دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گزارش کارآموزی

آشنایی با اصول مهندسی و توسعه نرم‌افزار

نگارش

محمد اژدری

استاد راهنما

دکتر مهدی راستی

شهریور ماه ۱۳۹۹

سپاس‌گزاری

اینجانب محمد اژدری مراتب امتنان خود را نسبت به استاد راهنما و شرکت بله که طی تدوین این گزارش و طی هر چه بهتر دوره کارآموزی همواره مرا یاری نموده‌اند، ابراز می‌دارم.

محمد اژدری
شهریور ماه ۹۹

فهرست مطالب

صفحه

فهرست اشکال و جداول

صفحه

فصل اول

مقدمه

مقدمه

امروزه مهندسی نرم‌افزار یکی از صنعت‌های مهم و حیاتی در دنیا به شمار می‌رود. از این رو شرکت‌ها و سرمایه‌گذاران بسیاری در این صنعت وارد شده‌اند. بنابراین جامعه دانشگاهی نیز سعی می‌کند در تربیت دانشجوهای خود به این صنعت توجه ویژه‌ای داشته باشد.

مطالبی که در دانشگاه‌های دنیا و مخصوصا در ایران تدریس می‌شوند، کاملا جنبه آکادمیک و تئوری دارند و ازاین‌رو فاصله صنعت و دانشگاه متاسفانه روزبه‌روز بیشتر می‌شود.

اما در این میان دانشگاه در رشته‌های مهندسی واحدی به نام کارآموزی دارد که در آن دانشجو وارد صنعت شده و به صورت واقعی مواردی را که آموخته است، انجام می‌دهد.

صنعت نرم‌افزار به شاخه‌های زیادی تقسیم‌بندی می‌شود. و تابه حال تکلونوژی‌های زیادی برایش توسعه داده شده است.

برنامه‌نویسی وب یکی از شاخه‌های نرم‌افزار به شمار می‌آید. همه وب‌سایت‌ها و برنامه‌هایی که در بستر اینترنت سرویس می‌دهند، در این حیطه قرار دارند.

خود برنامه‌نویسی وب به سه دسته کلی: فرانت‌اند، بکند و زیرساخت تقسیم می‌شود.

تخصص‌های حرفه‌ای افراد در بین این‌ها معمولا جدا می‌شود.

در این میان تخصص‌های بکند و زیرساخت می‌توانند قدری با هم ترکیب شوند. و با همدیگر همپوشانی داشته باشند. تا نهایتا با توسعه فرانت‌اند، منجر به تولد یک وب اپلیکیشن شوند.

فصل دوم

شرکت اپلیکیشن بله

پلتفرم مالی اجتماعی بله

همان‌طور که بیان شد، شرکت‌های بسیاری در صنعت نرم‌افزار، چه در ایران و چه در جهان فعالیت دارند. و سرویس‌هایی را اکثرا دربستر اینترنت به کاربران خود می‌دهند و از این راه کسب درآمد هم می‌کنند. در این میان اپلیکیشن بله، که یک پلتفرم مالی اجتماعی شناخته می‌شود، جهت تسهیل ارتباطات مالی افراد توسعه داده شده است و سرویس‌هایی هم از نوع پیامرسان و هم از نوع خدمات پرداختی ارائه می‌دهد. بله هم‌اکنون هفت میلیون کاربر دارد و روزبه‌روز به آن‌ها افزوده می‌شود. و امیدوار است که بتواند همواره خدمات بهتری را به مردم ایران ارائه دهد.

2-1- تاریخچه شرکت

در سال 1394 گروهی از فارغ‌التحصیلان دانشگاهی اقدام به توسعه بله نمودند و پس از آن با پیشرفت‌هایی که حاصل شد، این پلتفرم مورد حمایت بانک ملی قرار گرفت و هم اکنون در ساختمان سداد که از زیرمجموعه‌های نرم‌افزاری بانک ملی به شمار می‌آید، قرار دارد.

2-2- هدف

هدف شرکت تسهیل و ترکیب ارتباطات مالی افراد در بستر پیامرسان است. و تا این لحظه نیز برای رسیدن به آن تلاش‌های فنی و محصولی خود را انجام می‌دهد.

2-3- اهمیت

در دنیای امروزی با توجه به زیاد شدن برنامه‌های آنلاین و رغبت جامعه به داشتن سیستم‌های یک‌پارچه، این نیاز حس می‌شود که کاربران علاوه‌بر ارتباط معمولی، ارتباط‌های مالی خود را نیز آنلاین و در پیامرسان داشته باشند. مثلا برای یک‌دیگر درخواست پول ارسال کنند، با بانک خود ارتباط برقرار کنند و یا شارژ یا اینترنت خریداری نمایند.

2-4- چالش‌ها

بله نیز مانند اکثر شرکت‌های دیگر با چالش‌های فنی و محصولی روبه‌رو است و همیشه سعی در حل آن‌ها دارد. به لحاظ فنی توسعه پیام‌رسان از کارهای سخت و پرچالش در دنیای نرم‌افزار به شمار می‌رود. با توجه به حجم زیاد کاربران و توانایی برنامه در سرویس‌دهی بلادرنگ به آن‌ها، شرکت نیازمند استفاده از تلکونوژی‌های روز دنیا و توانایی فنی بالایی است.

برای حل این چالش‌ها شرکت هم در جذب نیروهای جدید و هم در یادگیرنده نگه داشتن فضای شرکت سعی و تلاش فراوان دارد.

فصل سوم

معماری میکروسرویس در مهندسی نرم‌افزار

درباره‌ی معماری میکروسرویس‌ها

میکروسرویس، واژه‌ای که این روزها شاید در اکثر شرکت‌های تکلونوژی دنیا اسمش شنیده می‌شود.

در این فصل به تعریف این معماری از منظر خودم و مزایا و معایب آن می‌پردازم و در فصل بعدی جزئیاتی را هم در این باره منتشر می‌کنم.

3-1- تعریف معماری میکروسرویس‌ها

من به موجودیتی سرویس می‌گویم که به تنهایی قابل توسعه، دیپلوی، تست و scale باشد.

برخلاف اسمش که میکرو در آن وجود دارد، هیچ‌وقت این تصور را نداشته باشید که یک سرویس هر چه قدر کوچک‌تر باشد بهتر است.

3-2- فرق معماری میکروسرویس‌ها با یکپارچه

بدیهی‌ترین فرق این است که در معماری مونولوتیک، یک سیستم از یک سرویس تشکیل شده است ولی در معماری میکروسرویس‌ها از چند سرویس.

3-3- برخی ویژگی‌ها، معایب و مزایا

مثل هر معماری دیگری، این معماری نیز مزایا و معایب خودش را دارد و به تناسب نیاز می‌تواند در یک سیستم یا پروژه استفاده بشود یا نشود.

3-3-1- هر سرویس به تنهایی قابل توسعه دادن است.

شاید بگویید مگر در بقیه حالت‌ها قابل توسعه نیست؟ در جواب باید بگویم که بله در برخی حالت‌ها اگر در زمان مناسبش اقدام درستی انجام نشود سرویس توانایی توسعه‌اش را از دست می‌دهد. حالا چگونه؟

فرض کنید که سیستم خیلی بزرگ دارید و چند تیم روی آن کار می‌کنند. حالا زیاد هم نگوییم مثلا سه تیم پنج نفره. هر چقدر هم کد یک پروژه تمیز باشد و وابستگی در آن کم باشد، یک برنامه‌نویس چون یک آدم است، می‌تواند اشتباه کند و اگر جایی اشتباه کند همین اشتباه او باعث خطا در سیستم شده و کل مجموعه را از کار می‌اندازد. و از جنبه دیگر، وقتی با یک سرویس سر و کار دارید و یک برنامه‌نویسی باشید که تازه به آن مجموعه اضافه شده‌اید شاید سال‌ها طول بکشد که از همه جای آن سر در بیاورید و عملا نمی‌توانید توسعه خاصی روی آن انجام دهید. ولی اگر این سیستم به چند سرویس شکسته شده باشد شما در یک تیم مجزا به توسعه سرویس خودتان می‌پردازید. و با بقیه کاری ندارید.

3-3-2- هر سرویس به تنهایی قابل دیپلوی کردن است.

همین سرویس بزرگ قبلی را تصور کنید، شما یک کامیت روی برنچ مستر انجام می‌دهید. در بهترین حالت احتمالا دو روز طول بکشد که کدی که شما زده‌اید بر روی نسخه پروداکشن اجرا شود. به نظرتان زمان خوبی است؟ اگر کدی که شما زده‌اید فقط یک خط باشد چه؟ واقعا فاجعه است! اما در میکروسرویس فاصله مرج تا دیپلوی می‌تواند حتی کمتر از یک دقیقه باشد. این یعنی چابکی به معنای واقعی کلمه!

3-3-3- هر سرویس به تنهایی قابل دیپلوی کردن است.

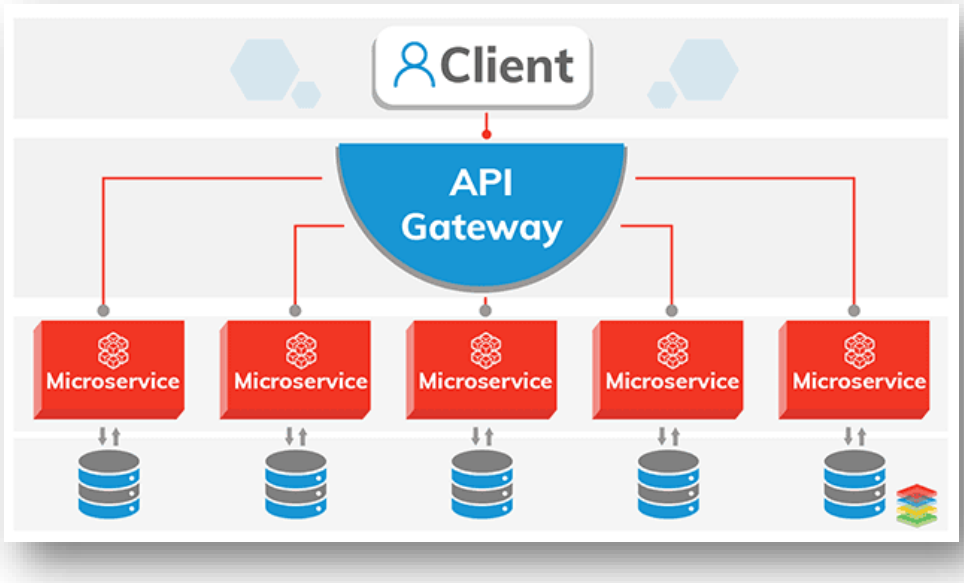
شما یک کدی را توسعه می‌دهید و انتظار دارید کارایی که می‌خواهید را داشته باشد و حتی برایش تست هم نوشته‌اید. اما از کجا می‌دانید که کد شما قسمت‌های مختلفی که سایر تیم‌ها روی آن‌ها کار می‌کنند تاثیر نمی‌گذارد آن‌ها را خراب نمی‌کند و برعکس. اما در میکروسرویس شما توسعه را در

سرویسی که برای خودتان است پیش می‌برید و تست‌های خودتان که پاس شد روی نسخه عملیاتی اجرایش میکنید. و حتی اگر خراب هم شود فقط روی این سرویس تاثیر می‌گذارد و کل سیستم مختل نمی‌شود.

3-3-4- هر سرویس به تنهایی قابل دیپلوی کردن است.

در حالت مونولوتیک اگر بار روی سرویس‌تان زیاد شود و بخواهید آن را حالا با هر روشی scale کنید، مجبور هستید برای همه سیستم این کار را انجام دهید. و این در حالی است که احتمال زیاد همه قسمت‌های آن مثلا همه apiها، بار زیادی ندارند و فی‌الواقع نیازی نیست که scale شوند.

ولی در معماری میکروسرویس‌ها شما می‌توانید هر سرویس را به طور جداگانه و در حد نیازش scale کرده و خروجی خوبی را به کاربران خود بدهید.



شکل ۳-1 نمونه‌ای نمای کلی یک معماری میکروسرویس.

کلا معماری میکروسرویس‌ها نکته‌های خوب زیاد دارد که در این حجم از کلام نمی‌گنجد و کتابی که در منابع معرفی کردم یکی از منابع خوب برای شناختن هر چه بهتر آن است.

اما در میان مزایا، این معماری مانند هر چیز دیگری معایبی را دارد. که به نظر من مهم‌ترین آن‌ها **پیچیدگی** است که به مجموعه اضافه می‌کند. از این جهت که اگر یک **کسب و کار نوپا** در همان اول کار دنبال میکروسرویس بودن باشد احتمال زیاد هم وقت زیادی صرف می‌کند و هم خطاها و شکست‌هایی در پی خواهد داشت. مهم‌ترین نکته در بین مرز معماری میکروسریس‌ها و مونولوتیک، شناختن و درک عمیق زمان درست مهاجرت از مونولوتیک به میکروسرویس‌ها است که اگر این زمان درست انتخاب شود، اتفاقات خوبی را برای یک کسب و کار نوپا از نظر فنی رقم می‌زند.

فصل چهارم

ارتباط بین میکروسرویس‌ها

ارتباط بین سرویس‌ها

در فصل قبل به معرفی و تعریف معماری میکروسرویس‌ها پرداختیم.

در این قسمت به نحوه ارتباط بین میکروسرویس‌ها می‌پردازیم.

این موضوع که سرویس‌هایی که در یک سیستم کار می‌کنند باید باهم در ارتباط باشند، امری بدیهی است. اما به طور کلی این ارتباط انواع و الگوهای خودش را دارد و لزوما هیچ‌کدام بر دیگری برتری نداشته و هر کدام بدی‌ها و خوبی‌های خود را دارند. نهایتا با تشخیص معمار نرم‌افزار که برحسب نیاز سیستم است، انتخاب و پیاده سازی می‌شوند.

به طور کلی ارتباط بین میکروسرویس‌ها از دو جنبه مختلف تقسیم‌بندی می‌شوند:

4-1- تقسیم‌بندی اول

4-1-1- ارتباط‌های همگام (Synchronous)

در حالت همگام، نحوه ارتباط به صورت request/response می‌باشد. سرویس‌ها براساس پروتکل‌هایی که توسعه داده شده‌اند، با هم ارتباط میگیرند. مثلا REST, Grpc, Graphql, ...

بدین صورت که یک سرویس در نقش کلاینت و سرویس دیگر در نقش سرور عمل می‌کنند. نکته مهم در این حالت این است که کلاینت منتظر جواب درخواستی که به سرور زده است می‌ماند و دیتایی که در جواب است، برایش مهم است. بنابراین در هنگام درخواست بلاک می‌شود.

4-1-2 ارتباط‌های ناهمگام

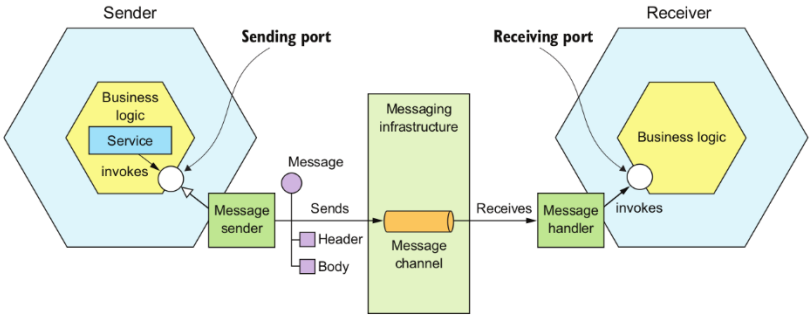
در این حالت نحوه ارتباط دو گونه است:

- به صورت request/response: که در این صورت کلاینت بازهم به همان روش‌های گفته شده در حالت قبل درخواست خود را به سرور ارسال می‌کند، ولیکن منتظر پاسخ نمی‌ماند و ادامه روندش را دارد. مثال‌های ضمنی این حالت اجرای یک دستور یا نوتیفیکیش در سرویس مقصد است که موفقیت یا عدم آن برای کلاینت اهمیتی ندارد.
- به صورت publish/subscribe: در این حالت یک Message Broker بین کلاینت و سرور قرار می‌گیرد و کلاینت پیام‌ها یا ایونت‌های خود را در آن publish کرده و سرور که روی آن subscribe کرده است، آن‌ها را خوانده و پردازش موردنظر خود را روی‌شان انجام می‌دهد. در این حالت نیز کلاینت بلاک نشده و پیام خود را انتشار داده و به کار خود ادامه می‌دهد.

4-2 تقسیم‌بندی دوم:

4-2-1 ارتباط‌های یک‌ب‌یک

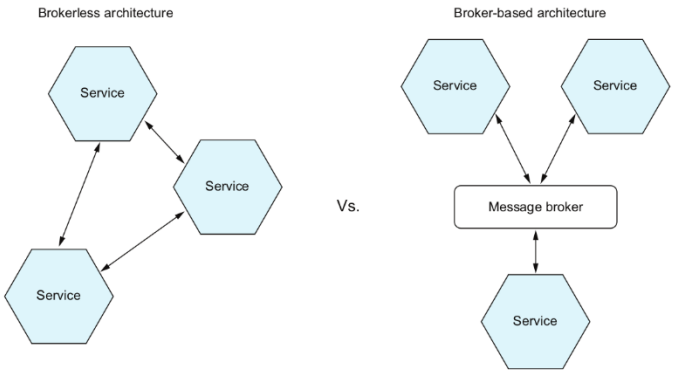
در این حالت یک سرویس با یک سرویس ارتباط می‌گیرد و می‌تواند هم همگام و به صورت request/responseباشد و هم ناهمگام که باز هم یا request/response یا به صورت Publish/subscribe باشد.



شکل 4-1 نمونه‌ای از ارتباط یک‌ب‌یک.

4-2-2- ارتباط‌های یک‌ب‌چند

در این نوع از ارتباط، دیگر مرسوم نیست که از حالت همگام و request/response استفاده شود. چرا که یک سرویس می‌خواهد با چند سرویس همزمان ارتباط برقرار کند. در این حالت، روش Publish/subscribeمناسب می‌باشد. به این صورت که سرویس مبدا، پیام خود را در Message Broker انتشار می‌دهد و سرویس‌های مقصد، هر کدام برحسب نیازشان پیام‌ها را خوانده و استفاده می‌کنند. توجه داشته باشید که این ارتباط هم می‌تواند به با یک Message Broker و هم میتواند با چند Message Channelصورت بگیرد. که هر کدام مزایا و معایب خودشان را دارند.



شکل 4-2 دو نوع ارتباط یک‌ب‌چند.

حالا برخی (نه همه) مزایا و معایب انواع ارتباطات گفته شده به صورت نکته‌وار می‌پردازیم.

- به صورت کلی برای **دردسترس بودن** (availability)بیشتر سیستم توصیه به استفاده از حالت ناهمگام در نحوه ارتباط‌ها توصیه می‌شود. چرا که در حالت همگام اگر یک سرویس از دسترس خارج شود، عملا کار سرویس کلاینت که منتظر جواب درخواستش است مختل شده و از دسترس خارج می‌شود. ولی در حالت ناهمگام این اتفاق نیوفتد و یک سرویس در درسترس نباشد و سرویس دیگر کار خودش را انجام دهد.

- یکی از معایب ارتباط یک به چند در حالتی که از Message Broker استفاده می‌شود، این است که ممکن است این MB به یک **گلوگاه** تبدیل بشود و از دسترس خارج شدن آن باعث مختل شدن کار همه سرویس‌هایی که به آن وابسته هستند بشود.

- در حالتی که شما از Message Broker استفاده می‌کنید، پیام‌هایی که در آن انتشار داده می‌شوند، تا زمانی که خوانده نشوند) .این قوانین درMB های مختلف متفاوت است (.حفظ می‌شود و عملا اگر سرویس استفاده کننده پیام‌ها از دسترس خارج شود و دوباره برگردد دیتایی را از دست نداده است. ولیکن اگر همین اتفاق برای حالت همگام اتفاق بیوفتد، یعنی درخواست از کلاینت ارسال شود و با ارور مواجه شود، یا سرور اصلا در درسترس نباشد، دیگر دیتا از بین رفته و قابلیت برگشت وجود ندارد. مگر اینکه در کلاینت نگه‌داری شود که اصلا توصیه به این کار نمی‌شود.

- در حالت Messaging شما از انعطاف‌پذیری خوبی برخوردار هستید. چون پیام‌ها ساختارهای متنوعی می‌توانند داشته باشند و حتی یک پیام می‌تواند توسط چند سرویس خوانده شده و استفاده شود.

- یکی از چالش‌های اصلی مطرح شده در حالت Messaging و ناهمگام وجود دارد، مدیریت خطاها و تراکنش‌هایی هست که اتفاق می‌افتد. مثلا رایت‌های دیتابیس و تغییراتی که ممکن است در یک سرویس اتفاق بیوفتد ولی در سرویس بعدی ادامه این کار انجام نشود و این سرویس متوجه آن نمی‌شود. برای این قضیه پترن‌هایی مانند Saga طراحی و مطرح شده است که انشالله در یک پست مجزا بررسی می‌شود.

فصل پنجم

راه انداختن سرویس‌های مانیتورینگ

مانیتورنگ یعنی چه؟

خیلی ساده بخواهم بگویم: همه ما می‌دانیم که وقتی سرویسی را توسعه می‌دهیم، قطعاً باید وضعیت آن سرویس همیشه جلوی چشممان باشد و اتفاق‌هایی را که برایش می‌افتد، رصد کنیم.

برای مثال، اگر یک وب سرویس داریم، باید بدانیم که در این لحظه میزان (rate) خطاهایش چقدر است یا اصلاً سرویس بالا هست یا نه!

شکی نیست که مانیتورینگ برای همهٔ سرویس‌ها ضروری است. حالا پرسش اینجاست که چطور سرویس خودمان را مانیتور کنیم؟

باید بگویم که خیلی ساده است؛ چراکه تاکنون شرکت‌های مختلف ابزارهای خیلی متنوعی را برای این کار توسعه داده‌اند.

یکی از این ابزارها که متن باز و رایگان است، prometheus نام دارد.

این ابزار به زبان Go توسعه داده شده و لینک سایت و گیت‌هابش اینجاست:

https://prometheus.io/

https://github.com/prometheus

کار اصلی این ابزار و احتمالاً خیلی از ابزارهای دیگر این است که برخی متریک‌هایی (metric) را که در روند توسعه کد سرویس تغییر داده‌ایم، بر روی نمودار نشان می‌دهند و تغییرات آن‌ها را نیز به نمایش می‌گذارند.

5-1- راه‌اندازی ابزار prometheus

برای این کار لازم است دانش اولیه‌ای درباره‌ی «داکر» داشته باشید؛ چون‌که می‌خواهم چگونگی بالا آوردن با داکرش را توضیح بدهم.

برای این کار اول از همه باید فایلی docker-compose.yaml مثل فایل زیر ایجاد کنیم:

```
version: "3"
services:
  prometheus:
    image: prom/prometheus:v2.12.0
    hostname: ""
    container_name: prometheus
    volumes:
    - ./volumes/static/entrypoint.sh:/bin/prometheus-entrypoint.sh
    - ./volumes/static/prometheus.yml:/etc/prometheus/prometheus.yml
    - ./volumes/dynamic/prometheus/:/var/lib/prometheus/data
    user: root
    entrypoint:
    - /bin/sh
    - /bin/prometheus-entrypoint.sh
    - --config.file=/etc/prometheus/prometheus.yml
    - --storage.tsdb.path=/var/lib/prometheus/data
    - --storage.tsdb.min-block-duration=1h
    - --storage.tsdb.max-block-duration=6h
    - --storage.tsdb.retention=7d
    - --web.console.libraries=/usr/share/prometheus/console_libraries
    - --web.console.templates=/usr/share/prometheus/consoles
    ports:
    - 9090:9090
```

اگر این را با کامند docker-compose up بالا بیاوریم، وب سرویس پرومتئوس توی پورت 9090 سرور بالا می‌آید. ولی الان مسئله این است که متریک‌ها را از کجا باید خواند؟ پس برای این هم باید کانفیگ‌هایی سِت شود:

بدین ترتیب که اول باید فایلی به اسم prometheus.yml ایجاد کنید با محتویات زیر:

```
global:
scrape_interval: 30s
scrape_configs
 - job_name: 'your_job
     static_configs:
       - targets: ['targer_host:target_port']
```

منظور از scrape_interval این است که پرومتئوسی که این‌طرف بالا آمده، چند ثانیه‌ای یک بار، برود و آن تارگت‌هایی (target) را که بهش می‌دهیم، بررسی کند.

منظور از job_name اسمی است که برای آن کار انتخاب می‌کنید و دلخواه شماست.

اما این هدف‌هایی (target) که برای کار خودتان تعریف می‌کنید، چه هستند؟ باید همان سروری را که متریک‌ها در آن ریخته شده‌اند، به آن بدهید.

اینجا پرانتزی باز می‌کنم و نکته‌ای را توضیح می‌دهم: دقت کنید که پرومتئوس اکسپورترهایی دارد که برایش توسعه داده شده‌اند و اینجا می‌توانید آن‌ها را ببینید. کاری که اکسپورترها انجام می‌دهند، این است که متریک‌های مدنظر خودشان را توی سروری می‌ریزند و می‌شود اینجا توی target های پرومتئوسی که می‌سازیم، آدرس آن سرورها را بهش بدهیم و آن‌ها را توی داشبوردمان نگه داریم.

بعد یک فایل entrypoint.sh با محتویات زیر ایجاد کنید:

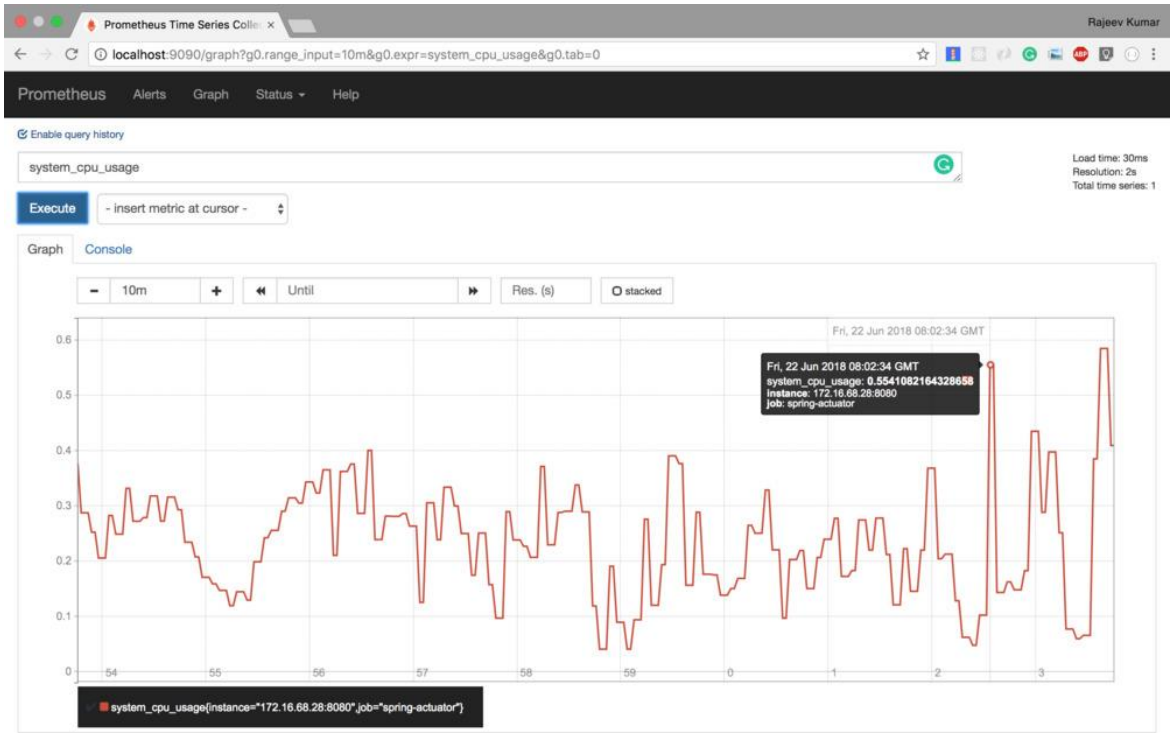
```
#!/bin/sh
#
# Entrypoint that adds `host.docker.internal` for Linux
# https://github.com/docker/for-linux/issues/264
HOST_DOMAIN="host.docker.internal"
ping -q -c1 $HOST_DOMAIN > /dev/null 2>&1
if [ $? -ne 0 ]; then
 HOST_IP=$(ip route | awk 'NR==1 {print $3}')
  echo -e "$HOST_IP\t$HOST_DOMAIN" >> /etc/hosts
  fi
  exec /bin/prometheus "$@"
```

بعد از آنکه این دو تا فایل را ایجاد کردید، دستورهای زیر را اجرا کنید: (دقت کنید که فرض بر این است که الان توی مسیری هستیم که فایل docker-compose.yaml بالا را گذاشتیم.)

```
sudo mkdir -p ./volumes/dynamic/
sudo mkdir -p ./volumes/static/
sudo cp prometheus.yml ./volumes/static/
sudo cp entrypoint.sh ./volumes/static/
```

بعد از این کار با دستور زیر داکرتان را اجرا کنید و بروید روی پورت 9090 (اگر توی داکر عوضش نکرده‌اید.) سرورتان را ببینید که بالا آمده است.

```
docker-compose up -d
```



شکل 5-1 نمونه‌ای از سرویس پرومئوس.

5-2- راه‌اندازی سرویس گرافانا

گرافانا ابزار متن بازی است که متریک‌ها را می‌خواند و آن‌ها را خیلی ساده و روشن در قالب نمودارهایی کارآمد و جذاب نمایش می‌دهد. برای آشنایی با نحوهٔ به‌کارگیری و راه‌اندازی آن اینجا را بخوانید.

خیلی ساده بخواهم بگویم، ابزار متن بازی (گیت هاب) است برای نمایش هرچه بهتر متریک‌ها.

گرافانا متریک‌ها را از برخی منابع (مثلاً پرومتئوس که در نوشتهٔ قبلی توضیح دادم) که خودمان به آن می‌دهیم، می‌خواند و می‌توان در آن نمودارهای خیلی متنوع و درعین‌حال خوشگلی :) کشید.

ازآنجاکه گرافانا ابزار متن باز است، پس یک docker image از آن در داکر هاب پیدا می‌شود.

حالا تنها کاری که لازم است بکنیم، این است که کانفیگ‌هایش را برای این ایمیج سِت کنیم و در سرورمان یا هر جای دیگری دلمان خواست، آن را بالا بیاوریم.

مراحل بالاآوردن گرافانا با داکر:

- مرحله اول:

باز هم مثل قبل یک فایل docker-compose.yaml داریم که باید آن را بالا بیاوریم. محتوای این فایل این است:

```
version: '3.1'
services:
  grafana:
    image: grafana/grafana:6.4.4
    container_name: grafana
  environment:
    - GF_SERVER_ROOT_URL=YOUR_SERVER_IP
    - GF_SECURITY_ADMIN_PASSWORD=asadasad
    - TZ=Asia/Tehran
  volumes:
    - ./volumes/dynamic/data:/var/lib/grafana
    - ./volumes/static:/etc/grafana
    - /var/log/docker/grafana-babr:/var/log/grafana
  ports:
    - 3000:3000
  user: "0"
```

خب مشخص است که اول می‌گوید: «برو ایمیج گرافانا رو بگیر و بیار و این environment_variable ها رو به آن بده.». بعد برخی چیزهایی را که لازم است، volume می‌کند.

همین جا توضیح می‌دهم که volume چیست؟ در هر کانتینر داکر که در حال اجرا شدن باشد، دیتاهایی وجود دارد که اگر کانتینر را پایین بیاوریم، آن‌ها پاک می‌شوند. برای اینکه جلوی این اتفاق گرفته شود، آن‌ها را اصطلاحاً volume می‌کنند. مثلاً می‌گویند فلان فایل‌ها توی فلان مسیر از سیستم ذخیره بشود که بعداً که کانتینر ریست شد، برود و دوباره آن‌ها را از آنجا بخواند.

خب بعدش هم می‌گوید که روی چه پورتی بالا بیاید و آن را bind می‌کند به پورت سرور و تمام!

- **مرحله دوم:**

فایلی هم به اسم grafana.ini هست که برای کانفیگ‌های بیشتر گرافانا استفاده می‌شود که چون خیلی حجیم است و قرار هم نیست برای کاربردهای ساده بخشی از آن را تغییر دهیم، از اینجا می‌توانید به آن دسترسی بیابید.

حالا که آن فایل را هم داریم، دستورات زیر را اجرا کنید:

```
sudo mkdir -p ./volumes/static/
sudo cp grafana.ini ./volumes/static/
docker-compose up -d
```

خب این‌ها هم که دیگر خودتان می‌دانید چی هستند! فایل‌هایی که باید در جای خودشان باشند.

باز هم تأکید می‌کنم که الان ما توی مسیری هستیم که فایل docker-compose.yaml وجود دارد.

خب الان گرافانای شما باید بالا آمده باشد. بروید توی localhost:3000 ببینیدش.

کاری که باید بکنید، این است که لاگین کنید (توی فایل docker-compose.yaml تعیین کردیم که یوزر و پسورد چیست!) و به بخش تنظیمات و قسمت اضافه‌کردن داشبورد جدید بروید و آنجا سورسش را پرومتئوسی بدهید که قبلاً در راه انداخته‌ایم.

شکل 5-2 نمونه‌ای از محیط گرافانا.

فصل ششم

جمع‌بندی و نتیجه‌گیری و پیشنهادات

جمع‌بندی و نتیجه‌گیری

با توجه به همه‌گیری و پیشرفت مهندسی نرم‌افزار یک شرکت برای اینکه بتواند در این حوزه حرفی برای گفتن داشته باشد، همواره باید در مسیر پیشرفت و یادگیری مطالب جدید باشد. این محیط یادگیری در شرکت بله کاملا فراهم است و کارمندان شرکت همواره در تکاپوی یادگیری و به‌کارگیری تکلونوژی‌ها و الگوریتم‌های جدید برای سرویس‌دهی هر چه بهتر هستند.

نهایتا یادگیری روش‌ها و الگوهای مهندسی نرم‌افزاری امری مهم در این زمینه به شمار می‌رود.

پیشنهادات

در انتها پیشنهاد می‌دهیم علاوه‌بر تقویت مهارت‌های فنی خود، همواره در تلاش برای تقویت مهارت‌های نرم مانند: کارتیمی، تصمیم‌گیری در مواقع حساس و آرامش و ... که بسیار مهم هستند، باشید.

به‌خصوص مهارت کارتیمی متاسفانه در سیستم آموزشی ما به خوبی آموزش داده نمی‌شود. و دانشجو ناگزیر است با تجربه و خودآموزی این مهارت‌ها را به دست آورد.

منابع و مراجع

https://www.manning.com/books/microservices-patterns

https://microservices.io/

https://www.docker.com/

https://prometheus.io/

https://grafana.com/

https://bale.ai/