How to containerize a Rust app with Docker

Containerize Rust apps with Docker to simplify deployment and ensure consistency across different environments.

Containerization provides the necessary benefits in terms of mobility, isolation, and efficiency. It allows consistent deployment of applications across different environments, and at the same time, ensures safety and stability through application isolation. It also optimizes resource usage, simplifies development and management, and enhances scalability.

Containerizing a Rust app with Docker provides a reliable and efficient method of packaging applications and dependencies into standalone and portable environments. It allows for seamless deployment on a variety of systems without regard to the underlying infrastructure.

Set up a simple web server in Rust with Actix

You can set up a simple web server in Rust with Actix and containerize the application with Docker. You will show a new port where you will access the server to receive queries.

Run this command to create a new Rust project with the Cargo package manager:

```
cargo new my-app
```

When creating a new Rust project, Cargo adds the cargo.toml file to the project's root directory. Open the cargo.toml file and add the Actix crate to the dependencies section as follows:

```
[dependencies] actix-web = "4.3.1"
```

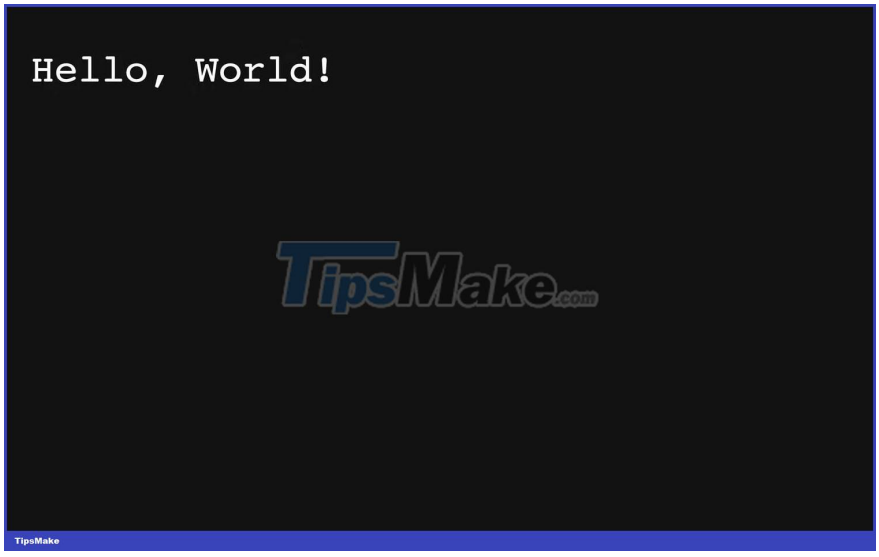
Here's how you can set up a simple server in Rust with crate Actix:

```
use actix_web::{get, App, HttpResponse, HttpServer, Responder}; // Nh?
p các ph?n ph? thu?c c?n thi?t t?
 framework Actix Web #[get("/")] async fn hello() -> impl Responder { // Xác
??nh hàm x? lý cho truy v?n GET t?i ???ng d?n g?c ("/") // Hàm này tr? v? m
?t ki?u tri?
n khai trait Responder HttpResponse::Ok().body("Hello, World!") // Tr? v?
 ph?n h?i HTTP v?i mã tr?ng thái 200 (OK) // và n?i dung c?a ph?n h?
i là "Hello, World!" } #[actix_web::main] async fn main() -> std::io::Result()> {
?i?m nh?p c?a ?ng d?ng HttpServer::new(|| { // T?o phiên b?n m?i c?
a HttpServer App::new().service(hello) // T?o phiên b?n m?i c?a App và ??
ng ký hàm hello }).bind("127.0.0.1:8080")?.run().await // Liên k?t server t
?i ??a ch? IP và c?ng này // Kh?i ??ng server và ??i quá trình hoàn t?t }
```

This program sets up a basic HTTP Web Server with Actix. The hello function is a function that handles the response to a GET query on port 8080 with 'Hello, World!'.

The main function sets up a server instance with the HttpServer::new function and binds the server to run on localhost 127.0.0.1:8080.

Now, run the cargo run command to run the web server. This is the result of opening the address on a web browser.



Writing Dockerfile for Rust app

To containerize a Rust app with Docker, you must create a Dockerfile and define commands for the containerization process.

Dockerfile has no extension. You just need to create a Dockerfile . You can also create .dockerignor files as abstract files in the active directory from the build process.

Define command in Dockerfile

The Docker file will contain commands to pull a base image from the Docker repository, set the working directory, copy the file, build dependencies/applications and create minimal thumbnails, export the port, and run the application.

```
# Dùng b?n m?i nh?t c?a ?nh c? s? Rust FROM rust:latest # ??t th? m?c ?
ang ho?t ??
ng trong container sang container to /my WORKDIR /usr/src/my-app # Sao chép file
? án Rust sang th? m?c ?ang ho?t ??ng COPY . . # Xây d?
ng app Rust RUN cargo build # ??t l?nh này ?? ch?y app Rust CMD cargo run
```

After defining the required commands for app containerization, you can build a container with this command:

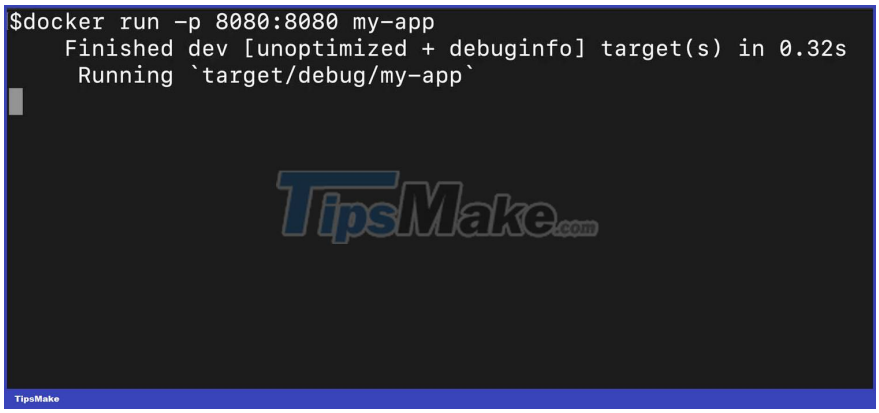
```
docker build -t my-app .
```

This command builds a Docker image for the app my-app with the tag my-app from the current directory.

You can use the dockerrun command to run Docker images.

```
docker run -p 8080:8080 my-app
```

The -p 8080:8080 option maps the host machine's port 8080 to the container's port 8080. Docker will forward traffic redirected to port 8080 on the host machine to port 8080 in the container.



You can send queries to this container over localhost port 8080 to invoke the web service.

Above is how to containerize a Rust app using Docker . Hope the article is useful to you.

You finished reading the article "**How to containerize a Rust app with Docker**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.