Take your monolith to microservices with Fastify

About Us

Tomas Della Vedova
Senior Software Engineer - Elastic

Matteo Collina
Technical Director - NearForm

fastify

```
'use strict'

const fastify = require(

fastify.get('/', async (
  return { hello: 'world
})

fastify.listen(3000, con
```

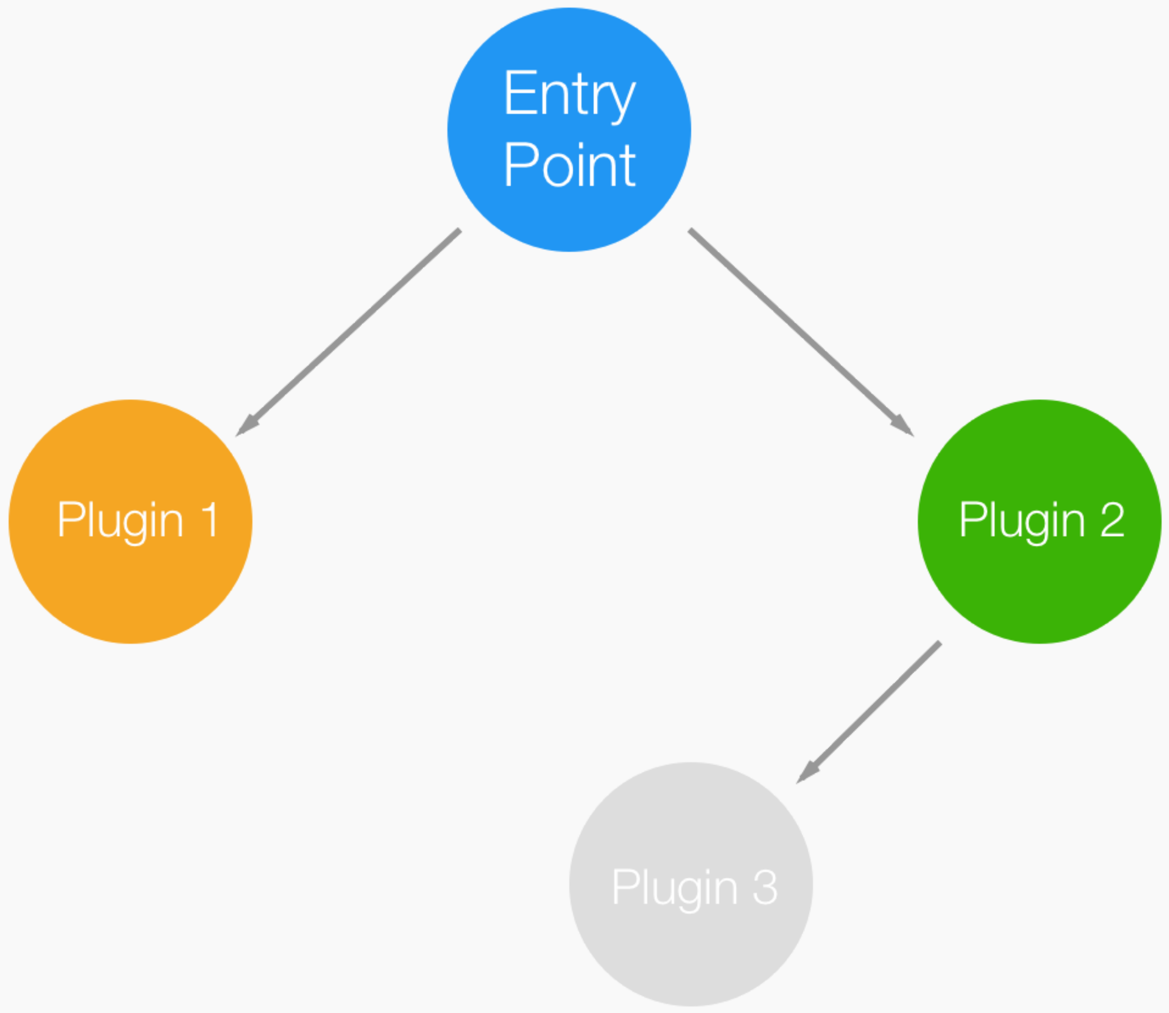
Fastify Plugins

A brief overview

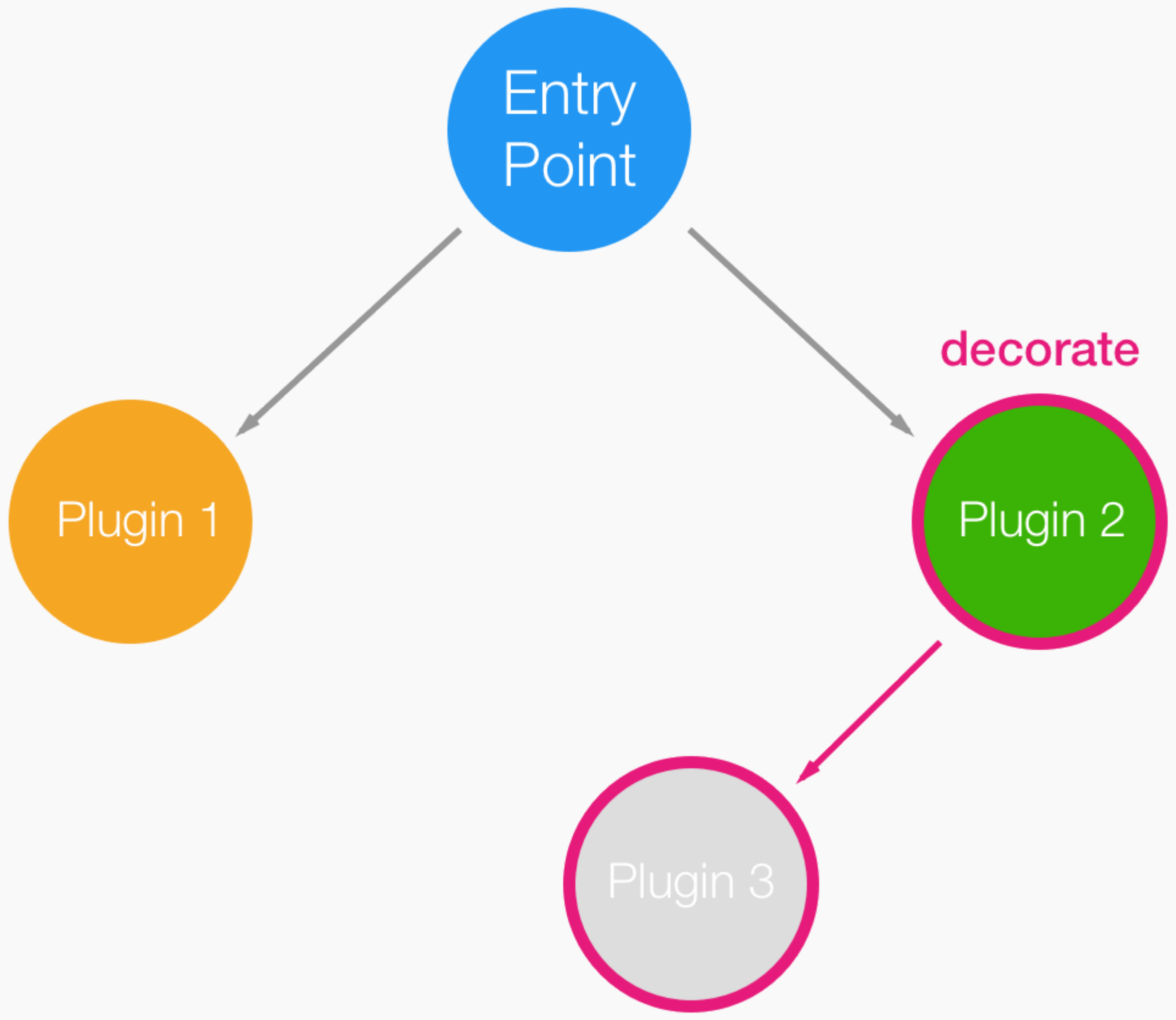
```
fastify.regist
  require('my-
  { options }
)
```

```
async function myPlugin (
  // register other plugi
  fastify register(    )

  // add hooks
  fastify addHook(    )

  // add decorator
  fastify decorate(    )

  // add routes
  fastify route(    )
}

module exports = myPlugir
```

Plugins: Architecture

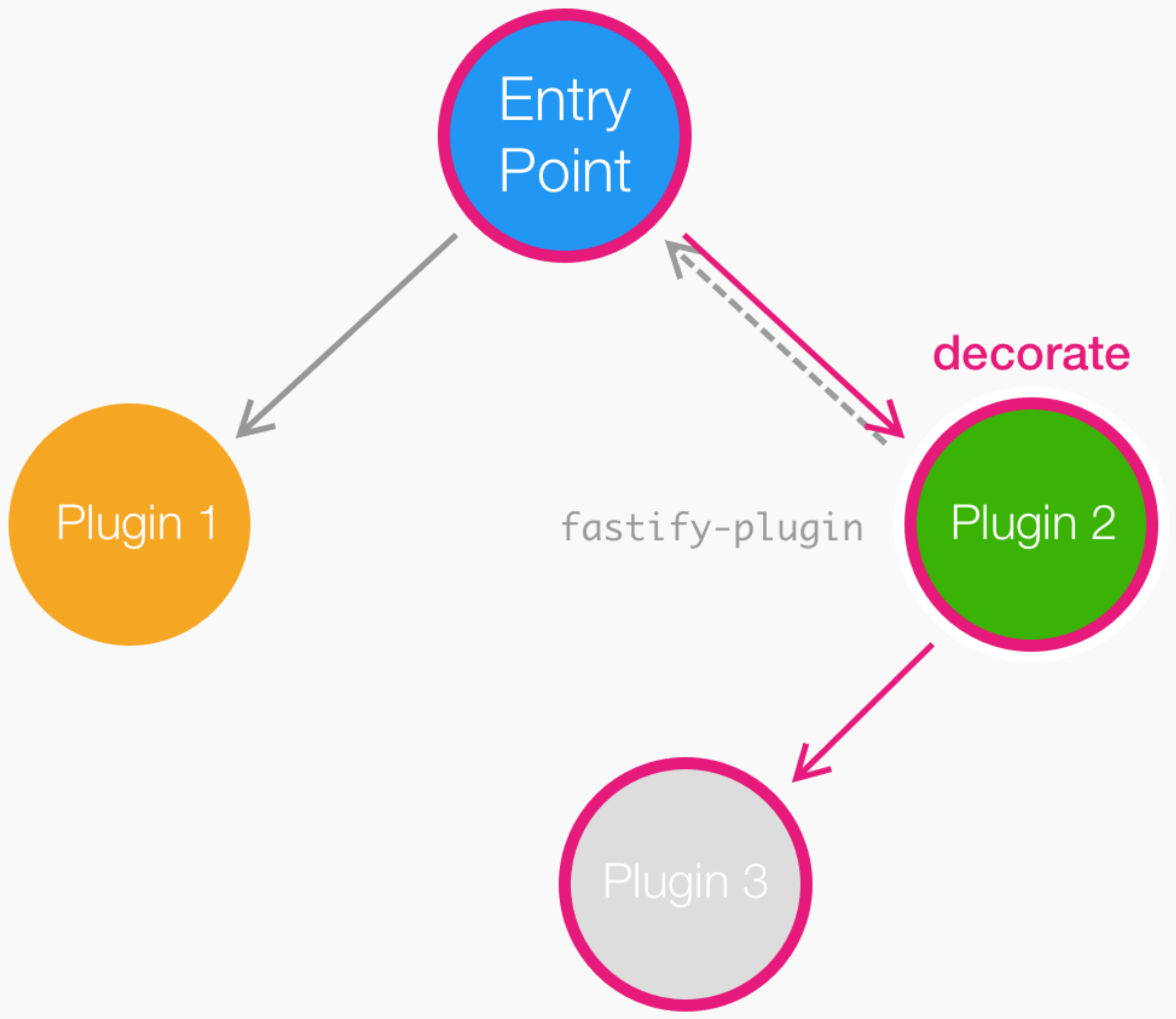


Plugins: Encapsulation

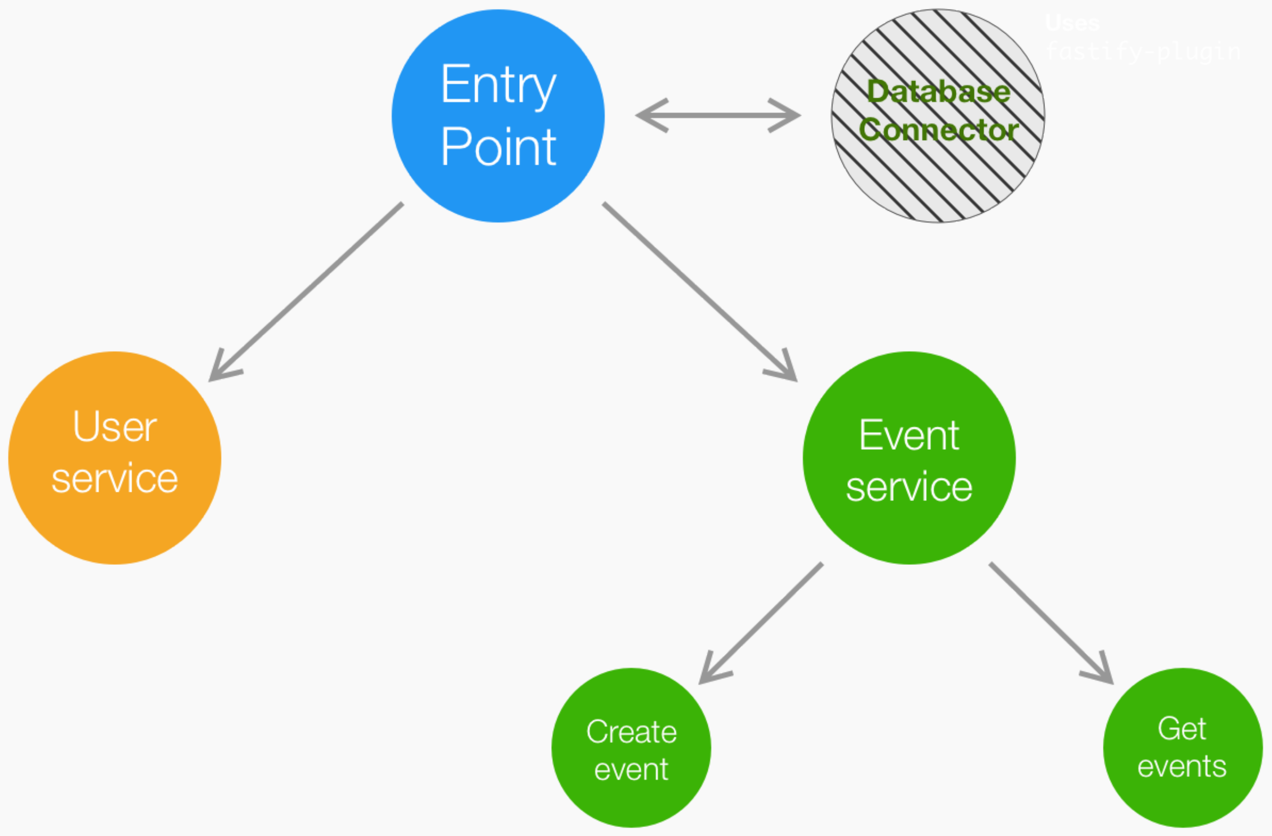


```
async function myPlugin (fas
  fastify.decorate('util', y
  // now you can use it with
}
```

Plugins: Encapsulation



Plugins: Real world



Encapsulation enables cool things

```
const fastify = require(

fastify.register(require
  prefix: '/v1'
})
```

Everything is a plugin

Let's start!

A little bit of configuration

Being consistent across microservices is a difficult task, to help you Fastify provides a powerful CLI.

```
npx fastify generate workshop
cd workshop
npm install
```

Project structure

- app.js: your entry point;
- services: the folder where you will declare all your endpoints;
- plugins: the folder where you will store all your custom plugins;
- test: the folder where you will declare all your test.

Scripts

- `npm start`: run your server;
- `npm run dev`: run your server with pretty logs (not suitable for production);
- `npm test`: run your test suite.

Exercise:

Generate the project and create
a simple status service that exposes a status route.

```
GET /status => { status: 'ok' }
```

Testing

```
const { test } = require('tap'
const Fastify = require('fasti
const app = require('../app')

test('Should expose a /status
  const fastify = Fastify()
  fastify.register(app)

  const response = await fasti
    method: 'GET',
    path: '/status'
  })

  t.strictEqual(response.statu
  t.deepEqual(
```

Exercise:

Test your application.

Let's add our database

```
npm i fastify-elasticsearch
npm i @delvedor/fastify-workshop-dataset
```

```
module.exports = async function (fastif
  fastify.register(AutoLoad, {
    dir: path.join(__dirname, 'plugins'
    options: Object.assign({}, opts)
  })

  fastify.register(AutoLoad, {
    dir: path.join(__dirname, 'services
    options: Object.assign({}, opts)
  })
}
```

Exercise:

Create an endpoint to index a new tweet,
the endpoint should return the tweet id.
Bonus: Add route validation.

You can use Hyperid for generating the IDs.

```
{
  id: String,
  text: String,
  user: String,
  time: DateString,
  topics: String[]
}
```

Exercise:

Test your application.

Exercise:

Create an endpoint to get a tweet by id.
Bonus: Add route validation.

```
GET /tweet/:id => {
                    id: String,
                    text: String,
                    user: String,
                    time: DateString,
                    topics: String[]
                  }
```

Exercise:

Test your application.

Exercise:

Create an endpoint to get a tweet timeline, ordered by time.
Bonus: Boost the results with a given topic.

```
GET /timeline => [{
                    id: String,
                    text: String,
                    user: String,
                    time: DateString,
                    topics: String[]
                 }]
```

Exercise:

Test your application.

Authentication

Protecting your application is important.

Exercise:

Create an authentication plugin that uses fastify-basic-auth.

Create also two fake users to test your application.

```
const users = {
  arya: 'stark', // Basic YXJ5YTpzdGFyaw==
  jon: 'snow' // Basic am9uOnNub3c=
}
```

```
'use strict'

const fp = require('fastify-plugin')
const basicAuth = require('fastify-b

const users = {
  arya: 'stark', // Basic YXJ5YTpzdG
  jon: 'snow' // Basic am9uOnNub3c=
}


async function basicAuthPlugin (fast
  fastify.register(basicAuth, { vali
  fastify.decorateRequest('user', nu

  async function validate (username,
    if (users[username] !== password
      return new Error('Invalid user
    }
```

Exercise:

Test your plugin.

Exercise:

Add the authentication to your routes.

```
fastify.route({
  method: 'GET',
  path: '/post/:id'
  handler: onGetPos
})
```

Exercise:

Update your test.

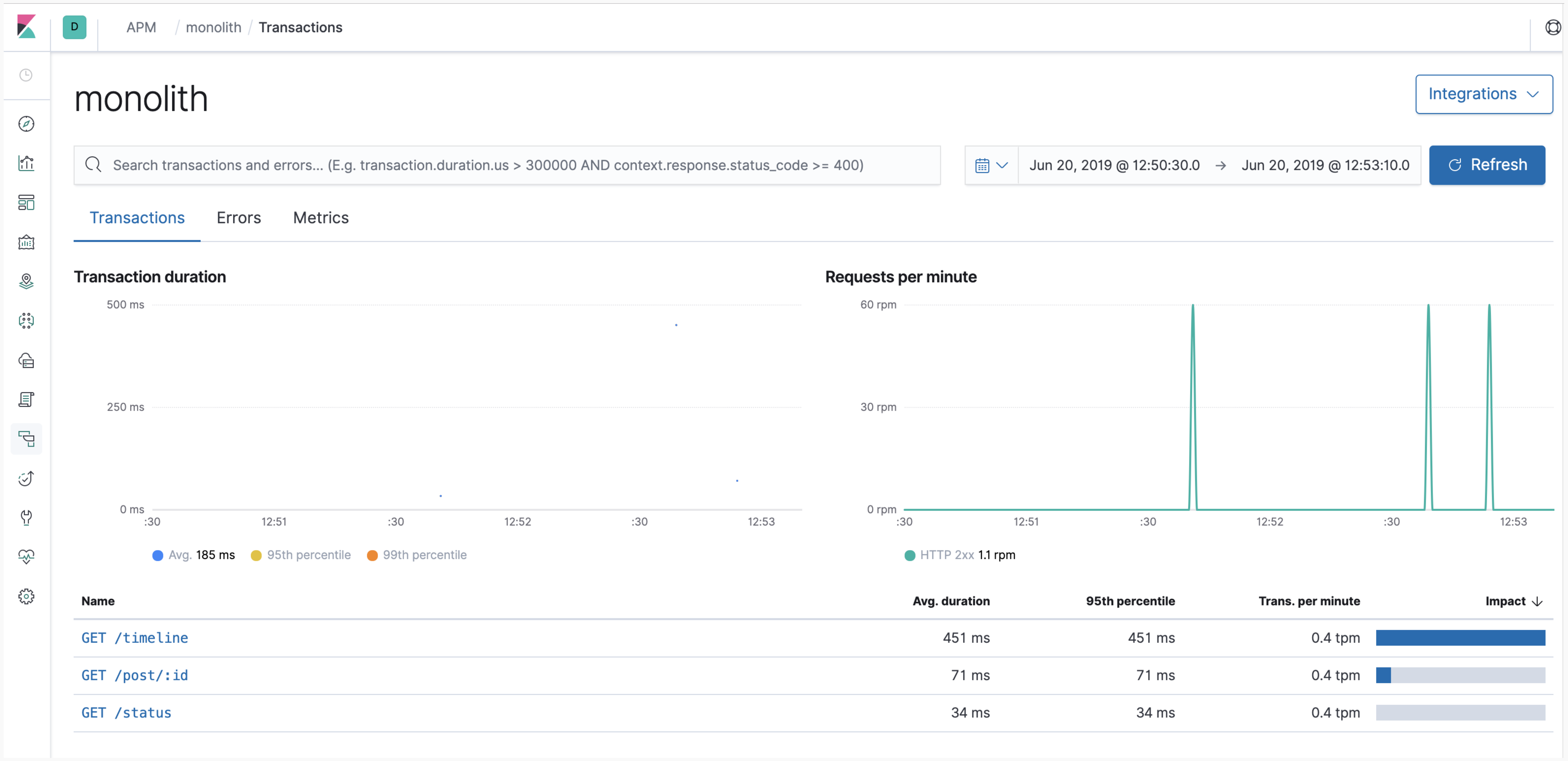
Let's talk about observability

```
npm i elastic-apm-node
```

```json
{
  "dev": "fastify start -l info -P app.js",
}
```

Exercise:

Open Kibana and run your application with APM.



Name	Avg. duration	95th percentile	Trans. per minute	Impact ↓
`GET /timeline`	451 ms	451 ms	0.4 tpm	
`GET /post/:id`	71 ms	71 ms	0.4 tpm	
`GET /status`	34 ms	34 ms	0.4 tpm	

From monolith to microservices

Let's begin!

Exercise:

- Duplicate the project three times
- Keep only one service per project
- Run the services!

Awesome!

Now update all your clients so they know
which address to call based on the service they need to use.

WRONG!

The infrastructure should be **transparent** to the client.

Exercise:

How can we fix this?

fastify

Thanks!