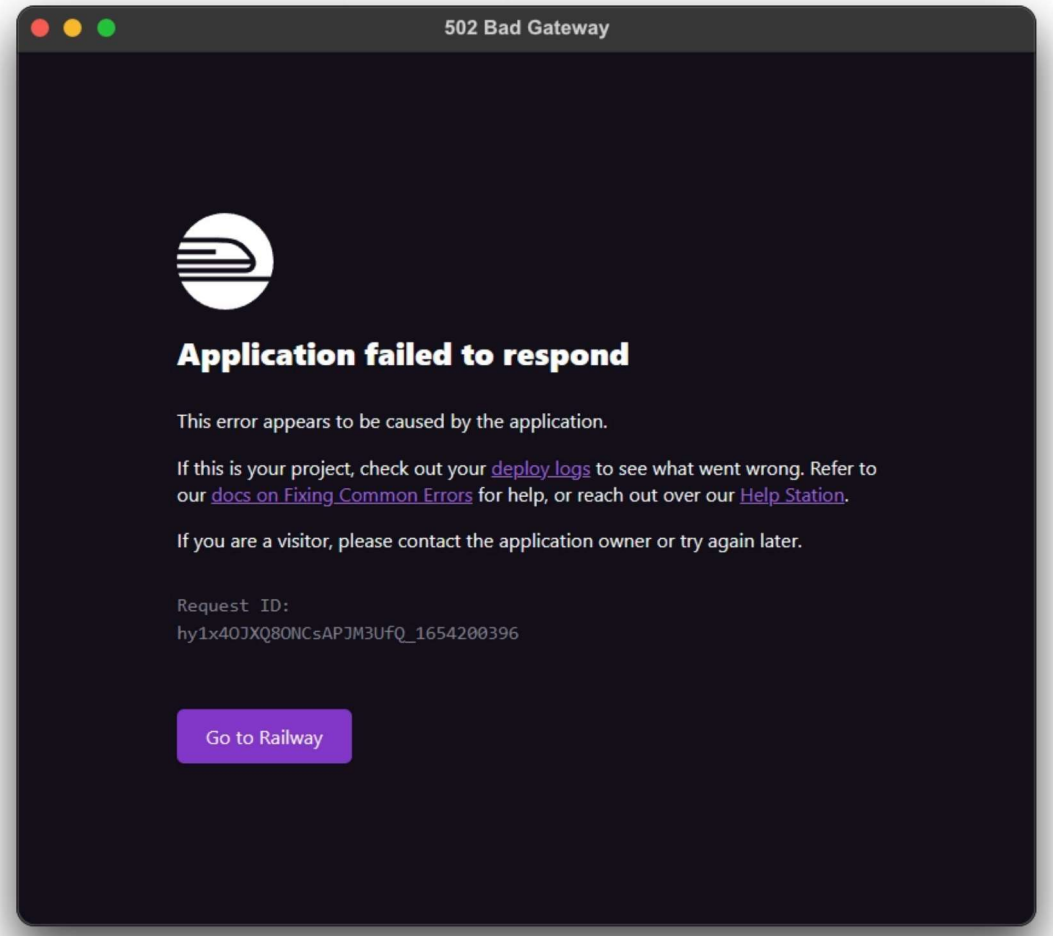
Search

Application Failed to Respond



What This Error Means

Seeing that your application failed to respond means that Railway's Edge Proxy cannot communicate with your application, causing your request to fail with a 502 (Bad Gateway) status code.

Why This Error Can Occur

There are a few reasons why this error can occur, the most common being that your application is not listening on the correct host or port.

Another common reason is that your target port is set to an incorrect value.

In some far less common cases this error can also occur if your application is under heavy load and is not able to respond to the incoming request.

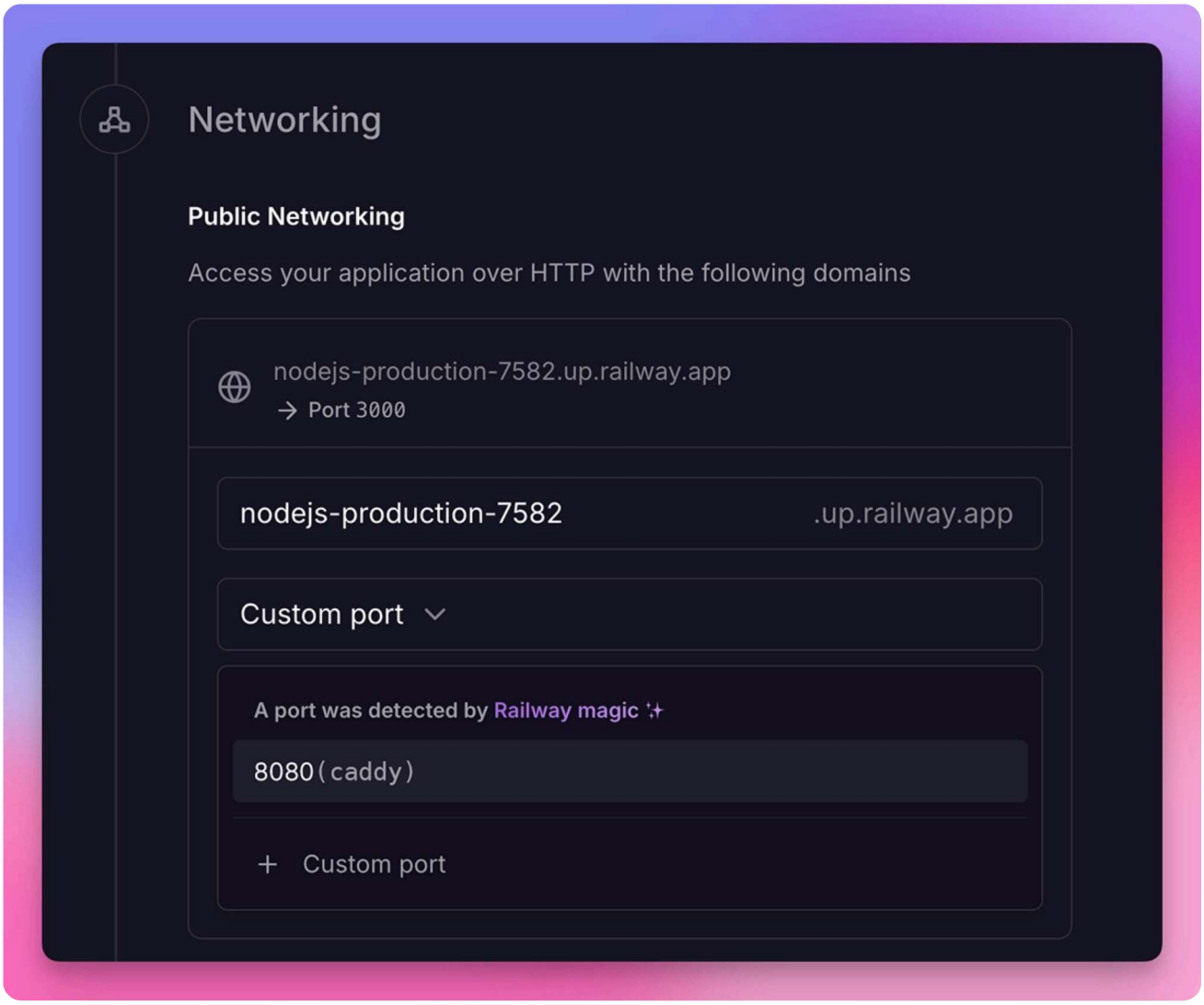
Possible Solutions

The correct solution depends on the cause of the error.

Target port set to the incorrect value

If your domain is using a target port, ensure that the target port for your public domain matches the port your application is listening on.

This setting can be found within your service settings.



In the screenshot above, the domain was previously incorrectly configured with port 3000, when the application was actually listening on port 8080.

Application Not Listening on the Correct Host or Port

Your web server should bind to the host `0.0.0.0` and listen on the port specified by the `PORT` environment variable, which Railway automatically injects into your application.

Start your application's server using:

- Host = `0.0.0.0`
- Port = Value of the `PORT` environment variable provided by Railway.

Below are some solution examples for common languages and frameworks.

Node / Express

```
// Use PORT provided in environment or default to 3000
const port = process.env.PORT || 3000;

// Listen on `port` and 0.0.0.0
app.listen(port, "0.0.0.0", function () {
  // ...
});
```

Node / Nest

```
// Use `PORT` provided in environment or default to 3000
const port = process.env.PORT || 3000;

// Listen on `port` and 0.0.0.0
async function bootstrap() {
  // ...
  await app.listen(port, "0.0.0.0");
}
```

Node / Next

Next needs an additional flag to listen on `PORT`:

```
next start --port ${PORT-3000}
```

Python / Gunicorn

`gunicorn` listens on `0.0.0.0` and the `PORT` environment variable by default:

```
gunicorn main:app
```

Python / Uvicorn

`uvicorn` needs additional configuration flags to listen on `0.0.0.0` and `PORT`:

```
uvicorn main:app --host 0.0.0.0 --port $PORT
```

Go / net/http

This example is for `net/http` in the Go standard library, but you can also apply this to other frameworks:

```
func main() {
  // ...
  // Use `PORT` provided in environment or default to 3000
  port := cmp.Or(os.Getenv("PORT"), 3000)

  log.Fatal(http.ListenAndServe((":" + port), handler))
  // ...
}
```

Application Under Heavy Load

If you think your application could be under heavy load, you can confirm this by checking the `Metrics` tab within your service panel.

For example, if you are running a Node.js application, and see that your vCPU usage has peaked at any point to around 1 vCPU, this is a good indication that your application is under heavy load given Node's single-threaded nature.

If this is the case, you can scale your application horizontally to handle more