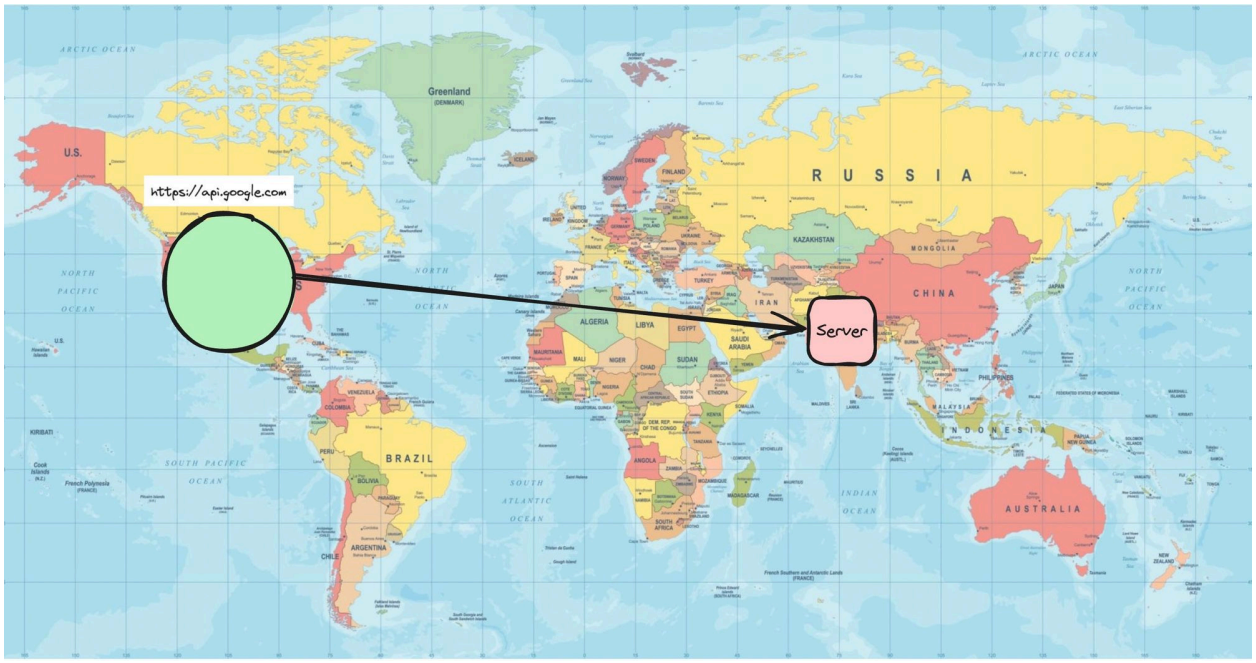
Serverless Fns

(https://projects.100xdevs.com/tracks/eooSv7InuwBO6wl9YA5w/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

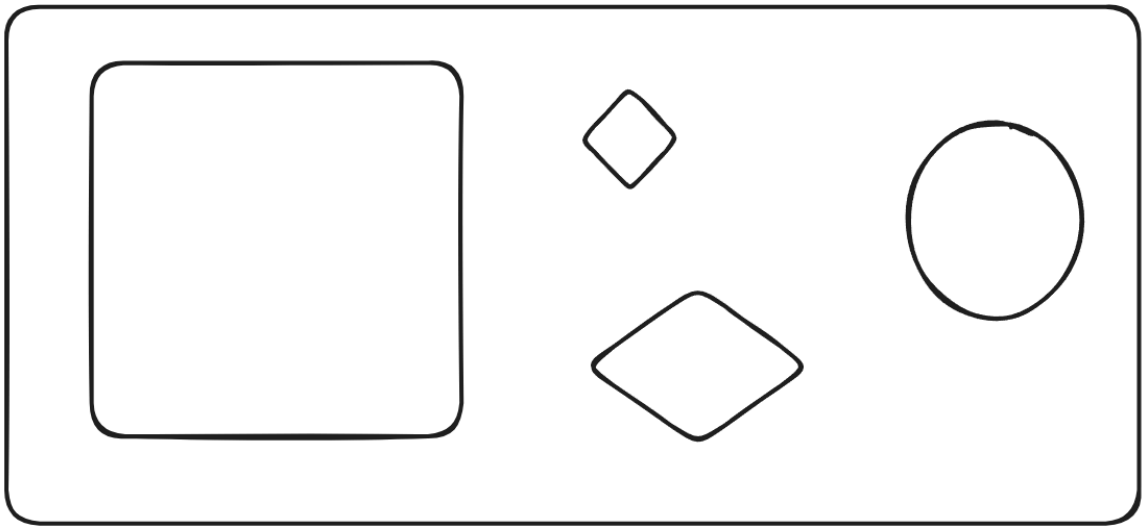
What if you could just write your **express routes** and run a command. The app would automatically

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

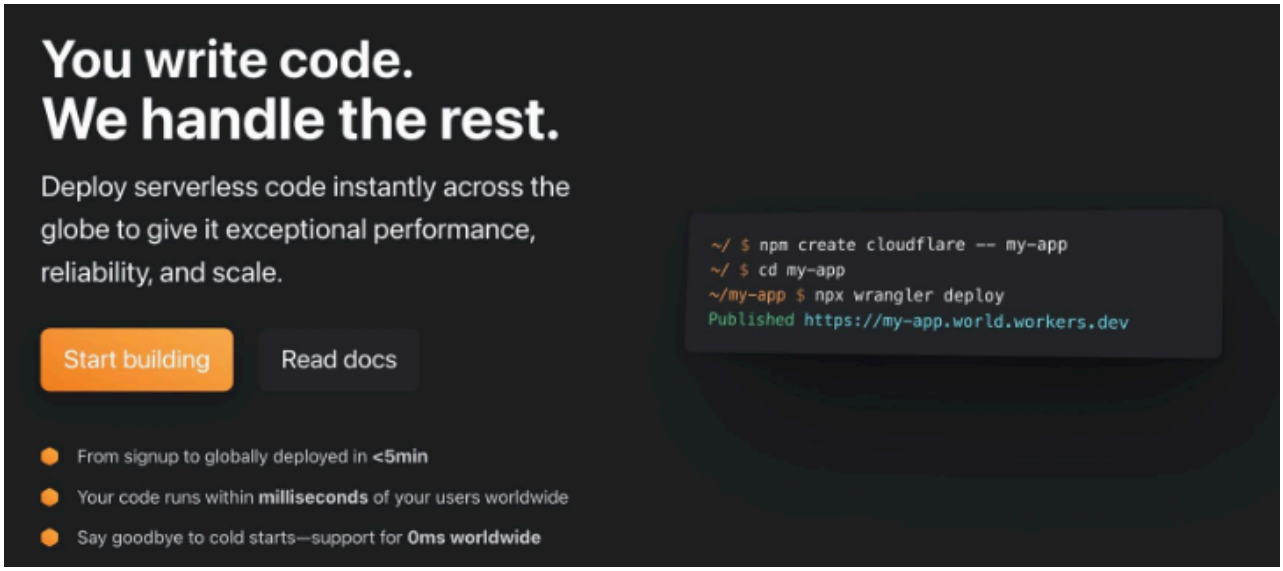
(https://aws.amazon.com/pm/lambda/?trk=5cc83e4b-8a6e-4976-92ff-7a6198f2fe76&sc_channel=ps&ef_id=CjwKCAiAt5euBhB9EiwAdkXWO-i-th4J3onX9ji-tPt_JmsBAQJLWYN4hzTF0Zxb084EkUBxSCK5vhoC-1wQAvD_BwE:G:s&s_kwcid=AL!4422!3!651612776783!e!!g!!aws%20lambda!19828229697!143940519541)

- ☐ Google cloud Functions

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

- ☐ Cloudfare Workers

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



We will use cloudfare (for now)

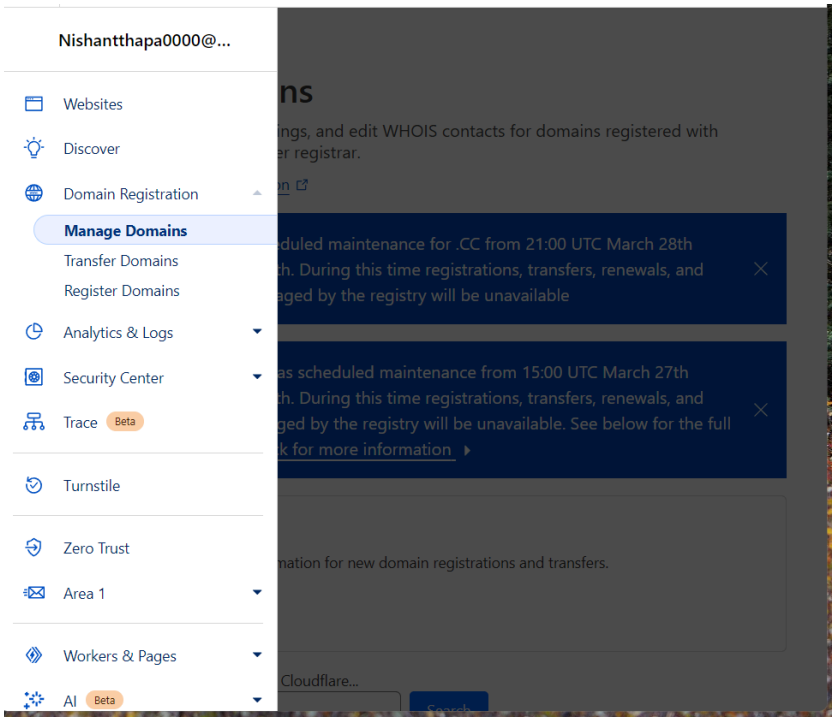
When should you use a serverless architecture

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

Cloudfare workers setup

Cloudfare prevent ddos attack .

Dashboard

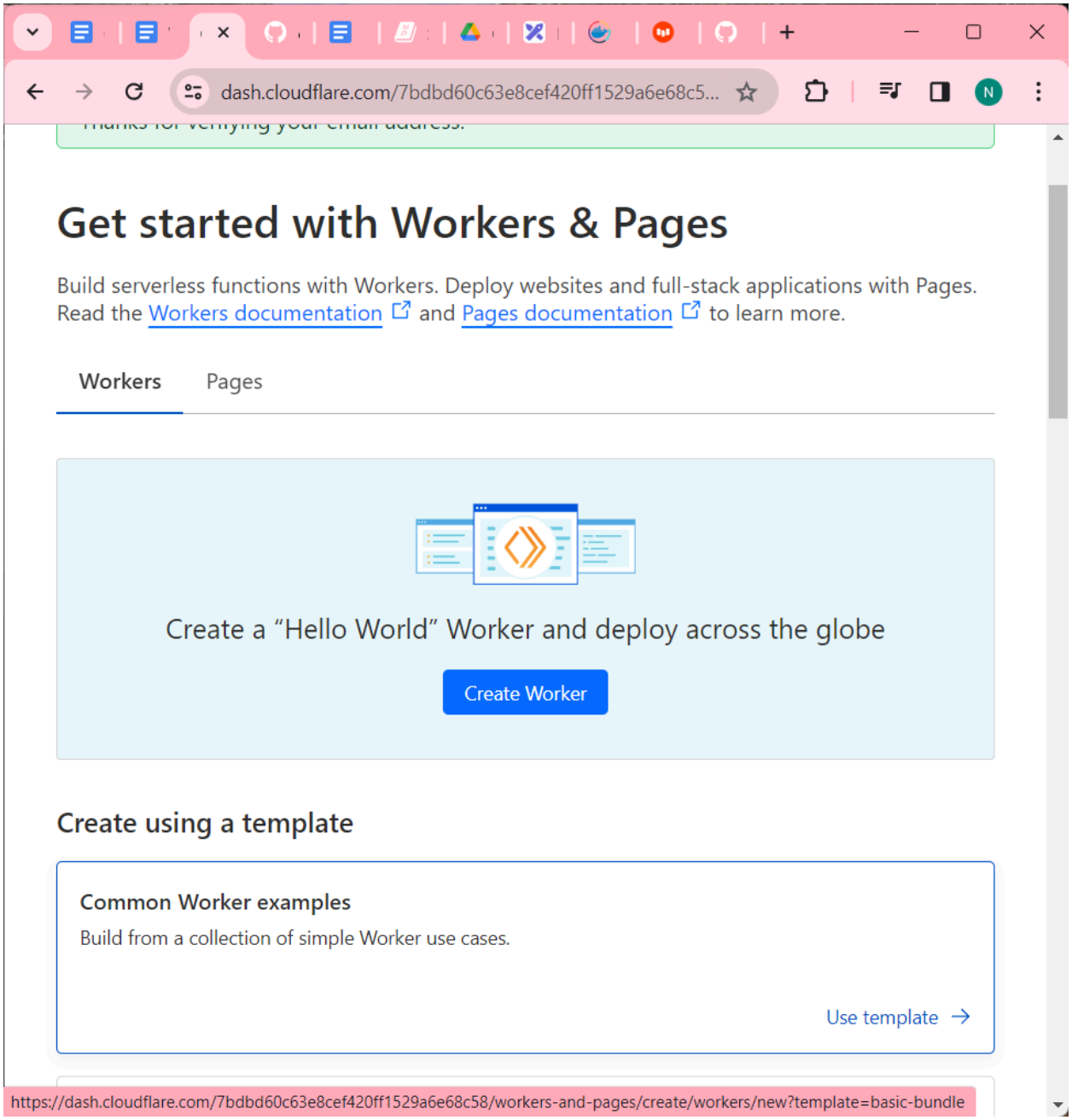


We will be go to Workers & Pages tab

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

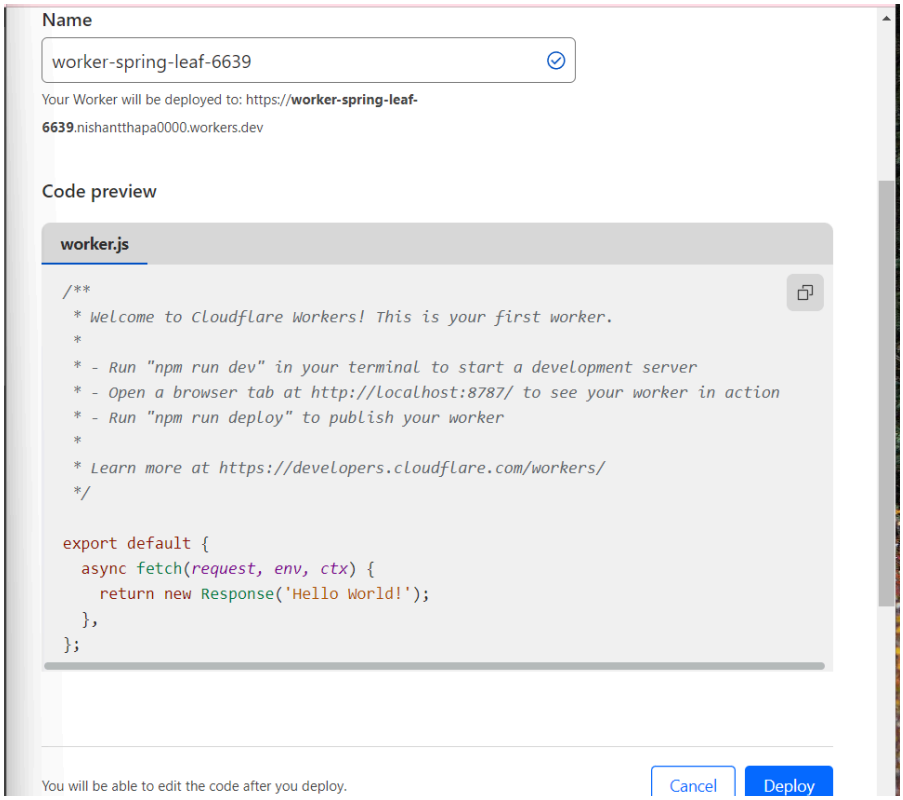
Pages documentation: https://developers.cloudflare.com/pages/

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

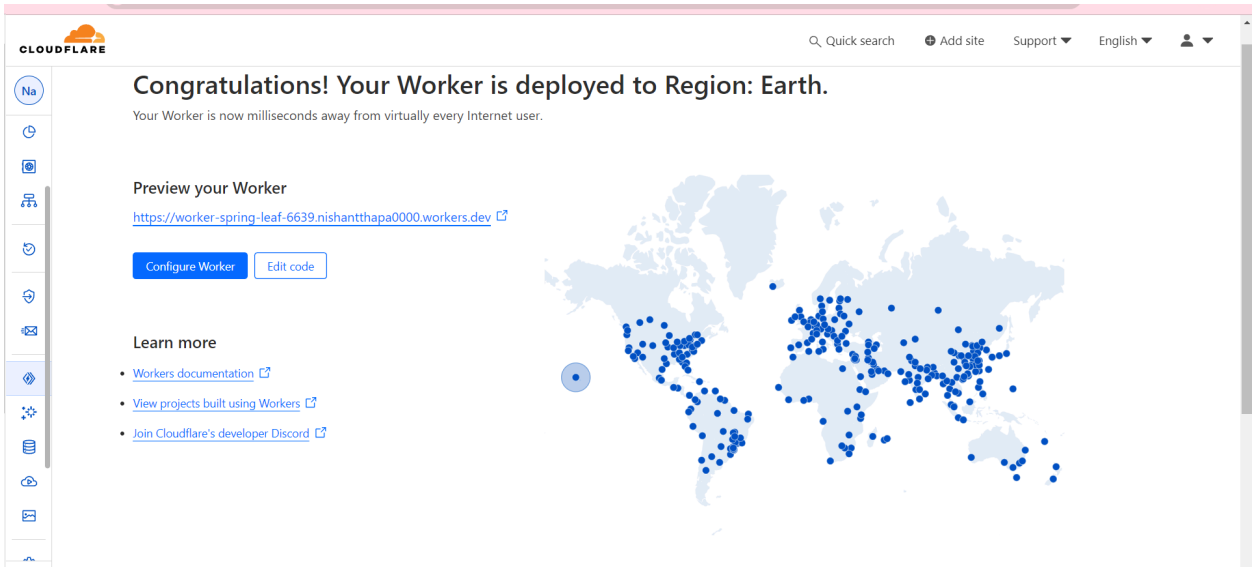


Try creating a test worker from the UI (Common worker example) and try hitting the URL at which it is deployed.

We wil deploy it

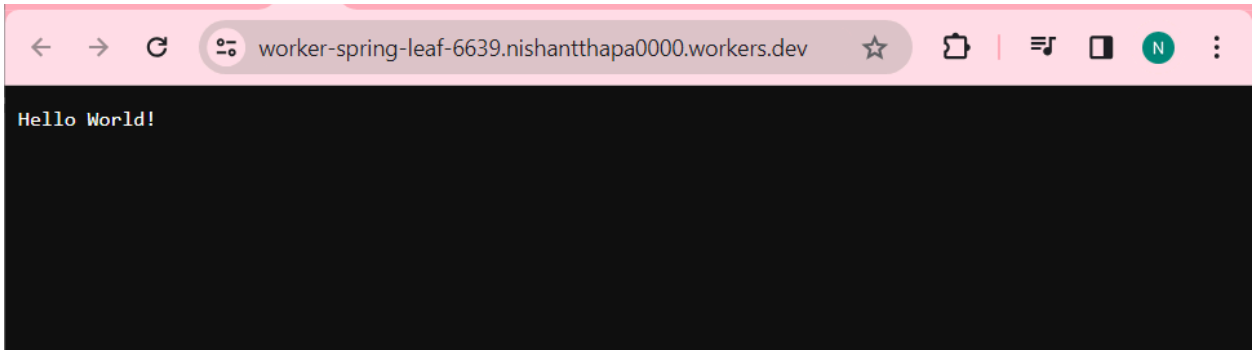


After we deploy



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

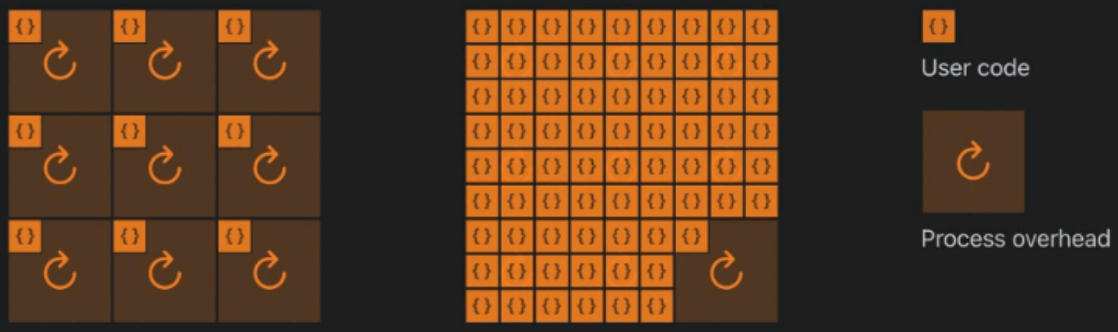
Worker runtime uses V8 engine.

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.



Traditional architecture

Workers V8 isolates

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 Cloudfare workers

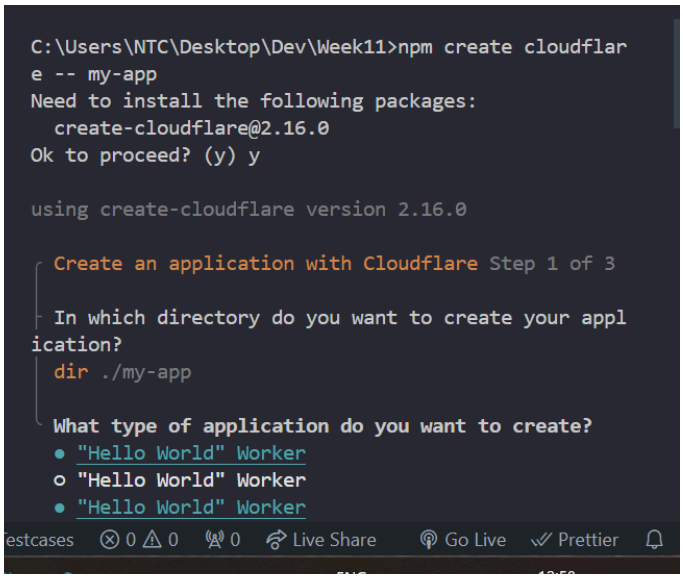
To create and deploy your application, you can take the following steps –

- 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



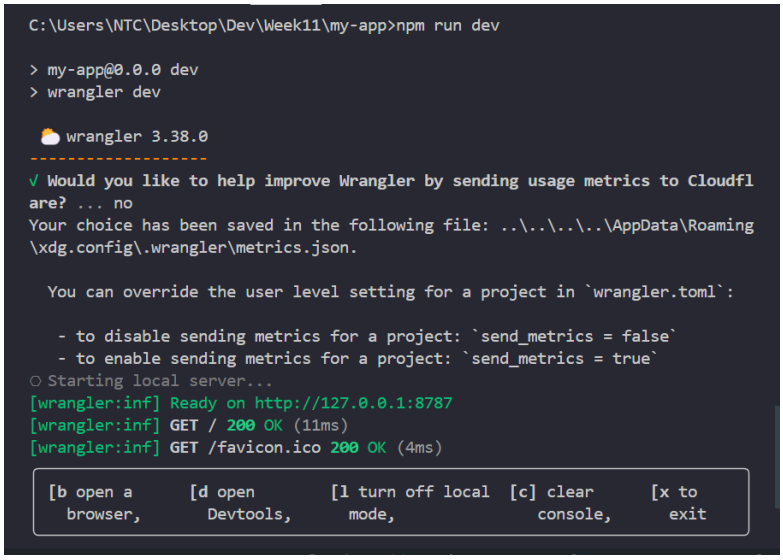
- Explore package.json dependencies\

"wrangler": "^3.0.0"

Notice express is not in dependencies

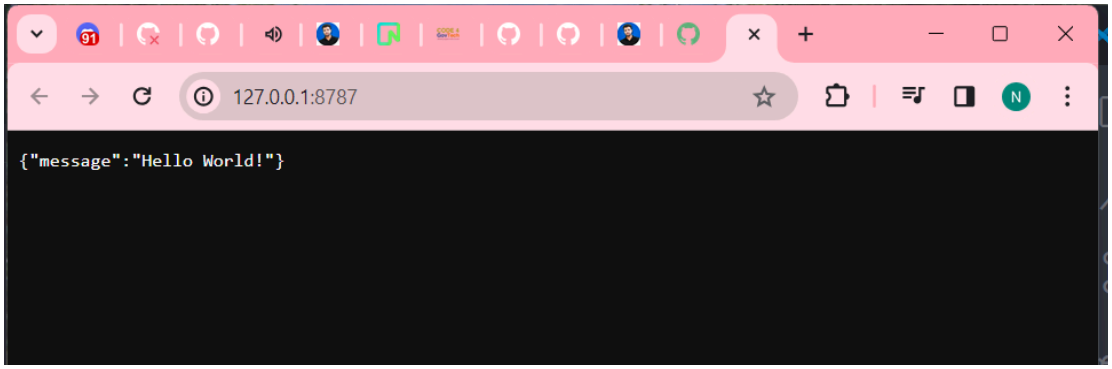
- Start the worker locally

npm run dev



- How to return json?

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



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 willl 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/au
th?response_type=code&client_id=54d11594-84e4-41aa-b438-e81b8fa78ee7&redirect
_uri=http%3A%2F%2Flocalhost%3A8976%2Foauth%2Fcallback&scope=account%3Aread%20
user%3Aread%20workers%3Awrite%20workers_kv%3Awrite%20workers_routes%3Awrite%2
0workers_scripts%3Awrite%20workers_tail%3Aread%20d1%3Awrite%20pages%3Awrite%2
0zone%3Aread%20ssl_certs%3Awrite%20constellation%3Awrite%20ai%3Aread%20offlin
e_access&state=1X-MgWW8N~Sb1iLhW6aP13W4nzu.Jnio&code_challenge=sfnmebECFVnLfu
s5YTPFcwAjYY8VhdLbU9uszrACS1g&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
Nishantthapa0000@gmail.com's Account	7bdbd60c63e8cef420ff1529a6e68c58

```
 Token Permissions: If scopes are missing, you may need to logout and re-lo
gin.
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>
```

To deploy it we do

npm run deploy

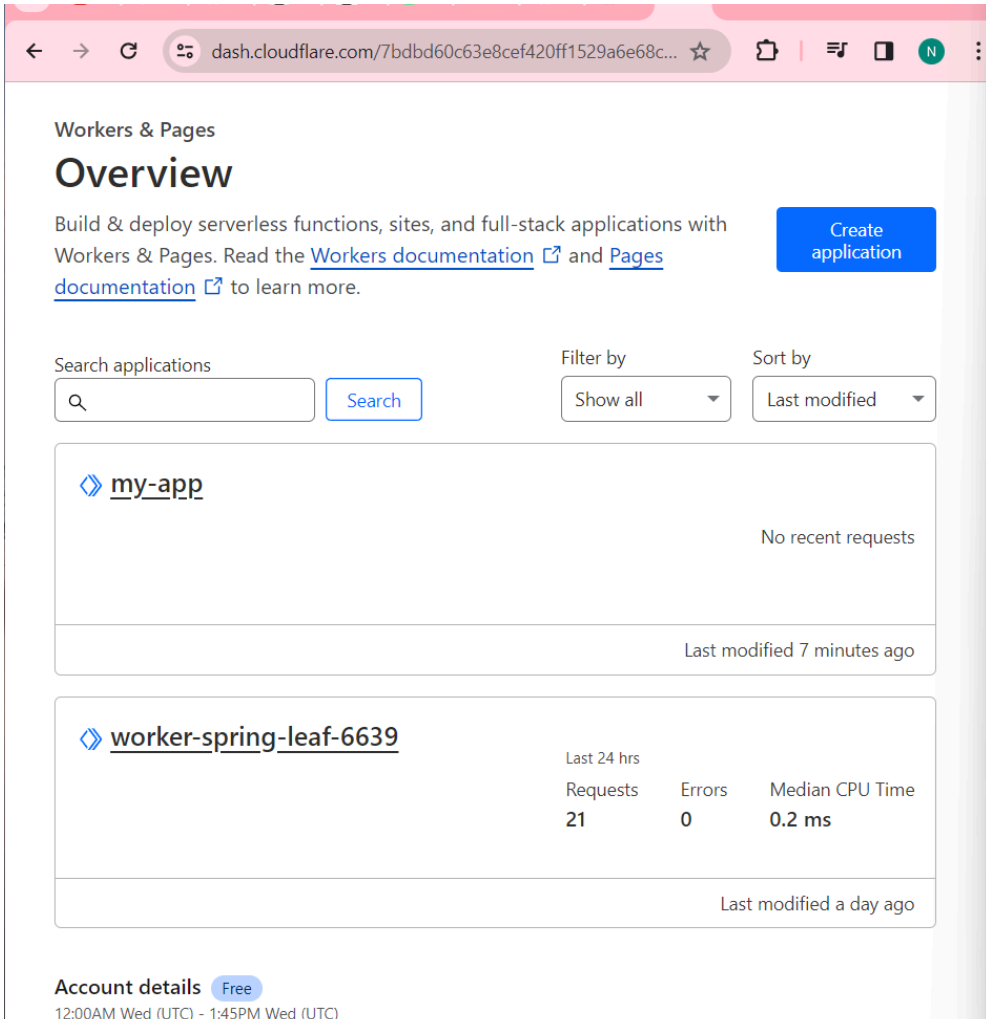
Give url where our backend is posted.

```
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;
}
```

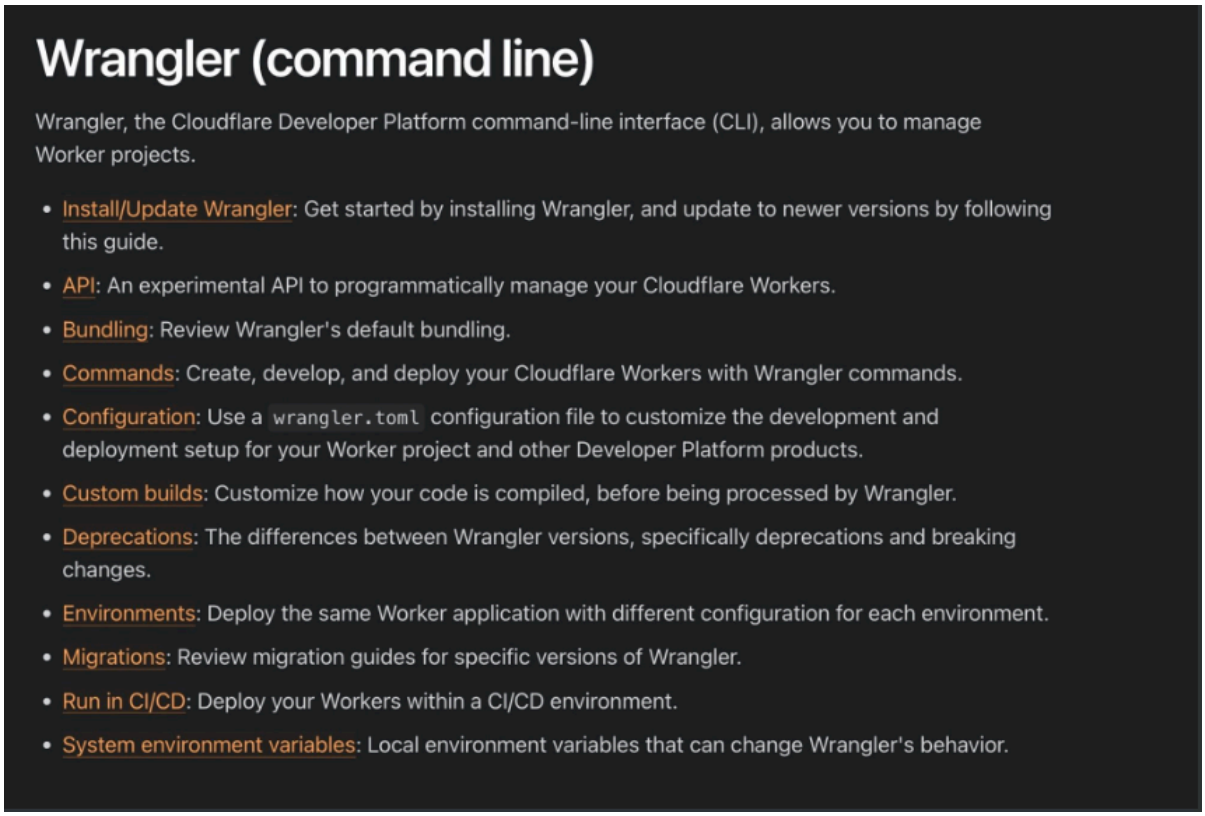
We can even see the workers on GUI of the Cloudfare



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

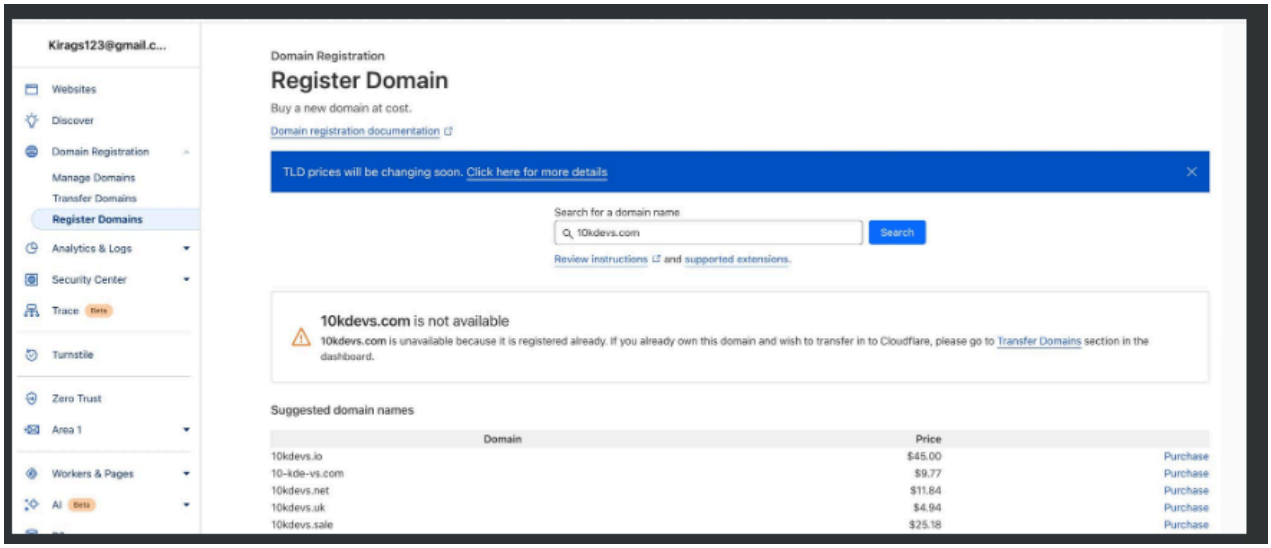
Step 2 : Deploy your worker

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

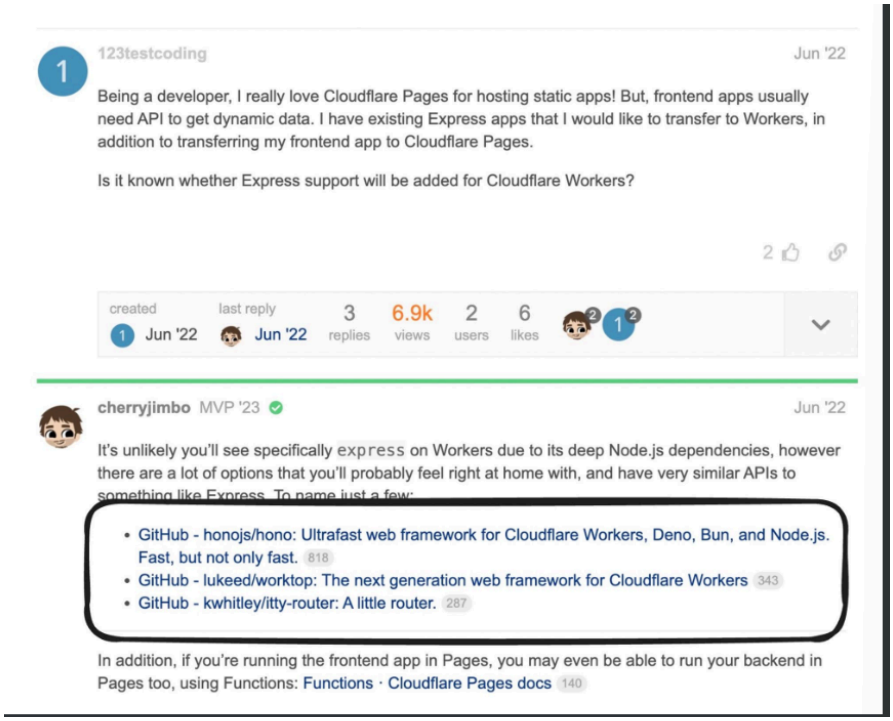


Adding express to it.

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

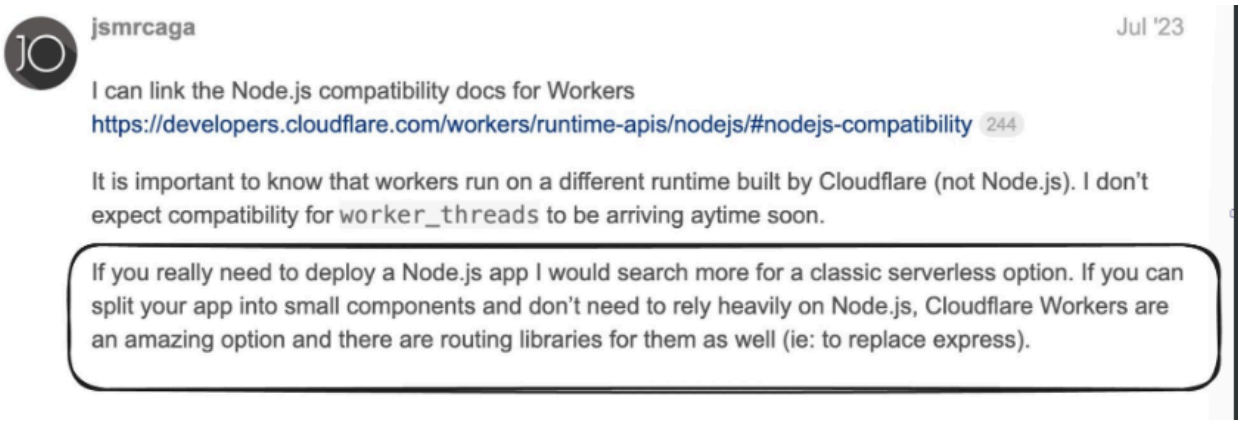
https://community.cloudflare.com/t/express-support-for-workers/390844



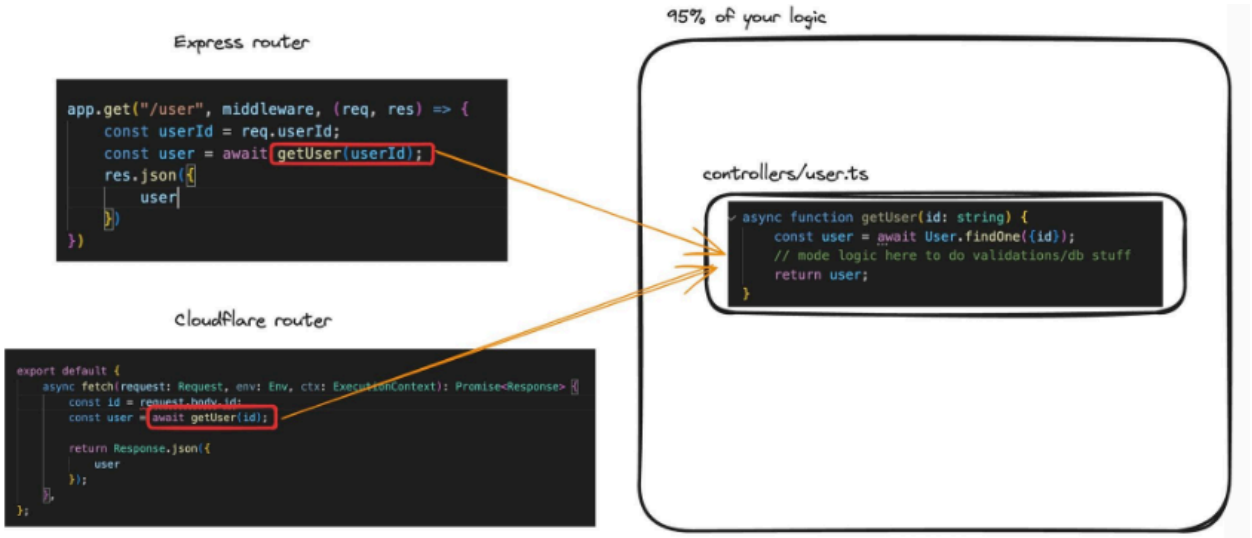
https://github.com/honojs/hono

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



Classic serverless option?

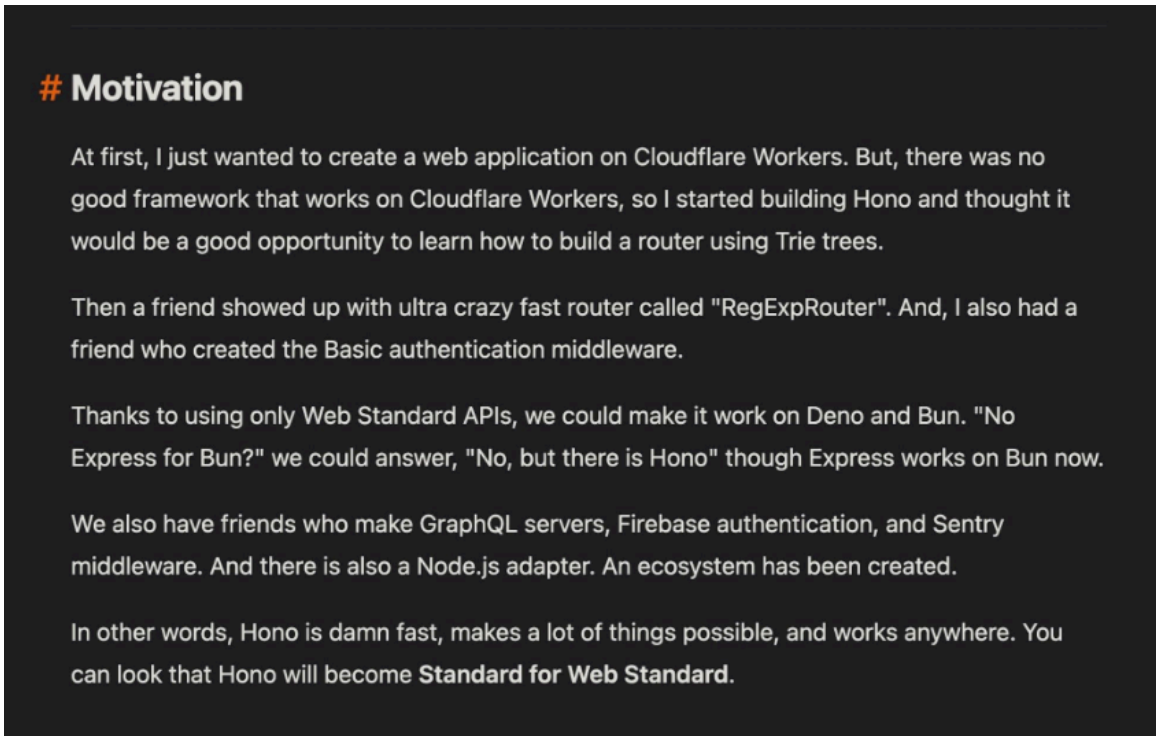


Encapsulate as many as we can in general functions

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.

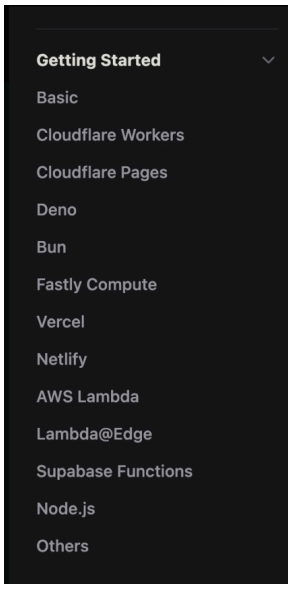
Then a friend showed up with ultra crazy fast router called "RegExpRouter". And, I also had a friend who created the Basic authentication middleware.

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??



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

Cloudfare examples

Authorization

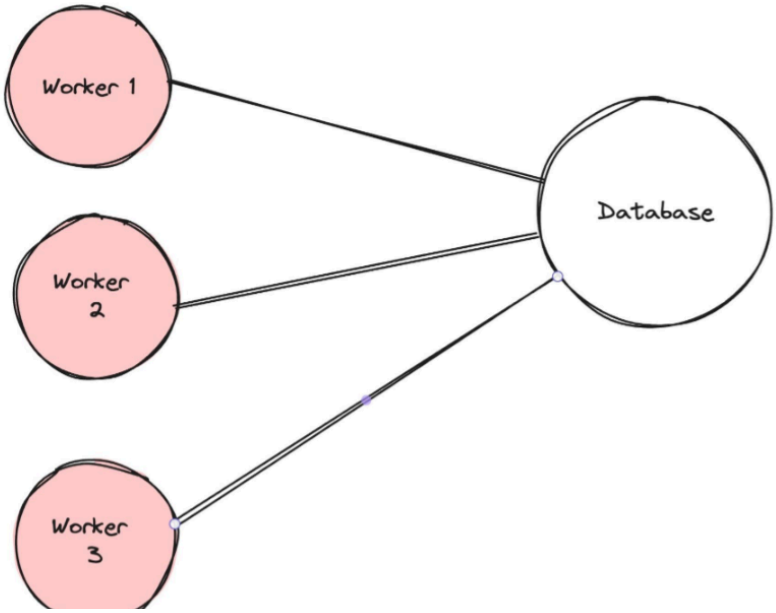
```
import { Hono } from 'hono'

const app = new Hono()

async function authMiddleware(c:any, next:any){
    // c is the contextr 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

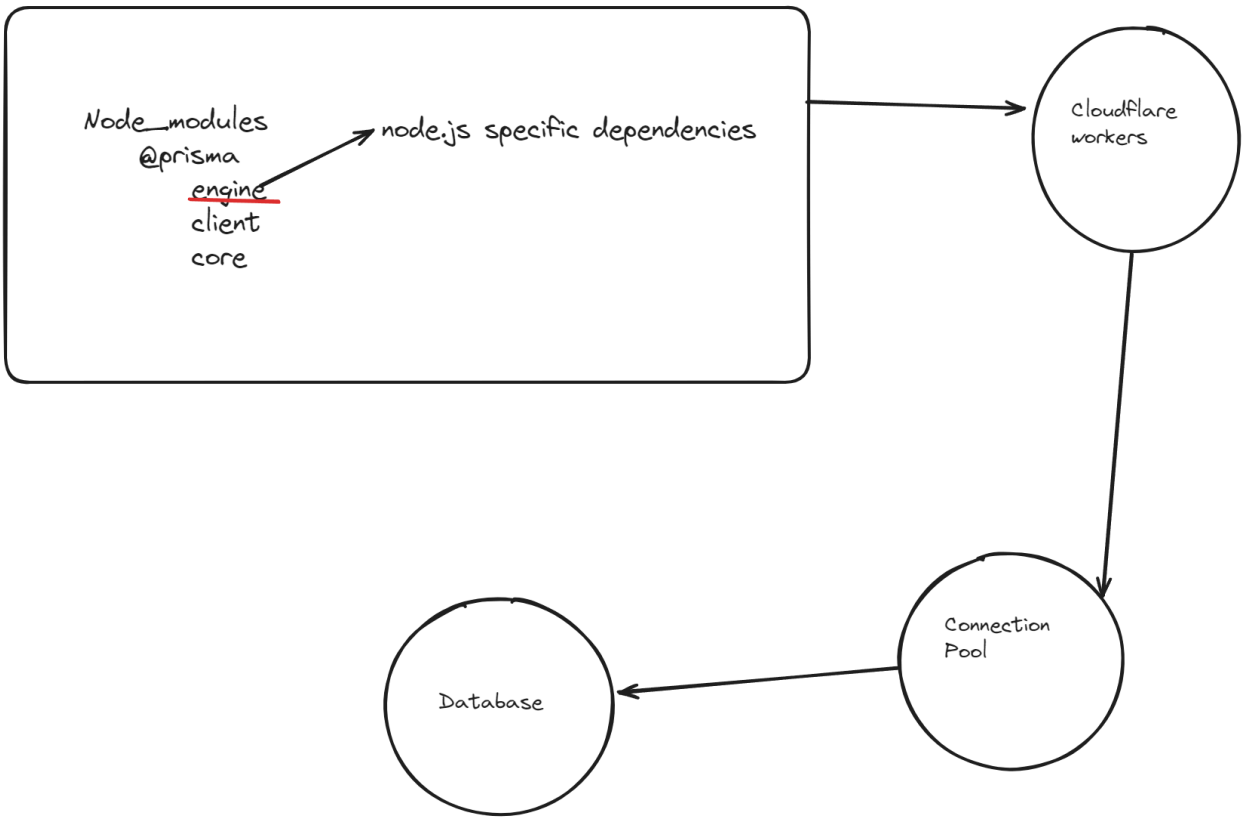


Prisma the library has dependencies that the cloudflare runtime doesn't understand.
There could be hundred workers running and all connecting to the DB

- **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")
}
```

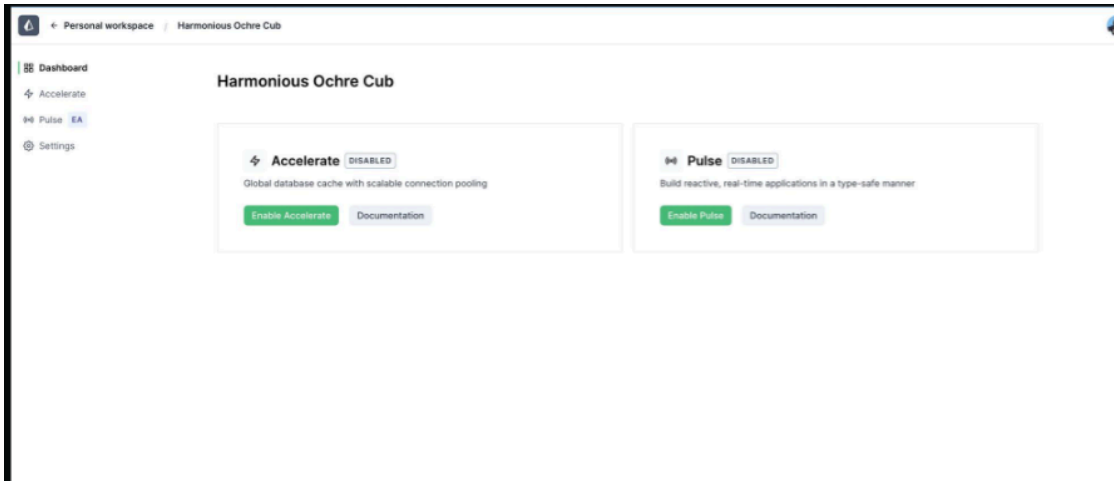
Here url are very important because we are not directly going to the database we are first going to the Connection pool then we will go to database.

npx prisma migrate dev --name init

Signup to prima accelerate

https://console.prisma.io/login

Enable Accelerate



And then Generate an API key

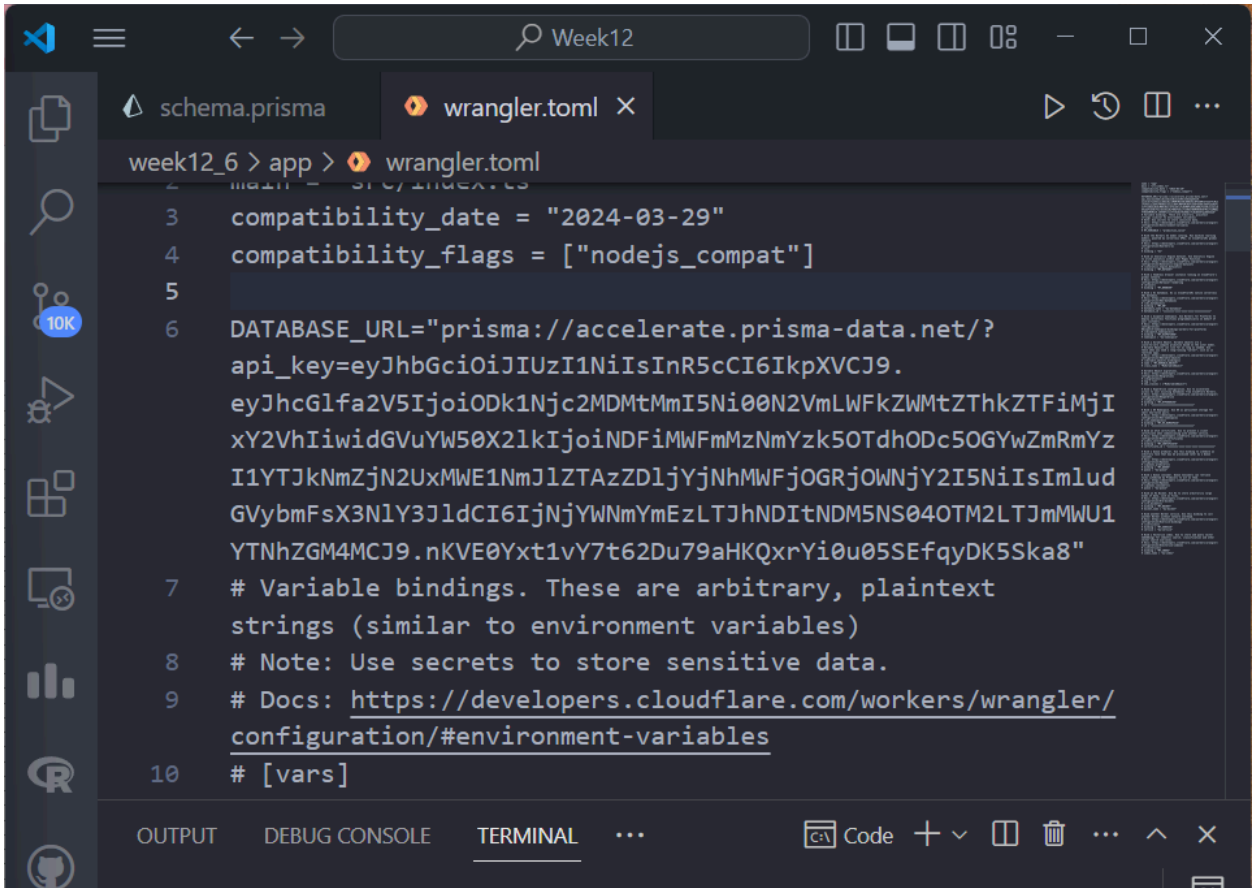
DATABASE_URL="prisma://accelerate.prisma-data.net/?api_key=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcGlfa2V5IjoiODk1Njc2MDMtMmI5Ni00N2VmLWFkZWMtZThkZTFiMjIxY2VhIiwidGVuYW50X2lkIjoiNDFiMWFmMzNmYzk5OTdhODc5OGYwZmRmYzI1YTJkNmZjN2UxMWE1NmJlZTAzZDljYjNhMWFjOGRjOWNjY2I5NiIsImludGVybmFsX3NlY3JldCI6IjNjYWNmYmEzLTJhNDItNDM5NS04OTM2LTJmMWU1YTNhZGM4MCJ9.nKVE0Yxt1vY7t62Du79aHKQxrYi0u05SEfqyDK5Ska8"

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



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