مشاهده‌ی کارایی در سیستم‌های مبتنی بر میکروسرویس

گزارش پروژه - قسمت دوم

استاد راهنما:

دکتر علیرضا شاملی

دانشجو:

علیرضا اروندی

گروه نرم‌افزار

دانشکده مهندسی و علوم کامپیوتر

نیم‌سال اول تحصیلی ۹۸-۹۹

فهرست مطالب

فهرست شکل‌ها

مقدمه

در این گزارش پروژه به توضیح بخش‌های مختلف پروژه می‌پردازیم. این گزارش شامل گزارش فاز دوم و سوم است یعنی پیاده سازی یک سیستم مبتنی بر معماری ریز سیستم‌ها[۱] و اضافه کردن یک سیستم گزارش‌گیری برای کنترل کارایی در برنامه اصلی و در نهایت در این پروژه از یک ابزار برای به تصویر کشیدن اتفاقات سیستم می‌پردازیم. این گزارش به چهار بخش اصلی تقسیم می‌شود. پیاده سازی سرویس مبتنی بر زیر سیستم‌ها، اجرای سرویس در یک محیط ایزوله مبتنی بر کانتینرها[۲]. اضافه کردن استاندارد اپن تریسینگ[۳] به سیستم‌ها، و نهایتا استفاده از زیپکین[۴] برای مشاهده‌ی کارایی سرویس‌ها.

---

۱ Microservices

۲ Containers

۳ OpenTracing

۴ Zipkin

سرویس مبتنی بر معماری زیر سیستم‌ها

معماری ریز سیستم‌ها یک معماری قدرتمند با بسیاری ویژگی خوب می‌باشد. این معماری به سرعت بخشیدن در توسعه‌ی نرم‌افزار می‌انجامد. در عین حال این معماری چالش های بسیاری با خود به همراه می آورد. به تیم های خودمختار کمک می‌کند تا بیشترین تاثیر را بتوانند به صورت مستقل بگذارند سیستم‌ها را طراحی کنند، بسازند و مستقر کنند. در عین حال نباید فراموش کنیم که این مسئله باعث می‌شود که پیچیدگی ها به سطح شبکه وارد شوند. یکی از این چالش‌ها خطایابی در سرویس‌ها است که به علت پخش بودن معماری است.

محیط‌های ایزوله (کانتینرها)

برنامه‌نویس‌ها همواره با این مشکل مواجه هستند که برنامه‌ای را روی سیستم خود کدنویسی، دیباگ، تست و اجرا کرده و از صحت عملکرد آن اطمینان حاصل می‌کنند اما همین که آن را به محیط یا پلتفرم دیگری همچون یک سرور دیپلوی می‌کنند، باگ‌ها و مشکلات عجیب‌وغریب بسیاری را تجربه می‌کنند و اینجا است که کانتینر خودنمایی می‌کند. به عبارت دیگر، کانتینر این اطمینان را حاصل می‌کند که نرم‌افزار فارغ از اینکه روی چه پلتفرمی دیپلوی[۵] گردد، کاملاً به درستی اجرا گردد و عملکرد یکسانی داشته باشد. حدود ۵ سال پیش فردی با نام سالامان هایکز[۶]، سازوکاری با نام داکر را راه‌اندازی کرد. هدف از این کار تعامل راحت‌تر با کانتینرها بود. این ایده با موفقیت روبرو شد و در سال ۲۰۱۴ پس از انتشار داکر[۷] شاهد افزایش محبوبیت آن بودیم. در نتیجه شرکت‌ها یکی پس از دیگری نرم‌افزارهای تحت سرور را به جای ماشین‌های مجازی در بستر داکر راه‌اندازی کردند. این روزها داکر و مبی[۸] که به عنوان مجموعه بالاسری داکر شناخته می‌شود، مخاطبین بسیار زیادی جذب کرده و بنا به گزارش داکر چیزی بیشتر از ۳.۵ اپلیکیشن در کانتینرها از طریق فناوری داکر قرار دارند و بیشتر از ۳.۷ میلیارد اپلیکیشن از این طریق دانلود شده است.

داکر امکانی را فراهم آورده است که پروسه‌ها و نرم‌افزارها به صورت مجزا در محیط کاملا ایزوله‌ای بر روی کرنل لینوکس راه‌اندازی شود که به این محیط و بسته‌ی ایزوله، کانتینر می‌گویند. کانتینر این امکان را برای برنامه نویسان و توسعه دهندگان اپلیکیشن ها فراهم می کند تا یک برنامه را با تمام ماژول ها و کامپوننت‌های وابسته آن (مانند کتابخانه ها ، توابع و ...) یکی کرده و به صورت یک پکیج درآورده تا آن برنامه تولید شده در پلتفرم ها و سیستمهای مختلف بدون مشکل اجرا شود، در حقیقت بدون نگرانی از تنظیمات و وابستگی‌های یک برنامه خاص در پلتفرم های دیگر، آن برنامه در هر محیطی اجرا شود.

داکر که در بالا به آن اشاره شد وظیفه مدیریت کانتینرها را به عهده دارد و بیشتر شبیه یک ماشین مجازی عمل میکند، تفاوت داکر با ماشین مجازی در این است که در یا ماشین مجازی برای اجرای اپلیکیشن و برنامه های مختلف که بخواهیم به صورت ایزوله و مجزا از هم کار کنند باید ماشین مجازی های مختلف ساخته شود که همین موضوع بار پردازشی و هدر رفت منابع سیستمی را روی سرور به همراه دارد. ولی در داکر روی یک ماشین مجازی خاص که میتواند دارای سیستم عامل ویندوز یا لینوکس باشد، ماژول داکر نصب شده و سپس روی سرویس داکر، کانتینرهای مختلف حاوی برنامه ها و اپلیکیشن های مختلف نصب و اجرا می‌شوند بدون اینکه کانتینرها به هم دسترسی داشته باشند. بدین صورت کانتینرها از هم ایزوله هستند و نیاز ما برای ایجاد چندین ماشین مجازی را مرتفع میسازند.

در این پروژه از داکر برای پیاده سازی سرویس‌ها استفاده شده‌است و حتی ابزارها هم تحت داکر استفاده می‌شوند. برای استفاده از داکر مشکلات مختلفی از جمله تحریم‌ها وجود دارد. برای رد شدن از این مشکل از یک سرویس ایرانی که پارس‌پک ارائه کرده استفاده شده است.

---

۵ Deploy

۶ Solomon Hykes

۷ Docker

۸ Moby

ردیابی توزیع شده

ابزارهای مشاهده قدیمی همچنان کار خود را می‌توانند انجام دهند اما در هر سرویس به صورت جداگانه و برای سرویس‌های توزیع شده این کار را نمی‌توانند انجام دهند به صورتی که ارتباط میان آنها را بررسی کنند. ردیابی توزیع شده، از طریق ایجاد کردن معاملات[۹] از خدمات توزیع شده و به دست آوردن اطلاعات از طریق ارتباطات، امکان ردیابی در معماری‌های توزیع شده را فراهم می کند.

ایده ردیابی توزیع شده یک ایده جدید نیست و در سال ۲۰۱۰ گوگل برای اولین بار از یک مقاله رونمایی کرد که در این حوزه بود. این ایده برای کنترل کارایی در سیستم‌ها و سرویس‌های گوگل بود که دپر[۱۰] نام داشت. در این مقاله دو مفهوم اصلی اسپن[۱۱] و تریس[۱۲] معرفی شد که در بخش استاندارد به صورت کامل به توضیح این بخش می‌پردازیم.

استاندارد اپن ترسینگ

برای پیاده‌سازی از این استاندارد استفاده می‌کنیم. این استاندارد برای کنترل کردن ریز سیستم‌ها است. در این استاندارد دو مفهوم اسپن و تریس را داریم که در ادامه هر کدام را توضیح می‌دهیم.

اسپن

اسپن یک واحد منطقی از کار در سیستم را نشان می دهد که دارای نام عملیات ، زمان شروع و مدت زمان است. ممکن است اسپن تو در تو و به ترتیب باشد تا روابط را مدل کنند. فراخوانی آر.پی.سی.[۱۳] مانند درخواست اچ.تی.تی.پی.[۱۴] یا فراخوانی از پایگاه داده نمونه ای از اسپن است ، اما می‌توان فراخوانی‌های داخلی را هم نمونه‌ای از اسپن در نظر گرفت. اسپن‌ها توسط ایونت‌ها[۱۵] در برنامه کنترل می‌شوند. می توان آنها را با داده های عملیاتی شروع کرد، به پایان رساند و گسترش داد که خطایابی را آسان تر می کند. به عنوان مثال وقتی یک فراخوانی اچ.تی.تی.پی. را به سرویس دیگری ایجاد می کنیم که می خواهیم شروع کنیم و وقتی جوابمان را دریافت کردیم می‌خواهیم آن را تمام کنیم می‌توانیم آن را با کد وضعیت و داده‌های دیگر تزئین کنیم که بتوان راحت تر دسته بندی کرد و بررسی کرد.

تریس

تریس توسط یک یا چند اسپن نمایش داده می‌شود. این یک مسیر اجرا از طریق سیستم است.

---

۹ Transactions

۱۰ Dapper

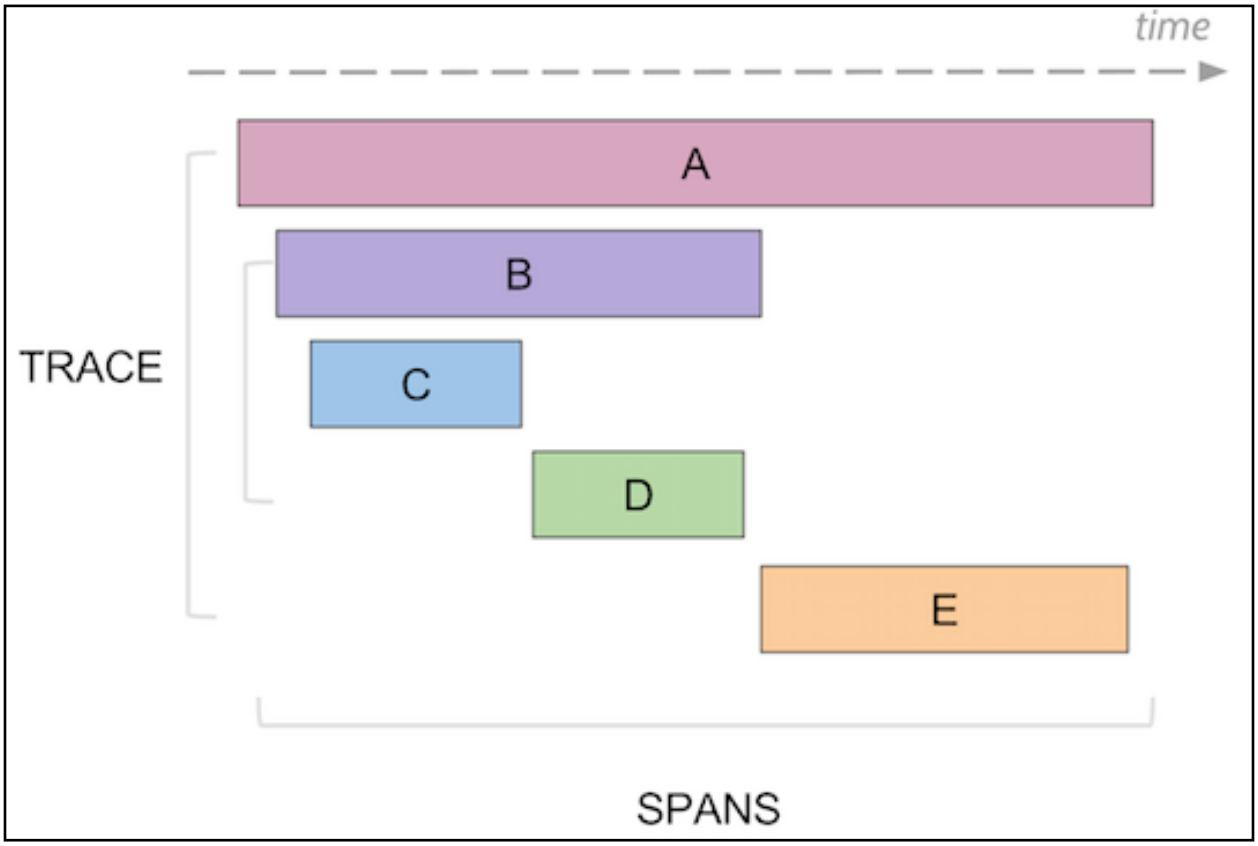
۱۱ Span

۱۲ Trace

۱۳ RPC

۱۴ HTTP

۱۵ Events



شکل شماره ۱

انتشار کانتکست[۱۶]

برای اینکه بتوانیم اسپن‌ها را به هم وصل کنیم نیاز داریم که یک کانتکست را بین آنها به اشتراک بگذاریم. برای مثال باید بتوانیم رابطه پدر و فرزند را بین اسپن‌ها برقرار کنیم. ارتباط بین پروسه‌ها می تواند به شکل‌های مختلفی و پروتکل‌های مختلفی باشد مثلاً فراخوانی‌های تحت اچ.تی.تی.پی.، آر.پی.سی. یا هر چیز دیگری باشد. برای به اشتراک گذاشتن کانتکست می‌توانیم از یک هدر[۱۷] استفاده کنیم.

مشاهده

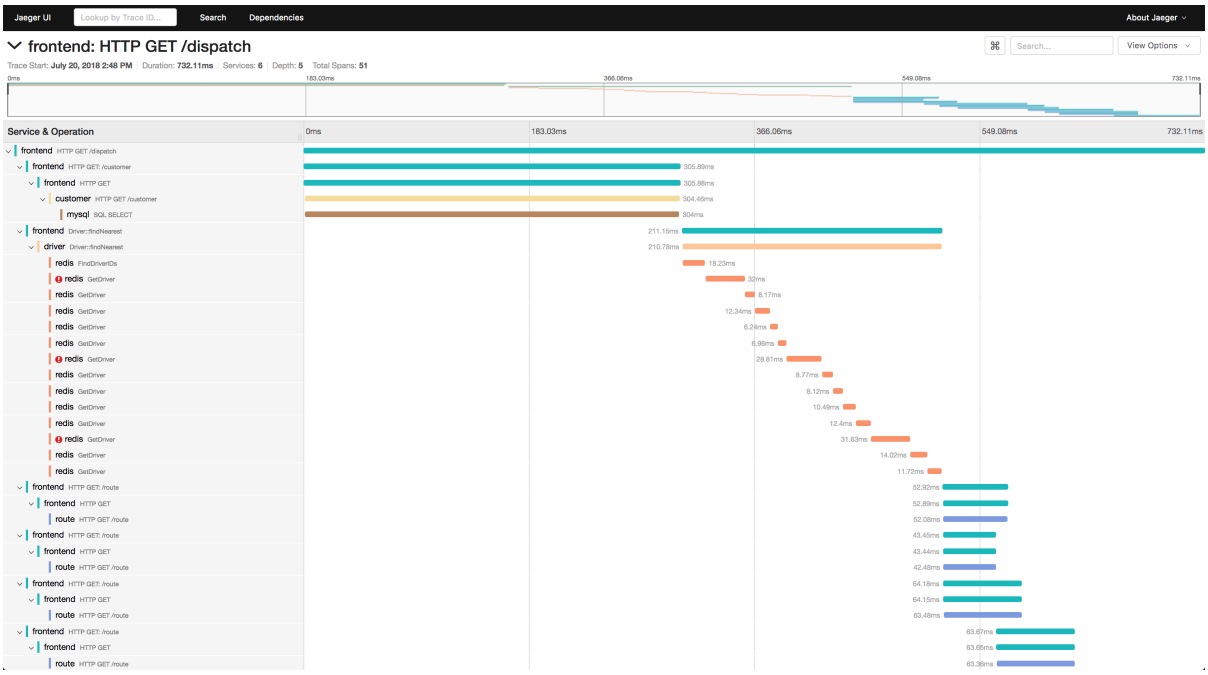
برای مشاهده و کنترل کردن سرویس‌ها باید بر اساس استاندارد اپن تریسینگ استفاده کنیم و سپس با برنامه‌هایی آنها را نمایش دهیم. برنامه‌های مختلفی برای نمایش وجود دارد. دو برنامه‌ی مهم جگر[۱۸] و زیپکین وجود دارد که در این پروژه از زیپکین استفاده می‌کنیم. در ادامه از صفحه‌ی اصلی هر دو برنامه یک شکل آمده.

---

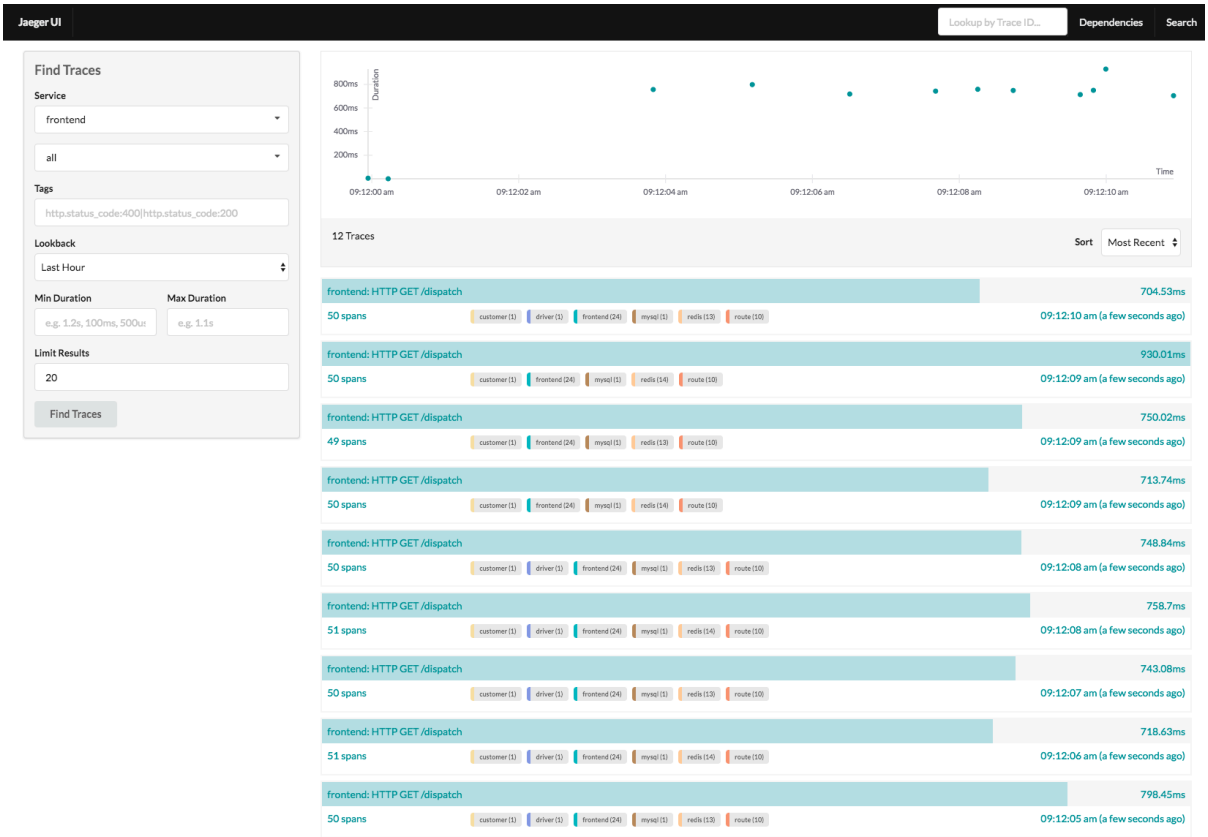
۱۶ Context Propagation

۱۷ Header

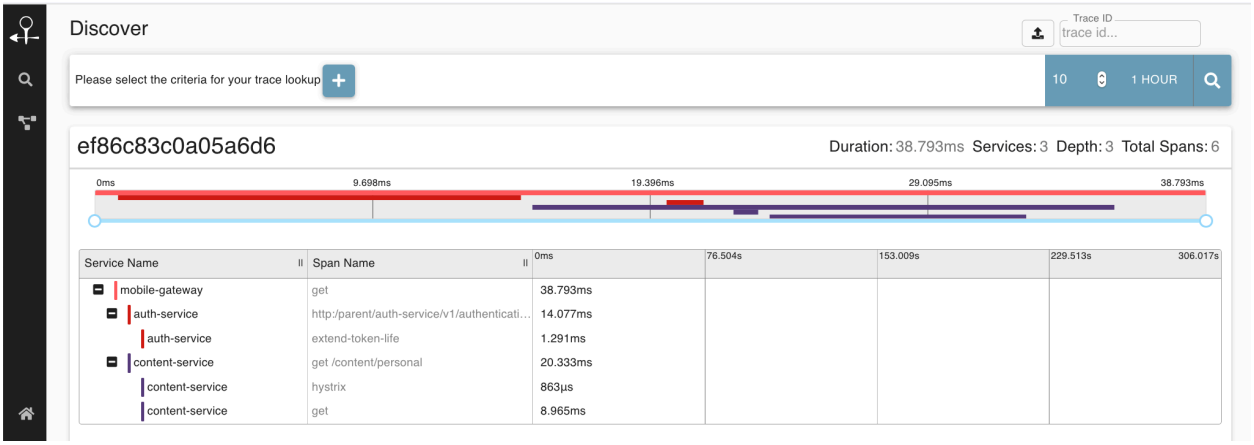
۱۸ Jaeger



شکل شماره ۲ – صفحه‌ای از جگر



شکل شماره ۳ – صفحه‌ی اصلی جگر



شکل شماره ۴ – صفحه‌ی اصلی زیپکین

پیاده‌سازی

در این بخش به توضیح پیاده‌سازی می‌پردازیم. برای پیاده سازی یک مجموعه سرویس با نودجی.اس.[۱۹] و اکسپرس[۲۰] نوشیم و با زیپکین آنها را نمایش می‌دهیم.

برای استفاده از داکر برای اجرای زیپکین استفاده می‌کنیم و سپس در مرحله‌ی بعد سرویس‌ها را هم وارد داکر می‌کنیم. ابتدا باید سرویس داکر را نصب و اجرا کنیم سپس باید داکر کامپوز[۲۱] را نصب کنیم. کد زیر را در فایل docker-compose.yml قرار می‌دهیم.

```yaml
version: "2"

services:
  storage:
    image: openzipkin/zipkin-mysql
    container_name: zipkin-playground-mysql
    environment:
      - MYSQL_HOST=mysql

  zipkin:
    image: openzipkin/zipkin
    container_name: zipkin-playground-zipkin
    environment:
      - STORAGE_TYPE=mysql
      - MYSQL_HOST=zipkin-playground-mysql
    ports:
      - 9411:9411
    depends_on:
      - storage
```

شکل شماره ۵ – کد داکر کامپوز

---

[۱۹] NodeJS

[۲۰] Express

[۲۱] Docker compose

این کد باعث می‌شود که دو کانتینر مای.اس.کیو.ال[۲۲] و زیپکین ساخته شوند و در یک شبکه باشند. حال یک سرویس با اکسپرس می‌سازیم. این سرویس یک آدرس دارد برای تست. کد این آدرس سرویس در شکل شماره ۶ آمده است.

```
JS service1.js > ...
1   const express = require("express");
2
3   const app = express();
4   const port = 3000;
5
6   app.get("/time", (req, res) => {
7     res.json({ currentDate: new Date().getTime() });
8   });
9
10  app.listen(port, () => console.log(`Date service listening on port ${port}`));
```

شکل شماره ۶ - سرویس شماره ۱

برای تریس کردن کد از کتابخانه‌ی زیپکین برای جاوااسکریپت استفاده می‌کنیم. این بخش را به کد اضافه می‌کنیم که در شکل شماره ۷ تا ۹ آمده است.

```
2
3   // Import zipkin stuff
4   const { Tracer, ExplicitContext, BatchRecorder, jsonEncoder } = require("zipkin");
5   const { HttpLogger } = require("zipkin-transport-http");
6   const zipkinMiddleware = require("zipkin-instrumentation-express").expressMiddleware;
7
8   const ZIPKIN_ENDPOINT = process.env.ZIPKIN_ENDPOINT || "http://localhost:9411";
9
```

شکل شماره ۷ - استفاده از کتابخانه‌ی زیپکین

```
10  // Get ourselves a zipkin tracer
11  const tracer = new Tracer({
12    ctxImpl: new ExplicitContext(),
13    recorder: new BatchRecorder({
14      logger: new HttpLogger({
15        endpoint: `${ZIPKIN_ENDPOINT}/api/v2/spans`,
16        jsonEncoder: jsonEncoder.JSON_V2,
17      }),
18    }),
19    localServiceName: "date-service",
20  });
```

شکل شماره ۸ - ساخت یک تریسر

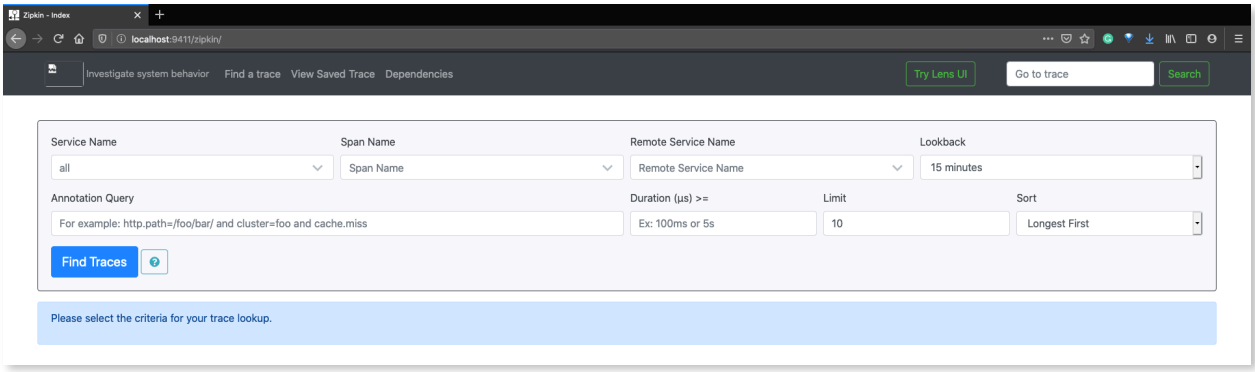
---

[۲۲] MySQL

```
25    // Add zipkin express middleware
26    app.use(zipkinMiddleware({ tracer }));
```

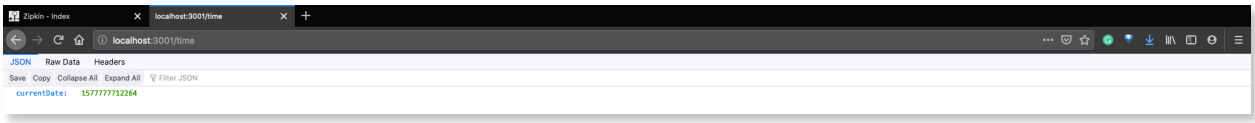
شکل شماره ۹ – اتصال به اکسپرس

تا اینجا فقط یک tracer ساخته‌ایم و آن را به اکسپرس وصل کرده‌ایم. از طرفی این کد اطلاعات را به سرور زیپکین هم می‌فرستد. با اجرای زیپکین در مرور گر می‌توانیم صفحه‌ی زیر را ببینیم.



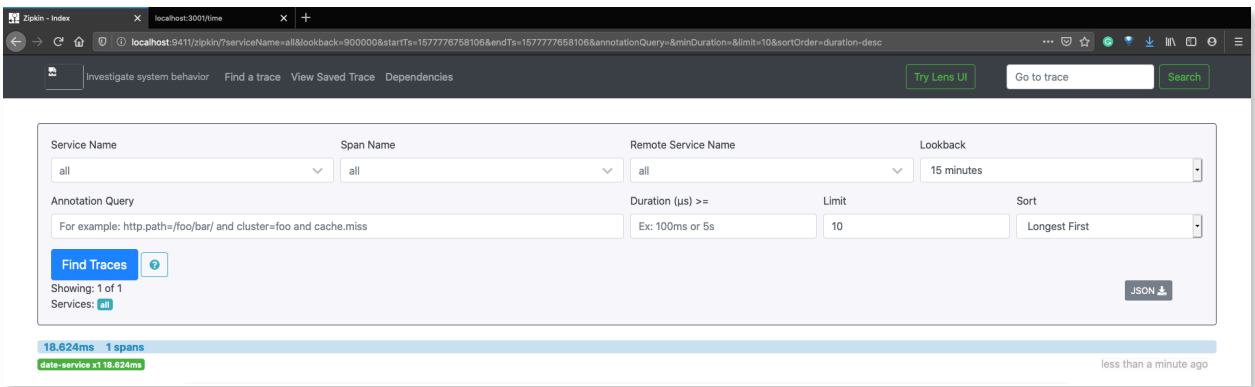
شکل شماره ۱۰ – زیپکین در پروژه

سپس در مرورگر سرویس را صدا کرده که در شکل زیر آمده.



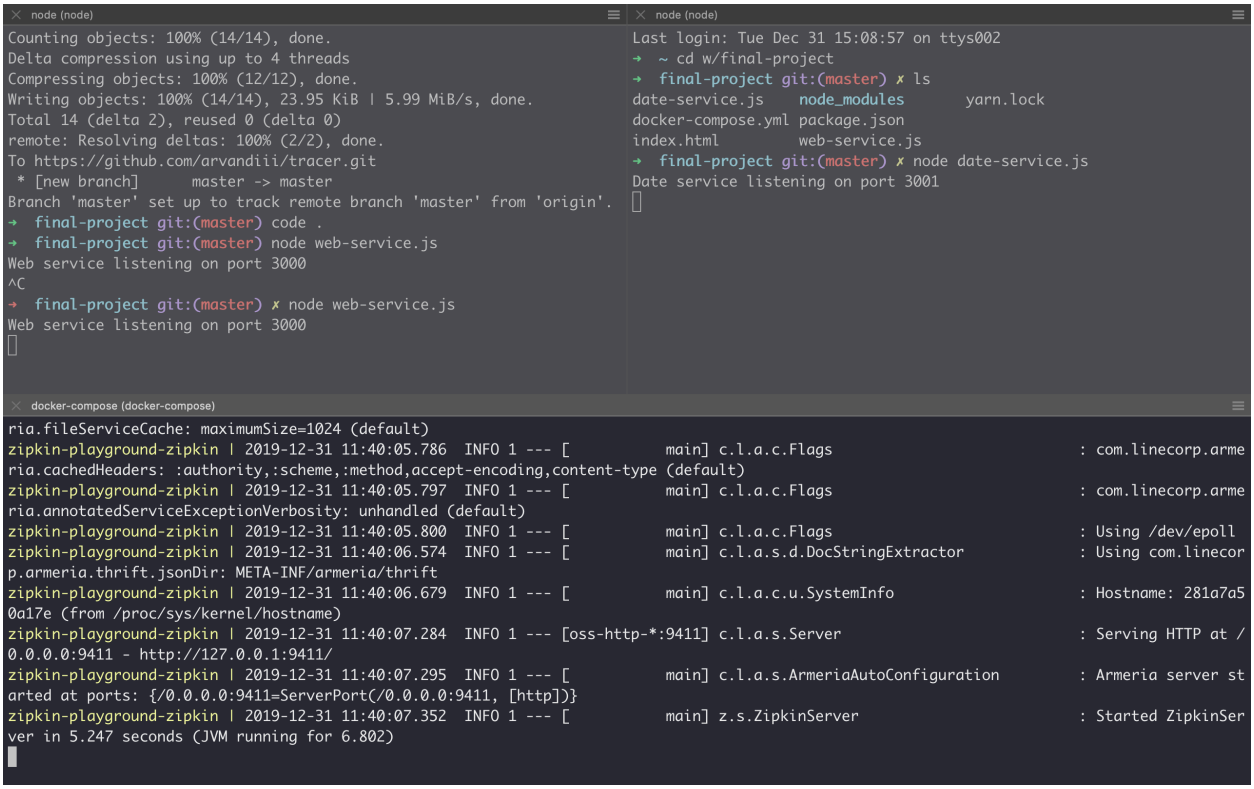
شکل شماره ۱۰ – فراخوانی سرویس

حال دوباره خروجی زیپکین را می‌بینیم در اینجا یک اسپن ایجاد شده که همان فراخوانی تابع است. این اطلاعات توسط سرویس به سرور زیپکین ارسال می‌شود و زیپکین آن را نمایش می‌دهد که در شکل زیر می‌بینید.



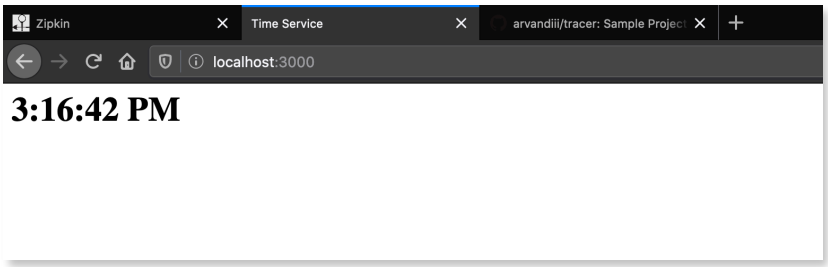
شکل شماره ۱۱ – زیپکین بعد از فراخوانی سرویس

این پیاده‌سازی ساده شمای کلی پروژه را نشان می‌دهد. برای پروژه بخش‌های مختلفی در نظر گرفته‌شده‌است. حال می‌خواهیم یک سرویس دیگر هم اضافه کنیم برای ایجاد پیچیدگی. در این مرحله با استفاده از کتابخانه‌ی اکسیوس[۲۳] از سرویس تعریف شده یک فراخوانی به سرویس دیگری که اضافه کردیم می‌زنیم. در ادامه سرویس دوم را هم می‌سازیم. این دو سرویس و زیپکین را اجرا می‌کنیم. که در تصویر زیر می‌بینید. بالا سمت راست سرویس اول و سمت چپ سرویس دوم اجرا شده‌است. در پایین هم داکر زیپکین بالا آمده.



شکل شماره‌ی ۱۲ - اجرای همزمان سرویس‌ها و زیپکین

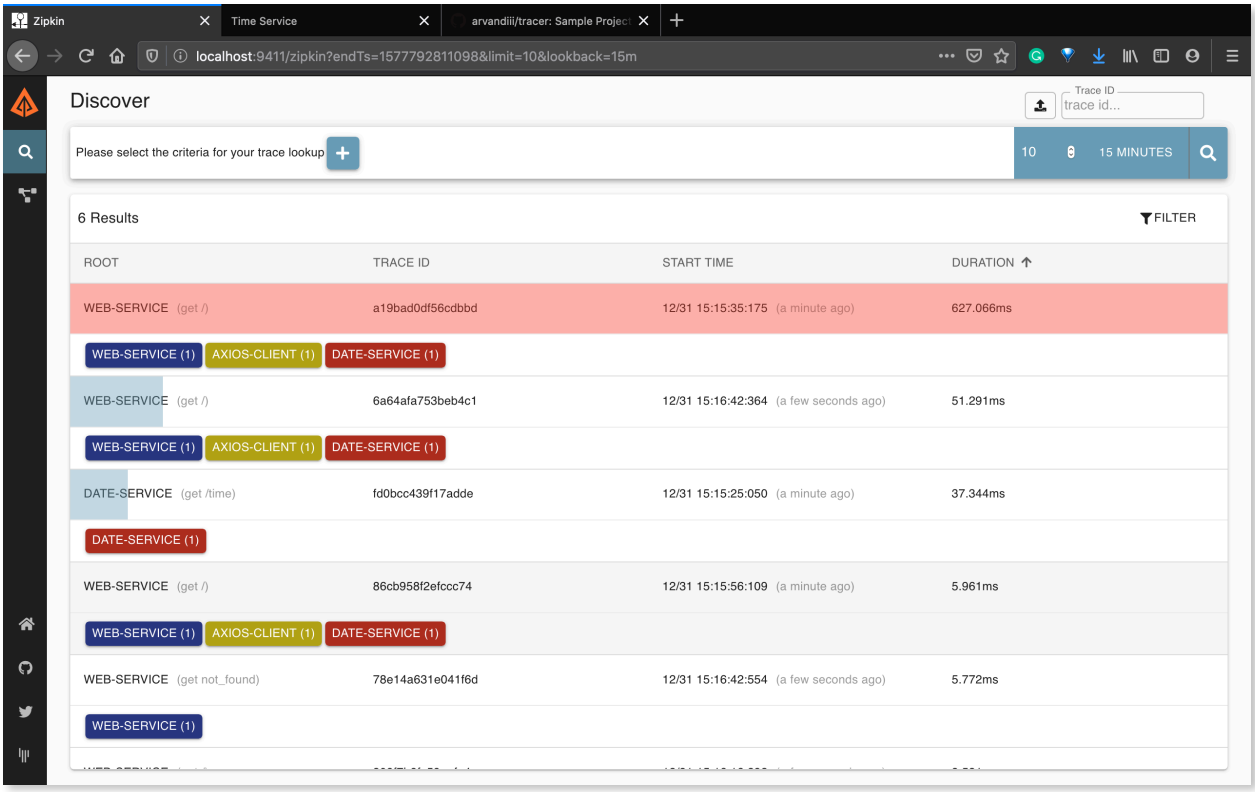
در ادامه‌ی پروژه این روال با کمک داکر اتومات می‌شود و با اجرای یک دستور داکر همه در کنار هم بالا می‌آیند. حال به تحلیل این سیستم می‌پردازیم. با باز کردن مرورگر و فراخوانی سرویس اول خروجی زیر را همان‌طور که انتظار می‌رفت دریافت می‌کنیم.



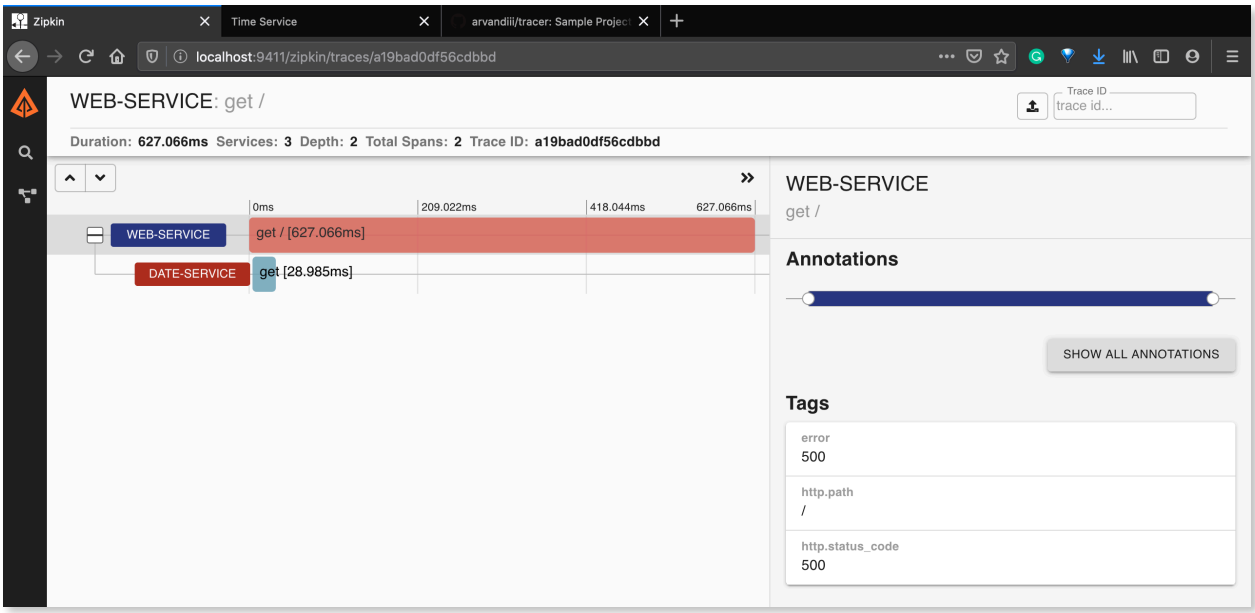
شکل شماره ۱۳ - خروجی سرویس اول

[۲۳] axios

حال لاگ‌ها را در زیپکین بررسی می‌کنیم. همان طور که در شکل ۱۴ مشاهده می‌کنیم این فراخوانی‌های تو در تو وجود دارند. این فراخوانی شامل سه بخش است که در شکل ۱۵ با جزئیات آمده‌است. این فراخوانی‌ها به ترتیب فراخوانی وب سرویس (آبی)، فراخوانی کلاینت اکسیوس (زرد) و نهایتا سرویس دوم (قرمز) است که در شکل زیر قابل مشاهده است.



شکل شماره ۱۴ – زیپکین پس از فراخوانی تو در تو



شکل شماره ۱۵ – جزئیات فراخوانی تو در تو

این خروجی نمایان‌گر صحت پروژه تا این مرحله است. در ادامه‌ی پروژه به خودکار سازی این سرویس‌ها توسط داکر می‌پردازیم و نهایتا با اضافه کردن سرویس‌های بیشتر و ایجاد پیچیدگی یک سرویس نسبتا واقعی را شبیه سازی می‌کنیم.