

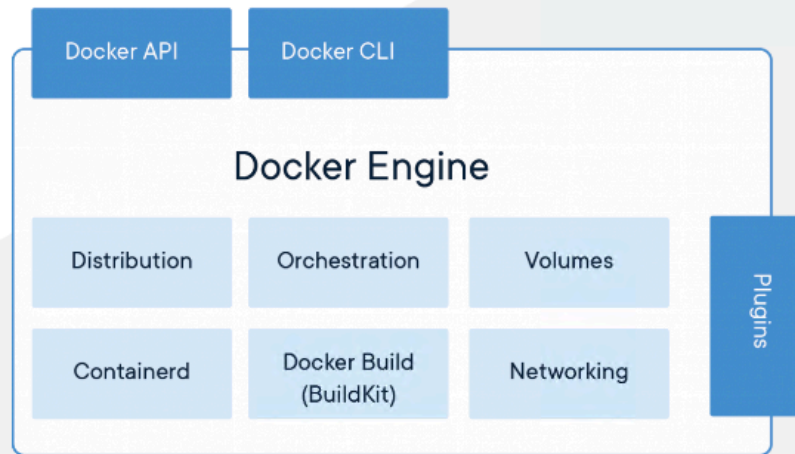
2024-07-24---docker-install

[#Link](#)

Docker

The Industry-Leading Container Runtime

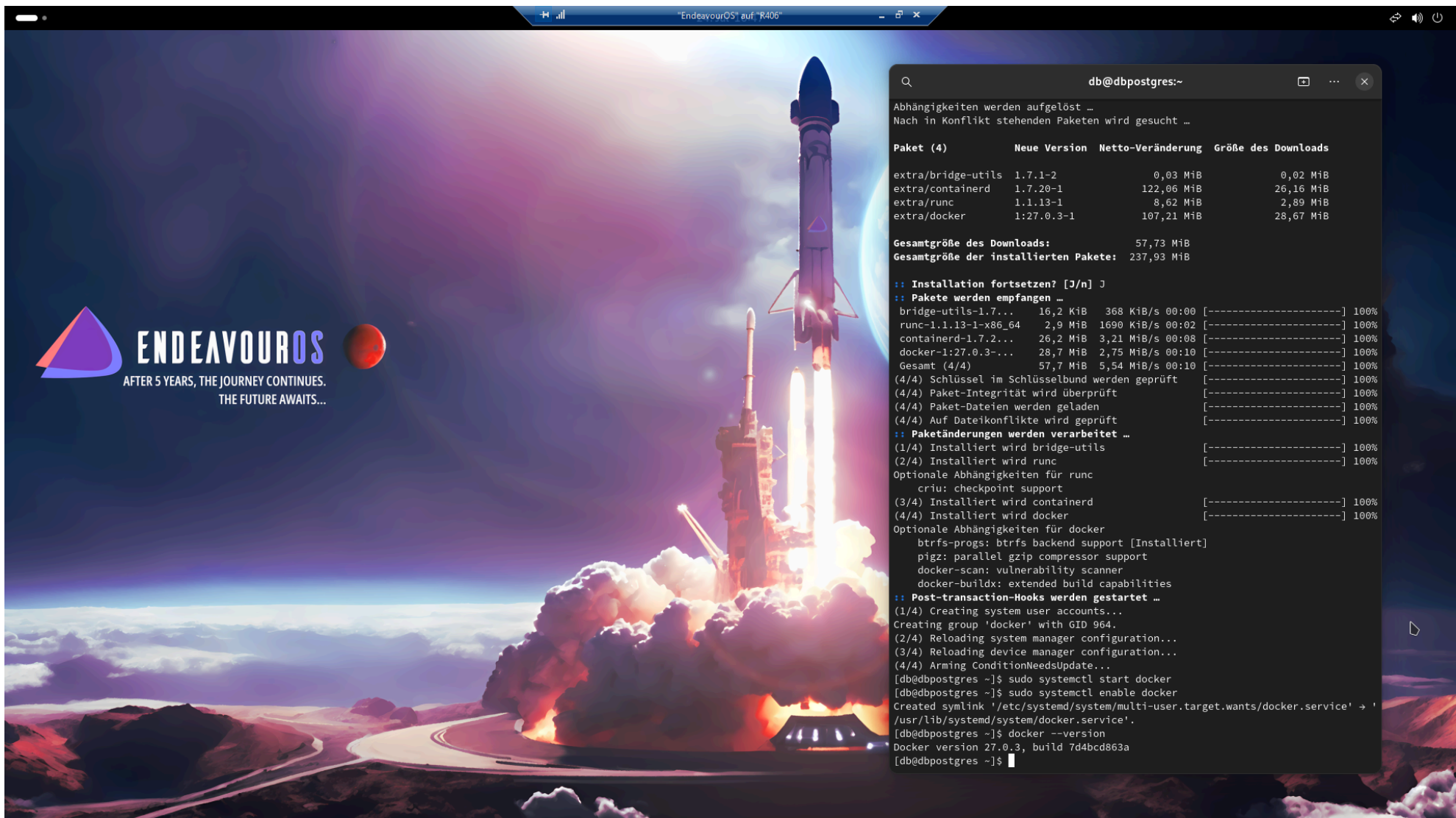
Docker Engine powers millions of applications worldwide, providing a standardized packaging format for diverse applications.



Docker Engine Sparked the Containerization Movement

Docker Engine is the industry's de facto container runtime that runs on various Linux ([CentOS](#), [Debian](#), [Fedora](#), [RHEL](#), and [Ubuntu](#)) and [Windows Server](#) operating systems. Docker creates simple tooling and a universal packaging approach that bundles up all application dependencies inside a container which is then run on Docker Engine. Docker Engine enables containerized applications to run anywhere consistently on any infrastructure, solving "dependency hell" for developers and operations teams, and eliminating the "it works on my laptop!" problem.

Installation



Installing Docker Engine on Arch Linux is a bit more involved than on other distributions, but it's still a relatively straightforward process. Here are the steps:

Method 1

- 1. **Update your system:**

```
sudo pacman -Syu
```

- 2. **Install the `docker` package:**

```
sudo pacman -S docker
```

- 3. **Start the Docker service:**

```
sudo systemctl start docker
```

- 4. **Enable the Docker service to start at boot:**

```
sudo systemctl enable docker
```

- 5. Method 1 are running ? You've successfully installed Docker Engine on your Arch Linux system

To verify that Docker is working correctly, you can run a few commands:

- 1. **Check the Docker version:**

```
docker --version // `For example: Docker version 27.0.3, build 7d4bcd863a`
```

- 2. **Run a test container:**

```
docker run -it ubuntu /bin/bash
```

- (this will start a new Ubuntu container and open a bash shell inside it)

3. List running containers:

```
docker ps
```

- (this should show you the test container you just started)

4. Stop the test container:

```
docker stop <container_id> // `replace <container_id> with the ID of the test container`
```

If everything is working as expected, you should see the Docker version, be able to run a test container, and list/stop the container using the above commands.

Now that you have Docker Engine installed, you can start exploring the world of containerization!

Arch Linux

- When you're running Arch Linux with kernel version 6.10.0-arch1-2.

When it comes to pulling a Docker image, the `arch` image is not a valid option.

The `arch` image is not an official Docker image, and it's not designed to run as a container.

Instead, you can use the following Docker images, which are compatible with Arch Linux:

1. `ubuntu` : A popular choice for running a Linux container. You can pull the latest Ubuntu image using `docker pull ubuntu` .
2. `debian` : Another popular Linux distribution that can run as a container. You can pull the latest Debian image using `docker pull debian` .

3. `alpine` : A lightweight Linux distribution that's often used as a base image for other containers. You can pull the latest Alpine image using `docker pull alpine` .

If you want to run a container with a similar environment to your host system (Arch Linux), you can use the `archlinux` image from the Docker Hub. However, please note that this image is not officially supported by Docker, and it might not be up-to-date.

To pull the `archlinux` image, use the following command:

```
docker pull archlinux/archlinux
```

Keep in mind that this image might not be optimized for running as a container, and you might encounter issues.

Some other useful Docker commands you might want to explore:

- List all available Docker images on your system:

```
docker images
```

- List all running containers

```
docker ps
```

- Run a new container from the specified image and open a bash shell inside it

```
docker run -it <image_name> /bin/bash
```

- Stop a running container:

```
docker stop <container_id>
```

- Remove a stopped container:

```
docker rm <container_id>
```

Feel free to ask if you have any more questions or need help with Docker!

Some next steps you might want to consider:

Pull a Docker-Image:

```
docker pull <image-name> // `docker pull ubuntu`
```

. and

Run a Docker-Image

```
docker run -it <image_name> // `docker run -it ubuntu`
```

.

```
docker run -it ubuntu /bin/bash
```

.

```
docker run -it alpine /bin/bash
```

Create a Dockerfile

- To create a `Dockerfile` and build a custom image, follow these steps:

****Step 1: Create a new directory**

```
mkdir my-alpine-image
```

. and navigate into it

```
cd my-alpine-image
```

Step 2: Create a `Dockerfile` in the new directory

```
touch Dockerfile
```

.

Step 3: Edit the `Dockerfile`

- Open the `Dockerfile` in your favorite editor

```
nano Dockerfile
```

.

- and add the following contents:

```
# Use an official Alpine Linux image as a base
```



```
FROM alpine:latest

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any necessary packages
RUN apk add --no-cache python

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME=John

# Run `command` when the container launches
CMD ["python", "app.py"]
```

Let me explain what each line does:

- `FROM alpine:latest` : We use the latest Alpine Linux image as a base.
- `WORKDIR /app` : We set the working directory in the container to `/app`.
- `COPY . /app` : We copy the current directory contents into the container at `/app`.
- `RUN apk add --no-cache python` : We install Python using the `apk` package manager.
- `EXPOSE 80` : We make port 80 available to the world outside this container.
- `ENV NAME=John` : We define an environment variable `NAME` with value `John`.
- `CMD ["python", "app.py"]` : We set the default command to run when the container launches, which is `python app.py`.

Step 4: Create a file `app.py` in the same directory

```
touch app.py
```

Step 5: Edit `app.py`

- Open `app.py` in your favorite editor and add some Python code, for example:

```
print("Hello, World!")  
print("My name is", NAME)
```

Step 6: Build the custom image

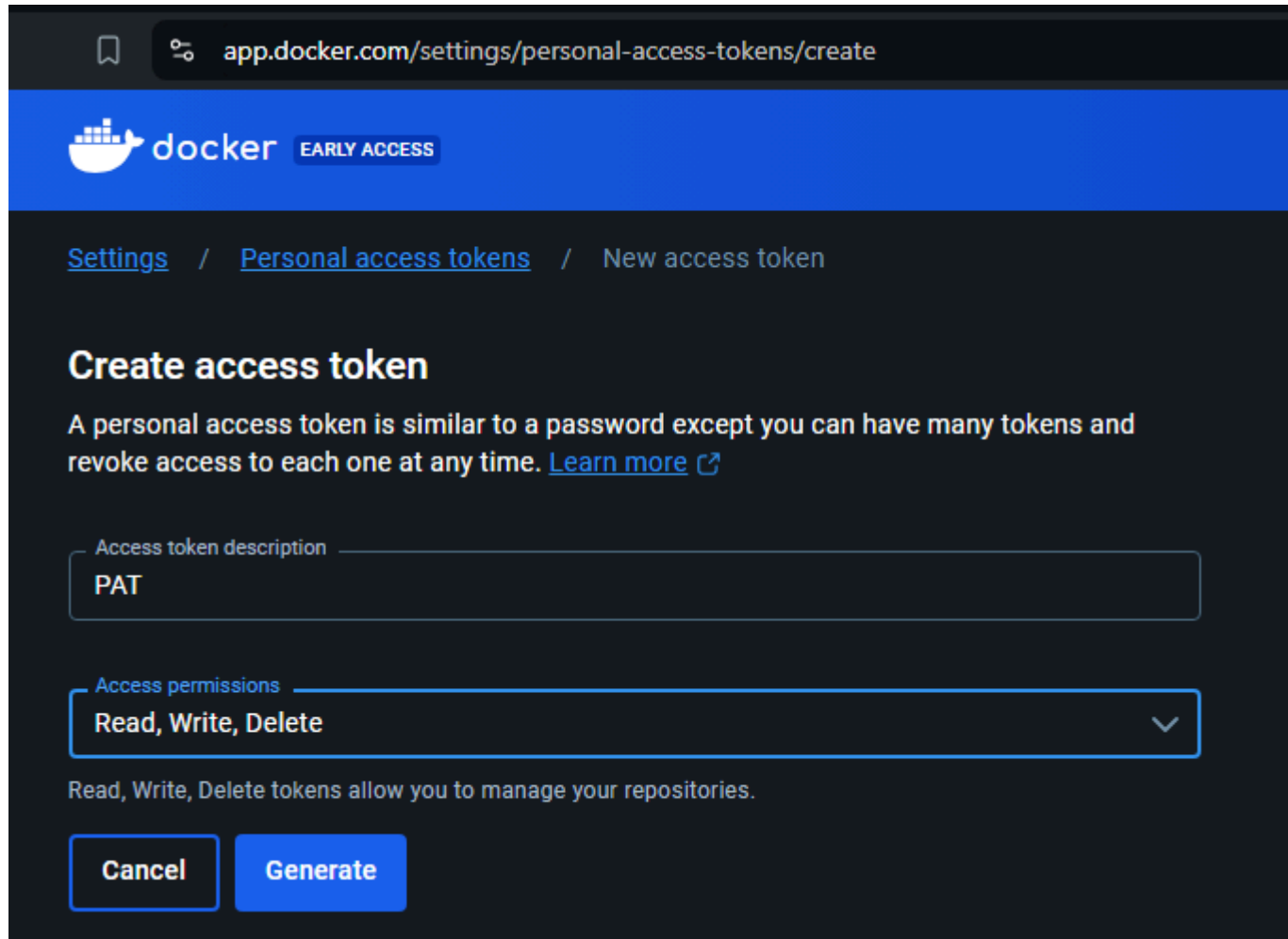
- Enable Experimental Features:

```
export DOCKER_CLI_EXPERIMENTAL=enabled
```

Step 7: login with Docker ID

- login in with your ID or E-Mail address to push and pull images from Docker Hub
- if you don't have a Docker ID - head over to <https://hub.docker.com/> to create one
- you can log in with your Password or Personal Access Token (`#PAT`)
- using a limited-scope PAT grants better security and is required for organizations using SSO
- learn more at <https://docs.docker.com/docker-id/> or <https://app.docker.com/signup/>
- Username: `db`
- Password: `docker`
- Method PAT: <https://app.docker.com/settings/personal-access-tokens/create>
 - Access token description `PAT`
 - Access permission `Read,Write,Delete`

- Click on -> Generate



The screenshot shows the Docker app interface on a dark theme. At the top, the browser address bar displays 'app.docker.com/settings/personal-access-tokens/create'. Below this is a blue header with the Docker logo and 'EARLY ACCESS' badge. The breadcrumb navigation shows 'Settings / Personal access tokens / New access token'. The main heading is 'Create access token'. A descriptive text explains that a personal access token is like a password but can be revoked, with a 'Learn more' link. There are two input fields: 'Access token description' with the value 'PAT' and 'Access permissions' with a dropdown menu showing 'Read, Write, Delete'. A note states that Read, Write, and Delete permissions allow managing repositories. At the bottom are 'Cancel' and 'Generate' buttons.

app.docker.com/settings/personal-access-tokens/create

docker EARLY ACCESS

[Settings](#) / [Personal access tokens](#) / New access token

Create access token

A personal access token is similar to a password except you can have many tokens and revoke access to each one at any time. [Learn more](#)

Access token description

PAT

Access permissions

Read, Write, Delete

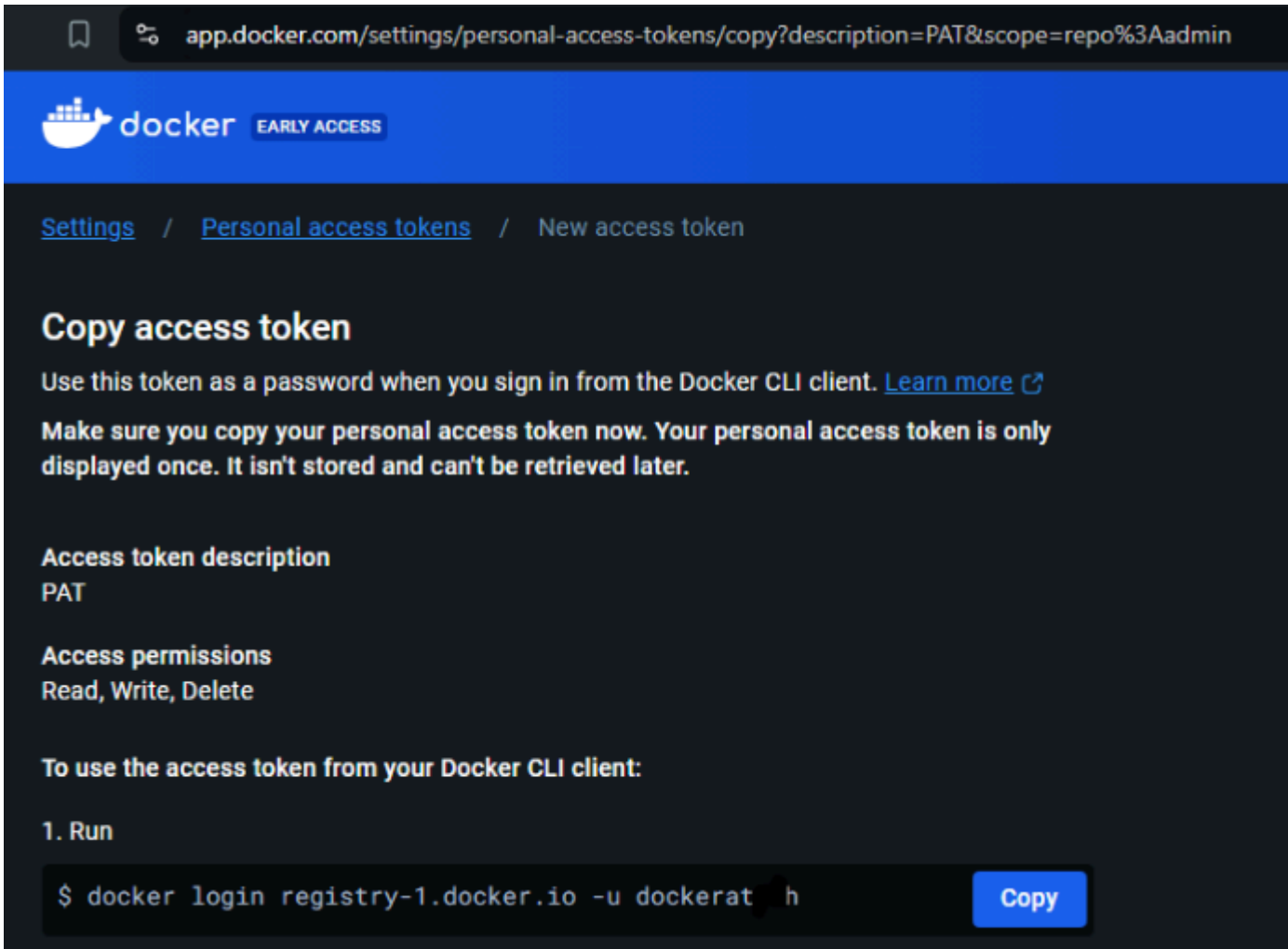
Read, Write, Delete tokens allow you to manage your repositories.

Cancel Generate

- to use the access token from your Docker CLI client:

```
docker login registry-1.docker.io -u dockeratydh
```

- and copy the access token in to your login

A screenshot of the Docker web interface showing the 'Copy access token' page. The browser address bar shows 'app.docker.com/settings/personal-access-tokens/copy?description=PAT&scope=repo%3Aadmin'. The Docker logo and 'EARLY ACCESS' badge are in the top left. Breadcrumbs show 'Settings / Personal access tokens / New access token'. The main heading is 'Copy access token'. Below it, text explains that the token is used as a password for the Docker CLI and is only shown once. It lists the 'Access token description' as 'PAT' and 'Access permissions' as 'Read, Write, Delete'. A section titled 'To use the access token from your Docker CLI client:' contains a numbered step '1. Run' followed by a terminal command: '\$ docker login registry-1.docker.io -u dockrat h'. A blue 'Copy' button is next to the command.

app.docker.com/settings/personal-access-tokens/copy?description=PAT&scope=repo%3Aadmin

docker EARLY ACCESS

[Settings](#) / [Personal access tokens](#) / New access token

Copy access token

Use this token as a password when you sign in from the Docker CLI client. [Learn more](#)

Make sure you copy your personal access token now. Your personal access token is only displayed once. It isn't stored and can't be retrieved later.

Access token description
PAT

Access permissions
Read, Write, Delete

To use the access token from your Docker CLI client:

1. Run


```
$ docker login registry-1.docker.io -u dockrat h
```

Copy

- At the password prompt, enter the personal access token.

```
dckr_pat_ R9IqBU-KvvHPwFCxxxkiM
```

app.docker.com/settings/personal-access-tokens

 **docker** EARLY ACCESS

?

⋮

D

[Settings](#) / Personal access tokens

✓

Successfully created personal access token PAT.

×

Personal access tokens

You can use a personal access token instead of a password for Docker CLI authentication. Create multiple tokens, control their scope, and delete tokens at any time. [Learn more](#)

Generate new token

Description	Scope	Status	Source ⓘ	Created	Last Used	
PAT	Read, Write, Delete	Active	Manual	Jul 24, 2024 at 14:16:15	Jul 24, 2024 at 14:25:27	⋮

Rows per page: 10 ▾

1-1 of 1

< >



db@dbpostgres:~/my-alpine-image



```
[db@dbpostgres my-alpine-image]$ docker login
Log in with your Docker ID or email address to push and pull images from Docker
Hub. If you don't have a Docker ID, head over to https://hub.docker.com/ to crea
te one.
You can log in with your password or a Personal Access Token (PAT). Using a limi
ted-scope PAT grants better security and is required for organizations using SSO
. Learn more at https://docs.docker.com/go/access-tokens/

Username: db
Password:
Get "https://registry-1.docker.io/v2/": unauthorized: incorrect username or pass
word
[db@dbpostgres my-alpine-image]$ docker login registry-1.docker.io -u dockeratyd
h
Password:
Get "https://registry-1.docker.io/v2/": unauthorized: incorrect username or pass
word
[db@dbpostgres my-alpine-image]$ docker login registry-1.docker.io -u dockeratyd
h
Password:
Get "https://registry-1.docker.io/v2/": unauthorized: incorrect username or pass
word
[db@dbpostgres my-alpine-image]$ docker login registry-1.docker.io -u dockeratyd
h
Password:
Error: Password Required
[db@dbpostgres my-alpine-image]$ docker login registry-1.docker.io -u dockeratyd
h
Password:
WARNING! Your password will be stored unencrypted in /home/db/.docker/config.js
on.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
[db@dbpostgres my-alpine-image]$
```

Step 8: Build the custom image

```
docker run -it my-alpine-image
```

This command runs a new container from the `my-alpine-image` image and opens a terminal session inside the container.

That's it! You've created a custom Docker image using a `Dockerfile` and built it from scratch!

Explore Docker Compose

- Install Docker Compose using:

```
sudo pacman -S docker-compose` and learn how to define and run multi-container applications
```

Alpine Container

- usefull commands:

```
docker run -it alpine /bin/bash
```

- `docker run` : This is the command to start a new Docker container.

- `-it` : This flag combination allows you to interact with the container as if you were sitting in front of it. `-i` allocates a pseudo-TTY, and `-t` enables terminal input/output.
- `alpine` : This is the name of the Docker image you want to use. In this case, it's the official Alpine Linux image.
- `/bin/bash` : This is the command to run inside the container. In this case, it starts a Bash shell.

When you run this command, Docker will create a new container from the Alpine image, and you'll be dropped into a Bash shell inside the container. From there, you can explore the container's file system, run commands, and more!