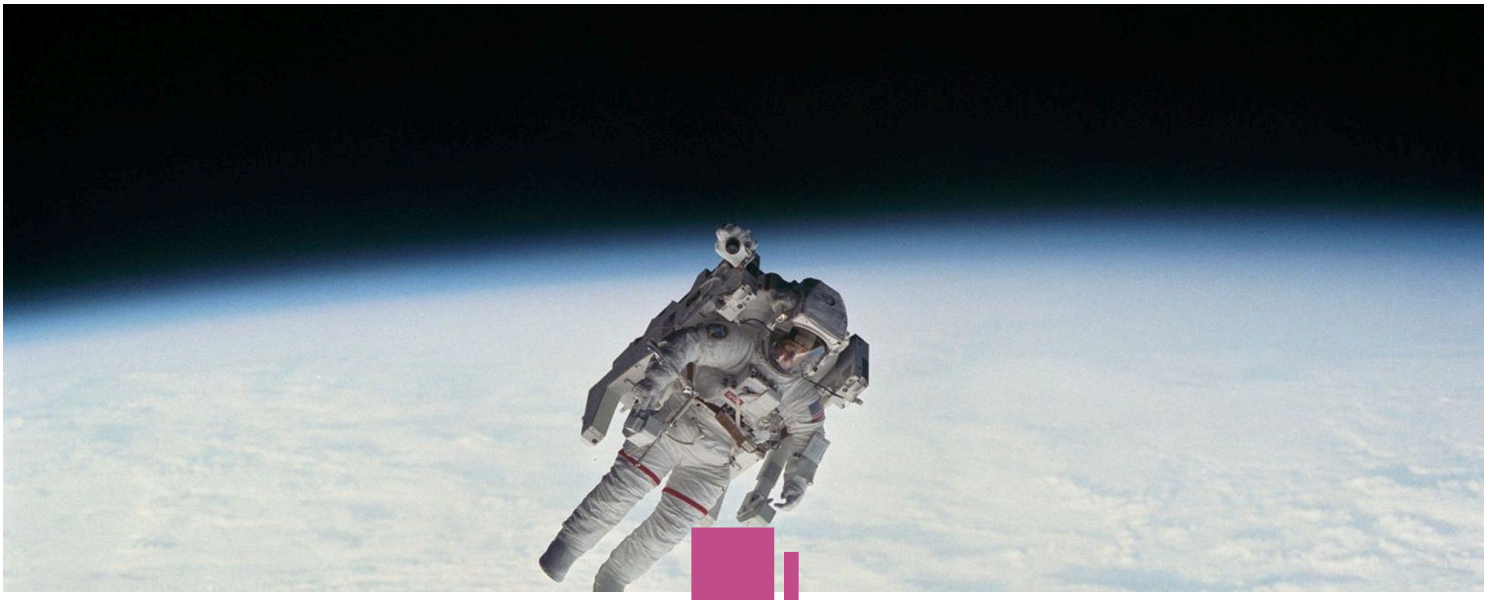


Week 12.6

Connection Pooling In Serverless ENV

In today's lecture, Harkirat explores `connection pooling in serverless environments`, emphasizing its importance for managing the high volume of database connections that platforms like `Cloudflare Workers` generate. The discussion also covered the challenges of using Prisma ORM in such settings and introduced `Prisma Accelerate` as a managed service for efficient connection pooling, ensuring scalable and stable database access for serverless applications.

The wrangler.toml File

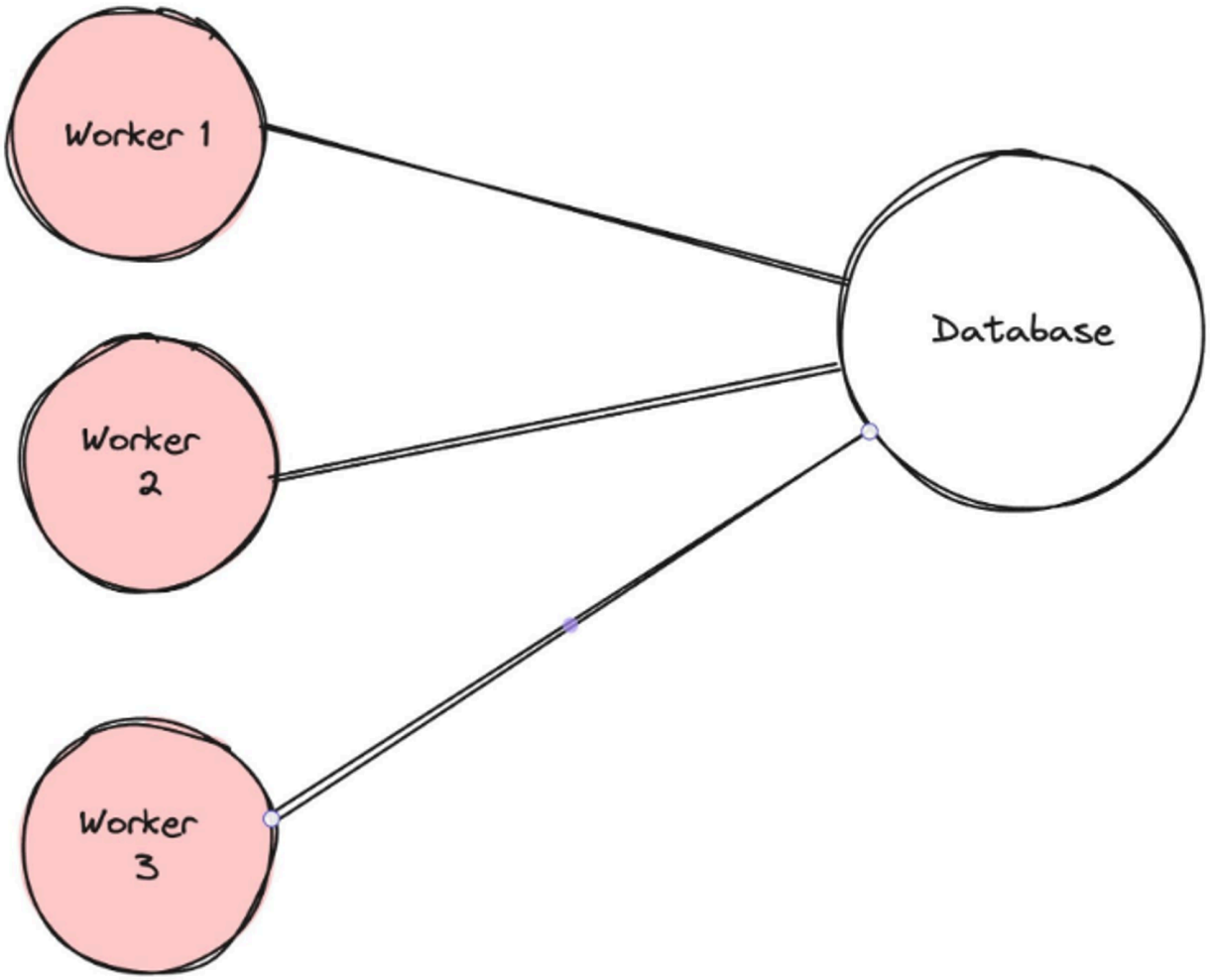
Connection Pooling

Connection pooling is a technique used to manage and reuse database connections efficiently. It is particularly relevant in serverless environments and applications with high levels of concurrency. Here's a detailed explanation of connection pooling and its importance:

What is Connection Pooling?

Connection pooling refers to the practice of maintaining a cache of database connection objects that can be reused by multiple clients. Instead of opening and closing a new connection for each user request, a connection pool allows a set of connections to be shared among requesting threads or processes. When a new request comes in, it can borrow a connection from the pool, use it for the database operation, and then return it to the pool for future use.

Serverless environments, such as Cloudflare Workers, present unique challenges for database connectivity. Since serverless functions can scale up rapidly and run in multiple regions, they can potentially open a large number of connections to the database. This can quickly exhaust the database's connection limit and degrade performance.

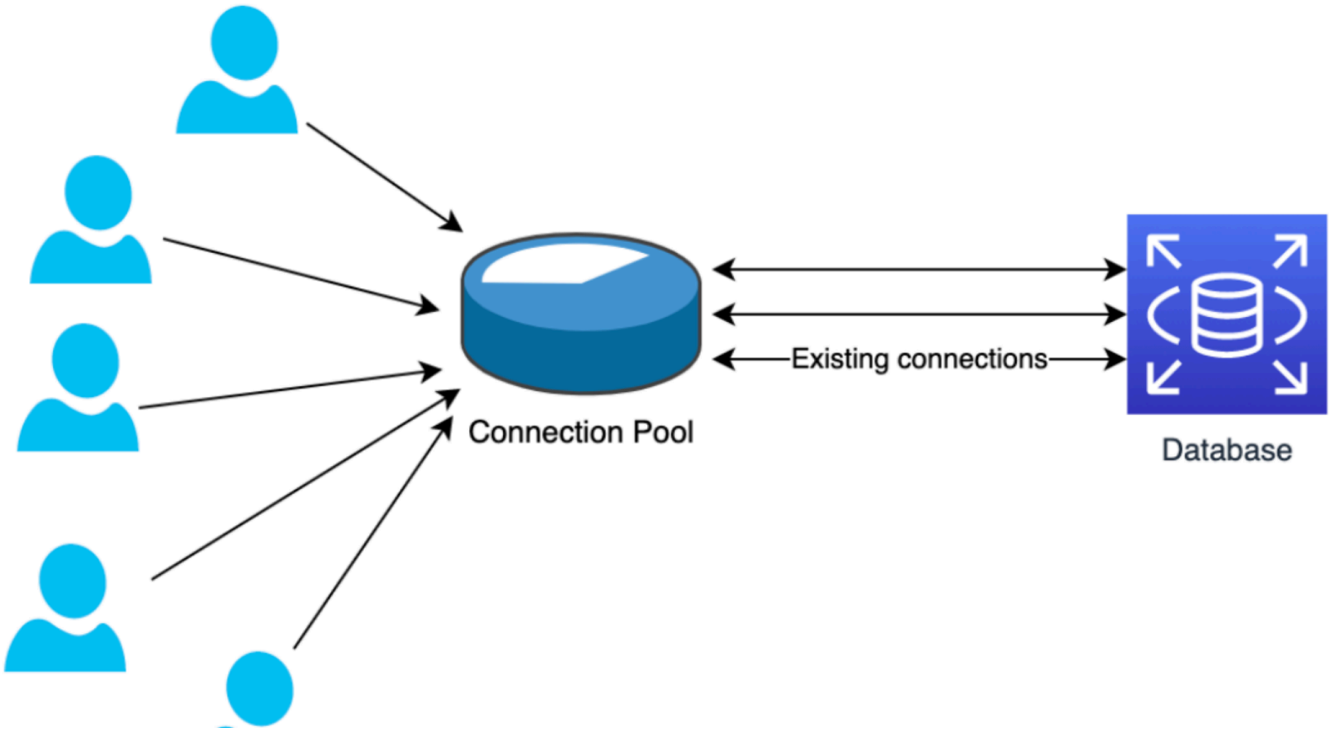


The above image illustrates a scenario where multiple workers (Worker 1, Worker 2, Worker 3) are each establishing their own connections to a single database. In a serverless environment, this can lead to a high number of simultaneous database connections, which can strain the database server and potentially exceed its connection limit. Using a connection pool can resolve this issue by acting as an intermediary that manages database connections on behalf of the workers. Here's how it works:

1. **Centralized Connection Management**: The connection pool maintains a set of open connections to the database. Instead of each worker opening its own connection, they request a connection from the pool.
2. **Efficient Resource Utilization**: When a worker needs to interact with the database, it checks out a connection from the pool, uses it for the database operation, and then returns it to the pool. This allows connections to be reused, reducing the overhead of establishing new connections for each operation.
3. **Controlled Concurrency**: The pool limits the number of active connections. If all connections in the pool are in use and another worker requests a connection, it will have to wait until a

connection is returned to the pool. This prevents the database from being overwhelmed with too many concurrent connections.

4. **Improved Performance**: Connection pooling improves the performance of the system by reducing the time spent on connection setup and teardown. It also helps in maintaining a stable number of connections, which can be tuned for optimal performance based on the database's capacity.
5. **Scalability**: As the number of workers increases, the connection pool can help scale the system more effectively. It ensures that the growth in the number of workers does not lead to a proportional increase in the number of database connections, which could otherwise lead to scalability issues.



> In summary, a connection pool collects all the database requests from the workers and manages the connections collectively. This approach streamlines the process of connecting to the database, enhances performance, and ensures that the database server is not overloaded with connection requests. It is a critical component in serverless architectures where the number of instances can fluctuate significantly, and efficient resource management is key to maintaining system stability and performance.

Connection Pooling In Prisma for Serverless ENV

Connection pooling in Prisma for serverless environments is a crucial feature that addresses the challenges of managing database connections in a scalable and efficient manner. Prisma Accelerate is a service that enhances Prisma's capabilities in serverless deployments, such as those on Cloudflare Workers. Here's a detailed explanation of the steps to set up Prisma with connection pooling using Prisma Accelerate:

1] Install Prisma in Your Project:

Add Prisma to your project as a development dependency using npm.

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

2] Initialize Prisma:

Set up the initial Prisma configuration files in your project.

```
npx prisma init
```

3] Create a Basic Schema:

Define your data model in the Prisma schema file. This includes specifying the database provider (e.g., PostgreSQL) and the URL, which will later be replaced with the Prisma Accelerate connection string.

```
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:

Generate the necessary database migrations based on your schema.

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

5] Sign Up to Prisma Accelerate:

Register for Prisma Accelerate, which provides a managed connection pooling solution for serverless applications.

6] Generate an API Key:

Obtain an API key from Prisma Accelerate and replace the placeholder in your `.env` file with the actual key.

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

7] Add Accelerate as a Dependency:

Install the Prisma Accelerate extension to enable connection pooling.

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

8] Generate the Prisma Client Without Binary Engines:

Create the Prisma client tailored for serverless environments.

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

9] Set Up Your Code:

Integrate Prisma Client with Prisma Accelerate in your serverless application code. Here's an example using the Hono framework:

```
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 = 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: "User created"})
})

export default app
```

> In this setup, Prisma Accelerate acts as the connection pool manager. It collects all the database requests from the serverless functions (workers) and manages the connections to the database. This ensures that the number of connections

> does not exceed the database's capacity and that the connections are efficiently reused, improving performance and scalability in serverless environments.

The `wrangler.toml` File

The `wrangler.toml` file is a configuration file used by Wrangler, which is a command-line interface (CLI) tool for Cloudflare Workers. In the context of using Prisma with connection pooling in a serverless environment like Cloudflare Workers, the `wrangler.toml` file would specify the settings and parameters required to deploy and configure the serverless application.

Here's how the `wrangler.toml` file relates to the setup:

- **Project Configuration**: The `wrangler.toml` file contains various configuration options for your Cloudflare Workers project, such as the account ID, project name, and environment variables.
- **Environment Variables**: You can define environment variables in the `wrangler.toml` file, which might include the `DATABASE_URL` for Prisma. This URL would be the connection string provided by Prisma Accelerate, which includes the API key and other necessary parameters for connection pooling.
- **Workers Script Settings**: The file also includes settings related to the Workers script itself, such as the entry point for the application code and any build commands that need to be run before deployment.
- **Dependencies**: If your Worker uses npm packages, such as `@prisma/client` and `@prisma/extension-accelerate`, you may need to specify how these dependencies are bundled with your Worker script.
- **Routes and Triggers**: The `wrangler.toml` file can also define routes and triggers that determine when your Worker is executed in response to HTTP requests.

> The `wrangler.toml` file is essential for deploying a serverless application that uses Prisma with connection pooling on Cloudflare Workers. It helps manage the deployment process and ensures that the necessary environment variables and settings are in place for the application to connect to the database using Prisma Accelerate's managed connection pooling service.