

Serverless Fns

(<https://projects.100xdevs.com/tracks/e0oSv7InuwBO6wl9YA5w/serverless-1>)

What are backend servers??



You might've used express to create a Backend server.

The way to run it usually is node index.js which starts a process on a certain port (3000 for example)

When you have to deploy it on the internet, there are a few ways -

Go to aws, GCP, Azure, Cloudflare

1. Rent a VM (Virtual Machine) and deploy your app (renting atleast a server)
2. Put it in an Auto scaling group (Zomato etc dont have single server and people hitting on it , it will autoscale as the user grows and have more than one servers) (renting more than one server)

3. Deploy it in a Kubernetes cluster

(Why should i buy a server suppose for \$20 dollar a month when we are getting suppose only 100 visit a day wont it be better to pay on the per request basis)
(Suppose i am hosting it in a single server and suddenly thousands etc request comes that single server wont be able to handle all the request, and we have to autoscale)

There are a few downsides to doing this -

1. Taking care of how/when to scale
2. Base cost even if no one is visiting your website
3. Monitoring various servers to make sure no server is down (what if this server goes down when i am not available)

What if, you could just write the code and someone else could take care of all of these problems?

AWS / GCP / Azure are cloud providers

We host it in cloud.

Very big data centers in India/ Us/ Uk and we can rent a small part of their server.

Before AWS, people buy hardware servers to host the applications.

What are serverless Backends

"Serverless" is a backend deployment in which the cloud provider dynamically manages the allocation and provisioning of servers. The term "serverless" doesn't mean there are no servers involved.

Instead, it means that developers and operators do not have to worry about the servers. We as developers don't worry about the server.

No one is visiting there will be no server.

Easier Definition

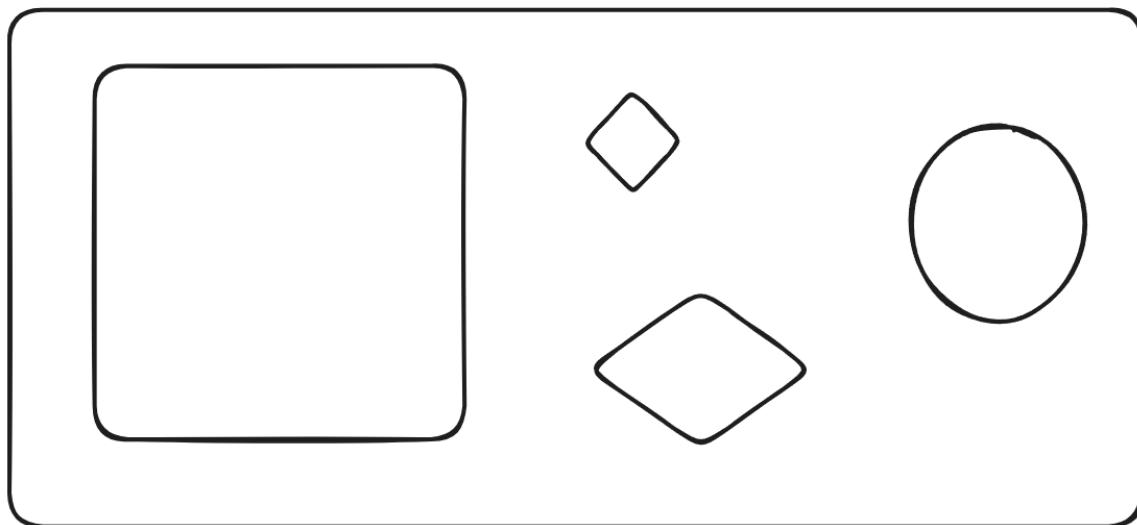
What if you could just write your **express routes** and run a command. The app would automatically

1. Deploy
2. Autoscale
3. Charge you on a per request basis (rather than you paying for VMs)

Problem with this approach

1. More expensive at scale
2. Cold start problem

Per request basis



There is no server not even miniscule server running if no one is visiting suppose ntc.com . What if suddenly someone comes (suppose no one has visited our app for long time), it need to start the server(very small container) . First time someone comes they will see high latency suppose 0.5 sec Few ways to fix is ping your website every 5 sec Or warm pool where we want minimum atleast one or two server running. This is cold start problem

Famous serverless Providers

There are many famous backend serverless providers —

- AWS lambda

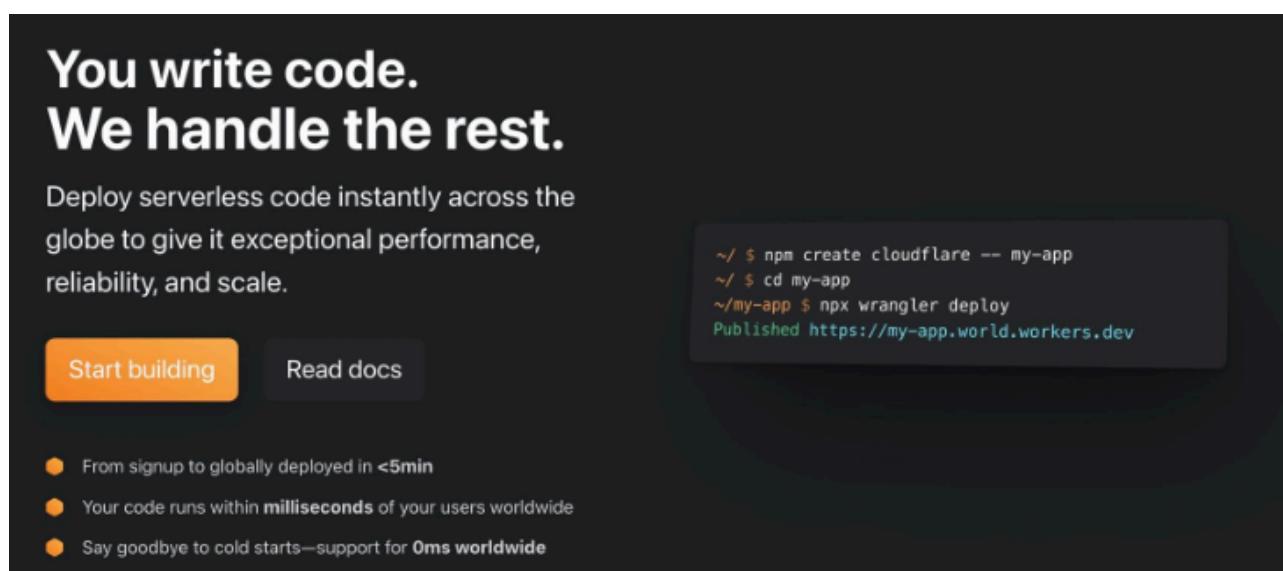
(https://aws.amazon.com/pm/lambda/?trk=5cc83e4b-8a6e-4976-92ff-7a6198f2fe76&sc_channel=ps&ef_id=CjwKCAiAt5euBhB9EiwAdkXWO-i-th4J3onX9ji-tPt_JmsBAQJLWYN4hzTF0Zxb084EkUBxSCK5vhoC-1wQAvD_BwE:G:s&s_kwcid=AL!4422!3!651612776783!e!!g!!aws%20lambda!19828229697!143940519541)

- Google cloud Functions

(<https://firebase.google.com/docs/functions>)

- Cloudflare Workers

(<https://workers.cloudflare.com/>)



The image shows the Cloudflare Workers landing page. It features a dark background with white text. At the top, it says "You write code. We handle the rest." Below that, it says "Deploy serverless code instantly across the globe to give it exceptional performance, reliability, and scale." There are two buttons: "Start building" (orange) and "Read docs". To the right, there is a terminal-like box showing deployment commands: `~/ $ npm create cloudflare -- my-app`, `~/ $ cd my-app`, `~/my-app $ npx wrangler deploy`, and `Published https://my-app.world.workers.dev`. Below the terminal box, there are three orange bullet points: "From signup to globally deployed in <5min", "Your code runs within milliseconds of your users worldwide", and "Say goodbye to cold starts—support for 0ms worldwide".

We will use cloudflare (for now)

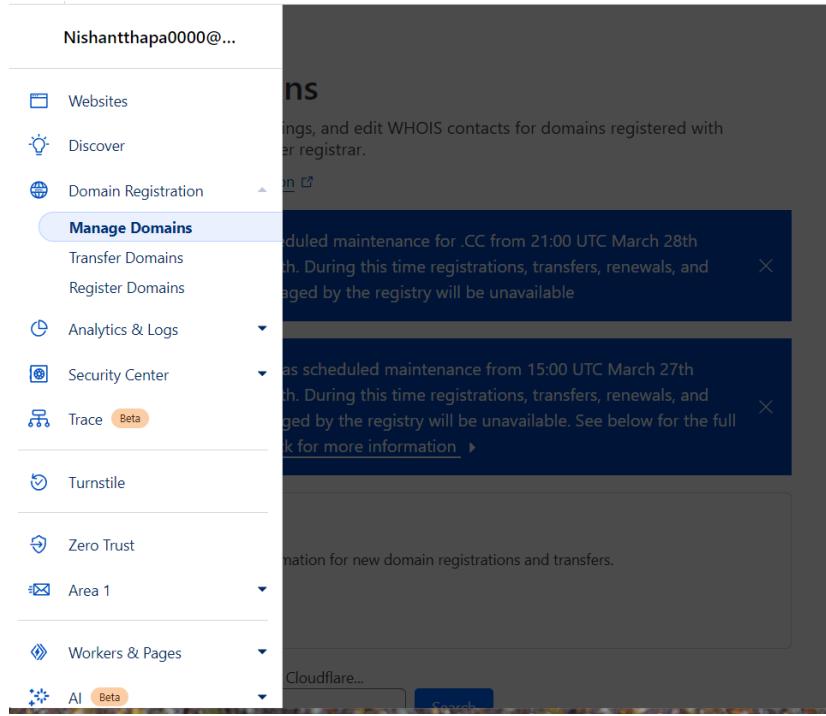
When should you use a serverless architecture

1. When you have to get off the ground fast and don't want to worry about deployments
2. When you can't anticipate the traffic and don't want to worry about autoscaling
3. If you have very low traffic and want to optimise for costs

Cloudflare workers setup

Cloudflare prevent ddos attack .

Dashboard



We will go to Workers & Pages tab

Workers documentation: <https://workers.cloudflare.com/docs>

Pages documentation: <https://developers.cloudflare.com/pages/>

Workers let us write some basic frontend and backend code and let us deploy it

Thanks for verifying your email address.

Get started with Workers & Pages

Build serverless functions with Workers. Deploy websites and full-stack applications with Pages. Read the [Workers documentation](#) and [Pages documentation](#) to learn more.

[Workers](#) [Pages](#)



Create a “Hello World” Worker and deploy across the globe

[Create Worker](#)

Create using a template

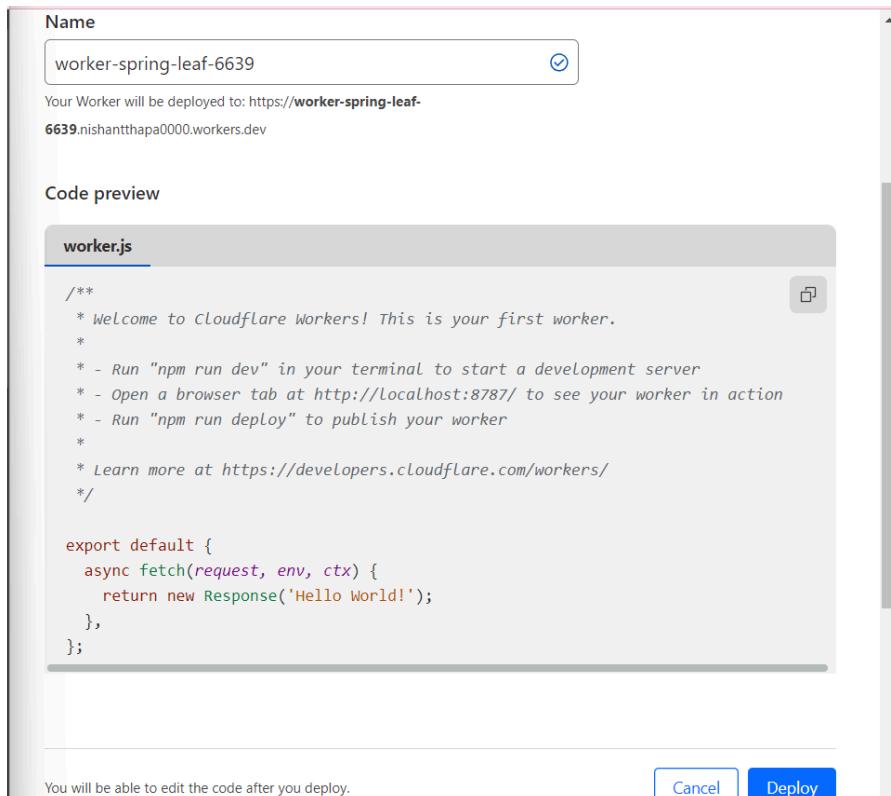
Common Worker examples
Build from a collection of simple Worker use cases.

[Use template →](#)

<https://dash.cloudflare.com/7bdbd60c63e8cef420ff1529a6e68c58/workers-and-pages/create/workers/new?template=basic-bundle>

Try creating a test worker from the UI (Common worker example) and try hitting the URL at which it is deployed.

We will deploy it



After we deploy

CLOUDFLARE

Congratulations! Your Worker is deployed to Region: Earth.

Your Worker is now milliseconds away from virtually every Internet user.

Preview your Worker

<https://worker-spring-leaf-6639.nishanthappa0000.workers.dev>

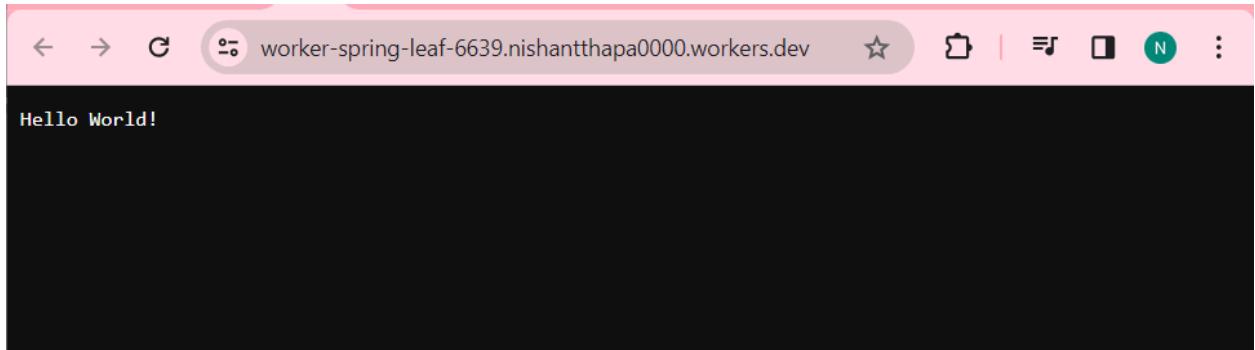
[Configure Worker](#) [Edit code](#)

Learn more

- [Workers documentation](#)
- [View projects built using Workers](#)
- [Join Cloudflare's developer Discord](#)

It shows the place where they can deploy the server based on from where the request coming.

When we preview our worker we see



How cloudflare worker work?

Blog about it:

<https://developers.cloudflare.com/workers/reference/how-workers-works/#:~:text=Though%20Cloudflare%20Workers%20behave%20similarly,used%20by%20Chromium%20and%20Node>

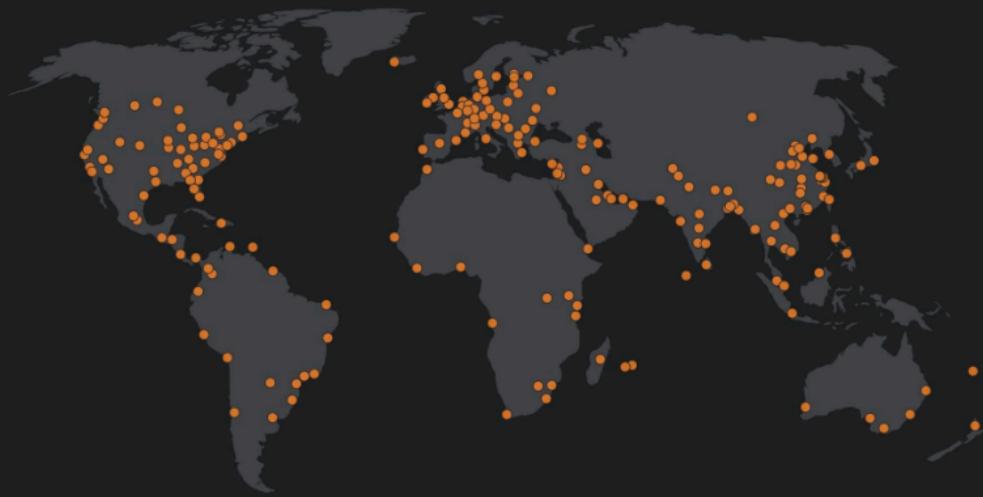
Cloudflare workers DONT use the Node.js runtime. They have created their own runtime. There are a lot of things that Node.js has
How it works

Worker runtime uses V8 engine.

How cloudflare workers work? (6 / 12)

Though Cloudflare Workers behave similarly to [JavaScript](#) in the browser or in Node.js, there are a few differences in how you have to think about your code. Under the hood, the Workers runtime uses the [V8 engine](#) — the same engine used by Chromium and Node.js. The Workers runtime also implements many of the standard [APIs](#) available in most modern browsers.

The differences between JavaScript written for the browser or Node.js happen at runtime. Rather than running on an individual's machine (for example, [a browser application](#) or [on a centralized server](#)), Workers functions run on [Cloudflare's Edge Network](#) — a growing global network of thousands of machines distributed across hundreds of locations.



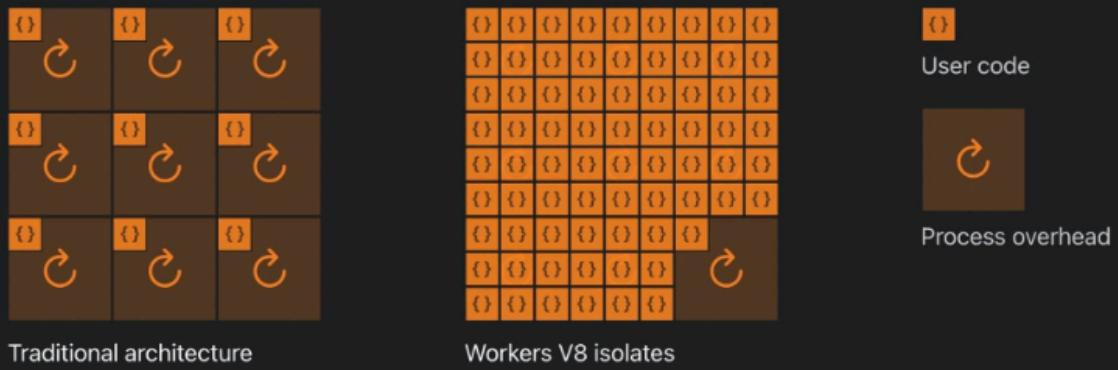
Each of these machines hosts an instance of the Workers runtime, and each of those runtimes is capable of running thousands of user-defined applications. This guide will review some of those differences.

Isolates Vs Container

Isolates

V8 [orchestrates isolates](#): lightweight contexts that provide your code with variables it can access and a safe environment to be executed within. You could even consider an isolate a sandbox for your function to run in.

A single runtime can run hundreds or thousands of isolates, seamlessly switching between them. Each isolate's memory is completely isolated, so each piece of code is protected from other untrusted or user-written code on the runtime. Isolates are also designed to start very quickly. Instead of creating a virtual machine for each function, an isolate is created within an existing environment. This model eliminates the cold starts of the virtual machine model.



Unlike other serverless providers which use [containerized processes](#) each running an instance of a language runtime, Workers pays the overhead of a JavaScript runtime once on the start of a container. Workers processes are able to run essentially limitless scripts with almost no individual overhead by creating an isolate for each Workers function call. Any given isolate can start around a hundred times faster than a Node process on a container or virtual machine. Notably, on startup isolates consume an order of magnitude less memory.

A given isolate has its own scope, but isolates are not necessarily long-lived. An isolate may be spun down and evicted for a number of reasons:

- Resource limitations on the machine.
- A suspicious script - anything seen as trying to break out of the Isolate sandbox.
- Individual [resource limits](#).

Initializing a Worker

We can't use Express when we use Cloudflare workers

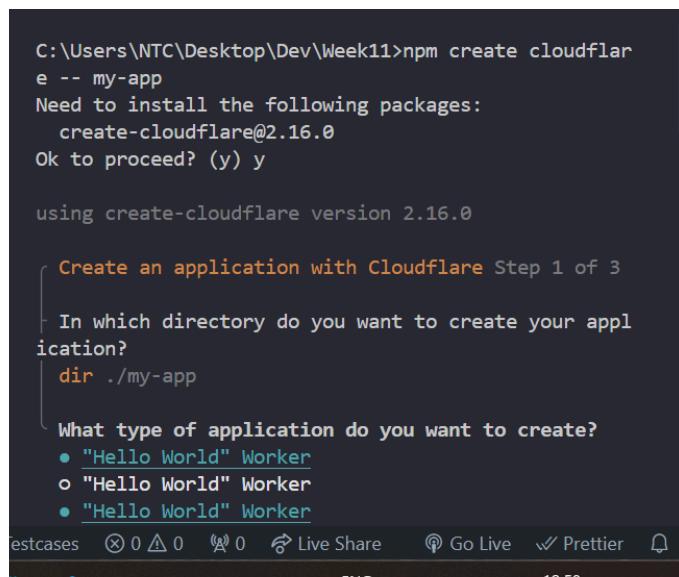
To create and deploy your application, you can take the following steps –

- Initialize a worker

```
npm create cloudflare -- my-app
```

Select no for Do you want to deploy your application?

Because we haven't login the cloudflare in our terminal



```
C:\Users\NTC\Desktop\Dev\Week11>npm create cloudflare -- my-app
Need to install the following packages:
  create-cloudflare@2.16.0
Ok to proceed? (y) y

using create-cloudflare version 2.16.0

  Create an application with Cloudflare Step 1 of 3

  In which directory do you want to create your application?
    dir ./my-app

  What type of application do you want to create?
    • "Hello World" Worker
    o "Hello World" Worker
    • "Hello World" Worker
```

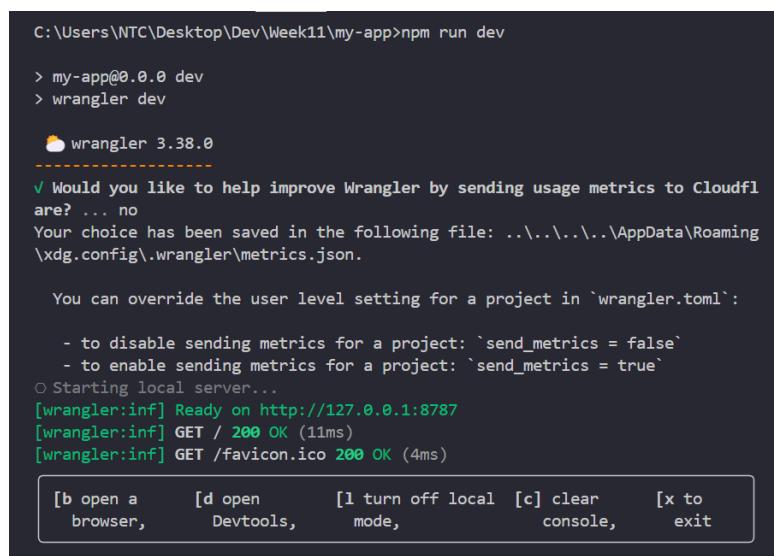
- Explore package.json dependencies\

```
"wrangler": "^3.0.0"
```

Notice express is not in dependencies

- Start the worker locally

```
npm run dev
```



```
C:\Users\NTC\Desktop\Dev\Week11\my-app>npm run dev

> my-app@0.0.0 dev
> wrangler dev

  wrangler 3.38.0
  -----
  ✓ Would you like to help improve Wrangler by sending usage metrics to Cloudflare? ... no
  Your choice has been saved in the following file: ..\..\..\..\AppData\Roaming\wrangler\metrics.json.

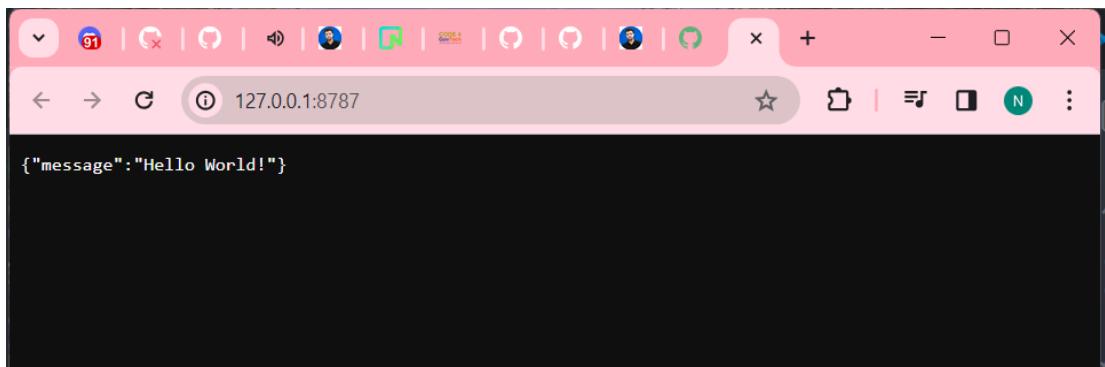
  You can override the user level setting for a project in `wrangler.toml`:
    - to disable sending metrics for a project: `send_metrics = false`
    - to enable sending metrics for a project: `send_metrics = true`

  ⚡ Starting local server...
  [wrangler:inf] Ready on http://127.0.0.1:8787
  [wrangler:inf] GET / 200 OK (11ms)
  [wrangler:inf] GET /favicon.ico 200 OK (4ms)

  [b] open a browser, [d] open Devtools, [l] turn off local mode, [c] clear console, [x] to exit
```

- How to return json?

```
export default {
    async fetch(request: Request, env: Env, ctx: ExecutionContext):
Promise<Response> {
    return Response.json({
        message: "hi"
    });
},
};
```



Where is the express code? HTTP Server?

Cloudflare expects you to just write the logic to handle a request

Creating an HTTP server on top is handled by cloudflare

How can I do routing?

In express routing is done as follows:-

```
import express from "express"
const app = express()

app.get("/route", (req, res) => {
    // handles a get request to /route
});
```

How can you do the same in the cloudflare environment?

```
export interface Env {
```

```
}

// app.get() what was the route, body , headers , query parameter the user
// were sending
// Request object will give access to all these thing in Wrangler
export default {
    async fetch(request: Request, env: Env, ctx: ExecutionContext): Promise<Response> {
        console.log(request.body);
        console.log(request.headers);

        // this will look quite bad when we have more than 20 routes
        if (request.method === "GET") {
            if(uri === '/users'){
                // handler the user request
            }
            return Response.json({
                message: "you sent a get request"
            });
        } else {
            return Response.json({
                message: "you did not send a get request"
            });
        }
    },
};

}
```

Cloudflare does not expect a routing library/http server out of the box. You can write a full application with just the constructs available above.

We will eventually see how you can use other HTTP frameworks (like express) in cloudflare workers.

[Login to cloudfare](#)

npx wrangler login

```
C:\Users\NTC\Desktop\Dev\Week11\my-app>npx wrangler login
wrangler 3.38.0
-----
Attempting to login via OAuth...
Opening a link in your default browser: https://dash.cloudflare.com/oauth2/auth?response_type=code&client_id=54d11594-84e4-41aa-b438-e81b8fa78ee7&redirect_uri=http%3A%2Flocalhost%3A8976%2Foauth%2Fcallback&scope=account%3Aread%20user%3Aread%20workers%3Awrite%20workers_kv%3Awrite%20workers_routes%3Awrite%20workers_scripts%3Awrite%20workers_tail%3Aread%20d1%3Awrite%20pages%3Awrite%20ozone%3Aread%20ssl_certs%3Awrite%20constellation%3Awrite%20ai%3Aread%20offline_access&state=1X-MgIWW8N~Sb1iLhW6aP13W4nzu.Jnio&code_challenge=sfnmebECFVnLfu5YTPFcwAjYY8VhdLbU9usxrACS1g&code_challenge_method=S256
Successfully logged in.
```

```
C:\Users\NTC\Desktop\Dev\Week11\my-app>
```

unctionMade* ⌂ ⌂ Run Testcases ⌂ 0 ▲ 0 ⌂ 0 ⌂ Live Share ⌂ Go Live ⌂ Prettier ⌂

npx wrangler whoami

Give details of account

Account Name	Account ID
Nishantthappa000@gmail.com's Account	7bdbd60c63e8cef420ff1529a6e68c58

```
⚠ Token Permissions: If scopes are missing, you may need to logout and re-login.
Scope (Access)
- account (read)
- user (read)
- workers (write)
- workers_kv (write)
- workers_routes (write)
- workers_scripts (write)
- workers_tail (read)
- d1 (write)
- pages (write)
- zone (read)
- ssl_certs (write)
- constellation (write)
- ai (read)
- offline_access
```

```
C:\Users\NTC\Desktop\Dev\Week11\my-app>
```

unctionMade* ⌂ ⌂ Run Testcases ⌂ 0 ▲ 0 ⌂ 0 ⌂ Live Share ⌂ Go Live ⌂ Prettier ⌂

To deploy it we do

npm run deploy

Give url where our backend is posted.

```
async function sum(a,b){  
    await debugger();  
    return a + bigINt;  
}
```

This function will not return a integer it will return a promise

```
async function sum(a,b): Promise<number>{  
    await debugger();  
    return a + bigINt;  
}
```

We can even see the workers on GUI of the Cloudflare

The screenshot shows the Cloudflare dashboard at dash.cloudflare.com. The top navigation bar includes icons for back, forward, search, and account management. The main title is "Workers & Pages". Below it, a large heading says "Overview". A sub-section titled "Build & deploy serverless functions, sites, and full-stack applications with Workers & Pages. Read the [Workers documentation](#) and [Pages documentation](#) to learn more." features a blue "Create application" button. On the left, there's a search bar with a magnifying glass icon and a "Search" button. To the right are "Filter by" and "Sort by" dropdown menus set to "Show all" and "Last modified".

The main content area displays two worker applications:

- my-app**: Shows "No recent requests" and was last modified 7 minutes ago.
- worker-spring-leaf-6639**: Shows metrics for the last 24 hours: Requests 21, Errors 0, Median CPU Time 0.2 ms. It was last modified a day ago.

At the bottom, there's an "Account details" section indicating a free plan and the time period 12:00AM Wed (UTC) - 1:45PM Wed (UTC).

Deploying a Worker

Now that we have written a basic HTTP server, let's get to the most interesting bit
– Deploying it on the internet

We use wrangler for this (Ref

<https://developers.cloudflare.com/workers/wrangler/>)

Wrangler (command line)

Wrangler, the Cloudflare Developer Platform command-line interface (CLI), allows you to manage Worker projects.

- [Install/Update Wrangler](#): Get started by installing Wrangler, and update to newer versions by following this guide.
- [API](#): An experimental API to programmatically manage your Cloudflare Workers.
- [Bundling](#): Review Wrangler's default bundling.
- [Commands](#): Create, develop, and deploy your Cloudflare Workers with Wrangler commands.
- [Configuration](#): Use a `wrangler.toml` configuration file to customize the development and deployment setup for your Worker project and other Developer Platform products.
- [Custom builds](#): Customize how your code is compiled, before being processed by Wrangler.
- [Deprecations](#): The differences between Wrangler versions, specifically deprecations and breaking changes.
- [Environments](#): Deploy the same Worker application with different configuration for each environment.
- [Migrations](#): Review migration guides for specific versions of Wrangler.
- [Run in CI/CD](#): Deploy your Workers within a CI/CD environment.
- [System environment variables](#): Local environment variables that can change Wrangler's behavior.

Step 1: Login to cloudflare via the wrangler cli

npx wrangler login

Step 2 : Deploy your worker

npm run deploy

Assigning a custom domain

You have to buy a plan to be able to do this

You also need to buy the domain on cloudflare/transfer the domain to cloudflare

Kirags123@gmail.c...

Domain Registration

Register Domain

Buy a new domain at cost.

Domain registration documentation

TLD prices will be changing soon. Click here for more details

Search for a domain name

10kdevs.com

Review instructions and supported extensions.

10kdevs.com is not available

10kdevs.com is unavailable because it is registered already. If you already own this domain and wish to transfer it to Cloudflare, please go to Transfer Domains section in the dashboard.

Suggested domain names

	Domain	Price	Purchase
1	10kdevs.io	\$46.00	Purchase
2	10-kde-vs.com	\$9.77	Purchase
3	10kdevs.net	\$11.84	Purchase
4	10kdevs.uk	\$4.94	Purchase
5	10kdevs.sale	\$25.18	Purchase

Adding express to it.

Why can't we use express? Why does it cloudflare doesn't start off with a simple express boiler plate?

Reason 1: Express heavily relies on Node.js

<https://community.cloudflare.com/t/express-support-for-workers/390844>

123testcoding Jun '22

1 Being a developer, I really love Cloudflare Pages for hosting static apps! But, frontend apps usually need API to get dynamic data. I have existing Express apps that I would like to transfer to Workers, in addition to transferring my frontend app to Cloudflare Pages.

Is it known whether Express support will be added for Cloudflare Workers?

2 3 6.9k 2 6

cherryjimbo MVP '23 Jun '22

It's unlikely you'll see specifically express on Workers due to its deep Node.js dependencies, however there are a lot of options that you'll probably feel right at home with, and have very similar APIs to something like Express. To name just a few:

- GitHub - honojs/hono: Ultrafast web framework for Cloudflare Workers, Deno, Bun, and Node.js. Fast, but not only fast. 818
- GitHub - lukeed/worktop: The next generation web framework for Cloudflare Workers 343
- GitHub - kwhiteley/itty-router: A little router. 287

In addition, if you're running the frontend app in Pages, you may even be able to run your backend in Pages too, using Functions: Functions · Cloudflare Pages docs 140

<https://github.com/honojs/hono>

You can split all your handler in a file

Create a generic handler that you can forward requests to from either express or hono or native cloudflare handler

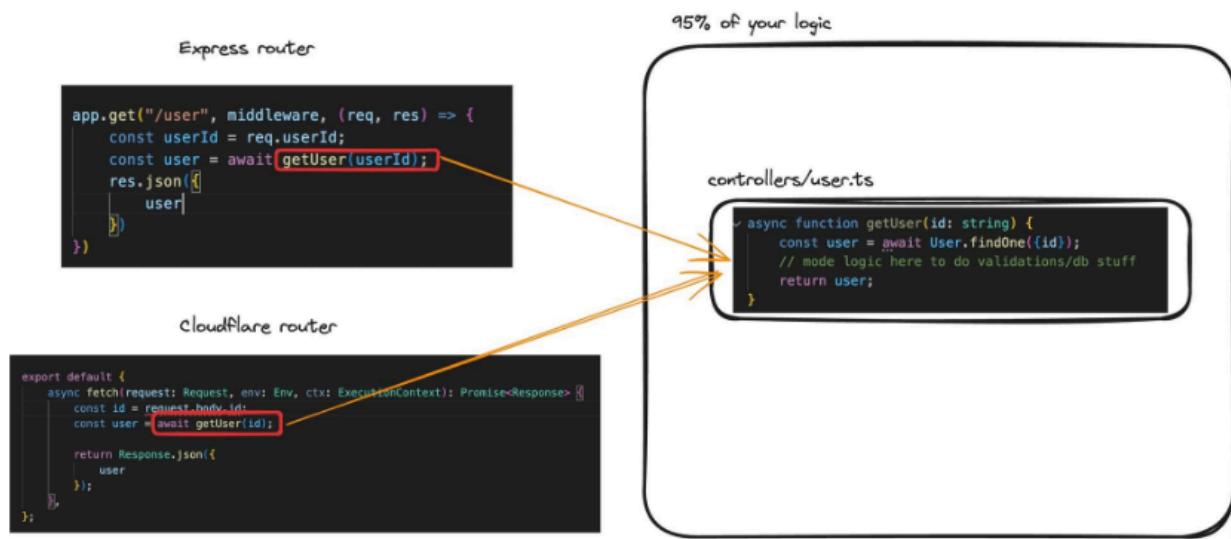
jsmrcaga Jul '23

I can link the Node.js compatibility docs for Workers
<https://developers.cloudflare.com/workers/runtime-apis/nodejs/#nodejs-compatibility> 244

It is important to know that workers run on a different runtime built by Cloudflare (not Node.js). I don't expect compatibility for worker_threads to be arriving anytime soon.

If you really need to deploy a Node.js app I would search more for a classic serverless option. If you can split your app into small components and don't need to rely heavily on Node.js, Cloudflare Workers are an amazing option and there are routing libraries for them as well (ie: to replace express).

Classic serverless option?



Encapsulate as many as we can in general functions

Make code as generic as possible into a file which doesn't depend much on express

We cannot use socket in workers

Using Hono

Motivation

At first, I just wanted to create a web application on Cloudflare Workers. But, there was no good framework that works on Cloudflare Workers, so I started building Hono and thought it would be a good opportunity to learn how to build a router using Trie trees.

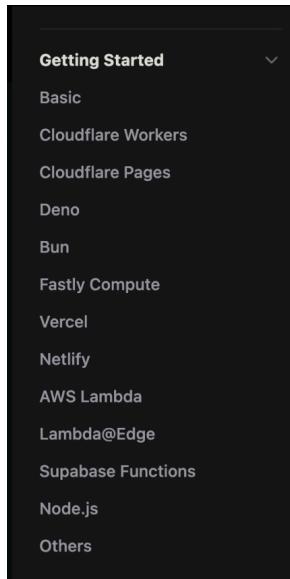
Then a friend showed up with ultra crazy fast router called "RegExRouter". And, I also had a friend who created the Basic authentication middleware.

Thanks to using only Web Standard APIs, we could make it work on Deno and Bun. "No Express for Bun?" we could answer, "No, but there is Hono" though Express works on Bun now.

We also have friends who make GraphQL servers, Firebase authentication, and Sentry middleware. And there is also a Node.js adapter. An ecosystem has been created.

In other words, Hono is damn fast, makes a lot of things possible, and works anywhere. You can look that Hono will become **Standard for Web Standard**.

What runtimes does it support??



Working with cloudflare workers -

Hono is routing something it make easy to write routing logic like express

1. Initialize a new app

```
npm create hono@latest my-app
```

2. Move to my-app and install the dependencies

```
cd my-app
```

```
npm i
```

```
import { Hono } from 'hono'

const app = new Hono()

app.get('/', (c) => {
  return c.text('Hello Hono!')
})

export default app
```

Very similar to express

c give us both res and req

Major things we want are body, headers, query parameters, middlewares, connecting to a database.

Getting inputs from user

```
import { Hono } from 'hono'

const app = new Hono()

app.get('/', async (c) => {
  const body = await c.req.json()
  console.log(body);
  console.log(c.req.header("Authorization"));
  console.log(c.req.query("param"));

  return c.text('Hello Hono!')
})
```

```
export default app
```

<https://hono.dev/getting-started/cloudflare-workers>

Cloudflare examples

Authorization

```
import { Hono } from 'hono'

const app = new Hono()

async function authMiddleware(c:any, next:any){
    // c is the context of this request, response
    if(c.req.header("authorizatiion")){
        // Do validation
        await next()
    }else{
        return c.text("You don't have acces")
    }
}

// even in fetch we had to await , why while conerting something to json
// why do we have to await it??
app.post('/',authMiddleware, async (c) => {
    const body = await c.req.json()
    console.log(body);
    console.log(c.req.header("Authorization"));
    // any header we might want to access to
    console.log(c.req.query("param"));

    return c.text('Hello Hono!')
})

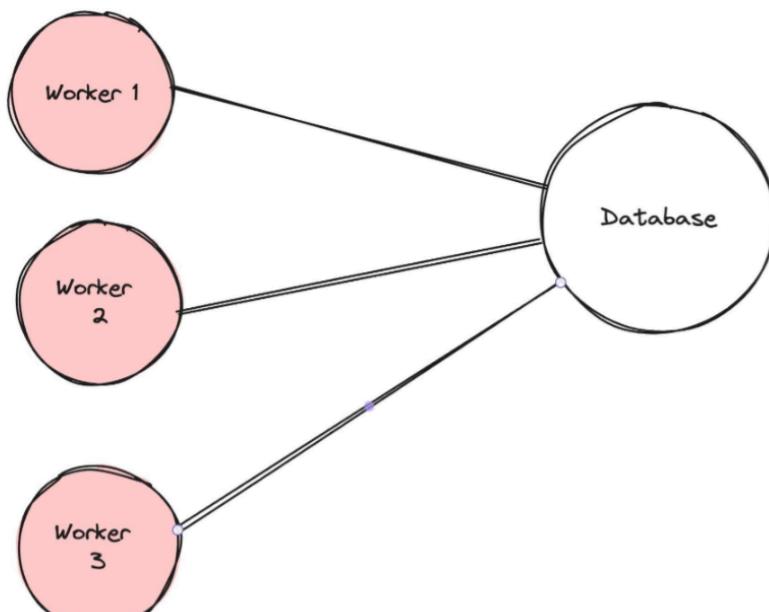
export default app
```

Connecting to DB

<https://www.prisma.io/docs/orm/prisma-client/deployment/edge/deploy-to-cloudflare-workers>

Serverless environment have one big problem when dealing with databases.

1. There can be many connections open to the DB since there can be multiple workers open in various region



Prisma the library has dependencies that the cloudflare runtime doesn't understand.

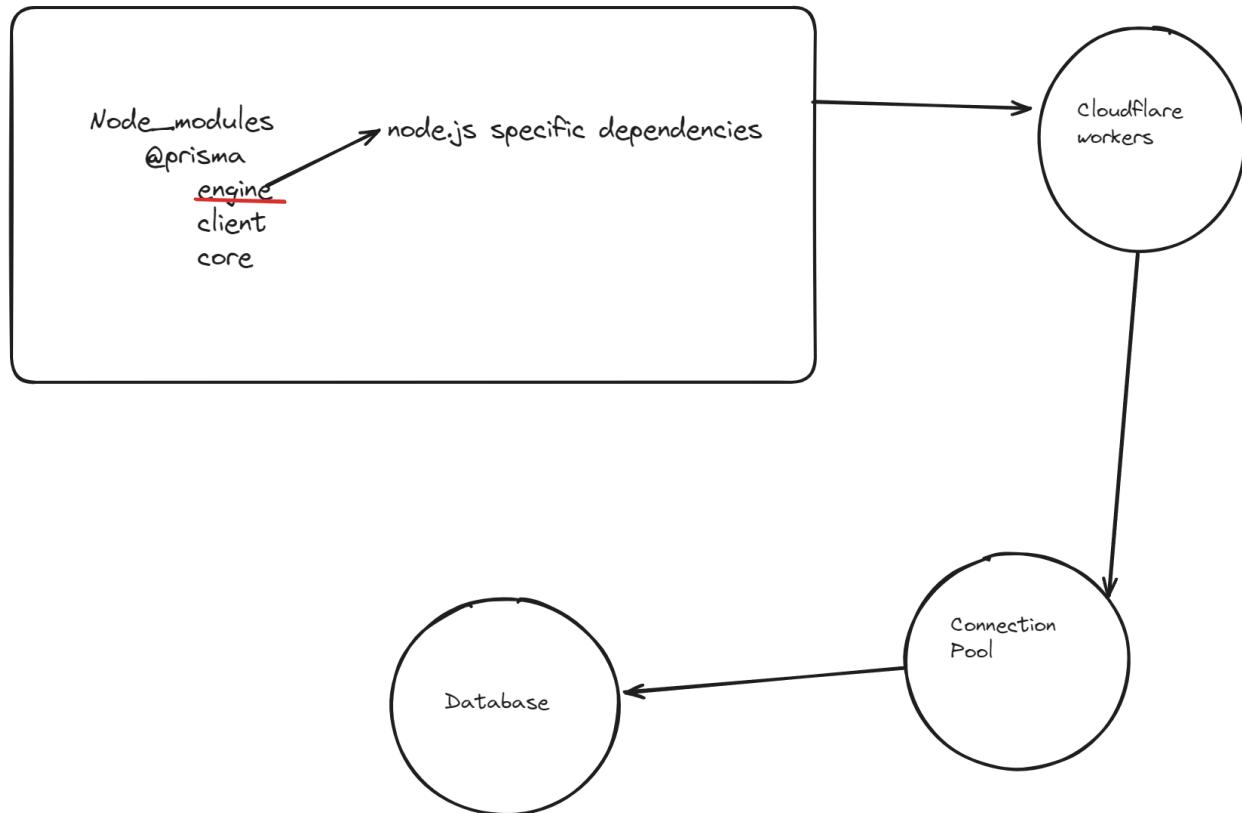
There could be hundred workers running and all connecting to the DB

- **Connection pooling in prisma for serverless env**

<https://www.prisma.io/docs/accelerate>

<https://www.prisma.io/docs/orm/prisma-client/deployment/edge/deploy-to-cloudflare-workers>

Now workers are first connected to the Connection Pool the only the allowed no. of connections are connected to the DB



npm create cloudflare@latest

npm install -save-dev prisma

npx prisma init

```

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

```

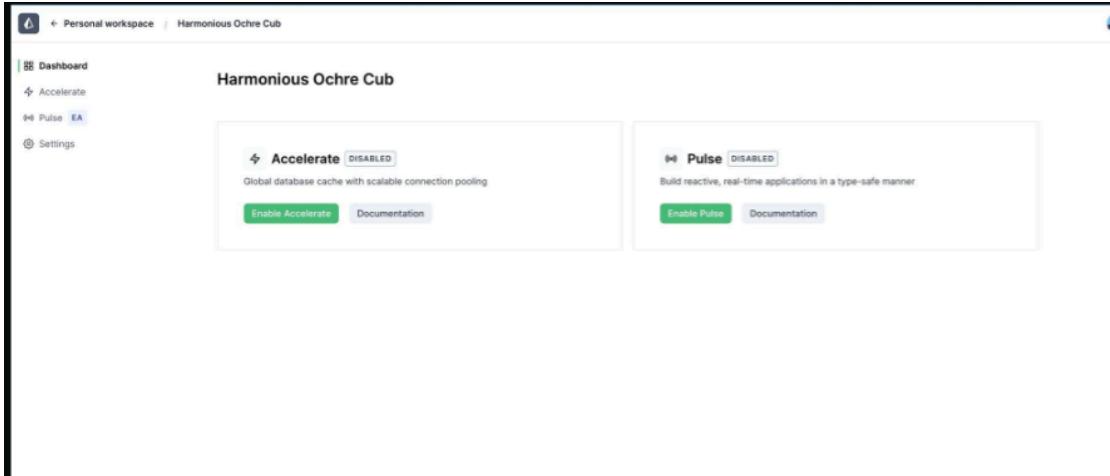
Here url are very important because we are not directly going to the database we are first going to the Connection pool then we will go to database.

npx prisma migrate dev --name init

Signup to prima accelerate

<https://console.prisma.io/login>

Enable Accelerate



And then Generate an API key

DATABASE_URL="prisma://["](https://accelerate.prisma-data.net/?api_key=eyJhbGciOiJIUzI1NlslnR5cCl6IkpxVCJ9.eyJhcGlfa2V5ljo1ODk1Njc2MDMtMml5Ni00N2VmLWFkZWMTZThkZTFiMjlxY2VhliwidGVuYW50X2kljoiNDFiMWFMzMNmYzk5OTdhODc5OGYwZmRmYzI1YTJkNmZjN2UxMWE1NmJZTAzZDljYjNhMWFjOGRjOWNjY2I5NlslmludGVybmFsX3NIY3JldCI6IjNjYWNmYmEzLTJhNDItNDM5NS04OTM2LTJmMWU1YTNhZGM4MCJ9.nKVE0Yxt1vY7t62Du79aHKQxrYi0u05SEfqyDK5Ska8)

This is the final url

In Cloudflare way to store env variable which we want to use in index.ts are in toml file

[vars]

DATABASE_URL

A screenshot of a code editor window titled "Week12". The left sidebar shows files: "schema.prisma" and "wrangler.toml". The main editor area shows the "wrangler.toml" file with the following content:

```
week12_6 > app > wrangler.toml
  main - src/index.ts
  3 compatibility_date = "2024-03-29"
  4 compatibility_flags = ["nodejs_compat"]
  5
  6 DATABASE_URL="prisma://accelerate.prisma-data.net/?api_key=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcGlfa2V5IjoiODk1Njc2MDMtMmI5Ni00N2VmLWFkZWMTZThkZTFiMjIxY2VhIiwidGVuYW50X2lkIjoiNDFiMWFMzMnMzYzk5OTdhODc5OGYwZmRmYzI1YTJkNmZjN2UxMWE1NmJlZTAzZD1jYjNhMWFjOGRjOWNjY2I5NiIsImludGVybmFsX3N1Y3JldCI6IjNjYwNmYmEzLTJhNDItNDM5NS04OTM2LTJmMWU1YTNhZGM4MCJ9.nKVE0Yxt1vY7t62Du79aHKQxrYi0u05SEfqryDK5Ska8"
  7 # Variable bindings. These are arbitrary, plaintext strings (similar to environment variables)
  8 # Note: Use secrets to store sensitive data.
  9 # Docs: https://developers.cloudflare.com/workers/wrangler/configuration/#environment-variables
 10 # [vars]
```

The bottom navigation bar includes tabs for "OUTPUT", "DEBUG CONSOLE", "TERMINAL", and "...".

A screenshot of a code editor window showing a ".env" file. The content is identical to the "wrangler.toml" file above, indicating environment variable overrides.

```
week12_6 > .env
 1 DIRECT_URL="postgresql://neondb_owner:tWcwiYRK8A3U@ep-fancy-pine-a5vggaa1.us-east-2.aws.neon.tech/neondb?sslmode=require"
 2 DATABASE_URL="prisma://accelerate.prisma-data.net/?api_key=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcGlfa2V5IjoiODk1Njc2MDMtMmI5Ni00N2VmLWFkZWMTZThkZTFiMjIxY2VhIiwidGVuYW50X2lkIjoiNDFiMWFMzMnMzYzk5OTdhODc5OGYwZmRmYzI1YTJkNmZjN2UxMWE1NmJlZTAzZD1jYjNhMWFjOGRjOWNjY2I5NiIsImludGVybmFsX3N1Y3JldCI6IjNjYwNmYmEzLTJhNDItNDM5NS04OTM2LTJmMWU1YTNhZGM4MCJ9.nKVE0Yxt1vY7t62Du79aHKQxrYi0u05SEfqryDK5Ska8"
```

npm install @prisma/extension-accelerate

npx prisma generate --no-engine

Smaller bundle size, connection pool consist some part of

```
import { Hono, Next } from 'hono'
```

```
import { PrismaClient } from '@prisma/client/edge'
import { withAccelerate } from '@prisma/extension-accelerate'
import { env } from 'hono/adapter'

const app = new Hono()

app.post('/', async (c) => {
    // Todo add zod validation here
    const body: {
        name: string;
        email: string;
        password: string
    } = await c.req.json()
    const { DATABASE_URL } = env<{ DATABASE_URL: string }>(c)

    const prisma = new PrismaClient({
        datasourceUrl: DATABASE_URL,
    })$.extends(withAccelerate())

    console.log(body)

    await prisma.user.create({
        data: {
            name: body.name,
            email: body.email,
            password: body.password
        }
    })

    return c.json({msg: "as"})
})

export default app
```

npx wrangler whoami