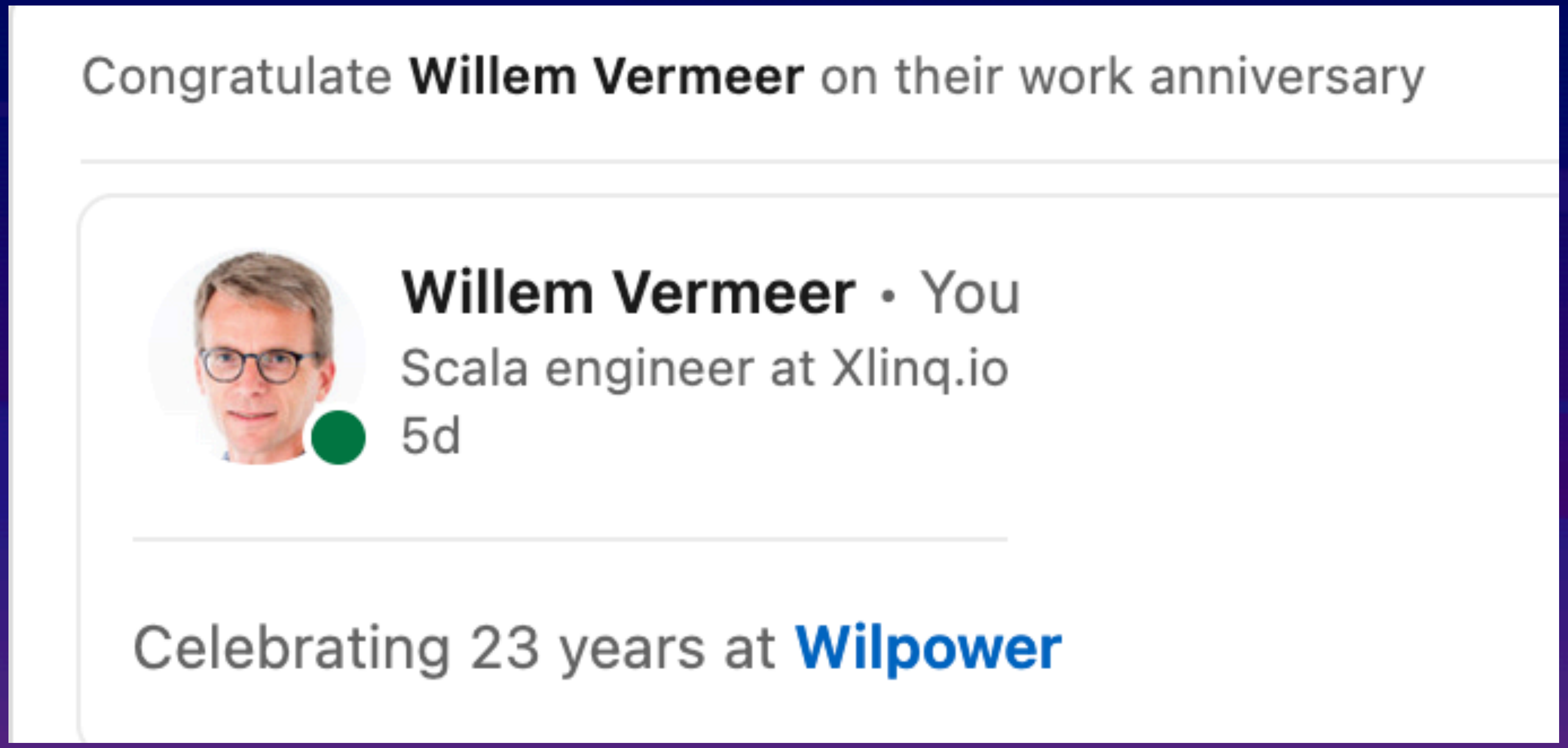
Performance Comparison

ZIO vs Akka-http vs Rust

Willem Vermeer - Friday April 21st 2023 - ZIOWorld

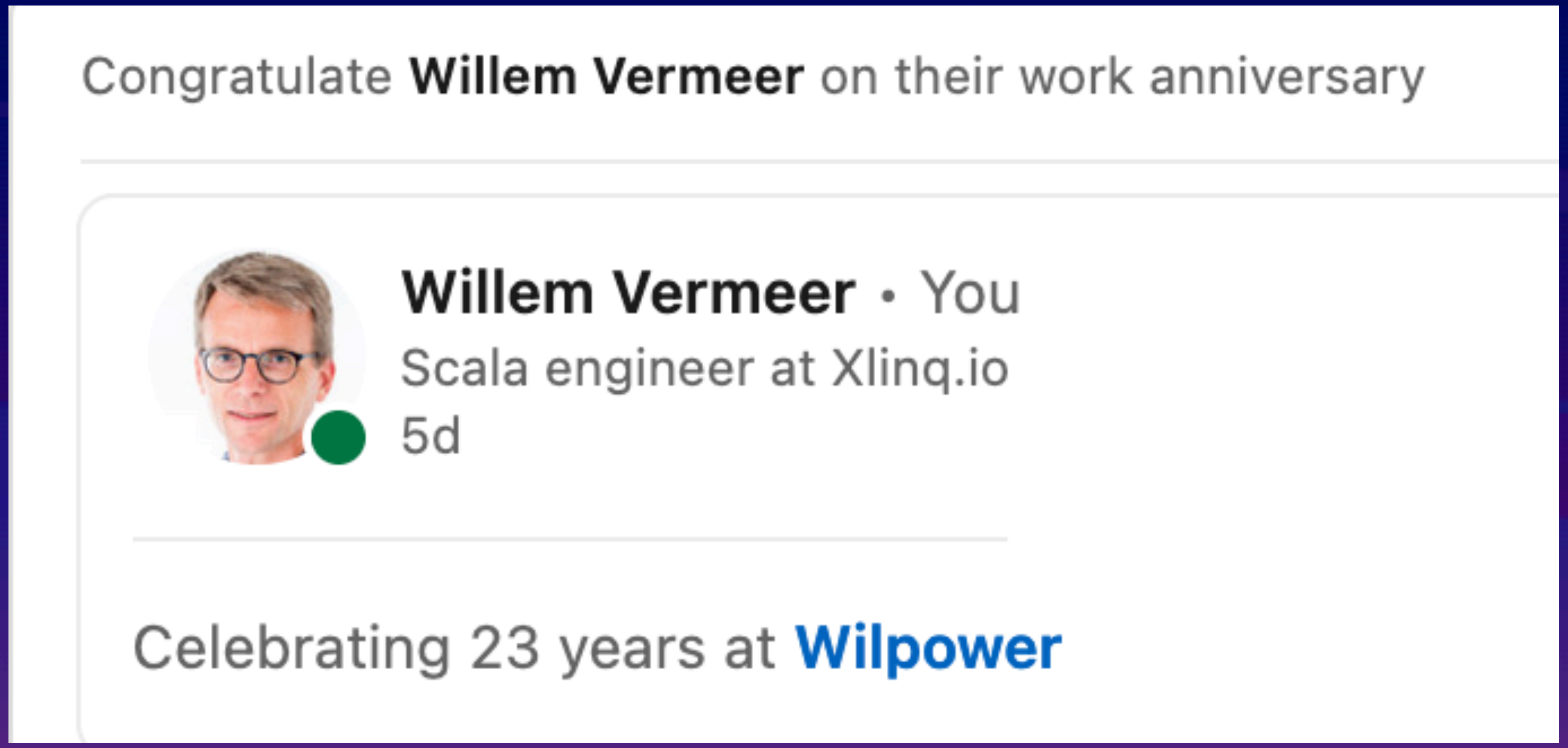
Hello!


Congratulate **Willem Vermeer** on their work anniversary

Willem Vermeer • You
Scala engineer at Xlinq.io
5d

Celebrating 23 years at **Wilpower**

Hello!


Congratulate **Willem Vermeer** on their work anniversary

Willem Vermeer • You
Scala engineer at Xlinq.io
5d

Celebrating 23 years at **Wilpower**

What happens in ZIO, stays in ZIO
functionalscala.com

Willem Vermeer - 3 December 2020
@willemvermeer

Motivation

- Using akka-http for several scala micro-services
- What are the alternatives?

Motivation

- Using akka-http for several scala micro-services
- What are the alternatives?
  - ZIO http production ready?

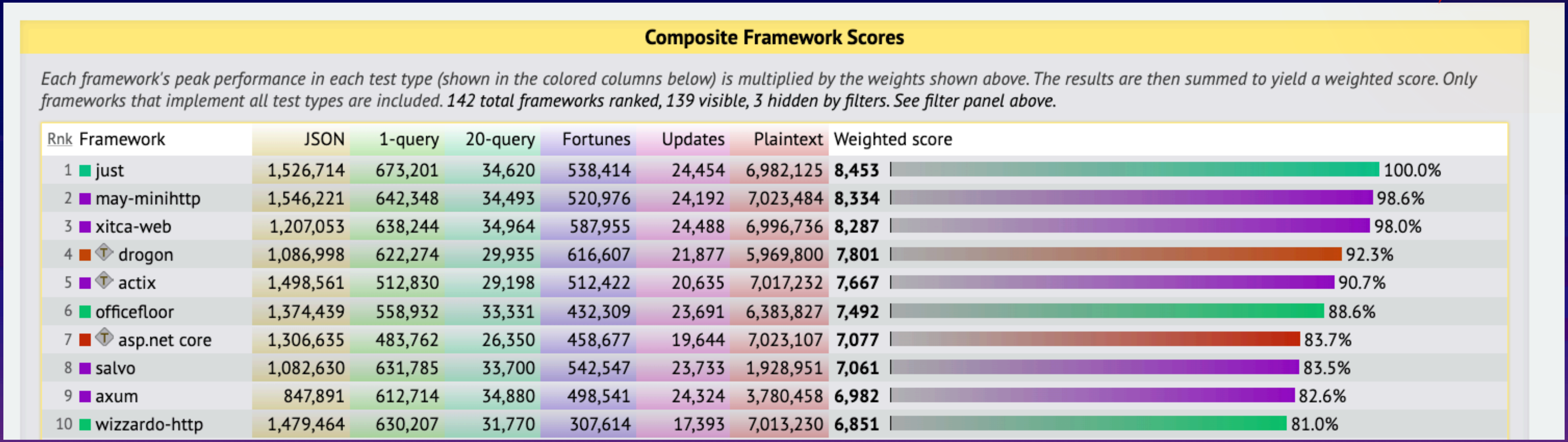
Motivation

- Using akka-http for several scala micro-services
- What are the alternatives?
  - ZIO http production ready?
  - Good excuse to learn Rust!
- How would these alternatives perform?

https://www.redbubble.com/i/t-shirt/rewrite-it-in-rust-the-rust-programming-language-by-MORGHANHENNESSY/128333416.IJ6Lo

Techempower benchmarks

Techempower benchmarks



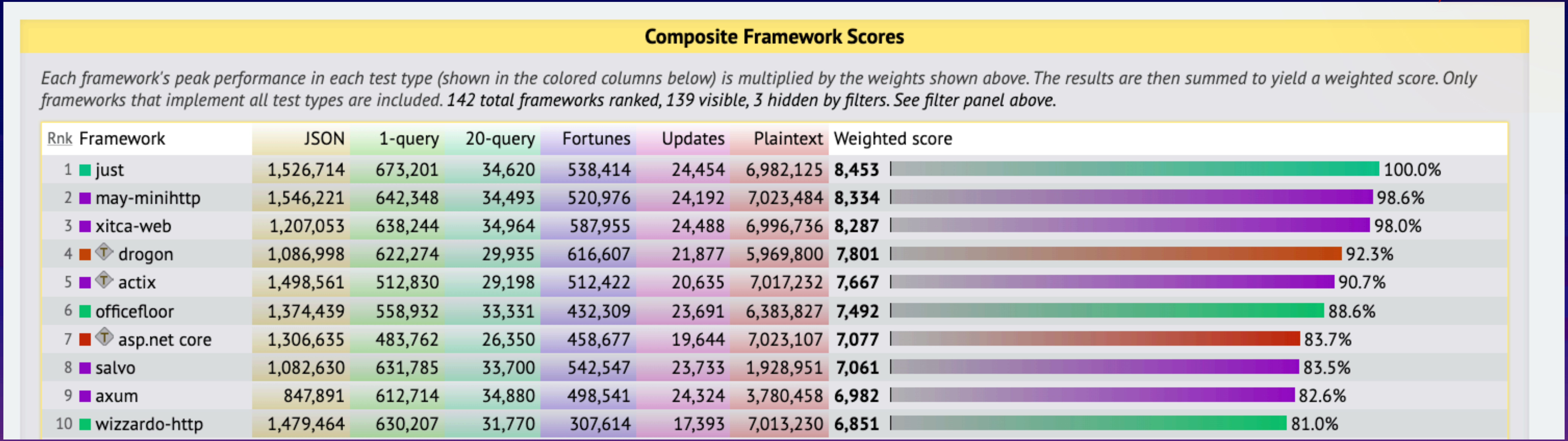
Composite Framework Scores

Each framework's peak performance in each test type (shown in the colored columns below) is multiplied by the weights shown above. The results are then summed to yield a weighted score. Only frameworks that implement all test types are included. 142 total frameworks ranked, 139 visible, 3 hidden by filters. See filter panel above.

Rnk	Framework	JSON	1-query	20-query	Fortunes	Updates	Plaintext	Weighted score	
1	🟩 just	1,526,714	673,201	34,620	538,414	24,454	6,982,125	**8,453**	100.0%
2	🟪 may-minihttp	1,546,221	642,348	34,493	520,976	24,192	7,023,484	**8,334**	98.6%
3	🟪 xitca-web	1,207,053	638,244	34,964	587,955	24,488	6,996,736	**8,287**	98.0%
4	🟧 T drogon	1,086,998	622,274	29,935	616,607	21,877	5,969,800	**7,801**	92.3%
5	🟪 T actix	1,498,561	512,830	29,198	512,422	20,635	7,017,232	**7,667**	90.7%
6	🟩 officefloor	1,374,439	558,932	33,331	432,309	23,691	6,383,827	**7,492**	88.6%
7	🟥 T asp.net core	1,306,635	483,762	26,350	458,677	19,644	7,023,107	**7,077**	83.7%
8	🟪 salvo	1,082,630	631,785	33,700	542,547	23,733	1,928,951	**7,061**	83.5%
9	🟪 axum	847,891	612,714	34,880	498,541	24,324	3,780,458	**6,982**	82.6%
10	🟩 wizzardo-http	1,479,464	630,207	31,770	307,614	17,393	7,013,230	**6,851**	81.0%

- https://www.techempower.com/benchmarks/#section=data-r21
- 5 out of top 10 are rust

Techempower benchmarks

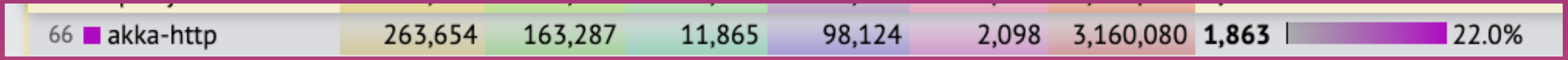


Composite Framework Scores

Each framework's peak performance in each test type (shown in the colored columns below) is multiplied by the weights shown above. The results are then summed to yield a weighted score. Only frameworks that implement all test types are included. 142 total frameworks ranked, 139 visible, 3 hidden by filters. See filter panel above.

Rnk	Framework	JSON	1-query	20-query	Fortunes	Updates	Plaintext	Weighted score
1	🟩 just	1,526,714	673,201	34,620	538,414	24,454	6,982,125	**8,453** 100.0%
2	🟪 may-minihttp	1,546,221	642,348	34,493	520,976	24,192	7,023,484	**8,334** 98.6%
3	🟪 xitca-web	1,207,053	638,244	34,964	587,955	24,488	6,996,736	**8,287** 98.0%
4	🟧 drogon	1,086,998	622,274	29,935	616,607	21,877	5,969,800	**7,801** 92.3%
5	🟪 actix	1,498,561	512,830	29,198	512,422	20,635	7,017,232	**7,667** 90.7%
6	🟩 officefloor	1,374,439	558,932	33,331	432,309	23,691	6,383,827	**7,492** 88.6%
7	🟥 asp.net core	1,306,635	483,762	26,350	458,677	19,644	7,023,107	**7,077** 83.7%
8	🟪 salvo	1,082,630	631,785	33,700	542,547	23,733	1,928,951	**7,061** 83.5%
9	🟪 axum	847,891	612,714	34,880	498,541	24,324	3,780,458	**6,982** 82.6%
10	🟩 wizzardo-http	1,479,464	630,207	31,770	307,614	17,393	7,013,230	**6,851** 81.0%

- https://www.techempower.com/benchmarks/#section=data-r21
- 5 out of top 10 are rust
- Akka-http entry

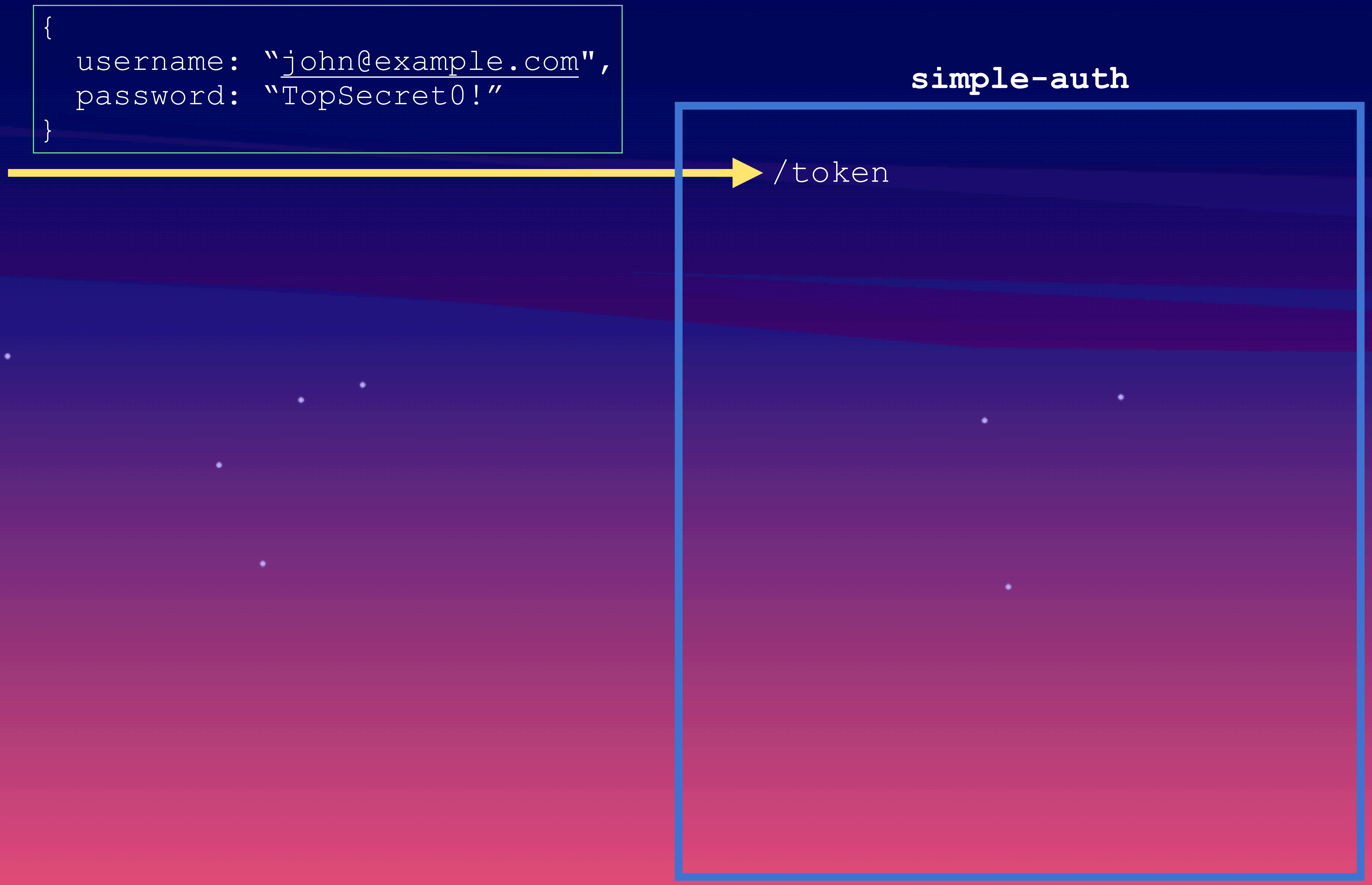


66	🟪 akka-http	263,654	163,287	11,865	98,124	2,098	3,160,080	**1,863** 22.0%

- Zio implementation 2 years old, not appearing in results

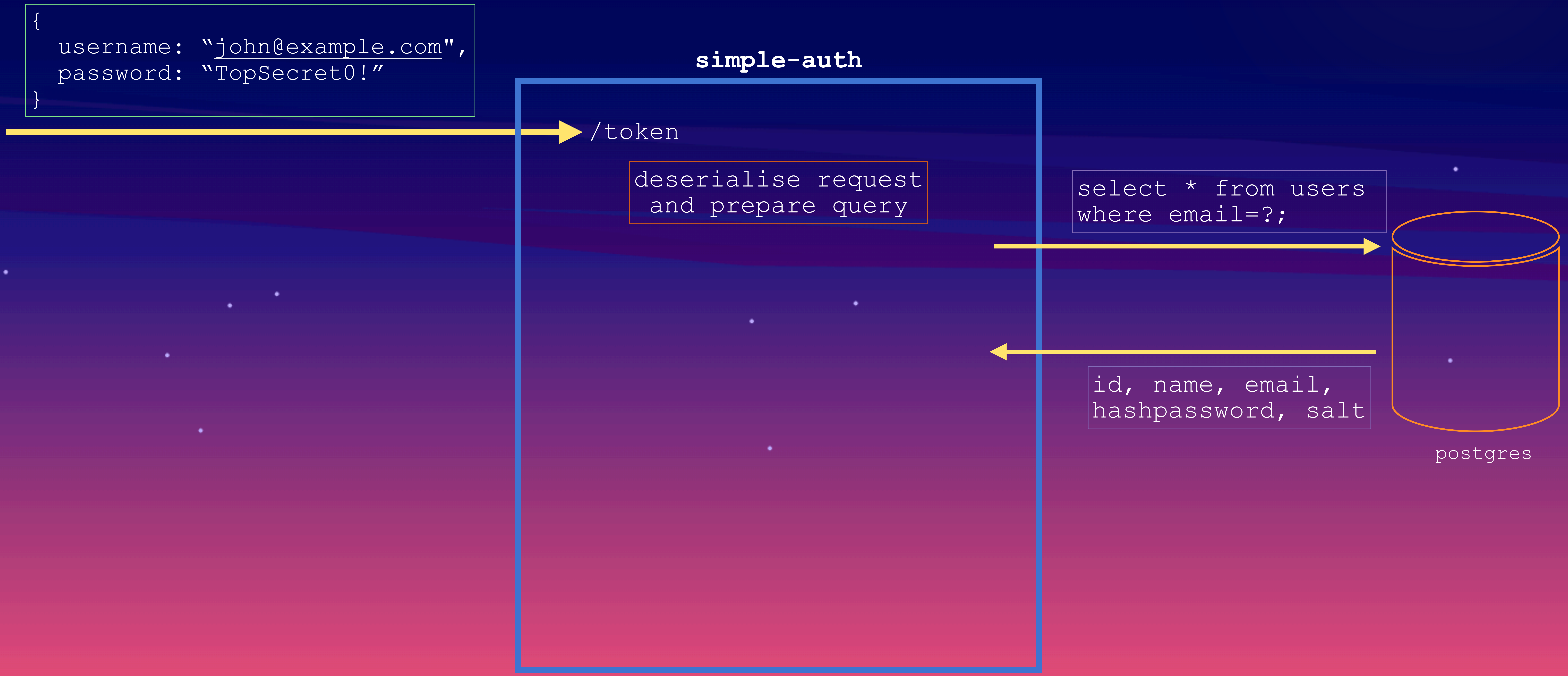
Use case - "Simple Auth"

Use case - “Simple Auth”



```
{
  username: “john@example.com",
  password: “TopSecret0!”
}
```

Use case - “Simple Auth”

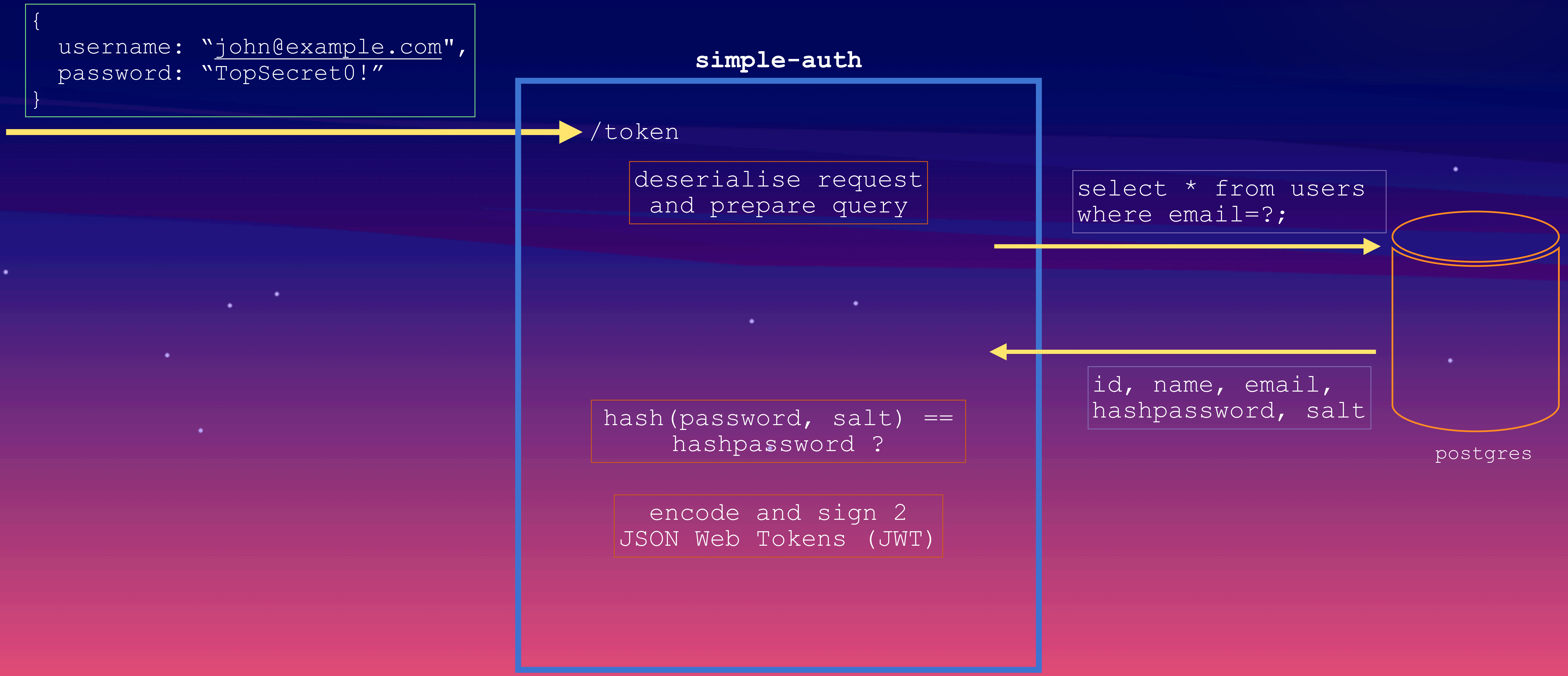


```
{
  username: “john@example.com",
  password: “TopSecret0!”
}
```

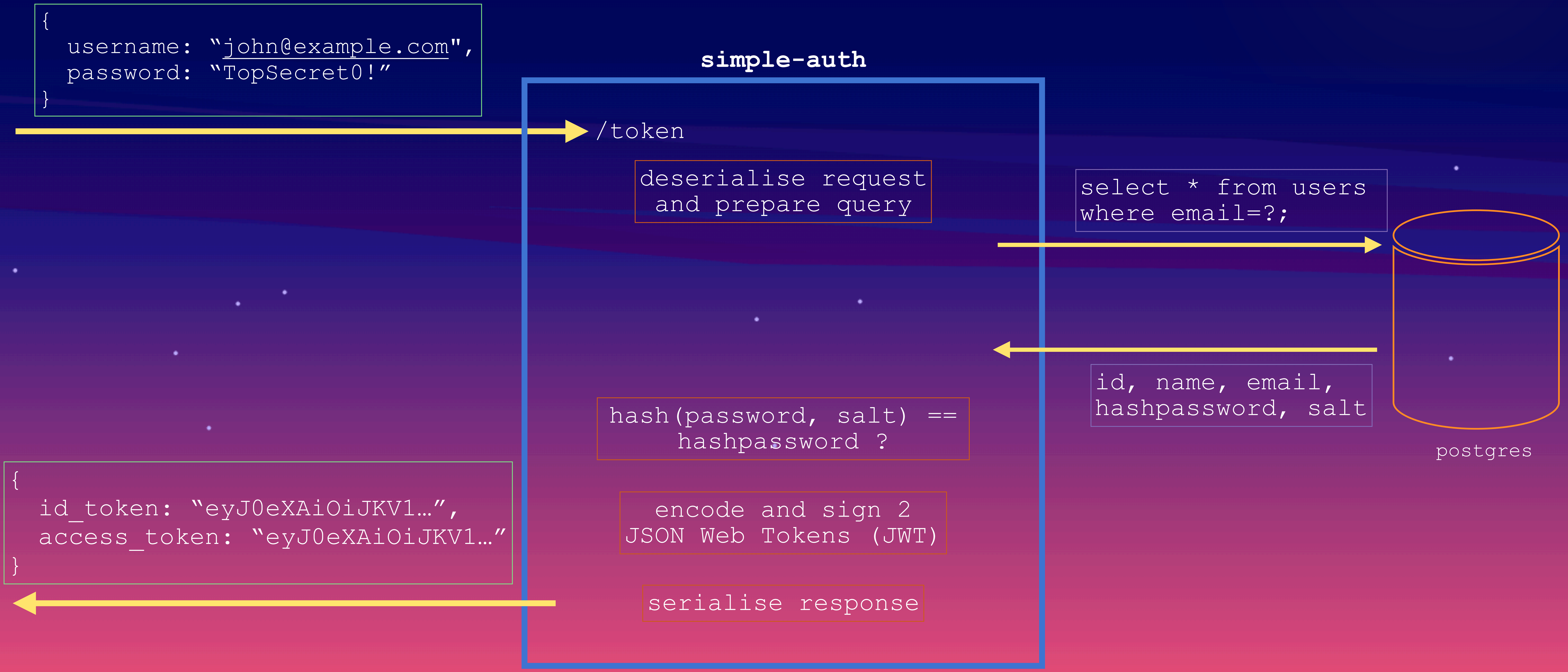
```
select * from users
where email=?;
```

```
id, name, email,
hashpassword, salt
```

Use case - “Simple Auth”


{
username: “john@example.com",
password: “TopSecret0!”
}
simple-auth
/token
deserialise request and prepare query
hash(password, salt) == hashpassword ?
encode and sign 2 JSON Web Tokens (JWT)
select * from users where email=?;
id, name, email, hashpassword, salt
postgres

Use case - “Simple Auth”


simple-auth
/token
deserialise request and prepare query
hash(password, salt) == hashpassword ?
encode and sign 2 JSON Web Tokens (JWT)
serialise response
postgres

```
{
  username: “john@example.com",
  password: “TopSecret0!”
}
```

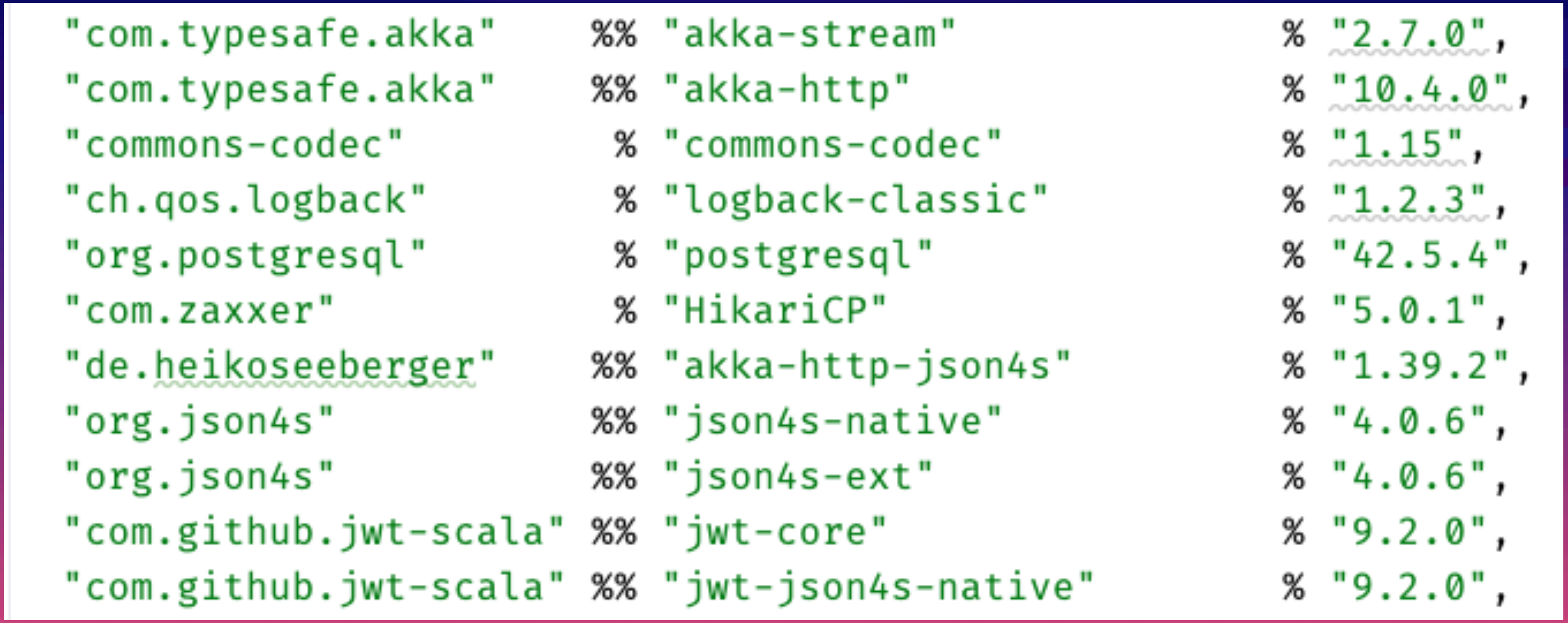
```
select * from users
where email=?;
```

```
id, name, email,
hashpassword, salt
```

```
{
  id_token: “eyJ0eXAiOiJKV1…”,
  access_token: “eyJ0eXAiOiJKV1…”
}
```

Akka-http implementation

build.sbt

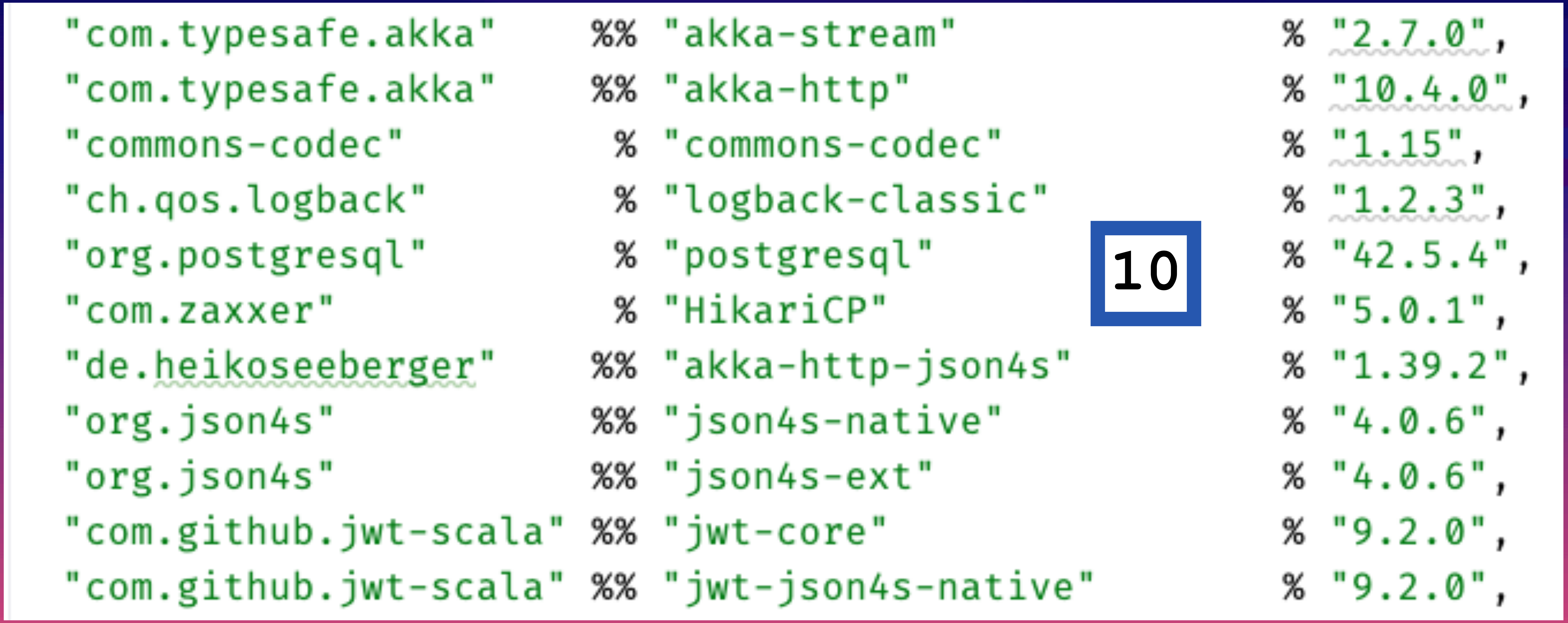


```
"com.typesafe.akka"     %% "akka-stream"              % "2.7.0",
"com.typesafe.akka"     %% "akka-http"                % "10.4.0",
"commons-codec"          % "commons-codec"            % "1.15",
"ch.qos.logback"         % "logback-classic"          % "1.2.3",
"org.postgresql"         % "postgresql"               % "42.5.4",
"com.zaxxer"             % "HikariCP"                 % "5.0.1",
"de.heikoseeberger"     %% "akka-http-json4s"         % "1.39.2",
"org.json4s"            %% "json4s-native"            % "4.0.6",
"org.json4s"            %% "json4s-ext"               % "4.0.6",
"com.github.jwt-scala" %% "jwt-core"                  % "9.2.0",
"com.github.jwt-scala" %% "jwt-json4s-native"         % "9.2.0",
```

https://github.com/willemvermeer/simple-auth

Akka-http implementation

build.sbt



```
"com.typesafe.akka"     %% "akka-stream"             % "2.7.0",
"com.typesafe.akka"     %% "akka-http"               % "10.4.0",
"commons-codec"          % "commons-codec"           % "1.15",
"ch.qos.logback"         % "logback-classic"         % "1.2.3",
"org.postgresql"         % "postgresql"              % "42.5.4",
"com.zaxxer"             % "HikariCP"                % "5.0.1",
"de.heikoseeberger"     %% "akka-http-json4s"        % "1.39.2",
"org.json4s"            %% "json4s-native"           % "4.0.6",
"org.json4s"            %% "json4s-ext"              % "4.0.6",
"com.github.jwt-scala"  %% "jwt-core"                % "9.2.0",
"com.github.jwt-scala"  %% "jwt-json4s-native"       % "9.2.0",
```

10

https://github.com/willemvermeer/simple-auth

Akka-http implementation

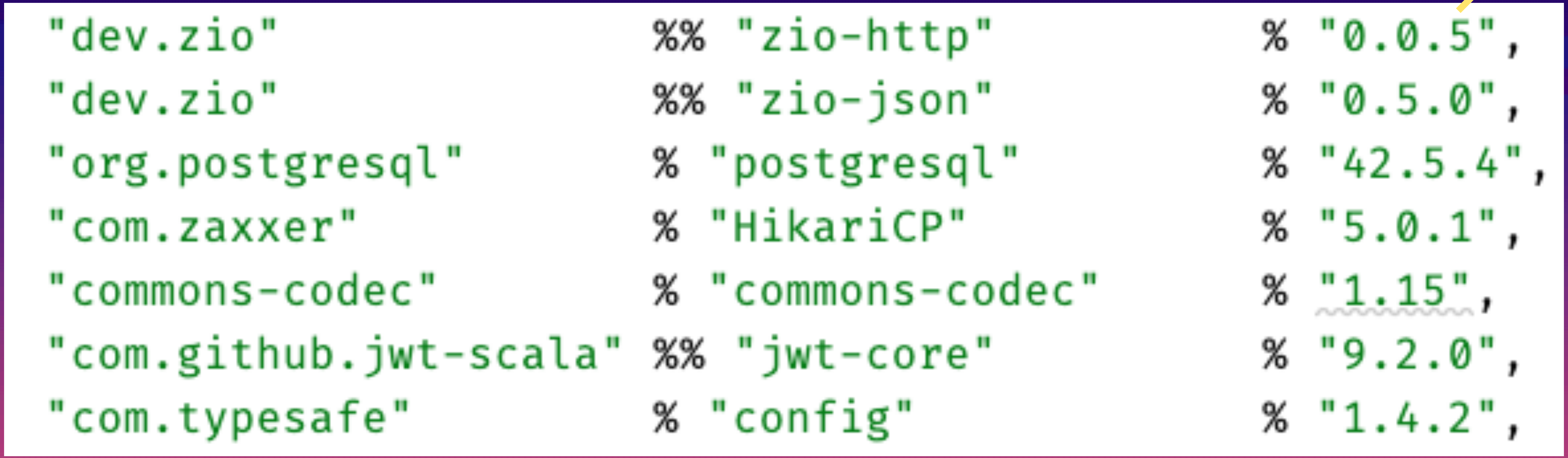
```
def route = path("token") {
  post {
    entity(as[TokenRequest]) { tokenRequest =>
      onComplete(for {
        userInfo <- Future.fromTry(DbQuery.userInfo(tokenRequest.username, dbPool))
        passwordOk <- Future.fromTry(
                        KeyTools
                          .verifyHashMatch(tokenRequest.password, userInfo.salt, userInfo.hashpassword)
                      )
        _          <- if (passwordOk) Future.successful(()) else Future.failed(new RuntimeException("Incorrect password"))
        tokenPair <- Future.fromTry(tokenCreator.createTokenPair(userInfo))
        response  <- Future.successful {
                        TokenResponse(id_token = tokenPair.id.rawToken, access_token = tokenPair.access.rawToken)
                      }
      } yield response) {
        case Success(value) =>
          complete(StatusCodes.OK, value)
        case Failure(failure) =>
          failure.printStackTrace()
          complete(StatusCodes.InternalServerError, failure.getMessage)
      }
    }
```

https://github.com/willemvermeer/simple-auth

ZIO implementation

build.sbt

zio v2.0.10



```
"dev.zio"                  %% "zio-http"          % "0.0.5",
"dev.zio"                  %% "zio-json"          % "0.5.0",
"org.postgresql"           % "postgresql"         % "42.5.4",
"com.zaxxer"               % "HikariCP"           % "5.0.1",
"commons-codec"            % "commons-codec"      % "1.15",
"com.github.jwt-scala"     %% "jwt-core"          % "9.2.0",
"com.typesafe"             % "config"             % "1.4.2",
```

https://github.com/willemvermeer/simple-auth

ZIO implementation

```scala
val app: Http[SimpleAuthConfig with HikariConnectionPool with TokenCreator, Nothing, Request, Response] =
  Http.collectZIO[Request] {
    case req @ Method.POST -> !! / "token" =>
      (for {
        tokenReq <- ZIO.absolve(req.body.asString(UTF_8).map(_.fromJson[TokenRequest]))
        pool     <- ZIO.service[HikariConnectionPool]
        userInfo <- ZIO.fromTry(DbQuery.userInfo(tokenReq.username, pool))
        pwOK     <- ZIO
                      .fromTry(
                        KeyTools
                          .verifyHashMatch(tokenReq.password, userInfo.salt, userInfo.hashpassword)
                      )
        _            <- ZIO.cond(pwOK, (), "Incorrect password")
        tokenCreator <- ZIO.service[TokenCreator]
        tokenPair    <- ZIO.fromTry(tokenCreator.createTokenPair(userInfo))
        response     = TokenResponse(tokenPair.id.rawToken, tokenPair.access.rawToken)
        resp         <- ZIO.succeed(Response.json(response.toJson))
      } yield resp).catchAll(ex => ZIO.succeed(Response.text(s"error $ex")))
  }
```

https://github.com/willemvermeer/simple-auth

Rust actix-web implementation

Cargo.toml

```toml
[package]
name = "actix-web-simple-auth"
version = "0.1.0"
edition = "2021"

[profile.release]
opt-level = 3

[dependencies]
socket2="0.4.9"
actix-web="4"
config = "0.13.1"
deadpool-postgres = {
 version = "0.10.2", features = ["serde"]
}
dotenv = "0.15.0"
tokio-pg-mapper = "0.2.0"
tokio-pg-mapper-derive = "0.2.0"
tokio-postgres = "0.7.6"
serde = { version = "1.0.158", features = ["derive"] }
serde_json = "1.0"
serde_with = "1.8"
uuid = { version = "1.3.1", features = ["v4"] }
derive_more = "0.99.17"
chrono = "0.4.23"
jsonwebtoken = "8.2.0"
hmac = "0.12"
sha2 = "0.10"
sha256 = "1.1"
base64 = "0.21"
hex = "0.4"

[build-dependencies]
platforms = "2.0.0"
```

https://github.com/willemvermeer/simple-auth

Rust actix-web implementation

Cargo.toml

```
[package]
name = "actix-web-simple-auth"
version = "0.1.0"
edition = "2021"

[profile.release]
opt-level = 3

[dependencies]
socket2="0.4.9"
actix-web="4"
config = "0.13.1"
deadpool-postgres = {
 version = "0.10.2", features = ["serde"]
}
dotenv = "0.15.0"
tokio-pg-mapper = "0.2.0"
tokio-pg-mapper-derive = "0.2.0"
tokio-postgres = "0.7.6"
serde = { version = "1.0.158", features = ["derive"] }
serde_json = "1.0"
serde_with = "1.8"
uuid = { version = "1.3.1", features = ["v4"] }
derive_more = "0.99.17"
chrono = "0.4.23"
jsonwebtoken = "8.2.0"
hmac = "0.12"
sha2 = "0.10"
sha256 = "1.1"
base64 = "0.21"
hex = "0.4"

[build-dependencies]
platforms = "2.0.0"
```

https://github.com/willemvermeer/simple-auth

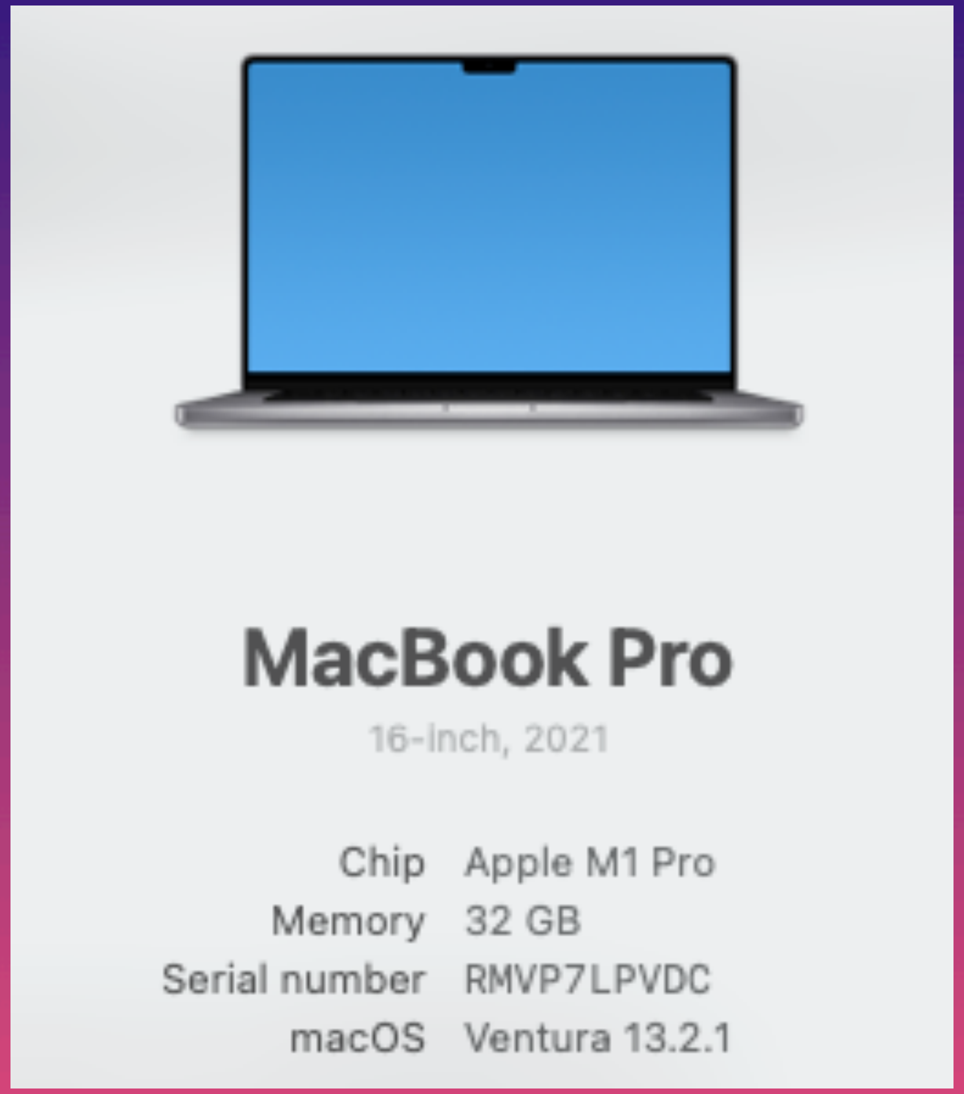
Rust actix-web implementation

```rust
pub async fn logon_user(
    logon_req: web::Json<LogonRequest>,
    state: web::Data<(Pool, EncodingKey)>,
) -> Result<HttpResponse, Error> {
    let user_info: LogonRequest = logon_req.into_inner();
    let (db_pool, encoding_key) = state.get_ref();
    let client: Client = db_pool.get().await.unwrap();
    let user_from_db = db::get_user(&client, &user_info).await?;

    let encoded = hash_password(&user_info.password, &user_from_db.salt);

    if encoded != user_from_db.hashpassword.to_string() {
        Ok(HttpResponse::InternalServerError().body("Incorrect password"))
    } else {
        let header = Header::new(Algorithm::RS256);
        let common_claims = JwtClaim::empty()
            .with_audience("simple-auth.example.com".to_string())
            .with_issuer("https://example.com".to_string())
            .issued_now()
            .expires_in(Duration::minutes(60).num_seconds().unsigned_abs());

        let id_claims = user_from_db.to_id_claims();
        let access_claims = AccessClaims { session_id: Uuid::new_v4().to_string(), };
        let token_pair = TokenPair::create(&encoding_key, &header, common_claims,
                                           id_claims, access_claims).unwrap();

        let response = TokenResponse {
            id_token: token_pair.id.raw,
            access_token: token_pair.access.raw,
        };

        Ok(HttpResponse::Ok().json(response))
    }
}
```

https://github.com/willemvermeer/simple-auth

Where to run?

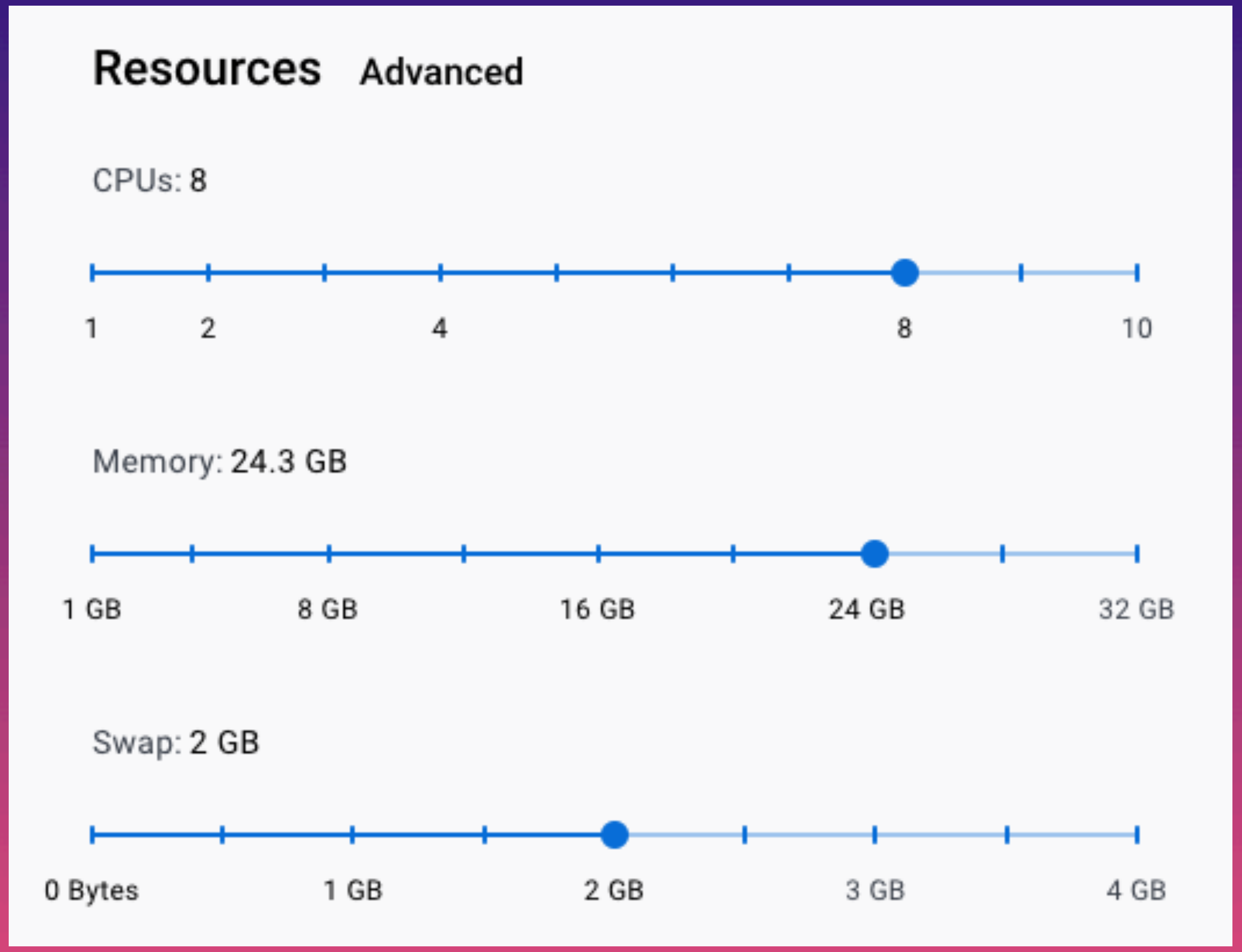
- Where to run?
  - Locally, locally in docker, in the cloud using fly.io



MacBook Pro

16-inch, 2021

Chip	Apple M1 Pro
Memory	32 GB
Serial number	RMVP7LPVDC
macOS	Ventura 13.2.1



Resources Advanced

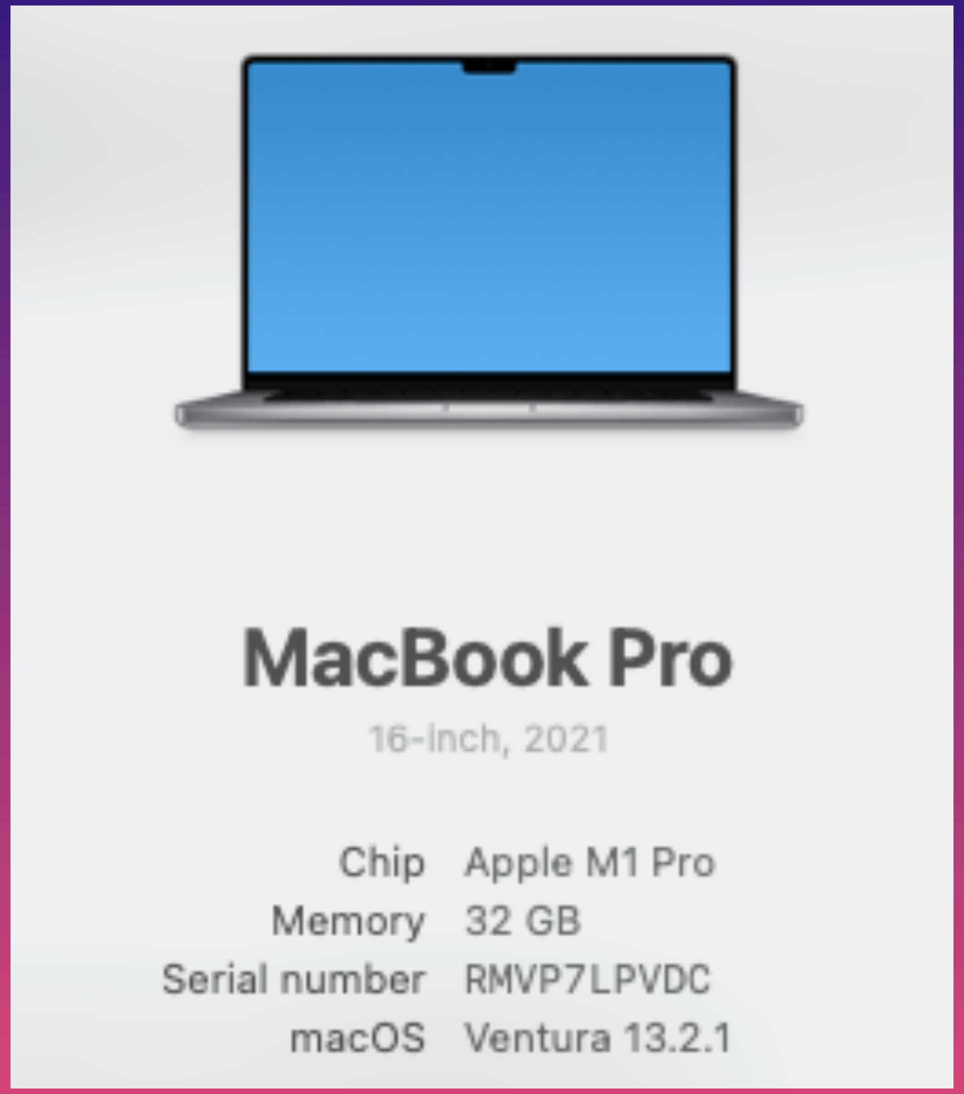
CPUs: 8

Memory: 24.3 GB

Swap: 2 GB

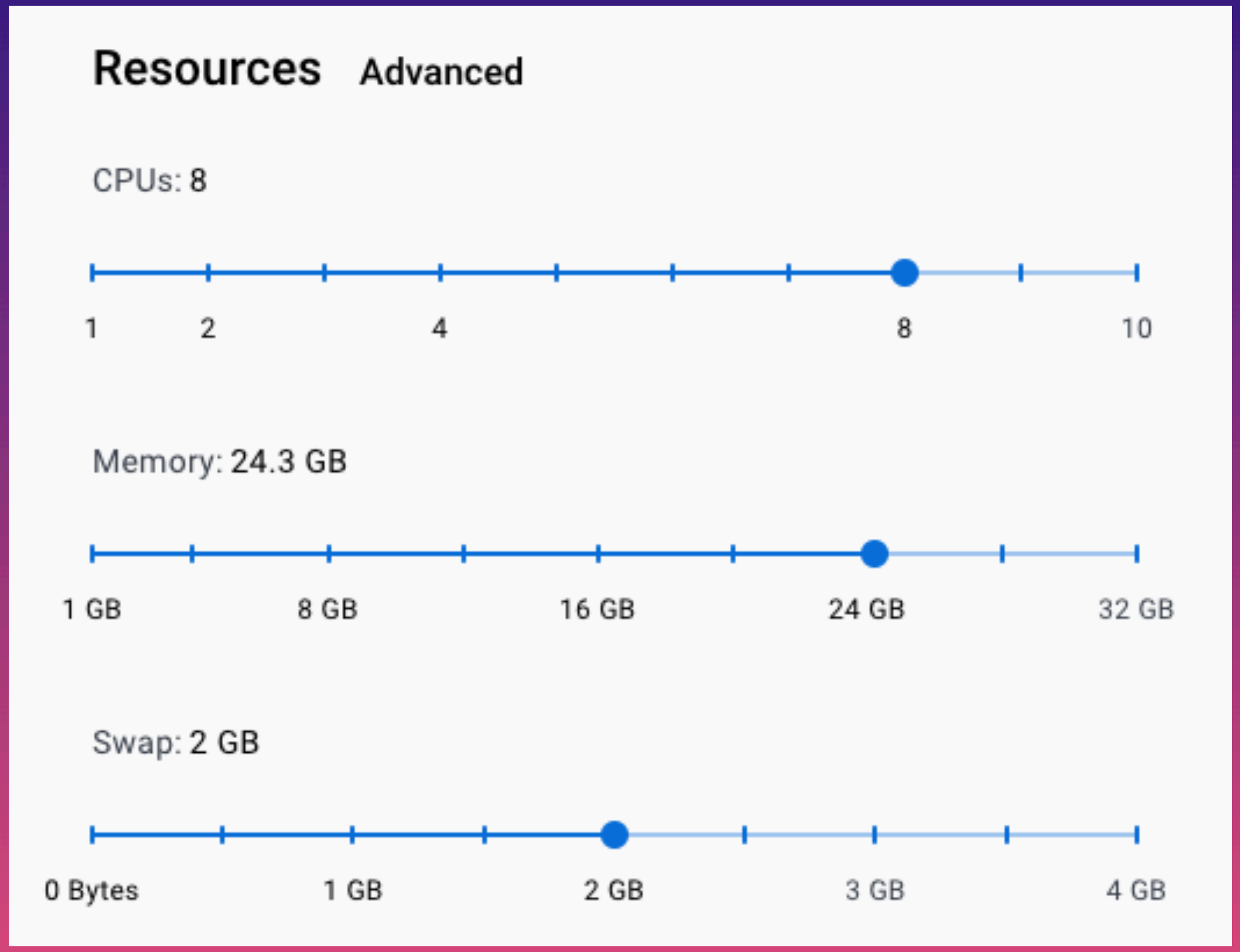
Where to run?

- Where to run?
  - Locally, locally in docker, in the cloud using fly.io



MacBook Pro
16-inch, 2021

Chip	Apple M1 Pro
Memory	32 GB
Serial number	RMVP7LPVDC
macOS	Ventura 13.2.1

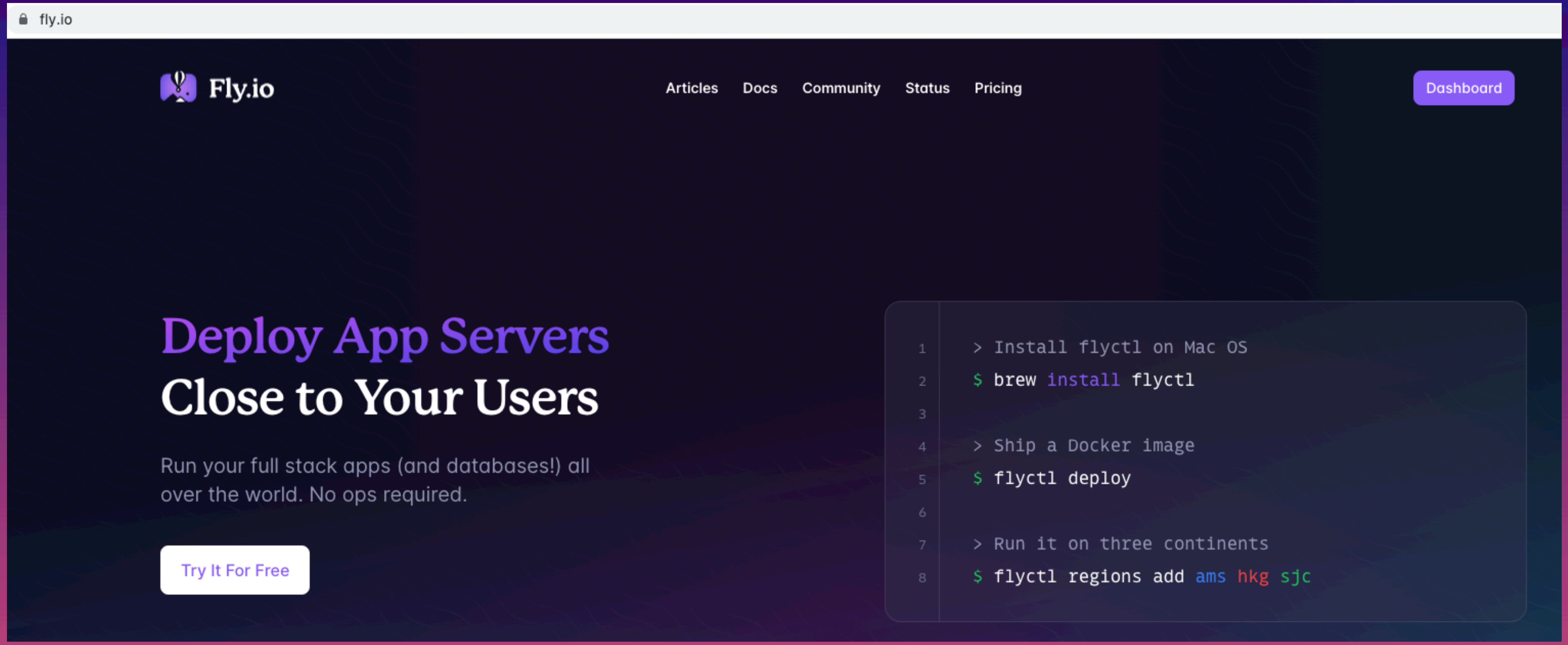


Resources Advanced

CPUs: 8

Memory: 24.3 GB

Swap: 2 GB



Deploy App Servers
Close to Your Users

Run your full stack apps (and databases!) all over the world. No ops required.

```
> Install flyctl on Mac OS
$ brew install flyctl

> Ship a Docker image
$ flyctl deploy

> Run it on three continents
$ flyctl regions add ams hkg sjc
```

```
'shared-cpu-1x'
       CPU Cores: 1
          Memory: 256 MB
```

```
'performance-8x'
       CPU Cores: 8
          Memory: 16384 MB
```

What/how to measure?

- Mainly: requests/per second
- Secondary: memory consumption

What/how to measure?

- Mainly: requests/per second
- Secondary: memory consumption
- Use ‘wrk’ tool (`brew install wrk`) to run the test

```
> more post-token.lua
wrk.method = "POST"
wrk.body   = '{"username":"john@example.com","password":"TopSecret0!"}'
wrk.headers["Content-Type"] = "application/json"

> wrk -s post-token.lua -d15 -t24 -c24 http://localhost:8781/token
Running 15s test @ http://localhost:8781/token
  24 threads and 24 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    20.93ms    6.61ms 143.98ms   73.37%
    Req/Sec    47.92      7.78    70.00     51.09%
  17309 requests in 15.10s, 18.44MB read
Requests/sec:   1146.13
Transfer/sec:      1.22MB
```

What/how to measure?

- Mainly: requests/per second
- Secondary: memory consumption
- Use ‘wrk’ tool (`brew install wrk`) to run the test

```
> more post-token.lua
wrk.method = "POST"
wrk.body   = '{"username":"john@example.com","password":"TopSecret0!"}'
wrk.headers["Content-Type"] = "application/json"

> wrk -s post-token.lua -d15 -t24 -c24 http://localhost:8781/token
Running 15s test @ http://localhost:8781/token
  24 threads and 24 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    20.93ms    6.61ms 143.98ms   73.37%
    Req/Sec    47.92      7.78    70.00     51.09%
  17309 requests in 15.10s, 18.44MB read
Requests/sec:   1146.13
Transfer/sec:      1.22MB
```

What/how to measure?

- Mainly: requests/per second
- Secondary: memory consumption
- Use ‘wrk’ tool (`brew install wrk`) to run the test

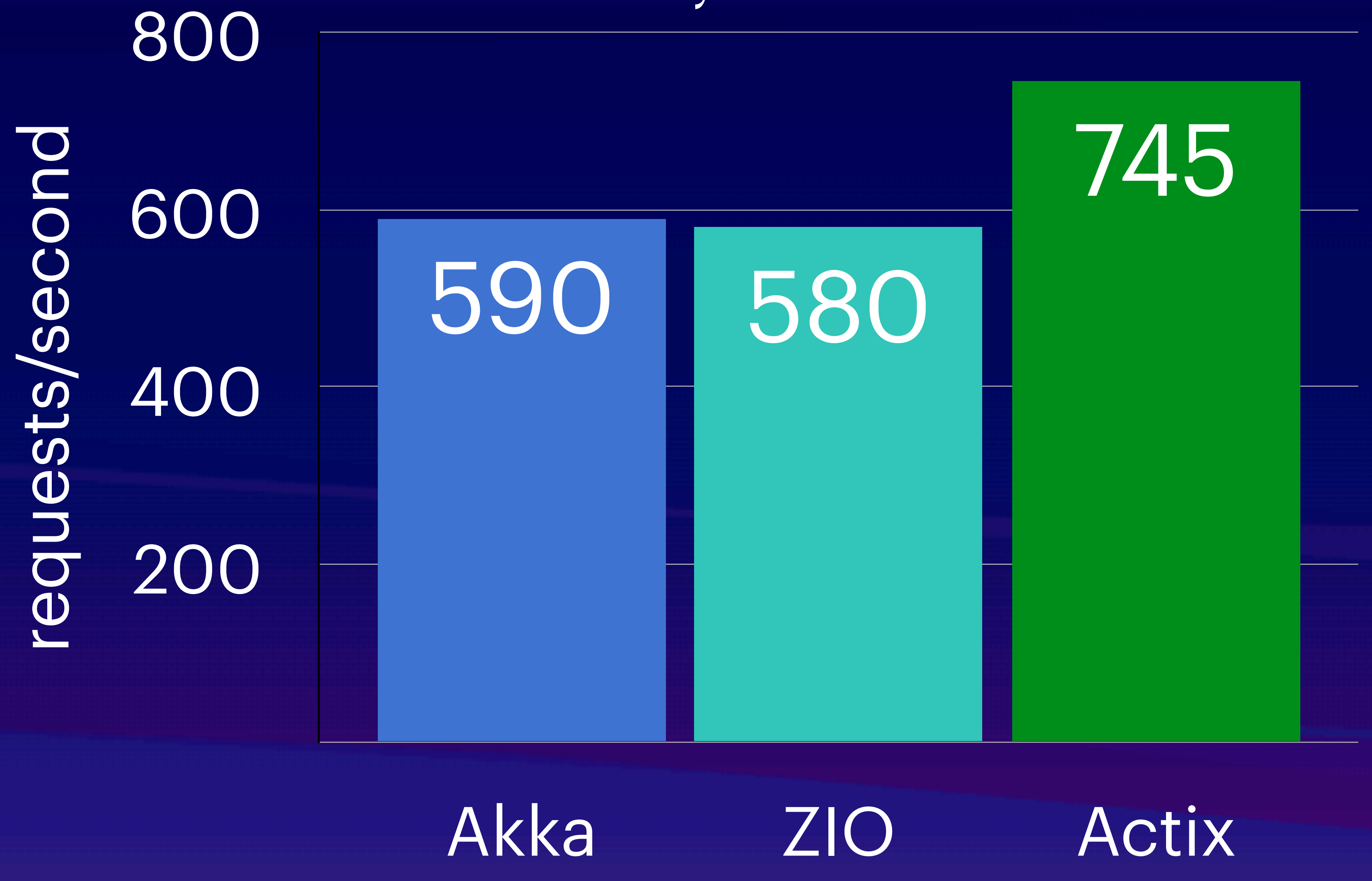
```
> more post-token.lua
wrk.method = "POST"
wrk.body   = '{"username":"john@example.com","password"
wrk.headers["Content-Type"] = "application/json"

> wrk -s post-token.lua -d15 -t24 -c24 http://localhost
Running 15s test @ http://localhost:8781/token
  24 threads and 24 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    20.93ms    6.61ms 143.98ms   73.37%
    Req/Sec    47.92      7.78     70.00     51.09%
  17309 requests in 15.10s, 18.44MB read
Requests/sec:   1146.13
Transfer/sec:      1.22MB
```

Test scenario (Techempower-style)

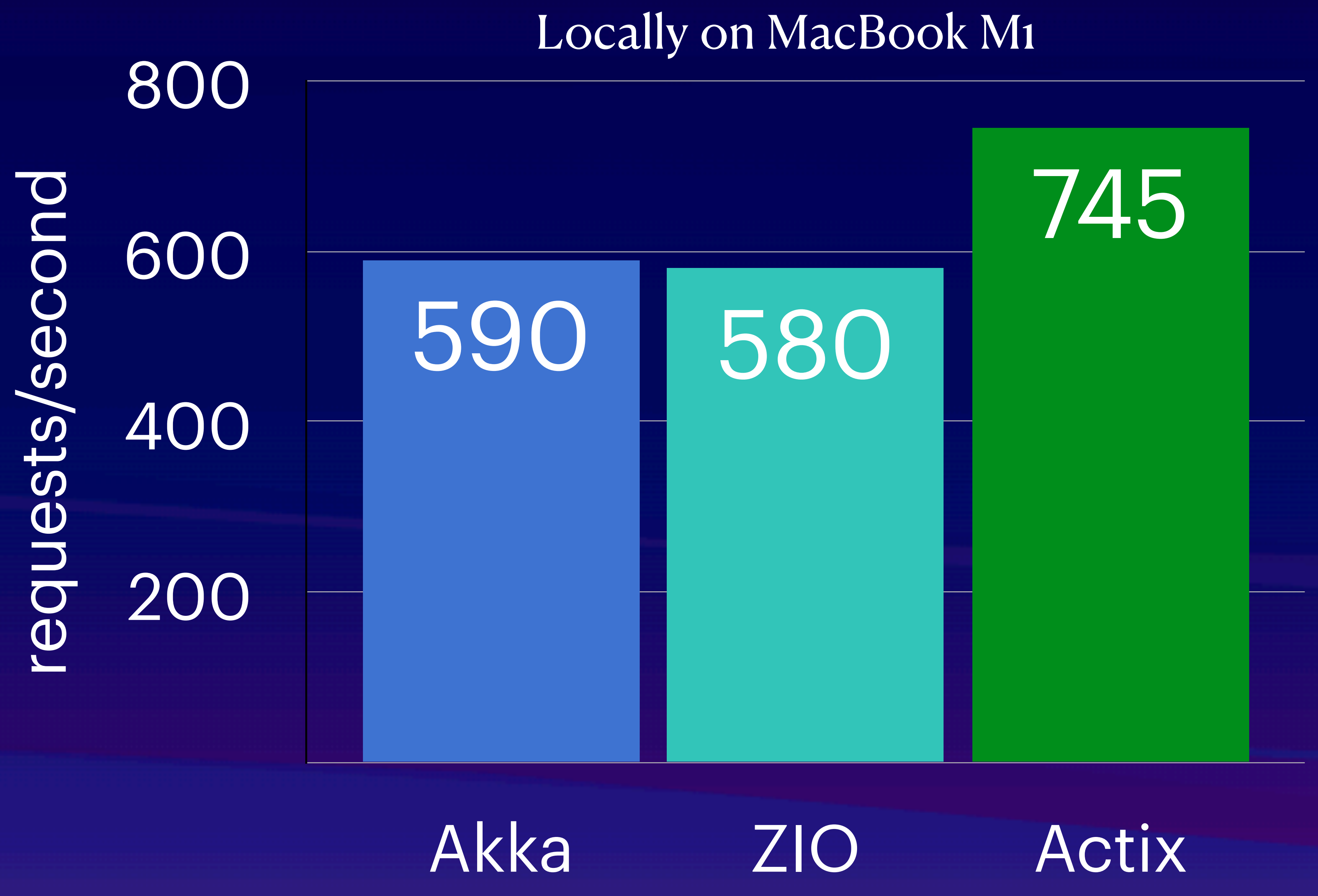
- start the app
- POST 1 or 2 req’s to check it’s up
- run `wrk` once for 15s to warmup
- run `wrk` again for 15s and measure

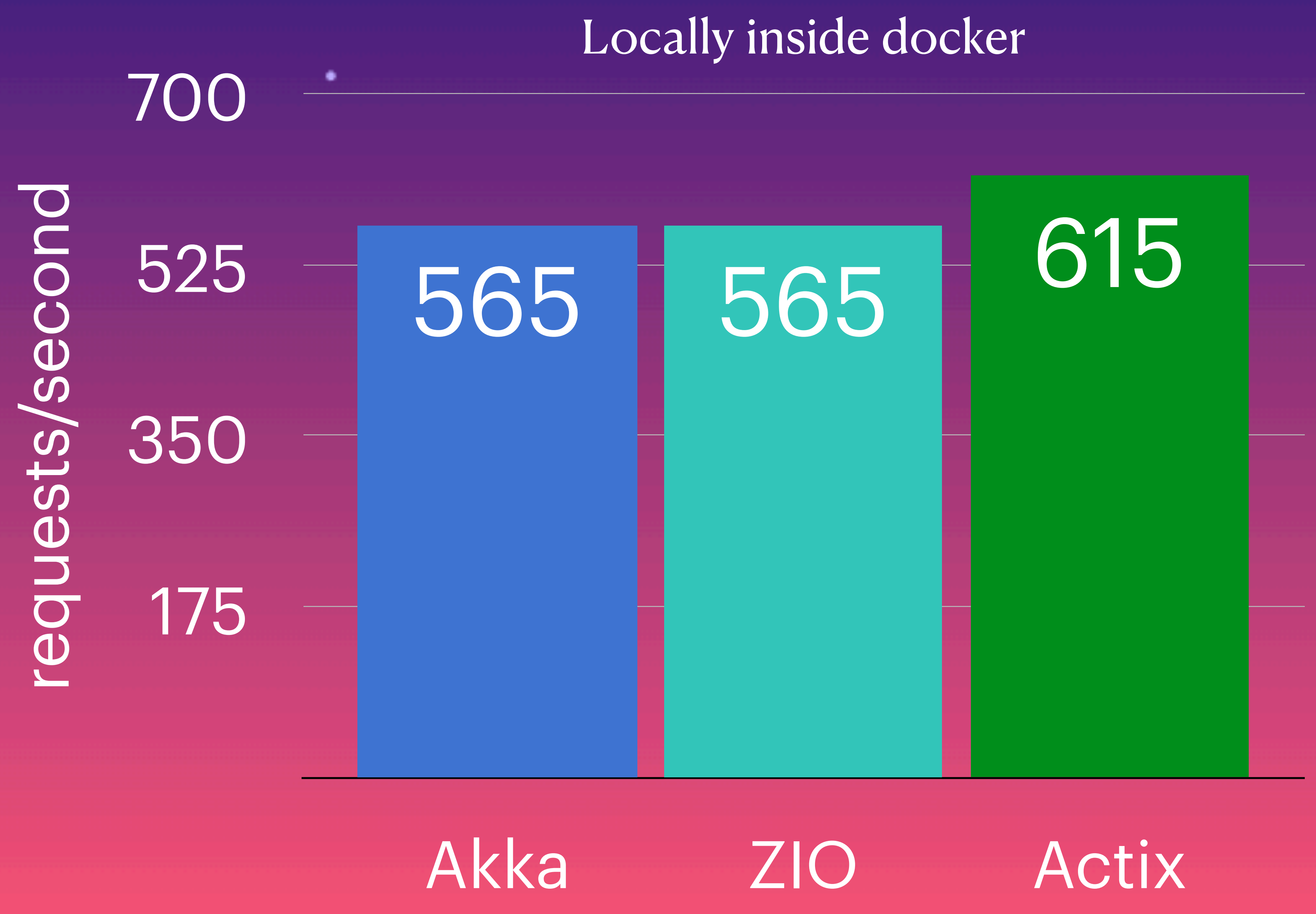
Results

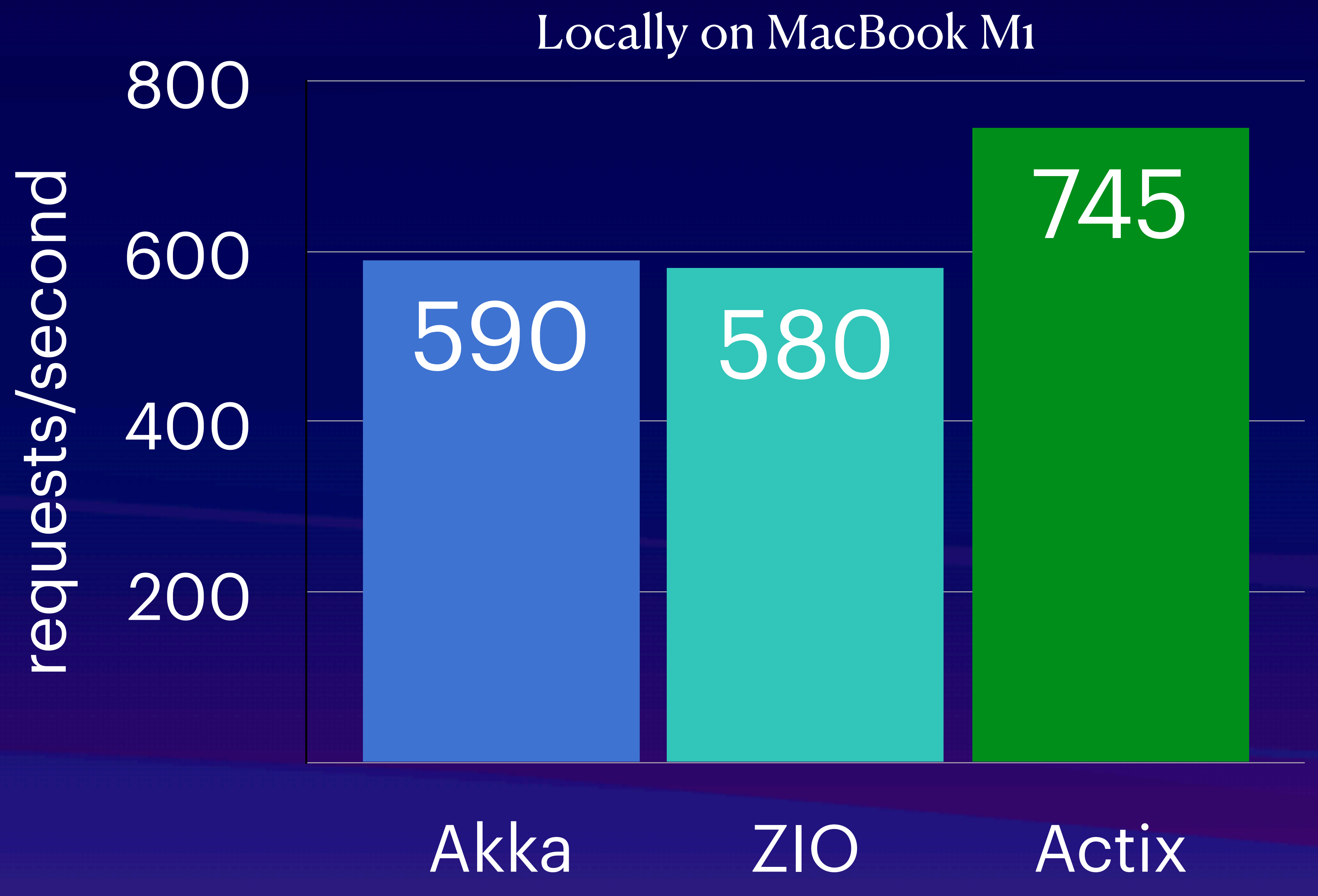
Locally on MacBook M1

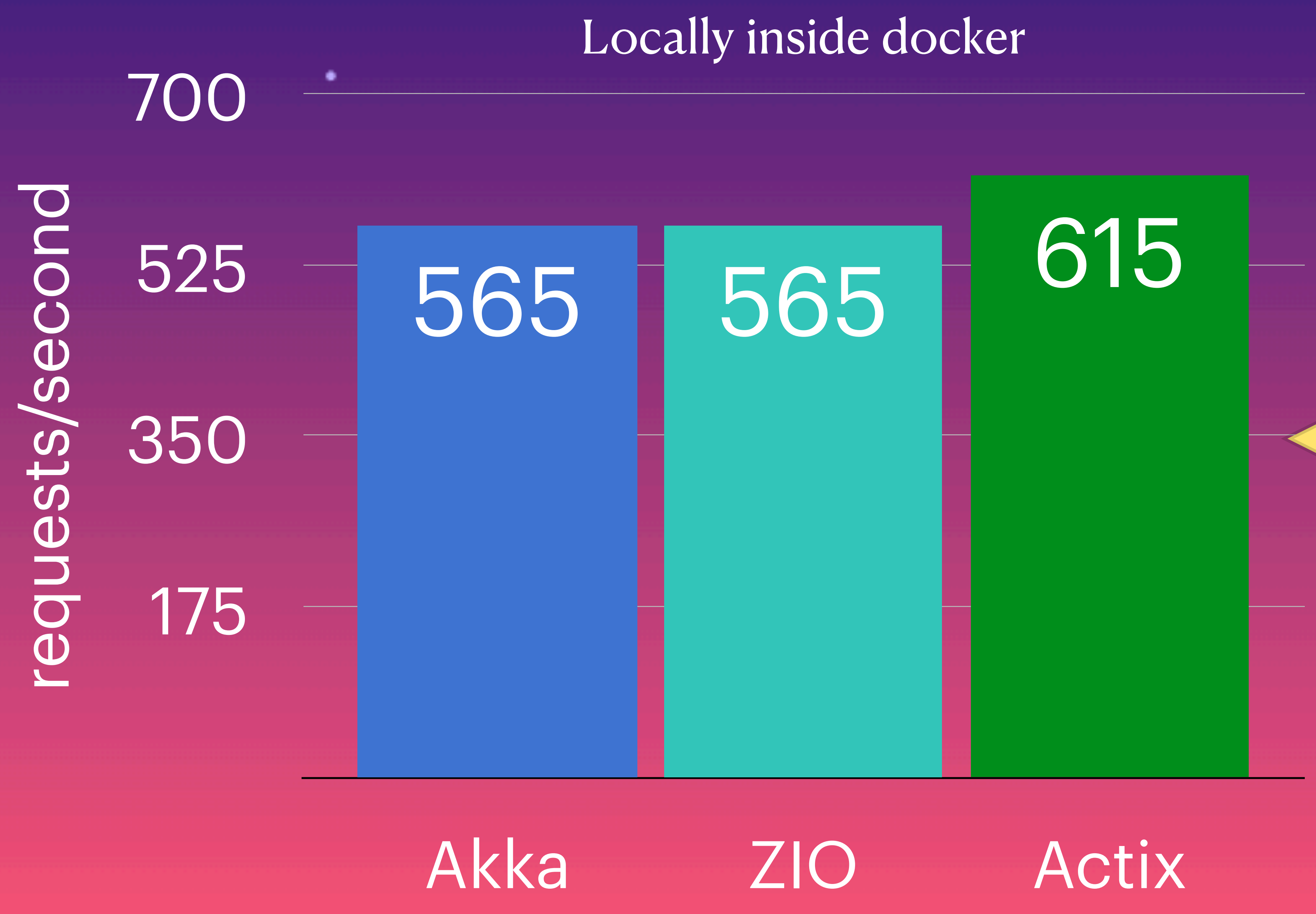


Locally inside docker

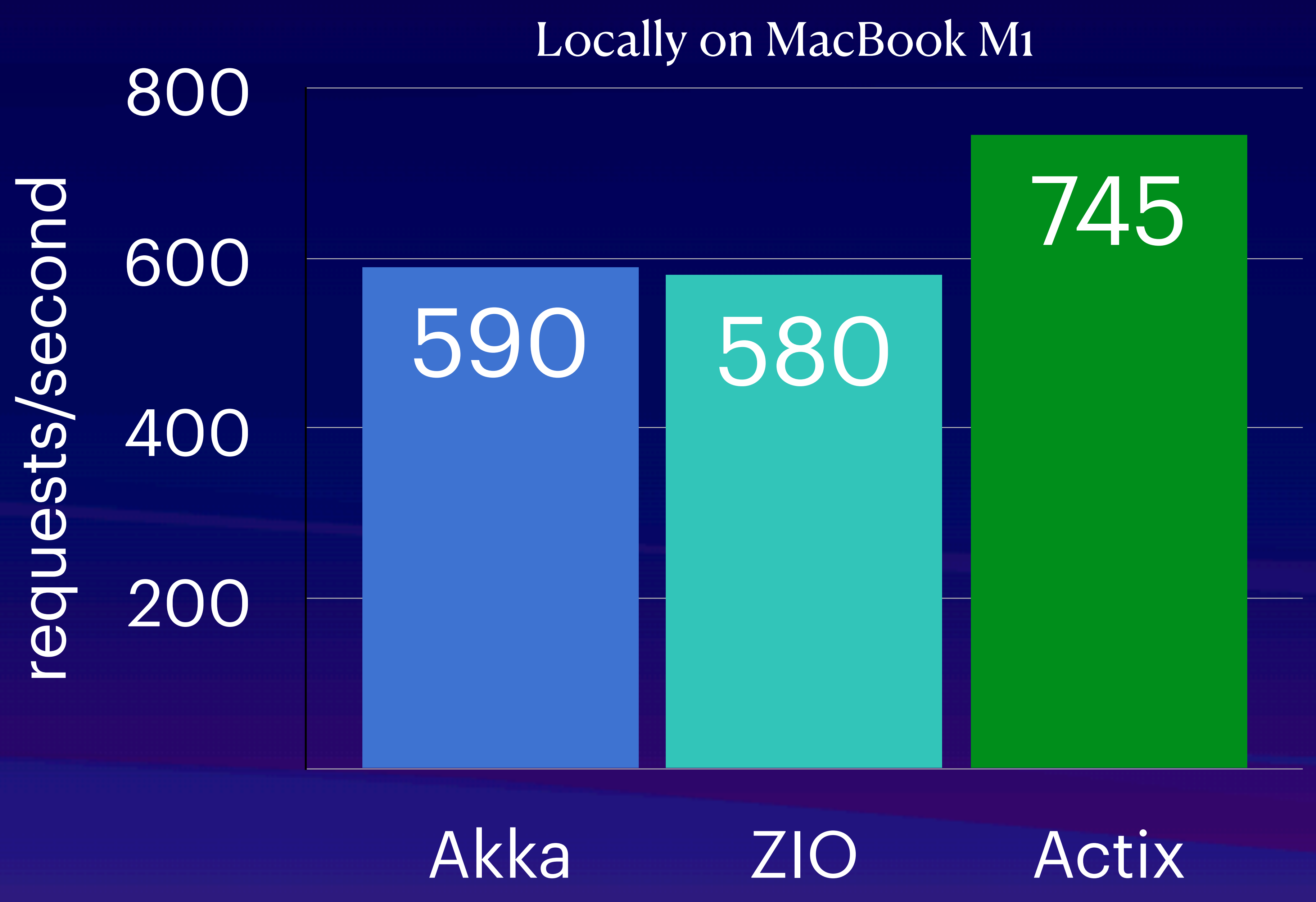

Locally on MacBook M1
requests/second
800
600
400
200
590
580
745
Akka
ZIO
Actix

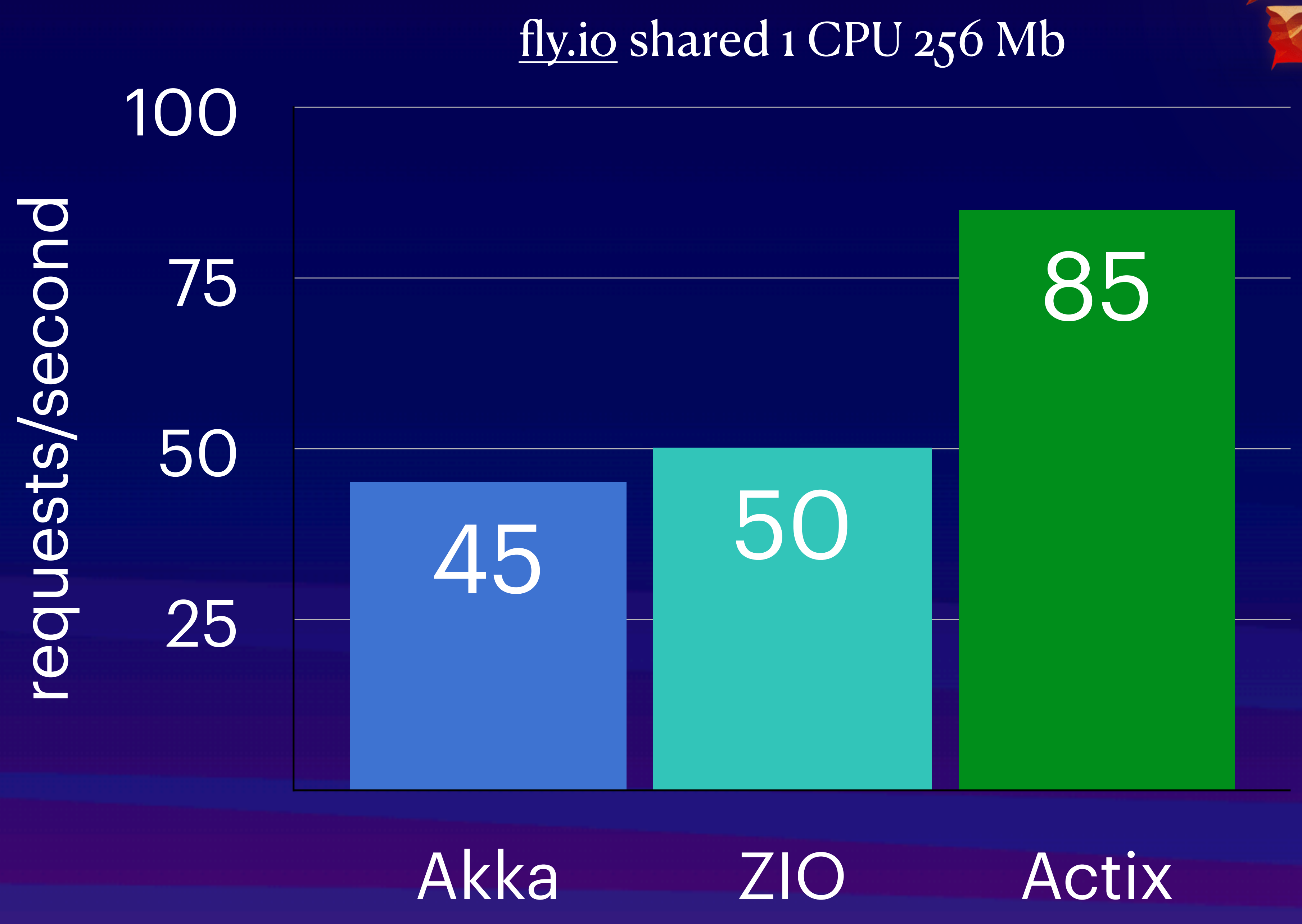

Locally inside docker
requests/second
700
525
350
175
565
565
615
Akka
ZIO
Actix

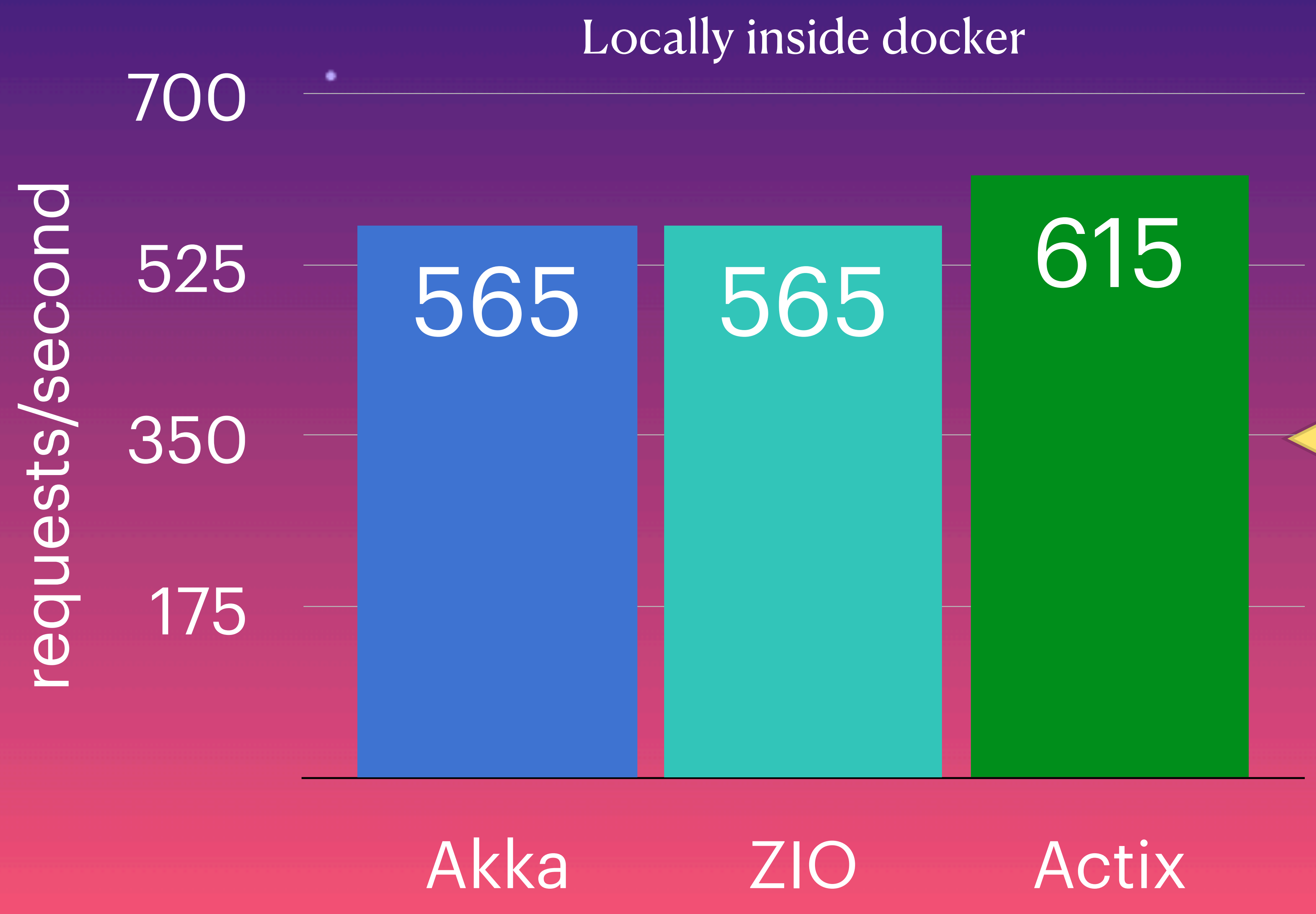




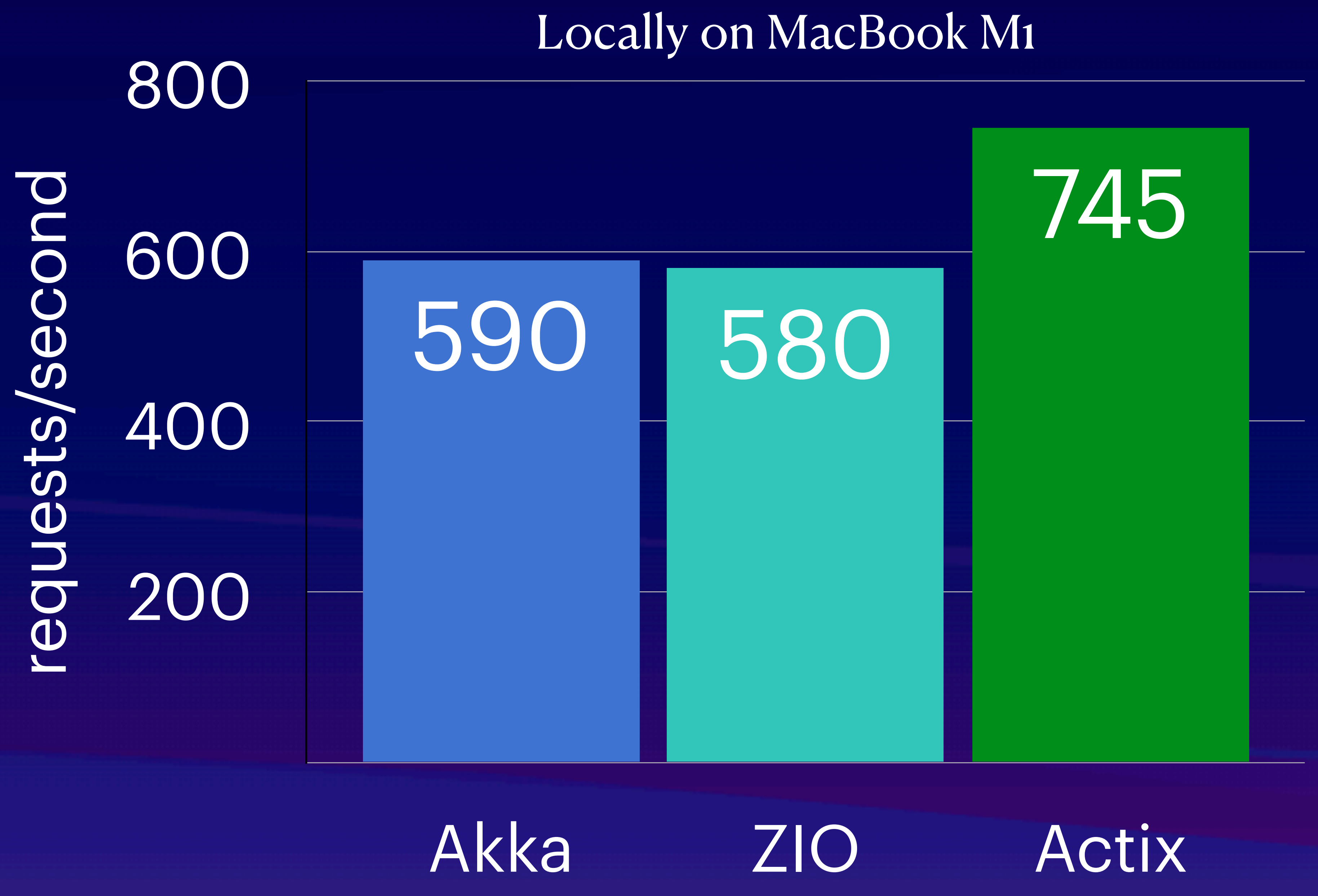
Memory usage:
Akka up to 260Mb
Zio up to 340 Mb
Rust up to 4 Mb (!)

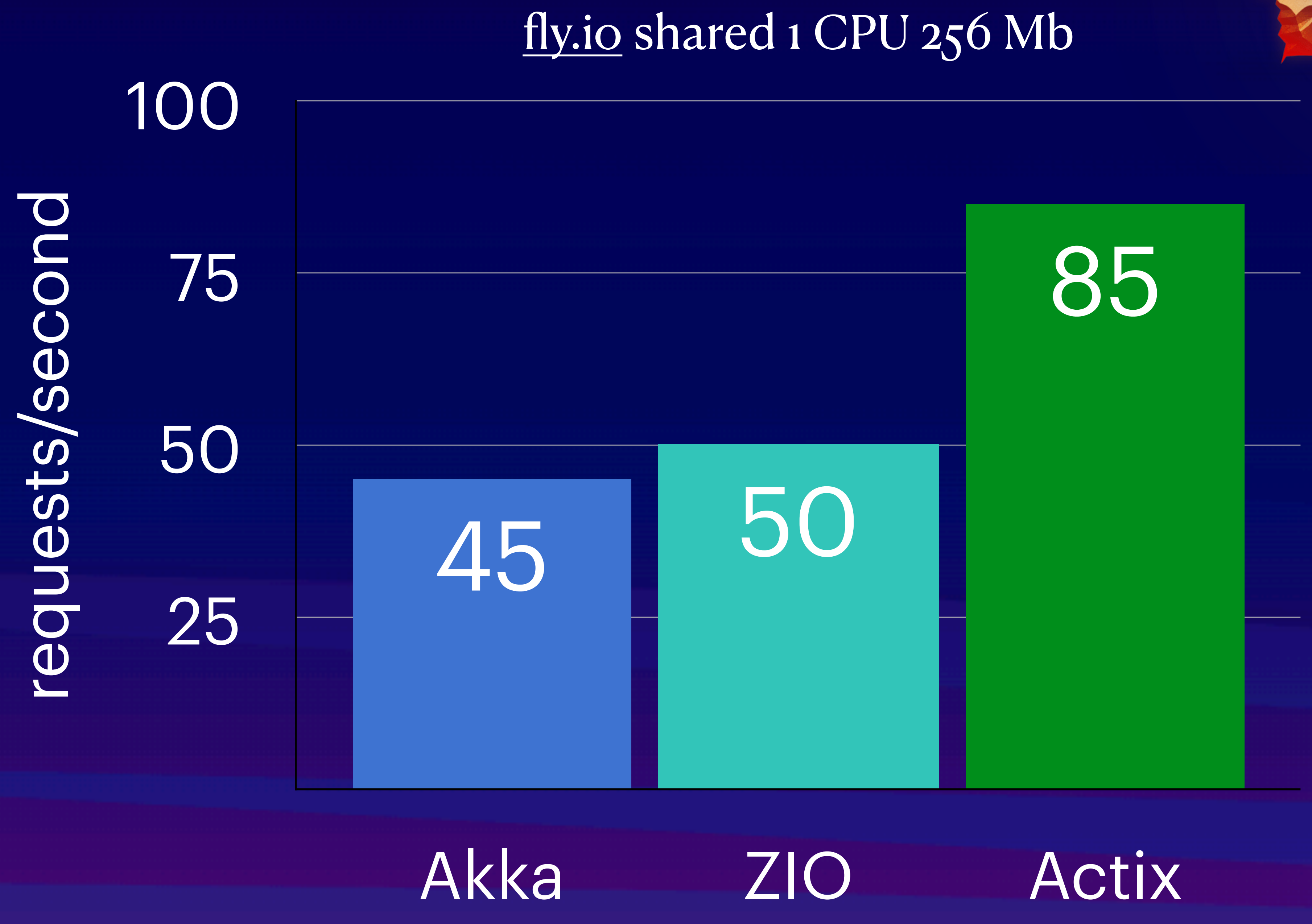


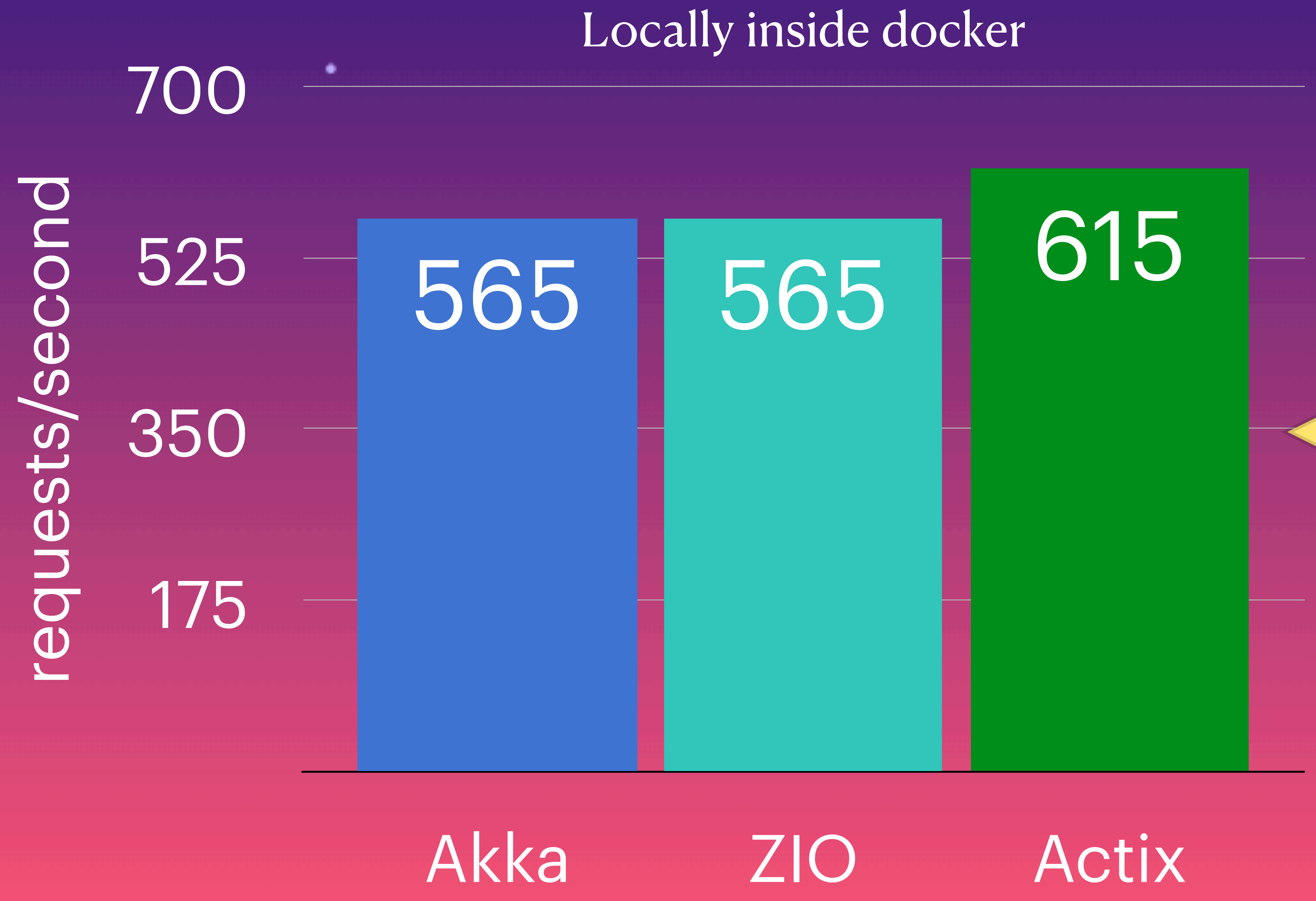




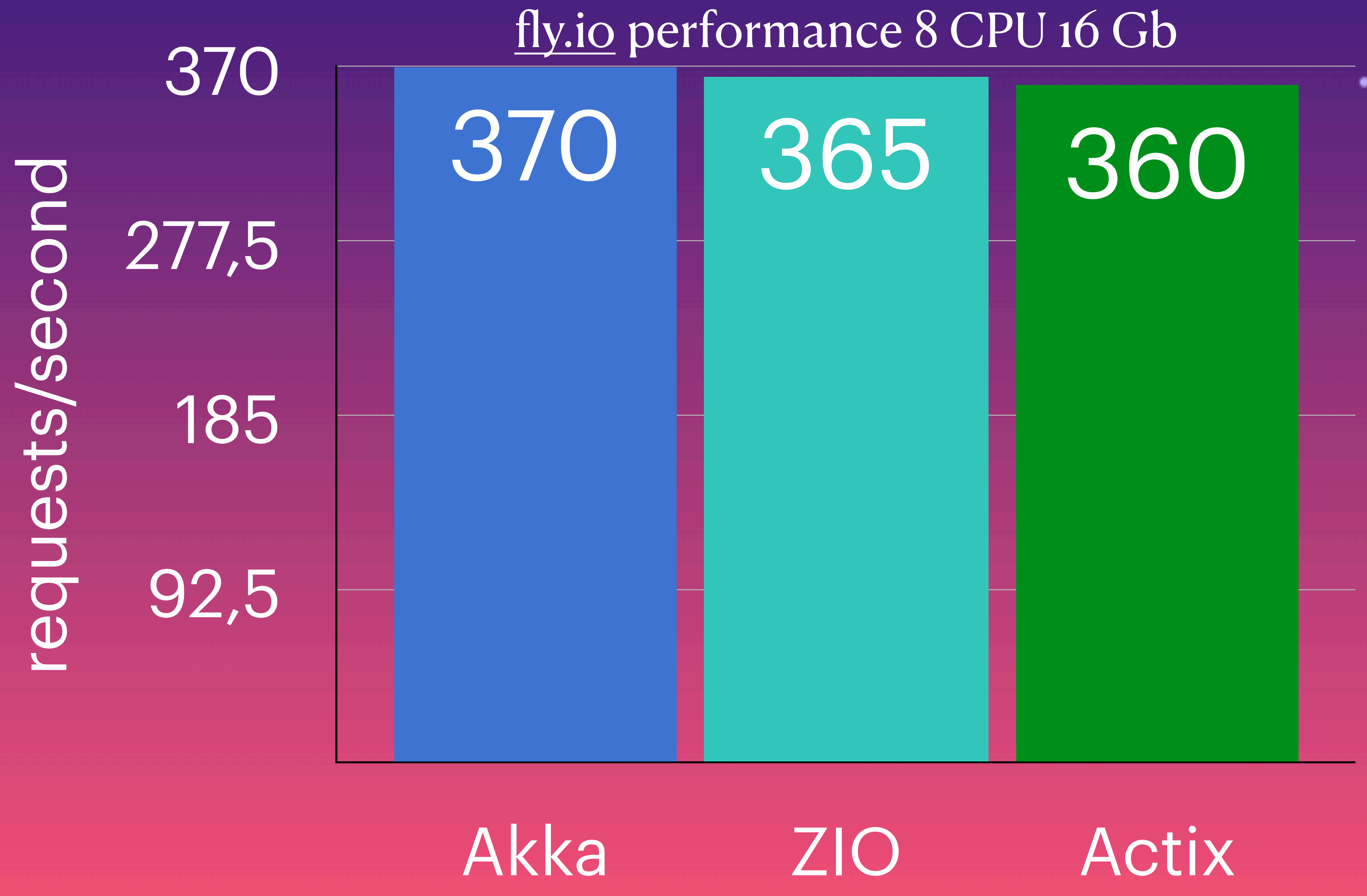
Memory usage:
Akka up to 260Mb
Zio up to 340 Mb
Rust up to 4 Mb (!)







Memory usage:
Akka up to 260Mb
Zio up to 340 Mb
Rust up to 4 Mb (!)



Conclusions

- ZIO holds up well compared to Akka-http

Conclusions

- ZIO holds up well compared to Akka-http
- Rust performs best, especially in resource constrained environments

Conclusions

- ZIO holds up well compared to Akka-http
- Rust performs best, especially in resource constrained environments
- .. draw your own conclusions, or, even better, repeat tests for your own use case 😀

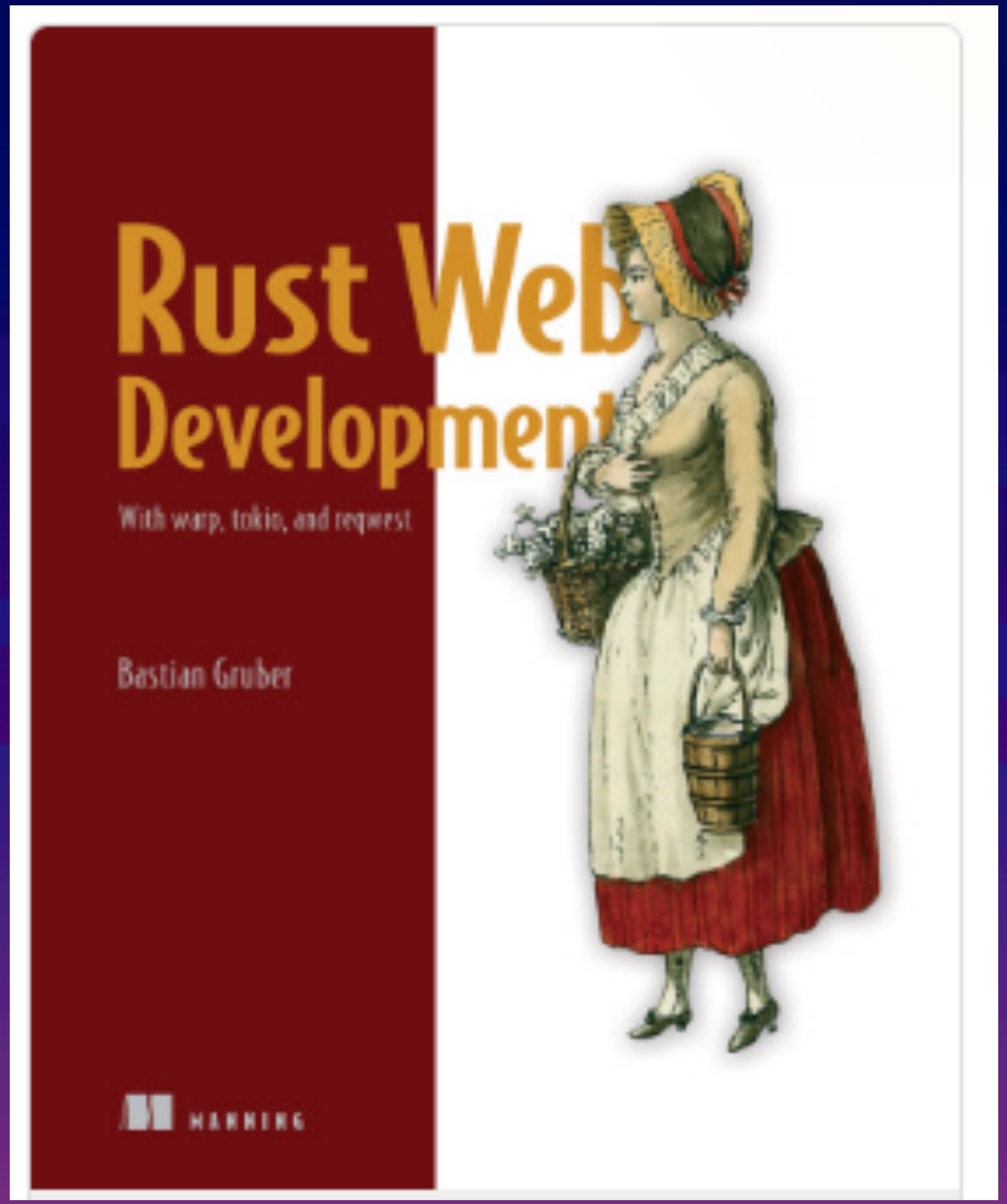
Acknowledgements

Acknowledgements

- All developers writing great frameworks such as Akka, ZIO and Rust/actix!

Acknowledgements

- All developers writing great frameworks such as Akka, ZIO and Rust/actix!


Rust Web Development

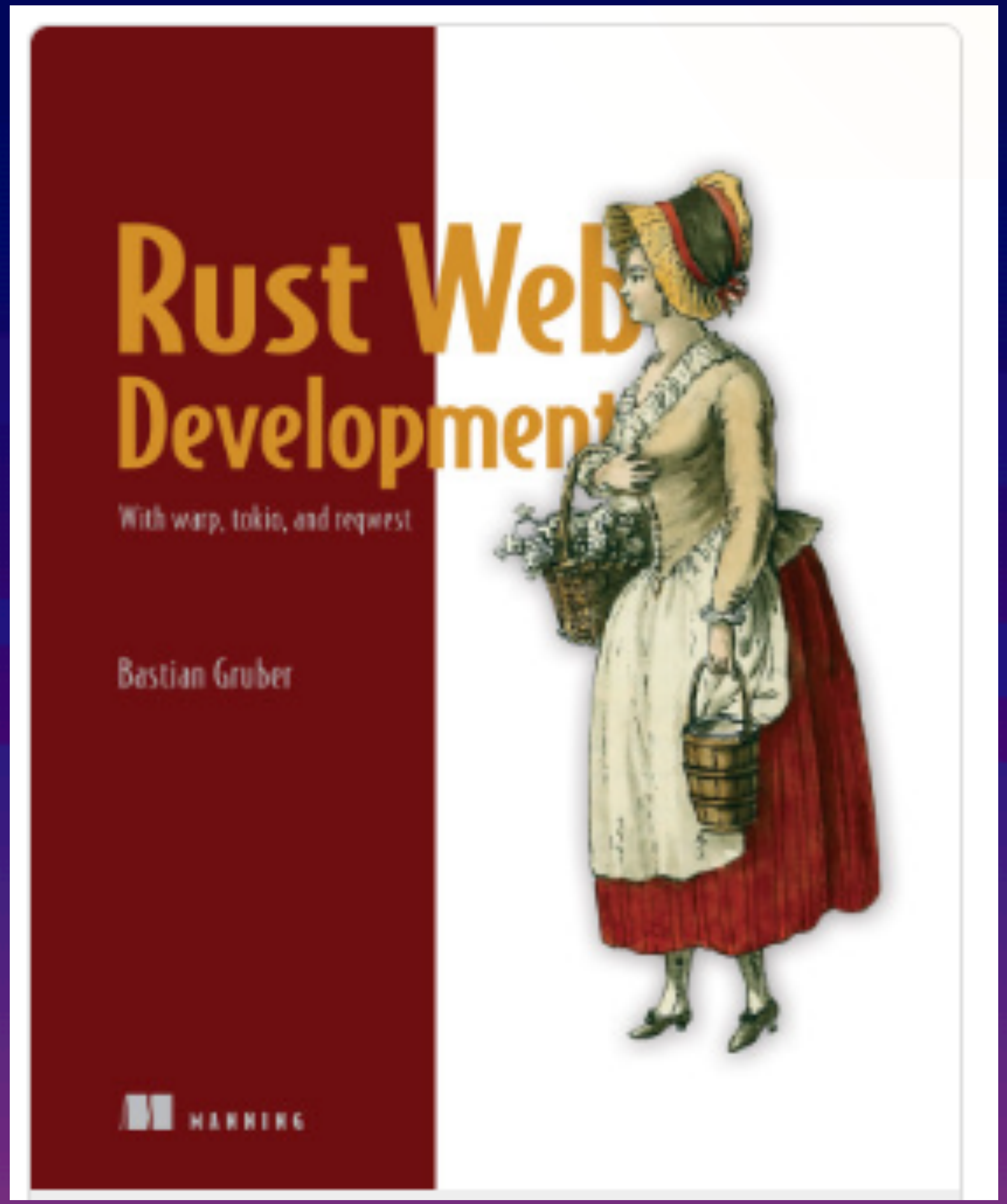
With warp, tokio, and request

Bastian Gruber

MANNING

Acknowledgements

- All developers writing great frameworks such as Akka, ZIO and Rust/actix!
- Oliver Tupran @olivertupran



Acknowledgements

- All developers writing great frameworks such as Akka, ZIO and Rust/actix!
- Oliver Tupran @olivertupran

- https://github.com/willemvermeer/simple-auth

@willemvermeer