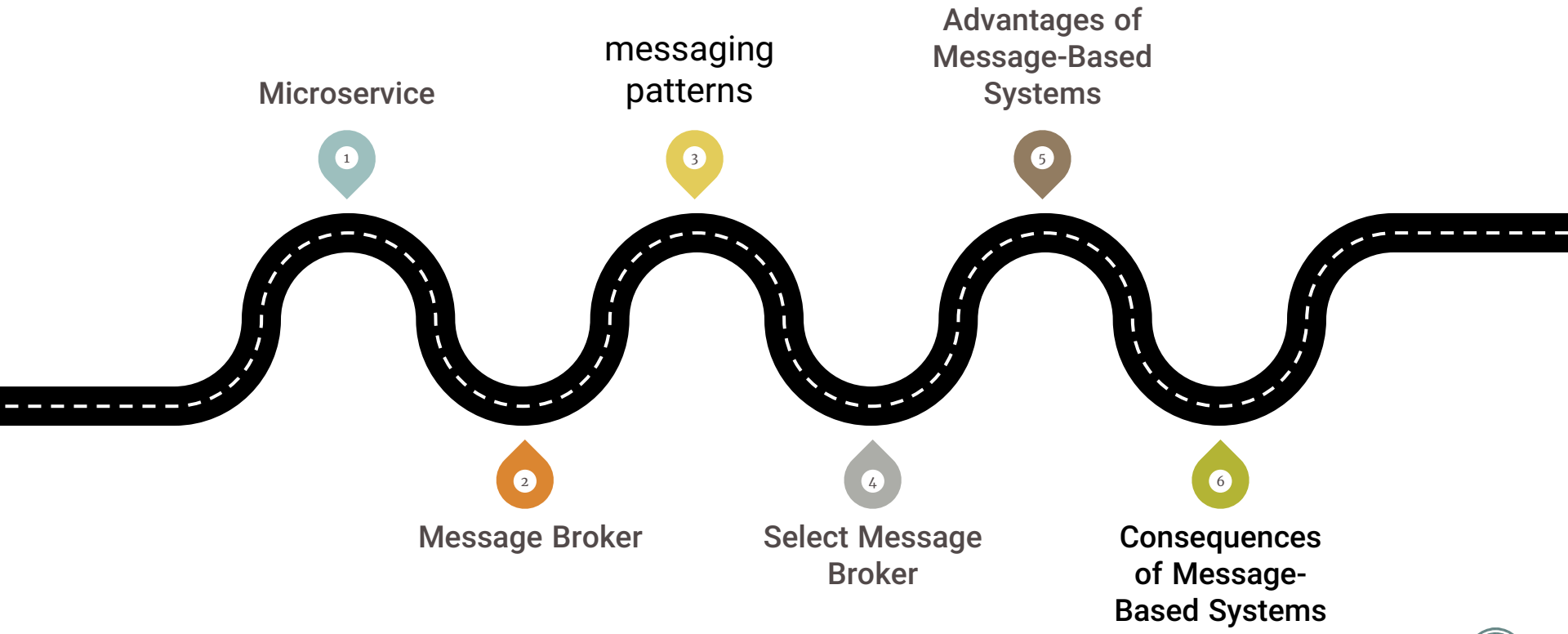
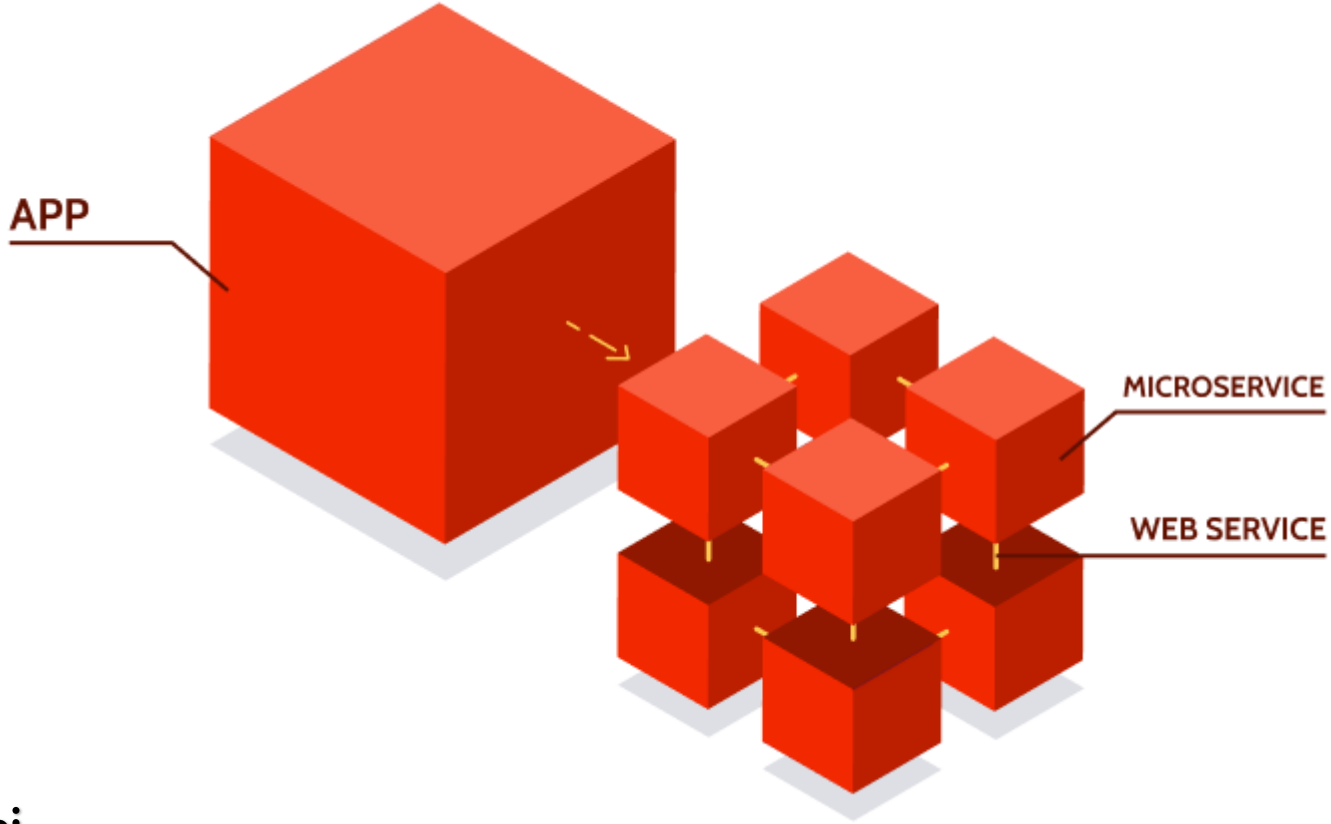
Roadmap

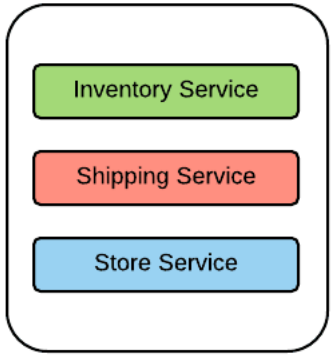

Microservice
1
messaging patterns
3
Advantages of Message-Based Systems
5
2
Message Broker
4
Select Message Broker
6
Consequences of Message-Based Systems

Micorservice

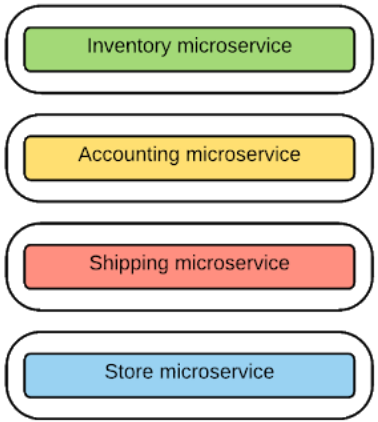


معماری Microservice

پایه و اساس معماری سرویس‌های **Microservice** ، توسعه یک نرم‌افزار واحد از مجموعه‌ای از سرویس‌های کوچک و مستقل است که هر کدام از این سرویس‌ها دارای پردازش‌های خاص خود باشند و به صورت مستقل توسعه و پیاده‌سازی شده باشند.
به طور کلی در این معماری، تمامی سرویس‌های موجود در یک نرم‌افزار یکپارچه و بزرگ به مجموعه‌ای از سرویس‌های مستقل و کوچک‌تر تقسیم می‌شوند. به عنوان مثال شکل‌های زیر را در نظر بگیرید که یک نرم‌افزار یکپارچه دارای سه قسمت اموال، حمل‌ونقل و فروشگاه (شکل سمت راست) به چهار سرویس مستقل تقسیم شده است (شکل سمت چپ). نکته جالب در این تغییر معماری می‌توان به اضافه شدن سرویس Accounting اشاره کرد.
سرویس Accounting که وظیفه احراز اصالت و مدیریت کاربران و ... را بر عهده دارد به صورت یک سرویس مجزا تعریف شده است. چرا که در حالت یکپارچه به دلیل این‌که تمامی سرویس‌ها در کنار یکدیگر قرار داشته‌اند، تمامی فرآیندهای احراز اصالت و مدیریت حساب‌های کاربری داخل همان نرم‌افزار و در کنار کدهای سرویس‌های موجود استفاده شده است، ولی در حالت **Microservice** حتما باید به صورت یک سرویس مستقل پیاده شود.



معماری یکپارچه



معماری میکرو سرویس

ادغام میکرو سرویس‌ها (ارتباط بین سرویس‌ها و پروسس‌های داخلی)

به طور قطع یکی از مهم‌ترین مباحث در طراحی یک معماری Microservice موفق، بررسی و پیاده‌سازی ارتباط بین سرویس‌های موجود است. به دلیل اینکه اکثر servicec ها از پروتکل HTTP و نوع دیتای JSON استفاده می‌کنند، به همین دلیل ادغام آن‌ها به دلیل سادگی این ساختارها راحت‌تر خواهد بود.

راه دیگر ارتباط بین service ها از طریق یک bus یا درگاه با کمترین میزان routing است تا هدایت درخواست به سرویس‌های داخلی را انجام دهد. به همین دلیل مدل‌های مختلفی برای ادغام میکرو سرویس‌ها ارائه شده است که در ادامه به صورت مختصر بررسی خواهند شد.

تبادل پیام در میکرو سرویس

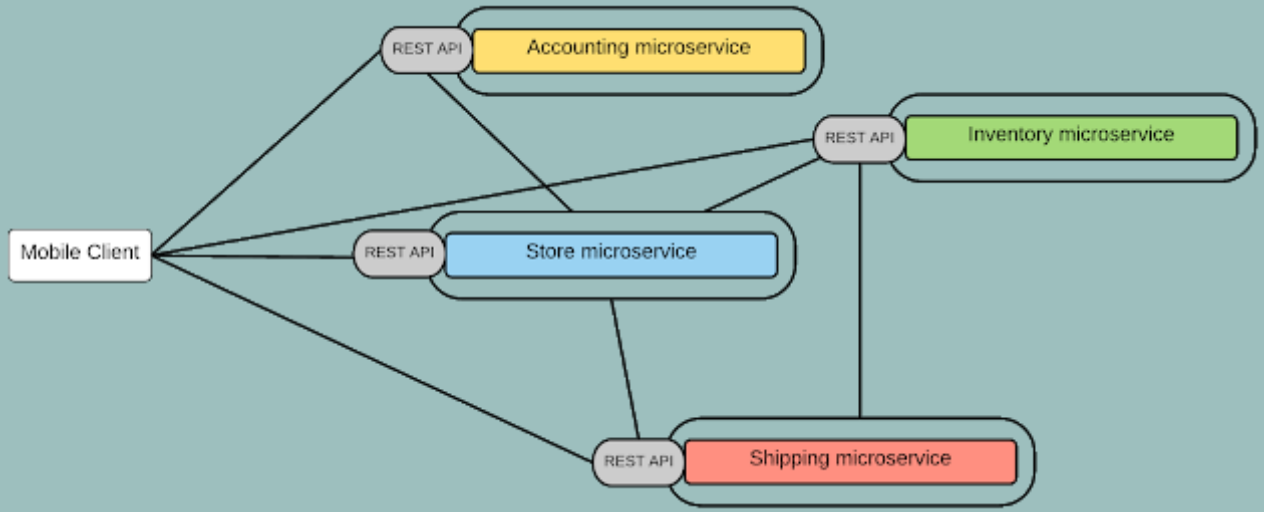
به طور کلی، در معماری microservice دو راه برای برقرای ارتباط بین سرویس ها وجود دارد:

1 - ارتباط همزمان(Synchronous or Point-to-Point):

2 - ارتباط غیرهمزمان (Asynchronous or publish-subscribe):

ارتباط point-to-point فراخوانی سرویس‌ها به صورت مستقیم

در این مدل ارتباط، هر کدام از سرویس‌ها به صورت مستقیم و با استفاده از پروتکل HTTP با هم در ارتباط هستند ، که نیاز به پاسخ فوری و مستقیم از سرور دارد. هر میکرو سرویس APIهای خود را ارائه کرده و در اختیار سایر سرویس‌ها قرار می‌دهد. حال هر کدام از سرویس‌ها در صورت نیاز از این APIها استفاده کرده و اطلاعات مورد نیاز را از سرویس مورد نظر دریافت می‌کنند. به عنوان مثال شکل زیر این معماری را نشان می‌دهد.



همانطور که مشاهده می‌کنید در صورتی که کاربر از اپ موبایل بخواهد اطلاعات فروشگاه را دریافت کند، درخواست خود را به سرویس فروشگاه ارسال می‌کند. حال سرویس فروشگاه در صورت نیاز برای دریافت اطلاعات حساب شخصی یا خریدهای وی به سرویس مربوطه یک درخواست HTTP ارسال می‌کند و اطلاعات مربوطه را دریافت کرده و از این اطلاعات در سرویس فعلی استفاده کرده و جواب مورد نظر را برای کاربر ارسال می‌کند.

Considerations when using REST

Tight Coupling

همیشه برخی از سرویس‌ها در اطراف interface (مخصوصاً در اطراف داده‌ها) وجود خواهد داشت، اما هنگام ساخت سرویس RESTful، توسعه‌دهنده فرض می‌کند که پیام فقط باید به یک مکان (سرویس) تحویل داده شود. وقتی سرویس یا مؤلفه دیگری در آینده ایجاد می شود و به داده این سرویس نیاز دارد چه اتفاقی می افتد؟ مطمئناً باید کد را به‌روزرسانی کنیم تا end point جدید را اضافه کنیم، اما این نقص را نشان می‌دهد: coupling غیر ضروری. در این سناریوها، ما باید API های RESTful بارها و بارها آبدیت کنیم .

Blocking

هنگام فراخوانی یک سرویس REST، سرویس ما در انتظار پاسخ مسدود می شود. این امر عملکرد برنامه را کاهش می دهد زیرا این thread می تواند درخواست های دیگر را پردازش کند.

2 - ارتباط غیرهمزمان (Asynchronous or publish-subscribe):

در این نوع ارتباط که از مکانیزم publish-subscribe استفاده می شود، نیازی به **پاسخ فوری** از سرور نیست و پیام ها (درخواست ها) به یک کارگزار پیام (مانند RabbitMQ) ارسال می شوند و سرویس مربوطه آن را گرفته و پردازش می کند. به عنوان مثال کاربران انتظار دریافت هیچ پاسخی از سمت سرور بلافاصله پس از ارسال ندارند. به عنوان مثال کاربری که لاگین کرده و نیاز است تا یک ایمیل فعال‌سازی برای وی ارسال شود. این ارسال ایمیل می‌تواند توسط سرویسی مجزا اجرا شود. در این سناریوها، پروتکل‌های پیام‌رسانی ناهمزمان مثل AMQP، STOMP یا MQTT بسیار مورد استفاده قرار می‌گیرند.

ویژگی های این ارتباط

Losse Coupling

با استفاده از Messaging ، به طور خاص مدل publish-subscribe ، سرویس ها از سایر خدمات اطلاعی ندارند. آنها از رویدادهای جدید مطلع می شوند، سپس آن اطلاعات را پردازش کرده و اطلاعات جدید را منتشر می کنند. سپس این اطلاعات جدید می تواند توسط هر تعداد سرویس مصرف شود. این Loss Coupling به میکروسرویس ها اجازه می دهد همیشه برای تغییرات بی پایان در برنامه در دسترس باشند.

Non-Blocking :

با برنامه های پیام رسانی Asynchronous، می توانیم به جای منتظر ماندن برای پاسخ، درخواستی ارسال کنیم و درخواست دیگری را پردازش کنیم.

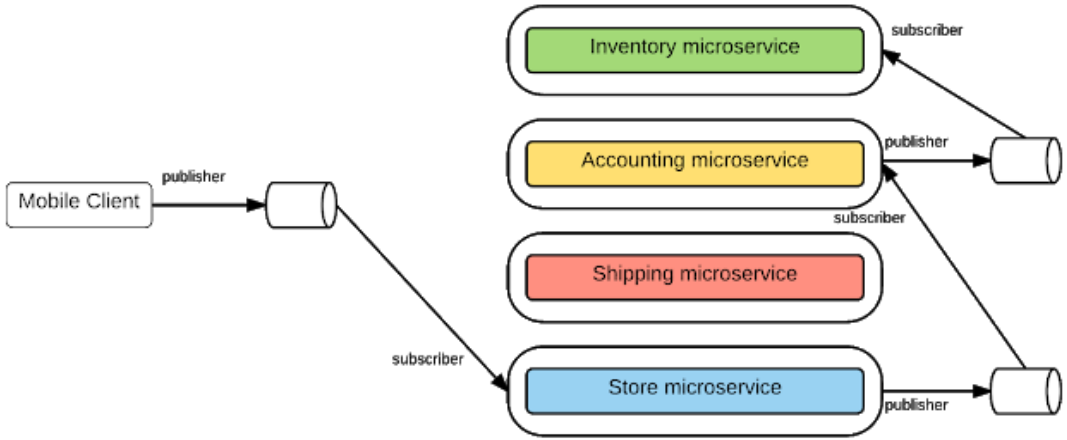
Simple to scale

معماری میکروسرویس به ما کمک می کند تا برنامه را در حین رشد مقیاس بندی کنیم. از آنجایی که هر سرویس کوچک است و فقط یک وظیفه را انجام می دهد، هر سرویس باید بتواند در صورت نیاز رشد کند یا کوچک شود. Event-driven architecture و مدل messaging، مقیاس‌پذیری میکروسرویس‌ها را آسان می‌کند، زیرا آنها جدا هستند و مسدود نمی‌شوند.

ارتباط با استفاده از message broker

یک راه دیگر در پیاده‌سازی ارتباطات ادغام سرویس‌ها با سناریوهای استفاده از کانال‌های پیامی **Asynchronous** است. به عنوان مثال درخواست‌ها به صورت یک طرفه ارسال شوند و با استفاده از مکانیزم publish-subscribe بر روی صف‌ها و تاپیک‌های مختلف پخش شوند. سپس سرویس‌هایی که مصرف کننده آن پیام هستند به تاپیک مربوطه subscribe شده و از این اطلاعات استفاده می‌کنند و دوباره در صورت نیاز اطلاعات را انتشار می‌دهند که این چرخه می‌تواند بین میکرو سرویس‌ها جریان داشته باشد.

یکی از مزیت‌های این مدل استقلال کامل سرویس publisher و subscriber است که broker میانی این اطلاعات را بافر کرده و در صورتی که سرویس شلوغ بود منتظر می‌ماند تا آن سرویس آماده شده و اطلاعات را در اختیار سرویس قرار دهد. شکل زیر این مدل را بهتر توضیح می‌دهد:



ارتباط پیامی غیر همزمان(**Asynchronous**) با استفاده از مدل pub-sub

Message Broker

Message Broker چیست؟

نرم افزاری است که برنامه ها، سیستم ها و سرویس ها را قادر می سازد تا با یکدیگر ارتباط برقرار کرده و تبادل اطلاعات داشته باشند. Message Broker با ترجمه پیام ها به پروتکل های پیام رسان رسمی این کار را انجام می دهد بنابراین به سرویس های وابسته این امکان را می دهد تا با یکدیگر "صحبت" کنند، حتی اگر به زبان های مختلف نوشته شده باشند.
Message Broker پیام های دریافتی را از برنامه فرستنده دریافت می کند و آن ها را برای برنامه گیرنده ارسال می کند. به این ترتیب فرستنده و گیرنده می توانند کاملا مجزا باشند.

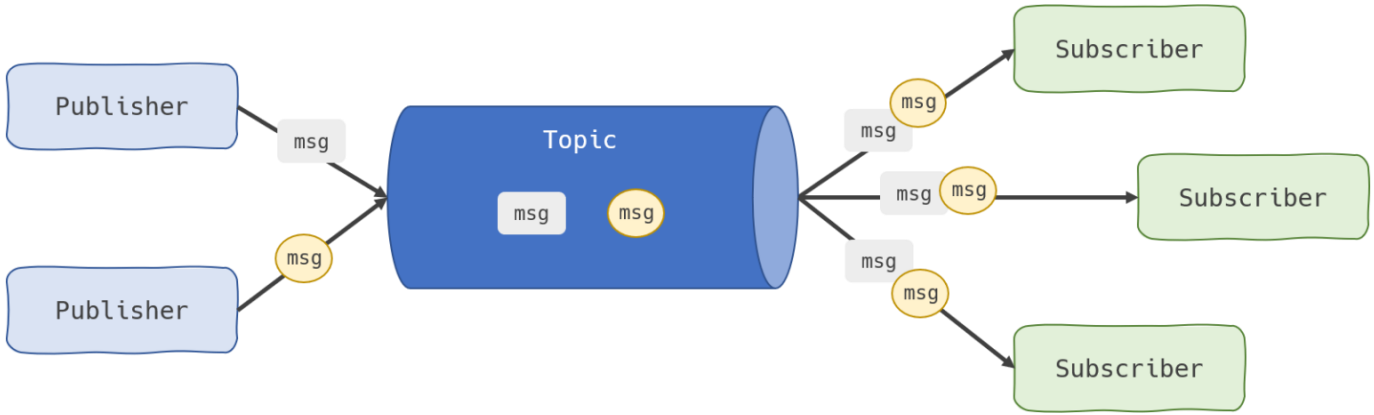
پیام

اطلاعاتی است که از یک برنامه ی فرستنده برای یک برنامه ی گیرنده ارسال می شود. پیام می تواند سیگنالی برای اطلاع رسانی به یک برنامه برای شروع پردازش یک کار باشد، یا اینکه به یک برنامه درباره اتمام کار برنامه دیگری خبر دهد. پیام همچنین می تواند اطلاعات حیاتی مورد نیاز یک برنامه را برای پردازش همان برنامه در خود نگه دارد.

A **Pub/Sub (Publish/Subscribe) system** and **queues** are both messaging patterns used in distributed systems to facilitate communication between different components or services.

Publish/subscribe

الگوی Pub/Sub یا (Publish/Subscribe) راهی برای ارتباط چند سرویس با یکدیگر از طریق انتشار پیام‌هایی برای یک topic است که سپس بین Subscriber های آن topic توزیع می‌شود.(همه Subscriber علاقه مند یک نسخه از پیام را دریافت می کنند.) تو این پترن یک پیام می‌تونه به صورت همزمان توسط چندین consumer دریافت و پردازش بشه. به عبارت دیگه یک publisher می‌تونه چندین consumer رو از یک رویداد خاصی باخبر کنه.

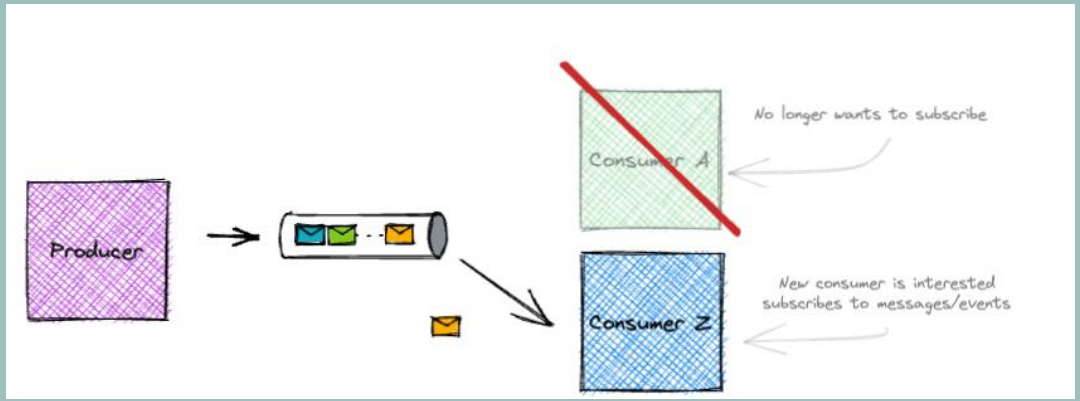


چون پیام های منتشر شده در یک موضوع بلافاصله به همه مشترکین تحویل داده می شود. در نتیجه، Pub/Sub گزینه های زیادی از نظر تحویل پیام ارائه نمی دهد.

Features

Flexible

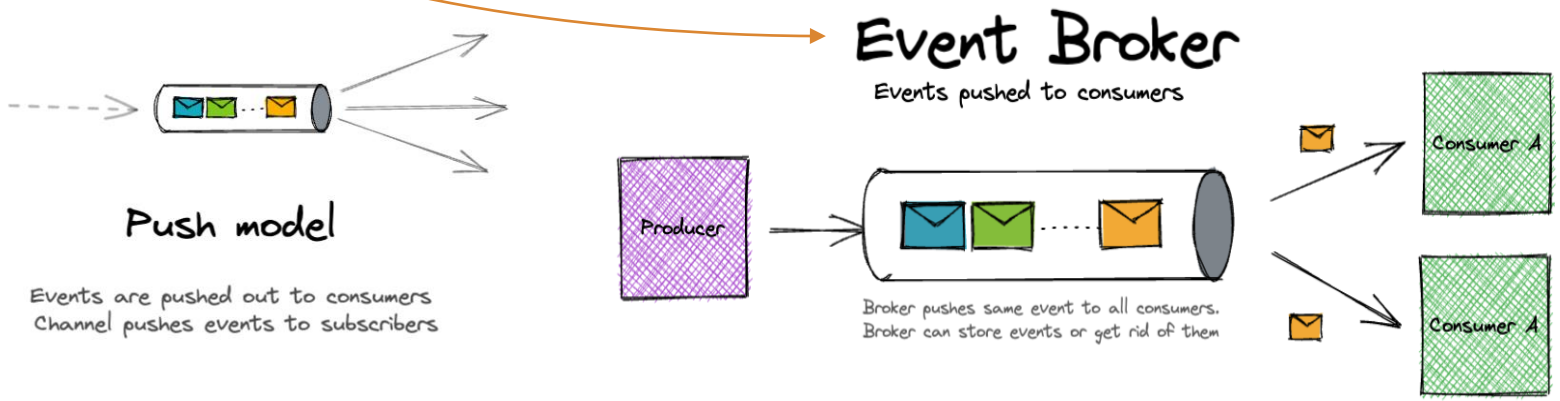
Consumers می آیند و می روند، به احتمال زیاد producer شما اهمیتی نمی دهد. این ویژگی به تیم‌ها انعطاف‌پذیری می‌دهد تا با تغییر نیازمندی‌های کسب‌وکار، Consumer ها را اضافه کنند یا حتی آنها را حذف کنند.



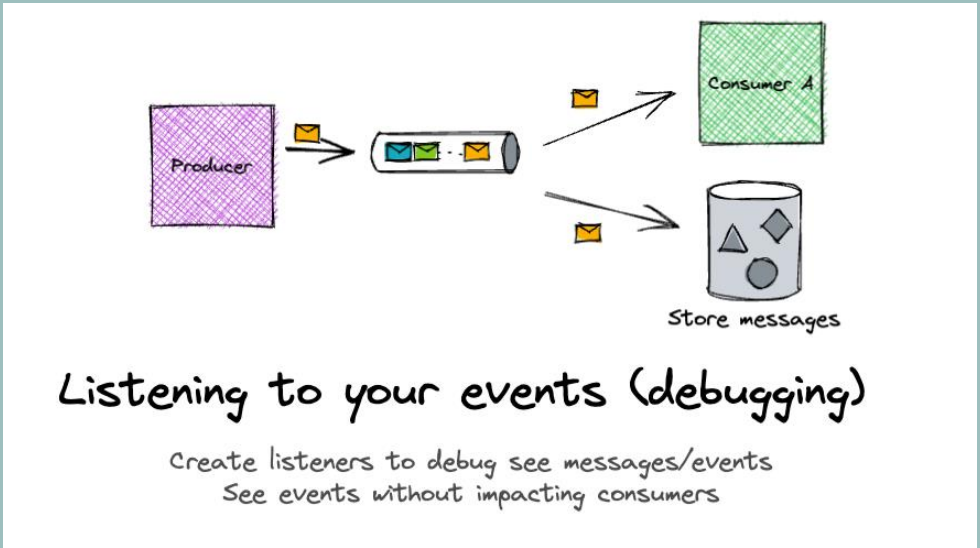
Push model

With pub/sub messages/events are often pushed to downstream consumers.

به‌جای اینکه consumers مجبور باشند رویدادهای یک صف/کانال را listen و pull کنند.(Message Queuing)



Listening to your events



As consumers are independent from each other (own copy of the message), you can create new subscribers onto the channel and use this for debugging.

ما دو نوع subscription بیشتر نداریم:

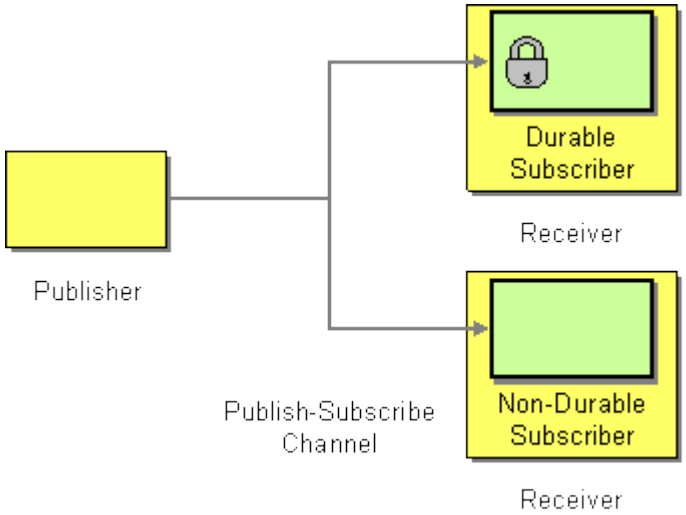
1. An ephemeral subscription (Non-durable subscription)

تو این حالت تا زمانی که consumer فعال هست، پیام ها توسط آنها در حال پردازش اما با shut down شدن consumer پیامهایی که باید توسط consumerپردازش میشدن از بین میرن.

2. A durable subscription

یک اشتراک بادوام پیام‌ها را برای یک مشترک غیرفعال ذخیره می‌کند و این پیام‌های ذخیره‌شده را هنگام اتصال مجدد مشترک تحویل می‌دهد. به این ترتیب، مشترک با وجود قطع ارتباط، هیچ پیامی را از دست نخواهد داد.

durable subscription هیچ تأثیری بر رفتار subscriber یا messaging system در زمانی که subscriber فعال است (به عنوان مثال، متصل) ندارد. subscriber متصل، چه subscription بادوام باشد و چه غیر بادوام، یکسان عمل می کند. **تفاوت در نحوه رفتار message broker** هنگام قطع ارتباط subscriber است.



Message Queuing

همه ی پیام ها توسط یک برنامه ی تولید کننده (Producer)، تولید می شوند و به یک صف پیام، push می شوند. این فرایند به Enqueuing معروف است. پیام های push شده در این صف باقی می مانند تا زمانی که یک برنامه ی مصرف کننده (Consumer) متصل شده و این پیام ها را واکشی کند. این فرایند به Dequeuing معروف است.

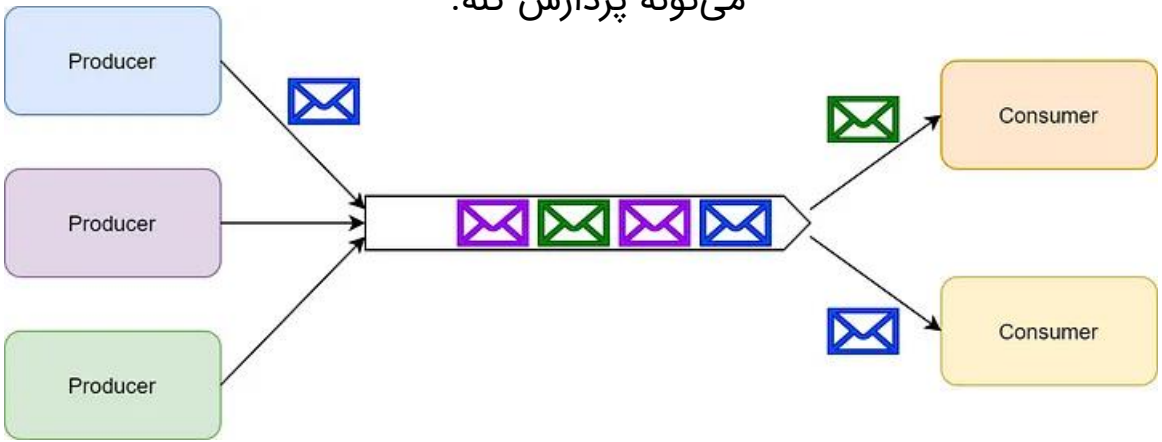
هر دو فرایند Enqueueو Dequeue به طور مستقل توسط برنامه های Producerو Consumer انجام می شوند و این امکان را داریم که یک پیام را در انتظار دریافت یک Consumer، در یک صف پیام نگه داریم. بنابراین فرایند pushشدن پیام به صف توسط برنامه Producerو مصرف شدن آن توسط برنامه Consumer، به عنوان Message Queueing (صف بندی پیام) شناخته می شود.

در نتیجه دو مفهوم مهم مطرح:

producers کسی یا برنامه ای هست که پیام اولیه رو تولید میکنه و به صف پردازش ارسال میکنه و consumersکسی یا برنامه ای هست که این پیام هارو به ترتیب از صف میگیره و پردازش میکنه

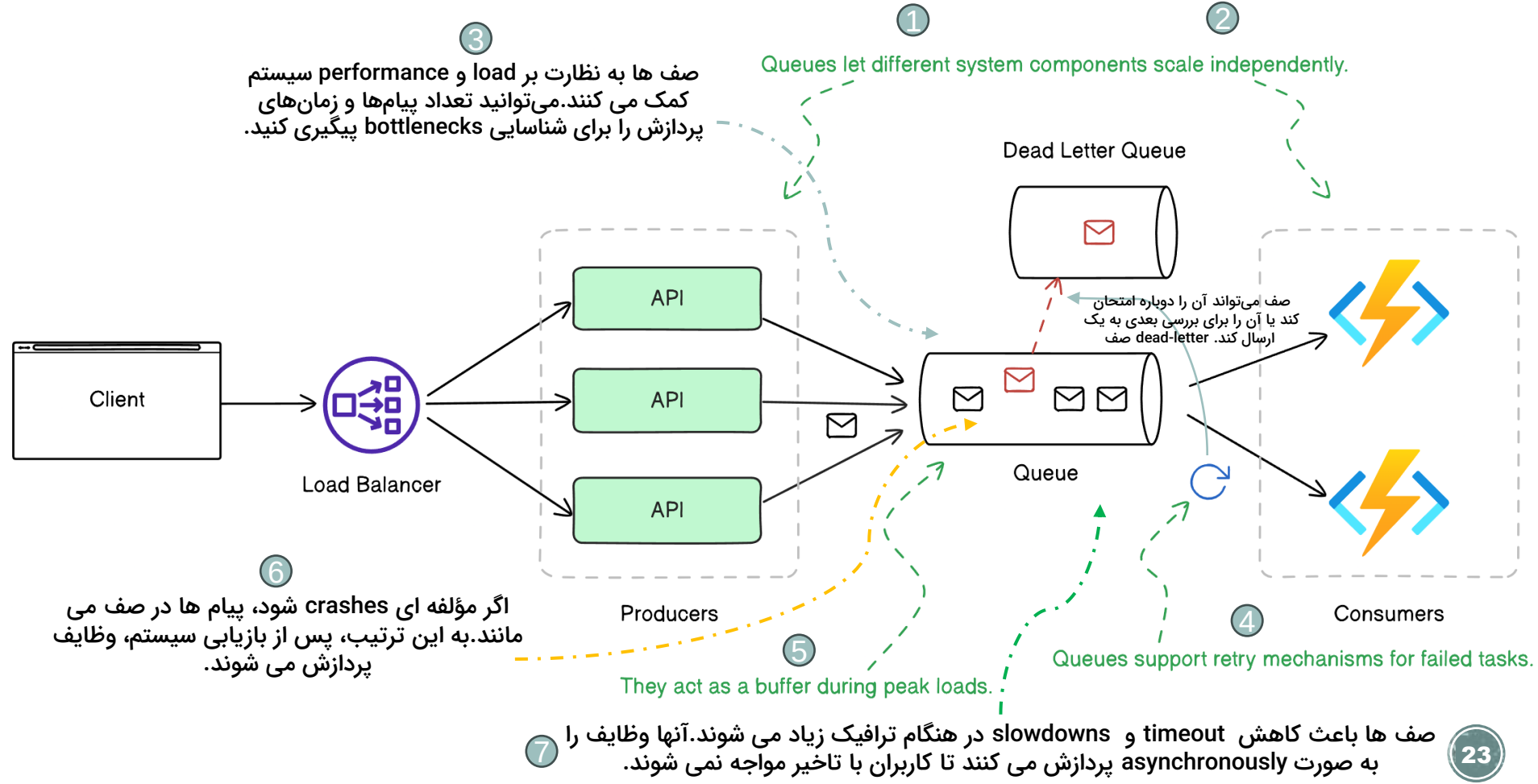
Message queueing

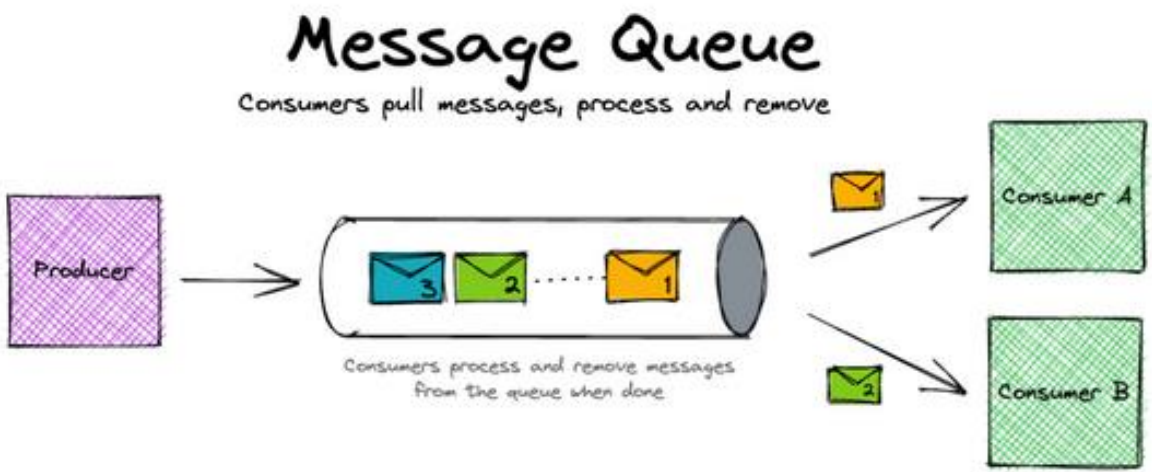
تو این پترن **queue (صف) ، producer و consumer** رو از هم جدا میکنه. پیام ها در یک صف قرار می گیرند و پیام ها بین consumer ها تقسیم می شوند و یک consumer پیام را مصرف می کند و آنها را پردازش می کند و پیام‌ها مصرف شده و پس از آن حذف می‌شوند. چندین producer می‌تونن به یک صف یکسان پیام ارسال کنن. همچنین زمانی که یک consumer درحال پردازش یک پیامه یا **locked**میشه یا از دسترس خارج میشه، علاوه بر این هر consumer فقط یک نوع پیام خاصی رو می‌تونه پردازش کنه.



به عنوان یک نکته جانبی، اگر مصرف کننده نتواند یک پیام خاص را پردازش کند، Message Broker معمولا پیام را به صف که برای مصرف کنندگان دیگر در دسترس قرار گرفته است، برمی گرداند. علاوه بر تفکیک زمانی، صف ها به ما اجازه می دهند تا تولیدکنندگان و مصرف کنندگان را **به طور مستقل مقیاس بندی** کنیم و **همچنین درجه ای از تحمل خطا** در برابر خطاهای پردازش را فراهم کنیم.

7 کاری که صف ها برای سیستم شما انجام می دهند:





Queues are often used when you want to distribute tasks or workload among multiple consumers. They can be used for load balancing and processing tasks in parallel.

هنگام انتخاب یک message broker جنبه های زیادی وجود دارد که باید در نظر بگیرید:

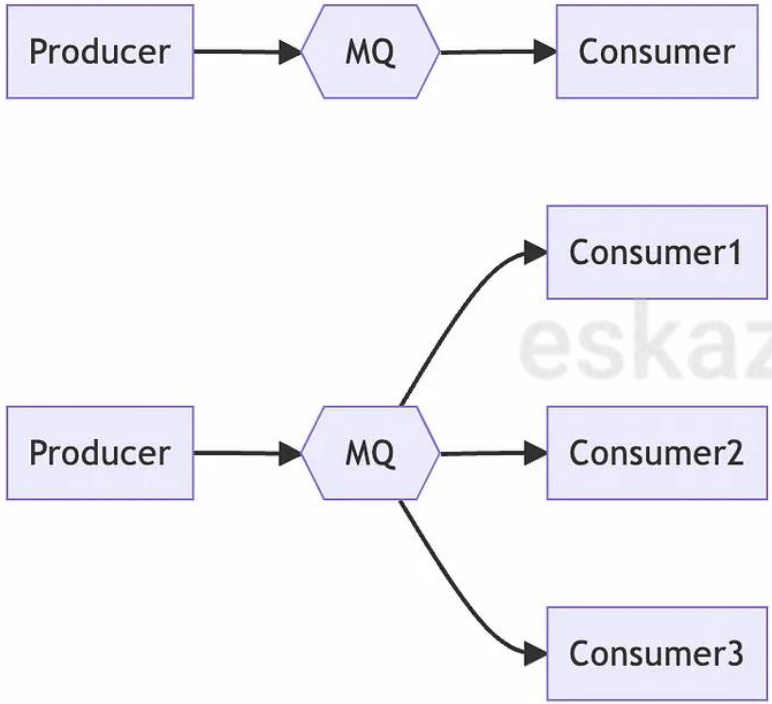
propagation
delivery
persistence
consumer groups

Propagation

Propagation (انتشار) به این معنی است که پیام ها چگونه توسط صف پیام منتقل می شوند!
۲ نوع انتشار وجود دارد:

- ✓ one-to-one
- ✓ one-to-many (fan-out)

Propagation



یک به یک بسیار ساده است. تولید کننده پیام **(producer)** پیامی به صف می فرستد و این پیام تنها توسط یک مصرف کننده (**consumer**) دریافت می شود. از سوی دیگر، **یک به چند(one-to-many)** پیامی است که می تواند به چندین مصرف کننده تحویل داده شود. لازم به ذکر است که تولیدکننده **(producer)** فقط یک پیام ارسال کرده است، اما پیام می تواند به **بسیاری از گیرنده ها** منتقل شود. چنین رفتاری را fan-out نیز می نامند.

Delivery

بیشتر Message Broker ها ضمانت تحویل خود را دارند. سه تضمین مشترک وجود دارد:

- ✓ At-most-once
- ✓ At-least-once
- ✓ Exactly-once

At-most-once

در بیشتر مواقع دستیابی به At-most-once نسبتا آسان است . می توان گفت که تمام سیستم های صف بندی این تضمین را دارند. در ضمانت تحویل حداکثر یکبار (At-most-once)، پیام حداکثر یک بار به گیرنده تحویل داده می شود. یعنی این احتمال وجود دارد که پیام به هیچ وجه ارسال نشود اما در صورت تحویل فقط یک بار ارسال می شود.(اگر مصرف کننده آن را دریافت کرد، اما به خوبی از پس آن بر نیامد ، پیام از بین برود، ناپدید می شود و بازیابی مجدد پیام غیرممکن است). این رویکرد اجتناب از تحویل های تکراری را در اولویت قرار می دهد

At-least-once

At-least-once توسط برخی از سیستم های Message Broker همچون RabbitMQ ، Kafka و ... استفاده می شود . در مقایسه با At-most-once، با حداقل یک بار ضمانت تحویل، پیام تضمین شده است که حداقل یک بار تحویل داده شود. این تضمین می کند که پیام گم نمی شود، اما امکان تحویل تکراری را باز می کند.برای دستیابی به حداقل یک بار تحویل، سیستم ممکن است از مکانیسم هایی مانند تلاش مجدد استفاده کند. اگر تلاشی برای تحویل ناموفق باشد، سیستم تلاش‌های بیشتری را تا موفقیت انجام می‌دهد.

Exactly-once

Exactly-once سخت ترین تضمین است. این تضمین می کند که یک پیام دقیقاً یک بار و تنها یک بار، بدون امکان تکرار یا از دست دادن، تحویل داده شود.دستیابی به تحویل دقیقاً یکبار چالش برانگیز است و اغلب شامل پروتکل ها و مکانیسم های پیچیده ای برای رسیدگی به خرابی ها و تلاش های مجدد بدون معرفی موارد تکراری است.

Persistence

Persistence (ماندگاری) به این معنی است که آیا پیام بعد از ارسال به سیستم ناپدید خواهد شد یا خیر.
همچنین سه نوع پایداری وجود دارد:

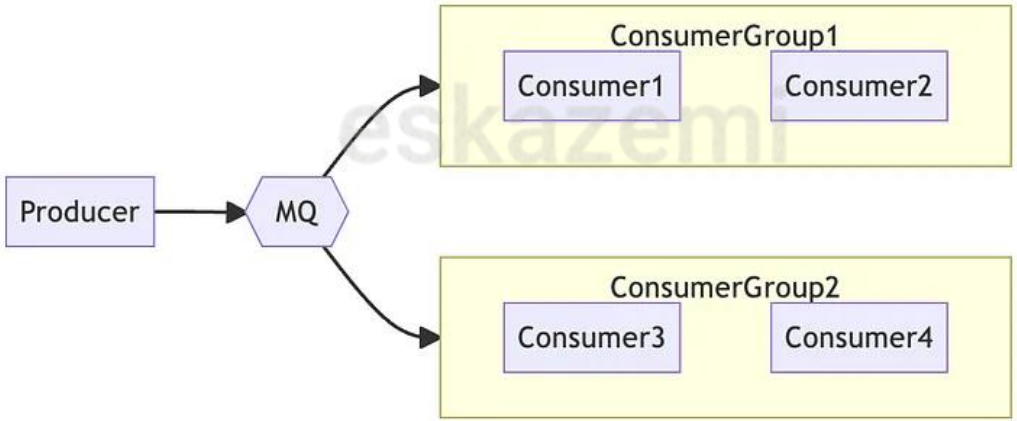
- ✓ In-memory
- ✓ In-disk
- ✓ Hybrid

نکته جالب این است که آیا باقی ماندن پیام ها در دیسک کندتر است؟ واقعا نه. بستگی به این دارد که پایداری چگونه اجرا شود. کافکا از LSM - Tree برای دستیابی به توان عملیاتی زیاد استفاده می کند؛ علاوه بر این، بهتر از RabbitMQ است که از حافظه (memory) استفاده می کند. مثال دیگری در Cassandra وجود دارد، Cassandra سرعت نوشتن بسیار بالایی دارد و از LSM - Tree نیز استفاده می کند.

Hybrid یک مورد خاص ، ترکیب in-memory و in-disk است. به منظور بهبود عملکرد نوشتن، سیستم صف بندی ابتدا روی حافظه می نویسد و سپس وارد دیسک می شود. RabbitMQ نمونه ای معمولی از یک Hybrid است. با این حال RabbitMQ قادر است به عنوان یک in-desk نیز پیکربندی شود.

Consumer Group

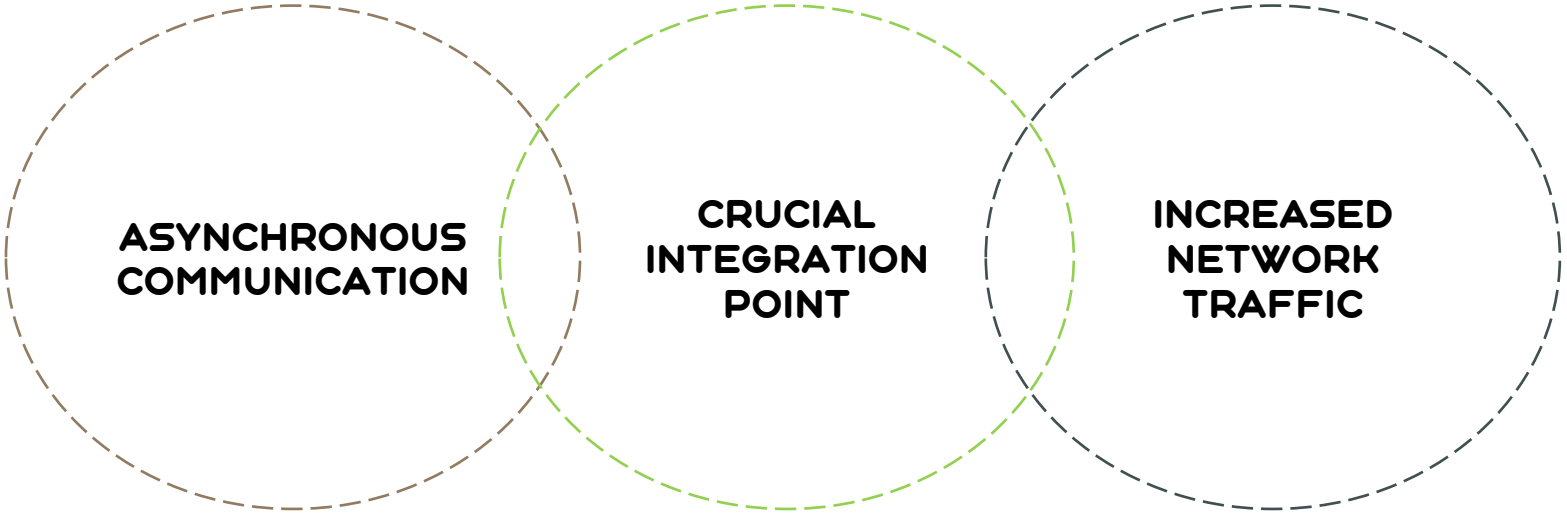
Consumer Group مهم ترین ویژگی یک Message Broker است. پردازش یک پیام معمولا زمان می برد، بنابراین ما باید از مصرف کنندگان بیشتری برای مقابله با پیام ها استفاده کنیم،. در صحنه هایی که مصرف کننده های گروهی داریم، هم هدف یک به یک و هم هدف یک به چند به جای یک مصرف کننده واحد به گروهی از مصرف کنندگان تبدیل می شوند.



Advantages of Message-Based Systems

1. Sender only needs to khow location of message broker
2. Multiple receivers possible
3. Easily add new receivers
4. Decoupling
5. Queueing

Consequences of Message-Based Systems



کدام message Broker برای شما مناسب است؟

بهترین message Broker برای شما به **نیازهای خاص شما** بستگی دارد. در اینجا عواملی وجود دارد که باید هنگام انتخاب message Broker در نظر بگیرید:

- ✓ **Data size**: If you need to process and store large amounts of data, then Kafka is a good choice.
- ✓ **Real time:** If you need to process data in real time and If you're dealing with massive amounts of data , then Kafka is also a good choice.
- ✓ **Ease of use :** If you are looking for an easy-to-use message broker, then RabbitMQ is a good option. Scalability: If you need a message broker that can scale to meet your needs, then ActiveMQ is a good choice.
- ✓ **Cost:** The cost of a message broker can vary depending on the features you need.

in eskazemi