دانشگاه صنعتی امیرکبیر

(پلی‌تکنیک تهران)

دانشکده مهندسی کامپیوتر

پروژه کارشناسی

رابط ارتباطی تحت وب خدمات زیرساخت بعنوان خدمت VCloud

نگارش

سیدمحمدفاطمی

استاد راهنما

دکتر محمود ممتازپور

اسفند ۱۴۰۱


بسم الله الرحمن الرحيم

صفحه فرم ارزیابی و تصویب پایان نامه- فرم تأیید اعضاء کمیته دفاع

در این صفحه فرم دفاع یا تایید و تصویب پایان نامه موسوم به فرم کمیته دفاع- موجود در پرونده آموزشی- را قرار دهید.

نکات مهم:

- نگارش پایان نامه/رساله باید به زبان فارسی و بر اساس آخرین نسخه دستورالعمل و راهنمای تدوین پایان نامه‌های دانشگاه صنعتی امیرکبیر باشد.(دستورالعمل و راهنمای حاضر)
- رنگ جلد پایان نامه/رساله چاپی کارشناسی، کارشناسی ارشد و دکترا باید به ترتیب مشکی، طوسی و سفید رنگ باشد.
- چاپ و صحافی پایان نامه/رساله بصورت پشت و رو(دورو) بلامانع است و انجام آن توصیه می شود.

دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)

به نام خدا

تعهدنامه اصالت اثر

تاریخ: اسفند ۱۴۰۱

اینجانب **سیدمحمدفاطمی** متعهد می‌شوم که مطالب مندرج در این پایان‌نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان‌نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان‌نامه متعلق به دانشگاه صنعتی امیرکبیر است. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

سیدمحمدفاطمی

امضا

تقدیم به

آن‌که جز به فضلش امیدی نیست...

سپاس‌گزاری

از پدر و مادرم که همواره در مواجهه با سختی‌های این دنیا دلسوزانه همراهم بوده‌اند؛
از استاد بزرگوارم جناب آقای دکتر محمود ممتازپور که با حسن خلق و گشاده‌رویی، رهنمودهای شبانه‌روزی خود را از من دریغ نکرده‌اند؛
و از سایر عزیزانی که در کنارشان این نتیجه حاصل آمد کمال تشکر و قدردانی را دارم.

سید محمد فاطمی
اسفند ۱۴۰۱

چکیده

امروزه الگوی مصرف منابع محاسباتی و شبکه‌ای بسیار به الگوی رایانش ابری نزدیک شده. با این وجود ارائه دهندگان خدمات زیرساخت در داخل کشور همچنان فاصله زیادی تا ارائه خدمات منطبق بر الگوی رایانش ابری دارند. یکی از خدمات معروف مورد نیاز کاربران، نیاز به زیرساخت به عنوان خدمت است. چالش پیاده‌سازی این خدمت، پیاده‌سازی زیرساخت و درگاه‌های مورد نیاز جهت تحویل خدمات به کاربر است. در مورد مسئله اول، راه‌حل‌های متن‌باز و صنعتی متعددی ارائه شده‌اند. ایراد استفاده مستقیم از این راه‌حل‌های بدون نیاز به تغییر و پیاده‌سازی درگاه ارتباطی خصوصی، عدم امکان شخصی‌سازی امکانات و نحوه ارائه خدمات است. به علت متفاوت بودن الگوهای ارائه‌ی خدمت و مدیریت کاربران، این ابزارها، پیاده‌سازی این قسمت از سامانه را برعهده‌ی کاربر استفاده کننده قرار داده‌اند. در این پروژه به بررسی نیازمندی‌های یک سامانه‌ی ارائه‌ی خدمات زیرساخت به عنوان خدمت پرداخته شده و با بررسی و مشخص کردن نیازمندی‌ها و فناوری‌ها، یک رابط برنامه‌نویسی تحت وب برای خدمات مطرح شده بر روی زیرساخت VMWare Cloud Director ارائه شده‌است. در مراحل مختلف طراحی، تصمیمات و گزینه‌های مختلف مطرح شده و دلیل تصمیم گیری‌ها و انتخاب فناوری شرح داده شده. پس از پیاده‌سازی، با طرح و اجرای ارزیابی‌های مختلف بر روی این راه‌حل، از صحت عملکرد آن در سناریوهای واقعی اطمینان حاصل پیدا شده. در انتها با بررسی کلیت سامانه، نقاط بهبود و جایگاه‌های قابل توسعه در پروژه شرح داده‌شده که بتوان در جهت‌های مختلف این پروژه را توسعه داد و خدمات جامع‌تری را پوشش داد.

واژه‌های کلیدی:

رایانش ابری، زیرساخت به عنوان خدمت، رابط برنامه‌نویسی تحت وب، معماری میکروسرویس

فهرست مطالب

عنوان صفحه

فهرست تصاویر

شکل صفحه

فهرست جداول

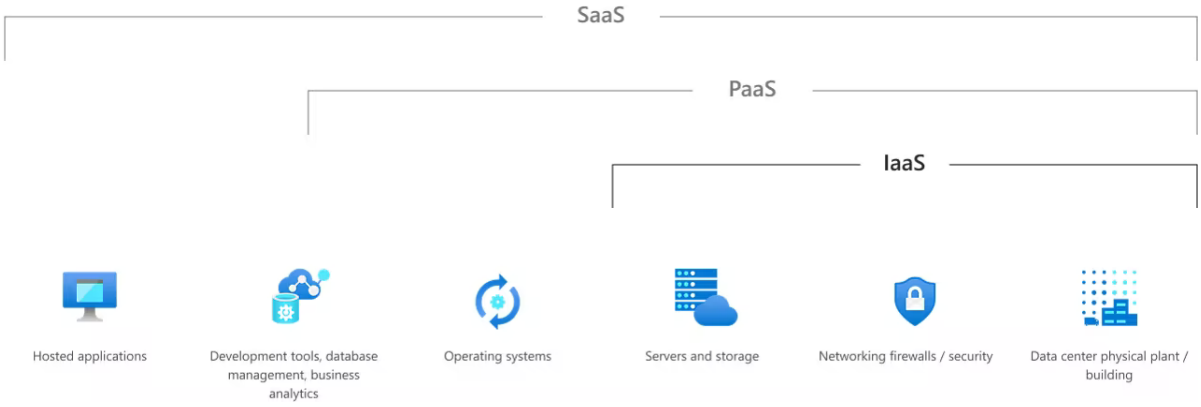
جدول صفحه

فصل اول

مقدمه

۱-۱ مقدمه

رایانش ابری[۱] یکی از حوزه‌های در حال تحول کامپیوتر است که به یک جنبه‌ی حیاتی از فناوری و مشاغل مدرن تبدیل شده است. این الگو همچون انقلابی، شیوه‌ی ارائه و مصرف منابع محاسباتی را متحول کرده است و فرصت‌های جدیدی را برای سازمان‌ها فراهم کرده تا کارایی و رقابت خود را بهبود بخشند. یکی از پایه‌ای‌ترین و رایج‌ترین شکل‌های رایانش ابری، زیرساخت به عنوان خدمت (IaaS)[۲] است که منابع محاسباتی مجازی مانند فضای ذخیره‌سازی، قدرت پردازش و پهنای باند را از طریق اینترنت در اختیار کاربران قرار می‌دهد[؟].



شکل ۱-۱: انواع مدل‌های رایانش ابری[؟]

این الگو را از دو دیدگاه می‌توان بررسی کرد. دیدگاه اول مختص کاربران مصرف‌کننده است که برای استفاده و مصرف منابع دیگر دغدغه‌ای نسبت به زیرساخت استفاده شده برای ارائه‌ی خدمات ندارند. دیدگاه دوم مربوط به ارائه دهندگان این خدمت است. از این دیدگاه، توسعه یک سکو[۳] کارآمد و قابل اعتماد که بتواند مشتریان را جذب و حفظ کند، بسیار مهم است. از این رو، شرکت‌ها و توسعه‌دهندگان سراسر دنیا همواره سعی در توسعه و فراهم‌کردن ابزارها با این هدف دارند. یکی از معروف‌ترین این شرکت‌ها، شرکت VMWare است که برنامه‌های فراوانی برای فراهم کردن زیرساخت ارائه‌ی خدمات ابری توسعه داده است. چالش بعدی ارائه‌دهندگان، فراهم کردن و طراحی یک رابط برنامه‌نویسی (API)[۴]

[1] Cloud Computing
[2] Infrastructure As A Service
[3] Platform
[4] Application Programming Interface

است که بتواند با مشتریان و ارائه دهنده زیرساخ[۵] تعامل داشته‌باشد و آن‌ها را قادر سازد منابع مجازی خود را مدیریت کنند.

۱-۲ تعریف مسئله

در این پروژه قصد داریم که خدمات جامع یک ارائه‌دهنده‌ی زیرساخت به عنوان خدمت را که بر بستر ابزار VMWare Cloud Director[۶] فراهم شده است را توسط یک رابط برنامه‌نویسی به کاربران ارائه دهیم. برای این هدف، باید خدمات کاربردی، مدیریتی، نظارتی و امنیتی را در یک لایه‌ی جدا تعریف کنیم و این لایه از برنامه‌ی خود را با ابزار زیرساختی در تعامل نگه داریم.

۱-۳ راه حل پیشنهادی

برای توسعه این محصول، بایستی در ابتدا با بررسی نیازمندی‌ها و محدودیت‌های زیرساخت و کاربران، اقدام به طراحی کلی سامانه کرد. با توجه به گسترده بودن این نیازمندی‌ها و توسعه‌پذیری جداگانه آن‌ها، باید طراحی کلی این سامانه به صورت پیمانه‌ای[۷] و با کمترین وابستگی میان واحدها باشد. در این گزارش ابتدا به طراحی و شرح قسمت‌های مختلف سامانه می‌پردازیم و سپس جزئیات فنی هر قسمت و تعامل قسمت‌ها را بررسی می‌کنیم.

هدف ما، پیاده‌سازی یک سامانه‌ی کاربردی کامل است که با استفاده از رابط برنامه‌نویسی تحت وب، بتوانیم خدمات را از آن دریافت کنیم. این خدمات شامل موارد زیر است.

۱. **احراز هویت**

۲. **خدمات ابری**

۳. **مدیریت و حسابداری**

این خدمات باید با دسترسی‌پذیری بالا[۸] و به صورت توزیع‌شده[۹] توسعه پیدا کنند که پاسخگوی نیازمندی‌های کاربران در شرایط خاص و فشارهای بسیار بالا باشد.

[5] Infrastructure Provider
[6] https://vmware.com/products/cloud-director.html
[7] Modular
[8] Highly Available
[9] Distributed

۱-۳-۱ احراز هویت

هدف از پیاده‌سازی این قسمت، ایجاد سطوح دسترسی مختلف برای کاربران و کنترل دسترسی کاربران به منابع است. در این سامانه، کاربران انتظار خدمات زیر در حوزه احراز هویت را دارند.

- **ایجاد حساب کاربری**: کاربر سامانه باید بتواند با نام کاربری و کلمه عبور، اقدام به ساخت حساب کاربری نماید. پس از ساخته‌شدن این حساب، دسترسی به منابع فقط با داشتن این اطلاعات هویتی امکان‌پذیر است.
- **سطوح دسترسی مختلف**: با توجه به گستردگی خدمات ارائه‌شده، سطوح دسترسی متفاوتی باید داخل سامانه قابل تعریف باشد.
- **مدیریت دسترسی**: مدیر سامانه باید بتواند موقتا یا دائما دسترسی کاربران را از سامانه بگیرد.
- **صحت‌سنجی درخواست‌ها**: پس از احراز هویت کاربر، تمامی درخواست‌های ورودی به سامانه باید صحت‌سنجی شوند.

۱-۳-۲ خدمات ابری

کاربرد اصلی این سامانه، ارائه خدمات IaaS در بستر یک رابط برنامه‌نویسی تحت وب است. این خدمات با تمرکز بر ارائه ماشین‌های مجازی و عملیات قابل تعریف بر روی آن شامل موارد زیر است.

- ساخت ماشین مجازی
- حذف ماشین مجازی
- ویرایش مشخصات ماشین مجازی
- ساخت Image و برنامه‌های قابل اجرا روی ماشین مجازی
- حذف و ویرایش برنامه‌های قابل اجرا روی ماشین مجازی
- ارسال دستورات روشن، خاموش، راه‌اندازی مجدد به ماشین‌های مجازی
- ارسال دستور تهیه نسخه‌های پشتیبان از ماشین مجازی

- تعریف شبکه‌های محلی
- تعریف شبکه‌های خصوصی
- اضافه کردن ماشین مجازی به یک شبکه

۱-۳-۳ مدیریت و حسابداری

دسته دیگری از خدمات که علاوه بر کاربران، برای مدیر سامانه نیز تعریف می‌شود، عملیات مربوط به مدیریت، نظارت و حسابداری است. لیست این عملیات به شرح زیر است.

- تعریف سهمیه[۱۰] منابع
- حذف سهمیه منابع
- بروزرسانی استفاده از سهمیه
- رصد کردن مصرف منابع
- دریافت گزارش از مصرف منابع
- تعریف اعتبار مالی
- تعریف هزینه‌های منابع
- اعتبارسنجی درخواست‌ها با توجه به سهمیه و اعتبار

۱-۳-۴ قابلیت‌های غیرعملکردی

علاوه بر نیازمندی‌ها و قابلیت‌های عملکردی که در بخش‌های گذشته مطرح شد، دسته دیگری از نیازمندی‌ها و قابلیت‌ها وجود دارد که جنبه‌ی غیرعملکردی دارند. این قابلیت‌ها مربوط به اجرا،امنیت، نگهداری و توسعه سامانه می‌شود.

در اجرای سامانه، باید به این نکته که بار سامانه بسته به تعداد کاربران و نوع درخواست‌ها تغییر می‌کند توجه داشت و تمهیدات لازم جهت اجرای مناسب برنامه در سناریوهای مختلف را تدارک دید. از

[10] Quota

جمله این تمهیدات، مقیاس کردن برنامه متناسب با بار، اجرای مجدد سامانه در صورت بروز خطا، تقسیم درخواست‌ها است.

گروه دیگری از قابلیت‌ها، مربوط به امنیت سامانه است. هدف از این قابلیت‌ها، شناسایی زودهنگام آسیب‌پذیری‌های ممکن و رفع آن‌ها در جهت بهبود امنیت سامانه در اجرای نهایی است. از جمله این قابلیت‌ها، سازوکار های مقاوت بر آسیب‌پذیری‌های پایگاه داده، جلوگیری و کاهش اثر حمله‌های شبکه‌ای است.

دسته آخر از قابلیت‌های غیر عملکردی، شامل امور نگهداری و توسعه سامانه است. از میان این قابلیت‌ها، ثبت وقایع به شیوه موثر و مناسب، ذخیره معیار[11]های مختلف برنامه، پیاده‌سازی داشبوردهای نظارتی از کلیدی‌ترین نیازمندی‌های سامانه هستند.

در ادامه جزئیات فنی و ابزارهای مورد استفاده جهت ارائه این قابلیت‌ها توضیح داده شده‌است.

۱-۴ کارهای مشابه

با توجه به محبوبیت رایانش ابری و خدمات ارائه زیرساخت، اکثر قریب به اتفاق شرکت‌های فعال در این حوزه اقدام به ارائه درگاه‌های ارائه خدمات ابری به صورت رابط برنامه‌نویسی کرده اند که در ادامه با این محصولات آشنا می‌شویم.

۱-۴-۱ نمونه‌های داخلی

از بین شرکت‌های داخلی فعال در این حوزه، شرکت ابرآروان اقدام به ارائه‌ی خدمات IaaS در قالب رابط ارتباطی تحت وب کرده است.

این رابط که در این آدرس[؟] تفسیر شده است، تمامی خدمات قابل تعریف برروی ماشین مجازی و منابع مجازی را مشخص کرده. با توجه به متن‌بسته بودن این سرویس، نمی‌توان بیش از این در مورد معماری و جزئیات پیاده‌سازی خدماتی که ارائه می‌شود، صحبت کرد.

یکی دیگر از شرکت‌های ارائه دهنده خدمات مشابه، شرکت آسیاتک است. مستندات مربوط به عملکرد سرویس ارائه خدمات ابری این شرکت، در این آدرس[؟] قرارداده شده.

[11] Metric

۱-۴-۲ نمونه‌های خارجی

شرکت‌های بزرگ فعال در حوزه‌ی خدمات ابری نظیر گوگل، مایکروسافت، آمازون، دیجیتال اوشن، اوراکل و آی‌بی‌ام، همگی سرویس ارائه‌ی خدمات IaaS را به صورت عمومی برای استفاده کاربران فراهم کرده‌اند. معماری و نحوه‌ی سرویس‌دهی تمامی این ارائه دهندگان به شکل مشابه و بر مبانی الگوی پرداخت بر اساس مصرف[۱۲] است. منتها به دلیل متن‌بسته بودن این سرویس‌ها، اطلاعات بیشتری در مورد جزئیات پیاده‌سازی آن‌ها در دسترس نیست. با این وجود، با بررسی دقیق‌تر و جزئی‌تر مستندات سرویس‌ها می‌توان کلیاتی در مورد طراحی و عملکرد سامانه‌ها به دست آورد که در فصل بعد به بررسی این موارد می‌پردازیم.

[12] Pay-As-You-Go

فصل دوم

اجزا و فناوری‌ها

در این فصل به بررسی ابعاد مختلف این پروژه و ابزارها و فناوری‌های مورد استفاده می‌پردازیم؛ سپس با بررسی جزئی هر یک از این اجزا و مقایسه با سایر گزینه‌ها، دلیل انتخاب خود را مطرح می‌کنیم.

۲-۱ زیرساخت به عنوان خدمت

برای ارائه‌ی خدمات رایانش ابری، نیازمند لایه‌ای از ابزارها و برنامه‌ها هستیم که با قرار گرفتن روی سخت‌افزار واقعی، منابع مجازی را برای ما فراهم کنند. ابزارهای مختلفی با معماری‌های مختلف با این هدف توسعه داده شده‌اند. شرکت‌های بزرگ فناوری همانند گوگل، آمازون و مایکروسافت این زیرساخت‌ها را به صورت مدیریت شده در اختیار کاربران قرار می‌دهند. سرویس‌های Google Cloud Engine، Amazon AWS و Microsoft Azure به ترتیب نام محصولات این شرکت‌ها با هدف ارائه خدمات رایانش ابری است.

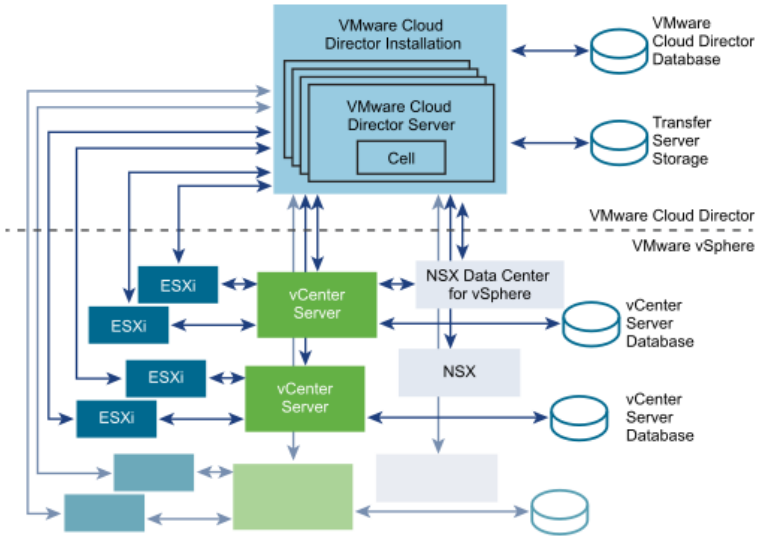
دسته دیگری از این برنامه‌ها، ماهیت خود میزبانی[1] دارند. به این شکل که بایستی مستقیما توسط کاربر بر روی سخت‌افزار واقعی نصب شوند. از جمله این محصولات می‌توان به VMWare vSphere، OpenStack و VMWare Cloud Director اشاره کرد.

از بین این ابزارها، با توجه به سابقه‌ی موفق شرکت VMWare در توسعه محصولات مربوط به مجازی‌سازی و همچنین جامع و کامل‌تر بودن امکاناتی که Cloud Director نسبت سایر محصولات ارائه می‌دهد، انتخاب ما برای انجام پروژه این محصول است.

[1] Self Hosted

۲-۱-۱ VMWare Cloud Director

برنامه Cloud Director برای تعریف مراکز داده نرم‌افزار پایه (SDDC)[۲] است که قابلیت‌های زیرساخت به عنوان خدمت (IaaS) را در اختیار مشتریان قرار می‌دهد. این ابزار به مشتریان اجازه می‌دهد تا منابع مجازی شده شامل ماشین‌های مجازی، شبکه‌ها و فضای ذخیره‌سازی را در یک محیط ابری مدیریت کنند. معماری VMware Cloud Director از چندین مؤلفه تشکیل شده است که با هم کار می‌کنند تا راه حلی بسیار در دسترس و مقیاس پذیر ارائه دهند. این اجزا در شکل۲-۱ مشخص شده‌اند.



شکل ۲-۱: اجزای تشکیل دهنده vCloud[؟]

مهم‌ترین مولفه‌ی تشکیل‌دهنده، خدمت‌گزار vCenter است که وظیفه‌ی مدیریت منابع مجازی در مرکزداده را بر عهده دارد. خدمت‌گزار vCenter با فراناظر[۳] vSphere ارتباط برقرار می‌کند تا عملیات کلیدی مانند ایجاد، استقرار و مدیریت ماشین‌ها و منابع مجازی را انجام دهد.

مولفه دیگر که برای مجازی‌سازی خدمات شبکه طراحی شده‌است، سکوی NSX است که وظیفه‌ی ارائه‌ی قابلیت‌های مجازی‌سازی شبکه به مرکزداده را بر عهده دارد. NSX مجازی‌سازی شبکه را با ایجاد شبکه‌های ایزوله از یکدیگر فراهم می‌کند و به چندین مستأجر[۴] اجازه می‌دهد تا زیرساخت فیزیکی مشابهی را بدون تداخل با یک‌دیگر به اشتراک بگذارند.

[2] Software Defined Data Center
[3] Hypervisor
[4] Tenant

مولفه‌ی مجازی‌سازی خدمات ذخیره‌سازی دائم، توسط vSAN ارائه شده‌است که یک سکوی ذخیره‌سازی نرم‌افزاری است که مشتریان را قادر می‌سازد منابع ذخیره‌سازی را در یک محیط مجازی مدیریت کنند. vSAN به مشتریان امکان ایجاد حجم[۵]های ذخیره‌سازی مجازی را می‌دهد که می‌تواند برای ذخیره‌ی فایل‌های ماشین مجازی و همچنین فضای ذخیره‌ی اشیا[۶] استفاده شود.

در نهایت، سکوی VMWare Cloud Director یک رابط برنامه‌نویسی ارائه می‌کند که به مشتریان اجازه می‌دهد زیرساخت ابری خود را مدیریت کنند. این رابط برنامه‌نویسی بر پایه‌ی استاندارد REST تعریف شده که می‌تواند برای عملیات مختلف مانند ایجاد و مدیریت ماشین‌های مجازی، شبکه‌ها و حجم‌های ذخیره‌سازی استفاده شود.

۲-۲ زبان برنامه‌نویسی

برای پیاده‌سازی سامانه‌ی مورد نظر باید از یک زبان برنامه‌نویسی وب استفاده کنیم. با توجه به حجم بالای درخواست‌ها و فشار بر روی خدمت‌گزار، سرعت زیاد و حجم و پیچیدگی کم در منطق سامانه، از معیارهای کلیدی انتخاب زبان برنامه‌نویسی است. یکی دیگر از این معیارها، محبوبیت و توسعه‌پذیری زبان برنامه‌نویسی در آینده برای نیازمندی‌های احتمالی و پیش‌بینی نشده است. با توجه به این معیارها، انتخاب ما برای توسعه‌ی این پروژه، زبان برنامه‌نویسی Go است.

۲-۲-۱ زبان برنامه‌نویسی Go

زبان Go یک زبان برنامه‌نویسی تابعی[۷] است که با گذر زمان محبوبیت زیادی کسب کرده و توسط بسیاری از شرکت‌ها و توسعه‌دهندگان برای ساخت برنامه‌های کاربردی مقیاس‌پذیر[۸] استفاده می‌شود. این زبان برای توسعه‌ی یک سرویس جامع ارائه‌ی خدمات IaaS، مزیت‌های قابل توجه و همچنین معایبی دارد که در ادامه به شرح دقیق‌تر آن‌ها می‌پردازیم.

[5] Volume
[6] Object Storage
[7] Functional
[8] Scalable

مزایای استفاده از Go برای پیاده‌سازی یک رابط برنامه‌نویسی مقیاس‌پذیر شامل این موارد اس[؟]:

۱. **هم‌زمانی[۹] به صورت ذاتی**: یکی از برجسته‌ترین مزایای Go، پشتیبانی قوی آن در مدیریت همزمانی است که آن را برای ساختن سیستم‌های بزرگ‌مقیاس که نیاز به استفاده‌ی کارآمد از منابع دارند، مناسب می‌کند. در یک محیط ابری که تقاضا برای منابع می‌تواند به سرعت تغییر کند، مدل پیاده‌سازی هم‌زمانی در Go، به‌کارگیری منابع را در حالتی ایده‌آل حفظ می‌کند.

۲. **مدیریت حافظه**: Go دارای یک جمع‌کننده زباله[۱۰] داخلی است که مدیریت حافظه را آسان‌تر می‌کند و بر خلاف زبان‌های سطح پایین مانند C و C++ این فرصت را به توسعه‌دهندگان می‌دهد تا روی عملکرد برنامه‌شان تمرکز کنند. این مورد همچنین به کاهش خطر نشت حافظه[۱۱] کمک می‌کند، که می‌تواند باعث ایجاد مشکلات پایداری و امنیتی شود.

۳. **سادگی**: Go دارای یک نحو[۱۲] ساده و شبیه به زبان‌های بسیار معروف Cو پایتون است که یادگیری و استفاده از آن را نسبت به زبان‌های دیگر برنامه‌نویسی آسان‌تر می‌کند.

۴. **کارایی**[۱۳]: Go برای کارایی بهینه طراحی شده‌است. برای سامانه‌ها و برنامه‌هایی که باید بار پویا و در مقیاس بالا را مدیریت کنند، یک گزینه مناسب محسوب می‌شود. کامپایل شدن برنامه‌ها و اجرای مستقیم بر روی سیستم عامل در کنار محیط اجرای بهینه، این زبان برنامه‌نویسی را از نظر زمان اجرا در کنار زبان‌های سطح پایینی همچون C++ قرار می‌دهد.[؟]

با وجود این مزیت‌ها، چالش‌هایی نیز برای استفاده از این زبان برنامه‌نویسی مطرح اس[؟]. از جمله:

۱. **محدود بودن جامعه**: باوجود محبوبیت رو به رشد این زبان برنامه‌نویسی، به دلیل سن کم‌تر نسبت به زبان‌های دیگر مانند JavaScript، Java و یا Python، این زبان برنامه‌نویسی از جامعه‌ی متن‌باز کوچک‌تری برخوردار است که به سبب آن محدودیت‌هایی برای کتاب‌خانه‌ها و قطعه کدهای خاص‌منظوره پدیدار شده است.

۲. **شیب یادگیری تند**: با وجود نحو آشنا و نزدیک Go، این زبان از اصول و ضرب‌المثل[۱۴]های خاصی پیروی می‌کند که ممکن است در چهارچوب‌ها و الگوهای زبان‌های دیگر حضور نداشته

[9] Concurrency
[10] Garbage Collector
[11] Memory Leak
[12] Syntax
[13] Performance
[14] Idiom

باشند. توسعه‌ی برنامه‌های شاخص و الگو، نیازمند رعایت این اصول است که این امر، اثرگذاری افراد با تجربه‌ی کم در توسعه‌ی برنامه‌های این زبان را کاهش می‌دهد.

با وجود این مسائل از بین گزینه‌های موجود، انتخاب نهایی ما زبان برنامه‌نویسی Go است.

۲-۳ استاندارد REST

در تمام برنامه‌های تحت وب، یکی از کلیدی‌ترین تصمیمات در طراحی، انتخاب پروتکل و استانداردهای مربوط به انتقال داده و اتصال برنامه به کاربر است. یکی از پرکاربردترین استانداردهای مورد استفاده در برنامه‌های تحت وب، استاندارد REST[۱۵] است.

عملیات محدود و وجود قوانین جامع، باعث سادگی تعریف و توسعه‌ی نقطه‌های دسترسی به منابع با این استاندارد شده. همچنین عدم نگهداری وضعیت بین درخواست‌ها باعث می‌شود که به سادگی بتوان برنامه‌ها را به صورت افقی نسبت به بار مقیاس کرد. با توجه به اینکه قوانین در این استاندارد کاملا مستقل از زیرساخت و سکوی توسعه و اجرا است، استفاده از این استاندارد، امکان استفاده‌ی گسترده‌ی بسیار وسیعی از کاربران از خدمات را فراهم می‌آورد.[؟]

همچنین زبان Go به صورت طبیعی از این استانداردها پیروی می‌کند و کتاب‌خانه و چهارچوب‌های متعددی از جمله چهارچوب Echo برای این منظور در جامعه‌ی متن‌باز این زبان، یافت می‌شود.

۲-۴ پایگاه ذخیره داده

برنامه‌ی ما نیازمند ذخیره‌سازی طیف گسترده‌ای از اطلاعات است. این اطلاعات شامل تنظیمات و اطلاعات مورد نیاز هر بخش از سامانه، رخدادها و وقایع رخ داده در سامانه و منابع و مدل‌های مورد استفاده‌ی کاربران می‌شود. برای ذخیره‌سازی این اطلاعات، انتخاب‌های متعددی برای پایگاه داده داریم که در این پروژه بسته به ماهیت داده و نوع وابستگی برنامه به آن، از دو دسته‌ی کلی پایگاه‌های داده رابطه‌ای و غیر رابطه‌ای استفاده می‌کنیم.

[15] Representational State Transfer

۲-۴-۱ پایگاه داده رابطه‌ای

پایگاه داده‌های رابطه‌ای، به دسته‌ای از پایگاه‌های داده گفته می‌شود که بر مبنای زبان پرسمان ساختاریافته (SQL[۱۶]) تعریف شده‌اند. در این پایگاه‌های داده، اطلاعات در جداولی با روابط تعریف‌شده، سازماندهی می‌شوند و پرسمان‌ها را می‌توان با استفاده از SQL برای بازیابی داده‌های خاص ایجاد کرد. نمونه‌هایی از پایگاه داده‌های SQL عبارتند از MySQL، Oracle و Microsoft SQL Server. پایگاه‌های داده‌ی SQL برای برنامه‌هایی که نیاز به تراکنش‌های پیچیده دارند و شامل مقادیر زیادی از **داده‌های ساختاریافته** هستند؛ مانند سیستم‌های مالی، سامانه‌های مدیریت کاربران و تعریف منابع با قابلبیت تغییر مکرر مناسب هستند.

به طور کلی، اطلاعات ساختاریافته‌ای را که می‌توان در جداول با ستون‌های ثابت تعریف کرد، در این پایگاه داده ذخیره می‌کنیم. در این پروژه، ما از پایگاه داده‌ی **PostgreSQL** که یکی از معروف‌ترین پایگاه داده‌های رابطه‌ای متن‌باز است استفاده می‌کنیم.

۲-۴-۲ پایگاه داده غیر رابطه‌ای

در مقابل پایگاه داده‌های رابطه‌ای، پایگاه داده‌های غیر رابطه‌ای (NoSQL[۱۷]) تعریف می‌شوند. این دسته از پایگاه‌های داده به گونه‌ای طراحی شده اند که بتوانند حجم بسیار زیادی از اطلاعات نیمه ساختار یافته[۱۸] را پردازش کنند. این اطلاعات همانگونه که از مفهومش برداشت می‌شود ساختار ثابت و جدول مانند ندارد، بلکه حالت‌های متنوعی از جمله کلید-مقدار، سند پایه، گراف پایه و ستون پایه را شامل می‌شود[؟].

در این پروژه ما رخدادهای سامانه که ساختار یکتایی ندارند و همچنین از حجم بسیار بالایی برخوردار هستند را در پایگاه داده غیر رابطه ای ذخیره می‌کنیم. ابزاری که برای این منظور استفاده می‌کنیم پایگاه داده‌ی **MongoDB** است که یک پایگاه داده‌ی متن‌باز است.

[16] Structured Query Language
[17] Not Only SQL
[18] Semi Structured Data

۲-۵ معماری میکروسرویس

معماری میکروسرویس[۱۹] یک الگوی طراحی[۲۰] محبوب است که در توسعه‌ی نرم‌افزارهای مدرن برای ایجاد برنامه‌های کاربردی مقیاس‌پذیر، قابل نگهداری و انعطاف‌پذیر استفاده می‌شود. این معماری بر اساس ایده‌ی تجزیه‌ی یک سیستم نرم‌افزاری پیچیده به سرویس‌های کوچکتر و مستقل است که می‌توانند به طور مستقل، توسعه یافته، استقرار یابند و مدیریت شوند. این رویکرد چندین مزیت از جمله بهبود مقیاس‌پذیری، افزایش تحمل خطا و سرعت توسعه را ارائه می‌دهد.

میکروسرویس مبتنی بر مفهوم معماری سرویس‌گرا (SOA[۲۱]) است که شامل تجزیه‌ی یک سیستم نرم‌افزاری پیچیده به اجزای کوچکتر و مستقل است. با این حال، معماری میکروسرویس، این مفهوم را با تعریف هر مؤلفه به عنوان یک سرویس جداگانه که برای انجام یک عملکرد واحد طراحی شده است، یک قدم جلوتر می‌برد. این رویکرد به درجه بسیار بالاتری از انعطاف‌پذیری منجر می‌شود، زیرا هر سرویس می‌تواند مستقل از سایر سرویس‌ها مدیریت و دچار دخل و تصرف شود.

بخشی از مزایای این معماری در برنامه‌های داده‌محور[۲۲] و رخدادمحور [۲۳] در زیر توضیح داده شده[؟][؟]:

- **سرعت توسعه نرم‌افزار**: با توجه به مستقل بودن واحدهای تعریف‌شده‌ی برنامه در این معماری، تیم‌ها و افراد مختلف می‌توانند مستقل و موازی بر توسعه سامانه کار کنند. این امر باعث افزایش قابل توجه سرعت و چابکی توسعه برنامه‌ها می‌شود. همچنین با توجه به تعریف روابط میان سرویس‌ها از قبل، اعمال و اجرای تغییرات درونی برنامه‌ها بسیار راحت‌تر و سریع‌تر انجام می‌شود.

- **مقیاس پذیری**: به دلیل جدا و منزوی بودن سرویس‌ها در این معماری، مقیاس هر سرویس را می‌توان متناسب با بار سامانه تنظیم کرد. این مورد به این معنی است که می‌توان قسمت‌های مختلف سامانه را در هر زمان کم و زیاد یا بزرگ و کوچک کرد.

- **تحمل خطا**: هرگونه خطا و اخلال در یک سرویس در معماری میکروسرویس، از قبل پیش‌بینی شده است و همواره می‌توان تمهیدات لازم جهت تحمل خطا را در سرویس‌های دیگر از پیش

[19] Microservice
[20] Design Pattern
[21] Service-Oriented Architecture
[22] Data Driven
[23] Event Driven

آماده کرد. همچنین اخلال در یک سرویس، فقط در همان سرویس اثرگذار است و عملکرد سایر سرویس‌های مستقل دچار اختلال نمی‌شود.

با وجود این مزایا، این معماری چالش‌ها و معایبی را نیز به دنبال دارد[؟][؟] که در ادامه به بررسی آنها می‌پردازیم.

- **پیچیدگی طراحی**: هنگام طراحی میکروسرویس‌های موجود در یک سامانه، باید تمامی روابط و هماهنگی‌های میان‌سرویسی را از قبل تعریف کرد. هرگونه خطا و کاستی در این مرحله، منجر به چالش‌های اساسی در مرحله اجرا و توسعه می‌شود که برطرف کردن آنها نیازمند هزینه‌ی بیشتری است. پس باید نهایت دقت را در مرحله طراحی میکروسرویس‌ها به خرج داد.
- **زیرساخت**: اجرا و پیاده‌سازی میکروسرویس‌ها نیازمند پیش‌نیازها و زیرساخت‌های خاص و ابزارهای متعدد است که هزینه استفاده از این معماری را برای برنامه‌هایی با مقیاس کم و دامنه امکانات محدود، زیاد می‌کند. نیاز این معماری به تضمین زیرساخت اجرایی و شبکه‌ای از جمله‌ی این پیش‌نیازها است.
- **حفظ سازگاری داده**[۲۴]: با گسترده شدن داده‌ها میان سرویس‌های مختلف و احتمال وجود خطا در هر قسم[۲۵] از سامانه، تضمین سازگاری داده بدون ضمانت سلامت زیرساخت امری بسیار دشوار است.
- **سربارهای اضافی**: استفاده از سرویس‌های متعدد برای سامانه‌ها با دامنه فعالیت‌های محدود، باعث به‌وجود آمدن سربارهای کد، شبکه و محاسبات می‌شود.

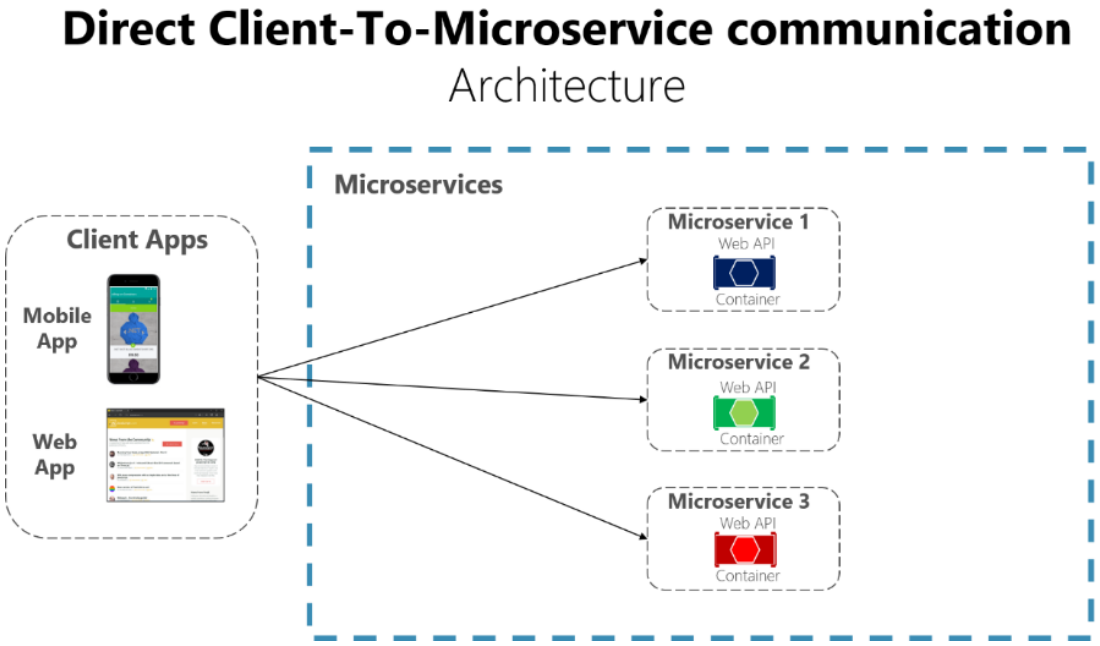
در نهایت، براساس موارد مطرح شده، برای تصمیم‌گیری در مورد استفاده از معماری میکروسرویس، باید دامنه‌ی فعالیت‌ها، فشار ناشی از بار در قسمت‌های مختلف، محدودیت‌های تیم توسعه و محدودیت‌های زیرساخت سامانه را در نظر گرفت. با در نظر گرفتن این موارد، در این پروژه معماری مورد استفاده‌ی ما، معماری میکروسرویس است.

[24] Data Consistency
[25] Partition

۲-۶ دروازه‌ی ورود رابط

الگوی طراحی دروازه‌ی ورود رابط[۲۶]، یکی از الگوهای رایجی است که در کنار میکروسرویس‌ها استفاده می‌شود. در این الگو، برنامه‌ای به عنوان یک پیش‌کار[۲۷] که وظیفه‌ی مسیریابی درخواست‌های ورودی را بر عهده دارد، قرار داده می‌شود. مدیریت بخش بزرگی از موارد امنیتی، نظارتی و سلامتی میکروسرویس‌ها بر عهده این لایه قرار می‌گیرد.

در حالت پیش‌فرض شکل۲-۲، کاربران سامانه مستقیما با میکروسرویس‌ها در ارتباط هستند که این امر ملاحظات جدی امنیتی و کارایی به دنبال دارد. در این حالت، تمامی میکروسرویس‌ها باید عملیات مشترکی همچون احراز هویت، واقعه‌نگاری ونظارت را پیاده‌سازی کنند.



شکل ۲-۲: معماری یک سامانه میکروسرویس بدون دروازه‌ی ورود رابط[؟]

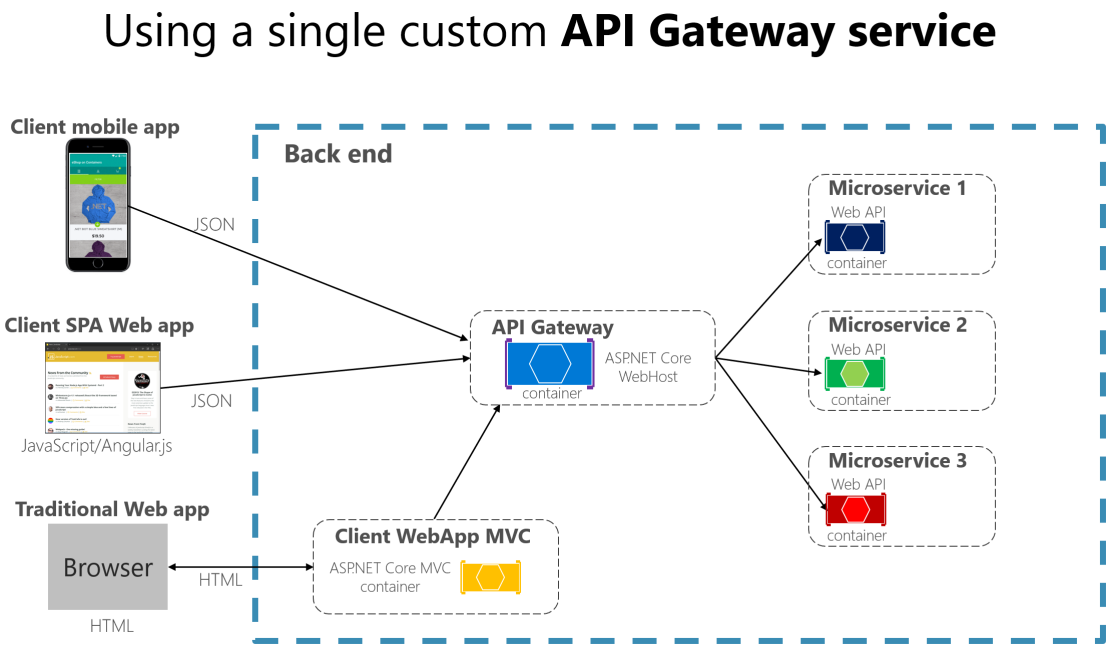
در مقابل معماری شکل۲-۲، معماری سامانه در صورت استفاده از یک دروازه‌ی ورود رابط، مانند شکل۲-۳ می‌شود. در این معماری، تعداد قابل توجهی از عملیات مشترک را می‌توان در لایه دروازه‌ی ورود انجام داد. به طور کلی، مزیت‌های استفاده از دروازه‌ی ورود رابط را می‌توان به شکل زیر بیان کرد[؟]:

- **مدیریت متمرکز**: هنگامی که دروازه‌ی مکاتبه و تعامل با تمامی کاربران سامانه ثابت و مشخص

[26] API Gateway
[27] Proxy

باشد، مدیریت نحوه‌ی این ارتباط کاملا مستقل از عملکرد درونی سامانه می‌شود و می‌توان نظارت و مدیریت کامل‌تری بر روی درخواست‌های کاربران و پاسخگویی به آن‌ها داشت.

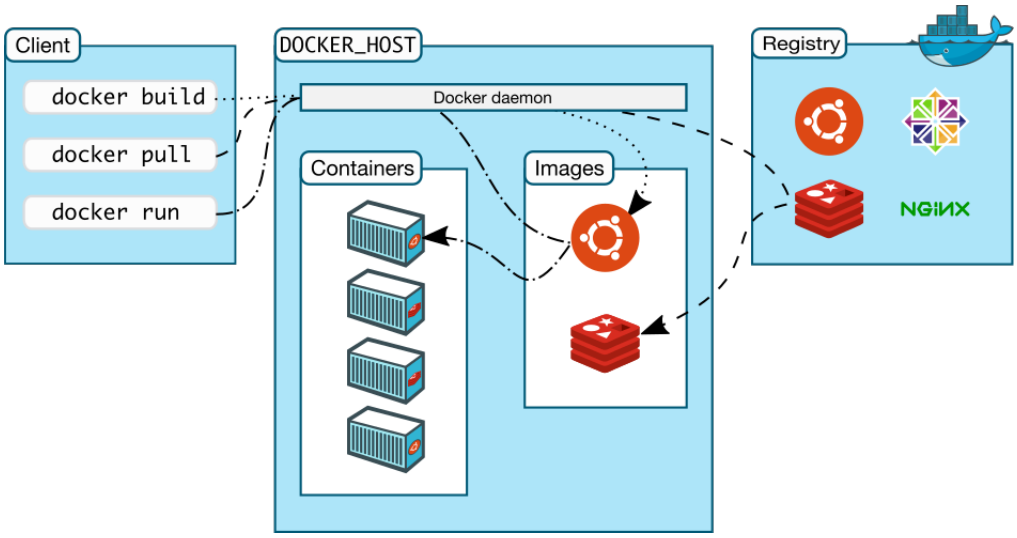
- **امنیت**: با وجود یک لایه‌ی کاملا مستقل در پیش‌روی کاربر، می‌توان مکانیزم‌های امنیتی نظیر احراز هویت و مجوزها را در کنار موارد امنیت شبکه یکجا مدیریت کرد. این گونه میکروسرویس‌ها، کاملا درون یک شبکه منزوی و غیر قابل دسترس از بیرون تجمیع می‌شوند که خطر ریسک‌های امنیتی را به شکل قابل توجهی کاهش می‌دهد.
- **تقسیم بار**: لایه دروازه‌ی ورود می‌تواند بار ورودی را بین نسخه‌های متعدد میکروسرویس‌ها تقسیم کند و در صورت نیاز اتصال به بخش‌های خاصی از سامانه را کمتر یا زیادتر کند.
- **نظارت**: با پیاده‌سازی واحدهای نظارتی در دروازه‌ی ورود، تمامی رفتار کاربران و نحوه برخورد سامانه با آن‌ها را یکجا و به طور متمرکز در اختیار خواهیم داشت که برای مقاصد ایرادیابی و تحلیلی فواید زیادی دارد.



شکل ۲-۳: معماری یک سامانه میکروسرویس با وجود یک دروازه‌ی ورود رابط[؟]

۲-۷ داکر

داکر[۲۸] یکی از کاربردی‌ترین ابزارها برای پیاده‌سازی معماری میکروسرویس است. این ابزار که خود از چندین بخش تشکیل شده، امکان بسته‌بندی برنامه‌ها در واحد کانتینر[۲۹] را فراهم می‌کند و کاربران می‌توانند این کانتینرها را در یک موتور کانتینر[۳۰] مثل موتور داکر[۳۱] اجرا کنند. با انجام این کار، کانتینرها به صورت مستقل از یکدیگر در یک محیط منزوی اجرا می‌شوند. فواید این مدل اجرا، شامل مقیاس پذیری راحت، قابل حمل بودن واحدهای اجرایی برنامه، سازگاری و انعطاف پذیری نسبت به محیط اجرا و ابزارها است. معماری کلی داکر و فرایند عملکرد این ابزار در شکل ۲-۴ مشخص شده است. کاربران با استفاده از برنامه docker با سرویس docker daemon که همواره در پس‌زمینه درحال اجرا است، در تعامل خواهند بود. این برنامه سپس تصاویر[۳۲] را از مخزن محلی و یا مرکزی دریافت کرده و در قالب کانتینرهای موتور داکر اجرا می‌کند.

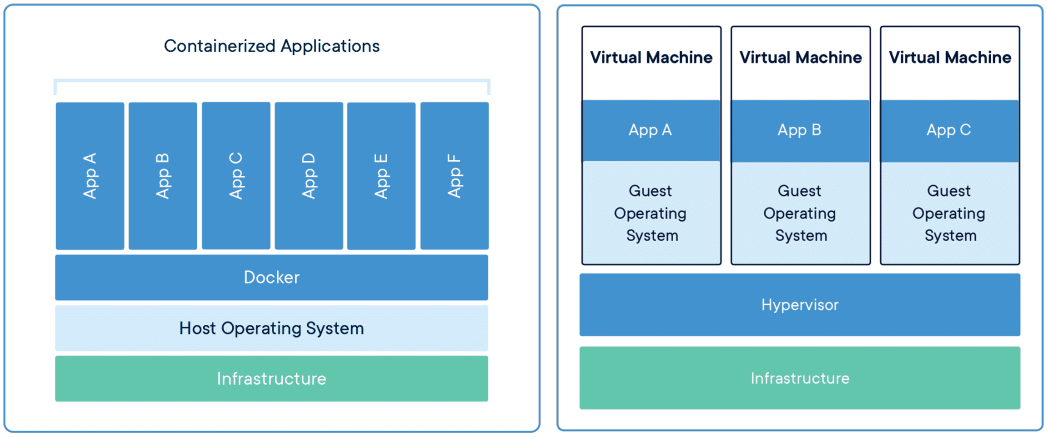


شکل ۲-۴: معماری داکر

تفاوت کانتینرهای داکر با ماشین‌های مجازی که در گذشته به همین منظور استفاده می‌شدند در شکل ۲-۵ قرار گرفته. برخلاف ماشین مجازی، کانتینرها توسط موتور کانتینر (در این شکل موتور داکر)

[28] Docker
[29] Container
[30] Container Engine
[31] Docker Engine
[32] Images

بر روی سیستم عامل میزبان اجرا می‌شوند. این مدل اجرا، باعث از بین رفتن سربارهای مجازی سازی دستورات و سخت افزار می‌شود.



شکل ۲-۵: لایه‌های کانتینر داکر

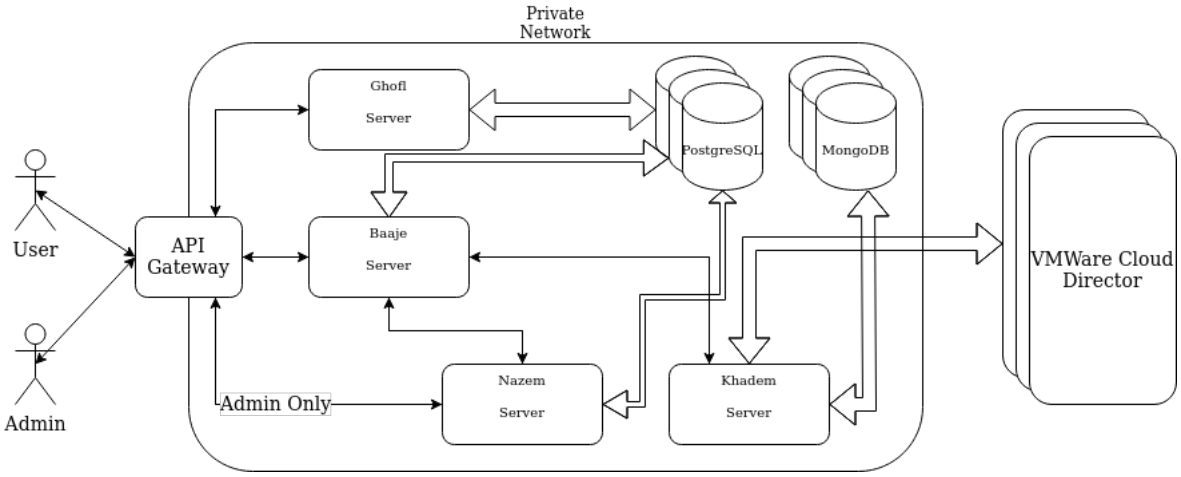
۲-۸ نتیجه گیری

در این پروژه، ما به عنوان زیرساخت ارائه خدمات IaaS از سرویس VMWare Cloud Director بهره گرفتیم.برای سرویس‌های مورد نیاز برای توسعه خدمات، از زبان برنامه‌نویسی Go و از چهارچوب‌ها و کتاب‌خانه‌های این زبان استفاده کردیم. رابط‌های سامانه با کاربران بیرون و تعاملات درونی سامانه از استاندارد REST پیروی می‌کنند و کتاب‌خانه Echo در زبان برنامه‌نویسی Go برای پیاده‌سازی این موضوع به کار گرفته شده. برای ذخیره داده از فناوری‌های پایگاه داده رابطه‌ای و غیر رابطه‌ای استفاده کردیم. راه اندازی پایگاه داده رابطه‌ای با ابزار PostgreSQL است و برای پایگاه داده غیر رابطه‌ای از ابزار MongoDB استفاده کرده‌ایم.

پیاده‌سازی سرویس‌ها در قالب معماری میکروسرویس انجام‌شد که تمامی قسمت‌های سامانه در قالب تصاویر داکر قرار گرفتند. این سرویس‌ها داخل یک شبکه داخلی پشت یک دروازه‌ی ورود رابط جای گرفتند که این دروازه‌ی ورود رابط، وظیفه‌های نظارت، احراز هویت و ثبت وقایع را برعهده دارد.

فصل سوم

طراحی و پیاده‌سازی سیستم



شکل ۳-۱: معماری کلی سامانه

در این فصل، به مراحلی که برای پیاده‌سازی این سیستم طی شده‌است، پرداخته شده‌است.

۳-۱ طراحی و معماری سیستم

با توجه به نیازمندی‌های مطرح شده برای خدمات سامانه، در ابتدا باید دامنه‌ی فعالیت‌های بخش‌های مختلف سامانه را مشخص کنیم. این تصمیم گیری باید به گونه‌ای باشد که بخش‌ها، کمترین وابستگی و جفت شدگی[1] را داشته باشند.

با توجه به استقلال نیازمندی‌ها و عملکردهای ذکر شده برای سامانه، معماری کلی این سامانه، معماری میکروسرویس است. در این پیاده‌سازی، هر نیازمندی در یک سرویس پیاده‌سازی شده است، یک سرویس جهت انجام امور احراز هویت، یک سرویس برای تعامل با کاربران و تحویل درخواست‌های آنان، یک سرویس جهت تعامل با زیرساخت ابری جهت اجرای دستورات و در نهایت یک سرویس جهت امور نظارتی و مدیریتی تعریف شده که همه این سرویس‌ها در یک شبکه‌ی خصوصی و پشت سر یک دروازه‌ی ورود رابط قرار گرفته. معماری کلی سامانه در شکل۳-۱ مشخص شده است. در ادامه‌ی این فصل جزئیات هر بخش از این معماری را دقیق‌تر بررسی می‌کنیم.

[1]Coupling

۳-۲ میکروسرویس‌ها

کل سامانه از چهار میکروسرویس تشکیل شده: قفل، ناظم، خادم و باجه. هر میکروسرویس، مخصوص رسیدگی به امور بخشی از سامانه است. این میکروسرویس‌ها در قالب معماری کد تک مخزنه[۲] پیاده‌سازی شده‌اند و دارای اشتراک‌های زیادی در کدهای پایه هستند.

۳-۲-۱ جزئیات فنی مشترک

در این بخش قصد داریم که این موارد مشترک را مطرح کنیم و سپس به بررسی جزئیات اختصاصی هر سرویس بپردازیم.

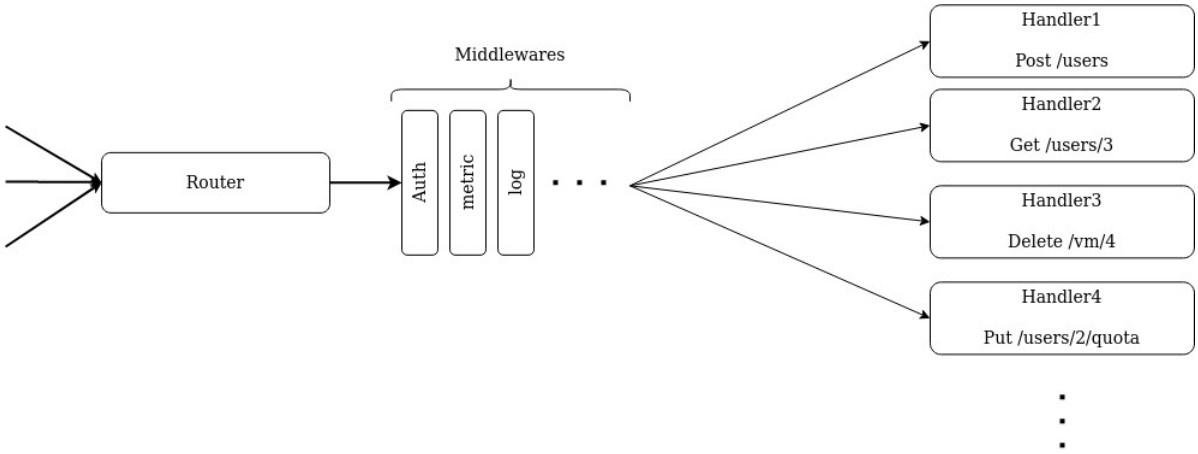
این موارد شامل:

- خدمت‌گزار وب
- برنامه‌ی تحت خط‌فرمان
- مدیریت تنظیمات
- تعامل با پایگاه داده و مدیریت مدل‌ها
- ساختار کد

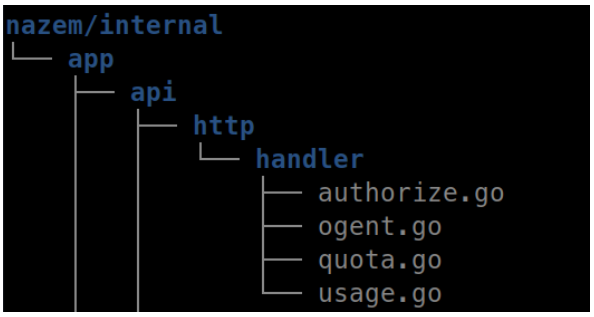
می‌شود که در ادامه‌ی این بخش به شرح جزئیات هریک می‌پردازیم.

اجرای یک خدمت‌گزار وب[۳] توسط چهارچوب Echo در زبان Go انجام می‌شود. نحوه عملکرد این چهارچوب به این شکل است که در ابتدا قوانین مربوط به مسیریاب[۴] تنظیم می‌شود، سپس برای هر مقصد[۵]، یک رسیدگی کننده[۶] تعریف می‌کنیم. به عنوان مثال برای عملیات ورود کاربر به سامانه، برای مسیر /api/login یک رسیدگی کننده تعریف می‌کنیم که عملیات لازم جهت ورود کاربر را اجرا می‌کند. این عملیات در این مثال شامل چک کردن ورودی درخواست، اتصال به پایگاه داده، چک کردن صحت اطلاعات فرستاده شده و ارسال یک کلید ورود برای کاربر است که همگی در قالب یک تابع به عنوان

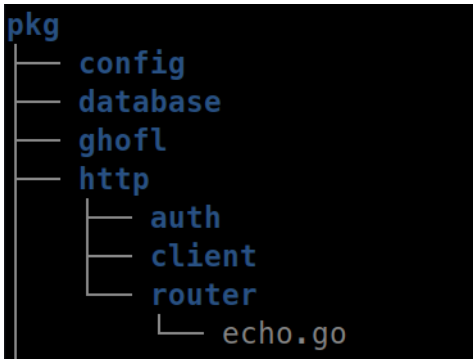
[2] Mono Repo
[3] Web Server
[4] Router
[5] Endpoint
[6] Handler



شکل ۳-۲: عملکرد چهارچوب Echo



شکل ۳-۳: پوشه‌بندی توابع رسیدگی‌کننده Echo



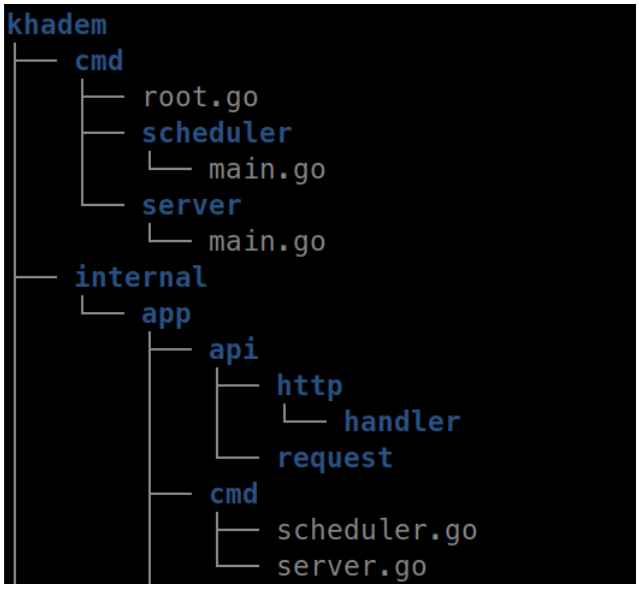
شکل ۳-۴: پوشه‌بندی مسیریاب Echo

رسیدگی کننده‌ی این مقصد تعریف می‌شوند. همچنین در تمامی خدمت‌گزارهای این سامانه، کاربران بایستی هویت خود را از طریق سربرگ‌های اجباری از پیش تعیین شده مشخص کنند و یک احراز هویت با کلید ثابت انجام دهند. این امر جهت بالا بردن امنیت دوچندان سامانه تعبیه شده‌است. در ساختار کد این پروژه، ثبت این مقصدها به همراه توابع رسیدگی کننده در بسته api/http/handler انجام می‌شود. جهت ثبت یک بسته جدید فقط کافیست که مشخصات مقصد و تابع رسیدگی کننده در فایلی داخل این بسته نوشته شود و با تابع RegisterEndpoints مقصد مورد نظر داخل مسیریاب Echo ثبت شود.

معماری کلی عملکرد Echo در شکل ۳-۲ و پوشه‌بندی موارد گفته شده در شکل ۳-۴ و ۳-۳ توضیح داده شده.

ایجاد برنامه‌های تحت خط‌فرمان[۷] توسط چهارچوب Cobra پیاده‌سازی شده. این چهارچوب به

[7] Command Line Application



شکل ۳-۵: ساختار پوشه‌بندی برنامه‌های خط فرمان در پروژه خادم

برنامه‌نویسان این اختیار را می‌دهد که برای برنامه‌های خود یک رابط خط فرمان[۸] ایجاد کنند. با این کار، برنامه‌نویس می‌تواند با کدهای نوشته شده برای پروژه، چندین فرمان قابل اجرا ایجاد کند. مزیت این‌کار در سناریوهایی که برنامه‌ی نوشته شده نیاز به اجرای چند فرمان مختلف به طور همزمان در پس‌زمینه داشته باشد مشخص می‌شود.

برنامه های خط فرمان طبق پوشه‌بندی شکل۳-۵ در پروژه‌ها قرار گرفته‌اند.

مدیریت تنظیمات برنامه توسط چهارچوب Viper انجام می‌شود. این چهارچوب، به کاربران این اجازه را می‌دهد که تنظیمات را در هریک از ساختارهای معروف JSON، YAML و یا TOML بنویسد و با قراردادن آن در یکی از آدرس‌های پیشفرض تنظیمات، آن‌ها را بر روی برنامه اعمال کند. ساختار انتخابی ما برای تنظیمات به صورت پیش‌فرض، ساختار YAML است که در یک فایل هم‌اسم پروژه، در پوشه ریشه پروژه قرار گرفته.

تنظیمات برنامه می‌تواند در یکی از آدرس‌های زیر باشد.

۱. ./<appname>.yaml

۲. /etc/<appname>/<appname>.yaml

۳. /.config/<appname>.yaml

[8]Command Line Interface

تعامل و ارتباطات با پایگاه داده در اپلیکیشن‌ها با دو چهارچوب gorm و ent انجام می‌شود. مزیت چهارچوب gorm، سادگی عملکرد و گستردگی پشتیبانی و جامعه متن‌باز آن است. هر دو چهارچوب به عنوان یک ORM[9] تعامل با پایگاه داده را بسیار ساده‌تر می‌کنند. به گونه‌ای که برای ارتباط با پایگاه‌داده به هیچ‌عنوان دغدغه‌ی نحوه تعریف روابط و ذخیره‌سازی اطلاعات را نداریم و فقط با مدل‌های تعریف شده‌ی داخل برنامه تعامل برقرار می‌کنیم. در کد این پروژه، ارتباطات با پایگاه داده جهت دریافت و ثبت اطلاعات در یک لایه‌ی کاملا مجزا با استفاده از الگوی شیئ دسترسی داده[10] پیاده‌سازی شده است. این الگوی طراحی به نویسندگان کد این امکان را می‌دهد، که توابع و نیازمندی‌های دسترسی به داده ذخیره‌شده را در یک لایه‌ی کاملا جداگانه تعریف کنند و لایه‌های دخل و تصرف داده را در کنار این لایه توسعه دهند[؟].

با توجه به این‌که چهارچوب ent توسط تولید کد رابط‌های مورد نیاز جهت تعامل با پایگاه داده را فراهم می‌کند، تمامی ریسک‌ها و خطرهای امنیتی مربوط به پایگاه داده به علت Type Safe بودن این رابط قابل کنترل می‌شود. این مورد یکی از مزیت‌های بسیار بزرگ استفاده از این چهارچوب است.

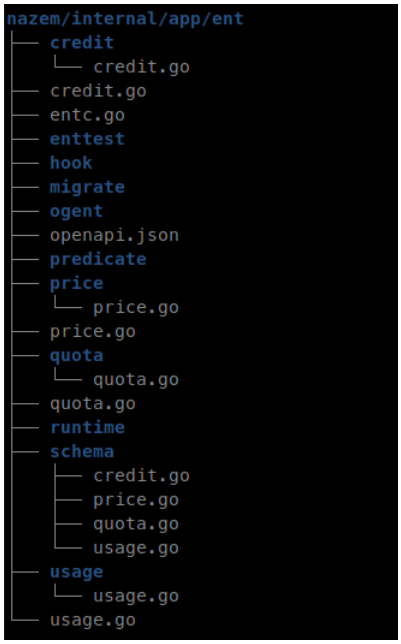
تمامی کدهای مربوط به تعامل با پایگاه داده مطابق شکل۳-۶، در پوشه ent قرار گرفته‌است. جزئیات مهم این پوشه‌بندی به شرح زیر می‌باشد.

- **schema**: این پوشه شامل تعاریف ساختار مدل‌ها در پایگاه داده است.
- **entc.go**: در این فایل، قوانین تولید و تفسیر API تعامل با پایگاه داده قرار داده شده است.
- **generate.go**: این فایل، نحوه تولید فایل‌های خودکار تولیدشده را مشخص می‌کند.
- **openapi.json**: این فایل، ساختار و مشخصات API تولیدشده را بر اساس استاندارد OpenAPI مشخص می‌کند.

با توجه به استفاده از الگوی تک‌مخزنه در نگهداری کدهای پروژه، تا حد ممکن کدها و منابع مشترک پروژه در پوشه pkg نگهداری می‌شود.

ساختار این پوشه در شکل۳-۷ مشخص شده، توضیحات پوشه‌های موجود در این ساختار به شرح زیر می‌باشد.

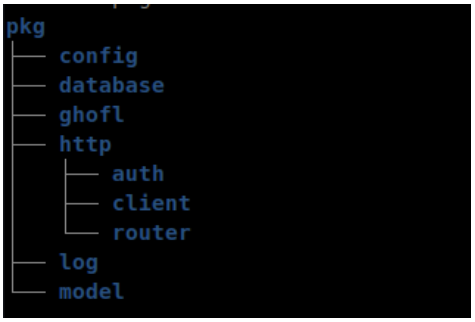
[9] Object-Relation Mapping
[10] Data Access Object



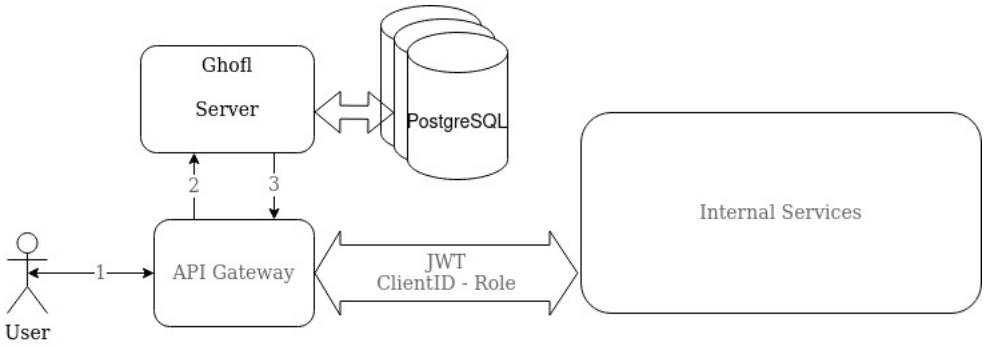
شکل ۳-۶: ساختار پوشه‌بندی چهارچوب ent

- **config**: نحوه خوانده شدن تنظیمات و قوانین مربوط به پردازش تنظیمات که بین پروژه‌ها مشترک است در فایل‌های این پوشه نوشته شده.
- **database**: کدهای مربوط به نحوه اتصال به پایگاه داده و مکانیزم‌های تلاش مجدد در فایل‌های این پوشه قرار داده شده، در صورت نیاز به پشتیبانی از پایگاه داده جدید، فقط کافیست نحوه اتصال به پایگاه داده را داخل این پوشه مشخص کنیم.
- **http**: میان‌افزارهای مشترک، تنظیمات مسیریاب اولیه برنامه، به همراه فایل‌های کاربر[11]های برنامه‌ها در این پوشه تعبیه شده.
- **log**: تنظیمات مربوط به راه‌اندازی ثبت کننده وقایع در این پوشه انجام می‌شود.
- **model**: مدل‌های مشترک نظیر ماشین مجازی یا تنظیمات آن در هر یک داخل فایلی داخل این پوشه قرار داده می‌شود.

[11] Client



شکل ۳-۷: ساختار پوشه‌بندی pkg



شکل ۳-۸: عملکرد سرویس قفل

۳-۲-۲ قفل

میکروسرویس مخصوص احراز هویت و تعیین سطح دسترسی در این سامانه، قفل نام دارد. دلیل نام‌گذاری این میکروسرویس به این اسم، شباهت عملکرد قفل و این سامانه است که هر دو وظیفه کنترل دسترسی به منابع پشت قفل را دارند.

در شکل۳-۸، کاربرد سرویس قفل در سامانه مشخص شده‌است. **تمامی درخواست‌های ورودی** به سامانه در ابتدا به سرویس قفل فرستاده می‌شوند و احراز هویت آن‌ها انجام می‌شود. این درخواست‌ها یا مانند درخواست ورود و ثبت‌نام ماهیت احراز هویت دارند و یا سایر درخواست‌های کاربری و مدیریتی هستند. دسته اول درخواست‌ها مستقیما به قفل فرستاده می‌شوند. دسته دوم درخواست‌ها نیز هنگام رسیدن به دروازه ورود، ابتدا به قفل فرستاده می‌شوند که سرایند احراز هویت JWT[۱۲] آنها ارزیابی شود. پس از این ارزیابی، در صورت موفقیت آمیزبودن. درخواست به سرویس‌های بالادستی فرستاده شده و

[12]JSON Web Token

درصورت ورود ناموفق کاربر با خطا مواجه خواهد شد.

۳-۲-۳ ناظم

میکروسرویس ناظم وظیفه نظارت و مدیریت امور مالی و گزارشات مصرف کاربران را برعهده دارد. این میکروسرویس یکی از کلیدی‌ترین سرویس‌های سامانه است که در تعامل زیاد با پایگاه داده است. معماری کلی این سرویس در شکل۳-۹ مشخص شده.

ناظم در **تعامل با باجه** در جایگاه یک اعتبارسنج[۱۳] درخواست عمل می‌کند. به این صورت که تمامی درخواست‌های مربوط به دخل و تصرف در منابع باید ابتدا از این سامانه تاییدیه بگیرند. گرفتن این تاییدیه به معنی اعتبار کافی و نبودن محدودیت بر روی کاربر برای منابع درخواستی است. هرگونه ایراد و خطا در این سرویس منجر توقف عملکرد کل سامانه است که این امر یک موضوع پیش‌بینی شده و قابل قبول است. کدهای مربوط به این اعتبار سنجی داخل بسته validation قرار گرفته و برای توسعه و تغییر این قوانین باید این بسته دچار تغییر شود.

در تعامل با دروازه‌ی ورود، درخواست‌های مدیر سامانه جهت تغییر محدودیت‌های کاربران یا گزارش گیری از وضعیت آنان را دریافت می‌کند و پاسخ مناسب را با توجه به اطلاعات ثبت شده در پایگاه داده می‌دهد.

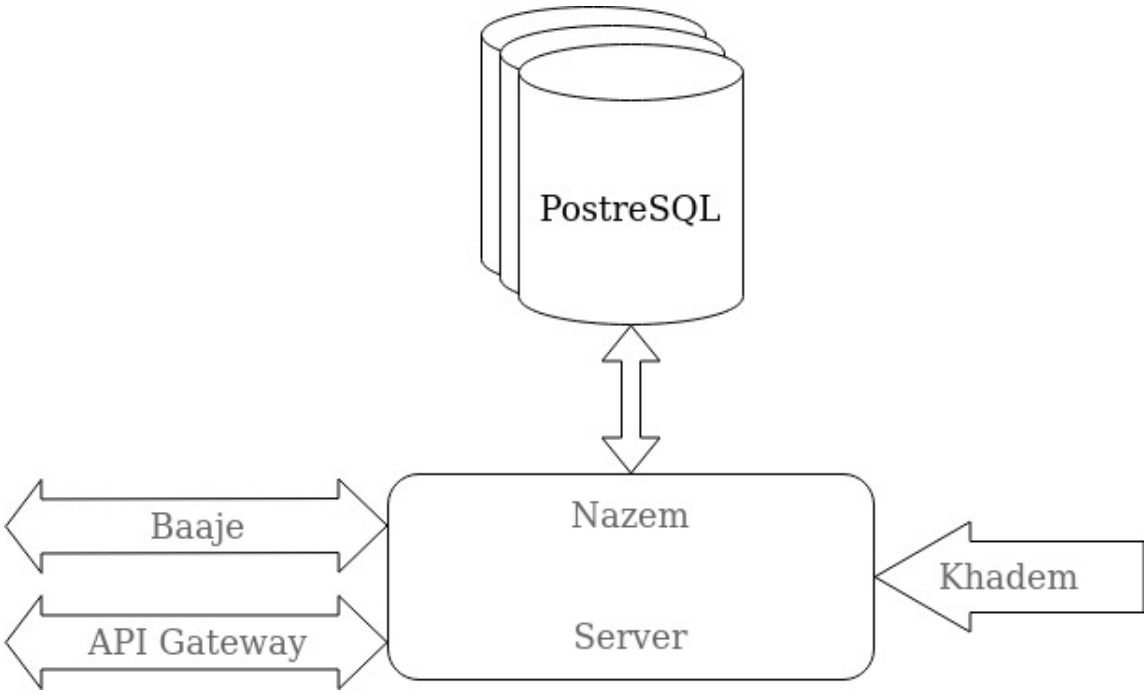
این سرویس برای درخواست‌های دریافتی از سرویس خادم، گزارش مصرف منابع را برای منابع دریافت می‌کند و با توجه به قوانین و قیمت گذاری‌های ثبت‌شده، اعتبار کاربر را تحت تاثیر قرار می‌دهد.

۴-۲-۳ باجه

این سرویس دروازه ارتباطی کاربران عادی سامانه است. تمامی درخواست‌ها و عملیات ممکن برای کاربر در این برنامه قرار گرفته. معماری این برنامه در شکل۳-۱۰ مشخص شده است. همانطور که در تصویر مشخص شده، این سرویس مستقیما از طریق دروازه‌ی ورود با کاربران در ارتباط است و برای انجام امور مربوط به کاربران با سرویس‌های خادم و ناظم به صورت داخلی در ارتباط است.

این سرویس اطلاعات مربوط به منابع تعریف شده برای کاربر را داخل پایگاه داده ذخیره کرده و در پردازش درخواست‌ها از این اطلاعات استفاده می‌کند. منابع تعریف شده داخل پایگاه داده شده داخل پیوست این گزارش موجود است.

[13] Validator



شکل ۳-۹: عملکرد سرویس ناظم

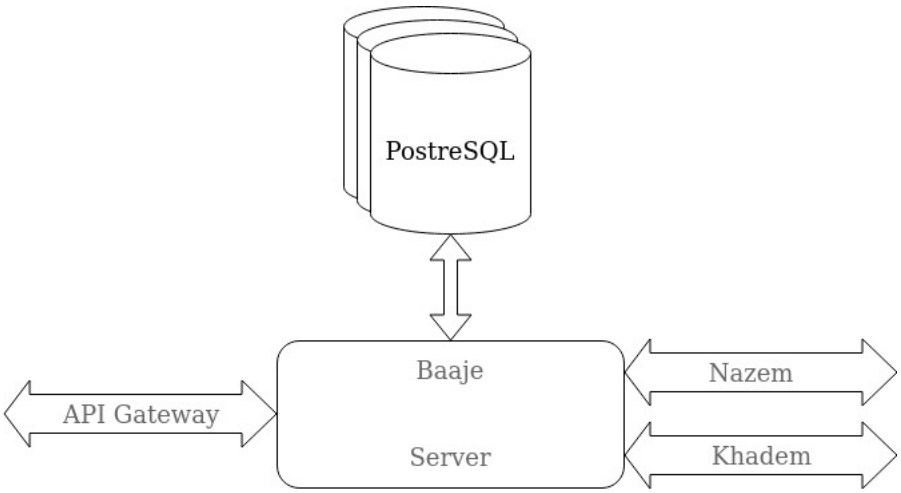
در مواجه با درخواست‌های مربوط به تغییر منابع مانند ساخت یا ویرایش ماشین مجازی، ابتدا اعتبارسنجی درخواست از طریق سرویس ناظم انجام می‌شود و در صورت موفقیت‌آمیز بودن این اعتبارسنجی، درخواست جهت اجرا به سرویس خادم ارسال می‌شود. پس از رسیدن جواب از سمت سرویس خادم، بسته به خروجی سرویس خادم، نتیجه درخواست کاربر داخل پایگاه داده ذخیره می‌شود.

۳-۲-۵ خادم

همانطور که قبلا گفته شد، سامانه برای ارائه خدمات IaaS نیازمند تعامل با یک ارائه دهنده زیرساخت است. این ارائه دهنده در سامانه‌ما VMWare Cloud Director است، میکروسرویس مسئول جهت تعامل با این برنامه خادم است. معماری خادم از دو قسمت اصلی تشکیل شده است. **خدمت‌گزار** و **زمان‌بند**[۱۴] دو بخش تشکیل دهنده این سرویس می‌باشند که کاملا مستقل و جداگانه اجرا می‌شوند. وظیفه خدمت‌گزار دریافت دستورات از سرویس باجه و سپس تفسیر و اجرای آن‌ها است. معماری کلی این پروژه در شکل ۳-۱۱مشخص شده است.

پایه کد این پروژه به طور کلی در بسته vcloud قرار داده شده. این بسته شامل تمامی عملیات قابل

[14]Scheduler



شکل ۳-۱۰: عملکرد سرویس باجه

اجرا بر روی cloud director است. در صورت نیاز به تکمیل و یا توسعه خدمات بایستی دستورات و توابع این بسته را تغییر داد. داخل این بسته از بسته توسعه نرم‌افزار (SDK[۱۵]) رسمی VMWare Cloud Director استفاده شده‌است. البته برای کاربردهای خاص و عملیات تخصصی دقیق، ممکن است که نیاز به تعامل مستقیم با API داشته باشیم. ولی برای خدمات فعلی سامانه، تمامی امکانات مورد نیاز داخل SDK موجود است.

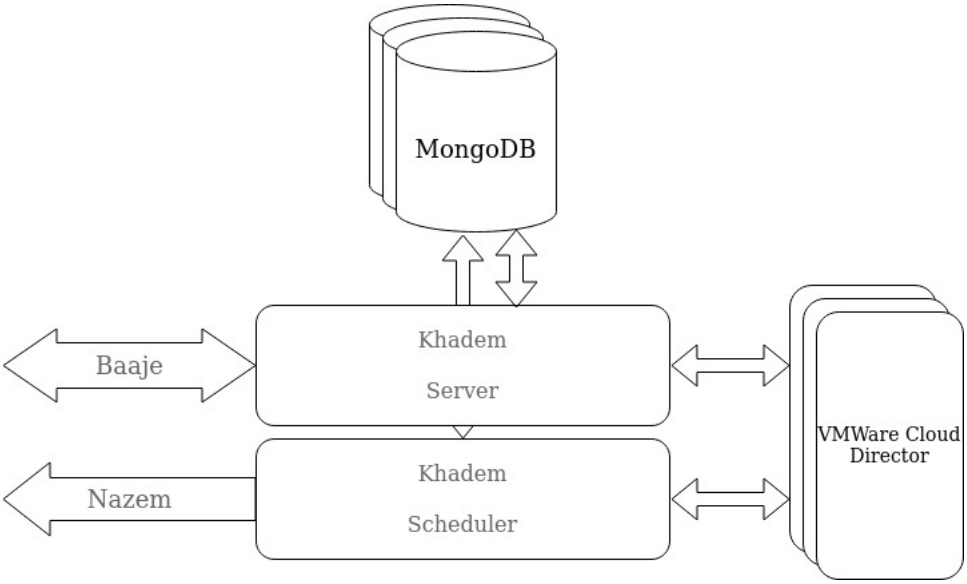
همچنین یک ثبت کننده وقایع داخل این سرویس تعبیه شده که تمامی درخواست‌های ارسالی به سرویس بالادستی را در یک پایگاه داده غیر رابطه‌ای ذخیره کند. این کار برای نظارت بیشتر و بازیابی از حادثه[۱۶] احتمالی بسیار مفید است.

برنامه **زمان‌بند** در پس‌زمینه به صورت مستقل اجرا شده و در فواصل زمانی مشخص وضعیت منابع تعریف شده داخل Cloud Director را رصد کرده و گزارش وضعیت را به سرویس ناظم جهت رسیدگی به امور مالی و نظارتی ارسال می‌کند. یک محدودیت مهم این سرویس، تک نسخه ای بودن آن در زمان اجرا است. برای قابلیت چند نسخه‌ای بودن این برنامه، بایستی یک مکانیزم و الگوریتم هماهنگی میان عناصر درحال اجرا پیاده‌سازی کنیم که منابع سیستم در هر لحظه تنها **توسط فقط یک نسخه** رصد و گزارش شود. پیاده‌سازی این موضوع با کمک امکانات کوبرنتیز امکان پذیر است ولی بخاطر پیچیدگی‌هایی که این پیاده‌سازی به همراه دارد در این پروژه از این‌کار صرف نظر شده‌است.

برنامه خدمت‌گزار در پس‌زمینه اجرا می‌شود و منتظر دستورات از سمت باجه می‌شود. پس از رسیدن دستورات، بسته به نوع دستور، توابع و پیاده‌سازی‌های موجود در بسته vcloud در برنامه اجرا می‌شود.

[15] Software Development Kit

[16] Incident Recovery



شکل ۳-۱۱: عملکرد سرویس خادم

با این کار، دستورات قابل فهم توسط cloud director ارسال شده و نتیجه به صورت برخط و یا غیر برخط به کاربر گزارش می‌شود.

۳-۳ دروازه‌ی ورود رابط

پیاده‌سازی دروازه‌ی ورود رابط در این سامانه با استفاده از ابزار قدرتمند و پایدار nginx صورت گرفته. nginx یک خدمت‌گزار وب نوشته شده با زبان C است که توانایی مدیریت تعداد بسیار زیادی از درخواست‌ها در لحظه را دارد. برای استفاده از این دروازه‌ی ورود باید قوانین مربوط به مسیریابی را در فرمت مخصوص بنویسیم. از آنجایی که مقیاس کردن این گزینه به صورت افقی سخت انجام می‌شود (باید قوانین مربوط به DNS و کاربران را تغییر دهیم.) بایستی منابع و ظرفیت‌های این برنامه را زیادتر از سایر قسمت‌های دیگر در نظر بگیریم.

۳-۴ نتیجه گیری

در این فصل به تشریح قسمت‌های مختلف برنامه و جزئیات پیاده‌سازی آن‌ها پرداخته شد. سامانه ما از چهار میکروسرویس که هریک وظیفه مخصوص به خود را دارد تشکیل شد. این میکروسرویس‌ها که در قالب بسته‌های داکر اجرا شدند با رابط‌های REST به یکدیگر متصل هستند. همه این میکروسرویس‌ها

در پشت یک دروازه‌ی ورود قرار گرفته و راه ارتباطی با شبکه خصوصی سامانه تنها این دروازه‌ی ورود است. اولین میکروسرویس درگیر با درخواست‌های کاربران، **قفل** هست که وظیفه احراز هویت تمامی درخواست‌های ورودی به دروازه‌ی ورود را دارد. میکروسرویس بعدی، **باجه** است که تمامی عملیات کاربران در آن تعریف شد. این میکروسرویس برای صحت سنجی درخواست‌های کاربران با میکروسرویس **ناظم** در ارتباط است و برای انجام این درخواست‌ها با میکروسرویس **خادم** ارتباط برقرار می‌کند.

فصل چهارم

بررسی و ارزیابی

در این فصل به ارزیابی‌ها و بررسی‌های انجام شده بر روی پروژه پرداخته خواهد شد.

۴-۱ تست‌های واحد

تست واحد[1] یکی از مهم‌ترین بخش‌های ارزیابی یک برنامه است. این مدل از ارزیابی به بررسی عملکرد یک واحد از برنامه می‌پردازد. یک واحد به معنی کوچک‌ترین بخش منزوی از یک برنامه است که به صورت مستقل عمل می‌کند و ورودی و خروجی مشخصی دارد. این واحد می‌تواند بخشی از یک تابع، کل تابع و یا مجموعه‌ای از توابع باشد. از مزایای تست واحد، می‌توان به صحت سنجی عملکرد واحدها نسبت به ورودی‌های مختلف اشاره کرد که کمک بسیار شایانی به اجراهای موفق می‌کند و ریسک عملکردهای دور از انتظار را بسیار کاهش می‌دهد.

در زبان برنامه‌نویسی Go، این ارزیابی‌ها در سطح بسته انجام می‌شوند. فایل‌های تست، با پسوند `_test.go` در نام فایل مشخص شده و توسط دستور go test اجرا می‌شود. در این پروژه، ما ارزیابی‌ها را برای واحدهای مختلفی از جمله توابع رسیدگی کننده به درخواست‌ها و یا مدل‌های تعامل با پایگاه داده انجام داده‌ایم. این ارزیابی‌ها با اجرا کردن توابع با انواع مختلف ورودی، خروجی تابع را با خروجی مورد انتظار مقایسه می‌کنند و در صورت وجود تفاوت ارزیابی را با وضعیت خطا تمام می‌کنند. با گسترده کردن ورودی‌های ارزیابی و تقلید[2] کردن نیازمندی‌ها، می‌توانیم از صحت عملکرد توابع در سناریوهای مختلف اطمینان حاصل کنیم. همچنین با استفاده از ابزارهای مختلفی که زبان Go برای این هدف در اختیار توسعه دهندگان قرار داده، همانند ابزار Coverage می‌توانیم درصد پوشش تست، یعنی تعداد خط‌های کد که این تست آن‌ها را مورد ارزیابی قرار داده پیدا کنیم و مطمین شویم که کل واحد مورد ارزیابی قرار گرفته است.

۴-۲ تست‌های ادغام

تست ادغام[3] به معنی تست کلی سامانه و ارزیابی عملکرد چندین بخش از سامانه در کنار یکدیگر است. این تست می‌تواند در داخل خود کد و همانند تست‌های واحد در قالب فایل‌های `_test.gi` باشد. و هم می‌تواند توسط ابزارهای خارجی و با نسخه درحال اجرای برنامه اجرا شود. یک مثال از این مدل ارزیابی،

[1] Unit Test
[2] Mock
[3] Integration Test

اجرای فرایند ثبت‌نام و ورود کاربر به صورت تعاملی[۴] با سامانه است. برای این‌کار، ابتدا با یک درخواست اقدام به ساخت حساب کاربری می‌کنیم. سپس با اطلاعات استفاده شده در مرحله قبل، اقدام به ورود و دریافت کلید ورود می‌کنیم. سپس با ارزیابی کلید ورود متوجه عملکرد این قسمت از سامانه می‌شویم.

با توجه به پیاده‌سازی رابط‌های ارتباطی وب در این پروژه با استاندارد Open API، تمامی مقصدهای تعریف شده. قابلیت اجرا با ورودی دلخواه توسط ابزارهای تست API نظیر Postman یا K6 را دارند.

۴-۳ تست بار

تست بار[۵] به دسته‌ای از تست‌ها گفته می‌شود که عملکرد سامانه و الگوی مصرف از منابع و میزان مصرف از منابع را در سناریوهای تحت فشار شدید ارزیابی می‌کنند. معمولا این تست‌ها عملکردی ساده از سامانه که می‌تواند یک تست ادغام باشد را به صورت موازی توسط چندین کاربر اجرا می‌کنند. نتایج این تست به توسعه دهندگان این امکان را می‌دهد که نسبت حجم درخواست به منابع مصرفی و سرعت پاسخگویی برنامه را متوجه شوند و در صورت نیاز سناریوهای مقیاس پذیری یا بهبودهای عملکردی را طراحی کنند.

این تست‌ها در این پروژه به دو صورت انجام شده، دسته اول از تست‌ها در قالب فایل‌های Benchmark اضافه شده. یکی از قابلیت‌های زبان برنامه‌نویسی Go، امکان تعریف Benchmark برای قسمت‌های مختلف برنامه است. این توابع همانند تست‌های واحد، می‌توانند واحد مشخصی از برنامه را با حجم متغیری از ورودی اجرا کنند و نتیجه را به تفکیک سرعت اجرا و منابع مصرفی برای کاربر نمایش دهند.

نوع دیگر انجام تست بار، توسط ابزارهای خارجی اجرا می‌شود و عملکرد کلی برنامه‌را در سناریوهای پیچیده‌تر ارزیابی می‌کند. یکی از ابزارهای مورد استفاده در این بخش، ابزار ۶ است. با نوشتن و طراحی سناریو و مشخص کردن نوع و حجم بار ورودی به سامانه، با اجرای این برنامه، می‌توانیم عملکرد سامانه را نسبت به حجم بار ورودی ارزیابی کنیم و در نهایت صحت عملکرد پاسخ‌ها را نیز دریافت کنیم.

۴-۴ محدودیت‌ها

با توجه به معماری استفاده‌شده در پیاده‌سازی این سامانه، تمامی خدمت‌گزارهای سامانه و پایگاه داده می‌توانند به صورت افقی مقیاس شوند تا پاسخگوی حجم‌های احتمالی زیاد باشند. با این وجود بخش‌هایی از سامانه محدودیت‌هایی در راستای حجم فشار کلی سامانه ایجاد می‌کنند.

[4] Interactive
[5] Load Test

اولین محدودیت سامانه، عدم امکان مقیاس کردن زمان‌بند سرویس خادم است. این زمان‌بند به صورت مستقیم با کاربران در تعامل نیست و در نتیجه حجم درخواست‌های ناگهانی بار اصلا تاثیرگذار نیست. اما بالا رفتن تعداد منابع ساخته‌شده، یا فشار زیاد شبکه، باعث اختلال در عملکرد و فشار بر این برنامه می‌شود. راه حل کمتر کردن این فشار، رزرو کردن منابع اختصاصی شبکه برای این برنامه در کنار زیاد کردن منابع محاسباتی پیش از اجرا است.

یکی دیگر از محدودیت‌های سامانه که مقیاس شدن برنامه در آن تاثیر کمی دارد، زیرساخت cloud director است. با توجه به ساختار تک نسخه‌ای بودن آن در برنامه، تمامی نسخه‌های سرویس خادم با یک نسخه‌از cloud director در ارتباط هستند. از آنجایی که عملیات تعریف شده در این برنامه، ذاتا زمان‌بر هستند و نیازمند اجرا به صورت آسنکرون هستند. اجرای حجم زیادی از درخواست‌ها و ارسال آن‌ها به سمت cloud director می‌تواند باعث بروز اختلال و اعمال فشار بر این سرویس شود. راه حل پیشنهادی جهت بهبود این محدودیت، تعریف سقف برای تعداد عملیات در حال اجرا در سمت خادم است. این‌کار می‌تواند با قرار دادن یک صف پیام[۶] بین سرویس باجه و خادم انجام شود.

[6]Message Queue

۴-۵ خطایابی

عملکرد مورد انتظار رابط‌های برنامه‌نویسی برنامه در فایل‌های Open API Specification نوشته‌شده، می‌توان این فایل‌ها را با استفاده از ابزار Swagger بررسی کرد. در صورتی که سامانه نتیجه‌ای غیر از نتیجه انتظار برگرداند. خطایابی با استفاده از کد وضعیت برگردانده شده به همراه پیام[۷] پاسخ می‌تواند انجام شود. در صورت نیاز به اطلاعات بیشتر، برنامه‌ها خطاهای رخداده را داخل Std Error محیط اجرایی چاپ می‌کنند. در صورت اجرا در قالب بسته‌های داکر، چک کردن این خطاها می‌تواند با دستور docker logs انجام شود. با توجه به سطوح مختلف اطلاع رسانی خطا، تنظیمات چاپ رخدادها را می‌شود داخل فایل config تغییر داد. درجات مختلف چاپ جزئیات در لیست زیر نوشته شده. در این لیست اولین عضو به معنی کمترین جزئیات و آخرین عضو به معنی بیشترین جزئیات است.

۱. `panic`
۲. `fatal`
۳. `error`
۴. `warning`
۵. `info`
۶. `debug`

[7] Message

فصل پنجم

جمع‌بندی، نتیجه‌گیری و پیشنهادات برای کارهای آتی

۵-۱ جمع‌بندی و نتیجه‌گیری

در این پروژه، سعی کردیم که یک رابط تحت وب برای ارائه خدمات IaaS بر بستر VMWare Cloud Director ارائه دهیم. در ابتدا با تحقیق و پیدا کردن نیازمندی‌ها، اقدام به طراحی معماری کلی سامانه و شناسایی ابزارها و نیازمندی‌ها کردیم. سپس با تحقیق در مورد این نیازمندی‌ها و مزیت‌های هر مدل از پیاده‌سازی، به یک طراحی نهایی رسیدیم. برای طراحی این پیاده‌سازی از الگوهای رایج توسعه نرم‌افزار تحت وب استفاده کردیم و به یک سامانه با قابلیت‌های قبول رسیدیم که به واسطه ساختار قطعه‌ای، قابلیت توسعه و تکمیل کردن به راحتی‌را دارد. سپس با ارزیابی‌های چندجانبه از صحت و توان عملکردی نرم‌افزار، پیش‌بینی تقریبی از اجرای این نرم‌افزار در سناریوهای واقعی به دست آوردیم.

۵-۲ پیشنهادات برای کارهای آتی

این پروژه، قابلیت تکمیل و توسعه از ابعاد مختلفی را داراست. ساختار میکروسرویسی و وابستگی اندک سرویس‌ها به یکدیگر، امکان توسعه و تغییر در میکروسرویس‌ها را به صورت جداگانه می‌دهد. در ادامه برخی پیشنهادات برای تکمیل و توسعه را به صورت جامع توضیح می‌دهیم.

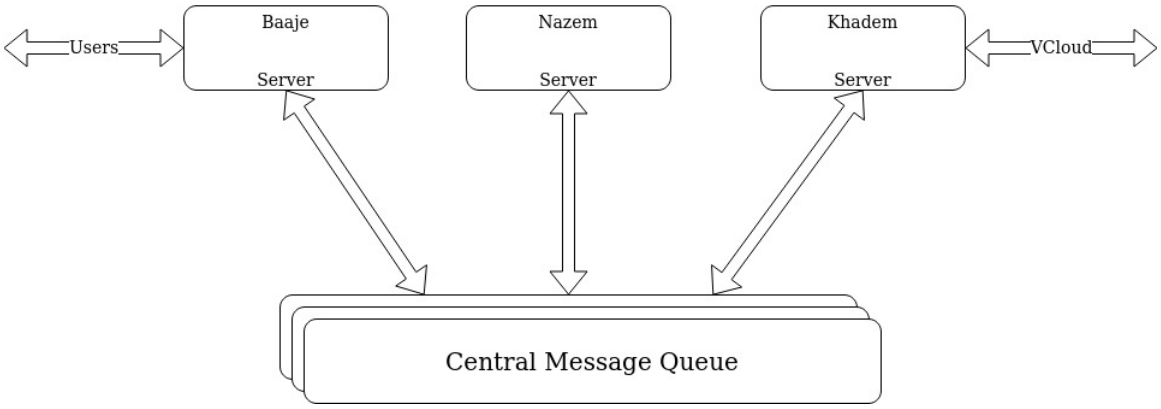
۵-۲-۱ تجزیه میکروسرویس ناظم

عملکرد سرویس ناظم در حال حاضر، دربرگیرنده تنظیمات و تصمیمات کسب‌وکار از جمله مصرف کاربران و امور مالی است. در صورت نیاز و ظهور نیازمندی‌های کسب‌وکاری بیشتر، می‌توان قسمت ثبت مصرف و تصمیمات مالی را جداکرد.

با این‌کار می‌توان مدل‌های مختلف پرداختی را پشتیبانی کرد. این جداسازی مزیت جداشدن منطق و محدودتر شدن فعالیت‌های میکروسرویس را به دنبال دارد. منتها سربار منابع مصرفی اضافی و افزایش تعداد درخواست‌ها در شبکه را نیز اضافه می‌کند.

۵-۲-۲ ارتباطات رخداد پایه

در حال حاضر، تمام ارتباطات سامانه به صورت سنکرون و توسط درخواست‌های HTTP انجام می‌شود. مزیت این روش سادگی پیاده‌سازی و دنبال کردن روند اجرای دستورات است. منتها با افزایش تعداد درخواست‌ها و محدودیت‌های شبکه ، این مشکل پیش می‌آید که بازیابی وضعیت سیستم از خطا دشوارتر



شکل ۵-۱: معماری سامانه در حالت آسنکرون

می‌شود. برای افزایش آستانه تحمل خطا در سرویس‌ها و امکان اضافه کردن تلاش مجدد[۱]، می‌توان ارتباط سرویس‌ها را به صورت آسنکرون و به صورت رخداد پایه انجام داد. مزیت این‌کار بالابردن در دسترس بودن سیستم[۲] می‌شود. برای پیاده‌سازی این مورد می‌توان از ابزارهای انتقال پیام نظیر Apache Kafka یا راه‌حل‌های سبک‌تر نظیر NATS استفاده کرد.

نحوه پیاده‌سازی API در میکروسرویس‌ها، امکان تغییر مکانیزم تعامل با بیرون را بسیار ساده کرده است. فقط کافی‌است که در پکیج‌های مربوطه، توابع مربوط به دریافت و ارسال اطلاعات را فارغ از نحوه پیاده‌سازی معرفی کنید. لایه‌های زیرین سرویس فقط نیازمند توابعی برای دریافت و ارسال اطلاعات هستند و وابستگی‌ای به نحوه دریافت و ارسال این اطلاعات ندارند.

معماری جدید سامانه در این حالت در شکل۵-۱ مشخص شده است.

۵-۲-۳ نظارت و گزارش رخدادها

جهت بهبود وضعیت نظارت بر سامانه، می‌توان از داشبوردهای گرافیکی ابزار گرافانا[۳] استفاده کرد. فقط کافیست که از اطلاعات متریک‌های سرور برای نمایش اطلاعات در قالب این داشبوردها استفاده‌کرد.

مزیت دیگر استفاده از این ابزار، امکان تعریف هشدار[۴] برای وضعیت‌های خاص و ناخواسته است. استفاده از این الگو، از روندهای رایج در فرهنگ SRE[۵] است. پیروی از این فرهنگ، بازیابی از فاجعه‌ها و ایرادات را بسیار سریع‌تر و آسان‌تر می‌کند.

[1] Retry
[2] Availability
[3] Grafana
[4] Alert
[5] Software Reliability Engineer

اضافه کردن Telemetry بسیار به درک و نظارت عملکرد میکروسرویس‌ها کمک می‌کند. ابزار پیشنهادی برای پیاده‌سازی این قابلیت، Jaeger و کتاب‌خانه‌های OpenTelemetry است.

۵-۲-۴ پشتیبانی از افزونه‌های VMWare Cloud Director

برخی از قابلیت‌های ابزار Cloud Director در قالب افزونه‌ها ارائه شده که به صورت پیش‌فرض فعال نیستند و نیازمند تغییراتی در تنظیمات هستند. با فعال سازی این افزونه‌ها می‌توان از امکانات بیشتری از این سرویس استفاده کرد. یکی از مثال‌های این افزونه‌ها، افزونه نظارت (Monitoring) است که نیازمند راه‌اندازی یک پایگاه Cassandra است. با تنظیم کردن و اضافه کردن این افزونه، امکان دریافت جزئیات مصرف منابع و گزارش گیری از آنها را مستقیما از سمت Cloud Director پیدا می‌کنیم.

البته پیاده‌سازی این این امکانات در سمت سرویس خادم با چالش‌هایی همراه است. چرا که این جزئیات در بسته توسعه نرم‌افزار پیش‌فرض تعبیه نشده و نیازمند تعامل مستقیم و خام برنامه با رابط برنامه‌نویسی vCloud است. البته این امر می‌تواند به سادگی و با تبدیل دستورات برنامه رابط کاربری تحت وب به دستورات cURL و سپس تبدیل دستورات cURL به درخواست‌های HTTP در زبان Go توسط برنامه Postman انجام شود. توسعه‌دهنده کافی است درخواست‌های ساخته‌شده را به قالب توابع موجود `VCloudManager` در بسته `vcloud` در برنامه تبدیل کند.

پیوست

مستندات ارائه دهندگان خدمات ابری خارجی

- Google: https://cloud.google.com/docs
- Microsoft: https://learn.microsoft.com/en-us/azure/virtual-machines
- DigitalOcean: https://docs.digitalocean.com/glossary/iaas
- Oracle: https://docs.oracle.com/en-us/iaas/Content/home.htm
- Amazon: https://docs.aws.amazon.com/ec2
- IBM: https://cloud.ibm.com/docs

۱- کدها و پیوست‌ها

تمامی کدها و پیوست‌های پروژه نظیر ساختار پایگاه‌داده، نیازمندی‌های اجرا و فایل‌های قابل اجرا در مخزن زیر در دسترس است.

`https://github.com/smf8/30bird`

Amirkabir University of Technology
(Tehran Polytechnic)

Department of Computer Engineering

Bachelor Thesis

VMWare VCloud IaaS REST Interface

By

Seyed Mohammad Fatemi

Supervisor

Dr. Mahmoud Momtazpour

March 2023