#90DaysOfDevOps Challenge - Day 16 - Docker for DevOps Engineers

Welcome to Day 16 of the #90DaysOfDevOps challenge. Today, we will dive into Docker, an essential tool for DevOps Engineers. Docker revolutionizes the way we build, ship, and run applications by providing a lightweight and portable containerization platform. By the end of this day, you will have a better understanding of Docker's fundamentals and its immense value in the DevOps world.

Why Docker is Important for DevOps Engineers

Docker has become a game-changer in the field of **DevOps**, revolutionizing the way applications are deployed, managed, and shipped. As a DevOps engineer, understanding Docker and its significance is crucial for **streamlining software delivery**, **improving scalability**, and ensuring consistent environments across various stages of the development and deployment lifecycle.

1. **Consistent Environments:** Docker allows DevOps engineers to create containerized environments that encapsulate the application and its dependencies. With Docker, you can ensure that your application runs consistently across different environments, including development, testing, staging, and production. This eliminates the notorious "it works on my machine" problem and provides a consistent foundation for collaboration between developers and operations teams.

2. **Scalability and Resource Efficiency:** Docker's containerization approach enables efficient resource utilization and scalability. By packaging applications into lightweight containers, you can deploy and scale them quickly, taking advantage of Docker's ability to spin up multiple containers on a single host machine. This scalability is essential for handling high traffic loads, optimizing infrastructure usage, and achieving cost-effectiveness.

3. **Rapid Deployment:** Docker simplifies the deployment process by packaging the application and its dependencies into a container. This container can then be easily deployed across different environments without worrying about compatibility issues. DevOps engineers can use Docker to automate the deployment process, enabling fast and reliable application delivery.

4. **Isolation and Security:** Docker provides isolated environments for applications, ensuring that they do not interfere with each other. Each container operates independently, creating a sandboxed environment that enhances security and reduces the risk of conflicts between applications or dependencies. Additionally, Docker's built-in security features, such as image scanning and access control, help DevOps engineers maintain a secure software supply chain.

5. **Continuous Integration and Delivery (CI/CD):** Docker plays a vital role in implementing CI/CD pipelines. With Docker, you can build container images that encapsulate the application and its dependencies. These images can then be tested, deployed, and rolled back easily, ensuring a smooth and automated CI/CD workflow. Docker integrates seamlessly with popular CI/CD tools, enabling efficient automation and orchestration of the software delivery process.

Docker has revolutionized the DevOps landscape by providing a powerful tool for containerization and application deployment. DevOps engineers can leverage Docker to achieve consistent environments, efficient resource utilization, rapid deployment, enhanced security, and streamlined CI/CD pipelines. By

mastering Docker, DevOps engineers can unlock the potential for scalable, reliable, and agile software delivery, empowering organizations to thrive in today's fast-paced digital landscape.

Tasks

1 - Using the `docker run` command

Start a new container and interact with it through the command line.

For example, you can run `docker run hello-world` to run a simple "Hello, World!" container and verify your Docker installation.

```
esteban@ubuntu-server-template:~$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

2 - Using the `docker inspect` command

View detailed information about a container or image. This command provides insights into the configuration, networking, and runtime details of a Docker object.

For instance, `docker inspect container_id` will display detailed information about a specific container.

```
esteban@ubuntu-server-template:~$ docker inspect 15fbaaa69b342541115813cf0e9988aa74715d5220b8c80c00bc2f0514609a3b
[
    {
        "Id": "15fbaaa69b342541115813cf0e9988aa74715d5220b8c80c00bc2f0514609a3b",
        "Created": "2023-06-06T18:44:48.469228822",
        "Path": "/hello",
        "Args": [],
        "State": {
            "Status": "exited",
            "Running": false,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 0,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2023-06-06T18:44:48.769677366Z",
            "FinishedAt": "2023-06-06T18:44:48.768868902Z"
        },
        "Image": "sha256:9c7a54a9a43cca047013b82af109fe963fde787f63f9e016fdc3384500c2823d",
```

3 - Using the `docker port` command

List the port mappings for a container. This command helps you identify the ports exposed by a container and map them to the corresponding host ports.

You can run `docker port container_name` to see the port mappings.

```
esteban@ubuntu-server-template:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                                       NAMES
ff9dca585b41   nginx     "/docker-entrypoint.…"   7 seconds ago   Up 6 seconds   0.0.0.0:8081->80/tcp, :::8081->80/tcp   web
esteban@ubuntu-server-template:~$ docker port ff9dca585b41
80/tcp -> 0.0.0.0:8081
80/tcp -> [::]:8081
```

4 - Using the `docker stats` command

View resource usage statistics for one or more containers. This command provides real-time information about CPU, memory, network, and disk usage of running containers.

Simply run `docker stats` to monitor the resource utilization of your containers.

```
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O     PIDS
ff9dca585b41   web       0.00%     2.719MiB / 3.832GiB   0.07%     1.02kB / 0B   0B / 24.6kB   3
```

5 - Using the `docker top` command

View the processes running inside a container. This command allows you to see the active processes within a container, similar to the `top` command in Linux.

Execute `docker top container_id` to view the running processes.

```
esteban@ubuntu-server-template:~$ docker top ff9dca585b41
UID                 PID                 PPID                C                   STIME               TTY                 TIME                CMD
root                17898               17878               0                   18:51               pts/0               00:00:00            nginx: master process nginx -g daemon off;
systemd+            17943               17898               0                   18:51               pts/0               00:00:00            nginx: worker process
systemd+            17944               17898               0                   18:51               pts/0               00:00:00            nginx: worker process
```

6 - Using the `docker save` command

Save an image to a tar archive. This command enables you to export Docker images to a file for sharing or storing.

Use `docker save -o image.tar image_name` to save the image to a tar archive.

```
esteban@ubuntu-server-template:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
nginx         latest    f9c14fe76d50   12 days ago   143MB
hello-world   latest    9c7a54a9a43c   4 weeks ago   13.3kB
esteban@ubuntu-server-template:~$ docker save -o image.tar nginx
esteban@ubuntu-server-template:~$ ls
 image.tar  'ystemctl status docker'
```

7 - Using the `docker load` command

Load an image from a tar archive. This command allows you to import a Docker image from a tar archive file.

Run `docker load -i image.tar` to load the image into your Docker environment.

```
esteban@ubuntu-server-template:~$ docker load -i image.tar
Loaded image: nginx:latest
```

Congratulations on completing Day 16 of the #90DaysOfDevOps challenge. Today, we dived into the exciting world of Docker for DevOps Engineers. We explored Docker's concepts and essential commands.

Get ready for Day 17, where we will dive into a thrilling Docker project designed specifically for DevOps Engineers. This project will allow you to apply your Docker knowledge and skills to solve real-world challenges in the DevOps realm. Stay tuned!