دانشکده مهندسی کامپیوتر

پیاده‌سازی درگاه ارتباط با رابط‌های برنامه‌نویسی

پایان‌نامه برای دریافت درجه‌ی کارشناسی در رشته‌ی مهندسی کامپیوتر

سعید طهماسبی ثالث

استاد راهنما:

دکتر وصال حکمی

شهریور ۱۳۹۹

تأییدیه‌ی هیأت داوران جلسه‌ی دفاع از پایان‌نامه

نام دانشکده: مهندسی کامپیوتر

نام دانشجو: سعید طهماسبی ثالث

عنوان: پیاده‌سازی درگاه ارتباط با رابط‌های برنامه‌نویسی

تاریخ دفاع: شهریور ۱۳۹۹

رشته: مهندسی کامپیوتر

ردیف	سمت	نام و نام خانوادگی	مرتبه دانشگاهی	دانشگاه یا مؤسسه	امضا
۱	استاد راهنما	دکتر وصال حکمی	استادیار	دانشگاه علم‌وصنعت ایران	
۲	داور داخلی				

تأییدیه‌ی صحت و اصالت نتایج

باسمه تعالی

اینجانب سعید طهماسبی ثالث به شماره دانشجویی ۹۵۵۲۱۳۱۵ دانشجوی رشته مهندسی کامپیوتر مقطع تحصیلی کارشناسی تأیید می‌نمایم که کلیه‌ی نتایج این پایان‌نامه حاصل کار اینجانب و بدون هرگونه دخل و تصرف است و موارد نسخه‌برداری‌شده از آثار دیگران را با ذکر کامل مشخصات منبع ذکر کرده‌ام. درصورت اثبات خلاف مندرجات فوق، به تشخیص دانشگاه مطابق با ضوابط و مقررات حاکم (قانون حمایت از حقوق مؤلفان و مصنفان و قانون ترجمه و تکثیر کتب و نشریات و آثار صوتی، ضوابط و مقررات آموزشی، پژوهشی و انضباطی) با اینجانب رفتار خواهد شد و حق هرگونه اعتراض درخصوص احقاق حقوق مکتسب و تشخیص و تعیین تخلف و مجازات را از خویش سلب می‌نمایم. در ضمن، مسؤولیت هرگونه پاسخگویی به اشخاص اعم از حقیقی و حقوقی و مراجع ذی‌صلاح (اعم از اداری و قضایی) به عهده‌ی اینجانب خواهد بود و دانشگاه هیچ‌گونه مسؤولیتی در این خصوص نخواهد داشت.

نام و نام خانوادگی: سعید طهماسبی ثالث

تاریخ و امضا:

مجوز بهره‌برداری از پایان‌نامه

بهره‌برداری از این پایان‌نامه در چهارچوب مقررات کتابخانه و با توجه به محدودیتی که توسط استاد راهنما به شرح زیر تعیین می‌شود، بلامانع است:

بهره‌برداری از این پایان‌نامه برای همگان بلامانع است.

بهره‌برداری از این پایان‌نامه با اخذ مجوز از استاد راهنما، بلامانع است.

بهره‌برداری از این پایان‌نامه تا تاریخ ..... ممنوع است.

نام استاد راهنما: دکتر وصال حکمی

تاریخ:

امضا:

چکیده

با توجه به رشد روزافزون کسب و کارهای مجازی و نیاز به ثبات و کارایی بالا در تامین درخواست‌های مشتریان، معماری‌های جدیدی برای توسعه نرم‌افزار مورد استفاده قرار می‌گیرند. ویژگی اصلی این معماری‌ها، مقیاس‌مندی و تاب‌آوری بالا برای پاسخ به مشتریان است.

معماری میکروسرویس، یکی از محبوب‌ترین معماری‌های جدید است. تمرکز این معماری بر مقیاس‌مندی، گسترش‌پذیری و عدم وابستگی خدمت‌های مختلف یک سامانه به یکدیگر است. نحوه‌ی دستیابی این معماری به ویژگی‌های ذکر شده، جداسازی عملکردهای مختلف سامانه به چند قسمت و اجرای خودمختار هر یک از این قسمت‌ها در یک محیط جداشده [1] است.

همانند بسیاری از معماری‌های دیگر، پیاده‌سازی درست معماری میکروسرویس، دارای چالش‌های مختلفی است. با توجه به ماهیت غیر متمرکز سامانه‌ها در این معماری، اتصال کاربران به خدمت‌های ارائه‌شده به سادگی گذشته نخواهد بود. زیرا کاربر برای استفاده از خدمت‌های مختلف نیاز به دریافت اطلاعات از قسمت‌های مختلف سامانه را دارد. از طرفی تغییر روش ارتباط کاربران با سامانه و به‌روزرسانی نرم‌افزارهای سمت کاربر، بسیار پرهزینه خواهد بود.

الگوی درگاه ارتباط با رابط‌های برنامه‌نویسی جهت حل این چالش طراحی شده است. در این الگو، کاربران همانند سابق درخواست‌های خود را تنها به یک سامانه‌ی میانی ارسال می‌کنند. وظیفه‌ی این سامانه‌ی میانی، دریافت درخواست و هدایت آن به سمت خدمت‌های مرتبط به درخواست است.

علی‌رغم وجود درگاه‌های ارتباط متن‌باز و تجاری مختلف، اکثر آن‌ها گسترش‌پذیری و قابلیت پیکربندی ضعیفی دارند. از این‌رو پیاده‌سازی یک درگاه ارتباط که خلاهای موجود را پوشش دهد، تمرکز اصلی این پروژه است.

کلید واژه‌ها: درگاه ارتباط با رابط‌های برنامه‌نویسی، معماری میکروسرویس، سامانه‌های غیر متمرکز، سامانه‌ی گسترش‌پذیر

[1] Isolated

فهرست مطالب

فهرست تصاویر

فهرست جداول

۱ مقدمه

۱.۱ تاریخچه

گسترش روزافزون راه‌های ارتباطی از طریق شبکه و اینترنت، باعث ایجاد کسب و کارهای بسیاری شده است. این فعالیت‌ها در حوزه‌های مختلفی از جمله درخواست خدمات حضوری، ارائه‌ی خدمات مجازی، شبکه‌های اجتماعی و ارتباطی و ... انجام می‌شوند.

با توجه به محبوبیت روزافزون خدمات مجازی در بین مردم، کسب و کارها نیز بزرگ و بزرگ‌تر می‌شوند. از این رو، ارائه‌ی خدمات مطمئن، سریع و با کیفیت مطلوب، بخشی از اولویت‌های صاحبان کسب و کارهاست.

در ابتدای گسترش فناوری‌های مربوط به خدمات مجازی، سامانه‌های یکپارچه [۱] و متمرکز [۲]، به راحتی جواب‌گوی درخواست‌های مشتریان بودند. ولی با توجه به رشد بسیار سریع تعداد مشتریان، راه‌حل‌های یکپارچه و متمرکز، کارایی خود را از دست دادند.

از طرفی، با توجه به پیچیده‌شدن نرم‌افزارهای یکپارچه در طول زمان، گسترش گروه‌های مهندسی جهت ارائه‌ی خدمات بیش‌تر و ایجاد تغییرات در نرم‌افزار، سخت و سخت‌تر می‌شود.

لذا اولین اقدام جهت تاب‌آوری میزان بار و درخواست مشتریان، سعی در بهینه‌سازی نرم‌افزارها و سخت‌افزارها در تمام سطوح ممکن است. این بهینه‌سازی‌ها اغلب بسیار پیچیده‌اند. به طوری که هزینه‌ی نیروی انسانی جهت این فعالیت‌ها، اغلب بسیار بیش‌تر از سود کسب‌شده در قبال آن‌هاست.

۲.۱ معماری‌های جدید توسعه‌ی نرم‌افزار

مطرح‌شدن معماری‌های جدید توسعه نرم‌افزار، از جمله معماری مبتنی بر خدمت [۳] ، معماری میکروسرویس [۴] و ... سعی در برطرف کردن این مشکلات دارند. مهم‌ترین ویژگی این معماری‌ها، توزیع‌پذیری، گسترش‌پذیری و غیر متمرکز بودن است.

هر کدام از معماری‌های مطرح شده، دارای مزایا و معایب و همچنین آسانی‌ها و سختی‌های مختص به خود هستند.

امروزه، معماری میکروسرویس در صنعت بسیار مورد استفاده قرار می‌گیرد. به طوری که بسیاری از کسب و کارهای بزرگ داخلی و خارجی از این معماری جهت توسعه محصول استفاده می‌کنند. همان‌طور که گفته شد این معماری دارای مزایا و معایب

[1] Monolithic
[2] Centralized
[3] Service Oriented
[4] Microservice

مختلفی است.

بررسی دقیق این معماری در قالب این گزارش نمی‌گنجد ولی جهت شفافیت هرچه بیش‌تر موضوع، در ادامه به برخی از معایب این معماری پرداخته شده است.

۳.۱ معایب معماری میکروسرویس

۱.۳.۱ افزایش هزینه‌های توسعه و نگه‌داری

با توجه به پیچیدگی‌های این معماری نسبت به سامانه‌های یکپارچه، زمان و هزینه، جهت تولید نرم‌افزارهایی که طبق این معماری ساخته شده باشند افزایش می‌یابد.

از طرفی، به دلیل ماهیت غیر متمرکز این معماری، نگه‌داری و تعمیرات نرم‌افزارهای مبتنی بر این معماری پرچالش‌تر و گران‌تر خواهد بود.

۲.۳.۱ بستر غیر قابل اطمینان شبکه

اجزای مختلف نرم‌افزارهای پیاده‌سازی شده بر اساس معماری میکروسرویس، نیازمند ارتباط با یکدیگر هستند. شبکه‌های کامپیوتری یکی از محبوب‌ترین و پراستفاده‌ترین بسترهای ارتباطی برای این منظور است. ولی با توجه به ماهیت این شبکه‌ها، نمی‌توان اطمینان کامل از برقراری ارتباط بین قسمت‌های مختلف نرم‌افزار حاصل کرد. به همین دلیل پیچیدگی‌هایی، جهت ایجاد اطمینان در عملکرد نرم‌افزارها، در پیاده‌سازی نرم‌افزار ایجاد می‌شود.

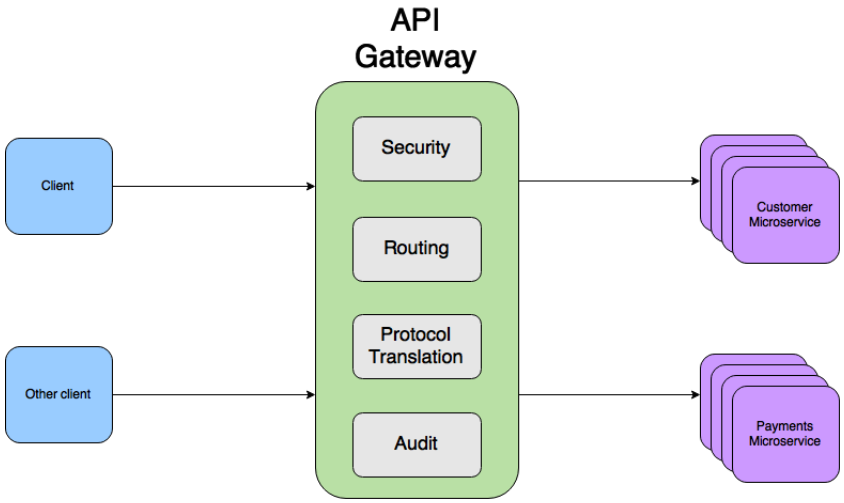
۳.۳.۱ سختی ارتباط کاربران با خدمات ارائه‌شده

در سامانه‌های یکپارچه، ارتباط کاربران و کسب و کار، از طریق یک راه ارتباط و تنها دو دستگاه برای کاربر و صاحب کسب و کار صورت می‌گرفت. ولی با توجه به ماهیت غیر متمرکز معماری میکروسرویس، کاربران باید با استفاده از چند راه ارتباطی، خدمات مورد نیاز خود را دریافت کنند. از طرفی به‌هنگام‌سازی دستگاه کاربر جهت ارتباط از طریق چند راه ارتباطی هزینه‌های هنگفتی را در بردارد. در این گزارش قصد بر آن است که یک راه حل برای این مشکل یافت‌شود و پس از پیاده‌سازی، نتایج آن بررسی شود.

۴.۱ الگوی درگاه ارتباط با رابط‌های برنامه‌نویسی [1]

این الگو یک الگوی مطرح برای حل مشکل ذکر شده است. [۱] فسلفه‌ی این الگو بر این مبنا است که کاربران، مطابق گذشته و معماری نرم‌افزارهای یکپارچه، تنها از طریق یک راه ارتباطی با یک سامانه‌ی میانی ارتباط برقرار کنند. وظیفه‌ی این سامانه، دریافت اطلاعات از بخش‌های مختلف به وسیله‌ی رابط‌های برنامه‌نویسی فراهم‌شده توسط هر قسمت، و منتقل کردن آن به کاربر است. طرح کلی این الگو مانند شکل ۱ است.

شکل ۱: طرح کلی الگوی درگاه ارتباط با رابط‌های برنامه‌نویسی



[1] API Gateway

۲ مروری بر منابع

۱.۲ مقدمه

در ابتدا ویژگی‌های یک درگاه ارتباط مناسب بررسی شده و سپس برخی از درگاه‌های شناخته‌شده مورد بررسی قرار می‌گیرند که چه میزان از ویژگی‌های یک درگاه ارتباط مناسب را دارا هستند و در نهایت دلایل برتری راه‌حل پیشنهادی به دیگر درگاه‌ها ذکر شده است.

۱.۱.۲ درگاه ارتباط مناسب

مهم‌ترین ویژگی‌های یک درگاه ارتباط مناسب عبارتند از:

- مسیردهی [۱]
- قابلیت پیکربندی با روش‌های مختلف
- مقیاس‌مندی [۲]
- گسترش‌پذیری [۳]
- توازن بار [۴]
- ثبت رخدادها [۵]
- رصد وضعیت سامانه [۶]

[1] Routing
[2] Scalability
[3] Extendability
[4] Load Balancing
[5] Logging events
[6] Monitoring

۲.۱.۲ مسیردهی

اساسی‌ترین نیاز یک درگاه ارتباطی، مسیردهی با استفاده از مولفه‌های مختلف درخواست از جمله نوع پروتکل، مسیر درخواست، سرتیترهای درخواست و... است. تنوع پشتیبانی از مولفه‌های مختلف در مسیردهی باعث می‌شود که درخواست‌ها دقیق‌تر به خدمت موردنظر برسند.

۳.۱.۲ قابلیت پیگربندی با روش‌های مختلف

با توجه به محیط‌های مختلف برای بارگذاری نرم‌افزارها و تغییر پی‌درپی نیازها، درگاه ارتباط باید اطلاعات موردنیاز برای مسیردهی به درخواست‌ها را از روش‌های مختلف کشف و دریافت کند. در این‌صورت در صورت اضافه و یا حذف کردن یک خدمت، نیازی به تغییر در پیکربندی نبوده و درگاه پیکربندی خود را با تغییرات جدید به روز می‌کند.

۴.۱.۲ مقیاس‌مندی

با بالا رفتن تعداد درخواست‌ها ممکن است که درگاه ارتباط تبدیل به تک نقطه‌ی شکست [1] شود. با قابلیت مقیاس‌مندی این امکان وجود دارد که به صورت هم‌زمان چند درگاه ارتباط مختلف را اجرا کرد و هرکدام از این درگاه‌ها به صورت مستقل به درخواست‌ها پاسخ دهند.

۵.۱.۲ گسترش‌پذیری

درگاه‌های ارتباط، معمولا به صورت پیش‌فرض طیف نیازهای عمومی تمام سامانه‌ها را پوشش می‌دهند. یکی از قابلیت‌های مناسب برای درگاه ارتباط، امکان گسترش‌پذیری و اضافه کردن نیاز مختص به یک سامانه به درگاه ارتباط است.

۶.۱.۲ توزان بار

برای جلوگیری از ایجاد تک نقطه‌ی شکست در معماری میکروسرویس، چند نسخه از یک خدمت اجرا می‌شود. درگاه ارتباط باید بتواند با روش‌های مختلف، بار ورودی به سامانه را بین این نسخه‌ها تقسیم کند.

[1] Single Point of Failure

۷.۱.۲ ثبت رخدادها و رصد وضعیت سامانه

در معماری میکروسرویس یک سامانه از اجزای مستقل و مختلفی تشکیل شده است که هرکدام وظیفه‌ی مشخصی دارند. یکی از چالش‌های موجود در این معماری رصد کردن خدمت‌های مختلف است. با توجه به اینکه درگاه ارتباط با تمام خدمت‌های یک سامانه در ارتباط است می‌تواند وضعیت تمام خدمت‌ها را در قالب‌های مختلف گزارش کند.

۲.۲ مروری بر ادبیات موضوع

در این بخش دو درگاه ارتباط محبوب و پرکاربرد Traefik و Kong مورد ارزیابی واقع می‌شوند و مزایا و معایب هر یک مشخص می‌شود.

۱.۲.۲ درگاه ارتباط Traefik

درگاه ارتباط Traefik یکی از محبوب‌ترین درگاه‌های ارتباط متن‌باز است که اکثر ویژگی‌های یک درگاه ارتباط مناسب را دارا است. این درگاه ارتباط، با زبان برنامه‌نویسی Golang توسعه داده شده است که این امر باعث بهینگی و عملکرد مناسب آن شده است.

مهم‌ترین ویژگی Traefik، سازگاری آن با محیط‌های مختلف بارگذاری نرم‌افزار است. این درگاه ارتباط، به صورت خودکار پیکربندی تمام خدمات را از محیط نرم‌افزار استخراج می‌کند که این امر باعث سادگی در استفاده از آن می‌شود.

بزرگ‌ترین مشکل Traefik عدم گسترش‌پذیری آن است. به صورتی که در صورت نیاز به اضافه کردن یک بخش اختصاصی به آن، راهی به غیر از تغییر در متن وجود ندارد.

۲.۲.۲ درگاه ارتباط Kong

Kong درگاه ارتباطی است که بر روی Nginx استوار شده است و در دو نسخه‌ی متن‌باز و تجاری قابل استفاده است. این درگاه ارتباط با استفاده از Nginx به عنوان هسته‌ی خود، قابلیت اتکای بالایی دارد و از تمام مزایای آن استفاده می‌کند.

یکی از مهم‌ترین ویژگی‌های Kong، رابط کاربری و پنل مدیریت آن است، که تغییرات در هنگام اجرا را میسر می‌سازد. پنل مدیریت Kong از ویژگی‌های نسخه‌ی تجاری آن است.

این درگاه ارتباط، با زبان برنامه‌نویسی Lua توسعه داده شده است که با توجه به محبوبیت کم‌تر این زبان، باعث شده است

جامعه‌ی برنامه‌نویسی کم‌تر به سراغ توسعه‌ی آن برود. از طرفی گسترش‌پذیری Kong، بدون تغییر در متن آن، امری ممکن اما دشوار است.

۳.۲ نتیجه‌گیری

با توجه به بررسی‌های انجام شده، خلا یک درگاه ارتباط با قابلیت گسترش‌پذیری بالا حس می‌شود. معماری این درگاه باید به صورتی باشد که بتوان بدون نیاز به ایجاد تغییر در متن، منطق خاص مورد استفاده در یک سامانه را به آن اضافه کرد.

هم‌چنین با توجه به تنوع در محیط‌های بارگذاری سامانه‌های با معماری میکروسرویس، درگاه ارتباط باید بتواند با اجزاهای محیطی مختلف ارتباط برقرار کرده و تنظیمات مربوط به خدمت‌های مختلف را از این محیط‌ها دریافت کند. نمونه‌هایی از محیط‌های مختلف عبارتند از سکوی [1] Docker، Kubernetes و... .

درگاه ارتباط پیاده‌سازی شده با تمرکز بر گسترش‌پذیری و قابلیت پیکربندی به عنوان شاخص‌های اصلی نسبت به سایر درگاه‌های موجود پیاده‌سازی شده است. جزئیات مربوط به نحوه‌ی پیاده‌سازی این ویژگی‌ها در فصل ۳ آمده است.

مقایسه‌ی درگاه‌های ارتباط مختلف، در جدول ۱ قابل مشاهده است.

جدول ۱: مقایسه‌ی درگاه‌های ارتباط مختلف

	Traefik	Kong	درگاه ارتباط پیشنهادی
مسیردهی	بالا	بالا	بالا
قابلیت پیکربندی با روش‌های مختلف	بالا	متوسط	بالا
مقیاس‌مندی	بالا	بالا	بالا
گسترش‌پذیری	پایین	متوسط	بالا
توازن بار	بالا	بالا	متوسط
ثبت رخدادها	بالا	بالا	متوسط
رصد وضعیت سامانه	بالا	متوسط	بالا
سادگی استفاده	بالا	متوسط	بالا

[1] Platform

۳ روش پیاده‌سازی

۱.۳ مقدمه

برای پیاده‌سازی یک سامانه، می‌توان دو رویکرد داشت. می‌توان از سامانه‌های از پیش موجود استفاده کرد و با حاصل کردن تغییرات لازم به مقصود رسید. و یا می‌توان با پیاده‌سازی سامانه از ابتدا، نیازهای را حل کرد.

از طرفی با توجه به ویژگی‌های ذکرشده در فصل ۲ ، راه حل انتخابی برای پیاده‌سازی یک درگاه ارتباط با رابط‌های برنامه‌نویسی باید معیارهای زیر را مدنظر خود قرار داده باشد:

- سرعت بالا
- کارایی بالا
- بهینه‌بودن راه‌حل
- گسترش‌پذیری
- تغییرپذیری
- سادگی و قابل فهم بودن

۲.۳ انتخاب روش و فناوری

استفاده از فناوری‌های موجود، مانند Nginx، HAProxy و ... و ایجاد تغییرات در آن‌ها جهت بر طرف کردن نیازهای مطرح شده در فصل ۲ ، می‌تواند یکی از راه‌حل های حل مسئله باشد. ولی با توجه به پایه‌ی پیاده‌سازی این محصولات بر اساس نیازهای گذشته، و همچنین عدم سازگاری این محصولات با محیط‌های ابری، جهت استفاده به عنوان درگاه ارتباط با رابط‌های برنامه‌نویسی، مشکلات زیادی برای حل مسئله وجود خواهد آمد. برخی از این مشکلات عبارتند از:

- اجبار به استفاده از راه‌حل‌های موقتی برای سازگاری این محصولات با فضای موجود
- پیچیدگی بیش از حد به دلیل وجود ویژگی‌های مرتبط با حوزه‌های دیگر

- احتمال عدم سازگاری تغییرات مورد نظر با ویژگی‌های اصلی محصول
- و ...

از این رو پیاده‌سازی یک درگاه، از ابتدا و با توجه به نیازهای جدید، راه حل منطقی تری به نظر می‌رسد.

با توجه به نیاز سامانه به دسترسی‌های سطح پایین، جهت ارتباط با لایه‌ی هفتم شبکه، و همچنین نیاز به سرعت بالا برای کنترل بار، زبان‌های برنامه‌نویسی محدودی مناسب پیاده‌سازی این محصول خواهند بود. برخی از این گزینه‌ها عبارتند از:

- زبان برنامه‌نویسی کامپایلری C/C++
- زبان برنامه‌نویسی کامپایلری Golang
- زبان برنامه‌نویسی مفسری Lua
- زبان برنامه‌نویسی Javascript (بستر Node.js)

معمولا زبان‌های برنامه‌نویسی سطح پایین‌تر، سرعت بالاتری نیز دارند. از این رو زبان‌های C و C++ از بالاترین سرعت برخوردار هستند. با این وجود، کاری‌هایی مانند مدیریت حافظه، مدیریت پردازه‌های هم‌روند و ... به عهده‌ی برنامه‌نویس است. این امر باعث پیچیدگی توسعه خواهد بود. از این رو، زبان برنامه‌نویسی Golang، که یک زبان برنامه‌نوسی سطح پایین محسوب می‌شود، به علت مدیریت حافظه خودکار و استفاده‌ی ساده از پردازه‌ها [1] و ریسمان‌ها [2] برای اجرای فرآیندهای هم‌روند، می‌تواند گزینه‌ی مناسبی جهت پیاده‌سازی باشد.

از زبان‌های مفسری نیز می‌توان برای توسعه‌ی نرم‌افزارهایی که در زمان اجرا نیاز به تغییر در خود دارند، استفاده کرد. ولی نقطه‌ی تاریک استفاده از زبان‌های برنامه‌نویسی مفسری، احتمال بالای رخداد خطاهای زمان اجرا خواهد بود. این ویژگی باعث از دست رفتن ثبات سامانه می‌شود.

نحوه‌ی انجام فرآیندهای هم‌روند نیز در انتخاب فناوری پیاده‌سازی این بخش از سامانه بسیار تاثیرگذار است. زیرا با توجه به نیاز به توان عملیاتی بالا، فناوری‌های تک رشته‌ای و یا تک پردازه‌ای باعث کاهش توان عملیاتی سامانه خواهند شد.

جدول ۲ شامل مقایسه‌ی زبان‌های برنامه‌نویسی ذکر شده با توجه به معیارهای ذکر شده است.

[1] Process
[2] Thread

جدول ۲: مقایسه‌ی زبان‌های برنامه‌نویسی جهت پیاده‌سازی درگاه ارتباط با رابط‌های برنامه‌نویسی

زبان برنامه‌نویسی	سرعت	سادگی	بهینگی	توان عملیاتی	مدیریت حافظه	مدیریت فرآیندهای هم‌روند
C/C++	بسیار بالا	پایین	بسیار بالا	بسیار بالا	خیر	بسیار سخت
Golang	بالا	متوسط	بالا	بالا	بله	آسان
Lua	بالا	متوسط	بالا	متوسط	بله	سخت
Javascript	متوسط	بالا	پایین	متوسط	بله	سخت

با توجه به جدول ۲ ، روش انتخابی برای حل مسئله، پیاده‌سازی سامانه از ابتدا و با استفاده از زبان برنامه‌نویسی Golang است. زیرا با اینکه سرعت کمتری نسبت به زبان برنامه‌نویسی C یا C++ دارد، ولی با توجه به عدم نیاز به مدیریت حافظه، مدیریت آسان‌تر فرآیندهای هم‌روند و همچنین سادگی و بهینگی قابل قبول، می‌توان از اختلاف سرعت این دو زبان برنامه‌نویسی چشم پوشی کرد.

۳.۳ پیاده‌سازی

۱.۳.۳ ساختار

ساختار برنامه باید به شکلی تعیین شود که برنامه‌نویس را در روند توسعه برنامه محدود نکند. علاوه بر این، یک ساختار خوب می‌تواند زمینه را جهت معماری گسترش‌پذیر و منعطف فراهم کند.

یکی از ساختارهای مناسب، در محیط توسعه‌ی Golang ، ساختار بسته مبنا است. این ساختار توسط بسیاری از محصولات صنعتی و آکادمیک استفاده شده است. همچنین برخی از ابزارهای از پیش‌آماده در محیط توسعه‌ی این زبان، از سازگاری کامل با این ساختار برخوردار هستند.

۲.۳.۳ ساختار بسته‌مبنا [1]

این ساختار، شامل سه پوشه‌ی اصلی cmd، internal و pkg است. پوشه‌های config و logging از جمله پوشه‌های دیگر این ساختار هستند که استفاده از آن‌ها توصیه شده است.

پوشه‌ی cmd حاوی برنامه‌ی اصلی و قابل اجرا است. این پوشه معمولا دارای برنامه‌های کوتاه و با منطق ساده هستند.

[1] Package Oriented

پوشه‌ی pkg حاوی بسته‌های مستقل و قابل استفاده توسط سایر برنامه هاست. بسته‌های موجود در این پوشه به تنهایی قابل استفاده اند و وابستگی آن‌ها به دیگر بسته ها بسیار کم است.

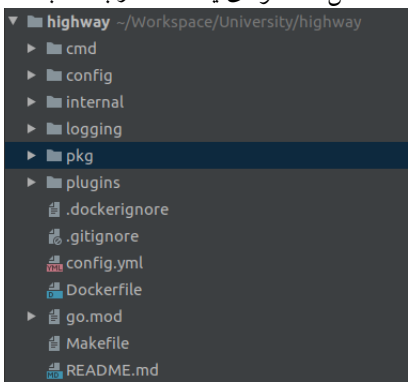
پوشه‌ی internal دارای بسته‌هایی است که از بسته‌های موجود در pkg استفاده کرده و منطق اصلی برنامه را پیاده‌سازی می‌کند. در این پوشه، رابط‌های ورودی و خروجی برنامه تعیین می‌شود.

بسته‌ی logging شامل برنامه‌های مربوط به ثبت اتفاقات سامانه است.

بسته‌ی config نیز دربردارنده‌ی برنامه‌های مربوط به پیکربندی و تنظیمات اجرای برنامه است. نحوه‌ی بارگذاری پیکربندی‌ها و همچنین استفاده از تنظیمات در قسمت‌های مختلف برنامه در این قسمت پیاده‌سازی می‌شود.

نمونه‌ی یک ساختار بسته‌مبنا در شکل ۲ قابل مشاهده است.

شکل ۲: نمونه‌ی یک ساختار بسته مبنا



۴.۳ مفاهیم

برای پیاده‌سازی یک درگاه ارتباط با رابط‌های برنامه‌نویسی، مفاهیم زیر پیشنهاد شده است:

- بطن [۱]
- خدمت [۲]
- متعادل‌کننده‌ی بار [۳]
- ضابطه [۴]
- میان‌افزار [۵]
- مسیریاب [۶]
- تامین کننده‌ی خدمت [۷]

در ادامه به توضیح هر یک از مفاهیم ذکر شده پرداخته می‌شود.

۱.۴.۳ بطن

یک بطن، کوچک‌ترین واحد در این سامانه است. هر بطن، یک داده ساختار شامل نام، آدرس، وزن و وضعیت است. این چهار ویژگی نشان‌دهنده‌ی یک نمونه از رابط‌های برنامه‌نویسی قابل استفاده توسط برنامه است. داده‌ساختار زیر، نشان‌دهنده‌ی بطن است.

```
type Backend struct {
    Name string
    Addr string
    Weight int8
    Status int
}
```

[1] Backend
[2] Service
[3] Load Balancer
[4] Rule
[5] Middleware
[6] Router
[7] Service Provider

۲.۴.۳ خدمت

یک خدمت دارای یک نام، چند بطن و یک متعادل‌کننده‌ی توزیع بار است. معمولا بطن‌های یک خدمت کاملا مشابه یکدیگر عمل می‌کنند. با توجه به قسمت ۳.۴.۳ ، خدمت با استفاده از متعادل‌کننده‌ی بار درخواست‌های را بین بطن‌های خود توزیع می‌کند.

قطعه کد زیر نشان‌دهنده‌ی داده‌ساختار یک خدمت است.

```
type Service struct {
    Name     string
    Backends []Backend
    LB       LoadBalancer
}
```

۳.۴.۳ متعادل‌کننده‌ی بار

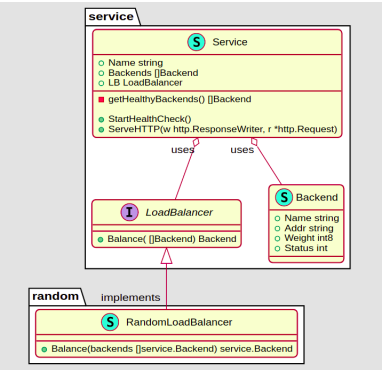
این موجودیت در برنامه از نوع یک واسط [۱] است. این واسط بدون در نظر گرفتن منطق پیاده‌سازی توزیع بار، رفتارهای مورد نیاز برای این عمل را مشخص می‌کند.

این واسط به طور مستقیم قابل استفاده نیست. بلکه نسخه‌های پیاده‌سازی شده‌ی این واسط را می‌توان به طور مستقیم استفاده کرد. برای مثال، الگوریتم توزیع تصادفی جهت توزیع بار در این برنامه پیاده‌سازی شده است. برنامه‌ی زیر نشان‌دهنده‌ی واسط متعادل‌کننده‌ی بار و شکل ۳ حاوی نمودار پیاده‌سازی نمونه‌ها از واسط هستند.

```
type LoadBalancer interface {
    Balance([]Backend) Backend
}
```

[1] Interface

شکل ۳: نمودار UML بسته‌ی Service



متعادل‌کننده‌های بار می‌توانند از وزن بطن‌های یک سرویس، برای توزیع وزن‌دار بار استفاده کنند.

۴.۴.۳ تامین کننده‌ی خدمت‌ها

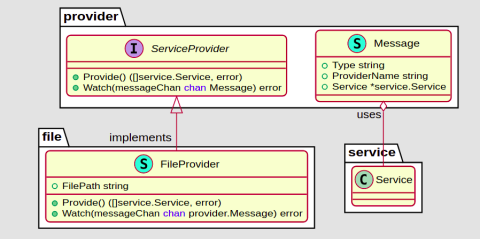
تامین‌کننده وظیفه تامین پیکربندی خدمت‌های مختلف را دارد. با توجه به تنوع در محیط‌های بارگذاری سامانه‌های با معماری میکروسرویس، درگاه ارتباط باید بتواند با اجزاهای محیطی مختلف ارتباط برقرار کرده و تنظیمات مربوط به خدمت‌های مختلف را از این محیط‌ها دریافت کند. قطعه کد زیر نشان‌دهنده‌ی واسط مربوط به تامین کننده‌ی خدمت است.

```
type ServiceProvider interface {
    Provide() ([]service.Service, error)
    Watch(messageChan chan<- Message) error
}
```

تامین‌کننده به صورت یک واسط تعریف شده است و نسخه‌های پیاده‌سازی شده‌ی مختلف آن‌ها، پیکربندی‌های مختلف را از محیط‌های متفاوت مانند Docker، Kubernetes و ... جمع‌آوری می‌کنند.

شکل ۴ نشان‌دهنده‌ی نمودار UML و نسخه‌های پیاده‌سازی شده از تامین‌کننده‌ی خدمت‌هاست.

شکل ۴: نمودار UML بسته‌ی Provider



۵.۴.۳ ضابطه

ضابطه‌ها داده‌ساختارهایی هستند که می‌توان از طریق آن‌ها به خدمات دسترسی پیدا کرد. در قالب این گزارش، ضابطه‌ها تنها برای پروتکل‌های HTTP و HTTPS طراحی شده‌اند. با توجه به ویژگی‌های این پروتکل‌ها، ضابطه‌های مختلفی می‌توان تعریف کرد. برخی از این ویژگی‌ها عبارتند از:

- نوع پروتکل (HTTP یا HTTPS)
- مسیر درخواست [۱]
- آدرس میزبان درخواست [۲]
- نوع درخواست [۳]
- سرتیترهای درخواست [۴]
- مولفه‌های پرس‌وجوی درخواست [۵]
- میان‌افزارها

کاربرد و نحوه‌ی استفاده از میان‌افزار ها در قسمت ۶.۴.۳ بررسی خواهد شد.

قطعه کد زیر، نمایانگر داده‌ساختار یک ضابطه است.

[1] Path
[2] Host
[3] Request Method
[4] Headers
[5] Query Params

```
type Rule struct {
    Service     *service.Service
    Schema      string
    PathPrefix  string
    Hosts       []string
    Methods     []string
    Headers     map[string]string
    Queries     map[string]string
    Middlewares []middlewares.Middleware
    handler     http.HandlerFunc
}
```

۶.۴.۳ میان‌افزار

گاهی نیاز است قبل از رسیدن درخواست به خدمت، تغییراتی در درخواست ایجاد شود. و یا برخی از سیاست‌های کنترلی مانند احراز هویت و تایید دسترسی قبل از رسیدن درخواست به خدمت بررسی شود.

در برخی از مواقع نیز این نیاز مطرح است که از نتایج درخواست‌ها مطلع شد و از آن‌ها برای تولید گزارشات و تهیه‌ی داده‌های آماری استفاده کرد.

میان‌افزارها این نیازها را برطرف خواهند کرد. میان‌افزارها نیز به صورت یک واسط تعریف شده‌اند و نسخه‌های پیاده‌سازی شده‌ی مختلف آن‌ها نیازهای مختلف را رفع خواهند کرد. برنامه‌ی زیر نشان‌دهنده‌ی شمای این واسط است.

```
type Middleware interface {
    Process(handler http.HandlerFunc) http.HandlerFunc
}
```

در سامانه‌ی پیاده‌سازی شده، میان‌افزار ها دو قسم اند. برخی از آن‌ها،که معمولا جز میان‌افزارهای پرطرفدار در صنعت هستند، به صورت پیشفرض پیاده‌سازی شده‌اند. میان‌افزارهای از پیش آماده در این سامانه عبارتند از:

- میان‌افزار CORS
- میان‌افزار محدودکننده‌ی نرخ درخواست [۱]

[1] Rate limiter

- میان‌افزار رصد سامانه

محدودکننده‌ی نرخ درخواست با استفاده از الگوریتم Token Bucket [۲] باعث می‌شود بتوان نرخ ارسال درخواست‌های کاربران را تحت کنترل قرار داد.

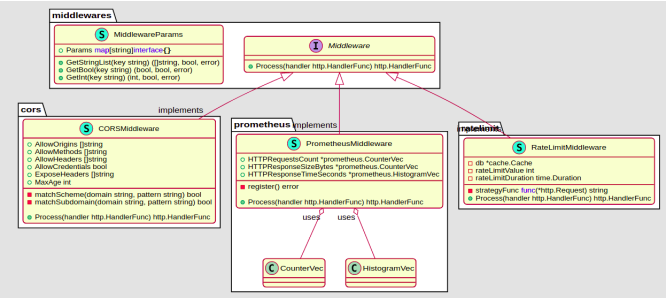
میان‌افزار CORS [۳] نیز با استفاده از استانداردهای تعیین‌شده برای درخواست‌های با مبدا و مقصد مختلف، دسترس‌پذیری این سامانه را از منابع مختلف تعیین می‌کند.

میان‌افزار رصد سامانه نیز، طبق استانداردهای نرم‌افزار Prometheus، [۴] امکان بررسی و تحلیل اطلاعات را فراهم می‌کند.

گروهی دیگر از میان‌افزارها نیز توسط کاربران سامانه تعریف و پیاده‌سازی شده و به داخل سامانه تزریق می‌شوند. این ویژگی باعث می‌شود استفاده کنندگان از این سامانه، هیچ‌گونه وابستگی به توسعه‌دهندگان اصلی برنامه نداشته باشند و میان‌افزارهای مربوط به نیازمندی‌های خاص خود را پیاده‌سازی کرده و در سامانه تزریق کنند.

شکل ۵ نشان‌دهنده‌ی نحوه‌ی پیاده‌سازی میان‌افزارها با استفاده از واسط تعیین‌شده است.

شکل ۵: نمودار UML بسته‌ی Middleware



۷.۴.۳ مسیریاب

یک درگاه ارتباط با رابط‌های برنامه‌نویسی را می‌توان یک مسیریاب لایه‌ی هفتم شبکه در نظر گرفت. بدین صورت که درخواست‌های ارسالی از سمت کاربران را به پاسخ‌دهنده‌های مربوط به آن درخواست‌ها هدایت می‌کند.

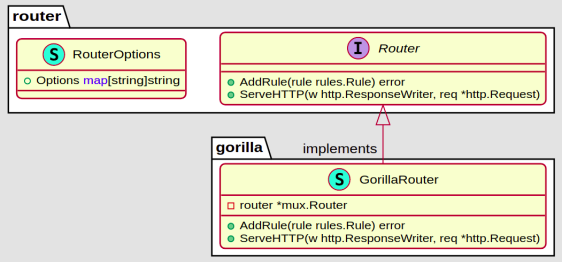
یک مسیریاب به طور واسط تعریف شده است و رفتارهای مورد نیاز آن، بدون توجه به منطق پیاده‌سازی آن مشخص شده است. برنامه‌ی زیر تعریف یک واسط مسیریاب را نشان می‌دهد.

```
type Router interface {
    AddRule(rule rules.Rule) error
    ServeHTTP(w http.ResponseWriter, req *http.Request)
}
```

مسیریاب‌ها با استفاده از ضابطه‌های تعریف‌شده، هر درخواست را به خدمت مورد نظر هدایت می‌کنند.

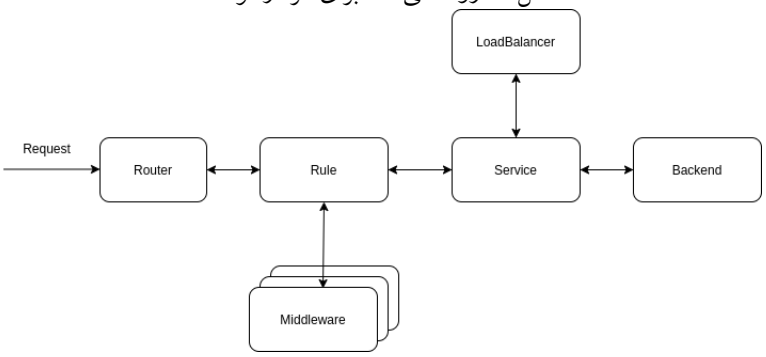
شکل ۶ نشان‌دهنده‌ی نحوه‌ی پیاده‌سازی یک مسیریاب با استفاده از واسط تعیین شده است.

شکل ۶: نمودار UML بسته‌ی Router



به طور کلی، شکل ۷ نشان‌دهنده‌ی روند طی‌شده برای درخواست کاربران را به طور خلاصه شرح می‌دهد.

شکل ۷: روند طی‌شده برای هر درخواست



۵.۳ ویژگی‌های سامانه

معماری ذکرشده در بخش ۴.۳ ، ویژگی‌های زیر را داراست:

- کمترین وابستگی ممکن

- آزمون پذیری بالا
- گسترش پذیری بالا
- مقیاس‌مندی بالا

۱.۵.۳ وابستگی

با استفاده درست از واسطها و همچنین طراحی ماژولار، بخش‌های مختلف سامانه کمترین وابستگی به یکدیگر را دارند. این ویژگی باعث می‌شود تغییرات احتمالی در آینده به راحتی هرچه تمام‌تر انجام شود.

۲.۵.۳ آزمون‌پذیری

با توجه به وابستگی کم اجزای سامانه به یکدیگر،هر یک از اجزا می‌توانند به تنهایی مورد آزمون و ارزیابی قرار گیرند. هم‌چنین استفاده از واسطها امکان ایستا کردن یک بخش و انجام آزمون واحد [1] بر روی بخش دیگر را امکان‌پذیر می‌کند.

۳.۵.۳ گسترش‌پذیری

با توجه به استفاده به موقع از واسطها، گسترش سامانه، از طریق پیاده‌سازی نسخه‌های جدید از واسطها، بسیار ساده خواهد بود. همچنین امکان تزریق میان‌افزارهای پیاده‌سازی شده توسط کاربران استفاده کننده، بدون نیاز به کامپایل مجدد نرم‌افزار، خاصیت گسترش پذیری نرم‌افزار را افزایش می‌دهد.

۴.۵.۳ مقیاس‌مندی

با توجه به عدم ذخیره‌ی حالت در سامانه، می‌توان تعداد نسخه‌های در حال اجرای سامانه را به طور خطی برای افزایش میزان توان عملیاتی سامانه، زیاد کرد.

[1] Unit test

۴ نتایج و تفسیر آن‌ها

پس از پیاده‌سازی معماری شرح‌داده‌شده در فصل ۳ ، آزمایش کیفیت و نحوه‌ی عملکرد سامانه ضروری به نظر می‌رسد. در آزمایش‌های انجام‌شده، سامانه‌ی پیاده‌سازی شده با یکی از محبوب‌ترین نمونه‌های صنعتی، به نام Kong، مورد مقایسه قرار گرفته شده است. در ادامه، به نحوه‌ی انجام آزمایش پرداخته شده است.

۱.۴ محیط آزمایش

با توجه به کاربرد وسیع درگاه‌های ارتباط با رابط‌های برنامه‌نویسی در محیط‌های ابری، انجام این آزمایش در محیط ابری به واقعیت نزدیک‌تر خواهد بود. از این رو تمام آزمایش‌ها در محیط ابری انجام شده‌است. آزمایش‌ها بر روی بستر OKD [1] انجام شده‌اند.

برای ایجاد بار بر روی سامانه‌های مورد آزمایش از سکوی Locust استفاده شده است. میزان منابع مصرفی این سکو برای ایجاد بار به شرح زیر است:

- میزان حافضه‌ی موقت مصرفی: ۱۱ گیگابایت
- تعداد هسته‌های پردازشی: ۲۲
- تعداد ایجادکننده‌های بار: ۱۰

۲.۴ شرح آزمایش‌ها

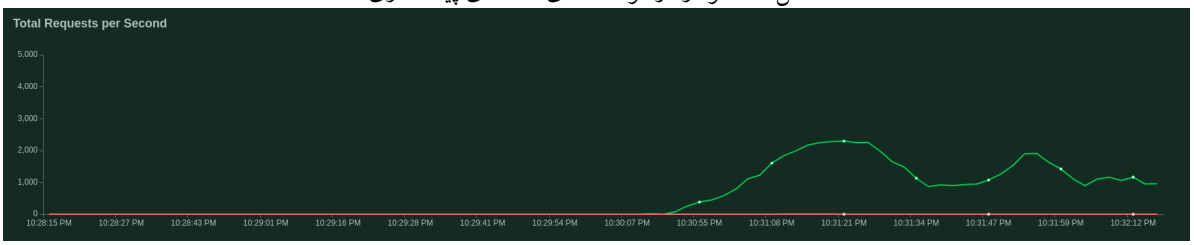
۱.۲.۴ میزان تحمل بار

برای بررسی میزان تحمل بار، پس از ساخت دو بطن مشابه و بدون سربار بالا، در قالب یک خدمت و استفاده از یک ضابطه‌ی مناسب به انجام آزمایش مبادرت شده است. این فرآیند در هر دو سامانه‌ی مورد ارزیابی انجام شده است.

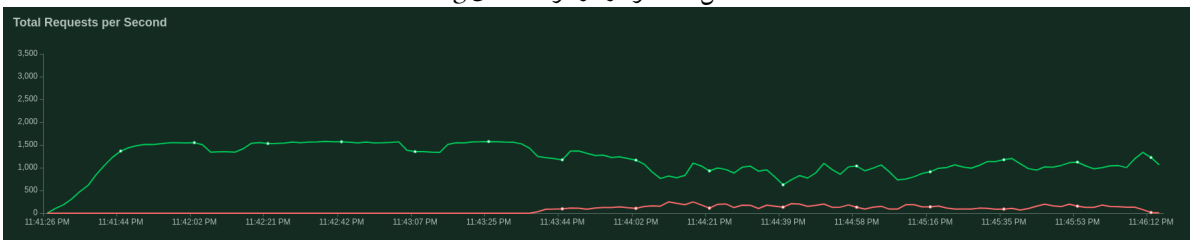
نتایج حاصل از تولید بار بر هر یک از سامانه‌ها، که در شکل‌های ۸ و ۹ قابل مشاهده است به شرح زیر است.

[1] Openshift Kubernetes Distribution

شکل ۸: نمودار درخواست‌های سامانه‌ی پیاده‌سازی شده



شکل ۹: نمودار درخواست‌های Kong



جدول ۳: مقایسه‌ی Kong و سامانه‌ی پیاده‌سازی شده

محصول مورد آزمایش	بیشینه مصرف حافظه‌ی موقت	بیشینه مصرف هسته‌های پردازشی	بیشینه میزان تحمل بار
سامانه‌ی پیاده‌سازی شده	۲۴۴ MB	۲	۲۳۳۰ rps
Kong	۲۰۴۸ MB	۲	۱۶۴۰ rps

با توجه به جدول شماره‌ی ۳ ، علی رغم کاهش ٪ ۸۸ ای میزان مصرف حافظه‌ی موقت، بیشینه میزان تحمل بار حدود ٪ ۴۲ افزایش یافته است. علاوه بر این طبق شکل شماره‌ی ۸ ، میزان درخواست‌های رد شده در سامانه‌ی پیاده‌سازی شده صفر است ولی در سامانه‌ی Kong شاهد رد شدن برخی از درخواست‌ها در اوج بار هستیم.

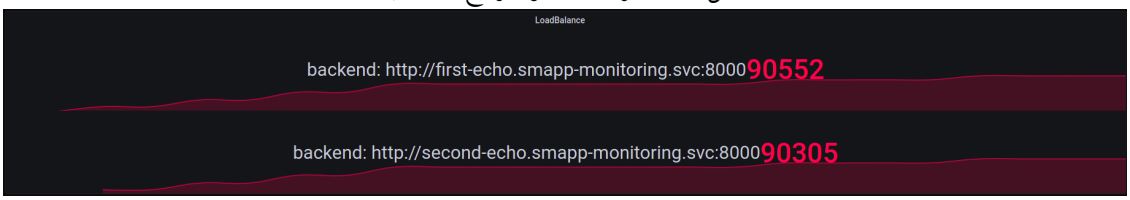
از این نظر، سامانه‌ی پیاده‌سازی شده بر Kong برتری کامل دارد.

۲.۲.۴ نحوه‌ی توزیع بار

با توجه به وجود دو بطن برای سرویس مورد آزمایش، نحوه‌ی عملکرد توزیع‌کننده‌ی بار نیز باید مورد آزمایش قرار گیرد. با فعال کردن میان‌افزار Prometheus بر روی یک ضابطه‌ی جدید و اجرای آزمایش بر روی آن و همچنین تحلیل نتایج حاصل، می‌توان نتایج را

در شکل شماره‌ی ۱۰ مشاهده کرد.

شکل ۱۰: نحوه‌ی عملکرد توزیع‌کننده‌ی بار

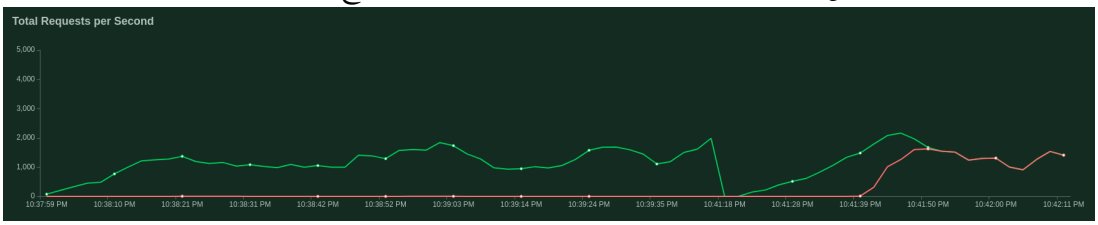


با توجه به شکل شماره‌ی ۱۰ ، از مجموع ۱۸۰۸۷۵ درخواست ارسال شده به سامانه، ۹۰۵۵۲ به بطن اول و ۹۰۳۰۵ به بطن دوم هدایت شده اند. در مجموع ٪ ۵۰/۰۷ درخواست ها به بطن اول و ٪ ۴۹/۹۳ از درخواست‌ها به بطن دوم هدایت شده‌اند. با توجه به وزن یکسان هر دو بطن، نحوه‌ی عملکرد توزیع‌کننده بار، قابل قبول است.

۳.۲.۴ کنترل نرخ درخواست‌ها

یکی از میان‌افزارهای پیاده‌سازی شده، کنترل کننده‌ی نرخ درخواست‌ها است. برای ارزیابی این میان‌افزار و عملکرد درست آن، با ساخت یک ضابطه‌ی جدید و فعال‌سازی این میان‌افزار در آن،به اجرای دوباره‌ی آزمایش مبادرت شده است. نحوه‌ی پاسخ به درخواست‌ها توسط سامانه، در شکل ۱۱ قابل مشاهده است.

شکل ۱۱: نحوه‌ی رفتار سامانه در حضور کنترل کننده‌ی نرخ درخواست‌ها

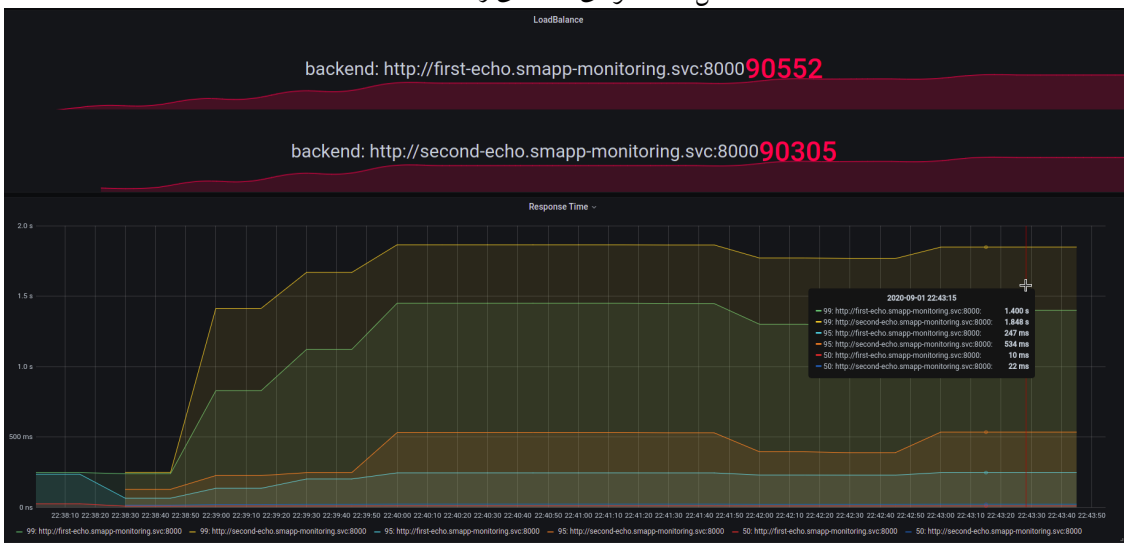


با توجه به شکل ۱۱ ، پس از تمام شدن تعداد درخواست مجاز هر فرستنده در زمان ۱۰:۴۱:۳۹، سامانه شروع به رد کردن درخواست‌های جدید می‌کند. از این رو، میان‌افزار پیاده‌سازی شده به درستی عمل می‌کند.

۴.۲.۴ رصد و تحلیل سامانه

برای بررسی عملکرد مطلوب میان‌افزار Prometheus جهت رصد و تحلیل این سامانه، این میان‌افزار در یک ضابطه‌فعال شده و ارزیابی مجددا انجام شده است. در شکل شماره‌ی ۱۲ ، شاهد نمونه‌ی صفحه‌ی رصد سامانه هستیم. در این صفحه نحوه‌ی عملکرد توزیع‌کننده‌ی بار و همچنین Quantile های مختلف مدت زمان پاسخ‌گویی سامانه محاسبه شده است.

شکل ۱۲: نمونه‌ی صفحه‌ی رصد سامانه



۵ پیشنهادها

پیاده‌سازی یک راه‌کار جامع برای محیط‌های ابری، دارای شرایط و سختی‌های مخصوص به خود است. برخی از ویژگی‌ها برای استفاده‌ی صنعتی از این نوع سامانه‌ها ضروری است. برخی از این ویژگی‌ها که در نسخه‌ی پیاده‌سازی شده وجود ندارند، عبارتند از:

- پنل و رابط برنامه‌نویسی مدیریت سامانه
- پشتیبانی از پروتکل‌های websocket، gRPC و...
- برخی میان‌افزارهای پرکاربرد
- یکپارچگی با برخی از سکوهای ابری

۱.۵ پنل و رابط برنامه‌نویسی مدیریت سامانه

وجود پنل و رابط‌های برنامه‌نویسی برای مدیریت پیکربندی‌ها و عدم نیاز به راه‌اندازی مجدد سامانه، علیرغم اینکه برخی از نسخه‌های تجاری فاقد آن هستند، یکی از ویژگی‌های ضروری یک درگاه ارتباط با رابط‌های برنامه‌نویسی است. وجود این ویژگی باعث عدم وابستگی سازمان‌های استفاد کننده به افراد با تخصص بالا است.

۲.۵ پنل و رابط برنامه‌نویسی مدیریت سامانه

در نسخه‌ی پیاده‌سازی شده، تنها پروتکل‌های HTTP و HTTPS پشتیبانی می‌شوند. امکان پشتیبانی از برخی از پروتکل‌ها،مانند gRPC، websocket، HTTP2 و ... برای بسیاری از سامانه‌ها ضروری هستند. ترجمه‌ی پروتکل‌ها به یکدیگر نیز یکی از ویژگی‌های جذاب و پرکاربرد در درگاه‌های ارتباط محسوب می‌شود.

۳.۵ میان‌افزارهای پرکاربرد

برخی از میان‌افزارهای پرکاربرد، هزینه‌ی پیاده‌سازی بخش‌های مختلف سامانه را کاهش می‌دهند. وجود این میان‌افزارها، باعث می‌شود استفاده از درگاه‌های ارتباط، از نظر زمانی و اقتصادی به‌صرفه و منطقی باشند. برخی از این میان‌افزارها عبارتند از:

- میان‌افزار Caching

- میان‌افزار احراز هویت [1] و کنترل دسترسی [2]
- میان‌افزارهای مربوط به امنیت سامانه
- میان‌افزارهای ترکیب پاسخ‌های خدمت‌های مختلف
- میان‌افزارهای تغییر درخواست و پاسخ درخواست

با توجه به معماری گسترش‌پذیر سامانه‌ی پیاده‌سازی شده، اضافه‌کردن میان‌افزارها، به سادگی امکان‌پذیراست. با تحلیل نیازهای واقعی در صنعت و استخراج نیازهای هر کدام از میان‌افزارها، می‌توان این میان‌افزارها را به طور پیش‌فرض در سامانه قرار داد.

۴.۵ یکپارچگی با برخی از سکوهای ابری

با توجه به رشد روزافزون محیط‌های ابری و استفاده‌ی روزمره‌ی آن‌ها توسط مشتریان مختلف، امکان یکپارچه‌سازی با سکوهای ابری معروف، یک ویژگی بسیار مهم برای درگاه‌های ارتباط خواهد بود. برخی از سکوهای معروف خدمات ابری عبارتند از:

- Docker
- Kubernetes
- Openshift
- Openstack
- Marathon
- Mesos

با توجه به معماری گسترش‌پذیر بخش تامین‌کنندگان، که در حال حاضر از تامین‌کننده‌ی فایل پشتیبانی می‌کند، می‌توان با کمی مطالعه‌ی رابط‌های برنامه‌نویسی سکوهای ذکر شده، تامین‌کنندگان جدیدی به سامانه اضافه کرد.

[1] Authentication
[2] Authorization

مراجع

[1] S. Newman, *Monolith to Microservices*. Nginx, 2018. [Online]. Available: https://www.nginx.com/blog/microservices-from-design-to-deployment-ebook-nginx/

[2] A. Tanenbaum and D. Wetherall, *Computer Networks*. Pearson Education, 2012. [Online]. Available: https://books.google.com/books?id=IRUvAAAAQBAJ

[3] A. Barth, "The web origin concept," Internet Requests for Comments, RFC, December 2011. [Online]. Available: https://tools.ietf.org/html/rfc6454

[4] S. Ntalampiras, D. Arsic, A. Stormer, T. Ganchev, I. Potamitis, and N. Fakotakis, "Prometheus database: A multimodal corpus for research on modeling and interpreting human behavior," in *2009 16th International Conference on Digital Signal Processing*, 2009, pp. 1–8.

Abstract

Due to the growth of online businesses and the need for stability and high efficiency for providing services for customers, new architectures should be used for software development. The main characteristics of these architectures are scalability and being fault-tolerant.

Microservice, among these architectures, is one of the most popular ones. The focus of this architecture is on scalability, extendability and independence of different services from each other. The way this architecture achieves the mentioned features is through separating different functionalities into different services and deploying them in isolated environments.

Similar to any architecture, the proper implementation of microservice architecture has several challenges. Due to the decentralized nature of the systems in this architecture, connecting users to the services, will not be as simple as in the past because the user needs to gather information from different parts of the system. Therefore, changing the way users communicate with the system and upgrading user applications will be expensive.

API Gateway pattern is designed to solve this challenge. In this pattern, users still send their requests to only one median system. The task of this median system is to receive requests and redirect them to associated services.

In spite of the existence of various open source and enterprise API Gateways, most of them lack extendability and configurability. The main purpose of this project is to implement an API Gateway that is extendable and configurable.

Keywords: API Gateway, Microservice architecture, Decentralized systems, Extendable systems.

Computer Engineering Department

Implementing an API (Application Programming Interface) Gateway

Bachelor of Science Thesis in Computer Engineering

Saeed Tahmasebi Sales

Supervisor:

Dr. Vesal Hakami

September 2020