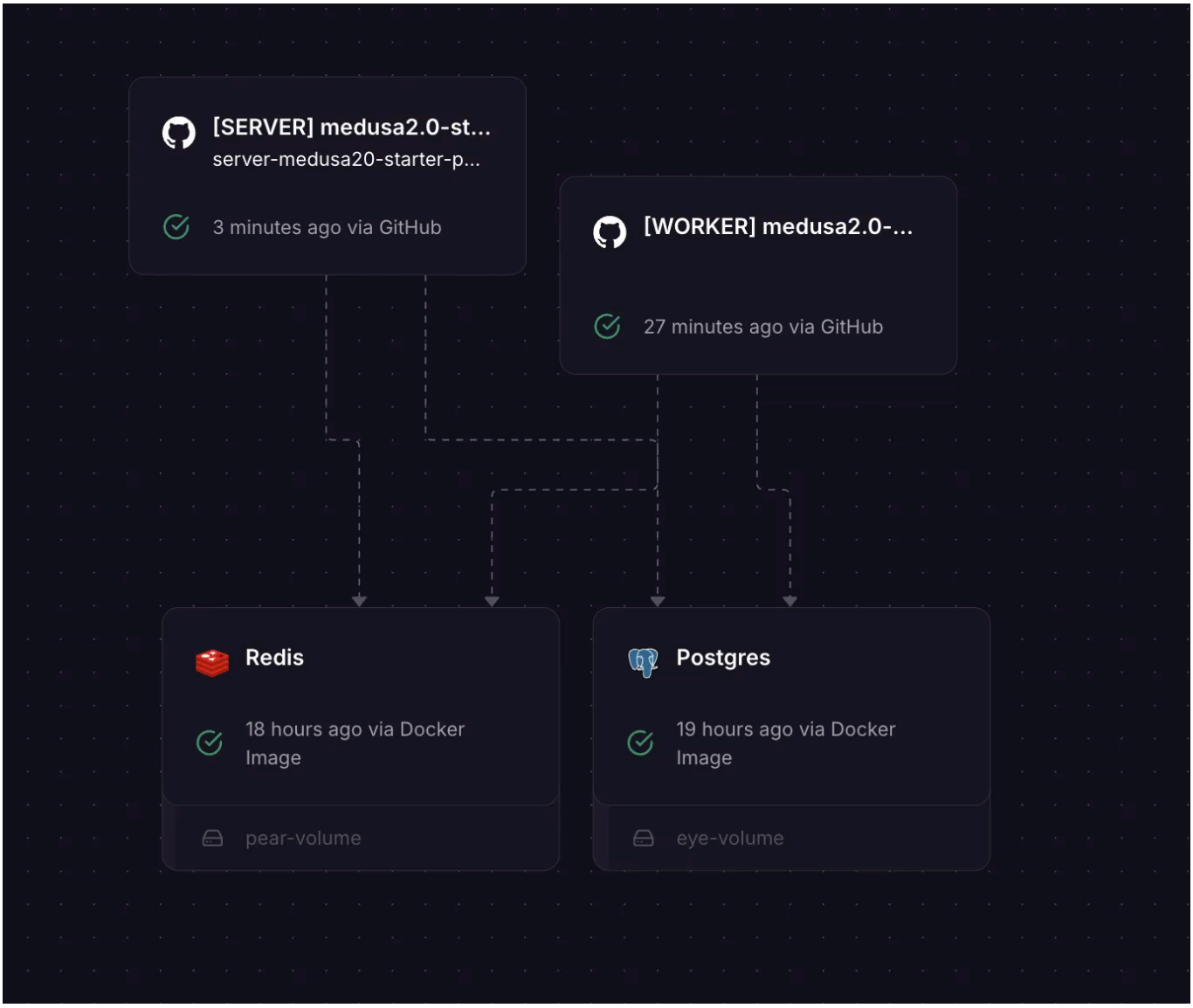
Medusa 2.0 e-commerce (Admin, Server and Worker) easy custom setup on Railway (+Stripe, Resend and Storage readiness)

PiotrDev Follow 13 min read · Dec 2, 2024

137 1



01 — Introduction to Medusa 2.0

For those who are particularly impatient, the **GitHub** repo is here :)

Medusa.js is an open-source, **headless e-commerce framework** designed to help developers build flexible, customizable e-commerce solutions. It provides an alternative to platforms like **Shopify** but with more freedom and complete control over the tech stack. Built with **Node.js**, Medusa focuses on

offering a highly **modular** and **API-driven** backend that can be integrated with various frontends, such as **Next.js** or **React.**

I tested Medusa 1.0 in Production with various projects and am now exploring the latest **2.0 version.**

Medusa 2.0 features a fully decoupled module system. This design allows us to integrate only the necessary components, thereby reducing complexity and facilitating incremental adoption. It also enables seamless module replacement or customization without affecting the overall system.

02- Introduction to Railway

Railway.app is a modern infrastructure platform that makes deploying, managing, and scaling applications easy without extensive DevOps effort. It offers features such as **automated service discovery**, **high-speed networking**, and support for multiple protocols — all designed to simplify the deployment process for developers.

Key Features:

- **Cost Efficiency**: Railway.app provides a **pay-as-you-go pricing model**, ensuring you only pay for what you use, which can be highly cost-effective for both small-scale projects and enterprise applications.
- **Ease of Scalability**: The platform is designed for **easy scalability**, allowing you to effortlessly scale services up or down based on demand without manual intervention.
- **Automated Service Discovery**: It automatically detects and configures services, streamlining deployment and reducing manual setup time.
- **High-Performance Networking**: **Railway.app offers fast and secure**

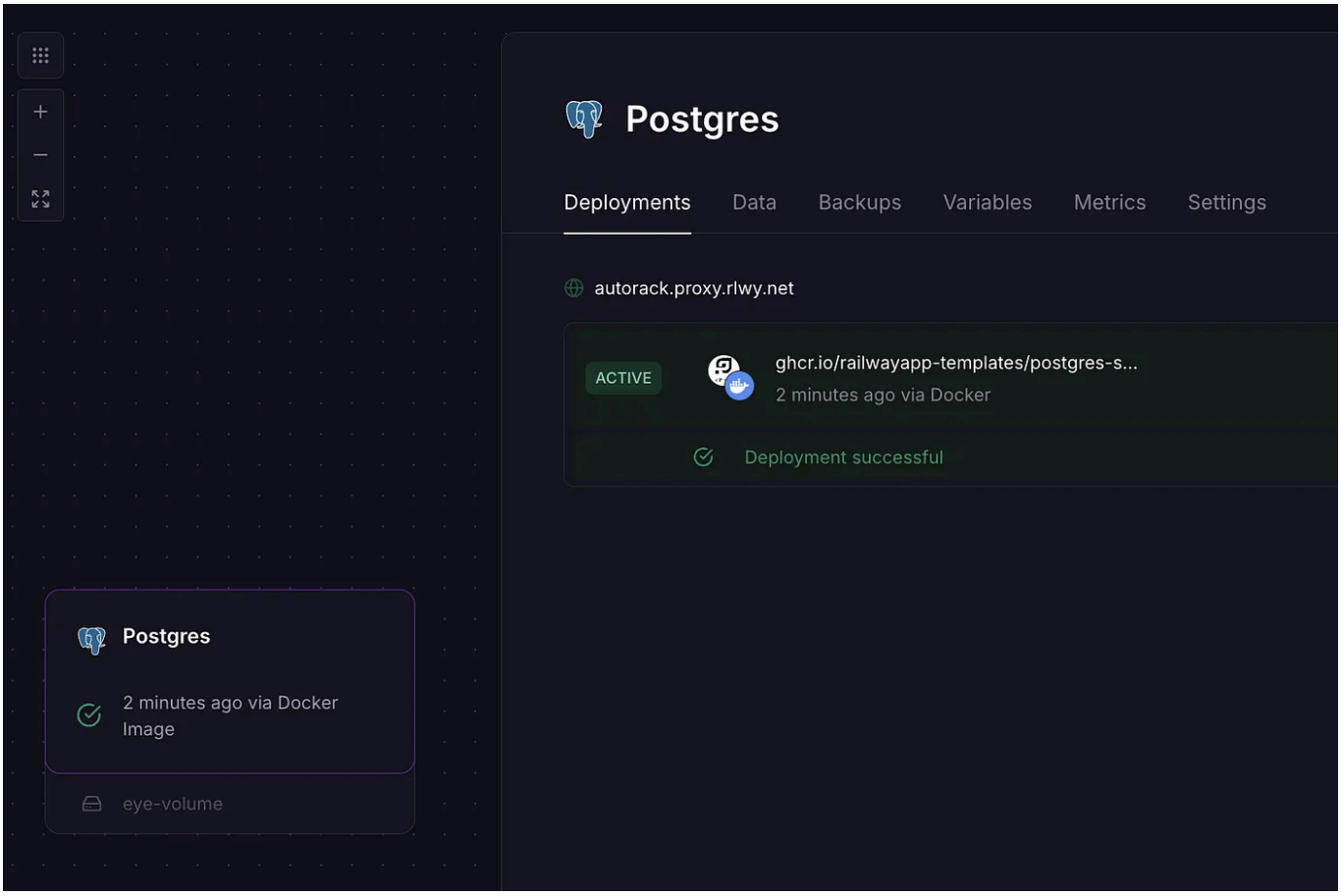
networking capabilities, including private networking for internal communication and public endpoints to expose applications to the internet.

Railway.app enables developers to focus on building and deploying applications with minimal infrastructure hassle, providing an intuitive and scalable alternative to traditional cloud providers.

03 — Setting Up PostgreSQL with Railway and Medusa locally

1. **Create a New Project**: Log in or register on [Railway.app] (https://railway.app), then click on **+ Create** to start a new, empty project.

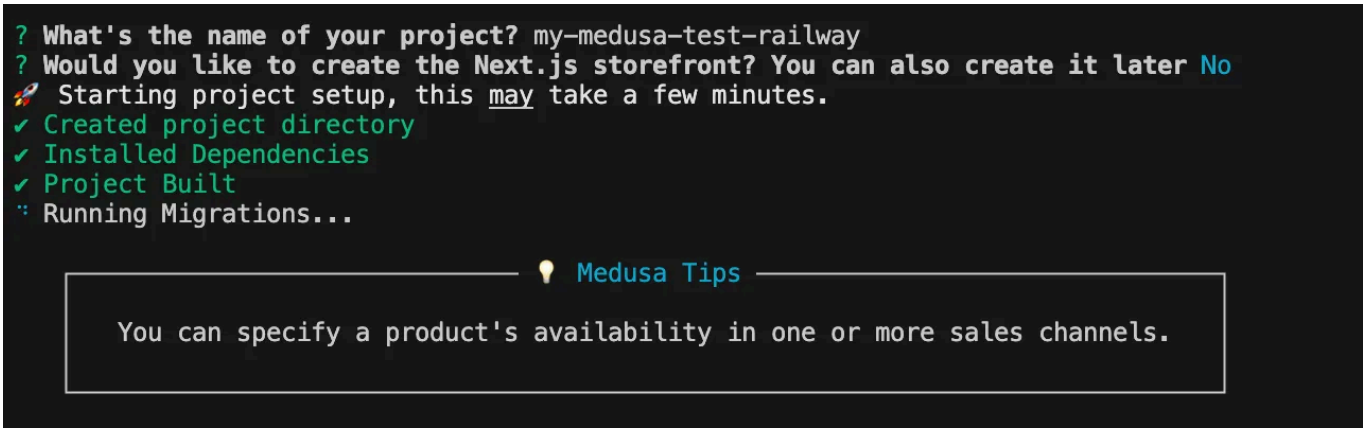
2. **Add a PostgreSQL Service**: Click **Add a Service** and select **PostgreSQL.** The deployment may take up to 2 minutes. Once complete, you'll see your PostgreSQL database ready on your Railway dashboard.



3. **Initialize Medusa Locally:** Run the following command in your local environment, replacing `<YOUR_PORT>` with your PostgreSQL port from Railway:

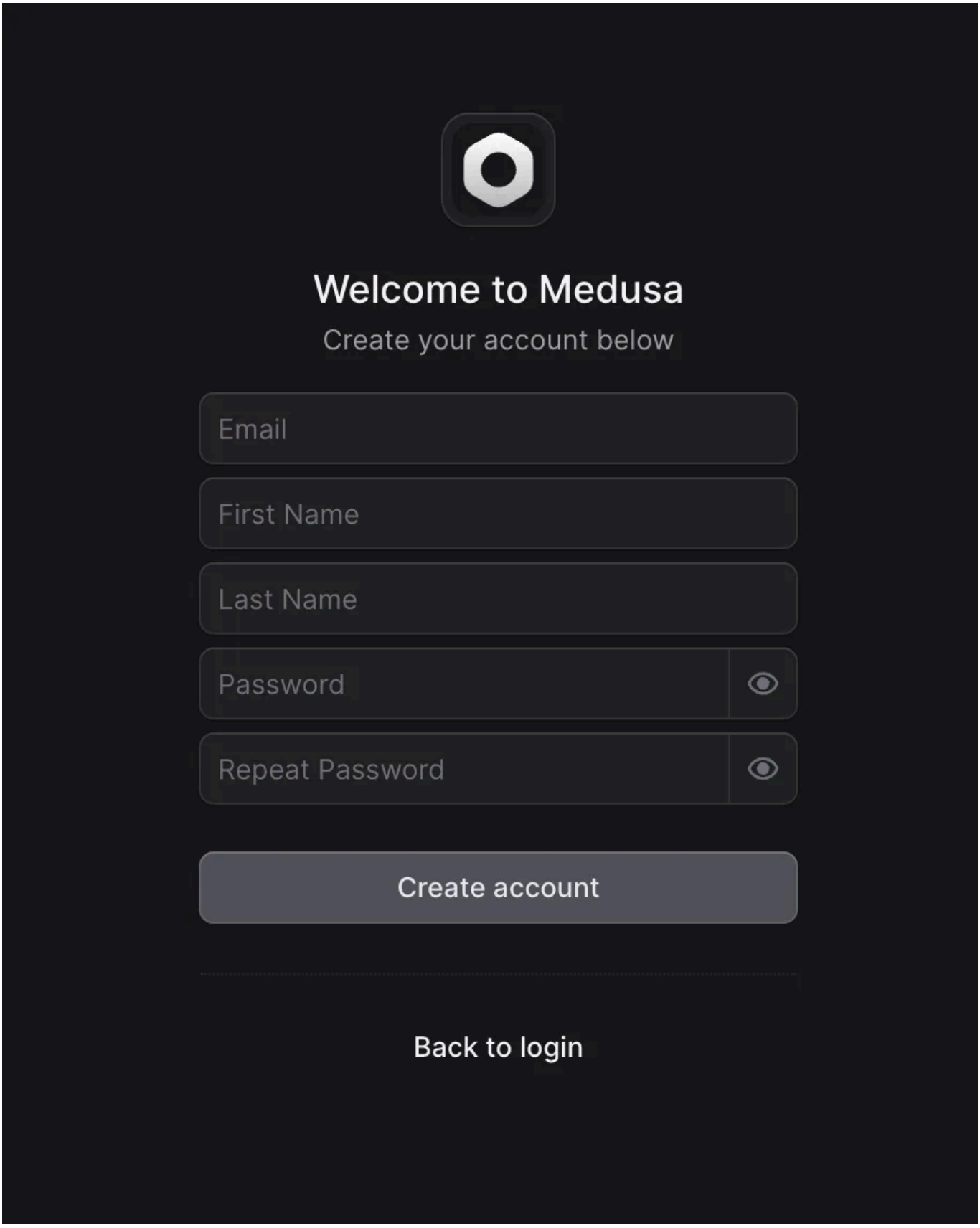
```
npx create-medusa-app@latest --db-url "postgres://user:password@localhost:<YOUR_
```

4. **Answer Setup Questions:** Follow the prompts to configure your Medusa project. If desired, you can also opt to install the **Starter Storefront** built with **Next.js 14** for a quick frontend setup.



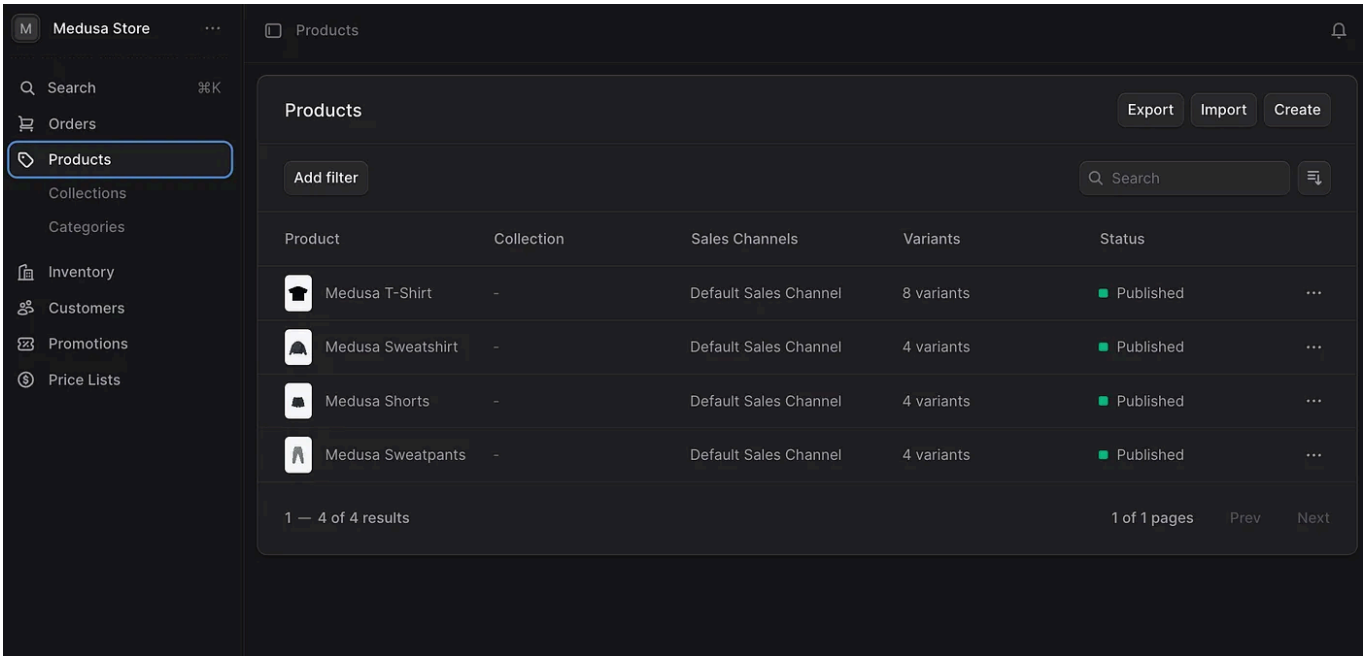
5. **Launch Medusa:** Once the setup completes, Medusa will automatically start on **port 9000**. You'll then be prompted to create your first user/admin account to manage the store.

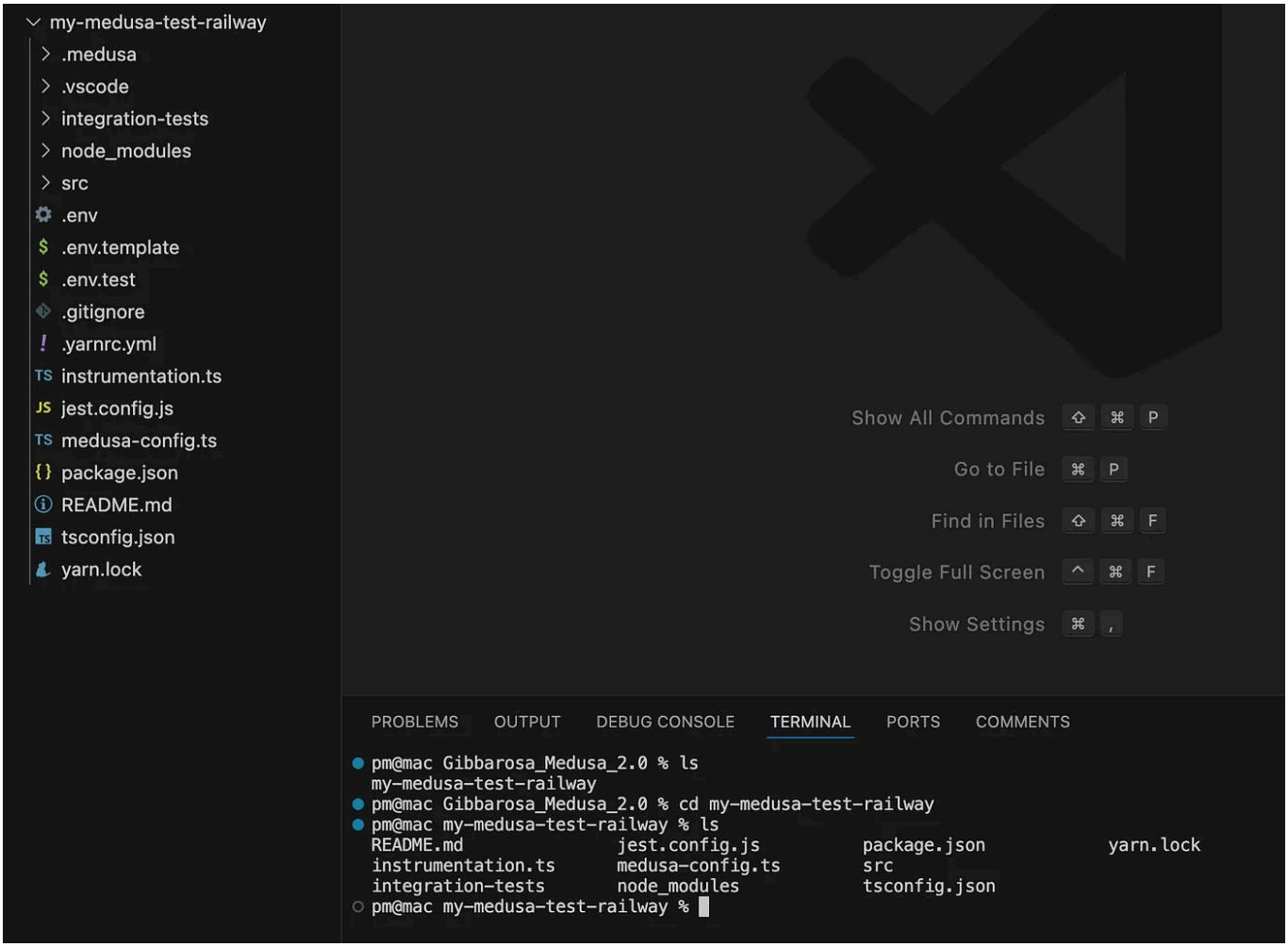
```
warn:    Local Event Bus installed. This is not recommended for production.
info:    Locking module: Using "in-memory" as default.
info:    No job to load from /Users/pm/Desktop/MedusaJS/Gibbarosa_Medusa_2.0/my-medusa-test-railway/node_modules/@medusajs/me
dusa/dist/jobs. skipped.
info:    Generated modules types
- Creating server
✔ Server is ready on http://localhost:9000 — 8ms
info:    Admin URL → http://localhost:9000/app
http:    HEAD /health ← – (200) – 1.505 ms
http:    GET /app/invite?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Imludml0ZV8wMUpEWVpEVjVUOEE0RFZLRzJIVEJLQjVGRSIsI
mVtYWlsIjoiYWRtaW5AbWVkdXNhLXRlc3QuY29tIiwiaWF0IjoxNzMyOTg1MTU0LCJleHAiOjE4MTkzODUxNTQsImp0aSI6ImZkYzNlMTlkLWM4ZDEtNDQ4MS1iOT
hjLTA5ZTE3ODY2ZjUwZiJ9.O7cbJBeYsPpDq7C3a6rq3YIOSfaL6qENPvLHUY2zSYo&first_run=true ← – (200) – 42.436 ms
http:    GET /app/entry.tsx ← http://localhost:9000/app/invite?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Imludml0ZV8
wMUpEWVpEVjVUOEE0RFZLRzJIVEJLQjVGRSIsImVtYWlsIjoiYWRtaW5AbWVkdXNhLXRlc3QuY29tIiwiaWF0IjoxNzMyOTg1MTU0LCJleHAiOjE4MTkzODUxNTQs
Imp0aSI6ImZkYzNlMTlkLWM4ZDEtNDQ4MS1iOThjLTA5ZTE3ODY2ZjUwZiJ9.O7cbJBeYsPpDq7C3a6rq3YIOSfaL6qENPvLHUY2zSYo&first_run=true (200)
 – 0.826 ms
http:    GET /app/index.css ← http://localhost:9000/app/entry.tsx (200) – 752.353 ms
```



6. **View Sample Products:** After logging in, you'll be able to see the **sample products** that were added from the initial seed data, giving you a starting

point to explore the platform.





04 — Enhancing Medusa Config: Adding Stripe Payments and Resend Notifications

In my **GitHub repository** you will find an extended version of the ***medusa-config.ts*** file. I have incorporated **Stripe Payments** and **Resend Notifications**, both of which enhance the functionality of the Medusa setup.

Key Additions:

1. **Stripe Payments**:

— **Stripe** is a popular payment processing platform that makes it easy to accept online payments. It provides secure, reliable payment solutions for e-commerce platforms. In the `***medusa-config.ts***` file, I have added Stripe as a **payment provider**, which allows our Medusa store to process payments seamlessly through **credit cards**, **debit cards**, and other popular payment methods.

2. **Resend Notifications**:

— **Resend** is a service used for **sending email notifications**, designed for simplicity and high deliverability. It integrates easily with platforms like Medusa to handle transactional emails such as order confirmations and password resets.
— In the repository, I've included a custom module in the `**src**` directory called `**resend-notification**`. This module is responsible for managing email notifications and is configured to work with **Resend**.
— Additionally, there are **subscribers** set up in the `**src**` directory to handle events, which trigger the email notifications through **Resend**. These subscribers are configured to listen to relevant events (like order creation) and ensure that customers receive the appropriate emails.

3. **Moved to `[Modules.FILE]` Notation:**

— I have also refactored the configuration to use the `**[Modules.FILE]**` notation, which aligns with Medusa's new **modular system**. This approach is:
— **Better:** It keeps the configuration more **organized** and makes the code easier to understand. Each module is encapsulated, which makes it clearer which part of the functionality each module handles.
— **Safer:** By isolating module configurations, it's easier to manage **dependencies** and **potential conflicts**. If changes are required, updating one module won't inadvertently affect others, minimizing the risk of errors.
— The `**[Modules.FILE]**` notation also enhances **scalability**— if you want to add or replace a specific module (e.g., swapping the file storage provider from S3 to another service), it's straightforward to adjust just that one part of the configuration without impacting the entire system.

These integrations are crucial for adding common e-commerce functionalities like secure payment handling and automatic email notifications, while the modular configuration ensures the setup remains scalable, maintainable, and safe.

Feel free to explore the changes in the `*medusa-config.ts*` and `*src*` directories to understand how these services were implemented and how they contribute to the e-commerce workflow.

05 — Deploying to Railway

What Are We Deploying?

1. **PostgreSQL Database**: We already have this set up on **Railway** (first step of this tutorial).
2. **Redis Database**: Used for caching and optimizing performance.

3. **Medusa Application (Server Mode):** Deploying Medusa in **server mode** along with the **Medusa Admin** dashboard.
4. **Medusa Application (Worker Mode):** Deploying a separate instance of Medusa in **worker mode** to handle background jobs and workflows efficiently.

Step 1 — Configure Medusa Application

Worker Mode Configuration

The `**workerMode**` setting determines how the Medusa application operates. We will deploy two separate Medusa instances: one in **server mode** and the other in **worker mode.**

Currently, we have the following configuration in our enhanced *medusa-config.ts*:

```
module.exports = defineConfig({
 projectConfig: {
    workerMode: process.env.MEDUSA_WORKER_MODE as "shared" | "worker" | "server"
  }
})
```

Later on, we will assign different values to the **MEDUSA_WORKER_MODE** environment variable for each deployment — ensuring one instance handles **server responsibilities** while the other is dedicated to **worker tasks.**

Configure Medusa Admin

We need to **disable Medusa Admin** in the **worker mode** instance while keeping it **enabled** in the **server mode** instance. This ensures that administrative tasks are only handled by the server deployment, avoiding redundancy.

Add the following configuration to *`medusa-config.ts`:*

```
module.exports = defineConfig({
   // ...
   admin: {
    disable: process.env.DISABLE_MEDUSA_ADMIN === "true"
   }
})
```

When deploying, set the **DISABLE_MEDUSA_ADMIN** environment variable to **true** for the worker instance, and **false** (or leave it unset) for the server instance.

Configure Redis URL

The **redisUrl** configuration specifies the connection URL to **Redis**, which is used to store the **Medusa server's session** data and manage caching effectively.

Add the following configuration to *medusa-config.ts:*

```
module.exports = defineConfig({
   projectConfig: {
      redisUrl: process.env.REDIS_URL,
```

```
    }
})
```

Make sure to set the **REDIS_URL** environment variable with your Redis connection URL to enable efficient session management and caching for your Medusa instance.

Step 2 — Add pre-deploy Script

Before starting the Medusa application in production, it is essential to **run database migrations** to ensure all tables and fields are up to date, and to **synchronize links** for consistency.

To automate this, add the following **pre-deploy script** to your *`package.json`:*

So, add the following script in *`package.json`:*

```
"scripts": {
   // ...
   "predeploy": "medusa db:migrate"
}
```

Step 3— Install Production Modules and Providers

By default, the Medusa application uses modules and providers that are more suitable for **development**, such as the **In-Memory Cache** and the **Local File Module Provider.** However, for production environments, it is highly recommended to use more robust modules that can handle scaling and reliability needs.

Replace development modules with **production-grade alternatives**, including:

- **Redis Cache Module:** Improves caching efficiency and scalability.
- **Redis Event Bus Module:** Facilitates real-time event-driven communication in a scalable manner.
- **Workflow Engine Redis Module:** Manages complex workflows and background jobs reliably.

These production modules **have already been added in the enhanced `*medusa-config.ts*`:**

```
import { Modules } from "@medusajs/framework/utils"

module.exports = defineConfig({
   // ...
   modules: [
      {
       resolve: "@medusajs/medusa/cache-redis",
       options: {
        redisUrl: process.env.REDIS_URL,
       },
      },
      {
       resolve: "@medusajs/medusa/event-bus-redis",
       options: {
        redisUrl: process.env.REDIS_URL,
       }
      },
      {
       resolve: "@medusajs/medusa/workflow-engine-redis",
       options: {
        redis: {
         url: process.env.REDIS_URL,
        }
       }
     }
   ]
```

```
})
```

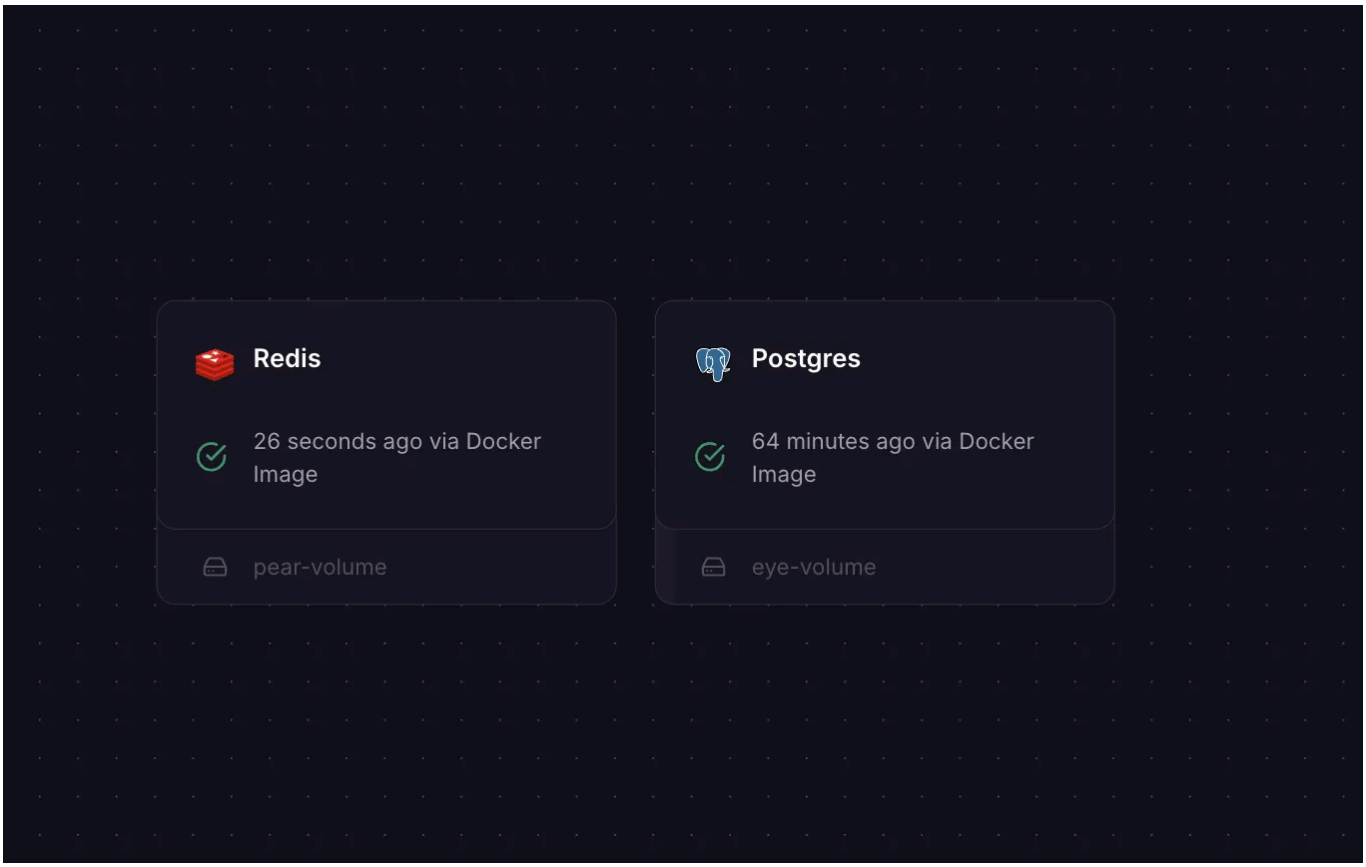
Using these modules will ensure the Medusa application is optimized for **performance** and **reliability** in production.

Step 4— Add Redis Database to Project

To add a Redis database service to your project on Railway:

1. **Click the Create Button:** Navigate to the top-right corner of the Railway dashboard and click + **Create.**
2. **Select Database:** Choose **Add Redis** from the available options to add a Redis service to your project.

This Redis instance will be used for caching, event management, and workflow orchestration in your Medusa application.



Step 5 — Deploy Medusa Application in Server Mode (with Admin)

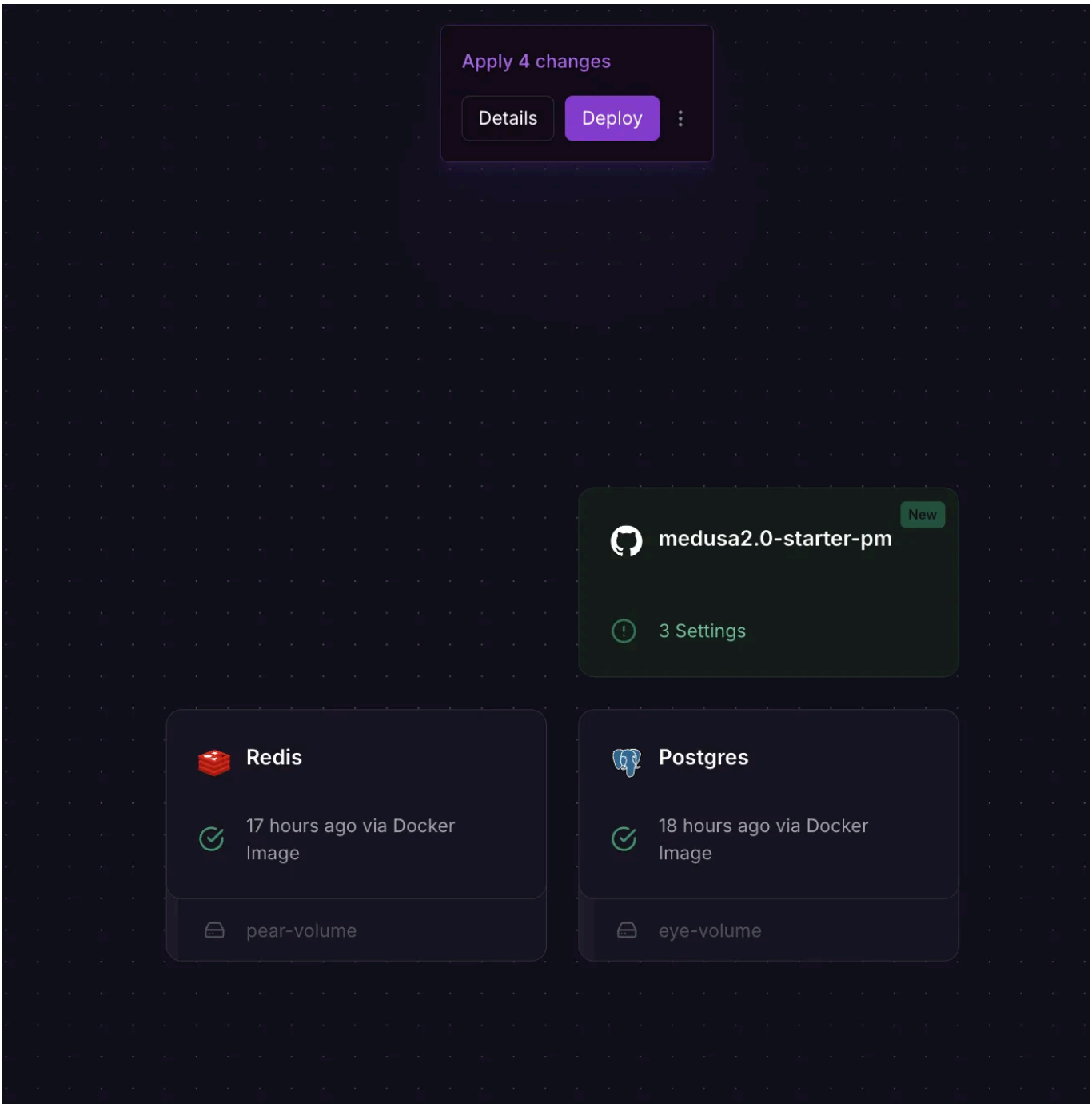
Server vs. Worker Mode Explained:

- **Server Mode:** In **server mode**, Medusa handles all **API requests**, processes **administrative tasks**, and serves both the **storefront** and **admin dashboard.** This is where customers and administrators interact with the e-commerce platform — managing products, processing orders, etc.
- **Worker Mode:** In **worker mode**, Medusa focuses on running **background tasks** such as managing events, processing long-running jobs, or handling workflows. Worker instances help offload time-consuming operations to keep the server instance responsive for incoming requests.

Create Service for Medusa in Server Mode:

1. **Click the Create Button:** In the Railway dashboard, click on the **+ Create** button.
2. **Choose GitHub Repository:** Select **GitHub Repo** as the source.
3. **Select Your Medusa Repository:** Choose the repository containing your Medusa application.

This will create a new service in your Railway project, deploying Medusa in **server mode** to handle API requests and the admin interface.



To configure your Medusa application in **server mode**, you need to add several **environment variables.** Follow these steps:

Add Environment Variables:

1. **Access the Medusa Service:** Click on the **Medusa server** card in your Railway project dashboard.

2. **Open Variables Tab:** Navigate to the **Variables** tab.
3. **Use the RAW Editor:** Click on **RAW Editor** and paste the following environment variables:

```
COOKIE_SECRET=supersecret # TODO: Generate a secure secret JWT_SECRET=supersecre
STORE_CORS= # Set to your storefront
URL ADMIN_CORS= # Set to the admin
URL AUTH_CORS= # Add storefront and admin URLs, separated by commas DISABLE_MEDU
MEDUSA_WORKER_MODE=server PORT=9000 DATABASE_URL=${{Postgres.DATABASE_URL}}
REDIS_URL=${{Redis.REDIS_URL}}?family=0
```

Explanation of Variables:

- **COOKIE_SECRET** & **JWT_SECRET:** These must be **randomly generated secrets** to ensure application security.
- **STORE_CORS:** The URL of your **storefront**. Leave empty for now if it's not yet set up.
- **ADMIN_CORS:** The URL for the **admin dashboard.** This will be the same as your server instance URL. You can add it later if it's not available at the moment.
- **AUTH_CORS:** The URLs of any apps authenticating users (e.g., storefront and admin). Separate multiple URLs with commas. This value can also be set later if needed.
- **DISABLE_MEDUSA_ADMIN:** Set to **false** to ensure the admin dashboard is included in the server application.
- **REDIS_URL:** Automatically populated using Railway's template syntax. Adding **?family=0** supports both **IPv6** and **IPv4** connections.

You can also add any other relevant environment variables necessary for your setup. Once you've entered everything, click the **Update Variables** button to save.

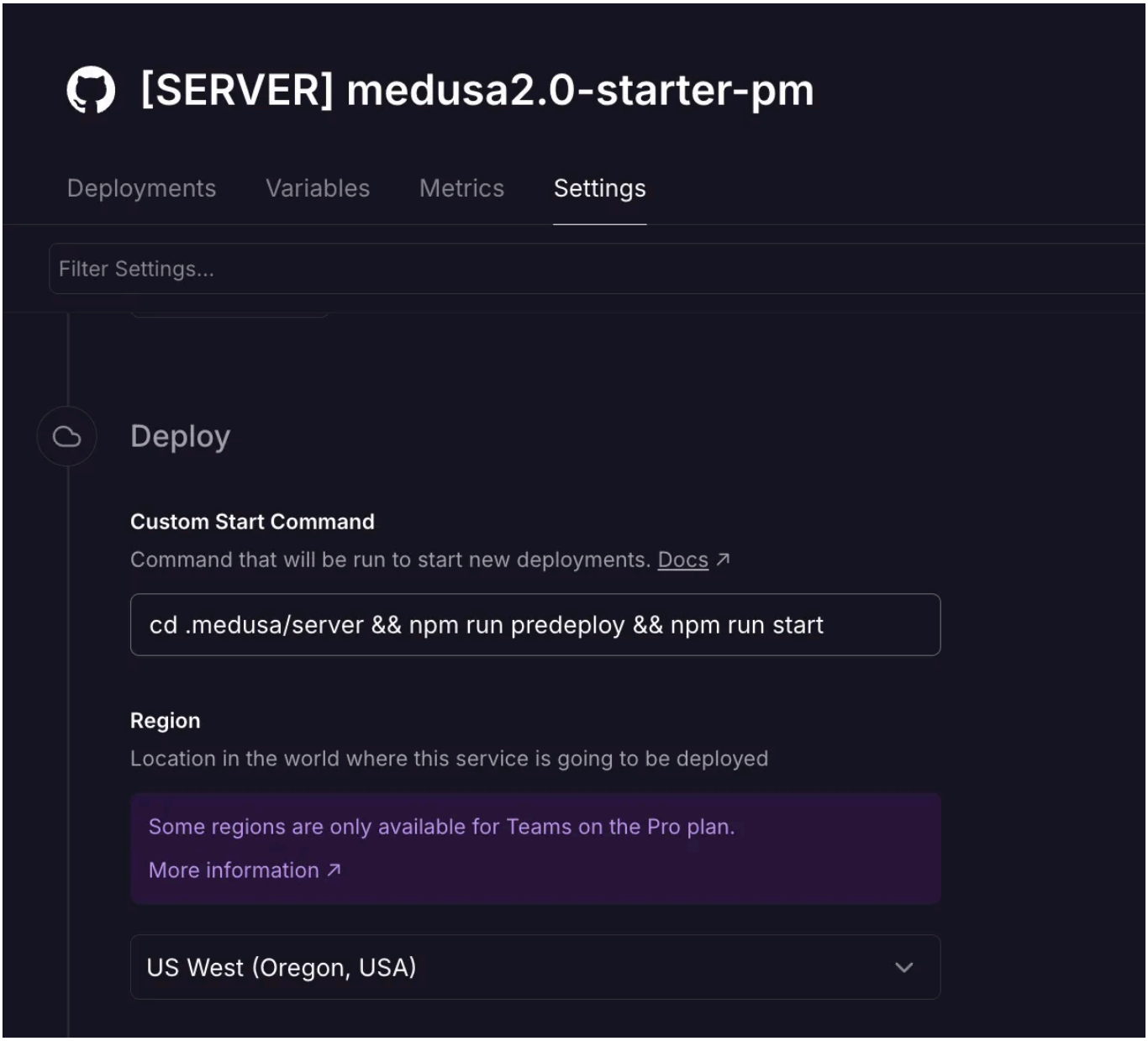
Set Start Command for Medusa Server Instance

To configure the **start command** for your Medusa application running in **server mode**, follow these steps:

Set Start Command:

1. **Access the Medusa Server Service**: Click on the **Medusa server** card in your Railway project dashboard.
2. **Open Settings Tab**: Navigate to the **Settings** tab.
3. **Locate Deploy Section**: Scroll down to the **Deploy** section.
4. **Enter Custom Start Command**: In the **Custom Start Command** field, enter the following:

```
cd .medusa/server && npm run predeploy && npm run start
```



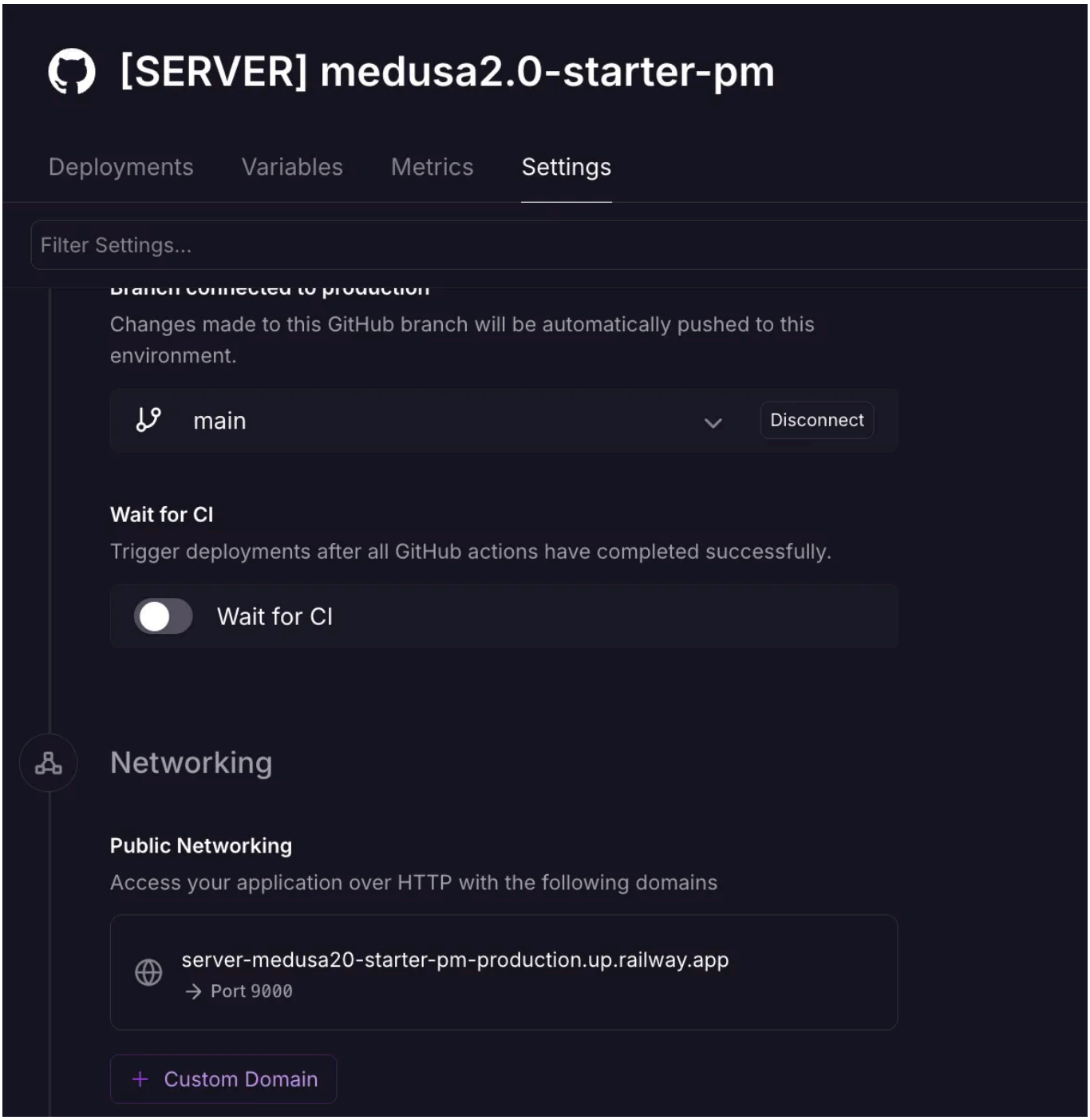
Deploy the Medusa Server Application:

To deploy your Medusa application in **server mode**, click the **Deploy** button at the top-center of the Railway project dashboard. The deployment process will take a few minutes to complete.

You can either **generate a random domain name** or set a **custom domain name** for your Medusa server. Follow these steps:

1. **Access Medusa Server Service:** Click on the **Medusa server** card in your Railway dashboard.
2. **Open Settings Tab**: Navigate to the **Settings** tab.
3. **Navigate to Networking Section**: Scroll down to the **Networking** section.
4. **Generate or Set Custom Domain**:
   — Under **Public Networking**, click **Generate domain** to create a random domain name.
   — Alternatively, click **Custom domain** to add your own custom domain.
   — Select **port 9000** to expose your Medusa server application properly.
5. **Save Changes**: Make sure to save your changes after configuring the domain.

This allows you to access your Medusa application through a public URL, either randomly generated or your own custom domain, ensuring ease of access for users and administrators.



Once the deployment is complete, you can verify if your Medusa application is up and running by visiting:

```
<RAILWAY_APP_URL>/health
```

Replace **<RAILWAY_APP_URL>** with the actual domain you configured. If everything is set up correctly, you should see a health status response confirming that your Medusa server is running smoothly.

Set Backend URL in Admin Configuration

After obtaining the URL for your Medusa application, update the **admin configuration** in *`medusa-config.ts`* (we already included this in our enhanced *`medusa-config.ts`*) to ensure that the admin dashboard can communicate with the backend:

Add Admin Backend URL Configuration:

```
module.exports = defineConfig({

   admin: {
    backendUrl: process.env.BACKEND_URL,
   }
})
```

Next, **push these changes** to your GitHub repository to reflect the updated configuration.

In your Railway dashboard, add or modify the following environment variables for the Medusa application in **server mode**:

Explanation:

- **ADMIN_CORS:** Set the value to the URL of your **Medusa application** to allow admin access from that domain.
- **AUTH_CORS:** Add the **Medusa application URL** here to enable authentication for the storefront and admin.
- **BACKEND_URL** Automatically generated from the **Railway public domain**. This allows the admin dashboard to know where the backend is located.

Once these environment variables are set correctly, click **Update Variables** to save the changes.

Step 6— Deploy Medusa Application in Worker Mode

The deployment process for the Medusa application in **worker mode** is similar to deploying it in **server mode,** with a few adjustments to the configuration.

Create Service for Medusa in Worker Mode:

1. **Click the Create Button:** On the Railway dashboard, click on the **+ Create** button.
2. **Select GitHub Repository:** Choose **GitHub Repo** as the source.
3. **Select Your Medusa Repository:** Pick the repository that contains your Medusa application.

This will create a new service in your Railway project, specifically for the **worker mode** deployment.

Add Environment Variables

To set up the required **environment variables** for the Medusa application in **worker mode:**

1. **Access Worker Service:** Click on the **Medusa worker** card in the Railway project dashboard.

2. **Open Variables Tab:** Navigate to the **Variables** tab.

3. **Use the RAW Editor:** Click **RAW Editor** and paste the following configuration:

```
COOKIE_SECRET=supersecret # TODO GENERATE SECURE SECRET
JWT_SECRET=supersecret  # TODO GENERATE SECURE SECRET
DISABLE_MEDUSA_ADMIN=true
MEDUSA_WORKER_MODE=worker # ====> KEY CHANGE (!)
PORT=9000
DATABASE_URL=${{Postgres.DATABASE_PUBLIC_URL}}
REDIS_URL=${{Redis.REDIS_PUBLIC_URL}}?family=0
```

Explanation of Environment Variables:

- **COOKIE_SECRET** & **JWT_SECRET:** Use **randomly generated secrets** to maintain security.
- **STORE_CORS, ADMIN_CORS, AUTH_CORS:** In **worker mode,** these values are optional as the worker doesn't interact directly with the storefront or admin interface.
- **DISABLE_MEDUSA_ADMIN:** Set to **true** to disable the admin panel for the worker instance, ensuring no administrative interface is built.
- **MEDUSA_WORKER_MODE:** Set to **worker** to indicate this instance runs in

worker mode, focusing on background tasks.
- **REDIS_URL:** Add **?family=0** to support both **IPv6** and **IPv4**.

After adding these variables, click the **Update Variables** button to save the configuration.

Set Start Command

To configure the **start command** for your Medusa application in **worker mode**, follow these steps:

1. **Access Worker Service:** Click on the **Medusa worker** card in your Railway project dashboard.

2. **Open Settings Tab:** Navigate to the **Settings** tab.

3. **Locate Deploy Section:** Scroll down to the **Deploy** section.

4. **Enter Custom Start Command:** In the **Custom Start Command** field, enter:

```
cd .medusa/server && npm run start
```

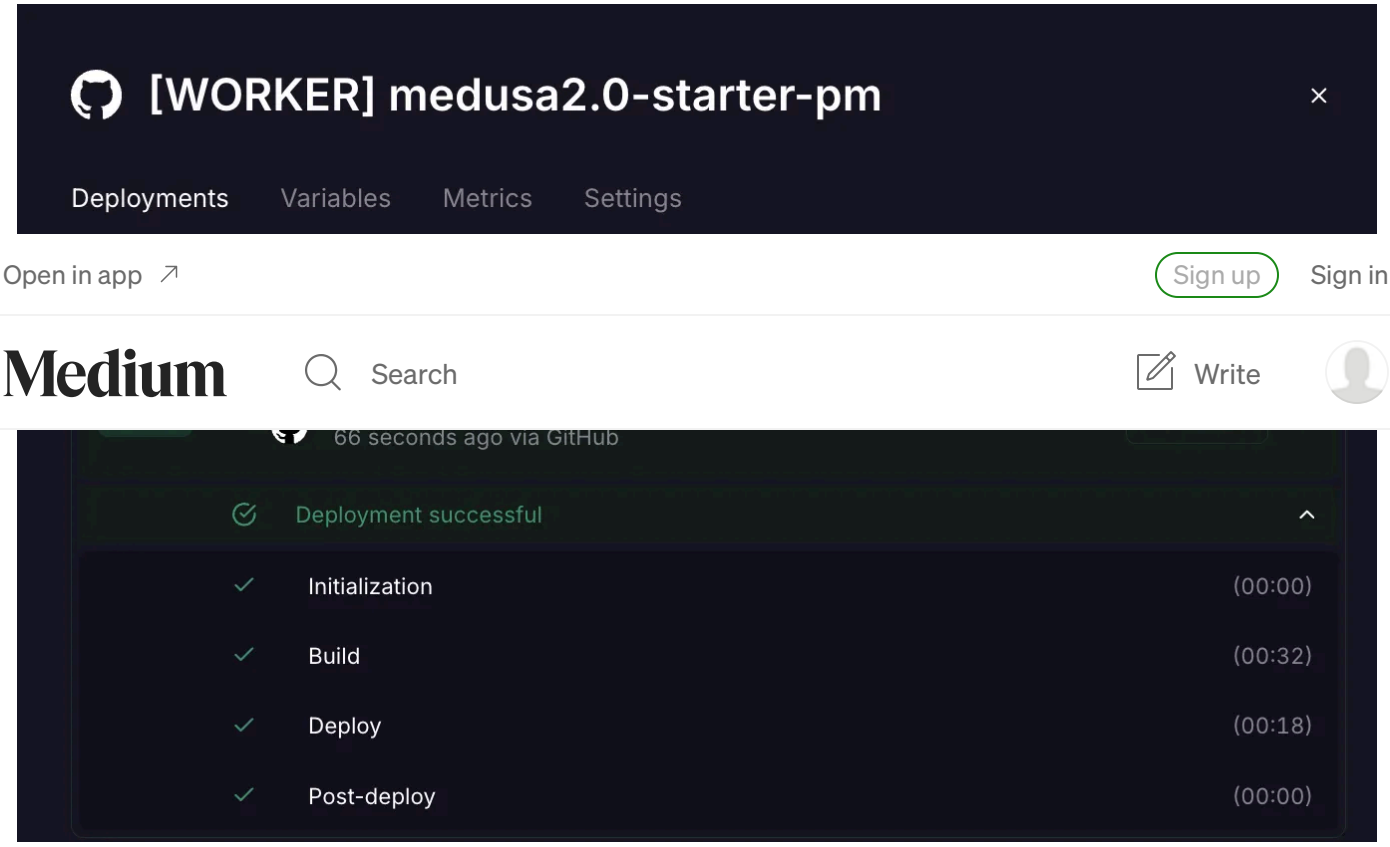
5. **Save the Command:** Click the **check mark** button to save the custom start command.

After setting up the environment variables and start command, it's time to deploy your Medusa application in **worker mode**.

Deploy Changes:

- Click the **Deploy** button at the top-center of the Railway project dashboard to deploy the worker instance of your Medusa application.
- The deployment process typically takes a few minutes.

Once complete, the worker instance will be running and ready to handle **background tasks** like **event handling**, **workflows**, and **job processing**, ensuring that the **server mode** instance remains responsive for API requests and other user interactions.



Step 7— Test the Deployed Application

Once the deployment is complete, it's important to verify that everything is running smoothly.

Test Health Check:

- Go to **<APP_URL>/health**, where **<APP_URL>** is the URL of your Medusa application in **server mode.**
- If the deployment was successful, you should see an **OK** response, indicating that the server is up and running.

Access Medusa Admin:

- You can also access the **Medusa Admin Dashboard** at **<APP_URL>/app.**
- This interface allows you to manage products, orders, and other e-commerce operations directly.

This simple test will confirm that your **Medusa server instance** is operational and the **admin dashboard** is accessible.

Happy coding :)

Piotr

Medusajs | E Commerce Solution | React | Web Development

Written by PiotrDev

89 followers · 14 following

15 years in Payments | Banking | Finance (FT500). MBA Warwick Business School. Full Stack Developer (JS, React, Python) | www.piotrmaciejewski.com

Follow

Responses (1)

Write a response

What are your thoughts?

Ahmed mostapha
Jan 30

how to install medusa 2 in online vps?

Reply

More from PiotrDev

PiotrDev

PiotrDev

NextAuth Credentials—easy signup & login with email &...

NEXTAUTH & ZOD INTRO

Feb 3, 2024 250 9

Python with Pgeocode and Pandas. Postal codes to geo coordinates.

In some projects, you may need to retrieve geo coordinates from the set of zip codes....

Mar 4, 2023 107 1

PiotrDev

Reverse Geocode with Python, real example with reverse_geocode.

Reverse Geocode takes a latitude/longitude coordinate and returns the country and city.

Mar 4, 2023 111 2

PiotrDev

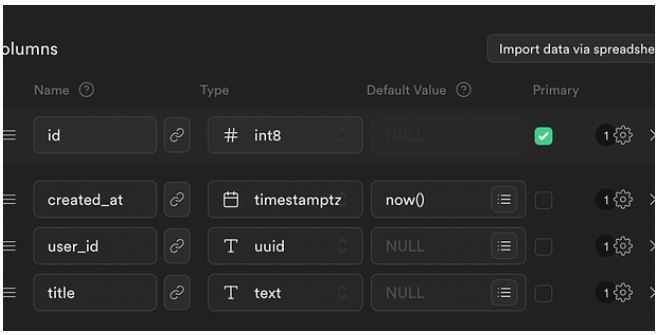
Implementing an AI-Powered Shopping Assistant with Vercel AI...

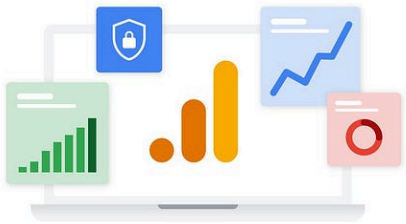
This tutorial is part of my experiments with MedusaJS to create modern, scalable, and...

Jun 10, 2024 30 1

See all from PiotrDev

Recommended from Medium





Girff

Authentication with Next.js and Supabase

In this guide, we'll build an example Next.js server-side rendered application with…

Dec 27, 2024

CodeZera

Integrating Mixpanel with Next.js

Let's be real, If your app has even slightly more going on than a basic CRUD, you'll 100…

Apr 7

In ENTech Solutions by Eric Popivker

Push Notifications in Next.js and Firebase with Demo and Full Code

Push notifications are a powerful way to re-engage users with your Next.js application….

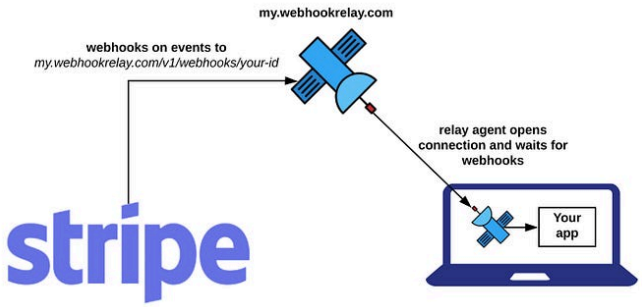
Mar 19 3

In T3CH by Adrian Pothuaud

Feature-Driven Development with TypeScript, Express, and Zod: A…

Today, I'm excited to share with you a game-changing approach that has transformed ho…

6d ago 60



Muhammad Ali Bhutta

How to Implement Stripe Webhooks and Listen for Events i...

Integrating Stripe into your application? Whether you're handling subscription event...

Jan 1

In Let's Code Future by CodeWithYog

Top 15 Open Source Next.js Projects Every Dev Should Clone i...

Hello, this is Yogesh, welcome in CodeWithYog. I spend a lot of time with code...

2d ago 120 6

See more recommendations