

# WHAT ARE BACKEND SERVERS?

They are like simple servers used in startups, hackathons etc initially its good but for scale its expensive so not used for scale

Vercel, Render are all serverless and good to use

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 -

1. Go to aws, GCP, Azure, Cloudflare

1. Rent a VM (Virtual Machine) and deploy your app

2. Put it in an Auto scaling group

3. Deploy it in a Kubernetes cluster

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

## WHAT IS SERVERLESS BACKENDS?

The better method to deploy things is to using asg, vm, kubernetes cluster and the other way is serverless

In this the cost is done based on the user request like you use this only when the number of users are less, and as a dev you don't have to worry about the servers and all they manager on its own

very hard to migrate from serverless to vms

its very difficult to move from serverless to actual db as it require time and money

"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.

## Easier definition

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

Deploy

Autoscale

Charge you on a per request basis (rather than you paying for VMs)

## Problems with this approach

More expensive at scale

Cold start problem == when you are deploying on serverless if imagine no one is hitting your be your backend will stop working and whenever imagine one user comes to your website then a container will start and then only your code will start to run which will take some time for the first user then for the other users it will be fast only

If no users than only the cold start problem

HOW TO SOLVE THIS?

Use a warm pool like using minimum 3-4 containers running but you have to pay money

Using a bot running somewhere , which hits the website every 5 seconds like that

## Famous serverless providers

There are many famous backend serverless providers -

AWS LAMBDA

Google Cloud Functions

Cloudflare Workers

### You write code. We handle the rest.

Deploy serverless code instantly across the globe to give it exceptional performance, reliability, and scale.

Start building

Read docs

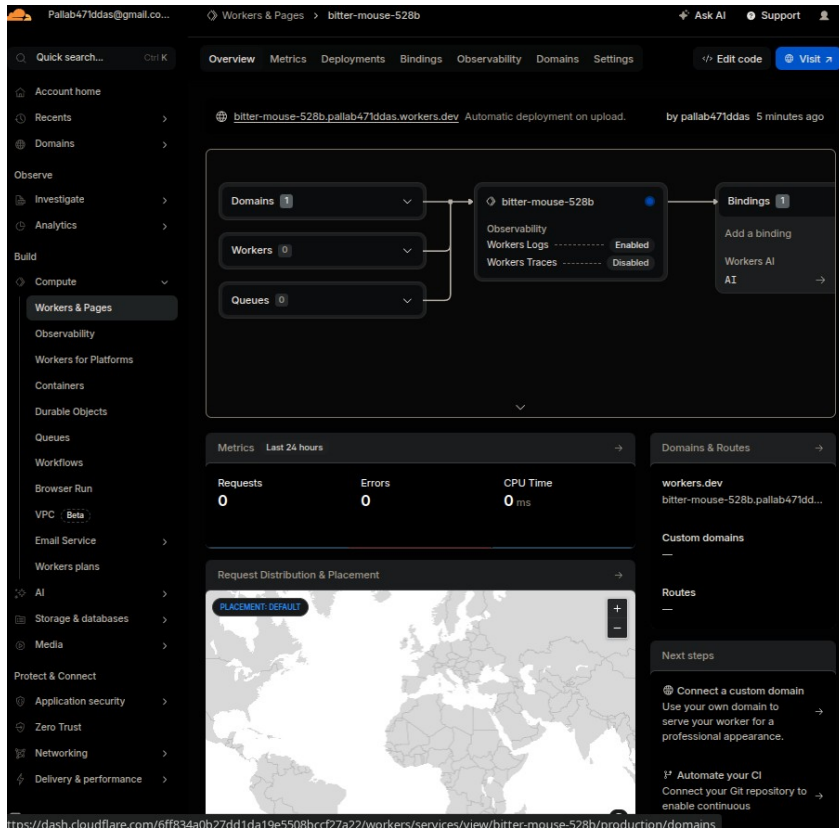
```
~/ $ npm create cloudflare -- my-app
~/ $ cd my-app
~/my-app $ npx wrangler deploy
Published https://my-app.world.workers.dev
```

- From signup to globally deployed in <5min
- Your code runs within milliseconds of your users worldwide
- Say goodbye to cold starts—support for 0ms worldwide

# When should you use a serverless architecture?

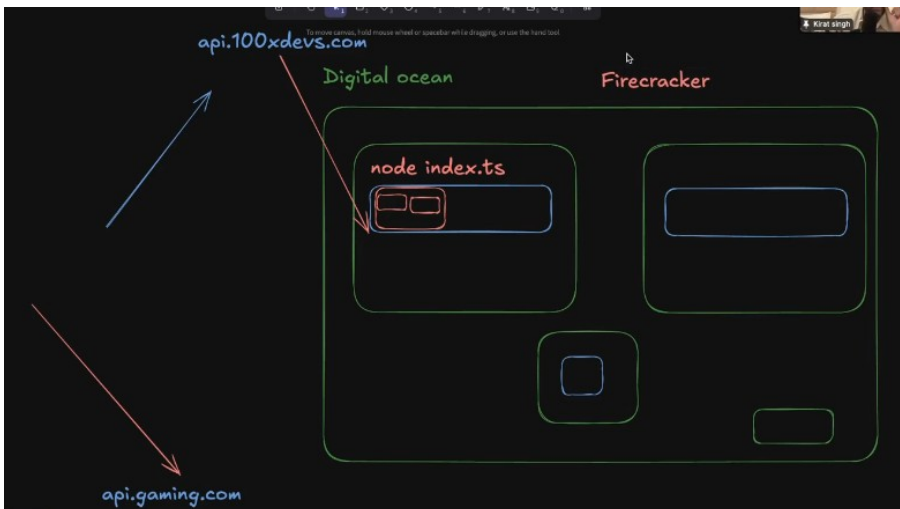
1. When you have to get off the ground fast and don't want to worry about deployments
2. When you can't anticipate the traffic and don't want to worry about autoscaling
3. If you have very low traffic and want to optimise for costs

Now creating a cloudflare worker and using that Dont buy url from the cloudflare they does like shady work if you like start making money from there url they send some random bills to pay



in this we can add env variable also in the settings part

# How cloudflare workers work?



Imagine you have digitalocean and you have to make a serverless server how you will do that?

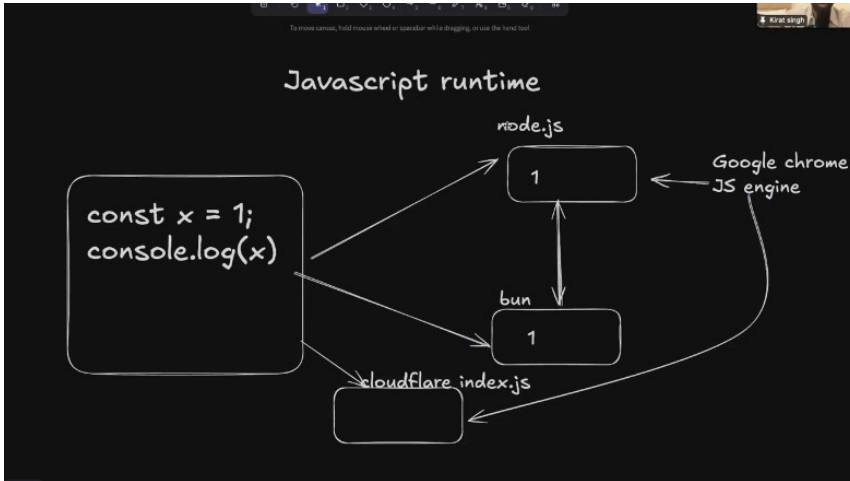
It means like for a website you have to create machine like which can hit 1 user also and 100 users also , if you give like 1 machine to everyone its a

waste of resources to fix that does this sounds like docker usage

only?? yes but the issue is docker also need some time to run and some computational power which is not feasible so this option is not that optimal so , so what is the approach?

AWS uses firecrackers to spin up these small machines but they use something very much related to node js which is taking a lot of time for them to involve new languages

You know what



whenever we write a js code it does not give output on its own it is interpreted by things like node js , bun js etc which like gives the output but this cloudflare, aws created there own interpreter based on the v8 engine(Google chrome engine) which helps them to

run the code in there machines and it also says it might not run all the code of the node js , its not fully compatiabile with the node js and also in the past bun was not also compatiabile but not its like 99%

## How Workers works

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.

there are many things which exist in browser which does not exist in node js but do exist in workers

so like our approach was we will create docker container inside a very big fat machine and then we will run all the codes but that is expensive but what we do in cloudflare is

we run a big cloudflare runtime which can run small small user code with the context in isolates

## 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](#).

think isolate like a docker container but much more light weight

its like they are running in one machine but the code cannot talk to each other so that the projects cannot call other projects api and all

## NOW CREATING A WORKER FROM TERMINAL

npm create cloudflare-- todo-app-week-32

then select the project  
then select the framework

then what tech

Workers, Assets(both fe and be), durable objects (storage very close to db but not actually db),  
we selected worker, ts and then the project starts

```
TS index.ts x
1  const main :ExportedHandler<Env>= {
2    fetch(request, env, ctx): Response {
3      return new Response("Hello World! | " + Math.random());
4    },
5  }
6  export default main;
7
```

in this cloudflare will handle everything like all the routes will be handled by this only and the look of the code is little bit different

```
const main: ExportedHandler<Env> = {
  // what cloudflare worker runtime sends the request to
  fetch(request, env, ctx): Response {
    if (request.method === "POST") {
      if (request.url.endsWith("/user")) {
        return new Response('this is a post request to /user');
      } else {
        return new Response('this is a post request to not /user');
      }
    } else {
      return new Response('this is a get request');
    }
  },
}
```

routing wagera hai nhi toh aisa ugly code likhna padtha hai yeh hai issue isme

**to deploy on workers we have to use wrangler of cloudflare which is like its cli  
do npx wrangler login**

**it will get the creds of wrangler  
run npm run deploy (the code is deployed on the internet)  
express does not work with the workers it worked with the node and earlier it used to not work with bun also but later it started working**

**for re deploy also it will not take time and if you just do npm run deploy it will work (helps to push the code fastly)**

### **Initializing a worker**

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

Explore package.json dependencies  
"wrangler": "^3.0.0"

Notice express is not a dependency there  
Start the worker locally  
npm run dev

### **How to return json?**

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

Question - 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

Question - 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 default {
```

```
  async fetch(request: Request, env: Env, ctx: ExecutionContext): Promise<Response>  
  {  
    console.log(request.body);  
    console.log(request.headers);  
  
    if (request.method === "GET") {  
      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.

### **Deploying a worker**

Now that you 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/>)  
notion image

Step 1 - Login to cloudflare via the wrangler cli

Step 2 - Deploy your worker

# 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.

## 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

The screenshot shows the Cloudflare 'Register Domain' page. A search bar contains '10kdevs.com' and a 'Search' button. A message states '10kdevs.com is not available' because it is registered. Below this, a table of 'Suggested domain names' is shown with columns for Domain, Price, and a 'Purchase' button.

Domain	Price	
10kdevs.io	\$45.00	Purchase
10-kde-vs.com	\$9.77	Purchase
10kdevs.net	\$11.84	Purchase
10kdevs.uk	\$4.94	Purchase
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?

1 123testcoding Jun '22

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 1 1

created last reply 3 6.9k 2 6 1 2

1 Jun '22 Jun '22 replies views users likes

## Reason 1 - Express heavily relies on Node.js

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

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

cherryjimbo MVP '23 Jun '22

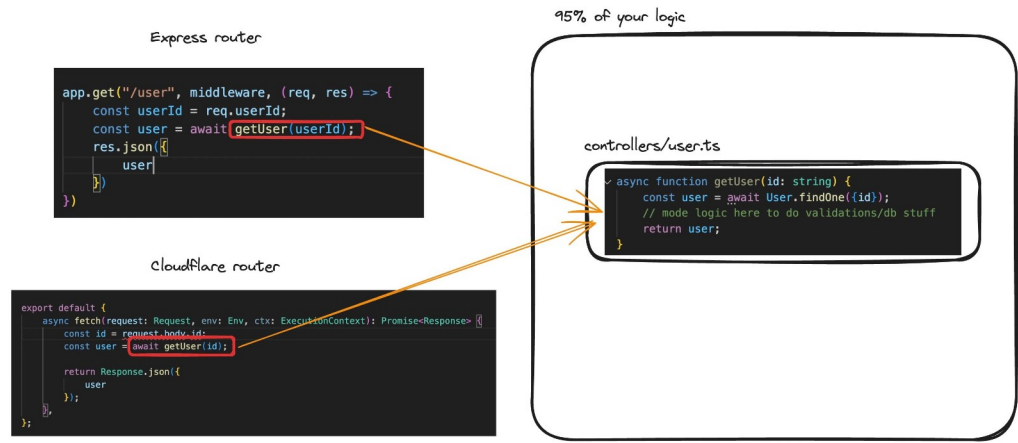
It's unlikely you'll see specifically express on Workers due to its deep Node.js dependencies, however there are a lot of options that you'll probably feel right at home with, and have very similar APIs to something like Express. To name just a few:

- 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 - kwhitley/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

## You can split all your handlers 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).

TO FIX THAT WE USE HONO FOR THAT  
SO HONO IS VERY HELPFULL AS ITS VERY SIMILAR TO  
EXPRESS AND IT DOES NOT HAVE APP.LISTEN(3000)

<https://hono.dev/docs/>

and in hono we get like single argument like in express we used to get req, res we get here only c  
it supports everything, fe, be , react type code, middlewares etc... everything

basically a fullstack application

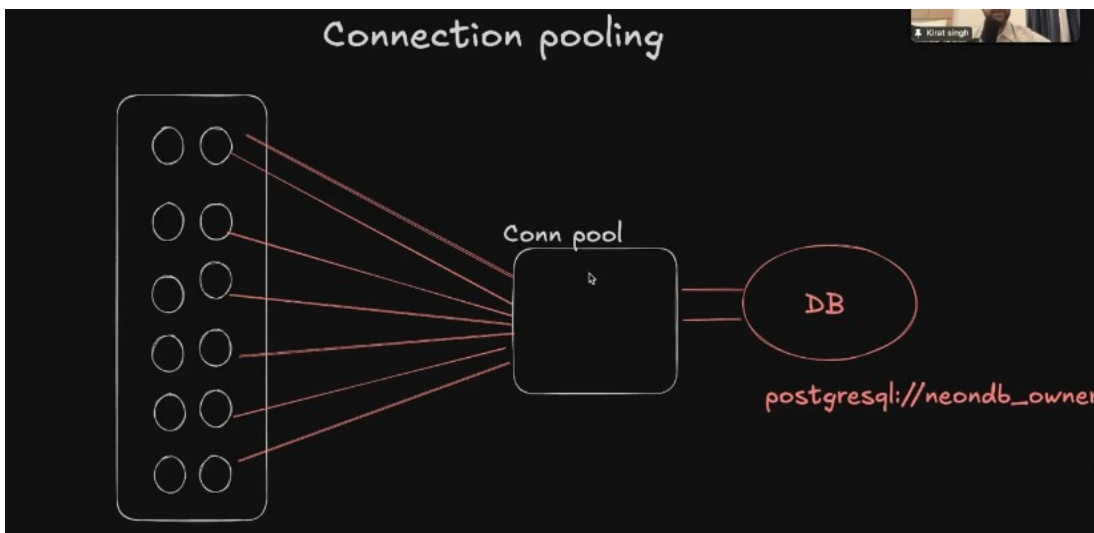
npm create [hono@latest](https://hono.dev/docs/) todo-app-latest

```
→ todo-app-week-32 cd ..
→ cohort3 npm create hono@latest todo-app-hono-week-32

> project@1.0.0 npx
> create-hono todo-app-hono-week-32

create-hono version 0.16.0
✓ Using target directory ... todo-app-hono-week-32
✓ Which template do you want to use? cloudflare-workers
✓ Do you want to install project dependencies? Yes
✓ Which package manager do you want to use? bun
✓ Cloning the template
✓ Installing project dependencies
👉 Copied project files
Get started with: cd todo-app-hono-week-32
→ cohort3 cd todo-app-week-32
```

WHAT IS CONNECTION POLLING?  
HOW DOES PRISMA MAKES MONEY THO?  
BCOZ OF CONNECTION POLLING



this option if you tick  
in connection pooling  
then its ok bcoz  
prisma used to use  
prisma acclerate  
which costs money

# Using hono

## What is Hono

<https://hono.dev/concepts/motivation>

### # 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 "RegExpRouter". 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?

Side photo

### Working with cloudflare workers -

1. Initialize a new app

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

1. Move to my-app and install the dependencies.

```
cd my-app
npm i
```

1. Hello World

```
import { Hono } from 'hono'
const app = new Hono()

app.get('/', (c) => c.text('Hello Cloudflare Workers!'))

export default app
```

### 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
```

### Getting Started

Basic

Cloudflare Workers

Cloudflare Pages

Deno

Bun

Fastly Compute

Vercel

Netlify

AWS Lambda

Lambda@Edge

Supabase Functions

Node.js

Others



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

Deploying

Make sure you're logged into cloudflare (wrangler login)

```
npm run deploy
```

## Middlewares



<https://hono.dev/guides/middleware>

### Creating a simple auth middleware

```
import { Hono, Next } from 'hono'
import { Context } from 'hono/jsx';

const app = new Hono()

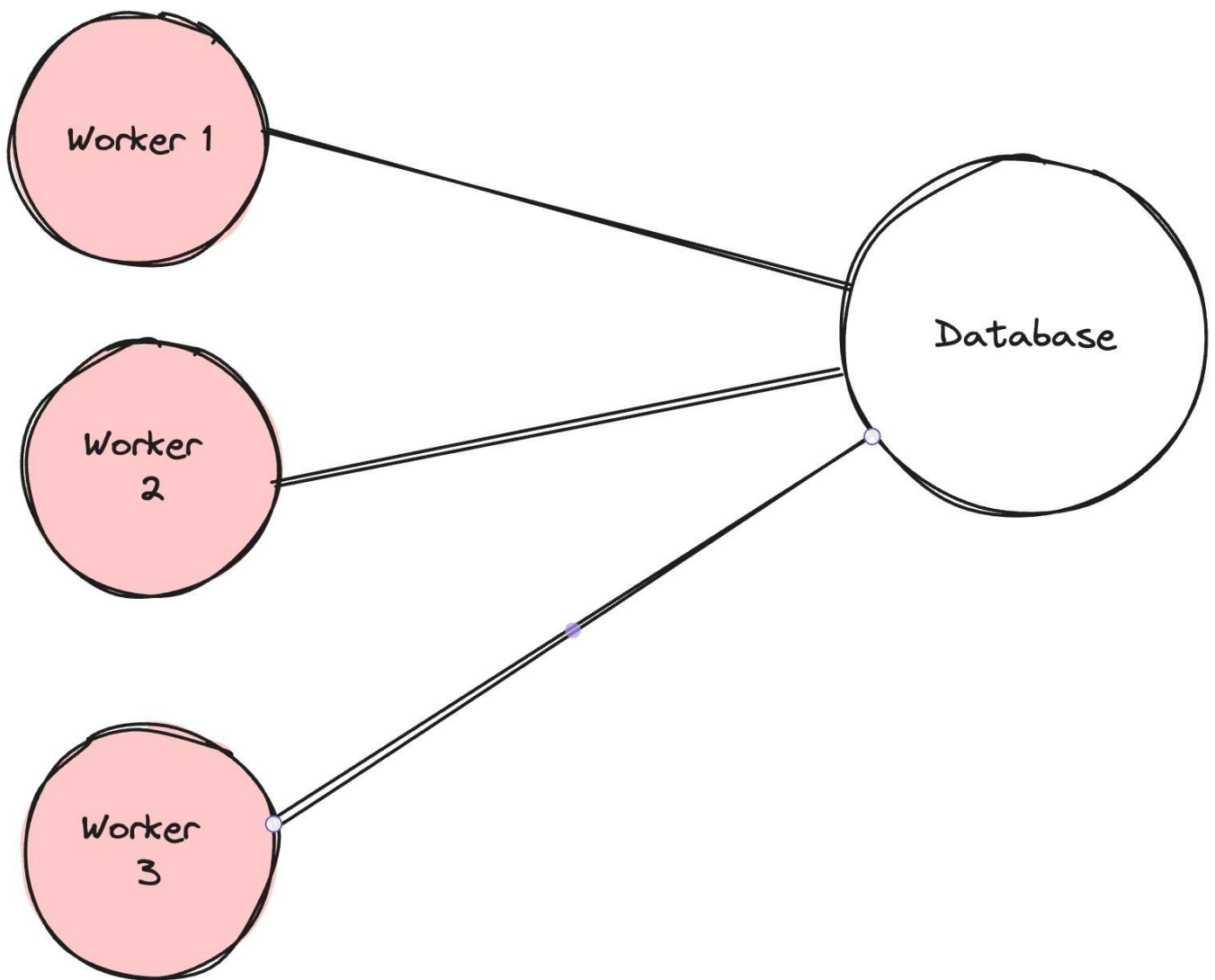
app.use(async (c, next) => {
  if (c.req.header("Authorization")) {
    // Do validation
    await next()
  } else {
    return c.text("You dont have acces");
  }
})

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

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

export default app
```

Notice you have to return the `c.text` value



## Connecting to DB

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

Serverless environments 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 regions
1. Prisma the library has dependencies that the cloudflare runtime doesn't understand.

## 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>

### 1. Install prisma in your project

```
npm install --save-dev prisma
```

### 2. Init Prisma

```
npx prisma init
```

### 3. Create a basic schema

```
generator client {  
  provider = "prisma-client-js"  
}  
  
datasource db {  
  provider = "postgresql"}
```

```

url      = env("DATABASE_URL")
}

model User {
  id      Int    @id @default(autoincrement())
  name    String
  email   String
  password String
}

```

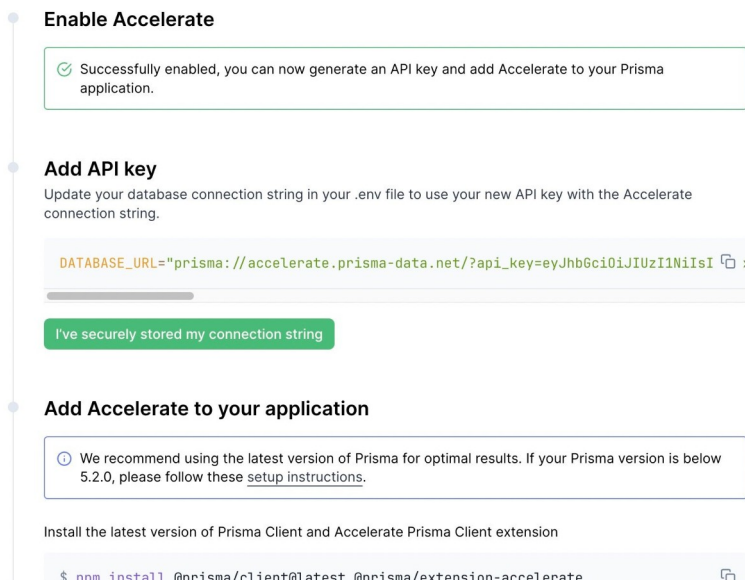
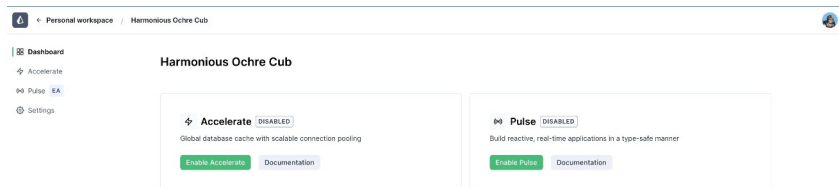
#### 4. Create migrations

```
npx prisma migrate dev --name init
```

#### 5. Signup to Prisma accelerate

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

#### Enable accelerate



Generate an API key

Replace it in .env

```

DATABASE_URL="prisma://
accelerate.prisma-data.net/?
api_key=your_key"

```

#### 5. Add accelerate as a dependency

```
npm install @prisma/extension-accelerate
```

#### 6. Generate the prisma client

```
npx prisma generate --no-engine
```

#### 7. Setup your code

```

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
```

1. No global variable in serverless
2. Connection pooling
3. Prisma accelerate