needs to be present in the body

Example: URL : `/employees/update/foo@bar.com`

```
{
  "name": "affan-test"
}
```

- Response

```
{
  "employee": {
      "age": Integer,
      "created_at": Timestamp,
      "email": String,
      "id": UUID,
      "name": String,
      "updated_at": Timestamp
  },
  "message": String
}
```

Example:

```
{
  "employee": {
      "age": "25",
      "created_at": "2025-05-15 01:41:35.721049",
      "email": "foo@bar.com",
      "id": "710972a0-a2b5-4c0e-8a24-8f120bc5957a",
      "name": "affan-test",
      "updated_at": "2025-05-20 22:29:27.659368"
  },
  "message": "success"
}
```

Delete

- URL : `/employees/delete/<email>`

- Description : Delete an employee entry that has email based on `<email>` parameter.
- Method : `DELETE`
- Parameters :
    - `email` : String
- Response : Response Code `204`

Deployment

For the deployment, i am using digitalocean cloud provider. The app can be acces through this address : http://170.64.206.172/

I am using the provider's VM product (droplet) with the following stack:

- App containerization using docker (will be explained further in section below)
- NGINX server as a reverse-proxy
- Postgresql database installed in the VM

Containerization

Here's the Dockerfile to create an image for the apps with explanation for each line

```
# Using existing python 3.9 as the base image
FROM python:3.9-slim-buster

# Installing dependency package for SQLAlchemy and psycopg2 as DB Driver
RUN apt-get update \
    && apt-get -y install libpq-dev gcc

# Exposing port that will be used
EXPOSE 5000

# Copying requirements file to the image
COPY requirements.txt .
# Installing the requirements
RUN python -m pip install -r requirements.txt

# Move to /app directory
WORKDIR /app
# Copying codebase to the /app directory
COPY . /app

# running the application
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]
```

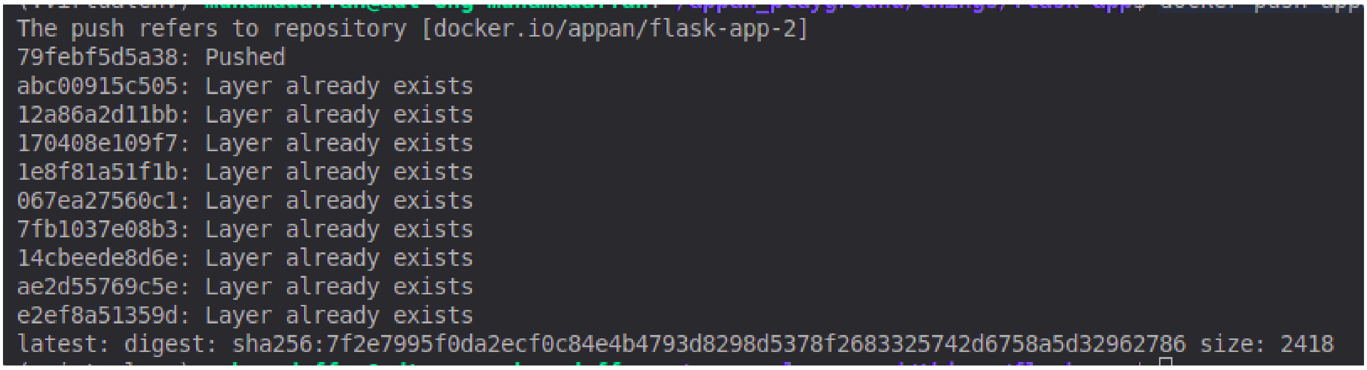
The image will be pushed to container registry in dockerhub with tag `appan/flask-app-2:latest`

Pushing image to registry

Command :

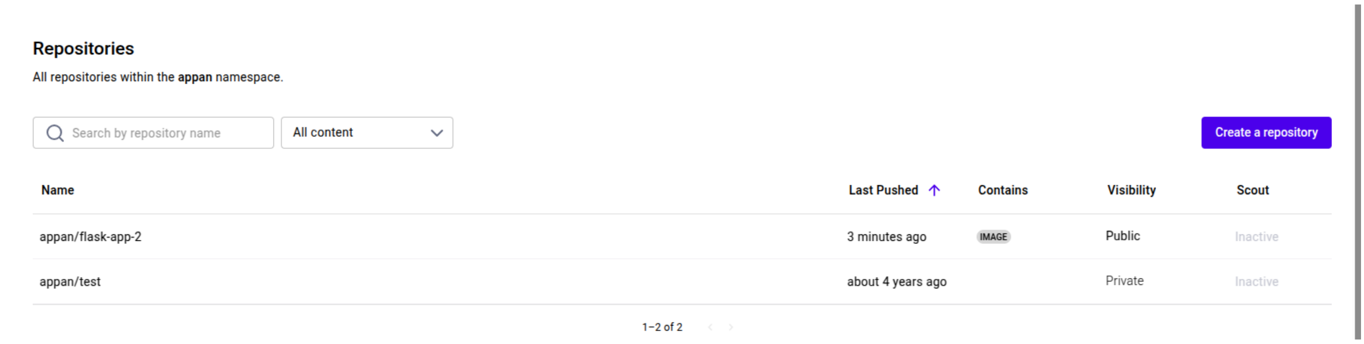
```
docker push appan/flask-app-2:latest
```

Result :



```
The push refers to repository [docker.io/appan/flask-app-2]
79febf5d5a38: Pushed
abc00915c505: Layer already exists
12a86a2d11bb: Layer already exists
170408e109f7: Layer already exists
1e8f81a51f1b: Layer already exists
067ea27560c1: Layer already exists
7fb1037e08b3: Layer already exists
14cbeede8d6e: Layer already exists
ae2d55769c5e: Layer already exists
e2ef8a51359d: Layer already exists
latest: digest: sha256:7f2e7995f0da2ecf0c84e4b4793d8298d5378f2683325742d6758a5d32962786 size: 2418
```

Result on dockerhub :



Repositories

All repositories within the **appan** namespace.

Name	Last Pushed ↑	Contains	Visibility	Scout
appan/flask-app-2	3 minutes ago	IMAGE	Public	Inactive
appan/test	about 4 years ago		Private	Inactive

1–2 of 2

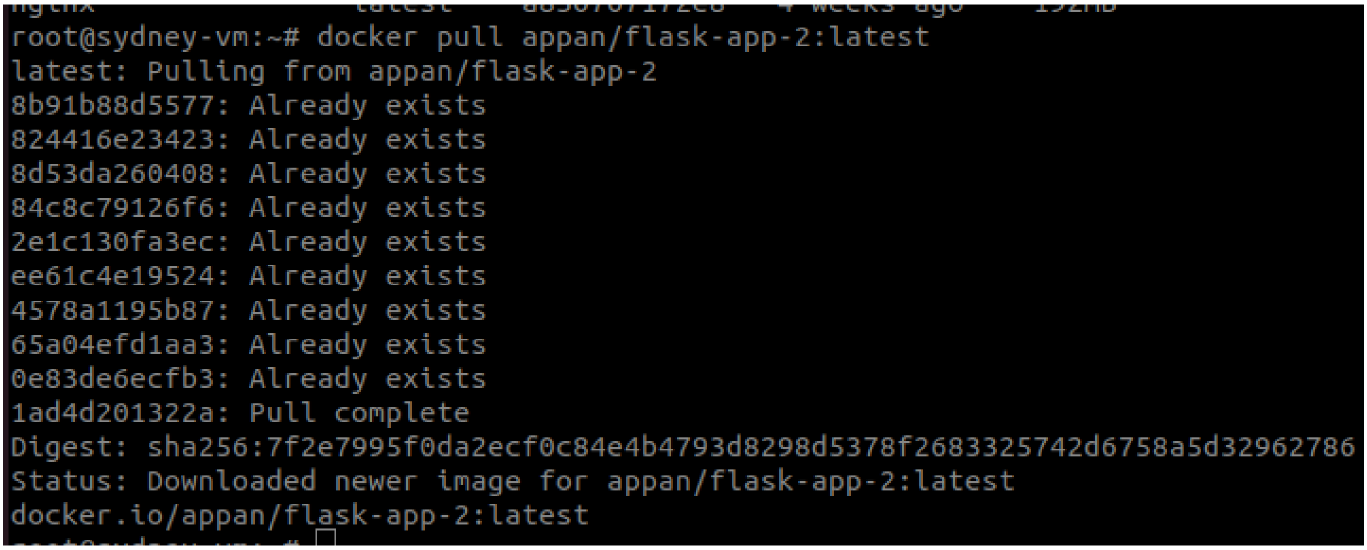
Pulling image from registry

To pull the image, i will be pulling it to my digitalocean VM

Command :

```
docker pull appan/flask-app-2:latest
```

Result :



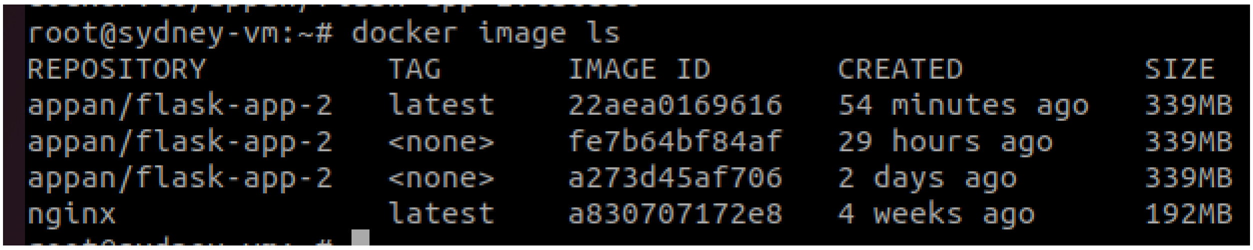
```
root@sydney-vm:~# docker pull appan/flask-app-2:latest
latest: Pulling from appan/flask-app-2
8b91b88d5577: Already exists
824416e23423: Already exists
8d53da260408: Already exists
84c8c79126f6: Already exists
2e1c130fa3ec: Already exists
ee61c4e19524: Already exists
4578a1195b87: Already exists
65a04efd1aa3: Already exists
0e83de6ecfb3: Already exists
1ad4d201322a: Pull complete
Digest: sha256:7f2e7995f0da2ecf0c84e4b4793d8298d5378f2683325742d6758a5d32962786
Status: Downloaded newer image for appan/flask-app-2:latest
docker.io/appan/flask-app-2:latest
```

Listing image

Command :

```
docker image ls
```

Result :



```
root@sydney-vm:~# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
appan/flask-app-2   latest    22aea0169616   54 minutes ago   339MB
appan/flask-app-2   <none>    fe7b64bf84af   29 hours ago     339MB
appan/flask-app-2   <none>    a273d45af706   2 days ago       339MB
nginx               latest    a830707172e8   4 weeks ago      192MB
```

Run and list container

Run container command :

```
docker run -p 5000:5000 -d --env-file .env --network=host --name flask-app
appan/flask-app-2:latest
```

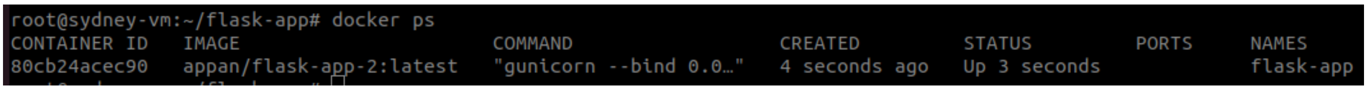
flags :

```
-p 5000:5000 : publish container port 5000 to host port 5000
-d : run the container in detached mode
--env-file .env : export .env file to the container
--network=host : using network mode "host" to enable accessing local
postgres database
```

List container command :

```
docker ps
```

Result :



```
root@sydney-vm:~/flask-app# docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS         PORTS     NAMES
80cb24acec90   appan/flask-app-2:latest   "gunicorn --bind 0.0…"   4 seconds ago    Up 3 seconds             flask-app
```

Kubernetes

Deployment

YAML deployment file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app-2
spec:
  selector:
    matchLabels:
      app: flask-app-2
  template:
    metadata:
      labels:
        app: flask-app-2
    spec:
      containers:
      - name: flask-app-2
        image: appan/flask-app-2
        resources:
          requests:
            memory: "64Mi"
            cpu: "100m"
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 5000
        env:
        - name: DB_USERNAME
          value: dev
        - name: DB_PASSWORD
          value: dev
        - name: DB_URL
          value: host.minikube.internal
        - name: DB_NAME
          value: flask_db
```

- `metadata.name`: The name of the deployment

- `spec.selector.matchLabels`: The labels selector for the deployment
- `template.metadata.labels` : The labels of the deployment
- `spec.containers` :
  - `name` : The name of the container that will be deployed in the pods
  - `image` : The name of the image that will be used to create the container
  - `resources.requests.memory` : The provisioned memory for each pod
  - `resources.requests.cpu` : The provisioned cpu for each pod
  - `resource.limits.memory` : The limit of memory that each pod can use
  - `resource.limits.cpu` : The limit of cpu that each pod can use
  - `ports.containerPort` : The ports that will be published by the container
  - `env.*` : The environment variables that will be used inside the pods

Accessing App From Outside The Cluster

To expose kubernetes pods over a network, we can use kubernetes object called `Service`. Kubernetes provides 4 types of Services which are :

- `ClusterIP`

  This service exposes pods over internal cluster IP which will make pods accesible through the cluster's internal network. Since it only expose on the cluster's internal, we can't access it from outside the cluster

- `NodePort`

  Expose a static port from the node in the cluster so that it can be reachable from network outside the cluster. With this service, we can access the pods through the node's IP address and the nodeport that we chose.

- `LoadBalancer`

  Expose the pods using an external loadbalancer. The loadbalancer will forward the request to the cluster's node. But since kubernetes does not support a loadbalancer component, we have to use a third-party provider for this service.

- `ExternalName`

  Creates a mapping from the service to an external DNS hostname.

For this project, I will be using `NodePort` service to expose the deployment and make it accessible from outside the cluster.

Service YAML file

```yaml
apiVersion: v1
kind: Service
metadata:
  name: flask-svc
spec:
  selector:
    app: flask-app-2
  type: NodePort
  ports:
  - port: 5000
    targetPort: 5000
    nodePort: 30000
```

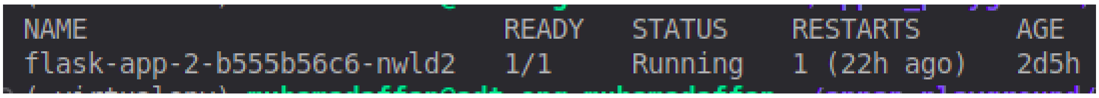
The service will expose node `30000` on the cluster and maps it towards port `5000` of the deployment's pods

List Pods

Command :

```
kubectl get pods
```

Result :



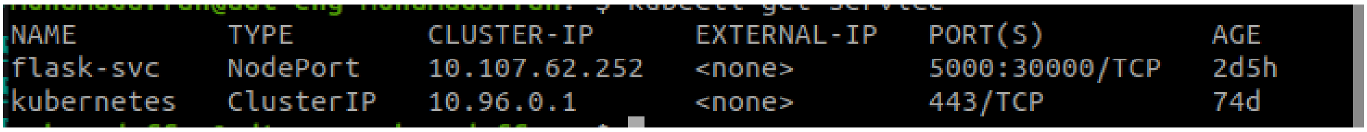
```
NAME                          READY   STATUS    RESTARTS      AGE
flask-app-2-b555b56c6-nwld2   1/1     Running   1 (22h ago)   2d5h
```

List Services

Command :

```
kubectl get service
```

Result :



```
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
flask-svc    NodePort    10.107.62.252   <none>        5000:30000/TCP   2d5h
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          74d
```

Pods vs Deployments

Pods is a smallest unit of work in kubernetes. It encapsulates one or more application's containers. The containers inside a pod shares resources such as storage and IP address. The containers can comunicate each other through localhost.

Deployments is a kubernetes object with the purpose to manage a set of pods. Deployments offer additional functionalities to manage pods such as :

- Self Healing

  Allowing pods to automatically replace crashed pods so it will match the desired number of replicas that's already been declared

- Scaling

  Adding / Reducing number of pods replicas

- Updates

  Updating all the pods replicas using update strategy such as `RollingUpdate` or `Recreate`

- Rollback

  Rolling back the pods the the previous version

Ansible

Pros and Cons

Pros

- Simple

  Ansible notebook implementation that is using YAML which have a rather declarative nature makes it easier to understand, more readable, and more simple to write for a sets of basic tasks.

- Agentless

  Ansible's push-based design through SSH means that it can easily run without having to actually install it on the target machines. This simplify running the notebook to any given machines without having to set it up first

- Security

  By communicating through SSH encryption, ansible offers an extended security in remote machines management.

- Flexible

  Ansible's flexibility to integrate with various system and platforms such as cloud providers (GCP, Azure, AWS etc.), containerization (Docker, Kubernetes etc.) Makes it reliable to use in various environments and use cases

- Scalability

  Ansible can handle managing execution on multiple machines, although it came with the downside of limiting the performances as the number of machine scales.

Cons

- Stateless

  Ansible's stateless nature which doesn't track the state beyond executing the described tasks makes it virtually difficult to track state of the machine/system that we are going to manage.

- SSH Overhead

Communicating over SSH might add an additional overhead latency that can slower the process compared to running using an agent-based tools.

- Performance

  Despite its ability to scale and manage multiple systems, it came with a catch that the growing number of systems being managed can take a toll towards its performance due to lack of the built-in state tracking and SSH connection overhead

- Complexity

  Although ansible is great for running sets of basic tasks, it might add more layer of complexity when the tasks being executed grow more and more complex.

- Limited Support on Windows

  Ansible has a limited support on windows compared to Unix/Linux which makes it more difficult to manage system if we are using windows.

Running Playbook

Here's the playbook YAML to install docker and run the container

```
- hosts: localhost
  become: true

  tasks:
    - name: Install packages
      apt:
        pkg:
          - apt-transport-https
          - ca-certificates
          - curl
          - software-properties-common

    - name: Add Docker GPG Key
      apt_key:
        url: https://download.docker.com/linux/ubuntu/gpg
        state: present

    - name: Add Docker Repository
      apt_repository:
        repo: deb https://download.docker.com/linux/ubuntu jammy stable
        state: present

    - name: Install Docker
      apt:
        pkg: docker-ce
        state: latest
        update_cache: true

    - name: Pull Image
```

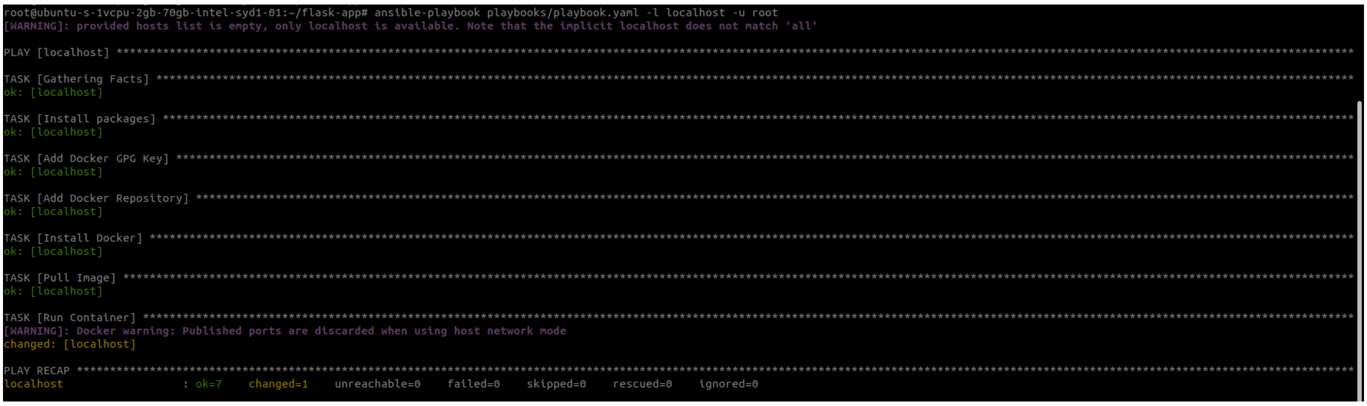
```
      community.docker.docker_image:
        name: "appan/flask-app-2:latest"
        source: pull

    - name: Run Container
      community.docker.docker_container:
        name: "flask-app"
        image: "appan/flask-app-2:latest"
        ports:
          - 5000:5000
        detach: true
        env_file: ../.env
        network_mode: host
```

To test it, I will be executing the playbook on an empty VM that still don't have docker installed

Executing the playbook

```
ansible-playbook playbooks/playbook.yaml -l localhost -u root
```

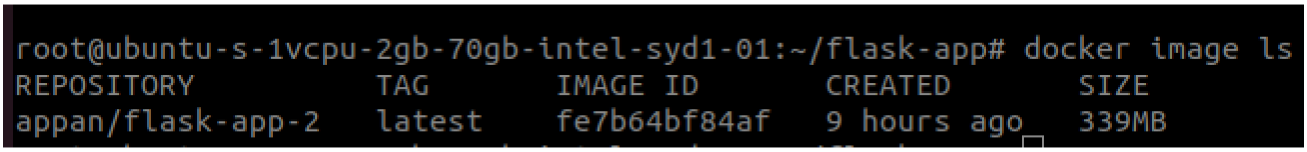
Result:



```
root@ubuntu-s-1vcpu-2gb-70gb-intel-syd1-01:~/flask-app# ansible-playbook playbooks/playbook.yaml -l localhost -u root
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [localhost] ************************************************************************************************************************

TASK [Gathering Facts] ******************************************************************************************************************
ok: [localhost]

TASK [Install packages] *****************************************************************************************************************
ok: [localhost]

TASK [Add Docker GPG Key] ***************************************************************************************************************
ok: [localhost]

TASK [Add Docker Repository] ************************************************************************************************************
ok: [localhost]

TASK [Install Docker] *******************************************************************************************************************
ok: [localhost]

TASK [Pull Image] ***********************************************************************************************************************
ok: [localhost]

TASK [Run Container] ********************************************************************************************************************
[WARNING]: Docker warning: Published ports are discarded when using host network mode
changed: [localhost]

PLAY RECAP ******************************************************************************************************************************
localhost                  : ok=7    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Checking the image

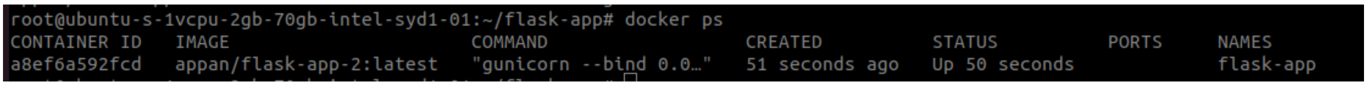
```
docker image ls
```



```
root@ubuntu-s-1vcpu-2gb-70gb-intel-syd1-01:~/flask-app# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED       SIZE
appan/flask-app-2   latest    fe7b64bf84af   9 hours ago   339MB
```

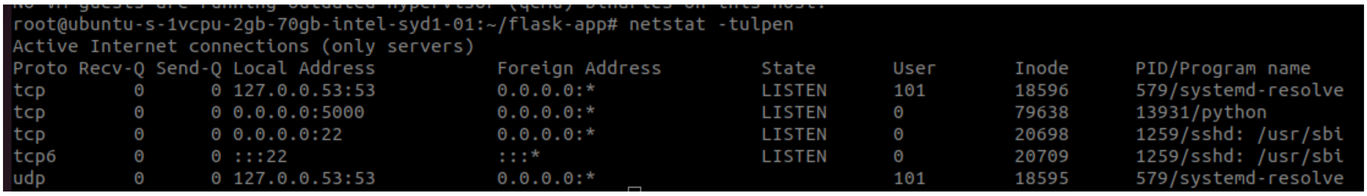
Ensuring the container is running

Container is running



```
root@ubuntu-s-1vcpu-2gb-70gb-intel-syd1-01:~/flask-app# docker ps
CONTAINER ID   IMAGE                         COMMAND                  CREATED          STATUS          PORTS     NAMES
a8ef6a592fcd   appan/flask-app-2:latest      "gunicorn --bind 0.0…"   51 seconds ago   Up 50 seconds             flask-app
```

Listening on port `5000`



```
root@ubuntu-s-1vcpu-2gb-70gb-intel-syd1-01:~/flask-app# netstat -tulpen
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       User       Inode      PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      101        18596      579/systemd-resolve
tcp        0      0 0.0.0.0:5000            0.0.0.0:*               LISTEN      0          79638      13931/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      0          20698      1259/sshd: /usr/sbi
tcp6       0      0 :::22                   :::*                    LISTEN      0          20709      1259/sshd: /usr/sbi
udp        0      0 127.0.0.53:53           0.0.0.0:*                           101        18595      579/systemd-resolve
```