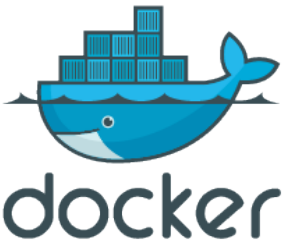
Getting Started

- Step-by-Step Guide: How to Setup Wastebin

Step-by-Step Guide: How to Setup Wastebin



This article will guide you through the process of building your own Wastebin image for the ARM64 architecture, specifically designed to run on a Raspberry Pi 5. Wastebin is a minimal pastebin application originally designed for AMD64 architecture. In this guide, you will learn how to build and run it on ARM64 using Docker Swarm with an SQLite3 database.

Building the Image for ARM64/v8 Architecture

Step 1: Download the Wastebin GitHub Repository

Start by cloning the official Wastebin GitHub repository:

```
git clone https://github.com/matze/wastebin.git
```

Step 2: Navigate to the Wastebin Directory

Navigate to the downloaded Wastebin repository:

```
cd /path/to/wastebinrepo/wastebin
```

Step 3: Build the ARM64 Image

To build an ARM64 image on an x86_64 host, run the following command:

```
sudo docker build --platform linux/arm64 -t wastebin:v2.5.0-arm64 -f Dockerfile.arm .
```

> Building the image may take some time. For example, it took around 322 seconds for me, so be patient.

Step 3.1: Verify the Image

Check if the image was built successfully:

```
docker images
```

You should see output like this:

```
wastebin            v2.5.0-arm64      796d3c8a13da   42 seconds ago   12.3MB
```

Step 4: Login to Docker Hub

Login to your Docker Hub account:

```
docker login
```

After logging in, visit https://login.docker.com/activate and enter your confirmation code.

Step 5: Push the Image to Docker Hub

You can now push your newly built image to Docker Hub.

Step 5.1: Tag the image using your Docker Hub username:

```
docker tag wastebin:v2.5.0-arm64 aeoneros/wastebin:v2.5.0-arm64
```

Step 5.2: Push the image to Docker Hub:

```
docker push aeoneros/wastebin:v2.5.0-arm64
```

> You can also access my prebuilt image on Docker Hub: Wastebin ARM64 Image.

Running Wastebin with Docker-Compose and Traefik

Step 1: Create a Directory for Persistent Data

You'll need a directory for storing Wastebin data. Create the following directories:

```
mkdir /mnt/glustermount/data/wastebin_data
sudo useradd -u 10001 wastebinuser
```

Step 2: Create the Docker Compose File

Now create and customize your `docker-compose.yaml` file:

```
nano docker-compose.yaml
```

Here is an example configuration:

```
version: "3.8"

services:
  wastebin:
    image: aeoneros/wastebin:v2.5.0-arm64
    environment:
      - WASTEBIN_DATABASE_PATH=/data/state.db # Don't change this, it maps inside the container
      - WASTEBIN_PASSWORD_SALT=${WASTEBIN_PASSWORD_SALT_HASH}
      - RUST_LOG=info
```

```
    volumes:
      - '/mnt/glustermount/data/wastebin_data:/data'
    networks:
      - management_net
    deploy:
      mode: replicated
      replicas: 1
      labels:
        - 'traefik.enable=true'
        - 'traefik.http.routers.wastebin.rule=Host(`wastebin.aeoneros.com`)'
        - 'traefik.http.routers.wastebin.entrypoints=websecure'
        - 'traefik.http.routers.wastebin.tls.certresolver=leresolver'
        - 'traefik.http.services.wastebin.loadbalancer.server.port=8088'
        - 'traefik.docker.network=management_net'

networks:
  management_net:
    external: true
```

> Ensure the `/wastebin_data` folder is writable by user `10001`.

Step 3: Adjust Permissions

Change the ownership and permissions of the storage directory to ensure proper access.

Step 3.1: Change ownership:

```
sudo chown 10001:10001 /mnt/glustermount/data/wastebin_data
```

Step 3.2: Set the correct permissions:

```
sudo chmod 700 /mnt/glustermount/data/wastebin_data
```

Verify the ownership and permissions:

```
ls -ld /mnt/glustermount/data/wastebin_data
```

You should see something like this:

```
drwx------ 10001 10001 ... /mnt/glustermount/data/wastebin_data
```

Step 4: Deploy the Stack

Deploy the Wastebin service using Docker Swarm:

```
docker stack deploy -c docker-compose.yaml wastebin
```

Configuration Options

The following environment variables can be used to configure Wastebin:

- **WASTEBIN_ADDRESS_PORT**: Sets the address and port (default: `0.0.0.0:8088`).
- **WASTEBIN_BASE_URL**: Determines the base URL for QR code display.
- **WASTEBIN_CACHE_SIZE**: Number of cached syntax-highlighted items (default: `128`).
- **WASTEBIN_DATABASE_PATH**: Path to the SQLite3 database (default: in-memory).
- **WASTEBIN_HTTP_TIMEOUT**: Maximum request processing time (default: `5 seconds`).
- **WASTEBIN_MAX_BODY_SIZE**: Maximum POST request size (default: `1 MB`).
- **WASTEBIN_MAX_PASTE_EXPIRATION**: Maximum paste lifetime (default: unlimited).
- **WASTEBIN_PASSWORD_SALT**: Salt for hashing user passwords.
- **WASTEBIN_SIGNING_KEY**: Key to sign cookies (random if not set).
- **WASTEBIN_TITLE**: HTML page title (default: `wastebin`).
- **RUST_LOG**: Logging level (e.g., `info`, `debug`).

Extra Information: WASTEBIN_PASSWORD_SALT

The `WASTEBIN_PASSWORD_SALT` environment variable provides additional security when hashing passwords. Here’s how it works:

What is a Password Hash?
A password hash is a secure, irreversible transformation of a user’s password, ensuring the password itself is not stored.

What is a Salt?
A salt is a random string added to the password before hashing, ensuring that even if two users have the same password, their hashes will differ.

Why Use a Salt?
Using a salt protects against certain attacks, like rainbow table attacks, by making it harder for attackers to crack passwords.

Do You Need to Set It?
For production environments, it’s recommended to set a unique, secure salt. You can generate a salt using:

```
openssl rand -base64 32
```

Conclusion

In this guide, you’ve learned how to build and deploy Wastebin for ARM64 architecture on a Raspberry Pi 5 using Docker Swarm. Wastebin’s minimal footprint, combined with features like encrypted pastes and QR code sharing, make it a versatile tool for managing and sharing data. With proper configuration, you can run it securely in production and adapt it to future versions as needed.