Integrating AsyncIO and AioHTTP for Scalable Systems

Dan Tofan
PhD

@dan_tofan | https://programmingwithdan.com

Building Scalable AioHTTP Servers

Routing optimization

Server configuration

Advanced scaling steps

AioHTTP Routing Optimization

Consistent naming

Logical grouping

Pattern-matching efficiency

AioHTTP Server Configuration

Backlog size

Keep alive timeout

Maximum request size

Request throttling

Advanced Scaling Steps

Gunicorn

Nginx

Kubernetes

Demo: Building a Scalable AioHTTP Server

Running multiple workers with Gunicorn

Real-time Data Processing with Asyncio and AioHTTP

Server-sent events

WebSockets

Understanding Server-sent Events

Persistent connection

One-way communication

Working with HTTP

Supported in browsers

Use Cases for Server-Sent Events

- News feeds
- Price updates
- Notifications
- Progress updates
- Live dashboards
- Sensor data

Understanding WebSockets

Persistent connection

Two-way communication

Low overhead

Supported in browsers

Use Cases for WebSockets

- Chat applications
- Real-time collaboration tools
- Multiplayer games
- Interactive dashboards

```
async def sse_handler(request):
 r = web.StreamResponse()

 r.headers['Content-Type'] =\
                     'text/event-stream'
 r.headers['Cache-Control'] = 'no-cache'
 r.headers['Connection'] = 'keep-alive'

 await r.prepare(request)

 while True:
   data = "Temperature: 70 \n"

   await r.write(data.encode('utf-8'))

   await asyncio.sleep(1)
```

- Initialize a streaming response
- Set headers
- Send headers to client
- Stream data events indefinitely
- Send event data chunk
- Non-blocking delay for 1 second

Demo: Streaming Real-time Data

Optimizing Database Operations

Using async database drivers

Implementing connection pools

Async Database Drivers

PostgreSQL
Use asyncpg

MySQL
Use aiomysql

SQLite
Use aiosqlite

Redis
Use aioredis

MongoDB
Use motor

Why Implement Connection Pools?

Manage lifecycle

Reduced latency

Better scalability

Implementing a PostgreSQL Connection Pool

```
async def init_pg_pool(app):
     app['pg_pool'] = await asyncpg.create_pool(
          dsn='postgresql://user:pass@server:port/db',
          min_size=5,
          max_size=10)

async def close_pg_pool(app):
     await app['pg_pool'].close()

app = web.Application()

app.on_startup.append(init_pg_pool)
app.on_cleanup.append(close_pg_pool)
```

Demo: Efficient Database Operations

Course Summary

Why asynchronous programming?

Implementing asynchronous HTTP with Aiohttp

Designing efficient workflows with AsyncIO

Integrating AsyncIO and Aiohttp for scalable systems