به نام خدا

درس تحلیل و طراحی سیستم‌ها

پروژه

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم سال اول ۰۲ـ۰۳

استاد:

دکتر علیرضا آقامحمدی

طراحان پروژه:

عرشیا اخوان، بهار اسدی، محمدطه جهانی‌نژاد، پارمیدا جوادیان، محمدعلی حسین‌نژاد، مهدی داوودزاده، آرمین دلگسار ماهر، رستا روغنی، حسین سمیعی، مهدی عباس‌تبار، امیرصدرا عبدالهی، سپهر کیانیان، محمدصادق مجیدی یزدی، امیرمهدی نامجو و کمیل یحیی‌زاده

فهرست

نکات قابل توجه

پروژه در دو فاز اصلی انجام می‌شود:

۱. فاز اول مربوط به طراحی سیستم است و در این مرحله نیازی به نوشتن کد و پیاده‌سازی سیستم نیست. در این فاز باید نموداری کلی از ساختار کلی سیستمی که قصد پیاده‌سازی آن را دارید را طراحی کرده و در یک مستند، دلایل خود برای طراحی این سیستم را توضیح دهید. در مواردی که چندین انتخاب برای یک مولفه وجود داشته باشد، باید توضیح بدهید که به چه دلیل از موردی که انتخاب کرده‌اید استفاده کرده و سایر گزینه‌ها را رد کرده‌اید.

نیازی به این که مستندی طولانی تهیه کنید وجود ندارد و یک مستند ۴ یا ۵ صفحه‌ای که به طور جامع و مانع طراحی شما را توضیح داده باشد، برای این منظور کافیست.

در فاز طراحی، فرض کنید که محدودیتی به لحاظ سخت‌افزاری وجود ندارد و ساختار بهینه‌ای که به نظرتان مناسب این پروژه است را طراحی کنید. در فاز دوم متناسب با محدودیت‌های عملیاتی سیستم خودتان، پیاده‌سازی را به تناسب در صورت لزوم تغییر خواهید داد.

به عنوان فعالیت امتیازی، می‌توانید امکان‌سنجی مالی انجام پروژه در مقیاس بزرگ را هم انجام بدهید. برای این کار باید از تعرفه نرخ پایه خدمات فنی‌ـ‌تخصصی انفورماتیک سال ۱۴۰۲ استفاده کنید. مستند مربوط به آن را می‌توانید از این لینک دریافت کنید.

۲. در فاز دوم، باید در طی چهار اسپرینت دو هفته‌ای، به پیاده‌سازی پروژه بپردازید. با توجه به استفاده از متدولوژی اسکرام، در این مراحل باید از طریق یک سیستم مدیریت وظیفه نظیر Jira، اقدام به ایجاد یک پروژه کرده و وظایف مختلف برای هر یک از اعضای تیم در هر اسپرینت تعریف بکنید. برای این وظایف باید سطوح اولویت تعریف شده و در صورت لزوم وابستگی آن‌ها به یکدیگر مشخص بشود و وظایف به طور مشخص به اعضای مختلف تیم داده بشود. به علاوه باید دسته‌بندی‌هایی برای وظایف ایجاد بشود و هر وظیفه در دسته‌بندی(های) مناسب خود قرار بگیرد.

نحوه مشخص کردن اولویت و نمایش آن در سامانه و موارد مشابه به صلاحدید تیم شما خواهد بود اما این موضوع باید انجام بشود و در مورد روش خود دلیل قانع‌کننده داشته باشید.

برای رسیدگی به کارهای هر گروه یک دستیار آموزشی مشخص شده است. در مواردی که در مورد پروژه ابهام دارید، می‌توانید ابهام را به کمک این دستیار آموزشی برطرف کنید و متناسب با آن پروژه را پیش ببرید.

با توجه به استفاده از متدولوژی اسکرام، انتظار می‌رود که جلسات Planning و Retrospective در ابتدا و انتهای هر اسپرینت برگزار شده و گزارش آن جلسات به دستیار آموزشی ارائه بشود. به علاوه جلسات Daily بین اعضای تیم باید برگزار بشود و گزارش آن به دستیار آموزشی ارائه بشود. در مورد جلسات Daily نیازی به این که حتما به صورت روزانه جلسات برگزار بشوند وجود ندارد، ولی باید بازه‌های زمانی مشخصی برای این موضوع در هر اسپرینت مشخص بشود و متناسب با آن، جلسات برگزار شده و گزارش‌ها ارائه بشود.

در انتهای هر اسپرینت، در یک جلسه با دستیار آموزشی، باید وضعیت پیشرفت پروژه در آن اسپرینت را ارائه داده و وضعیت وظایف انجام شده در آن اسپرینت و وظایفی که تکمیل نشده‌اند و باید به اسپرینت بعدی منتقل بشوند را به دستیار آموزشی ارائه کنید. زمان این جلسات با هماهنگی بین اعضای گروه و دستیار آموزشی مشخص می‌شود اما به طور کلی نباید بیش از دو روز از انتهای اسپرینت فاصله داشته باشد.

در نهایت توجه کنید که نحوه تنظیم زمان شروع اسپرینت‌ها بر عهده خود شماست و با توجه به دروس دیگر، نیازی نیست که کاملا پشت سر هم باشند، ولی باید آن‌ها را طوری تنظیم کنید که پایان آخرین اسپرینت حداکثر ۱۵ بهمن ماه بوده و بازه دو هفته‌ای هر اسپرینت به صورت پیوسته باشد.

مهلت تحویل فاز اول پروژه **۱۴ آذرماه** بوده و پس از آن اسپرینت‌های دو هفته‌ای آغاز می‌شوند.

مقدمه

پروژه مهم‌ترین بخش درس تحلیل و طراحی سیستم‌ها است. هدف از پروژه یادگیری و تثبیت مفاهیم مختلفی است که در کلاس درس آن‌ها را فراگرفته یا خواهید گرفت. از جمله این موارد می‌توان به متدولوژی اسکرام، اصول تحلیل و طراحی سیستم‌های نرم‌افزاری، خط‌لوله CI/CD و مهارت‌های کار تیمی اشاره کرد.

پروژه طراحی شده در این ترم، پیاده‌سازی یک سیستم صف پیام‌رسان مشابه Kafka یا RabbitMQ است که در ادامه به تفضیل در مورد آن توضیح داده شده است. نکته حائز اهمیت این است که هدف پروژه ارزیابی مهارت‌های برنامه‌نویسی شما نیست، بلکه هدف اصلی ارزیابی توانایی‌های تحلیل و طراحی سیستم‌ها و همچنین کار تیمی و پیاده‌سازی آموخته‌های خود در فضایی شبیه یک کار صنعتی واقعی است. با توجه به مدت زمان درس، عملا محصولی که در انتها تولید می‌شود MVP کلی یک سیستم صف پیام‌رسان است و در نتیجه انتظار پیاده‌سازی ویژگی‌ها و قابلیت‌های غیرضروری وجود ندارد، بلکه خواسته اصلی پیاده‌سازی مناسب هسته اصلی این سیستم متناسب با نکات گفته شده در این مستند و کلاس درس و رعایت اصول کار تیمی است. در ادامه ابتدا به معرفی سیستم‌های صف پرداخته‌ایم و سپس خواسته‌های اصلی پروژه آورده شده است.

صف چیست

صف‌های پیام‌رسانی از ابزارهای کلیدی در معماری سیستم‌های توزیع‌شده به شمار می‌روند و نقش مهمی را در برقراری ارتباط مطمئن و کارآمد بین مولفه‌های مختلف یک سیستم ایفا می‌کنند. این صف‌ها به تولیدکننده‌های پیام اجازه می‌دهند که پیام‌های خود را فارغ از این‌که مصرف‌کننده‌ها در آن لحظه آماده‌ی پردازش هستند یا نه، بفرستند. (این قابلیت در اصطلاح پردازش آفلاین [1] نامیده می‌شود به این معنی که سیستم اصلی که با داده کار می‌کند لزوما در همان لحظه داده‌ها را پردازش نمی‌کند. این اصطلاح در مقابل پردازش آنلاین [2] قرار دارد که در آن سیستم در لحظه، داده ورودی را مصرف و پردازش می‌کند.) پیام‌ها تا زمانی که مصرف‌کننده قادر به دریافت و پردازش‌شان باشد، در صف نگهداری می‌شوند.

در ادامه نمونه‌هایی از کاربرد صف در سیستم‌ها آورده‌شده‌اند.

۱. تعادل بار [3]: در سیستم‌هایی با ترافیک بالا، صف‌ها می‌توانند به توزیع یکنواخت درخواست‌ها بین سرورهای مختلف کمک کنند تا از لود اضافه روی یک سرور خاص جلوگیری شود.

۲. تضمین تحویل پیام: در شرایطی که ارتباط شبکه ناپایدار باشد یا مصرف‌کننده با مشکل

[1] Offline Processing
[2] Online Processing
[3] Load Balancing

مواجه باشد، صف‌ها اطمینان حاصل می‌کنند که پیام‌ها از دست نروند و در نهایت به مقصد برسند.

۳. جداسازی مسئولیت‌ها [۴]: صف‌ها به جداسازی مولفه‌های تولیدکننده و مصرف‌کننده کمک می‌کنند، به این معنی که تغییرات در یک بخش از سیستم بر بخش‌های دیگر تأثیر نمی‌گذارد.

۴. پردازش غیرهمزمان [۵]: کارهای سنگین و زمان‌بر می‌توانند به صف افزوده شوند تا در پس‌زمینه و بدون تأثیر بر عملکرد کلی سیستم اجرا شوند.

۵. مقاومت در برابر خطا [۶]: با استفاده از صف‌ها، حتی در صورت خرابی یک مولفه، پیام‌ها از دست نمی‌روند و می‌توان پس از رفع مشکل مجدداً آن‌ها را پردازش کرد.

نمونه سیستم‌های صف

سیستم‌های صف متعددی وجود دارند که هر کدام از آن‌ها نقاط قوت و ضعف خود را دارند. Kafka و RabbitMQ از جمله پرکاربردترین این سیستم‌ها هستند که در ادامه بعضی از ویژگی‌های آن‌ها آورده‌شده‌اند.

Apache Kafka

۱. پردازش پیام‌ها با حجم و سرعت بالا: کافکا برای مدیریت داده‌های حجیم و نیز پیام‌های با سرعت بالا طراحی شده است.

۲. دوام و تحمل خطا: کافکا با استفاده از مکانیزم Replication اطمینان حاصل می‌کند که داده‌ها حتی در صورت خرابی سرورها از دست نخواهند رفت.

۳. معماری انتشار_اشتراک [۷]: کافکا از هر دو مدل پخش و صف پشتیبانی می‌کند. این امر به تولیدکنندگان و مصرف‌کنندگان این امکان را می‌دهد که به‌صورت مستقل عمل کنند.

ویژگی‌های مذکور باعث شده‌است کافکا برای پردازش بی‌درنگ [۸] و جمع‌آوری لاگ‌های سرویس‌ها و سرورها مناسب باشد.

RabbitMQ

۱. انعطاف‌پذیری: از الگوهای پیام‌رسانی مختلف پشتیبانی می‌کند.

[4] Decoupling
[5] Asynchronous Processing
[6] Fault-Tolerance
[7] Publish-Subscribe
[8] Real-time

۲. قابلیت اطمینان: از مکانیزم‌های تایید پیام و تراکنش‌ها استفاده می‌کند تا از تحویل پیام اطمینان یابد.

۳. مقیاس‌پذیری: با افزودن گره‌های [۹] بیشتر به خوشه‌ها [۱۰]، امکان افزایش منابع و کارایی را فراهم می‌کند.

ویژگی‌های بالا باعث شده‌است RabbitMQ برای سیستم‌های هشدار و نیز ارتباط بین سرویس‌های مختلف در یک سیستم توزیع‌شده گزینه مناسبی باشد.

[9] Node
[10] Cluster

پروژه

هدف اصلی این پروژه هدایت دانشجویان برای طراحی یک صف و راه‌اندازی کلاینت آن است. بنای این پروژه بر زیرساخت کانتینری [11] است که گسترش [12] یکپارچه و مقیاس‌پذیری [13] در محیط‌های مختلف را با استفاده از ابزارهایی مانند کوبرنیتز [14] یا داکر سوارم [15] یا داکر کامپوز [16] حاصل می‌کند.

کلاینت که یکی از بخش‌های اصلی پروژه است، باید سه عملکرد اساسی push، pull و subscribe را دربرگیرد که در ادامه این موارد دقیق‌تر توضیح داده می‌شوند.

در سمت سرور، تأکید بر ایجاد یک معماری مقیاس‌پذیر [17] و مقاوم در برابر خطا است. از دانشجویان انتظار می‌رود که مفاهیمی مانند پارتیشن‌بندی [18] و تکرار [19] را بررسی کنند تا از انعطاف‌پذیری [20] و مقیاس‌پذیری سیستم با تغییر لود اطمینان حاصل کنند. این اصول برای ساختن یک پایه قوی که بتواند در مقابل خرابی گره‌ها [21] مقاومت کند و در عین حال یکپارچگی [22] و در دسترس بودن [23] داده‌ها را تضمین کند، حیاتی هستند.

نظارت [24] یکی دیگر از جنبه‌های مهم این پروژه است. دانشجویان بهتر است یک پشته نظارتی [25] را که توسط Orchestrating System مدیریت می‌شود، پیاده‌سازی کنند. این کار دید کاملی حول عملکرد و سلامت سامانه به ما می‌دهد. ضروری است که تنظیمات نظارت [26] بتواند در مورد رویدادهای مهم مانند استفاده زیاد از دیسک هشدار [27] دهد.

دانشجویان با انجام این پروژه، می‌توانند از چارچوب‌های نظری به طراحی و تجزیه و تحلیل عملی در سیستم‌ها گذر کنند. آن‌ها در طراحی این سیستم مراحل مختلفی را، از تجزیه و تحلیل نیازمندی‌ها تا طراحی معماری، پیاده‌سازی و آزمایش یک سیستم مدیریت صف قابل‌اعتماد، مقیاس‌پذیر و انعطاف‌پذیر طی خواهند کرد.

توجه کنید که Performance سیستم در این پروژه برای ما مهم نیست.

[11] Containerized
[12] Deployment
[13] Scalability
[14] Kubernetes
[15] Docker Swarm
[16] Docker Compose
[17] Scalable
[18] Partitioning
[19] Replication
[20] Resilience
[21] Node Failure
[22] Integrity
[23] Availability
[24] Monitoring
[25] Monitoring Stack
[26] Monitoring Setup
[27] Alert

کلاینت

کلاینت باید به دو زبان مختلف پیاده‌سازی شده و یکی از این دو زبان باید پایتون باشد. به این معنی که اجرای اصلی منطق کلاینت باید در محیط پایتون قابل‌دسترسی و قابل‌استفاده باشد.

اگر می‌خواهید از زبان دیگری برای پیاده‌سازی کلاینت‌ها استفاده کنید، برای راحتی کار می‌توانید منطق کلاینت‌ها را با زبانی غیر از پایتون نوشته و سپس یکی از آن‌ها را با یک wrapper پایتون ارائه کنید.

در اینجا wrapper به معنی یک لایه کد است که امکان فراخوانی و استفاده از قابلیت‌های پیاده‌سازی‌شده در یک زبان را در زبانی دیگر (در اینجا پایتون) فراهم می‌کند.

حال سه قابلیت اصلی کلاینت را بررسی می‌کنیم:

```
Push: push(key: String, value: [Byte])
```

پارامترها:

یکی از پارامترهای این تابع key است که به‌صورت یک رشته است. این رشته به عنوان یک شناسه یکتا برای پیامی که به صف push می‌شود، عمل می‌کند. این رشته می‌تواند برای ارجاع، دسته‌بندی یا تقسیم‌بندی پیام‌ها در صف استفاده شود.

پارامتر دیگر این تابع value است که به‌صورت آرایه‌ای از بایت‌ها است که داده‌های پیامی را که به صف push می‌شوند، نشان می‌دهد. این فرمت داده دودویی امکان ارسال هر نوع داده‌ای را فراهم می‌کند.

عملکرد:

هنگامی که تابع push فراخوانی می‌شود، یک پیام جدید با key و value مشخص‌شده ایجاد و برای پردازش یا ذخیره‌شدن به سرور صف ارسال می‌شود.

همچنین لازم به ذکر است این تابع باید به شکل blocking پیاده‌سازی شود.

```
Pull: pull() -> (key: String, value: [Byte])
```

پارامترها:

این تابع پارامتری ندارد.

عملکرد:

هنگامی که این تابع فراخوانی می‌شود، یک جفت (key, value) که در سر صف قرار دارد، از صف خارج شده و تابع آن‌ها را باز می‌گرداند.

همچنین لازم به ذکر است این تابع باید به شکل blocking پیاده‌سازی شود، به این معنی که پس از فراخوانی، فراخواننده آنقدر صبر می‌کند تا این عملیات به اتمام برسد. برای

اطلاعات بیشتر از این رویه، می‌توانید به اینجا مراجعه کنید.

```
Subscribe: subscribe(f: func(key: String, value: [Byte]))
```

پارامترها:

تنها پارامتر این تابع خود یک تابع f است. تابع f یک key و یک value دریافت می‌کند و سپس پیام را پردازش می‌کند. لازم به ذکر است این تابع خروجی ندارد و یک تابع در سمت کلاینت است.

عملکرد:

پس از اتمام فراخوانی تابع subscribe، هر زمانی که یک جفت (key, value) به صف push شد، تابع f روی این جفت فراخوانی می‌شود.

روند اجرای تابع subscribe به‌صورت غیرمسدودکننده [۲۸] است. روند اجرای تابع f هم می‌تواند به صورت غیرمسدودکننده باشد.

نکته‌ی جالب توجه این است که فرآیند unsubscribe به طور ضمنی [۲۹] انجام می‌شود. درواقع هرموقع کلاینت سوکت را بست، توابعی که از سمت آن کلاینت subscribe شده‌بودند، باید unsubscribe شوند.

[28] Non Blocking
[29] Implicit

سرور

می‌توانید سمت سرور را با زبان دلخواه‌تان پیاده‌سازی کنید. در سمت سرور باید یک صف مادر قرار داده‌شود. عملکردهای زیر عملکردهای اصلی‌ای هستند که برای سرور در نظر گرفته شده‌اند. برای انجام نیازمندی‌های غیرعملکردی [30] ممکن است عملکردهای دیگری نیاز به پیاده‌سازی داشته‌باشند.

عملکرد: سرور باید توانایی پاسخ دادن به تمامی API Call هایی که از سمت کلاینت تعریف می‌شود را داشته باشد. در پیاده سازی سرور، طراحی یک API ساده ولی کارآمد بسیار مهم است. در طراحی API خود باید روشی برای نوشتن (Push) و خواندن (Pull) پیام از سرور فراهم کنید. لازم به ذکر است که کلاینت‌های شما صرفا از این APIها استفاده می‌کنند تا یک لایه انتزاع [31] برای استفاده‌ی راحت‌تر از سرور در کد را فراهم کنند. این به این معنی است که API سرور شما باید به تنهایی قابل استفاده باشد و نیازمند پیاده‌سازی کلاینت از سمت شما نباشد.

[30] Non-Functional
[31] abstraction

نیازمندی‌های غیرعملکردی

Scalability

در طراحی نرم‌افزار مقیاس‌پذیری مفهوم مهمی است که باید به آن پرداخته‌شود. به‌طور ساده، مقیاس‌پذیری به توانایی سیستم در مدیریت‌کردن افزایش لود کاری روی سرور گفته می‌شود. این افزایش می‌تواند نتیجه‌ی اضافه‌شدن کاربرهای جدید به سیستم و یا افزایش تعداد تقاضاها در بازه زمانی خاصی باشد. در واقع یک سیستم مقیاس‌پذیر می‌تواند در شرایط خاص به راحتی منابع خود را گسترش دهد، با نوسان‌های لود سازگار شود و به کار خود با همان کیفیت قبل ادامه‌دهد.

به‌طور خاص برای یک صف ویژگی مقیاس‌پذیری بسیار مهم است زیرا یک صف اغلب در هسته‌ی اپلیکیشن‌های مختلف قرار دارد و مسئول مدیریت پیام‌ها، تسک‌ها و یا ایونت‌ها است. پس همانطور که برنامه‌ها رشد می‌کنند، سیستم صف باید به‌خوبی به درخواست‌های فزاینده برای پردازش و تحویل این پیام‌ها رسیدگی کند.

دو شکل اصلی مقیاس‌پذیری وجود دارد:

۱. مقیاس‌پذیری عمودی [۳۲]: این روش شامل افزایش ظرفیت یک ماشین یا گره از طریق افزایش منابعی مانند حافظه و پردازنده است. درحالی که این رویکرد می‌تواند عملکرد را تا حدی بهبود بخشد، محدودیت‌های عملی دارد و ممکن است درنهایت منجر به کاهش بازدهی شود.

۲. مقیاس‌پذیری افقی [۳۳]: این رویکرد شامل اضافه‌کردن گره‌ها و یا ماشین‌های بیشتر به سیستم است تا به آن اجازه دهد لود کاری را در چند گره توزیع کند. هنگامی که لود روی سیستم افزایش می‌یابد، گره‌های جدید را می‌توان با الگوریتمی اضافه کرد تا در نتیجه عملکرد و ظرفیت بهبود یابد. این رویکرد اغلب با مقیاس‌پذیری خطی همراه است، بدین معنی که با اضافه‌کردن گره‌های بیشتر می‌توان انتظار افزایش متناسب ظرفیت سیستم را داشت.

در این پروژه تاکید روی دستیابی به مقیاس‌پذیری افقی است. برای این هدف باید به چند جنبه‌ی فنی آن همانند توزیع بار [۳۴]، مقیاس‌پذیری پویا [۳۵] و تکثیر داده‌ها [۳۶] توجه کنیم.

یکی از اهداف اصلی پیاده‌سازی ویژگی مقیاس‌پذیری در پروژه، مقیاس‌پذیری خطی در نهایت [۳۷] است. این هدف بدین منظور است که اطمینان حاصل‌شود که با اضافه‌کردن گره‌های بیشتر به سیستم، ظرفیت آن به‌طور خطی و یا نزدیک‌به‌خطی افزایش یابد. دستیابی

[32] Vertical Scaling
[33] Horizontal Scaling
[34] Load Balancing
[35] Dynamic Scaling
[36] Data Replication
[37] Eventually Linearly Scalable

به این هدف به‌دلیل عواملی مانند انسجام داده [۳۸] و توزیع پیام [۳۹] و هماهنگی بین گره‌ها می‌تواند چالش‌های فنی داشته باشد. برای حل این چالش‌ها راه‌حل‌هایی همانند Efficient Message Routing، Effective Load Balancing و Stateless Design وجود دارند که با استفاده از آن‌ها می‌توان مقیاس‌پذیری سیستم را در نهایت خطی و یا تقریبا_خطی کرد.

Fault Tolerance

یکی دیگر از مفاهیمی که باید به آن توجه‌کنید مقاوم بودن سیستم در برابر خطا است که در ادامه به آن پرداخته‌شده‌است.

تعریف: به‌طور کلی یک سیستم مقاوم در برابر خطا باید بتواند در صورت بروز مشکل برای یک یا چند مولفه‌ی خوشه، به کار خود ادامه‌دهد و کاربر تا حد امکان متوجه ضعف ایجاد شده در سیستم نشود.

تعمیم به پروژه: در این پروژه از شما می‌خواهیم که خوشه صف خود را تا حدی مقاوم در برابر خطا پیاده‌سازی‌کنید. به این‌صورت که اگر یکی از گره‌ها به مشکل خورد، نباید دسترسی‌پذیری سیستم از بین برود و نباید داده‌ها را از دست‌بدهیم. نکاتی که می‌توانید در نظر بگیرید: در هنگام بروز مشکل برای یک مولفه می‌توانید فرض‌کنید که درخواستی در لحظه به خوشه نمی‌آید و نگران درخواست‌هایی که در بازه زمانی fail-over می‌آیند، نباشید. در واقع ما در این فرآیند تنها می‌خواهیم که داده‌ای از دست‌ندهیم و بعد از مدتی بتوانیم به خوشه درخواست pull یا push بفرستیم. روش‌های مختلفی برای پیاده‌سازی این ویژگی وجود دارد که می‌توان به تکثیر [۴۰] و Erasure Coding اشاره‌کرد.

Consensus

یکی دیگر از ویژگی‌هایی که باید در خوشه وجود داشته باشد، مفهوم Consensus است. در واقع گره‌های هر خوشه باید از وضعیت خوشه باخبر باشند و با یکدیگر اتفاق‌نظر داشته‌باشند. به بیان دیگر، باید با یکدیگر هماهنگ [۴۱] باشند. هماهنگ شدن زمانی خیلی مهم می‌شود که یک گره بخواهد از خوشه به هر دلیلی خارج شود و یا به خوشه اضافه شود. به‌طور کلی دو روش برای هماهنگ نگه‌داشتن خوشه وجود دارد:

۱. Leader-less

۲. Leader-full

[38] Data Consistency
[39] Message Distribution
[40] Replication
[41] Synchronize

در روش leader-full یک گره به عنوان master انتخاب می‌شود و کارهای هماهنگ شدن گره‌ها و اطلاع‌رسانی به سایر گره‌ها را بر عهده می‌گیرد. مکانیزم‌های مختلفی برای انتخاب master و طریق عملکرد آن داریم. raft و kraft از الگوریتم‌های معروف این روش هستند که در آن‌ها یک گره به عنوان رهبر انتخاب می‌شود و وظیفه هماهنگ نگه داشتن باقی گره‌ها را دارد.

در روش leader-less گره‌ای به عنوان master نداریم و در واقع همه گره‌ها به نحوی در اشتراک اطلاعات در رابطه با وضعیت خوشه دخیل هستند که در این زمینه می‌توان به الگوریتم‌های hash-ring و gossiping اشاره کرد.

Monitoring

مورد مهمی که در زیرساخت یک پروژه اهمیت دارد، قابلیت نظارت کردن آن توسط افراد است. یعنی باید بتوان به‌صورت مداوم در هر لحظه دلخواه جزییاتی از سیستم را بررسی کرد و در صورت نیاز کارهایی برای بهبود عملکرد سیستم انجام داد.

برای مثال در ابزاری مثل RabbitMQ می‌توان از طریق پنل موجود در ابزار، تعداد صف‌ها و تعداد پیام‌های هرکدام را در لحظه دید و پردازش موجود در صف‌ها را تحلیل کرد. همچنین Prometheus ابزاری است که می‌توانید به کمک آن به‌صورت پیوسته از زیرساخت مورد استفاده در سیستم خود یا سرورتان اطلاعات جمع‌آوری کرده و به کمک Grafana داده‌ها را به‌صورت بصری تحلیل و بررسی کنید. این ابزار نمودارهای کاربردی متعددی دارد که با تحلیل آن‌ها می‌توانید لود موجود در سیستم و downtime ها روی هرکدام از اجزای زیرساخت را ببینید.

در این پروژه شما باید از ابزار Prometheus استفاده کنید و زیرساخت پروژه‌تان را به آن متصل کنید. با این ابزار شما می‌توانید اجزای سیستم خود را بررسی و تحلیل کنید، کوئری [۴۲] بزنید و با توجه به نیازتان هشدار [۴۳] بسازید تا در شرایط خاص ابزار به شما وجود مورد حادی را اطلاع دهد.

برای این بخش باید تعدادی متریک را پیاده‌سازی کنید و در هر کدام از متریک‌ها موارد گفته‌شده را اضافه کنید:

- آمار کلی خوشه‌ها (total-cluster-stats)
  - تعداد گره‌ها
  - سایز کلی دیسک
  - مقدار دیسک مصرف‌شده

[42] Query
[43] Alert

- تعداد کل پیام‌ها

- آمار عملکرد (performance-stats)
  - تعداد push در ثانیه سیستم
  - تعداد pull در ثانیه سیستم
  - میزان throughput کل سیستم
  - میزان تاخیر پردازش سیستم [۴۴]

- جزییات هر گره (per-instance-metrics)
  - سایز کلی دیسک
  - مقدار دیسک مصرف‌شده
  - تعداد پیام‌های ذخیره‌شده
  - تعداد push در ثانیه
  - تعداد pull در ثانیه
  - میزان throughput
  - میزان تاخیر پردازش

همچنین باید یک هشدار بسازید که وقتی در متریک total-cluster-stats میزان دیسک مصرف‌شده از درصد خاصی (x%) بالاتر رفت، فعال شود.

[44] Consume Lag

زیرساخت

Portability

کل سیستم پیاده‌سازی شده توسط شما باید قابل‌حمل [۴۵] باشد. به‌طور خلاصه قابل‌حمل بودن [۴۶] به این معنی است که برنامه شما باید قابلیت اجرا در سیستم‌ها و محیط‌های مختلف نرم‌افزاری را داشته باشد. برای این موضوع می‌توانید از ابزارهای containerization مانند Docker استفاده کنید.

CI/CD

در پیاده‌سازی این بخش به این مورد توجه کنید که فرآیند ایجاد کانتینرها باید به عنوان بخشی از خطلوله‌ی CI پیاده‌سازی شود. در این پروژه پیاده‌سازی یک خط لوله CI/CD لازم است تا فرآیندهای ساخت و راه‌اندازی به طور خودکار انجام شوند. توصیه می‌شود برای این کار از ابزارهایی مانند GitLab CI/CD یا Jenkins یا Github Actions استفاده کنید.

Orchestrator

برای خودکارسازی در راه‌اندازی و مدیریت کانتینرها و ارتباط بین آن‌ها لازم است از ابزارهای Orchestrator در پروژه خود استفاده کنید. برخی از متداول‌ترین این ابزارها عبارتند از Docker Swarm، Kubernetes ، Docker Compose و Nomad

Simplicity

همانطور که پیش‌تر اشاره‌شد، سیستم پیاده‌سازی‌شده باید مقایس‌پذیر باشد. نکته دیگری که لازم است به آن توجه کنید این است که فرآیند متوازن کردن پروژه باید تا حد امکان ساده و efficient باشد. به این معنی که انجام کاری مانند اضافه کردن گره‌ها باید به سادگی انجام گیرد. برای مثال اگر از kubernetes در پروژه استفاده می‌کنید، این کار باید به سادگی تغییر عدد Replica باشد. دیگر Orchestrator ها هم امکانات مشابهی در اختیار شما قرار می‌دهند. Ansible playbookنیز ابزارهای ساده‌ای برای Configuration Managementارائه می‌دهد که با استفاده از آن‌ها می‌توانید فرآیند scaling را به سادگی انجام دهید.

Monitroing Stack

پیش‌تر به معیارهایی که باید در سامانه نظارتی قابل‌مشاهده باشند اشاره شد. لازم است به این نکته توجه کنید که بالا آوردن پشته نظارتی نیز باید به عنوان بخشی از سیستم اصلی باشد و توسط orchestrating system مدیریت شود.

در مورد پیاده‌سازی سیستم نظارتی به نکات زیر توجه کنید:

- سیستم نظارتی شما باید قابلیت هشدار هنگام افزایش بیش از اندازه مصرف دیسک را

[45] portable
[46] Portability

داشته‌باشد. به این معنی که اگر disk usage سیستم از درصدی بالاتر برود، سیستم به کاربر هشدار دهد. این هشدار می‌تواند به صورت ارسال یک ایمیل، اجرای یک هشدار در محیط رابط کاربری سیستم و یا هر روش دیگری پیاده‌سازی شود.

- در مورد معیارهای بخش total-cluster-stacks، نمایش داده‌ها به‌صورت صرفاً عددی کافی نیست و نیاز است این متریک‌ها با نمودارهای مناسب بصری شوند.