



# Effortlessly building global-scale applications with Cloudflare

ENEI 2025

Luís Duarte - Systems Engineer



## Luís Duarte

Systems Engineer @ ETI  
21y

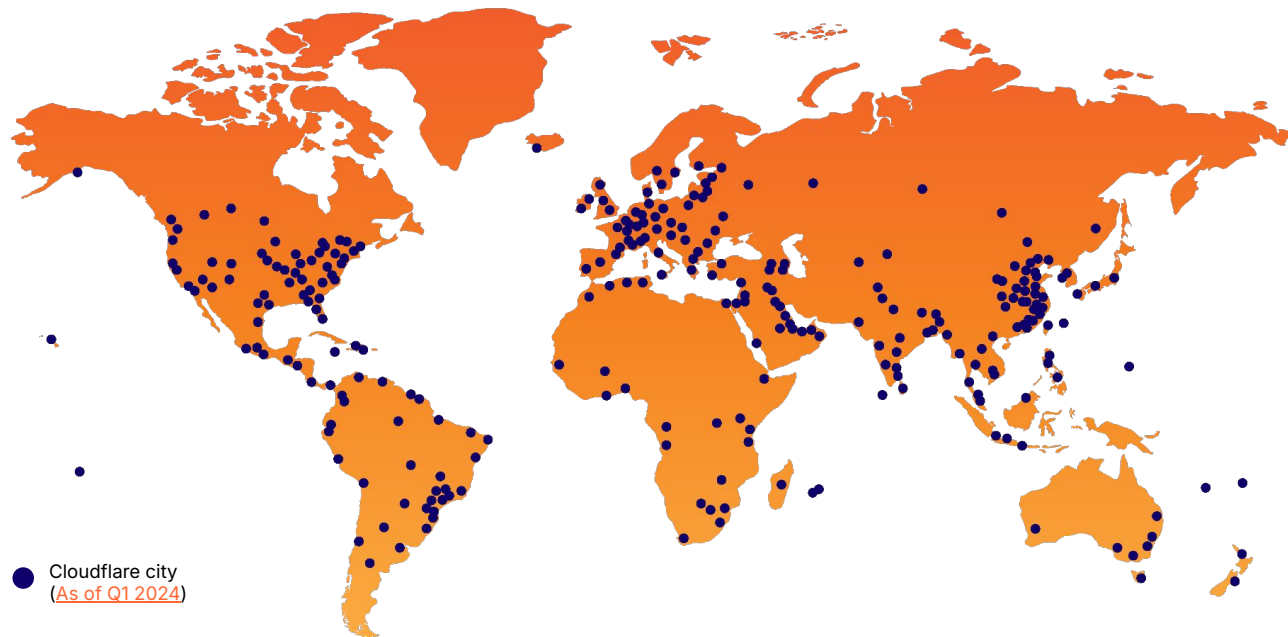
- Workflows
- Email Routing
- workers-rs

Make this a conversation! Ask a question at any point :)

**Follow along at:**  
**[enei25.luisduarte.me](https://enei25.luisduarte.me)**

# Cloudflare is the only composable, Internet-native platform...

...that delivers local capabilities with global scale.



## 320

cities in 120+ countries,  
including mainland China

## 150+

AI inference locations powered by GPUs

## 13,000+

networks directly connect to Cloudflare,  
including every major ISP, cloud  
provider, and enterprise

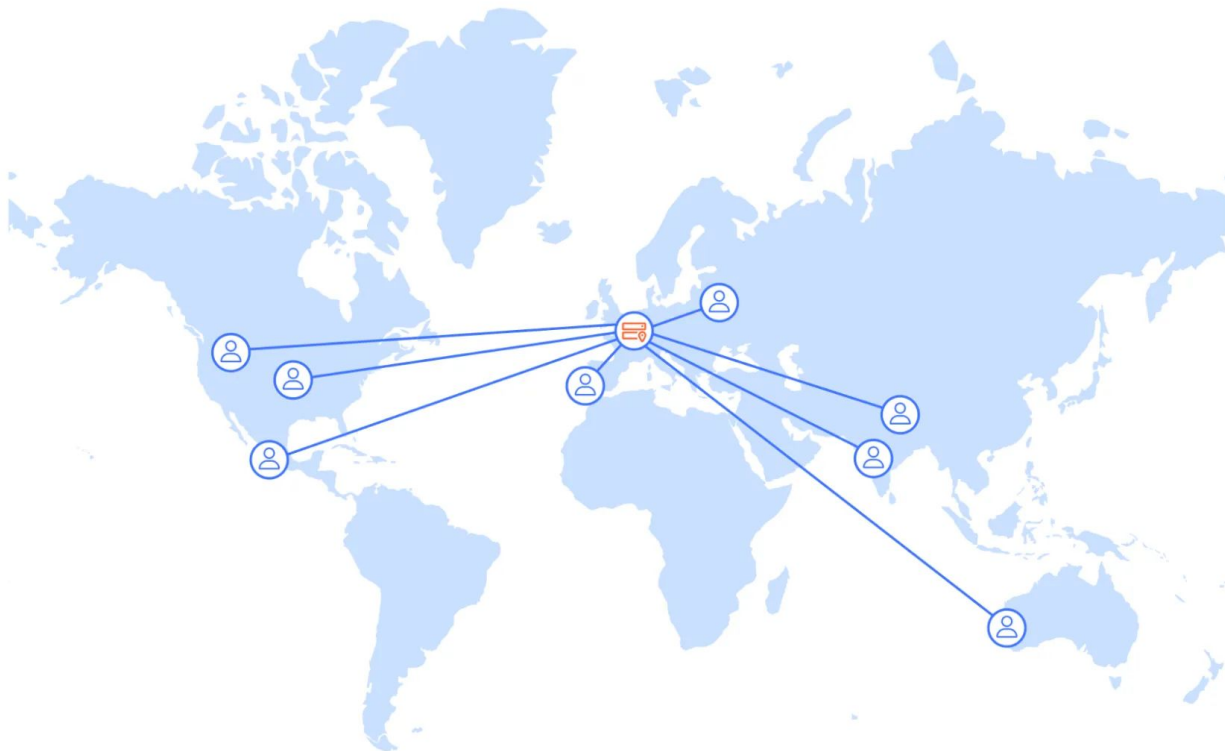
## 280 Tbps

global network edge capacity,  
consisting of transit connections,  
peering and private network  
interconnects

## ~50 ms

from 95% of the world's  
Internet-connected population

# How traditional applications are deployed



# Cloudflare's approach to global-first applications



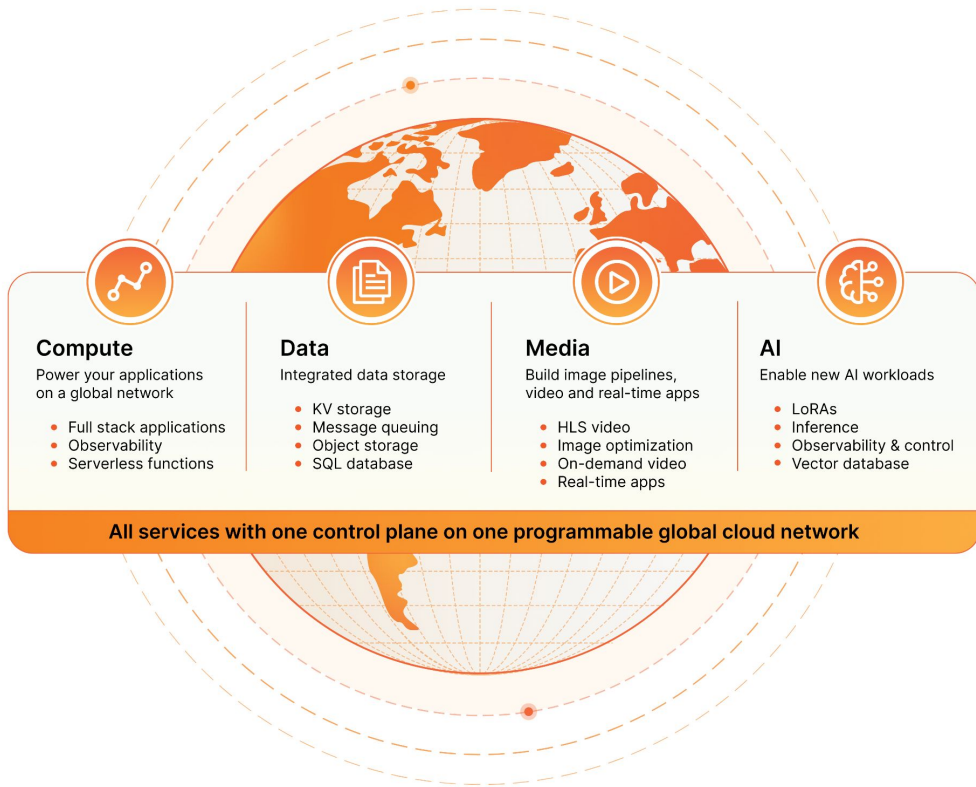
# Deploy applications to region: Earth\*

One unified platform for building full-stack and AI applications uniquely powered by an intelligent, programmable global cloud network.





This enables organizations to ...

- ✓ **regain control,**
- ✓ **lower costs, and**
- ✓ **reduce the risks**

... of a complex and disjointed IT environment.



# Building blocks to create a **full-stack** application 2025

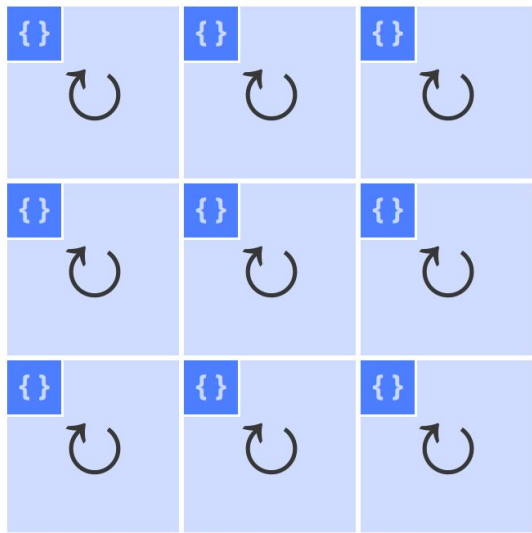
	 CLOUDFLARE	 aws	 Microsoft Azure	 Google Cloud
Serverless Compute	Workers	Lambda / Lambda at Edge/ Cloudfront Functions	Functions	Cloud Functions
Object Storage	R2	S3	Blob Storage	Cloud Storage
Web Development	Pages	Amplify	App Service	Firebase
Queue	Queues	SQS	Queues	Cloud Tasks
Durable Execution	Workflows	Step Function	Durable Functions	Workflows
Relational Database	D1	RDS	Azure SQL Database	Google Cloud SQL
Database Proxy	Hyperdrive	RDS Proxy	Azure SQL Gateway	Cloud SQL Auth Proxy
NoSQL Database	Workers KV / Durable Objects	DynamoDB / DocumentDB / Keyspaces	Azure Cosmos DB	Bigtable / Firestore / Firebase
Analytic Database	Analytic Engine	Redshift	Azure Synapse	BigQuery
Data Streaming	Pipelines	Kinesis	Azure Stream Analytics	Dataflow
API Gateway	API Gateway	API Gateway	API Gateway	API Gateway
Video Streaming	Stream	IVS	Azure Media Services	Live Streaming



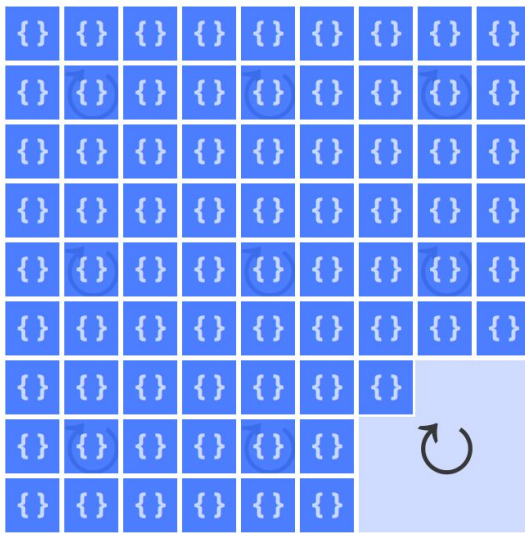
# Cloudflare Workers

- Serverless compute, based on V8 isolates running JavaScript code or WASM. Python support in beta.
- Integrated within the entire Cloudflare network, with dedicated runtime APIs making most of Cloudflare developer platform products available inside your Worker code, like direct access to the CDN cache.
- Fast cold starts, running everywhere on Cloudflare's network.
- The core of workers is open-source: <https://github.com/cloudflare/workerd>
- We focus on the use of **Web Standard APIs** - so there isn't much vendor lock in

# Workers - how they work?



Traditional architecture



Workers V8 isolates



User code



Process overhead

Fun fact: we spin up isolates while the TLS handshake is happening so, in most cases, it appears that cold starts don't exist at all (also an isolate takes just a few milliseconds to spin up)!

# Workers - easy to write

Workers are just Javascript/Typescript you can get started with just 4 lines:

```
export default {  
  async fetch(request, env, ctx) {  
    return new Response("Hello World!");  
  },  
};
```





## Workers - limits

Because Workers run globally (and we can have millions of instances of your code running) they have the following resource limits:

Feature	Workers Free	Workers Paid
<a href="#">Request</a>	100,000 requests/day 1000 requests/min	No limit
<a href="#">Worker memory</a>	128 MB	128 MB
<a href="#">CPU time</a>	10 ms	5 min HTTP request 15 min <a href="#">Cron Trigger</a>
<a href="#">Duration</a>	No limit	No limit for Workers. 15 min duration limit for <a href="#">Cron Triggers</a> , <a href="#">Durable Object Alarms</a> and <a href="#">Queue Consumers</a>

**This works for stateless  
applications - what  
about stateful ones?**








# Building blocks to create a **full-stack** application 2025

	 CLOUDFLARE	 aws	 Microsoft Azure	 Google Cloud
Serverless Compute	Workers	Lambda / Lambda at Edge/ Cloudfront Functions	Functions	Cloud Functions
Object Storage	R2	S3	Blob Storage	Cloud Storage
Web Development	Pages	Amplify	App Service	Firebase
Queue	Queues	SQS	Queues	Cloud Tasks
Durable Execution	Workflows	Step Function	Durable Functions	Workflows
Relational Database	D1	RDS	Azure SQL Database	Google Cloud SQL
Database Proxy	Hyperdrive	RDS Proxy	Azure SQL Gateway	Cloud SQL Auth Proxy
NoSQL Database	Workers KV / Durable Objects	DynamoDB / DocumentDB / Keyspaces	Azure Cosmos DB	Bigtable / Firestore / Firebase
Analytic Database	Analytic Engine	Redshift	Azure Synapse	BigQuery
Data Streaming	Pipelines	Kinesis	Azure Stream Analytics	Dataflow
API Gateway	API Gateway	API Gateway	API Gateway	API Gateway
Video Streaming	Stream	IVS	Azure Media Services	Live Streaming

# The actor model

A way to build programs that do many things at once (concurrency).

Actors are like little workers:

-  They get messages
-  They decide what to do
-  They can create other actors
-  They send messages to others
-  No shared memory = no messy bugs with multiple things changing the same data.
-  Great for apps that need to scale or handle errors well (like chat apps or games).
-  Used in: Erlang, Elixir, Akka (Scala/Java) and Durable Objects 🧐

# Durable Objects

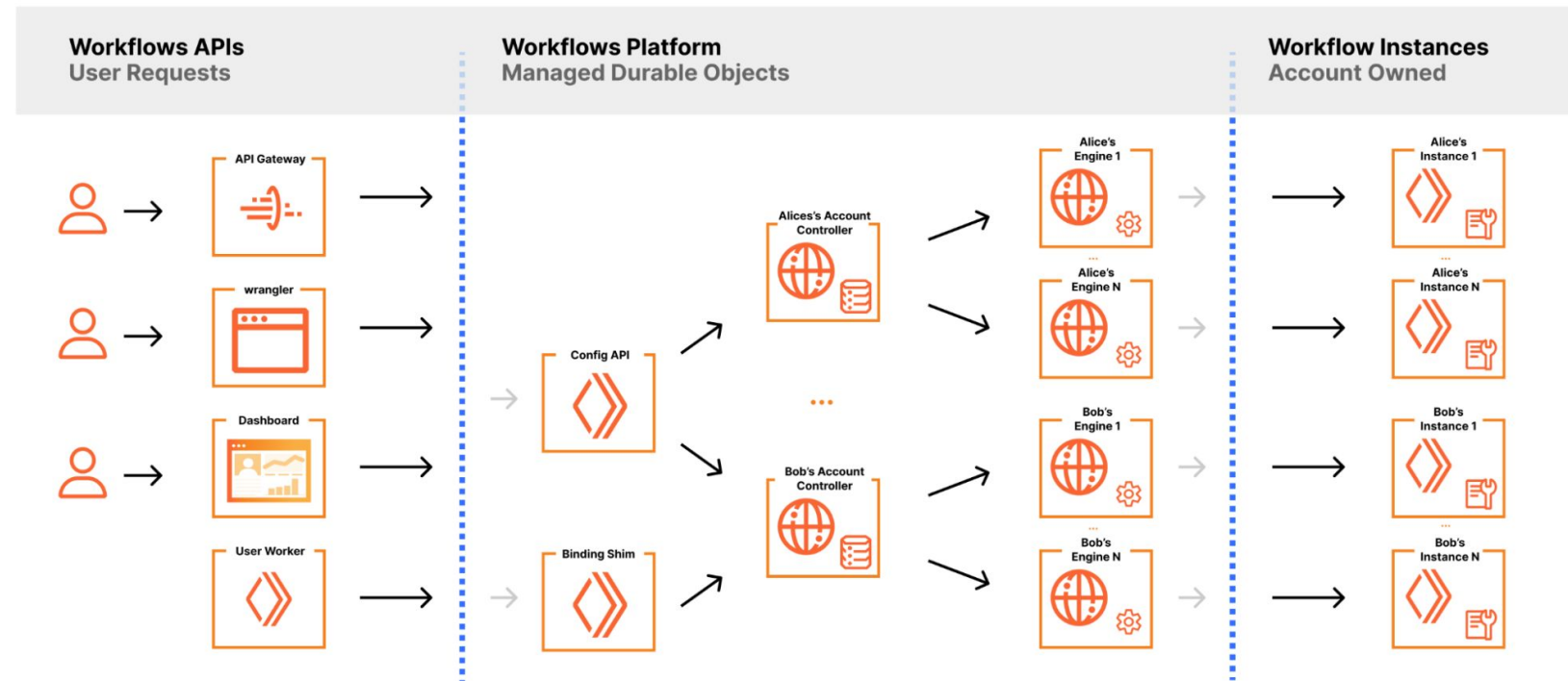
- Durable Objects combine compute, durable storage across requests, and Cloudflare's edge network. Actors at the infrastructure level.
  - Read [Zero-latency SQLite storage in every Durable Object](#) blog post.
- Each Durable Object (instance) is essentially a mini-computer with 128MB of memory, running your Worker code, with persisted durable storage (10GB) with KV or SQLite API.
- Create millions of them, programmatically! 🚀
- Each Durable Object can be created in the "region" of your choice or by default close to your request location.
  - <https://where.durableobjects.live/>



# Durable Objects - their capabilities

- DOs can be WebSockets servers just by using the standard WebSockets API. The advantage here is that they can hibernate if there's no activity in the active connection - meaning that you are only billed for the DOs if there are messages passing.
- DOs can set alarms to be woken up at a specific time in the future.
- DOs supports JSRPC (Remote Procedure Call) meaning that you can only use Javascript methods and properties to communicate with other DOs or Workers.

# Durable Objects - the building block of Workflows



# Making Durable Objects scale

As you can imagine, you can't have a single JS thread handling all the load, so you need a way to **shard/divide** your application.

As an example, Workflows are sharded by:

1. Account
2. Workflow run/instance (for storing the workflow state)

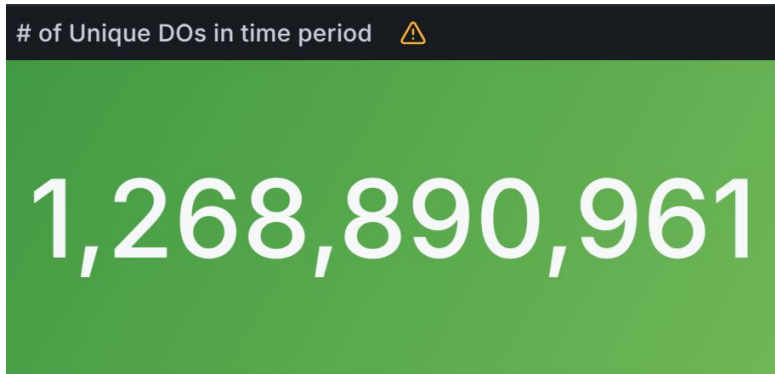
So, in order to use DOs, we must think if/how to logically divide the application (not always possible!)



*What happens if you only  
use 1 DO*

# Making Durable Objects scale

You have to find a balance on logically dividing parts of your application - the tradeoff is added latency. But when done correctly, you can horizontally scale with ease.



An internal application has literally billions of DOs active at the same time!



# Onto the practical part!

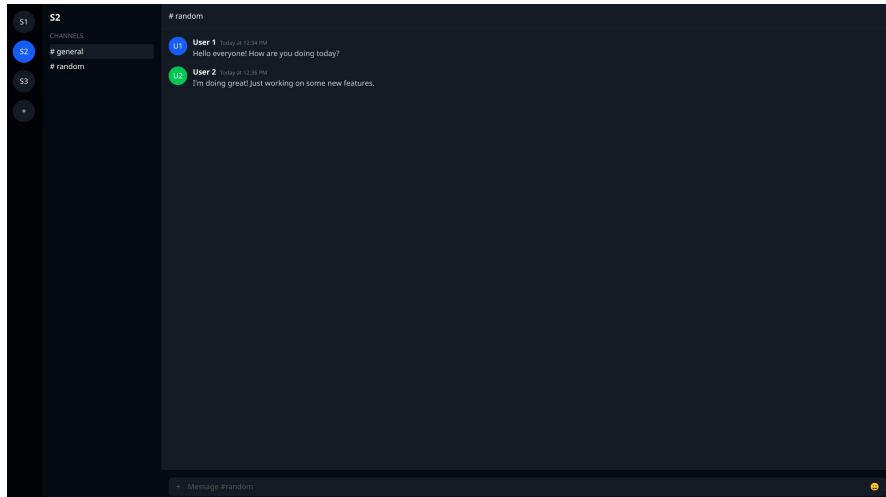
# What are you going to need?

- Node.js 20+
- pnpm
- If you wish to deploy your app to production, a free Cloudflare account (optionally, since you can run just locally)

# Accord - a real-time chat application

Let's imagine that you've entered a new startup - Accord - it's a real-time chat application for large communities, and you've tasked to make the **backend** for the **servers** and **chatrooms**.

(Yes, it's the opposite to Discord)



## Accord - the stack



**Vite + React + Hono + Cloudflare**



# What are we going to build?

- API - create new Server
- API - create new Channel
- WS - receive and send messages
- (and pipe it to the frontend - I've already created the frontend for you)

We're going to skip a few things such as Authentication/Authorization, account creation and so on.

## Accord - clone the template!

<https://github.com/LuisDuarte1/accord-template>

**How would you design  
Accord with Durable  
Objects or other  
products?**

# My Proposal

- We use D1 for tracking the server list (at scale, this is relatively okay to do since servers aren't created/deleted that often)
- We also use D1 for tracking channel lists, for the same reason (and to save time)
- We create a single DO namespaces: **Channels**
- **Channels** keep actual messages of that channel - this will be the DO that will use **WebSockets** for real-time messaging.

Feel free to use another architecture if you see that it provides other advantages that you'll like :)

## Accord endpoints

- `/api/servers` - GET and POST
- `/servers/:server_id/channels` - GET and POST
- `/servers/:server_id/channels/:channel_id/messages` - GET
- `/servers/:server_id/channels/:channel_id/connect` - WebSockets upgrade endpoint

It's the ones that are used in the frontend - feel free to change them at your will

# Time to build!

# Useful documentation

- Hono docs: <https://hono.dev/docs/>
- D1 docs:  
<https://developers.cloudflare.com/d1/best-practices/import-export-data/>  
<https://developers.cloudflare.com/d1/worker-api/d1-database/>
- Durable Object docs:  
<https://developers.cloudflare.com/durable-objects/get-started/>  
<https://developers.cloudflare.com/durable-objects/best-practices/websockets/>  
<https://developers.cloudflare.com/durable-objects/best-practices/access-durable-objects-storage/>

# Thank you!

# Any questions?

✕ - @imluisduarte

Email:  
[ping@luisduarte.me](mailto:ping@luisduarte.me)