

الگوهای مُدرن در ترم افزار

دانیال خسروی

جواد جعفری

ترجمه و تأليف:

۷۱.۱.۲

الگوهای مُدرن در نرم افزار

طراحی سیستم مقدماتی

منابع:

مقالات ByteByteGO

کتاب system design – karan

مقالات مايكروسافت

ترجمه و تأليف:

دانیال خسروی

جواد جعفری

آخرین بهروزرسانی ۴ مرداد ۱۴۰۳



С нами бог

System-design.ir

سخن نویسندگان:

تاجهان بود از سرآدم فراز کس نبود از راز دانش بی نیاز
مردمان بخود اندرهزمان راهِ دانش را به هرگونه زبان
گرد کردن و گرامی داشتند تا به سکن اندرهی بنشستند
دانش اندر دل چراغ روشنست وزهمه بد، بر تن تو، جو شنست

استفاده از این کتاب رایگان است و هر گونه کپی برداری از آن با ذکر منابع بلامانع است.

مقدمه

در این کتاب از چند منع مختلف به خصوص مقالات [bytebytego system](#) و کتاب [design](#) اثر [karan](#) و برخی مقالات سایت مایکروسافت استفاده شده است. در واقع این کتاب مقدمه بر کتاب قبلی که توسط دانیال و جواد ترجمه شده است و «طراحی سیستم‌های نرم‌افزاری» نام دارد، می‌باشد. در واقع محتوی این کتاب ترکیبی از چند منع به همراه نظرات نویسنده‌های آن است. این کتاب به صورت رایگان بوده و استفاده از مطالب آن با توجه به ذکر منع بلامانع است.

تشکر می‌کنیم از خانواده‌هایمان برای هزاران دلیلی که به ما کمک کردن به خصوص برای فرصلهایی که از آن‌ها دریغ کردیم و روی نوشتمن این کتاب گذاشتم.

تشکر می‌کنم از «[حسین میرجبرئیلی](#)» و «[ابراهیم بیاگوی](#)» برای بررسی متن و پیشنهادات سازنده در ساختار این کتاب.

برای ارتباط با نویسنده‌گان این کتاب و بیان پیشنهادات با دانیال خسروی و جواد جعفری ما بسیار تلاش کردیم تا متن ترجمه شده بدون خطأ و اشتباه باشد؛ اما راهیافتن خطأ در کتاب گریزناپذیر است. مترجمین بسیار ممنون خواهند شد که این اشتباهات را توسط ایمیل یادآوری کنید.

راههای ارتباطی با ما:
وبلاگ کتاب‌ها - [system-design.ir](#)

دانیال خسروی d.kh@mail.com - [github](#)
جواد جعفری javadsdn@outlook.com - [github](#)

فهرست مطالب

۲۱	طراحی سیستم چیست؟
۲۲	معنای اعداد نسخه‌های نرم‌افزاری چیست؟
۲۴	مدل‌های توسعه نرم‌افزار: آبشاری، چابک، تکراری، مارپیچ و RAD
۲۶	پروتکل‌های ارتباطی
۲۷	شبکه تحویل محتوا CDN
۳۰	جستجوی سیستم نام دامنه (DNS)
۳۲	انواع کوئیری در سیستم DNS
۳۲	انواع رکوردها
۳۴	زیر دامنه (Subdomain)
۳۴	DNS Zones
۳۴	DNS Caching
۳۵	Reverse DNS
۳۶	۶ نوع از رایج‌ترین سرورها
۳۷	GraphQL در مقابل REST API
۴۰	مقایسه بین RPC و RESTful
۴۱	نحوه عملکرد gRPC
۴۳	چیست؟ Webhook
۴۵	HTTP 1.0 -> HTTP 1.1 -> HTTP 2.0 -> HTTP 3.0 (QUIC)
۴۷	مقایسه SOAP و REST و GraphQL و RPC
۴۸	توسعه مبتنی بر کد در مقابل توسعه مبتنی بر API
۵۰	HTTP کدهای وضعیت
۵۱	تفاوت URN, URI, URL

۵۳	اجزای مهم یک URL
۵۴	وقتی یک URL را در مرورگر خود تایپ می‌کنید چه اتفاقی می‌افتد؟
۵۷	تفاوت بین کوکی (Cookie) و سشن (Session)
۵۸	مرورگرهای مدرن چگونه کار می‌کنند؟
۶۰	بررسی روش‌های API
۶۰	چگونه دسترسی امن به API وب برای وبسایت خود طراحی کنیم؟
۶۳	چگونه API‌های مؤثر و امن طراحی کنیم؟
۶۴	بهبود عملکرد API
۶۶	HTTP, POST, GET ... مفاهیم رایج
۶۸	Tقلیب‌نامه‌ی REST API
۷۱	۱۲ نکته برتر برای امنیت API
۷۳	۹ نوع تست API
۷۵	چگونه از Pagination در طراحی API استفاده کنیم؟
۷۷	موارد مهم درباره هدرهای HTTP
۷۹	الگوهای معما ری نرم‌افزار
۸۱	بررسی معما ری های N-Tier
۸۴	چگونه الگوهای طراحی را یاد بگیریم؟
۸۶	الگوهای طراحی کلیدی (Design Patterns)
۸۹	تسلط بر اصول طراحی - SOLID
۸۹	۱۲ فاکتور مهم در طراحی اپلیکیشن
۹۲	تفاوت بین Process و Thread چیست؟
۹۴	مقایسه هم زمان (Concurrency) با موازی سازی (Parallelism)
۹۷	چگونه برنامه های کامپیوتری اجرا می شوند؟
۹۹	چگونه C++, جاوا و پایتون کار می کنند؟
۱۰۰	اصطلاح برتر برنامه نویسی

۱۰۳.....	۱۰۳..... ۱۰۴..... ۱۰۵..... ۱۰۶..... ۱۰۷..... ۱۰۸..... ۱۰۹..... ۱۱۰.....	۱۰۳..... ۱۰۴..... ۱۰۵..... ۱۰۶..... ۱۰۷..... ۱۰۸..... ۱۰۹..... ۱۱۰.....	۱۰۳..... ۱۰۴..... ۱۰۵..... ۱۰۶..... ۱۰۷..... ۱۰۸..... ۱۰۹..... ۱۱۰.....	۱۰۳..... ۱۰۴..... ۱۰۵..... ۱۰۶..... ۱۰۷..... ۱۰۸..... ۱۰۹..... ۱۱۰.....
۱۱۲.....				سیستم نظارت و هشدار برای متريکها
۱۱۴.....				قابلیت مشاهده: لاغ‌گیری، ردیابی و معیارها
۱۱۶.....				کدام پایگاهداده برای سیستم جمع‌آوری متريکها مناسب است؟
۱۱۹.....				مدل‌های Push در مقابل Pull
۱۲۲.....				ذخیره‌سازی
۱۲۲.....				چرا یک درایو حالت جامد (SSD) سریع است؟
۱۲۵.....				بررسی ذخیره‌سازهای مختلف
۱۲۶.....				ذخیره‌ساز بلوکی (Block Storage)
۱۲۷.....				ذخیره‌سازی شیء (Object storage)
۱۲۷.....				ذخیره‌سازی فایل (File storage)
۱۲۸.....				حجم (Volumes)
۱۲۸.....				ذخیره‌ساز متصل به شبکه (NAS)
۱۲۸.....				سیستم فایل توزیع شده (HDFS) Hadoop
۱۲۹.....				آرایه افروندگی دیسک‌های مستقل (RAID)
۱۳۱.....				اعداد تأخیر که باید بدانید.
۱۳۳.....				انواع حافظه و ذخیره‌سازی
۱۳۵.....				طراحی S3
۱۳۸.....				بارگذاری فایل‌های بزرگ
۱۴۱.....				کد تصحیح خطأ (Erasure coding)

۱۴۳	بررسی پایگاهداده
۱۴۳	چه پایگاهداده‌ای را باید استفاده کنم؟
۱۴۴	پایگاهداده‌های مختلف در سرویس‌های ابری
۱۴۵	راهنمای انتخاب پایگاهداده مناسب
۱۴۷	پایگاهداده‌های SQL
۱۴۹	پایگاهداده‌های NoSQL
۱۵۰	سندگرا (Document)
۱۵۰	کلید - مقدار (Key-Value)
۱۵۱	پایگاهداده گرافی (Graph)
۱۵۲	سری زمانی (Time Series)
۱۵۳	ستون گسترده (Wide-column)
۱۵۴	چندمدلی (Multi-model)
۱۵۵	?Column-based یا row-based
۱۵۷	پایگاهداده‌های NoSQL در مقابل SQL
۱۶۱	تکثیر پایگاهداده (Database Replication)
۱۶۵	ایندهکس‌ها (indexes)
۱۶۷	تراکنش (Transaction)
۱۷۰	۸ ساختار داده‌ای که پایگاهداده‌ها را قادر تمند می‌کنند.
۱۷۲	چگونه یک عبارت SQL در پایگاهداده اجرا می‌شود؟
۱۷۴	دستورات اصلی پایگاهداده SQL
۱۷۶	نرم‌السازی و غیر نرم‌السازی
۱۸۳	CAP
۱۸۵	PACELC
۱۸۶	سطوح ایزولاسیون پایگاهداده
۱۸۸	نمایش یک کوئری در SQL
۱۸۹	زبان SQL

۱۹۰.....	ACID به چه معناست؟
۱۹۲.....	BASE به چه معناست؟
۱۹۴.....	۶ مدل برتر پایگاهداده
۱۹۷.....	چگونه داده‌های حساس را در یک سیستم مدیریت کنیم؟
۲۰۰.....	پایگاهداده‌های بُرداری
۲۰۲.....	۶ مورد برتر از کاربردهای Elasticsearch
۲۰۴.....	تراکنش توزیع شده (Distributed Transactions)
۲۰۸.....	فردراسیون پایگاهداده (Database Federation)
۲۱۰.....	چگونه داده‌ها را در سیستم‌های توزیع شده مدیریت کنیم؟
۲۱۳.....	پارتبیشن‌بندی افقی و پارتبیشن‌بندی عمودی
۲۱۷.....	۴ الگوریتم برتر Sharding
۲۱۹.....	الگوهای خواندن داده‌های replica شده ۱
۲۲۱.....	الگوهای خواندن داده‌های replica شده ۲
۲۲۳.....	تعادل بین تأخیر زمانی و یکپارچگی
۲۲۵.....	۷ استراتژی ضروری برای مقیاس‌بزیری پایگاهداده
۲۲۶.....	مقیاس‌دهی ۱۰۰ برابری در Postgres
۲۲۸.....	بن‌بست (Deadlock) چیست؟
۲۳۰.....	تفاوت‌های بین قفل‌های پایگاهداده
۲۳۳.....	قفل‌گذاری خوش‌بینانه
۲۳۵.....	تفاوت پایگاهداده‌های serverless با پایگاهداده‌های ابری سنتی
۲۳۷.....	چرا PostgreSQL محبوب‌ترین است؟
۲۳۹.....	بررسی کش (Cache)
۲۳۹.....	کش و memory پردازنده
۲۴۰.....	کاربردهای کش
۲۴۴.....	چه زمانی از کش نباید استفاده کرد؟
۲۴۵.....	بررسی برخی مفاهیم در کش:

۲۴۶.....	بهترین استراتژی‌های کش کردن چیست؟
۲۴۸.....	چرا Redis این قدر سریع است؟
۲۴۹.....	Redis در مقابل Memcached
۲۵۰.....	چه کاربردهایی دارد؟ Redis
۲۵۲.....	حمله گم شدن داده در Cache
۲۵۴.....	۸ استراتژی برتر برای خروج از کش
۲۵۷.....	تکامل معماری Redis
۲۶۰.....	وقتی سیستم کش دچار مشکل می‌شود؟
۲۶۲.....	: پایگاه داده Redis in-memory
۲۶۴	مقالات Big data
۲۶۶.....	تکامل Big data
۲۶۷.....	بررسی Data Pipelines
۲۶۹.....	سرعت رشد بالا داده‌ها
۲۷۱	معماری‌های میکروسرویس
۲۷۱.....	(تک‌هسته‌ای) monolith
۲۷۲.....	ماژولار Monolith
۲۷۲.....	میکروسرویس‌ها (Microservices)
۲۷۷.....	میکروسرویس در مقابل معماری سرویس‌گرا (SOA)
۲۷۸.....	چرا به میکروسرویس نیاز ندارید؟
۲۸۰.....	بررسی یک نمونه از معماری میکروسرویس
۲۸۲.....	بهترین روش‌های میکروسرویس
۲۸۳.....	چگونه میکروسرویس‌ها با یکدیگر همکاری و تعامل دارند؟
۲۸۵.....	فناوری رایج برای میکروسرویس‌ها چیست؟
۲۸۷.....	چه کاری انجام می‌دهد؟ API gateway
۲۹۱.....	۹ اصل اساسی قبل از ساختن میکروسرویس‌ها

۲۹۴ ۹ جزء ضروری یک برنامه میکروسرویس

۲۹۶ **بررسی Load Balancing**

۳۰۲ چرا Nginx یک پروکسی «معکوس» نامیده می شود؟

۳۰۴ الگوریتم های رایج توزیع بار (Load Balancing)

۳۰۶ کلید استفاده از توزیع کننده بار (Load Balancer)

۳۰۸ برآورد تقریبی در طراحی سیستم

۳۱۰ **بررسی صفات پیام**

۳۱۰ صفات پیام (Message Queue)

۳۱۱ ۴ مورد از متدائل ترین انواع صفات

۳۱۳ Message Brokers

۳۱۵ Enterprise Service Bus (ESB)

۳۱۷ تحول معماری صفات پیام: از IBM MQ به Pulsar تا RabbitMQ

۳۱۹ حداقل یکبار، حداقل یکبار و دقیقاً یکبار

۳۲۴ ۶ الگوی برتر پیام رسانی ابری

۳۲۷ چرا Kafka سریع است

۳۲۹ ۵ مورد از کاربردهای برتر Kafka

۳۳۱ آیا Kafka می تواند پیام ها را گم کند؟

۳۳۴ مقایسه Kafka و Rabbitmq

۳۳۵ پردازش Stream در مقابل پردازش Batch

۳۳۷ چرا به message brokers نیاز داریم؟

۳۳۹ **سرویس ها و معماری های ابری**

۳۴۰ چیست cloud native

۳۴۲ چیست IaaS/PaaS/SaaS

۳۴۳ خوشه بندی (Clustering)

۳۴۸ مفاهیم و تعاریف الگوهای طراحی cloud native

۳۵۳.....	الگوهای طراحی cloud native
۳۵۳.....	الگوی سفیر - پروکسی (Ambassador - Proxy)
۳۵۳.....	الگوی Bulkhead
۳۵۴.....	کش جانبی (Cache-aside)
۳۵۵.....	قطع کننده مدار (Circuit Breaker)
۳۵۷.....	تراکنش جبرانی (Compensating Transaction)
۳۵۸.....	مصرف کنندگان رقیب (Competing Consumers)
۳۵۸.....	تجمعیع منابع محاسباتی (Compute Resource Consolidation)
۳۵۹.....	معماری رویدادمحور (Event-Driven Architecture - EDA)
۳۶۱.....	منبع رویداد (Event Sourcing)
۳۶۴.....	تفکیک مسئولیت فرمان و کوئری (CQRS)
۳۶۶.....	مخزن پیکربندی خارجی (External Configuration Store)
۳۶۷.....	هویت فدراتیو (Federated Identity)
۳۶۷.....	الگوی بک‌اند برای فرانت‌اند (Backend For Frontend - BFF)
۳۶۸.....	دروازه‌بان (Gatekeeper)
۳۶۸.....	Health Endpoint Monitoring
۳۶۹.....	جدول ایندکس (Index Table)
۳۷۰.....	انتخاب رهبر (Leader Election)
۳۷۰.....	دید تحقق یافته (Materialized View)
۳۷۱.....	لوله‌ها و فیلترها (Pipes and Filters)
۳۷۲.....	صف اولویت‌دار (Priority Queue)
۳۷۳.....	ترازوی بارگذاری مبتنی بر صف (Queue-based Load Leveling)
۳۷۳.....	تکرار مجدد (Retry)
۳۷۶.....	پیکربندی زمان اجرا (Runtime Reconfiguration)
۳۷۶.....	سرپرسی عامل زمان‌بندی (Scheduler Agent Supervisor)
۳۷۷.....	قطعه‌بندی (Sharding)

۳۷۸.....	میزبانی محتوای استاتیک (Static Content Hosting)
۳۷۸.....	محدودسازی (Throttling)
۳۷۹.....	بررسی Valet Key
۳۸۰.....	ضد الگوهای Cloud Native
۳۸۲.....	بازیابی فاجعه (Disaster Recovery)
۳۸۴.....	کشف سرویس (Service Discovery)
۳۸۷.....	کاهش هزینه‌های ابری
۳۸۹.....	۹ الگوی برتر معماری برای جریان داده و ارتباط
۳۹۲.....	Event Sourcing چیست؟ چرا با CRUD متفاوت است؟
۳۹۵.....	چگونه Event Sourcing را در سیستم‌ها ادغام کنیم؟
۳۹۷.....	SSO (ورود یکباره به چند سیستم) چیست؟
۳۹۹.....	کدام ارائه‌دهنده ابری باید هنگام ساخت یک راهکار داده‌های بزرگ استفاده شود؟
۴۰۱.....	AWS Lambda پشت‌صخنه

۴۰۴ CI/CD

۴۰۵.....	بخش اول - CI/CD با SDLC
۴۰۵.....	بخش دوم - تفاوت بین CD و CI
۴۰۶.....	بخش سوم - pipeline CI/CD
۴۰۷.....	Netflix Tech Stack (CI/CD Pipeline)
۴۰۹.....	چرخه توسعه نرم‌افزار چابک
۴۱۲.....	چگونه پیکربندی‌ها را در یک سیستم مدیریت کنیم؟
۴۱۴.....	مسیر کاری GitOps
۴۱۶.....	استراتژی‌های استقرار
۴۱۸.....	DevOps بررسی
۴۱۸.....	کتاب‌های DevOps
۴۱۹.....	DevOps در مقابل SRE در مقابل مهندسی پلتفرم

۴۲۱.....	Docker چگونه کار می کند؟
۴۲۴.....	Kubernetes یا k8s چیست؟
۴۲۶.....	چهار نوع برتر سرویس های Kubernetes در یک نمودار
۴۲۸.....	Kubernetes در مقابل Docker
۴۳۰.....	تفاوت ها بین مجازی سازی (VMware) و کانتینر سازی (Docker) چیست؟
۴۳۲.....	DevSecOps چیست؟
۴۳۴.....	بررسی GIT
۴۳۴.....	چطور دستورات گیت کار می کنند؟
۴۳۵.....	گیت چگونه کار می کند؟
۴۳۶.....	تفاوت Git rebase و Git merge
۴۳۷.....	بررسی دستورات ویژه در git
۴۳۹.....	انواع استراتژی های branch
۴۴۰.....	یک خط تغییر، زمان clone را تا ۹۹ درصد کاهش داد!
۴۴۳.....	بررسی Linux
۴۴۳.....	سیستم فایل لینوکس به زبان ساده
۴۴۵.....	۱۸ دستور پر کاربرد لینوکس که باید بدانید
۴۴۷.....	فرایند بوت لینوکس به زبان ساده
۴۴۹.....	مجوزهای فایل های لینوکس
۴۵۱.....	ارتباط بین فرایندها IPC
۴۵۳.....	برگه تقلب تجزیه لاغ
۴۵۵.....	۵ اصل مهم در لینوکس
۴۵۷.....	بررسی مباحث شبکه
۴۵۷.....	آشنایی با TCP/IP و مدل OSI
۴۵۸.....	تفاوت TCP و UDP
۴۶۱.....	IPv4 در مقابل IPv6، تفاوت ها چیست؟

۴۶۵.....	۸ پروتکل محبوب در شبکه
۴۶۷.....	مسیریابی ترافیک اینترنت.....
۴۶۸.....	داده‌ها چگونه بین برنامه‌ها منتقل می‌شوند؟.....
۴۷۰	بررسی امنیت در وب
۴۷۰.....	چگونه HTTPS کار می‌کند؟.....
۴۷۳.....	تفاوت بین تأیید اعتبار مبتنی بر JWT و Session چیست؟.....
۴۷۵.....	احراز هویت و دسترسی امن با OAuth 2.0.....
۴۷۷.....	۴ روش برتر برای مکانیزم‌های احراز هویت.....
۴۷۸.....	رمزگذاری در مقابل رمزگذاری در مقابل توکن‌سازی
۴۸۰.....	رمزگذاری متقارن در مقابل رمزگذاری نامتقارن.....
۴۸۲.....	OAuth 2.0، SSO و JWT توکن، کوکی، Google Authenticator چطور کار می‌کند؟.....
۴۸۴.....	ذخیره‌سازی رمزها
۴۸۶.....	توکن امن وب جیسون (JWT) به زبان ساده.....
۴۸۸.....	تأثیر دومرحله‌ای Google Authenticator چه می‌باشد؟.....
۴۹۱.....	۶ مورد از مهم‌ترین کاربردهای فایروال
۴۹۳.....	هر آنچه در مورد Cross-Site Scripting (XSS) نیاز دارد بدانید.....
۴۹۴.....	چه اتفاقی می‌افتد وقتی ssh hostname را تایپ می‌کنید؟.....
۴۹۶.....	روش‌های REST API Authentication
۴۹۸.....	فایروال به زبان ساده.....
۵۰۰	الگوریتم‌های کاربردی در سیستم‌های توزیع شده
۵۰۰.....	الگوریتم‌هایی مهم در طراحی سیستم
۵۰۲.....	Quadtree
۵۰۴.....	چگونه رستوران‌های نزدیک را به کمک Geohash پیدا می‌کنیم؟.....
۵۰۷.....	۱۰ الگوریتم گراف که باید بدانید.....
۵۱۰.....	۱۰ الگوریتم مهم در دنیای ما.....

بررسی طراحی سیستم‌ها در دنیای واقعی	۵۱۲
چهار شگفت‌انگیز طراحی سیستم چه چیزهایی هستند؟	۵۱۲
چگونه یک وب‌سایت را برای پشتیبانی از میلیون‌ها کاربر مقیاس‌پذیر کنیم؟	۵۱۴
راز موفقیت آمازون، نتفلیکس و اوبر چیست؟	۵۱۸
طراحی Google Docs	۵۲۰
تکنولوژی مورداستفاده‌ی نتفلیکس	۵۲۲
معماری توییتر	۲۰۲۲
معماری توییتر در ۲۰۱۳	۵۲۵
تکامل غیرمعمول معماری API نتفلیکس	۵۲۷
نکته مصاحبه طراحی سیستم	۵۲۸
تحول معماری میکروسرویس Airbnb طی ۱۵ سال گذشته	۵۳۰
Microrepo در مقابل Monorepo	۵۳۲
چگونه وب‌سایت Stack Overflow را طراحی می‌کنید؟	۵۳۵
چرا مانیتورینگ ویدئوی آمازون از Monolithic Serverless به تغییر یافت؟	۵۳۷
چگونه Disney Hotstar ۵ میلیارد ایموجی را در طول یک تورنمنت ثبت می‌کند؟	۵۴۰
چگونه Discord تریلیون‌ها پیام را ذخیره می‌کند.	۵۴۲
چگونه پخش زنده ویدئویی در یوتیوب، TikTok live یا Twitch کار می‌کند؟	۵۴۴
چگونه یک سیستم به طور خودکار باگ‌ها را برای ما شناسایی و رفع کند؟	۵۴۷
چگونه با اسکن کردن کد QR از کیف پول دیجیتال پرداخت کنیم؟	۵۴۹
سوال مصاحبه: طراحی Gmail	۵۵۱
مسیر دریافت ایمیل	۵۵۲
مسیر ارسال ایمیل	۵۵۴
سوال مصاحبه: طراحی Google Maps	۵۵۶
رندر کردن نقشه	۵۵۸
موتور کدنویسی هوش مصنوعی	۵۶۰
موتور کدنویسی هوش مصنوعی - جدول زمانی ChatGPT	۵۶۱

..... ۵۶۳	چگونه سیستم‌های شبیه به ChatGPT کار می‌کنند؟
..... ۵۶۶	مدیریت یک خرابی گسترده
..... ۵۶۷	چگونه شناسه‌های منحصر به فرد تولید کنید؟
..... ۵۶۹	چگونه از آدرس‌های تکراری در ابعاد گوگل جلوگیری کنیم؟
..... ۵۷۱	نحوه ارسال نوتیفیکیشن در slack
..... ۵۷۲	چگونه آمازون نرم‌افزارهای خود را می‌سازد و اداره می‌کند؟
..... ۵۷۴	بررسی طراحی سیستم‌های مالی و بانکی
..... ۵۷۴	مبادله خارجی در پرداخت
..... ۵۷۶	مطابقت سفارش‌های خرید و فروش
..... ۵۷۸	طراحی بورس اوراق بهادر
..... ۵۸۰	طراحی یک سیستم پرداخت
..... ۵۸۲	طراحی یک سیستم فروش/حراج
..... ۵۸۴	چگونه یک بورس اوراق بهادر مدرن به تأخیرهای میکروثانیه دست می‌یابد؟
..... ۵۸۵	شبکه پرداخت SWIFT
..... ۵۸۷	فرایند جایه‌جایی پول در نرم‌افزارهای بانکی
..... ۵۹۰	تطبیق و سازگاری در پرداخت بانکی ۱
..... ۵۹۲	بررسی پایگاهداده مناسب برای سرور جمع‌آوری متريک‌ها
..... ۵۹۴	تطبیق و سازگاری در پرداخت بانکی ۲
..... ۵۹۶	جلوگیری از شارژ شدن دوباره
..... ۵۹۸	امنیت در پرداخت بانکی
..... ۵۹۹	سایر کتاب‌ها

طراحی سیستم چیست؟

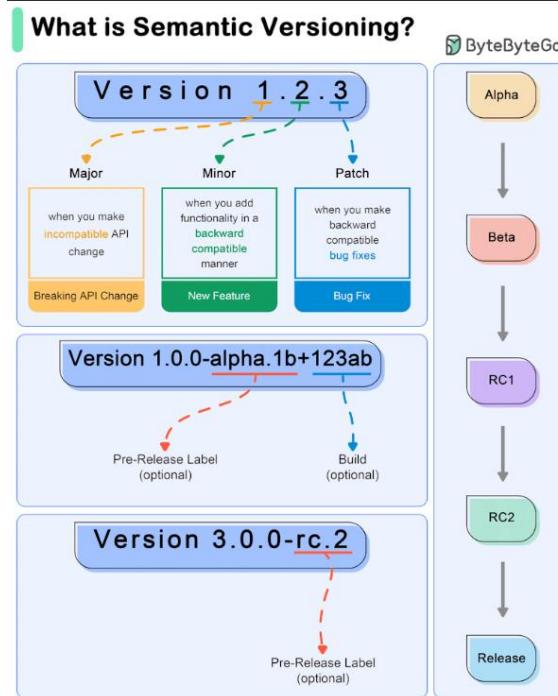
پیش از شروع این دوره، بباید در مورد اینکه طراحی سیستم چیست صحبت کنیم. طراحی سیستم فرایند تعریف معماری، رابط‌ها و داده‌ها برای سیستمی است که نیازمندی‌های خاصی را برآورده می‌کند. طراحی سیستم از طریق سیستم‌های منسجم و کارآمد نیازهای کسب‌وکار یا سازمان شما را برآورده می‌کند. این مسئله نیازمند رویکردی سیستمی برای ساخت و مهندسی سیستم‌ها است. یک طراحی سیستم خوب نیازمند آن است که به همه چیز فکر کنیم، از زیرساخت تا پایین به داده‌ها و نحوه ذخیره آنها.

چرا طراحی سیستم این‌قدر مهم است؟

طراحی سیستم به ما کمک می‌کند راه حلی را تعریف کنیم که نیازهای کسب‌وکار را برآورده کند. این یکی از اولین تصمیماتی است که می‌توانیم هنگام ساخت یک سیستم بگیریم. اغلب تفکر از سطح بالا ضروری است؛ زیرا اصلاح این تصمیمات بعداً بسیار دشوار است. همچنین با تکامل سیستم، استدلال و مدیریت تغییرات معماری را آسان‌تر می‌کند.

معنای اعداد نسخه‌های نرم افزاری چیست؟

نسخه‌بندی معنایی (Semantic Versioning - SemVer) یک طرح نسخه‌بندی برای نرم افزار است که هدف آن انتقال معنا در مورد تغییرات اساسی در یک انتشار است. از یک شماره نسخه سه‌قسمتی استفاده می‌کند: **.MAJOR.MINOR.PATCH**.



نسخه اصلی (MAJOR)

هنگامی که تغییرات ناسازگار در API وجود داشته باشد، افزایش می‌یابد.

نسخه فرعی (MINOR): هنگامی که قابلیت‌ها به صورت سازگار با گذشته اضافه می‌شود، افزایش می‌یابد.

نسخه اصلاح (PATCH): هنگامی که رفع اشکالات سازگار با گذشته انجام می‌شود، افزایش می‌یابد.

Workflow مثال

۱. گام توسعه اولیه

با نسخه ۱.۰ شروع کنید.

۲. اولین انتشار پایدار

به یک انتشار پایدار بررسید: ۱.۰.۰

۳. تغییرات بعدی

- ۴. انتشار اصلاحی (Patch Release): برای نسخه ۱.۰.۰ نیاز به رفع اشکال است.

به ۱.۰.۱ بهروزرسانی کنید.

- ۵. انتشار فرعی (Minor Release): یک ویژگی جدید سازگار با گذشته به ۱.۰.۳

اضافه می‌شود. به ۱.۰.۱ بهروزرسانی کنید.

- ۶. انتشار اصلی (Major Release): یک تغییر قابل توجه که با گذشته سازگار نیست

در ۱.۰.۲ معرفی می‌شود. پس به ۲.۰.۰ بهروزرسانی کنید.

۴. نسخه‌های ویژه و پیش انتشار

:**Pre-release Versions** (:

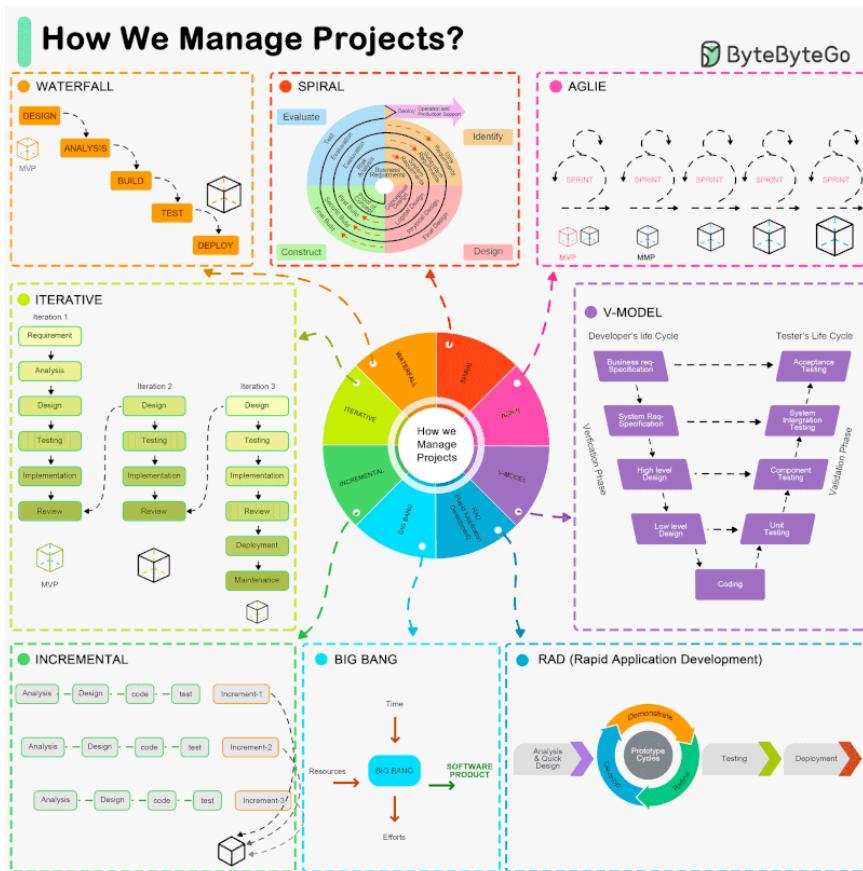
. 1.0.0-alpha, 1.0.0-beta, 1.0.0-rc.1

:**Build Metadata**

. 1.0.0+20130313144700

مدل‌های توسعه نرم افزار: آبشاری، چابک، تکراری، مارپیچ و RAD

چرخه عمر توسعه نرم افزار (SDLC) چارچوبی است که فرایند توسعه نرم افزار را به صورت سیستمی ترسیم می‌کند. در اینجا برخی از رایج‌ترین مدل‌ها آورده شده است:



۱. مدل آبشاری (Waterfall Model)

رویکردی خطی و ترتیبی پروژه را به گام‌های مجزا تقسیم می‌کند: که به طور معمول نیازمندی‌ها، طراحی، پیاده‌سازی، تأیید و نگهداری هستند.

۲. مدل چابک (Agile Model)

توسعه در بخش‌های کوچک و قابل مدیریت به نام sprint انجام می‌شود. متداول‌ترین رایج چابک شامل Extreme Programming - XP و Kanban ، Scrum است.

۳. مدل V (مدل اعتبارسنجی و تأیید):

توسعه‌ای بر اساس مدل آبشاری هر گام توسعه با یک‌فاز تست مرتبط است که شکل V را تشکیل می‌دهد.

۴. مدل تکراری (Iterative Model)

بر ساخت سیستمی به صورت تکراری تمرکز دارد. هر تکرار بر اساس تکرار قبلی ساخته می‌شود تا به محصول نهایی برسد.

۵. مدل مارپیچی (Spiral Model)

توسعه تکراری را با جنبه‌های سیستمی مدل آبشاری ترکیب می‌کند. هر چرخه شامل برنامه‌ریزی، تحلیل ریسک، مهندسی و ارزیابی است.

۶. مدل Big Bang

تمام کد با حداقل برنامه‌ریزی انجام می‌شود و کل نرم‌افزار یکجا ادغام و تست می‌گردد. این مورد به دلیل ریسک بالا، استفاده کمی دارد.

۷. مدل RAD (توسعه سریع برنامه‌های کاربردی):

بر نمونه‌سازی سریع و بازخورد سریع تأکید دارد. بر توسعه و تحويل سریع تمرکز دارد.

۸. مدل افزایشی (Incremental Model)

محصول به صورت افزایشی طراحی، پیاده‌سازی و آزمایش می‌شود تا محصول نهایی تکمیل شود و شباهت‌هایی با مدل تکراری دارد.

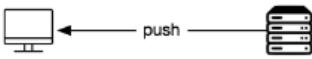
هر یک از این مدل‌ها مزایا و معایب خاص خود را دارند و انتخاب مدل مناسب اغلب به الزامات و محدودیت‌های خاص پروژه بستگی دارد.

پروتکل‌های ارتباطی

سبک‌های معماری API نحوه تعامل اجزای مختلف یک رابط برنامه‌نویسی اپلیکیشن (API^(۱)) با یکدیگر را تعریف می‌کنند. در نتیجه، با ارائه یک رویکرد استاندارد برای طراحی و ساخت API‌ها، کارایی، قابلیت اطمینان و سهولت ادغام با سایر سیستم‌ها را تضمین می‌کنند. در اینجا پرکاربردترین آن‌ها آورده شده‌اند:

API Architecture Styles

ByteByteGo.com

Style	Illustration	Use Cases
SOAP		XML-based for enterprise applications
RESTful		Resource-based for web servers
GraphQL		Query language reduce network load
gRPC		High performance for microservices
WebSocket		Bi-directional for low-latency data exchange
Webhook		Asynchronous for event-driven application

انواع معماری API

SOAP: بالغ، جامع، مبتنی بر XML که برای برنامه‌های سازمانی مناسب است.

RESTful: محبوب، پیاده‌سازی آسان، با استفاده از متدهای HTTP که ایده‌آل برای سرویس‌های وب است.

GraphQL: زبان کوئری‌ها، درخواست داده‌های خاص کاهش سربار شبکه، پاسخ‌های سریع‌تر را شامل می‌شود.

gRPC: مدرن، با کارایی بالا، با استفاده از Protocol Buffers مناسب برای معماری میکروسرویس‌ها است.

WebSocket: برقراری ارتباطات دوطرفه، پایدار و لحظه‌ای که بسیار ایده‌آل برای تبادل داده با تأخیر کم است.

Webhook: مبتنی بر رویداد، فراخوانی‌های HTTP غیرهم‌زمان که سیستم‌ها را در زمان وقوع رویدادها مطلع می‌کند.

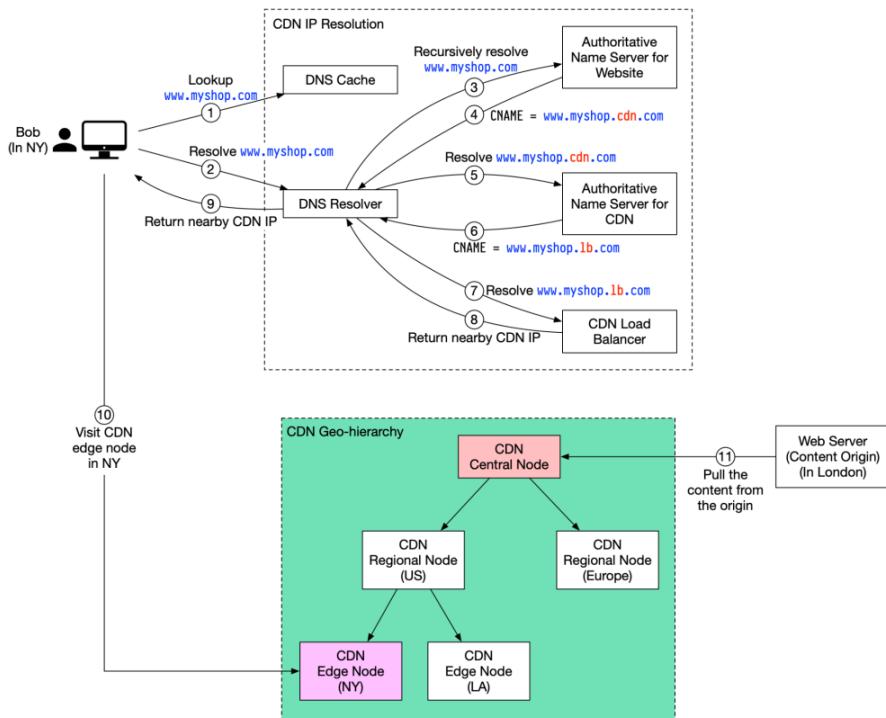
شبکه تحویل محتوا CDN

شبکه تحویل محتوا (CDN^۱) به سرورهایی که به صورت جغرافیایی توزیع شده‌اند (که سرورهای لبه نیز نامیده می‌شوند) اشاره دارد که محتواهای استاتیک و دینامیک را به سرعت تحویل می‌دهند. بیایید نگاهی بیندازیم که چگونه کار می‌کند.

فرض کنید باب که در نیویورک زندگی می‌کند می‌خواهد از یک وبسایت تجارت الکترونیک که در لندن مستقر است بازدید کند. اگر درخواست به سرورهایی که در لندن واقع شده‌اند برود، پاسخ بسیار کند خواهد بود؛ بنابراین، ما سرورهای CDN را نزدیک به جایی که باب زندگی می‌کند مستقر می‌کنیم، و محتوا از سرور CDN نزدیک بارگذاری خواهد شد.

نمودار زیر این فرایند را نشان می‌دهد:

content delivery network ^۱



- شخصی آدرس `www.myshop.com` را در مرورگر وارد می‌کند. مرورگر نام دامنه را در کش مربوط به DNS محلی جستجو می‌کند.
- اگر نام دامنه در کش DNS محلی وجود نداشته باشد، مرورگر به DNS resolver به می‌رود تا نام دامنه را بررسی کند. DNS resolver معمولاً در ارائه‌دهنده خدمات اینترنتی (ISP) قرار دارد.
- یک DNS resolver به صورت بازگشتی نام دامنه را `resolve` می‌کند. سرانجام، از سرور نام معتبر می‌خواهد تا نام دامنه را `resolve` کند.
- اگر از CDN استفاده نکنیم، سرور نام معتبر آدرس IP برای `www.myshop.com` را باز می‌گرداند. اما با استفاده از CDN، سرور نام معتبر که یک نام مستعار دارد که به `www.myshop.cdn.com` اشاره دارد (نام دامنه سرور CDN برمی‌گردد).

۵. این DNS resolver از سرور نام معتبر می‌خواهد تا www.myshop.cdn.com را resolve کند.

۶. Server name معتبر نام دامنه را برای متعادل‌کننده بار CDN به آدرس www.myshop.lb.com باز می‌گرداند.

۷. DNS resolver از متعادل‌کننده بار CDN می‌خواهد تا www.myshop.lb.com را resolve کند. متعادل‌کننده بار یک سرور لب CDN بهینه را بر اساس آدرس IP کاربر، محتوای درخواستی و بار سرور انتخاب می‌کند.

۸. آدرس IP CDN load balancer را برای www.myshop.lb.com باز می‌گرداند.

۹. اکنون سرانجام آدرس IP واقعی را برای بازدید دریافت می‌کنیم. آدرس IP را به مرورگر باز می‌گرداند.

۱۰. مرورگر به سرور لب CDN مراجعه می‌کند تا محتوا را بارگذاری کند. دو نوع محتوا در سرورهای CDN ذخیره می‌شود: محتوای استاتیک و محتوای دینامیک. اولی شامل صفحات استاتیک، تصاویر و ویدئوها است؛ دومی شامل نتایج محاسبات لبه است.

۱۱. اگر کش سرور لب CDN حاوی محتوا نباشد، به سمت سرور CDN منطقه‌ای می‌رود. اگر محتوا هنوز پیدا نشود، به سمت سرور مرکزی CDN می‌رود یا حتی به مبدأ - سرور وب در شهر لندن. به این شیوه توزیع CDN گفته می‌شود، جایی که سرورها به صورت جغرافیایی مستقر شده‌اند.

جستجوی سیستم نام دامنه (DNS)

DNS مانند یک دفترچه آدرس عمل می‌کند. نام‌های دامنه قابل خواندن برای انسان، به عنوان مثال (google.com) را به آدرس‌های IP قابل خواندن برای ماشین (142.251.46.238) ترجمه می‌کند.

برای دستیابی به مقیاس‌پذیری بهتر، سرورهای DNS در یک ساختار درختی سلسله‌مراتبی سازماندهی شده‌اند.

سه سطح اصلی سرورهای DNS وجود دارد:

۱. (.). آدرس‌های IP سرورهای نام دامنه سطح بالا (TLD^۱) . Root name server

را ذخیره می‌کند. در سراسر جهان ۱۳ Root name server منطقی وجود دارد.

۲. (.). آدرس‌های IP سرورهای نام معتبر را ذخیره می‌کند. چندین TLD name server

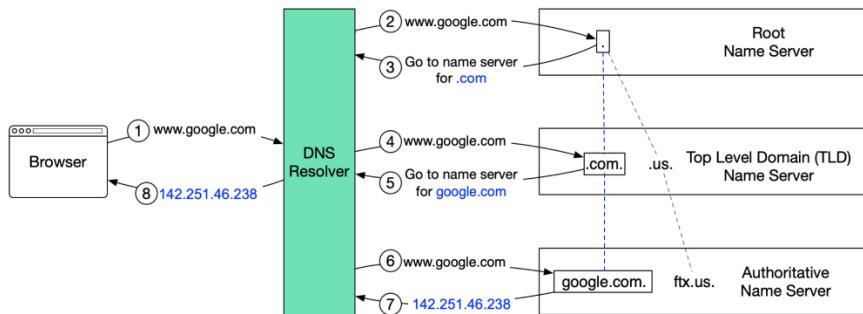
نوع نام TLD وجود دارد. برای مثال، TLD عمومی (.com, .org) و کد کشور (.test) و TLD آزمایشی (.us)

۳. (.). پاسخ‌های واقعی به درخواست DNS را ارائه Authoritative name server

می‌دهد. می‌توانید name serverهای معتبر را با ثبت‌کننده نام دامنه مانند Namecheap، GoDaddy و غیره ثبت کنید.

دیگر اگر زیر نشان می‌دهد که جستجوی DNS در پشت‌صحنه چگونه کار می‌کند:

¹ Top Level Domain



۱. DNS resolver در مرورگر تایپ می‌شود و مرورگر نام دامنه را به ارسال می‌کند.
۲. DNS resolver از یک root name server مربوط به DNS درخواست می‌کند.
۳. سرور ریشه به DNS resolver با آدرس یک سرور DNS سطح TLD پاسخ می‌دهد. در این مورد، برابر com است.
۴. سپس DNS resolver درخواستی به com.TLD ارسال می‌کند.
۵. سرور TLD با آدرس IP سرور نام دامنه google.com (سرور نام معتبر) پاسخ می‌دهد.
۶. DNS resolver یک درخواست به سرور نام دامنه ارسال می‌کند.
۷. آدرس IP برای google.com سپس از سرور نام به DNS resolver بازگردانده می‌شود.
۸. DNS resolver به مرورگر وب با آدرس IP (142.251.46.238) دامنه درخواست شده در ابتدا پاسخ می‌دهد.

جستجوهای DNS به طور میانگین بین ۲۰ تا ۱۲۰ میلی ثانیه طول می‌کشند تا تکمیل شوند (YSlow بر اساس).

انواع کوئی در سیستم DNS

سیستم DNS دارای سه نوع کوئی است:

بازگشتی (Recursive): در یک کوئی بازگشتی، یک کلاینت DNS نیازمند آن است که یک سرور DNS با رکورد منبع درخواستی یا یک پیام خطأ در صورت عدم یافتن رکورد توسط Resolver به کلاینت پاسخ دهد.

تکرارشونده (Iterative): در یک کوئی تکرارشونده، یک کلاینت DNS یک نام میزبان DNS را ارائه می‌دهد و DNS Resolver بهترین پاسخی را که می‌تواند ارائه می‌دهد. اگر Resolver رکوردهای DNS مربوطه را در کش خود داشته باشد، آنها را برمی‌گرداند. در غیر این صورت، کلاینت DNS را به root server یا name server معتبر دیگری که به نزدیک‌ترین به ناحیه DNS موردنیاز است را ارجاع می‌دهد. سپس کلاینت DNS باید کوئی را مستقیماً علیه سرور DNS که به آن ارجاع داده شده است را تکرار کند.

غیربازگشتی (Non-recursive): یک کوئی غیربازگشتی کوئی‌ای است که resolver قبل‌پاسخ آن را می‌داند. این کوئی یا بلافصله یک رکورد DNS را برمی‌گرداند؛ زیرا قبل‌آن را در یک کش محلی ذخیره کرده است، یا یک سرور نام DNS را که برای رکورد معتبر است (به این معنی که قطعاً IP صحیح را برای آن نام میزبان در اختیار دارد) را کوئی می‌کند. در هر دو مورد، نیازی به دورهای اضافی کوئی (مانند کوئی‌های بازگشتی یا تکرارشونده) نیست. بلکه پاسخی بلافصله به کلاینت برگردانده می‌شود.

انواع رکوردها

رکوردهای DNS (همچنین به عنوان فایل‌های منطقه شناخته می‌شوند) دستورالعمل‌هایی هستند که در سرورهای معتبر DNS وجود دارند و اطلاعاتی در مورد یک دامنه از جمله آدرس IP مرتبط با آن دامنه و نحوه رسیدگی به درخواست‌های آن دامنه را ارائه می‌دهند.

این رکوردها از مجموعه‌ای از فایل‌های متنی با فرمت شناخته شده به عنوان DNS syntax تشکیل شده‌اند. DNS syntax فقط رشتہ‌ای از کاراکترها است که به عنوان دستوراتی برای

سرور DNS استفاده می‌شود که به سرور DNS بگوید چه کاری انجام دهد. تمام رکوردهای DNS همچنین دارای "TTL" هستند و نشان می‌دهد که یک سرور DNS چند بار آن رکورد را refresh می‌کند.

رکوردهای زیادی وجود دارد، بباید به برخی از رایج‌ترین آنها نگاه کنیم:

- A (Address record): این رکوردی است که حاوی آدرس IP یک دامنه است.
- AAAA (IP Version 6 Address record): رکوردی که حاوی آدرس IPv6 برای یک دامنه است (برخلاف رکوردهای A که آدرس IPv4 را ذخیره می‌کنند).
- CNAME (Canonical Name record): یک دامنه یا زیر دامنه را به دامنه دیگری هدایت می‌کند، آدرس IP را ارائه نمی‌دهد.
- MX (Mail exchanger record): ایمیل را به یک سرور ایمیل هدایت می‌کند.
- TXT (Text Record): این رکورد به admin اجازه می‌دهد تا یادداشت‌های متنی را در رکورد ذخیره کند. این رکوردها اغلب برای امنیت ایمیل استفاده می‌شوند.
- NS (Name Server records): سرور نام را برای یک رکورد DNS ذخیره می‌کند.
- SOA (Start of Authority): اطلاعات admin در مورد یک دامنه را ذخیره می‌کند.
- SRV (Service Location record): پورتی را برای سرویس‌های خاص مشخص می‌کند.
- PTR (Reverse-lookup Pointer records): در جستجوهای معکوس^۱، یک نام دامنه خاص را ارائه می‌دهد.
- CERT (Certificate record): گواهینامه‌های کلید عمومی را ذخیره می‌کند.

time-to-live^۱

reverse lookups^۲

زیر دامنه (Subdomain)

زیر دامنه بخشی اضافی از نام دامنه اصلی ما است و معمولاً برای جداسازی منطقی یک وب سایت به بخش های مختلف استفاده می شود. ما می توانیم چندین زیر دامنه یا دامنه فرعی روی دامنه اصلی ایجاد کنیم. به عنوان مثال، blog.example.com که در آن blog زیر دامنه، example دامنه اصلی و com دامنه سطح بالا (TLD) است. نمونه های مشابه می توانند careers.example.com یا support.example.com باشند.

DNS Zones

یک DNS Zones بخشی مجزا از فضای نام دامنه است که به یک نهاد حقوقی مانند شخص، سازمان یا شرکتی که مسئول نگهداری از DNS Zones است، واگذار شده است. یک DNS Zones همچنین یک administrative function است که کنترل دقیق اجزای DNS مانند سرورهای نام های معتبر را امکان پذیر می کند.

DNS Caching

کش DNS (گاهی اوقات DNS resolver cache نامیده می شود) یک پایگاهداده موقتی است که توسط سیستم عامل نگهداری می شود و حاوی سوابقی از تمام بازدیدهای اخیر و تلاش های بازدید از وب سایتها و سایر دامنه های اینترنتی است. به عبارت دیگر، کش DNS تنها حافظه ای از جستجوهای اخیر DNS است که رایانه ما می تواند هنگام تلاش برای یافتن نحوه بارگیری وب سایت به سرعت به آن مراجعه کند.

سیستم نام دامنه (DNS) برای هر رکورد DNS یک زمان انقضا (TTL) تعیین می کند. تعداد ثانیه هایی را که رکورد را می توان توسط یک سرویس گیرنده یا سرور DNS در کش نگه داشت را مشخص می کند. هنگامی که رکورد در حافظه کش ذخیره می شود، هر مقدار TTL همراه با آن نیز ذخیره می شود. سرور همچنان به روزرسانی TTL رکورد ذخیره شده در کش ادامه می دهد و هر ثانیه را شمارش معکوس می کند. هنگامی که به صفر برسد، رکورد

از حافظه کش پاک می‌شود. در آن زمان، اگر درخواستی برای آن رکورد دریافت شود، سرور DNS باید فرایند resolve را آغاز کند.

Reverse DNS

جستجوی معکوس DNS در واقع یک کوئری DNS برای نام دامنه مرتبط با یک آدرس IP معین است. این کار بر عکس جستجوی معمول DNS انجام می‌شود که در آن سیستم DNS برای بازگرداندن یک آدرس IP مورد پرس‌وجو قرار می‌گیرد. فرایند حل معکوس^۱ از یک آدرس IP از رکوردهای PTR استفاده می‌کند. اگر سرور رکورد PTR نداشته باشد، نمی‌تواند جستجوی معکوس (reverse lookup) را حل کند.

reverse lookup به طور رایج توسط سرورهای ایمیل استفاده می‌شود. سرورهای ایمیل بررسی می‌کنند و می‌بینند که آیا یک پیام ایمیل قبل از آوردن آن به شبکه آنها از یک سرور معتبر آمده است. بسیاری از سرورهای ایمیل پیام‌های هر سروری که از جستجوی معکوس پشتیبانی نمی‌کند یا از سروری که به احتمال زیاد قانونی نباشد را رد می‌کنند. توجه: جستجوی معکوس DNS به طور جهانی پذیرفته نشده است؛ زیرا برای عملکرد عادی اینترنت ضروری نیستند.

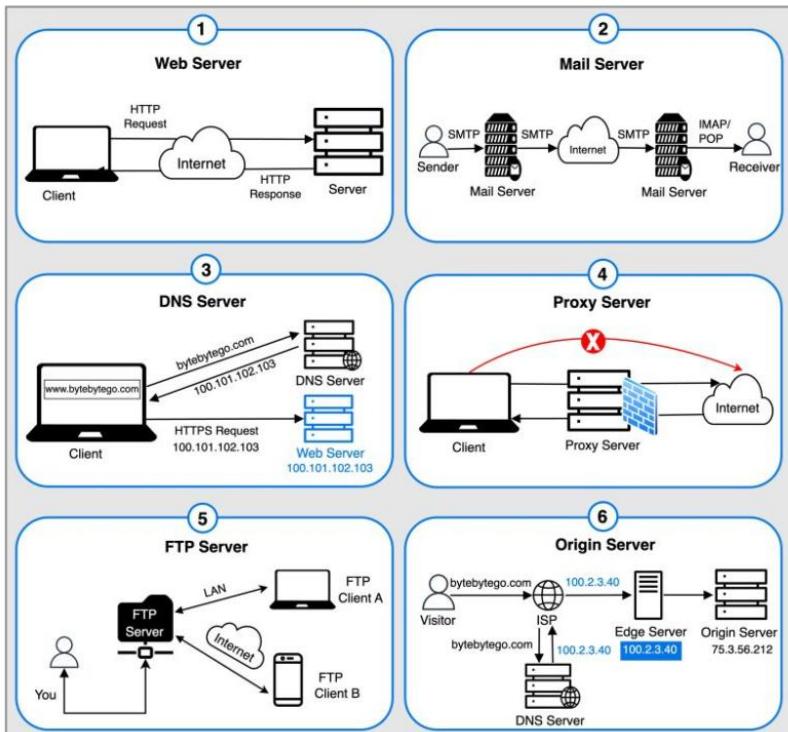
مثال‌ها

در اینجا برخی از راهکارهای مدیریت DNS پرکاربرد آورده شده است:

- Route53
- Cloudflare DNS
- Google Cloud DNS
- Azure DNS
- NS1

۶ نوع از رایج‌ترین سرورها

Top 6 Most Commonly Used Server Types blog.bytebytogo.com



۱. **Web Server:** وب‌سایت‌ها را میزبانی می‌کند و محتوای وب را از طریق اینترنت به کاربران تحویل می‌دهد.

۲. **Mail Server:** ارسال، دریافت و مسیریابی ایمیل‌ها را در سراسر شبکه‌ها مدیریت می‌کند.

۳. سرور DNS: نام‌های دامنه (مانند bytebytogo.com) را به آدرس‌های IP ترجمه می‌کند و به کاربران اجازه می‌دهد تا با نام‌های قابل خواندن توسط انسان به وب‌سایت‌ها دسترسی پیدا کنند.

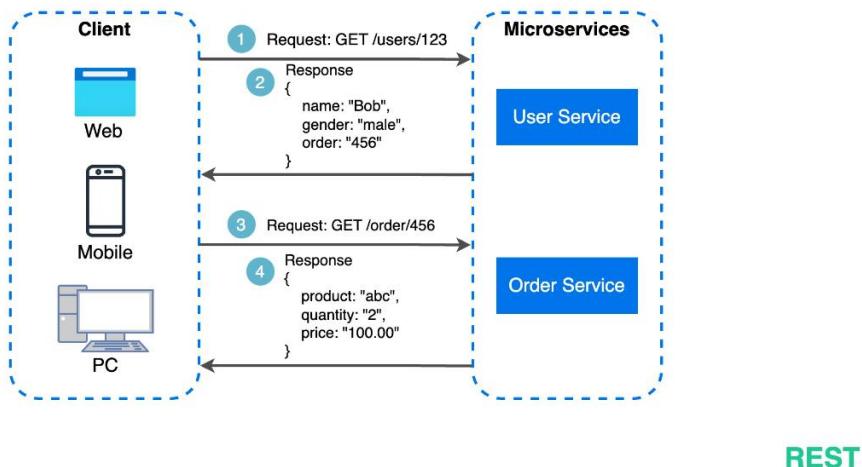
۴. **Proxy Server**: یک سرور واسطه است که به عنوان دروازه‌ای بین کاربران و سایر سرورها عمل می‌کند و امنیت اضافی، بهینه‌سازی عملکرد و ناشناس بودن را فراهم می‌کند.

۵. **FTP Server**: انتقال فایل بین کاربران و سرورها را از طریق شبکه تسهیل می‌کند.

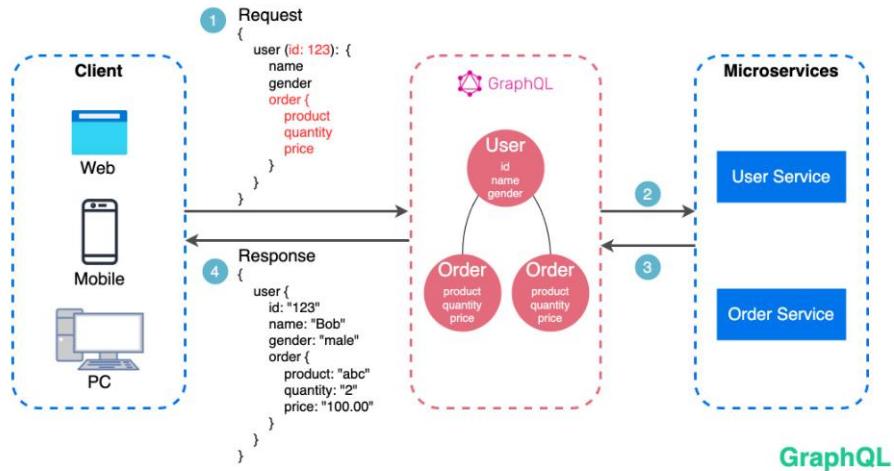
۶. **Origin Server**: منبع اصلی محتوایی را که در سرورهای لبه^۱ برای تحویل سریع تر به کاربران نهایی ذخیره و توزیع می‌شود را می‌بینی می‌کند.

GraphQL در مقابل REST API

در طراحی REST و GraphQL هر کدام نقاط قوت و ضعف خاص خود را دارند. شکل زیر مقایسه‌ای کوتاه بین REST و GraphQL را نشان می‌دهد.



^۱ edge servers

**:REST**

از متدهای استاندارد HTTP مانند DELETE, PUT, POST, GET برای عملیات

-

CRUD استفاده می‌کند.

-

برای مواردی که به رابطه‌های ساده و یکنواخت بین سرویس‌ها/برنامه‌های جداگانه نیاز دارد، به خوبی کار می‌کند.

-

اجرای استراتژی‌های Caching در آن ساده است.

-

نقشه‌ضعف آن این است که ممکن است برای جمع‌آوری داده‌های مرتبط از endpoint‌های جداگانه، نیاز به رفت و برگشت‌های متعدد باشد.

-

:GraphQL

یک نقطه انتهایی^۱ واحد را برای کلاینت‌ها فراهم می‌کند تا دقیقاً داده‌هایی را که نیاز دارند را پرس و جو/کوئری کنند.

-

کلاینت‌ها فیلد‌های دقیق موردنیاز در کوئری‌های تودرتو را مشخص می‌کنند و سرور بارهای مفیدی را بر می‌گرداند که فقط حاوی همان فیلد‌ها هستند.

-

¹ endpoint

- از Mutation برای تغییر داده‌ها و Subscription برای اعلان‌های لحظه‌ای و بلاذرنگ پشتیبانی می‌کند.
- برای جمع‌آوری داده از منابع متعدد عالی است و با نیازهای frontend در حال تغییر سریع به خوبی کار می‌کند.
- با این حال، پیچیدگی را به سمت کلاینت منتقل می‌کند و در صورت عدم محافظت مناسب، می‌تواند به کوئری‌های مخرب اجازه اجرا دهد.
- استراتژی‌های Caching می‌توانند پیچیده‌تر از REST باشند.

بهترین انتخاب بین REST و GraphQL به نیازهای خاص برنامه و تیم توسعه بستگی دارد. GraphQL برای نیازهای پیچیده یا در حال تغییر frontend مناسب است، در حالی که REST برای برنامه‌هایی که فرادردادهای ساده و سازگار و یکپارچه را ترجیح می‌دهند، مناسب است.

هیچ رویکرد API یک راه حل قطعی نیست. ارزیابی دقیق الزامات و مبادلات برای انتخاب سبک مناسب مهم است. هر دو REST و GraphQL گزینه‌های معتبر برای نمایش داده‌ها و قدرت دادن به اپلیکیشن‌های مدرن هستند.

مقایسه بین RESTful و RPC

RPC vs. RESTful



ByteByteGo.com

	RPC	RESTful
Coupling	Strong coupling	Weak coupling
Data format	Binary thrift, protobuf, Avro	Text XML, JSON
Communication protocol	TCP	HTTP/1.1, HTTP/2
Performance	High	Lower than RPC
Interface definition language (IDL)	thrift, protobuf	Swagger
Client code generation	Auto-generated stub	Auto-generated stub
Language framework	gRPC, thrift	SpringMVC, JAX-RS
Developer friendliness	not human readable hard to debug	human readable easy to debug

ارتباط بین سیستم‌های نرم افزاری مختلف می‌تواند با استفاده از پروتکل‌های RPC^۱ یا RESTful^۲ برقرار شود که به سیستم‌های مختلف اجازه می‌دهد در محاسبات توزیع شده با هم کار کنند. این دو پروتکل عمدتاً از نظر فلسفه طراحی با هم تفاوت دارند. RPC امکان فراخوانی رویه‌های از راه دور^۳ روی یک سرور را به عنوان اگر آن‌ها رویه‌های محلی^۴ بودند را فراهم می‌کند، در حالی که برنامه‌های RESTful مبتنی بر منابع هستند و با استفاده از متدهای HTTP با این منابع تعامل می‌کنند. هنگام انتخاب بین RPC و RESTful نیازهای برنامه خود را در نظر بگیرید. RPC ممکن است برای یک رویکرد بیشتر عمل گرای اعمالیات سفارشی مناسب‌تر باشد، در حالی که RESTful برای یک رویکرد مبتنی بر منابع استاندارد که از متدهای HTTP استفاده می‌کند، انتخاب بهتری خواهد بود.

Remote Procedure Call^۱

Representational State Transfer^۲

remote procedures^۳

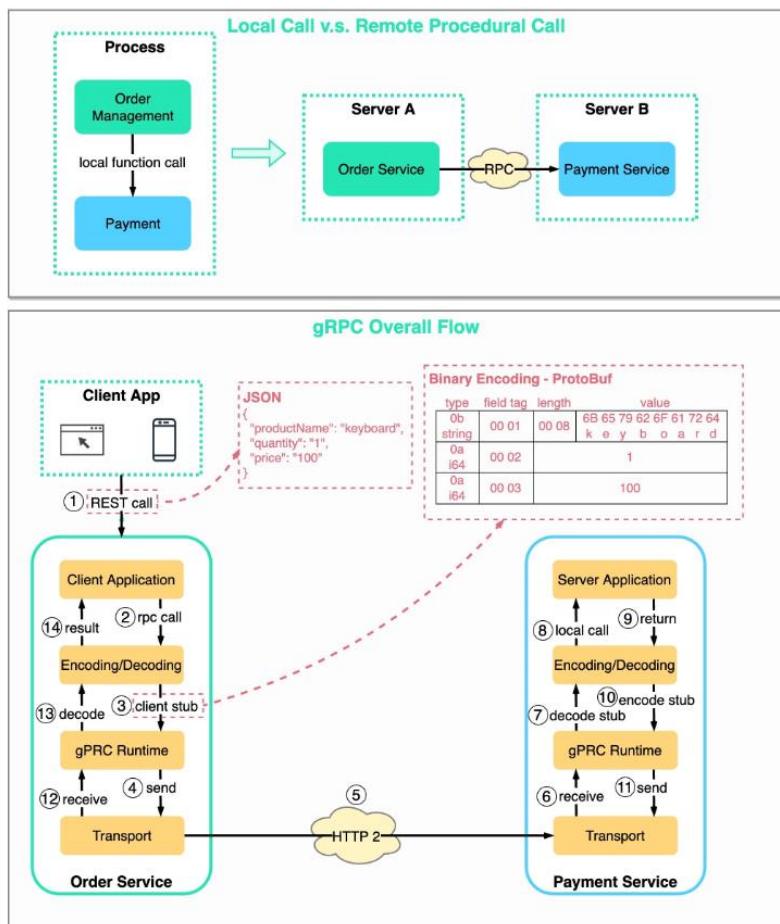
local procedures^۴

نحوه عملکرد gRPC

gRPC مخفف "Remote Procedure Call" (فراخوان رویه راه دور) است. به این دلیل «remote» نامیده می‌شود که امکان برقراری ارتباط بین سرویس‌های از راه دور را فراهم می‌کند، زمانی که سرویس‌ها در یک معماری میکروسرویس در سرورهای مختلف مستقر می‌شوند. از نظر کاربر، مانند یک فراخوانی تابع محلی عمل می‌کند. شکل زیر جریان کلی داده را برای gRPC نشان می‌دهد.

How does gRPC Work?

 blog.bytebytogo.com



مرحله ۱: یک تماس REST از کلاینت برقرار می شود. بدنه درخواست معمولاً به صورت JSON است.

مراحل ۲ تا ۴: سرویس سفارش (کلاینت gRPC) تماس REST را دریافت می کند، آن را تغییر می دهد و یک تماس RPC به سرویس پرداخت برقرار می کند. gRPC استاب کلاینت را به فرمت باینری کدگذاری می کند و آن را به لایه حمل و نقل سطح پایین ارسال می کند.

مراحل ۵: gRPC بسته ها را از طریق شبکه از طریق HTTP2 ارسال می کند. به دلیل کدگذاری باینری و بهینه سازی شبکه، گفته می شود gRPC حدود ۵ برابر سریع تر از JSON است.

مراحل ۶ تا ۸: سرویس پرداخت (سرور gRPC) بسته ها را از شبکه دریافت می کند، آنها را رمزگشایی می کند و برنامه سرور را فراخوانی می کند.

مراحل ۹ تا ۱۱: نتیجه از برنامه سرور برگردانده می شود و کدگذاری شده و به لایه حمل و نقل ارسال می شود.

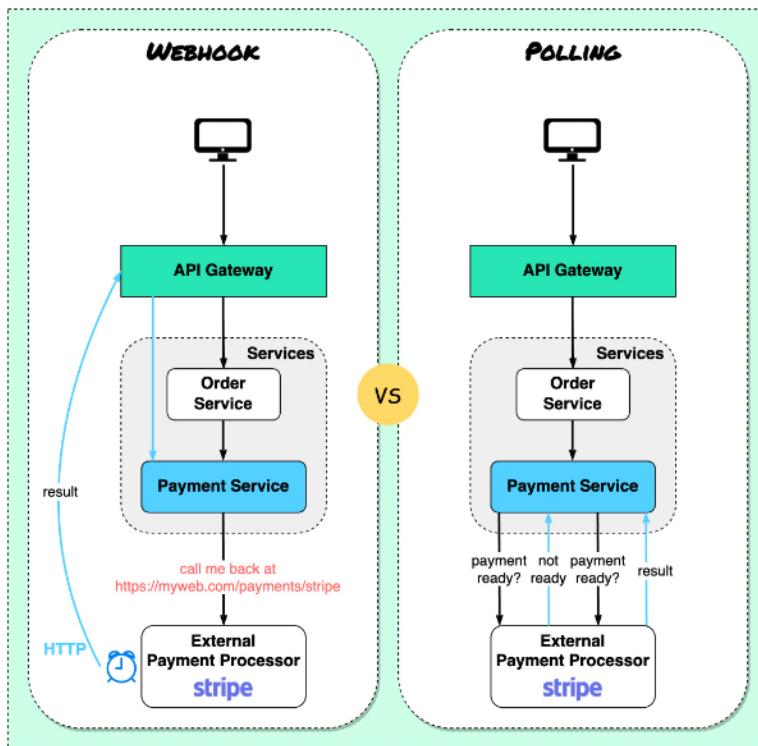
مراحل ۱۲ تا ۱۴: سرویس سفارش بسته ها را دریافت می کند، آنها را رمزگشایی می کند و نتیجه را به برنامه کلاینت ارسال می کند.

چیست؟ Webhook

شکل زیر مقایسه‌ای بین Polling و Webhook را نشان می‌دهد.

What is a Webhook?

 blog.bytebytogo.com



فرض کنید یک وبسایت تجارت الکترونیک را اجرا می‌کنیم. کاربرها از طریق API gateway سفارش‌هایی را به سرویس سفارش ارسال می‌کنند که برای انجام تراکنش‌های پرداخت به سرویس پرداخت می‌رود. سپس سرویس پرداخت برای تکمیل تراکنش‌ها با یک ارائه‌دهنده خدمات پرداخت خارجی (PSP^۱) صحبت می‌کند.

دو راه برای مدیریت ارتباطات با PSP خارجی وجود دارد:

^۱ payment service provider

Short Polling .۱

پس از ارسال درخواست پرداخت به PSP، سرویس پرداخت همچنان از PSP در مورد وضعیت پرداخت سؤال می‌کند. پس از چند دور، PSP سرانجام با وضعیت پرداخت را بازمی‌گرداند.

دو اشکال دارد:

- Polling مدام وضعیت، منابع سرویس پرداخت را مصرف می‌کند.
- سرویس خارجی مستقیماً با سرویس پرداخت ارتباط برقرار می‌کند که باعث ایجاد آسیب‌پذیری‌های امنیتی می‌شود.

Webhook .۲

می‌توانیم یک Webhook در سرویس خارجی ثبت کنیم. این کار به این معنی است: «هنگامی که در مورد درخواستی تقاضای بهروزرسانی دارید، با یک URL خاص تماس بگیرید» وقتی PSP پردازش را تکمیل کرد، درخواست HTTP را برای بهروزرسانی وضعیت پرداخت، فراخوانی می‌کند.

به این ترتیب، الگوی برنامه‌نویسی تغییر می‌کند و سرویس پرداخت دیگر نیازی به مصرف منابع برای بررسی وضعیت پرداخت ندارد. اگر PSP هرگز تماسی برقرار نکرد چه؟ ما می‌توانیم یک کار زمانبندی شده^۱ برای بررسی وضعیت پرداخت هر ساعت تنظیم کنیم. Webhook‌ها اغلب به عنوان API‌های معکوس^۲ یا Push API شناخته می‌شوند؛ زیرا سرور درخواست‌های HTTP را به کلاینت ارسال می‌کند. هنگام استفاده از Webhook، باید به سه نکته توجه کنیم:

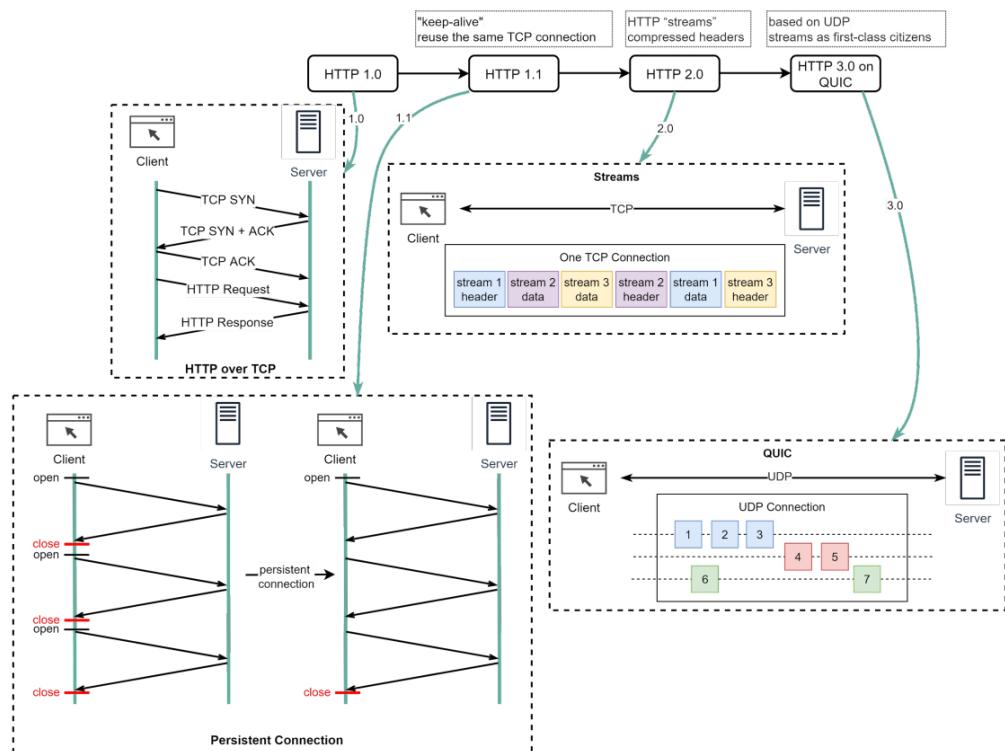
۱. ما باید یک API مناسب برای فراخوانی توسط سرویس خارجی طراحی کنیم.
۲. به دلایل امنیتی باید قوانین مناسب را در API gateway تنظیم کنیم.
۳. باید URL صحیح را در سرویس خارجی ثبت کنیم.

^۱ housekeeping job

^۲ Reverse API

HTTP 1.0 -> HTTP 1.1 -> HTTP 2.0 -> HTTP 3.0 (QUIC)

هر نسل از HTTP چه مشکلی را حل می‌کند؟
نمودار زیر ویژگی‌های کلیدی را نشان می‌دهد.



- HTTP 1.0 در سال ۱۹۹۶ نهایی و کاملاً مستند شد. هر درخواست به همان سرور مورد نظر نیاز به یک اتصال TCP جداگانه دارد.
- HTTP 1.1 در سال ۱۹۹۷ منتشر شد. اتصال TCP می‌تواند برای استفاده مجدد باز نگه داشته شود (اتصال پایدار)، اما مشکل (head-of-line) HOL را حل نمی‌کند.

HOL blocking – هنگامی که تعداد درخواست‌های موازی مجاز در مرورگر تمام شود، درخواست‌های بعدی باید تا زمان اتمام قبلي‌ها صبر کنند.

- HTTP 2.0 در سال ۲۰۱۵ منتشر شد. این نسخه مشکل HOL را از طریق multiplexing درخواست حل می‌کند که مسدود شدن HOL در لایه برنامه^۱ را از بین می‌برد، اما HOL همچنان در لایه حمل و نقل^۲ (TCP) وجود دارد. همانطور که در نمودار می‌بینید، HTTP 2.0 مفهوم «جريان‌های^۳» HTTP را معرفی کرد: انتزاعی که به HTTP کردن transport مختلف در همان اتصال TCP اجازه می‌دهد. هر جريان نیاز ندارد که به ترتیب ارسال شود.

- HTTP 3.0 در سال ۲۰۲۰ برای اولین بار منتشر شد. این جانشین پیشنهادی است. از QUIC به جای TCP برای پروتکل حمل و نقل^۴ زیربنایی استفاده می‌کند، بنابراین مسدود شدن HOL در لایه transport را برطرف می‌کند. transport QUIC بر پایه UDP است. به عنوان شهر وندان درجه اول^۵ در لایه

^۱ application layer

^۲ transport

^۳ streams

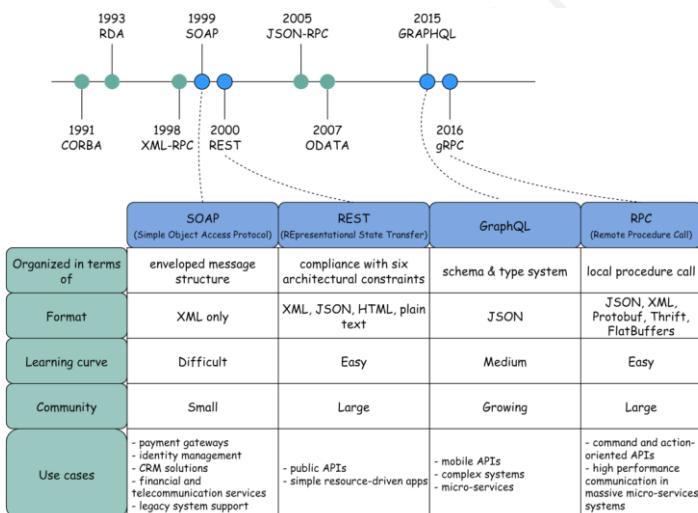
^۴ transport protocol

^۵ first-class citizens

جريان‌ها را معرفی می‌کند. اتصال‌های QUIC جريان‌ها را به اشتراک می‌گذارند، بنابراین به دستدادن‌های^۱ اضافی و شروع‌های آهسته نیست، اما جريان‌های QUIC به صورت مستقل تحويل داده می‌شوند به طوری که در بیشتر موارد از دست رفتن بسته‌ها که یک جريان را تحت تأثیر قرار می‌دهد، بر سایر موارد تأثیر نمی‌گذارد.

مقایسه REST و SOAP و GraphQL

نمودار زیر خط زمانی API و مقایسه سبک‌های API را نشان می‌دهد.



با گذشت زمان، سبک‌های معماری API مختلفی منتشر شده است. هر کدام از آنها الگوهای خاص خود را برای استانداردسازی تبادل داده دارند.

می‌توانید موارد استفاده هر سبک را در نمودار بررسی کنید.

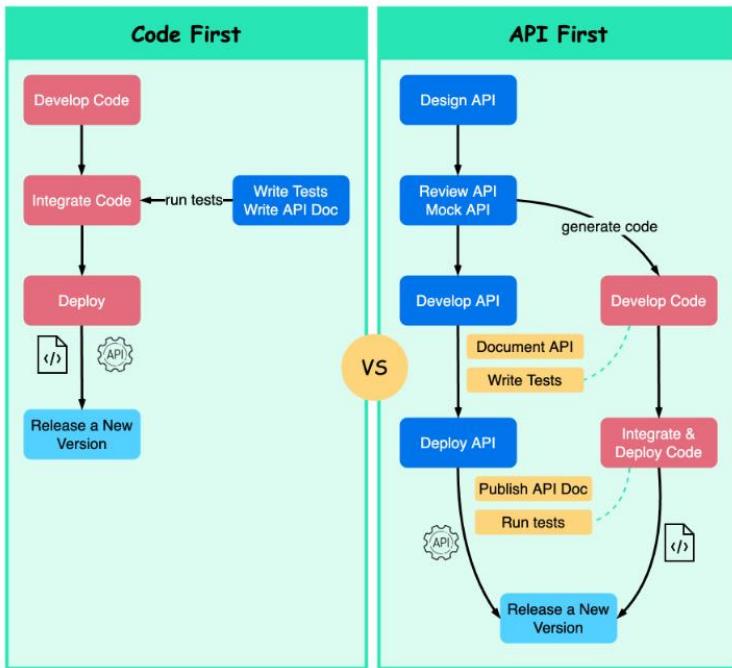
منبع: <https://lnkd.in/gFgi33RY>

^۱ - در هر اتصال رایانه‌ای مقداری بار اضافی وجود دارد که در اصطلاح دست‌دهی (handshaking) نامیده می‌شود و بدین معنی است که مودم از کامپیوتر سرور سؤال می‌کند، آیا داده‌ها را دریافت کرده‌است و سرور پاسخ مثبت یا منفی می‌دهد.

توسعه مبتنی بر کد در مقابل توسعه مبتنی بر API

شکل زیر تفاوت های بین توسعه مبتنی بر کد (Code-First) و توسعه مبتنی بر API (API-First) را نشان می دهد. چرا باید طراحی مبتنی بر API را در نظر بگیریم؟

Code First v.s API First Development  blog.bytebytogo.com



مزایای توسعه مبتنی بر API

کاهش پیچیدگی سیستم: میکروسرویس ها پیچیدگی سیستم را افزایش می دهند و ما سرویس های جداگانه ای برای ارائه توابع مختلف سیستم داریم. در حالی که این نوع معماری، جداسازی و تفکیک وظایف را تسهیل می کند، باید ارتباطات مختلف بین سرویس ها را

مدیریت کنیم. بهتر است قبل از نوشتن کد، به پیچیدگی سیستم فکر کنیم و مرزهای سرویس‌ها را بادقت تعریف کنیم.

زبان مشترک برای تیم‌های توسعه‌ای مجزا؛ تیم‌های عملکردی جداگانه باید به یک زبان صحبت کنند و تیم‌های عملکردی اختصاصی فقط مسئول اجزاء و سرویس‌های خودشان هستند. توصیه می‌شود سازمان از طریق طراحی API به یک زبان مشترک دست یابد.

اعتبارسنجی طراحی API قبل از نوشتمن کد: ما می‌توانیم قبل از نوشتمن کد، در خواست‌ها و پاسخ‌ها را شبیه‌سازی کنیم تا طراحی API را اعتبارسنجی کنیم.

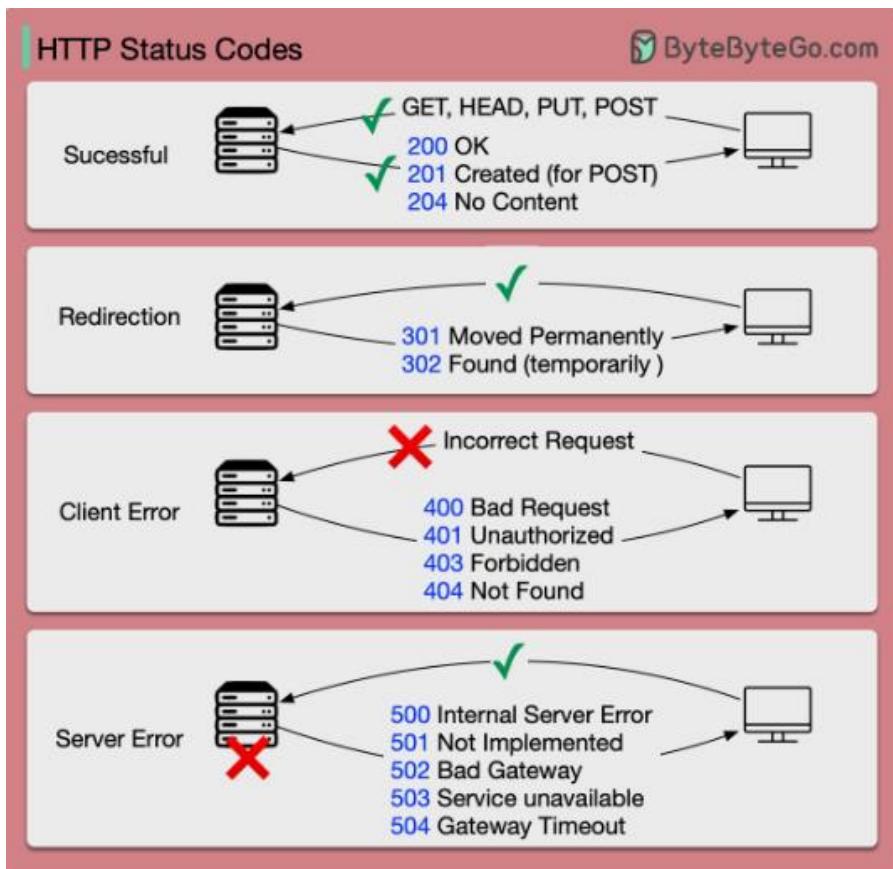
بهبود کیفیت نرم‌افزار و بهره‌وری توسعه‌دهنده: از آنجایی که بیشتر ابهامات را در ابتدای پروژه برطرف کرده‌ایم، کل فرایند توسعه روان‌تر است و کیفیت نرم‌افزار به طور قابل توجهی بهبود می‌یابد. توسعه‌دهنده‌گان نیز از این فرایند راضی هستند زیرا می‌توانند به جای مذاکره در مورد تغییرات ناگهانی، روی توسعه عملکردی تمرکز کنند.

کاهش احتمال مواجه شدن با غافلگیری در اواخر چرخه عمر پروژه: از آنجایی که API را ابتدا طراحی کرده‌ایم، تست‌ها می‌توانند هم‌زمان با توسعه کد طراحی شوند. به‌نوعی، با استفاده از توسعه مبتنی بر API، TDD¹ را نیز داریم.

کدهای وضعیت HTTP

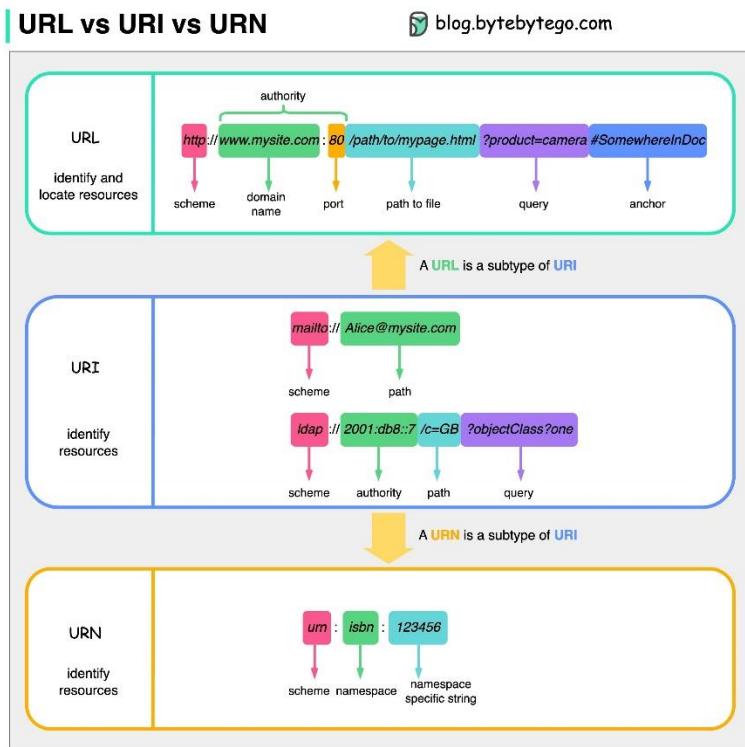
کدهای پاسخ برای HTTP به پنج دسته تقسیم می‌شوند:

- اطلاع‌رسانی (۱۰۰-۱۹۹)
- موفقیت (۲۰۰-۲۹۹)
- هدایت مجدد (۳۰۰-۳۹۹)
- خطای کاربر (۴۰۰-۴۹۹)
- خطای سرور (۵۰۰-۵۹۹)



تفاوت URN، URI، URL

شکل زیر مقایسه‌ای از URL، URI و URN را نشان می‌دهد.



:URI

URI مخفف Uniform Resource Identifier است. این شناسه، یک منبع منطقی یا فیزیکی را در وب شناسایی می‌کند. URL و URN زیرمجموعه‌های URI هستند. URL یک منبع را مکان‌یابی می‌کند، درحالی‌که URN یک منبع را نام‌گذاری می‌کند.

یک URI از بخش‌های زیر تشکیل شده است:

scheme: [//authority] path [?query][#fragment]

:URL

URL مخفف Uniform Resource Locator است و مفهوم کلیدی در HTTP محسوب می‌شود. آدرس یک منبع منحصر به فرد در وب است. این آدرس می‌تواند با پروتکل‌های دیگری مانند FTP و JDBC نیز استفاده شود. هنگامی که شما یک URL را در نوار آدرس مرورگر وب تایپ می‌کنید، در حال دسترسی به یک «منبع» هستید، نه فقط یک صفحه وب تنها.

آدرس‌های اینترنتی (URL) از چندین جزء تشکیل شده‌اند:

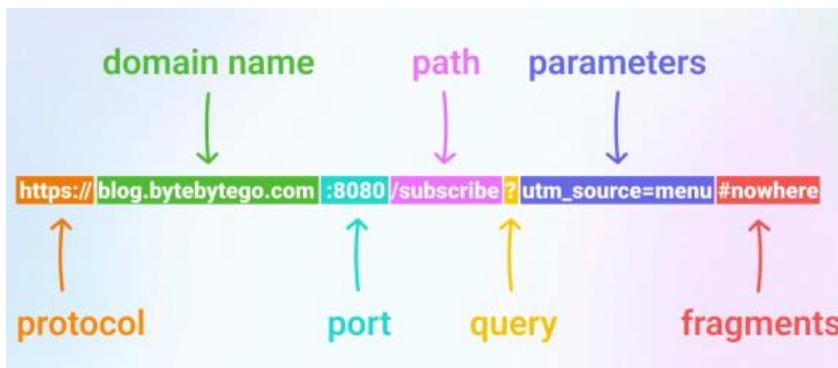
- پروتکل یا scheme، مانند http، https و ftp
- نام دامنه و پورت که با یک نقطه (.) جدا شده‌اند.
- مسیر رسیدن به منبع که با اسلش (/) جدا شده است.
- پارامترها که با علامت سوال (?) شروع می‌شوند و از جفت‌های کلید - مقدار مانند a=b&c=d تشکیل شده‌اند.
- قطعه یا لنگر که با علامت هشتگ (#) نشان داده می‌شود و برای نشانه‌گذاری یک بخش خاص از منبع استفاده می‌شود.

:URN

URN مخفف Uniform Resource Name است و از scheme urn استفاده می‌کند. URN‌ها برای مکان‌یابی یک منبع قابل استفاده نیستند. یک مثال ساده که در شکل آمده است، از یک فضای نام (namespace) و یک رشته خاص مرتبط با آن فضا نام تشکیل شده است. اگر می‌خواهید جزئیات بیشتری در مورد این موضوع بدانید، توصیه می‌کنم توضیحات [W3C](#) را مطالعه کنید.

اجزای مهم یک URL

Uniform Resource Locator (URL) یک اصطلاح آشنا برای اکثر افراد است، زیرا برای پیدا کردن منابع در اینترنت استفاده می‌شود. وقتی یک URL را در نوار آدرس مرورگر تایپ می‌کنید، به یک "resource" دسترسی پیدا می‌کنید، نه فقط یک صفحه وب.

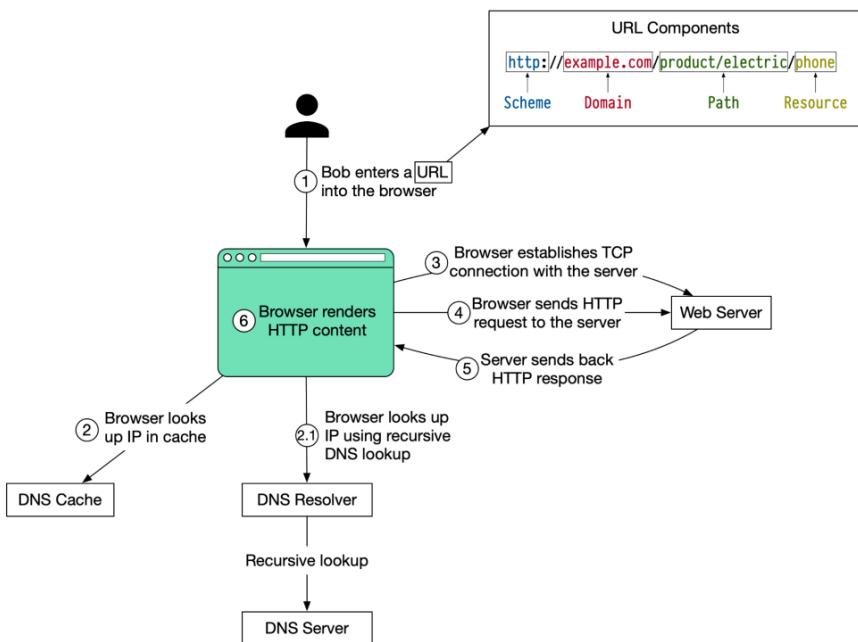


URL‌ها شامل چندین جزء هستند:

- پروتکل یا scheme، مانند http و https و .ftp
- نام دامنه و پورت، با یک نقطه (.) جدا شده‌اند.
- مسیر به resource، با یک اسلش (/) جدا شده است.
- پارامترها، که با یک علامت سوال (?) شروع می‌شوند و شامل جفت‌های کلید-مقدار هستند، مانند a=b&c=d
- قطعه یا anchor، نشان داده شده با یک علامت پوند (#)، که برای بوکمارک کردن یک بخش خاص از resource استفاده می‌شود.

وقتی یک URL را در مرورگر خود تایپ می‌کنید چه اتفاقی می‌افتد؟

- آقای Bob یک URL را در مرورگر وارد می‌کند و Enter را می‌زند. در این مثال، URL از ۴ قسمت تشکیل شده است:



- طرح^۱ - **https://**. این به مرورگر می‌گوید که با استفاده از HTTPS به سرور وصل شود.
- دامنه^۲ - **example.com**. این نام دامنه سایت است.

scheme^۱
domain^۲

- مسیر^۱ - **product/electric**. این مسیر روی سرور به منع درخواست شده است: تلفن.
 - منع^۲ - **phone**. این نام منبعی است که باب می‌خواهد بازدید کند.
۲. مرورگر با استفاده از جستجوی سیستم نام دامنه (DNS) آدرس IP را برای دامنه جستجو می‌کند. برای اینکه فرایند جستجو سریع شود، داده‌ها در لایه‌های مختلف کش می‌شوند: حافظه کش مرورگر، حافظه کش سیستم‌عامل، حافظه کش شبکه محلی و حافظه کش ISP
۳. اگر آدرس IP در هیچ یک از حافظه‌های کش یافت نشد، مرورگر به سرورهای DNS می‌رود تا یک جستجوی بازگشتی DNS انجام دهد تا زمانی که آدرس IP پیدا شود.
۴. حالا که آدرس IP سرور را داریم، مرورگر یک اتصال TCP با سرور برقرار می‌کند.
۵. مرورگر یک درخواست HTTP به سرور ارسال می‌کند. درخواست به این صورت است:
- ```
GET /phone HTTP/1.1
Host: example.com
```
۶. سرور درخواست را پردازش می‌کند و پاسخ را ارسال می‌کند. برای یک پاسخ موفق (کد وضعیت ۲۰۰ است). پاسخ HTML ممکن است به این صورت باشد:

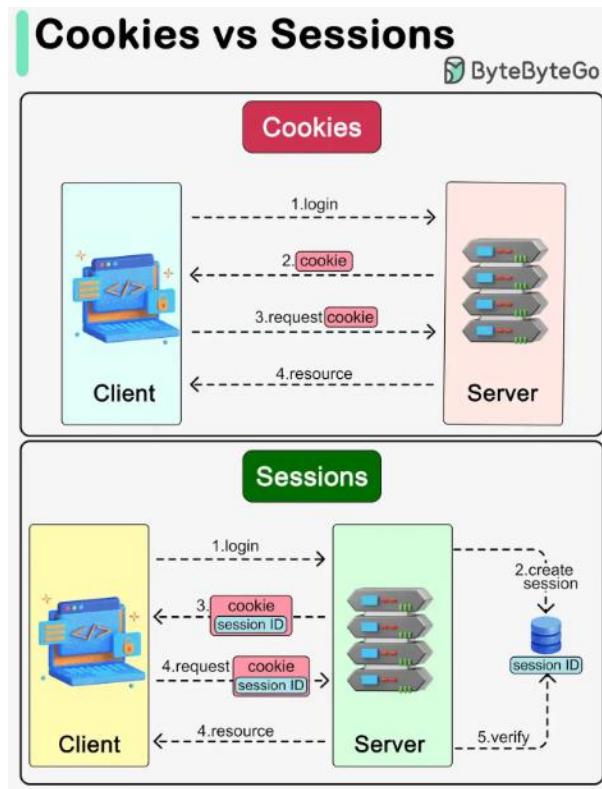
path<sup>۱</sup>  
resource<sup>۲</sup>

```
HTTP/1.1 200 OK
Date: Sun, 30 Jan 2022 00:01:01 GMT
Server: Apache
Content-Type: text/html; charset=utf-8
<!DOCTYPE html>
<html lang="en">
 <head>
 <title>Hello world</title>
 </head>
 <body>
 <h1>Hello world</h1>
 </body>
</html>
```

۵. حالا مرورگر محتوای HTML را به شیوه مناسبی نمایش می‌دهد.

## تفاوت بین کوکی (Cookie) و سشن (Session)

کوکی ها و سشن ها هر دو برای حمل اطلاعات کاربر بر روی درخواست های HTTP استفاده می شوند، این اطلاعات شامل وضعیت ورود به سیستم کاربر، سطح دسترسی کاربر و غیره هستند.



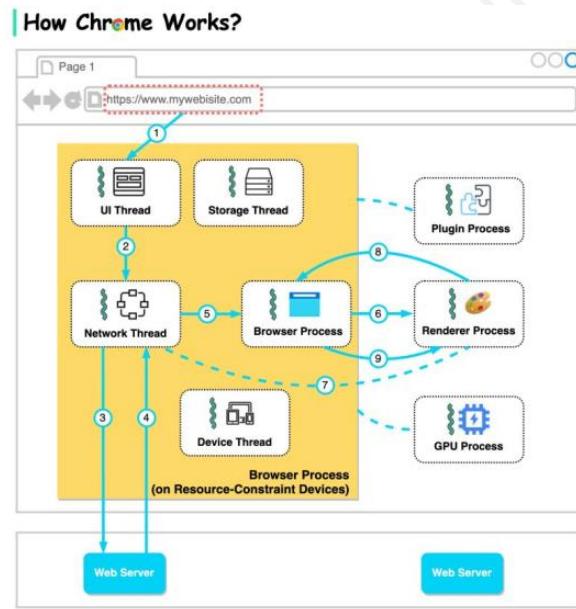
### کوکی (Cookie)

کوکی ها معمولاً دارای محدودیت اندازه (معمولاً 4 کیلوبایت) هستند. آنها قطعات کوچکی از اطلاعات را حمل می کنند و بر روی دستگاه های کاربر ذخیره می شوند. کوکی ها با هر درخواست بعدی کاربر ارسال می شوند. کاربران می توانند انتخاب کنند که کوکی ها را در مرورگر خود مسدود کنند.

## سِشن (Session)

برخلاف کوکی‌ها، سشن‌ها در سمت سرور ایجاد و ذخیره می‌شوند. معمولاً یک شناسه سشن منحصر به فرد در سرور تولید می‌شود که به یک سشن کاربر خاص متصل است. این شناسه سشن با یک کوکی به سمت کاربر برگردانده می‌شود. سشن‌ها می‌توانند حجم بیشتری از داده را نگهداری کنند. از آنجایی که داده‌های سشن توسط کاربر مستقیماً قابل دسترسی نیستند، سشن امنیت بیشتری دارد.

## مروگرهای مدرن چگونه کار می‌کنند؟



نمودار زیر معماری یک مروگر مدرن را نشان می‌دهد. این بر اساس درک ما از «نگاهی درون به مروگر وب مدرن» منتشر شده توسط تیم کروم است. به طور کلی ۴ فرایند وجود دارد: فرایند مروگر، فرایند رندر، فرایند GPU و فرایند افزونه.

- فرایند مرورگر نوار آدرس که بوک‌مارک‌ها، دکمه‌های عقب و جلو و غیره را کنترل می‌کند.
  - فرایند ریندر هر چیزی را داخل تب که یک وب‌سایت نمایش داده می‌شود کنترل می‌کند.
  - فرایند GPU که وظایف GPU را مدیریت می‌کند.
  - فرایند افزونه‌ها: افزونه‌های مورداستفاده توسط وب‌سایت‌ها را کنترل می‌کند.
- فرایند مرورگر با فرایندهای دیگر هماهنگ می‌شود. زمانی که کروم روی سخت‌افزار قدرتمندی اجرا می‌شود، ممکن است هر سرویس در فرایند مرورگر را به رشته‌های مختلف تقسیم کند، همان‌طور که در نمودار زیر نشان‌داده شده است. به این عمل سرویس‌سازی (Servicification) می‌گویند. حالا باید مراحل وارد کردن یک URL در کروم را بررسی کنیم.

مرحله ۱: کاربر یک URL را در مرورگر وارد می‌کند. این کار توسط رشته UI انجام می‌شود.

مرحله ۲: وقتی کاربر کلید enter را می‌زند، UI thread یک تماس شبکه برای دریافت محتوای سایت آغاز می‌کند.

مراحل ۳ و ۴: رشته شبکه از پروتکل‌های شبکه مناسب عبور کرده و محتوا را بازیابی می‌کند.

مرحله ۵: زمانی که network thread پاسخ‌ها را دریافت می‌کند، چند بایت اول جریان را بررسی می‌کند. اگر یک فایل HTML باشد، توسط فرایند مرورگر به فرایند رندر ارسال می‌شود.

مراحل ۶ تا ۹: یک IPC از فرایند مرورگر به فرایند رندر ارسال می‌شود تا مسیردهی را انجام دهد. یک خط داده بین network thread و renderer process ایجاد می‌شود تا renderer process بتواند داده‌ها را دریافت کند. زمانی که browser process تأیید را از renderer process دریافت کرد که انجام شده است، مسیردهی کامل می‌شود و مرحله بارگذاری document آغاز می‌شود.

لينک‌ها:

[/https://developer.chrome.com/blog/inside-browser-part1](https://developer.chrome.com/blog/inside-browser-part1)

[/https://developer.chrome.com/blog/inside-browser-part2](https://developer.chrome.com/blog/inside-browser-part2)

[/https://developer.chrome.com/blog/inside-browser-part3](https://developer.chrome.com/blog/inside-browser-part3)

[/https://developer.chrome.com/blog/inside-browser-part4](https://developer.chrome.com/blog/inside-browser-part4)

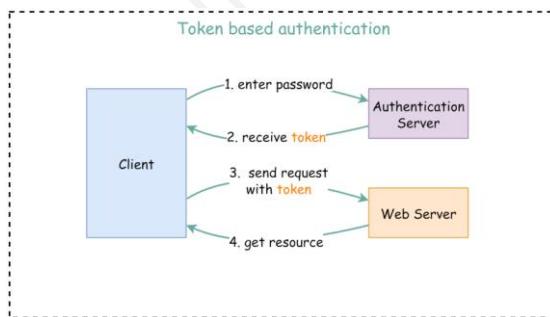
## بررسی روش‌های API

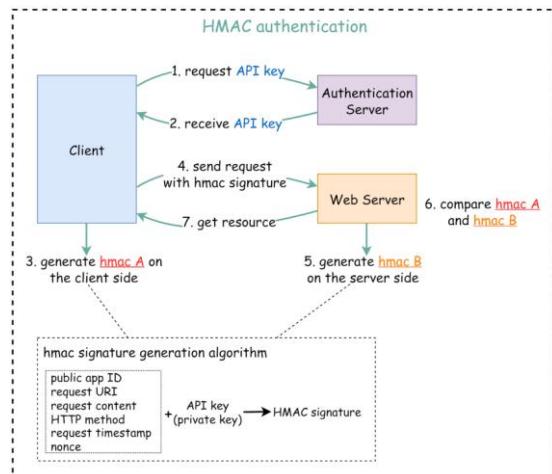
### چگونه دسترسی امن به API وب برای وب‌سایت خود طراحی کنیم؟

زمانی که دسترسی به API وب را برای کاربران باز می‌کنیم، باید اطمینان حاصل کنیم که هر فراخوانی API احراز هویت شده است. این بدان معناست که کاربر باید همان شخصی باشد که ادعا می‌کند. در این پست، ما دو روش رایج را بررسی می‌کنیم:

۱. احراز هویت مبتنی بر توکن
۲. احراز هویت HMAC (کد احراز پیام مبتنی بر هش<sup>۱</sup>)

دیاگرام زیر نشان می‌دهد که این روش‌ها چگونه کار می‌کنند.





## مبتنی بر توکن

مرحله ۱ - کاربر رمز عبور خود را در وارد می‌کند و رمز عبور را به سرور احراز هویت ارسال می‌کند.

مرحله ۲ - سرور احراز هویت، اطلاعات اعتباری را تأیید می‌کند و یک توکن با زمان انقضای تولید می‌کند.

مراحل ۳ و ۴ - حالا کاربر می‌تواند درخواست‌ها را برای دسترسی به منابع سرور به همراه توکنی در هدر HTTP ارسال کند. این دسترسی تا زمان انقضای توکن معتبر است.

## HMAC مبتنی بر

این مکانیزم با استفاده از یک تابع هش (MD5 یا SHA256) یک کد احراز پیام (امضا<sup>۱</sup>) تولید می‌کند.

مراحل ۱ و ۲ - سرور دو کلید تولید می کند، یکی Public APP ID (کلید عمومی<sup>۱</sup>) و دیگری API Key (کلید خصوصی<sup>۲</sup>) است.

مراحله ۳ - حالا ما یک امضای HMAC در سمت کلاینت (hmac A) تولید می کنیم. این امضا با مجموعه ای از ویژگی هایی که در دیاگرام ذکر شده است تولید می شود.

مراحله ۴ - کلاینت درخواست ها را برای دسترسی به منابع سرور با hmac A در هدر HTTP ارسال می کند.

مراحله ۵ - سرور درخواستی را دریافت می کند که شامل داده های درخواست و Header احراز هویت است. سرور ویژگی های لازم را از درخواست استخراج می کند و از کلید API که در سمت سرور ذخیره شده است برای تولید یک امضا (hmac B) استفاده می کند.

مراحله ۶ و ۷ - سرور hmac A (تولید شده در سمت کلاینت) و hmac B (تولید شده در سمت سرور) را مقایسه می کند. اگر مطابقت داشته باشند، منبع درخواستی به کلاینت بازگردانده می شود.

سؤال - چگونه احراز هویت HMAC یکپارچگی داده ها را تضمین می کند؟ چرا ما «درخواست را در تولید امضای HMAC قرار می دهیم»

## چگونه API‌های مؤثر و امن طراحی کنیم؟

شکل زیر نمونه‌های متداول طراحی API را با یک مثال سبد خرید نشان می‌دهد.

### Design Effective & Safe APIs

 blog.bytebytogo.com

Design a Shopping Cart		
Use resource names (nouns)	<span style="color:red;">X</span> GET /querycarts/123	<span style="color:green;">✓</span> GET /carts/123
Use plurals	<span style="color:red;">X</span> GET /cart/123	<span style="color:green;">✓</span> GET /carts/123
Idempotency	<span style="color:red;">X</span> POST /carts	<span style="color:green;">✓</span> POST /carts {requestId: 4321}
Use versioning	<span style="color:red;">X</span> GET /carts/v1/123	<span style="color:green;">✓</span> GET /v1/carts/123
Query after soft deletion	<span style="color:red;">X</span> GET /carts	<span style="color:green;">✓</span> GET /carts? includeDeleted=true
Pagination	<span style="color:red;">X</span> GET /carts	<span style="color:green;">✓</span> GET /carts? pageSize=xx&pageToken=xx
Sorting	<span style="color:red;">X</span> GET /items	<span style="color:green;">✓</span> GET /items? sort_by=time
Filtering	<span style="color:red;">X</span> GET /items	<span style="color:green;">✓</span> GET /items? filter=color:red
Secure Access	<span style="color:red;">X</span> X-API-KEY=xxx	<span style="color:green;">✓</span> X-API-KEY = xxx X-EXPIRY = xxx X-REQUEST-SIGNATURE = xxx <i>hmac(URL + QueryString + Expiry + Body)</i>
Resource cross reference	<span style="color:red;">X</span> GET /carts/123? item=321	<span style="color:green;">✓</span> GET /carts/123/items/321
Add an item to a cart	<span style="color:red;">X</span> POST /carts/123? addItem=321	<span style="color:green;">✓</span> POST /carts/123/items/add {itemId: "items/321"}
Rate limit	<span style="color:red;">X</span> No rate limit - DDoS	<span style="color:green;">✓</span> Design rate limiting rules based on <i>IP, user, action group etc</i>

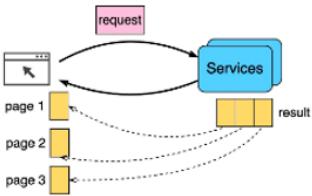
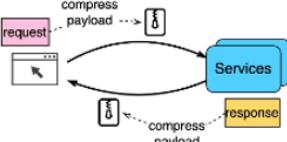
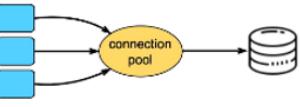
توجه داشته باشید که طراحی API فقط طراحی مسیر URL نیست. در اکثر موقع، نیاز به انتخاب نام منابع، شناسه‌ها و الگوهای مسیر مناسب داریم. طراحی فیلدہای هدر HTTP مناسب یا طراحی قوانین محدودیت نرخ مؤثر<sup>۱</sup> در API gateway بسیار مهم است.

rate-limitin<sup>1</sup>

## بهبود عملکرد API

در اینجا پنج ترفند رایج برای بهبود عملکرد API آورده شده است:

### How to Improve API Performance?

PAGINATION		<ul style="list-style-type: none"><li>شماره گذاری ترتیبی صفحات.</li><li>تعداد زیادی از نتایج را کنترل می‌کند.</li></ul>
ASYNC LOGGING		<ul style="list-style-type: none"><li>لاگ‌های مرزوط را به یک بافر حلقه.</li><li>lock-free ارسال گردد و بروگردانید.</li><li>به صورت دوره‌ای روی دیسک پذیرشی.</li><li>توان عملیاتی بالاتر و تاخیر کمتر.</li></ul>
CACHING		<ul style="list-style-type: none"><li>داده‌های پرکاربرد را به جای پایگاه داده در کش ذخیره کنید.</li><li>هنگامی که حافظه کش وجود ندارد، پایگاه داده را کوتیری کنید.</li></ul>
PAYLOAD COMPRESSION		<ul style="list-style-type: none"><li>پایای افزایش سرعت دانلود و اپلود، حجم داده‌ها را کاهش دهد.</li></ul>
CONNECTION POOL		<ul style="list-style-type: none"><li>باز و سریع شدن اتصالات DB سریار.</li><li>قابل توجه را اضافه نمی‌کند.</li><li>یک connection pool تعدادی اتصال باز را برای بزرگ‌نمایی استفاده مجدد حفظ می‌کند.</li></ul>

Reference: Rapid API

### صفحه‌بندی (Pagination)

این یک بهینه‌سازی رایج است که زمانی که حجم نتیجه زیاد باشد، مورد استفاده قرار می‌گیرد. نتایج به صورت جریان به کلاینت بر می‌گردند تا پاسخگویی سرویس بهبود یابد.

### Asynchronous Logging

Log هم زمان برای هر فرآخوانی با دیسک سروکار دارد و می‌تواند سرعت سیستم را آهسته کند. Log نام زمان ابتدا لگ‌ها را به یک بافر بدون قفل ارسال می‌کند و بلافاصله بر می‌گردد. لگ‌ها به صورت دوره‌ای روی دیسک نوشته می‌شوند. این کار سریار I/O را به میزان قابل توجهی کاهش می‌دهد.

### Caching

ما می‌توانیم داده‌هایی که به طور مکرر به آن‌ها دسترسی پیدا می‌شود را در یک کش ذخیره کنیم. کلاینت می‌تواند قبل از مراجعه مستقیم به پایگاهداده، ابتدا کش را بررسی کند. در صورت عدم وجود داده در کش (Cache Miss)، کلاینت می‌تواند از پایگاهداده کوئری کند. کش‌هایی مانند Redis داده‌ها را در حافظه ذخیره می‌کنند، بنابراین دسترسی به داده‌ها بسیار سریع‌تر از پایگاهداده است.

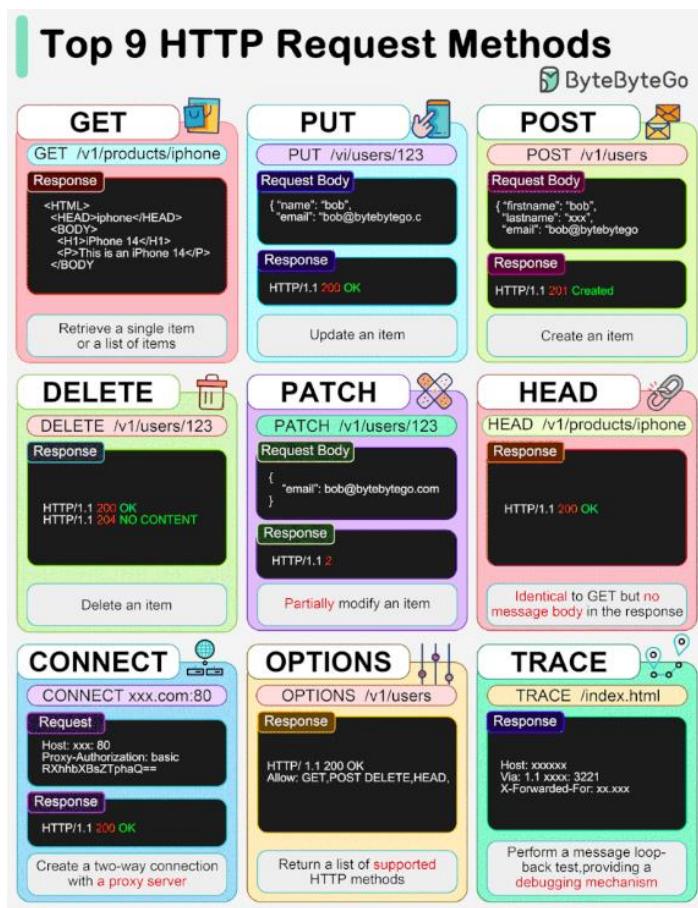
### Payload فشرده‌سازی

در خواست‌ها و پاسخ‌ها را می‌توان با استفاده از gzip و غیره فشرده کرد تا اندازه داده‌های ارسالی بسیار کمتر شود. این کار باعث سرعت بخشیدن به آپلود و دانلود می‌شود.

### Connection Pool

هنگام دسترسی به منابع، اغلب نیاز به بارگذاری داده‌ها از پایگاهداده داریم. باز و بسته کردن اتصالات پایگاهداده سریار قابل توجهی ایجاد می‌کند؛ بنابراین، باید از طریق یک connection pool به پایگاهداده متصل شویم. connection pool مسئول مدیریت چرخه عمر اتصالات است.

## HTTP ... مفاهیم رایج PUT . POST . GET



### ۱. HTTP GET

این روش یک منبع را از سرور بازیابی می‌کند و یک فرایند idempotent است، به این معنی که درخواست‌های یکسان متعدد، نتیجه‌ی یکسانی را بر می‌گرداند.

### ۲. HTTP PUT

این روش یک منبع را بروزرسانی یا ایجاد می‌کند و یک فرایند idempotent است، یعنی درخواست‌های یکسان متعدد، یک منبع را به روزرسانی می‌کنند.

### ۳. HTTP POST

این روش برای ایجاد منابع جدید استفاده می‌شود و یک فرایند idempotent نیست، انجام دو درخواست POST یکسان، باعث ایجاد تکراری از منبع می‌شود.

#### ۴. HTTP DELETE

این روش برای حذف یک منبع استفاده می‌شود و یک فرایند idempotent است، یعنی درخواست‌های یکسان متعدد، همان منبع را حذف می‌کنند.

#### ۵. HTTP PATCH

این روش تغییرات جزئی را روی یک منبع اعمال می‌کند.

#### ۶. HTTP HEAD

این روش یک پاسخ مشابه با درخواست GET را درخواست می‌کند، اما بدون بدنی پاسخ (داده‌های ارسالی از سرور).

#### ۷. HTTP CONNECT

این روش یک تونل (ارتباط ویژه) به سمت سروری که توسط منبع هدف شناسایی شده، ایجاد می‌کند.

#### ۸. HTTP OPTIONS

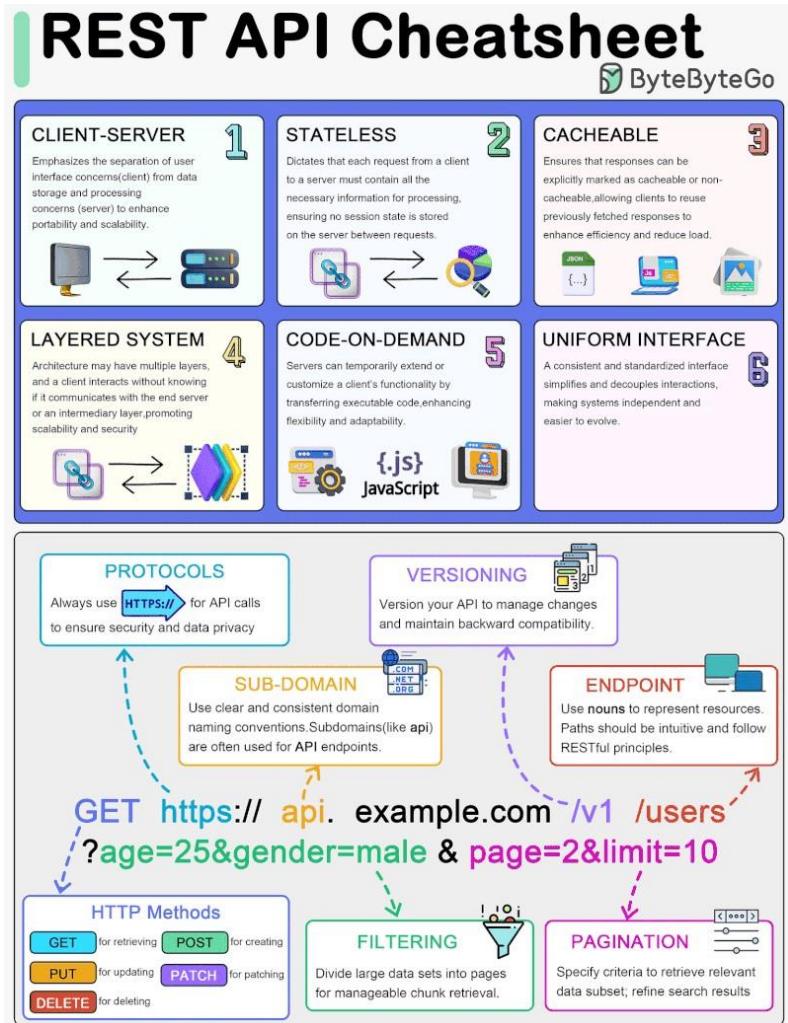
این روش گرینه‌های ارتباطی برای منبع هدف را توصیف می‌کند.

#### ۹. HTTP TRACE

این روش یک تست حلقه‌ی پیام را در طول مسیر به سمت منبع هدف انجام می‌دهد.

## نقاب نامه‌ی REST API

این راهنمای کمک به شما در درک دنیای API‌های RESTful به روشی واضح و جذاب طراحی شده است.



محتوای این راهنمای:

- بررسی شش اصل اساسی طراحی REST API

- درک عمیق اجزای کلیدی مانند متدهای HTTP، پروتکل‌ها، نسخه‌بندی و موارد دیگر
- تمرکز ویژه بر جنبه‌های کاربردی مانند صفحه‌بندی، فیلتر کردن و طراحی Endpoint

## اصول کلی REST API

**Client-Server**: بر جداسازی دغدغه‌های رابط کاربری (کلاینت) از ذخیره‌سازی و پردازش داده‌ها (서버) برای بهبود قابلیت حمل و مقیاس‌پذیری تأکید می‌کند.

**Stateless**: حکم می‌کند که هر درخواست از یک کلاینت به سرور باید حاوی تمام اطلاعات لازم برای پردازش باشد و سرور نباید روی خود، داده‌ای را برای درخواست‌های بعدی نگه‌داری کند.

**Cacheable**: این امکان را برای کلاینت‌ها فراهم می‌کند تا پاسخ‌هایی که صراحتاً به عنوان قابل کش شدن<sup>۱</sup> یا غیر قابل کش شدن<sup>۲</sup> علامت‌گذاری شده‌اند را دوباره استفاده کنند تا بازده را افزایش و بارگذاری روی سرور را کاهش دهند.

**سیستم چندلایه**: معماری ممکن است چندلایه باشد و یک کلاینت بدون اینکه بداند با سرور نهایی یا یک لایه واسط ارتباط برقرار می‌کند، با آن تعامل می‌کند که این امر باعث افزایش مقیاس‌پذیری و امنیت می‌شود.

**Code-on-Demand**: سرورها می‌توانند با انتقال کد قابل اجرا، قابلیت‌های یک کلاینت را به طور موقت گسترش یا سفارشی کنند و بدین ترتیب انعطاف‌پذیری و تطبیق‌پذیری را افزایش دهند.

**Uniform Interface**: یک رابط سازگار و استاندارد باعث ساده‌سازی و مجزا کردن تعاملات می‌شود و سیستم‌ها را مستقل و تکامل‌پذیرتر می‌کند.

cacheable<sup>۱</sup>

non-cacheable<sup>۲</sup>

**پروتکل ها:** همیشه برای تماس های API از HTTPS:// استفاده کنید تا امنیت و محروم انگی داده ها را تضمین کنید.

**API های Versioning:** خود را نسخه بندی کنید تا تغییرات را مدیریت کرده و سازگاری با نسخه های قبلی را حفظ کنید.

**Subdomain:** از قراردادهای نام گذاری دامنه ای واضح و سازگار استفاده کنید. برای نقاط انتهایی API، اغلب از زیرمجموعه ها (مانند api) استفاده می شود.

**endpoint:** از اسم ها برای نشان دادن منابع استفاده کنید. مسیرها باید بصری باشند و از اصول RESTful پیروی کنند.

**مثال:**

GET [invalid URL removed]  
متد های HTTP

• GET برای بازیابی

• POST برای ایجاد

• PUT برای به روز رسانی

• PATCH برای وصله کردن

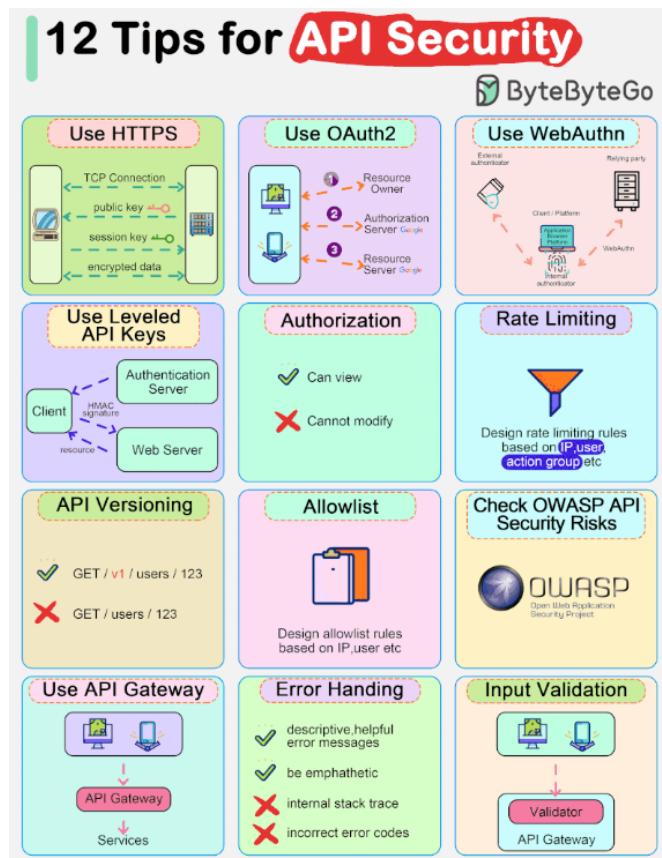
• DELETE برای حذف

**Filtering:** مجموعه داده های بزرگ را برای بازیابی بخش های قابل مدیریت به صفحات تقسیم کنید.

**pagination:** برای بازیابی زیرمجموعه هی مرتب طی از داده ها، معیار را مشخص کنید؛ نتایج جستجو را اصلاح کنید.

## ۱۲ نکته برتر برای امنیت API

در این مقاله، ۱۲ نکته کلیدی برای تقویت امنیت API شما ارائه شده است.

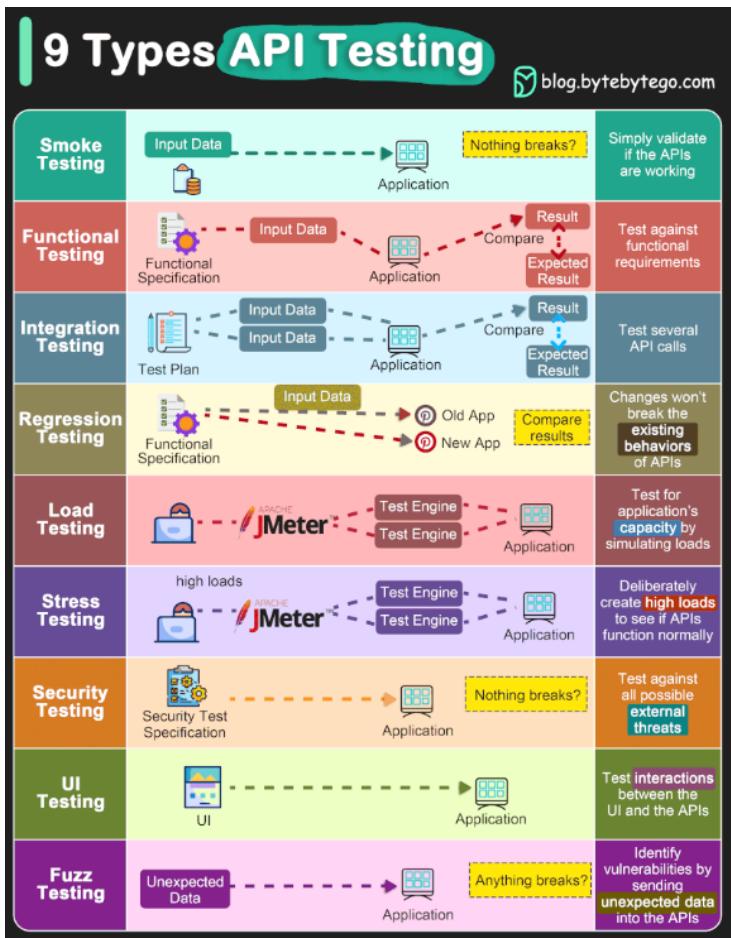


- **HTTPS**: امنیت ارتباطات را با رمزگاری داده‌ها تأمین می‌کند و از شنود و دستکاری اطلاعات جلوگیری می‌کند.
- **OAuth2**: یک پروتکل استاندارد برای مجوزدهی است که به اپلیکیشن اجازه می‌دهد بدون نیاز به ذخیره گذر واژه‌های کاربر، به منابع سرور دسترسی پیدا کنند.
- **WebAuthn**: استانداردی جدید برای احراز هویت بدون رمز عبور است که امنیت بالاتری را نسبت به روش‌های سنتی مبتنی بر رمز عبور ارائه می‌دهد.

- **کلیدهای API با سطوح دسترسی:** به برنامه‌های مختلف، کلیدهای API با سطح دسترسی مناسب با نیازهایشان اختصاص دهید.
- **مجوزدهی:** دسترسی به داده‌ها و عملکردهای API را فقط برای کاربران و برنامه‌های مجاز محدود کنید.
- **محدودیت نرخ:** تعداد درخواست‌هایی را که یک کاربر یا برنامه می‌تواند در یک دوره زمانی خاص ارسال کند، محدود کنید تا از حملات DoS (حملات منع سرویس) جلوگیری شود.
- **نسخه‌بندی API:** نسخه‌های مختلفی از API خود را ارائه دهید تا سازگاری با برنامه‌های موجود را در جین معرفی تغییرات جدید حفظ کنید.
- **فهرست مجاز:** فقط به IP‌های خاص یا دامنه‌های مجاز اجازه دهید به API شما دسترسی داشته باشند.
- **بررسی ریسک‌های امنیتی API توسط OWASP:** از لیست ریسک‌های امنیتی API که توسط سازمان OWASP<sup>۱</sup> تهیه شده است، برای شناسایی و رفع آسیب‌پذیری‌های احتمالی در API خود استفاده کنید.
- **API Gateway:** یک لایه میانی ایجاد کنید که ترافیک API را مدیریت کرده و امنیت را تقویت نماید.
- **مدیریت خطأ:** خطاهای را به صورت ایمن و با جزئیات مناسب مدیریت کنید تا از افشای اطلاعات حساس جلوگیری شود.
- **اعتبارسنجی ورودی:** داده‌های ورودی از کاربران و اپلیکیشن را قبل از پردازش در API به طور کامل اعتبارسنجی کنید تا از حملات تزریق کد<sup>۲</sup> و سایر آسیب‌پذیری‌ها جلوگیری شود.

## ۹ نوع تست API

تست API فرایندی است که برای اطمینان از عملکرد صحیح و ایمن API‌ها انجام می‌شود. در این بخش، ۹ نوع رایج از تست API را بررسی می‌کنیم:

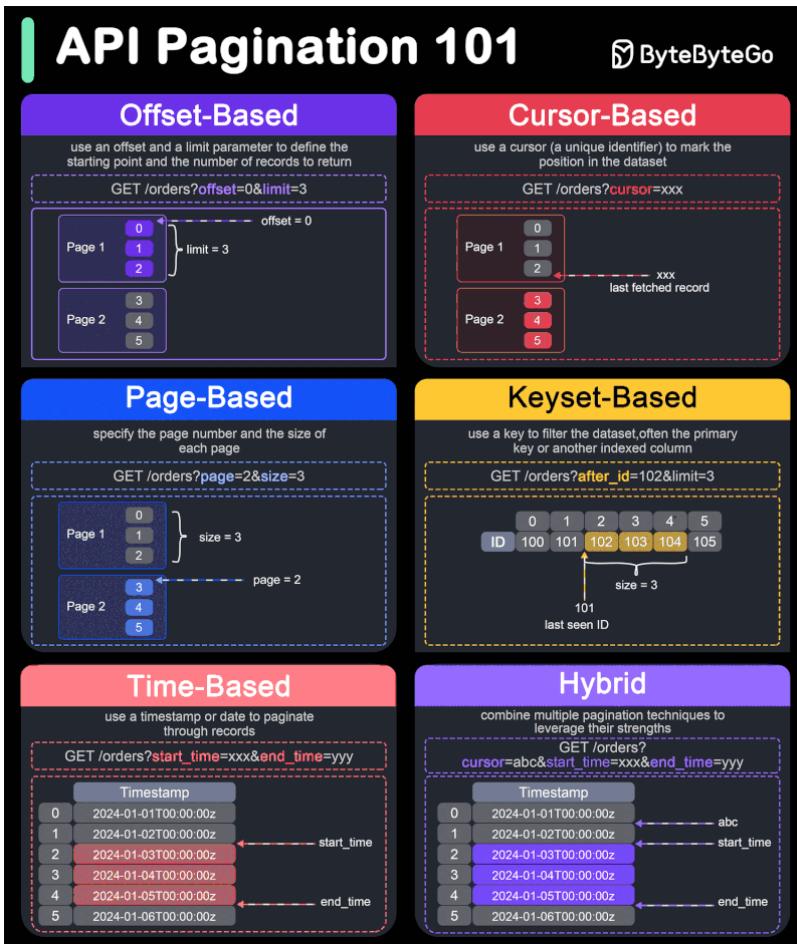


۱. تست دود (Smoke Testing): این تست پس از تکمیل توسعه API انجام می‌شود. در این تست بررسی می‌شود که آیا API‌ها کار می‌کنند و مشکلی وجود ندارد.
۲. تست عملکردی (Functional Testing): این تست بر اساس الزامات عملکردی، یک برنامه تست ایجاد می‌کند و نتایج را با نتایج موردنظر مقایسه می‌کند.

۳. تست یکپارچه‌سازی (Integration Testing): این تست، چندین فراخوانی API را برای انجام تست‌های سراسری ترکیب می‌کند. در این تست، ارتباطات درون سرویس و انتقال داده‌ها مورد بررسی قرار می‌گیرد.
۴. تست رگرسیون (Regression Testing): این تست اطمینان می‌دهد که رفع اشکال یا ویژگی‌های جدید نباید باعث خرابی رفتارهای موجود در API‌ها شوند.
۵. تست بار (Load Testing): این تست عملکرد برنامه را با شبیه‌سازی بارهای مختلف مورد بررسی قرار می‌دهد. سپس با کمک این تست می‌توان ظرفیت برنامه را محاسبه کرد.
۶. تست استرس (Stress Testing): این تست به‌عمد، بارهای بالایی را روی API‌ها ایجاد می‌کند و بررسی می‌کند که آیا API‌ها قادر به عملکرد عادی هستند یا خیر.
۷. تست امنیتی (Security Testing): این تست، API‌ها را در برابر تمام تهییدات خارجی احتمالی مورد بررسی قرار می‌دهد.
۸. تست رابط کاربری (UI Testing): این تست، تعامل رابط کاربری با API‌ها را آزمایش می‌کند تا اطمینان حاصل شود که داده‌ها به درستی نمایش داده می‌شوند.
۹. تست تزریق داده مخرب (Fuzz Testing): این تست، داده‌های ورودی نامعتبر یا غیرمنتظره را به API تزریق می‌کند و سعی می‌کند API را خراب کند. به‌این‌ترتیب، آسیب‌پذیری‌های API شناسایی می‌شوند.

## چگونه از API در طراحی Pagination استفاده کنیم؟

در طراحی API برای مدیریت کارآمد مجموعه داده‌های بزرگ و بهبود عملکرد پیش‌گفتار مهم است. در اینجا به شش تکنیک محبوب Pagination اشاره می‌کنیم:



### .۱. Offset-based Pagination

این روش از یک آفست ( نقطه شروع ) و یک پارامتر محدودیت ( تعداد رکوردهای قابل بازگشت ) برای تعریف نقطه شروع و تعداد رکوردهایی که باید برگردانده شوند، استفاده می‌کند.

**مثال:** GET /orders?offset=0&limit=3

**مزایا:** پیاده‌سازی و درک آن ساده است.

**معایب:** برای آفست‌های بزرگ، با اسکن و ردشدن از ردیف‌ها، می‌تواند ناکارآمد شود.

### **:Cursor-based Pagination .۲**

این روش از یک Cursor (منحصر به فرد) برای علامت‌گذاری موقعیت در مجموعه داده استفاده می‌کند. به طور معمول، Cursor یک رشته کدگذاری شده است که به یک رکورد خاص اشاره می‌کند.

**مثال:** GET /orders?cursor=xxx

**مزایا:** برای مجموعه داده‌های بزرگ کارآمدتر است، زیرا نیازی به اسکن رکوردهای ردشده ندارد.

**معایب:** پیاده‌سازی و درک آن کمی پیچیده‌تر است.

### **:Page-based Pagination .۳**

این روش شماره صفحه و اندازه هر صفحه را مشخص می‌کند.

**مثال:** GET /items?page=2&size=3

**مزایا:** پیاده‌سازی و استفاده آسان است.

**معایب:** برای شماره صفحات بزرگ، مشکلات عملکردی مشابه با Pagination مبتنی بر آفست را دارد.

### **:Keyset-based Pagination .۴**

این روش از یک کلید برای فیلتر کردن مجموعه داده استفاده می‌کند که اغلب کلید اصلی یا ستون دیگری با ایندکس است.

**مثال:** GET /items?after\_id=102&limit=3

**مزایا:** برای مجموعه داده‌های بزرگ کارآمد است و از مشکلات عملکردی با آفست‌های بزرگ جلوگیری می‌کند.

معایب: نیاز به یک کلید منحصر به فرد و با اندیس دارد و پیاده‌سازی آن می‌تواند پیچیده باشد.

#### **:Time-based Pagination ۵**

این روش از یک زمان‌سنج یا تاریخ برای Pagination روی رکوردها استفاده می‌کند.

**مثال:** GET /items?start\_time=xxx&end\_time=yyy

مزایا: برای مجموعه‌داده‌های مرتب شده بر اساس زمان مفید است، در صورت اضافه شدن رکوردهای جدید، اطمینان می‌دهد که هیچ رکوردی از دست نمی‌رود.

معایب: نیاز به زمان‌سنج و تاریخ قابل اعتماد و سازگار دارد.

#### **:Hybrid Pagination ۶**

این روش چندین تکنیک Pagination را برای استفاده از نقاط قوت آن‌ها ترکیب می‌کند. به عنوان مثال این روش ترکیبی از Pagination مبتنی بر Cursor و time-based است که در رکوردهای مرتب شده بر اساس زمان اسکرول کارآمد است.

**مثال:** GET /items?cursor=abc&start\_time=xxx&end\_time=yyy

مزایا: می‌تواند بهترین عملکرد و انعطاف‌پذیری را برای مجموعه‌داده‌های پیچیده ارائه دهد.

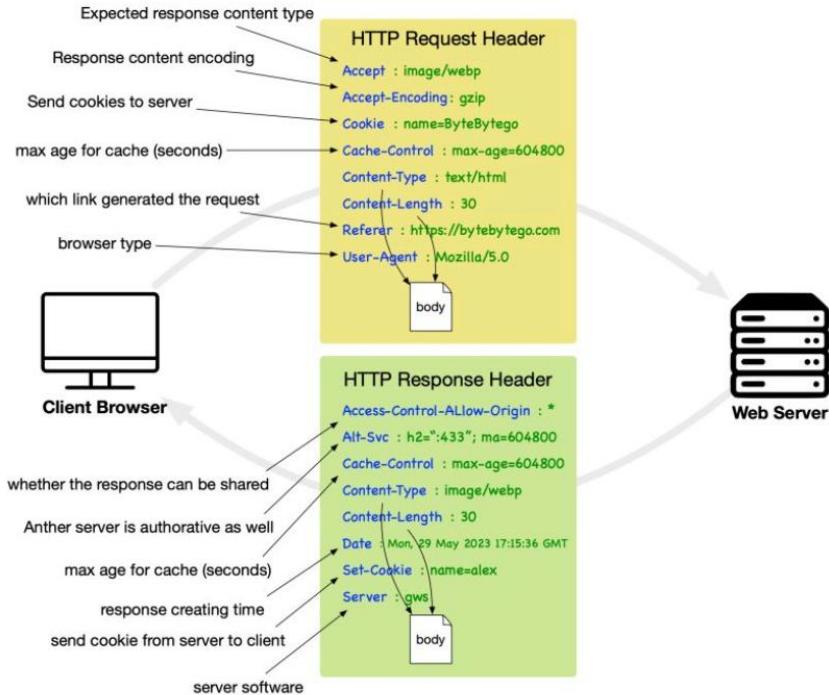
معایب: پیاده‌سازی آن پیچیده‌تر است و نیاز به طراحی دقیق دارد.

## **موارد مهم درباره هدرهای HTTP**

درخواست‌های HTTP مانند درخواست کردن چیزی از یک سرور هستند و پاسخ‌های HTTP پاسخ‌های سرور هستند. این کار مانند ارسال پیام و دریافت پاسخ است.

هدر درخواست HTTP دارای تعدادی اطلاعات اضافی است که هنگام ارسال درخواست فرستاده می‌شود، مانند اطلاعاتی در مورد نوع داده‌ای که ارسال می‌کنید یا چه کسی داده‌ها را ارسال کرده است، می‌باشد. اما در هدرهای پاسخ، سرور اطلاعاتی درباره پاسخ‌ها مانند نوع داده‌ای که دریافت می‌کنید یا اینکه چه دستورالعمل‌های خاصی دارید را ارائه می‌دهد.

## What's inside the HTTP Header?

 ByteByteGo.com


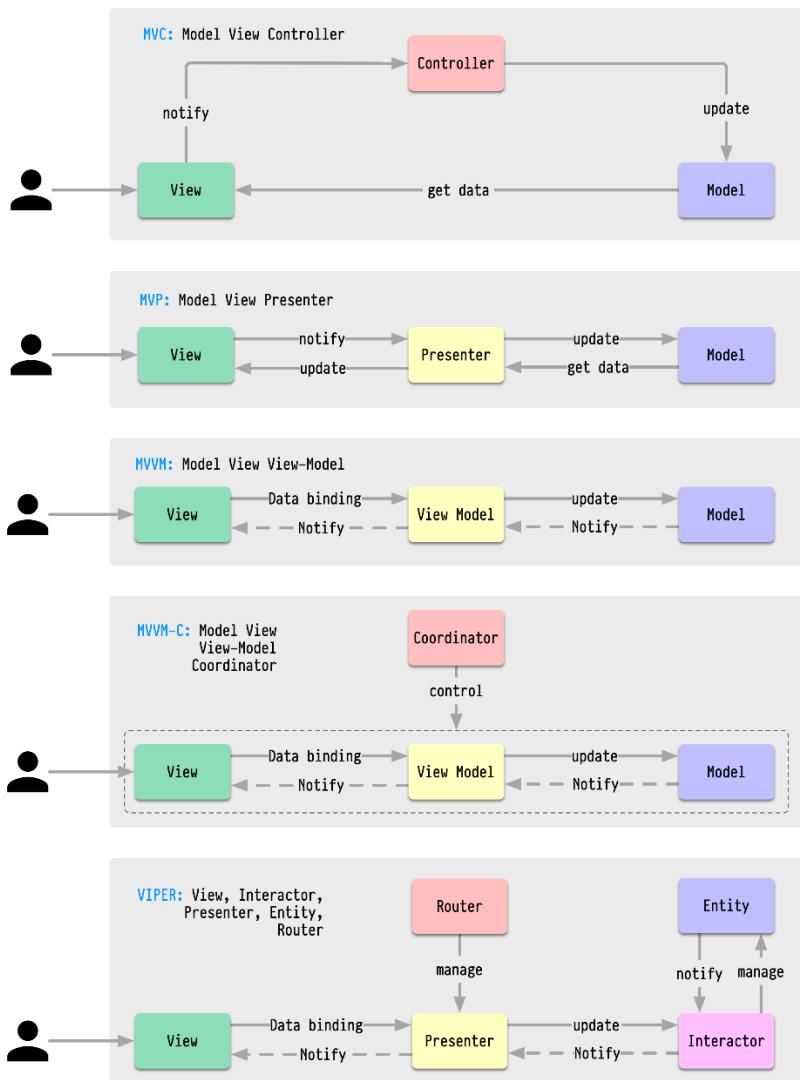
هدر نقش حیاتی در فعال کردن ارتباط کلاینت-سرور هنگام ساخت برنامه‌های RESTful دارد. برای ارسال اطلاعات صحیح با درخواست‌های خود و تفسیر صحیح پاسخ‌های سرور، باید این هدرها را درک کنید.

# الگوهای معماری نرم افزار

الگوهای MVC، MVP، MVVM و VIPER از جمله رایج ترین الگوهای مورد استفاده در توسعه اپلیکیشن‌ها در پلتفرم‌های iOS و اندروید هستند.

MVC, MVP, MVVM, VIPER patterns

 ByteByteGo.com



توسعه دهنده‌گان آنها را برای غلبه بر محدودیت‌های الگوهای قبلی، این الگوها را معرفی کرده‌اند. حالا، آنها چه فرقی با هم دارند؟

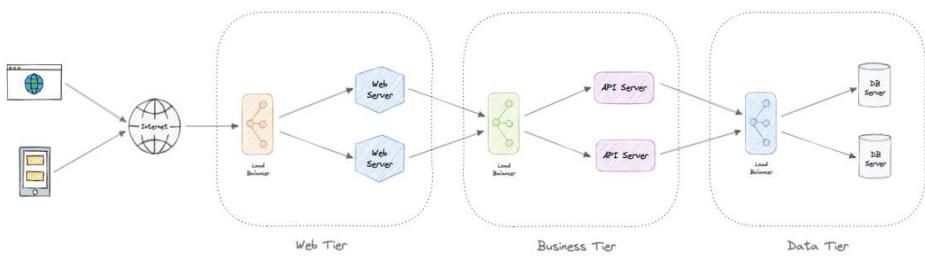
- MVC، قدیمی‌ترین الگو است و قدمتی تقریباً ۵۰ ساله دارد.
- در هر الگو، یک "view" (V) مسئول نمایش محتوا و دریافت ورودی کاربر است.
- اکثر الگوهای شامل یک "model" (M) برای مدیریت داده‌های کسب و کار هستند.
- "view-model" و "presenter" و "Controller" مترجم‌هایی هستند که بین "view" و "model" به خصوص ("entity"<sup>1</sup>) در الگوی VIPER شراکت می‌کنند.

---

<sup>1</sup> موجودیت

## بررسی معماری های N-Tier

معماری چندلایه (N-tier) یک برنامه‌ی نرم افزاری را به لایه‌های منطقی و طبقات فیزیکی تقسیم می‌کند. لایه‌ها روشی برای جداسازی مسئولیت‌ها و مدیریت وابستگی‌ها هستند. هر لایه وظیفه خاصی دارد. یک لایه بالاتر می‌تواند از سرویس‌های لایه پایین‌تر استفاده کند، اما برعکس این موضوع امکان‌پذیر نیست.



لایه‌ها از نظر فیزیکی جدا شده‌اند و روی ماشین‌های جداگانه‌ای اجرا می‌شوند. یک لایه می‌تواند مستقیماً به لایه دیگر فراخوانی بزند یا از پیام‌رسانی ناهزم زمان<sup>۱</sup> استفاده کند. اگرچه ممکن است هر لایه در طبقه خاص خود میزبانی<sup>۲</sup> شود، اما اجباری در این کار نیست. چندین لایه ممکن است روی یک لایه میزبانی شوند. جداسازی فیزیکی طبقات، مقیاس‌پذیری و انعطاف‌پذیری را بهبود می‌بخشد، اما تأخیر ناشی از ارتباطات اضافی شبکه را نیز به همراه دارد.

یک معماری چندلایه می‌تواند از دو نوع باشد:  
معماری لایه بسته<sup>۳</sup>: در این نوع، یک لایه فقط می‌تواند لایه بعدی را که مستقیماً پایین‌تر از آن است را فراخوانی کند.

<sup>۱</sup> asynchronous messaging

<sup>۲</sup> hosted

<sup>۳</sup> Closed Layer Architecture

معماری لایه باز<sup>۱</sup>: در این نوع، یک لایه می‌تواند هر یک از لایه‌های پایین‌تر از خود را فرآخوانی کند. معماری لایه بسته وابستگی‌های بین لایه‌ها را محدود می‌کند. با این حال، ممکن است ترافیک شبکه غیرضروری ایجاد کند و این مورد به خصوص زمانی که یک لایه به سادگی در خواست‌ها را به لایه بعدی منتقل می‌کند، اهمیت می‌یابد.

بیایید به چند نمونه از معماری‌های چندلایه را بررسی کنیم:

**معماری سه لایه**: معماری سه لایه به طور گسترده مورد استفاده قرار می‌گیرد و از لایه‌های مختلف زیر تشکیل شده است:

لایه نمایش<sup>۲</sup>: با تعاملات کاربر با برنامه سروکار دارد.

لایه منطق کسب‌وکار<sup>۳</sup>: داده‌ها را از لایه Presentation دریافت می‌کند، طبق business logic اعتبارسنجی می‌کند و به لایه داده منتقل می‌کند.

لایه دسترسی به داده<sup>۴</sup>: داده‌ها را از لایه business logic دریافت می‌کند و عملیات لازم را روی پایگاه داده انجام می‌دهد.

### معماری دو لایه

در این معماری، لایه Presentation روی کلاینت اجرا می‌شود و با یک بانک اطلاعاتی ارتباط برقرار می‌کند. هیچ لایه business logic یا لایه دیگری بین کلاینت و سرور وجود ندارد.

### معماری تک لایه

این نوع ساده‌ترین معماری است، زیرا معادل اجرای برنامه روی یک رایانه شخصی است. تمام اجزای موردنیاز برای اجرای یک برنامه در یک برنامه یا سرور واحد قرار دارند.

Open Layer Architecture <sup>۱</sup>

Presentation layer <sup>۲</sup>

Business Logic layer <sup>۳</sup>

Data Access layer <sup>۴</sup>

**مزایا:**

- در اینجا برخی از مزایای استفاده از معماری چندلایه آورده شده است:
- می‌تواند پارامترهای در دسترس بودن<sup>۱</sup> را بهبود بخشد.
  - سطح امنیت بهتری به همراه می‌آورد زیرا لایه‌ها می‌توانند مانند فایروال عمل کنند.
  - امکان مقیاس‌پذیری<sup>۲</sup> لایه‌های مجرما در صورت نیاز را فراهم می‌کند.
  - پارامترهای نگهداری از برنامه را بهبود می‌بخشد زیرا افراد مختلف می‌توانند لایه‌های مختلف را مدیریت کنند.

**معایب:**

- در زیر برخی از معایب استفاده از معماری چندلایه آورده شده است:
- افزایش پیچیدگی در کل سیستم.
  - افزایش تأخیر شبکه با افزایش تعداد لایه‌ها.
  - پرهزینه بودن به دلیل اینکه هر لایه هزینه سخت‌افزاری خاص خود را دارد.
  - مدیریت امنیت شبکه دشوار است.

## چگونه الگوهای طراحی را یاد بگیریم؟

علاوه بر خواندن کدهای خوب نوشته شده، یک کتاب خوب مانند یک معلم خوب به ما راهنمایی می‌کند.

من کتاب "Head First Design Patterns"، ویرایش دوم را پیشنهاد می‌کنم.



وقتی مسیرم را در مهندسی نرم‌افزار شروع کردم، در کتاب کلاسیک "Design Patterns" نوشته گروه Gang of Four Patterns ساخت بود. خوشبختانه، من کتاب "Head First Design Patterns" را در کتابخانه مدرسه پیدا کردم. این کتاب Design Patterns زیادی برای من حل کرد. وقتی دوباره به کتاب برگشتم، همه چیز آشنا و قابل فهم تر به نظر می‌رسید.

سال گذشته، من ویرایش دوم کتاب "Head First Design Patterns" را خریداری کردم و آن را خواندم. در اینجا چند چیز هست که در مورد این کتاب دوست دارم:

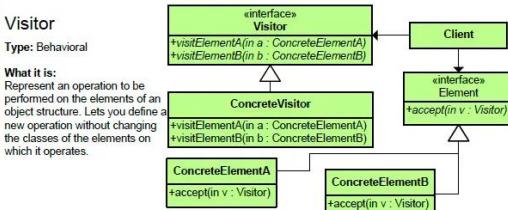
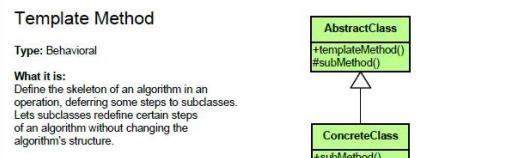
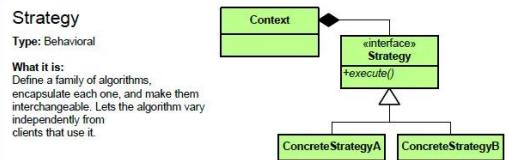
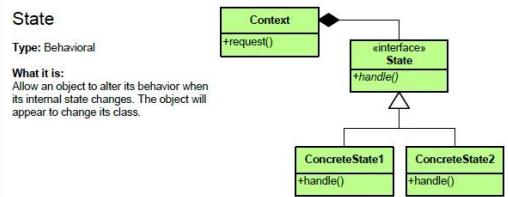
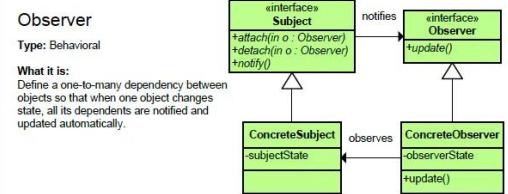
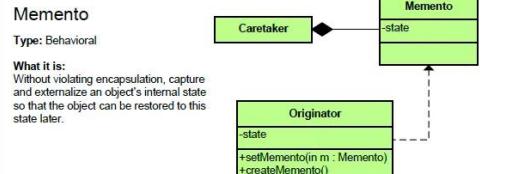
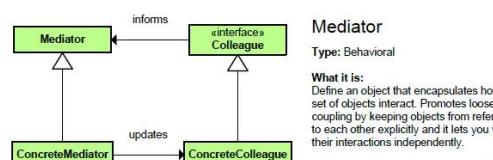
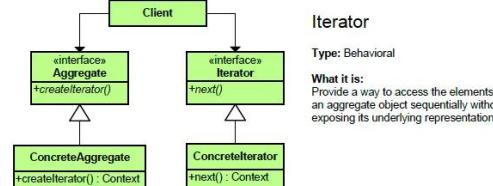
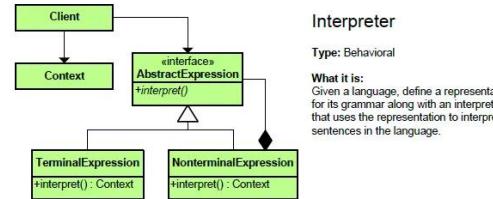
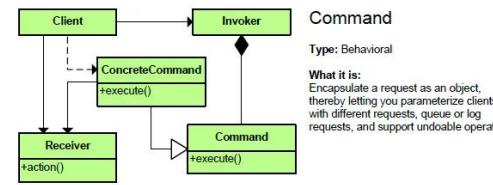
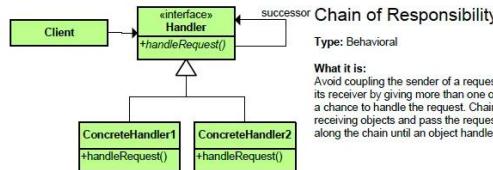
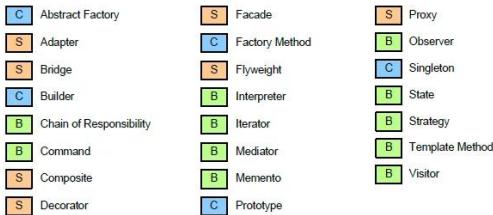
- این کتاب به چالش طبیعت انتزاعی و «نامرئی» نرم‌افزار پرداخته است. ساخت نرم‌افزار دشوار است؛ زیرا ما نمی‌توانیم معماری آن را بینیم؛ جزئیات آن در کد و فایل‌های باینری جاسازی شده‌اند. درک الگوهای طراحی نرم‌افزار حتی دشوارتر است؛ زیرا این‌ها انتزاع‌های سطح بالاتری از نرم‌افزار هستند. این کتاب با استفاده از بصری‌سازی این مشکل را حل می‌کند. تقریباً در هر صفحه‌ای تعداد زیادی دیاگرام، فلش و نظر وجود دارد. اگر متن را درک نکنم، مشکلی نیست. دیاگرام‌ها چیزها را بسیار خوب توضیح می‌دهند.
- همه ما سؤالاتی داریم که وقتی اولین‌بار مهارت جدیدی را یاد می‌گیریم، از پرسیدن آن‌ها می‌ترسیم. شاید فکر کنیم این یک سؤال آسان است. این کتاب در مقابله با الگوهای طراحی از دید دانشجو خوب عمل می‌کند. این کتاب با پرسیدن سؤالات ما و پاسخ‌دادن واضح به آن‌ها، ما را راهنمایی می‌کند. در کتاب یک استاد و همچنین یک دانشجو وجود دارد.

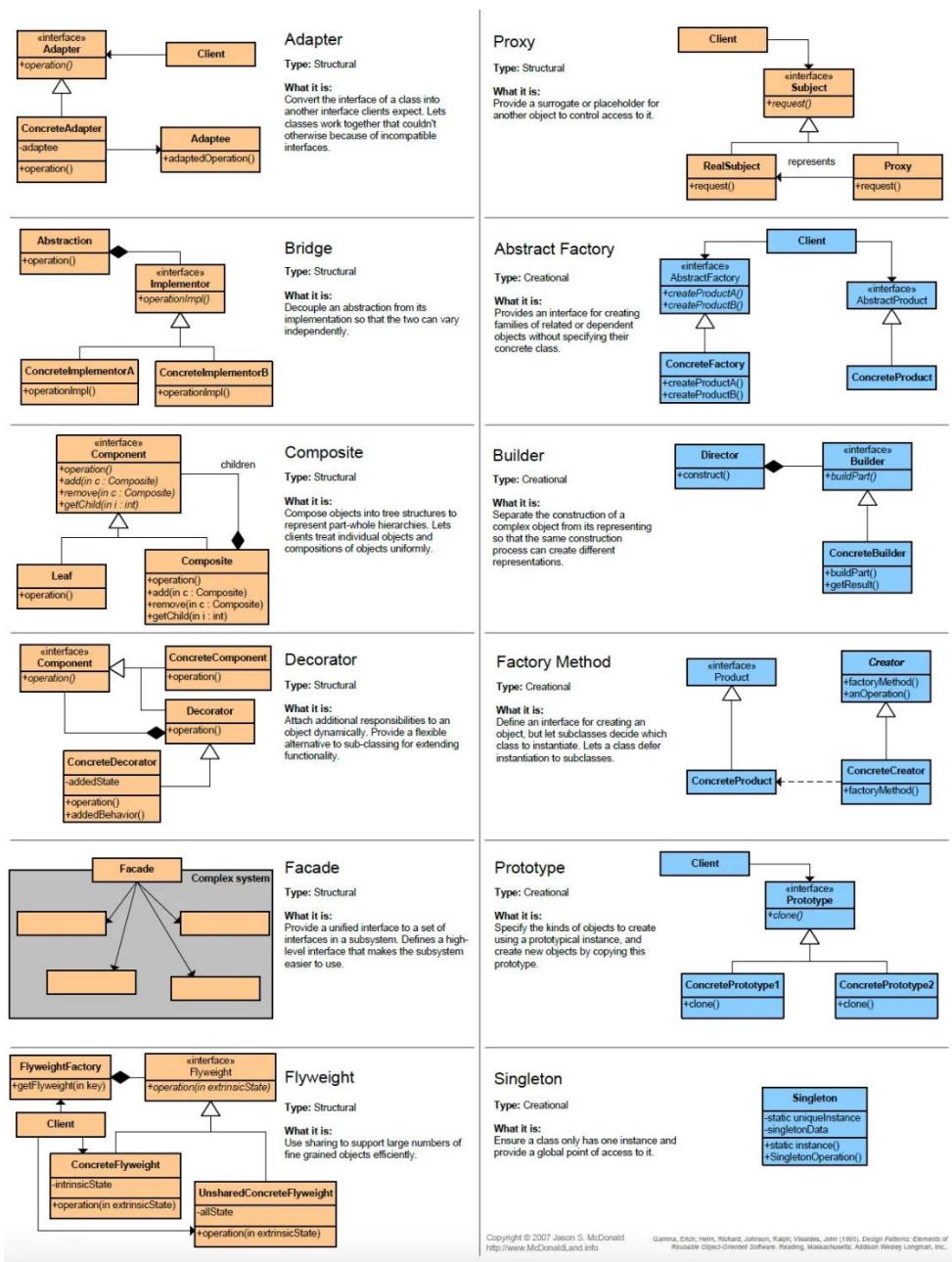
## الگوهای طراحی کلیدی (Design Patterns)

الگوها راه حل های قابل استفاده مجدد برای مشکلات رایج طراحی هستند که منجر به یک فرایند توسعه روان تر و کارآمدتر می شوند. آنها به عنوان الگوهایی برای ساختارهای نرم افزاری بهتر عمل می کنند. اینها برخی از محبوب ترین الگوها هستند:

مترجم: در حال نگارش کتابی با این موضوع هستیم.

18 Key Design Patterns Every Developer Should Know			ByteByteGo.com
<b>Abstract Factory</b> Family creator Create groups of related items	<b>Builder</b> Lego master Build object step by step	<b>Prototype</b> Cloner Create copies from examples	
<b>Singleton</b> The one and only With just one instance	<b>Adapter</b> Universal plug Connect different interfaces	<b>Bridge</b> Connector Link what is to how it works	
<b>Composite</b> Tree builder Create tree-like structure	<b>Decorator</b> Customizer Add new features to existing object	<b>Facade</b> One-stop shop Single interface to all functions	
<b>Flyweight</b> Space saver Share small, reusable items	<b>Proxy</b> Middle man Represent another object	<b>Chain of responsibility</b> Replayer Relay requests until it is handles	
<b>Command</b> Task wrapper Turn a request into object	<b>Iterator</b> Explorer Assess element one by one	<b>Mediator</b> Hub Simplify communication between classes	
<b>Memento</b> Capsule Capture and store object state	<b>Observer</b> Broadcaster Notify others about the change	<b>Visitor</b> Guests Explore an object without changing it	

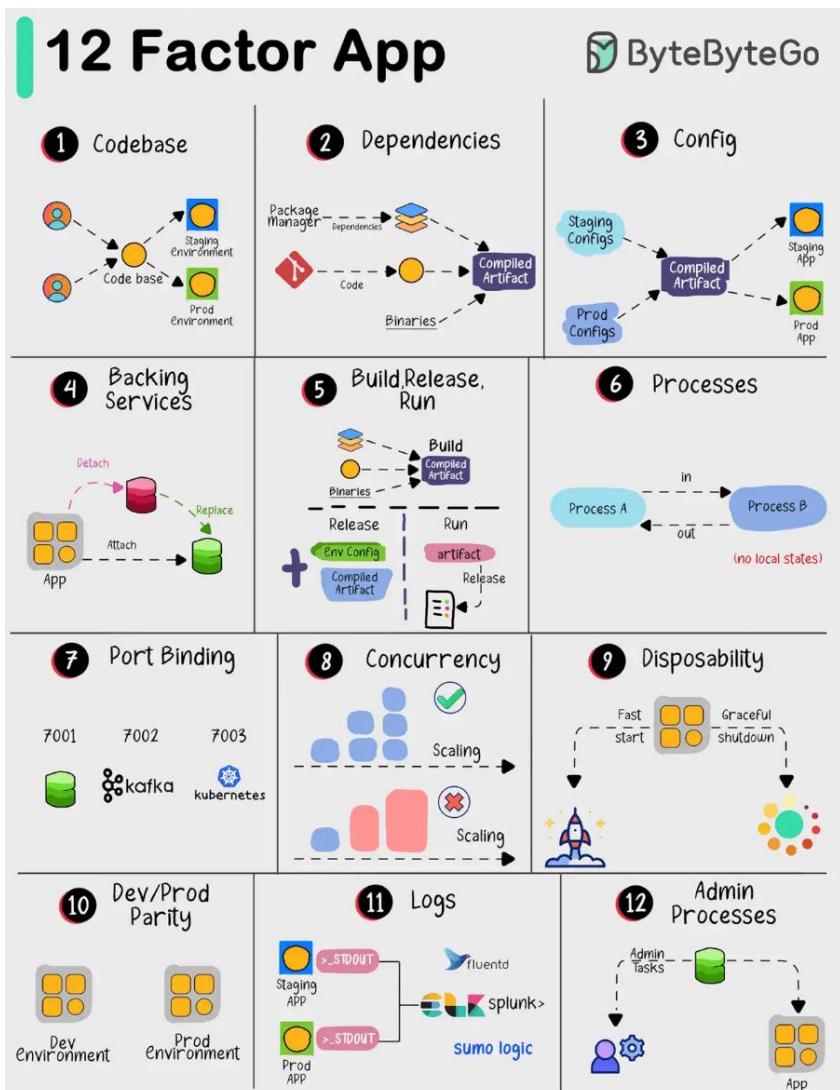




## تسلط بر اصول طراحی - SOLID

مترجم: در حال نگارش کتابی با این موضوع هستیم.

## ۱۲ فاکتور مهم در طراحی اپلیکیشن



"۱۲ فاکتور مهم در اپلیکیشن‌ها" مجموعه‌ای از بهترین شیوه‌ها را برای ساخت اپلیکیشن‌های نرم افزاری مدرن ارائه می‌دهد. به کاربستن این ۱۲ اصل می‌تواند به توسعه‌دهندگان و تیم‌ها در ساخت اپلیکیشن‌های قابل اعتماد، مقیاس‌پذیر و قابل مدیریت کمک کند. در اینجا خلاصه‌ای از هر اصل آمده است:

#### :پایگاه کد (Codebase)

تمام کدهای خود را در یک مکان نگه دارید و با استفاده از کتلر نسخه مانند Git آن‌ها را مدیریت کنید.

#### :وابستگی‌ها (Dependencies)

لیستی از تمام چیزهایی که برنامه شما برای کارکرد صحیح نیاز دارد تهیه کنید و مطمئن شوید که به راحتی قابل نصب هستند.

#### :تنظیمات (Config)

تنظیمات مهم مانند اعتبارنامه‌های پایگاه‌داده را جدا از کد خود نگه دارید تا بتوانید آنها را بدون بازنویسی کد تغییر دهید.

#### :سروریس‌های پشتیبان (Backing Services)

از سرویس‌های دیگر (مانند پایگاه‌داده یا پردازشگرهای پرداخت<sup>(۱)</sup> به عنوان اجزای جداگانه‌ای که برنامه شما به آنها متصل می‌شود، استفاده کنید.

#### :Build, Release, Run

تفاوت مشخصی بین آماده‌سازی برنامه، انتشار آن و اجرای آن در محیط عملیاتی قائل شوید.

#### :Processes

برنامه خود را به گونه‌ای طراحی کنید که هر بخش به یک کامپیوتر یا حافظه خاصی وابسته نباشد. این شبیه ساخت قطعات پازل است که به هم متصل می‌شوند.

### **:Port Binding**

اجازه دهید برنامه شما از طریق یک پورت شبکه قابل دسترسی باشد و مطمئن شوید که اطلاعات مهم را در یک کامپیوتر واحد ذخیره نمی کند.

### **:Concurrency**

با اضافه کردن نسخه های بیشتر از یک task مشابه، مانند استخدام کارگران بیشتر برای یک رستوران پر از مشتری، برنامه خود را قادر به مدیریت task های بیشتر کنید.

### **یکبار مصرف بودن (Disposability)**

برنامه شما باید به سرعت راه اندازی و به درستی خاموش شود، مانند خاموش کردن کلید برق به جای کشیدن دوشاخه از پریز.

### **هم ارزی توسعه / تولید (Dev/Prod Parity)**

اطمینان حاصل کنید که آنچه برای توسعه برنامه خود استفاده می کنید بسیار شبیه به چیزی است که در تولید استفاده می کنید تا از غافلگیری جلوگیری شود.

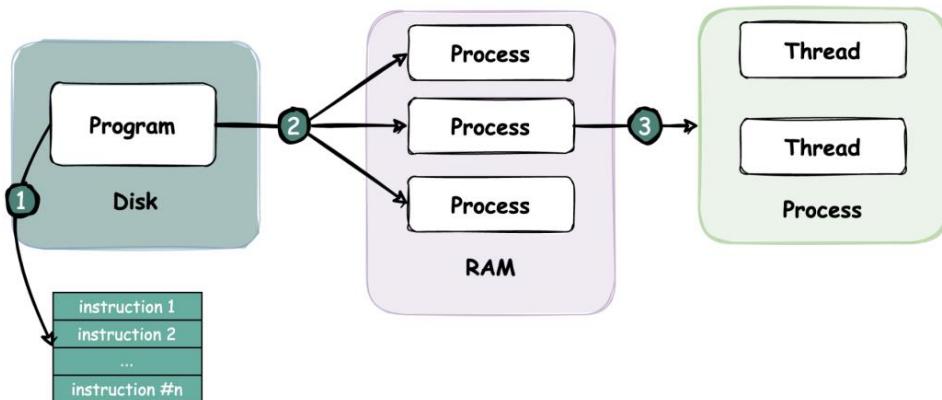
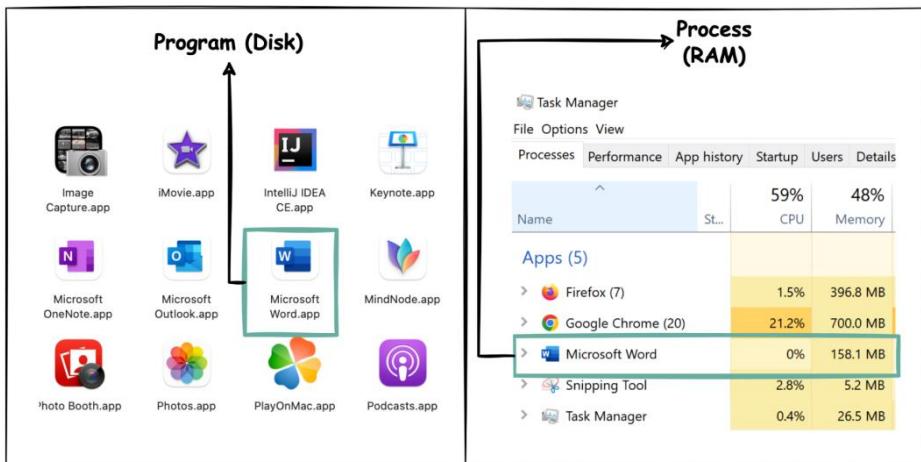
### **لاغ ها (Logs)**

رکوردي از اتفاقات رخداده در برنامه خود نگه دارید تا بتوانید مشکلات را درک و حل کنید، مانند یک دفتر خاطرات برای نرم افزار خود عمل کنید.

### **فرایندهای مدیریتی:**

تسک های ویژه و خاص را جدا از اپلیکیشن خود اجرا کنید.

## تفاوت بین Thread و Process چیست؟



برای درک بهتر این سؤال، ابتدا بیایید نگاهی به این بیندازیم که یک برنامه چیست. یک برنامه یک فایل قابل اجرا است که شامل مجموعه‌ای از دستورات عمل‌ها بوده و به صورت غیرفعال روی دیسک ذخیره می‌شود. یک برنامه می‌تواند دارای چندین فرایند باشد. به عنوان مثال، مرورگر کروم برای هر تب جداگانه یک Process متفاوت ایجاد می‌کند.

یک Process به معنای اجرای یک برنامه است. زمانی که یک برنامه در حافظه بارگذاری شده و فعال می‌شود، آن برنامه به یک Process تبدیل می‌شود. Process به برخی منابع ضروری مانند رجیسترها، شمارنده برنامه و پسته<sup>۱</sup> نیاز دارد.

یک رشته (Thread) کوچک‌ترین واحد اجرایی درون یک Process است. فرایند زیر رابطه بین برنامه، Thread و Process را توضیح می‌دهد.

- برنامه شامل مجموعه‌ای از دستورات عمل‌ها است.

برنامه در حافظه بارگذاری می‌شود و به یک یا چند Process در حال اجرا تبدیل می‌شود.

زمانی که یک Process شروع می‌شود، حافظه و منابع به آن اختصاص داده می‌شود. یک Process می‌تواند دارای یک یا چند Thread باشد. به عنوان مثال، در برنامه مایکروسافت ورد، یک Thread ممکن است مسئول بررسی املاء و Thread دیگر مسئول واردکردن متن به سند باشد.

#### تفاوت‌های اصلی بین Process و Thread

- Process‌ها معمولاً مستقل هستند، درحالی‌که Thread‌ها به عنوان زیرمجموعه‌هایی از یک Process وجود دارند.

هر Process دارای فضای حافظه خود است. Thread‌هایی که به یک Process تعلق دارند، حافظه مشترکی را به اشتراک می‌گذارند.

یک Process دارای سربار عملیاتی سنگین است. ایجاد و خاتمه‌دادن به آن زمان بیشتری می‌برد.

تغییر زمینه<sup>۲</sup> بین Process‌ها هزینه‌برتر است.

ارتباط بین Thread‌ای برای Thread‌ها سریع‌تر است.

stack<sup>۱</sup>

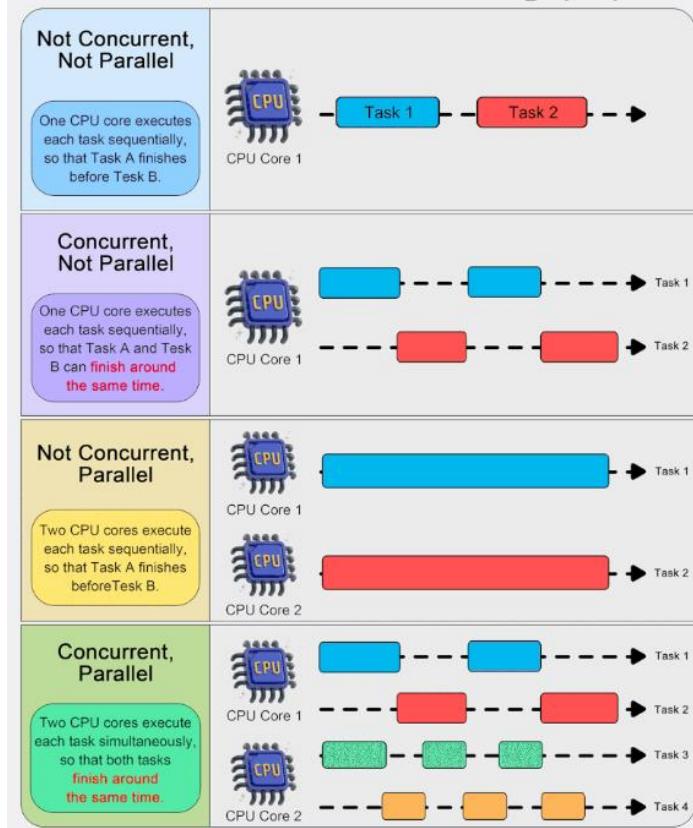
Context switch<sup>۲</sup>

## مقایسه همزمان (Parallelism) با موازی‌سازی (Concurrency)

در طراحی سیستم، درک تفاوت بین همزمان و موازننگاری بسیار مهم است.

### Concurrency is **Not** Parallelism

ByteByteGo



همان‌طور که «راب پایک» (یکی از خالقان زبان برنامه‌نویسی GoLang) بیان کرد: «همزمانی<sup>۱</sup> در مورد برخورده و تعامل با بسیاری از کارها به طور همزمان است. موازی‌سازی<sup>۲</sup> در مورد

Concurrency<sup>۱</sup>  
Parallelism<sup>۲</sup>

انجام بسیاری از کارها به طور هم‌زمان است». این تمایز تأکید می‌کند که هم‌زمان بیشتر در مورد شیوه طراحی<sup>۱</sup> یک برنامه دارای کاربرد است، درحالی‌که موازی‌سازی در مورد روش‌های اجرا<sup>۲</sup> است.

Concurrency در مورد برخورد با چندین کار به طور هم‌زمان عمل می‌کند و این شامل ساختار دادن یک برنامه برای رسیدگی به چندین وظیفه یا task به طور هم‌زمان است به خصوص جایی که task‌ها می‌توانند در دوره‌های زمانی خاصی دارای همپوشانی در شروع، اجرا و تکمیل فرایندهای خود باشند، اما لزوماً اجرای این فرایندها در یک لحظه واحد نیستند.

Concurrency در مورد ترکیب فرایندهای اجرای مستقل است و توانایی یک برنامه را برای مدیریت چندین کار با پیشرفت در آنها بدون نیاز به تکمیل یکی قبل از شروع دیگری توصیف می‌کند.

از سوی دیگر، موازی‌سازی (Parallelism) به اجرای هم‌زمان محاسبات متعدد اشاره دارد. این تکنیک اجرای هم‌زمان دو یا چند وظیفه task یا محاسبه است که با استفاده از چندین پردازنده یا هسته در یک رایانه برای انجام چندین عملیات به طور هم‌زمان انجام می‌شود. موازنۀ سازی به سخت‌افزاری با چندین واحد پردازش نیاز دارد و هدف اصلی آن افزایش توان عملیاتی و سرعت محاسباتی یک سیستم است.

از نظر عملی، هم‌زمان به یک برنامه اجازه می‌دهد تا حتی در یک پردازنده تک‌هسته‌ای، نسبت به ورودی پاسخگو بماند و کارهای پس‌زمینه را انجام دهد و چندین عملیات را به‌ظاهر هم‌زمان را مدیریت کند. این امر به‌ویژه در عملیات **I/O-bound** و با تأخیر بالا

design<sup>۱</sup>

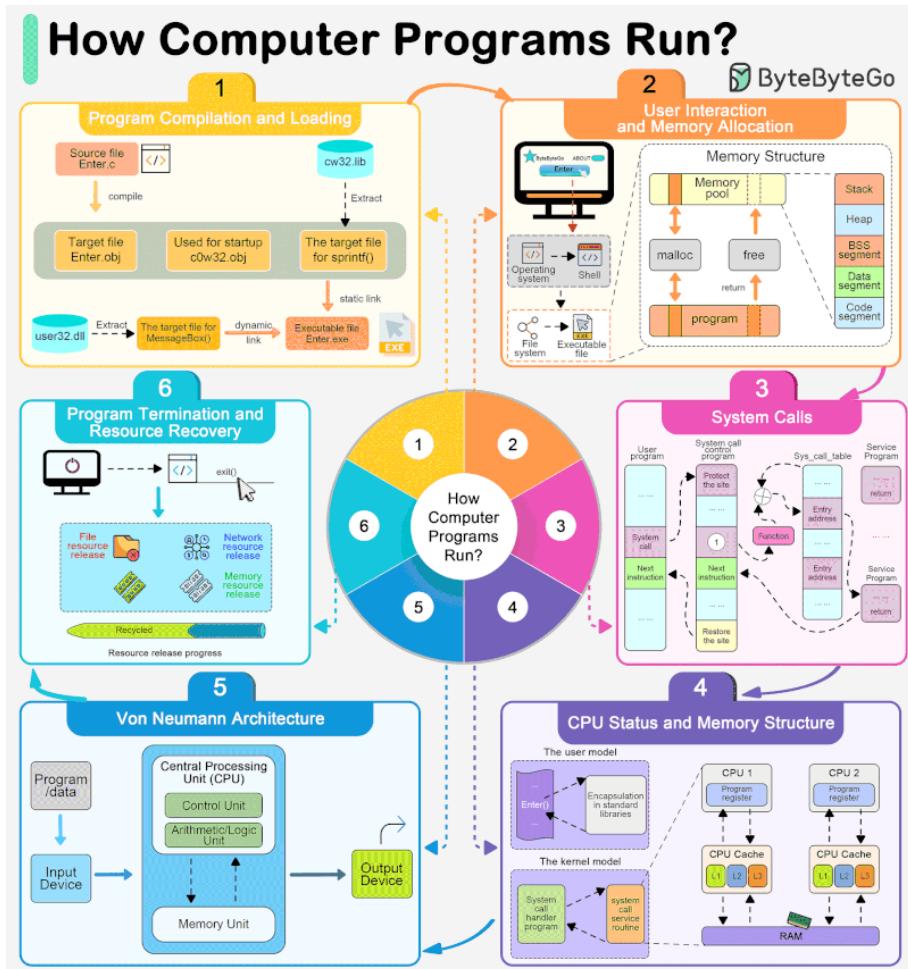
execution<sup>۲</sup>

مفید است، جایی که برنامه‌ها باید منتظر رویدادهای خارجی مانند تعامل با فایل، شبکه یا کاربر باشند.

معمولًاً با توانایی انجام چندین عملیات به طور همزمان در کارهای وابسته به Parallelism CPU که سرعت محاسباتی و توان عملیاتی به عنوان یک گلوگاه هستند، بسیار مهم است. برنامه‌هایی که نیاز به محاسبات ریاضی سنگین، تحلیل داده‌ها، پردازش تصویر و پردازش بلادرنگ دارند، می‌توانند به طور قابل توجهی از اجرای موازی بهره‌مند شوند.

## چگونه برنامه های کامپیوتری اجرا می شوند؟

این نمودار، مراحل اجرا ی برنامه را نشان می دهد:

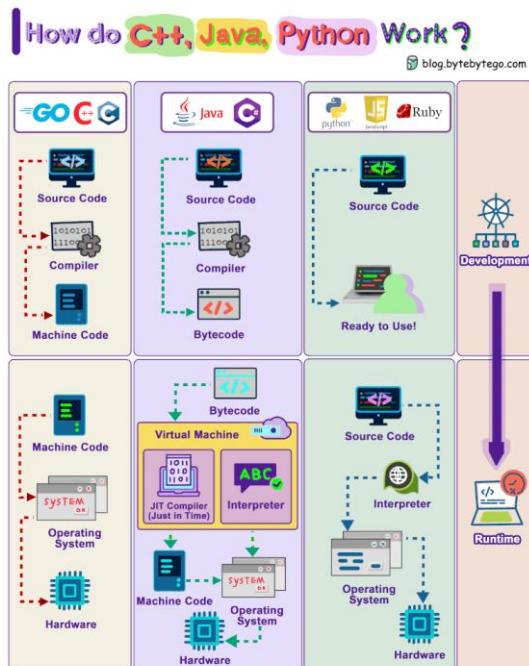


- تعامل کاربر و شروع فرمان با دو بار کلیک روی یک برنامه، کاربر از طریق رابط کاربری گرافیکی به سیستم عامل دستور اجرای یک برنامه را می دهد.
- پیش بارگذاری برنامه پس از آغاز درخواست اجرا، سیستم عامل ابتدا فایل اجرایی برنامه را بازیابی می کند.

- سیستم عامل این فایل را از طریق سیستم فایل پیدا کرده و برای آماده سازی اجرا، آن را در حافظه بارگذاری می کند.
- حل وابستگی و بارگذاری اکثر برنامه های مدرن به تعدادی کتابخانه مشترک مانند کتابخانه های پیوند پویا (DLL) وابسته هستند.
- سیستم عامل باید این وابستگی ها را شناسایی و بارگذاری کند تا برنامه به درستی اجرا شود.
- تخصیص فضای حافظه سیستم عامل مسئول اختصاص فضای حافظه است. سیستم عامل برای اجرای برنامه، میزان مناسبی از حافظه را به آن اختصاص می دهد.
- راه اندازی محیط زمان اجرا پس از اختصاص حافظه، سیستم عامل و محیط اجرا (برای مثال JVM جاوا یا دات نت فریم ورک) منابع مختلف موردنیاز برای اجرای برنامه را راه اندازی می کنند.
- فراخوانی های سیستمی و مدیریت منابع نقطه ورود یک برنامه (معمولأً تابعی به نام main) فراخوانده می شود تا اجرای کد نوشته شده توسط برنامه نویس آغاز شود.
- این فراخوانی ها به سیستم عامل اجازه می دهند تا با سخت افزار و سایر منابع سیستم تعامل داشته باشد.
- معماری Neumann در معماری فون نویمان، CPU دستورالعمل های ذخیره شده در حافظه را اجرا می کند.
- برنامه به صورت مجموعه ای از دستورالعمل ها است که آنها را یک به یک پردازش می کند.
- خاتمه برنامه در نهایت، هنگامی که برنامه کار خود را به پایان رساند یا کاربر به طور فعال برنامه را خاتمه دهد، برنامه وارد گام پاک سازی می شود. این شامل بستن توصیفگرهای فایل باز، آزاد کردن منابع شبکه و بازگرداندن حافظه به سیستم می شود.

## چگونه C++, Java و پایتون کار می کنند؟

این نمودار نحوه کار کامپایل و اجرا را نشان می دهد:



زبان‌های کامپایل شده: توسط کامپایلر به کد ماشین ترجمه می‌شوند. کد ماشین می‌تواند مستقیماً توسط CPU اجرا شود. مثال‌ها: Go, C++, C, C#.

زبان‌های بایت‌کد: مانند جاوا، ابتدا کد منبع را به بایت‌کد کامپایل می‌کنند، سپس JVM برنامه را اجرا می‌کند. گاهی اوقات کامپایلر (JIT Just-In-Time) برای سرعت‌بخشیدن به اجرا، کد منبع را به کد ماشین کامپایل می‌کند. مثال‌ها: جاوا، #C#.

زبان‌های تفسیری: کامپایل نمی‌شوند. آن‌ها در زمان اجرا توسط مفسر تفسیر می‌شوند. مثال‌ها: پایتون، جاوا اسکریپت، روئی

به طور کلی، زبان‌های کامپایل شده سریع‌تر از زبان‌های تفسیری اجرا می‌شوند.

## ۸ اصطلاح برتر برنامه‌نویسی

Top 8 programming paradigms - PART I		
 <p><b>Imperative Programming</b></p>	<p>describes a sequence of steps that change the program's state.</p> <ul style="list-style-type: none"> <li>Mutable State</li> <li>Procedural Approach</li> <li>Sequential Execution</li> <li>Explicit Flow Control</li> <li>Close to Hardware</li> </ul>	 <pre>public static void main(String[ ] args) {     int a = 0;     int b = 1;     int c = 2;     int sum = a + b + c; }</pre>
 <p><b>Object-Oriented Programming</b></p>	<p>encapsulates data (attributes) and behavior (methods or functions).</p> <ul style="list-style-type: none"> <li>Objects and Classes</li> <li>Encapsulation</li> <li>Inheritance</li> <li>Polymorphism</li> <li>Modularity</li> </ul>	 <pre>class Person {     private int age;     private String firstName;     private String lastName;      public Person(int age, String firstName, String lastName) {         this.age = age;         this.firstName = firstName;         this.lastName = lastName;     } }</pre>
 <p><b>Functional Programming</b></p>	<p>treats computation as the evaluation of mathematical functions</p> <ul style="list-style-type: none"> <li>Immutable Data</li> <li>Referential Transparency</li> <li>First-Class Functions</li> <li>Lazy Evaluation</li> <li>Recursion</li> </ul>	 <pre>ifExpression(     isEven(         randomNumber()     ),     "Heads",     "Tails" );</pre>
 <p><b>Generic Programming</b></p>	<p>creating reusable, flexible, and type-independent code</p> <ul style="list-style-type: none"> <li>Type Parameterization</li> <li>Code Reusability</li> <li>Templates and Generics</li> <li>Compile-time Polymorphism</li> </ul>	 <pre>public class List&lt;T&gt; {     ... }</pre>

### ۱. برنامه‌نویسی دستوری (Imperative Programming)

برنامه‌نویسی دستوری، توالی‌ای از مراحل را توصیف می‌کند که وضعیت برنامه را تغییر می‌دهند. زبان‌هایی مانند C، C++، جاوا، پایتون (تا حدودی) و بسیاری دیگر از سبک‌های برنامه‌نویسی دستوری پشتیبانی می‌کنند.

## ۲. برنامه‌نویسی اعلانی (Declarative Programming)

برنامه‌نویسی اعلانی بر بیان منطق و عملکردها بدون توصیف صریح جریان کتترل تأکید می‌کند. برنامه‌نویسی تابعی (Functional Programming) شکلی محبوب از برنامه‌نویسی اعلانی است.

## ۳. برنامه‌نویسی شی‌ء گرا (Object-Oriented Programming - OOP)

برنامه‌نویسی شی‌ء گرا (OOP) حول مفهوم اشیاء می‌چرخد که داده (صفات) و رفتار (متدها یا توابع) را در بر می‌گیرند. زبان‌های رایج برنامه‌نویسی شی‌ء گرا شامل جاوا، C++, پایتون، روبي و سی‌شارپ (C#) هستند.

## ۴. برنامه‌نویسی جنبه‌گرا (Aspect-Oriented Programming - AOP)

هدف برنامه‌نویسی جنبه‌گرا (AOP) مازولار کردن دغدغه‌هایی است که بر روی بخش‌های مختلف یک سیستم نرم‌افزاری تأثیر می‌گذارد. AspectJ یکی از شناخته شده‌ترین فریم‌ورک‌های AOP است که جاوا را با قابلیت‌های AOP گسترش می‌دهد.

## ۵. برنامه‌نویسی تابعی (Functional Programming - FP)

برنامه‌نویسی تابعی محاسبات را به عنوان ارزیابی توابع ریاضی در نظر می‌گیرد و بر استفاده از داده‌های غیرقابل تغییر<sup>۱</sup> و عبارات اعلانی<sup>۲</sup> تأکید می‌کند. زبان‌هایی مانند هاسکل، لیسب، ارلنگ و برخی ویژگی‌ها در زبان‌هایی مانند جاوا اسکریپت، پایتون و اسکالا از الگوهای برنامه‌نویسی تابعی پشتیبانی می‌کنند.

## ۶. برنامه‌نویسی واکنشی (Reactive Programming)

<sup>۱</sup> immutable

<sup>۲</sup> declarative

برنامه‌نویسی واکنشی با جریان‌های داده ناهم‌زمان<sup>۱</sup> و انتشار تغییرات سروکار دارد. اپلیکیشن‌های رویدادمحور<sup>۲</sup> و پردازش جریان داده<sup>۳</sup> از برنامه‌نویسی واکنشی بهره می‌برند.

## ۷. برنامه‌نویسی عمومی (Generic Programming)

برنامه‌نویسی عمومی با اجازه‌دادن به نوشتن الگوریتم‌ها و ساختارهای داده بدون مشخص کردن نوع داده‌هایی که روی آن‌ها عمل می‌کنند و به دنبال ایجاد کد قابل استفاده مجدد، انعطاف‌پذیر و مستقل از type است. برنامه‌نویسی عمومی به طور گسترده در کتابخانه‌ها و فریم‌ورک‌ها برای ایجاد ساختارهای داده؛ مانند لیست‌ها، پسته‌ها، صفحه‌ها و الگوریتم‌هایی مانند مرتب‌سازی و جستجو استفاده می‌شود.

## ۸. برنامه‌نویسی هم‌زمان (Concurrent Programming)

برنامه‌نویسی هم‌زمان به اجرای هم‌زمان چندین وظیفه/task یا فرایند می‌پردازد و باعث بهبود عملکرد و استفاده از منابع می‌شود. برنامه‌نویسی هم‌زمان در اپلیکیشن مختلفی از جمله سرورهای چندرشته‌ای<sup>۴</sup>، پردازش موازی، سرورهای وب هم‌زمان و محاسبات با کارایی بالا<sup>۵</sup> مورد استفاده قرار می‌گیرد.

---

۱ asynchronous

۲ Event-driven applications

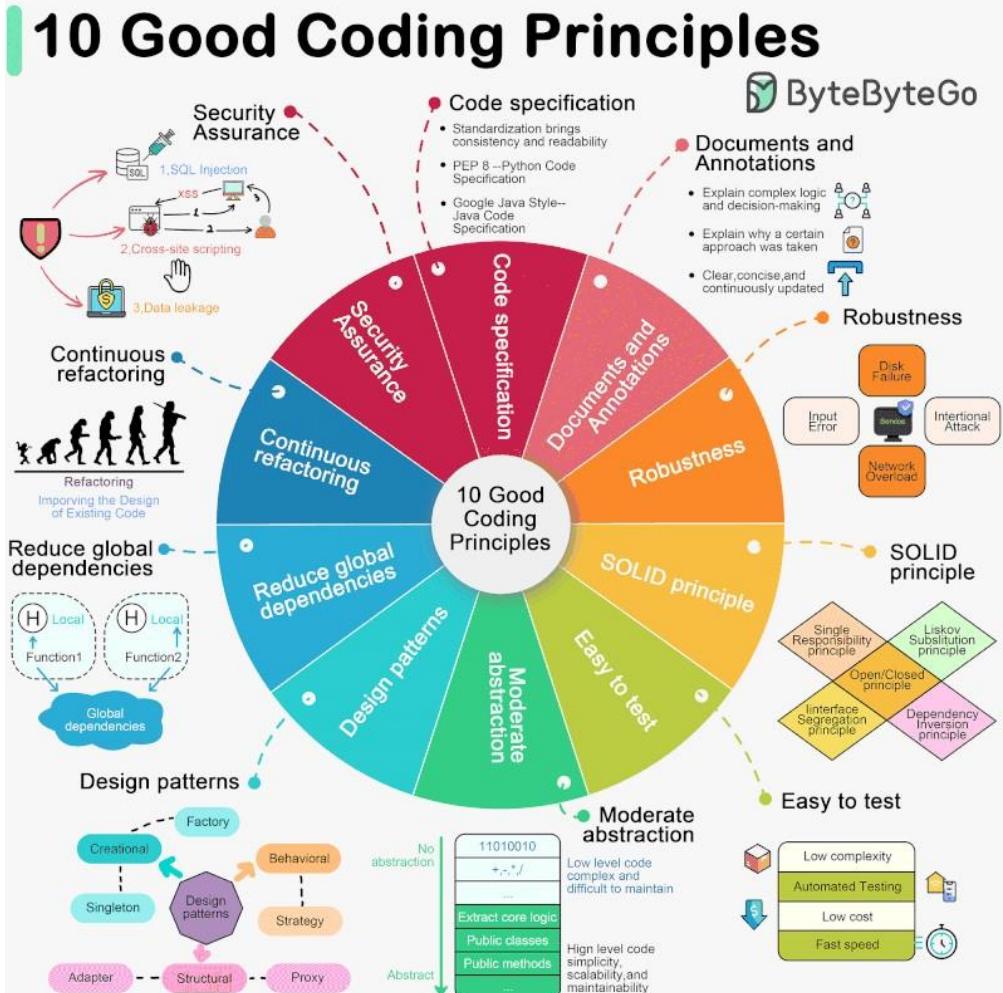
۳ streaming data processing

۴ multi-threaded

۵ high-performance computing

## ۱۰ اصول مهم برنامه نویسی برای بهبود کیفیت کد

توسعه نرم افزار نیازمند طراحی سیستم مناسب و استانداردهای کدنویسی است. در این لیست ۱۰ اصل مهم برنامه نویسی برای بهبود کیفیت کد ارائه شده است:



### ۱. پیروی از مشخصات کد:

هنگام نوشتن کد، مهم است که از استانداردهای شناخته شده‌ای مانند "PEP 8" یا "Java Style" تبعیت کنیم. پایبندی به مجموعه‌ای از توافقات در مورد مشخصات کد، تضمین‌کننده‌ی کیفیت، ثبات و خوانایی کد است.

## ۲. مستندسازی و کامنت‌گذاری:

کد خوب باید به طور واضح مستندسازی و کامنت‌گذاری شود تا منطق و تصمیمات پیچیده توضیح داده شوند. کامنت‌ها باید به جای توضیح «چه کاری<sup>۱</sup>» انجام می‌شود به «چرا یعنی<sup>۲</sup>» انتخاب یک رویکرد خاص بپردازند. مستندات و کامنت‌ها باید واضح، مختصر و به طور مداوم به روزرسانی شوند.

## ۳. استحکام (Robustness):

کد خوب باید بتواند با انواع موقعیت‌های غیرمنتظره و ورودی‌ها بدون خرابی یا ایجاد نتایج غیرقابل‌پیش‌بینی برخورد کند. متداول‌ترین رویکرد، گرفتن و مدیریت استثنایات<sup>۳</sup> است.

## ۴. پیروی از اصول SOLID:

این پنج اصل که با حروف اول آن‌ها (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) کلمه‌ی SOLID را تشکیل می‌دهند، سنگ بنای نوشتن کدی هستند که قابلیت مقیاس‌پذیری و نگهداری آسان دارد.

## ۵. ساده بودن تست:

قابلیت تست‌پذیری نرم‌افزار از اهمیت ویژه‌ای برخوردار است. کد خوب باید به راحتی قابل تست باشد و این امر هم از طریق کاهش پیچیدگی هر کامپوننت و هم از طریق پشتیبانی از تست خودکار برای اطمینان از رفتار مورد انتظار، حاصل می‌شود.

## ۶. Abstraction:

انتزاع نیازمند استخراج منطق هسته برنامه و پنهان‌کردن پیچیدگی است، بنابراین کد را انعطاف‌پذیرتر و عمومی‌تر می‌کند. کد خوب باید سطح متوسطی از انتزاع داشته باشد و نه

What<sup>۱</sup>

Why<sup>۲</sup>

Exceptions<sup>۳</sup>

طراحی بیش از حد و نه نادیده‌گرفتن قابلیت توسعه و نگهداری بلندمدت برنامه یا کد مورد استفاده.

۷. استفاده از الگوهای طراحی، اما در استفاده از آنها زیاده‌روی نکنید:  
الگوهای طراحی می‌توانند به حل برخی مشکلات رایج کمک کنند. با این حال، هر الگو سناریوهای قابل اجرای خاص خود را دارد. استفاده بیش از حد یا نادرست از الگوهای طراحی ممکن است کد شما را پیچیده‌تر و درک آن را دشوار کند.

۸. کاهش وابستگی‌های سراسری:  
اگر از متغیرهای سراسری<sup>۱</sup> و نمونه‌ها استفاده کنیم، ممکن است در وابستگی‌ها و مدیریت وضعیت گیج‌کننده گرفتار شویم. کد خوب باید بر روی وضعیت محلی و پاس‌دادن پارامترها تکیه کند. توابع باید بدون عوارض جانبی<sup>۲</sup> باشند.

۹. بازنگری مستمر (Refactoring):  
کد خوب قابل نگهداری و توسعه‌پذیر است. بازنگری مستمر با شناسایی و رفع مشکلات در اساعی وقت، بدھی فنی را کاهش می‌دهد.

۱۰. امنیت در اولویت است:  
کد خوب باید از آسیب‌پذیری‌های رایج امنیتی اجتناب کند.

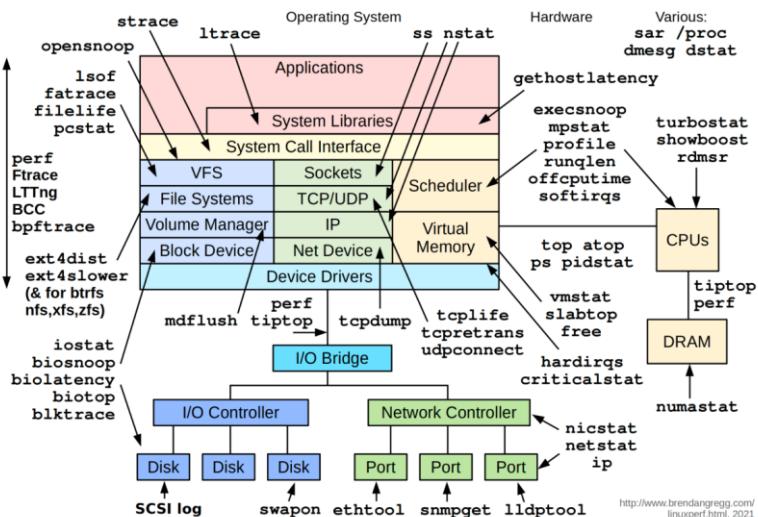
instances<sup>۱</sup>

Side-effect<sup>۲</sup>

چگونه یک فرایند خاصی که بیش از حد منابع مصرف می‌کند را تشخیص دهیم؟

نمودار زیر ابزارهای مفید در یک سیستم لینوکس را نشان می‌دهد.

### Linux Performance Observability Tools



- 'vmstat' - اطلاعاتی در مورد فرایندها، حافظه، صفحه‌بندی، I/O بلوکی، تله‌ها و فعالیت CPU ارائه می‌دهد.
- آمار CPU و ورودی/خروجی سیستم را گزارش می‌کند.
- داده‌های آماری مربوط به پروتکل‌های IP، TCP، UDP و ICMP را نمایش می‌دهد.
- 'lsof' - فایل‌های باز سیستم جاری را لیست می‌کند.
- استفاده از منابع سیستم از جمله CPU، حافظه، I/O دستگاه، تعویض threadها و غیره را برای همه یا فرایندهای مشخص شده نظارت می‌کند.

## ۹ مورد رایج که مصرف CPU را به اوج می رساند.

نمودار زیر مقصران رایجی را نشان می دهد که می توانند منجر به استفاده ۱۰۰٪ از CPU شوند. درک این موارد می تواند به تشخیص مشکلات و بهبود کارایی سیستم کمک کند.



- ۱ Infinite Loops .۱
- ۲ Background Processes .۲
- ۳ High Traffic Volume .۳
- ۴ Resource-Intensive Applications .۴
- ۵ Insufficient Memory .۵
- ۶ Concurrent Processes .۶
- ۷ Busy Waiting .۷
- ۸ Regular Expression Matching .۸
- ۹ Malware and Viruses .۹

## بهترین روش‌های تست عملکرد سیستم

تست عملکرد سیستم، گامی اساسی در فرایند توسعه و مهندسی نرم‌افزار به شمار می‌رود. این تست اطمینان حاصل می‌کند که یک سیستم یا اپلیکیشن طبق انتظار عمل کرده و نیازهای کاربر را برآورده سازد و به طور قابل اعتمادی کار کند.

Process	Illustration	Tools
Unit Testing		
Integration Testing		
System Testing		
Load Testing		
Error Testing		
Test Automation		

در اینجا به بهترین روش‌ها برای تست عملکرد سیستم می‌پردازیم:

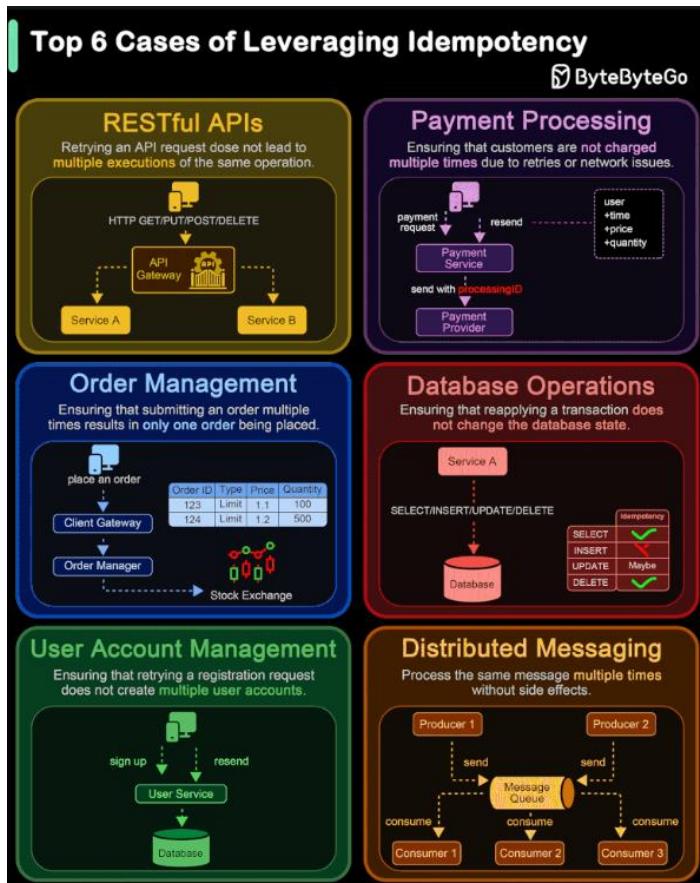
۱. تست واحد (Unit Testing): تضمین می‌کند که اجزای کد به صورت جداگانه و

مستقل به درستی کار می‌کنند.

۲. تست یکپارچه (Integration Testing): بررسی می کند که اجزای مختلف سیستم به طور یکپارچه و بدون مشکل با هم کار می کنند.
۳. تست سیستم (System Testing): انطباق کل سیستم با نیازمندی هایی کاربر و عملکرد آن را ارزیابی می کند.
۴. تست بار (Load Testing): توانایی سیستم در مدیریت حجم کاری بالا را آزمایش می کند و مشکلات عملکردی را شناسایی می کند.
۵. تست خطا (Error Testing): چگونگی مدیریت ورودی های نامعتبر و شرایط خطا توسط نرم افزار را ارزیابی می کند.
۶. تست خودکار (Test Automation): اجرای سناریوهای تست را برای کارایی، تکرار پذیری و کاهش خطا را به صورت خودکار انجام می دهد.

## ۶ مورد برتر برای اعمال Idempotency

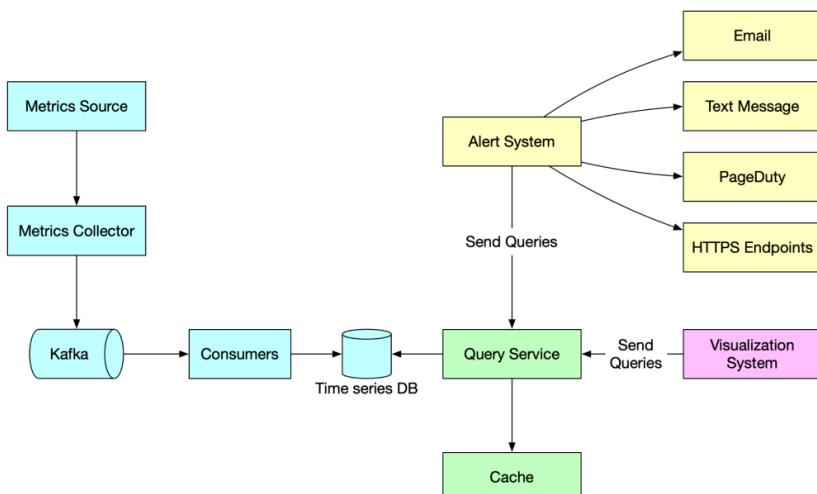
ایدمپوتنسی<sup>۱</sup> در سناریوهای مختلف، به خصوص زمانی که عملیات ممکن است دوباره امتحان شوند یا چندین بار اجرا شوند، ضروری است. در اینجا ۶ مورد برتر از سناریوهایی که در آن‌ها حیاتی است، آورده شده است:



۱. درخواست‌های API RESTful: باید اطمینان حاصل کنیم که تکرار مجدد یک درخواست API منجر به اجرای چندباره همان عملیات نشود. برای حفظ وضعیت ثابت منابع، از روش‌های ایدمپوتنس (مانند PUT و DELETE) استفاده کنید.
۲. پردازش‌های پرداخت مالی<sup>۱</sup>: باید اطمینان حاصل کنیم که به دلیل تکرار مجدد یا مشکلات شبکه از مشتریان چندین بار هزینه دریافت نشود. درگاه‌های پرداخت اغلب نیاز به تکرار تراکنش‌ها دارند؛ Idempotency اطمینان می‌دهد که تنها یک‌بار هزینه دریافت شود.
۳. دستگاه‌های مدیریت سفارش‌ها: باید اطمینان حاصل کنیم که ارسال چندباره یک سفارش منجر به ثبت تنها یک سفارش شود. یک مکانیزم ایمن برای جلوگیری از کسر یا بهروزرسانی تکراری موجودی طراحی کنیم.
۴. عملیات پایگاهداده: باید اطمینان حاصل کنیم که اعمال مجدد یک تراکنش، وضعیت پایگاهداده را فراتر از کاربرد اولیه تغییر ندهد.
۵. مدیریت حساب کاربری: باید اطمینان حاصل کنیم که تکرار مجدد درخواست ثبت‌نام منجر به ایجاد چندین حساب کاربری نشود. همچنین باید اطمینان حاصل کنیم که درخواست‌های متعدد برای بازنشانی رمز عبور منجر به یک عمل بازنشانی واحد شود.
۶. سیستم‌های توزیع شده و پیام‌رسانی: باید اطمینان حاصل کنیم که پردازش مجدد پیام‌ها از صفر منجر به پردازش تکراری نشود. هندرلهایی<sup>۲</sup> را پیاده‌سازی کنیم که بتوانند یک پیام را چندین بار بدون عوارض جانبی پردازش کنند.

## سیستم نظارت و هشدار برای متريکها

یک سیستم واضح در مورد سلامت زيرساخت برای اطمینان از بالا بودن سطوح دسترس پذيری<sup>۱</sup> و قابلیت اطمینان دارد. نمودار زیر نحوه کار آن را در سطح بالا توضیح می دهد.



**منبع متريکها<sup>۲</sup>:** اين می تواند سرورهای اپليکيشن، پايگاهداده های SQL، صفحه های پيام و غيره باشد.

**جمع کننده متريکها<sup>۳</sup>:** اين سیستم داده های متريک را جمع آوری كرده و آنها را در پايگاهداده سري زمانی می نويسد.

<sup>۱</sup> availability

<sup>۲</sup> Metrics source

<sup>۳</sup> Metrics collector

**پايگاهداده سري زمانی**<sup>۱</sup>: اين پايگاهداده، داده‌های متريک را به صورت سري زمانی ذخیره می‌کند. معمولاً<sup>۲</sup> يك رابط کوئري سفارشی برای تحليل و خلاصه‌سازی حجم زيادي از داده‌های سري زمانی ارائه می‌دهد. شاخص‌هایي<sup>۳</sup> را در مورد برچسب‌ها<sup>۴</sup> حفظ می‌کند تا جستجوی سريع داده‌های سري زمانی بر اساس برچسب‌ها را تسهيل کند.

**کافکا**: کافکا به عنوان يك پلتفرم ارسال پیام توزيع شده با قابلیت اطمینان و مقیاس‌پذیری بالا استفاده می‌شود. آن سرویس‌های جمع‌آوری داده و پردازش داده را از یکدیگر جدا می‌کند.

**صرف‌کنندگان**<sup>۵</sup>: مصرف‌کنندگان یا سرویس‌های پردازش جريانی مانند Apache Storm Flink و Spark داده‌ها را پردازش می‌کنند و به پايگاهداده سري زمانی ارسال می‌کنند.

**سرویس کوئري**<sup>۶</sup>: اين سرویس، کوئري‌های ساده‌ای ایجاد می‌کند و بازیابی داده از پايگاهداده سري زمانی را فراهم می‌کند. اين باید يك لایه باریکی باشد اگر يك پايگاهداده سري مانی خوب انتخاب کنيم. همچنین می‌تواند به طور كامل با رابط کوئري خود پايگاهداده سري زمانی جايگزين شود.

**سیستم هشدار**<sup>۷</sup>: اين اعلان‌های هشدار را به مقاصد هشدار مختلف ارسال می‌کند.

**سیستم مصورسازی**<sup>۸</sup>: اين متريک‌ها را به صورت انواع نمودارها/چارت‌ها نشان می‌دهد.

---

<sup>۱</sup> Time-series database

<sup>۲</sup> indexes

<sup>۳</sup> labels

<sup>۴</sup> Consumers

<sup>۵</sup> Query service

<sup>۶</sup> Alerting system

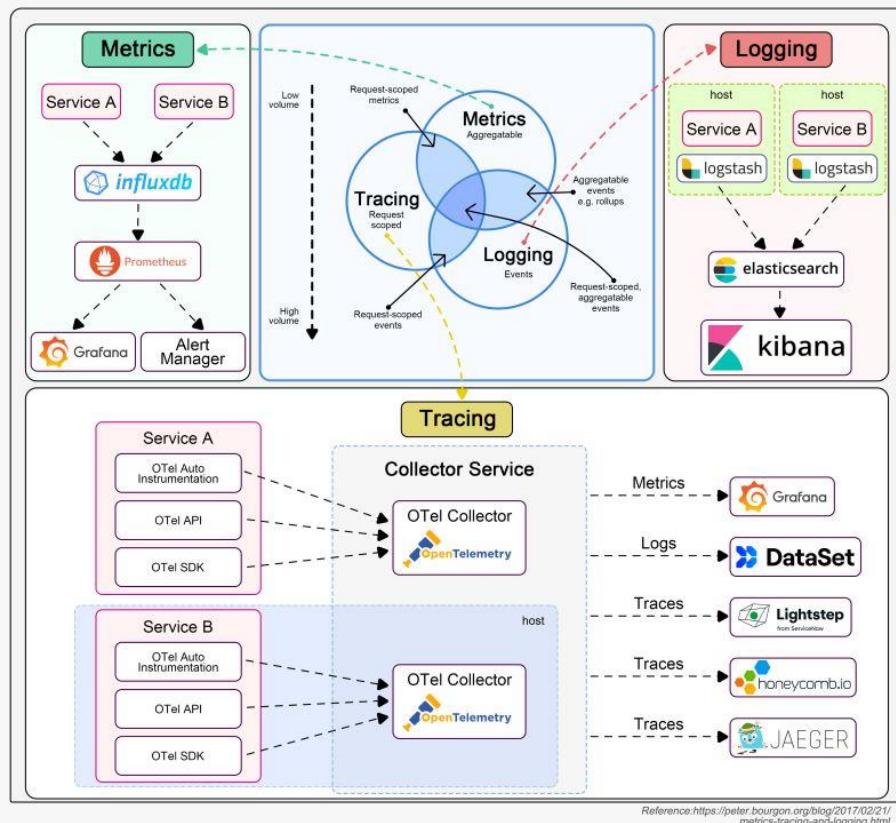
<sup>۷</sup> Visualization system

## قابلیت مشاهده: لاگ‌گیری، ردیابی و معیارها

قابلیت مشاهده<sup>۱</sup> به توانایی درک و تحلیل عملکرد یک سیستم نرم‌افزاری از طریق جمع‌آوری و تجزیه و تحلیل داده‌ها گفته می‌شود. سه ستون اصلی قابلیت مشاهده عبارت‌اند از: لاگ‌گیری، ردیابی و متريک‌ها<sup>۲</sup>.

# Logging, Tracing, Metrics

ByteByteGo



Observability<sup>۱</sup>  
logging, tracing, metrics<sup>۲</sup>

## لاغ‌گیری (Logging)

لاغ‌گیری شامل ضبط وقایع مجزا در داخل یک سیستم، مانند درخواست‌های ورودی یا دسترسی به پایگاه‌داده است. این فرایند معمولاً حجم بالایی از داده را تولید می‌کند. ابزار ELK (شامل Elasticsearch، Logstash و Kibana) به طور رایج برای ساخت پلتفرم‌های تحلیل لاغ استفاده می‌شود. پیاده‌سازی فرمتهای لاغ‌گیری استاندارد در سراسر تیم‌ها امکان جستجوی کارآمد در مجموعه‌داده‌های لاغ را فراهم می‌سازد.

## ردیابی (Tracing)

ردیابی، بینش و درکی کلی از مسیر پیموده شده درخواست‌ها در سراسر اجزای سیستم مانند API‌ها، توزیع‌کننده‌های بار<sup>۱</sup>، سرویس‌ها و پایگاه‌های داده را ارائه می‌دهد. این فرایند در شناسایی گلوگاه‌های<sup>۲</sup> عملکردی بسیار مهم است. OpenTelemetry یک رویکرد یکپارچه برای پیاده‌سازی لاغ‌گیری، ردیابی و متریک‌ها در یک معماری واحد ارائه می‌دهد.

## متریک‌ها

متریک‌ها، نقاط داده‌ی تجمعی را نشان می‌دهند که بیانگر وضعیت عملیاتی یک سیستم هستند، از جمله نرخ کوئری، پاسخگویی API و میزان تأخیر در سرویس‌ها. این داده‌های سری زمانی در پایگاه‌داده‌ای مانند InfluxDB کوئری می‌شوند و اغلب توسط ابزارهای مانند Prometheus پردازش می‌شوند که از کوئری و هشداردهی<sup>۳</sup> بر اساس متریک‌های خاص پشتیبانی می‌کند. مصورسازی<sup>۴</sup> و هشدار بر اساس متریک‌ها می‌تواند در پلتفرم‌های مانند Grafana انجام شود که با مکانیسم‌های هشداردهی مختلف مانند ایمیل، پیام کوتاه یا Slack ادغام می‌شود.

Load Balancers<sup>۱</sup>

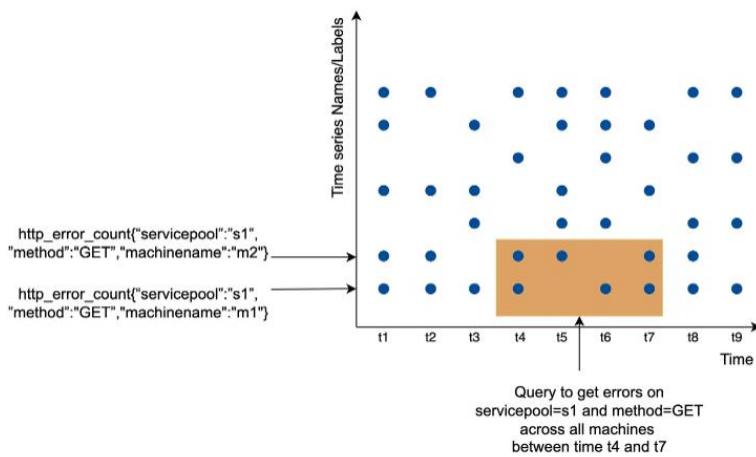
Bottlenecks<sup>۲</sup>

alerting<sup>۳</sup>

Visualization<sup>۴</sup>

## کدام پایگاهداده برای سیستم جمعآوری متريکها مناسب است؟

اين يكى از مهمترین سؤالاتى است که باید در يك مصاحبه به آن پاسخ دهيم.



## الگوی دسترسی به داده

همان طور که در نمودار نشان داده شده است، هر برچسب<sup>۱</sup> در محور  $\text{y}$  نماینده یک سری زمانی (که به طور منحصر به فرد توسط نامها و برچسبها شناسایی می شود) در حالی که محور  $\text{x}$  نشان دهنده زمان است. حجم بار کاری در سمت نوشتن داده ها سنگین است. همان طور که می بینید، می تواند در هر لحظه نقاط داده های سری زمانی زیادی نوشته شود. میلیون ها متريک عملیاتی در روز نوشته می شود و بسیاری از متريکها با فرکانس بالا جمع آوری می شوند، بنابراین ترافیک در سمت نوشتن داده بدون شک سنگین است. در همان زمان، بار خواندن به صورت لحظه ای<sup>۲</sup> است. هم سرویس های نمایش و هم سرویس های هشدار، کوئری های را به پایگاهداده ارسال می کنند و بسته به الگوهای دسترسی نمودارها و هشدارها، حجم خواندن می تواند شدید باشد.

<sup>۱</sup>label  
<sup>۲</sup>spiky

## انتخاب پایگاهداده مناسب

سیستم ذخیره داده قلب طراحی هر سیستمی است. توصیه نمی شود که سیستم ذخیره سازی خودتان را بسازید یا از یک سیستم ذخیره سازی همه منظوره (MySQL) برای این کار استفاده کنید. یک پایگاهداده همه منظوره، در تئوری می تواند از داده های سری زمانی پشتیبانی کند، اما نیاز به تنظیم سطح حرفه ای دارد تا در مقیاس مورد نظر کار کند. به طور خاص، یک پایگاهداده رابطه ای برای عملیات هایی که معمولاً روی داده های سری زمانی انجام می شود، بهینه نشده است. برای مثال، محاسبه میانگین متحرک<sup>۱</sup> در یک پنجره زمانی چرخشی<sup>۲</sup> نیازمند SQL پیچیده است که خواندن آن دشوار است. علاوه بر این، برای پشتیبانی از برچسب گذاری داده ها، باید برای هر برچسب یک نمایه اضافه کنیم. علاوه بر این، یک پایگاهداده رابطه ای همه منظوره در برابر بار نوشتن سنگین مداوم عملکرد خوبی ندارد. در مقیاس مورد نظر، باید تلاش زیادی برای تنظیم پایگاهداده صرف کنیم و حتی پس از آن، ممکن است عملکرد خوبی نداشته باشد.

NoSQL چطور؟ در حالت تئوری، چند پایگاهداده NoSQL می توانند داده های سری زمانی را به طور مؤثر مدیریت کنند. برای مثال، هم Bigtable و هم Cassandra می توانند برای داده های سری زمانی استفاده شوند. با این حال، این نیازمند دانش عمیق از عملکرد داخلی هر NoSQL برای طراحی یک طرح قابل مقیاس برای ذخیره و کوئری مؤثر داده های سری زمانی است. با وجود پایگاهداده های سری زمانی در مقیاس صنعتی که به راحتی در دسترس هستند، استفاده از یک پایگاهداده NoSQL همه منظوره جذاب نیست.

<sup>۱</sup> moving average

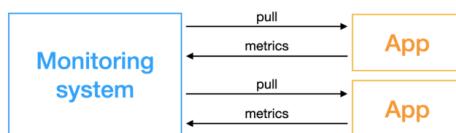
<sup>۲</sup> rolling time window

سیستم‌های ذخیره‌سازی زیادی وجود دارند که برای داده‌های سری زمانی بهینه‌سازی شده‌اند. بهینه‌سازی به ما امکان می‌دهد از سرورهای بسیار کمتری برای مدیریت همان حجم از داده‌ها استفاده کنیم. بسیاری از این پایگاه‌داده‌ها همچنین دارای رابطه‌های کوئری سفارشی هستند که به طور خاص برای تجزیه و تحلیل داده‌های سری زمانی طراحی شده‌اند و استفاده از آنها بسیار راحت‌تر از SQL است. برخی حتی ویژگی‌هایی برای مدیریت نگهداری داده و تجمعیع داده ارائه می‌دهند.

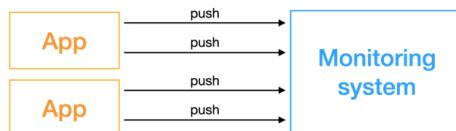
## مدل‌های Push در مقابل Pull

دو روش برای جمع‌آوری داده‌های مربوط به متریک وجود دارد: push یا pull. این که کدام روش بهتر است، بحثی دائمی است و پاسخ روشنی وجود ندارد.

Pull-based system



Push-based monitoring system



سیستم مانیتورینگ Pull-based، همانطور که از نام آن پیداست، یک سیستم ناظری است که به طور فعال شاخص‌ها را به دست می‌آورد و اشیایی که نیاز به نظارت دارند باید قابلیت دسترسی از راه دور و remote را داشته باشند. سیستم‌های مانیتورینگ Push-based به طور فعال داده‌ها را به دست نمی‌آورند، اما اشیاء نظارت شده به طور فعال شاخص‌های خود را ارسال می‌کنند. بین این دو روش از بسیاری جهات تفاوت وجود دارد. برای ساخت و انتخاب سیستم‌های مانیتورینگ باید مزایا و معایب این دو روش را از قبل درک کرده و طرح مناسب را برای اجرا انتخاب کنیم. در غیر این صورت، هزینه نگهداری بعدی برای اطمینان از پایداری سیستم نظارت و هزینه استقرار و نگهداری آن بسیار زیاد خواهد بود.

در ادامه ما به مدل pull نگاهی خواهیم انداخت. شکل ۱ جمع‌آوری داده‌ها با مدل pull روی HTTP را نشان می‌دهد. ما جمع‌کننده‌های اختصاصی متریک‌ها را داریم که به صورت دوره‌ای مقادیر متریک‌ها را از برنامه‌های در حال اجرا دریافت می‌کنند.

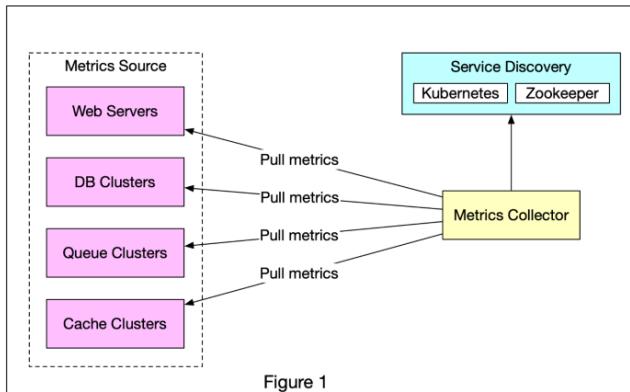


Figure 1

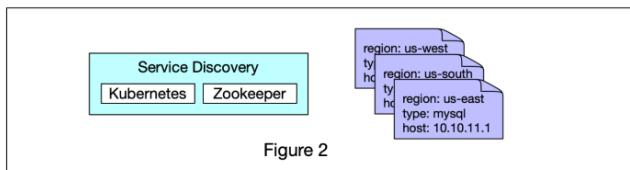


Figure 2

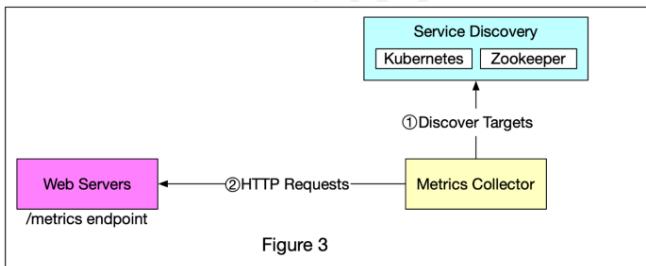


Figure 3

در این رویکرد، جمع‌کننده متريک باید لیست کامل نقاط انتهایی سرويس<sup>۱</sup> را که داده‌ها از آنها جمع‌آوری می‌شود را بداند. یک رویکرد ساده استفاده از یک فایل برای نگهداری اطلاعات DNS/IP برای هر نقطه انتهایی سرويس در سرورهای «جمع‌کننده متريک»<sup>۲</sup> است. در حالی‌که اين اидеه ساده است، اما نگهداری از اين رویکرد در محطي با مقیاس بزرگ که سرورها به طور مکرر اضافه یا حذف می‌شوند تا حدودی دشوار است و ما می‌خواهیم اطمینان حاصل کنیم که جمع‌کننده‌های متريک جمع‌آوری داده‌های متريک را از سرورهای جدید از دست ندهند.

<sup>1</sup> service endpoint<sup>2</sup> metric collector

خبر خوب این است که ما یک راه حل قابل اعتماد، مقیاس‌پذیر و قابل نگهداری از طریق Service Discovery ارائه شده توسط Kubernetes یا Zookeeper وغیره در اختیار داریم که در آن سرویس‌ها میزان دردسترس بودن<sup>۱</sup> خود را ثبت می‌کنند و هر زمان که لیست نقاط انتهایی سرویس تغییر کند، جمع‌کننده متریک می‌تواند توسط مؤلفه Service Discovery مطلع شود. Service Discovery شامل قوانین پیکربندی در مورد زمان و مکان جمع‌آوری معیارها است، همان‌طور که در شکل ۲ نشان داده شده است.

شکل ۳ مدل pull را به صورت جزئی توضیح می‌دهد.

۱. جمع‌کننده متریک که متادیتای<sup>۲</sup> مربوط به پیکربندی endpoint‌ها سرویس را از

دریافت می‌کند. متادیتا شامل فاصله زمانی pull، آدرس‌های

IP و پارامترهای زمان توقف و تلاش مجدد وغیره است.

۲. جمع‌کننده متریک داده‌های متریک را از طریق یک نقطه انتهایی HTTP از پیش

تعريف شده (برای مثال، /metrics) دریافت می‌کند. برای در معرض قراردادن نقطه

نهایی<sup>۳</sup>، معمولاً باید یک کتابخانه کلاینت به سرویس اضافه شود. در

شکل ۳ بیانگر سرویسی از وب سرورها است.

۳. (اختیاری) جمع‌کننده متریک یک نوتیفیکیشن رویداد<sup>۴</sup> تغییر را با

ثبت می‌کند تا هر زمان که نقاط انتهایی سرویس تغییر کند،

یک بهروزرسانی دریافت کند. از طرف دیگر، جمع‌کننده متریک می‌تواند به طور

دوره‌ای تغییرات نقطه پایانی<sup>۵</sup> را بررسی کند.

---

availability<sup>۱</sup>

Metadata<sup>۲</sup>

expose the endpoint<sup>۳</sup>

event notification<sup>۴</sup>

endpoint<sup>۵</sup>

## ذخیره‌سازی

### چرا یک درایو حالت جامد (SSD) سریع است؟

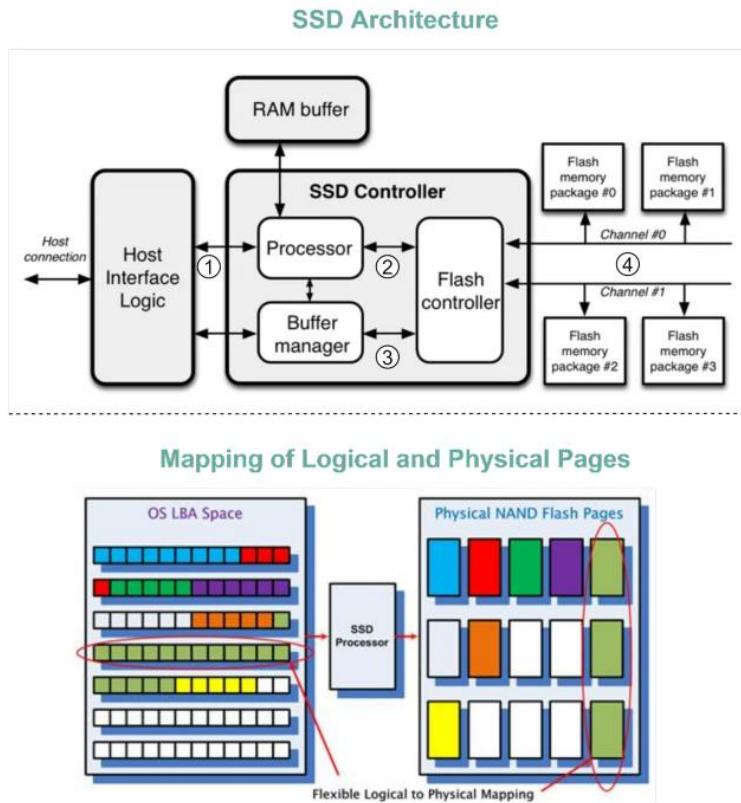
یک درایو حالت جامد<sup>۱</sup> در مقایسه با یک درایو دیسک سخت، داده‌ها را تا ۱۰ برابر سریع‌تر خوانده و تا ۲۰ برابر سریع‌تر می‌نویس [۱].

یک SSD یک دستگاه ذخیره‌سازی داده مبتنی بر حافظه فلاش است. بیت‌ها در سلول‌هایی که از ترانزیستورهای گیت شناور<sup>۲</sup> ساخته شده‌اند ذخیره می‌شوند. SSD‌ها کاملاً از اجزای الکترونیکی ساخته شده‌اند، هیچ قطعات متحرک یا مکانیکی مانند درایوهای سخت (HDD) ندارند. [۲]. نمودار زیر معماری SSD را نشان می‌دهد.

---

<sup>۱</sup> solid-state drive

<sup>۲</sup> floating-gate transistors



مرحله ۱: دستورات کاربر از طریق host interface می‌آیند [۲]. رابط می‌تواند سریال SATA یا PCI Express (PCIe)

مرحله ۲: پردازنده در کنترلر SSD دستورات را دریافت و به کنترلر فلش ارسال می‌کند. [۲]

مرحله ۳: SSDها همچنین حافظه RAM داخلی دارند، معمولاً برای اهداف کش و ذخیره اطلاعات نگاشت<sup>۱</sup> از آن استفاده می‌کنند. [۲]

مرحله ۴: بسته‌های حافظه فلش NAND در گروه‌ها و در چندین کانال سازماندهی شده‌اند.

نمودار دوم نحوه نگاشت صفحات منطقی و فیزیکی را نشان می‌دهد و چرا این معماری سریع است. کنترلر SSD از چندین قطعه FLASH را که به طور موازی کار می‌کند و باعث بهبود چشمگیر پهنه‌ای باند زیربنایی می‌شود، تشکیل شده است. هنگامی که نیاز به نوشتן بیش از یک صفحه (Page) داریم، کنترلر SSD می‌تواند آنها را به طور موازی بنویسد [۳]، در حالی که یک HDD یک head واحد دارد و تنها می‌تواند از یک head در یک زمان بخواند. هر بار که یک HOST Page نوشته می‌شود، کنترلر SSD یک صفحه فیزیکی برای نوشتן داده پیدا می‌کند و این نگاشت ثبت می‌شود. با این نگاشت، هر بار که HOST یک Page را می‌خواند، SSD می‌داند از کجا باید داده‌ها را از FLASH بخواند [۳].

سؤال - تفاوت‌های اصلی بین SSD و HDD چیست؟ اگر به معماری علاقه‌مند هستید، توصیه می‌کنم مقاله "Coding for SSDs" از Emmanuel Goossaert در منبع [۲] را بخوانید.

منابع:

[۱] SSD یا HDD: کدام یک برای شما مناسب است؟  
<https://www.avg.com/en/signal/ssd-hdd-which-is-best>

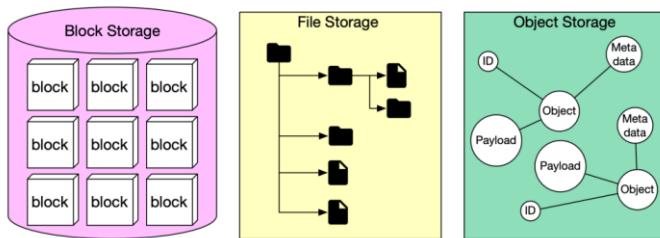
[۲] کدنویسی برای SSD ها:  
<https://codecapsule.com/2014/02/12>

## بررسی ذخیره‌سازی‌های مختلف

سیستم‌های ذخیره‌سازی به سه دسته کلی تقسیم می‌شوند:

- Block storage
- file storage
- object storage

نمودار زیر مقایسه سیستم‌های ذخیره‌سازی مختلف را نشان می‌دهد.



شکل ۱-۲۴

در جدول زیر به مقایسه این سه نوع ذخیره‌ساز پرداخته شده است:

	Block storage	File storage	Object storage
Mutable Content	Y	Y	N (object versioning is supported, in-place update is not)
Cost	High	Medium to high	Low
Performance	Medium to high, very high	Medium to high	Low to medium
Consistency	Strong consistency	Strong consistency	Strong consistency
Data access	SAS/iSCSI/FC	Standard file access, CIFS/SMB, and NFS	RESTful API
Scalability	Medium scalability	High scalability	Vast scalability
Good for	Virtual machines (VM), high-performance applications like database	General-purpose file system access	Binary data, unstructured data

جدول ۱-۱

## ذخیره‌ساز بلوکی (Block Storage)

ذخیره‌سازی بلوکی<sup>۱</sup> اولین نوع ذخیره‌سازهایی است که در دهه ۱۹۶۰ معرفی شد. ابزارهای ذخیره‌سازی رایج مانند هارددیسک (HDD) و درایو حالت جامد (SSD) که به صورت فیزیکی به سرورها متصل می‌شوند، همگی به عنوان ذخیره‌سازی بلوکی شناخته می‌شوند. ذخیره‌سازی بلوکی، بلوک‌های خام را به صورت یک حجم<sup>۲</sup> به سرور ارائه می‌دهد. این انعطاف‌پذیرترین و کارآمدترین شکل ذخیره‌سازی است. سرور می‌تواند بلوک‌های خام را فرمت‌بندی کند و از آن‌ها به عنوان یک سیستم فایل استفاده نماید یا اینکه کنترل این بلوک‌ها را به یک برنامه واگذار کند. برخی از برنامه‌ها مانند پایگاه‌داده یا ماشین مجازی این بلوک‌ها را به صورت مستقیم مدیریت می‌کنند تا حداکثر کارایی را از آن‌ها به دست آورند. ذخیره‌سازی بلوکی محدود به ابزارهای فیزیکی متصل به سرور نیست. این نوع ذخیره‌سازی می‌تواند از طریق یک شبکه پرسرعت یا پروتکل‌های اتصال استاندارد صنعتی مانند فیبر کانال (FC) و iSCSI به سرور متصل شود. از لحاظ مفهومی، ذخیره‌سازی بلوکی متصل به شبکه نیز همچنان بلوک‌های خام را ارائه می‌دهد. برای سرورها، این روش مشابه با ذخیره‌سازی بلوکی متصل به صورت فیزیکی عمل می‌کند. چه به شبکه متصل باشد و چه به صورت فیزیکی به سیستم متصل باشند، ذخیره‌سازی بلوکی به طور کامل در اختیار یک سرور واحد قرار دارد و یک منبع اشتراکی به حساب نمی‌آید. همین‌طور Block Storage از متادیتا محدود استفاده می‌کند؛ اما برای عملیات خواندن/نوشتن بر روی شناسه‌های منحصر به فرد اختصاص‌داده شده به هر بلوک تکیه می‌کند. این کاهش اضافه‌بار انتقال داده را کاهش می‌دهد و به سرور اجازه می‌دهد به طور کارآمد به داده‌ها در ذخیره‌سازی بلوک دسترسی پیدا کند و آنها را بازیابی کند. به دلیل محدود بودن متادیتا ذخیره‌سازی بلوک، این نوع ذخیره‌سازی تأخیر فوق العاده کمی را که برای بارهای کاری که به عملکرد بالا نیاز دارند، سرعت مناسبی را ارائه می‌دهد. این برای برنامه‌های حساس به تأخیر مانند پایگاه‌های داده لازم است.

1 Block Storage  
2 volume

## ذخیره‌سازی شیء (Object storage)

ذخیره‌سازی شیء<sup>۱</sup> که ذخیره‌ساز حبابی یا Blob Storage نیز نامیده می‌شود، یک فناوری نسبتاً جدید است. این نوع از ذخیره‌سازی با اولویت قراردادن دوام (نگهداری ایمن اطلاعات برای مدت طولانی)، مقیاس‌پذیری عظیم و هزینه پایین، به طور عمده از کارایی (سرعت) صرف‌نظر می‌کند. این نوع ذخیره‌سازی برای داده‌هایی که زیاد مورد استفاده قرار نگرفته (به‌اصطلاح داده‌های سرد) طراحی شده و عمدهاً برای بایگانی و پشتیبان‌گیری مورد استفاده قرار می‌گیرد. در ذخیره‌سازی شیء، تمامی اطلاعات به صورت شیء (Object) در یک ساختار تخت (بدون سلسله‌مراتب) ذخیره می‌شوند. در نتیجه، برخلاف روش‌های سنتی، خبری از ساختار درختی پوشیده‌ها نیست. دسترسی به داده‌ها عموماً از طریق یک رابط برنامه‌نویسی (RESTful API) فراهم می‌شود. سرعت ذخیره‌سازی شیء نسبت به سایر انواع روش‌های ذخیره‌سازی پایین‌تر است. اکثر ارائه‌دهندگان خدمات ابری عمومی، سرویس ذخیره‌سازی object را ارائه می‌دهند، مانند سرویس S3 شرکت آمازون (AWS)، سرویس ذخیره‌سازی Azure Blob بلوکی گوگل (Google Block Storage) و سرویس ذخیره‌سازی object آژور (Azure Storage).

## ذخیره‌سازی فایل (File storage)

ذخیره‌سازی فایل<sup>۲</sup> روی بستر ذخیره‌سازی بلوکی بنا شده است. این نوع از ذخیره‌سازی، سطح بالاتری از انتزاع<sup>۳</sup> را برای مدیریت آسان‌تر فایل‌ها و پوشیده‌ها ارائه می‌دهد. در این روش، داده‌ها به عنوان فایل تحت یک ساختار سلسله‌مراتبی از پوشیده‌ها ذخیره می‌شوند. ذخیره‌سازی فایل، متداول‌ترین راهکار ذخیره‌سازی برای مقاصد عمومی است. این نوع از ذخیره‌سازی را می‌توان با استفاده از پروتکل‌های رایج شبکه‌ای در سطح فایل، مانند NFS و SMB/CIFS و را برای تعداد زیادی از سرورها قابل دسترسی کرد. سروورهایی که به ذخیره‌سازی فایل دسترسی پیدا

1 Object Storage

2 File Storage

3 abstraction

می‌کنند، نیازی به مدیریت پیچیدگی‌های مربوط به بلوک‌ها، فرمتبندی حجم و غیره ندارند. سادگی ذاتی ذخیره‌سازی فایل، آن را به راه حلی ایده‌آل برای اشتراک‌گذاری تعداد زیادی از فایل‌ها و پوشش‌ها درون یک سازمان تبدیل می‌کند.

### **حجم (Volumes)**

حجم (Volume) مقدار ثابتی از فضای ذخیره‌سازی روی یک دیسک یا نوار است. اصطلاح Volume اغلب به عنوان مترادف با خود ذخیره‌سازی استفاده می‌شود، اما ممکن است یک دیسک به تنها‌ی حاوی بیش از یک Volume باشد یا یک Volume روی بیش از یک دیسک قرار گیرد.

### **ذخیره‌ساز متصل به شبکه (NAS)**

یک (NAS) دستگاهی برای ذخیره‌سازی است که به شبکه متصل می‌شود و امکان ذخیره و بازیابی داده‌ها از یک مکان مرکزی برای کاربران مجاز شبکه را فراهم می‌کند. دستگاه‌های NAS انعطاف‌پذیر هستند، به این معنی که با نیاز به فضای ذخیره‌سازی بیشتر، می‌توانیم به حجم موجود اضافه کنیم. این سیستم سریع‌تر و کم‌هزینه‌تر از روش‌های دیگر است و تمام مزایای یک Public Cloud را در محل ارائه می‌دهد و به ما کنترل کاملی می‌دهد.

### **سیستم فایل توزیع شده (HDFS) Hadoop**

سیستم فایل توزیع شده هادوپ (HDFS - Hadoop Distributed File System) یک سیستم فایل توزیع شده است که برای اجرا روی سخت‌افزار معمولی طراحی شده است. HDFS دارای تحمل خطای بالایی است و برای استقرار روی سخت‌افزار کم‌هزینه طراحی شده است. HDFS دسترسی با توان عملیاتی بالا به داده‌های برنامه را فراهم می‌کند و برای برنامه‌هایی که مجموعه‌داده‌های بزرگی دارند مناسب است. این سیستم شباهت‌های زیادی با سیستم‌های فایل توزیع شده موجود دارد.

HDFS برای ذخیره‌سازی این فایل‌های بسیار بزرگ در سراسر ماشین‌ها در یک خوشه (cluster) بزرگ طراحی شده است. هر فایل به عنوان یک توالی از بلوک‌ها ذخیره می‌شود، همه بلوک‌های یک فایل به جز آخرین بلوک دارای اندازه یکسان هستند. بلوک‌های یک فایل برای تحمل خطا تکثیر می‌شوند.

### آرایه افزونگی دیسک‌های مستقل (RAID)

روشی برای ذخیره داده‌های یکسان RAID (Redundant Array of Independent Disks) روی چندین هارد دیسک یا درایو حالت جامد (SSD) برای محافظت از داده‌ها در صورت خرابی درایو است.

با این حال، سطوح مختلف RAID وجود دارد و همه آنها هدف ارائه افزونگی<sup>۱</sup> را ندارند. باید برخی از سطوح RAID را بررسی کنیم:

- RAID 0: همچنین به عنوان striping شناخته می‌شود، داده‌ها به طور مساوی در همه درایوهای آرایه تقسیم می‌شوند.
- RAID 1: همچنین به عنوان mirroring شناخته می‌شود، حداقل دو درایو حاوی یک کپی دقیق از مجموعه داده‌ها هستند. اگر یک درایو خراب شود، سایرین همچنان کار می‌کنند.
- RAID 5: RAID با parity striping. این مورد نیاز به استفاده از حداقل ۳ درایو دارد، داده‌ها در چندین درایو مانند RAID 0 دارد اما همچنین دارای برابری توزیع شده در سراسر درایوها نیز است.
- RAID 6: RAID با برابری دوگانه. RAID 5 شبیه ۶ است، اما داده‌های برابری روی دو درایو نوشته می‌شوند.

- RAID 1 و RAID 0: ترکیبی از striping به همراه mirroring از striping با روی داده ها روی درایو های ثانویه در عین حال استفاده از striping همه داده ها را در مجموعه از درایو ها برای افزایش سرعت انتقال داده، امنیت را فراهم می کند.

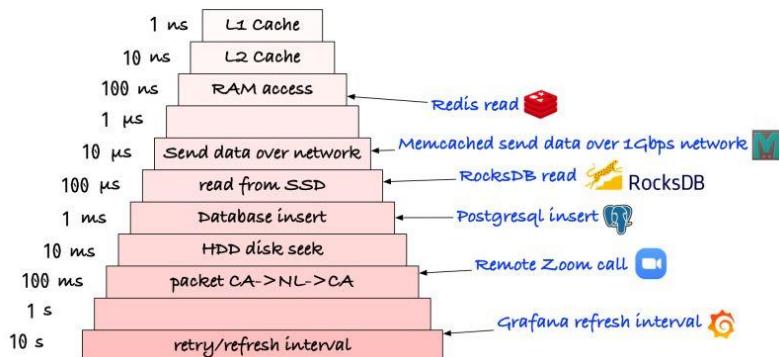
Features	RAID 0	RAID 1	RAID 5	RAID 6	RAID 10
Description	Striping	Mirroring	Striping with Parity	Striping with double parity	Striping and Mirroring
Minimum Disks	2	2	3	4	4
Read Performance	High	High	High	High	High
Write Performance	High	Medium	High	High	Medium
Cost	Low	High	Low	Low	High
Fault Tolerance	None	Single-drive failure	Single-drive failure	Two-drive failure	Up to one disk failure in each sub-array
Capacity Utilization	100%	50%	67%-94%	50%-80%	50%

## اعداد تأخیر که باید بدانید.

توجه داشته باشید که این اعداد کاملاً دقیق نیستند. آنها بر اساس برخی معیارهای آنلاین اعداد تأخیر Jeff Dean + برخی منابع دیگر تعیین شده‌اند.

### Latency Numbers You Should Know

ByteByteGo.com



## کش‌های L1 حدود ۱ نانوثانیه و L2 در حدود ۱۰ نانوثانیه

برای مثال: این کش‌ها معمولاً روی تراشه ریزپردازنده ساخته می‌شوند. مگر اینکه به طور مستقیم با سخت‌افزار کار کنند، احتمالاً نیازی به نگرانی در مورد آنها ندارید.

## دسترسی به RAM در حدود ۱۰۰ نانوثانیه

برای مثال: حدود ۱۰۰ نانوثانیه طول می‌کشد تا داده‌ها از حافظه خوانده شوند. Redis یک «ذخیره‌گاه داده‌ی درون حافظه» است، بنابراین خواندن داده از Redis حدود ۱۰۰ نانوثانیه طول می‌کشد.

ارسال ۱ کیلوبایت داده از طریق شبکه با سرعت ۱ گیگابیت بر ثانیه: حدود ۱۰ میکروثانیه  
برای مثال: حدود ۱۰ میکروثانیه طول می‌کشد تا ۱ کیلوبایت داده از Memcached از طریق شبکه ارسال شود.

<sup>۱</sup> in-memory data store

### خواندن از SSD حدود ۱۰۰ میکروثانیه

برای مثال: RocksDB یک ذخیره‌گاه کلید - مقدار یا در اصطلاح K/V مبتنی بر دیسک است، بنابراین تأخیر خواندن روی SSD حدود ۱۰۰ میکروثانیه است.

### عملیات درج در پایگاهداده: حدود ۱ میلی ثانیه

برای مثال: Commit در PostgreSQL ممکن است ۱ میلی ثانیه طول بکشد. پایگاهداده باید اداده‌ها را ذخیره کند، فهرست (index) ایجاد کند و لاغ‌ها را پاک کند. همه این اقدامات زمان بر هستند.

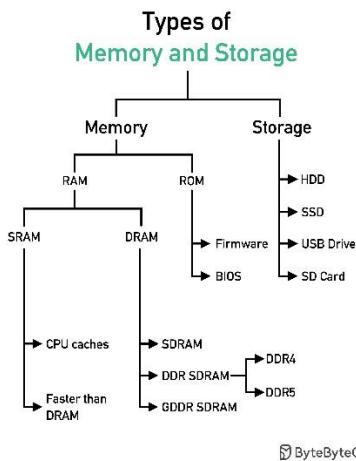
### ارسال packet از کانادا به هلند و سپس بازگشت به کانادا: ۱۰۰ میلی ثانیه

برای مثال: اگر یک تماس تصویری با مسافت طولانی در Zoom داشته باشیم، تأخیر ممکن است حدود ۱۰۰ میلی ثانیه باشد.

### تکرار داخلی یا به روزرسانی: ۱ تا ۱۰ ثانیه

برای مثال: در یک سیستم مانیتورینگ، فاصله زمانی به روزرسانی معمولاً روزی ۵ تا ۱۰ ثانیه تنظیم می‌شود (مقدار پیش‌فرض در Grafana).

## انواع حافظه و ذخیره‌سازی



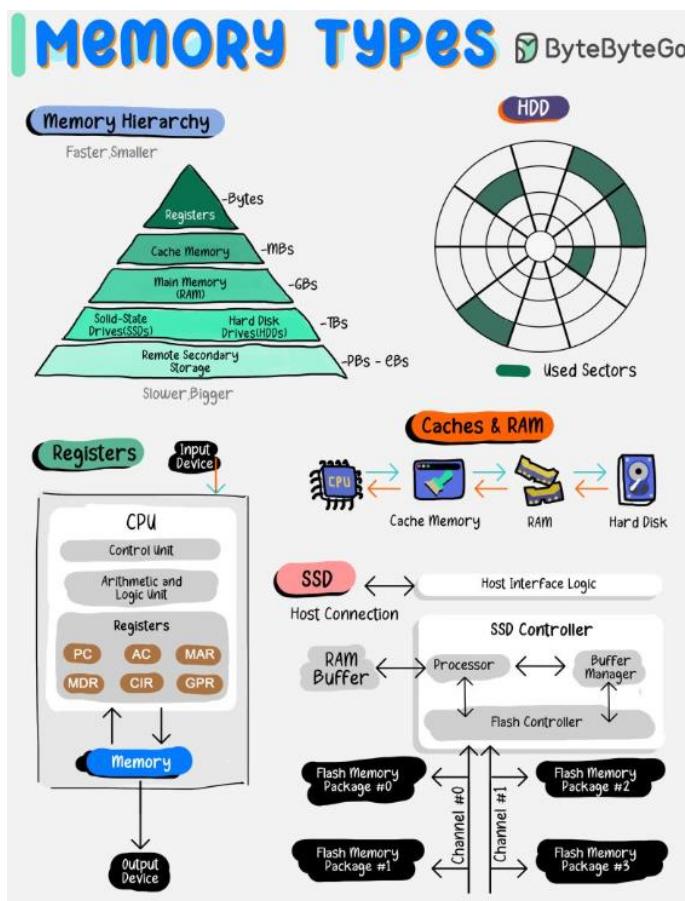
### انواع حافظه را می‌شناسید؟

حافظه‌ها از نظر سرعت، اندازه و عملکرد متفاوت هستند و یک معماری چندلایه ایجاد می‌کنند که هزینه را با نیاز به دسترسی سریع به داده متعادل می‌کند. با درک نقش‌ها و قابلیت‌های هر نوع حافظه، توسعه‌دهندگان و معماران سیستم می‌توانند سیستم‌هایی را طراحی کنند که به طور مؤثر از نقاط قوت هر لایه ذخیره‌سازی استفاده کنند و در نتیجه عملکرد کلی سیستم و تجربه کاربری بهبود یابد.

برخی از انواع رایج حافظه عبارت‌اند از:

۱. رجیسترها: ذخیره‌سازی بسیار کوچک و فوق العاده سریع درون CPU برای دسترسی فوری به داده‌ها.
۲. حافظه کش (Cache): حافظه کوچک و سریعی که در نزدیکی CPU قرار دارد تا سرعت بازیابی داده را افزایش دهد.
۳. حافظه اصلی (RAM): ذخیره‌سازی اولیه و بزرگ‌تر برای برنامه‌ها و داده‌های در حال اجرا.

۴. درایوهای حالت جامد (SSD): ذخیره‌سازی سریع و قابل اعتماد بدون قطعات متحرک که برای داده‌های دائمی استفاده می‌شود.
۵. درایوهای دیسک سخت (HDD): هارد دیسک‌ها درایوهای مکانیکی با ظرفیت بالا برای ذخیره‌سازی بلندمدت هستند.
۶. حافظه ثانویه از راه دور: ذخیره‌سازی خارج از سایت برای پشتیبان‌گیری و باگانی داده‌ها که از طریق شبکه قابل دسترسی است.



## S3 طراحی

قبل از اینکه به طراحی بپردازیم، اجازه دهید برخی از اصطلاحات را تعریف کنیم.

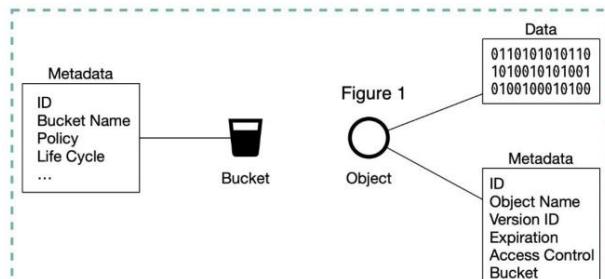
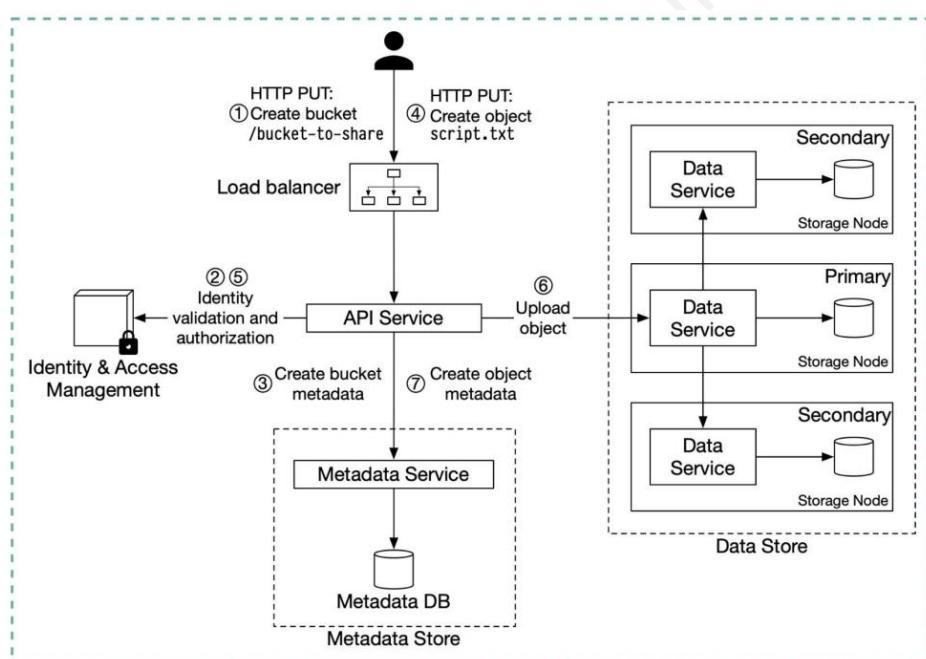


Figure 1



چه اتفاقی می‌افتد وقتی یک فایل را به Amazon S3 آپلود می‌کنید؟ بباید یک سیستم object storage مانند S3 طراحی کنیم. قبل از اینکه به طراحی پردازیم، اجازه دهید برخی از اصطلاحات را تعریف کنیم.

پیمانه<sup>۱</sup>. یک پیمانه منطقی برای Object است. نام پیمانه به صورت سراسری منحصر به فرد است. برای بارگذاری داده‌ها به S3، ابتدا باید یک پیمانه ایجاد کنیم.

یک Object یک قطعه داده فردی است که ما در یک پیمانه ذخیره می‌کنیم که حاوی داده‌های شیء<sup>۲</sup> (که به آن محموله<sup>۳</sup> تیز گفته می‌شود) و متادیتا است. داده‌های Object می‌تواند هر توالی از بایت‌هایی باشد که می‌خواهیم ذخیره کنیم. متادیتا یک مجموعه از جفت‌های key-value است که Object را توصیف می‌کنند.

- یک S3 Object شامل موارد زیر است (شکل ۱):
  - Metadata: این گزینه قابل تغییر است و حاوی ویژگی‌هایی مانند ID، نام bucket.
  - نام Object و غیره است.
  - Object data: این مورد غیرقابل تغییر است و داده‌های واقعی را شامل می‌شود.

در S3، یک Object در یک پیمانه ساکن است. مسیر به این صورت است: /bucket-to-share/script.txt/. پیمانه فقط متادیتا دارد. Object دارای متادیتا و داده‌های واقعی است.

**Bucket<sup>۱</sup>**

Object data<sup>۲</sup>

payload<sup>۳</sup>

در نمودار زیر (شکل ۲) نحوه بارگذاری فایل توضیح داده شده است. در این مثال، ابتدا یک پیمانه با نام "bucket-to-share" ایجاد می‌کنیم و سپس فایلی با نام "script.txt" را در پیمانه بارگذاری می‌کنیم.

۱. کلاینت یک درخواست HTTP PUT برای ایجاد پیمانه با نام

"bucket-to-share" ارسال می‌کند. این درخواست به سرویس API ارسال

می‌شود.

۲. سرویس API سرویس مدیریت هویت و دسترسی (IAM) را برای اطمینان از مجاز بودن کاربر و داشتن اجازه نوشتن فراخوانی می‌کند.

۳. سرویس API با ذخیرهساز متاداده برای ایجاد یک ورودی با اطلاعات پیمانه در پایگاهداده متاداده تماس می‌گیرد. پس از ایجاد ورودی، پیام موفقیت به کلاینت ارسال می‌شود.

۴. پس از ایجاد پیمانه، کلاینت یک درخواست HTTP PUT برای ایجاد یک Object با نام "script.txt" ارسال می‌کند.

۵. سرویس API هویت کاربر را تأیید می‌کند و اطمینان می‌دهد که کاربر مجوز نوشتن در پیمانه را دارد.

۶. پس از موفقیت در اعتبارسنجی، سرویس API داده‌های Object را در محموله HTTP PUT به پیمانه داده ارسال می‌کند. ذخیرهساز داده‌ها محموله را به عنوان یک شیء ذخیره می‌کند و آن Object را بازمی‌گرداند.

۷. سرویس API با ذخیرهساز متاداده برای ایجاد یک ورودی جدید در پایگاهداده متاداده تماس می‌گیرد. این ورودی حاوی متاداده مهمی مانند object\_name (Object) مربوط به bucket\_id (UUID) و object\_id (UUID) است.

## بارگذاری فایل‌های بزرگ

چگونه می‌توانیم زمانی که فایل‌های بزرگ را در سرویس ذخیره‌سازی شیء مانند S3 بارگذاری می‌کنیم، عملکرد را بهینه کنیم؟

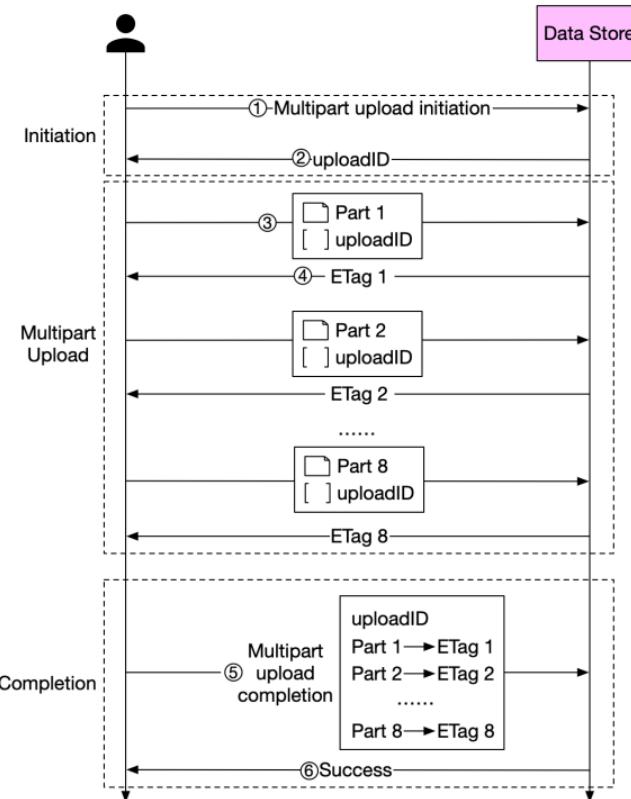
قبل از پاسخ به این سؤال، بباید ببینیم چرا نیاز داریم این فرایند را بهینه کنیم. برخی از فایل‌ها ممکن است بزرگ‌تر از چند گیگابایت باشند. امکان بارگذاری مستقیم چنین شیء بزرگی وجود دارد، اما ممکن است زمان زیادی طول بکشد. اگر در میانه بارگذاری، اتصال شبکه قطع شود، باید از ابتدا شروع کنیم. راه حل بهتری این است که شیء بزرگ را به قطعات کوچک‌تر تقسیم کنیم و آنها را به صورت مستقل بارگذاری کنیم. پس از بارگذاری تمام قطعات، ذخیره‌گاه شیء<sup>۱</sup> تمامی اشیاء<sup>۲</sup> را دوباره جمع می‌کند. این فرایند بارگذاری چندبخشی<sup>۳</sup> نامیده می‌شود.

نمودار زیر نشان می‌دهد که بارگذاری چندبخشی چگونه کار می‌کند:

object store<sup>۱</sup>

objects<sup>۲</sup>

multipart upload<sup>۳</sup>



۱. کلاینت با object storage ارتباط می‌گیرد تا یک بارگذاری چندبخشی را آغاز کند.
۲. ذخیره‌گاه داده یک uploadID را بازمی‌گرداند که بارگذاری را به طور منحصر به فرد شناسایی می‌کند.
۳. کلاینت فایل بزرگ را به شیوه‌های کوچک‌تر تقسیم می‌کند و شروع به بارگذاری می‌کند. فرض کنید اندازه فایل ۱.۶ گیگابایت است و کلاینت آن را به ۸ قسمت تقسیم می‌کند، بنابراین هر قسمت ۲۰۰ مگابایت است. کلاینت قسمت اول را همراه با uploadID دریافتی در مرحله ۲ به ذخیره‌گاه داده بارگذاری می‌کند.

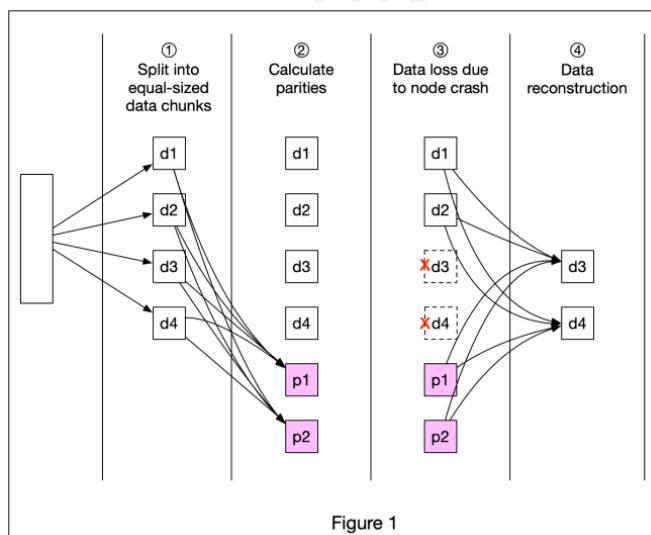
۴. هنگامی که یک قسمت بارگذاری می شود، ذخیره گاه داده یک ETag بازمی گرداند که در واقع md5 checksum آن قسمت است. این برای تأیید بارگذاری های چندبخشی استفاده می شود.

۵. پس از بارگذاری تمام قطعات، کلاینت یک درخواست تکمیل بارگذاری چندبخشی را ارسال می کند که شامل uploadID، شماره قطعات و ETag ها است.

۶. ذخیره گاه داده object را از قطعات آن بر اساس شماره قطعه دوباره جمع می کند. از آنجایی که object واقعاً بزرگ است، این فرایند ممکن است چند دقیقه طول بکشد. پس از تکمیل جمع آوری، یک پیام موفقیت را به کلاینت بازمی گرداند.

## کد تصحیح خطای (Erasure coding)

یک تکنیک بسیار عالی که معمولاً در ذخیره‌سازی اشیاء<sup>۱</sup> مانند S3 برای بهبود ماندگاری<sup>۲</sup> استفاده می‌شود، رمزگذاری محو نامیده می‌شود. بیایید نگاهی بیندازیم که چگونه کار می‌کند. کدگذاری تصحیح خطای<sup>۳</sup> تعاملی با ماندگاری داده به شیوه‌ای متفاوت از تکثیر<sup>۴</sup> داده‌ها دارد. این روش داده‌ها را به قطعات کوچک‌تر (قرار داده شده روی سرورهای مختلف) تقسیم می‌کند و افزونگی برابر ایجاد می‌کند. در صورت خرابی، می‌توانیم از داده‌هایی که چند قطعه شده از نظر اندازه برابر هم‌دیگر هستند برای بازسازی داده‌ها استفاده کنیم. بیایید نگاهی به یک مثال عینی (کدگذاری تصحیح خطای ۴ + ۲) بیندازیم که در شکل ۱ نشان داده شده است.



---

object storage<sup>۱</sup>

<sup>۲</sup> ماندگاری یا دوام به توانایی یک محفصه‌ول فیزیکی برای کارکردن بدون نیاز به نگهداری یا تعمیر بیش از اندازه در هنگام رویارویی با چالش‌های عملکرد عادی در طول عمر طراحی آن است.

Erasure coding<sup>۳</sup>

replication<sup>۴</sup>

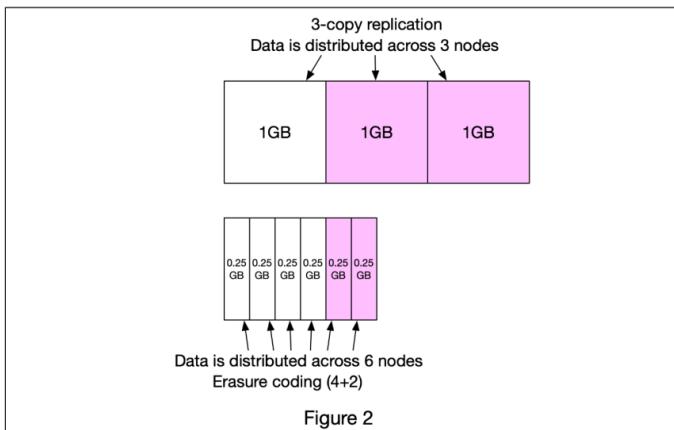


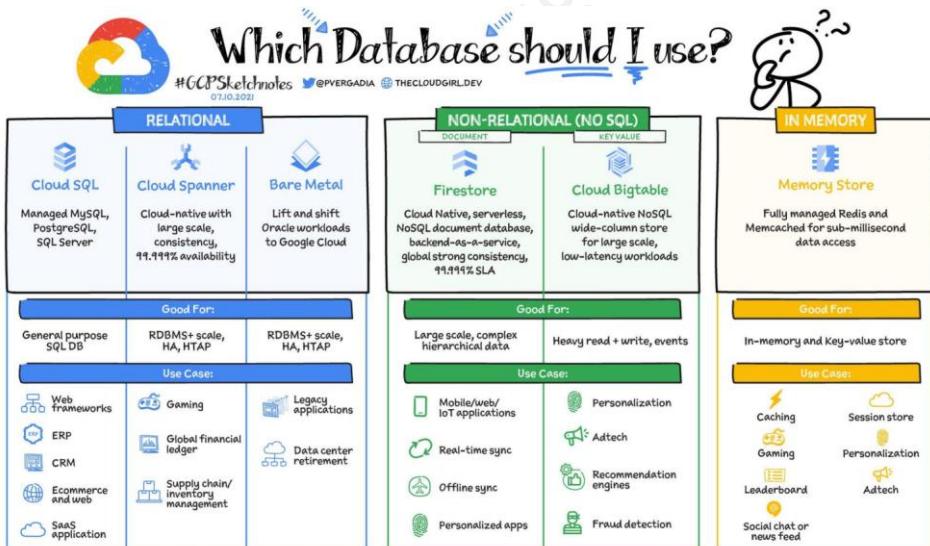
Figure 2

۱. داده‌ها به چهار تکه داده به اندازه برابر  $d_1$ ,  $d_2$ ,  $d_3$  و  $d_4$  تقسیم می‌شوند.
  ۲. از فرمول ریاضی برای محاسبه برابری  $p_1$  و  $p_2$  استفاده می‌شود. برای ارائه یک مثال بسیار ساده برابر است با:  $p_1 = d_1 + 2*d_2 - d_3 + 4*d_4$  و  $p_2 = -d_1 + 5*d_2 + d_3 - 3*d_4$ .
  ۳. داده‌های  $d_3$  و  $d_4$  به دلیل خرابی گره از دست رفته‌اند.
  ۴. از فرمول‌های ریاضی برای بازسازی داده‌های از دست رفته  $d_3$  و  $d_4$  با استفاده از مقادیر شناخته شده  $d_1$ ,  $d_2$ ,  $p_1$  و  $p_2$  استفاده می‌شود.
- کدگذاری تصحیح خطأ به چه مقدار فضای اضافی نیاز دارد؟ برای هر دو تکه داده، به یک بلوک برابری نیاز داریم، بنابراین سربار ذخیره‌سازی ۵۰٪ است (شکل ۲). در حالی که در تکثیرسازی ۳ کپی، سربار ذخیره‌سازی ۲۰٪ است (شکل ۲).
- آیا کدگذاری تصحیح خطأ ماندگاری داده‌ها را افزایش می‌دهد؟ فرض کنیم یک گره نرخ خرابی سالانه‌ای حدود ۸۱٪ دارد. بر اساس محاسبه انجام شده توسط Backblaze، کدگذاری تصحیح خطأ می‌تواند ۱۱ رقم ماندگاری داده‌ها را در مقابل تکثیر ۳ کپی که می‌تواند ۶ رقم ماندگاری را فراهم کند را به دست آورد.

## بررسی پایگاهداده

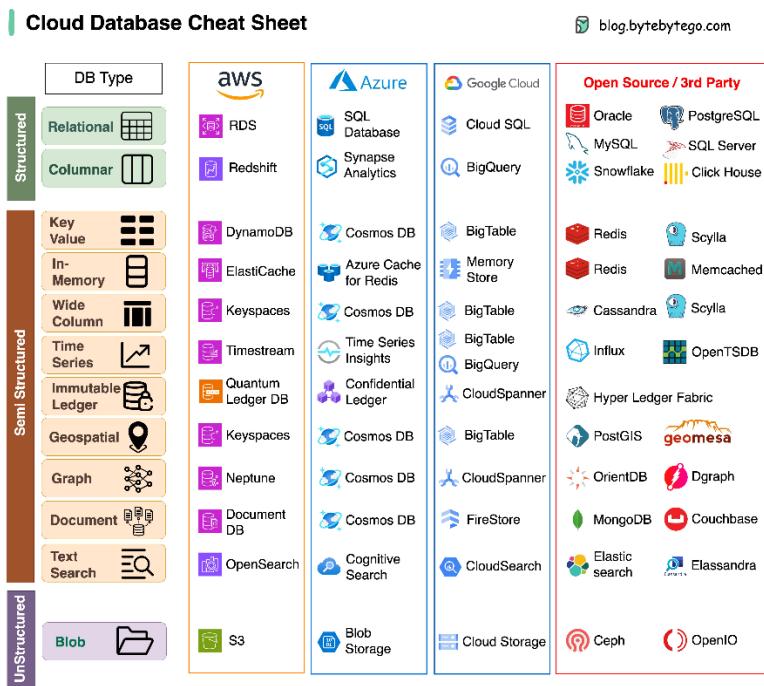
### چه پایگاهداده‌ای را باید استفاده کنم؟

این یکی از مهم‌ترین سوالاتی است که معمولاً باید در یک مصاحبه به آن پاسخ دهیم. انتخاب پایگاهداده مناسب کار سختی است. Google Cloud اخیراً یک مقاله عالی منتشر کرد که در آن گزینه‌های مختلف پایگاهداده موجود در Google Cloud را خلاصه کرده و توضیح داده است که کدام موارد استفاده برای هر گزینه پایگاهداده مناسب‌تر است.



## پایگاهداده‌های مختلف در سرویس‌های ابری

انتخاب پایگاهداده مناسب برای پروژه شما یک کار پیچیده است. بسیاری از گزینه‌های پایگاهداده که هر کدام برای موارد استفاده متمایز مناسب هستند، می‌توانند به سرعت منجر به خستگی در تصمیم‌گیری شوند.

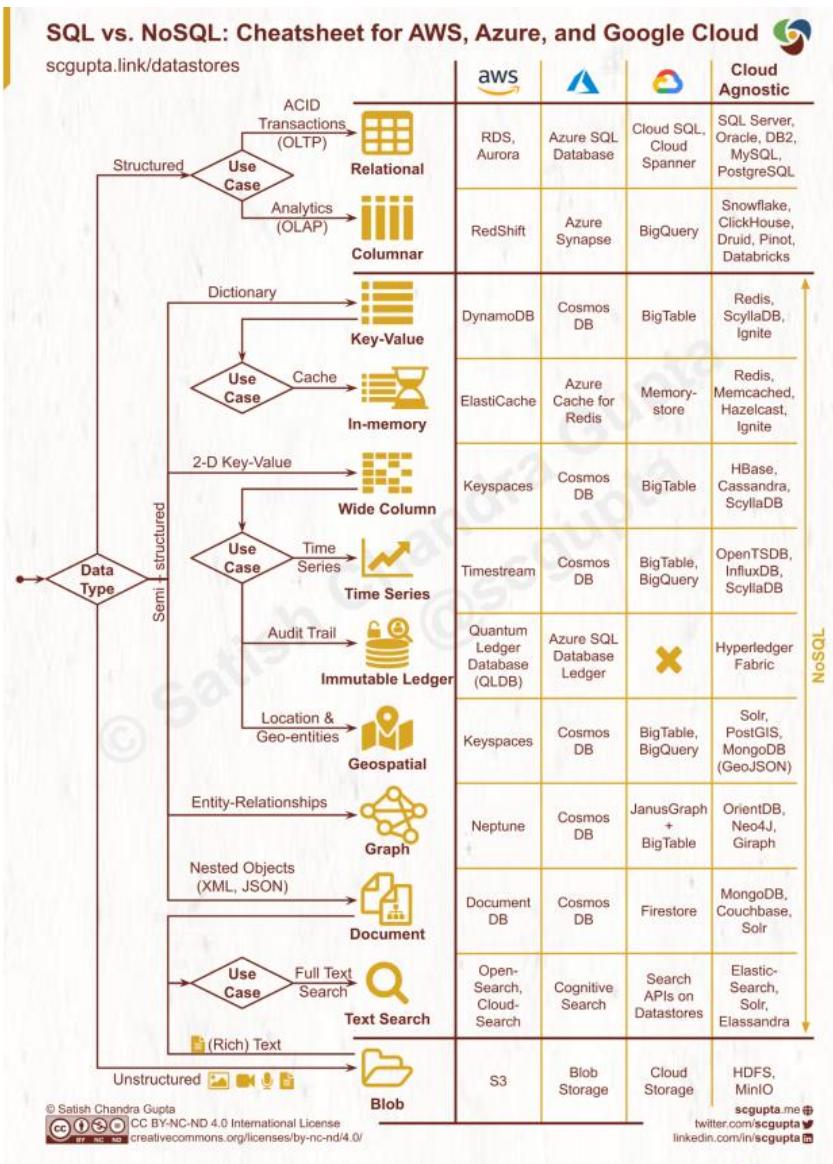


امیدواریم این متن، جهت‌گیری کلی را برای شناسایی سرویس مناسب که با نیازهای پروژه شما مطابقت دارد و از ایجاد مشکل جلوگیری کند.

توجه: گوگل مستندات محدودی برای موارد استفاده از پایگاهداده خود دارد. با اینکه ما تمام تلاش خود را برای بررسی آنچه در دسترس بود انجام دادیم و به بهترین گزینه رسیدیم، ممکن است برخی از ورودی‌ها نیاز به دقت بیشتری داشته باشند.

## راهنمای انتخاب پایگاهداده مناسب

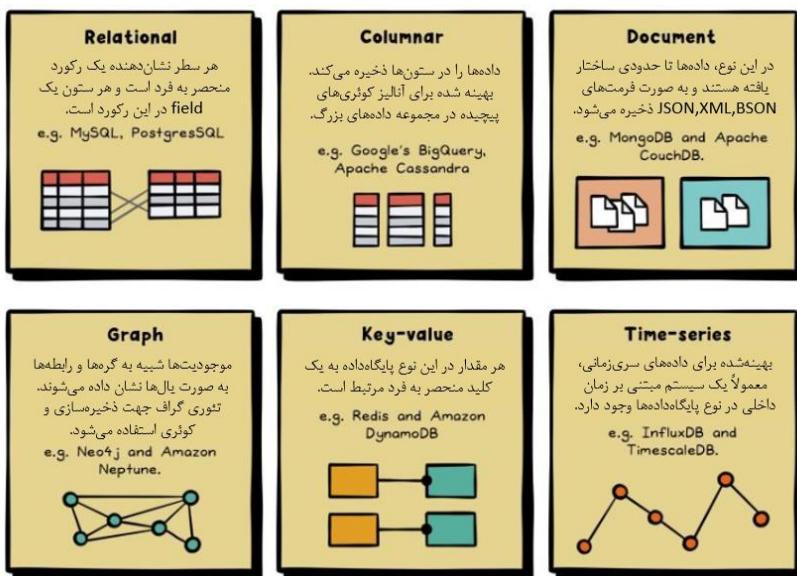
انتخاب یک پایگاهداده تعهدی بلندمدت است، بنابراین تصمیم‌گیری نباید به سادگی صورت گیرد. نکته مهمی که باید به خاطر داشت این است که پایگاهداده مناسب را برای کار مناسب انتخاب کنید.



داده‌ها می‌توانند ساختار یافته (طرح جدول SQL)، نیمه‌ساختار یافته (JSON، XML و غیره) و بدون ساختار (Blob) باشند.

## Types of Databases

by levelupcoding.co



دسته‌بندی‌های رایج پایگاه‌داده‌ها شامل:

رابطه‌ای، ستونی، کلید - مقدار، در حافظه<sup>۱</sup>، ستون گسترده<sup>۲</sup>، سری زمانی، دفترچه ثبت

غیرقابل تغییر<sup>۳</sup>، جغرافیایی، گراف، سندگرا، جستجوی متن<sup>۴</sup>، Blob

In memory<sup>۱</sup>

Wide column<sup>۲</sup>

Immutable ledger<sup>۳</sup>

Text search<sup>۴</sup>

## پایگاهداده‌های SQL

پایگاهداده‌های SQL (رابطه‌ای) مجموعه‌ای از آیتم‌های داده‌ای با روابط از پیش تعریف شده بین آن‌ها است. این آیتم‌ها به صورت مجموعه‌ای از جداول با ستون‌ها و ردیف‌ها سازماندهی می‌شوند. جدول‌ها برای نگهداری اطلاعات در مورد شیء/*Object*‌هایی که باید در پایگاهداده نشان داده شوند، استفاده می‌شوند. هر ستون در یک جدول نوع خاصی از داده را نگه می‌دارد و یک فیلد مقدار واقعی یک ویژگی را ذخیره می‌کند. ردیف‌های جدول مجموعه‌ای از مقادیر مرتبط با یک Object یا موجودیت را نشان می‌دهند.

هر ردیف در یک جدول می‌تواند با یک شناسه منحصر به فرد به نام کلید اصلی مشخص شود و ردیف‌ها بین جداول متعدد را می‌توان با استفاده از کلیدهای خارجی مرتبط کرد. به این داده‌ها می‌توان از طرق مختلف بدون سازماندهی مجدد جداول پایگاهداده دسترسی داشت. پایگاه‌های داده SQL معمولاً از مدل یکپارچگی<sup>۱</sup> ACID پیروی می‌کنند.

## نمایه‌های مادی (Materialized Views)

یک Materialized Views یک مجموعه‌داده پیش محاسبه شده<sup>۲</sup> است که از یک کوئری ویژه بدست‌آمده و برای استفاده بعدی ذخیره می‌شود. از آنجایی که داده‌ها از قبل محاسبه شده‌اند، کوئری‌ها یک Materialized Views سریع‌تر از اجرای یک کوئری در جدول پایه مربوط به view است. این تفاوت عملکرد زمانی که یک کوئری به طور مکرر اجرا می‌شود یا به اندازه کافی پیچیده باشد، می‌تواند قابل توجه باشد.

<sup>۱</sup> در ادامه همین فصل توضیح داده می‌شود.

<sup>۲</sup> pre-computed

همچنین این کار زیرمجموعه داده‌ها<sup>۱</sup> را فعال می‌کند و عملکرد کوئری‌های پیچیده را که روی مجموعه داده‌های بزرگ اجرا می‌شوند را بهبود می‌بخشد که در نتیجه باعث کاهش بار روی شبکه می‌شود. کاربردهای دیگری برای Materialized Views وجود دارد، اما معمولاً برای بهبود کارایی<sup>۲</sup> و تکثیر<sup>۳</sup> استفاده می‌شوند.

### N+1 مسئله کوئری

مشکل کوئری N+1 زمانی رخ می‌دهد که لایه دسترسی به داده N دستور SQL اضافی را برای دریافت همان داده‌ای که هنگام اجرای کوئری اصلی SQL قابل بازیابی بوده است را اجرا می‌کند. هرچه مقدار N بزرگ‌تر باشد در نتیجه کوئری‌های بیشتری اجرا می‌شود و performance آن بیشتر می‌شود.

این امر به طور معمول در ابزارهای GraphQL و ORM<sup>۴</sup> دیده می‌شود و می‌توان با بهینه‌سازی کوئری SQL یا استفاده از یک بارگذار<sup>۵</sup> داده که درخواست‌های متوالی را دسته‌بندی می‌کند و یک درخواست داده واحد را نیز انجام می‌دهد، برطرف شود.

**مزایا:**

بیایید به برخی از مزایای استفاده از پایگاه‌های داده رابطه‌ای نگاه کنیم:

- ساده و دقیق
- دسترسی
- یکپارچه داده‌ها
- انعطاف‌پذیری

data subsetting<sup>۱</sup>

performance<sup>۲</sup>

replication<sup>۳</sup>

Object-Relational Mapping<sup>۴</sup>

data loader<sup>۵</sup>

### معایب:

در زیر معایب استفاده از پایگاههای داده رابطه‌ای آورده شده است:

- نگهداری پرهزینه
- تکامل طرح schema/ دشوار
- مشکلات عملکردی (join، denormalization و غیره)
- مقیاس‌پذیری دشوار به دلیل مقیاس‌پذیری افقی ضعیف

### مثال‌ها:

در اینجا برخی از پایگاههای داده رابطه‌ای رایج آورده شده است:

- PostgreSQL
- MySQL
- MariaDB
- Amazon Aurora

## پایگاهداده‌های NoSQL

NoSQL دارای یک دسته‌بندی گسترده است که شامل هر پایگاهداده‌ای می‌شود که از SQL به عنوان زبان دسترسی اصلی به داده‌ها استفاده نمی‌کند. این نوع پایگاهداده‌ها گاهی اوقات به عنوان پایگاهداده‌ها غیررابطه‌ای نیز شناخته می‌شوند. برخلاف پایگاهداده‌های رابطه‌ای، داده‌ها در یک پایگاهداده NoSQL مجبور نیستند با یک schema از پیش تعریف شده مطابقت داشته باشند. پایگاههای داده NoSQL از مدل یکپارچگی BASE پیروی می‌کنند.

در زیر انواع مختلف پایگاههای داده NoSQL آورده شده است:

**سندگرا (Document)**

یک پایگاهداده سندی (همچنین به عنوان یک پایگاهداده سندگرا<sup>۱</sup> یا یک ذخیره‌کننده سند شناخته می‌شود) پایگاه داده‌ای است که اطلاعات را در اسناد<sup>۲</sup> ذخیره می‌کند. آنها پایگاه‌های داده‌ای با کاربری عمومی هستند که برای انواع موارد استفاده برای اپلیکیشن‌های تراکنش‌گرا و تحلیلی کاربرد دارند.

**مزایا:**

- ساده و انعطاف‌پذیر
- مقیاس‌پذیری افقی آسان
- بدون طرح‌واره (Schemaless)

**معایب:**

- بدون طرح‌واره (Schemaless)
- غیرابطه‌ای

**مثال‌ها:**

- MongoDB
- Amazon DocumentDB
- CouchDB

**کلید - مقدار (Key-Value)**

یکی از ساده‌ترین انواع پایگاه‌های داده NoSQL، پایگاه‌های داده کلید - مقدار (key-value) است که داده‌ها را به عنوان گروهی از زوج‌های کلید - مقدار که هر کدام از دو مورد داده تشکیل شده است، ذخیره می‌کنند. آنها همچنین گاهی اوقات به عنوان یک ذخیره کلید - مقدار شناخته می‌شوند.

**مزایا:**

- ساده و با کارایی بالا
- مقیاس‌پذیری بالا برای حجم بالای ترافیک
- مدیریت Session
- جستجوی بهینه

معایب:

- اجرای عملیات ساده در ایجاد، خواندن، بروزرسانی و حذف (در اصطلاح (CRUD) جهت اعمال روی داده‌ها.
- مقادیر قابل فیلتر نیستند.
- فاقد قابلیت‌های indexing و اسکن.
- برای کوئری‌های پیچیده بهینه نشده است.

مثال‌ها:

- Redis
- Memcached
- Amazon DynamoDB
- Aerospike

## پایگاهداده گرافی (Graph)

پایگاهداده گرافی یک پایگاهداده NoSQL است که از ساختارهای گراف برای کوئری‌های معنایی<sup>۱</sup> با گره‌ها، یال‌ها و ویژگی‌ها برای نمایش و ذخیره داده به جای جداول یا اسناد استفاده می‌کند.

این گراف، آیتم‌های داده موجود در ذخیره را به مجموعه‌ای از گره‌ها و یال‌ها مرتبط می‌کند، یال‌ها نشان‌دهنده روابط بین گره‌ها هستند. این روابط به داده‌های موجود در ذخیره اجازه

می‌دهد تا به طور مستقیم به هم مرتبط شوند و در بسیاری از موارد، با یک عملیات بازیابی شوند.

مزایا:

- سرعت کوئری بالا
- چابک و انعطاف‌پذیر
- نمایش صریح داده

معایب:

- پیچیده
- بدون زبان کوئری استاندارد

کاربرد:

- کشف تقلب
- هسته برنامه‌های توصیه‌گر
- شبکه‌های اجتماعی
- شبکه mapping

مثال‌ها:

- Neo4j
- ArangoDB
- Amazon Neptune
- JanusGraph

### سری زمانی (Time Series)

یک پایگاهداده سری زمانی پایگاه داده‌ای است که برای داده‌های دارای time-stamped یا سری زمانی بهینه شده است.

مزایا:

- درج و بازیابی سریع
- ذخیره‌سازی کارآمد داده

### حوزه‌های کاربرد

- داده‌های اینترنت اشیاء
- تجزیه و تحلیل متريک‌ها
- نظارت بر برنامه‌ها
- درک روندهای مالی

مثال‌ها:

- InfluxDB
- Apache Druid

### ستون گسترده (Wide-column)

پایگاه‌های داده ستون گسترده که همچنین به عنوان ذخیره‌سازهای ستون گسترده شناخته می‌شوند، بدون schema هستند. داده‌ها به جای ردیف‌ها و ستون‌ها، در خانواده‌های ستونی ذخیره می‌شوند.

مزایا:

- مقایسه‌پذیری بالا، می‌تواند با پتابایت داده‌ها کار کند
- ایده‌آل برای برنامه‌هایی با داده‌های بزرگ و بلادرنگ

معایب:

- گران‌قیمت
- صرف زمان زیاد در نوشتمن داده

حوزه‌های کاربردی:

- تجزیه و تحلیل کسب و کار
- ذخیره سازی داده مبتنی بر ویژگی

مثال‌ها:

- BigTable
- Apache Cassandra
- ScyllaDB

### چندمدلی (Multi-model)

پایگاه‌های داده چندمدلی، مدل‌های پایگاه داده مختلف (رابطه‌ای، گراف، کلید – مقدار، سند و غیره) را در یک ساختار یکپارچه ترکیب می‌کنند. این بدان معناست که آنها می‌توانند انواع مختلف داده‌ها، ایندکس‌ها، کوئری‌ها را در خود جای دهند و داده‌ها را در بیش از یک مدل ذخیره کنند.

مزایا:

- انعطاف‌پذیری
- مناسب برای پروژه‌های پیچیده
- داده‌های یکپارچه<sup>۱</sup>

معایب:

- پیچیده
- کم تجربه تر

مثال‌ها:

- ArangoDB
- Azure Cosmos DB
- Couchbase

## Column-based یا row-based

پایگاههای داده در دو ساختار اساسی قرار می‌گیرد آنها می‌توانند ردیف محور (Row-based) یا ستون محور (Column-based) باشند. این تصمیم کلیدی نحوه ذخیره‌سازی داده‌های شما (در ردیف‌ها یا ستون‌ها) را تعیین می‌کند و تأثیر قابل توجهی بر عملکرد و مقیاس‌پذیری دارد.

### پایگاهداده‌های ردیف محور

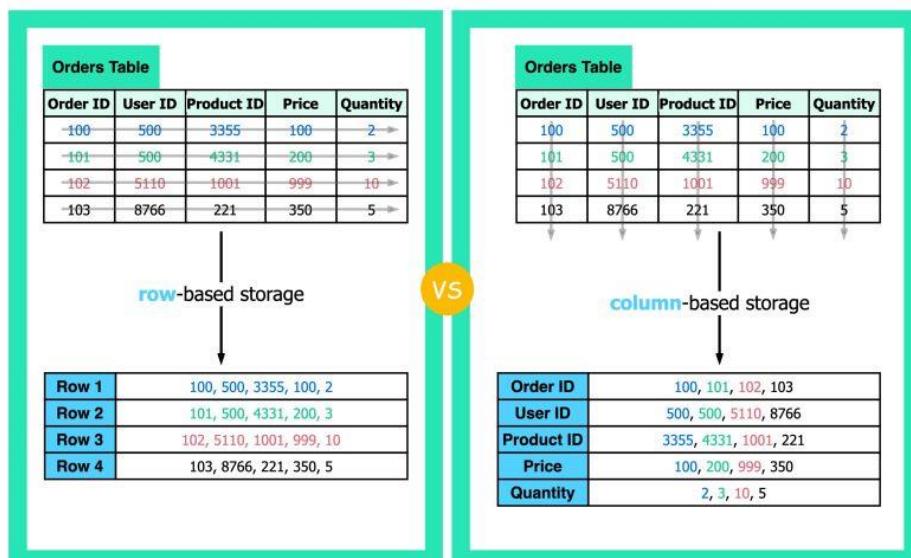
پایگاههای داده ردیف محور، داده‌ها را در ردیف‌ها ذخیره می‌کنند. تمام اطلاعات مربوط به یک رکورد باهم ذخیره می‌شوند و بازیابی کل ردیف را آسان می‌سازند.

### پایگاهداده‌های ستون محور

پایگاههای داده ستون محور، اطلاعات را به جای ردیف‌ها، بر اساس ستون‌ها ذخیره می‌کنند؛ بنابراین یک ستون حاوی تمام مقادیر یک فیلد در سراسر بسیاری از ردیف‌ها و رکوردها خواهد بود.

#### Row-based DB v.s Column-based DB

 blog.bytebytogo.com



### چرا این موضوع برای عملکرد کوئری اهمیت دارد؟

ساختار پایگاهداده نقش مهمی در عملکرد کوئری ایفا می‌کند. پایگاههای داده ردیف محور، بازیابی کارآمد کل ردیف‌ها را امکان‌پذیر می‌کنند. این امر آنها را برای پردازش تراکنش‌هایی که نیاز به دسترسی به رکوردهای کامل دارند، مناسب‌تر می‌کند.

ذخیره‌سازهای ستون محور در کوئری‌های تجمعی (مانند جمع یا میانگین) که در بسیاری از ردیف‌ها جستجو می‌کنند، عملکرد بهتری دارند. زیرا یک ستون حاوی تمام داده‌های مرتبط برای محاسبه آمار است. کوئری‌های تحلیلی با کاهش حجم داده‌هایی که نیاز به اسکن دارند، بسیار سریع‌تر اجرا می‌شوند.

### چرا این موضوع برای مقیاس‌پذیری اهمیت دارد؟

پایگاههای داده ستون محور اغلب برای حجم کاری تحلیلی مقیاس‌پذیر‌تر هستند، زیرا داده‌ها از قبل بر اساس ستون ذخیره شده‌اند. افزایش حجم برای ذخیره‌سازی دسته‌های بزرگ و جدید داده‌ها با جهت‌گیری ستونی آسان‌تر است. فشرده‌سازی ستونی همچنین فضای قابل توجهی را برای حجم عظیم داده‌ها ذخیره می‌کند. این ویژگی، پایگاههای داده ستون محور را برای «داده‌های بزرگ» مناسب می‌کند.

اما بهروزرسانی داده‌ها می‌تواند کندرت باشد؛ بنابراین، ذخیره‌سازهای ردیف محور ممکن است برای تراکنش‌های حجمی که به طور مرتب رکوردها را اضافه یا بهروزرسانی می‌کنند، مناسب‌تر باشند. این ساختار با این سناریوهای استفاده همخوانی بیشتری دارد.

### از همان ابتدا درست انتخاب کنید!

بسیاری از برنامه‌های مدرن با حجم داده زیاد، از هر دو پایگاهداده ردیف محور و ستون محور استفاده می‌کنند. Application تراکنشی برای رسیدگی به عملیات تجاری حیاتی، Postgres برای ثبات و انطباق با ACID به سیستم‌های ردیف محور مانند MySQL و PostgreSQL متکی هستند. حجم کاری تحلیلی برای هوش تجاری تمایل به استفاده از انبارهای داده ستون محور مانند ClickHouse برای عملکرد در مقیاس بالا دارند.

## پایگاه‌داده‌های NoSQL در مقابل SQL

در اینجا برخی از تفاوت‌های سطح بالا بین SQL و NoSQL آورده شده است:

[blog.amigoscode.com](http://blog.amigoscode.com)

### RELATIONAL VS NOSQL DATABASES



در دنیای پایگاه‌های داده، دو نوع راه حل اصلی وجود دارد: پایگاه‌های داده SQL (رابطه‌ای) و NoSQL (غیررابطه‌ای). هر دوی آنها در نحوه ساخت و نوع اطلاعاتی که ذخیره می‌کنند و حتی نحوه ذخیره آنها متفاوت هستند. پایگاه‌های داده رابطه‌ای ساختاریافته و دارای

طرح‌واره‌های<sup>۱</sup> از پیش تعریف شده هستند، درحالی‌که پایگاه‌های داده غیررابطه‌ای بدون ساختار، توزیع شده و دارای طرح‌واره‌های پویا هستند.

### تفاوت‌های سطح بالا

#### ۱. ذخیره‌سازی

داده‌ها را در جداول ذخیره می‌کند، جایی که هر ردیف نشان‌دهنده یک موجودیت و هر ستون نشان‌دهنده یک نقطه داده در مورد آن موجودیت است. پایگاه‌های داده NoSQL مدل‌های ذخیره‌سازی داده متفاوتی مانند کلید – مقدار، گراف، سند و غیره دارند.

#### ۲. schema

در SQL، هر رکورد با یک طرح‌واره/Schema ثابت مطابقت دارد، به این معنی که ستون‌ها باید قبل از ورود داده تصمیم‌گیری و انتخاب شوند و هر ردیف باید برای هر ستون داده داشته باشد. schema را می‌توان بعداً تغییر داد، اما این کار مستلزم اصلاح پایگاه‌داده با استفاده از<sup>۲</sup> migrations است. درحالی‌که در NoSQL، طرح‌واره‌ها پویا هستند. فیلد‌ها<sup>۳</sup> را می‌توان در حین کار اضافه کرد و هر رکورد (معادل آن) نیازی به حاوی داده برای هر فیلد ندارد.

#### ۳. کوئری

پایگاه‌های داده SQL از زبان پرس‌وجوی ساختاریافته<sup>۴</sup> (SQL) برای تعریف و دست‌کاری داده‌ها استفاده می‌کنند که بسیار قدرتمند است. SQL یک زبان استاندارد است که در اکثر پایگاه‌های داده رابطه‌ای استفاده می‌شود. در یک پایگاه‌داده NoSQL،

<sup>۱</sup> schemas

<sup>۲</sup> مهاجرت

<sup>۳</sup> Fields

<sup>۴</sup> structured query language

کوئری‌ها بر روی مجموعه‌ای از اسناد یا سایر ساختارهای داده خاص به پایگاهداده خاص تمرکز دارند. هر پایگاهداده NoSQL ممکن است syntax (زبان) خاص خود را برای کوئری داشته باشد.

#### ۴. مقیاس‌پذیری (Scalability)

در اکثر موارد رایج، پایگاههای داده SQL به صورت عمودی مقیاس‌پذیر<sup>۱</sup> هستند که می‌تواند بسیار گران شود. امکان مقیاس‌بندی یک پایگاهداده رابطه‌ای در چندین سرور وجود دارد، هرچند این یک فرایند پیچیده و زمانبر است. از طرف دیگر، پایگاههای داده NoSQL به صورت افقی مقیاس‌پذیر<sup>۲</sup> هستند، به این معنی که می‌توانیم به راحتی سرورهای بیشتری را به زیرساخت پایگاهداده NoSQL خود برای مدیریت ترافیک بالا اضافه کنیم. هر سخت‌افزار معمولی ارزان‌قیمتی یا نمونه‌های ابری می‌توانند میزبان پایگاههای داده NoSQL باشند، بنابراین از نظر هزینه بسیار مغروبه‌صرفه‌تر از مقیاس‌پذیری عمودی هستند. بسیاری از فناوری‌های NoSQL همچنین داده‌ها را به طور خودکار در سراسر سرورها توزیع می‌کنند.

#### ۵. قابلیت اطمینان

اکثر قریب به اتفاق پایگاهداده‌های رابطه‌ای با ACID سازگار هستند. ACID مخفف Consistency (اتمسیت)، Atomicity (یکپارچگی)، Isolation (جداسازی) و Durability (دوام) است که تضمین‌هایی را برای صحت و امنیت تراکنش‌های داده ارائه می‌دهد؛ بنابراین، هنگامی که صحبت از قابلیت اطمینان داده و تضمین ایمن انجام تراکنش‌ها می‌شود، پایگاههای داده SQL همچنان گزینه بهتری هستند. اکثر راه حل‌های NoSQL کارایی و مقیاس‌پذیری را برای مطابقت بیشتر با ACID، قربانی می‌کنند.

vertically scalable<sup>۱</sup>

horizontally scalable<sup>۲</sup>

## دلایل انتخاب

همان‌طور که همیشه، ما باید فناوری‌ای را انتخاب کنیم که بهتر با نیازمندی‌های ما مطابقت داشته باشد؛ بنابراین، بباید به برخی از دلایل انتخاب پایگاه‌داده مبتنی بر SQL یا NoSQL نگاه کنیم:

### برای SQL:

- داده‌های ساختاریافته با schema سخت
- داده‌های رابطه‌ای
- نیاز به پیوندهای<sup>۱</sup> پیچیده.
- تراکنش‌ها.
- جستجو بر اساس ایندکس‌ها بسیار سریع است.

### برای NoSQL:

- schema پویا یا انعطاف‌پذیر
- داده‌های غیر رابطه‌ای
- عدم نیاز به اتصالات پیچیده
- حجم کاری بسیار پر از داده
- توان عملیاتی بسیار بالا برای IOPS (عملیات ورودی/خروجی در ثانیه).

---

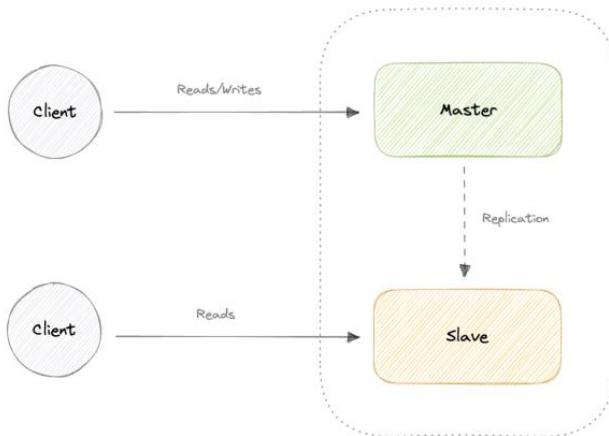
<sup>۱</sup> joins

## تکثیر پایگاهداده (Database Replication)

تکثیر یا Replication روندی است که شامل به اشتراک گذاری اطلاعات برای اطمینان از سازگاری و یکپارچگی داده‌ها بین منابع اضافی<sup>۱</sup> مانند چندین پایگاهداده، برای بهبود قابلیت اطمینان، تحمل خطا<sup>۲</sup> یا قابلیت‌های دسترسی است.

### Master-Slave Replication

Master، خواندن و نوشتمن را انجام می‌دهد و عملیات نوشتمن را به یک یا چند slave که فقط خواندن را انجام می‌دهند، تکثیر می‌کند. slave‌ها همچنین می‌توانند به صورت درختی (tree-like) توابع بیشتری را تکثیر کنند. اگر Master از کار بیفتد، سیستم می‌تواند تا زمانی که یک slave به عنوان Master ارتقا یابد یا یک Master جدید انتخاب و استفاده شود، در حالت فقط خواندن (read-only) به کار خود ادامه دهد.



مزایا:

- تهیه نسخه پشتیبان از کل پایگاهداده بدون تأثیرگذاری قابل توجه روی Master
- اپلیکیشن‌ها می‌توانند بدون تأثیرگذاری از slave‌ها بخوانند.

<sup>۱</sup> redundant

<sup>۲</sup> fault-tolerance

- slave را می توان بدون هیچ گونه خرابی و downtime به صورت آفلاین درآورد و دوباره با Master همگام سازی کرد.

معایب:

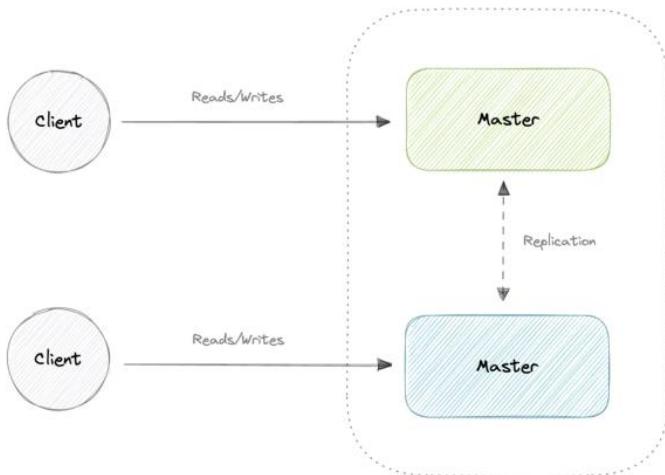
- تکثیر به سخت افزار بیشتری نیاز دارد و پیچیدگی بیشتری را به همراه دارد.
- با وجود خرابی یا downtime، احتمال از دست رفتن داده ها هنگام خرابی Master وجود دارد.
- همچنین همه عملیات نوشتن باید در یک ساختار master-slave روی Master انجام شود.
- هرچه تعداد slave های خواندنی<sup>۱</sup> بیشتر باشد، ما باید موارد بیشتری را تکثیر کنیم که باعث افزایش تأخیر تکثیر<sup>۲</sup> خواهد شد.

### Master-Master Replication

هر دو Master، خواندن/نوشتن را انجام می دهند و با یکدیگر هماهنگ و همگام سازی می شوند. اگر هر یک از Master ها از کار بیفتند، سیستم می تواند به کار خود با توجه به هر دو روش های خواندن و نوشتن ادامه دهد.

read slaves<sup>۱</sup>

replication lag<sup>۲</sup>



مزایا:

- اپلیکیشن‌ها می‌توانند از هر دو Master بخوانند.
- بار نوشتن را در سراسر گره‌های Master توزیع می‌کنند.
- انتقال خودکار و failover/بازیابی سریع

معایب:

- پیکربندی و استقرار آن به سادگی master-slave نیست.
- به دلیل همگام‌سازی، یکپارچگی ضعیف یا تأخیر در نوشتن داده‌ها بیشتر است.
- با اضافه شدن گره‌های نوشتن بیشتر و افزایش تأخیر، حل تعارض (conflict resolution) اهمیت پیدا می‌کند.

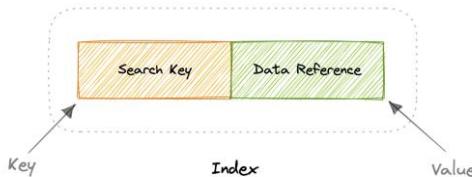
تکثیر همزمان در مقابل غیرهمزمان<sup>۱</sup>

تفاوت اصلی بین تکثیر هم‌زمان و غیره‌هم‌زمان، نحوه نوشتن داده‌ها در نمونه به صورت replica/ تکثیر است. در تکثیر هم‌زمان، داده‌ها به طور هم‌زمان در حافظه اصلی و replica نوشته می‌شوند. بدین ترتیب، نسخه اصلی و replica باید همیشه همگام باقی بمانند.

در مقابل، تکثیر غیره‌هم‌زمان، داده‌ها را پس از اینکه قبلاً در حافظه اصلی نوشته شده‌اند در replica کپی می‌کند. اگرچه فرایند تکثیر ممکن است تقریباً در زمان واقعی رخ دهد، اما تکثیر در یک برنامه زمان‌بندی شده<sup>۱</sup> رایج‌تر و از نظر هزینه‌ای بهصرفه‌تر است.

## ایندکس‌ها (indexes)

ایندکس‌ها در پایگاه‌های داده بسیار شناخته شده هستند و از آن‌ها برای بهبود سرعت عملیات بازیابی داده در ذخیره‌گاه داده<sup>۱</sup> استفاده می‌شود. ایندکس با افزایش سربار ذخیره‌سازی و نوشتمن کنتر (از آنجایی که ما نه تنها باید داده‌ها را بنویسیم؛ بلکه باید ایندکس را نیز به روزرسانی کنیم) به نفع خواندن سریع‌تر، معاوضه یا بدء‌بستان<sup>۲</sup> می‌کند. ایندکس‌ها برای یافتن سریع داده‌ها بدون نیاز به بررسی تک‌تک ردیف‌های یک جدول پایگاهداده استفاده می‌شوند. ایندکس‌ها را می‌توان با استفاده از یک یا چند ستون از یک جدول پایگاهداده ایجاد کرد که اساساً برای جستجوی تصادفی سریع و دسترسی کارآمد به رکوردهای مرتب‌سازی شده، کاربرد دارد.



یک ایندکس یک ساختار داده است که می‌توان آن را به عنوان یک فهرست در نظر گرفت که ما را به محل زندگی داده‌های واقعی هدایت می‌کند و در واقع نوعی متادیتا هستند؛ بنابراین، هنگامی که یک ایندکس روی یک ستون از یک جدول ایجاد می‌کنیم، آن ستون و یک اشاره‌گر به کل آن ردیف را در ایندکس ذخیره می‌کنیم. ایندکس‌ها همچنین برای ایجاد نماهای مختلف از داده‌های مشابه استفاده می‌شوند. برای مجموعه داده‌های بزرگ، این یک راه عالی برای مشخص کردن فیلترهای مختلف یا طرح‌های مرتب‌سازی بدون نیاز به ایجاد چندین کپی اضافی از داده‌ها است.

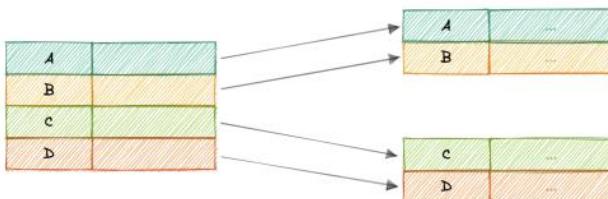
data store<sup>۱</sup>

trade-offs<sup>۲</sup>

یکی از ویژگی‌های ایندکس‌های پایگاهداده این است که می‌توانند متراکم (dense) یا پراکنده (sparse) باشند. هر یک از این ویژگی‌های ایندکس با مزايا و معایب خاص خود همراه است. بیایید نحوه عملکرد هر نوع ایندکس را بررسی کنیم:

### ایндکس متراکم (Dense Index)

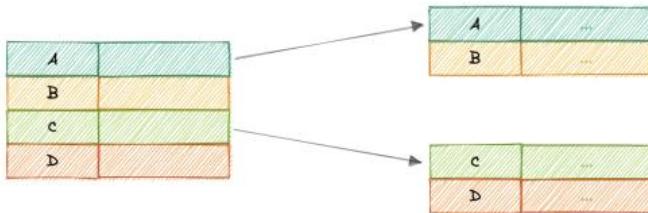
در یک ایندکس متراکم، برای هر ردیف از جدول یک رکورد ایندکس ایجاد می‌شود. رکوردها را می‌توان مستقیماً پیدا کرد؛ زیرا هر رکورد از ایندکس، مقدار کلید جستجو و اشاره‌گر به رکورد واقعی را نگه می‌دارد.



ایндکس‌های dense در زمان نوشتن به نگهداری بیشتری نسبت به ایندکس‌های sparse نیاز دارند. از آنجایی که هر ردیفی باید یک ورودی داشته باشد، پایگاهداده باید ایندکس را در درج، بهروزرسانی و حذفها را نگهداری کند. داشتن یک ورودی برای هر ردیف به این معنی است که ایندکس‌های متراکم به حافظه بیشتری نیاز دارند. مزیت یک ایندکس متراکم این است که مقادیر را می‌توان به سرعت فقط با جستجوی دو دویی پیدا کرد. ایندکس‌های متراکم همچنین هیچ الزام مرتب‌سازی بر روی داده‌ها اعمال نمی‌کنند.

### ایндکس پراکنده (Sparse Index)

در یک ایندکس پراکنده، رکوردها فقط برای برخی از رکوردها ایجاد می‌شوند. ایندکس‌های پراکنده در زمان نوشتن به نگهداری کمتری نسبت به ایندکس‌های متراکم نیاز دارند؛ زیرا آن‌ها فقط حاوی زیرمجموعه‌ای از مقادیر هستند. این بار نگهداری کمتر به این معنی است که درج، بهروزرسانی و حذفها سریع‌تر خواهد بود.



داشتن ورودی‌های کمتر همچنین به این معنی است که ایندکس از حافظه کمتری استفاده می‌کند. در این حالت، یافتن داده‌ها کندر است؛ زیرا یک اسکن در سراسر صفحه به طور معمول پس از جستجوی دودویی انجام می‌شود. ایندکس‌های پراکنده همچنین هنگام کار با داده‌های مرتب‌سازی شده به صورت اختیاری هستند.

### تراکنش (Transaction)

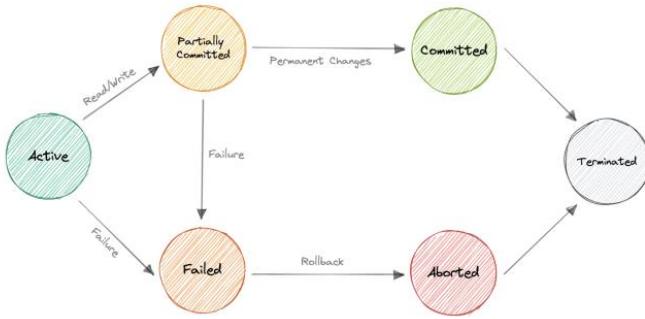
تراکنش مجموعه‌ای از عملیات پایگاهداده است که به عنوان یک «واحد منفرد کاری»<sup>۱</sup> در نظر گرفته می‌شود. عملیات موجود در یک تراکنش یا همه با موفقیت انجام می‌شوند یا همه با شکست مواجه می‌شوند. به این ترتیب، مفهوم تراکنش از یکپارچگی داده‌ها تا زمانی که بخشی از سیستم با مشکل مواجه می‌شود، پشتیبانی می‌کند. همه پایگاه‌های داده تراکنش‌های ACID را پشتیبانی نمی‌کنند، معمولاً<sup>۲</sup> به این دلیل است که آن‌ها اولویت را به سایر بهینه‌سازی‌هایی می‌دهند که پیاده‌سازی آن‌ها با هم دشوار یا از نظر تئوری غیرممکن است.

به طور معمول، پایگاه‌های داده relational از تراکنش‌های ACID پشتیبانی می‌کنند و پایگاه‌های داده non-relational از ACID پشتیبانی نمی‌کنند (استثنایی وجود دارد).

### وضعیت (States)

یک تراکنش در یک پایگاهداده می‌تواند در یکی از حالات زیر باشد:

<sup>1</sup> single unit of work



### فعال (Active)

در این حالت، تراکنش در حال اجرا است. این حالت اولیه هر تراکنش است.

### Partially Committed

زمانی که یک تراکنش آخرین عملیات خود را اجرا می‌کند، گفته می‌شود که در حالت جزئی قرار دارد. Commit

### Committed

اگر یک تراکنش تمام عملیات خود را با موفقیت انجام دهد به آن Committed گفته می‌شود و همه اثرات آن اکنون به طور دائم در سیستم پایگاهداده برقرار شده است.

### شکست‌خورده (Failed)

اگر هر یک از بررسی‌های انجام شده توسط سیستم بازیابی پایگاهداده با شکست مواجه شود، گفته می‌شود که تراکنش در حالت شکست‌خورده یا Failed قرار دارد. یک تراکنش شکست‌خورده دیگر نمی‌تواند ادامه یابد.

### لغو شده (Aborted)

اگر هر یک از بررسی‌ها با شکست مواجه شود و تراکنش به حالت شکست‌خورده برسد، مدیر بازیابی تمام عملیات نوشتن را در پایگاهداده لغو می‌کند تا پایگاهداده را به حالت اولیه خود قبل از اجرای تراکنش بازگرداند. تراکنش‌ها در این حالت لغو می‌شوند. مازوں بازیابی پایگاهداده می‌تواند پس از لغو یک تراکنش، یکی از دو عملیات را انتخاب کند:

- راهاندازی مجدد تراکنش
- پایان‌دادن به تراکنش
- پایان‌یافته<sup>۱</sup>

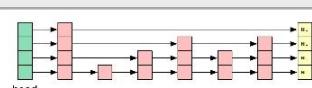
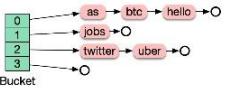
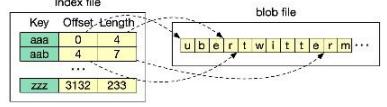
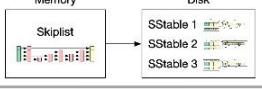
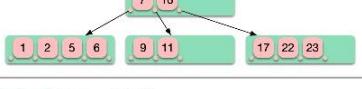
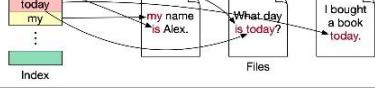
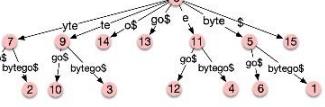
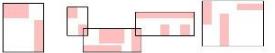
اگر هیچ بازگشتی وجود نداشته باشد یا تراکنش از حالت Commit انجام شده باشد، سیستم سازگار و یکپارچه است و برای یک تراکنش جدید آماده است و تراکنش قدیمی پایان‌یافته است.

## ۸ ساختار داده‌ای که پایگاه‌داده‌ها را قادر تمند می‌کنند.

این مسئله بسته به مورد استفاده شما متفاوت خواهد بود. داده‌ها می‌توانند در حافظه موقت یا روی دیسک فهرست‌بندی شوند. به طور مشابه، فرمتهای داده؛ مانند اعداد، رشته‌ها، مختصات جغرافیایی و غیره متفاوت هستند. سیستم ممکن است دارای حجم نوشتاری بالا یا خواندنی بالا باشد. همه این عوامل بر انتخاب فرمت فهرست پایگاه‌داده شما تأثیر می‌گذارند.

### 8 Data Structures That Power Your Databases

 ByteByteGo.com

Types	Illustration	Use Case	Note
Skiplist		In-memory	used in Redis
Hash index		In-memory	Most common in-memory index solution
SSTable		Disk-based	Immutable data structure. Seldom used alone
LSM tree		Memory + Disk	High write throughput. Disk compaction may impact performance
B-tree		Disk-based	Most popular database index implementation
Inverted index		Search document	Used in document search engine such as Lucene
Suffix tree		Search string	Used in string search, such as string suffix match
R-tree		Search multi-dimension shape	Such as the nearest neighbor

در ادامه برخی از محبوب‌ترین ساختارهای داده‌ای که برای فهرست‌بندی داده‌ها استفاده می‌شوند، آورده شده است:

- Skiplist: یک نوع شاخص<sup>۱</sup> رایج در حافظه موقتِ داخلی<sup>۲</sup> است که معمولاً در پایگاهداده Redis استفاده می‌شود.
- Hash index: پیاده‌سازی بسیار رایج ساختار داده «نگاشت»<sup>۳</sup> یا «مجموعه»<sup>۴</sup>.
- جدول ساختاریافته قابل تغییر (SSTable): پیاده‌سازی غیرقابل تغییر «نگاشت» روی دیسک است.
- LSM tree: برابر است با حاصل SSTable + Skiplist که توان عملیاتی بالایی در نوشتن داده‌ها ایجاد می‌کند.
- B-tree: راه حل مبتنی بر دیسک. عملکرد خواندن/نوشتن یکنواخت.
- ایندکس وارونه (Inverted index): برای فهرست‌بندی اسناد استفاده می‌شود. در پایگاهداده‌ای به نام Apache Lucene استفاده می‌شود.
- درخت پسوند (Suffix tree): برای جستجوی الگوهای رشته‌ای string استفاده می‌شود.
- R-tree: جستجوی چندبعدی، مانند یافتن نزدیک‌ترین همسایه.

index<sup>۱</sup>

in-memory<sup>۲</sup>

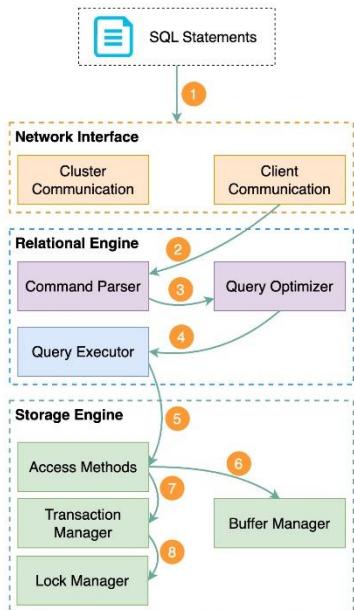
Map<sup>۳</sup>

Collection<sup>۴</sup>

## چگونه یک عبارت SQL در پایگاهداده اجرا می‌شود؟

نمودار زیر این فرایند را نشان می‌دهد. توجه داشته باشید که معماری پایگاهداده‌های مختلف متفاوت است، این نمودار برخی از طرح‌های رایج را نشان می‌دهد.

How is SQL Executed in DB? [blog.bytebytego.com](http://blog.bytebytego.com)



۱. یک دستور SQL از طریق پروتکل لایه انتقال<sup>۱</sup> (مانند TCP) به پایگاهداده ارسال می‌شود.
۲. دستور SQL به تجزیه‌گر دستور<sup>۲</sup> ارسال می‌شود، جایی که تجزیه و تحلیل نحوی و معنایی<sup>۳</sup> انجام می‌شود و پس از آن یک درخت کوئری<sup>۴</sup> ایجاد می‌شود.

layer protocol<sup>۱</sup>  
command parser<sup>۲</sup>  
syntactic and semantic analysis<sup>۳</sup>  
query tree<sup>۴</sup>

۳. درخت کوئری به بهینهساز<sup>۱</sup> ارسال می‌شود. بهینهساز یک برنامه اجرایی ایجاد می‌کند.
۴. برنامه اجرایی به اجراکننده (executor) ارسال می‌شود. executor داده‌ها را پس از اجرا بازیابی می‌کند.
۵. روش‌های دسترسی، منطق بازیابی داده‌های موردنیاز برای اجرا را فراهم می‌کنند و داده‌ها را از موتور ذخیره‌سازی بازیابی می‌کنند.
۶. روش‌های دسترسی تصمیم می‌گیرند که آیا دستور SQL فقط خواندنی است یا خیر. اگر پرس‌وجو/کوئری فقط خواندنی باشد (مثلاً دستور SELECT)، برای پردازش بیشتر به مدیر بافر<sup>۲</sup> منتقل می‌شود. مدیر بافر داده‌ها را در حافظه کش یا فایل‌های داده جستجو می‌کند.
۷. اگر دستور INSERT یا UPDATE باشد، برای پردازش بیشتر به مدیر تراکنش<sup>۳</sup> منتقل می‌شود.
۸. در طول یک تراکنش، داده‌ها در حالت قفل هستند. این امر توسط مدیر قفل<sup>۴</sup> تضمین می‌شود. همچنین خواص ACID تراکنش را تضمین می‌کند.

optimizer<sup>۱</sup>

buffer manager<sup>۲</sup>

transaction manager<sup>۳</sup>

lock manager<sup>۴</sup>

## دستورات اصلی پایگاهداده SQL

یک پایگاهداده از سه نوع شیء تشکیل شده است:

Database –

Table –

Index –

	Create	Read	Update	Delete
 Database	<code>CREATE DATABASE name;</code>			<code>DROP DATABASE name;</code>
 Table	<code>CREATE TABLE name {     col_1 int,     col_2 varchar(255),     col_3 string, };</code>	<code>SELECT * from name WHERE col_1 = 2;</code>	<code>UPDATE name SET col_1 = 3 WHERE col_3 = "a";</code>  <code>INSERT INTO name VALUES (1,"a","b")</code>	<code>DELETE FROM name WHERE col_3 = "a";</code>  <code>DROP TABLE name;</code>
 Index	<code>CREATE INDEX index_name ON name (     col_1,     col_2, );</code>			<code>DROP INDEX index_name</code>

ایندکس هر نوع شی دارای چهار عملیات (به نام CRUD) است:

Create –

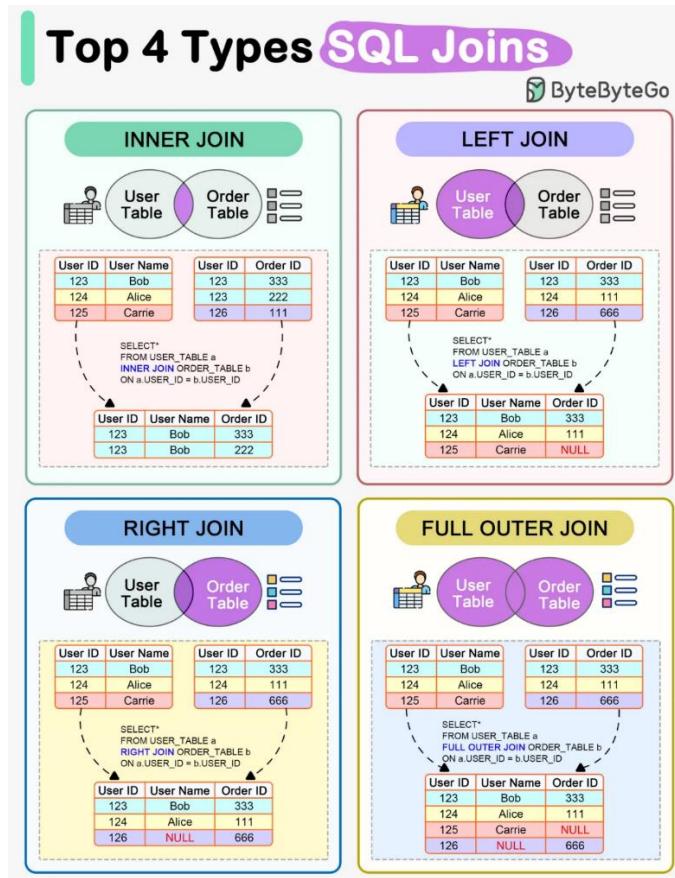
Read –

Update –

Delete –

بنابراین، در مجموع ۱۲ دسته از دستورات SQL وجود دارد. برخی از دسته‌ها به دلیل استفاده کمتر از نمودار حذف شده‌اند. اکیداً توصیه می‌شود با دسته‌های باقی‌مانده آشنا شوید.

چگونه SQL Joins کار می‌کنند؟



## INNER JOIN

رکوردهای مطابق را در هر دو جدول برمی‌گرداند.

## LEFT JOIN

تمام رکوردها را از جدول چپ و رکوردهای مطابق را از جدول راست برمی‌گرداند.

## RIGHT JOIN

تمام رکوردها را از جدول راست و رکوردهای مطابق را از جدول چپ برمی‌گرداند.

## اتصال خارجی کامل (FULL OUTER JOIN)

تمام رکوردهایی را برمی‌گرداند که مطابقی در جدول چپ یا راست داشته باشند.

## نرمال‌سازی و غیر نرمال‌سازی

قبل از اینکه ادame دهیم، بباید به برخی از اصطلاحات رایج در نرمال‌سازی و غیر نرمال‌سازی<sup>۱</sup>

نگاهی بیندازیم:

### کلیدها

- کلید اصلی (Primary key): ستون یا گروهی از ستون‌ها که برای شناسایی منحصر به فرد هر ردیف جدول قابل استفاده هستند.
- کلید مرکب (Composite key): کلید اصلی که از چندین ستون تشکیل شده است.
- کلید فراگیر (Super key): مجموعه تمام کلیدهایی که می‌توانند همه سطرهای موجود در یک جدول را به طور منحصر شناسایی کنند.
- کلید کاندید (Candidate key): ویژگی‌هایی که سطرهای را در یک جدول به طور منحصر شناسایی می‌کنند.
- کلید خارجی (Foreign key): این کلید اشاره به کلید اصلی یک جدول دیگر می‌کند.
- کلید جایگزین (Alternate key): کلیدهایی که کلید اصلی نیستند به عنوان کلیدهای جایگزین شناخته می‌شوند.
- کلید جانشین (Surrogate key): یک مقدار تولید شده توسط سیستم که به طور منحصر به فرد در یک جدول را شناسایی می‌کند، زمانی که هیچ ستون دیگری قادر به نگهداری ویژگی‌های یک کلید اصلی نباشد.

### وابستگی‌ها

- وابستگی جزئی (Partial dependency): زمانی رخ می‌دهد که کلید اصلی برخی از ویژگی‌های دیگر را تعیین کند.

- وابستگی تابعی (Functional dependency): رابطه‌ای است که بین دو ویژگی وجود دارد، به طور معمول بین کلید اصلی و ویژگی غیرکلیدی<sup>۱</sup> در یک جدول.
- وابستگی تابعی متعدد (Transitive functional dependency): زمانی اتفاق می‌افتد که برخی ویژگی‌های غیرکلیدی، برخی ویژگی‌های دیگر را تعیین کند.

### ناهنجری‌ها (Anomalies)

ناهنجری پایگاهداده زمانی اتفاق می‌افتد که به دلیل برنامه‌ریزی نادرست یا ذخیره همه چیز در یک پایگاهداده تحت (flat database)، نقصی در پایگاهداده وجود داشته باشد. این کار به‌طورکلی با فرایند نرم‌السازی برطرف می‌شود.  
سه نوع ناهنجاری پایگاهداده وجود دارد:

- ناهنجاری درج (Insertion anomaly): زمانی اتفاق می‌افتد که بدون وجود سایر ویژگی‌ها، قادر به درج برخی ویژگی‌ها در پایگاهداده نباشیم.
- ناهنجاری بهروزرسانی (Update anomaly): در صورت وجود افزونگی داده<sup>۲</sup> و بهروزرسانی جزئی رخ می‌دهد. به عبارت دیگر، یک بهروزرسانی صحیح پایگاهداده به اقدامات دیگری مانند اضافه کردن، حذف یا هر دو نیاز دارد.
- ناهنجاری حذف (Deletion anomaly): جایی که حذف برخی از داده‌ها نیازمند حذف سایر داده‌ها است.

مثال

بیایید جدول زیر را در نظر بگیریم که نرم‌السازی نشده است:

<sup>۱</sup> non-key

<sup>۲</sup> data redundancy

ID	Name	Role	Team
1	Peter	Software Engineer	A
2	Brian	DevOps Engineer	B
3	Hailey	Product Manager	C
4	Hailey	Product Manager	C
5	Steve	Frontend Engineer	D

تصور کنید فرد جدیدی به نام «John» را استخدام کرده‌ایم؛ اما ممکن است بلافضله تیمی به او اختصاص داده نشود. این باعث ایجاد یک ناهنجاری درج<sup>۱</sup> می‌شود؛ زیرا ویژگی تیم هنوز وجود ندارد.

در مرحله بعد، فرض کنید «Hailey» از تیم C ترفع می‌گیرد، برای اعمال آن تغییر در پایگاهداده، برای حفظ یکپارچگی<sup>۲</sup> باید ۲ ردیف را بهروزرسانی کنیم که می‌تواند باعث ایجاد یک ناهنجاری بهروزرسانی شود.

در نهایت، ما می‌خواهیم تیم B را حذف کنیم، اما برای انجام این کار نیاز به حذف اطلاعات اضافی مانند نام و نقش<sup>۳</sup> در این جدول داریم، این نمونه‌ای از یک ناهنجاری حذف<sup>۴</sup> است.

### نرمال‌سازی (Normalization)

نرمال‌سازی فرایند سازماندهی داده‌ها در یک پایگاهداده است. این شامل ایجاد جدول و برقراری روابط بین آن جداول طبق قوانینی است که برای محافظت از داده‌ها و همچنین انعطاف‌پذیرتر کردن پایگاهداده با حذف افزونگی و وابستگی‌های ناسازگار طراحی شده است.

insertion<sup>۱</sup>

consistency<sup>۲</sup>

role<sup>۳</sup>

deletion<sup>۴</sup>

## چرا به نرمال‌سازی نیاز داریم؟

هدف از نرمال‌سازی، حذف داده‌های تکراری و اطمینان از یکپارچگی داده‌ها<sup>۱</sup> است. یک پایگاهداده کاملاً نرمال شده به ساختار خود اجازه می‌دهد تا برای پذیرش انواع جدید داده‌ها بدون تغییر بیش از حد در ساختار موجود، گسترش یابد. در نتیجه، برنامه‌هایی که با پایگاهداده تعامل دارند، کمترین تأثیر را می‌پذیرند.

## فرم‌های نرمال

فرم‌های نرمال مجموعه‌ای از خط‌مشی‌ها برای اطمینان از نرمال بودن پایگاهداده هستند. بیایید به برخی از مهم‌ترین فرم‌های نرمال بپردازیم:

### 1NF (First Normal Form)

برای اینکه یک جدول در اولین فرم نرمال (1NF) باشد، باید از قوانین زیر پیروی کند:

- گروه‌های تکراری مجاز نیستند.
- هر مجموعه از داده‌های مرتبط را با یک کلید اصلی شناسایی کنید.
- مجموعه‌داده‌های مرتبط باید یک جدول جداگانه داشته باشند.
- مخلوط کردن انواع داده‌ها در یک ستون مجاز نیست.

### 2NF (Second Normal Form)

برای اینکه یک جدول در فرم دوم نرمال (2NF) باشد، باید از قوانین زیر پیروی کند:

- شرایط فرم اول نرمال (2NF) را برآورده کند.
- نباید وابستگی جزئی داشته باشد.

### NF3 (Third Normal Form)

برای اینکه یک جدول در فرم سوم نرمال (3NF) باشد، باید از قوانین زیر پیروی کند:

- شرایط فرم دوم نرمال (3NF) را برآورده کند.

- وابستگی های تابعی متعدد مجاز نیستند.

### **BCNF (Boyce-Codd Normal Form)**

فرم نرم‌ال Boyce-Codd (یا BCNF) نسخه کمی قوی‌تر از فرم سوم نرم‌ال (3NF) است که برای رسیدگی به انواع خاصی از ناهنجاری‌ها که در تعریف اولیه 3NF به آنها پرداخته نشده، استفاده می‌شود. گاهی اوقات به آن 3.5NF نیز گفته می‌شود.

برای اینکه یک جدول در فرم نرم‌ال Boyce-Codd (BCNF) باشد، باید از قوانین زیر پیروی کند:

- شرایط فرم سوم نرم‌ال (3NF) را برآورده کند.
- برای هر وابستگی عاملی  $X \rightarrow Y$  باشد  $Y$  super key باشد.

فرم‌های نرم‌ال دیگری مانند 6NF، 5NF و 4NF نیز وجود دارند، اما در اینجا به آنها نمی‌پردازیم.

در یک پایگاهداده رابطه‌ای، اغلب گفته می‌شود یک رابطه «نرم‌ال» است اگر فرم سوم نرم‌ال را رعایت کند. اکثر روابط 3NF عاری از ناهنجاری‌های درج، به روزرسانی و حذف<sup>۱</sup> هستند. همان‌طور که در مورد بسیاری از قوانین و مشخصات رسمی وجود دارد، سناریوهای دنیای واقعی همیشه اجازه رعایت کامل را نمی‌دهند. اگر تصمیم دارید یکی از سه قانون اول نرم‌ال‌سازی را نقض کنید، مطمئن شوید که برنامه شما هر مشکلی که ممکن است رخ دهد، مانند داده‌های تکراری و وابستگی‌های ناسازگار را پیش‌بینی کند.

### **مزایای نرم‌ال‌سازی**

در اینجا برخی از مزایای نرم‌ال‌سازی آورده شده است:

<sup>۱</sup>insertion, update, and deletion

- کاهش افزونگی دادهها
- طراحی بهتر دادهها
- افزایش یکپارچگی دادهها
- اعمال یکپارچگی ارجاعی<sup>۱</sup>

### معایب نرمالسازی

بیایید به برخی از معایب نرمالسازی نگاهی بیندازیم:

- طراحی دادهها پیچیده است.
- عملکرد کندتر
- سربار نگهداری
- نیاز به اتصالات (joins) بیشتر دارد

### غیر نرمالسازی (Denormalization)

غیر نرمالسازی یک تکنیک بهینهسازی پایگاهداده است که در آن دادههای تکراری را به یک یا چند جدول اضافه میکنیم. این میتواند به ما کمک کند از اتصالات پرهزینه در یک پایگاهداده رابطه‌ای اجتناب کنیم. این روش در تلاش است تا عملکرد خواندن را به قیمت افت عملکرد نوشتمن بهبود بخشد. برای اجتناب از اتصالات پرهزینه، نسخه‌های تکراری از دادهها در چندین جدول نوشته میشوند.

زمانی که دادهها با تکنیک‌هایی مانند فدراسیون<sup>۲</sup> و شارдинگ<sup>۳</sup> توزیع می‌شوند، مدیریت اتصالات در سراسر شبکه، پیچیدگی را بیشتر میکند. غیر نرمالسازی ممکن است نیاز به

<sup>۱</sup> referential integrity

<sup>۲</sup> federation

<sup>۳</sup> sharding

چنین اتصالات پیچیده‌ای را دور بزند. توجه داشته باشید که غیر نرمال‌سازی به معنای معکوس کردن نرمال‌سازی نیست.

### مزایای غیر نرمال‌سازی

بیایید به برخی از مزایای غیر نرمال‌سازی نگاهی بیندازیم:

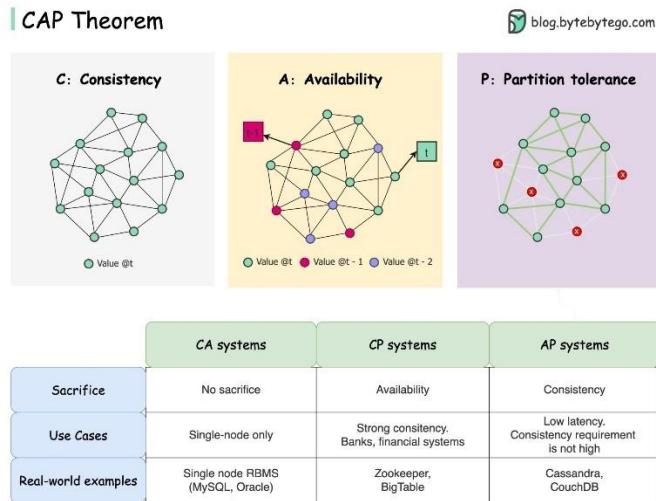
- بازیابی داده‌ها سریع‌تر است.
- نوشتمن پرس‌وجو آسان‌تر است.
- کاهش تعداد جداول
- مدیریت راحت‌تر
- معايب غیر نرمال‌سازی

در زیر برخی از معايب غیرعادی سازی آورده شده است:

- درجه‌ها و بهروزرسانی‌های گران‌قیمت.
- پیچیدگی طراحی پایگاه‌داده را افزایش می‌دهد.
- افزونگی داده‌ها را افزایش می‌دهد.
- احتمال ناهمانگی و ناسازگاری داده‌ها بیشتر است.

## CAP تئوری

قضیه CAP یکی از اصطلاحات بسیار مشهور در علوم کامپیوتر است، اما ممکن است توسعه‌دهندگان مختلف درک‌های متفاوتی از آن داشته باشند. بیایید بررسی کنیم که این قضیه چیست و چرا می‌تواند گیج‌کننده باشد.



قضیه CAP بیان می‌کند که یک سیستم توزیع شده نمی‌تواند به طور هم‌زمان بیش از دو مورد از این سه تضمین را ارائه دهد.

**یکپارچگی (Consistency):** سازگاری یا یکپارچگی به این معنی است که همه کاربران، صرف نظر از اینکه به کدام گره متصل می‌شوند، داده‌های یکسانی را به طور هم‌زمان می‌بینند.

**دسترس پذیری (Availability):** در دسترس‌پذیری به این معنی است که هر کاربری که درخواست داده می‌کند، پاسخی دریافت می‌کند، حتی اگر برخی از گره‌ها از کارافتاده باشند.

**تحمل پارتیشن (Partition Tolerance):** پارتیشن به معنای قطع ارتباط بین دو گره است. تحمل پارتیشن به این معنی است که سیستم همچنان به کار خود ادامه می‌دهد، حتی اگر پارتیشن‌های شبکه وجود داشته باشد.

فرمول‌بندی «دو مورد از سه» می‌تواند مفید باشد، اما این ساده‌سازی ممکن است گمراه‌کننده باشد.

۱. انتخاب یک پایگاه‌داده آسان نیست. توجیه انتخاب ما صرفاً بر اساس قضیه CAP

کافی نیست. برای مثال، شرکت‌ها صرفاً به دلیل اینکه **Cassandra** یک سیستم

است، آن را برای برنامه‌های چت انتخاب نمی‌کنند. لیستی از ویژگی‌های خوب

وجود دارد که کاساندرا را به گزینه مناسبی برای ذخیره پیام‌های چت تبدیل می‌کند.

ما باید عمیق‌تر بررسی کنیم.

۲. به نقل از مقاله «قضیه CAP دوازده سال بعد: چگونه «قوانين»<sup>۱</sup> تغییر کرده‌اند»:

(قضیه CAP تنها بخش کوچکی از فضای طراحی را ممنوع می‌کند: در

دسترس‌پذیری و سازگاری کامل در صورت وجود پارتبیشن‌ها که نادر هستند.»

۳. این قضیه در مورد ۱۰۰٪ در دسترس‌پذیری و یکپارچگی است. بحث واقع‌بینانه‌تر،

مبادلات بین تأخیر و یکپارچگی در زمانی که هیچ پارتبیشن شبکه‌ای وجود ندارد،

خواهد بود. برای جزئیات بیشتر به قضیه PACELC مراجعه کنید.

### آیا قضیه CAP واقعاً مفید است؟

فکر می‌کنم این قضیه همچنان مفید است؛ زیرا ذهن ما را به مجموعه‌ای از بحث‌های مبادله

باز می‌کند، اما این تنها بخشی از ماجراست. برای انتخاب پایگاه‌داده مناسب، باید عمیق‌تر

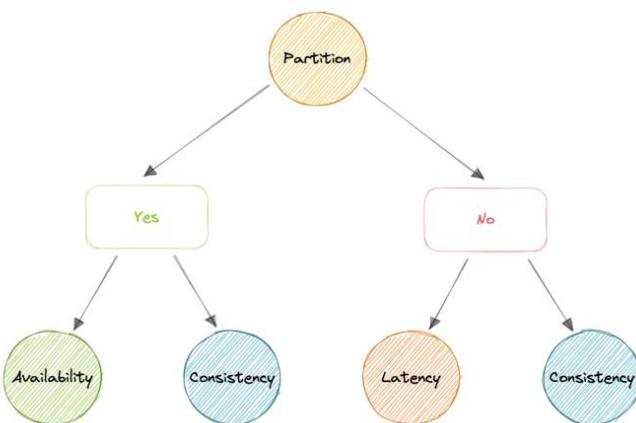
بررسی کنیم.

## PACELC تئوری

تئوری PACELC توسعه‌ای بر تئوری CAP است. تئوری CAP بیان می‌کند که در صورت بندی شبکه (Partition - P) در یک سیستم توزیع شده، باید بین دردسترس بودن (Availability - A) و یکپارچگی (Consistency - C) یکی را انتخاب کرد.

قضیه‌ی PACELC با معرفی تأخیر (Latency - L) به عنوان یک ویژگی اضافی یک سیستم توزیع شده، قضیه‌ی CAP را گسترش می‌دهد. این قضیه بیان می‌کند که در غیر این صورت (Else - E)، حتی زمانی که سیستم به طور عادی و بدون پارتیشن‌بندی اجرا می‌شود باید بین تأخیر (Latency - L) و یکپارچگی (Consistency - C) یکی را انتخاب کرد.

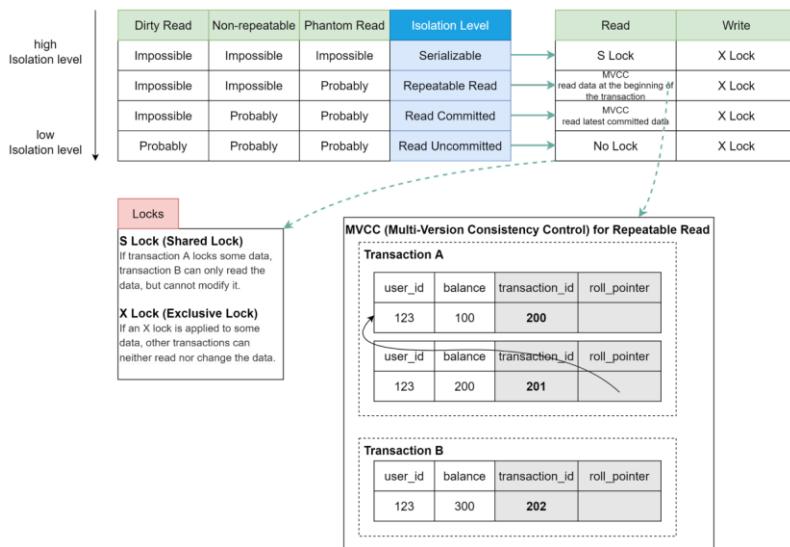
تئوری PACELC برای اولین بار توسط Daniel J. Abadi توصیف شد.



تئوری PACELC برای رفع محدودیت کلیدی قضیه‌ی CAP توسعه داده شد، زیرا هیچ تمهدی برای عملکرد یا تأخیر در نظر نمی‌گیرد. برای مثال، طبق تئوری CAP، یک پایگاهداده در صورتی در دسترس در نظر گرفته می‌شود که یک پرس و جو / کوئری پس از ۳۰ روز پاسخی را برگرداند. بدیهی است که چنین تأخیری برای هر اپلیکیشن دنیای واقعی غیرقابل قبول است.

## سطوح ایزولاسیون پایگاهداده

سطوح ایزولاسیون پایگاهداده چیست؟ و برای چه چیزی استفاده می‌شود؟ ایزولاسیون پایگاهداده به تراکنش‌ها اجازه می‌دهد که به عنوان اینکه هیچ تراکنش دیگری در حال اجرا نیست، اجرا شوند. نمودار زیر چهار سطح ایزولاسیون را نشان می‌دهد.



قابلیت سریالسازی: این بالاترین سطح ایزولاسیون است. تراکنش‌های همزمان تضمین می‌شود که به صورت متوالی اجرا شوند.

قابلیت تکرارپذیری خواندن: داده‌هایی که در طول تراکنش خوانده می‌شوند، همان‌طور که تراکنش شروع شده است، ثابت می‌مانند.

خواندن: تغییرات داده‌ها فقط پس از commit تراکنش قابل خواندن هستند.

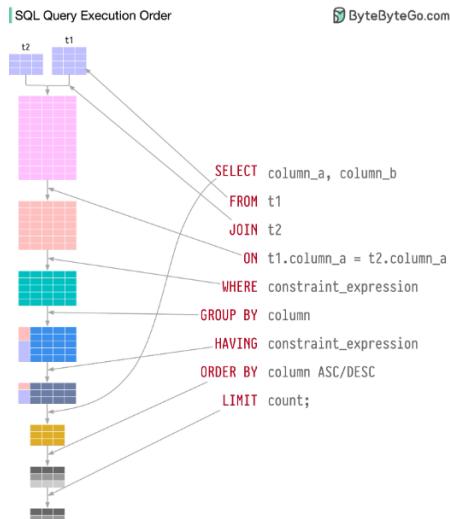
خواندن غیر commit شده: تغییرات داده‌ها می‌توانند توسط سایر تراکنش‌ها قبل از commit تراکنش خوانده شوند.

ایزوولاسیون توسط MVCC (کنترل سازگاری چند نسخه‌ای) و قفل‌ها تضمین می‌شود. به عنوان مثال نمودار زیر قابلیت تکرارپذیری در خواندن پایگاهداده را نشان می‌دهد تا نحوه کارکرد MVCC را نشان دهد:

دو ستون پنهان برای هر ردیف وجود دارد: transaction\_id و roll\_pointer. وقتی تراکنش A شروع می‌شود، یک نمای خواندن<sup>۱</sup> جدید با شناسه<sup>۲</sup> تراکنش = ۲۰۱ ایجاد می‌شود. به سرعت و پس از آن، تراکنش B شروع می‌شود و یک نمای خواندن جدید با شناسه تراکنش = ۲۰۲ ایجاد می‌شود.

اکنون تراکنش A مانده را به ۲۰۰ تغییر می‌دهد، یک ردیف جدید از لایگ ایجاد می‌شود و اشاره کننده‌ی roll\_pointer به ردیف قدیمی اشاره می‌کند. قبل از اینکه تراکنش A commit شود، تراکنش B داده مانده را می‌خواند. تراکنش B می‌یابد که شناسه تراکنش ۲۰۱ هنوز commit نشده است، پس ردیف بعدی commit شده (transaction\_id=200) را می‌خواند. حتی زمانی که تراکنش A commit می‌شود، تراکنش B همچنان بر اساس نمای خواندن ایجاد شده هنگام شروع تراکنش B داده را می‌خواند؛ بنابراین تراکنش B همیشه داده معادل = ۱۰۰ را می‌خواند.

## نمایش یک کوئری در SQL

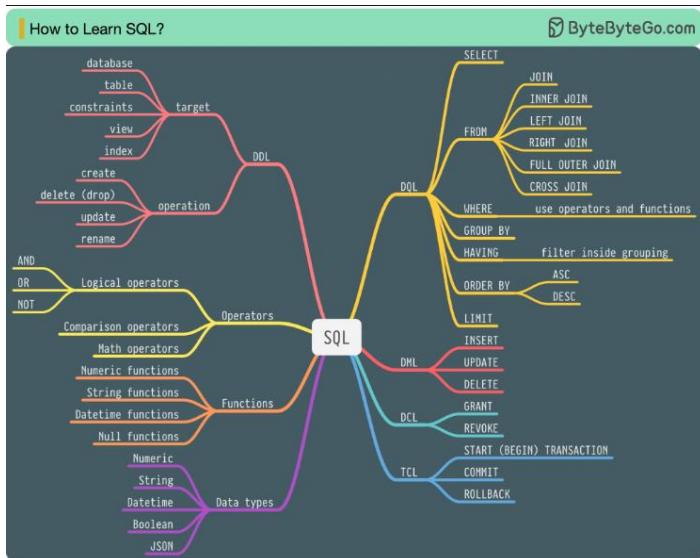


دستورهای SQL توسط سیستم پایگاهداده در چندین مرحله اجرا می‌شوند، از جمله:

- تجزیه دستور SQL و بررسی صحت آن.
  - تبدیل SQL به یک نمایش داخلی، مانند رابطه ریاضیاتی در جبر.<sup>۱</sup>
  - بهینه‌سازی نمایش داخلی و ایجاد یک برنامه اجرایی که از اطلاعات index استفاده می‌کند.
  - اجرای برنامه و بازگرداندن نتایج.
- اجرای SQL بسیار پیچیده است و شامل ملاحظات زیادی مانند:
- استفاده از شاخص‌ها<sup>۲</sup> و کش‌ها
  - ترتیب پیوستن جدول‌ها<sup>۳</sup>
  - کترل همزمان
  - مدیریت تراکنش

## SQL زبان

در سال ۱۹۸۶ SQL (زبان پرس‌وجوی ساختاریافته<sup>۱</sup>) به یک استاندارد تبدیل شد. در طول ۴۰ سال بعد، به زبان غالب برای سیستم‌های مدیریت پایگاهداده رابطه‌ای تبدیل شد. خواندن آخرین استاندارد (ANSI SQL 2016) است.



پنج مؤلفه از زبان SQL وجود دارد:

DML: زبان تعریف داده، مانند CREATE, ALTER, DROP

DQL: زبان پرس‌وجوی داده، مانند SELECT

DML: زبان دست‌کاری داده، مانند INSERT, UPDATE, DELETE

DCL: زبان کنترل داده، مانند GRANT, REVOKE

TCL: زبان کنترل تراکنش، مانند COMMIT, ROLLBACK

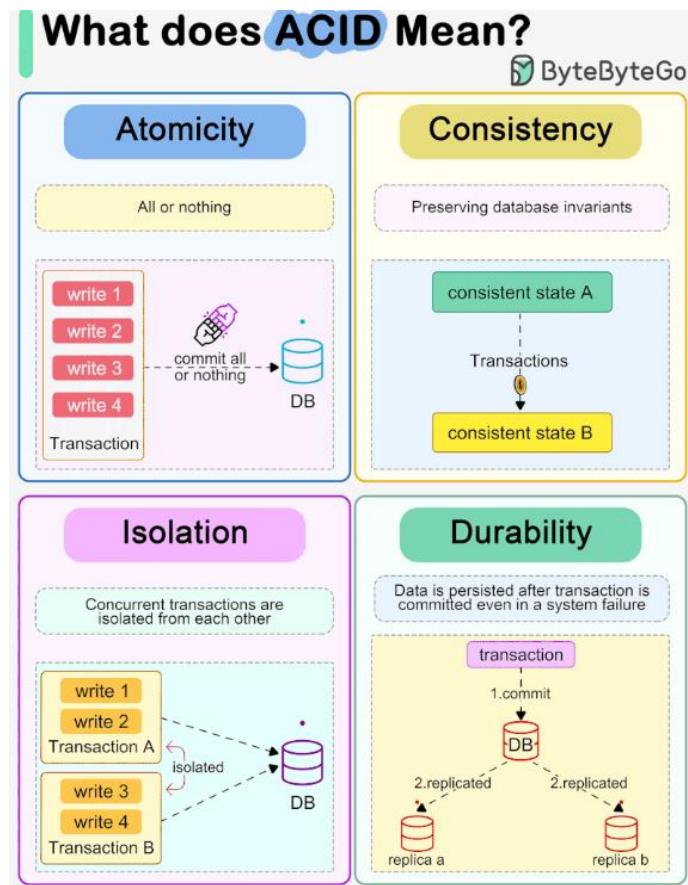
به عنوان یک مهندس backend، ممکن است نیاز داشته باشید که بیشتر این موارد را بدانید.

به عنوان یک تحلیلگر داده، ممکن است نیاز به درک خوبی از DQL داشته باشید. موضوعاتی

را انتخاب کنید که برای شما مرتبط‌تر هستند.

## بِ چه معناست؟ ACID

نمودار زیر مفهوم ACID را در متن تراکنش‌های پایگاه‌داده توضیح می‌دهد.



### : (Atomicity)

نوشته‌های داده در یک تراکنش پایگاه‌داده به صورت یکجا اجرا می‌شوند و نمی‌توان آنها را به بخش‌های کوچک‌تر تقسیم کرد. اگر در هنگام اجرای تراکنش خطایی رخ دهد، نوشته‌های تراکنش لغو و rollback می‌شوند.

بنابراین، اتمیته به معنای «همه یا هیچ» است.

### یکپارچگی (Consistency):

برخلاف مفهوم «یکپارچگی» در قضیه CAP که به معنای دریافت آخرین نسخه نوشته شده در هر بار خواندن یا دریافت خطا است، یکپارچگی در اینجا به معنای حفظ الزامات نامتغيرهای پایگاهداده<sup>۱</sup> است. هر داده‌ای که توسط یک تراکنش نوشته می‌شود، باید طبق تمام قوانین تعریف شده معتبر باشد و پایگاهداده را در یک وضعیت مطلوب نگه دارد.

### جداسازی (Isolation):

زمانی که هم‌زمان نوشتمنداده در پایگاهداده از دو تراکنش مختلف وجود داشته باشد، این دو تراکنش از یکدیگر جدا (ایزوله) می‌شوند. سخت‌گیرانه‌ترین جداسازی، «قابلیت سریال‌سازی» است، جایی که هر تراکنش عمل می‌کند انگار تنها تراکنش در حال اجرا در پایگاهداده است. با این حال، پیاده‌سازی این روش در عمل دشوار است، بنابراین ما اغلب از سطوح پایین‌تر جداسازی استفاده می‌کنیم.

### دوم (Durability):

داده‌ها حتی در صورت خرابی سیستم، پس از commit شدن تراکنش، همچنان باقی می‌مانند. در یک سیستم توزیع شده، این به معنای تکرار داده‌ها در برخی گره‌های دیگر است.

## BASE به چه معناست؟

با افزایش حجم داده و نیاز به دردسترس بودن بالا، رویکرد طراحی پایگاهداده نیز به طور چشمگیری تغییر کرده است. برای افزایش قابلیت مقیاس دهی و در عین حال دردسترس بودن بالا، ما منطق را از پایگاهداده به سرورهای جداگانه منتقل می کنیم. به این ترتیب، پایگاهداده مستقل تر شده و بر فرایند واقعی ذخیره سازی داده تمرکز می کند.

در دنیای پایگاهداده های NoSQL، تراکشن های ACID کمتر رایج هستند؛ زیرا برخی از پایگاههای داده، الزامات یکپارچگی فوری، تازگی داده و دقت را برای به دست آوردن مزایای دیگر مانند مقیاس پذیری و انعطاف پذیری کاهش داده اند. ویژگی های BASE بسیار سهل تر از ضمانت های ACID هستند، اما نگاشت یک به یک مستقیمی بین این دو مدل یکپارچگی وجود ندارد. باید این اصطلاحات را درک کنیم:

### (Basic Availability) دردسترس بودن پایه

پایگاهداده به نظر اکثر موقعیت کار می کند و در دسترس است.

### (Soft-state) وضعیت نرم

ذخیره سازها مجبور نیستند سازگاری و یکپارچگی سریع داده ها در سمت نوشتن داده ها داشته باشند و همچنین لزومی ندارد که نسخه های مختلف در تمام موقعیت سازگاری و یکپارچگی متقابل داشته باشند.

### (Eventual Consistency) یکپارچگی تدریجی

ممکن است داده ها بلافاصله در همه پایگاهداده های توزیع شده به صورت یکپارچه و یکسان نباشند، اما به صورت تدریجی و با گذشت زمان داده های در همه پایگاهداده های مستقر شده یکسان و یکپارچه می شوند. خواندن در سیستم حتی ممکن است به دلیل عدم سازگاری و یکپارچگی، پاسخ صحیح را ارائه ندهد، اما همچنان امکان پذیر است.

### تبادل ACID در برابر BASE

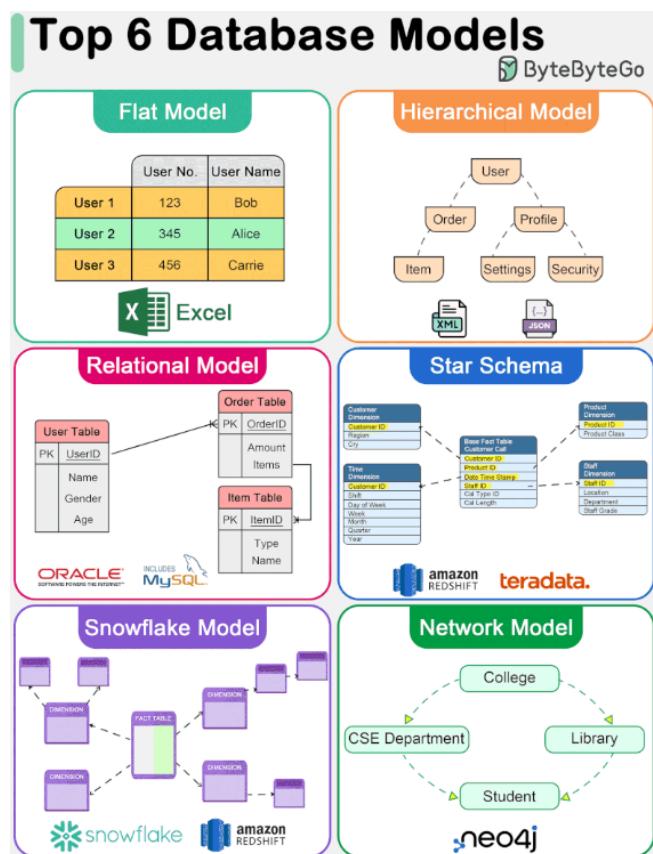
آیا برنامه‌ی ما به یک مدل یکپارچگی ACID یا BASE نیاز دارد یا نه پاسخ واضح و اصحی ندارد. هر دو مدل برای برآورده کردن نیازهای مختلف طراحی شده‌اند. هنگام انتخاب پایگاهداده، باید ویژگی‌های هر دو مدل و نیازهای برنامه‌ی خود را در نظر داشته باشیم.

با توجه به یکپارچگی ضعیف در ساختار BASE، توسعه‌دهندگان در صورت انتخاب یک ذخیره‌ساز BASE برای برنامه‌ی خود، باید دانش و دقت بیشتری در مورد داده‌های یکپارچه داشته باشند. آشنایی با رفتار BASE پایگاهداده‌ی انتخابی و کار با آن محدودیت‌ها ضروری است.

از طرف دیگر، برنامه‌ریزی در مورد محدودیت‌های BASE گاهی اوقات می‌تواند در مقایسه با سادگی تراکنش‌های ACID، یک نقطه‌ی ضعف عمله باشد. یک پایگاهداده‌ی کاملاً ACID برای مواردی که قابلیت اطمینان و یکپارچگی داده‌ها ضروری است گزینه‌ای کاملاً مناسب است.

## ۶ مدل برتر پایگاهداده

نمودار زیر، ۶ مدل برتر پایگاهداده را نشان می‌دهد.



### : (Flat Model)

مدل تخت یکی از ساده‌ترین انواع مدل‌های پایگاهداده است. این مدل، داده‌ها را در یک جدول واحد سازماندهی می‌کند که در آن هر سطر نشان‌دهنده یک رکورد و هر ستون نشان‌دهنده یک ویژگی است. این مدل شبیه به یک صفحه گسترده<sup>۱</sup> بوده و درک و پیاده‌سازی

آن ساده است. با این حال، این مدل توانایی مدیریت کارآمد روابط پیچیده بین موجودیت‌های داده<sup>۱</sup> را ندارد.

### مدل سلسله‌مراتبی (Hierarchical Model)

مدل سلسله‌مراتبی، داده‌ها را در یک ساختار درختی سازماندهی می‌کند، به‌گونه‌ای که هر رکورد یک والد (parent) واحد دارد اما می‌تواند چندین فرزند (child) داشته باشد. این مدل برای سناریوهایی با رابطه واضح «والد - فرزند<sup>۲</sup>» بین موجودیت‌های داده کارآمد است. با این حال، این مدل در مدیریت روابط «many-to-many» با مشکل مواجه می‌شود و می‌تواند پیچیده و انعطاف‌ناپذیر شود.

### مدل رابطه‌ای (Relational Model):

مدل رابطه‌ای که توسط E.F. Codd در سال ۱۹۷۰ معرفی شد، داده‌ها را در جداول رابطه‌ای نمایش می‌دهد که از سطراها و ستون‌ها تشکیل شده است. این مدل از طریق استفاده از کلیدها و نرمال‌سازی، از یکپارچگی داده<sup>۳</sup> پشتیبانی کرده و از تکرار<sup>۴</sup> داده‌ها جلوگیری می‌کند. قدرت مدل رابطه‌ای در انعطاف‌پذیری آن و سادگی زبان پرس‌وجوی/کوئری آن یعنی SQL (زبان پرس‌وجوی ساخت‌یافته<sup>۵</sup>) نهفته است که باعث می‌شود این مدل به براستفاده‌ترین مدل پایگاهداده برای سیستم‌های پایگاهداده سنتی تبدیل شود. این مدل به طور مؤثر روابط «many-to-many» را مدیریت می‌کند و از پرس‌وجوهای پیچیده و تراکنش‌ها پشتیبانی می‌کند.

### طرح ستاره (Star Schema):

data entities <sup>۱</sup>

parent-child <sup>۲</sup>

data integrity <sup>۳</sup>

redundancy <sup>۴</sup>

Structured Query Language <sup>۵</sup>

یک مدل داده تخصصی است که در انبارداری داده<sup>۱</sup> برای کاربردهای پردازش تحلیلی آنلاین (OLAP) استفاده می‌شود. این مدل شامل یک جدول مرکزی fact table است که حاوی داده‌های قابل اندازه‌گیری و مقداردهی است و با جداول «بعد»<sup>۲</sup> احاطه شده است که حاوی ویژگی‌های توصیفی مرتبط با داده‌های fact table هستند. این مدل برای عملکرد کوئیری در برنامه‌های تحلیلی بهینه‌سازی شده است و با به حداقل رساندن تعداد اتصالات (join) موردنیاز برای پرس‌وجوها، سادگی و بازیابی سریع داده را ارائه می‌دهد.

#### مدل دانه برفی (Snowflake Model)

مدل Snowflake گونه‌ای از اسکیما ستاره است که در آن جداول بعد به چندین جدول مرتبط نرمال‌سازی می‌شوند و بدین ترتیب، تکرار<sup>۳</sup> داده‌ها را کاهش داده و یکپارچگی داده<sup>۴</sup> را بهبود می‌بخشد. این منجر به ساختاری می‌شود که شبیه به یکدانه برفی است. در حالی که اسکیما دانه برفی به دلیل افزایش تعداد اتصالات (join) ممکن است منجر به پرس‌وجوهای پیچیده‌تری شود، اما از نظر کارایی ذخیره‌سازی مزایایی دارد و می‌تواند در سناریوهایی که جداول با ابعاد بزرگ یا به طور مکرر به روز می‌شوند، مفید باشد.

#### مدل شبکه‌ای (Network Model)

مدل داده شبکه‌ای به هر رکورد اجازه می‌دهد تا والدین و فرزندان متعددی داشته باشد و ساختاری گراف مانند را تشکیل دهد که می‌تواند روابط پیچیده بین موجودیت‌های داده را نشان دهد. این مدل با مدیریت کارآمد روابط «many-to-many»، بر برخی از محدودیت‌های مدل سلسله مراتبی غلبه می‌کند.

<sup>۱</sup> data warehousing

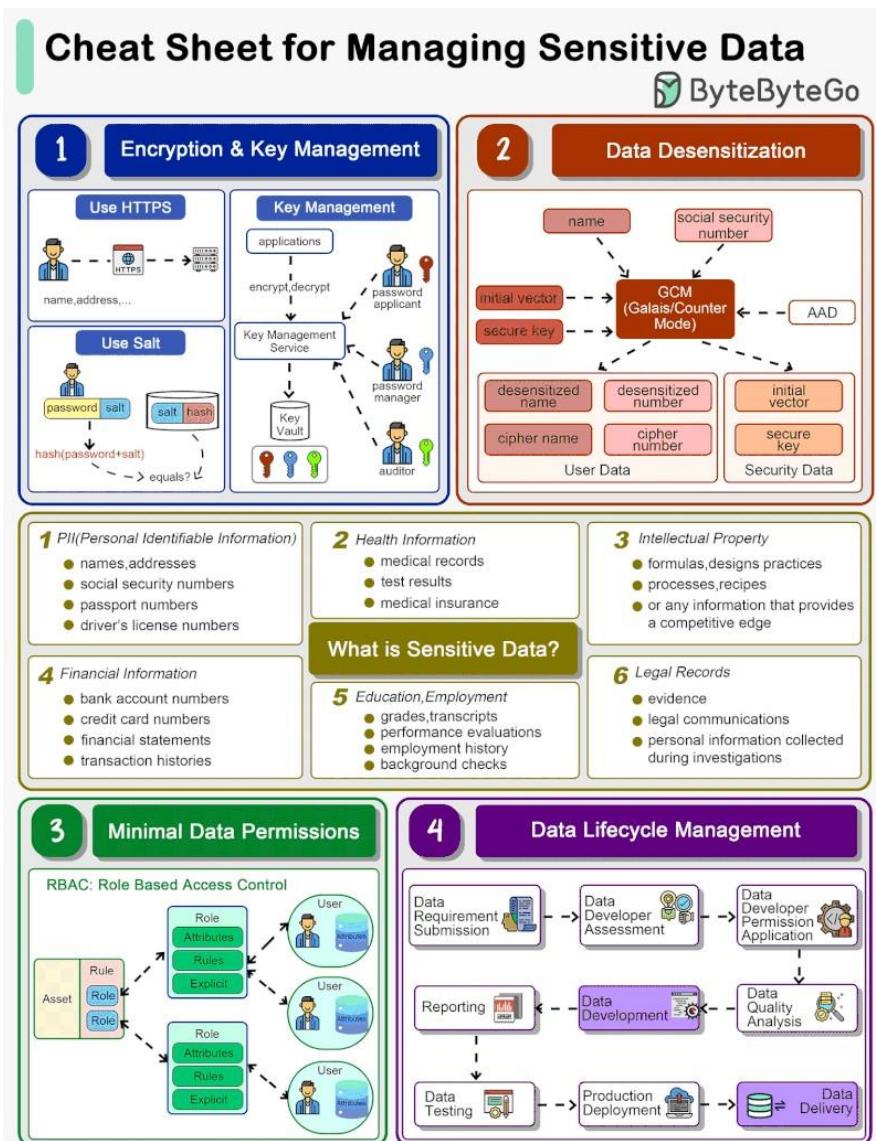
<sup>۲</sup> dimension table

<sup>۳</sup> redundancy

<sup>۴</sup> data integrity

## چگونه داده‌های حساس را در یک سیستم مدیریت کنیم؟

این تقلب نامه فهرستی از دستورالعمل‌ها را برای مدیریت داده‌های حساس در یک سیستم ارائه می‌کند.



## داده‌های حساس چیست؟

اطلاعات قابل شناسایی شخصی (PII)، اطلاعات سلامت، مالکیت فکری، اطلاعات مالی، سوابق تحصیلی و حقوقی، همگی داده‌های حساس هستند. بسیاری از کشورها قوانین و مقرراتی را برای الزام به محافظت از داده‌های حساس وضع کرده‌اند. برای مثال، «مقررات عمومی حفاظت از داده» (GDPR<sup>۲</sup>) در اتحادیه اروپا، قوانین سخت‌گیرانه‌ای را برای حفاظت از داده‌ها و حفظ حریم خصوصی تعیین می‌کند. عدم رعایت چنین مقرراتی می‌تواند منجر به جریمه‌های سنگین، اقدامات قانونی و تحریم علیه نهاد خاطی شود. هنگامی که سیستم‌ها را طراحی می‌کنیم، باید آن‌ها را برای حفاظت از داده‌ها طراحی کنیم.

## رمزگذاری و مدیریت کلید

انتقال داده باید با استفاده از SSL رمزگذاری شود. رمزهای عبور نباید به صورت متن ساده ذخیره شوند.

برای ذخیره‌سازی کلید، ما نقش‌های مختلفی را از جمله متقاضی رمز عبور، مدیر رمز عبور و حسابرس<sup>۳</sup> طراحی می‌کنیم که هر کدام یک بخش از کلید را در اختیار دارند. برای باز کردن قفل به هر سه کلید نیاز خواهیم داشت.

## حذف حساسیت داده‌ها (Data Desensitization)

حذف حساسیت داده که همچنین به عنوان ناشناس‌سازی داده<sup>۴</sup> یا پاک‌سازی داده<sup>۵</sup> شناخته می‌شود، به فرایند حذف یا اصلاح اطلاعات شخصی از یک مجموعه داده اشاره دارد تا افراد به راحتی قابل شناسایی نباشند. این عمل برای محافظت از حریم خصوصی افراد و اطمینان از

<sup>۱</sup> Personal Identifiable Information

<sup>۲</sup> General Data Protection Regulation

<sup>۳</sup> auditor

<sup>۴</sup> data anonymization

<sup>۵</sup> data sanitization

انطباق با قوانین و مقررات حفاظت از داده‌ها بسیار مهم است. حذف حساسیت داده اغلب هنگام به اشتراک گذاشتن داده‌ها به صورت خارجی، مانند تحقیق یا تجزیه و تحلیل آماری یا حتی به صورت داخلی در یک سازمان برای محدود کردن دسترسی به اطلاعات حساس، مورد استفاده قرار می‌گیرد.

الگوریتم‌هایی مانند GCM داده‌های رمزگذاری شده و کلیدها را به طور جداگانه ذخیره می‌کنند تا هکرها نتوانند داده‌های کاربر را رمزگشایی کنند.

### مجوزهای کمترین دسترسی به داده (Minimal Data Permissions)

برای محافظت از داده‌های حساس، باید حداقل مجوز را به کاربران اعطا کنیم. ما اغلب کنترل دسترسی مبتنی بر نقش (RBAC<sup>1</sup>) را برای محدود کردن دسترسی به کاربران مجاز بر اساس نقش‌هایشان در سازمان طراحی می‌کنیم. این یک مکانیزم کنترل دسترسی پرکاربرد است که مدیریت مجوزهای کاربر را ساده می‌کند و اطمینان می‌دهد که کاربران فقط به اطلاعات و منابعی که برای نقش‌هایشان ضروری است دسترسی دارند.

### مدیریت چرخه عمر داده (Data Lifecycle Management)

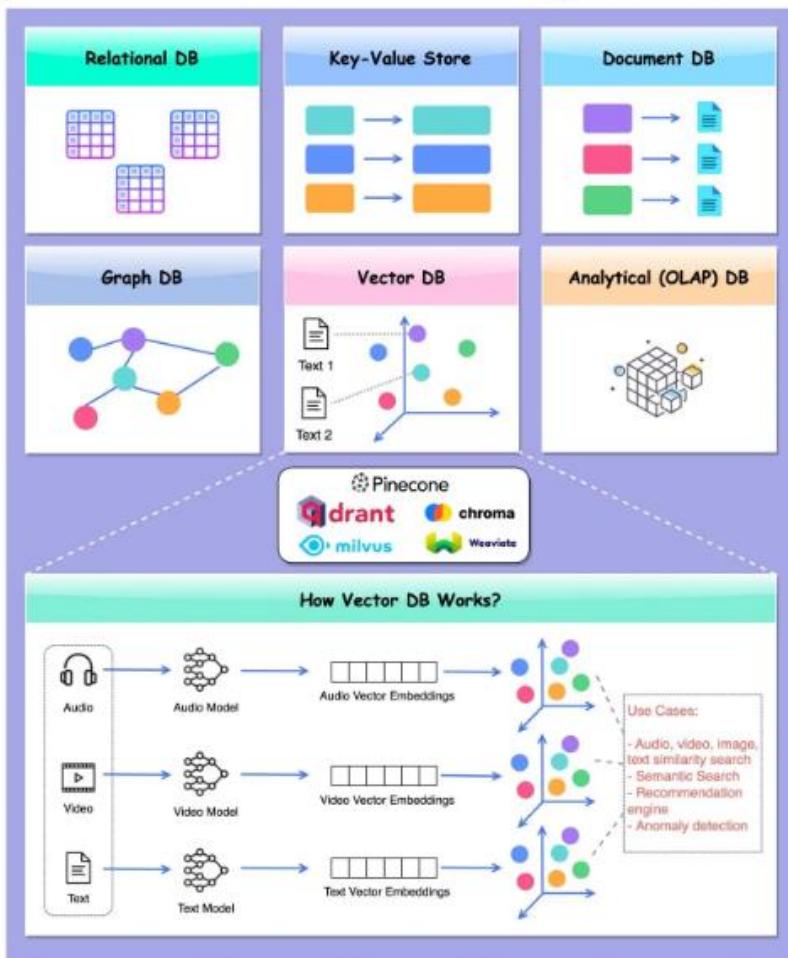
هنگامی که محصولات داده محور؛ مانند گزارش‌ها یا فیدهای داده را توسعه می‌دهیم، باید فرایندی برای حفظ کیفیت داده طراحی کنیم و به توسعه دهنده‌گان داده در هین توسعه باید مجوزهای لازم اعطا شود. پس از آنلاین شدن داده‌ها، این مجوزها باید لغو شوند.

## پایگاه‌داده‌های بُرداری

پایگاه‌داده‌های بُرداری<sup>۱</sup> در حال حاضر بسیار مطرح هستند، اما پایگاه‌داده بُرداری چیست؟ نمودار زیر مقایسه‌ای بین پایگاه‌داده بُرداری و سایر انواع پایگاه‌های داده را نشان می‌دهد.

### What is Vector DB?

 blog.bytebytogo.com



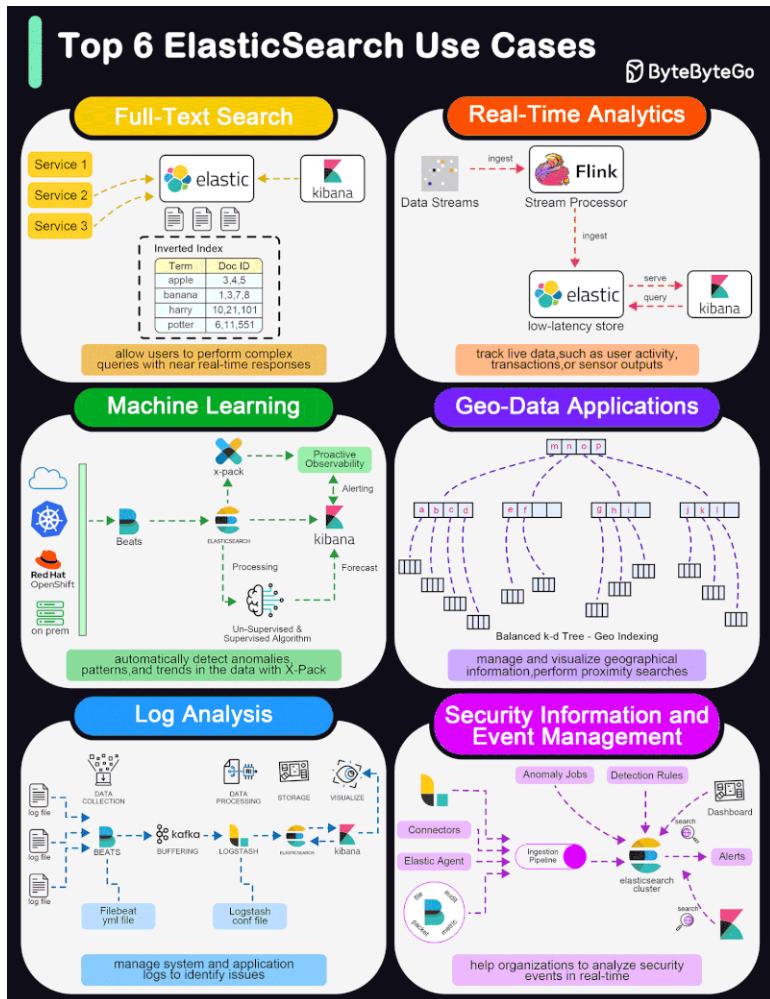
یک پایگاهداده برداری، جاسازی‌های برداری<sup>۱</sup> را برای بازیابی سریع و جستجوی شباهت فهرست‌بندی و ایندکس و ذخیره می‌کند و دارای قابلیت‌هایی مانند عملیات CRUD، فیلترکردن بر اساس متادیتا و مقیاس‌دهی افقی<sup>۲</sup> است.

پیشرفت‌های اخیر در هوش مصنوعی عمومی (AGI) باعث محبوبیت پایگاه‌های داده‌برداری شده است. یک پایگاهداده برداری، بردارهای با ابعاد بالا را که از داده‌های ساختارنیافرته مختلف مانند صدا، تصویر، ویدئو و متن استخراج شده‌اند را ذخیره می‌کند. سپس می‌توانیم شباهت بین داده‌های غیرساختاری را محاسبه کنیم. موارد استفاده معمولی عبارت‌انداز:

- یافتن تصاویر یا متون مشابه
- توصیه محصولات مشابه
- تشخیص ناهنجاری‌ها
- ذخیره موقت جاسازی‌ها برای حجم زیادی از ورودی

## ۶ مورد برتر از کاربردهای Elasticsearch

به دلیل قابلیت‌های جستجوی قدرتمند و همه‌کاره خود به طور گسترده‌ای Elasticsearch مورد استفاده قرار می‌گیرد. نمودار زیر ۶ مورد از مهم‌ترین موارد استفاده را نشان می‌دهد:



۱. جستجوی کامل متن (Full-Text Search): Elasticsearch به دلیل قابلیت‌های جستجوی قوی، مقیاس‌پذیر و سریع در سناریوهای جستجوی کامل متن می‌درخشد.

این امر به کاربران امکان می‌دهد تا پرسش‌های پیچیده را با پاسخگویی تقریباً بلاذرنگ انجام دهند.

۲. **تحلیل Real-Time:** قابلیت Elasticsearch برای انجام تحلیل لحظه‌ای، آن را برای داشبوردهایی که داده‌های زنده را ردیابی می‌کنند، مانند فعالیت کاربر، تراکنش‌ها یا خروجی‌های سنسور، مناسب می‌کند.

۳. **یادگیری ماشین:** با افزوده شدن ویژگی یادگیری ماشین در Elasticsearch، X-Pack می‌تواند به طور خودکار ناهنجاری‌ها، الگوها و روندها را در داده‌ها شناسایی کند.

۴. **اپلیکیشن‌های مبتنی بر Geo-Data:** Elasticsearch از طریق قابلیت‌های فهرست‌بندی و جستجوی مکانی، از داده‌های مکانی پشتیبانی می‌کند. این برای برنامه‌هایی که نیاز به مدیریت و تجسم اطلاعات جغرافیایی مانند نقشه‌برداری و خدمات مبتنی بر مکان دارند، مفید است.

۵. **تحلیل داده‌های Log و رویدادها:** سازمان‌ها از Elasticsearch برای جمع‌آوری، نظارت و تجزیه و تحلیل لگ‌ها و داده‌های رویداد از منابع مختلف استفاده می‌کنند. ELK Stack یک جزء کلیدی شامل Elasticsearch، Logstash و Kibana است که برای مدیریت لگ‌های سیستم و برنامه به منظور شناسایی مشکلات و نظارت بر سلامت سیستم محبوب است.

۶. **مدیریت اطلاعات و رویداد امنیتی<sup>۱</sup> (SIEM):** Elasticsearch را می‌توان به عنوان ابزاری برای SIEM استفاده کرد و به سازمان‌ها در تجزیه و تحلیل رویدادهای امنیتی به صورت لحظه‌ای کمک کند.

## تراکنش توزیع شده (Distributed Transactions)

تراکنش توزیع شده مجموعه‌ای از عملیات روی داده است که بر روی دو یا چند پایگاهداده انجام می‌شود. این تراکنش به طور معمول در سراسر گره‌های جداگانه‌ای که توسط یک شبکه به هم متصل شده‌اند، هماهنگ<sup>۱</sup> و هدایت می‌شود، اما ممکن است چندین پایگاهداده روی یک سرور واحد را نیز در بر گیرد.

چرا به تراکنش‌های توزیع شده نیاز داریم؟

برخلاف تراکنش ACID روی یک پایگاهداده واحد، یک تراکنش توزیع شده شامل تغییر داده‌ها روی چندین پایگاهداده است. در نتیجه، پردازش تراکنش توزیع شده پیچیده‌تر است، زیرا پایگاهداده باید تأیید یا لغو و بازگشت عقب بازگردد<sup>۲</sup> تغییرات در یک تراکنش را به عنوان یک واحد مستقل هماهنگ<sup>۳</sup> کند. به عبارت دیگر، همه گره‌ها باید تأیید کنند یا همه باید لغو کنند و کل تراکنش به عقب بازگردد<sup>۴</sup>. پس به همین دلیل است که به تراکنش‌های توزیع شده نیاز داریم.

حالا بباید به برخی از راه حل‌های رایج برای تراکنش‌های توزیع شده نگاهی بیندازیم:

### تأیید دو مرحله‌ای (Two-Phase Commit)

پروتکل Commit دو مرحله‌ای (PC2) یک الگوریتم توزیع شده است که همه فرایندهایی را که در یک تراکنش توزیع شده برای تأیید یا لغو و rollback تراکنش شرکت می‌کنند را هماهنگ می‌کند.

coordinated<sup>۱</sup>

rollback<sup>۲</sup>

coordinate<sup>۳</sup>

rollback<sup>۴</sup>

این پروتکل حتی در بسیاری از موارد خرابی موقت سیستم به هدف خود می‌رسد و از این‌رو به طور گسترده استفاده می‌شود. با این حال، در برابر همه پیکربندی‌های خرابی احتمالی مقاوم نیست و در موارد نادر، برای رفع نتیجه به دخالت دستی نیاز است. این پروتکل به یک گره هماهنگ‌کننده<sup>۱</sup> نیاز دارد که اساساً بر تراکنش در گره‌های مختلف نظارت و هماهنگی ایجاد می‌کند. هماهنگ‌کننده سعی می‌کند در دو مرحله اجماع<sup>۲</sup> را بین مجموعه‌ای از فرایندها برقرار کند، از این‌رو نام آن دو مرحله‌ای است.

### فازها

تأیید دو مرحله‌ای از مراحل زیر تشکیل شده است:

#### فاز آماده‌سازی (Prepare phase)

فاز آماده‌سازی شامل جمع‌آوری اجماع از هر یک از گره‌های شرکت‌کننده توسط گره هماهنگ‌کننده است. مگر اینکه هر یک از گره‌ها اعلام کنند که آماده هستند، تراکنش لغو خواهد شد.

#### فاز تأیید (Commit phase)

اگر همه شرکت‌کنندگان به هماهنگ‌کننده پاسخ دهند که آماده هستند، سپس هماهنگ‌کننده از همه گره‌ها می‌خواهد تا تراکنش را تأیید و commit کنند. اگر خرابی رخ دهد، تراکنش لغو خواهد شد (rollback).

### معایب

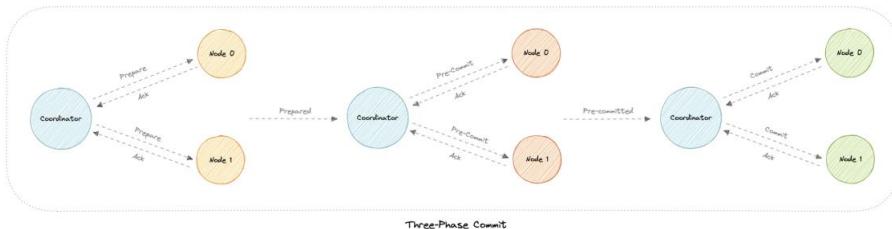
مشکلات زیر ممکن است در پروتکل تأیید دو مرحله‌ای ایجاد شوند:

coordinator<sup>۱</sup>

consensus<sup>۲</sup>

- اگر یکی از گره‌ها خراب شود چه اتفاقی می‌افتد؟
- اگر خود هماهنگ‌کننده خراب شود چه اتفاقی می‌افتد؟
- این یک پروتکل مسدودکننده است!

### تأیید سه مرحله‌ای (Three-Phase Commit)



تأیید سه مرحله‌ای (PC<sup>3</sup>) توسعه‌ای از تأیید دو مرحله‌ای است که در آن فاز تأیید به دو فاز تقسیم می‌شود. این به حل مشکل مسدودشدن<sup>۲</sup> که در پروتکل تأیید دو مرحله‌ای رخ می‌دهد، کمک می‌کند.

#### فازها

Commit و تأیید سه مرحله‌ای از مراحل زیر تشکیل شده است:

#### فاز آماده‌سازی (Prepare phase)

این فاز همانند فاز آماده‌سازی در تأیید دو مرحله‌ای است.

#### فاز پیش - تأیید (Pre-commit phase)

همانگ‌کننده پیام پیش - تأیید را صادر می‌کند و همه گره‌های شرکت‌کننده باید آن را تأیید کنند. اگر یک شرکت‌کننده نتواند این پیام را به موقع دریافت کند، تراکنش لغو می‌شود.

<sup>۱</sup> blocking protocol

<sup>۲</sup> blocking

### فاز تأیید (Commit phase)

این مرحله نیز مشابه پروتکل تأیید دو مرحله‌ای است.

چرا فاز پیش - تأیید مفید است؟

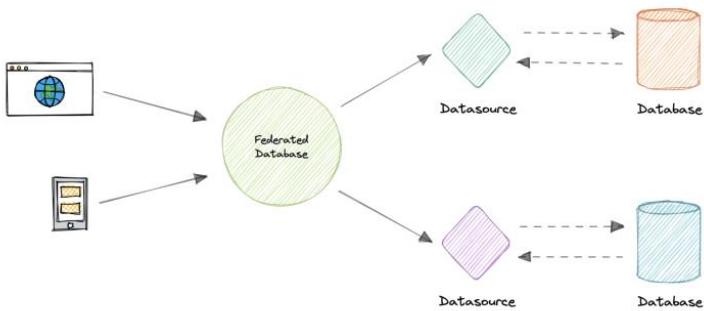
فاز پیش - تأیید موارد زیر را انجام می‌دهد:

اگر گره‌های شرکت‌کننده در این فاز یافت شوند، به این معنی است که هر شرکت‌کننده فاز اول را تکمیل کرده است و تکمیل فاز آماده‌سازی تضمین شده است.

اکنون هر فاز می‌تواند زمانش تمام و دچار timeout شود و از انتظارهای نامحدود جلوگیری کند.

## فدراسیون پایگاهداده (Database Federation)

فدراسیون پایگاهداده (یا پارتیشن‌بندی فانکشنال<sup>۱</sup>) پایگاههای داده را بر اساس عملکرد (Function) تقسیم می‌کند. معماری فدراسیون باعث می‌شود چندین پایگاهداده فیزیکی مجزا، به عنوان یک پایگاهداده منطقی واحد برای کاربران نهایی ظاهر شوند.



تمام اجزا در یک فدراسیون توسط یک یا چند طرح‌واره فدرال<sup>۲</sup> به هم مرتبط می‌شوند که اشتراک داده‌ها در سراسر فدراسیون را نشان می‌دهند. این Schema‌های فدرال برای مشخص کردن اطلاعاتی که می‌توان توسط اجزای فدراسیون به اشتراک گذاشته شود و برای فراهم کردن یک پایه‌ی مشترک برای ارتباط بین آنها استفاده می‌شوند. فدراسیون همچنین یک نمای کلی و متحده از داده‌های حاصل از منابع متعدد ارائه می‌دهد. منابع داده برای سیستم‌های فدرال می‌توانند شامل پایگاههای داده و اشکال مختلف دیگر داده‌های ساختاریافته و غیرساختاریافته باشند.

### ویژگی‌ها:

- شفافیت (Transparency): پایگاهداده فدرال تفاوت‌های کاربر و پیاده‌سازی‌های منابع داده زیربنایی را پنهان می‌کند؛ بنابراین، کاربران نیازی به دانستن محل ذخیره داده‌ها ندارند.

<sup>۱</sup> functional partitioning

<sup>۲</sup> Federal Schema

- ناهمگنی (Heterogeneity): منابع داده می‌توانند از جهات مختلفی متفاوت باشند.  
یک سیستم پایگاهداده فدرال می‌تواند سخت‌افزار، پروتکل‌های شبکه، مدل‌های داده و غیره را مدیریت کند.
- قابلیت توسعه (Extensibility): ممکن است برای پاسخگویی به نیازهای در حال تغییر کسب‌وکار به منابع جدید نیاز باشد. یک سیستم پایگاهداده فدرال خوب نیاز دارد که افزودن منابع جدید را آسان کند.
- استقلال (Autonomy): پایگاهداده فدرال منابع داده موجود را تغییر نمی‌دهند، رابطه‌ها باید بدون تغییر باقی بمانند.
- یکپارچه‌سازی داده‌ها (Data Integration): یک پایگاهداده فدرال می‌تواند داده‌ها را از پروتکل‌های مختلف، سیستم‌های مدیریت پایگاهداده و غیره ادغام کند.

#### مزایا:

در اینجا برخی از مزایای پایگاه‌های داده فدرال آورده شده است:

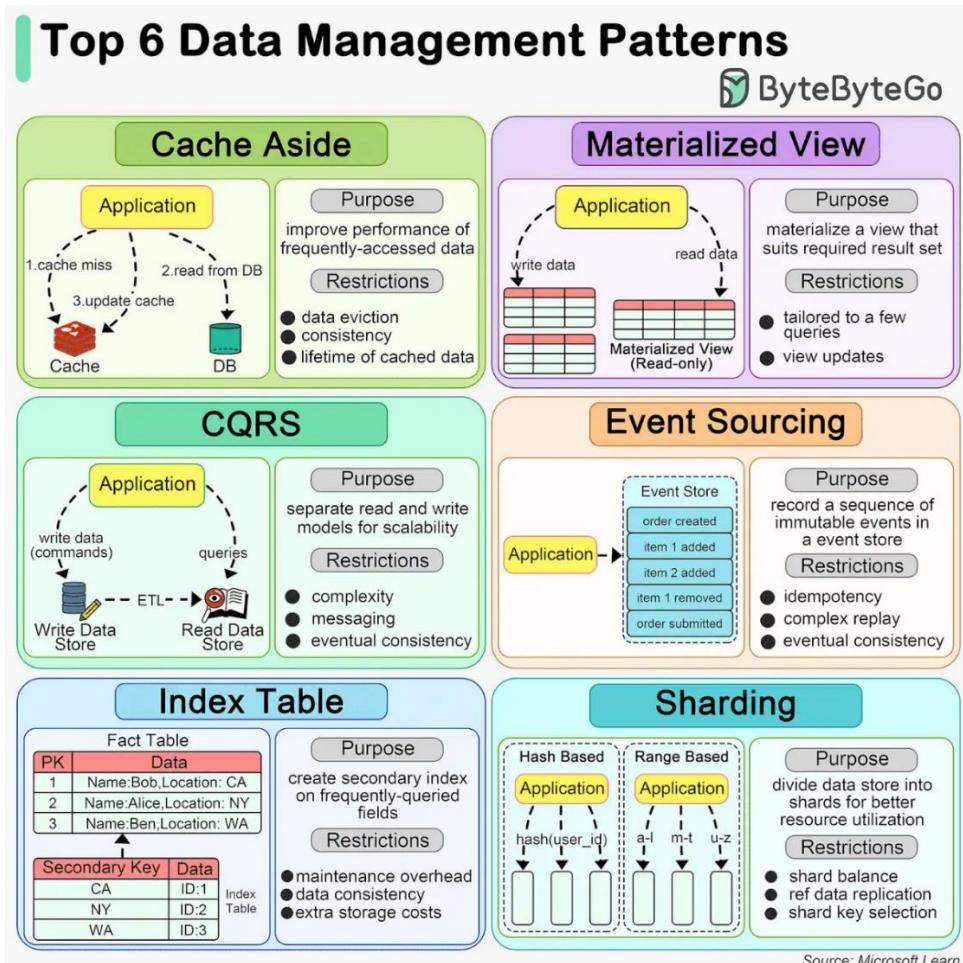
- اشتراک داده انعطاف‌پذیر
- استقلال بین اجزای پایگاهداده
- دسترسی به داده‌های ناهمگن به صورت یکپارچه
- عدم وابستگی شدید Application به پایگاه‌های داده قدیمی

#### معایب:

- افزودن سخت‌افزار بیشتر و پیچیدگی بیشتر.
- الحق داده<sup>۱</sup> از دو پایگاهداده پیچیده است.
- وابستگی به منابع داده مستقل.
- عملکرد کوئری<sup>۲</sup> و مقیاس‌پذیری

## چگونه داده‌ها را در سیستم‌های توزیع شده مدیریت کنیم؟

در اینجا ۶ الگوی برتر مدیریت داده آورده شده است



۱. **Cache Aside:** زمانی که یک برنامه نیاز به دسترسی به داده دارد، ابتدا حافظه کش را بررسی می‌کند. اگر داده موجود نباشد (عدم وجود کش)، داده‌ها را از مخزن داده دریافت می‌کند، آن را در کش ذخیره می‌کند و سپس داده‌ها را به کاربر باز می‌گرداند. این الگو

به طور خاص برای سناریوهایی مفید است که در آنها داده‌ها به طور مکرر خوانده می‌شوند؛ اما کمتر به روزرسانی می‌شوند.

**۲. نمای مادی Materialized View:** یک شیء پایگاهداده است که حاوی نتایج یک پرس‌وجو/کوئری است. این مورد به صورت فیزیکی ذخیره می‌شود به این معنی که داده‌ها در واقع محاسبه شده و روی دیسک ذخیره می‌شوند به جای اینکه در هر درخواست به صورت پویا ایجاد شوند. این کار می‌تواند زمان پرس‌وجو/کوئری برای محاسبات پیچیده یا تجمع‌هایی را که در غیر این صورت نیاز به محاسبه در لحظه دارند را به طور قابل توجهی سرعت بخشد. **Materialized View** به ویژه در اپلیکیشن‌های داده<sup>۱</sup> و سناریوهای هوش تجاری که عملکرد پرس‌وجو مهم است، موردی مناسب هستند.

**۳. CQRS: الگوی CQRS** یک الگوی معماری است که مدل‌های خواندن و نوشتن داده را از هم جدا می‌کند. این بدان معناست که ساختارهای داده‌ای که برای پرس‌وجو/کوئری (خواندن) داده‌ها استفاده می‌شوند از ساختارهای مورد استفاده برای به روزرسانی داده‌ها (نوشتن) جدا هستند. این جداسازی به امکان بهینه‌سازی هر عملیات به صورت مستقل، بهبود عملکرد، مقیاس‌پذیری و امنیت منجر می‌شود. CQRS می‌تواند به ویژه در سیستم‌های پیچیده‌ای که عملیات خواندن و نوشتن الزامات بسیار متفاوتی دارند، مفید باشد.

**۴. منع رویداد (Event Sourcing):** منع رویداد الگویی است که در آن تغییرات در وضعیت برنامه به صورت توالی رویدادها ذخیره می‌شود. به جای اینکه فقط وضعیت فعلی داده‌ها را در یک دامنه ذخیره کنید، Event Sourcing یک log از تمام تغییرات (رویدادها) را که در طول زمان رخداده است را ذخیره می‌کند. این کار امکان بازسازی وضعیت‌های گذشته را برای برنامه فراهم می‌کند و یک دنباله قبالتی بررسی از تغییرات را

ارائه می‌دهد. Event Sourcing در سناریوهایی که نیاز به تراکنش‌های تجاری پیچیده، قابلیت بررسی و امکان rollback یا تکرار رویدادها دارند، بسیار مناسب است.

**۵. جدول شاخص (Index Table):** الگوی جدول ایندکس شامل ایجاد جدول‌های اضافی در پایگاهداده است که برای عملیات کوئری خاصی بهینه‌سازی شده‌اند. این جداول به عنوان شاخص‌های ثانویه عمل می‌کنند و برای سرعت‌بخشیدن به بازیابی داده‌ها بدون نیاز به اسکن کامل مخزن داده اصلی طراحی شده‌اند. جدول‌های ایندکس به طور خاص در سناریوهایی با مجموعه‌داده‌های بزرگ و جایی که پرس‌وجوهای خاصی به طور مکرر انجام می‌شوند، مفید هستند.

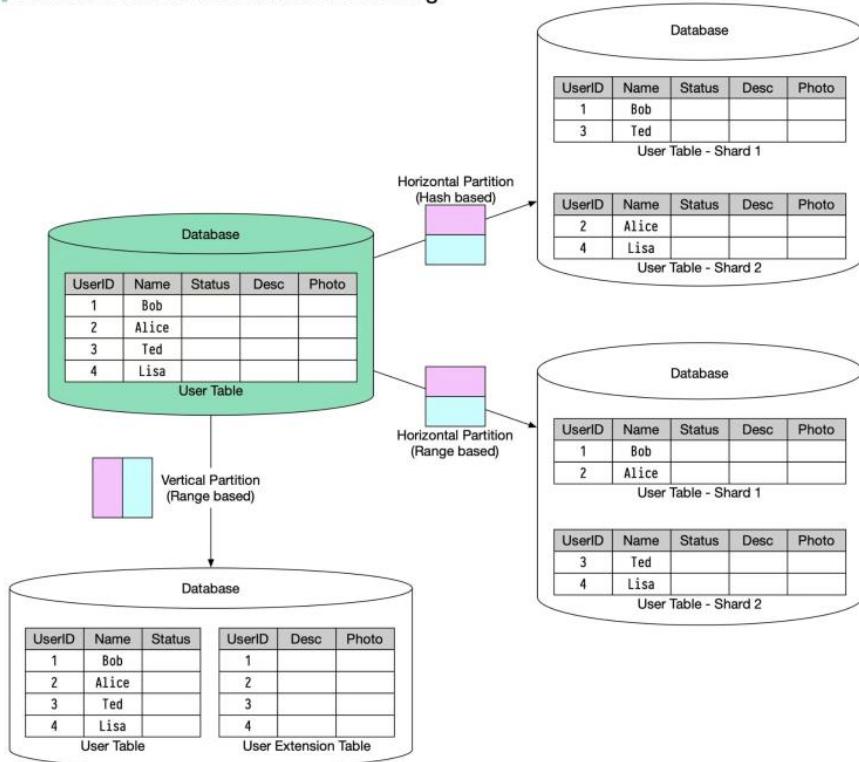
**۶. Sharding:** الگوی Sharding یک الگوی پارتیشن‌بندی داده است که در آن داده‌ها به قطعات کوچک‌تر و قابل مدیریت‌تر یا در اصطلاح «شاردها» تقسیم می‌شوند که هر کدام می‌توانند روی سرورهای پایگاه‌داده مختلف ذخیره شوند. از این الگو برای توزیع داده‌ها در چندین ماشین برای بهبود مقیاس‌پذیری و عملکرد استفاده می‌شود. شارдинگ به طور خاص در برنامه‌های با حجم بالا مؤثر است، زیرا به مقیاس‌بندی افقی، توزیع بار در چندین سرور برای رسیدگی به کاربران و تراکنش‌های بیشتر، امکان می‌دهد.

## پارتیشن‌بندی افقی و پارتیشن‌بندی عمودی

در بسیاری از برنامه‌های مقیاس بزرگ، داده‌ها به بخش‌هایی تقسیم می‌شوند که می‌توانند به طور جداگانه دسترسی داشته باشند. دو استراتژی معمول برای بخش‌بندی / پارتیشن‌بندی داده‌ها وجود دارد.

- پارتیشن‌بندی عمودی: به این معنی که برخی از ستون‌ها به جدول‌های جدید منتقل می‌شوند. هر جدول حاوی همان تعداد ردیف اما ستون‌های کمتر است (نمودار زیر را ببینید).
- پارتیشن‌بندی افقی (که اغلب شارдинگ<sup>۱</sup> نامیده می‌شود): یک جدول را به چندین جدول کوچک‌تر تقسیم می‌کند. هر جدول یک ذخیره‌گاه داده‌ای جداگانه است و همان تعداد ستون‌ها را دارد، اما تعداد ردیف‌ها کمتر است (نمودار زیر را ببینید).

### Vertical & Horizontal Database Sharding



پارتیشن‌بندی افقی به طور گسترده‌ای استفاده می‌شود، بنابراین اجازه دهد نگاهی دقیق‌تر داشته باشیم.

### الگوریتم مسیریابی

الگوریتم مسیریابی تصمیم می‌گیرد که کدام پارتیشن (shard) داده‌ها را ذخیره می‌کند. پارتیشن‌بندی بر اساس محدوده<sup>۱</sup>! این الگوریتم از ستون‌های مرتب شده، مانند اعداد صحیح، اعداد بزرگ و timestamp ها، برای جداکردن ردیف‌ها استفاده می‌کند. برای مثال، نمودار زیر از ستون شناسه کاربر برای پارتیشن‌بندی محدوده استفاده می‌کند: شناسه‌های کاربر ۱ و ۲ در بخش ۱ هستند، شناسه‌های کاربر ۳ و ۴ در بخش ۲ هستند.

### پارتبیشن‌بندی بر اساس هش

این الگوریتم یک تابع هش را به یک ستون یا چندین ستون اعمال می‌کند تا تصمیم بگیرد که کدام ردیف به کدام جدول برود. برای مثال در نمودار قبل از "User ID mod 2"<sup>۱</sup> به عنوان یک تابع هش استفاده می‌کند. شناسه‌های کاربر ۱ و ۳ در بخش<sup>۲</sup> شماره ۱ هستند، شناسه‌های کاربر ۲ و ۴ در بخش شماره ۲ هستند.

### پارتبیشن‌بندی مبتنی بر لیست

در پارتبیشن‌بندی مبتنی بر لیست، هر پارتبیشن بر اساس لیستی از مقادیر روی یک ستون به جای مجموعه‌ای از بازه‌های پیوسته‌ی مقادیر، تعریف و انتخاب می‌شود.

### پارتبیشن‌بندی مبتنی بر بازه<sup>۳</sup>

پارتبیشن‌بندی مبتنی بر بازه یا Range، داده‌ها را بر اساس بازه‌های مقادیر کلید پارتبیشن‌بندی به پارتبیشن‌های مختلف نگاشت می‌کند. به عبارت دیگر، ما جدول را به گونه‌ای پارتبیشن‌بندی می‌کنیم که هر پارتبیشن حاوی سطرهایی در یک بازه‌ی خاص باشد که توسط کلید پارتبیشن‌بندی تعریف می‌شود. بازه‌ها باید پیوسته باشند؛ اما همپوشانی نداشته باشند، جایی که هر بازه یک مرز پایین و بالای غیرمشمول را برای یک پارتبیشن مشخص می‌کند. هر مقدار کلید پارتبیشن‌بندی که مساوی یا بالاتر از مرز بالای بازه باشد به پارتبیشن بعدی اضافه می‌شود.

shard<sup>۱</sup>

Range<sup>۲</sup>

### پارتیشن‌بندی ترکیبی<sup>۱</sup>

همان‌طور که از نام آن پیداست، پارتیشن‌بندی ترکیبی، داده‌ها را بر اساس دو یا چند تکنیک پارتیشن‌بندی تقسیم می‌کند. در اینجا ابتدا داده‌ها را با استفاده از یک تکنیک پارتیشن‌بندی می‌کنیم و سپس هر پارتیشن با استفاده از همان روش یا روش دیگری به زیرمجموعه‌های پارتیشن‌بندی تقسیم می‌شود.

### مزایا sharding

امکان مقیاس‌بندی افقی را تسهیل می‌کند. پارتیشن‌بندی امکان افزودن ماشین‌هایی بیشتر برای توزیع بار را فراهم می‌کند.

زمان پاسخ‌دهی را کوتاه می‌کند. با پارتیشن‌بندی یک جدول به چندین جدول، کوئری‌ها از میان تعداد کمتری ردیف عبور می‌کنند و نتایج بسیار سریع‌تر بازگردانده می‌شوند.

### معایب sharding

- عملیات مرتب‌سازی پیچیده‌تر است. معمولاً لازم است داده‌ها از بخش‌های مختلف بازیابی شده و در کد برنامه مرتب شوند.
- توزیع غیریکنواخت. برخی از shashardها ممکن است داده‌های بیشتری نسبت به سایر shashardها داشته باشند (به این حالت hotspot نیز گفته می‌شود).

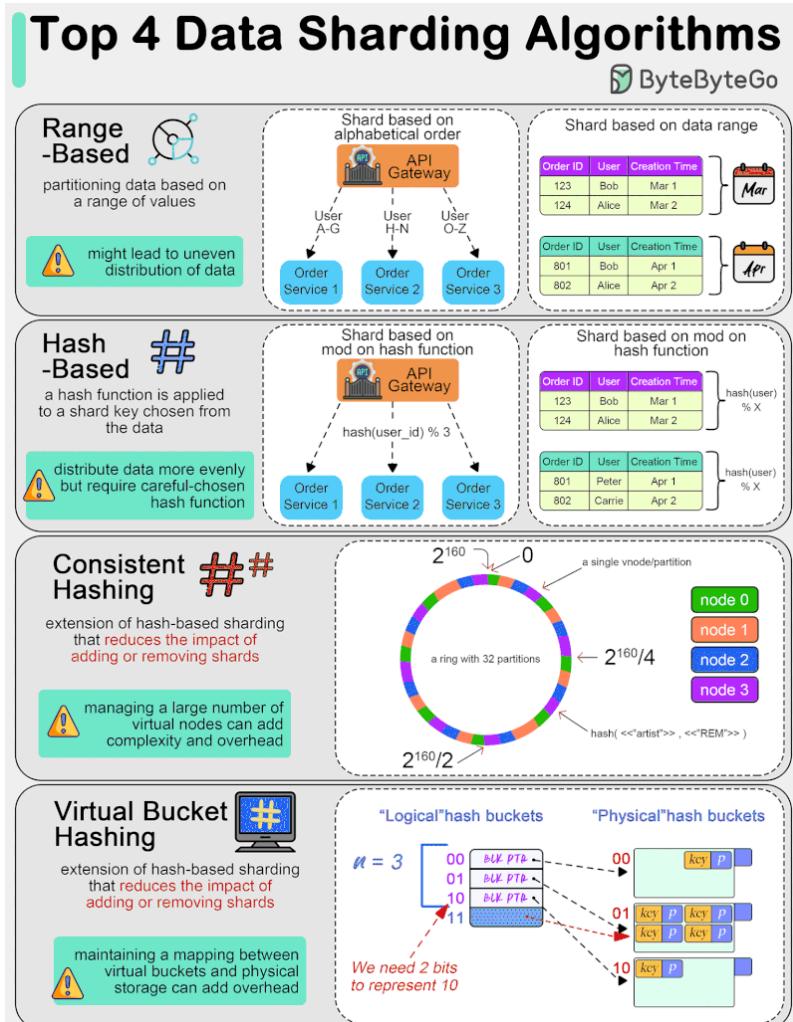
### چه زمانی از Sharding استفاده کنیم؟

- استفاده از سخت‌افزار موجود به جای سخت‌افزارهای پیشرفته‌تر.
- نگهداری داده‌ها در مناطق جغرافیایی مجزا.
- مقیاس‌پذیری سریع با افزودن shashardهای بیشتر.
- عملکرد بهتر به دلیل اینکه هر ماشین بار کمتری را تحمل می‌کند.

- زمانی که اتصالات هم زمان بیشتری موردنیاز است.

## ۴ الگوریتم برتر Sharding

با حجم عظیمی از داده‌ها سروکار داریم. اغلب نیاز داریم داده‌ها را به قطعات کوچک‌تر و قابل مدیریت‌تری به نام «تکه» یا Shard تقسیم کیم. در اینجا برخی از الگوریتم‌های Sharding داده که به طور معمول استفاده می‌شوند، آورده شده است:



### **:Range-Based Sharding**

این روش شامل پارتیشن‌بندی داده‌ها بر اساس محدوده‌ای از مقادیر است. به عنوان مثال، داده‌های مشتری را می‌توان بر اساس حروف الفباوی نام خانوادگی یا داده‌های تراکنش را بر اساس بازه‌های زمانی Sharding کرد.

### **:Hash-Based Sharding**

در این روش، یک تابع درهم‌سازی (Hash Function) روی یک کلید Shard انتخاب شده از داده‌ها (مانند شناسه مشتری یا شناسه تراکنش) اعمال می‌شود.

این روش تمایل دارد داده‌ها را نسبت به Sharding مبتنی بر محدوده به طور یکنواخت‌تری در سراسر Shard‌ها توزیع کند. با این حال، برای اجتناب از تصادم درهم‌سازی (Collision)، باید یک تابع درهم‌سازی مناسب انتخاب کنیم.

### **:Consistent Hashing**

این روش، توسعه‌ای از Sharding مبتنی بر درهم‌سازی است که تأثیر اضافه یا حذف Shard‌ها را کاهش می‌دهد. این روش داده‌ها را به طور یکنواخت‌تری توزیع می‌کند و میزان داده‌ای را که هنگام اضافه یا حذف تکه‌ها نیاز به جابه‌جایی دارند، به حداقل می‌رساند.

### **:Virtual Bucket Sharding**

داده‌ها به پیمانه‌های مجازی نگاشت<sup>۱</sup> می‌شوند و سپس این پیمانه‌ها به Shard‌های فیزیکی نگاشت می‌شوند. این نگاشت دوستخی، مدیریت انعطاف‌پذیرتر تکه‌ها و توازن مجدد بدون جابه‌جایی قابل توجه داده‌ها را فراهم می‌کند.

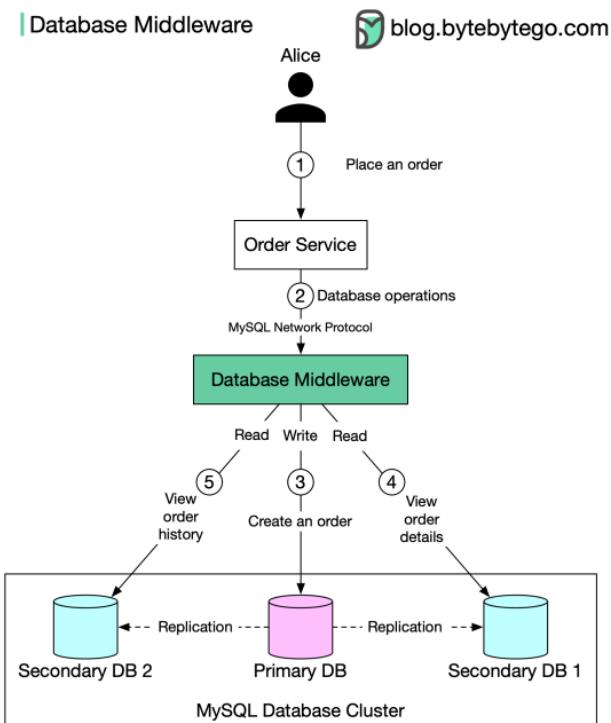
## الگوهای خواندن داده‌های replica شده

دو روش رایج برای پیاده‌سازی الگوی نسخه replica خواندن وجود دارد:

۱. تعییه منطق مسیریابی در کد برنامه.
۲. استفاده از میان‌افزار پایگاهداده.

در اینجا روی گزینه ۲ تمرکز می‌کنیم. میان‌افزار مسیریابی واضحی بین برنامه و سرورهای پایگاهداده را فراهم می‌کند. می‌توانیم منطق مسیریابی را بر اساس قواعد خاصی مانند کاربر، طرح، دستورات و غیره سفارشی کنیم.

نمودار زیر این تنظیمات را نشان می‌دهد:



۱. زمانی که آلیس سفارشی در آمازون ثبت می‌کند، درخواست به سرویس سفارش ارسال می‌شود.

۲. سرویس سفارش به طور مستقیم با پایگاهداده تعامل ندارد. در عوض، آن درخواست‌های پایگاهداده را به میان‌افزار پایگاهداده ارسال می‌کند.
  ۳. میان‌افزار پایگاهداده نوشتمنها را به پایگاهداده اصلی هدایت می‌کند. داده‌ها به دو نسخه تکراری replica می‌شوند.
  ۴. آلیس جزئیات سفارش را مشاهده می‌کند (خواندن). درخواست از طریق میان‌افزار ارسال می‌شود.
  ۵. آلیس سابقه سفارش‌های اخیر را مشاهده می‌کند (خواندن). درخواست از طریق میان‌افزار ارسال می‌شود.
- میان‌افزار پایگاهداده به عنوان یک پروکسی بین برنامه و پایگاهداده‌ها عمل می‌کند. آن از پروتکل شبکه استاندارد MySQL برای ارتباط استفاده می‌کند.

#### مزایا:

کد برنامه ساده‌تر می‌شود. برنامه نیازی به آگاهی از توپولوژی پایگاهداده و مدیریت دسترسی مستقیم به پایگاهداده ندارد.

سازگاری بهتر. میان‌افزار از پروتکل شبکه MySQL استفاده می‌کند. هر کلاینت سازگار با MySQL می‌تواند به راحتی به میان‌افزار متصل شود. این امر انتقال پایگاهداده را آسان‌تر می‌کند.

#### معایب:

افزایش پیچیدگی سیستم. یک میان‌افزار پایگاهداده یک سیستم پیچیده است. ازانجایی که تمام درخواست‌های پایگاهداده از طریق میان‌افزار می‌گذرند، معمولاً نیاز به یک تنظیم با قابلیت دسترسی بالا برای جلوگیری از یک نقطه شکست تک دارد.

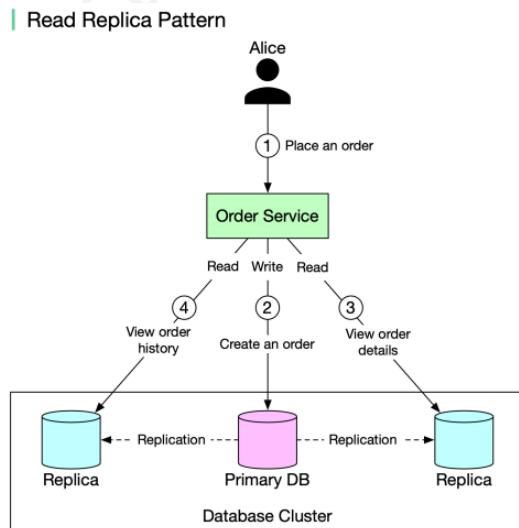
لایه میان‌افزار اضافی به معنای تأخیر شبکه اضافی است؛ بنابراین، این لایه نیازمند عملکرد عالی است.

## الگوهای خواندن داده‌های replica ۲ شده

در این متن، ما درباره یک الگوی ساده اما رایج طراحی پایگاهداده صحبت می‌کنیم: الگوی نسخه replica خواندن.

در این تنظیمات، تمام دستورات تغییردهنده داده؛ مانند درج، حذف یا بهروزرسانی به پایگاهداده اصلی ارسال می‌شوند و خواندن‌ها به نسخه‌های تکراری و تکثیر شده جهت خواندن فرستاده می‌شوند. نمودار زیر این تنظیمات را نشان می‌دهد:

۱. زمانی که آلیس در آمازون سفارشی ثبت می‌کند، درخواست به سرویس سفارش ارسال می‌شود.
۲. سرویس سفارش یک رکورد درباره سفارش در پایگاهداده اصلی ایجاد می‌کند (نوشتن). داده‌ها به دو نسخه تکراری تکثیر می‌شوند.
۳. آلیس جزئیات سفارش را مشاهده می‌کند. داده‌ها از یک نسخه تکثیر شده سرویس داده می‌شوند (خواندن).
۴. آلیس سابقه سفارش‌های اخیر را مشاهده می‌کند. داده‌ها از یک نسخه تکثیر شده سرویس داده می‌شوند (خواندن).



یک مشکل عمده در این تنظیمات وجود دارد: تأخیر تکثیر!

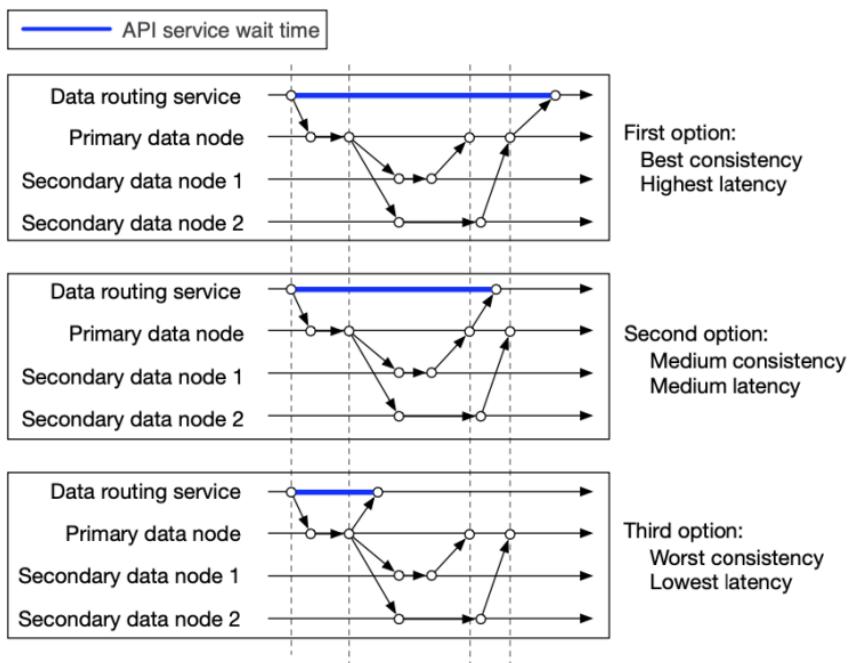
در شرایط خاصی (تأخیر شبکه، اشیاع سرور و غیره)، ممکن است داده‌ها در نسخه‌های تکثیر شده، چند ثانیه یا حتی چند دقیقه عقب باشند. در این حالت، اگر آلیس بلاfaciale پس از ثبت سفارش، وضعیت سفارش را بررسی کند (درخواست پرس‌وجو از نسخه تکثیر شده سرویس داده می‌شود) و حتی ممکن است اصلاً سفارشی را مشاهده نکند. این باعث سردرگمی آلیس می‌شود. در این حالت، ما به «هماهنگی خواندن پس از نوشتن» نیاز داریم.

#### راه حل‌های ممکن برای کاهش این مشکل:

۱. خواندن‌های حساس به تأخیر به پایگاهداده اصلی ارسال می‌شوند.
۲. خواندن‌هایی که بلاfaciale پس از نوشتن‌ها انجام می‌شوند و به پایگاهداده اصلی هدایت می‌شوند.
۳. یک پایگاهداده رابطه‌ای معمولاً راهی برای بررسی اینکه آیا یک replica با اصلی همگام شده است یا خیر، فراهم می‌کند. اگر داده‌ها به روز هستند، از نسخه تکثیر شده پرس‌وجو کنید. در غیر این صورت، درخواست خواندن را رد کنید یا از پایگاهداده اصلی بخوانید.

## تعادل بین تأخیر زمانی و یکپارچگی

در ک تعداد ها نه تنها در مصاحبه های طراحی سیستم بلکه در طراحی سیستم های دنیای واقعی نیز بسیار مهم است. هنگامی که درباره تکثیر داده<sup>۱</sup> صحبت می کنیم، یک تعادل بینیادی بین تأخیر<sup>۲</sup> و یکپارچگی<sup>۳</sup> وجود دارد. این موضوع در نمودار زیر نشان داده شده است.



۱. داده پس از ذخیره شدن در هر سه گره به عنوان ذخیره شده موفق در نظر گرفته می شود. این رویکرد دارای بهترین یکپارچگی اما بیشترین تأخیر است.

data replication<sup>۱</sup>

latency<sup>۲</sup>

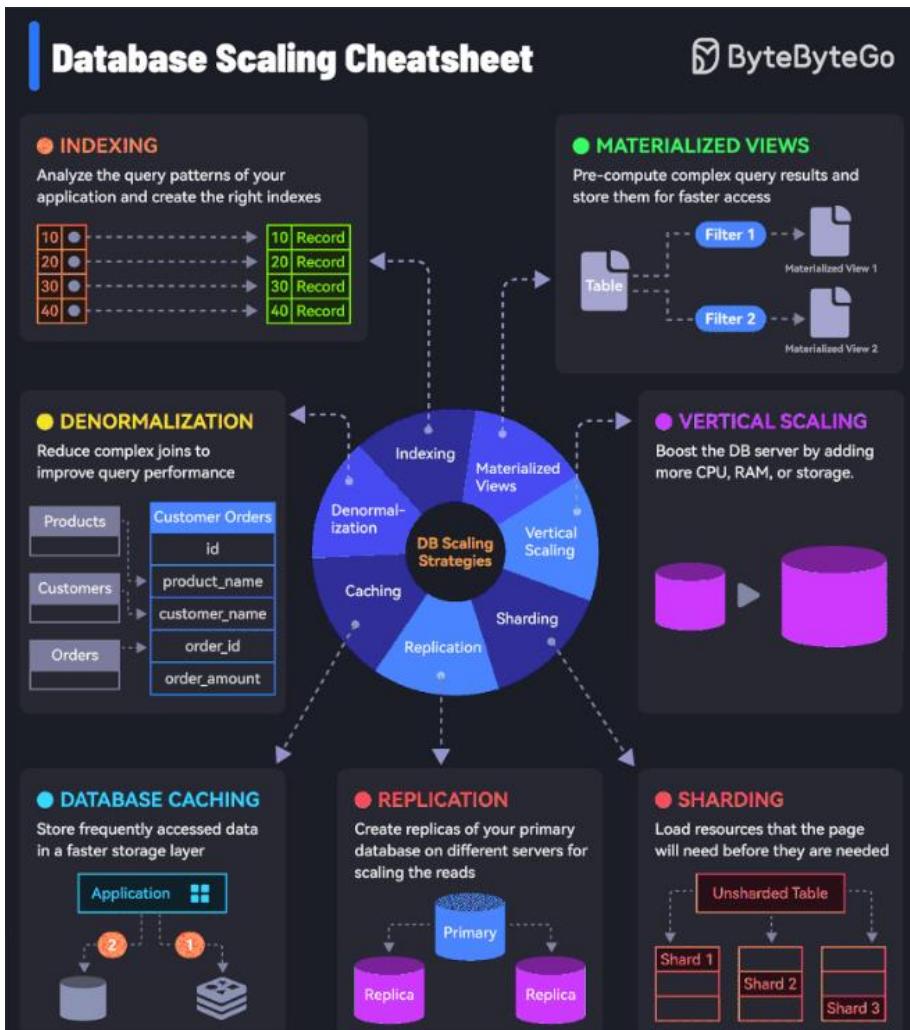
consistency<sup>۳</sup>

۲. داده پس از ذخیره شدن در گرده اصلی و یکی از گردهای فرعی به عنوان ذخیره شده موفق در نظر گرفته می شود. این رویکرد دارای یکپارچگی و تأخیر متوسط است.

۳. داده پس از ذخیره شدن در گرده اصلی به عنوان ذخیره شده موفق در نظر گرفته می شود. این رویکرد دارای بدترین یکپارچگی اما کمترین تأخیر است.

هر دو مورد ۲ و ۳ نوعی از یکپارچگی تدریجی<sup>۱</sup> هستند.

## ۷ استراتژی ضروری برای مقیاس‌پذیری پایگاهداده



- ایندکس‌گذاری (Indexing): الگوهای پرس‌وجوی (query) برنامه خود را بررسی کنید و ایندکس‌های مناسب ایجاد کنید.
- نماهای مادی (Materialized Views): نتایج کوئری‌های پیچیده را پیش محاسبه کنید و آن‌ها را برای دسترسی سریع‌تر ذخیره کنید.

- Denormalization: برای بهبود عملکرد کوئری‌ها، الحاق‌های پیچیده (join) را کاهش دهید.
- مقیاس‌گذاری عمودی (Vertical Scaling): با اضافه کردن CPU، رم (RAM) یا فضای ذخیره‌سازی بیشتر، سرور پایگاه‌داده خود را تقویت کنید.
- کش (Caching): داده‌های پرکاربرد را در یک لایه ذخیره‌سازی سریع‌تر ذخیره کنید تا بار پایگاه‌داده را کاهش دهید.
- تکثیر (Replication): برای مقیاس‌گذاری خواندن (read)، نسخه‌هایی از پایگاه‌داده اصلی خود را در سرورهای مختلف ایجاد کنید.
- شارдинگ (Sharding): جدول‌های پایگاه‌داده خود را به قطعات کوچک‌تر تقسیم کنید و آن‌ها را بین سرورها پخش کنید. این روش برای مقیاس‌پذیری نوشتمن و همچنین خواندن استفاده می‌شود.

## مقیاس‌دهی ۱۰۰ برابری در Postgres

با افزایش ۳ میلیون کاربر ماهانه، تعداد کاربران شرکت Figma از سال ۲۰۱۸ تا کنون ۲۰۰ درصد افزایش یافته است. در نتیجه، پایگاه‌داده Postgres آن شاهد رشد خیره‌کننده ۱۰۰ برابری بوده است. این رشد کاربران چگونه مدیریت شده است؟

### ۱. Replication و Vertical Scaling

فیگما از یک پایگاه‌داده Amazon RDS واحد و بزرگ استفاده می‌کرد. به عنوان اولین قدم، آنها به بزرگ‌ترین نمونه در دسترس (از r5.12xlarge به r5.24xlarge) ارتقا دادند. همچنین برای مقیاس‌بندی ترافیک خواندن، چندین نسخه خواندنی (Read Replica) ایجاد کردند و PgBouncer را به عنوان یک Connection Pooler برای محدود کردن تأثیر تعداد رویه‌رشد اتصالات اضافه کردند.

### ۲. Vertical Partitioning

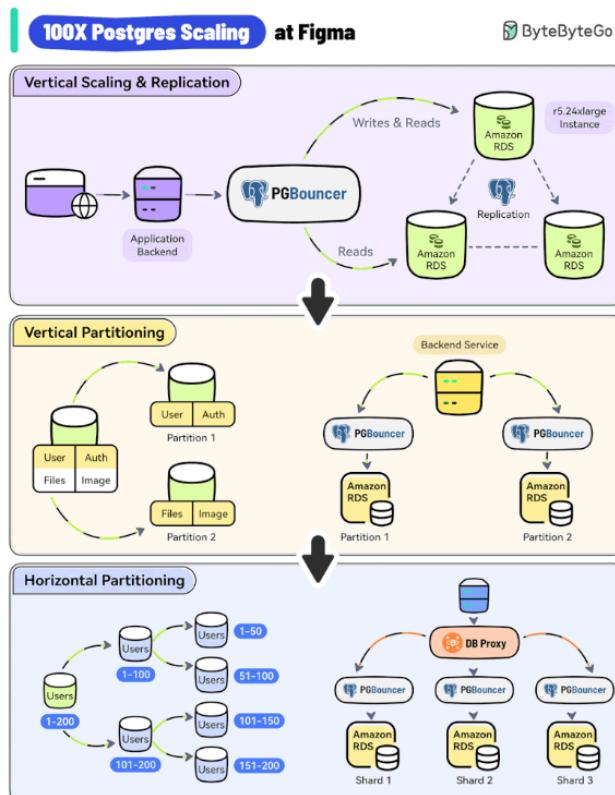
گام بعدی پارتیشن‌بندی عمودی بود.

آنها جدول‌های با ترافیک بالا مانند "Organizations" و "Figma Files" را به پایگاه‌های داده جداگانه خود migrate کردند. برای مدیریت اتصالات به این پایگاه‌های داده معجزاً، از چندین نمونه PgBouncer استفاده شد.

### ۳. Horizontal Partitioning

با گذشت زمان، برخی از جدول‌ها از چندین تراپایت داده و میلیاردها ردیف عبور کردند. فرایند Vacuum در Postgres به یک مشکل تبدیل شد و حداقل IOPS از محدودیت‌های Amazon RDS در آن زمان فراتر رفت. برای حل این مشکل، فیگما پارتیشن‌بندی افقی را با تقسیم‌کردن جداول بزرگ در چندین پایگاه‌داده فیزیکی اجرا کرد.

سرвис جدیدی به نام DBProxy برای مدیریت مسیریابی و اجرای کوئری ساخته شد.



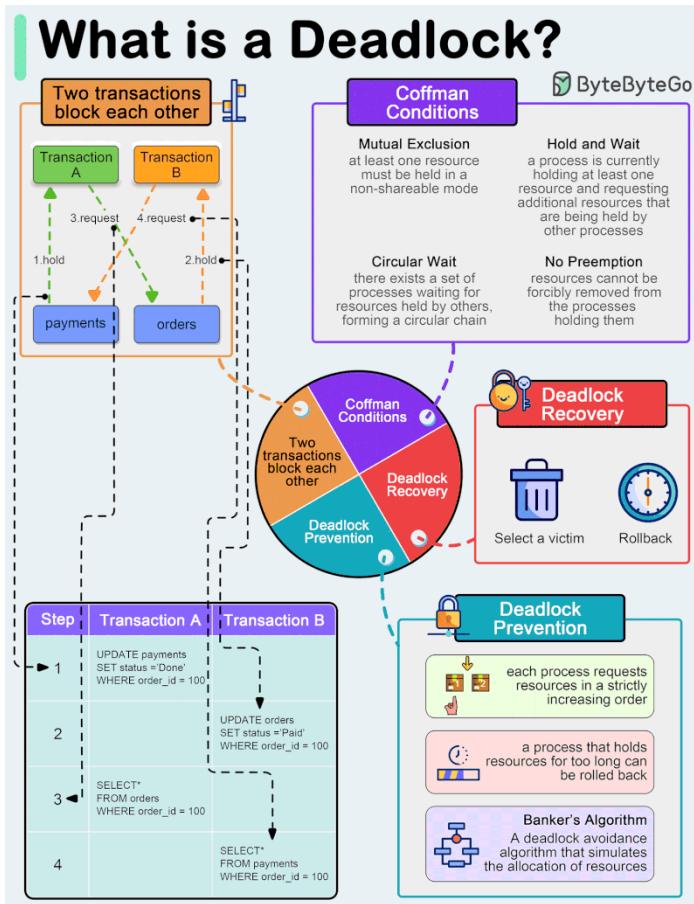
## بنبست (Deadlock) چیست؟

بنبست یا Deadlock زمانی رخ می‌دهد که دو یا چند تراکنش در انتظار آزادشدن قفل‌هایی روی منابعی هستند که برای ادامه‌ی پردازش به آن‌ها تیاز دارند. این منجر به وضعیتی می‌شود که هیچ یک از تراکنش‌ها نمی‌توانند پیشرفت کنند و در نهایت به طور نامحدودی منتظر می‌مانند.

### شرایط Coffman

شرایط Coffman که به نام ادوارد جی. کافمن که اولین بار آنها را در سال ۱۹۷۱ شرح داد، نام‌گذاری شده است، چهار شرط ضروری را توصیف می‌کند که باید هم‌زمان برای رخ دادن بنبست وجود داشته باشند:

- استثنا متقابل (Mutual Exclusion): یک منبع در یک‌زمان تنها می‌تواند در اختیار یک تراکنش قرار گیرد.
- نگهداشتن و انتظار (Hold and Wait): یک تراکنش می‌تواند در حالی که منابع خاصی را در اختیار دارد، منتظر منابع دیگری بماند.
- عدم تصاحب (No Preemption): منابعی که قبلاً توسط یک تراکنش تصاحب شده‌اند، نمی‌توانند به طور اجباری از آن سلب مالکیت شونند.
- انتظار دوره‌ای (Circular Wait): یک زنجیره‌ی انتظار وجود داشته باشد که در آن هر تراکنش در انتظار منبعی است که توسط تراکنش بعدی در زنجیره نگهداری می‌شود.



## پیشگیری از Deadlock

مرتب‌سازی منابع: ترتیبی کلی برای تمام انواع منابع تعیین کنید و مشخص که هر فرایند منابع را به ترتیب کاملاً افزایشی درخواست کند. زمان خروج (Timeout): فرایندی که منابع را برای مدت طولانی در اختیار دارد، می‌تواند لغو و از نو آغاز شود.

الگوریتم بانکدار (Banker's Algorithm): الگوریتمی برای اجتناب از بن‌بست است که تخصیص منابع به فرایندها را شبیه‌سازی کرده و به تصمیم‌گیری در مورد ایمن بودن اعطای

در خواست منابع بر اساس در دسترس بودن آتی منابع کمک می‌کند و بدین ترتیب از حالات نایمن جلوگیری می‌کند.

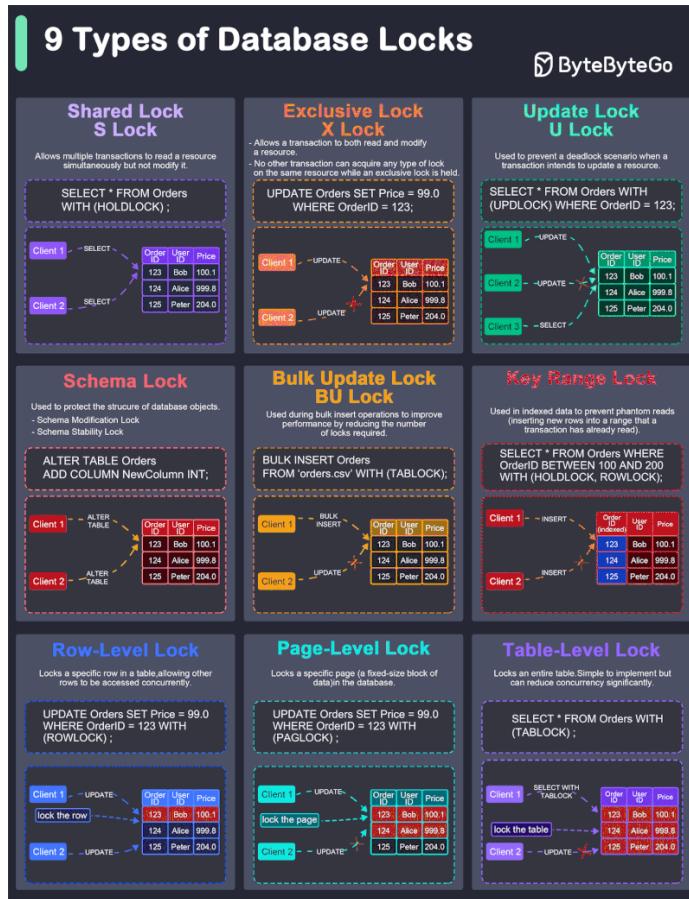
### بازیابی از **Deadlock**

**انتخاب قربانی:** اکثر سیستم‌های مدیریت پایگاهداده (DBMS) و سیستم‌عامل‌های مدرن الگوریتم‌های پیشرفته‌ای را برای تشخیص بن‌بست و انتخاب قربانی پیاده‌سازی می‌کنند و اغلب به کاربران اجازه می‌دهند تا معیارهای انتخاب قربانی را از طریق تنظیمات پیکربندی سفارشی‌سازی کنند. این انتخاب می‌تواند بر اساس استفاده از منابع، اولویت تراکنش، هزینه‌ی لغو و غیره باشد.

**لغو (Rollback):** پایگاهداده ممکن است کل تراکنش را لغو کند یا فقط به اندازه‌ی کافی آن را لغو کند تا بن‌بست را بشکند. تراکنش‌های لغو شده می‌توانند به طور خودکار توسط سیستم مدیریت پایگاهداده مجددأ راه‌اندازی شوند.

### تفاوت‌های بین قفل‌های پایگاهداده

در مدیریت پایگاهداده، قفل‌ها سازوکارهایی هستند که برای اطمینان از یکپارچگی و سازگاری داده‌ها، از دسترسی هم‌زمان به داده‌ها جلوگیری می‌کنند. در اینجا انواع رایج قفل‌های مورداستفاده در پایگاهداده آورده شده است:



۱. قفل اشتراکی (**Shared Lock - S Lock**): به چند تراکنش اجازه می‌دهد تا به طور هم‌زمان یک منبع را بخوانند؛ اما آن را تغییر ندهند. سایر تراکنش‌ها نیز می‌توانند روی همان منبع یک قفل اشتراکی دریافت کنند.

## ۲. قفل انحصاری (**Exclusive Lock - X Lock**)

به یک تراکنش اجازه می‌دهد هم یک منبع را بخواند و هم آن را تغییر دهد. هیچ تراکنش دیگری نمی‌تواند در حالی که یک قفل انحصاری در اختیار است، هیچ نوع قفلی روی همان منبع دریافت کند.

### ۳. قفل بهروزرسانی (Lock - U Lock update)

برای جلوگیری از سناریوی بنبست (Deadlock) زمانی که یک تراکنش قصد بهروزرسانی یک منبع را دارد، استفاده می‌شود.

### ۴. قفل اسکیما (Schema Lock)

برای محافظت از ساختار اشیاء پایگاهداده استفاده می‌شود.

### ۵. قفل بهروزرسانی انبوه (Bulk Update Lock - BU Lock)

برای بهبود عملکرد با کاهش تعداد قفل‌های موردنیاز، در طول عملیات درج انبوه استفاده می‌شود.

### ۶. قفل محدوده کلیدی (Key-Range Lock)

برای جلوگیری از خواندن‌های phantom (به معنی درج ردیف‌های جدید در محدوده‌ای که یک تراکنش قبلًا خوانده است) در داده‌های ایندکس شده استفاده می‌شود.

### ۷. قفل سطح سطر (Row-Level Lock)

یک سطر خاص را در یک جدول قفل می‌کند و به سایر سطرهای اجازه می‌دهد به طور همزمان دسترسی داشته باشند.

### ۸. قفل سطح صفحه (Page-Level Lock)

یک صفحه خاص (یک بلوک داده با اندازه ثابت) را در پایگاهداده قفل می‌کند.

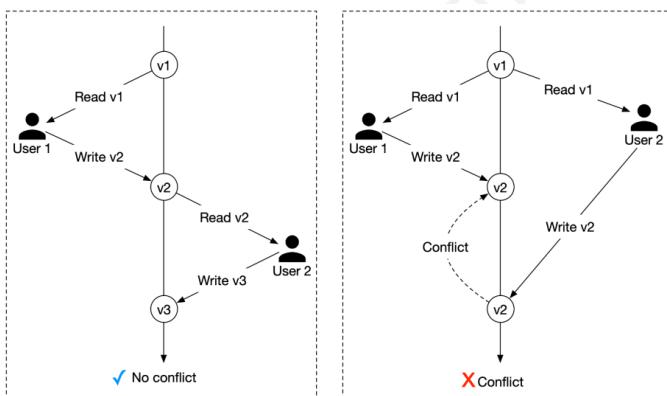
### ۹. قفل سطح جدول (Table-Level Lock)

یک کل جدول را قفل می‌کند. پیاده‌سازی آن ساده است، اما می‌تواند به طور قابل توجهی همزمانی (concurrency) را کاهش دهد.

## قفل‌گذاری خوش‌بینانه

قفل‌گذاری خوش‌بینانه<sup>۱</sup> که به آن کنترل هم‌زمان خوش‌بینانه نیز گفته می‌شود، به کاربران متعدد به صورت هم‌زمان اجازه می‌دهد تا تلاش کنند منابع مشابه را به‌روزرسانی کنند.

دو روش رایج برای پیاده‌سازی قفل‌گذاری خوش‌بینانه وجود دارد: شماره نسخه<sup>۲</sup> و timestamp. شماره نسخه به‌طورکلی گزینه بهتری محسوب می‌شود، زیرا ساعت سرور ممکن است با گذشت زمان دقیق نباشد. ما چگونگی کار قفل‌گذاری خوش‌بینانه با استفاده از شماره نسخه را توضیح می‌دهیم. دیاگرام زیر یک مورد موفق و یک مورد شکست را نشان می‌دهد.



۱. یک ستون جدید به نام «نسخه» به جدول پایگاهداده اضافه می‌شود.
۲. قبل از اینکه کاربر یک ردیف پایگاهداده را اصلاح کند، برنامه شماره نسخه آن ردیف را می‌خواند.
۳. هنگامی که کاربر ردیف را به‌روزرسانی می‌کند، برنامه شماره نسخه را یک واحد افزایش داده و آن را در پایگاهداده بازنویسی می‌کند.

Optimistic locking<sup>۱</sup>  
version number<sup>۲</sup>

۴. یک بررسی اعتبارسنجی از پایگاهداده در نظر گرفته می‌شود؛ شماره نسخه بعدی باید یک واحد از شماره نسخه فعلی بیشتر باشد. اگر این بررسی اعتبار شکست بخورد، تراکنش لغو شده و کاربر باید از مرحله ۲ دوباره شروع کند.

قفل گذاری خوشبینانه معمولاً سریع‌تر از قفل گذاری بدینانه است؛ زیرا ما پایگاهداده را قفل نمی‌کنیم. با این حال، عملکرد قفل گذاری خوشبینانه هنگامی که درخواست‌های هم‌زمان زیاد است، به شدت کاهش می‌یابد.

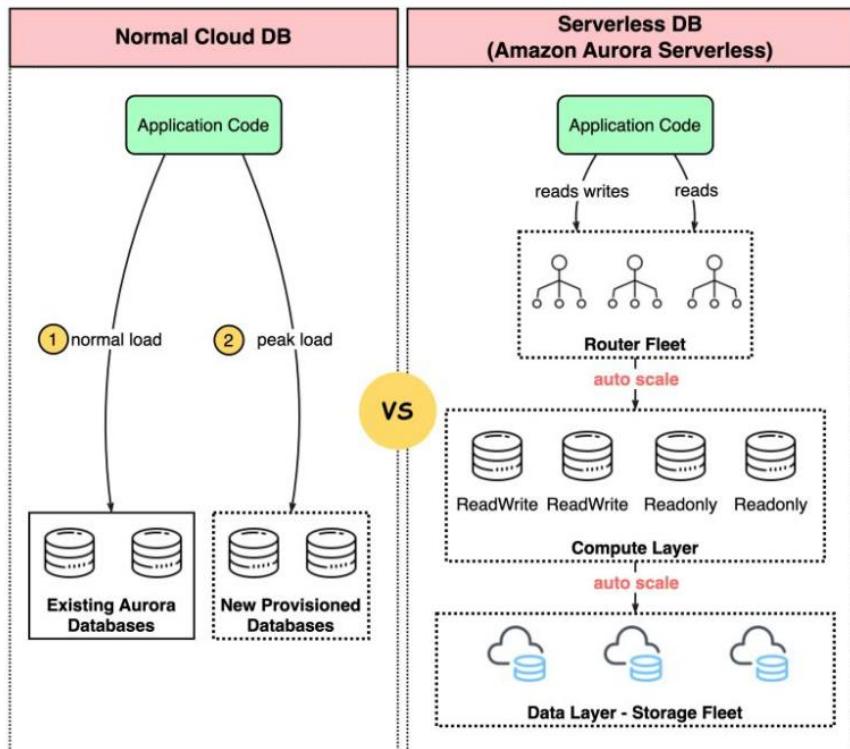
برای درک علت، حالتی را در نظر بگیرید که در آن بسیاری از مشتریان در یک‌زمان سعی در رزرو یک اتاق هتل دارند. از آنجایی که محدودیتی برای تعداد مشتریانی که می‌توانند تعداد اتاق‌های موجود را بخوانند وجود ندارد، همه آنها تعداد اتاق‌های موجود و شماره نسخه فعلی را بازخوانی می‌کنند. هنگامی که مشتریان مختلف رزرو می‌کنند و نتایج را به پایگاهداده بازمی‌نویسنند، تنها یکی از آنها موفق خواهد شد و بقیه مشتریان پیام شکست بررسی نسخه را دریافت می‌کنند. این مشتریان باید دوباره تلاش کنند. در دور بعدی تلاش‌های مجدد، فقط یک مشتری موفق است و بقیه باید دوباره تلاش کنند. اگرچه نتیجه نهایی درست است، اما تلاش‌های مکرر باعث تجربه ناخوشایند کاربر می‌شود.

## تفاوت پایگاه‌داده‌های serverless با پایگاه‌داده‌های ابری سنتی.

در نمودار زیر نشان داده شده است دارای یک پیکربندی است که به صورت خودکار مقیاس پذیر<sup>۱</sup> می‌شود و برای سرویس Amazon Aurora به طور مناسبی در دسترس است.

### What is Serverless DB?

 blog.bytebytogo.com



در Aurora Serverless توانایی مقیاس‌بندی ظرفیت را به طور خودکار به جهت بالا یا پایین بر اساس نیازهای business را دارد. به عنوان مثال، یک وب‌سایت تجارت الکترونیک که

auto-scaling<sup>۱</sup>

برای یک تبلیغات بزرگ آماده می‌شود، می‌تواند بار را در عرض چند میلی ثانیه به چندین پایگاهداده مقیاس‌بندی کند. در مقایسه با پایگاهداده‌های ابری معمولی که نیاز به تهیه و مدیریت نمونه‌های پایگاهداده دارند، Aurora Serverless می‌تواند به طور خودکار راه اندازی و خاموش شود.

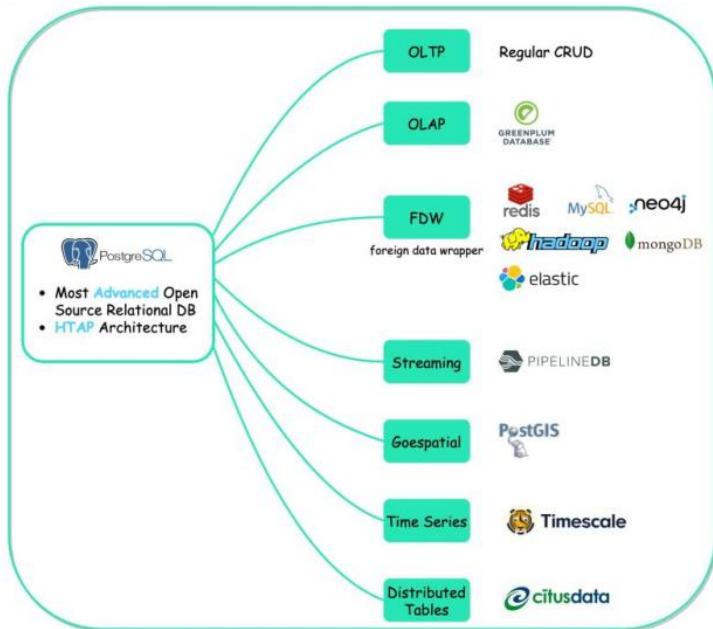
با جدا کردن<sup>۱</sup> لایه محاسباتی از لایه ذخیره سازی داده، Aurora Serverless قادر است هزینه‌ها را به صورت دقیق‌تری محاسبه کند. علاوه بر این، Aurora Serverless می‌تواند ترکیبی از نمونه‌های تامین شده و serverless باشد و امکان تبدیل پایگاهداده‌های تامین شده موجود را به بخشی از مجموعه serverless را فراهم کند.

---

<sup>۱</sup> decoupling

## چرا PostgreSQL محبوب ترین است؟

نمودار زیر موارد استفاده بسیاری را توسط PostgreSQL نشان می‌دهد این پایگاهداده تقریباً شامل تمام موارد استفاده مورد نیاز توسعه‌دهندگان است.



### OLTP (Online Transaction Processing)

ما می‌توانیم از PostgreSQL برای عملیات CRUD (ایجاد، خواندن، بهروزرسانی، حذف) استفاده کنیم.

### OLAP (Online Analytical Processing)

ما می‌توانیم از PostgreSQL برای پردازش تحلیلی استفاده کنیم. PostgreSQL بر اساس معماری HTAP<sup>1</sup> (پردازش ترکیبی تراکنش/تحلیلی) است، بنابراین می‌تواند هم OLTP و هم OLAP را به خوبی مدیریت کند.

Hybrid transactional/analytical processing <sup>1</sup>

**FDW (Foreign Data Wrapper)**

FDW یک افزونه موجود در PostgreSQL است که به ما امکان دسترسی به یک جدول یا طرح در یک پایگاهداده از پایگاهداده دیگر را می‌دهد.

**Streaming PipelineDB**

یک افزونه PostgreSQL برای جمع‌بندی سری زمانی با کارایی بالا است که برای پشتیبانی از برنامه‌های گزارش‌دهی و تحلیل در زمان بلادرنگ طراحی شده است.

**Geospatial PostGIS**

یک گسترش پایگاه داده مکانی برای پایگاه داده شیء-رابطه‌ای PostgreSQL است. این پشتیبانی از اشیاء جغرافیایی را اضافه می‌کند و اجازه می‌دهد تا پرس‌وچوهای مکانی در SQL اجرا شوند.

**سری زمانی**

Timescale PostgreSQL را برای سری زمانی و تحلیل گسترش می‌دهد. به عنوان مثال، توسعه‌دهندگان می‌توانند جریان‌های حجمی داده‌های مالی را با داده‌های تجاری دیگر ترکیب کنند تا برنامه‌های جدید بسازند و نگرش‌های منحصر به فردی پیدا کنند.

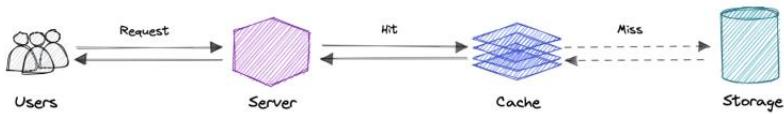
**جدول‌های توزیع شده**

CitusData با توزیع داده‌ها و کوئری‌ها، Postgres را مقیاس‌پذیر می‌کند.

## بررسی کش (Cache)

یکی از گفته‌های معروف در علوم کامپیوتر این است که «تنهای دو مشکل سخت در علوم کامپیوتر وجود دارد: نامعتبر کردن کش<sup>۱</sup> و نام‌گذاری چیزها».

هدف اصلی کش، افزایش عملکرد بازیابی داده با کاهش نیاز به دسترسی به لایه ذخیره‌سازی کندر در سایر لایه‌های پایین‌تر<sup>۲</sup> است. یک کش با به خطر انداختن ظرفیت به نفع سرعت، معمولاً زیرمجموعه‌ای از داده‌ها را به طور موقت ذخیره می‌کند و این برخلاف منطق پایگاه‌داده‌ایی است که داده‌های آن‌ها معمولاً کامل و بادوام هستند.



کش‌ها از اصل مرجع مکان‌یابی<sup>۳</sup> استفاده می‌کنند که می‌گوید «داده‌هایی که به تازگی درخواست شده‌اند به احتمال زیاد دوباره درخواست خواهند شد».

## کش و memory پردازنده

مانند حافظه memory/ رایانه، کش یک حافظه جمع‌وجور و با عملکرد سریع است که داده‌ها را در سلسله‌مراتبی از سطوح ذخیره می‌کند، از سطح یک شروع می‌شود و به ترتیب از آنجا پیش می‌رود. آن‌ها به صورت L1, L2, L3 و غیره برچسب‌گذاری می‌شوند. همچنین در صورت درخواست، داده‌های جدیدتر در کش نیز نوشته می‌شود، مانند زمانی که به روزرسانی انجام شده و محتوای جدید نیاز به ذخیره‌شدن در کش دارد و جایگزین محتوای قدیمی‌تر می‌شود که ذخیره شده بود.

cache invalidation<sup>۱</sup>

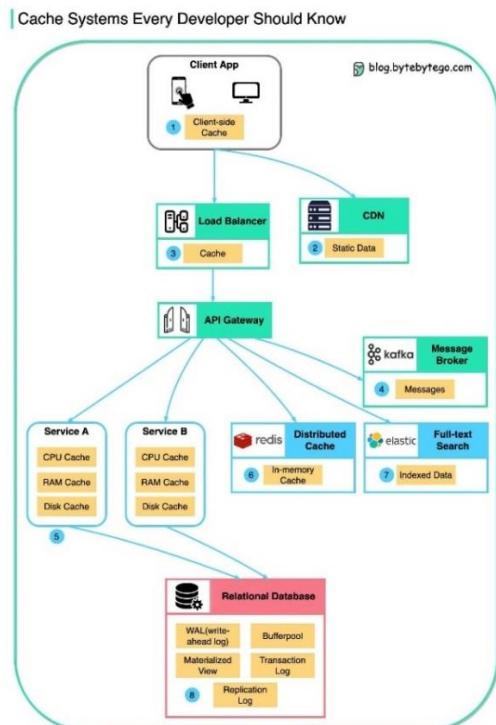
underlying slower storage<sup>۲</sup>

locality of reference<sup>۳</sup>

مهم نیست که کش خوانده شود یا نوشته شود، این کار به صورت بلوک به بلوک انجام می‌شود. هر بلوک همچنین دارای تگی است که شامل مکان ذخیره داده در کش است. هنگامی که داده‌ای از کش درخواست می‌شود، جستجویی از طریق تگ‌ها برای یافتن محتوای خاص موردنیاز در سطح یک (L1) حافظه انجام می‌شود. اگر داده صحیح پیدا نشد، جستجوهای بیشتری در L2 انجام می‌شود. اگر داده در آنجا پیدا نشد، جستجوها در L3 و سپس L4 و بیشتری در L2 انجام می‌شود. اگر داده در آنجا پیدا نشد، جستجوها در L3 و سپس L4 و بیشتری در L2 انجام می‌شود. اگر داده در آنجا پیدا نشد، جستجوها در L3 و سپس L4 و بیشتری در L2 انجام می‌شود. اگر داده در آنجا پیدا نشد، جستجوها در L3 و سپس L4 و بیشتری در L2 انجام می‌شود. اگر داده در آنجا پیدا نشد، جستجوها در L3 و سپس L4 و بیشتری در L2 انجام می‌شود.

## کاربردهای کش

این نمودار نشان می‌دهد که در یک معماری معمولی کجا داده‌ها را کش می‌کنیم.



۱. اپلیکیشن‌های کلاینت<sup>۱</sup>: پاسخ‌های HTTP توسط مرورگر قابل کش شدن هستند. ما برای اولین بار داده‌ها را از طریق HTTP درخواست می‌کنیم و با یک سیاست انقضا در هدر HTTP برگردانده می‌شود؛ ما دوباره درخواست داده می‌کنیم و برنامه کلاینت ابتدا سعی می‌کند داده‌ها را از کش مرورگر بازیابی کند.
۲. شبکه توزیع محتوا (CDN): در واقع CDN منابع استاتیک/ثابت وب را کش می‌کند. کاربران می‌توانند داده‌ها را از یک گره CDN نزدیک به محل اتصال خود به اینترنت بازیابی کنند.
۳. سیستم نام دامنه (DNS<sup>۲</sup>): یکی دیگر از کاربردهای کش است.
۴. متعادل‌کننده بار<sup>۳</sup>: متعادل‌کننده بار نیز می‌تواند منابع را کش کند.
۵. زیرساخت پیام‌رسانی<sup>۴</sup>: کارگزاران پیام ابتدا پیام‌ها را روی دیسک ذخیره می‌کنند و سپس مصرف‌کنندگان آنها را با سرعت خود بازیابی می‌کنند. بسته به سیاست نگهداری، داده‌ها برای مدت زمانی در خوشه‌های کافکا کش می‌شوند.
۶. سرویس‌ها: در یک سرویس لایه‌های متعددی از کش وجود دارد. اگر داده‌ها در کش CPU نشده باشند، سرویس سعی می‌کند داده‌ها را از حافظه بازیابی کند. گاهی اوقات سرویس یک کش سطح دوم برای ذخیره داده‌ها روی دیسک دارد.
۷. کش توزیع شده<sup>۵</sup>: کش توزیع شده مانند Redis جفت‌های key-value را برای چندین سرویس در حافظه نگه می‌دارد. این عملکرد خواندن/نوشتن بسیار بهتری نسبت به پایگاه‌داده ارائه می‌دهد. در واقع کش توزیع شده سیستمی است که RAM

---

Client apps<sup>۱</sup>

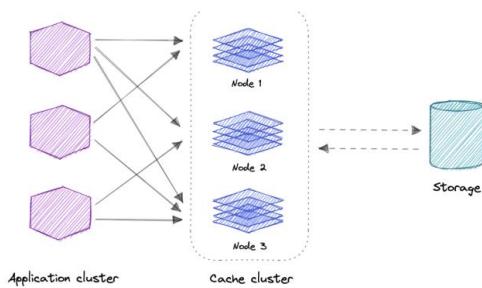
Domain Name System<sup>۲</sup>

Load Balancer<sup>۳</sup>

Messaging infra<sup>۴</sup>

Distributed Cache<sup>۵</sup>

چندین کامپیوتر شبکه‌ای را در یک ذخیره‌گاه داده<sup>۱</sup> درون حافظه<sup>۲</sup> واحد جمع‌آوری می‌کند و از آن به عنوان کش داده برای فراهم‌کردن دسترسی سریع به داده استفاده می‌کند. در حالی که اکثر کش‌ها به طور سنتی در یک سرور فیزیکی یا یک جزء سخت‌افزاری قرار دارند، یک کش توزیع شده می‌تواند با اتصال چندین کامپیوتر به هم فراتر از محدودیت‌های حافظه یک کامپیوتر واحد رشد کند.



۸. جستجوی متن کامل<sup>۳</sup>: گاهی اوقات نیاز به استفاده از جستجوهای متن کامل مانند

ابزاری شبیه Elastic Search برای جستجوی اسناد یا جستجوی لاغ‌ها داریم. یک

کبی از داده‌ها نیز در موتور جستجو فهرست‌بندی<sup>۴</sup> می‌شود.

۹. پایگاهداده: حتی در پایگاهداده، سطوح مختلفی از کش داریم:

- لاغ پیش‌نویس<sup>۵</sup>: داده‌ها قبل از ساخت یک ایندکس B-Tree، ابتدا در

WAL نوشته می‌شوند.

۱ data store

۲ in-memory

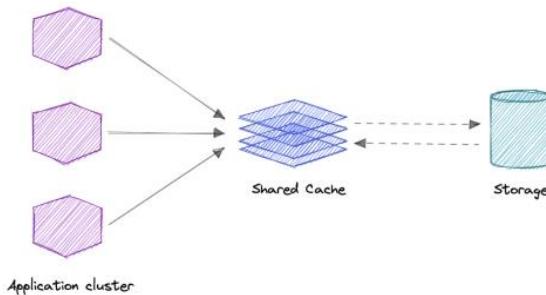
۳ Full-text Search

۴ indexed

۵ Write-ahead Log

- **Bufferpool:** یک ناحیه حافظه اختصاص داده شده برای کش کردن نتایج کوئری کاربرد دارد.
- **Materialized View:** نتایج پرس و جو را پیش محاسبه می کند و آنها را در جداول پایگاهداده برای عملکرد بهتر پرس و جو / کوئری ذخیره می کند.
- **Transaction log:** تمام تراکنش ها و بهروزرسانی های پایگاهداده را ثبت می کند.
- **Replication Log:** برای ثبت وضعیت تکثیر در یک خوشه پایگاهداده استفاده می شود.

۱۰. کش سراسری<sup>۱</sup>: همان طور که از نام آن پیداست، ما یک کش مشترک واحد خواهیم داشت که تمام گره های برنامه از آن استفاده خواهند کرد. هنگامی که داده هی در خواستی در کش سراسری یافت نشد، مسئولیت کش این است که قطعه گم شده داده را از مخزن داده دیگری پیدا کند.



در این سناریو، کل داده بر روی یک سرور واحد یا حافظه مشترک ذخیره نمی شود، بلکه مفهوم «کش سراسری» این است که تمام گره های برنامه یک فضای نام مشترک<sup>۲</sup> را می بینند و می توانند به داده در هر کجا که در زیرساخت کش توزیع شده قرار دارد، دسترسی داشته باشند. کش توزیع شده زیربنایی مسئولیت

¹ Global Cache

² shared namespace

همگام‌سازی داده‌ها بین گره‌های مختلف و اطمینان از بهروز بودن آن‌ها است. کش سراسری مزایای قابل توجهی مانند کاهش بار روی مخزن داده‌ی پایینی و بهبود عملکرد کلی برنامه را ارائه می‌دهد.

### چه زمانی از کش نباید استفاده کرد؟

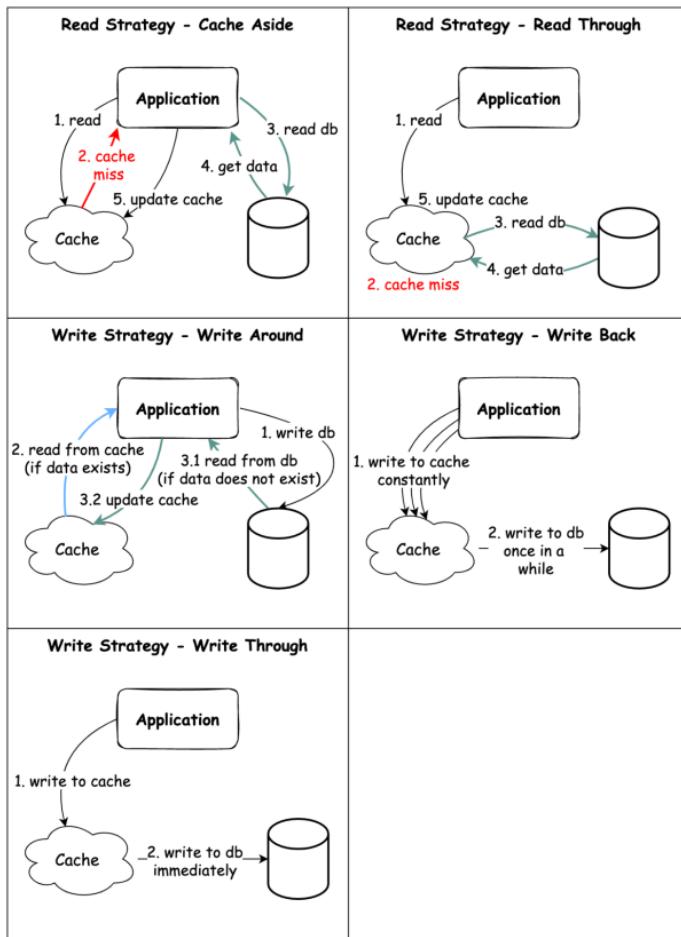
- باید به برخی سناریوها نگاه کنیم که در آن‌ها نباید از کش استفاده کنیم:
- دسترسی آهسته به کش: زمانی که دسترسی به کش بهاندازه دسترسی به مخزن داده اصلی زمان می‌برد، استفاده از کش مفید نیست.
- داده‌ای با تکرار کم (تصادفی بودن بالا): زمانی که درخواست‌ها تکرار کمی داشته باشند (تصادفی بودن بالا)، کشینگ به خوبی کار نمی‌کند، زیرا عملکرد کش از الگوهای دسترسی مکرر به حافظه ناشی می‌شود.
- داده‌های با تغییر مکرر: هنگامی که داده‌ها به طور مکرر تغییر می‌کنند، استفاده از کش مفید نیست، زیرا نسخه کش شده با نسخه اصلی همگام‌سازی نمی‌شود و هر بار باید به مخزن داده اصلی دسترسی پیدا کرد.

مهم است که توجه داشته باشید که کش نباید به عنوان ذخیره دائمی داده استفاده شود. کش‌ها تقریباً همیشه در حافظه فرار<sup>۱</sup> پیاده‌سازی می‌شوند؛ زیرا سریع‌تر هستند و بنابراین باید به عنوان داده‌های موقت در نظر گرفته شوند.

## بررسی برخی مفاهیم در کش:

- **Cache Invalidation:** نامعتبر کردن کش فرایندی است که در آن سیستم کامپیوتری ورودی‌های کش را نامعتبر اعلام کرده و آنها را حذف یا جایگزین می‌کند. اگر داده‌ها تغییر کنند، باید در کش باطل شوند، در غیر این صورت، این امر می‌تواند باعث رفتار ناسازگار برنامه شود.
- **Cache Hit:** وضعیتی را توصیف می‌کند که محتوا با موفقیت از کش پردازش می‌شود. تگ‌ها به سرعت در حافظه جستجو می‌شوند و هنگامی که داده‌ها یافت و خوانده می‌شوند، یک Cache Hit در نظر گرفته می‌شود.
- **Cache Hit:** همچنین می‌تواند سرد، گرم یا داغ توصیف شود. در هر یک از این موارد، سرعت خواندن داده توصیف می‌شود.
- **Hot Cache:** این حالتی است که داده‌ها با سریع‌ترین سرعت ممکن از حافظه خوانده شده‌اند. این زمانی اتفاق می‌افتد که داده‌ها از L1 بازیابی شوند.
- **Cold Cache:** با این حال، کندترین سرعت ممکن برای خواندن داده، همچنان یک "Cache Hit" موفق محسوب می‌شود. داده‌ها فقط در سطوح پایین‌تر سلسله‌مراتب حافظه مانند L3 یا پایین‌تر یافت می‌شوند.
- **Warm Cache:** برای توصیف داده‌ای که در L2 یا L3 یافت می‌شوند، از اصطلاح "Warm Cache" استفاده می‌شود. این به سرعت کش داغ نیست، اما همچنان از کش سرد سریع‌تر است. به طور کلی، "Warm" نامیدن کش برای بیان این است که کندتر از کش داغ است و به کش سرد نزدیک‌تر است.
- **Cache Miss:** به زمانی اشاره دارد که داده‌ها در حافظه جستجو می‌شود و داده‌ای پیدا نمی‌شود. هنگامی که این اتفاق می‌افتد، محتوا منتقل شده و در کش نوشته می‌شود.

## بهترین استراتژی‌های کش کردن چیست؟



نمودار بالا نشان می‌دهد که این ۵ استراتژی چگونه کار می‌کنند. برخی از استراتژی‌های کش کردن می‌توانند با هم استفاده شوند.

خواندن داده از کش:

- Cache aside
- Read Through

نوشتن داده در کش:

۱. کش مستقیم (Write-through cache):

داده‌ها به طور هم‌زمان در کش و پایگاه داده مربوطه نوشته می‌شوند.

مزایا: بازیابی سریع، انسجام کامل داده بین کش و ذخیره‌سازی.

معایب: تأخیر بالاتر برای عملیات نوشتند.

۲. کش کناری (Write-around cache):

در این حالت نوشتند مستقیماً به پایگاه داده یا ذخیره‌سازی دائمی می‌روند و کش را دور می‌زنند.

مزایا: این کار ممکن است تأخیر را کاهش دهد.

معایب: این باعث افزایش خطاهای cache miss می‌شود؛ زیرا سیستم کش در صورت بروز خطای کش باید اطلاعات را از پایگاه داده بخواند. در نتیجه، این کار می‌تواند در مورد برنامه‌هایی که به سرعت اطلاعات را می‌نویسند و دوباره می‌خوانند، منجر به تأخیر بیشتر در خواندن شود. خواندن از حافظه پشتیبان کنترل اتفاق می‌افتد و تأخیر بالاتری را تجربه می‌کند.

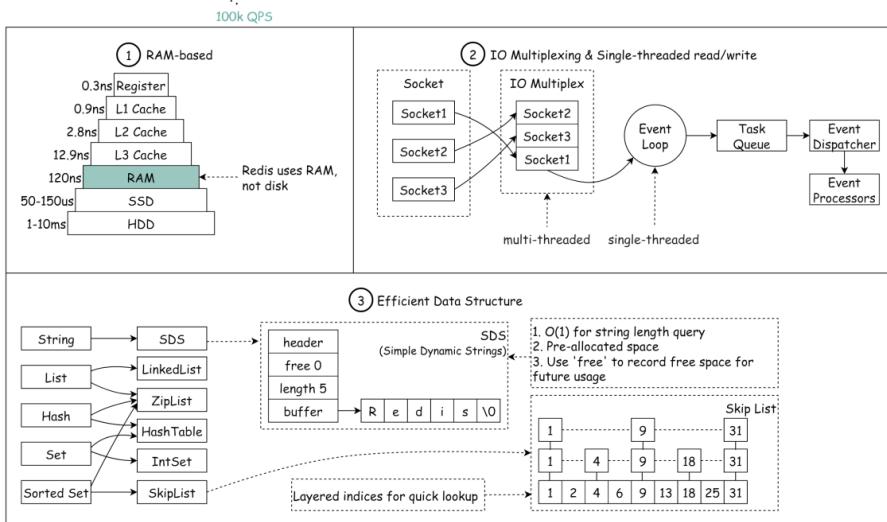
۳. کش با نوشتند پشتی (Write-back cache):

در این حالت نوشتند فقط در لایه کش انجام می‌شود و به محض تکمیل نوشتند در کش، تأیید می‌شود. سپس کش به طور ناهم‌زمان این نوشتند را با پایگاه داده همگام‌سازی می‌کند.

مزایا: این منجر به کاهش تأخیر و توان عملیاتی بالا برای برنامه‌هایی با حجم نوشتند بالا می‌شود.

معایب: در صورت خرابی لایه کش، خطر از دست رفتن داده‌ها وجود دارد. ما می‌توانیم این کار را بیش از یکبار که نوشتمن را در کش تأیید می‌کند، بهبود بخسیریم.

## چرا این قدر سریع است؟ Redis



۱. Redis یک پایگاه داده مبتنی بر RAM است. دسترسی RAM حداقل ۱۰۰۰ برابر سریع‌تر از دسترسی تصادفی به دیسک است.

۲. Redis از چندین I/O و حلقه اجرایی تک نخی<sup>۱</sup> برای افزایش عملکرد اجرایی بهره می‌برد.

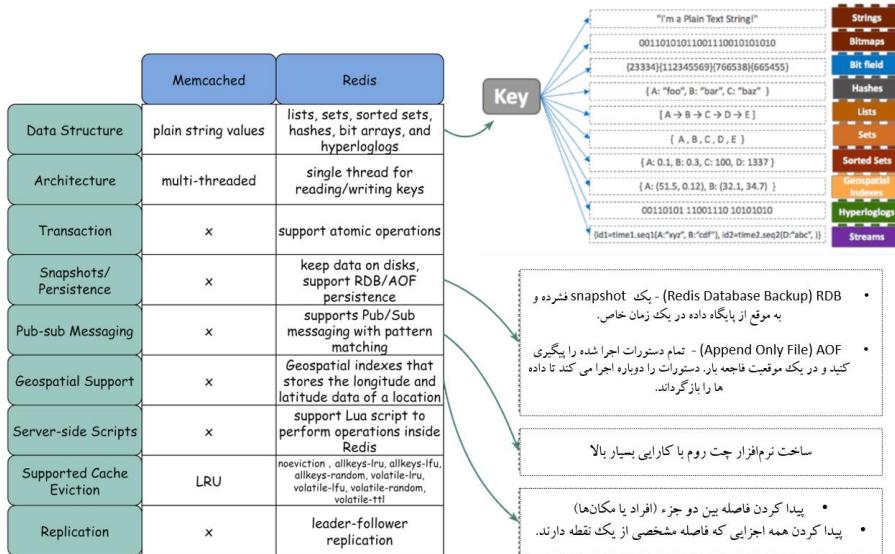
۳. Redis از چندین ساختار داده سطح پایین کارآمد بهره می‌برد.  
سؤال: یک ذخیره‌گاه حافظه درونی<sup>۲</sup> محبوب دیگر Memcached است. آیا تفاوت‌های بین Redis و Memcached را می‌دانید؟

<sup>۱</sup> single-threaded

<sup>۲</sup> in-memory store

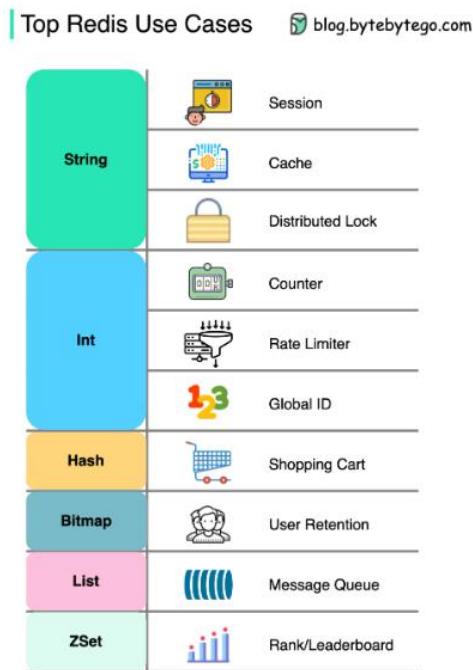
## Redis در مقابل Memcached

Redis در مقابل Memcached نمودار زیر تفاوت‌های کلیدی را نشان می‌دهد. مزایای ساختارهای داده‌ای، Redis را یک انتخاب خوب برای موارد زیر می‌کند:



- ضبط تعداد کلیک‌ها و نظرات برای هر پست (.hash).
- مرتب‌سازی لیست کاربران نظر دهنده و حذف تکراری کاربران (.zset).
- کش کردن تاریخچه رفتار کاربران و فیلتر رفتارهای مخرب (.zset, hash).
- ذخیره اطلاعات boolean داده‌های بسیار بزرگ در فضای کوچک. برای مثال، وضعیت ورود، وضعیت عضویت (.bitmap).

## چه کاربردهایی دارد؟ Redis



فراتر از صرفاً کش کردن است. Redis همانطور که در نمودار نشان داده شده است، Redis را می‌توان در سناریوهای مختلفی استفاده کرد.

- می‌توانیم از Redis برای بهاشتراك‌گذاری داده‌های Session کاربر بین سرویس‌های مختلف استفاده کنیم.
- می‌توانیم از Redis برای کش کردن اشیاء یا صفحات، بهخصوص برای داده‌های پرمخاطب، استفاده کنیم.
- می‌توانیم از Cache (Redis) برای گرفتن قفل بین سرویس‌های قفل توزیع شده؛ می‌توانیم از یک رشته Redis برای گرفتن قفل بین سرویس‌های توزیع شده استفاده کنیم.

- شمارنده (Counter): می‌توانیم تعداد لایک‌ها یا تعداد خواندن مقالات را بشماریم.
- محدودکننده نرخ<sup>۱</sup>: می‌توانیم برای IP‌های کاربری خاص، محدودکننده نرخ اعمال کنیم.
- تولیدکننده شناسه جهانی<sup>۲</sup>: می‌توانیم از Redis Int برای شناسه جهانی استفاده کنیم.
- سبد خرید<sup>۳</sup>: می‌توانیم از Redis Hash برای نمایش جفت‌های key-value در سبد خرید استفاده کنیم.
- محاسبه نگهداری کاربر<sup>۴</sup>: می‌توانیم از Bitmap برای نمایش ورود روزانه کاربر و محاسبه نگهداری کاربر استفاده کنیم.
- صف پیام<sup>۵</sup>: می‌توانیم از List برای صف پیام استفاده کنیم.
- رتبه‌بندی (Ranking): می‌توانیم از ZSet برای مرتب کردن مقالات استفاده کنیم.

---

Rate limiter<sup>۱</sup>

Global ID generator<sup>۲</sup>

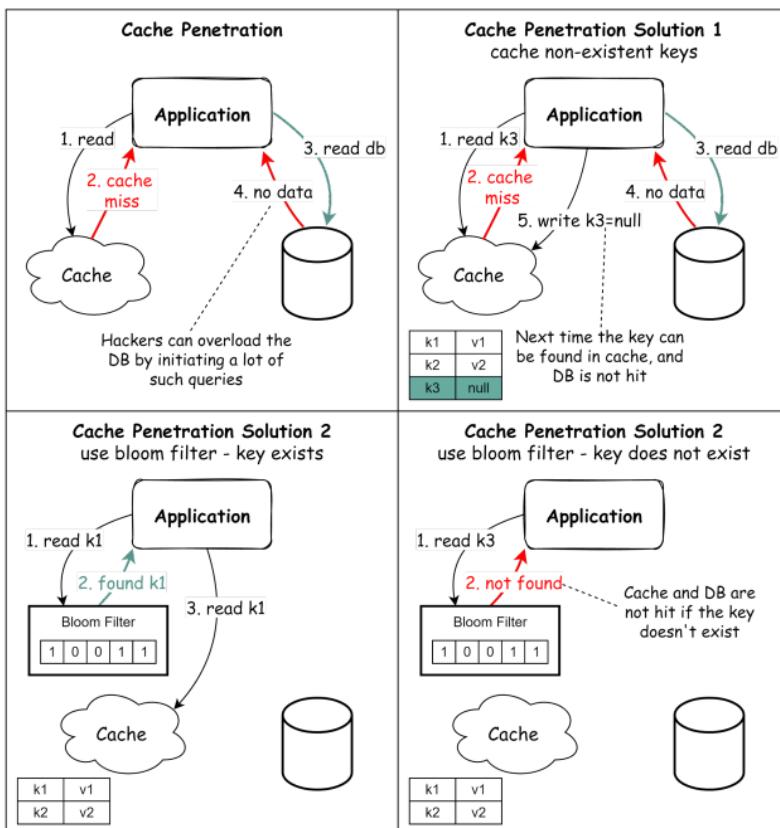
Shopping cart<sup>۳</sup>

Calculate user retention<sup>۴</sup>

Message queue<sup>۵</sup>

## حمله گم شدن داده در Cache

کش کردن عالی است اما مانند بسیاری از چیزها در زندگی، هزینه‌ای هم دارد. یکی از مشکلات، حمله گم شدن داده در کش است. این اصطلاح به سنتاریویی اشاره دارد که داده‌ای برای بازیابی نه در پایگاهداده وجود دارد و نه در کش ذخیره شده است؛ بنابراین هر درخواستی در نهایت به پایگاهداده می‌رسد و هدف استفاده از کش را نقض می‌کند. اگر یک کاربر مخرب تعداد زیادی درخواست با چنین کلیدهایی ارسال کند، می‌تواند به راحتی پایگاهداده را اشبع کند. نمودار زیر این فرایند را نشان می‌دهد.



دو رویکرد متدائل برای حل این مشکل وجود دارد:

- کلیدهای کش با مقدار null. یک 'TTL' کوتاه برای کلیدهای با مقدار null تنظیم کنید.
- استفاده از فیلتر بلوم. فیلتر بلوم یک ساختار داده است که به سرعت می‌تواند به ما بگوید آیا یک عنصر در یک مجموعه وجود دارد یا خیر. اگر کلید وجود داشته باشد، درخواست ابتدا به کش می‌رود و در صورت نیاز از پایگاهداده پرس‌وجو / کوئری می‌کند. اگر کلید در مجموعه‌داده وجود نداشته باشد، به این معنی است که کلید نه در کش و نه در لایه پایگاهداده وجود ندارد. در این حالت، پرس‌وجو به کش یا لایه پایگاهداده نمی‌رسد.

## ۸ استراتژی برتر برای خروج از کش

استراتژی‌های خروج از کش (Cache Eviction)، رویکردهایی برای مدیریت فضای ذخیره‌سازی کش هستند و تعیین می‌کنند که کدام آیتم‌ها در صورت پرشدن کش، باید حذف شوند. در این مقاله، ۸ استراتژی برتر برای خروج از کش را بررسی می‌کنیم.



### (کمترین زمان دسترسی اخیر - LRU)

استراتژی LRU، کم کاربردترین آیتم‌های اخیر را ابتدا حذف می‌کند. این رویکرد بر این اصل استوار است که آیتم‌هایی که اخیراً به آن‌ها دسترسی پیدا کرده‌ایم، به احتمال زیاد در آینده نزدیک دوباره مورد نیاز خواهند بود.

### (بیشترین زمان دسترسی اخیر - MRU)

برخلاف LRU، الگوریتم MRU، پربازدیدترین آیتم‌های اخیر را ابتدا حذف می‌کند. این استراتژی در سناریوهایی مفید است که در آن‌ها احتمال دسترسی مجدد به تازه‌ترین موارد بازدید شده، کمتر باشد.

### (Segmented LRU) SLRU

SLRU، کش را به دو بخش تقسیم می‌کند: بخش آزمایشی و بخش محافظت‌شده. آیتم‌های جدید ابتدا در بخش آزمایشی قرار می‌گیرند. اگر دوباره به یک آیتم در بخش آزمایشی دسترسی پیدا شود، به بخش محافظت‌شده ارتقا پیدا می‌کند.

### (کمترین دفعات دسترسی - LFU)

الگوریتم LFU آیتم‌هایی را که کمترین دفعات دسترسی را داشته‌اند، حذف می‌کند.

### (اولین ورودی، اولین خروجی - FIFO)

FIFO یکی از ساده‌ترین استراتژی‌های کش است که در آن، کش مانند یک صف عمل می‌کند و صرف نظر از الگوی دسترسی یا تعداد دفعات بازدید، قدیمی‌ترین آیتم‌ها را ابتدا حذف می‌کند.

### (TTL) مدت زمان باقی‌ماندن -

TTL، گرچه به طور مستقیم یک استراتژی خروج نیست، روشی است که در آن به هر آیتم کش یک طول عمر مشخص اختصاص داده می‌شود. پس از منقضی شدن این زمان، آیتم به طور خودکار از کش حذف می‌گردد.

### (Two-Tiered Caching) کش دوستحی

در استراتژی کش دوستحی، از یک کش درون حافظه<sup>۱</sup> برای لایه اول و یک کش توزیع شده<sup>۲</sup> برای لایه دوم استفاده می‌شود. این روش باعث بهبود کارایی و ظرفیت کلی کش می‌شود.

### (RR - Random Replacement) جایگزینی تصادفی

الگوریتم جایگزینی تصادفی، به صورت تصادفی یک آیتم کش را انتخاب کرده و برای ایجاد فضای بیشتر برای آیتم‌های جدید، آن را حذف می‌کند. این روش ساده است و نیازی به ردیابی الگوهای دسترسی یا تعداد دفعات بازدید شده در گذشته را ندارد.

---

in-memory cache<sup>۱</sup>

distributed cache<sup>۲</sup>

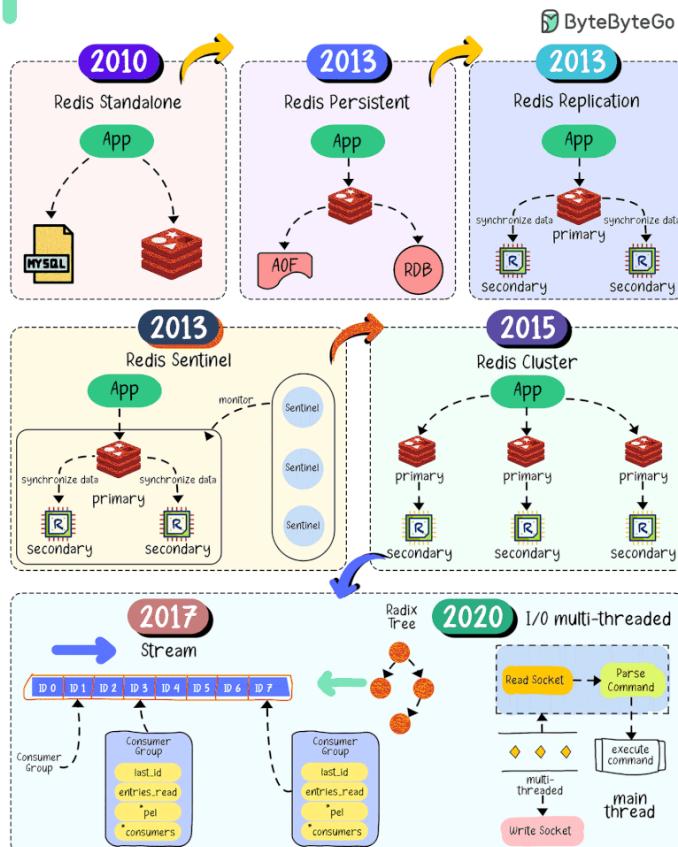
## تکامل معماری Redis

همان طور که ضرب المثل می‌گوید، روم هم در یک روز ساخته نشده است!

چگونه معماری Redis تکامل پیدا کرده است؟

در واقع Redis یک کش محبوب درون حافظه است. چگونه به معماری امروزی خود تکامل یافته است؟

### How Redis Architecture Evolves ?



### Standalone Redis – ۲۰۱۰

زمانی که Redis 1.0 در سال ۲۰۱۰ منتشر شد، معماری آن بسیار ساده بود. معمولاً<sup>۱</sup> به عنوان یک کش برای برنامه‌های تجاری استفاده می‌شد.

با این حال، Redis داده‌ها را در حافظه ذخیره می‌کند. هنگامی که Redis را مجدداً راهاندازی می‌کنیم، تمام داده‌ها از بین می‌روند و ترافیک مستقیماً به پایگاهداده برخورد می‌کند.

### Persistence – ۲۰۱۳

با انتشار Redis 2.8 در سال ۲۰۱۳، محدودیت‌های قبلی برطرف شد. Redis اسپشات‌های درون حافظه RDB را برای ماندگاری داده‌ها معرفی کرد. همچنین از (فایل الحاقی<sup>۱</sup>) پشتیبانی می‌کند، جایی که هر دستور نوشتن در یک فایل AOF نوشته می‌شود.

### Replication – ۲۰۱۳

Redis 2.8 همچنین برای افزایش دردسترس‌بودن، تکثیر را اضافه کرد. نمونه اصلی درخواست‌های خواندن و نوشتن بلادرنگ را مدیریت می‌کند، در حالی که نمونه‌بردار داده‌های نمونه اصلی را همگام‌سازی می‌کند.

### Sentinel – ۲۰۱۳

Sentinel را برای نظارت بر نمونه‌های Redis به صورت بلادرنگ معرفی کرد. این یک سیستم است که برای کمک به مدیریت نمونه‌های Redis طراحی شده است. این failover monitoring, notification, automatic و .configuration provider

### Cluster – ۲۰۱۵

در سال ۲۰۱۵ Redis 3.0 منتشر شد. این نسخه خوشه‌های Redis را اضافه کرد. یک خوشه Redis یک راه حل پایگاهداده توزیع شده است که داده‌ها را از طریق sharding

مدیریت می‌کند. داده‌ها به ۱۶۳۸۴ شکاف تقسیم می‌شوند و هر گره مسئول بخشی از یک slot است.

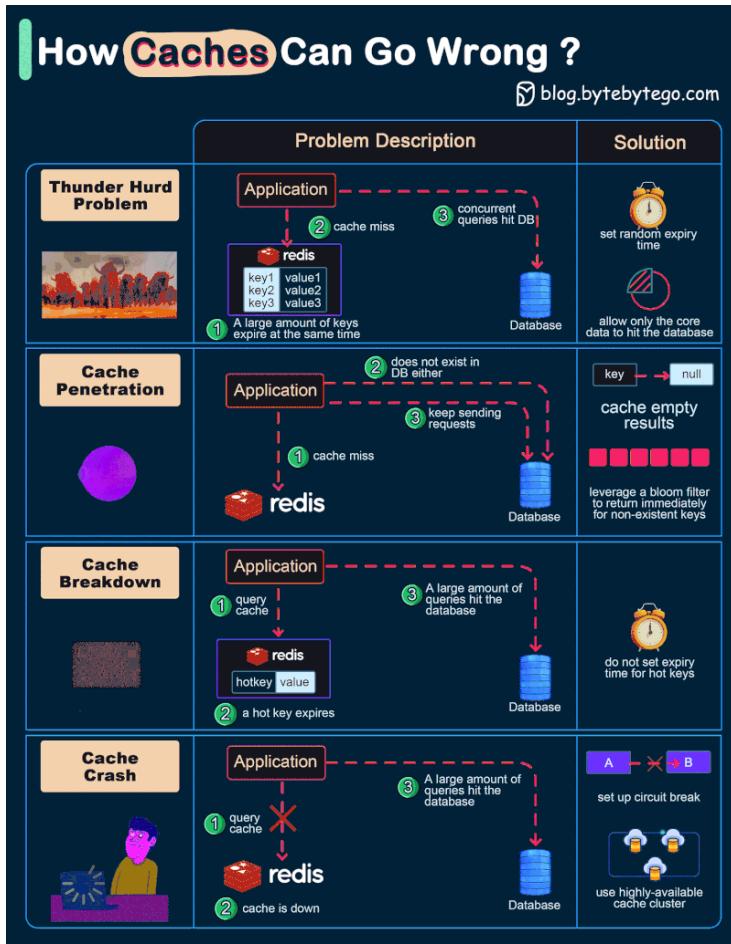
### نگاهی به آینده

همین طور Redis به دلیل عملکرد بالا و ساختارهای داده غنی که به طور چشمگیری پیچیدگی توسعه یک برنامه تجاری را کاهش می‌دهد، محبوب است. در سال ۲۰۱۷، نسخه Redis 5.0 منتشر شد و نوع داده جریان (stream) را اضافه کرد.

در سال ۲۰۲۰، نسخه Redis 6.0 منتشر شد و ورودی/خروجی چندرشته‌ای را در ماژول شبکه معرفی کرد. مدل Redis به ماژول شبکه و ماژول پردازش اصلی تقسیم می‌شود. به نظر می‌رسد توسعه‌دهندگان Redis، ماژول شبکه را به گلوگاهی در سیستم تبدیل کرده‌اند.

## وقتی سیستم کش دچار مشکل می‌شود؟

چگونه سیستم‌های کش دچار مشکل می‌شوند؟



### ۱. مشکل Thunder Herd

این اتفاق زمانی می‌افتد که تعداد زیادی از کلیدها در کش به طور همزمان منقضی شوند. سپس درخواست‌های کوئری مستقیماً به پایگاهداده برخورد می‌کنند که باعث اضافه‌بار روی پایگاهداده می‌شود. برای کاهش این مشکل دوراه وجود دارد: یکی اینکه از تنظیم زمان انقضای یکسان برای کلیدها اجتناب کنید و یک عدد تصادفی در پیکربندی اضافه کنید. راه

دیگر این است که فقط اجازه دهید داده‌های اصلی و business به پایگاهداده دسترسی پیدا کنند و از دسترسی داده‌های غیرهسته‌ای به پایگاهداده تا زمانی که کش دوباره راهاندازی شود، جلوگیری کنید.

#### ۲. نفوذ به کش (Cache Penetration):

این اتفاق زمانی می‌افتد که کلید در کش یا پایگاهداده وجود نداشته باشد و برنامه نمی‌تواند داده‌های مرتبط را از پایگاهداده برای بهروزرسانی کش بازیابی کند. این مشکل فشار زیادی را هم بر روی کش و هم بر روی پایگاهداده وارد می‌کند. دو پیشنهاد برای حل این مشکل وجود دارد: یکی اینکه برای کلیدهای غیر موجود، مقدار null را در کش ذخیره کنید تا از دسترسی به پایگاهداده جلوگیری شود. راه دیگر استفاده از فیلتر بلوم<sup>۱</sup> برای بررسی وجود کلید است، در صورتی که کلید وجود نداشته باشد، می‌توانیم از دسترسی به پایگاهداده اجتناب کنیم.

#### ۳. فروپاشی کش (Cache Breakdown):

این مورد مشابه مشکل Thunder Herd است. این زمانی اتفاق می‌افتد که یک hot key منقضی شود. تعداد زیادی از درخواست‌ها به پایگاهداده برخورده باشند. از آنجایی که hot key ها ۸۰ درصد از درخواست‌ها را تشکیل می‌دهند، ما برای آن‌ها زمان انقضا تنظیم نمی‌کنیم.

#### ۴. خرابی کش (Cache Crash):

این زمانی اتفاق می‌افتد که کش از کار بیفتد و تمام درخواست‌ها به پایگاهداده هدایت شوند. دوراه برای حل این مشکل وجود دارد: یکی اینکه یک قطع‌کننده مدار<sup>۲</sup> تنظیم کنیم و زمانی که کش از کار افتاد، سرویس‌های برنامه نمی‌توانند به کش یا پایگاهداده دسترسی پیدا کنند. راه دیگر راهاندازی یک خوشه<sup>۳</sup> برای کش است تا در دسترس بودن کش را بهبود بخشد.

---

<sup>۱</sup>Bloom Filter

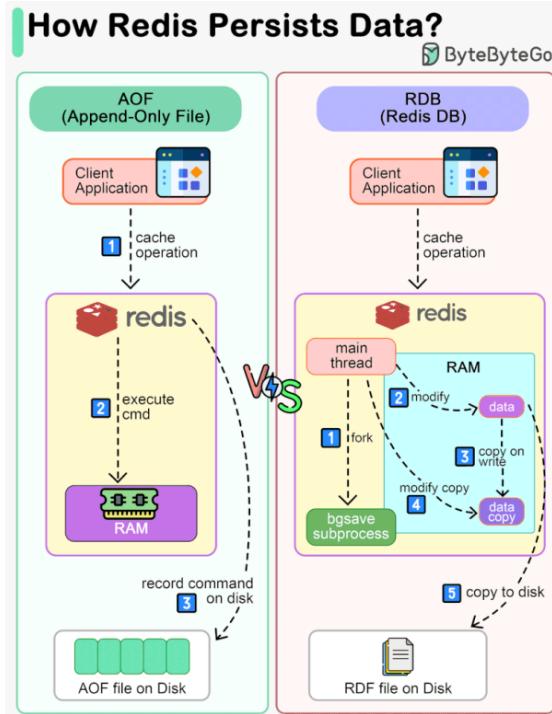
<sup>۲</sup>circuit breaker

<sup>۳</sup>cluster

## in-memory داده پایگاه: Redis

یک پایگاه داده درون حافظه<sup>۱</sup> است و اگر سرور خاموش شود، داده‌ها از بین می‌روند. نمودار زیر دو روش برای پایدارسازی داده‌های Redis روی دیسک را نشان می‌دهد:

۱. RDB (Redis Database) و ۲. AOF (Append-Only File).



توجه داشته باشید که پایدارسازی داده‌ها روی مسیر بحرانی انجام نمی‌شود و فرایند نوشتمن در Redis را مسدود نمی‌کند.

**AOF •**

برخلاف یک لاغ write-ahead، Redis، یک لاغ AOF است. ابتدا دستورات را برای تغییر داده‌ها در حافظه اجرا می‌کند و سپس آن را در فایل لاغ می‌نویسد. لاغ AOF دستورات را به جای داده‌ها ثبت می‌کند. طراحی مبتنی بر رویداد، بازیابی داده‌ها را ساده می‌کند. علاوه بر این، AOF دستورات را پس از اجرای دستور در حافظه ثبت می‌کند، بنابراین عملیات نوشتمن فعلی را مسدود نمی‌کند.

### RDB •

محدودیت AOF این است که دستورات را به جای داده‌ها persists می‌کند. وقتی از لاغ AOF برای بازیابی استفاده می‌کنیم، کل لاغ باید اسکن شود. وقتی اندازه لاغ بزرگ است، زمان زیادی برای بازیابی می‌برد. بنابراین Redis روش دیگری برای پایدارسازی داده‌ها ارائه می‌دهد و RDB یک از آن روش‌ها است. درواقع RDB اسنپشات‌های<sup>۱</sup> از داده‌ها را در نقاط زمانی مشخص ثبت می‌کند. وقتی سرور نیاز به بازیابی دارد، اسنپشات‌های داده‌ها را می‌تواند مستقیماً برای بازیابی سریع به حافظه بارگذاری شود.

مرحله ۱: thread اصلی sub-process مربوط به bgsave را ایجاد می‌کند که تمام داده‌های درون حافظه thread (in-memory) را به اشتراک می‌گذارد. bgsave داده‌ها را از thread اصلی می‌خواند و آن را در فایل RDB می‌نویسد.

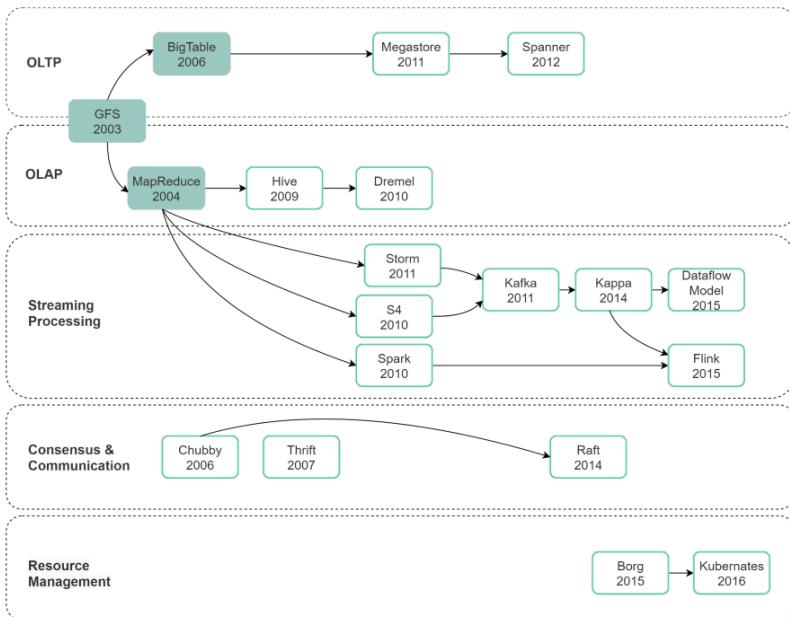
مراحل ۲ و ۳: اگر thread اصلی داده‌ها را تغییر دهد، یک کپی از داده‌ها ایجاد می‌شود. مرحله ۴ و ۵: thread اصلی سپس روی کپی داده‌ها عمل می‌کند. در همین حال، sub-process مربوط به bgsave همچنان داده‌ها را در فایل RDB می‌نویسد.

### • ترکیبی

معمولًاً در سیستم‌های تولید، می‌توانیم یک رویکرد ترکیبی را انتخاب کنیم، که در آن از RDB برای ثبت اسنپشات‌های داده‌ها در فواصل زمانی مشخص استفاده می‌کنیم و از AOF برای ثبت دستورات از آخرین اسنپشات‌های استفاده می‌کنیم.

# مقالات Big data

در زیر یک نمودار از مقالات مهم Big data و تکامل تکنیک‌های آن در طول زمان است.



کادرهای سبز پرنگ، ۳ مقاله معروف گوگل هستند که پایه و اساس چارچوب Big data را بنا نهادند. در سطح بالا:

تکنیک‌های Big data = داده عظیم + محاسبات عظیم

بیایید به تکامل OLTP نگاه کنیم. BigTable یک سیستم ذخیره‌سازی توزیع شده برای داده‌های ساختاریافته ارائه داد؛ اما برخی از ویژگی‌های پایگاه‌داده رابطه‌ای را حذف کرد. سپس Megastore طرح‌ها و تراکنش‌های ساده را بازگرداند؛ Spanner یکپارچگی داده را بازگرداند.

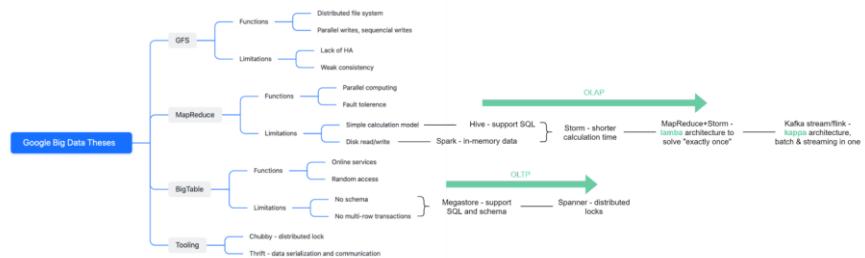
اکنون باید به تکامل OLAP نگاه کنیم. برنامه‌نویسی آسانی نداشت، بنابراین Hive با معرفی یک زبان کوئری شبیه SQL این مشکل را حل کرد. اما هنوز در لایه‌های زیرین خود از MapReduce استفاده می‌کرد، بنابراین برنامه‌ای فوق العاده‌ای نبود. در سال ۲۰۱۰، Dremel یک موتور کوئری تعاملی ارائه داد.

پردازش جریانی<sup>۱</sup> برای حل مشکل تأخیر در OLAP به وجود آمد. معروف‌ترین معماری لامبда<sup>۲</sup> بر اساس Storm و MapReduce بود، جایی که پردازش جریانی و پردازش دسته‌ای جریان‌های پردازش متفاوتی داشتند. سپس مردم شروع به ساخت پردازش جریانی با apache Kafka کردند. معماری **Kappa** در سال ۲۰۱۴ پیشنهاد شد، جایی که پردازش جریانی و پردازش دسته‌ای در یک جریان ادغام شدند. گوگل در سال ۲۰۱۵ مدل جریان داده را منتشر کرد که یک استاندارد انتزاعی برای پردازش جریانی بود و Flink این مدل را پیاده‌سازی کرد.

برای مدیریت یک گروه بزرگ از منابع سرور رایج، ما نیاز به مدیریت منابع Kubernetes داریم.

## تکامل Big data

امیدوارم همه در تعطیلات، زمان خوبی را با دوستان و خانواده گذرانده باشید. اگر به دنبال مطالعه هستید، مقالات کلاسیک مهندسی یک شروع خوب است.

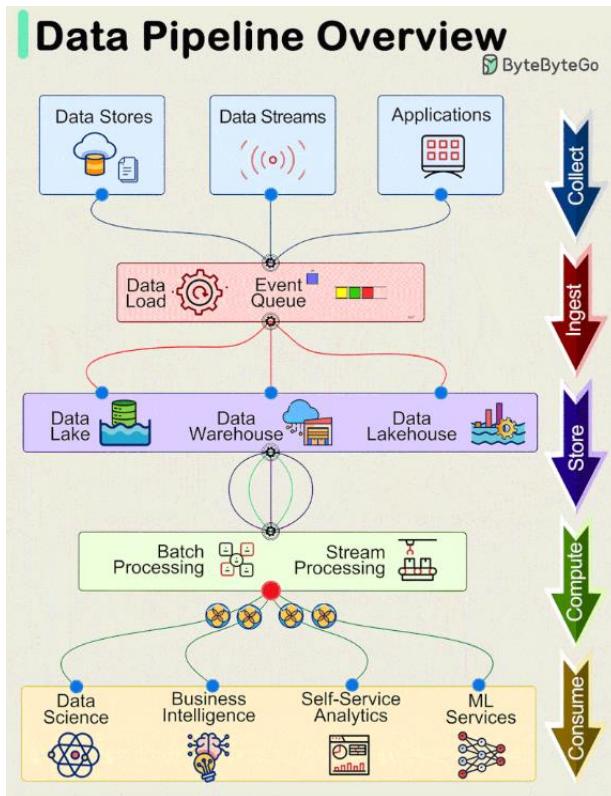


بسیاری اوقات زمانی که با کار مشغول هستیم، تنها بر اطلاعات پراکنده تمرکز می‌کنیم که به ما می‌گوید «چگونه» و «چه چیزی» برای برآورده کردن نیازهای فوری مان برای انجام کارها لازم داریم. با این حال، خواندن کلاسیک‌ها به ما کمک می‌کند تا «علت‌های» پشت صحنه را بدانیم و به ما می‌آموزد چگونه مسائل را حل کنیم و تصمیمات بهتری بگیریم یا حتی به پروژه‌های open source کمک کنیم. بیایید Big data را به عنوان مثال در نظر بگیریم. حوزه Big data در ۲۰ سال گذشته پیشرفت زیادی داشته است. این از ۳ مقاله گوگل (لینک‌ها را در نظرات ببینید) شروع شد که چالش‌های مهندسی واقعی را در مقیاس گوگل حل می‌کرد:

- GFS (۲۰۰۳) - ذخیره‌سازی داده بزرگ
- MapReduce (۲۰۰۴) - مدل محاسباتی
- BigTable (۲۰۰۶) - سرویس‌های آنلاین

## Data Pipelines بررسی

خطوط لوله داده (Data Pipelines) جزء اساسی مدیریت و پردازش کارآمد داده در سیستم‌های مدرن هستند. این خطوط لوله معمولاً شامل ۵ مرحله اصلی هستند: جمع‌آوری، جذب، ذخیره، محاسبه و مصرف.



### ۱. جمع‌آوری (Collect)

داده‌ها از ذخیره‌گاه داده<sup>۱</sup>، جریان‌های داده<sup>۲</sup> و اپلیکیشن‌ها جمع‌آوری می‌شوند. این منابع می‌توانند به صورت remote از دستگاه‌ها، برنامه‌ها یا سیستم‌های تجاری نتیجه بگیرند.

<sup>۱</sup> data stores

<sup>۲</sup> data streams

۲. جذب (Ingest):

در طی فرایند جذب، داده‌ها به سیستم‌ها بارگذاری شده و درون صفحه‌های رویداد<sup>۱</sup> سازماندهی می‌شوند.

۳. ذخیره (Store):

پس از جذب، داده‌های سازماندهی شده در انبارهای داده<sup>۲</sup>، data lakehouse و data lake به همراه سیستم‌های مختلف مانند پایگاه‌داده‌ها، برای اطمینان از ذخیره‌سازی پس از جذب، نگهداری می‌شوند.

۴. محاسبه (Compute):

داده‌ها برای مطابقت با استانداردهای شرکت، تحت تجمعیع، پاکسازی و دستکاری قرار می‌گیرند. این شامل کارهایی مانند تبدیل فرمت، فشرده‌سازی داده و پارتبین‌بندی می‌شود. این مرحله از تکنیک‌های پردازش دسته‌ای و جریانی استفاده می‌کند.

۵. مصرف (Consume):

داده‌های پردازش شده از طریق ابزارهای تجزیه و تحلیل و تجسم، انبارهای داده عملیاتی، موتورهای تصمیم‌گیری، Application‌های کاربرمحور، داشبوردها، علم داده، سرویس‌های یادگیری ماشین، هوش تجاری و تجزیه و تحلیل خود سرویس در دسترس مصرف قرار می‌گیرد.

کارایی و اثربخشی هر مرحله در موفقیت کلی عملیات مبنی بر داده در یک سازمان نقش بسزایی دارد.

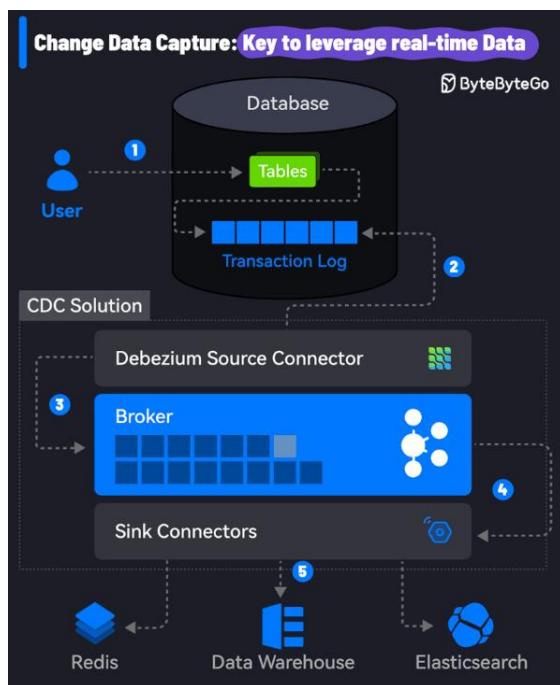
---

۱ event queue

۲ data warehouse

## سرعت رشد بالا داده‌ها

۹۰ درصد از داده‌های جهان در دو سال گذشته ایجاد شده است و این رشد سرعت بیشتری خواهد گرفت. با این حال، بزرگ‌ترین چالش، استفاده از این داده‌ها به صورت بلاذرنگ است. تغییرات مداوم داده‌ها باعث می‌شود پایگاه‌های داده، دریاچه‌های داده<sup>۱</sup> و انبارهای داده<sup>۲</sup> همگام‌سازی نشوند.



CDC<sup>۳</sup> یا تغییر داده‌برداری می‌تواند به شما در غلبه بر این چالش کمک کند. تغییرات اعمال شده بر روی داده‌ها در یک پایگاه داده را شناسایی و ضبط می‌کند و به شما امکان تکثیر و همگام‌سازی داده‌ها در چندین سیستم را می‌دهد.

تغییرات داده‌برداری چگونه کار می‌کند؟

data lakes<sup>۱</sup>

data Warehouses<sup>۲</sup>

Change Data Capture<sup>۳</sup>

در اینجا یک توضیح مرحله‌به‌مرحله وجود دارد:

۱. **تغییرات داده:** تغییری در داده‌های پایگاهداده منع ایجاد می‌شود. این تغییر می‌تواند یک عملیات درج (insert)، بهروزرسانی (update) یا حذف (delete) روی یک جدول باشد.
۲. **ضبط تغییرات:** یک ابزار CDC، گزارش تراکنش‌های پایگاهداده را برای ضبط تغییرات، مانیتور می‌کند. این ابزار از یک source connector برای اتصال به پایگاهداده و خواندن گزارش‌ها استفاده می‌کند.
۳. **پردازش تغییر:** تغییرات ضبط شده پردازش و به فرمتی مناسب برای سیستم‌های پایین‌دستی<sup>۱</sup> تبدیل می‌شوند.
۴. **انتشار تغییر:** تغییرات پردازش شده در یک صفحه<sup>۲</sup> منتشر شده و به سیستم‌های هدف، مانند انبارهای داده، پلتفرم‌های تحلیلی، حافظه‌های کش توزیع شده مانند Redis و غیره منتقل می‌شوند.
۵. **یکپارچه‌سازی بلاذرنگ:** ابزار CDC از sink connector برای دریافت گزارش و بهروزرسانی سیستم‌های هدف استفاده می‌کند. تغییرات به صورت بلاذرنگ دریافت می‌شوند و امکان تجزیه و تحلیل داده‌ها و تصمیم‌گیری بدون تداخل را فراهم می‌کنند.

کاربران فقط نیاز به انجام مرحله ۱ دارند و سایر مراحل به صورت واضح انجام می‌شوند. یکی از راه حل‌های محبوب CDC، استفاده از Kafka Connect با Debezium است که از Kafka broker به عنوان Kafka broker برای انتقال جریان تغییرات داده از سیستم منع به سیستم‌های هدف استفاده می‌کند. Debezium دارای اتصال‌دهنده‌هایی برای اکثر پایگاه‌های داده مانند Oracle، PostgreSQL، MySQL

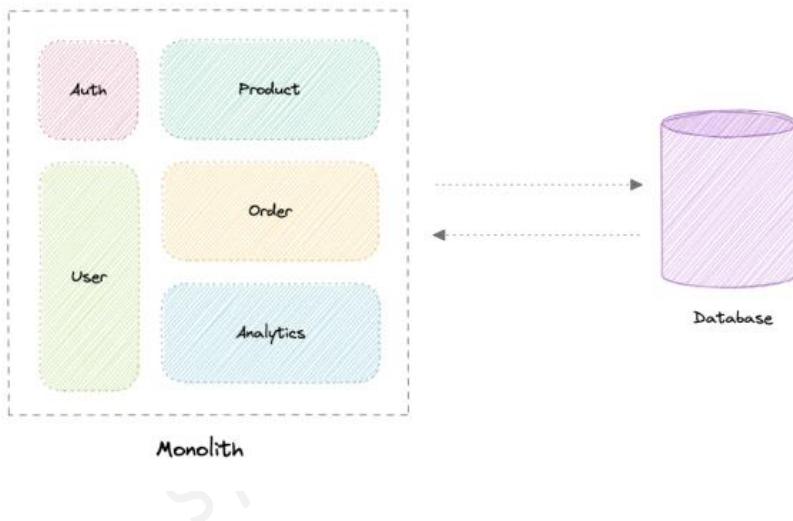
<sup>۱</sup> downstream systems

<sup>۲</sup> message queue

# معماری‌های میکروسرویس

## (تک‌هسته‌ای) monolith

یک monolith یک برنامه‌ی مستقل و تک‌هسته‌ای است. این برنامه به عنوان یک واحد جداگانه ساخته شده است و نه تنها برای یک وظیفه‌ی خاص، بلکه برای انجام تمام مراحل موردنیاز برای بروزرسانی کسب‌وکار مسئول است.



مزایا:

در ادامه برخی از مزایای monolith‌ها آورده شده است:

- توسعه یا اسکالرزدایی ساده.
- ارتباط سریع و قابل اعتماد.
- مانیتورینگ و تست آسان.
- پشتیبانی از تراکنش‌های ACID

## معایب

- برخی از معایب رایج monolith‌ها عبارت‌اند از:
- با بزرگ‌شدن کد، نگهداری آن سخت‌تر می‌شود.
  - برنامه‌ی وابستگی شدید به اجزای خود دارد و گسترش آن سخت است.
  - نیاز به متعهد ماندن به یک stack فناوری خاص دارد.
  - در هر بروزرسانی، کل برنامه دوباره مستقر می‌شود.
  - کاهش قابلیت اطمینان به دلیل اینکه یک باگ واحد می‌تواند کل سیستم را از کار بیندازد.
  - مقیاس‌پذیری یا به کارگیری فناوری‌های جدید دشوار است.

## ماژولار Monolith

یک تک‌هسته‌ای ماژولار رویکردی است که در آن ما یک برنامه‌ی واحد (بخش Monolith) را می‌سازیم و مستقر می‌کنیم، اما آن را به‌گونه‌ای می‌سازیم که کد را به ماژول‌های مستقل برای هر یک از ویژگی‌های موردنیاز در برنامه‌ی ما تقسیم می‌کند.

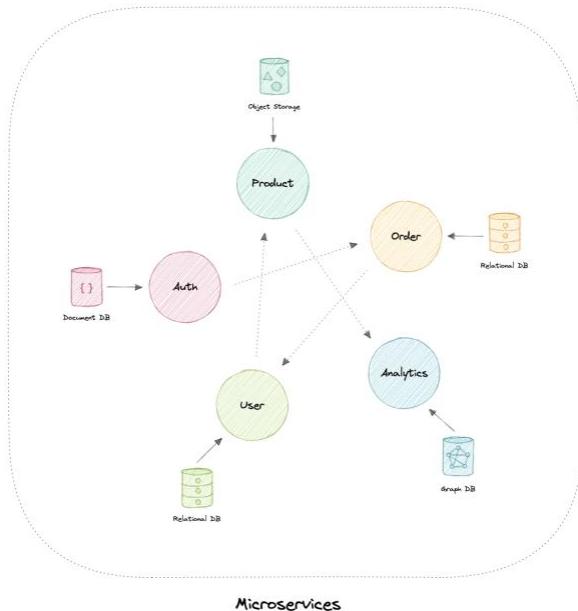
این رویکرد وابستگی‌های یک ماژول را به‌گونه‌ای کاهش می‌دهد که بتوانیم یک ماژول را بدون تأثیر بر ماژول‌های دیگر ارتقا دهیم یا تغییر دهیم. هنگامی که به درستی پیاده‌سازی و انجام شود، می‌تواند در درازمدت بسیار مفید باشد؛ زیرا پیچیدگی ناشی از نگهداری یک تک‌هسته‌ای را با رشد سیستم کاهش می‌دهد.

## میکروسرویس‌ها (Microservices)

یک معماری میکروسرویس از مجموعه‌ای از سرویس‌های کوچک و مستقل تشکیل شده است که در آن هر سرویس کاتینر شده هستند و باید یک قابلیت تجاری واحد را در یک محدوده‌ی اختصاصی<sup>۱</sup> پیاده‌سازی کند. یک محدوده‌ی اختصاصی یک تقسیم‌بندی طبیعی از

<sup>۱</sup> bounded context

منطق کسب‌وکار است که یک مرز صریح را فراهم می‌کند که در آن یک مدل دامنه<sup>۱</sup> وجود دارد.



هر سرویس دارای یک codebase جداگانه است که می‌تواند توسط یک تیم توسعه‌ی کوچک مدیریت شود. سرویس‌ها می‌توانند به طور مستقل مستقر شوند و یک تیم می‌تواند یک سرویس موجود را بدون نیاز به بازسازی و استقرار مجدد کل برنامه بهروزرسانی کند. سرویس‌ها مسئول نگهداری داده‌های خود یا وضعیت خارجی (پایگاهداده بهزای هر سرویس) هستند. این با مدل سنتی که در آن یک‌لایه‌ی داده‌ی جداگانه مسئولیت پایداری داده‌ها را بر عهده دارد، متفاوت است.

### ویژگی‌ها:

معماری میکروسرویس دارای ویژگی‌های زیر است:

<sup>1</sup> domain model

**وابستگی ضعیف<sup>۱</sup>:** سرویس‌ها باید وابستگی ضعیفی نسبت به هم داشته باشند تا بتوانند به طور مستقل مستقر و مقیاس‌بندی<sup>۲</sup> شوند. این کار منجر به تمرکز زدایی تیم‌های توسعه و در نتیجه امکان توسعه و استقرار سریع‌تر برنامه با محدودیت‌ها و وابستگی‌های عملیاتی کم‌تر می‌شود.

**کوچک اما متمرکز:** این در مورد محدوده و مسئولیت‌ها است و نه اندازه‌ی برنامه، یک سرویس باید روی یک مشکل خاص متمرکز شود. اساساً، «یک کار را انجام می‌دهد و آن را به خوبی انجام می‌دهد». در حالت ایده‌آل، آن‌ها می‌توانند مستقل از زیرساخت پایه‌ای باشند. مناسب برای هدف واحد در **کسب و کار**: معماری میکروسرویس معمولاً حول قابلیت‌ها و اولویت‌های کسب و کار سازماندهی می‌شود.

**انعطاف‌پذیری و تحمل خطأ:** سرویس‌ها باید به گونه‌ای طراحی شوند که در صورت خرابی یا خطأ همچنان عمل کنند. در محیط‌هایی با سرویس‌های قابل استقرار مستقل، تحمل خطأ از بالاترین اهمیت برخوردار است.

**قابل نگهداری:** سرویس‌ها باید به راحتی قابل نگهداری و تست باشند؛ زیرا سرویس‌هایی که قابل نگهداری نیستند، بازنویسی خواهند شد.

**مزایا:**

در اینجا برخی از مزایای معماری میکروسرویس‌ها آورده شده است:

- **سرویس‌ها با پیوستگی ضعیف:** سرویس‌ها با یکدیگر وابستگی کمی دارند و می‌توانند به صورت مستقل مستقر و مقیاس‌بندی شوند. این امر باعث چابکی بیشتر تیم‌های توسعه شده و امکان توسعه و استقرار سریع‌تر را فراهم می‌کند.
- **استقرار مستقل سرویس‌ها:** هر سرویس را می‌توان بدون نیاز به بهروزرسانی کل برنامه، به طور مستقل توسعه، تست و مستقر کرد. این منجر به چرخه‌ی انتشار سریع‌تر و ریسک کمتر می‌شود.

<sup>۱</sup> Loosely Coupled

<sup>۲</sup> deployed and scaled

- **چابکی بالا برای تیم‌های توسعه‌ی مختلف:** تیم‌های توسعه می‌توانند به طور مستقل روی سرویس‌های مختلف کار کنند که منجر به چابکی و بهره‌وری بیشتر می‌شود.
- **بهبود تحمل خطأ و جداسازی داده‌ها:** خرابی یک سرویس بر سایر سرویس‌ها تأثیر نمی‌گذارد و داده‌های هر سرویس نیز جدا از سایر سرویس‌ها مدیریت می‌شود. این باعث افزایش پایداری و امنیت کلی سیستم می‌شود.
- **مقیاس‌پذیری بهتر:** هر سرویس را می‌توان به طور مستقل بر اساس نیازهایش مقیاس‌بندی کرد. این باعث استفاده‌ی بهینه‌تر از منابع و عملکرد کلی بهتر می‌شود.
- **عدم وابستگی بلندمدت به یک stack فناوری خاص:** هر سرویس می‌تواند با فناوری مناسب خود ساخته شود، بدون اینکه بر سایر سرویس‌ها تأثیر بگذارد. این انعطاف‌پذیری بیشتری را در انتخاب فناوری فراهم می‌کند.

#### معایب

همان‌طور که اشاره شد، معماری میکروسرویس مجموعه‌ی خاص خود از چالش‌ها را به همراه دارد:

- **پیچیدگی یک سیستم توزیع شده:** مدیریت یک سیستم با چندین سرویس مستقل می‌تواند پیچیده باشد و نیازمند زیرساخت و ابزار نظارت بیشتری است.
- **مشکل بودن تست:** تست تعامل بین سرویس‌ها می‌تواند چالش‌برانگیز باشد و نیازمند سناریوهای مختلف است.
- **هزینه‌ی بالای نگهداری:** نگهداری چندین سرویس مجزا، از جمله سرورها و پایگاه‌های داده، می‌تواند هزینه‌ی بیشتری نسبت به یک سیستم monolith داشته باشد.
- چالش‌های ارتباطی بین سرویس‌ها: برقراری ارتباط قابل اعتماد و کارآمد بین سرویس‌ها می‌تواند پیچیده باشد.
- **یکپارچگی و انسجام داده‌ها:** اطمینان از یکپارچگی و انسجام داده‌ها در سراسر سرویس‌های مختلف نیازمند توجه و مدیریت دقیق است.

- شلوغی شبکه و تأخیرهای مرتبط: برقراری ارتباط بین سرویس‌های مختلف می‌تواند باعث ایجاد ترافیک شبکه و تأخیر شود.

### احتیاط: مواظب Monolith توزیع شده باشید

تک‌هسته‌ای/Monolith توزیع شده سیستمی شبیه معماری میکروسرویس است، اما در عین حال مانند یک برنامه‌ای تک‌هسته‌ای وابستگی شدیدی در درون اجزای خود دارد. معماری میکروسرویس مزایای زیادی به همراه دارد. اما در هنگام ساخت آن، احتمال زیادی وجود دارد که با یک تک‌هسته‌ای توزیع شده مواجه شویم.

اگر هر یک از موارد زیر در مورد میکروسرویس‌های ما صدق کند، آن‌ها فقط یک تک‌هسته‌ای توزیع شده هستند:

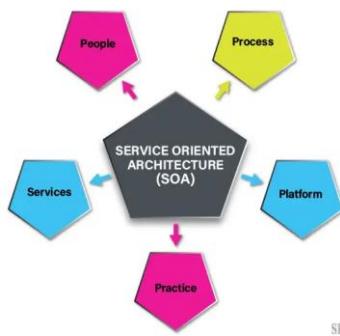
- نیاز به ارتباط با تأخیر زمانی کم: برقراری ارتباط بین سرویس‌ها باید با تأخیر کم انجام شود. این کار می‌تواند انعطاف‌پذیری و مقیاس‌پذیری را محدود کند.
- سرویس‌ها به راحتی مقیاس‌پذیر نیستند: مقیاس‌پذیری هر سرویس به صورت مستقل دشوار است.
- وابستگی بین سرویس‌ها: سرویس‌ها وابستگی شدیدی به یکدیگر دارند و نمی‌توانند به طور مستقل عمل کنند.
- اشتراک منابع مشابه مانند پایگاه‌داده: سرویس‌ها از منابع مشترکی مانند پایگاه‌داده استفاده می‌کنند که می‌تواند منجر به گلوبال و مشکلات مدیریت آن شود.
- سیستم‌هایی با پیوستگی قوی<sup>۱</sup>: سرویس‌ها به شدت به یکدیگر پیوسته هستند و تغییرات در یک سرویس می‌تواند بر عملکرد سایر سرویس‌ها تأثیر بگذارد.

یکی از دلایل اصلی ساخت یک برنامه با استفاده از معماری میکروسرویس، داشتن قابلیت مقیاس‌پذیری است؛ بنابراین، میکروسرویس‌ها باید سرویس‌های وابستگی سست داشته باشند

که امکان مستقل بودن هر سرویس را فراهم می‌کند. معماری تک‌هسته‌ای توزیع شده این قابلیت را از بین می‌برد و باعث می‌شود اکثر اجزا به یکدیگر وابسته شوند که پیچیدگی طراحی را افزایش می‌دهد.

### میکروسرویس در مقابل معماری سرویس‌گرا (SOA)

شاید در اینترنت با معماری سرویس‌گرا<sup>۱</sup> (SOA) برخورد کرده باشید، گاهی اوقات حتی به جای معماری میکروسرویس‌ها از آن یاد می‌شود، اما این دو با هم متفاوت هستند و تمایز اصلی بین این دو رویکرد به محدوده کاری آن‌ها برمی‌گردد.



معماری سرویس‌گرا (SOA) روشی را برای استفاده‌ی مجدد از اجزای نرم‌افزاری از طریق واسطه‌های سرویس (service interfaces) تعریف می‌کند. این واسطه‌ها از استانداردهای ارتباطی مشترک استفاده می‌کنند و بر به حداقل رساندن قابلیت استفاده‌ی مجدد سرویس‌های برنامه تمرکز دارند، در حالی که میکروسرویس‌ها به عنوان مجموعه‌ای از کوچک‌ترین واحدهای سرویس مستقل ساخته می‌شوند که بر استقلال تیم و جداسازی (decoupling) تمرکز دارند.

## چرا به میکروسرویس نیاز ندارید؟

بنابراین، ممکن است تعجب کنید، به نظر می‌رسد از ابتدا مونولیت‌ها ایده‌ی خوبی نیستند، پس چرا کسی از آن‌ها استفاده می‌کند؟

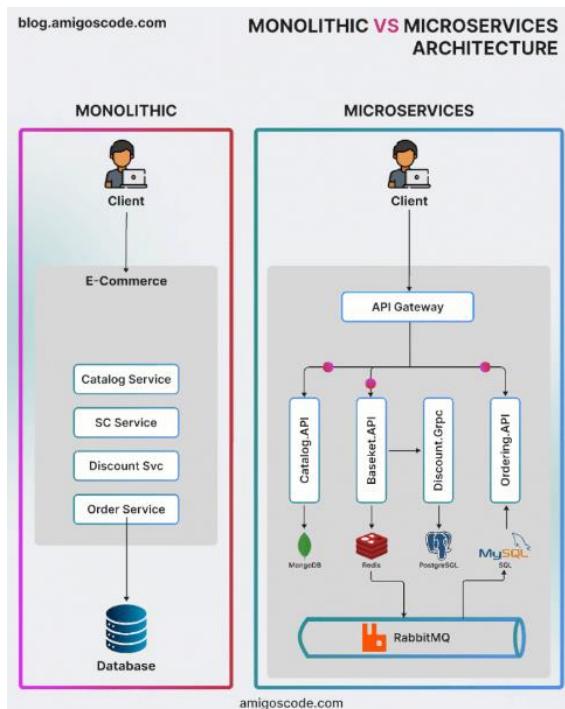


خب، بستگی دارد. در حالی‌که هر رویکردی مزایا و معایب خاص خود را دارد، توصیه می‌شود هنگام ساخت یک سیستم جدید با یک monolith شروع کنید. درک این نکته مهم است که میکروسرویس‌ها یک راه حل جادویی نیستند، بلکه یک مشکل سازمانی را حل می‌کنند. معماری میکروسرویس به همان اندازه که در مورد فناوری است، در مورد اولویت‌های سازمانی و تیم شما نیز هست.

قبل از اینکه تصمیم به حرکت به سمت معماری میکروسرویس بگیرید، باید از خود سؤالاتی مانند موارد زیر بپرسید:

- آیا تیم برای کار مؤثر روی یک codebase مشترک خیلی بزرگ است؟
- آیا تیم‌ها منتظر اتمام تسک‌های تیم‌های دیگر هستند؟
- آیا میکروسرویس‌ها ارزش تجاری واضحی را برای ما به ارمغان می‌آورند؟
- آیا کسب‌وکار من به اندازه‌ی کافی برای استفاده از میکروسرویس‌ها بالغ است؟
- آیا معماری فعلی ما با بار اضافی ارتباط ما را محدود می‌کند؟

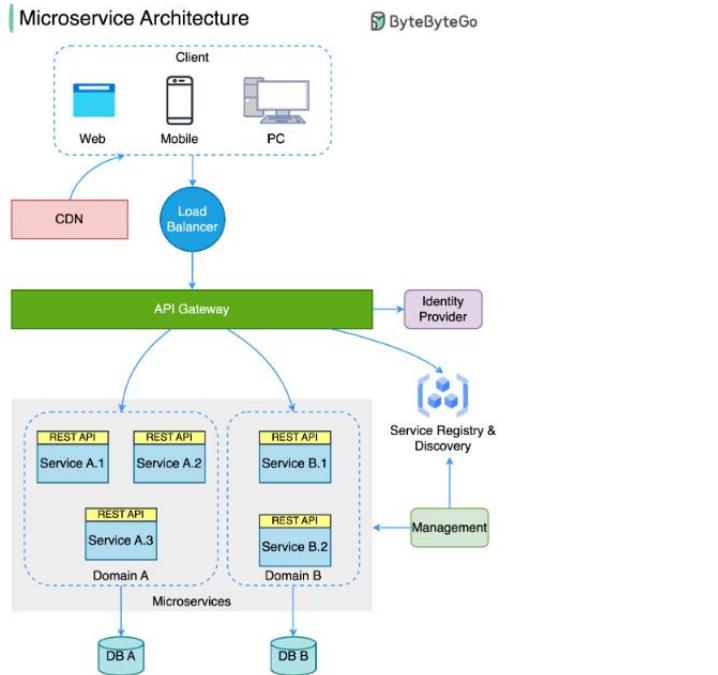
اگر برنامه‌ی شما نیازی به تجزیه‌شدن به میکروسرویس‌ها ندارد، به این موردنیاز ندارید و لزومی ندارد که همه‌ی برنامه‌ها به میکروسرویس‌ها تجزیه شوند.



ما اغلب از شرکت‌هایی مانند نتفلیکس و استفاده‌ی آن‌ها از میکروسرویس‌ها الهام می‌گیریم، اما این نکته را نادیده می‌گیریم که ما نتفلیکس نیستیم. آن‌ها از قبل برای اینکه راه حلی مناسب حوزه بازاری خود داشته باشند، تکرارها و مدل‌های زیادی را پشت سر گذاشته‌اند و این معماری زمانی برای آن‌ها قابل قبول شد که مشکلی را که سعی در حل آن داشتن شناسایی و حل کردند. به همین دلیل، درک عمیق این موضوع که آیا کسب و کار شما واقعاً به میکروسرویس نیاز دارد، ضروری است. چیزی که سعی می‌کنم بگویم این است که میکروسرویس‌ها راه حل‌هایی برای مشکلات پیچیده هستند و اگر کسب و کار شما مسائل پیچیده‌ای ندارد، به آن‌ها نیاز ندارید.

## بررسی یک نمونه از معماری میکروسرویس

تصویر زیر نمونه‌ای از یک معماری میکروسرویس را نشان می‌دهد.



- backend: این مؤلفه ترافیک ورودی را بین سرویس‌های **Load Balancer** • مختلف توزیع می‌کند.

- شبکه توزیع محتوا (CDN): شبکه توزیع محتوا مجموعه‌ای از سرورهای توزیع شده جغرافیایی است که محتوا استاتیک را برای ارائه سریع‌تر در خود جای می‌دهند. کاربران ابتدا در CDN به دنبال محتوا می‌گردند، سپس به سرویس‌های پکاند هدایت می‌شوند.

- API Gateway: این بخش درخواست‌های ورودی را مدیریت کرده و آنها را به سرویس‌های مرتبط هدایت می‌کند. دروازه API با تأمین‌کننده هویت و service discovery ارتباط برقرار می‌کند.

- تأمین‌کننده هويت (Identity Provider): اين بخش احراز هويت<sup>۱</sup> و مجوزدهی<sup>۲</sup> کاربران را مدیرiyت می‌کند.
- سرویس‌های Registry و Discovery: ثبت و کشف و شناسایی میکروسرویس‌ها در این قسمت اتفاق می‌افتد و API Gateway برای برقراری ارتباط با سرویس‌های مرتبط را در این قسمت شروع به جستجو می‌کند.
- مدیرiyت: اين بخش مسئولیت نظارت بر سرویس‌ها را بر عهده دارد.
- میکروسرویس‌ها: میکروسرویس‌ها در حوزه‌های مختلف طراحی و مستقر می‌شوند. هر حوزه پایگاهداده اختصاصی خود را دارد. API Gateway از طریق API یا پروتکل‌های دیگر با میکروسرویس‌ها صحبت می‌کند و میکروسرویس‌های درون یک حوزه با استفاده از RPC<sup>۳</sup> با هم دیگر ارتباط برقرار می‌کنند.

#### مزایای میکروسرویس‌ها:

- امکان طراحی، استقرار و مقیاس‌بندی افقی سریع<sup>۴</sup>.
- نگهداری مستقل هر حوزه توسط یک تیم اختصاصی.
- در نتیجه، امکان سفارشی‌سازی نیازمندی‌های کسب‌وکار در هر حوزه و پشتیبانی بهتر از آن‌ها.

authentication<sup>۱</sup>

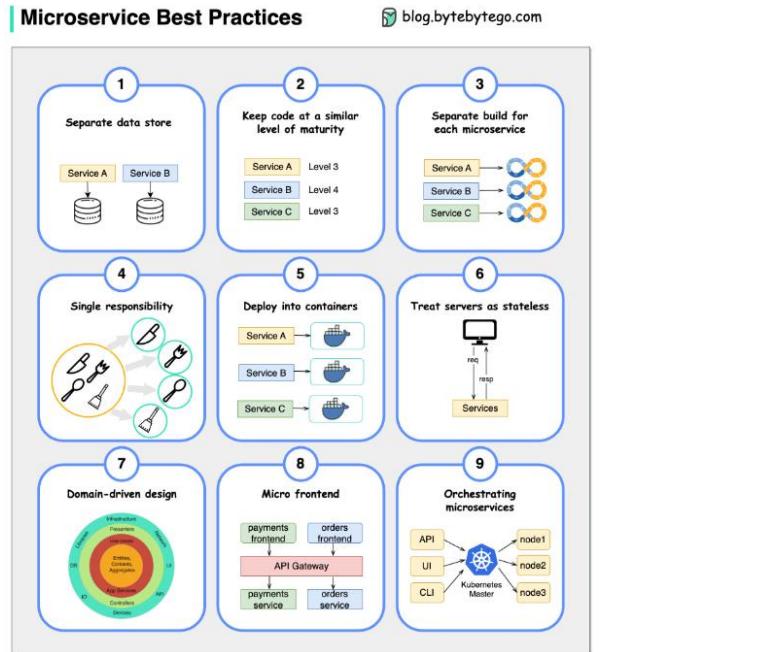
authorization<sup>۲</sup>

Remote Procedure Call<sup>۳</sup>

horizontally scale<sup>۴</sup>

## بهترین روش‌های میکروسرویس

یک تصویر به اندازه هزار کلمه حرف می‌زند:



۹ مورد از بهترین شیوه‌ها برای توسعه میکروسرویس‌ها  
هنگام توسعه میکروسرویس‌ها، باید بهترین شیوه‌های زیر را دنبال کنیم:

استفاده از ذخیره‌سازی داده مجزا برای هر میکروسرویس.

حفظ سطح مشابهی از مسیر تکامل کد.

ساخت مجزا برای هر میکروسرویس.

اختصاص دادن یک مسئولیت واحد به هر میکروسرویس.

استقرار به کمک کانتینرها

طراحی سرویس‌های Stateless

پذیرفتن Domain-Driven Design

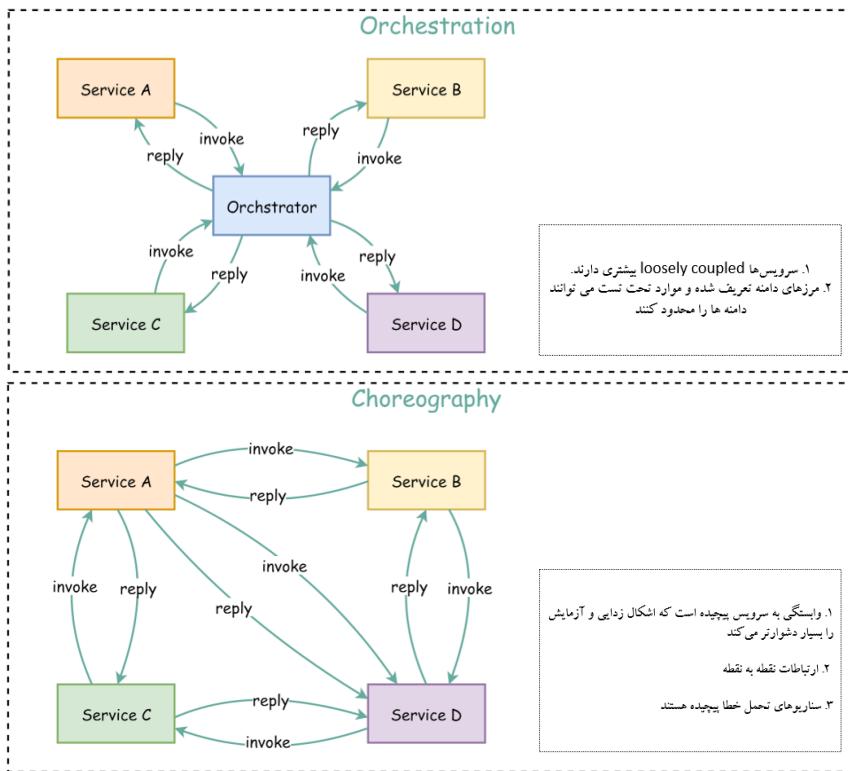
طراحی رابط Micro Frontend

ارکستراسیون میکروسرویس‌ها

## چگونه میکروسرویس‌ها با یکدیگر همکاری و تعامل دارند؟

دو روش وجود دارد: **choreography** و **orchestration**.

دیاگرام زیر همکاری میکروسرویس‌ها با یکدیگر را نشان می‌دهد.



**choreography** مانند داشتن یک کریوگراف<sup>۱</sup> است که تمام قوانین را تعیین می‌کند. سپس رقصندگان روی صحنه (میکروسرویس‌ها) طبق آنها تعامل دارند. کریوگرافی سرویس این تبادل پیام‌ها و قوانینی را که میکروسرویس‌ها طبق آن تعامل دارند، توصیف می‌کند. ارکستراسیون<sup>۲</sup> اما کمی متفاوت است. ارکسترатор به عنوان مرکز اقتدار عمل می‌کند. مسئول فرآخوانی و ترکیب سرویس‌ها است. این تعاملات بین تمام سرویس‌های شرکت‌کننده را

<sup>۱</sup> "choreograph" یا کورئوگرافی فرایند طراحی حرکات رقص است. این کلمه معمولاً در زمینه هنر رقص استفاده می‌شود. کورئوگراف هنرمندی است که حرکات رقص را طراحی می‌کند.

<sup>۲</sup> orchestration

توصیف می‌کند. دقیقاً مانند یک رهبر ارکستر است که نوازنده‌گان را در یک سمفونی موسیقی هدایت می‌کند. الگوی ارکستراسیون همچنین شامل مدیریت تراکنش بین سرویس‌های مختلف است.

### **:orchestration مزایای**

۱. قابلیت اطمینان<sup>۱</sup> – ارکستراسیون مدیریت تراکنش و رسیدگی به خطاهای را به صورت داخلی دارد، در حالی که کریوگرافی ارتباطات نقطه‌ای به نقطه‌ای است و سناریوهای تحمل خطا بسیار پیچیده‌تر هستند.

۲. مقیاس‌پذیری<sup>۲</sup> – هنگام افزودن یک سرویس جدید به ارکستراسیون، تنها ارکسترатор نیاز به تغییر قوانین تعامل دارد، در حالی که در کریوگرافی تمام سرویس‌های تعاملی باید تغییر کنند.

### **:orchestration محدودیت‌هایی**

۱. کارایی<sup>۳</sup> – تمام سرویس‌ها از طریق یک ارکسترатор مرکز صحبت می‌کنند، بنابراین تأخیر بیشتر از کریوگرافی است. همچنین، پنهانی باند محدود به ظرفیت ارکسترатор است.

۲. نقطه ضعف و شکست تکی<sup>۴</sup> – اگر ارکسترатор خراب شود، هیچ سرویسی نمی‌تواند با یکدیگر صحبت کند. برای کاهش این مسئله، ارکسترатор باید بهشت در دسترس<sup>۵</sup> باشد.

مورد استفاده واقعی: Netflix Conductor یک ارکسترатор میکروسرویس است و شما می‌توانید جزئیات بیشتری در مورد طراحی ارکسترатор بخوانید.

Reliability<sup>۱</sup>

Scalability<sup>۲</sup>

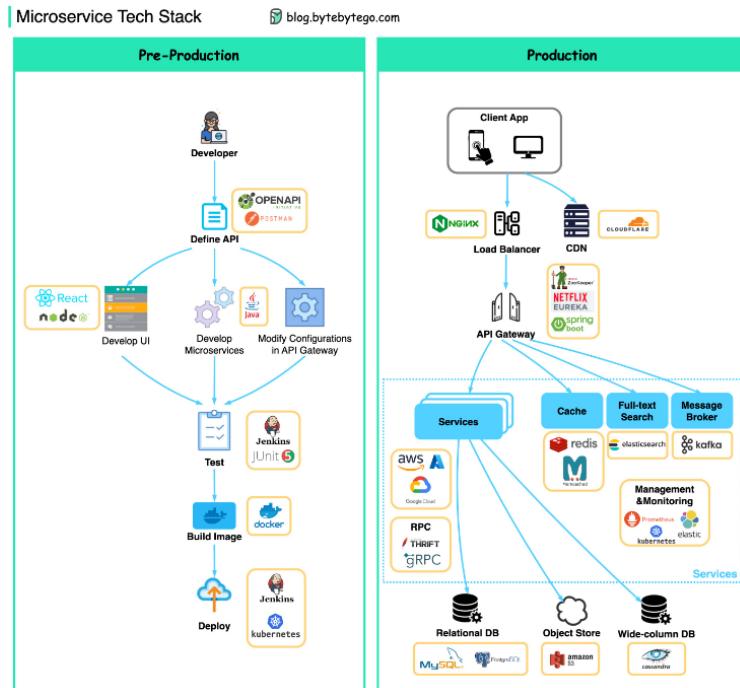
Performance<sup>۳</sup>

Single point of failure<sup>۴</sup>

highly available<sup>۵</sup>

## فناوری رایج برای میکروسرویس‌ها چیست؟

در زیر نموداری از فناوری میکروسرویس برای فاز توسعه و production را مشاهده می‌کنید.



### Pre-Production

- **تعريف API** - این کار قراردادی بین frontend و backend برقرار می‌کند. برای این کار می‌توانیم از OpenAPI یا Postman استفاده کنیم.
- **توسعه** - React یا Node.js برای توسعه frontend و جاوا/پایتون/گو برای توسعه API محبوب هستند. همچنین، باید بر اساس تعاریف API، تنظیمات API backend را تغییر دهیم.
- **یکپارچه‌سازی مستمر<sup>۱</sup>** - Jenkins و JUnit برای تست خودکار. این کد در یک تصویر Docker بسته‌بندی شده و به عنوان میکروسرویس مستقر می‌شود.

## Production

- Nginx یک انتخاب رایج برای توزیع کننده بارها است. Cloudflare یک CDN (شبکه توزیع محتوا) ارائه می‌دهد.
- API Gateway - می‌توانیم از Spring Boot برای دروازه<sup>۱</sup> و از Eureka/Zookeeper برای کشف سرویس<sup>۲</sup> استفاده کنیم.
- Microsoft AWS، Google GCP یا Azure میکروسرویس‌ها روی ابرها مستقر می‌شوند. ما گرینه‌هایی بین Redis<sup>۳</sup> که از که یک انتخاب رایج برای ذخیره‌سازی جفت کلید - مقدار است و برای کش بسیار مورد استفاده قرار می‌گیرد. همچنین Elasticsearch برای جستجوی متن کامل استفاده می‌شود.
- ارتباطات - برای برقراری ارتباط سرویس‌ها با یکدیگر، می‌توانیم از زیرساخت پیام‌رسانی Kafka یا RPC استفاده کنیم.
- پایداری داده<sup>۴</sup> - می‌توانیم از PostgreSQL یا MySQL برای پایگاه‌داده رابطه‌ای و Amazon S3 برای ذخیره‌سازی اشیا<sup>۵</sup> استفاده کنیم. همچنین در صورت نیاز، برای ذخیره‌سازی ستون‌گسترده (Wide-Column Store) Cassandra می‌توان از استفاده کرد.
- مدیریت و مانیتورینگ - برای مدیریت تعداد زیادی از میکروسرویس‌ها از ابزارهای رایج Kubernetes، Prometheus و Elastic Stack Ops شامل هستند.

Gateway<sup>۱</sup>

service discovery<sup>۲</sup>

Full-text Search<sup>۳</sup>

Persistence<sup>۴</sup>

object store<sup>۵</sup>

## چه کاری انجام می‌دهد؟ API gateway

یک ابزار مدیریت API است که بین یک سرویس client و مجموعه‌ای از سرویس‌های backend قرار می‌گیرد. این یک نقطه‌ی ورود واحد به سیستمی است که معماری داخلی سیستم را در بر می‌گیرد و یک API ارائه می‌دهد که برای هر سرویس کلاینتی مناسب‌سازی شده است. همچنین مسئولیت‌های دیگری مانند احراز هویت<sup>۱</sup>، مانیتورینگ، توزیع بار، گش، محدود کردن سرعت<sup>۲</sup>، لاغ‌گیری<sup>۳</sup> و غیره را بر عهده دارد.

### چرا به API Gateway نیاز داریم؟

جزئیات API‌هایی که توسط میکروسرویس‌ها ارائه می‌شوند، اغلب با نیازهای یک سرویس گیرنده متفاوت است. میکروسرویس‌ها معمولاً API‌های با جزئیات دقیق ارائه می‌دهند، به این معنی که سرویس گیرنده‌ها باید با چندین سرویس تعامل داشته باشند. از این‌رو، یک دروازه‌ی API می‌تواند یک نقطه‌ی ورود واحد برای همه سرویس گیرنده‌ها با برخی ویژگی‌های اضافی و مدیریت بهتر ارائه دهد.

### ویژگی‌ها

- Authorization و Authentication
- Service discovery
- Reverse Proxy
- Caching
- Security
- Retry and Circuit breaking
- Load balancing
- Logging, Tracing
- API composition

authentication<sup>۱</sup>

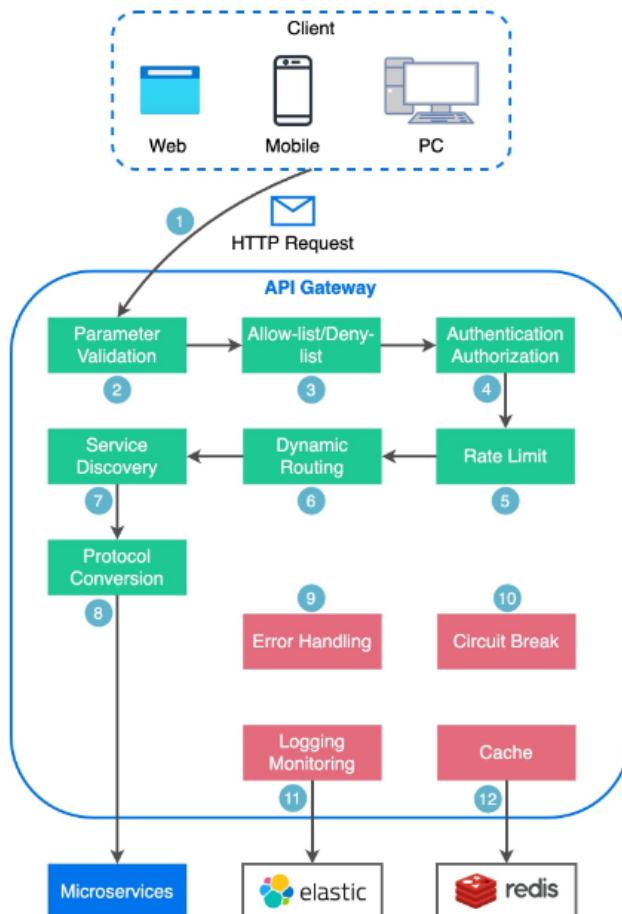
throttling<sup>۲</sup>

logging<sup>۳</sup>

- Rate limiting and throttling
- Versioning
- Routing
- blacklisting یا IP whitelisting

شکل زیر جزئیات را نشان می‌دهد.

### What does API Gateway do? blog.bytebytego.com



مرحله ۱ - کلاینت یک درخواست HTTP به دروازه API ارسال می‌کند.

مرحله ۲ - API gateway ویژگی‌های موجود در درخواست‌های HTTP را تجزیه و اعتبارسنجی می‌کند.

مرحله ۳ - API gateway بررسی های لیست مجاز/لیست سیاه را انجام می دهد.  
مرحله ۴ - API gateway برای احراز هویت<sup>۱</sup> و مجوز دادن<sup>۲</sup> با یک ارائه دهنده اعتبار

صحبت می کند.

مرحله ۵ - قوانین محدودیت نرخ<sup>۳</sup> روی درخواست ها اعمال می شود. در صورت عبور از حد مجاز، درخواست رد می شود.

مراحل ۶ و ۷ - اکنون که درخواست بررسی های اولیه را پشت سر گذاشته است، API gateway با مطابقت مسیر، سرویس مرتبط را برای مسیریابی پیدا می کند.

مرحله ۸ - درخواست را به پروتکل مناسب تبدیل می کند و آن را به میکروسرویس های backend ارسال می کند.

مراحل ۹ تا ۱۲: API gateway می تواند خطاهای را به درستی مدیریت کند و در صورت نیاز به زمان بیشتر برای بازیابی به کمک<sup>۴</sup> circuit breaker با خرابی ها مقابله کند. همچنین می تواند از ساختار ELK (Elastic-Logstash-Kibana) برای لاگ گیری و مانیتورینگ استفاده کند. همچنین گاهی اوقات داده ها را در API gateway کش می کنیم.

در زیر ابزارهای مورد استفاده در API gateway را مشاهده می کنیم:

Backend: درخواست های API ورودی را به سرویس Request Routing •

مناسب هدایت می کند.

Load Balancing: درخواست ها را در چندین سرور توزیع می کند تا اطمینان

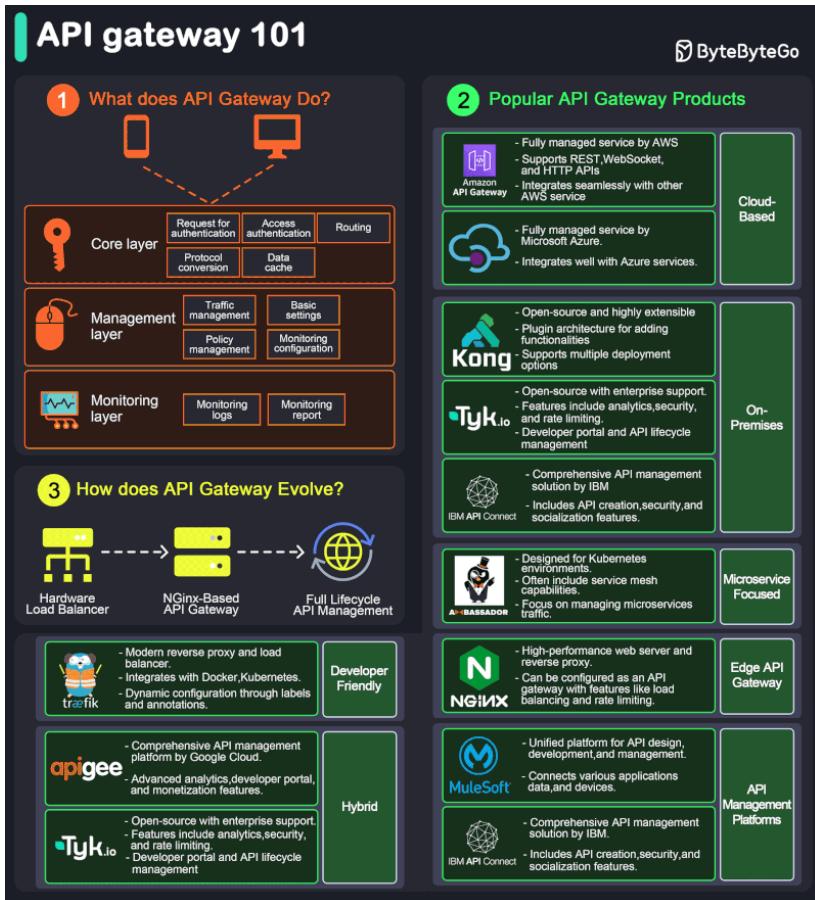
حاصل شود که هیچ یک از سرورها زیر فشار بیش از حد مجاز نیستند.

<sup>۱</sup> authentication

<sup>۲</sup> authorization

<sup>۳</sup> rate limiting

<sup>۴</sup> این روش قطع کننده نقش یک واسط را بین درخواست دهنده (client) و پاسخ دهنده ایجاد می کند که هنگامی که خرابی ها به یک آستانه مشخص رسیدند حالت قطع کننده فعال شده و دیگر درخواستی برای پاسخ دهنده (سرویس مورد نظر) ارسال نمی شود و مستقیماً خود این الگو پاسخ مورد نظر را برای کاربر ارسال می کند.



امنیت: اقدامات امنیتی مانند احراز هویت، مجوز، و رمزگذاری داده ها را اجرا می کند.

تعداد درخواست هایی را که کاربر می تواند در • **Throttling و Rate Limiting**

یک بازه زمانی معین انجام دهد را کنترل می کند.

چندین درخواست API backend را در یک درخواست • **API Composition**

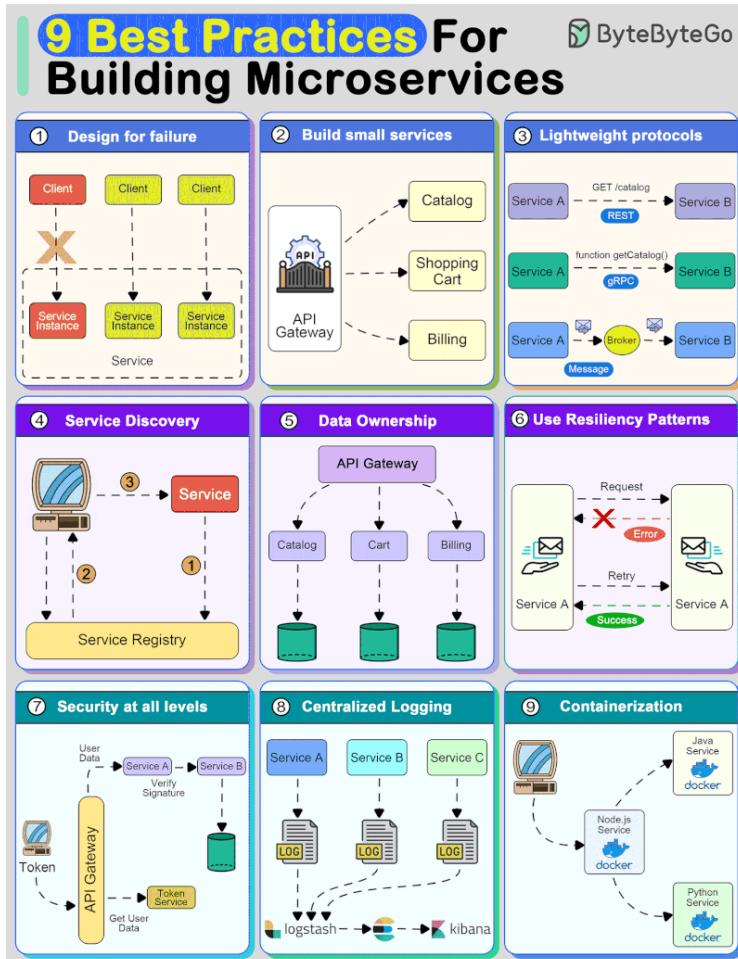
برای بهینه سازی در کارایی برنامه را به یکدیگر ترکیب می کند.

پاسخ ها را به طور موقت ذخیره می کند تا نیاز به پردازش مکرر آنها • **Caching**

را کاهش دهد.

## ۹ اصل اساسی قبل از ساختن میکروسرویس‌ها

ایجاد یک سیستم با استفاده از میکروسرویس‌ها بدون رعایت برخی اصول کلیدی، بسیار دشوار است.



در اینجا ۹ اصل اساسی که باید قبل از ساختن میکروسرویس‌ها بدانید، آورده شده است:

## ۱. طراحی برای تحمل خطأ

یک سیستم توزیع شده با میکروسرویس‌ها در معرض خطأ قرار خواهد گرفت و شما باید سیستم را به‌گونه‌ای طراحی کنید که در سطوح مختلف مانند زیرساخت، پایگاهداده و سرویس‌های مجزا، تحمل خطأ داشته باشد. برای مقابله با خطاهای از قطع کننده مدار (circuit breaker) یا روش‌های تنزل و کاهش تدریجی<sup>۱</sup> استفاده کنید.

## ۲. ساختن سرویس‌های کوچک

یک میکروسرویس نباید چندین کار را به طور همزمان انجام دهد. یک میکروسرویس خوب به‌گونه‌ای طراحی شده است که یک کار را به‌خوبی انجام دهد.

## ۳. استفاده از پروتکل‌های سبک برای ارتباط

ارتباط، هسته‌ی یک سیستم توزیع شده است. میکروسرویس‌ها باید با استفاده از پروتکل‌های سبک<sup>۲</sup> با یکدیگر ارتباط برقرار کنند. گزینه‌ها شامل REST، gRPC یا کارگزاران پیام<sup>۳</sup> هستند.

## ۴. پیاده‌سازی کشف سرویس service discovery

میکروسرویس‌ها برای برقراری ارتباط با یکدیگر نیاز به کشف همدیگر در شبکه دارند. service discovery را با استفاده از ابزارهایی مانند Eureka، Consul یا سرویس‌های Kubernetes پیاده‌سازی کنید.

<sup>۱</sup> degradation Graceful

<sup>۲</sup> lightweight

<sup>۳</sup> message brokers

## ۵. مالکیت داده<sup>۱</sup>

در میکروسرویس‌ها، داده‌ها باید توسط سرویس‌های مستقل، مالکیت و مدیریت شوند. هدف باید کاهش وابستگی بین سرویس‌ها باشد تا بتوانند به طور مستقل توسعه یابند.

## ۶. استفاده از الگوهای انعطاف‌پذیر

الگوهای انعطاف‌پذیر خاصی را برای بهبود در دسترس بودن سرویس‌ها پیاده‌سازی کنید. نمونه‌ها: خط‌مشی‌های تلاش مجدد<sup>۲</sup>، کش و محدود کردن نرخ<sup>۳</sup>.

## ۷. امنیت در تمام سطوح

در یک سیستم مبتنی بر میکروسرویس، سطح حمله بسیار گسترده است. شما باید در هر سطح از مسیر ارتباطی سرویس، موارد امنیتی را پیاده‌سازی کنید.

## ۸. لاغ مرکز<sup>۴</sup>

لاغ‌ها برای یافتن مشکلات در یک سیستم مهم هستند. با وجود سرویس‌های متعدد، استفاده از آن‌ها حیاتی می‌شوند. یک سیستم لاغ مرکز برای جمع‌آوری لاغ‌ها در تمامی سرویس‌ها راه‌اندازی کنید.

## ۹. استفاده از تکنیک‌های کانتینر سازی

برای مستقر کردن میکروسرویس‌ها به صورت ایزوله، از تکنیک‌های کانتینر سازی استفاده کنید. ابزارهایی مانند Docker و Kubernetes می‌توانند در این زمینه کمک کنند، زیرا برای ساده‌سازی مقیاس‌دهی و استقرار میکروسرویس‌ها در نظر گرفته شده‌اند.

<sup>۱</sup> Data Ownership

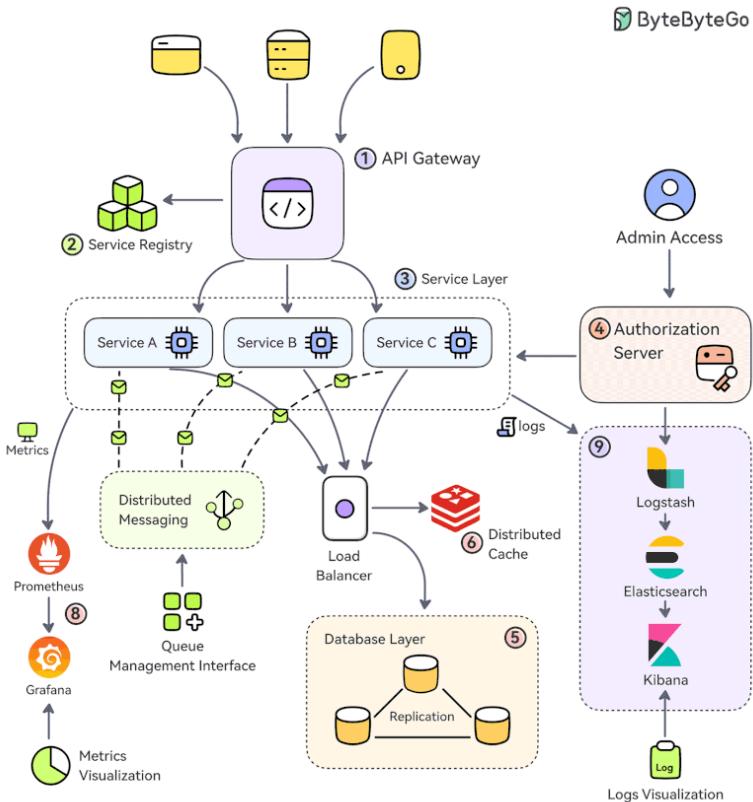
<sup>۲</sup> retry policies

<sup>۳</sup> rate limiting

<sup>۴</sup> Centralized logging

## ۹ جزء ضروری یک برنامه میکروسرویس

### 9 Essential Components of Production Microservice App



### .۱ API Gateway

این Gateway یک نقطه ورود یکپارچه برای Application های کلاینت فراهم می کند و مسئولیت هایی همچون مسیریابی، فیلتر کردن و توزیع بار را برعهده دارد.

### .۲ Service Registry

رجیستر سرویس حاوی جزئیات تمام سرویس ها است. دروازه با استفاده از سرویس Zookeeper، سایر سرویس را شناسایی می کند. ابزارهایی مانند Eureka، Consul و غیره در این مورد کاربرد دارند.

### ۳. Service Layer :

هر میکروسرویس یک عملکرد تجاری خاص را ارائه می‌دهد و می‌تواند روی چندین نمونه اجرا شود. این سرویس‌ها را می‌توان با استفاده از فریم‌ورک‌هایی مانند Spring Boot و غیره ساخت. NestJS

### ۴. سرور تأیید اعتبار (Authorization Server) :

برای ایمن‌سازی میکروسرویس‌ها و مدیریت هویت و کنترل دسترسی استفاده می‌شود. ابزارهایی مانند Okta، Azure AD، Keycloak و Keycloak می‌توانند در این زمینه کمک کنند.

### ۵. ذخیره‌سازی داده (Data Storage) :

پایگاه‌داده‌هایی مانند PostgreSQL و MySQL می‌توانند داده‌های برنامه‌ای را که توسط سرویس‌ها تولید می‌شوند را ذخیره کنند.

### ۶. کش توزیع شده (Distributed Caching) :

کش کردن رویکردهای عالی برای افزایش عملکرد برنامه است. گزینه‌ها شامل راه حل‌های کش مانند Memcached، Couchbase، Redis و غیره است.

### ۷. ارتباط ناهم‌زمان میکروسرویس‌ها (Async Microservices Communication) :

از پلتفرم‌هایی مانند Kafka و RabbitMQ برای پشتیبانی از ارتباطات ناهم‌زمان بین میکروسرویس‌ها استفاده کنید.

### ۸. مصوّر سازی متريک‌ها (Metrics Visualization) :

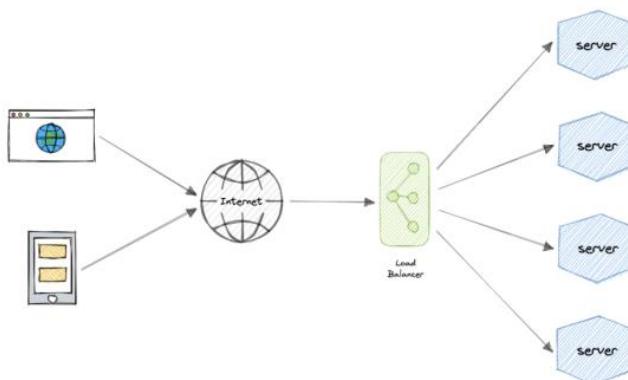
ميکروسرویس‌ها را می‌توان برای انتشار متريک‌ها به کمک Prometheus پيکربندی کرد و ابزارهایی مانند Grafana می‌توانند به نمایش گرافیکی متريک‌ها کمک کنند.

### ۹. جمع‌آوری و نمایش لاغ (Log Aggregation and Visualization) :

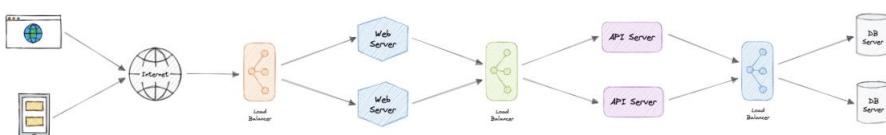
лаг‌های تولید شده توسط سرویس‌ها با استفاده از Logstash جمع‌آوری، در Kibana نمایش داده می‌شوند. Elasticsearch

## بررسی Load Balancing

توزیع بار به ما امکان می‌دهد ترافیک ورودی شبکه را بین چندین منبع توزیع کنیم تا با ارسال درخواست‌ها فقط به منابع آنلاین، دردسترس بودن و قابلیت اطمینان بالا را تضمین کنیم. این انعطاف‌پذیری را برای افزودن یا کم کردن منابع بسته به تقاضا فراهم می‌کند. برای مقیاس‌پذیری و افزونگی بیشتر، می‌توانیم در هر لایه از سیستم خود، توزیع بار را امتحان کنیم:



اما چرا؟ وبسایت‌های مدرن با ترافیک بالا باید به صدها هزار، اگر نه میلیون‌ها درخواست هم‌زمان از کاربران یا سرویس‌گیرندها پاسخ دهند. برای مقیاس‌دهی مفرونه به صرفه برای پاسخگویی به این حجم بالا، بهترین روشی رایج محاسبات مدرن به طور کلی نیازمند افزودن سرورهای بیشتر است.



یک توزیع‌کننده‌ی بار می‌تواند جلوی سرورها قرار بگیرد و درخواست‌های کاربران را در تمام سرورهایی که قادر به برآورده کردن آن درخواست‌ها هستند، به روشنی که سرعت و

استفاده از ظرفیت را به حداکثر برساند، مسیریابی کند. این کار تضمین می‌کند که هیچ سروری بیش از حد کار نکند که می‌تواند عملکرد را کاهش دهد. اگر یک سرور از کار بیفتاد، توزیع‌کننده‌ی بار ترافیک را به سرورهای آنلاین باقیمانده هدایت می‌کند. هنگامی که یک سرور جدید به گروه سرور اضافه می‌شود، توزیع‌کننده‌ی بار به طور خودکار شروع به ارسال درخواست‌ها به آن می‌کند.

### توزیع بار کاری (Workload distribution)

- این قابلیت اصلی ارائه شده توسط یک توزیع‌کننده‌ی بار است و چندین متغیر رایج دارد:
- مبتنی بر میزبان (Host-based): درخواست‌ها را بر اساس نام میزبان درخواست‌شده توزیع می‌کند.
  - مبتنی بر مسیر (Path-based): با استفاده از کل URL برای توزیع درخواست‌ها به جای صرفاً نام میزبان.
  - مبتنی بر محتوا (Content-based): محتوای پیام یک درخواست را بررسی می‌کند. این امکان توزیع بر اساس محتوایی مانند مقدار یک پارامتر را می‌دهد.

### لایه‌ها

به‌طورکلی، توزیع‌کننده‌های بار در یکی از دو سطح زیر عمل می‌کنند:

لایه شبکه (Network layer): این توزیع‌کننده‌ی باری است که در لایه انتقال شبکه کار می‌کند که به عنوان لایه ۴ نیز شناخته می‌شود. این مسیریابی را بر اساس اطلاعات شبکه مانند آدرس‌های IP انجام می‌دهد و قادر به انجام مسیریابی مبتنی بر محتوا نیست. اینها اغلب دستگاه‌های سخت‌افزاری اختصاصی هستند که می‌توانند با سرعت بالا کار کنند.

لایه Application: این توزیع‌کننده‌ی باری است که در لایه Application کار می‌کند و همچنین به عنوان لایه ۷ شناخته می‌شود. توزیع‌کننده‌های بار می‌توانند درخواست‌ها را به طور

کامل بخوانند و مسیریابی مبتنی بر محتوا را انجام دهند. این امکان مدیریت بار را بر اساس درک کامل ترافیک فراهم می‌کند.

### انواع توزیع کننده بار

بیایید به انواع مختلف توزیع کننده‌های بار نگاه کنیم:

### نرم‌افزاری

توزیع کننده‌های بار نرم‌افزاری معمولاً نسبت به نسخه‌های سخت‌افزاری آسان‌تر مستقر می‌شوند. آن‌ها مقرّون به صرفه‌تر و انعطاف‌پذیرتر هستند و در کنار محیط‌های توسعه‌ی نرم‌افزار استفاده می‌شوند. رویکرد نرم‌افزاری انعطاف‌پذیری پیکربندی توزیع کننده‌ی بار را با نیازهای خاص محیط ما به ما می‌دهد. افزایش انعطاف‌پذیری ممکن است به قیمت صرف زمان بیشتر برای راه‌اندازی توزیع کننده‌ی بار تمام شود. در مقایسه با نسخه‌های سخت‌افزاری که رویکردی بیشتر شبیه جعبهٔ بسته ارائه می‌دهند، توزیع کننده‌های نرم‌افزاری آزادی بیشتری برای انجام تغییرات و ارتقای به ما می‌دهند. توزیع کننده‌های بار نرم‌افزاری به طور گسترده مورد استفاده قرار می‌گیرند و به عنوان راه حل‌های قابل نصب که نیاز به پیکربندی و مدیریت دارند یا به عنوان یک سرویس ابری مدیریت شده در دسترس هستند.

### سخت‌افزاری

همان‌طور که از نامش پیداست، یک توزیع کننده‌ی بار سخت‌افزاری برای توزیع ترافیک شبکه و برنامه بر روی سخت‌افزار فیزیکی و درون - سازمانی (on-premises) تکیه می‌کند. این دستگاه‌ها می‌توانند حجم زیادی از ترافیک را مدیریت کنند، اما اغلب قیمت بالایی دارند و از نظر انعطاف‌پذیری نسبتاً محدود هستند. توزیع کننده‌های بار سخت‌افزاری شامل

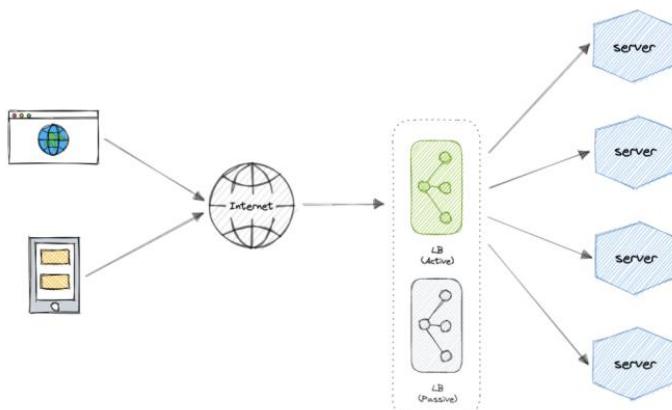
سیستم عامل اختصاصی هستند که نیاز به نگهداری و به روزرسانی با انتشار نسخه‌های جدید و وصله‌های امنیتی دارد.

## DNS

توزیع بار DNS روشی برای پیکربندی یک دامنه در سامانه نام دامنه (DNS) است به گونه‌ای که درخواست‌های کاربران به دامنه در گروهی از سرورها توزیع شود. متأسفانه، توزیع بار DNS مشکلات ذاتی ای دارد که قابلیت اطمینان و کارایی آن را محدود می‌کند. مهم‌تر از همه، DNS خرابی‌ها و قطعی‌های شبکه یا سرور و یا خطاهای را بررسی نمی‌کند. حتی اگر سرورها از کارافتاده یا در دسترس نباشند، همچنان همان مجموعه آدرس‌های IP را برای یک دامنه برمی‌گردانند.

## افزونگی توزیع کننده‌های بار

همان‌طور که احتمالاً حدس زده‌اید، خود توزیع کننده‌ی بار می‌تواند یک نقطه‌ی بالقوه خرابی کل سیستم باشد. برای غلبه بر این مشکل، می‌توان از توزیع کننده‌ی بار دوم یا بار N به صورت خوش‌های استفاده کرد.



و اگر تشخیص خرابی وجود داشته باشد و توزیع کننده بار فعال با مشکل مواجه شود، توزیع کننده بار دیگری به صورت غیرفعال می‌تواند جایگزین آن شود که باعث می‌شود سیستم ما تحمل پذیر در برابر خطأ<sup>۱</sup> شود.

### قابلیت‌های توزیع کننده‌های بار

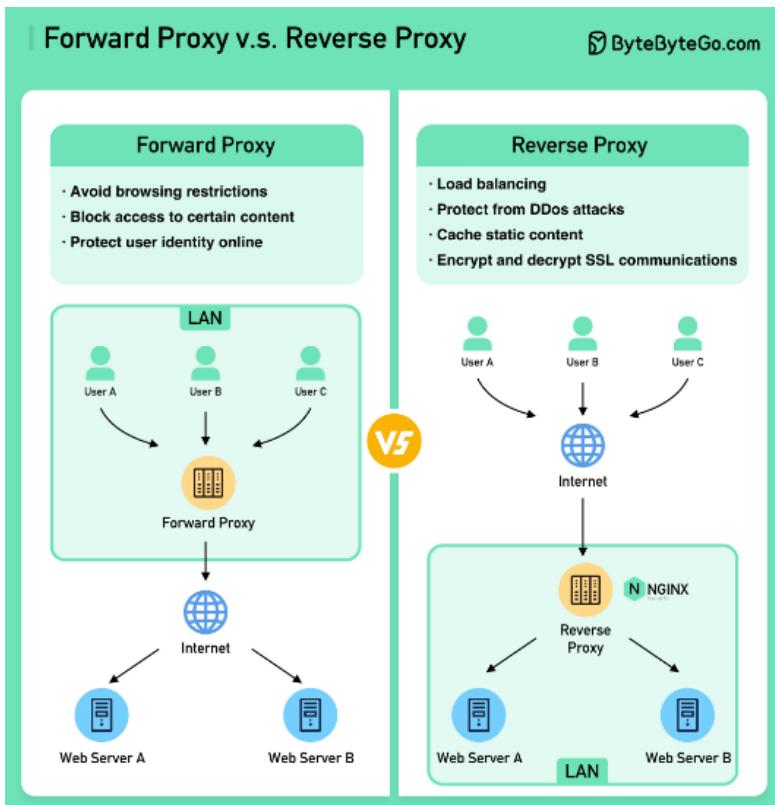
- در اینجا برخی از قابلیت‌های رایج موردنیاز توزیع کننده‌های بار آورده شده است:
  - مقیاس‌پذیری خودکار (Autoscaling): راه اندازی و خاموش کردن منابع بر اساس شرایط تقاضا.
  - جلسات چسبنده (Sticky sessions): توانایی اختصاص کاربر یا دستگاه یکسان به یک منبع یکسان برای حفظ وضعیت جلسه (session state) روی آن منبع.
  - بررسی سلامت (Healthchecks): بررسی اینکه آیا یک منبع از کارافتاده یا عملکرد ضعیفی دارد تا آن را از توزیع بار خارج کند.
  - اتصالات پایدار (Persistence connections): به یک سرور اجازه می‌دهد تا یک اتصال پایدار با یک سرویس گیرنده مانند WebSocket برقرار کند.
  - رمزنگاری (Encryption): پشتیبانی از اتصالات رمزنگاری شده مانند TLS و SSL.
  - گواهینامه‌ها (Certificates): ارائه گواهینامه به سرویس گیرنده و احراز هویت گواهینامه‌های سرویس گیرنده.
  - فشرده‌سازی (Compression): فشرده‌سازی پاسخ‌ها.
  - کش (Caching): یک توزیع کننده بار لایه کاربردی ممکن است قابلیت کش کردن پاسخ‌ها را ارائه دهد.
  - لاغ‌گیری (Logging): لاغ‌گیری متادیتای درخواست و پاسخ می‌تواند به عنوان یک ردیاب مهم یا منبعی برای داده‌های تجزیه و تحلیل عمل کند.

- ردیابی درخواست (Request tracing): اختصاص یک شناسه‌ی منحصر به فرد به هر درخواست برای اهداف لگ‌گیری، مانیتورینگ و عیب‌یابی.
- تغییر مسیر (Redirects): توانایی تغییر مسیر یک درخواست ورودی بر اساس عواملی مانند مسیر درخواستی.
- پاسخ ثابت (Fixed response): برگرداندن یک پاسخ ایستا برای یک درخواست مانند یک پیام خطای.

#### مثال

- Amazon Elastic Load Balancing
- Azure Load Balancing
- GCP Load Balancing
- DigitalOcean Load Balancer
- Nginx
- HAProxy

## چرا Nginx یک پروکسی «معکوس» نامیده می شود؟



یک پروکسی مستقیم<sup>۱</sup> سروری است که بین دستگاههای کاربر و اینترنت قرار می‌گیرد.

یک پروکسی مستقیم به طور معمول برای موارد زیر استفاده می‌شود:

- محافظت از کاربران
- دورزدن محدودیتهای مرورگر
- مسدود کردن دسترسی به محتوای خاص

forward proxy<sup>۱</sup>

پروکسی معکوس<sup>۱</sup> سروری است که درخواستی را از کاربر می‌پذیرد، درخواست را به سرورهای وب ارسال می‌کند و نتایج را به کاربر برمی‌گرداند، گویی سرور پروکسی درخواست را پردازش کرده است.

یک پروکسی معکوس برای موارد زیر مفید است:

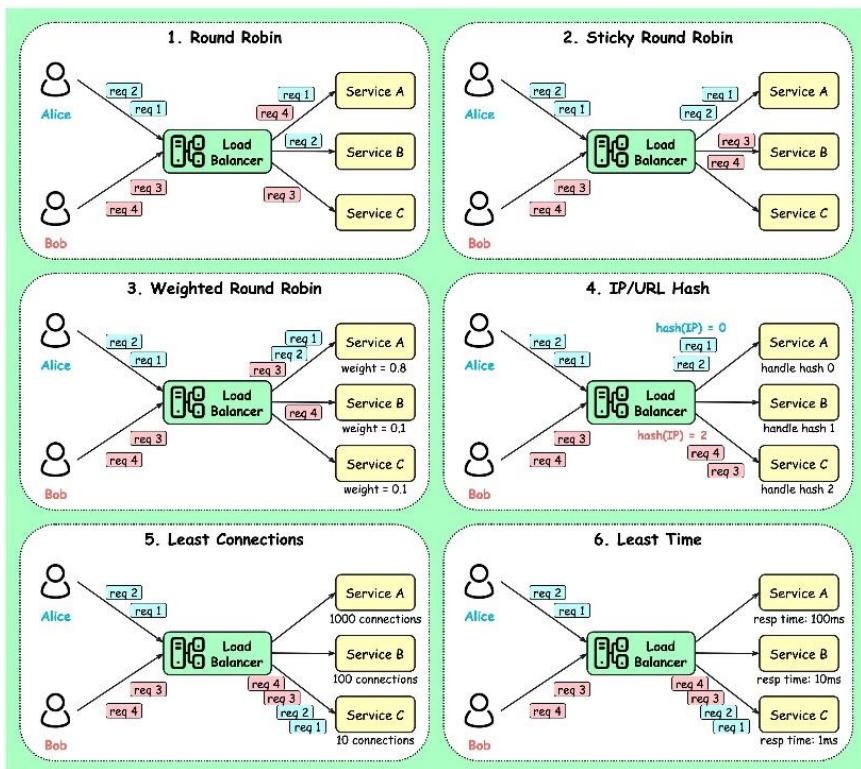
- محافظت از سرورها
- توزیع بار
- کش کردن محتوای استاتیک
- رمزگذاری و رمزگشایی ارتباطات SSL

## الگوریتم‌های رایج توزیع بار (Load Balancing)

شکل زیر ۶ الگوریتم رایج را نشان می‌دهد.

### Load Balancing Algorithms

blog.bytebytogo.com



### الگوریتم‌های ایستا (Static Algorithms)

۱. **Round Robin:** درخواست‌های کلاینت به ترتیب توالی به نمونه‌های مختلف

سرویس فرستاده می‌شوند. سرویس‌ها معمولاً Stateless در نظر گرفته می‌شوند.

**Round Robin**: این الگوریتم، بهبودیافته‌ی الگوریتم **Sticky Round Robin** .<sup>۴</sup>

است. اگر اولین درخواست Alice به سرویس A برود، درخواست‌های بعدی او نیز به سرویس A فرستاده می‌شوند.

(Weight): مدیر می‌تواند برای هر سرویس وزن (Weighted Round Robin •

تعیین کند. سرویس‌هایی با وزن بالاتر، درخواست‌های بیشتری را نسبت به سایر سرویس‌ها رسیدگی می‌کنند.

**Hash**: این الگوریتم یک تابع Hash را روی IP یا URL درخواست‌های ورودی •

اعمال می‌کند. درخواست‌ها بر اساس نتیجه تابع هش به نمونه‌های سرویس مرتب مسیریابی می‌شوند.

### الگوریتم‌های پویا (Dynamic Algorithms)

۱. کمترین اتصال (Least Connections): درخواست جدید به نمونه سرویسی که

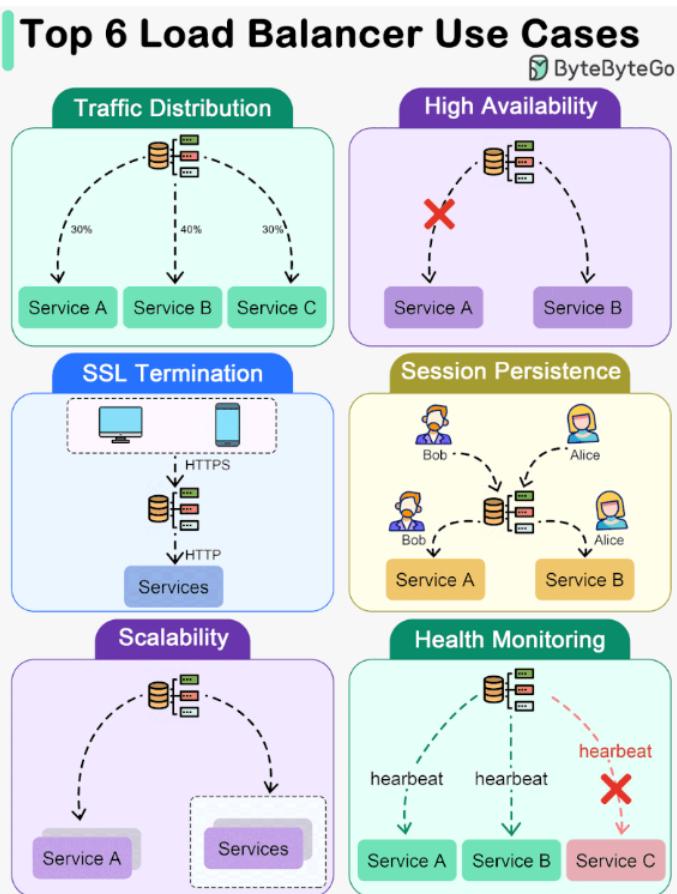
کمترین اتصال هم‌زمان را دارد، فرستاده می‌شود.

۲. کمترین زمان پاسخ (Least Response Time): درخواست جدید به نمونه

سرویسی با سریع‌ترین زمان پاسخ، فرستاده می‌شود.

## کلید استفاده از توزیع کننده بار (Load Balancer)

نمودار زیر ۶ مورد از مهم ترین سناریوهای استفاده از توزیع کننده بار را نشان می دهد.



### توزیع ترافیک

توزیع کننده های بار، ترافیک ورودی را به طور مساوی بین سرورهای مختلف توزیع می کنند و از تحت فشار قرار گرفتن بیش از حد یک سرور خاص جلوگیری می کنند. این امر به حفظ عملکرد بهینه، قابلیت ارتقا و قابلیت اطمینان برنامه ها یا وب سایت ها کمک می کند.

## در دسترس‌پذیری بالا<sup>۱</sup>

توزیع کننده‌های بار با هدایت مجدد ترافیک از سرورهای معیوب یا از کارافتاده به سرورهای سالم، در دسترس‌پذیری سیستم را افزایش می‌دهند. این کار حتی در صورت بروز مشکل در برخی سرورها، تضمین‌کننده ارائه بدون وقهه در سرویس است.

## خاتمه‌دادن به SSL

توزیع کننده‌های بار می‌توانند وظایف رمزگذاری و رمزگشایی SSL/TLS را از سرورهای backend خارج کرده و در نتیجه بار کاری آنها را کاهش داده و عملکرد کلی را بهبود بخشنند.

## پایداری Session

برای برنامه‌هایی که نیاز به حفظ Session کاربر در یک سرور خاص دارند، توزیع کننده‌های بار می‌توانند اطمینان حاصل کنند که درخواست‌های بعدی از یک کاربر به همان سرور ارسال شود.

## مقیاس‌پذیری<sup>۲</sup>

توزیع کننده‌های بار با مدیریت مؤثر ترافیک ورودی، مقیاس‌پذیری افقی را تسهیل می‌کنند. سرورهای اضافی به راحتی قابل اضافه‌شدن به مجموعه هستند و توزیع کننده بار ترافیک را در همه سرورها توزیع می‌کند.

## ناظارت بر سلامت سیستم<sup>۳</sup>

توزیع کننده‌های بار به طور مداوم سلامت و عملکرد سرورها را کنترل می‌کنند و برای حفظ عملکرد بهینه، سرورهای معیوب یا ناسالم را از مجموعه خارج می‌کنند.

---

High Availability<sup>۱</sup>

Scalability<sup>۲</sup>

Health Monitoring<sup>۳</sup>

## برآورد تقریبی در طراحی سیستم

اخیراً چند مهندس از من پرسیدند که آیا واقعاً در یک مصاحبه طراحی سیستم به برآورد تقریبی نیاز داریم یا خیر. فکر می‌کنم روشن کردن این موضوع مفید خواهد بود. برآوردها مهم هستند؛ زیرا ما به آنها برای درک مقیاس سیستم و توجیه طراحی نیاز داریم. این کار به پاسخ دادن به سؤالاتی مانند موارد زیر کمک می‌کند:

- آیا واقعاً به یک راه حل توزیع شده نیاز داریم؟
- آیا یک لایه حافظه کش ضروری است؟
- آیا باید از تکثیر داده یا شارдинگ استفاده کنیم؟

اینجا مثالی از چگونگی شکل دهی برآوردها به تصمیم طراحی آورده شده است.

یکی از پرسش‌های مصاحبه، طراحی سرویس مجاورتی<sup>۱</sup> و نحوه مقیاس دهی شاخص جغرافیایی - فضایی است که بخشی کلیدی از آن است. چند پاراگراف را که نوشتیم تا نشان دهیم چرا پرش به طراحی sharding بدون برآوردها ایده بدی است، اینجا آورده‌ایم: یکی از اشتباهات رایج در مقیاس‌بندی شاخص جغرافیایی - فضایی، پرش سریع به طرح شارдинگ بدون درنظر گرفتن اندازه واقعی داده‌های جدول است. در این مورد ما کل مجموعه داده‌ها برای جدول شاخص جغرافیایی - فضایی بزرگ نیست (شاخص Quadtree تنها ۱.۷۱ گیگابایت حافظه نیاز دارد و نیاز به ذخیره‌سازی برای شاخص geohash مشابه است). کل شاخص جغرافیایی - فضایی به راحتی می‌تواند در مجموعه کاری یک سرور پایگاه داده مدرن جای بگیرد. با این وجود، بسته به حجم خواندن، یک سرور پایگاه داده ممکن است CPU یا پهنانی باند شبکه کافی برای ارائه سرویس‌ها به همه درخواست‌های خواندن نداشته باشد. اگر این چنین باشد، لازم است بار خواندن را بین چندین سرور پایگاه داده توزیع کنیم.

دو رویکرد کلی برای توزیع بار یک سرور پایگاهداده رابطه‌ای وجود دارد. می‌توانیم تکرارهای خواندن اضافه کنیم یا پایگاهداده را تقسیم‌بندی کنیم.

بسیاری از مهندسان در طول مصاحبه‌ها دوست دارند درباره شارдинگ صحبت کنند. اما ممکن است شارдинگ برای جدول geohash گزینه خوبی نباشد. شارдинگ پیچیده است. منطق شارдинگ باید به لایه Application اضافه شود. گاهی اوقات، شارдинگ تنها گزینه است. اما در این مورد، ازانجایی که همه چیز در مجموعه کاری یک سرور پایگاهداده جای می‌گیرد، هیچ دلیل فنی قوی برای شارد کردن داده‌ها بین چندین سرور وجود ندارد. در این مورد، رویکرد بهتر داشتن یک سری نسخه تکراری خواندن برای کمک به بار خواندن است. این روش برای توسعه و نگهداری بسیار ساده‌تر است؛ بنابراین، ما توصیه می‌کنیم جدول نمایه مکانی - فضایی را از طریق نسخه‌های تکراری مقیاس‌پذیر کنید.

## بررسی صفت پیام

### صف پیام (Message Queue)

صف پیام نوعی ارتباط سرویس به سرویس است که ارتباط ناهمزمان را تسهیل می‌کند. این صفت به صورت ناهمزمان پیام‌ها را از تولیدکننده‌ها دریافت می‌کند و آن‌ها را به مصرف‌کننده‌ها ارسال می‌کند.

صف‌ها برای مدیریت مؤثر درخواست‌ها در سیستم‌های توزیع شده‌ی بزرگ مقیاس استفاده می‌شوند. در سیستم‌های کوچک با بار پردازش کم و پایگاه‌های داده‌ی کوچک، نوشتن‌ها می‌توانند به طور قابل پیش‌بینی سریعی انجام شوند. با این حال، در سیستم‌های پیچیده‌تر و بزرگ‌تر، نوشتن‌ها می‌توانند مدت زمانی تقریباً غیرقابل تبیین داشته باشند.



### نحوه‌ی کار

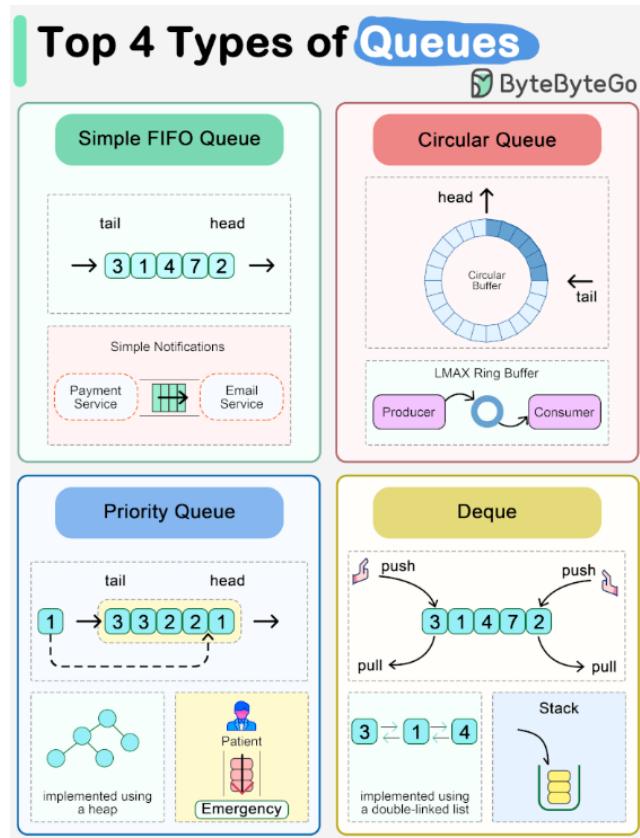
پیام‌ها تا زمانی که پردازش و حذف شوند، در صفت ذخیره می‌شوند. هر پیام تنها یک‌بار توسط یک مصرف‌کننده‌ی واحد پردازش می‌شود. نحوه‌ی کار به شرح زیر است:

یک تولیدکننده، یک کار را در صفت منتشر می‌کند، سپس کاربر را از وضعیت کار مطلع می‌کند.

یک مصرف‌کننده، کار را از صفت برمی‌دارد، آن را پردازش می‌کند، سپس علامت می‌دهد که کار تکمیل شده است.

## ۴ مورد از متدائل ترین انواع صف

صفها ساختارهای داده محبوب هستند که به طور گسترده در سیستم‌ها استفاده می‌شوند. نمودار زیر ۴ نوع مختلف از صفحه‌ای را که اغلب استفاده می‌کنیم نشان می‌دهد.



### ۱. صف ساده FIFO (اولین ورودی، اولین خروجی)

یک صف ساده از قانون FIFO (اولین ورودی، اولین خروجی) پیروی می‌کند. یک عنصر جدید در انتهای صف قرار داده می‌شود و یک عنصر از ابتدای صف حذف می‌شود. اگر بخواهیم هر زمان که پاسخ پرداخت را دریافت می‌کنیم، برای کاربران اعلان ایمیل ارسال کنیم، می‌توانیم از یک صف FIFO استفاده کنیم. ایمیل‌ها به همان ترتیب پاسخ‌های پرداخت ارسال می‌شوند.

## ۲. صف حلقوی (Circular Queue)

صف حلقوی همچنین بافر حلقوی یا رینگ بافر نامیده می‌شود. آخرین عنصر آن به اولین عنصر متصل است. درج در ابتدای صف و حذف در انتهای صف انجام می‌شود. یک پیاده‌سازی معروف، رینگ بافر با تأخیر کم LMAX نام دارد. اجزای مرتبط از طریق یک رینگ بافر با هم صحبت می‌کنند. این مورد در حافظه پیاده‌سازی شده و بسیار سریع است.

## ۳. صف اولویت‌دار (Priority Queue)

عناصر موجود در صف اولویت‌دار دارای اولویت‌های از پیش تعريف شده هستند. ما عنصری با بالاترین (یا پایین‌ترین) اولویت را از صف خارج می‌کنیم. به صورت داخلی، این مورد با استفاده از min heap یا max heap پیاده‌سازی می‌شود، جایی که عنصری با بالاترین یا پایین‌ترین اولویت در ریشه heap قرار دارد.

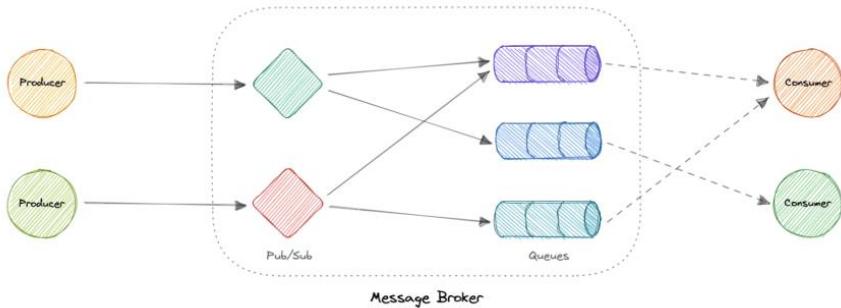
یک مثال مورد استفاده معمولی از این نوع صف به اختصاص دادن بیمارانی با بالاترین سطح وخامت به اتاق اورژانس در حالی که سایرین به اتاق‌های معمولی منتقل می‌شوند، است.

## ۴. صف دوطرفه (Deque)

صف دوطرفه همچنین Deque نامیده می‌شود. درج و حذف می‌تواند هم در ابتدای صف (head) و هم در انتهای صف (tail) اتفاق بیفتد. Deque از هر دو FIFO و LIFO (آخرین ورودی، اولین خروجی) پشتیبانی می‌کند، بنابراین می‌توانیم از آن برای پیاده‌سازی یک ساختار داده پشته (stack) استفاده کنیم.

## Message Brokers

کارگزار پیام (Message Brokers) نرم افزاری است که به برنامه ها، سیستم ها و سرویس ها امکان می دهد تا با یکدیگر ارتباط برقرار کرده و اطلاعات را ردوبل کنند. کارگزار پیام این کار را با ترجمه پیام ها بین پروتکل های پیام رسانی رسمی انجام می دهد. این کار به سرویس های وابسته به هم اجازه می دهد تا مستقیماً با یکدیگر « صحبت » کنند، حتی اگر به زبان های مختلف نوشته شده باشند یا روی پلتفرم های متفاوتی اجرا شوند.



Message Broker می توانند پیام ها را اعتبار سنجی، ذخیره، مسیر یابی و به مقصد های مناسب تحویل دهند. آن ها به عنوان واسطه ای بین سایر برنامه ها عمل می کنند و به فرستنده کان اجازه می دهند پیام ارسال کنند بدون اینکه بدانند گیرنده کان کجا هستند، فعال هستند یا خیر، یا چند تا از آن ها وجود دارند. این امر منجر به جداسازی فرایندها و سرویس ها در سیستم ها می شود.

## مدل های Message Brokers

دو الگوی اساسی توزیع پیام یا سبک های پیام رسانی ارائه می دهند: پیام رسانی نقطه به نقطه (Point-to-Point messaging): این الگوی توزیع است که در

صف های پیام با رابطه هی یک به یک بین فرستنده و گیرنده هی پیام استفاده می شود.

پیام رسانی انتشار - اشتراک (Publish-subscribe messaging): در این الگوی توزیع پیام، که اغلب با عنوان "pub/sub" از آن یاد می شود، تولید کننده هی هر پیام آن را در یک

موضوع (topic) منتشر می‌کند و مصرف‌کنندگان پیام متعدد در موضوعاتی که می‌خواهند پیام‌ها را از آن‌ها دریافت کنند، مشترک می‌شوند.

### Event streaming در مقابل Message brokers

Message Broker می‌توانند از دو یا چند الگوی پیام‌رسانی، از جمله صفاتی پیام و pub/sub پشتیبانی کنند، در حالی که پلتفرم‌های جریان رویداد فقط الگوهای توزیع به سبک pub/sub را ارائه می‌دهند. پلتفرم‌های جریان رویداد که برای استفاده با حجم زیادی از پیام‌ها طراحی شده‌اند، به راحتی مقیاس‌پذیر هستند. آن‌ها قادرند جریان‌هایی از رکوردها را در دسته‌هایی به نام موضوع (topic) مرتب کرده و برای مدت زمان معینی ذخیره کنند. با این حال، برخلاف Message Broker، پلتفرم‌های Event streaming نمی‌توانند تضمین تحويل پیام یا ردیابی اینکه کدام مصرف‌کنندگان پیام‌ها را دریافت کرده‌اند را ارائه دهنند. پلتفرم‌های جریان رویداد مقیاس‌پذیری بیشتری نسبت به کارگزاران پیام ارائه می‌دهند، اما ویژگی‌های کمتری برای اطمینان از تحمل خط‌الرأس مانند ارسال مجدد پیام‌ها و همچنین قابلیت‌های محدود‌تر مسیریابی و صفت‌بندی پیام دارند.

### Enterprise Service Bus در مقابل Message Brokers

زیرساخت ESB (Enterprise Service Bus) پیچیده است و می‌تواند ادغام و نگهداری آن پرهزینه باشد. عیب‌یابی آن‌ها در صورت بروز مشکل در محیط‌های production دشوار است، مقیاس‌پذیری آن‌ها آسان نیست و به روزرسانی آن‌ها خسته‌کننده است. در حالی که کارگزاران پیام جایگزینی سبک‌وزن برای ESB‌ها هستند که عملکرد مشابهی را ارائه می‌دهند، یعنی مکانیزمی برای ارتباط بین سرویس‌ها، با هزینه‌ی کمتری. آن‌ها برای استفاده در معماری‌های میکروسرویس که با کاهش محبوبیت ESB‌ها رواج بیشتری پیدا کرده‌اند، بسیار مناسب هستند.

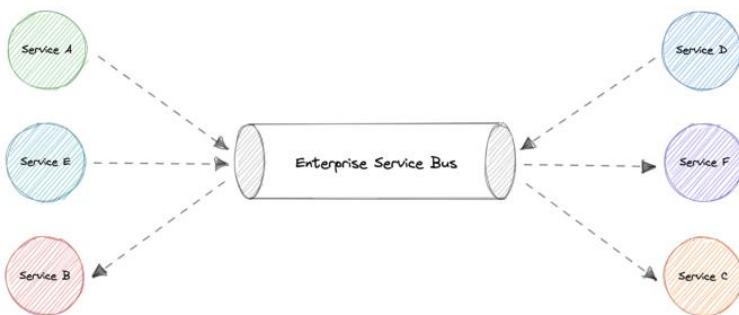
نمونه‌ها

در اینجا برخی از کارگزاران پیام رایج آورده شده است:

- NATS
- Apache Kafka
- RabbitMQ
- ActiveMQ

## Enterprise Service Bus (ESB)

یک الگوی معماری است که در آن یک مؤلفه‌ی نرم‌افزاری مرکزی، یکپارچه‌سازی بین برنامه‌ها را انجام می‌دهد. این الگو تبدیل مدل‌های داده، مدیریت اتصال، مسیریابی پیام، تبدیل پروتکل‌های ارتباطی و به طور بالقوه مدیریت ترکیب درخواست‌های متعدد را انجام می‌دهد. ESB می‌تواند این یکپارچه‌سازی‌ها و تبدیل‌ها را به عنوان یک رابطه‌ی سرویس برای استفاده‌ی مجدد توسط برنامه‌های جدید در دسترس قرار دهد.



## مزایای ESB

به طور تئوری، یک ESB مرکزی، پتانسیل استانداردسازی و ساده‌سازی چشمگیر ارتباط، پیام‌سازی و یکپارچه‌سازی بین سرویس‌ها در سراسر سازمان را ارائه می‌دهد. در اینجا برخی از مزایای استفاده از ESB آورده شده است:

- بهبود بهرهوری توسعهدهنده: به توسعهدهنگان این امکان را می‌دهد تا فناوری‌های جدید را در یک بخش از برنامه ادغام کنند بدون اینکه به سایر بخش‌های برنامه دست بزنند.
- مقیاسپذیری ساده‌تر و مقرون به صرفه‌تر: اجزا می‌توانند به طور مستقل از سایر اجزا scale شوند.
- انعطاف‌پذیری بیشتر: خرابی یک جزء بر سایر اجزا تأثیر نمی‌گذارد و هر میکروسرویس می‌تواند الزامات در دسترس بودن خاص خود را رعایت کند بدون اینکه در دسترس بودن سایر اجزا در سیستم به خطر بیفتد.

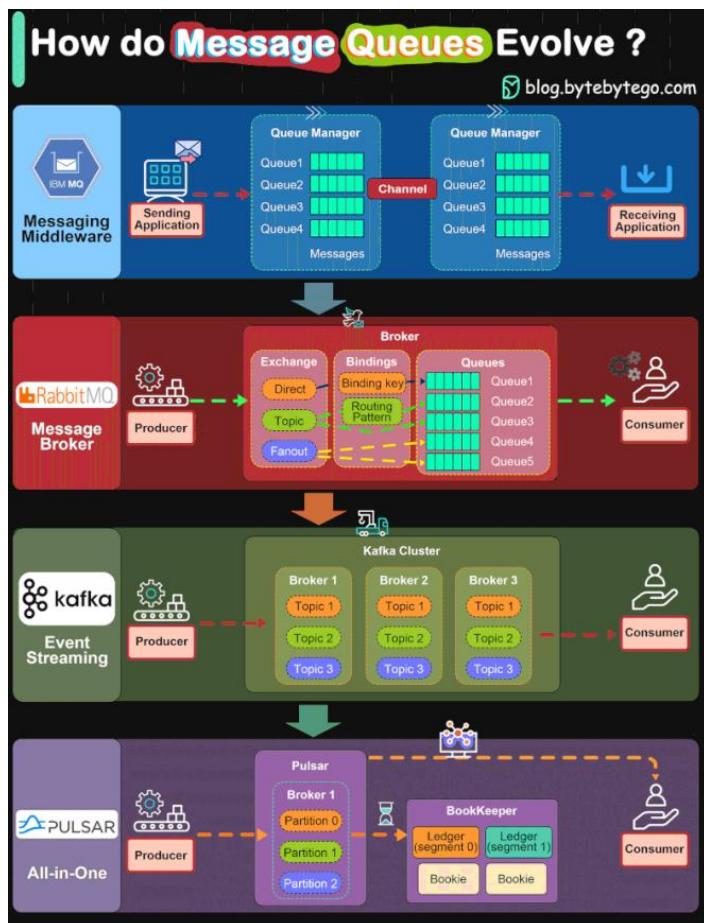
### معایب ESB

در حالی که ESB‌ها با موفقیت در بسیاری از سازمان‌ها مستقر شدند، در بسیاری از سازمان‌های دیگر، ESB به عنوان یک گلوگاه<sup>۱</sup> شناخته شد. در اینجا برخی از معایب استفاده از ESB آورده شده است:

- ایجاد تغییرات یا بهبودهایی در یکپارچه‌سازی می‌تواند یکپارچه‌سازی‌های دیگری را که از همان یکپارچه‌سازی استفاده می‌کنند، بی‌ثبات کند.
- یک نقطه‌ی بالقوه‌ی خرابی می‌تواند همه‌ی ارتباطات را مختل کند.
- به روزرسانی‌های ESB اغلب بر یکپارچه‌سازی‌های موجود تأثیر می‌گذارد، بنابراین برای انجام هر به روزرسانی آزمایش قابل توجهی موردنیاز است.
- ESB به صورت مرکزی مدیریت می‌شود که همکاری بین تیم‌ها را به چالش می‌کشد.
- پیچیدگی بالای پیکربندی و نگهداری.

## تحول معماری صفت پیام: از Pulsar تا RabbitMQ به IBM MQ

صف پیام<sup>۱</sup> ابزاری است که به برنامه‌ها امکان برقراری ارتباط با یکدیگر از طریق تبادل پیام‌های ناهم‌زمان<sup>۲</sup> را می‌دهد. در طول زمان، معماری‌های صفت پیام برای برآورده کردن نیازهای فراینده سیستم‌های توزیع شده تکامل یافته است. بیایید نگاهی به برخی از بازیگران اصلی در این تحول بیندازیم:



message queue<sup>۱</sup>

Asynchronous<sup>۲</sup>

**:IBM MQ**

محصول IBM MQ در سال ۱۹۹۳ با نام اولیه MQSeries راهاندازی شد و سپس در سال ۲۰۰۲ به WebSphere MQ تغییر نام یافت. در نهایت، در سال ۲۰۱۴ با نام نهایی IBM MQ شناخته شد. این محصول بسیار موفق است و به طور گستردگی در بخش مالی مورد استفاده قرار می‌گیرد. درآمد آن در سال ۲۰۲۰ همچنان به ۱ میلیارد دلار رسید.

**:RabbitMQ**

معماری IBM MQ با RabbitMQ متفاوت است و شباهت بیشتری به مفاهیم Kafka دارد. در این معماری، تولیدکننده پیامی را با نوع مبادله مشخص (distributed topic) یا (direct) یا به یک مبادله منتشر می‌کند. سپس مبادله پیام را بر اساس ویژگی‌های مختلف پیام و نوع مبادله، به صفوّف مناسب هدایت می‌کند. مصرف‌کنندگان نیز بر اساس نیاز خود پیام‌ها را از صفوّف دریافت می‌کنند.

**:Kafka**

در اوایل سال ۲۰۱۱، شرکت لینکدین پلتفرم توزیع شده جریان رویداد<sup>۱</sup> به نام Kafka را به صورت متن‌باز در اختیار عموم قرارداد. این نام برگرفته از نام فرانتس کافکا، نویسنده مشهور است. همان‌طور که از نام آن پیداست، Kafka بر روی نوشتن داده‌ها بهینه‌سازی شده است. این پلتفرم با توان عملیاتی بالا و تأخیر کم، بستر مناسبی برای مدیریت فیدهای داده بلاذرنگ<sup>۲</sup> فراهم می‌کند. همچنین یک log رویداد یکپارچه را برای فعل کردن جریان رویداد ارائه می‌دهد و به طور گستردگی در شرکت‌های اینترنتی مورد استفاده قرار می‌گیرد. Kafka مفاهیمی مانند تولیدکننده (producer)، کارگزار (broker)، موضوع (topic)، پارسیشن

distributed event streaming platform<sup>۱</sup>real-time<sup>۲</sup>

(partition) و مصرف‌کننده (consumer) را تعریف می‌کند. سادگی و تحمل خطای آن باعث شده است تا جایگزین محصولات قبلی مانند صفحه‌های پیام مبتنی بر AMQP شود.

## Pulsar

پلتفرم Pulsar که در اصل توسط یاهو توسعه یافته است، یک پلتفرم یکپارچه پیام‌رسانی و استریم است. در مقایسه با Kafka ویژگی‌های مفید بسیاری را از سایر محصولات ترکیب می‌کند و از قابلیت‌های گسترده‌ای پشتیبانی می‌کند. همچنین، معماری Pulsar بیشتر مبتنی بر محیط cloud-native است و از مقیاس‌بندی خوبه<sup>۱</sup> و migration partition و غیره پشتیبانی بهتری ارائه می‌دهد. معماری Pulsar دارای دو لایه است: لایه ارائه‌دهنده سرویس<sup>۲</sup> و لایه پایداری<sup>۳</sup> و ثبات Pulsar به طور پیش‌فرض از ذخیره‌سازی چندلایه پشتیبانی می‌کند، جایی که می‌توانیم از ذخیره‌سازی اشیاء<sup>۴</sup> ارزان‌تر مانند AWS S3 برای نگهداری پیام‌ها در درازمدت استفاده کنیم.

## حداکثر یکبار، حداقل یکبار و دقیقاً یکبار

در معماری مدرن، سیستم‌ها به بلوک‌های ساختمانی کوچک و مستقل با رابطه‌های تعریف شده بین آنها تقسیم می‌شوند. صفحه‌ای پیام ارتباط و هماهنگی را برای این بلوک‌های

cluster scaling<sup>۱</sup>

serving layer<sup>۲</sup>

persistent layer<sup>۳</sup>

object storage<sup>۴</sup>

ساختمانی فراهم می‌کنند. امروز در مورد معانی مختلف تحویل دهی<sup>۱</sup> پیام: حداکثر یکبار، حداقل یکبار و دقیقاً یکبار بحث خواهیم کرد.

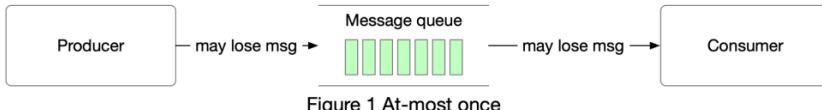


Figure 1 At-most once

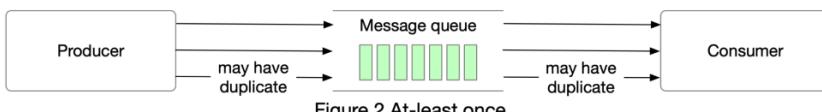


Figure 2 At-least once

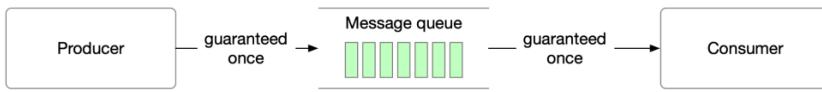


Figure 3 Exactly-once

### حداکثر یکبار<sup>۲</sup>

همان‌طور که از نام آن پیداست، حداکثر یکبار به این معنی است که پیام بیش از یکبار تحویل داده نخواهد شد. ممکن است پیام‌ها گم شوند؛ اما دوباره ارسال نمی‌شوند. این چگونگی کار تحویل حداکثر یکبار در سطح بالا است.

**موارد استفاده:** برای موارد استفاده‌ای مانند متريک‌های نظارتی که از دست‌رفتن مقدار کمی از داده‌ها قابل قبول است اين روش مناسب است.

### حداقل یکبار<sup>۳</sup>

delivery<sup>۱</sup>

At-most once<sup>۲</sup>

At-least once<sup>۳</sup>

با این معنای تحویل داده، قابل قبول است که پیام بیش از یکبار تحویل داده شود، اما هیچ پیامی نباید از دست برود.

موارد استفاده: با حداقل یکبار، پیام‌ها گم نمی‌شوند؛ اما ممکن است همان پیام چندین بار تحویل داده شود. اگرچه از دیدگاه کاربر ایده‌آل نیست، اما معنای تحویل حداقل یکبار معمولاً برای موارد استفاده‌ای که تکرار داده مشکل بزرگی نیست یا امکان حذف تکرار در سمت مصرف‌کننده وجود دارد، کافی است. به عنوان مثال، با یک کلید منحصر به فرد در هر پیام، می‌توان از نوشتن داده‌های تکراری در پایگاه داده جلوگیری کرد.

### دقیقاً یکبار<sup>۱</sup>

دقیقاً یکبار دشوارترین معنای تحویل دهی برای پیاده‌سازی است. برای کاربران بسیار مورد پسند است، اما هزینه زیادی برای عملکرد و پیچیدگی سیستم دارد.

موارد استفاده: موارد استفاده مرتبط با امور مالی (پرداخت، معامله، حسابداری و غیره). دقیقاً یکبار هنگامی که تکرار قابل قبول نیست و سرویس پایین‌دست یا شخص ثالث از idempotency پشتیبانی نمی‌کند، بسیار مهم است.

**صفهای نامه‌های مرده (Dead-letter Queues)**

صف نامه‌های مرده، صفتی است که صفحه‌ای دیگر می‌توانند پیام‌هایی را که نمی‌توانند با موفقیت پردازش شوند به آن ارسال کنند. این کار بررسی بیشتر این پیام‌ها را بدون مسدود کردن پردازش صفحه یا صرف چرخه‌ی CPU روی پیامی که ممکن است هرگز به طور موفقیت‌آمیزی مصرف نشود را آسان می‌کند.

### ترتیب (Ordering)

<sup>۱</sup>Exactly once

اکثر صفحه‌ای پیام، ترتیب با «بهترین تلاش<sup>۱</sup>» را ارائه می‌دهند که تضمین می‌کند پیام‌ها عموماً به همان ترتیبی که ارسال می‌شوند، تحویل داده شوند و یک پیام حداقل یکبار تحویل داده شود.

### (Poison-pill Messages) پیام‌های قرص سمی

قرص‌های سمی پیام‌های ویژه‌ای هستند که می‌توانند دریافت شوند، اما قابل پردازش نیستند. آن‌ها مکانیزمی هستند که برای سیگنال دادن به یک مصرف‌کننده برای پایان دادن به کارش استفاده می‌شوند تا دیگر منتظر ورودی‌های جدید نباشد و شبیه بستن یک سوکت در مدل سرویس کلاینت/سرور هستند.

### امنیت

صفهای پیام برنامه‌هایی را که سعی می‌کنند به صفت دسترسی پیدا کنند، احراز هویت می‌کنند. این کار به ما امکان می‌دهد پیام‌ها را هم روی شبکه و هم در خود صفت رمزگاری کنیم.

### Task Queues

صفهای task، task و داده‌های مرتبط با آنها را دریافت می‌کنند، آنها را اجرا می‌کنند و سپس نتایجشان را تحویل می‌دهند. آنها می‌توانند از زمان‌بندی پشتیبانی کنند و برای اجرای کارهای محاسباتی فشرده در پس‌زمینه استفاده شوند.

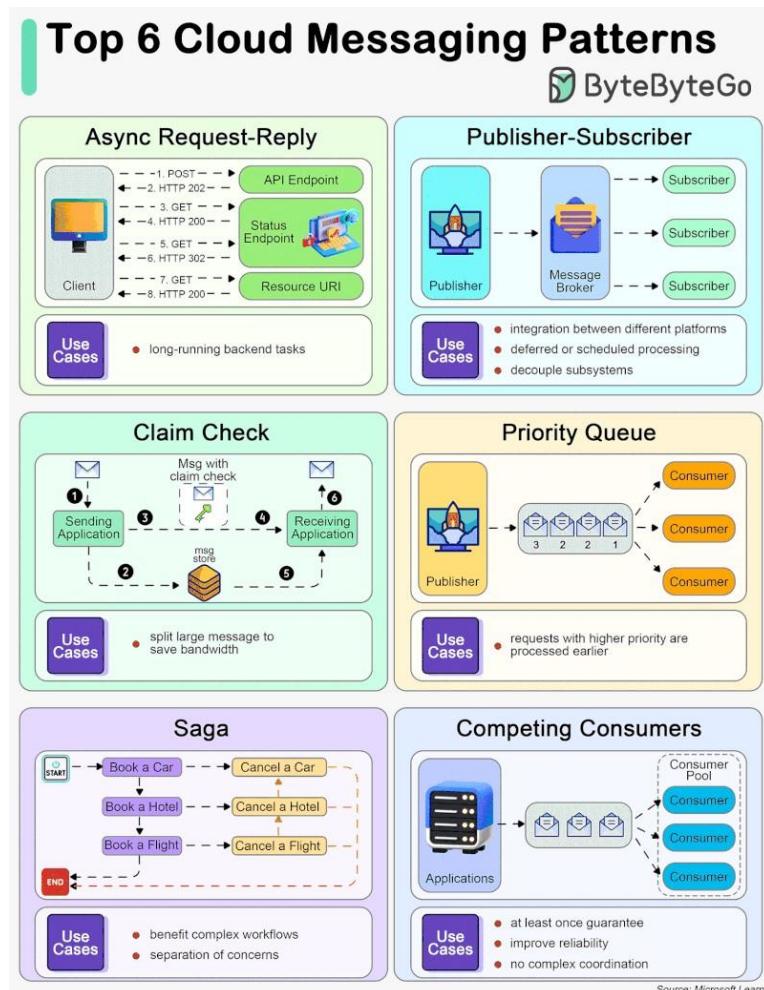
### (Backpressure) فشار معکوس

اگر صفحه شروع به رشد قابل توجهی کنند، اندازه صفت می‌تواند بزرگ‌تر از حافظه شود و در نتیجه باعث خطاهاي حافظه‌ی کش (cache misses)، خواندن دیسک و حتی عملکرد

کندر شود. فشار معکوس با محدود کردن اندازه صفت می‌تواند به این موضوع کمک کند و در نتیجه نرخ توان عملیاتی بالا و زمان پاسخگویی خوبی را برای کارهایی که قبلاً در صفت هستند، حفظ کند. هنگامی که صفت پر می‌شود، سرویس‌گیرنده‌ها یک کد وضعیت server busy یا HTTP 503 دریافت می‌کنند تا بعداً دوباره امتحان کنند. سرویس‌گیرنده‌ها می‌توانند بعداً دوباره درخواست را امتحان کنند، شاید با یک استراتژی مشابه exponential backoff.

## ۶ الگوی برتر پیام رسانی ابری

سرویس‌ها چگونه با یکدیگر ارتباط برقرار می‌کنند؟ نمودار زیر ۶ الگوی پیام رسانی ابری را نشان می‌دهد.



### :Asynchronous Request-Reply

این الگو به منظور ارائه کنترل و صحت عملکرد برای تَسک‌هایی که مدت زمان طولانی در backend در حال اجرا هستند، طراحی شده است. این الگو پردازش را از کلاینت‌های فرانت‌اند جدا می‌کند.

در نمودار بالا، کلاینت یک فراخوان هم‌زمان به API می‌زند که باعث راه‌اندازی یک عملیات طولانی مدت در بک‌اند می‌شود. API کد وضعیت 202 HTTP (پذیرفته شده) را بر می‌گرداند که نشان می‌دهد درخواست برای پردازش دریافت شده است.

### :Publisher-Subscriber

هدف این الگو جدایکردن فرستنده‌ها از گیرنده‌ها و جلوگیری از مسدود یا بلاک شدن فرستنده برای انتظار پاسخ است.

### :Claim Check

این الگو مشکل انتقال پیام‌های بزرگ را حل می‌کند. این الگو کل payload پیام را در یک پایگاه‌داده ذخیره می‌کند و تنها مرجع پیام را که بعداً برای بازیابی payload از پایگاه‌داده استفاده می‌شود را ارسال می‌کند.

### :Priority Queue

این الگو درخواست‌های ارسال شده به سرویس‌ها را اولویت‌بندی می‌کند تا درخواست‌های با اولویت بالاتر سریع‌تر از درخواست‌های با اولویت پایین‌تر دریافت و پردازش شوند.

### :Saga

Saga برای مدیریت یکپارچگی داده<sup>۱</sup> در سراسر سرویس‌های مختلف در سیستم‌های توزیع‌شده، به‌ویژه در معماری‌های میکروسرویس که در آن هر سرویس پایگاه‌داده خود را مدیریت می‌کند، استفاده می‌شود.

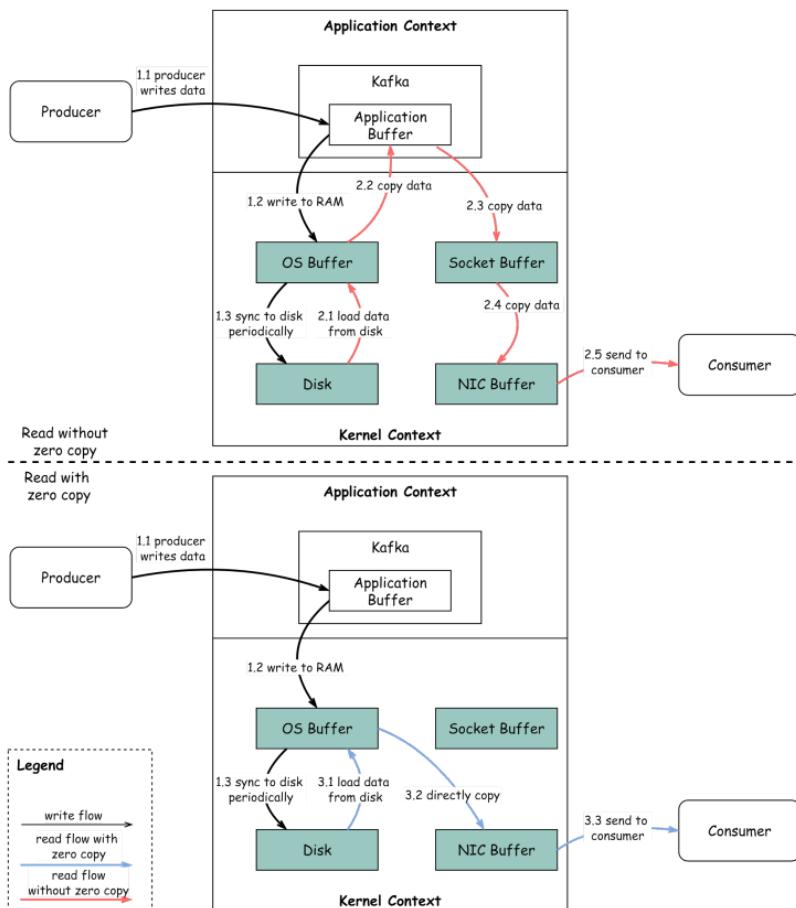
الگوی Saga چالش حفظ انسجام و یکپارچگی داده را بدون نیاز به تراکنش‌های توزیع شده که مقیاس‌بندی آن‌ها دشوار است و می‌تواند بر عملکرد سیستم تأثیر منفی بگذارد را برطرف می‌کند.

### :**(Competing Consumers) مصرف‌کنندگان رقیب**

این الگو به چندین مصرف‌کننده هم‌زمان امکان پردازش پیام‌های دریافتی در یک کانال پیام‌رسانی را می‌دهد. در واقع به کمک این الگو دیگر نیازی به پیکربندی هماهنگی پیچیده بین مصرف‌کنندگان نیست با این حال، این الگو نمی‌تواند ترتیب پیام‌ها را تضمین کند.

## چرا Kafka سریع است

کافکا با استفاده از I/O ترتیبی و اصل کپی صفر<sup>۱</sup>، به برتری انتقال پیام با تأخیر زمانی کم دست پیدا می‌کند. همین تکنیک‌ها به طور معمول در بسیاری از پلتفرم‌های پیام‌رسانی/جریان اطلاعاتی دیگر نیز استفاده می‌شوند. نمودار زیر نحوه انتقال داده بین تولیدکننده و مصرف‌کننده و مفهوم کپی صفر را نشان می‌دهد.



این نمودار نشان می‌دهد که چگونه داده‌ها بین تولیدکننده و مصرفکننده منتقل می‌شود و کپی صفر به چه معناست.

مراحل بدون کپی صفر (مراحل ۲.۱ تا ۲.۵):

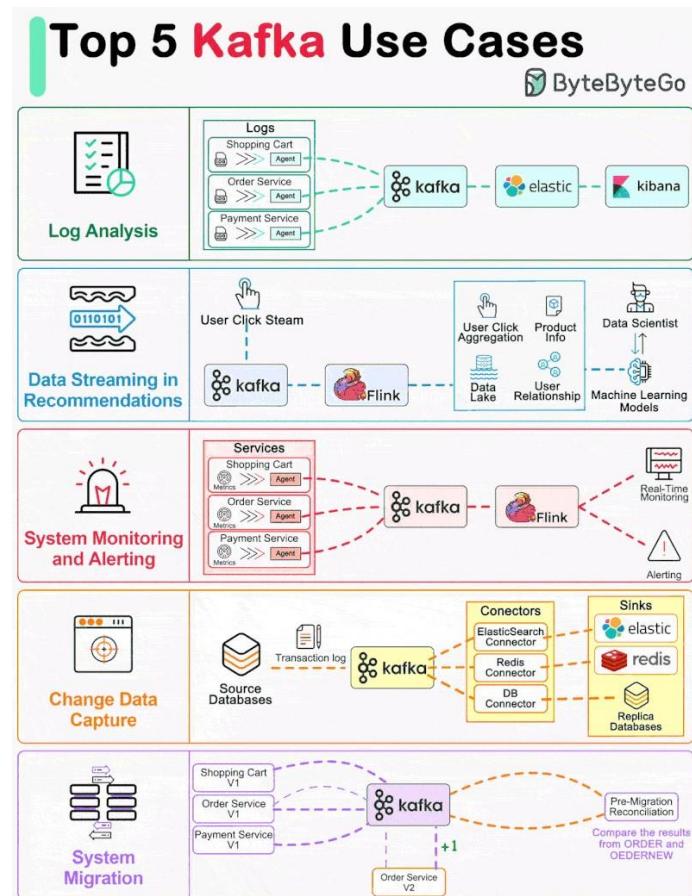
- ۱.۱ - ۱.۳: تولیدکننده داده‌ها را روی دیسک می‌نویسد.
  - ۲: مصرفکننده داده‌ها را بدون کپی صفر می‌خواند.
  - ۲.۱: داده‌ها از دیسک به کش (cache) سیستم‌عامل بارگذاری می‌شوند.
  - ۲.۲: داده‌ها از کش سیستم‌عامل به Application کافکا کپی می‌گردند.
  - ۲.۳: Application کافکا داده‌ها را به socket buffer کپی می‌کند.
  - ۲.۴: داده‌ها از بافر سوکت به کارت شبکه کپی می‌شوند.
  - ۲.۵: کارت شبکه داده‌ها را به سمت مصرفکننده ارسال می‌کند.
- مراحل با کپی صفر (مراحل ۳.۱ تا ۳.۳):
- ۳: مصرفکننده داده‌ها را با کپی صفر می‌خواند.
  - ۳.۱: داده‌ها از دیسک به کش سیستم‌عامل بارگذاری می‌شوند.
  - ۳.۲: کش سیستم‌عامل با استفاده از دستور `sendfile` مستقیماً داده‌ها را به کارت شبکه کپی می‌کند.
  - ۳.۳: کارت شبکه داده‌ها را به سمت مصرفکننده ارسال می‌کند.
- کپی صفر میانبری است برای صرفه‌جویی در کپی‌های متعدد داده‌ها بین محیط برنامه<sup>۱</sup> و محیط کرنل<sup>۲</sup> صورت می‌پذیرد.

<sup>۱</sup> application context

<sup>۲</sup> kernel context

## ۵ مورد از کاربردهای برتر Kafka

Kafka در اصل برای پردازش حجم عظیمی از لگ‌ها (log) ساخته شده است. این سیستم پیام‌ها را تا زمان انقضای نگه می‌دارد و به مصرف‌کنندگان<sup>۱</sup> اجازه می‌دهد تا پیام‌ها را با سرعت ذاتی خودشان دریافت کنند.



حالا باید به محبوب‌ترین موارد استفاده از Kafka نگاهی بیندازیم:

consumers<sup>۱</sup>

**پردازش و تحلیل لاغ Kafka** : به دلیل توانایی مقیاس‌پذیری بالا و قابلیت اطمینان، به طور گسترده‌ای برای تجزیه و تحلیل لاغ‌ها از منابع مختلف مانند سرورها، اپلیکیشن‌ها و سرویس‌های میکروسرویس استفاده می‌شود.

**Data streaming in recommendations** : در سیستم‌های توصیه‌گر، Kafka برای جریان داده‌های کاربر به صورت بلاذرنگ استفاده می‌شود. این داده‌ها می‌توانند شامل بازدیدهای صفحه، رتبه‌بندی‌ها و سایر تعاملات کاربران باشند. این داده‌ها سپس برای ایجاد توصیه‌های شخصی‌سازی شده برای هر کاربر مورد استفاده قرار می‌گیرند.

**سیستم نظارت و هشدار<sup>۱</sup>** : از Kafka برای جمع‌آوری و انتقال داده‌های نظارت از زیرساخت‌ها یا اپلیکیشن‌های مختلف استفاده می‌شود. این داده‌ها می‌توانند برای تشخیص مشکلات، شناسایی الگوهای غیرمنتظره و ارسال هشدار به تیم عملیات<sup>۲</sup> مورد استفاده قرار گیرند.

CDC<sup>۳</sup> : با استفاده از Kafka، می‌توان تغییرات در پایگاه‌داده‌ها را به صورت بلاذرنگ ضبط و به سیستم‌های پایین‌دستی منتقل کرد. این به این معنی است که سایر اپلیکیشن‌ها می‌توانند همیشه از آخرین وضعیت داده‌ها مطلع باشند که این امر برای همگام‌سازی داده‌ها و به روزرسانی سیستم‌های وابسته بسیار مفید است.

**System migration** : Kafka می‌تواند برای انتقال اینمن و قابل اعتماد داده‌ها بین سیستم‌های قدیمی و جدید در طول فرایند migration سیستم استفاده شود. این کار با جریان داده‌ها از Kafka و سپس مصرف داده‌ها توسط سیستم جدید انجام می‌شود. سیستم قدیمی به Kafka و قابلیت مقیاس‌پذیری بالا و تحمل خطا، فرایند migration را کارآمدتر می‌کند.

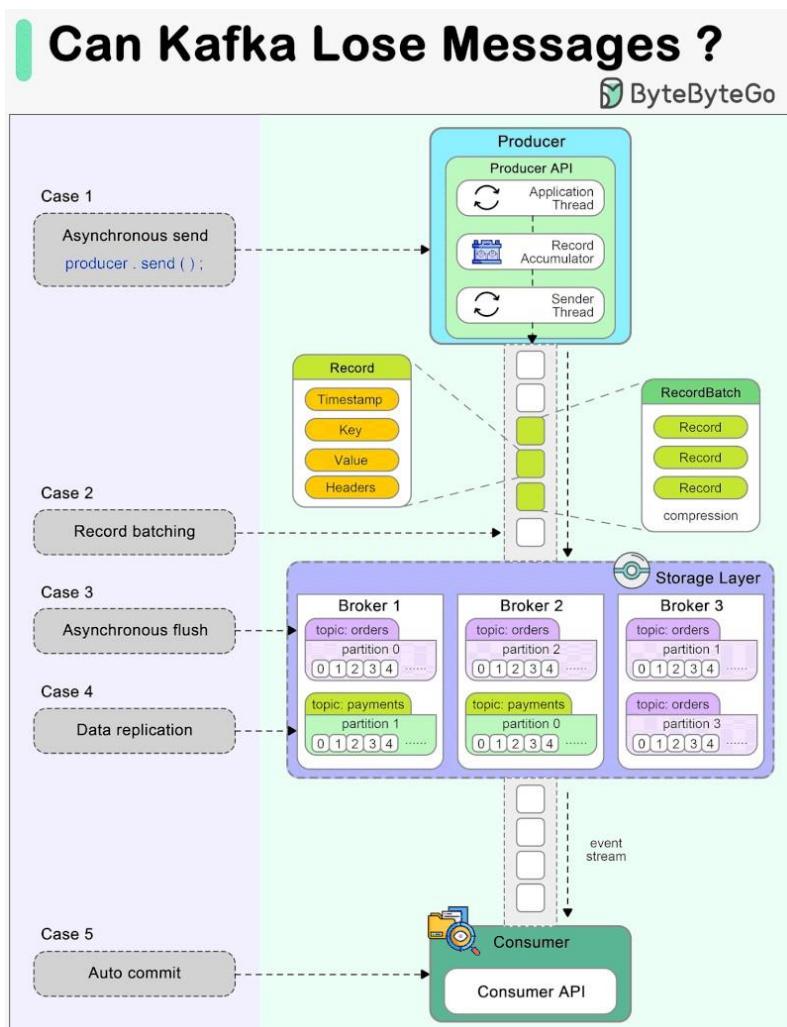
¹ System monitoring and alerting

² operations team

³ Change data capture

## آیا Kafka می‌تواند پیام‌ها را گم کند؟

یکی از مهم‌ترین جنبه‌های ساخت سیستم‌های قابل اعتماد، مدیریت خطأ است. در این متن موضوع مهمی را بررسی می‌کنیم: آیا Kafka به طور کلی از گم شدن پیام‌ها جلوگیری می‌کند؟



باور رایج در میان بسیاری از توسعه‌دهندگان این است که Kafka به دلیل ماهیت طراحی خود، تضمین می‌کند که هیچ پیامی گم نمی‌شود. با این حال، درک دقیق معماری و پیکربندی Kafka برای درک واقعی چگونگی و زمان گم شدن پیام‌ها و مهم‌تر از آن، نحوه جلوگیری از چنین سناریوهایی ضروری است.

شكل بالا نشان می‌دهد که چگونه یک پیام می‌تواند در طول چرخه عمر خود در Kafka گم شود.

### Producer

زمانی که () producer.send را برای ارسال پیام فراخوانی می‌کنیم، پیام مستقیماً به broker ارسال نمی‌شود. دو thread<sup>۱</sup> و یک صفحه<sup>۲</sup> در فرایند ارسال پیام دخیل هستند:

- Application thread
- Record accumulator
- Sender thread (I/O thread)

برای اطمینان از ارسال پیام‌ها به کارگزار، نیاز به پیکربندی صحیح «تأییدیه‌ها» (acks) و retries<sup>۲</sup> برای تولیدکننده داریم.  
کارگزار (Broker) :

یک خوشه کارگزار<sup>۳</sup> در حالت عادی نباید پیام‌ها را گم کند. با این حال، لازم است درک کنیم که چه شرایط حاد و بحرانی ممکن است منجر به گم شدن پیام شود:

<sup>۱</sup> queue

<sup>۲</sup> تلاش‌های مجدد

<sup>۳</sup> broker cluster

پیام‌ها معمولاً برای دستیابی به توان عملیاتی ورودی/خروجی (I/O) بالاتر به صورت ناهم‌زمان<sup>۱</sup> روی دیسک نوشته می‌شوند، بنابراین اگر سیستم قبل از نوشتن شدن پیام‌ها از کار بیفتد، پیام‌ها گم می‌شوند.

تکثیر‌شونده‌ها<sup>۲</sup> در خوشة Kafka باید به درستی پیکربندی شوند تا یک نسخه معتبر از داده‌ها را نگه دارند. قطعی بودن<sup>۳</sup> در همگام‌سازی داده‌ها مهم است.

#### صرف‌کننده (Consumer):

Kafka روش‌های مختلفی برای تأیید دریافت پیام‌ها ارائه می‌دهد. تأیید خودکار<sup>۴</sup> ممکن است پردازش رکوردها را قبل از اینکه واقعاً پردازش شوند را تأیید کند. هنگامی که صرف‌کننده در حین پردازش از کار می‌افتد، ممکن است برخی از رکوردها هرگز پردازش نشوند.

بهترین روش این است که ترکیبی از تأییدهای هم‌زمان و ناهم‌زمان را انجام دهیم، جایی که از تأییدهای ناهم‌زمان در حلقه پردازش برای توان عملیاتی بالاتر و تأییدهای هم‌زمان در مدیریت استثناء<sup>۵</sup> برای اطمینان از همیشه تأییدشدن آخرين موقعیت استفاده می‌کنیم.

asynchronous <sup>۱</sup>

replica <sup>۲</sup>

determinism <sup>۳</sup>

Auto-committing <sup>۴</sup>

exception handling <sup>۵</sup>

## مقایسه Kafka و Rabbitmq

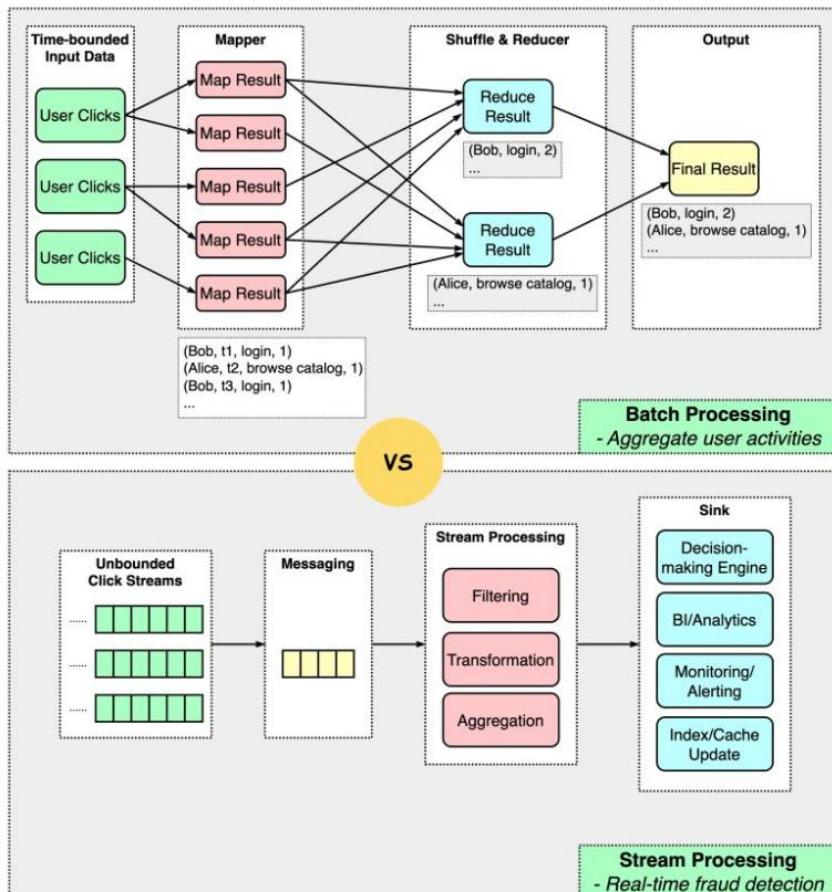
ویژگی	RabbitMQ	Kafka
معماری	مسیریابی پیچیده پیام، push model	طراحی مبتنی بر پارسیشن برای پردازش جریان با کارایی بالا به صورت بلاذرنگ، pull model
مدیریت پیام	صرف کنندگان پیگیری بازیابی پیام را با ردیاب آفست صرف پیام را انجام می‌دهد، حذف پیام پس از صرف آن، پشتیبانی از اولویت پیام	آن انجام می‌دهد، Kafka حفظ پیام بر اساس سیاست‌ها نگهداری خاص پردازش می‌کند، بدون اولویت پیام
عملکرد	تأخیر کم، هزاران پیام در ثانیه	انتقال بلاذرنگ، تا میلیون‌ها پیام در ثانیه
زبان برنامه‌نویسی و پروتکل	پشتیبانی از طیف گسترده‌ای از زبان‌ها و پروتکل‌های قدیمی	انتخاب محدود زبان، پروتکل باینری روی TCP
توصیف	بر اساس پروتکل (Advanced Message Queuing Protocol)	پلتفرم توزیع شده جریان رویداد
محبوبیت	در سیستم‌های سازمانی و مالی محبوب است	انتخاب محبوب برای معماهی های مبتنی بر رویداد و میکروسرویس‌ها
کارایی	۳۰ تا ۴۰ هزار پیام در ثانیه	۲ میلیون پیام در ثانیه
اندازه بار مفید	بدون محدودیت	محدودیت پیش‌فرض ۱ مگابایت (قابل تنظیم)
نگهداری پیام	مبتنی بر تأیید	مبتنی بر رویکرد خاص (مثلاً نگهداری ۲ روز)
نوع تبادل	مستقیم، header، topic، fanout	مبتنی بر انتشار و اشتراک
کاربردها	سیستم‌های سازمانی، مالی	مانیتورینگ، لاگ‌ها، داده‌های عملیاتی، سهام

## پردازش Stream در مقابل پردازش Batch

نمودار زیر یک سناریوی معمول با کلیک‌های کاربر را نشان می‌دهد:

### Batch v.s. Stream Processing

[blog.bytebytogo.com](http://blog.bytebytogo.com)



- پردازش Batch: ما فعالیت‌های کلیک کاربر را در پایان روز جمع‌آوری می‌کنیم.
- پردازش Stream: ما کلاهبرداری‌های احتمالی را با Stream‌های کلیک کاربر به صورت بلادرنگ تشخیص می‌دهیم.

هر دو مدل پردازش در پردازش داده‌های بزرگ استفاده می‌شوند. تفاوت‌های اصلی عبارت‌اند از:

### ورودی

پردازش Batch روی داده‌های با شرایط مرزی در زمان<sup>۱</sup> کار می‌کند به این معنی که ورودی داده‌ها در نهایت، پایان می‌یابد.

پردازش Stream روی جریان داده کار می‌کند که مرزی ندارد.  
به موقع بودن

پردازش Batch در سناریوهایی استفاده می‌شود که داده‌ها نیازی به پردازش در بلاذرنگ ندارند. پردازش Stream می‌تواند نتایج پردازش را هم‌زمان با تولید داده ایجاد کند.

### خروجی

پردازش Batch معمولاً نتایج یکباره تولید می‌کند، به عنوان مثال در گزارش‌ها. خروجی‌های پردازش Stream می‌توانند به موتورهای تصمیم‌گیری کلاهبرداری، ابزارهای نظارت، ابزارهای تحلیلی یا بهروزرسانی‌های index/cache متصل شوند.

### تحمل خطأ

پردازش Batch خطاهای را بهتر تحمل می‌کند زیرا Batch را می‌توان روی یک مجموعه داده ورودی ثابت به صورت مجدد پخش کرد. پردازش Stream چالش‌برانگیز‌تر است؛ زیرا داده‌های ورودی به طور مداوم در جریان هستند. برخی از روش‌ها برای حل این مشکل وجود دارد:

الف - میکرو دسته‌بندی<sup>۲</sup> که Stream داده را به بلوک‌های کوچک‌تر تقسیم می‌کند (استفاده شده در Spark):

ب - Checkpoint که هر چند ثانیه یک علامت برای بازگشت به گذشته<sup>۳</sup> ایجاد می‌کند (استفاده شده در Flink).

---

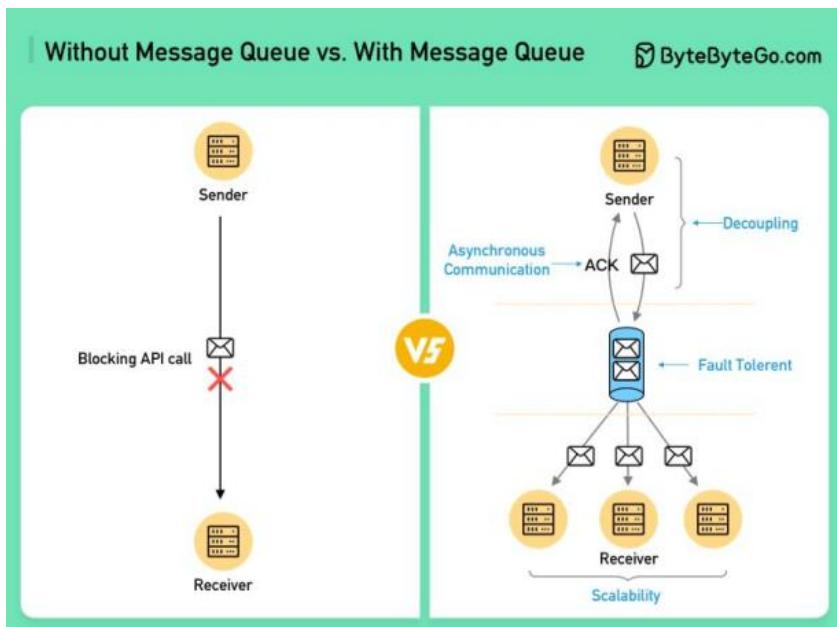
<sup>۱</sup> time-bounded

<sup>۲</sup> Microbatching

<sup>۳</sup> roll back

## چرا به message brokers نیاز داریم؟

می‌دانیم message brokers نقش مهمی در ساخت سیستم‌های توزیع شده یا میکروسرویس‌ها برای بهبود عملکرد، مقیاس‌پذیری و قابلیت نگهداری آنها دارند.



- جداسازی message brokers : با ایجاد جداسازی<sup>۱</sup> بین اجزای نرم‌افزاری، توسعه، استقرار و مقیاس‌پذیری مستقل را تقویت می‌کنند. نتیجه آن نگهداری و عیب‌یابی آسان‌تر است.
- ارتباط ناهمگام<sup>۲</sup> : یک واسطه پیام به اجزای مختلف اجرازه می‌دهد بدون انتظار برای پاسخ ارتباط برقرار کنند که باعث افزایش کارایی سیستم و امکان توزیع بار مؤثر می‌شود.

Decoupling<sup>۱</sup>

Asynchronous communication<sup>۲</sup>

- Message broker ها با ارائه buffering و پایداری پیام<sup>۱</sup>، اطمینان حاصل می کنند که پیام ها در هنگام خرابی اجزا از بین نمی روند.
  - مقیاس پذیری: Message broker ها می توانند حجم بالایی از پیام ها را مدیریت کنند و به سیستم شما اجازه می دهند با افزودن نمونه های بیشتر از Message broker ها در صورت نیاز، به صورت افقی مقیاس بندی شوند.
- به طور خلاصه، یک Message brokers می تواند کارایی، مقیاس پذیری و قابلیت اطمینان را در معماری شما بهبود بخشد. در نظر گرفتن استفاده از یک Message brokers می تواند به موفقیت بلندمدت برنامه شما کمک زیادی کند. همیشه به تصویر بزرگ تر فکر کنید و اینکه انتخاب های طراحی شما چگونه بر کل پروژه تأثیر می گذارد.

# سرویس‌های و معماری‌های ابری

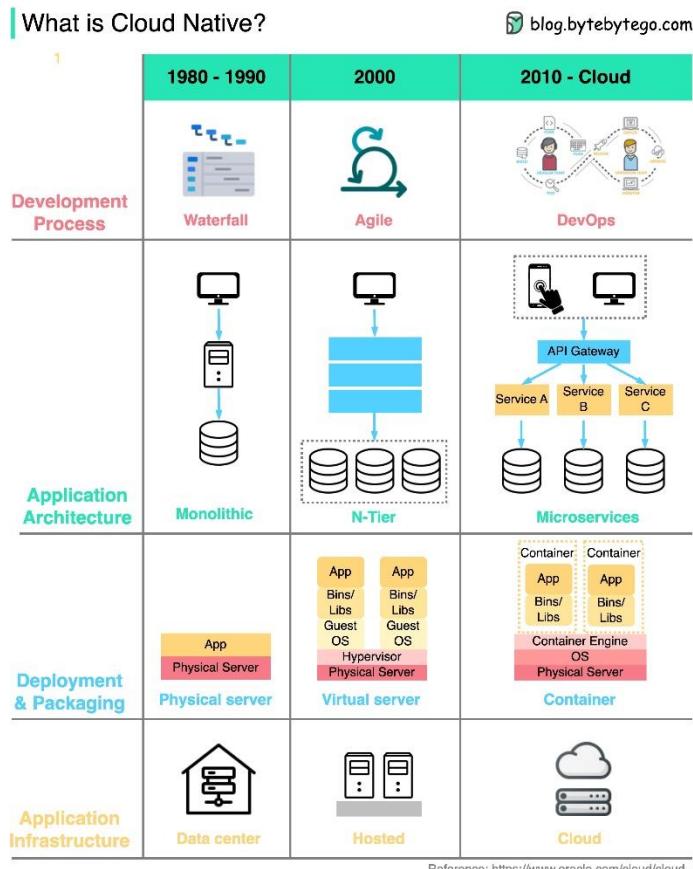
یک تصویر زیبا از سرویس‌های مختلف ابری (نسخه ۲۰۲۳).

Cloud Comparison Cheat Sheet [blog.bytebytogo.com](http://blog.bytebytogo.com)

AWS	Azure	Google Cloud	ORACLE CLOUD
Elastic Compute Cloud (EC2)	Virtual Machine	Compute Engine	Virtual Machine
Elastic Kubernetes Service (EKS)	Azure Kubernetes Service (AKS)	Google Kubernetes Engine (GKE)	Oracle Container Engine
Lambda	Azure Functions	Cloud Functions	OCI Functions
Simple Storage Service (S3)	Blob Storage	Cloud Storage	Object Storage
Elastic Block Store	Managed Disk	Persistent Disk	Persistent Volume
Elastic File System	File Storage	File Store	File Storage
Virtual Private Cloud	Virtual Network	Virtual Private Cloud	Virtual Cloud Network
Route 53	DNS	Cloud DNS	DNS
Elastic Load Balancing	Load Balancer	Cloud Load Balancing	Load Balancer
Web Application Firewall	Web Application Firewall	Cloud Armor	Web Application Firewall
RDS	SQL Database	Cloud SQL	ATP
DynamoDB	Cosmos DB	Firebase Realtime Database	NoSQL Database
Redshift	Synapse Analytics	BigQuery	Autonomous Data Warehouse
Elastic MapReduce	HDInsight	Dataproc	Big Data
Kinesis	Streaming Analytics	Dataflow	Streaming
SageMaker	Machine Learning	Vertex AI	Data Science
Glue	Data Factory	Data Fusion	Data Integration
EventBridge	Event Grid	Eventarc	Events
Simple Queuing Service	Storage Queues	Pub/Sub	Streaming
Simple Notification Service	Service Bus	Firebase Cloud Messaging	Notifications
CloudWatch	Monitor	Cloud Monitoring	Monitoring
CloudFormation	Resource Manager	Deployment Manager	Resource Manager
IAM	Active Directory	Cloud Identity	IAM
KMS	Key Vault	Cloud KMS	Vault

## چیست؟ cloud native

در زیر نموداری است که تکامل معماری و فرایندها را از دهه ۱۹۸۰ نشان می‌دهد.



سازمان‌ها با استفاده از فناوری‌های ابری «مبتنی بر Cloud<sup>1</sup>» می‌توانند بر روی cloud های عمومی، خصوصی و ترکیبی، اپلیکیشن‌های مقیاس‌پذیر بسازند و اجرا کنند.

این بدین معناست که این اپلیکیشن‌ها برای استفاده از قابلیت‌های cloud طراحی شده‌اند، بنابراین در برابر بار/load مقاوم بوده و به راحتی مقیاس‌پذیر هستند.

#### شامل ۴ جنبه است: «Cloud native»

۱. فرایندهای توسعه: این فرایند از آبشاری (Waterfall) به چاپک (Agile) و سپس

به DevOps تغییریافته است.

۲. معماری اپلیکیشن: معماری از یکپارچه<sup>۱</sup> به میکروسرویس‌ها<sup>۲</sup> تغییریافته است. هر

سرویس برای کوچک بودن و تطبیق‌پذیری با منابع محدود در containerهای ابری طراحی شده است.

۳. استقرار و بسته‌بندی: اپلیکیشن‌ها قبلًاً روی سرورهای فیزیکی مستقر می‌شدند.

سپس، در حدود سال ۲۰۰۰، اپلیکیشن‌هایی که به تأخیر حساس نبودند را معمولاً روی سرورهای مجازی مستقر می‌کردند. در اپلیکیشن‌های مبتنی بر cloud، آنها در قالب Docker image بسته‌بندی شده و در containerها مستقر می‌شوند.

۴. زیرساخت اپلیکیشن: اپلیکیشن‌ها به جای سرورهای اختصاصی، به صورت گستره

روی زیرساخت ابری مستقر می‌شوند.

## IaaS/PaaS/SaaS چیست؟

نمودار زیر تفاوت بین IaaS (زیرساخت به عنوان یک سرویس<sup>۱</sup>، PaaS (پلتفرم به عنوان یک سرویس<sup>۲</sup>) و SaaS (نرم‌افزار به عنوان یک سرویس<sup>۳</sup>) را نشان می‌دهد.



در یک برنامه غیر ابری، ما مالک تمام ساخت‌افزارها و نرم‌افزارها هستیم و مدیریت آن‌ها را بر عهده داریم. می‌گوییم برنامه و می‌گوییم برنامه به صورت on-premises اجرا می‌شود. با محاسبات ابری، ارائه‌دهندگان سرویس‌های ابری سه مدل مختلف برای استفاده ما فراهم می‌کنند: SaaS، PaaS، IaaS.

سرویس IaaS به ما دسترسی به زیرساخت‌های ارائه‌دهندگان ابرمانند سرورها، ذخیره‌سازی و شبکه را می‌دهد. ما برای سرویس زیرساخت هزینه پرداخت می‌کنیم و نرم‌افزارهای پشتیبان را برای برنامه خود نصب و مدیریت می‌کنیم.

سرویس PaaS حتی فراتر نیز می‌رود. این یک پلتفرم با انواع میان‌افزارها، چارچوب‌ها و ابزارها برای ساخت برنامه ما فراهم می‌کند. ما فقط روی توسعه برنامه و داده‌ها تمرکز می‌کنیم. سرویس SaaS اجازه اجرای برنامه در ابر را می‌دهد. ما هزینه ماهانه یا سالانه برای استفاده از محصول SaaS پرداخت می‌کنیم.

<sup>۱</sup> Infrastructure-as-a-Service

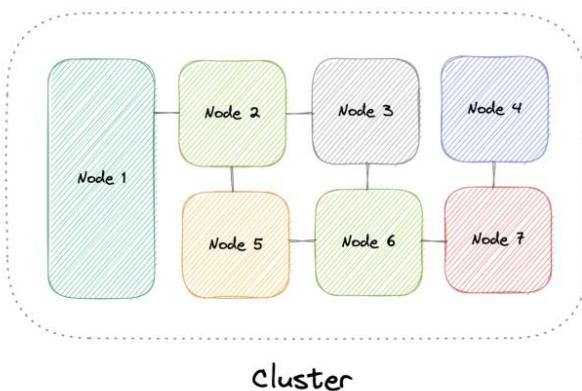
<sup>۲</sup> Platform-as-a-Service

<sup>۳</sup> Software-as-a-Service

## خوشه‌بندی (Clustering)

یک خوشه کامپیوتری گروهی از دو یا چند کامپیوتر یا گره<sup>۱</sup> است که به صورت موازی برای رسیدن به یک هدف مشترک اجرا می‌شوند. این امر امکان توزیع حجم کاری متشكل از تعداد زیادی از وظایف و تسکوهای مستقل و قابل موازی‌سازی بین گره‌های موجود در خوشه را فراهم می‌کند. در نتیجه، این وظایف می‌توانند از حافظه و قدرت پردازش ترکیبی هر رایانه برای افزایش عملکرد کلی استفاده کنند.

برای ساخت یک خوشه کامپیوتری، گره‌های جداگانه باید برای امکان ارتباط بین گره‌ها به یک شبکه متصل شوند. سپس می‌توان از نرم‌افزار برای اتصال گره‌ها به هم و تشکیل یک خوشه استفاده کرد. هر خوشه ممکن است یک دستگاه ذخیره‌سازی مشترک یا فضای ذخیره‌سازی محلی روی هر گره داشته باشد.



node<sup>۱</sup>

به طور معمول، حداقل یک گره به عنوان گره رهبر<sup>۱</sup> تعیین می‌شود و به عنوان نقطه ورود به خوشه عمل می‌کند. گره رهبر ممکن است مسئولیت واگذاری کارهای ورودی به سایر گرهها و در صورت لزوم، جمع‌آوری نتایج و بازگرداندن پاسخ به کاربر را بر عهده داشته باشد.

به طور ایده‌آل، یک خوشه به گونه‌ای عمل می‌کند که گویی یک سیستم واحد است. کاربری که به خوشه دسترسی پیدا می‌کند و نباید بداند که این سیستم یک خوشه یا یک ماشین مستقل است.

علاوه بر این، یک خوشه باید برای به حداقل رساندن تأخیر و جلوگیری از گلوگاه‌ها در ارتباط گره به گره طراحی شود.

## أنواع خوشه‌ها

خوشه‌های کامپیوتري به طور کلی به سه نوع طبقه‌بندی می‌شوند:

۱. در دسترس پذیری بالا یا جبران‌کننده خود کار<sup>۲</sup>:

این نوع خوشه برای افزایش در دسترس بودن سرویس‌ها و برنامه‌ها طراحی شده است. در صورت خرابی یک گره، گره دیگر وظایف آن را بر عهده می‌گیرد تا وقفه‌ای در سرویس دهی رخ ندهد.

۲. توزیع کننده بار<sup>۳</sup>:

leader node<sup>۱</sup>

Highly available or fail-over<sup>۲</sup>

Load balancing<sup>۳</sup>

این نوع خوش‌بازی توزیع بار کاری در چندین گره و بهبود عملکرد کلی سیستم طراحی شده است.

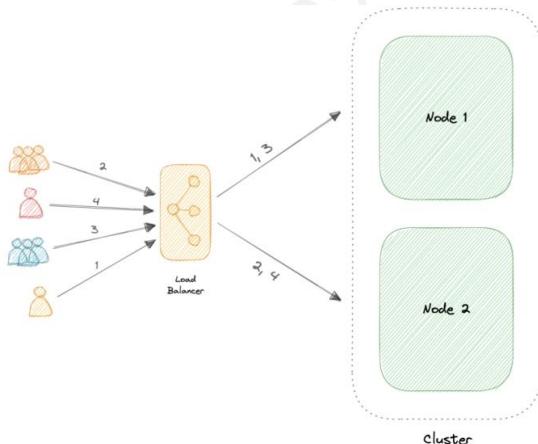
### ۳. محاسبات با عملکرد بالا (HPC<sup>۱</sup>):

این نوع خوش‌بازی برای انجام محاسبات علمی و مهندسی پیچیده که نیاز به قدرت پردازش عظیم دارند، طراحی شده است.

## تنظیمات

دو پیکربندی رایج خوش‌بندی با در دسترس پذیری بالا (HA<sup>۲</sup>؛ فعال - فعال) و فعال - غیرفعال (active-passive) هستند.

(Active-Active - فعال - فعال)



یک خوش‌بازی - فعال به طور معمول از حداقل دو گره تشکیل شده است که هر دو به طور هم‌زمان یک نوع سرویس مشابه را به طور فعال اجرا می‌کنند. هدف اصلی یک خوش‌بازی - فعال دستیابی به توزیع بار است. یک توزیع کننده بار<sup>۳</sup>، حجم کاری را در همه گره‌ها توزیع می‌کند تا اضافه‌بار شدن روی یک گره خاص جلوگیری شود. از آنجایی که گره‌های بیشتری

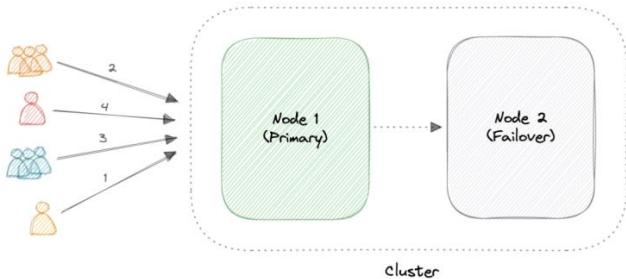
High-performance computing<sup>۱</sup>

high availability<sup>۲</sup>

load balancing<sup>۳</sup>

برای سرویس‌دهی در دسترس هستند، در نتیجه‌ی آن بهبودی در توان عملیاتی و زمان پاسخگویی نیز حاصل می‌شود.

### خوش‌فعال - غیرفعال (Active-Passive)



مانند پیکربندی خوش‌فعال - فعال، یک خوش‌فعال - غیرفعال نیز حداقل از دو گره تشکیل شده است. با این حال، همان‌طور که از نام فعال - غیرفعال پیداست، همه گره‌ها فعال نخواهند بود. به عنوان مثال، در مورد دو گره، اگر گره اول در حال حاضر فعال باشد، گره دوم باید غیرفعال یا در حالت آماده‌باش باشد.

### مزایای خوش‌بندی

چهار مزیت کلیدی رایانه‌های خوش‌بندی به شرح زیر است:

- High availability: خوش‌بندی با داشتن یک گره پشتیبان در صورت خرابی گره اصلی به دردسترس بودن مداوم سرویس کمک می‌کند.
- Scalability: خوش‌های را می‌توان به راحتی با افزودن گره‌های بیشتر برای برآوردن نیازهای پردازش و ذخیره‌سازی، مقیاس‌دهی داد.
- Performance: گره‌های متعدد در یک خوش‌بندی با توزیع بار کاری، منجر به بهبود عملکرد کلی سیستم می‌شوند.

- صرفه‌جویی در هزینه: خوش‌های می‌توانند جایگزین سخت‌افزار گران‌قیمت شوند، زیرا چندین سرور با منابع متوسط می‌توانند عملکرد مشابه یک سرور گران‌قیمت را ارائه دهند.

### توزيع کننده بار<sup>۱</sup> در مقابل خوش‌بندی

توزیع کننده بار برخی ویژگی‌های مشترک با خوش‌بندی دارد، اما آنها فرایندهای متفاوتی هستند. خوش‌بندی افزونگی<sup>۲</sup> را فراهم می‌کند و ظرفیت و دردسترس‌بودن<sup>۳</sup> را افزایش می‌دهد. سرورهای موجود در یک خوش‌ه از یکدیگر آگاه هستند و برای رسیدن به یک هدف مشترک با همکار می‌کنند، اما با توزیع بار، سرورها از وضعیت یکدیگر آگاه نیستند. در عوض، آنها به درخواست‌هایی که از توزیع کننده بار دریافت می‌کنند واکنش نشان می‌دهند. ما می‌توانیم توزیع کننده بار را همراه با خوش‌بندی به کار بگیریم، اما همچنین در مواردی که سرورهای مستقل با هدف مشترکی مانند اجرای یک وب‌سایت، برنامه‌ی تجاری، سرویس و ب یا برخی منابع فناوری اطلاعات دیگر را دارند، قابل اجرا است.

### چالش‌های خوش‌بندی

واضح‌ترین چالشی که خوش‌بندی ارائه می‌کند، افزایش پیچیدگی نصب و نگهداری است. سیستم عامل، برنامه و وابستگی‌های آن باید روی هر گره نصب و به روزرسانی شوند. این امر در صورتی که گره‌های موجود در خوش‌بندی ناهمگن نباشند، حتی پیچیده‌تر می‌شود. همچنین باید بدقت از میزان استفاده از منابع برای هر گره نظارت کرد و لایک‌ها باید تجمعی شوند تا اطمینان حاصل شود که نرم‌افزار به درستی کار می‌کند.

علاوه بر این، مدیریت ذخیره‌سازی دشوارتر می‌شود، یک دستگاه ذخیره‌سازی مشترک باید از بازنویسی داده‌ها توسط گره‌ها بر روی یکدیگر جلوگیری کند و بانک‌های اطلاعاتی توزیع شده باید همگام‌سازی شوند.

Load balancing<sup>۱</sup>

Redundancy<sup>۲</sup>

availability<sup>۳</sup>

## مفاهیم و تعاریف الگوهای طراحی cloud native

### ۱. دردسترس بودن (Availability)

دردسترس بودن<sup>۱</sup>، میزان زمانی را که سیستم دارای عملکرد مناسب بوده و برای کاربران قابل دسترسی است. عواملی مانند خطاهای سیستم، مشکلات زیرساخت، حملات مخرب و بار روی سیستم بر دردسترس بودن تأثیرگذار است. دردسترس بودن معمولاً به صورت درصد زمان بالابودن (uptime) اندازه‌گیری می‌شود. اپلیکیشن‌های ابری معمولاً توافق‌نامه سطح سرویس (SLA<sup>2</sup>) را برای کاربران فراهم می‌کنند، به این معنی که برنامه‌ها باید به‌گونه‌ای طراحی و اجرا شوند که دردسترس بودن را به حداقل برسانند.

### ۲. مدیریت داده (Data Management)

مدیریت داده عنصر کلیدی اپلیکیشن‌های ابری است و بر اکثر ویژگی‌های کیفی تأثیر می‌گذارد. داده‌ها معمولاً به دلایل مختلفی از جمله عملکرد مناسب، مقیاس‌پذیری یا دردسترس بودن در مکان‌های مختلف و روی چندین سرور میزبانی می‌شوند که این امر می‌تواند چالش‌های متعددی را به همراه داشته باشد. به عنوان مثال، یکپارچگی داده<sup>۳</sup> باید حفظ شود و داده‌ها معمولاً باید در مکان‌های مختلف همگام‌سازی<sup>۴</sup> شوند.

### ۳. طراحی و پیاده‌سازی (Design and Implementation)

طراحی خوب شامل عواملی مانند انسجام و هم‌خوانی در طراحی و استقرار اجزا<sup>۵</sup>، قابلیت نگهداری برای ساده‌سازی مدیریت و توسعه و قابلیت استفاده مجدد برای امکان استفاده از اجزا و زیرسیستم‌ها در سایر اپلیکیشن‌ها و سناریوهای می‌شود. تصمیماتی که در مرحله طراحی

Availability<sup>۱</sup>

service level agreement<sup>2</sup>

data consistency<sup>۳</sup>

synchronize<sup>۴</sup>

component<sup>۵</sup>

و پیاده‌سازی گرفته می‌شود معمولاً تأثیر بسزایی بر کیفیت و کل هزینه مالکیت (TCO<sup>۱</sup>) اپلیکیشن‌های و خدمات میزبانی شده در محیط ابری دارد.

#### ۴. پیام‌رسانی:

ماهیت توزیع شده اپلیکیشن‌های ابری، نیازمند زیرساخت پیام‌رسانی است که اجزا و سرویس‌ها را به هم متصل کند و بسیار ایده‌آل است که این اتصال با پیوستگی ضعیف<sup>۲</sup> باشد تا مقیاس‌پذیری به حداقل برسد. پیام‌رسانی ناهم‌زمان<sup>۳</sup> به طور گسترده استفاده می‌شود و مزایای زیادی به همراه دارد، اما همچنین چالش‌هایی مانند ترتیب پیام‌ها، مدیریت پیام‌های مخرب<sup>۴</sup> و idempotency<sup>۵</sup> و موارد دیگر را به همراه دارد.

#### ۵. مدیریت و نظارت:

اپلیکیشن‌های ابری در یک مرکز داده<sup>۶</sup> به صورت remote اجرا می‌شوند که در آنجا کنترل کاملی روی زیرساخت یا در برخی موارد سیستم عامل ندارید. این امر می‌تواند مدیریت و نظارت را نسبت به استقرار روی سرورهای داخل شرکت یا در اصطلاح on-premises دشوارتر کند. اپلیکیشن‌ها باید اطلاعات زمان اجرا<sup>۷</sup> را در معرض دید قرار دهند تا مدیران و اپراتورها بتوانند از آن برای مدیریت و نظارت بر سیستم استفاده کنند و همچنین از نیازهای تجاری در حال تغییر و سفارشی‌سازی پشتیبانی کنند بدون اینکه نیاز به توقف یا استقرار مجدد برنامه باشد.

<sup>۱</sup> total cost of ownership

<sup>۲</sup> loosely coupled

<sup>۳</sup> asynchronous messaging

<sup>۴</sup> poison message

<sup>۵</sup> در ریاضیات و علوم کامپیوتر، به ویژگی خشی بودن تکرار یک رابطه (یا تابع یا عملیات) یا «خودتوانی» گفته می‌شود؛ یعنی اینکه تکرار، در نتیجه نهایی یا محتوای اطلاعات تغییری ایجاد نکند.

<sup>۶</sup> data center

<sup>۷</sup> runtime

## ۶. کارایی و مقیاس‌پذیری:

کارایی<sup>۱</sup>، نشانگر پاسخگویی سیستم برای اجرای هر اقدامی در یک بازه زمانی مشخص است، در حالی که مقیاس‌پذیری، توانایی سیستم برای مدیریت افزایش بار بدون تأثیر بر عملکرد یا امکان افزایش آسان منابع در دسترس است. اپلیکیشن‌های ابری معمولاً<sup>۲</sup> با حجم کاری متغیر و اوج فعالیت مواجه می‌شوند. پیش‌بینی این موارد، به ویژه در سناریوی چند مستأجر<sup>۳</sup> (multi-tenant)، تقریباً غیرممکن است. در عوض، اپلیکیشن‌ها باید بتوانند طرح‌هایی برای پاسخگویی به اوج تقاضای مصرف‌کننده‌ها در حالت مقیاس افقی (scale out) و با کاهش تقاضا مقیاس عمودی (scale in) داشته باشند. مقیاس‌پذیری نه تنها مربوط به نمونه‌های محاسباتی<sup>۴</sup>، بلکه به سایر عناصر مانند ذخیره‌سازی داده، زیرساخت پیام‌رسانی و موارد دیگر نیز مربوط می‌شود.

## ۷. انعطاف‌پذیری (Resiliency):

قابلیت بازیابی و توانایی یک سیستم برای مدیریت و بازیابی از خرابی‌ها به صورت نرم را انعطاف‌پذیری یا Resiliency گویند. ماهیت میزبانی ابری<sup>۴</sup>، جایی که برنامه‌ها اغلب چند - مستأجر (multi-tenant) هستند و از سرویس‌های پلتفرمی مشترک استفاده می‌کنند و برای منابع و پنهانی باند رقابت می‌کنند و حتی از طریق اینترنت ارتباط برقرار می‌کنند و روی سخت‌افزار معمولی اجرا می‌شوند، بنابراین همه‌ی این‌ها به این معنی است که احتمال بروز

## Performance<sup>۱</sup>

<sup>۲</sup> در مهندسی رایانه، tenants (در لغت به معنای مستأجر یا ساکن) به واحدهای جداگانه‌ای از یک سیستم یا برنامه اطلاق می‌شود که به طور مستقل عمل می‌کنند و منابع را به اشتراک می‌گذارند. این مفهوم شبیه به آپارتمان‌های یک ساختمان است که هر کدام واحد مجزا با حریم خصوصی و منابع مختص به خود هستند، اما همگی در یک ساختمان مشترک و زیر یک سقف قرار دارند.

<sup>۳</sup> compute instances

<sup>۴</sup> cloud hosting

خطاهای گذرا و دائمی افزایش می‌یابد. تشخیص خرابی‌ها و بازیابی سریع و کارآمد برای حفظ قابلیت بازیابی ضروری است.

#### ۸. امنیت:

امنیت یا Security به معنی توانایی یک سیستم برای جلوگیری از اقدامات مخرب یا تصادفی خارج از محدوده استفاده طراحی شده و جلوگیری از افشا یا ازدست‌رفتن اطلاعات است. برنامه‌های ابری در اینترنت و خارج از مرزهای قابل اعتماد درون‌سازمانی قرار دارند که اغلب برای عموم باز و قابل دسترسی هستند و ممکن است به کاربران غیر مورد اعتماد سرویس دهند. برنامه‌ها باید به گونه‌ای طراحی و پیاده‌سازی شوند که از حملات مخرب محافظت کنند و دسترسی را تنها به کاربران مجاز محدود کنند و از داده‌های حساس محافظت نمایند.

### بررسی SLI، SLO و SLA

SLI و SLO به شرکت‌ها امکان می‌دهند تا وعده‌هایی را که برای یک سرویس به کاربرانش داده شده است، تعریف، پیگیری و نظارت کنند. SLI، SLO و SLA با هم باید به تیم‌ها کمک کنند تا اعتماد کاربر به سرویس‌های خود را با تأکید بیشتر بر بهبود مستمر فرایندهای مدیریت و پاسخگویی به حادثه، افزایش دهند.

#### SLA (Service Level Agreement)

SLA یا توافق‌نامه سطح سرویس، توافقی است که بین یک شرکت و کاربران آن برای یک سرویس خاص منعقد می‌شود. SLA وعده‌های متفاوتی را که شرکت به کاربران در رابطه با معیارهای خاص، مانند دردسترس‌بودن<sup>۱</sup> سرویس و سایر موارد دیگر می‌دهد را تعریف می‌کند. SLA اغلب توسط تیم تجاری یا حقوقی شرکت نوشته می‌شود.

#### SLO (Service Level Objective)

<sup>۱</sup> availability

SLO یا هدف سطح سرویس، توافقی است که یک شرکت به کاربران در رابطه با یک معیار خاص مانند پاسخ به حادثه یا زمان کارکرد بدون خرابی (uptime) می‌دهد. SLOها در داخل یک SLA به عنوان توافق‌های جداگانه‌ای وجود دارند که در توافقنامه کامل کاربر گنجانده شده‌اند. SLO هدف خاصی است که سرویس باید برای مطابقت با SLA براورده کند. SLOها همیشه باید ساده و به طور واضح تعریف شده و به راحتی قابل اندازه‌گیری باشند تا مشخص شود که آیا هدف براورده شده است یا خیر.

### SLI (Service Level Indicator)

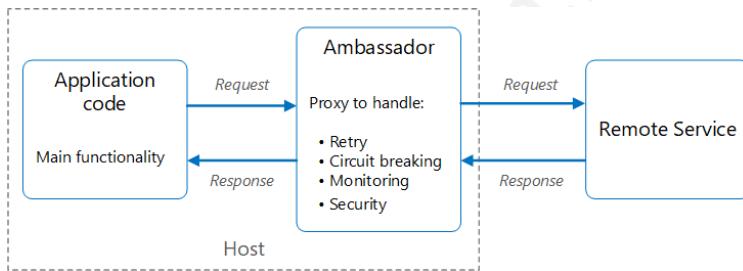
SLI یا شاخص سطح سرویس، یک معیار کلیدی است که برای تعیین اینکه آیا الزامات SLO براورده شده است یا خیر، استفاده می‌شود. این مقدار اندازه‌گیری شده‌ی معیاری است که در SLO توضیح داده شده است. برای اینکه در انطباق با SLA باقی بمانند، مقدار همیشه باید مقدار تعیین شده توسط SLO را براورده کند یا از آن بیشتر شود.

## الگوهای طراحی cloud native

در این مبحث معماری‌های cloud native و میکروسرویس به صورت خلاصه معرفی شده است. برای بررسی بیشتر به نسخه قبل از انتشار کتاب [cloud design pattern](#) مراجعه کنید.

### الگوی سفیر - پروکسی (Ambassador - Proxy)

الگوی Ambassador بر روی انتقال تمام وظایف اصلی که برای برنامه‌ها حیاتی هستند به جز business logic تمرکز می‌کند تا برنامه‌ها بتوانند فقط روی موارد استفاده حیاتی تمرکز کنند. این الگو به عنوان یک واسطه بین برنامه‌ها و سرویس‌هایی که با آن‌ها ارتباط برقرار می‌کند عمل می‌کند و وظایفی مانند لاغ‌گیری، مانیتورینگ یا مدیریت retry‌ها و محدودسازی نرخ را بر عهده می‌گیرد.



به عنوان مثال، کوبرتیز از Envoy به عنوان یک برای ارتباط ساده بین سرویس‌ها استفاده می‌کند.

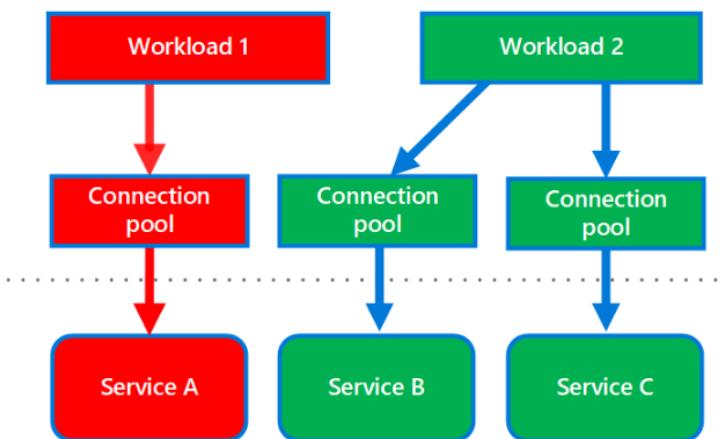
### الگوی Bulkhead

اجزای یک سیستم را ایزوله می‌کند تا خرابی در یک جزء باعث خرابی کل سیستم نشود. نام آن برگرفته از پارتیشن‌های برش خورده (بالک‌ها) در بدنه کشتی است. اگر بدنه کشتی آسیب بیند، فقط قسمت آسیب دیده پر از آب می‌شود که این کار از غرق شدن کشتی جلوگیری می‌کند.

الگوی Bulkhead در سناریوهای زیر بسیار مفید است:  
ایزولاسیون منابع: زمانی که نیاز به ایزوله کردن منابع مورد استفاده توسط سرویس‌های back-end مصرف‌کننده برای جلوگیری از رقابت منابع دارد.

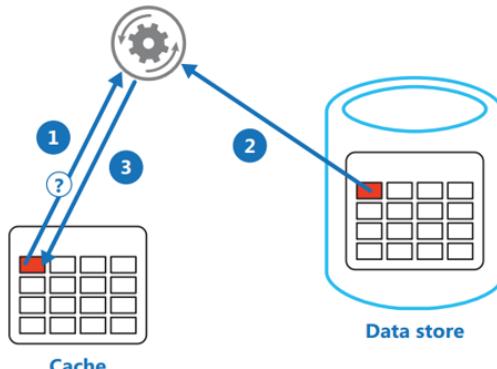
ایزولاسیون مصرف‌کننده بحرانی: برای محافظت از مصرف‌کنندگان بحرانی (سرویس‌ها) در برابر مصرف‌کنندگان استاندارد، اطمینان از در دسترس بودن و پاسخگویی سرویس‌های بحرانی حتی در زمان‌های اوج مصرف بار یا خرابی.

محافظت در برابر شکست زنجیره‌ای: برای محافظت از برنامه خود در برابر شکست‌های زنجیره‌ای که می‌تواند زمانی رخ دهد که مشکلات در یک سرویس بر سایرین تأثیر می‌گذارند.



### کش جانبی (Cache-aside) :

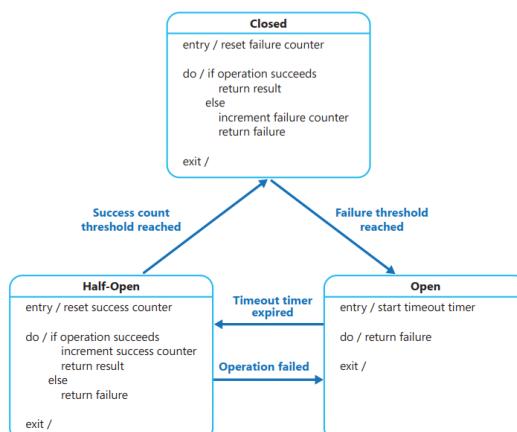
داده را بر اساس تقاضا از یک بانک اطلاعاتی به کش بارگذاری کنید. این الگو می‌تواند کارایی<sup>۱</sup> را بهبود بخشد و همچنین به حفظ انسجام بین داده‌های موجود در کش و داده‌های موجود در بانک اطلاعاتی اصلی کمک کند.



- ۱- بروزی کش؛ در ابتدا، برنامه بروزی می‌کند که آیا آیتم مورد نظر در حال حاضر در کش وجود دارد یا غیره.
- ۲- در صورت نبود آیتم در کش، آن را از پانک اطلاعاتی اصلی که داده‌ها را در خود جای داده است، می‌خواند.
- ۳- ذخیره آیتم در کش؛ بین از خواندن آیتم از پانک اطلاعاتی، یک کپی از آن در کش ذخیره می‌شود تا در خواسته‌های بعدی برای همان آیتم بتواند به طور بالقوه سریع‌تر از کش بازیابی شوند.

## قطع کننده مدار (Circuit Breaker)

هنگام اتصال به یک سرویس یا منبع remote، خطاهایی را که ممکن است برای رفع آن‌ها به زمان متغیر و نامعلومی نیاز باشد را مدیریت کنید. این الگو می‌تواند پایداری و قابلیت بازیابی یک برنامه را بهبود بخشد و منطق آن جلوگیری از تکرار مداوم خرابی در طول تعمیر و نگهداری یا خرابی موقت سیستم خارجی یا مشکلات غیرمنتظره سیستم را در بر می‌گیرد.



ایدهی اساسی پشت قطع کننده مدار بسیار ساده است. ما یک فرآخوانی تابع محافظت شده را در یک object قطع کننده محصور می‌کنیم که خرابی‌ها را کنترل می‌کند. هنگامی که خرابی‌ها به آستانه خاصی بررسند، قطع کننده قطع می‌شود و تمام تماس‌های بعدی با قطع کننده با خطا بازگشت می‌یابد، بدون اینکه تماس محافظت شده اصلاً برقرار شود. همچنین، معمولاً در صورت قطع شدن قطع کننده، به نوعی هشدار نظارتی نیاز خواهیم داشت.

این رایج است که سیستم‌های نرم‌افزاری تماس‌های remote نرم‌افزاری که در فرآیندهای مختلف اجرا می‌شود، احتمالاً روی ماشین‌های مختلف در سراسر شبکه برقرار کنند. یکی از تفاوت‌های بزرگ بین تماس‌های درون حافظه و تماس‌های remote این است که تماس‌های remote می‌توانند با شکست مواجه شوند یا بدون پاسخ تا رسیدن به محدودیت زمان وقفه، معلق بمانند. بدتر از آن این است که اگر فرآخواننده‌های زیادی روی یک سرویس دهنده بی‌پاسخ داشته باشیم، می‌توانیم منابع حیاتی را تمام کنیم که منجر به خرابی‌های متوالی در چندین سیستم می‌شود.

### حالاتی قطع کننده (States of Circuit Breaker)

#### بسته

هنگامی که همه چیز عادی است، قطع کننده‌ها بسته می‌مانند و تمام درخواست‌ها طبق معمول به سرویس‌ها منتقل می‌شوند. اگر تعداد خرابی‌ها از آستانه فراتر رود، قطع کننده قطع می‌شود و وارد حالت باز می‌شود.

#### باز

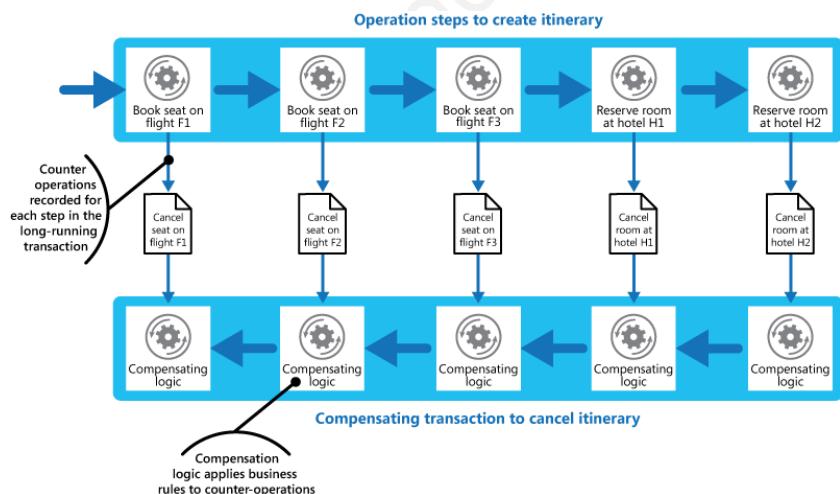
در این حالت، قطع کننده بلا فاصله و بدون فرآخوانی سرویس‌ها، خطایی را برمی‌گرداند. قطع کننده‌ها پس از سپری شدن یک دوره زمانی خاص به حالت نیمه‌باز می‌روند. معمولاً یک سیستم نظارتی وجود دارد که در آن زمان timeout مشخص می‌شود.

#### نیمه‌باز

در این حالت، قطع کننده اجازه می‌دهد تعداد محدودی از درخواست‌ها از سرویس عبور کند و عملیات را فراخوانی کند. اگر درخواست‌ها موفقیت‌آمیز باشند، قطع کننده به حالت بسته می‌رود. اما اگر درخواست‌ها همچنان با شکست مواجه شوند، به حالت باز بر می‌گردد.

### تراکنش جبرانی (Compensating Transaction)

اگر یک یا چند عملیات در یک سری از مراحل که با هم یک عملیات یکپارچگی تدریجی<sup>۱</sup> را تعریف می‌کنند با شکست مواجه شوند، آن‌گاه کار انجام‌شده توسط آن مراحل را لغو کنید. عملیاتی که از مدل یکپارچگی تدریجی پیروی می‌کنند را به طور معمول در برنامه‌های میزبانی‌شده در محیط ابری<sup>۲</sup> که فرایندهای تجاری و گردش کار پیچیده را اجرا می‌کنند، یافت می‌شوند.

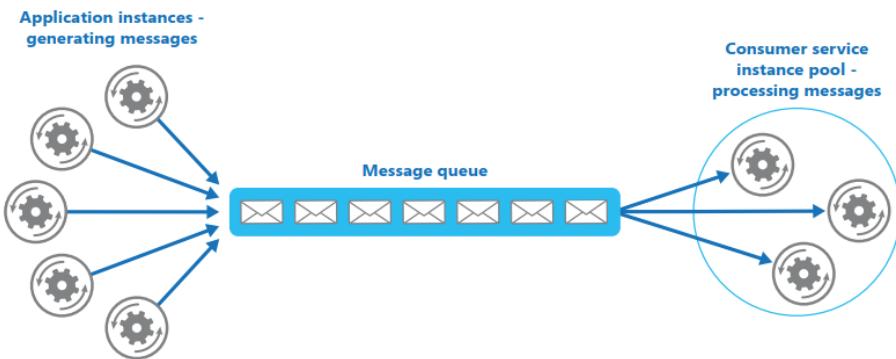


eventually consistent<sup>۱</sup>

cloud-hosted<sup>۲</sup>

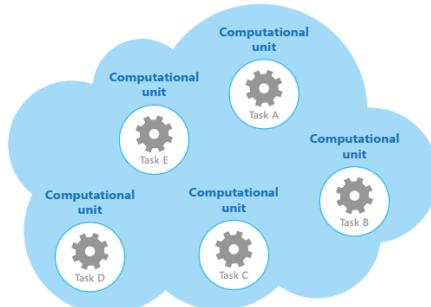
### مصرف کنندگان رقیب (Competing Consumers)

این امکان را به چندین مصرف کننده همزمان می دهد تا پیام هایی را که در یک کانال پیام رسانی مشابه دریافت می شوند را پردازش کنند. این الگو به یک سیستم امکان می دهد تا چندین پیام را به طور همزمان پردازش کند تا توان عملیاتی را بهینه کند و قابلیت مقیاس پذیری و availability را بهبود بخشد و بار کاری<sup>۱</sup> را متعادل کند.



### تجمیع منابع محاسباتی (Compute Resource Consolidation)

چندین کار یا عملیات را در یک واحد محاسباتی واحد ادغام کنید. این الگو می تواند استفاده از منابع محاسباتی را افزایش دهد و هزینه ها و سربار مدیریت مرتبط با انجام پردازش محاسباتی در برنامه های میزبانی شده در ابر را کاهش دهد.



## معماری رویدادمحور (Event-Driven Architecture - EDA)

معماری رویدادمحور (EDA) در مورد استفاده از رویدادها به عنوان راهی برقراری ارتباط درون یک سیستم است. به طور کلی، از یک کارگزار پیام (message broker) برای انتشار و مصرف رویدادها به صورت ناهم‌زمان استفاده می‌شود. ناشر (publisher) از اینکه چه کسی رویداد را مصرف (consuming) می‌کند بی‌اطلاع است و مصرف‌کننده‌ها (consumers) از هم بی‌خبرند. معماری رویدادمحور به سادگی راهی برای دستیابی به پیوستگی سیستم<sup>۱</sup> بین سرویس‌ها درون یک سیستم است.

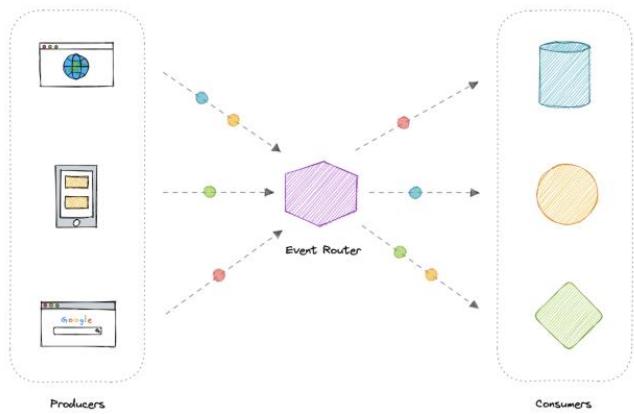
### رویداد چیست؟

یک رویداد (Event) یک نقطه‌ی داده است که نشان‌دهنده‌ی تغییرات وضعیت در یک سیستم است. این رویداد مشخص نمی‌کند چه اتفاقی باید بیفتد و چگونه این تغییر باید سیستم را اصلاح کند، بلکه فقط به سیستم در مورد یک تغییر وضعیت خاص اطلاع می‌دهد. هنگامی که یک کاربر عملی را انجام می‌دهد یک رویداد را راهاندازی (trigger) می‌کند.

### اجزا

معماری‌های رویدادمحور دارای سه جزء کلیدی هستند:

- تولیدکنندگان رویداد (Event producers): رویدادی را برای مسیریاب منتشر می‌کند.
- مسیریاب‌های رویداد (Event routers): رویدادها را فیلتر می‌کند و آن‌ها را به مصرف‌کننده‌ها هدایت می‌کند.
- مصرف‌کنندگان رویداد (Event consumers): از رویدادها برای اعمال تغییرات در سیستم استفاده می‌کند.



توجه: نقاط در نمودار نشان‌دهنده رویدادهای مختلف در سیستم هستند.

### الگوهای Event-Driven

راههای مختلفی برای پیاده‌سازی معماری رویدادمحور وجود دارد و اینکه از کدام روش استفاده می‌کنیم به مورد استفاده بستگی دارد، اما در اینجا چند نمونی رایج آورده شده است:

- Sagas •
- Publish-Subscribe •
- Event Sourcing •
- (CQRS) (Command and Query Responsibility Segregation) •

### مزایای Event-Driven

- جداسازی تولیدکنندگان و مصرف‌کنندگان پیام‌ها از یکدیگر
- مقیاس‌پذیری و توزیع‌پذیری بالا
- اضافه کردن آسان مصرف‌کنندگان جدید
- بهبود چابکی<sup>۱</sup>

### چالش‌های Event-Driven

<sup>۱</sup> agility

در اینجا برخی از چالش‌های معماری رویدادمحور آورده شده است:

- نیازمندی به تضمین تحويل پیام‌ها
- مدیریت خطاب دشوار است
- سیستم‌های رویدادمحور به‌طورکلی پیچیده هستند
- پردازش رویدادها دقیقاً یکبار و به ترتیب باید انجام شود
- موارد استفاده Event-Driven

در زیر برخی از موارد استفاده‌ی رایج که در آن‌ها معماری‌های رویدادمحور مفید هستند، آورده شده است:

- متادادتا و متريکها
- لاغ‌های سرور و امنیت
- ادغام سیستم‌های ناهمگن
- توزیع گسترده و پردازش موازی

#### مثال‌ها

در اینجا برخی از فناوری‌های پرکاربرد برای پیاده‌سازی معماری‌های رویدادمحور آورده شده است:

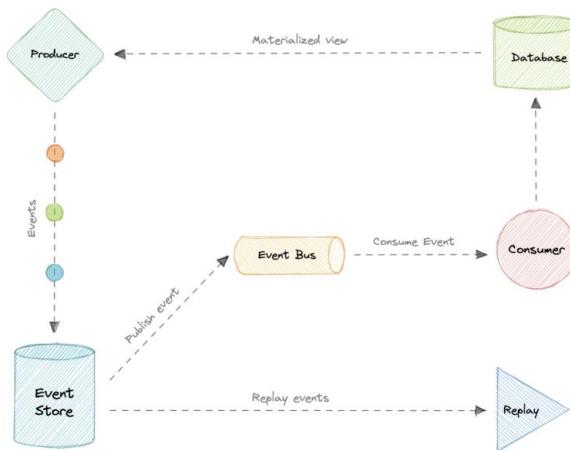
- NATS
- Apache Kafka
- Amazon EventBridge
- Amazon SNS
- Google PubSub

### منبع رویداد (Event Sourcing)

به‌جای ذخیره فقط وضعیت فعلی، از یک ذخیره‌سازی صرفاً افزودنی<sup>۱</sup> برای ثبت کل مجموعه رویدادهایی که اقدامات انجام شده روی داده‌ها در یک دامنه را توصیف می‌کنند، استفاده

<sup>۱</sup> append-only store یکی از ویژگی‌های ذخیره‌سازی داده‌های کامپیوتراست که داده‌های جدید را می‌توان به ذخیره‌سازی اضافه کرد، اما در جایی که داده‌های موجود تغییرناپذیر است.

کنید تا بتوان از این ذخیره‌ساز برای تحقق اشیاء دامنه<sup>۱</sup> استفاده کرد. این الگو می‌تواند وظایف را در دامنه‌های پیچیده با اجتناب از الزام به همگام‌سازی مدل داده و دامنه‌ی کسب‌وکار ساده کند و scalability، performance و پاسخگویی<sup>۲</sup> را بهبود بخشد؛ انسجام و یکپارچگی را برای داده‌های تراکنشی فراهم کند و ردیابی و تاریخچه کاملی را برای انجام اقدامات جبرانی احتمالی حفظ نماید.



### تفاوت منبع رویداد با معماری رویدادمحور<sup>۳</sup>

در واقع منبع رویداد یا Event Sourcing دائمًا با معماری رویدادمحور (EDA) اشتباه گرفته می‌شود. معماری رویدادمحور در مورد استفاده از رویدادها برای برقراری ارتباط بین مرزهای سرویس است. به طور کلی، از یک کارگزار پیام برای انتشار و مصرف رویدادها به صورت ناهمزمان در سایر مرزها استفاده می‌شود.

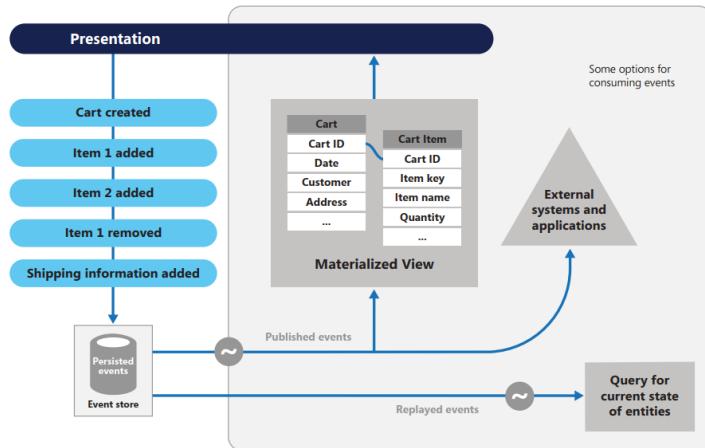
در حالی‌که Event Sourcing در مورد استفاده از رویدادها به عنوان یک وضعیت است که رویکردهٔ متفاوت برای ذخیره‌سازی داده است. به جای ذخیره وضعیت فعلی، ما قصد داریم

domain objects<sup>۱</sup>

responsiveness<sup>۲</sup>

Event-Driven Architecture - EDA<sup>۳</sup>

به جای آن رویدادها را ذخیره کنیم. همچنین، Event Sourcing یکی از چندین الگو برای پیاده‌سازی معماری رویدادمحور<sup>۱</sup> است.



### مزایای Event Sourcing

- مناسب برای گزارش داده‌های لحظه‌ای.
- ایده‌آل برای ایمنی در برابر خرابی، داده‌ها را می‌توان از ذخیره‌ساز رویداد بازسازی کرد.
- بسیار انعطاف‌پذیر، هر نوع پیامی قابل ذخیره است.
- روش ترجیحی برای دستیابی به عملکرد لایه‌های حسابرسی برای سیستم‌های با انطباق بالا.

### معایب Event Sourcing

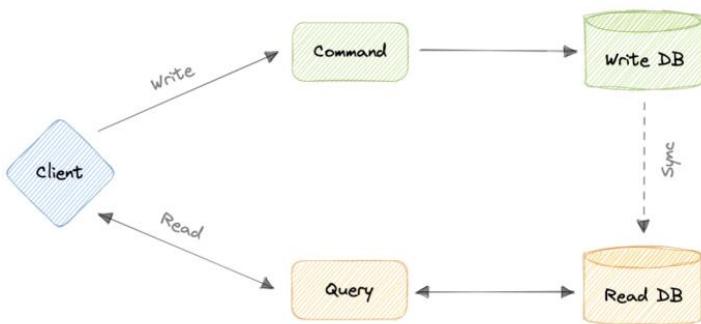
- نیاز به یک زیرساخت شبکه‌ی بسیار کارآمد دارد.
- نیاز به یک روش قابل اعتماد برای کنترل فرمات‌های پیام، مانند یک schema registry دارد.
- رویدادهای مختلف حاوی داده‌های متفاوتی خواهند بود.

<sup>۱</sup> event-driven architecture

## تفکیک مسئولیت فرمان و کوئری (CQRS):

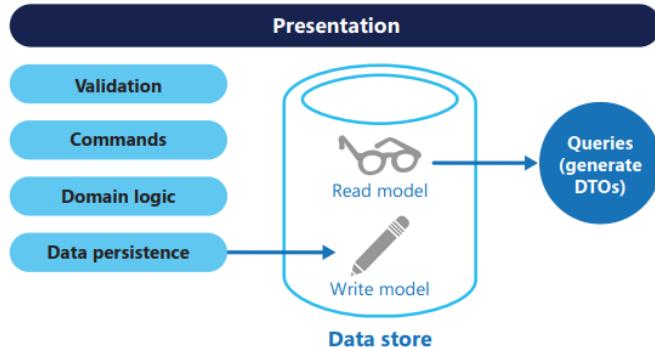
تفکیک مسئولیت فرمان و کوئری یا در اصطلاح CQRS<sup>1</sup>: عملیات خواندن داده را از عملیات بهروزرسانی داده با استفاده از رابطهای جداگانه تفکیک کنید. این الگو می‌تواند عملکرد، قابلیت مقیاس‌پذیری و امنیت را به حداقل برساند و از تکامل سیستم در طول زمان از طریق انعطاف‌پذیری بالاتر پشتیبانی کند و همچنین از بروز درگیری‌های ادغام ناشی از دستورات بهروزرسانی در سطح دامنه جلوگیری کند. به طور خلاصه CQRS یک الگوی معماری است که اقدامات یک سیستم را به دستورات (Command) و کوئری (Query) تقسیم می‌کند. این الگو برای اولین‌بار توسط Greg Young توصیف شد.

در CQRS، یک دستور یک دستورالعمل یا راهکاری برای انجام یک کار خاص است. این یک قصد برای تغییر چیزی است و هیچ مقداری را برنمی‌گرداند، فقط نشان‌دهنده‌ی موقوفیت یا شکست است و یک کوئری درخواستی برای اطلاعاتی است که وضعیت سیستم را تغییر نمی‌دهد یا هیچ اثر جانی ایجاد نمی‌کند.



اصل اساسی CQRS جداسازی دستورات و کوئری‌ها است. آن‌ها نقش‌های اساساً متفاوتی را در یک سیستم انجام می‌دهند و جداسازی آن‌ها به این معنی است که هر کدام را می‌توان در صورت نیاز بهینه کرد که سیستم‌های توزیع‌شده واقعاً می‌توانند از آن بهره‌مند شوند.

<sup>1</sup> Command and Query Responsibility Segregation



## Event Sourcing با CQRS

الگوی CQRS اغلب همراه با الگوی ذخیره‌سازی رویداد (Event Sourcing) استفاده می‌شود. سیستم‌های مبتنی بر CQRS از مدل‌های داده‌ی خواندن و نوشتمن جداگانه استفاده می‌کنند که هر کدام متناسب با وظایف مرتبط هستند و اغلب در مخازن فیزیکی جداگانه قرار دارند.

هنگامی که با الگوی ذخیره‌سازی رویداد استفاده می‌شود، مخزن رویدادها، مدل نوشتمن است و منبع رسمی اطلاعات است. مدل خواندن یک سیستم مبتنی بر CQRS و materialized view داده‌ها را به طور معمول به عنوان نماهای views بسیار غیرنرمال شده ارائه می‌دهد.

## مزایای CQRS

- امکان مقیاس‌بندی مستقل کارهای خواندن و نوشتمن.
- مقیاس‌بندی، بهینه‌سازی و تغییرات معماری آسان‌تر.
- نزدیک‌تر به منطق کسب‌وکار با پیوستگی ضعیف (loose coupling).
- برنامه می‌تواند در هنگام کوئری زدن از join‌های پیچیده اجتناب کند.
- مرزهای واضح بین رفتار سیستم.

## معایب CQRS

- طراحی برنامه‌ی پیچیده‌تر.
- خرابی پیام یا پیام‌های تکراری ممکن است رخ دهد.

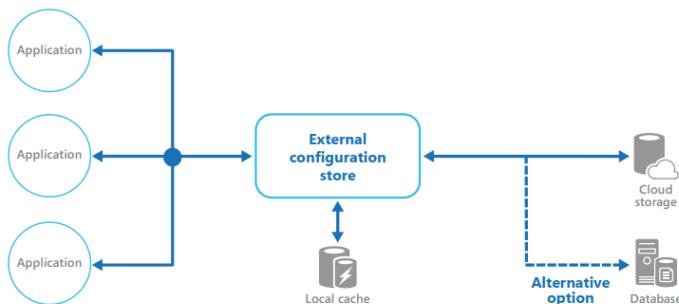
- برخورد با یکپارچگی تدریجی<sup>۱</sup> یک چالش است.
- افزایش تلاش‌ها برای نگهداری سیستم.

## موارد استفاده CQRS

عملکرد خواندن داده باید به طور جداگانه از عملکرد نوشتن داده بهینه‌سازی شود. انتظار می‌رود که سیستم در طول زمان تکامل یابد و ممکن است حاوی چندین نسخه از مدل باشد یا در جایی که قوانین business برنامه به طور مرتب تغییر می‌کنند. یکپارچه‌سازی با سایر سیستم‌ها، به ویژه در ترکیب با ذخیره‌سازی رویداد<sup>۲</sup>، جایی که خرابی موقت یک زیرسیستم نباید بر درسترس‌بودن سایر سیستم‌ها تأثیر بگذارد. امنیت بهتر برای اطمینان از اینکه تنها موجودیت‌های محدود شده و مناسب، عملیات نوشتن را روی داده‌ها انجام می‌دهند.

### مخزن پیکربندی خارجی (External Configuration Store) :

اطلاعات پیکربندی را از بسته‌ی استقرار برنامه<sup>۳</sup> به یک مکان مرکزی منتقل کنید. این الگو می‌تواند فرصت‌هایی را برای مدیریت و کنترل آسان‌تر داده‌های پیکربندی و اشتراک‌گذاری داده‌های پیکربندی در سراسر برنامه‌ها و نمونه‌های برنامه فراهم کند.



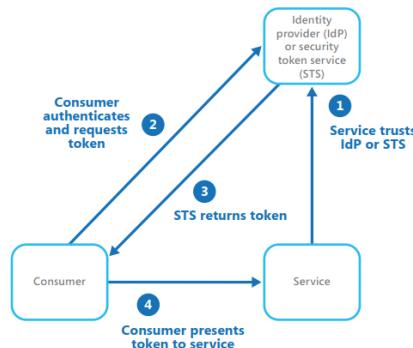
<sup>۱</sup> Eventual Consistency

<sup>۲</sup> event sourcing

<sup>۳</sup> application deployment package

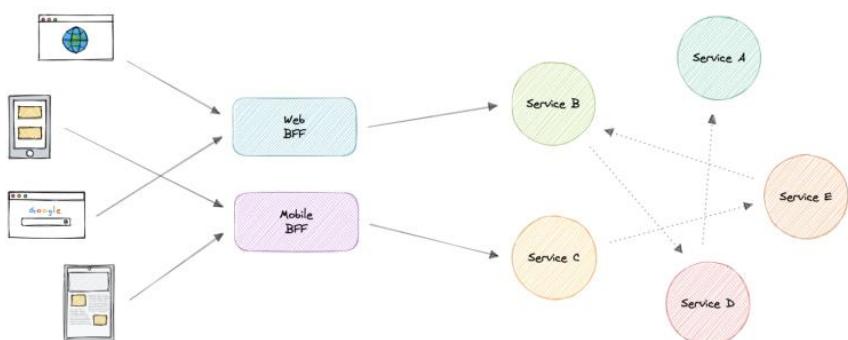
## هویت فدراتیو (Federated Identity)

احراز هویت را به یک ارائه‌دهندهٔ هویت خارجی واگذار کنید. این الگو می‌تواند توسعه را ساده کند، الزامات مدیریت کاربر را به حداقل برساند و تجربه کاربری برنامه را بهبود بخشد.



## الگوی بک‌اند برای فرانت‌اند (Backend For Frontend - BFF)

در الگوی بک‌اند برای فرانت‌اند (Backend For Frontend - BFF)، ما سرویس‌های Backend مجازابی برای استفاده توسط اپلیکیشن‌های فرانت‌اند یا رابط‌های کاربری خاص ایجاد می‌کنیم. این الگو زمانی مفید است که بخواهیم از سفارشی‌سازی یک بک‌اند واحد برای چندین رابط کاربری اجتناب کنیم. این الگو برای اولین‌بار توسط "Sam Newman" توصیف شد.

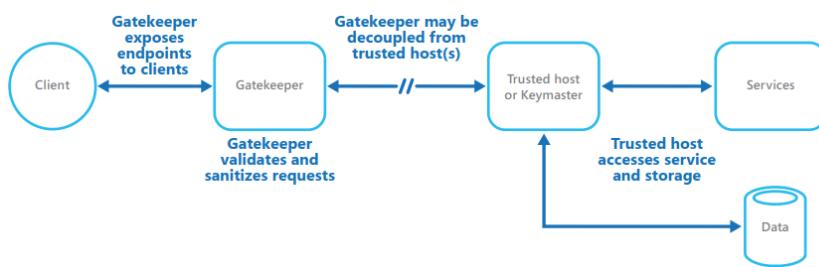


همچنین، گاهی اوقات خروجی داده‌های برگردنده شده توسط میکروسرویس‌ها به رابط کاربری، به فرمت دقیق یا فیلتر شده موردنیاز برای رابط کاربری نیست. برای حل این مشکل، فرانت‌اند باید منطقی برای تغییر قالب داده‌ها داشته باشد پس بنابراین ما می‌توانیم از BFF برای انتقال بخشی از این منطق به لایه میانی استفاده کنیم.

عملکرد اصلی الگوی بک‌اند برای فرانت‌اند، دریافت داده‌های موردنیاز از سرویس مناسب، فرمت‌دهی داده‌ها و ارسال آن به فرانت‌اند است. GraphQL به عنوان یک بک‌اند برای فرانت‌اند (BFF) عملکرد بسیار خوبی دارد.

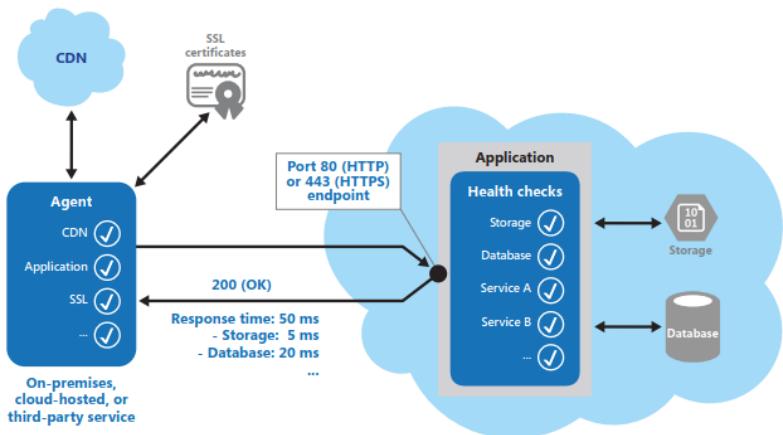
### دروازه‌بان (Gatekeeper)

از برنامه‌ها و سرویس‌ها با استفاده از یک نمونه‌ی host اختصاصی که به عنوان واسطه بین سرویس‌گیرنده‌ها و برنامه یا سرویس عمل می‌کند، اعتبارسنجی و پاکسازی درخواست‌ها را انجام می‌دهد و درخواست‌ها و داده‌ها را بین آن‌ها منتقل می‌کند، محافظت کنید. این الگو می‌تواند یک لایه امنیتی اضافی فراهم کند و سطح حمله‌ی سیستم را محدود نماید.



### Health Endpoint Monitoring

بررسی‌های عملکردی را در داخل یک برنامه پیاده‌سازی کنید که ابزارهای خارجی بتوانند در فواصل زمانی منظم از طریق endpointها در معرض دید به آن‌ها دسترسی داشته باشند. این الگو می‌تواند به تأیید عملکرد صحیح برنامه‌ها و سرویس‌ها کمک کند.



## جدول ایندکس (Index Table)

بر روی فیلدهای موجود در بانک‌های اطلاعاتی که به طور مکرر توسط معیارهای کوئری مرجع داده می‌شوند، ایندکس ایجاد کنید. این الگو می‌تواند با اجازه‌دادن به برنامه‌ها برای بازیابی سریع‌تر داده‌ها از بانک اطلاعاتی، عملکرد کوئری‌ها را بهبود بخشد.

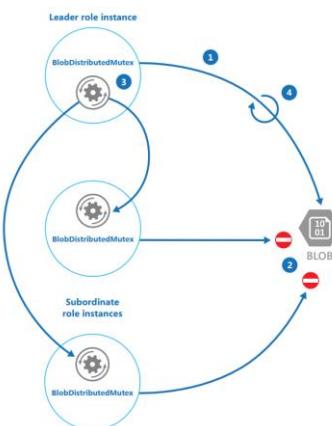
Fact Table

The diagram shows the relationship between an Index Table and a Fact Table. The Index Table (left) has a composite key (Town, LastName) and contains data for towns like Chicago, Portland, and Redmond, and last names like Clarke, Jones, Smith, Brown, Green. The Fact Table (right) has a primary key (Customer ID) and contains customer data with columns for LastName and Town. Arrows show the mapping from the composite key in the Index Table to the primary key in the Fact Table, indicating that multiple rows in the Index Table can map to a single row in the Fact Table.

Index Table		Fact Table	
Composite Key (Town, LastName)	Customer Reference (ID) and commonly queried data	Primary Key (Customer ID)	Customer Data
Chicago, Clarke	ID: 1000, ...	1	LastName: Smith, Town: Redmond, ...
Chicago, Jones	ID: 2, ...	2	LastName: Jones, Town: Seattle, ...
Chicago, Smith	ID: 3, ...	3	LastName: Robinson, Town: Portland, ...
...	...	4	LastName: Brown, Town: Redmond, ...
Portland, Clarke	ID: 4, ...	5	LastName: Smith, Town: Chicago, ...
Portland, Robinson	ID: 5, ...	6	LastName: Green, Town: Redmond, ...
Redmond, Brown	ID: 6, ...	7	LastName: Clarke, Town: Portland, ...
Redmond, Green	ID: 7, ...	8	LastName: Smith, Town: Redmond, ...
Redmond, Smith	ID: 8, ...	9	LastName: Jones, Town: Chicago, ...
Seattle, Jones	ID: 9, ...	...	...
...	...	1000	LastName: Clarke, Town: Chicago, ...
		...	...

### انتخاب رهبر (Leader Election)

با انتخاب یک نمونه به عنوان رهبر که مسئولیت مدیریت سایر نمونه‌ها را بر عهده می‌گیرد، اقدامات انجام شده توسط مجموعه‌ای از نمونه‌های تسک<sup>۱</sup> همکاری‌کننده در یک برنامه‌ی توزیع شده را هماهنگ<sup>۲</sup> کنید. این الگو می‌تواند به اطمینان حاصل کند که وظایف و تسک‌ها با یکدیگر تداخل نداشته باشند و باعث رقابت برای منابع مشترک نشوند یا ناخواسته با کاری که سایر نمونه‌های تسک انجام می‌دهند تداخل نکنند.



### دید تحقق یافته (Materialized View)

هنگامی که داده‌ها به گونه‌ای فرمت‌بندی شده‌اند که از عملیات کوئری موردنیاز پشتیبانی نمی‌کنند، Materialized View بر روی داده‌ها در یک یا چند بانک اطلاعاتی ایجاد کنید. این الگو می‌تواند از کوئری و استخراج داده کارآمد پشتیبانی کند و عملکرد برنامه را بهبود بخشد. در محاسبه، یک دید تحقق یافته یک شی پایگاه‌داده است که شامل نتایج یک کوئری است. برای مثال، ممکن است یک کپی محلی از داده‌های از راه دور باشد، یا ممکن است زیر مجموعه‌ای از ردیف‌ها و / یا ستون‌های یک جدول یا نتیجه اتصال باشد، یا ممکن است یک خلاصه با استفاده از یک تابع تجمعی باشد.

<sup>۱</sup> task instances

<sup>۲</sup> Coordinate

فرآیند ایجاد یک دیدگاه تحقق یافته گاهی تحقق یافته نامیده می‌شود. این شکلی از ذخیره نتایج یک کوئری است، مشابه با یادآوری ارزش یک عملکرد در زبان‌های کاربردی و گاهی اوقات به عنوان شکلی از پیش محاسبه توصیف می‌شود. همانند دیگر اشکال پیش محاسبه، کاربران پایگاهداده معمولاً از دیدگاه‌های مادیت شده برای دلایل عملکرد استفاده می‌کنند، به عنوان یک شکل بهینه‌سازی. در یک دید تحقق یافته، شاخص‌ها می‌توانند بر روی هر ستون ساخته شوند. در مقابل، در یک دیدگاه نرم‌افزار، به طور معمول تنها امکان بهره‌برداری از شاخص‌ها بر روی ستون‌هایی که مستقیماً از (یا دارای یک نقشه برای) ستون‌های ایندکس شده در جداول پایه می‌آیند، وجود دارد؛ اغلب این عملکرد به هیچ وجه ارائه نمی‌شود.

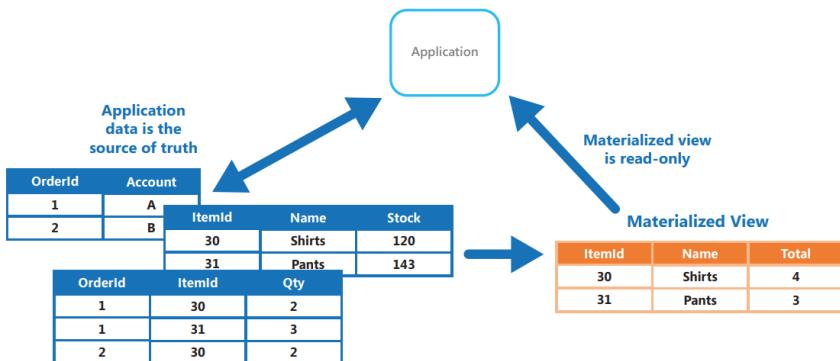


FIGURE 1  
The Materialized View pattern

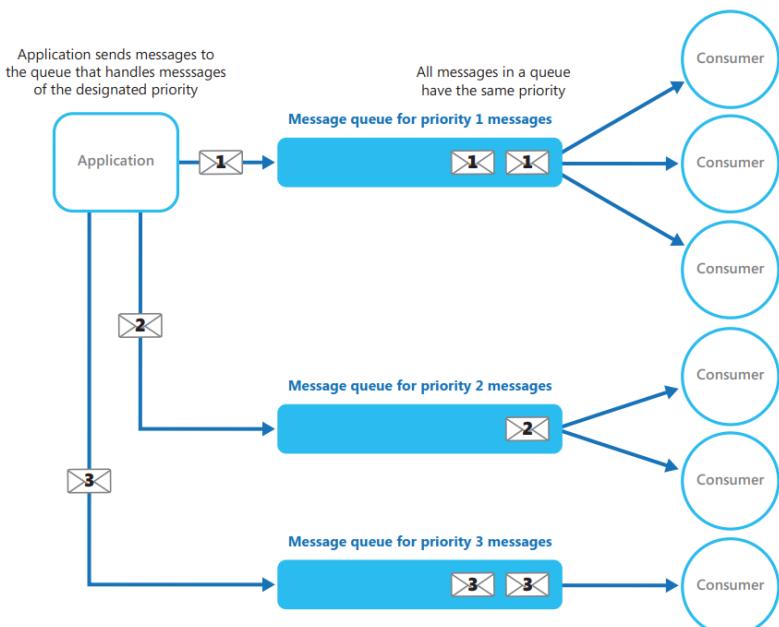
### لوله‌ها و فیلترها (Pipes and Filters):

یک task که پردازش پیچیده‌ای را انجام می‌دهد به مجموعه‌ای از عناصر گستته که قابل استفاده مجدد هستند، تفکیک کنید. این الگو می‌تواند با اجازه‌دادن به عناصر task performance، که پردازش را انجام می‌دهند تا به طور مستقل مستقر و مقیاس شوند، scalability و قابلیت استفاده مجدد را بهبود بخشد.



## صف اولویت دار (Priority Queue):

در خواستهای ارسال شده به سرویس‌ها را اولویت‌بندی کنید تا درخواست‌های با اولویت بالاتر نسبت به درخواست‌های با اولویت پایین‌تر سریع‌تر دریافت و پردازش شوند. این الگو در برنامه‌هایی که سطوح ضمانت خدمات متفاوتی را به انواع خاصی از سرویس‌گیرنده‌ها را ارائه می‌دهند، مفید است.



### ترازوی بارگذاری مبتنی بر صف (Queue-based Load Leveling):

از یک صف که به عنوان یک بافر بین یک task و سرویسی که فرخوانی می‌کند استفاده کنید تا بارهای سنگین گزرا را که ممکن است باعث خرابی سرویس یا timeout شدن کار شود را هموار و تعديل کند. این الگو می‌تواند به حداقل رساندن تأثیر اوج‌های تقاضا بر دردسترس بودن<sup>۱</sup> و پاسخگویی<sup>۲</sup> برای هر دو مورد تَسک و سرویس کمک کند.

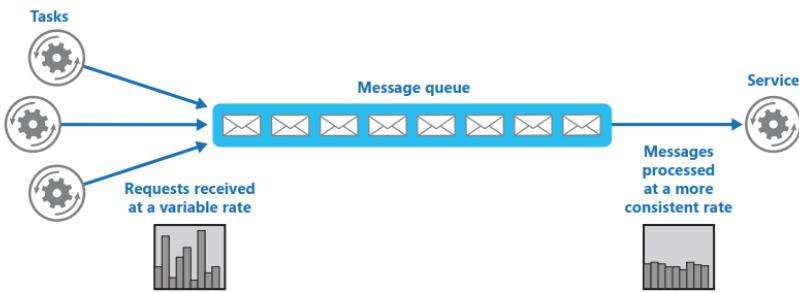


FIGURE 1  
Using a queue to level the load on a service

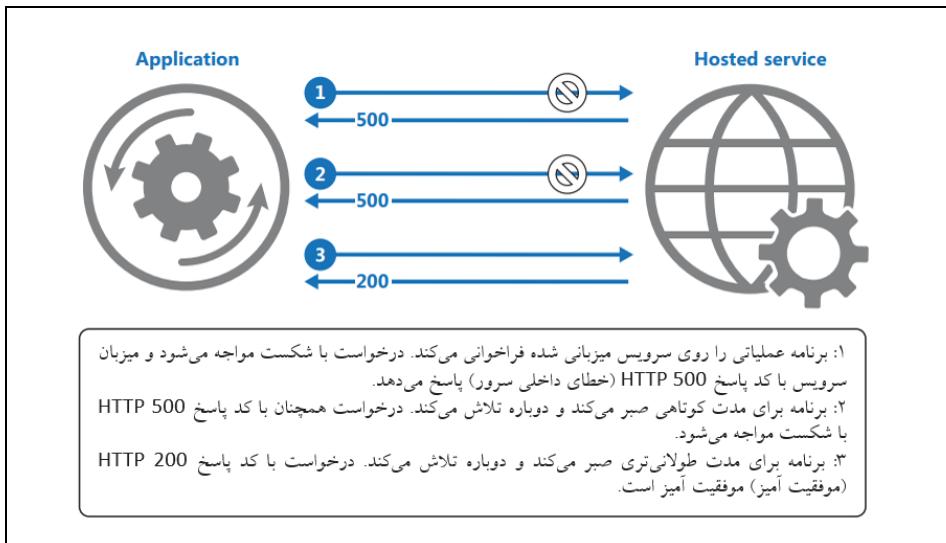
### تکرار مجدد (Retry):

با تکرار مجدد عملیات به منظور انتظار برگزرا بودن خرابی، یک برنامه را قادر سازید تا خرابی‌های موقت را هنگام اتصال به سرویس یا منع شبکه مدیریت کند. این الگو می‌تواند پایداری<sup>۳</sup> برنامه را بهبود بخشد.

<sup>۱</sup> availability

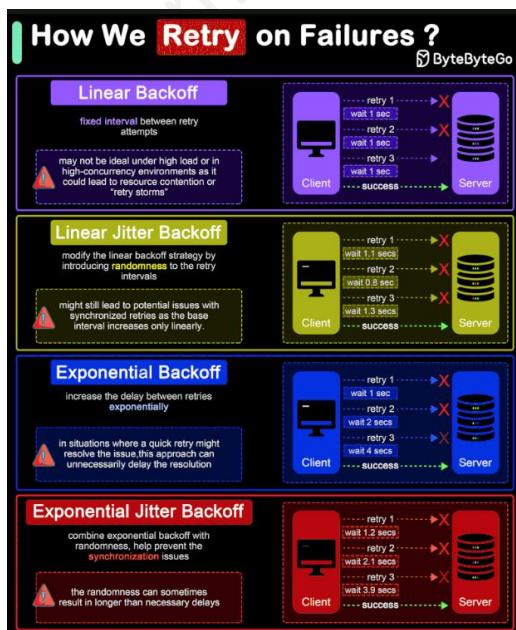
<sup>۲</sup> responsiveness

<sup>۳</sup> Stability



### چگونه در صورت وقوع خرابی retry کنیم؟

در سیستم های توزیع شده و برنامه های تحت شبکه ای، استراتژی های تکرار مجدد (retry) برای رسیدگی مؤثر به خطاهای گذرا و عدم ثبات شبکه بسیار مهم هستند. این نمودار ۴ استراتژی رایج برای تکرار مجدد را نشان می دهد.

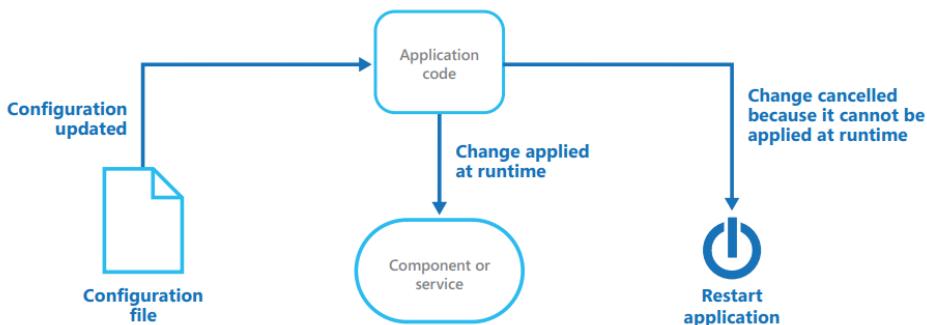


- تأخیر خطی معکوس (Linear Backoff): تأخیر خطی معکوس شامل انتظار برای یک فاصله‌ی ثابت که به طور تصاعدی افزایش می‌یابد در بین retry‌ها است. مزایا: ساده برای اجرا و درک است.
- معایب: ممکن است در شرایط بارگذاری بالا یا در محیط‌های با تراکم کار بالا ایده‌آل نباشد، زیرا می‌تواند منجر به رقابت برای منابع یا «طوفان‌های تکرار مجدد (retry storms)» شود.
- تأخیر با لرزش خطی (Linear Jitter Backoff): تأخیر با لرزش خطی، استراتژی تأخیر خطی معکوس را با معرفی تصادفی بودن به فواصل تکرار مجدد، تغییر می‌دهد. این استراتژی همچنان تأخیر را به صورت خطی افزایش می‌دهد، اما لرزش (jitter) تصادفی به هر فاصله اضافه می‌کند. مزایا: تصادفی بودن به پخش شدن تلاش‌های تکرار مجدد در طول زمان کمک می‌کند و شанс تکرارهای هم‌زمان در سراسر نمونه‌ها را کاهش می‌دهد.
- معایب: اگرچه بهتر از تأخیر خطی معکوس ساده است، این استراتژی ممکن است همچنان به دلیل افزایش خطی فاصله‌ی اصلی، منجر به مشکلات بالقوه با تکرارهای هم‌زمان شود.
- تأخیر نمایی معکوس (Exponential Backoff): تأخیر نمایی معکوس شامل افزایش تصاعدی تأخیر بین تکرارها است. این فاصله ممکن است از ۱ ثانیه شروع شود، سپس به ۲ ثانیه، ۴ ثانیه، ۸ ثانیه و غیره افزایش یابد که معمولاً تا حد اکثر تأخیر ادامه دارد. این رویکرد نسبت به تأخیر خطی معکوس، در فاصله‌گذاری تکرارها تهاجمی‌تر است. مزایا: به طور قابل توجهی بار روی سیستم و احتمال تصادم یا همپوشانی در تلاش‌های تکرار مجدد را کاهش می‌دهد و آن را برای محیط‌های با بارگذاری بالا مناسب می‌کند. معایب: در مواردی که تکرار سریع ممکن است مشکل را حل کند، این رویکرد می‌تواند به طور غیرضروری حل را به تعویق بیندازد.

- تأخیر نمایی با لرزش (**Exponential Jitter Backoff**): تأخیر نمایی با لرزش، تأخیر نمایی معکوس را با تصادفی بودن ترکیب می‌کند. پس از هر بار تکرار، فاصله‌ی تأخیر به صورت تصاعدی افزایش می‌یابد و سپس یک لرزش تصادفی اعمال می‌شود. لرزش می‌تواند افزایشی (اضافه کردن مقدار تصادفی به تأخیر نمایی) یا ضرب (ضرب کردن تأخیر نمایی در یک عامل تصادفی) باشد. مزایا: تمام مزایای تأخیر نمایی را به همراه مزیت کاهش بیشتر تصادفات تکرار به دلیل معرفی لرزش ارائه می‌دهد. معایب: تصادفی بودن گاهی اوقات می‌تواند منجر به تأخیرهای طولانی‌تر از حد ضروری شود، به خصوص اگر Jitter قابل توجه باشد.

### پیکربندی زمان اجرا (**Runtime Reconfiguration**)

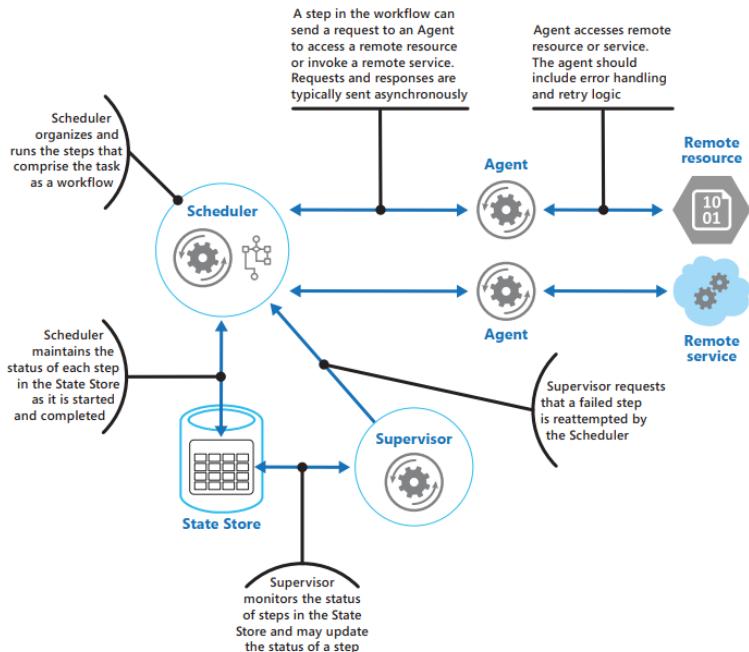
یک برنامه را به گونه‌ای طراحی کنید که بتوان آن را بدون نیاز به استقرار مجدد یا راهاندازی مجدد برنامه، بازپیکربندی کرد. این کار به حفظ درسترس بودن و به حداقل رساندن زمان خرابی کمک می‌کند.



### سرپرست عامل زمان‌بندی (**Scheduler Agent Supervisor**)

مجموعه‌ای از اقدامات را در سراسر مجموعه توزیع شده‌ای از سرویس‌ها و سایر منابع از remote هدایت کنید و تلاش کنید تا در صورت خرابی هر یک از این اقدامات، به طور واضح با خطاهای برخورد کنید یا اثرات کار انجام شده را اگر سیستم نمی‌تواند از یک خطأ

بازیابی شود را لغو کنید. این الگو می‌تواند با توانمندسازی سیستم توزیع شده برای بازیابی و تلاش مجدد اقداماتی که به دلیل استثنایات گذرا و خطاهای طولانی مدت و خرابی‌های فرایندی که با شکست مواجه می‌شوند، به یک سیستم توزیع شده قابلیت بازیابی اضافه کند.



### قطعه‌بندی (Sharding)

یک بانک اطلاعاتی را به مجموعه‌ای از پارتیشن‌های افقی به نام shard تقسیم کنید. این الگو می‌تواند قابلیت ارتقا را هنگام ذخیره و دسترسی به حجم زیادی از داده‌ها بهبود بخشد.

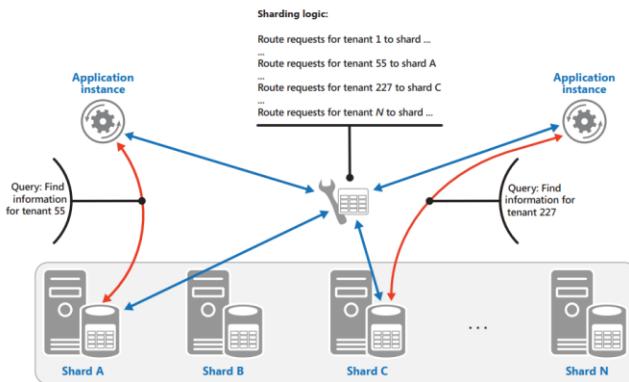
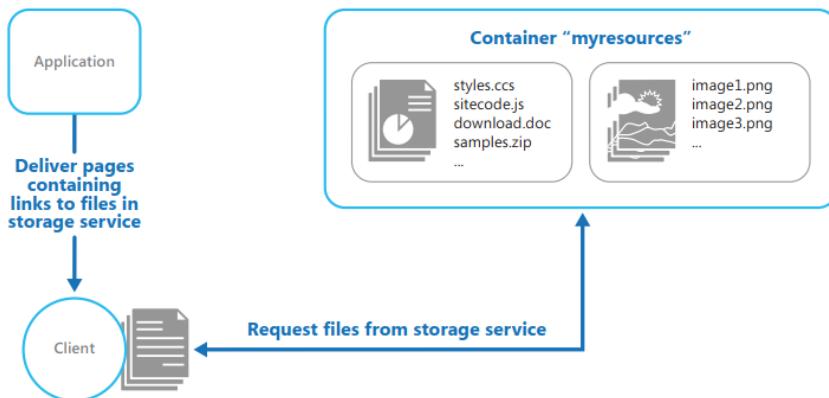


FIGURE 1  
Sharding tenant data based on tenant IDs

### میزبانی محتوا استاتیک (Static Content Hosting)

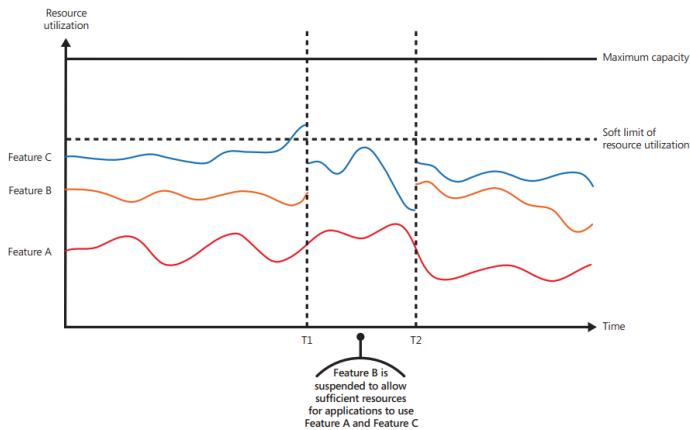
محتوا استاتیک را در یک سرویس ذخیره‌سازی ابری (cloud-based storage) مستقر کنید که می‌تواند این محتوا را مستقیماً به کاربر تحویل دهد. این الگو می‌تواند نیاز به نمونه‌های محاسباتی بالقوه گران قیمت را کاهش دهد.



### محدودسازی (Throttling)

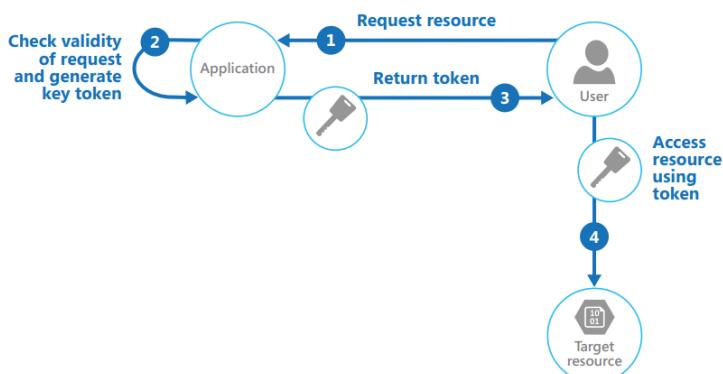
مصرف منابع را که توسط یک نمونه از یک برنامه، یک مستأجر یا کل سرویس استفاده می‌شود، کنترل کنید. این الگو حتی زمانی که افزایش تقاضا بار شدیدی را بر منابع اعمال

می‌کند، به سیستم اجازه می‌دهد تا به کار خود ادامه دهد و به توافقنامه‌های سطح خدمات (SLA) دست یابد.



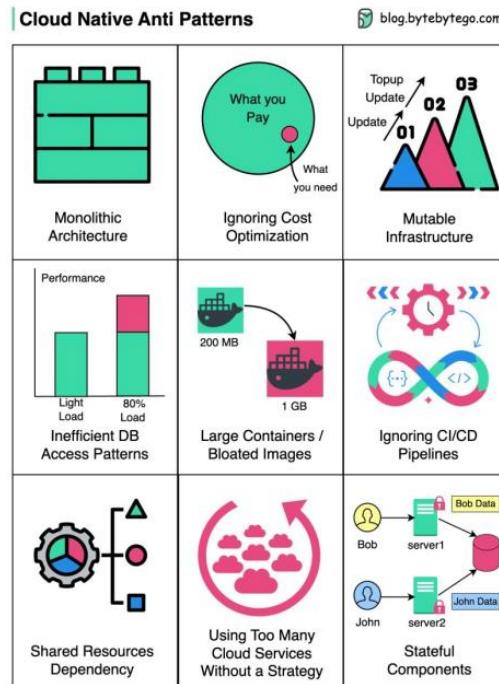
## :Valet Key بورسی

برای اینکه عملیات انتقال داده را از کد برنامه خارج کنید باید از توکن یا کلیدی استفاده کنید که به سرویس‌گیرنده‌ها دسترسی مستقیم و محدودی به یک منبع یا سرویس خاص را ارائه می‌دهد. این الگو به طور خاص در برنامه‌هایی که از سیستم‌های ذخیره‌سازی یا صفاتی میزبانی شده در ابر استفاده می‌کنند مناسب است و می‌تواند هزینه را به حداقل برساند و قابلیت ارتقا و عملکرد را به حداقل بررساند.



## ضد الگوهای Cloud Native

با آگاهی از این آنتیپترن‌ها و پیروی از بهترین الگوهای cloud-native، می‌توانید برنامه‌های cloud-native قوی‌تر، مقیاس‌پذیرتر و مقرون‌به‌صرفه‌تر طراحی، ساخته و راهاندازی کنید.



### ۱. معماری یکپارچه (Monolithic Architecture)

یک برنامه بزرگ و کاملاً کوپل شده که روی ابر اجرا می‌شود و مانع مقیاس‌پذیری و چابکی می‌شود

### ۲. نادیده‌گرفتن هزینه‌ها:

سرویس‌های ابری می‌توانند گران باشند و عدم بهینه‌سازی هزینه‌ها می‌تواند منجر به افزایش هزینه‌ها شود

### ۳. زیرساخت تغییرپذیر (Mutable Infrastructure)

- اجزای زیرساخت باید به عنوان یکبار مصرف در نظر گرفته شوند و هرگز در جای خود تغییر نکنند.

- عدم پذیرش این رویکرد می‌تواند منجر به انحراف پیکربندی، افزایش نگهداری و کاهش قابلیت اطمینان شود.

#### ۴. الگوهای دسترسی به پایگاهداده ناکارآمد:

استفاده از کوئری‌های بیش از حد پیچیده یا عدم وجود ایندکس پایگاهداده می‌تواند منجر به کاهش عملکرد و گلوگاه‌های پایگاهداده شود

#### ۵. کانتینرهای بزرگ یا Image‌های حجمی:

ایجاد کانتینرهای بزرگ یا استفاده از Image‌های حجمی می‌تواند زمان استقرار را افزایش دهد، منابع بیشتری مصرف کند و مقیاس‌بندی برنامه را آهسته کند

#### ۶. نادیده‌گرفتن CI/CD Pipeline:

استقرارها دستی و مستعد خطای شوند و سرعت و فرکانس انتشار نرم‌افزار را کاهش می‌دهند

#### ۷. وابستگی به منابع مشترک:

برنامه‌ایی که به منابع مشترک مانند پایگاهداده‌ها وابسته هستند می‌توانند ایجاد رقابت و گلوگاه کنند و بر عملکرد کلی تأثیر بگذارند

#### ۸. استفاده از بیش از حد سرویس‌های ابری بدون استراتژی:

در حالی که ارائه‌دهندگان ابری طیف گسترده‌ای از خدمات را ارائه می‌دهند، استفاده بیش از حد از آن‌ها بدون یک استراتژی واضح می‌تواند پیچیدگی ایجاد کند و مدیریت برنامه را دشوارتر کند.

#### ۹. اجزای Stateful:

اتکا به حالت پایدار<sup>۱</sup> در برنامه‌ها می‌تواند پیچیدگی ایجاد کند، مقیاس‌پذیری را کاهش دهد و تحمل خطا را محدود کند.

<sup>۱</sup> persistent state

## بازیابی فاجعه (Disaster Recovery)

بازیابی فاجعه یا در اختصار DR فرایندی برای بازیابی دسترسی و عملکرد زیرساخت پس از وقایعی مانند بلایای طبیعی، حملات سایبری یا حتی اختلالات تجاری است. بازیابی فاجعه بر تکثیر<sup>۱</sup> داده‌ها و پردازش رایانه‌ای در یک مکان خارج از محل که تحت تأثیر فاجعه قرار نگرفته، تکیه می‌کند. هنگامی که سرورها به دلیل فاجعه از کار می‌افتدند، یک کسب‌وکار باید داده‌های ازدست‌رفته را از مکان دوم که در آن از داده‌ها پشتیبان تهیه شده بازیابی کند. در حالت ایده‌آل، یک سازمان می‌تواند پردازش رایانه‌ای خود را نیز به آن مکان از راه دور منتقل کند تا عملیات خود را ادامه دهد.

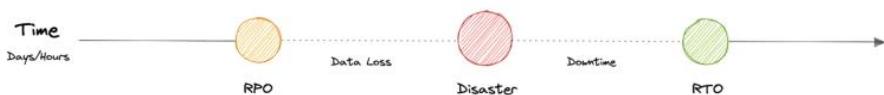
بازیابی فاجعه اغلب در مصاحبه‌های طراحی سیستم به طور فعال مورد بحث قرار نمی‌گیرد، اما داشتن درک اولیه از این موضوع مهم است. شما می‌توانید در مورد بازیابی فاجعه از AWS Well-Architected Framework اطلاعات بیشتری کسب کنید.

بازیابی فاجعه می‌تواند مزایای زیر را داشته باشد:

- به حداقل رساندن وقفه و خرابی
- محدود کردن خسارت
- بازیابی سریع
- حفظ بهتر کاربران

## اصطلاحات DR

بیایید برخی از اصطلاحات مهم مرتبط با بازیابی فاجعه را بررسی کنیم:



<sup>۱</sup> replication

### **RTO (Recovery Time Objective)**

هدف زمان بازیابی (RTO) حداقل تأخیر قابل قبول بین قطع سرویس و بازیابی سرویس است. این امر تعیین می‌کند که چه زمانی یک پنجره زمانی قابل قبول در نظر گرفته می‌شود که سرویس در دسترس نیست.

### **RPO (Recovery Point Objective)**

هدف نقطه بازیابی (RPO) حداقل زمان قابل قبول از آخرین نقطه بازیابی داده است. این امر تعیین می‌کند که چه میزان از دست رفتن داده‌ها بین آخرین نقطه بازیابی و قطع سرویس قابل قبول است.

### **استراتژی‌ها**

انواع مختلفی از استراتژی‌های بازیابی فاجعه (DR) می‌توانند بخشی از یک برنامه بازیابی فاجعه باشند.

#### **پشتیبان‌گیری (Back-up)**

این ساده‌ترین نوع بازیابی فاجعه است و شامل ذخیره‌سازی داده‌ها در خارج از محل یا روی یک درایو قابل جابه‌جای است.

#### **Cold Site**

در این نوع بازیابی فاجعه، یک سازمان زیرساخت اولیه را در یک سایت دوم راهاندازی می‌کند.

#### **Hot Site**

یک سایت گرم (Hot Site) در همه حال، از کپی‌های به روز اطلاعات نگهداری می‌کند. راهاندازی سایت‌های گرم زمان‌بر است و نسبت به سایت‌های سرد هزینه‌ی بیشتری دارد، اما به طور چشمگیری زمان خرابی را کاهش می‌دهد.

## (Service Discovery) کشف سرویس

کشف سرویس یا Service Discovery فرایند شناسایی سرویس‌ها درون یک شبکه‌ی کامپیوتری است. پروتکل کشف سرویس (SDP<sup>1</sup>) یک استاندارد شبکه‌ای است که کشف شبکه‌ها را با شناسایی منابع انجام می‌دهد.

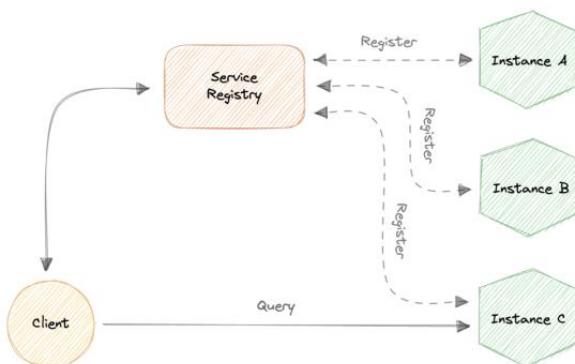
### چرا به Service Discovery نیاز داریم؟

در یک اپلیکیشن یکپارچه<sup>2</sup>، سرویس‌ها از طریق متدها یا فراخوانی‌های رویه‌ای در سطح زبان، یکدیگر را فراخوانی می‌کنند. با این حال، برنامه‌های مدرن مبتنی بر میکروسرویس معمولاً در محیط‌های مجازی‌سازی شده یا کانتینری شده اجرا می‌شوند که در آن‌ها تعداد نمونه‌های یک سرویس و مکان‌های آن‌ها به صورت پویا تغییر می‌کند. در نتیجه، ما به مکانیسمی نیاز داریم که به سرویس گیرنده‌های سرویس اجازه دهد تا درخواست‌هایی را به مجموعه‌ای از نمونه‌های سرویس نایاب‌دار و پویا ارسال کنند.

### پیاده‌سازی‌ها

دو الگوی اصلی کشف سرویس وجود دارد:

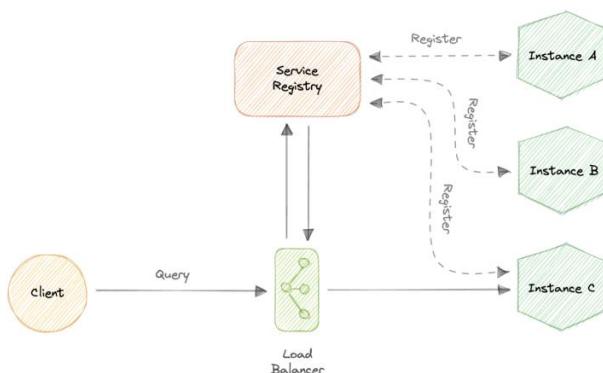
Client-side discovery



Service Discovery Protocol<sup>1</sup>  
Monolithic application<sup>2</sup>

در این رویکرد، سرویس‌گیرنده با پرس‌وجو از یک رجیستری سرویس، مکان سرویس دیگری را به دست می‌آورد. این رجیستری مسئول مدیریت و ذخیره‌سازی مکان‌های شبکه‌ی تمام سرویس‌ها است.

### Server-side discovery



در این رویکرد، ما از یک جزء واسطه مانند توزیع‌کننده بار<sup>۱</sup> استفاده می‌کنیم. سرویس‌گیرنده از طریق یک توزیع‌کننده بار درخواستی را به سرویس ارسال می‌کند و سپس توزیع‌کننده بار درخواست را به یک نمونه‌ی سرویس در دسترس ارسال می‌کند.

### Service Registry

یک رجیستری سرویس اساساً یک پایگاهداده است که حاوی مکان‌های شبکه‌ای نمونه‌های سرویس است که سرویس‌گیرنده‌گان می‌توانند به آن‌ها دسترسی پیدا کنند. یک رجیستری سرویس باید به شدت در دسترس<sup>۲</sup> باشد و به طور مداوم به روزرسانی شود.

### Service Registration

ما همچنین به روشی برای بدست‌آوردن اطلاعات سرویس که اغلب به عنوان ثبت سرویس شناخته می‌شود، نیاز داریم. باید به دو رویکرد احتمالی ثبت سرویس نگاه کنیم:

<sup>۱</sup> load balancer

<sup>۲</sup> highly available

### ثبت خودکار (Self-Registration)

هنگام استفاده از مدل ثبت خودکار، یک نمونه‌ی سرویس مسئول ثبت و لغو ثبت خود در رجیستری سرویس است. علاوه بر این، در صورت لزوم، یک نمونه سرویس درخواست‌های ضربان قلب<sup>۱</sup> (heartbeat) ارسال می‌کند تا ثبت‌نام خود را فعال نگه دارد.

### ثبت شخص ثالث (Third-party Registration)

با نظرسنجی از محیط استقرار یا مشترک شدن در رویدادها، رجیستری، تغییرات در نمونه‌های در حال اجرا را ردیابی می‌کند. هنگامی که یک نمونه‌ی سرویس جدید در دسترس را تشخیص می‌دهد، آن را در پایگاهداده خود ثبت می‌کند. رجیستری سرویس همچنین نمونه‌های سرویس خاتمه‌یافته را لغو ثبت می‌کند.

### Service mesh

ارتباط سرویس به سرویس در یک برنامه‌ی توزیع شده ضروری است، اما مسیریابی این ارتباط هم در داخل و هم بین خوشه‌های برنامه با افزایش تعداد سرویس‌ها پیچیده‌تر می‌شود. سرویس مش، ارتباط مدیریت شده، قابل مشاهده و ایمن بین سرویس‌های مجزا را امکان‌پذیر می‌کند. این کار به کمک یک پروتکل service discovery برای شناسایی سرویس‌ها کار می‌کند. Envoy و Istio برخی از متداول‌ترین فناوری‌های سرویس mesh هستند.

### مثال

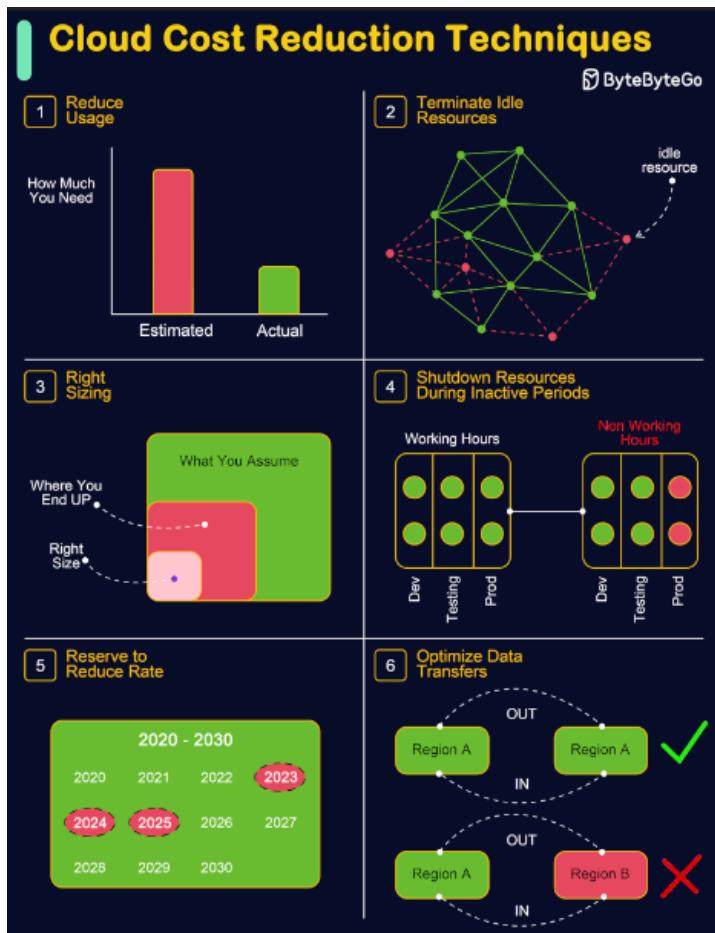
در اینجا برخی از ابزارهای زیرساخت service discovery راچ آورده شده است:

etcdb	•
Consul	•
Apache Thrift	•
Apache Zookeeper	•

<sup>۱</sup> در علم کامپیوتر، ضربان قلب یک سیگنال دوره‌ای است که توسط سخت‌افزار یا نرم‌افزار برای نشان دادن عملکرد عادی یا همگام‌سازی سایر بخش‌های یک سیستم کامپیوتری تولید می‌شود.

## کاهش هزینه‌های ابری

هزینه غیرمنطقی رایانش ابری، بزرگ‌ترین چالش بسیاری از سازمان‌ها در مواجهه با پیچیدگی‌های رایانش ابری است. مدیریت کارآمد این هزینه‌ها برای بهینه‌سازی استفاده از ابر و حفظ سلامت مالی حیاتی است. تکنیک‌های زیر می‌توانند به کسب‌وکارها در کنترل و به حداقل رساندن هزینه‌های ابری خود کمک کنند.



۱. کاهش مصرف:

حجم و مقیاس منابع را برای اطمینان از کارایی بدون خدشه‌دار کردن عملکرد برنامه‌ها (به عنوان مثال، کوچک‌کردن نمونه‌ها، به حداقل رساندن فضای ذخیره‌سازی، ادغام سرویس‌ها) تنظیم کنید.

## ۲. خاتمه دادن به منابع بلا استفاده:

منابعی را که به طور فعال مورد استفاده قرار نمی‌گیرد، مانند نمونه‌های غیرفعال، پایگاه‌های داده یا واحدهای ذخیره‌سازی را پیدا کرده و حذف کنید.

## ۳. تغییر اندازه مناسب:

اندازه نمونه‌ها را برای برآوردن نیازهای برنامه‌های خود تنظیم کنید تا از عدم استفاده یا استفاده بیش از حد اطمینان حاصل شود.

## ۴. خاموش کردن منابع در زمان‌های کم‌کار:

برای خاموش کردن منابع غیرضروری در زمان عدم استفاده، به خصوص در دوره‌های فعالیت کم، مکانیزم‌های خودکار یا زمان‌بندی تعیین کنید.

## ۵. رزرو برای کاهش نرخ:

مدل‌های قیمت‌گذاری به صرفه مانند نمونه‌های رزرو یا برنامه‌های صرفه‌جویی را که با نیازهای خاص حجم کاری شما مطابقت دارند، اتخاذ کنید.

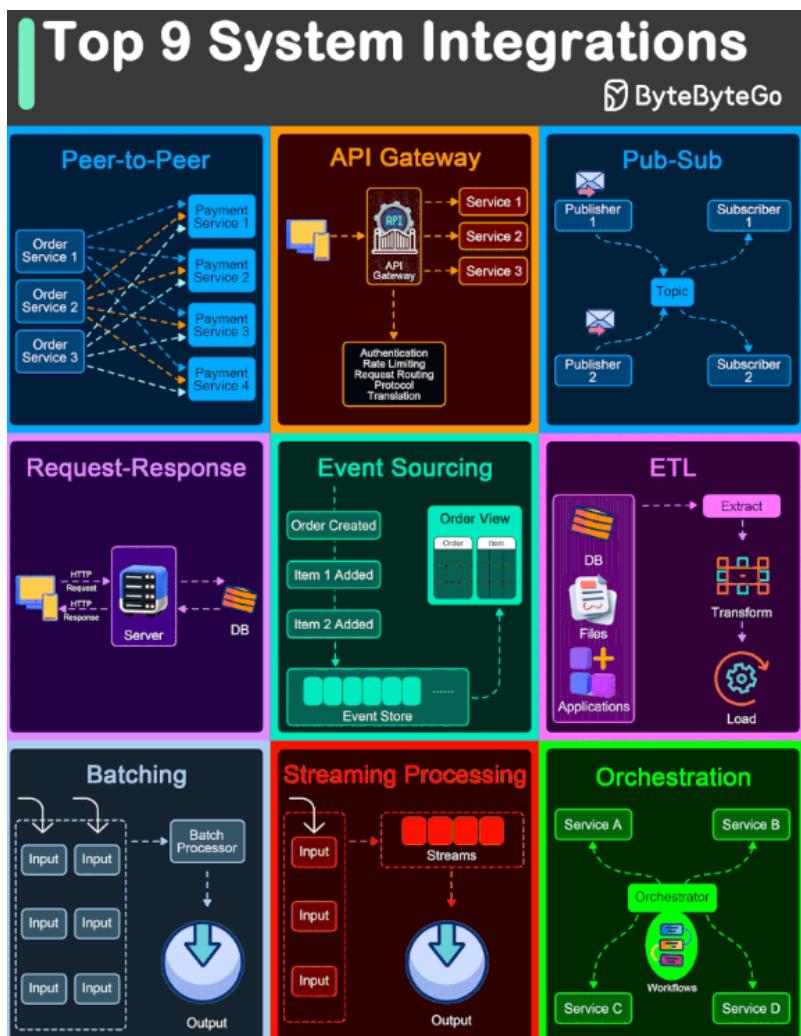
نکته اضافی: برای صرفه‌جویی بیشتر در هزینه، استفاده از Spot Instances و گزینه‌های ذخیره‌سازی سطح پایین‌تر را در نظر بگیرید.

## ۶. بهینه‌سازی انتقال داده:

از روش‌هایی مانند فشرده‌سازی داده‌ها و شبکه‌های تحویل محتوا (CDN) برای کاهش هزینه‌های پهنای باند استفاده کنید و منابع را به صورت استراتژیک برای کاهش هزینه‌های انتقال داده قرار دهید و بر انتقالات درون منطقه تمرکز کنید.

## ۹ الگوی برتر معماری برای جریان داده و ارتباط

الگوهای معماری، رویکردهایی برای طراحی و ساختار سیستم‌های نرم‌افزاری هستند که باعث بهبود قابلیت نگهداری، مقیاس‌پذیری، انعطاف‌پذیری و درک کلی سیستم می‌شوند. این الگوها راه حل‌هایی از پیش طراحی شده‌ای برای مسائل رایج در معماری نرم‌افزار ارائه می‌دهند. در این مقاله، ۹ الگوی برتر معماری برای جریان داده و ارتباط را بررسی می‌کنیم.



## ۱. Peer-to-Peer

الگوی همتا به همتا شامل ارتباط مستقیم بین دو جزء بدون نیاز به یک هماهنگ‌کننده مرکزی است. در این الگو، هر دو جزء می‌توانند همزمان در نقش ارسال‌کننده و گیرنده عمل کنند. به عنوان مثال، شبکه‌های اشتراک فایل، بیت تورنت و بلاکچین از این الگو استفاده می‌کنند.

## ۲. API Gateway

دروازه API به عنوان یک نقطه ورود واحد برای تمامی درخواست‌های سمت کاربر (client) به سرویس‌های پس‌زمینه (backend) یک برنامه عمل می‌کند. دروازه API مسئولیت مسیریابی درخواست‌ها به سرویس‌های مناسب، اعمال امنیت و مانیتورینگ را بر عهده دارد.

## ۳. انتشار - مشترک (Pub-Sub)

الگوی انتشار - مشترک، تولیدکنندگان پیام (publishers) را از مصرف‌کنندگان<sup>۱</sup> پیام (مشترکان) از طریق یک کارگزار پیام (message broker) تفکیک می‌کند. ناشران پیام‌های خود را به broker ارسال می‌کنند و مشترکان<sup>۲</sup> بدون نیاز به دانستن اینکه چه کسی پیام‌ها را منتشر کرده است، می‌توانند در صفحه دریافت پیام‌ها قرار بگیرند و آن‌ها را دریافت کنند.

## ۴. درخواست - پاسخ (Request-Response)

این یکی از اساسی‌ترین الگوهای یکپارچه‌سازی است. در این الگو، یک سمت کاربر (client) درخواستی را به یک سرور ارسال می‌کند و منتظر پاسخ می‌ماند. سرویس وب ساده‌ای که داده‌ها را بر اساس درخواست کاربر برمی‌گرداند، نمونه‌ای از این الگو است.

## ۵. منبع رویداد (Event Sourcing)

Event Sourcing شامل ذخیره‌سازی تغییرات وضعیت یک برنامه به عنوان یک توالی از رویدادها است. با داشتن یک لیست از تمام رویدادهایی که رخداده است، می‌توان در هر زمان وضعیت فعلی برنامه را بازسازی کرد. این الگو برای ردیابی تاریخچه تغییرات و قابلیت بازیابی اطلاعات مفید است.

<sup>۱</sup> consumers

<sup>۲</sup> subscribers

#### ۶. استخراج، تبدیل، بارگذاری (ETL):

ETL یک الگوی یکپارچه‌سازی داده است که برای جمع‌آوری داده از منابع مختلف، تبدیل آن به یک قالب ساختاریافته و بارگذاری آن در یک پایگاهداده مقصد استفاده می‌شود. این الگو برای انتقال داده‌ها از سیستم‌های قدیمی به سیستم‌های جدید یا ادغام داده‌ها از منابع مختلف مفید است.

#### ۷. دسته‌بندی (Batching):

دسته‌بندی شامل جمع‌آوری داده در طول یک دوره زمانی یا تا زمان رسیدن به یک آستانه خاص قبل از پردازش آن به عنوان یک گروه واحد است. این الگو باعث بهینه‌سازی مصرف منابع و بهبود کارایی پردازش داده‌های حجمی می‌شود.

#### ۸. پردازش جریان (Streaming Processing):

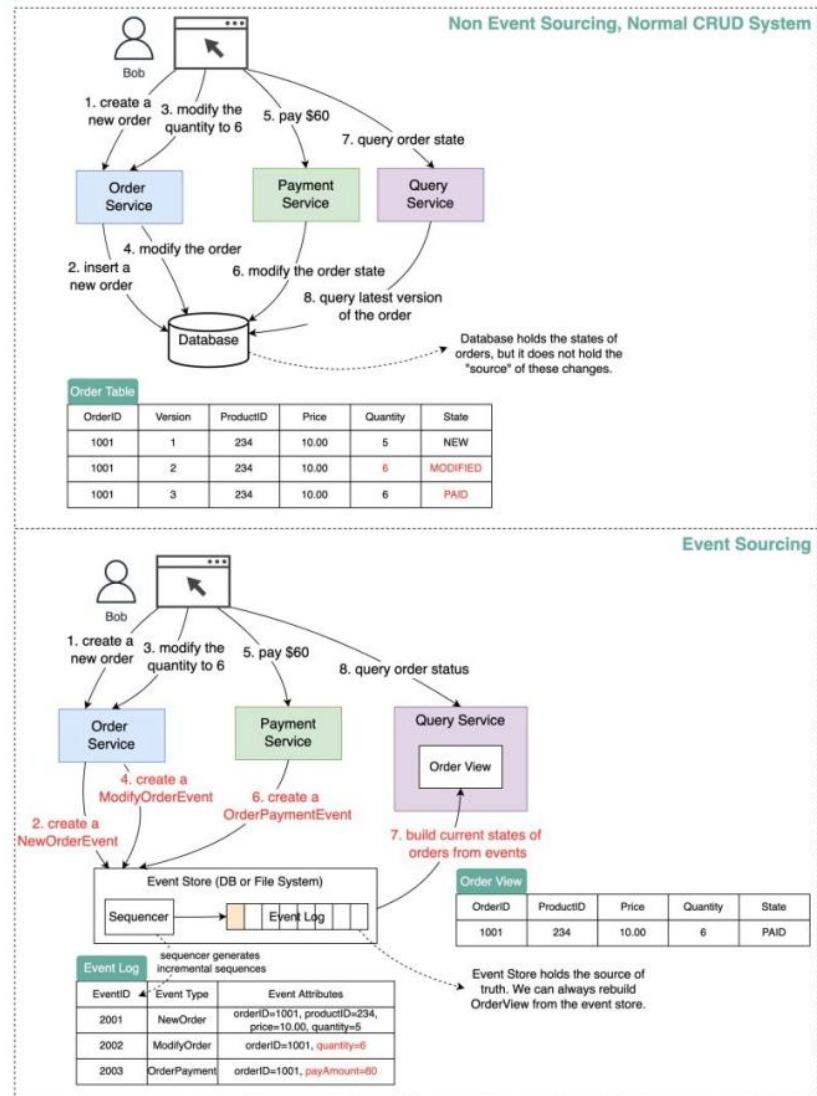
پردازش جریانی امکان دریافت، پردازش و تجزیه و تحلیل مداوم جریان‌های داده<sup>۱</sup> در زمان واقعی را فراهم می‌کند. این الگو برای ستابیلیتی مانند تجزیه و تحلیل داده‌های حسگرها، پردازش تراکنش‌های مالی و تجزیه و تحلیل داده‌های رسانه‌های اجتماعی مفید است.

#### ۹. هماهنگ‌کننده (Orchestration):

هماهنگ‌کننده شامل یک هماهنگ‌کننده مرکزی (یک ارکسترатор) است که تعاملات بین اجزای توزیع شده یا سرویس‌ها را برای دستیابی به یک گردش کار یا فرایند تجاری مدیریت می‌کند. ارکسترатор مسئولیت تجزیه یک فرایند پیچیده به کارهای کوچک‌تر، اختصاص دادن آنها به اجزای مناسب و اطمینان از اجرای آنها در ترتیب صحیح و مناسب را بر عهده دارد.

## چیست؟ چرا با CRUD متفاوت است؟ Event Sourcing

نمودار زیر مقایسه‌ای بین طراحی سیستم CRUD معمولی و طراحی سیستم Event Sourcing را نشان می‌دهد. ما از یک سیستم تجارت الکترونیک که قابلیت ثبت سفارش و پرداخت را دارد برای نمایش نحوه عملکرد Event Sourcing استفاده می‌کنیم.



الگوی Event Sourcing برای طراحی سیستمی با قابلیت تکرارپذیری<sup>۱</sup> استفاده می‌شود. این رویکرد، فلسفه طراحی سیستم‌های معمولی را تغییر می‌دهد. اما چگونه کار می‌کند؟ به جای ذخیره وضعیت سفارش در پایگاه داده، طراحی Event Sourcing رویدادهایی را که منجر Event Store به تغییر وضعیت می‌شوند در Event Store (ذخیره‌ساز رویداد) ثبت می‌کند. Event Store یک لاغ فقط الحقیقی<sup>۲</sup> است. رویدادها باید با اعداد افزایشی ترتیب‌بندی شوند تا ترتیب وقوع آن‌ها تضمین شود. وضعیت‌های سفارش از رویدادها بازسازی می‌شوند و در OrderView (نمایشگر سفارش) نگهداری می‌شوند. اگر OrderView از کار بیفتاد، همیشه می‌توانیم به Event Store که منبع حقیقت است برای بازیابی وضعیت‌های سفارش اعتماد کنیم.

بیایید مراحل را با جزئیات بررسی کنیم.

### ● Event Sourcing

مراحل ۱ و ۲: باب می‌خواهد یک محصول بخرد. سفارش ایجاد و در پایگاه داده درج می‌شود.

مراحل ۳ و ۴: باب می‌خواهد مقدار را از ۵ به ۶ تغییر دهد. سفارش با یک وضعیت جدید اصلاح می‌شود.

مراحل ۵ و ۶: باب مبلغ ۶۰ دلار برای سفارش پرداخت می‌کند. سفارش تکمیل می‌شود و وضعیت به «پرداخت شده» تغییر می‌ابد.

مراحل ۷ و ۸ باب آخرین وضعیت سفارش را استعلام می‌کند. سرویس پرس‌وجو وضعیت را از پایگاه داده بازیابی می‌کند.

### ● Event Sourcing

determinism<sup>۱</sup>  
append-only<sup>۲</sup>

مراحل ۱ و ۲: باب می‌خواهد یک محصول بخرد. یک رویداد NewOrderEvent ایجاد، ترتیب‌بندی و با eventID=2001 در Event Store ذخیره می‌شود.

مراحل ۳ و ۴: باب می‌خواهد مقدار را از ۵ به ۶ تغییر دهد. یک ModifyOrderEvent ایجاد، ترتیب‌بندی و با eventID=2002 در Event Store ذخیره می‌شود.

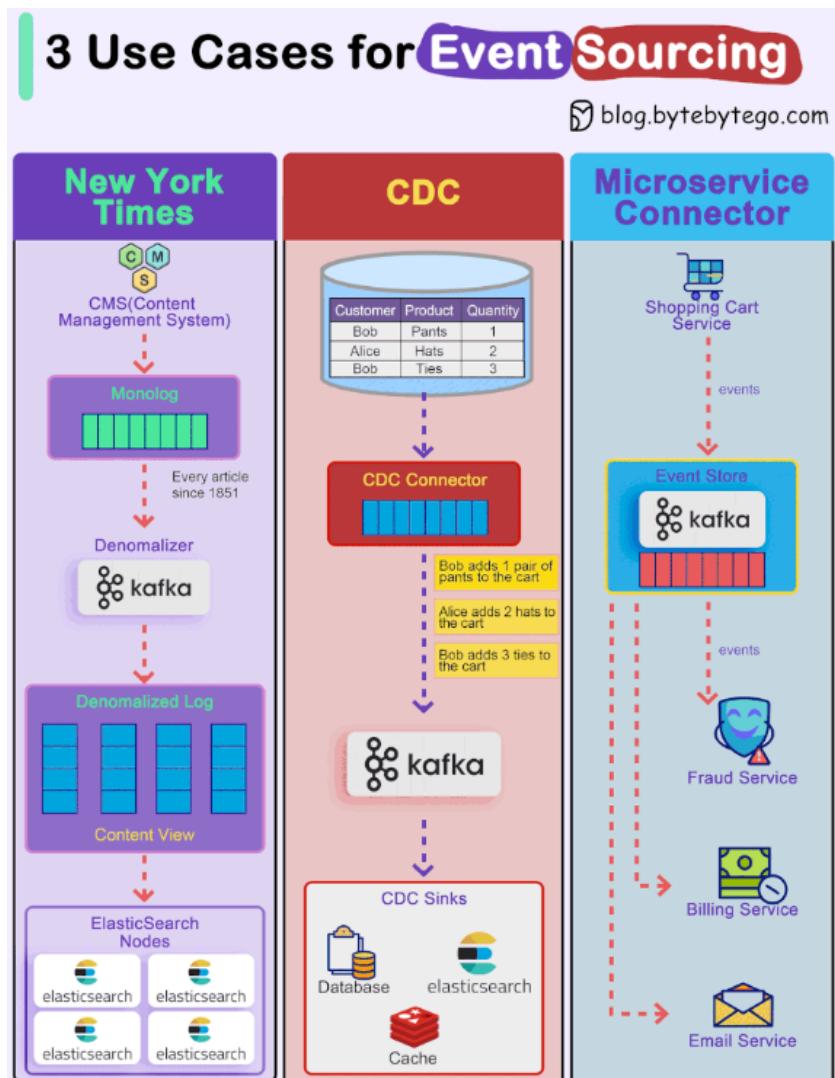
مراحل ۵ و ۶: باب مبلغ ۶۰ دلار برای سفارش پرداخت می‌کند. یک OrderPaymentEvent ایجاد، ترتیب‌بندی و با eventID=2003 در Event Store ذخیره می‌شود. توجه داشته باشید که انواع مختلف رویداد دارای ویژگی‌های رویداد متفاوتی هستند.

مرحله ۷: رویدادهای منتشر شده از Event Store را دریافت می‌کند و آخرين وضعیت سفارش‌ها را می‌سازد. اگرچه OrderView تعداد ۳ رویداد را دریافت می‌کند، اما این رویدادها را به صورت تک به تک اعمال می‌کند و آخرين وضعیت را نگه می‌دارد.

مرحله ۸: باب از OrderService وضعيت سفارش را استعلام می‌کند. سپس OrderView از OrderView استعلام می‌گیرد. OrderView می‌تواند در حافظه یا حافظه کش قرار داشته باشد و نیازی به ذخیره‌سازی دائمی ندارد، زیرا می‌توان آن را از Event Store بازیابی کرد.

## چگونه Event Sourcing را در سیستم‌ها ادغام کنیم؟

منبع رویداد، الگوی برنامه‌نویسی را از ذخیره‌سازی حالت (state) به ذخیره‌سازی رویدادها تغییر می‌دهد. مخزن رویداد (event store) منبع اصلی داده است. بیایید به سه مثال نگاه کنیم:



## ۱. نیویورک تایمز

وبسایت روزنامه نیویورک تایمز، از سال ۱۸۵۱، هر مقاله، تصویر و نام نویسنده را در یک مخزن رویداد<sup>۱</sup> ذخیره می‌کند. سپس داده‌های خام به شکل‌های مختلف غیرطبیعی ElasticSearch (denormalize) شده و برای جستجو در وبسایت به گره‌های مختلف منتقل می‌شوند.

## ۲. رهگیری تغییر داده (CDC - Change Data Capture)

یک کانکتور CDC داده‌ها را از جدول‌ها استخراج کرده و آن‌ها را به رویداد تبدیل می‌کند. این رویدادها به Kafka منتقل می‌شوند و سایر مصرف‌کننده‌ها (در تصویر به عنوان sinks) این رویدادها را از Kafka دریافت می‌کنند.

## ۳. کانکتور میکروسرویس

ما همچنین می‌توانیم از الگوی منبع رویداد (event sourcing) برای انتقال رویدادها بین میکروسرویس‌ها استفاده کنیم. برای مثال، سرویس سبد خرید رویدادهای مختلفی را برای اضافه یا حذف کالا از سبد ایجاد می‌کند. broker یا کارگزار Kafka به عنوان مخزن رویداد عمل می‌کند و سایر سرویس‌ها از جمله سرویس تشخیص تقلب، سرویس صورت‌حساب و سرویس ایمیل، رویدادها را از مخزن رویداد دریافت می‌کنند. از آنجایی که رویدادها منبع اصلی داده هستند، هر سرویس می‌تواند مدل دامنه (domain model) خود را به طور مستقل تعیین کند.

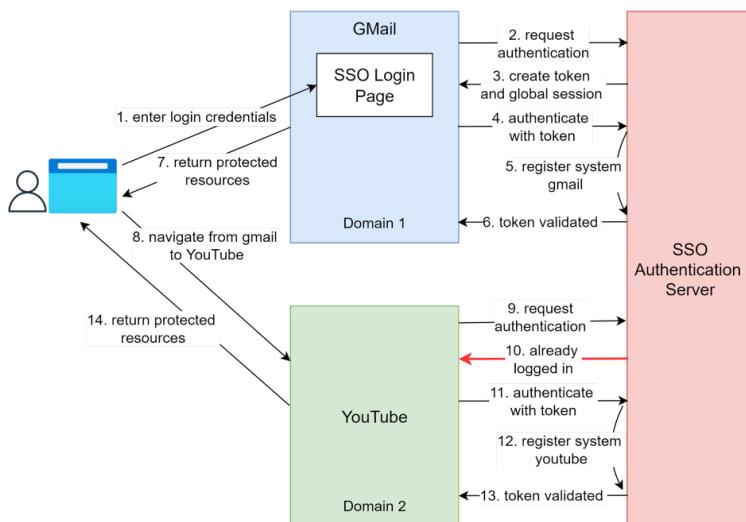
### <sup>۱</sup> event store

۲ نرم‌السازی فرایند و یا تیکنیکی است در طراحی یا طراحی مجدد یک پایگاه داده، جهت بهینه‌سازی و حذف افزونگی داده‌ها و تضمین این امر در فرم‌های مختلف نرم‌السازی می‌باشد. پایگاه داده‌ای که نرم‌السازی نشده است، ممکن است شامل داده‌ای باشد که بدون دلیل خاصی در جداول بانک اطلاعاتی آن به صورت تکراری موجود و ذخیره سازی شده است که این امر میتواند بانک اطلاعاتی را در مواردی چون؛ مصرف حافظه بانک اطلاعاتی، سرعت اجرای کوئری‌ها، بازدهی بروزرسانی پایگاه داده و مهم‌تر از همه تمامیت و درستی داده‌ها را دچار مشکل سازد.

## (ورود یکباره به چند سیستم) چیست؟

یکی از دوستان اخیراً تجربه ناخوشایند خروج از تعدادی از وبسایت‌هایی که روزانه از آن‌ها استفاده می‌کند را پشت سر گذاشت. این اتفاق برای میلیون‌ها کاربر وب آشنا است و فرایندی خسته‌کننده برای حل آن وجود دارد. این ممکن است شامل سعی در بهیادآوردن رمزهای عبور فراموش شده‌ی طولانی، یا تایپ کردن نام حیوانات خانگی دوران کودکی برای پاسخ‌دادن به سوالات امنیتی باشد. SSO این ناراحتی‌ها را برطرف می‌کند و زندگی آنلاین را بهبود می‌بخشد. اما چگونه کار می‌کند؟

به طور اساسی، ورود یکباره به چند سیستم (SSO<sup>۱</sup>) یک طرح احراز هویت است. این امکان را به کاربر می‌دهد که با استفاده از یک شناسه وارد سیستم‌های مختلف شود. دیاگرام زیر نشان می‌دهد که SSO چگونه کار می‌کند.



مرحله ۱: کاربر به سرویس ایمیلی مثل Gmail مراجعه می‌کند. Gmail متوجه می‌شود که کاربر وارد سیستم نشده است و بنابراین کاربر را به سرور احراز هویت SSO هدایت می‌کند، که در آنجا نیز مشخص می‌شود کاربر وارد سیستم نشده است. در نتیجه، کاربر به صفحه ورود SSO هدایت می‌شود، جایی که او اطلاعات ورود خود را وارد می‌کند.

مراحل ۲ تا ۳: سرور احراز هویت SSO، اعتبار اطلاعات را تأیید می‌کند، جلسه ساری<sup>۱</sup> برای کاربر ایجاد می‌کند و یک توکن ایجاد می‌کند.

مراحل ۴ تا ۷: Gmail توکن را در سرور احراز هویت SSO تأیید می‌کند. سرور احراز هویت، سیستم Gmail را ثبت می‌کند و نتیجه «معتبر» برمی‌گرداند. Gmail منبع محافظت شده را به کاربر برمی‌گرداند.

مراحل ۸ از Gmail، کاربر به وبسایت دیگری که متعلق به Google است، مثلاً YouTube می‌رود.

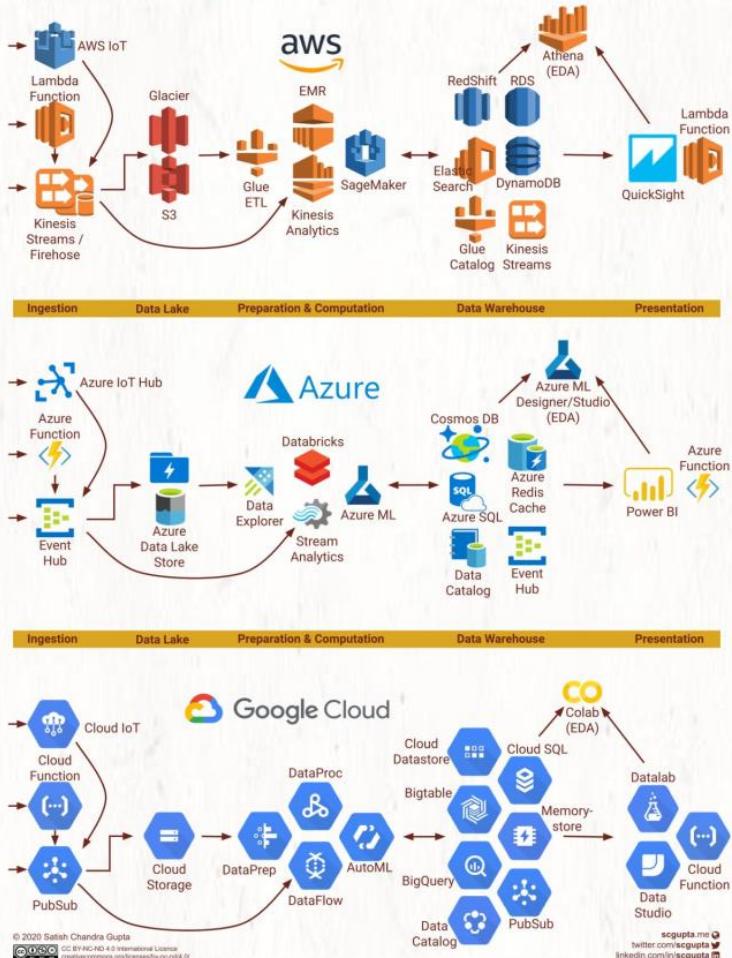
مراحل ۹ تا ۱۰: یوتیوب متوجه می‌شود که کاربر وارد سیستم نشده است و سپس احراز هویت را درخواست می‌کند. سرور احراز هویت SSO متوجه می‌شود که کاربر قبلًاً وارد شده است و توکن را برمی‌گرداند.

مراحل ۱۱ تا ۱۴: یوتیوب توکن را در سرور احراز هویت SSO تأیید می‌کند. سرور احراز هویت، سیستم یوتیوب را ثبت می‌کند و نتیجه «معتبر» برمی‌گرداند. یوتیوب منبع محافظت شده را به کاربر برمی‌گرداند. فرایند کامل شده و کاربر دوباره به حساب خود دسترسی پیدا می‌کند.

## کدام ارائه‌دهنده ابری باید هنگام ساخت یک راهکار داده‌های بزرگ استفاده شود؟

دیگرام زیر مقایسه دقیقی از AWS و Microsoft Azure و Google Cloud می‌دهد.

**Big Data Pipelines on AWS, Microsoft Azure, and GCP**  
[scgupta.link/big-data-pipeline](http://scgupta.link/big-data-pipeline)



بخش‌های مشترک راهکارها:

۱. جذب داده‌های ساختاریافته یا بدون ساختار.
۲. ذخیره‌سازی داده‌های خام.
۳. پردازش داده‌ها، شامل فیلتر کردن، تبدیل، نرمال‌سازی و غیره.
۴. انبار داده<sup>۱</sup>، شامل ذخیره‌سازی کلید - مقدار، پایگاه‌داده رابطه‌ای، پایگاه‌داده OLAP و غیره.
۵. لایه ارائه<sup>۲</sup> با داشبوردها و اعلان‌های بلاذرنگ. جالب است که بینیم فروشنده‌گان مختلف به عنوان مثال، مرحله اول و مرحله آخر هر دو از محصول<sup>۳</sup> serverless استفاده می‌کنند. این محصول در AWS به نام «لامبدا<sup>۴</sup>» و در Azure و Google Cloud به نام «فانکشن<sup>۵</sup>» خوانده می‌شود.

منبع: S.C. Gupta's post

---

Data warehouse<sup>۱</sup>

Presentation layer<sup>۲</sup>

<sup>۳</sup> Serverless در واقع به معنای نبودن سرور تلقی نمی‌شود، بلکه به معنای این است که کاربر نیاز به تهیه سرور و مدیریت و نگهداری از آن را نخواهد داشت.

lambda<sup>۴</sup>

function<sup>۵</sup>

## AWS Lambda پشت‌صحنه

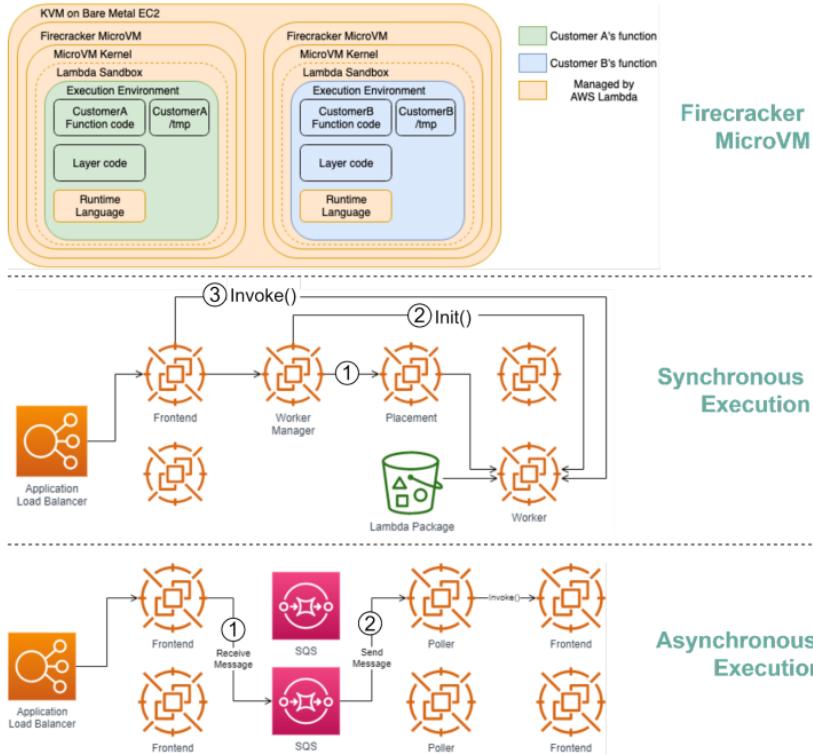
یکی از داغترین موضوعات در سرویس‌های ابری است. چگونه AWS Lambda پشت‌صحنه کار می‌کند؟

یک سرویس محاسباتی Serverless ارائه شده توسط Amazon Web Services (AWS) است که توابع را در پاسخ به رویدادها اجرا می‌کند.

## Firecracker MicroVM

محرك همه توابع Lambda است [۱]. این یک فناوری مجازی‌سازی است که در آمازون توسعه یافته و به زبان **RUST** نوشته شده است. نمودار زیر مدل جداسازی برای AWS Lambda worker را نشان می‌دهد.

توابع Lambda در یک محیط خصوصی اجرا می‌شوند که یک محیط کاربر لینوکس حداقلی، برخی از کتابخانه‌ها و ابزارهای رایج را فراهم می‌کند. این محیط اجرایی (کارگر<sup>۱</sup>) را در نمونه‌های EC2 ایجاد می‌کند.



چگونه Lambda ها شروع و فرآخوانی می شوند؟ دو روش وجود دارد.

### اجرای همزمان

مرحله ۱: مدیریت کنندهی worker با سرویس جایگذاری که مسئول قرار دادن یک بار کاری<sup>۱</sup> در محل برای میزبانی داده شده است (تأمین شده در sandbox) ارتباط برقرار می کند و آن را به مدیریت کنندهی worker بازمی گرداند.

مرحله ۲: مدیریت کنندهی worker سپس می تواند *Init* را فرآخوانی کند تا تابع را برای اجرا با بارگیری بسته Lambda از S3 و تنظیم زمان اجرای Lambda آماده کند.

مرحله ۳: حال Frontend Worker قادر به فرآخوانی *Invoke* است.

<sup>۱</sup> workload

## اجرای غیرهمزمان

مرحله ۱: برنامه توزیع کننده بار<sup>۱</sup> فراخوانی را به یک Frontend در دسترس ارسال می‌کند که رویداد را در یک صف داخلی (SQS) قرار می‌دهد.

مرحله ۲: مجموعه‌ای از نظرسنجی‌ها<sup>۲</sup> به این صف داخلی اختصاص داده شده است که مسئول نظرسنجی<sup>۳</sup> آن و انتقال رویداد به یک Frontend به طور همزمان هستند. پس از قرار گرفتن در Frontend، از الگوی تماس همزمان فراخوانی<sup>۴</sup> که قبلًا توضیح دادیم پیروی می‌کند [۲]

سؤال: می‌توانید مورداد استفاده‌ای برای AWS Lambda تصور کنید؟

منابع:

[۱]AWS Lambda whitepaper:

<https://docs.aws.amazon.com/whitepapers/latest/security-overview-aws-lambda/lambda-executions.html>

[۲]Behind the scenes, Lambda:

<https://www.bschaatsbergen.com/behind-the-scenes-lambda/>

منبع تصویر: [۱][۲]

Load Balancer <sup>۱</sup>

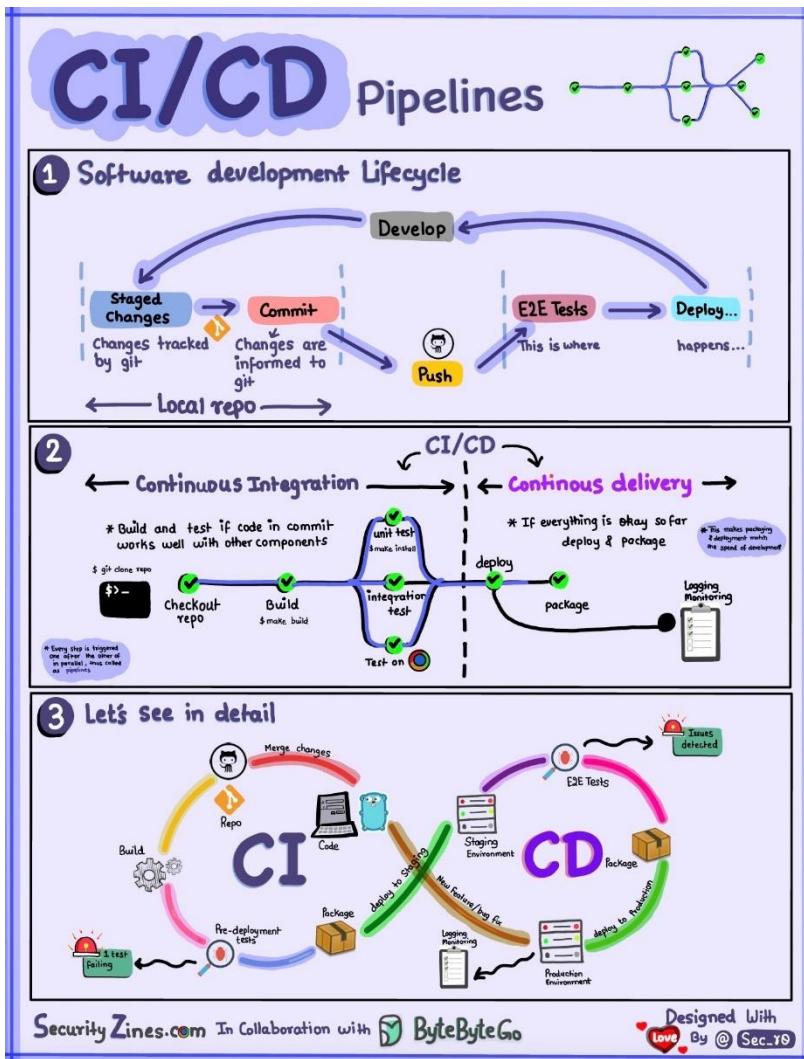
Pollers <sup>۲</sup>

polling <sup>۳</sup>

synchronous invocation call <sup>۴</sup>

# CI/CD

چرخه خط لوله<sup>۱</sup> CI/CD به زبان ساده:



Pipeline<sup>1</sup>

## بخش اول - CI/CD با SDLC

گرددش کار توسعه نرم افزار (SDLC<sup>۱</sup>) شامل چندین مرحله کلیدی است: توسعه، تست، استقرار و نگهداری. CI/CD این مراحل را خودکار و یکپارچه می‌کند تا انتشارهایی سریع‌تر و قابل اعتمادتری را امکان‌پذیر سازد.

هنگامی که کد به مخزن git منتقل می‌شود، فرایند ساخت و آزمایش خودکار را آغاز می‌کند. موارد تست انتها به انتها (e2e<sup>۲</sup>) برای اعتبارسنجی کد اجرا می‌شوند. در صورت موفقیت تست‌ها، کد به طور خودکار در محیط استیجینگ/پروداکشن<sup>۳</sup> مستقر<sup>۴</sup> می‌شود. در صورت بروز مشکل، کد برای رفع اشکال به توسعه برگردانده می‌شود. این اتوماسیون بازخورد سریعی به توسعه‌دهنگان ارائه می‌دهد و خطر وجود اشکال در محیط production را کاهش می‌دهد.

## بخش دوم - تفاوت بین CI و CD

یکپارچه‌سازی مدام<sup>۵</sup> (CI) فرایند ساخت، تست و ادغام را خودکار می‌کند. هر زمان کدی برای تشخیص زودهنگام مشکلات ادغام commit شود در آن زمان تست‌ها را اجرا می‌کند. این امر منجر به commit‌های مکرر کد و بازخورد سریع می‌شود.

تحویل مدام<sup>۶</sup> (CD) فرایندهای انتشار مانند تغییرات زیرساخت<sup>۷</sup> و استقرار<sup>۸</sup> را خودکار می‌کند و اطمینان حاصل می‌کند که نرم افزار از طریق گرددش<sup>۹</sup> کار خودکار در هر زمان به طور قابل اعتماد منتشر شود. CD همچنین ممکن است مراحل تست دستی و تأیید موردنیاز قبل از استقرار در production را خودکار کند.

<sup>۱</sup> software development life cycle

<sup>۲</sup> End-to-end

<sup>۳</sup> staging/production

<sup>۴</sup> deploy

<sup>۵</sup> Continuous Integration

<sup>۶</sup> Continuous Delivery

<sup>۷</sup> infrastructure

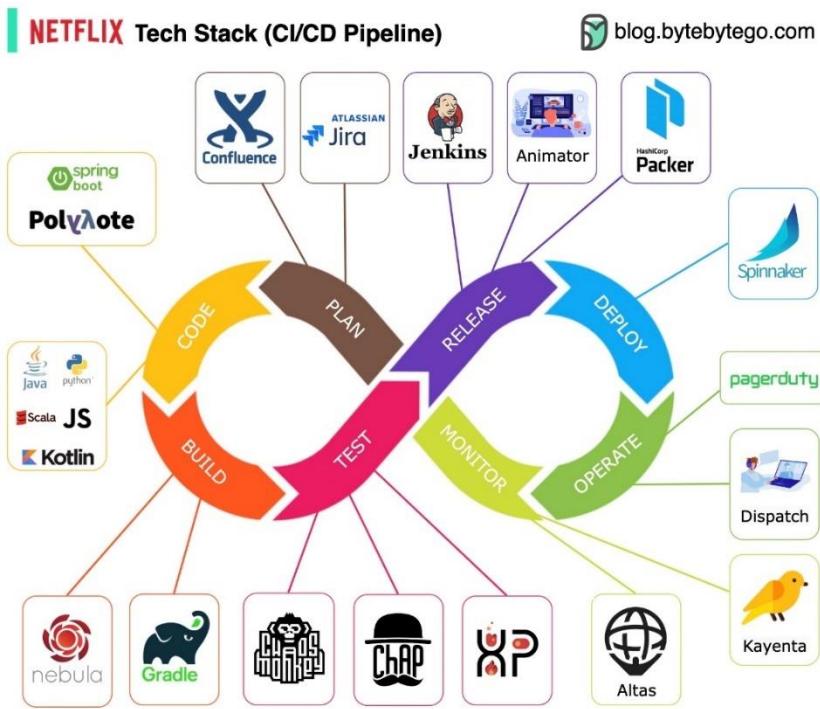
<sup>۸</sup> deployment

### بخش سوم - pipeline CI/CD

یک pipeline CI/CD معمولی دارای چندین مرحله متصل است:

- توسعه‌دهنده تغییرات کد را در source control (مثلاً گیت) ثبت می‌کند.
- سرور CI تغییرات را تشخیص داده و build را آغاز می‌کند.
- کد کامپایل شده و تست می‌شود (تست‌های واحد، تست‌های ادغام<sup>(۱)</sup>).
- نتایج تست به توسعه‌دهنده گزارش می‌شود.
- در صورت موفقیت، برنامه در محیط‌های staging/production مستقر می‌شوند.
- ممکن است قبل از انتشار آزمایش‌ها بیشتری روی production انجام شود.
- سیستم CD تغییرات تأیید شده را در production مستقر می‌کند

## Netflix Tech Stack (CI/CD Pipeline)



برنامه‌ریزی<sup>۱</sup>: مهندس‌های نتفلیکس از JIRA برای برنامه‌ریزی و از Confluence برای مستندسازی استفاده می‌کند.

جawa زبان برنامه‌نویسی اصلی برای سرویس backend است، در حالی که از زبان‌های دیگری برای موارد مختلف استفاده می‌شود.

از Gradle عمدهاً برای build استفاده می‌شود و افزونه‌های Gradle برای پشتیبانی از موارد مختلف ساخته شده‌اند.

**Packaging**: بسته و وابستگی‌ها برای انتشار در یک AMI<sup>۱</sup> بسته‌بندی می‌شوند.  
 تست: تست بر فرهنگ تولید<sup>۲</sup> با تمرکز بر ساخت ابزارهای آشفته<sup>۳</sup> تأکید می‌کند.  
 استقرار<sup>۴</sup>: نتفلیکس از Spinnaker اختصاصی خود برای canary rollout<sup>۵</sup> استفاده می‌کند.  
 نظارت<sup>۶</sup>: متریک‌های مانیتورینگ در Atlas متمرکز شده‌اند و از Kayenta برای تشخیص ناهنجاری‌ها استفاده می‌شود.  
 گزارش حادثه: حوادث بر اساس اولویت ارسال می‌شوند و از PagerDuty برای رسیدگی به حوادث استفاده می‌شود.

---

Amazon Machine Image<sup>۱</sup>  
produc chaos tools tion<sup>۲</sup>

chaos tools<sup>۳</sup>

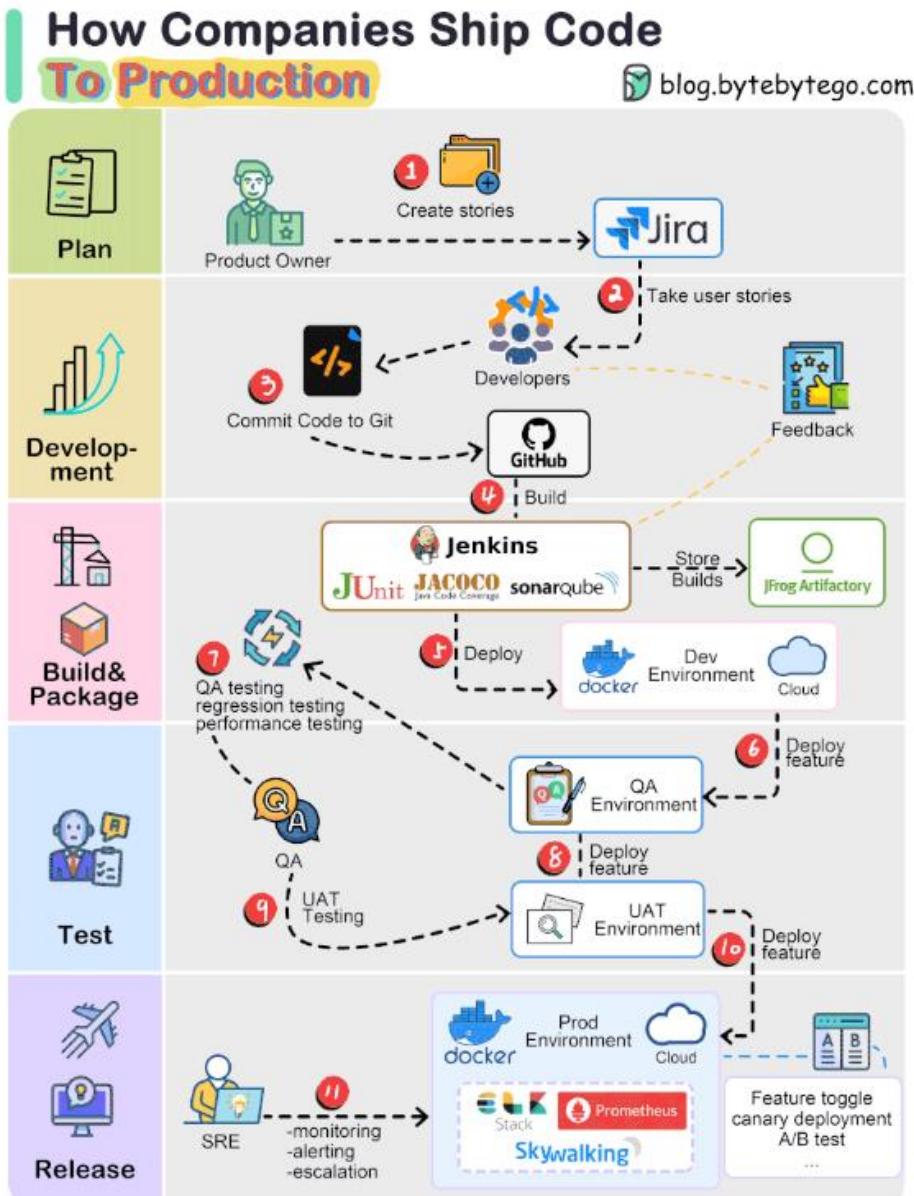
Deployment<sup>۴</sup>

<sup>۵</sup> استقرار چرخشی

Monitoring<sup>۶</sup>

## چرخه توسعه نرم افزار چابک

شکل زیر یک چرخه توسعه نرم افزار چابک را به عنوان یک نمونه نشان می دهد.



## ۱. صاحب محصول (product owner)

بر اساس نیازمندی‌ها، داستان‌های کاربری (creating user stories) را ایجاد می‌کند.

## ۲. تیم توسعه

داستان‌های کاربری را از لیست کارهای معوق (Backlog) برمی‌دارد و آن‌ها را برای یک‌چرخه‌ی توسعه‌ی دوهفته‌ای در یک اسپرینت (Sprint) قرار می‌دهد.

## ۳. توسعه‌دهندگان

کد منبع (source code) را در مخزن کد گیت (Git) ثبت و Commit می‌کنند.

## ۴. Jenkins به کمک Build

ساخت یا Build نرم افزار در Jenkins آغاز می‌شود. کدهای برنامه باید تست‌های واحد (unit tests)، حد لازم پوشش‌دادن سطح زیادی از کد جهت تست<sup>۱</sup> و دروازه‌های SonarQube<sup>۲</sup> را با موفقیت پشت سر بگذارد.

## ۵. artifactory

پس از Build موفقیت‌آمیز، خروجی در artifactory<sup>۳</sup> ذخیره می‌شود. سپس، ساخت در محیط توسعه مستقر می‌شود.

## ۶. تیم‌های تست

ممکن است تیم‌های توسعه‌ی متعددی روی ویژگی‌های مختلف کار کنند. ویژگی‌ها باید به طور مستقل تست شوند، بنابراین در محیط‌های QA1 و QA2 مستقر می‌شوند.

---

## ۱. code coverage threshold

که قبل SonarSource نامیده می‌شد) یک پلت فرم منبع باز است که توسط SonarQube<sup>۲</sup> برای بازرسی مدام کیفیت کد برای انجام بررسی‌های خودکار با تجزیه و تحلیل استاتیک کد برای شناسایی اشکالات و خطاهای کد در ۲۹ زبان برنامه نویسی توسعه یافته است. SonarQube گزارش‌هایی در مورد کدهای تکراری، استانداردهای کدگذاری، تست‌های واحد، پوشش کد، پیچیدگی کد، نظرات، اشکالات و توصیه‌های امنیتی ارائه می‌دهد.

یک اصطلاح معروف برای اشاره به مدیریت repository<sup>۳</sup> است که همه منابع بازیزی شما را سازماندهی می‌کند. این منابع می‌تواند شامل Artifactory<sup>۴</sup> به صورت ریموت، کتابخانه‌های اختصاصی و سایر منابع شخص ثالث باشد. یک مدیر مخزن همه این منابع را به یک مکان واحد می‌کشد.

## ۷. تیم تضمین کیفیت

تیم تضمین کیفیت، محیط‌های تست جدید را انتخاب کرده و تست‌های تضمین کیفیت، تست‌های رگرسیون<sup>۱</sup> و عملکردی را انجام می‌دهد.

## ۸. محیط پذیرش نهایی کاربران (UAT - User Acceptance Testing)

پس از اینکه build های QA تأیید تیم تضمین کیفیت را با موفقیت پشت سر گذاشته‌اند، در محیط پذیرش نهایی کاربران یا به‌اصطلاح UAT در مستقر می‌شوند.

## ۹. انتشار (release)

در صورت موفقیت تست‌های UAT، build ها به کاندیدای release تبدیل می‌شوند و طبق برنامه در محیط production مستقر خواهند شد.

## ۱۰. تیم مهندسی قابلیت اطمینان سایت (SRE - Site Reliability Engineering)

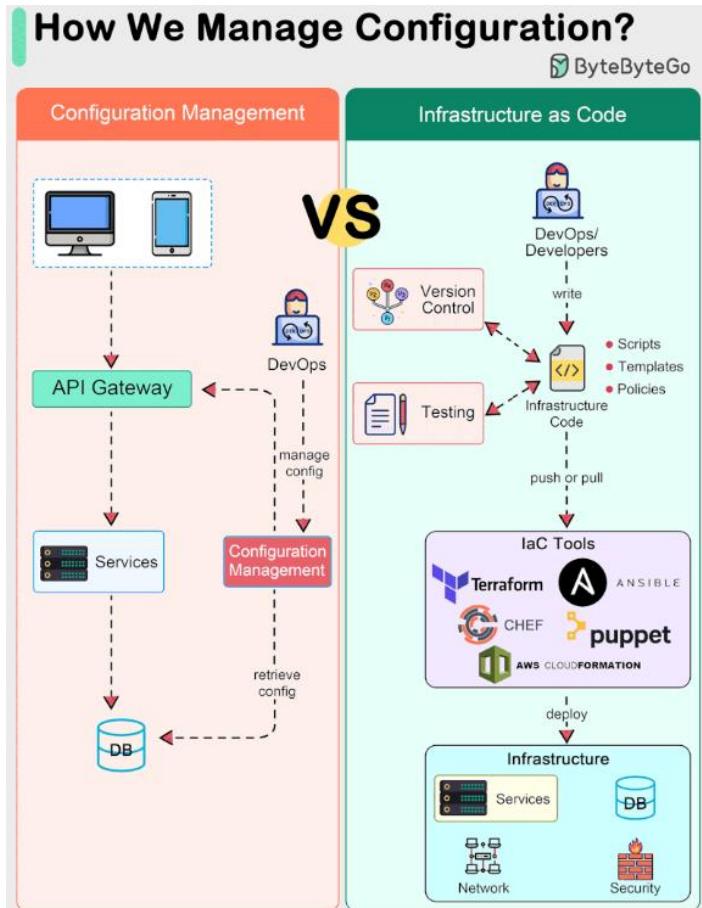
که تیم مهندسی قابلیت اطمینان سایت است.

---

<sup>۱</sup> آزمون رگرسیون نوعی روش برای آزمون نرم‌افزار است که هدف از آن پیدا کردن اشکالات نرم‌افزاری جدید یا رگرسیون‌ها در نواحی مشغول به کار و همینطور نواحی غیر فعال سیستم، پس از اعمال کردن تغییراتی نظیر بهینه‌سازی، اعمال وصله، ایجاد تغییر در پیکربندی نرم‌افزار و غیره است.

## چگونه پیکربندی‌ها را در یک سیستم مدیریت کنیم؟

این نمودار مقایسه‌ای بین مدیریت پیکربندی سنتی و IaC (زیرساخت به عنوان کد<sup>۱</sup>) را نشان می‌دهد.



### مدیریت پیکربندی

این روش برای مدیریت و تأمین زیرساخت فناوری اطلاعات از طریق فرایندهای سیستمی و تکرارپذیر طراحی شده است. این کار برای اطمینان از عملکرد سیستم طبق انتظارات بسیار مهم است.

مدیریت پیکربندی سنتی بر حفظ وضعیت مطلوب موارد پیکربندی سیستم، مانند سرورها، دستگاههای شبکه و اپلیکیشن‌ها پس از تدارک و راهاندازی آنها تمرکز دارد. این کار معمولاً شامل تنظیمات اولیه دستی توسط DevOps است. تغییرات با دستورات گام‌به‌گام مدیریت می‌شوند.

### IaC چیست؟

از سوی دیگر، IaC نشان‌دهنده تغییر در نحوه تأمین و مدیریت زیرساخت است و راهاندازی و تغییرات زیرساخت را به عنوان شیوه‌های توسعه نرم‌افزار در نظر می‌گیرد. IaC با شروع و مدیریت سیستم از طریق کد، تأمین زیرساخت را به صورت خودکار در می‌آورد. این کار اغلب از رویکرد اعلانی<sup>۱</sup> استفاده می‌کند، جایی که وضعیت مطلوب زیرساخت شرح داده شده است.

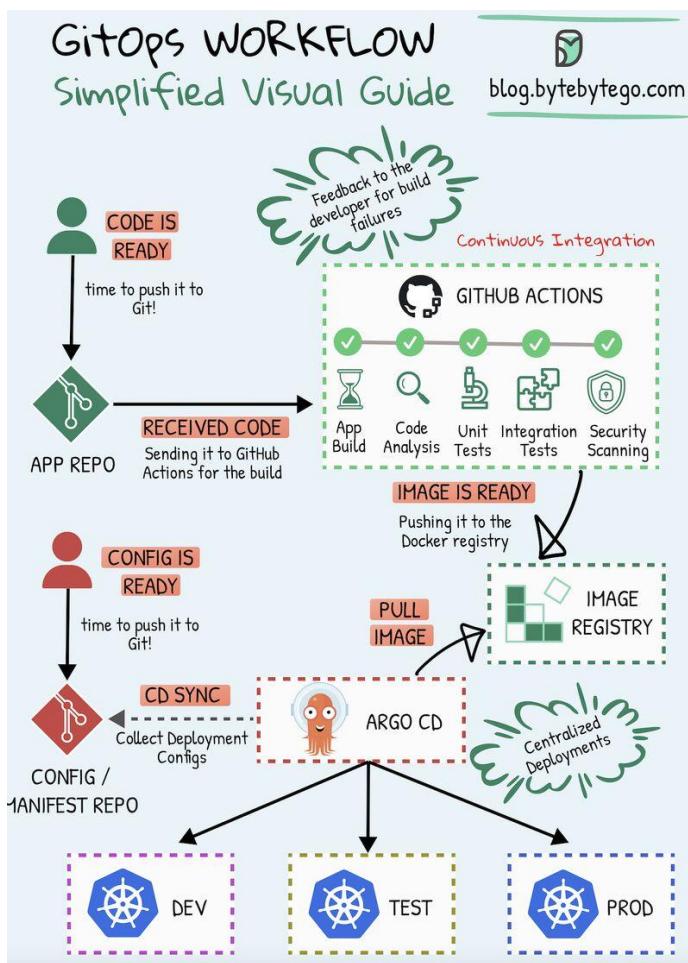
ابزارهایی مانند Puppet، Chef، AWS CloudFormation، Terraform و declarative source controlled<sup>۲</sup> برای تعریف زیرساخت در فایل‌های کد که تحت کنترل منبع هستند استفاده می‌شوند. IaC نشان‌دهنده تکاملی به سمت اتوماسیون، تکرارپذیری و کاربرد شیوه‌های توسعه نرم‌افزار در مدیریت زیرساخت است.

<sup>۱</sup>declarative

<sup>۲</sup>source controlled

## مسیر کاری GitOps

GitOps با معرفی گیت به عنوان هسته مرکزی برای مدیریت و خودکارسازی کل چرخه عمر برنامه‌ها و زیرساخت‌ها، تغییری در نحوه مدیریت نرم افزار و زیرساخت ایجاد کرد.



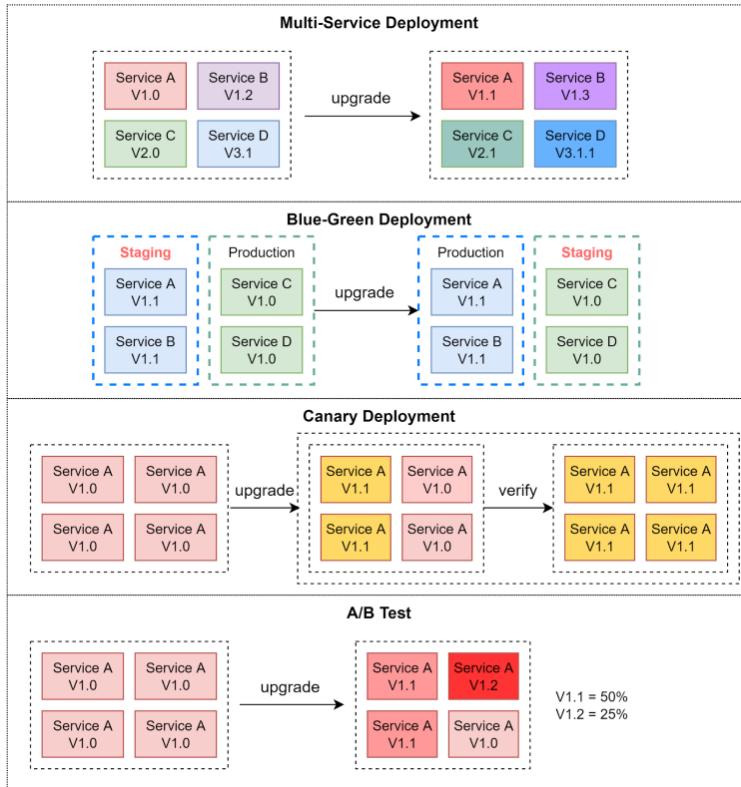
(CI/CD) این بر اساس اصول کنترل نسخه، همکاری و ادغام و استقرار مداوم یا در اصطلاح ساخته شده است.

## ویژگی‌های مهم عبارت‌اند از:

۱. کنترل نسخه و همکاری: متمرکزکردن کدها، پیکربندی‌ها و زیرساخت در گیت برای کنترل و همکاری.
۲. سیستم اعلانی<sup>۱</sup>: توصیف وضعیت مطلوب سیستم برای کنترل نسخه<sup>۲</sup> آسان‌تر.
۳. تحويل خودکار<sup>۳</sup>: خودکارسازی استقرار از طریق فرایندهای مبتنی بر گیت که به طور نزدیک با خطوط لوله CI/CD ادغام می‌شوند.
۴. زیرساخت تغییرناپذیر: ایجاد تغییرات از طریق گیت به جای محیط زنده برای جلوگیری از ناسازگاری‌ها.
۵. قابلیت مشاهده و بازخورد: نظارت بر سیستم‌ها به صورت بلاذرنگ برای هماهنگ‌کردن وضعیت واقعی با وضعیت اعلام شده در گیت.
۶. امنیت و انطباق: پیگیری تغییرات در گیت برای امنیت و انطباق، با دسترسی خاصی مانند role-based access برای کنترل بیشتر.

## استراتژی‌های استقرار

استقرار<sup>۱</sup> یا ارتقای<sup>۲</sup> سرویس‌ها فرایندی حساس است. در این پست، ما استراتژی‌های کاهش خطر را بررسی می‌کنیم. دیاگرام زیر استراتژی‌های رایج را نشان می‌دهد.



استقرار چند - سرویسی<sup>۳</sup> در این مدل، تغییرات جدید به طور همزمان در چندین سرویس مستقر می‌شوند. این رویکرد آسان برای پیاده‌سازی است. اما از آنجاکه همه

Deploying<sup>۱</sup>  
upgrading<sup>۲</sup>

Multi-Service Deployment<sup>۳</sup>

سرویس‌ها هم‌زمان ارتقا می‌بابند، مدیریت و تست وابستگی‌ها<sup>۱</sup> دشوار است. همچنین، بازگشت ایمن<sup>۲</sup> به حالت قبلی تیز دشوار است.

### استقرار آبی - سبز (Blue-Green Deployment)

در استقرار آبی - سبز، ما دو محیط یکسان داریم: یکی محیط تست (آبی) و دیگری تولید<sup>۳</sup> (سبز). محیط تستی یک نسخه جلوتر از production است. پس از اتمام آزمایش‌ها در محیط تستی، ترافیک کاربران به محیط آزمایشی منتقل می‌شود و محیط تستی به production تبدیل می‌شود. این استراتژی استقرار، بازگشت به حالت قبلی را ساده می‌کند، اما داشتن دو محیط production یکسان ممکن است پر هزینه باشد.

### استقرار کاناری (Canary Deployment)

استقرار کاناری سرویس‌ها را به تدریج و هر بار برای زیرمجموعه‌ای از کاربران ارتقا می‌دهد. این روش ارزان‌تر از استقرار آبی - سبز است و بازگشت به حالت قبلی راحتی دارد. با این حال، از آنجاکه محیط آزمایشی وجود ندارد، ما مجبوریم در محیط production آزمایش کنیم. این فرایند پیچیده‌تر است؛ زیرا ما باید کاناری را زیر نظر داشته باشیم در حالی که به تدریج کاربران بیشتری را از نسخه قدیمی دور می‌کنیم.

### آزمایش A/B

در آزمایش A/B، نسخه‌های مختلفی از سرویس‌ها به طور هم‌زمان در production اجرا می‌شوند. هر نسخه یک «آزمایش» را برای زیرمجموعه‌ای از کاربران اجرا می‌کند. آزمایش A/B یک روش ارزان برای آزمایش ویژگی‌های جدید در تولید است. ما باید فرایند استقرار را کنترل کنیم تا در صورتی که برخی ویژگی‌ها به طور تصادفی به کاربران فرستاده شوند، اقدام کنیم.

<sup>۱</sup> test dependencies

<sup>۲</sup> rollback safely

<sup>۳</sup> production

# بررسی DevOps

## کتاب‌های DevOps

برخی از کتاب‌های DevOps که من آنها را روشنگر می‌یابم:

- **Accelerate<sup>1</sup>** - یافته‌ها و علم پشت سنجش عملکرد تحويل نرم‌افزار را ارائه می‌دهد.
- **Continuous Delivery<sup>2</sup>** - مدیریت معماری خودکار و مهاجرت داده را معرفی می‌کند. همچنین مشکلات کلیدی و راه حل‌های بهینه در هر زمینه را مشخص می‌کند.
- **Site Reliability Engineering<sup>3</sup>** - کتاب معروف SRE گوگل. آن چرخه کامل توسعه، انتشار و پایش گوگل را توضیح می‌دهد و نحوه مدیریت بزرگ‌ترین سیستم‌های نرم‌افزاری جهان را شرح می‌دهد.
- **Effective DevOps** - راه‌های مؤثر برای بهبود هماهنگی تیم را ارائه می‌دهد.
- **The Phoenix Project** - یک رمان کلاسیک درباره اثربخشی و ارتباطات انسانی: کار IT مانند کار کارخانه تولیدی است و باید یک سیستم برای روان‌سازی جریان کار ایجاد شود. خواندن آن بسیار جالب است!
- **کتابچه راهنمای DevOps** - توسعه محصول، تضمین کیفیت، عملیات IT و امنیت اطلاعات را معرفی می‌کند.

کتاب DevOps محبوب شما کدام است؟

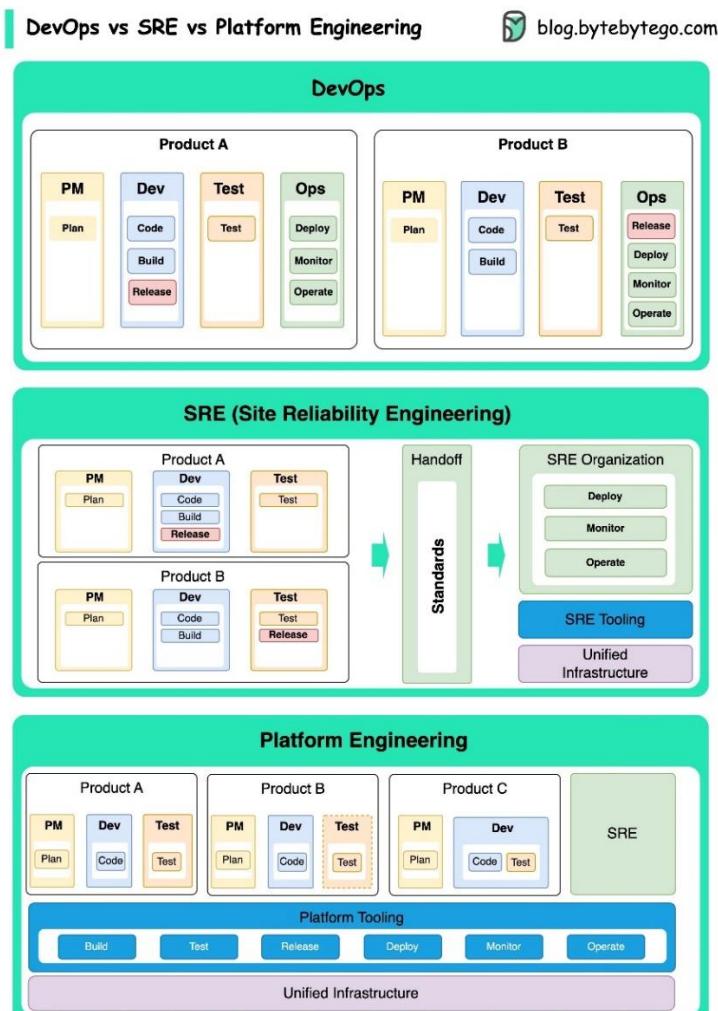
<sup>1</sup> شتاب دهی

<sup>2</sup> تحويل مدام

<sup>3</sup> مهندسی قابلیت اطمینان سایت

## در مقابل SRE در مقابل مهندسی پلتفرم DevOps

مفهوم SRE و مهندسی پلتفرم در زمان‌های مختلف پدید آمده‌اند و توسط افراد و سازمان‌های مختلف توسعه یافته‌اند.



DevOps به عنوان یک مفهوم در سال ۲۰۰۹ توسط «پاتریک دبوآ» و «اندرو شفر» در کنفرانس Agile معرفی شد. آنها به دنبال پر کردن شکاف بین توسعه نرم افزار و عملیات با ترویج فرهنگ همکاری و مسئولیت مشترک برای کل چرخه عمر توسعه نرم افزار بودند.

SRE یا مهندسی قابلیت اطمینان سایت<sup>۱</sup>، برای اولین بار توسط گوگل در اوایل دهه ۲۰۰۰ برای رسیدگی به چالش‌های عملیاتی در مدیریت سیستم‌های پیچیده و مقیاس بزرگ، انجام شد. گوگل برای بهبود قابلیت اطمینان و کارایی سرویس‌های خود اقدام به توسعه شیوه‌ها و ابزارهای SRE مانند سیستم مدیریت خوشه با نام Borg و سیستم نظارت معروف به Monarch کرد.

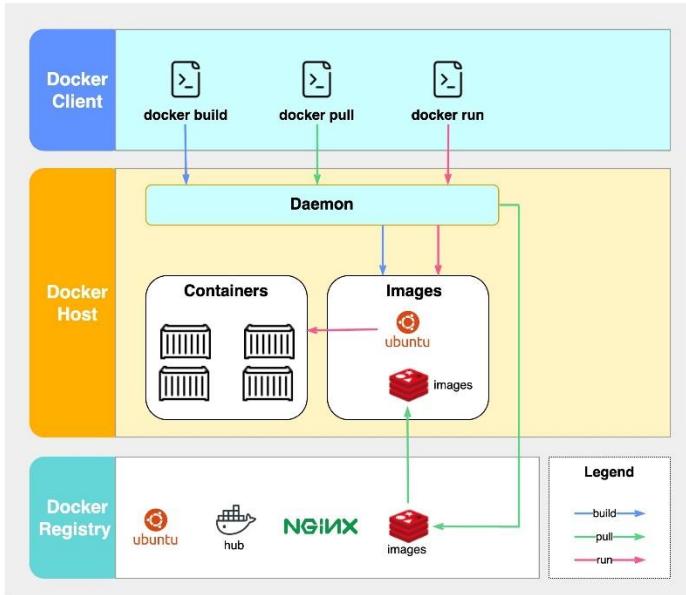
مهندسي پلتفرم<sup>۲</sup> مفهومی جدیدتر است که بر اساس مهندسی SRE بنا شده است. مبدأ دقیق مهندسی پلتفرم کمتر مشخص است، اما به طور کلی به عنوان توسعه‌ای از شیوه‌های DevOps و SRE در نظر گرفته می‌شود، با تمرکز بر ارائه یک پلتفرم جامع برای توسعه محصول که کل جنبه‌های کسب و کار را پشتیبانی می‌کند.

این نکته قابل توجه است که در حالی که این مفاهیم در زمان‌های مختلف پدید آمده‌اند، همگی آن‌ها به روند کلی بهبود همکاری، اتوماسیون و کارایی در توسعه و عملیات نرم افزار مرتبط هستند.

## چگونه کار می‌کند؟ Docker

نمودار زیر معماری Docker و نحوه عملکرد آن را در هنگام اجرای docker build و نحوه عملکرد آن را در هنگام اجرای docker run و docker pull نشان می‌دهد.

### How does Docker Work? blog.bytebytogo.com

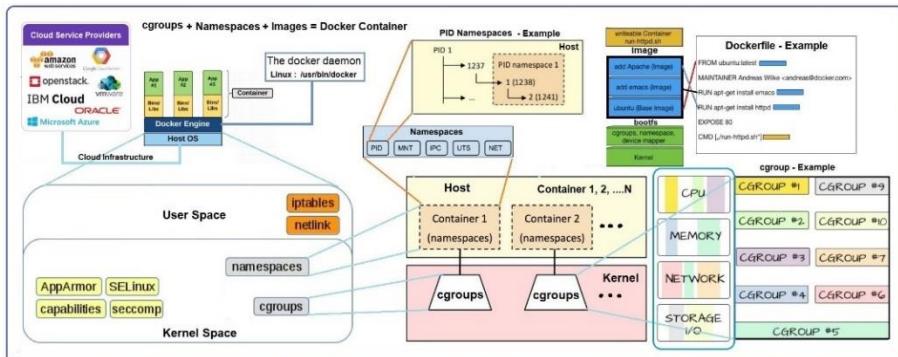


در معماری Docker سه جزء وجود دارد:

- Docker client: که Docker daemon با Docker client صحبت می‌کند.
- Docker host: این Docker daemon به درخواست‌های API داکر گوش می‌دهد و Docker host آبجکت‌های Docker مانند image, containerها، شبکه‌ها و volume را مدیریت می‌کند.
- Docker registry: یک Docker registry رجیستری docker image را ذخیره می‌کند. Docker Hub یک رجیستری عمومی است که هر کسی می‌تواند از آن استفاده کند.

باید دستور docker run را به عنوان مثال در نظر بگیریم.

۱. Docker یک image را از رجیستری می‌کشد.
۲. Docker یک کانتینر جدید ایجاد می‌کند.
۳. Docker یک سیستم فایل خواندن - نوشتمن را به کانتینر اختصاص می‌دهد.
۴. Docker یک رابط شبکه برای اتصال کانتینر به شبکه پیش‌فرض ایجاد می‌کند.
۵. Docker کانتینر را راهاندازی می‌کند.



از اجزای مهم دیگر در داکر به موارد زیر می‌توان اشاره کرد:

### گروههای کنترلی (cgroups)

گروههای کنترلی یا cgroups یک قابلیت کرنل است که برای محدود کردن استفاده از منابع توسط یک فرآیند یا مجموعه ای از فرآیندها استفاده می‌شود. این قابلیت به داکر چهار ویژگی اصلی می‌دهد: محدود کردن منابع، اولویت‌بندی منابع، اندازه‌گیری استفاده از منابع و کنترل گروهی از فرآیندها.

### فضاهای نام (Namespaces)

فضاهای نام یک قابلیت کرنل است که مجازی‌سازی process‌های سبک وزن را برای کانتینرها فراهم می‌کند. این مورد به داکر کمک می‌کند تا این منابع را برای یک کانتینر ایزوله کند - شناسه‌های فرآیند، نام میزبان، شناسه‌های کاربر، دسترسی به شبکه، IPC و سیستم فایل‌ها. فضاهای نام اجازه می‌دهند تا یک «دیدگاه»<sup>۱</sup> به process ارائه شود که همه چیز

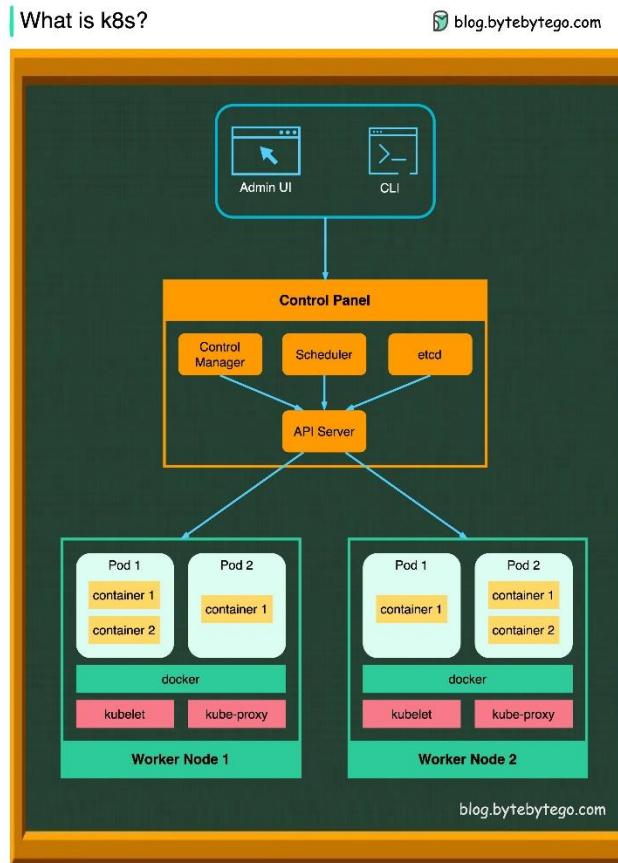
خارج از آن فضاهای نام را پنهان می کند، بنابراین به آن یک محیط جداگانه برای اجرا می دهد، process ها نمی توانند فرآیندهای دیگر را ببینند یا با آنها تداخل کنند.

### Docker Daemon

در داکر (usr/bin/docker/) مسئول مدیریت توابع کرنل مثل گروههای کنترلی Namespace و همانگسازی ها است.

## چیست؟ Kubernetes یا k8s

یک cluster یا خوشه k8s شامل مجموعه‌ای از ماشین‌های worker به نام node/گره است که اپلیکیشن کانتینری شده را اجرا می‌کنند. هر خوشه حداقل یک نود کارگر/worker دارد.



control node (های) کارگر Pod هایی را میزبانی می‌کنند که اجزای بار کاری برنامه هستند.<sup>۱</sup> worker nodes<sup>۲</sup> و Pod ها را در خوشه مدیریت می‌کند. در محیط‌های عملیاتی، plane

<sup>۱</sup> صفحه کنترل

<sup>۲</sup> نودهای کارگر

control plane معمولاً روی چندین کامپیوتر اجرا می‌شود و یک خوشه معمولاً چندین node را اجرا می‌کند که تحمل خطا و دردسترس بودن بالا را ارائه می‌دهد.

### اجزای صفحه کنترل (Control Plane)

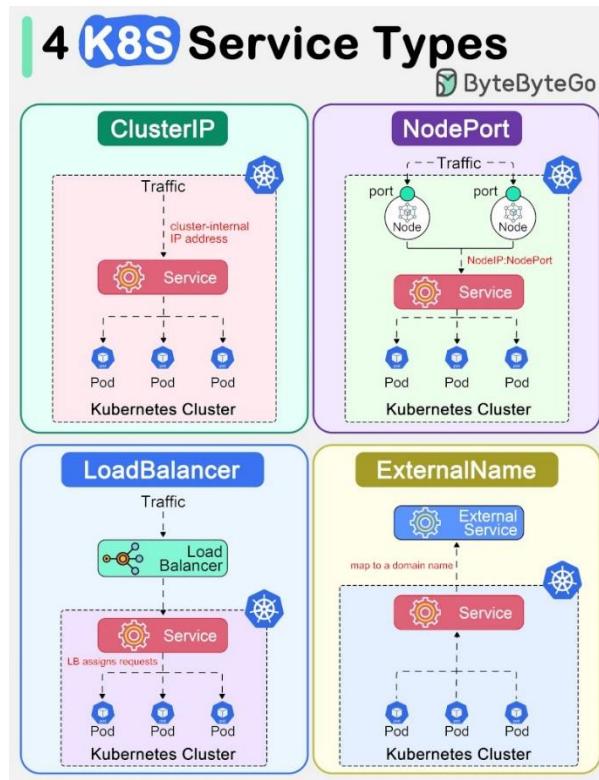
۱. API Server: سرور API با تمام اجزای خوشه k8s ارتباط برقرار می‌کند. تمام عملیات روی Pod‌ها با صحبت‌کردن با سرور API انجام می‌شود.
۲. زمانبندی کننده (Scheduler): زمانبندی کننده بار کاری Pod‌ها را بررسی می‌کند و بار را روی Pod‌های تازه ایجاد شده اختصاص می‌دهد.
۳. مدیر کنترلر (Controller Manager): مدیر کنترلر، کنترلرها از جمله کنترلر نود، کنترلر Job، کنترلر EndpointSlice و کنترلر ServiceAccount را اجرا می‌کند.
۴. Etcd: در واقع etcd یک ذخیره‌ساز کلید - مقدار<sup>۱</sup> است که به عنوان پشتیبان تمام داده‌های خوشه برای Kubernetes استفاده می‌شود.

### نودها (Nodes)

۱. Pod (پاد): یک Pod گروهی از کانتینرها است و کوچک‌ترین واحدی است که آن را مدیریت می‌کند. Pod‌ها یک آدرس IP واحد دارند که به هر کانتینر درون Pod اعمال می‌شود.
۲. Kubelet: یک عامل agent است که روی هر node در خوشه اجرا می‌شود و اطمینان حاصل می‌کند که کانتینرها در یک Pod در حال اجرا هستند.
۳. Kube-Proxy: کیوب - پراکسی یک پروکسی شبکه است که روی هر node در خوشه شما اجرا می‌شود و ترافیک ورودی به یک نود از سرویس را مسیریابی می‌کند. در خواست‌ها برای کار را به کانتینرهای صحیح ارسال می‌کند.

## چهار نوع برتر سرویس‌های Kubernetes در یک نمودار

نمودار زیر ۴ روش برای نمایش یک سرویس را نشان می‌دهد.



در Kubernetes، یک Service روشی برای نمایش و در معرض قراردادن<sup>۱</sup> یک برنامه شبکه‌ای در خوشه<sup>۲</sup> است. ما از یک سرویس برای استفاده از مجموعه Pods موجود در شبکه استفاده می‌کنیم تا کاربران بتوانند با آن تعامل داشته باشند.

exposing<sup>۱</sup>  
cluster<sup>۲</sup>

چهار نوع سرویس Kubernetes وجود دارد: LoadBalancer، NodePort، ClusterIP و ExternalName. خاصیت "type" در مشخصات سرویس تعیین می‌کند که سرویس چگونه در شبکه نمایش داده شود.

### **ClusterIP**

در واقع ClusterIP نوع سرویس پیش‌فرض و رایج‌ترین آن است. Kubernetes یک آدرس IP داخلی خوشه را به سرویس ClusterIP اختصاص می‌دهد. این باعث می‌شود سرویس فقط در داخل خوشه قابل دسترسی باشد.

### **NodePort**

این سرویس را با اضافه کردن یک پورت در کل خوشه روی ClusterIP آن را در خارج از خوشه در معرض قرار می‌دهد یا در اصطلاح فنی آن پورت را expose می‌کند. ما می‌توانیم این نوع سرویس را با *NodePort* درخواست کنیم.

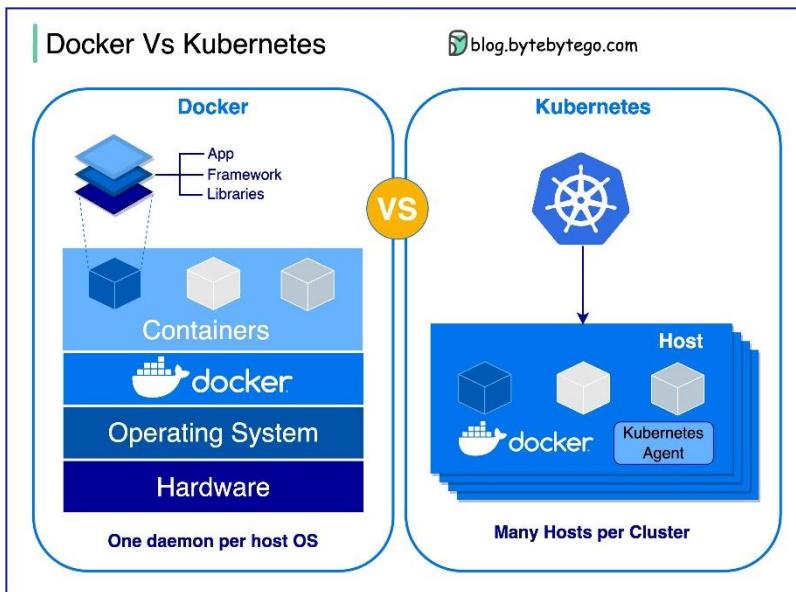
### **LoadBalancer**

این سرویس را با استفاده از توزیع کننده ابر در سمت ارائه‌دهنده<sup>۱</sup>، به صورت خارجی expose می‌شود.

### **ExternalName**

این نام یک دامنه را به یک سرویس اختصاص می‌دهد که معمولاً برای ایجاد یک سرویس در Kubernetes برای نشان‌دادن یک پایگاه‌داده خارجی استفاده می‌شود.

## Kubernetes در مقابل Docker



### چیست؟ Docker

Docker یک پلتفرم تا حدودی متن باز است که به شما امکان می دهد اپلیکیشن های خود را در کانتینرهای ایزوله بسته بندی، توزیع و اجرا کنید. Docker بر روی کانتینر سازی تمرکز دارد و محیط های سبکی را ارائه می دهد که برنامه ها و وابستگی های آنها را در بر می گیرد.

### چیست؟ Kubernetes

Kubernetes که اغلب به عنوان K8s شناخته می شود، یک پلتفرم متن باز برای ارکستراسیون<sup>۱</sup> کانتینرها است. این پلتفرم یک چارچوب برای خود کار سازی استقرار، مقیاس بندی و مدیریت اپلیکیشن های کانتینری شده در سراسر یک خوشه از گره ها (Nodes) را ارائه می دهد.

### این دو چگونه با هم تفاوت دارند؟

<sup>۱</sup> در واقع ارکستراسیون مشابه رهبر ارکست هست که نوازده ها را هماهنگ می کند.

**Docker**: در واقع Docker در سطح یک کانتینر منفرد روی یک میزبان<sup>۱</sup> با یک سیستم عامل واحد عمل می‌کند.

شما باید هر میزبان را به صورت دستی مدیریت کنید و راهاندازی شبکه‌ها، خط‌مشی‌های امنیتی و ذخیره‌سازی برای چندین کانتینر مرتبط می‌تواند پیچیده باشد.

**Kubernetes**: باید توجه داشت که Kubernetes در سطح خوش‌هه<sup>۲</sup> عمل می‌کند. این پلتفرم اپلیکیشن‌های کانتینری شده متعدد را در سراسر چندین میزبان مدیریت می‌کند و اتوماسیون‌هایی را برای کارهایی مانند توزیع بار، مقیاس‌بندی و اطمینان از وضعیت مطلوب اپلیکیشن‌ها فراهم می‌کند.

به طور خلاصه، Docker بر روی کانتینر سازی و اجرای کانتینرها روی میزبان‌های منفرد<sup>۳</sup> تمرکز دارد، در حالی که Kubernetes در مدیریت و ارکستراسیون کانتینرها در جهت مقیاس‌پذیری روی یک خوش‌هه از میزبان‌ها تخصص دارد.

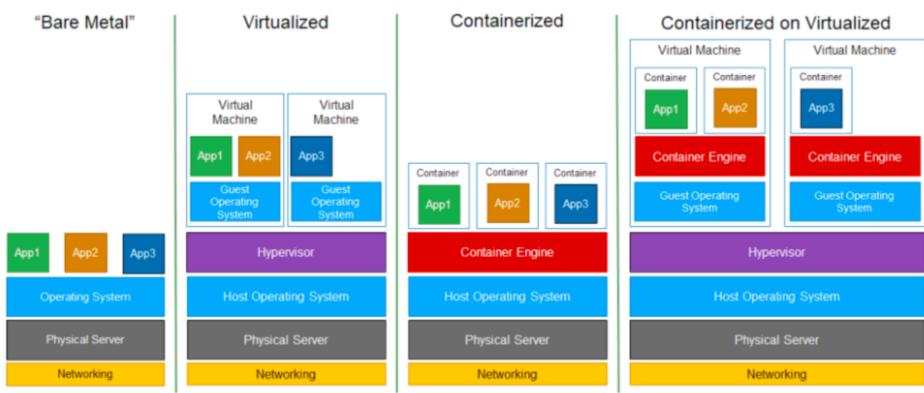
host<sup>۱</sup>

cluster<sup>۲</sup>

individual hosts<sup>۳</sup>

## تفاوت‌ها بین مجازی‌سازی (VMware) و کانتینر سازی (Docker) چیست؟

دیاگرام زیر معماری لایه‌ای مجازی‌سازی و کانتینر سازی را نشان می‌دهد.



«ماجذی‌سازی فناوری‌ای است که به شما امکان می‌دهد چندین محیط شبیه‌سازی شده یا منابع اختصاصی را از یک سیستم سخت‌افزاری فیزیکی واحد ایجاد کنید» [۱].

«کانتینر سازی، بسته‌بندی کد نرم‌افزار به همراه تمامی اجزای ضروری مانند کتابخانه‌ها، چارچوب‌ها و سایر وابستگی‌ها است تا آن‌ها در «کانتینر<sup>۱</sup>» خود مجزا شوند» [۲].

تفاوت‌های اصلی عبارت‌اند از:

- در مجازی‌سازی، هایپرواایزر<sup>۲</sup> یک لایه انتزاعی بر روی سخت‌افزار ایجاد می‌کند، به‌طوری‌که چندین سیستم‌عامل می‌توانند در کنار یکدیگر اجرا شوند. این تکنیک به عنوان نسل اول محاسبات ابری در نظر گرفته می‌شود.

<sup>۱</sup> container

<sup>۲</sup> hypervisor

- کانتینر سازی به عنوان نسخه سبک وزن مجازی سازی در نظر گرفته می شود که سیستم عامل را به جای ساخت افزار مجازی می کند. بدون وجود هایپروایزر، کانتینرها از تخصیص منابع سریع تری برخوردار هستند. تمام منابع (از جمله کد، وابستگی ها) که برای اجرای برنامه یا میکروسرویس لازم هستند، به طور مشترک بسته بندی می شوند، به طوری که برنامه ها می توانند در هر جایی اجرا شوند.

سؤال: چه تفاوت هایی در عملکرد بین مجازی سازی، کانتینر سازی و bare-metal<sup>۱</sup> در تولید مشاهده کرده اید؟

منبع تصویر: <https://lnkd.in/gaPYcGTz>

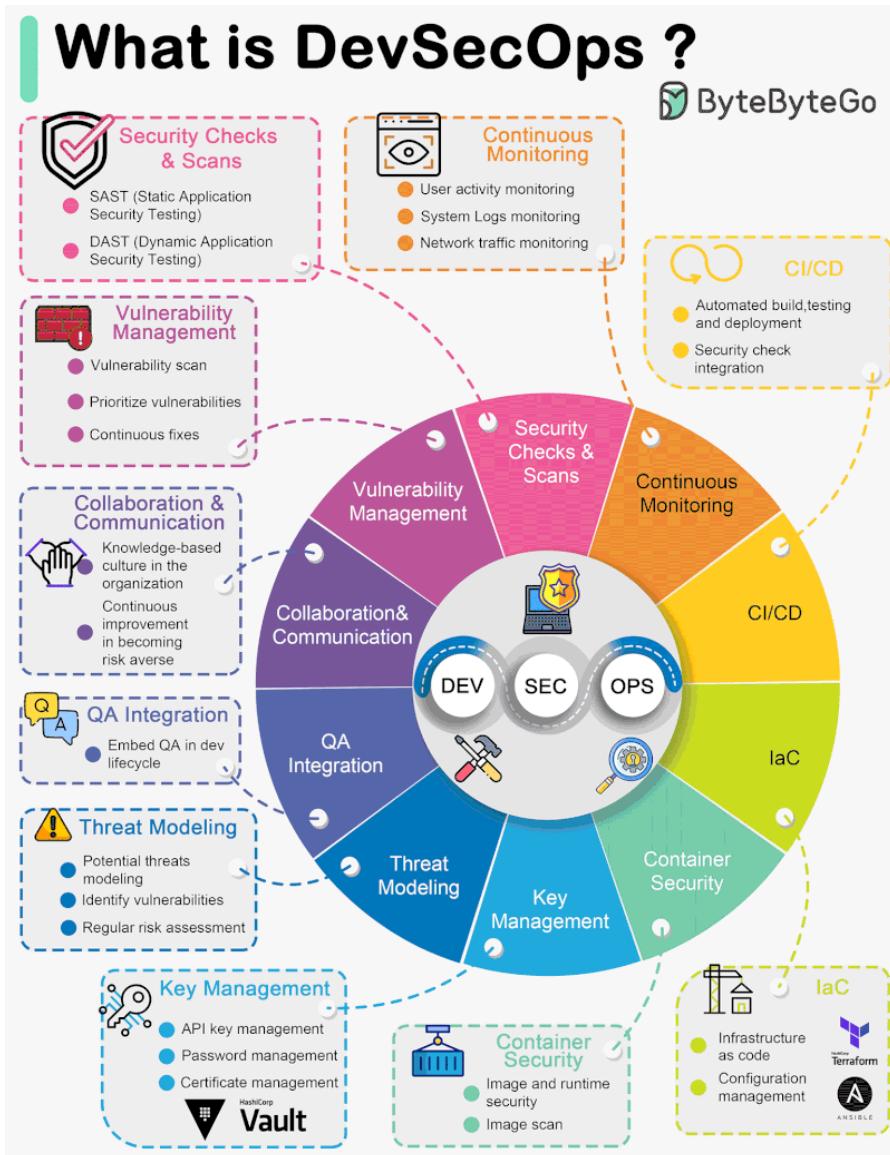
منابع:

[۱] درک مجازی سازی: <https://lnkd.in/gtQY9gkx>

[۲] کانتینر سازی چیست؟: [https://lnkd.in/gm4Qv\\_x2](https://lnkd.in/gm4Qv_x2)

<sup>۱</sup> Bare Metal به محیطی گفته می شود که در آن یک سیستم عامل یا برنامه به صورت مستقیم بر روی ساخت افزار نصب می شود و نه درون سیستم عامل میزان. این محیط به صورت مستقیم با هارد دیسک و سخت افزارهای دیگر کامپیوتر ارتباط برقرار می کند.

## چیست؟ DevSecOps



به عنوان یک تکامل طبیعی از شیوه‌های DevOps با تمرکز بر ادغام امنیت در فرایند توسعه و استقرار نرم افزار ظهر کرد. اصطلاح 'DevSecOps' نشان‌دهنده همگرایی

رویه‌های توسعه (Dev)، امنیت (Sec) و عملیات (Ops) است که بر اهمیت امنیت در طول چرخه عمر توسعه نرم‌افزار تأکید می‌کند.

نمودار زیر مفاهیم مهم در DevSecOps را نشان می‌دهد.

Automated Security Checks . ۱

Continuous Monitoring . ۲

CI/CD Automation . ۳

Infrastructure as Code (IaC) . ۴

Container Security . ۵

Secret Management . ۶

Threat Modeling . ۷

Quality Assurance (QA) Integration . ۸

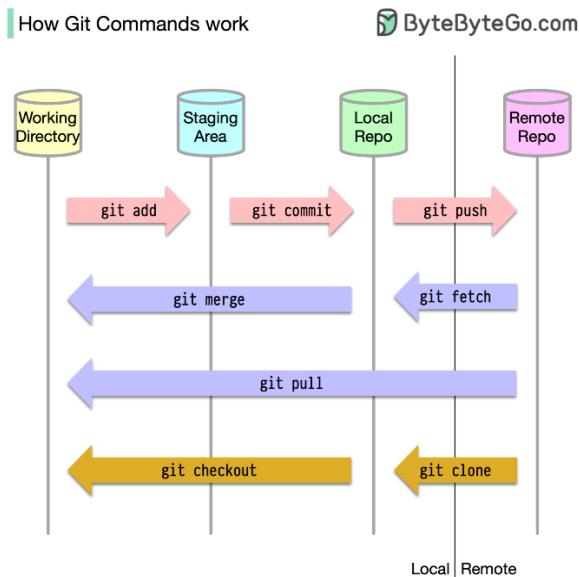
Collaboration and Communication . ۹

Vulnerability Management . ۱۰

# بررسی GIT

## چطور دستورات گیت کار می‌کنند؟

برای شروع، شناسایی محل ذخیره کد ما ضروری است. تصور رایج این است که فقط دو مکان وجود دارد – یکی در یک سرور ریموت مانند GitHub و دیگری در دستگاه محلی ما. با این حال، این کاملاً درست نیست. گیت سه مخزن محلی را روی دستگاه ما نگهداری می‌کند، به این معنی که کد ما را می‌توان در چهار مکان یافت:

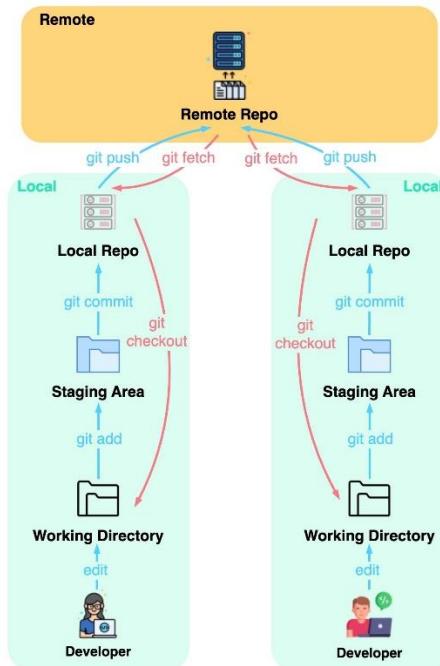


- جایی که ما فایل‌ها را ویرایش می‌کنیم: Working directory
- محل مرحله‌بندی (Staging area): مکان موقتی که فایل‌ها برای commit بعدی در آنجا نگه‌داری می‌شوند.
- حاوی کدی است که commit شده است: Local repository
- سرور که کد را ذخیره می‌کند: Remote repository
- اکثر دستورات گیت عمدتاً فایل‌ها را بین این چهار مکان جابه‌جا می‌کنند.

## گیت چگونه کار می‌کند؟

نمودار زیر گردش کار گیت را نشان می‌دهد.

| How does Git Work?  blog.bytebytogo.com



گیت یک سیستم کنترل نسخه توزیع شده است.

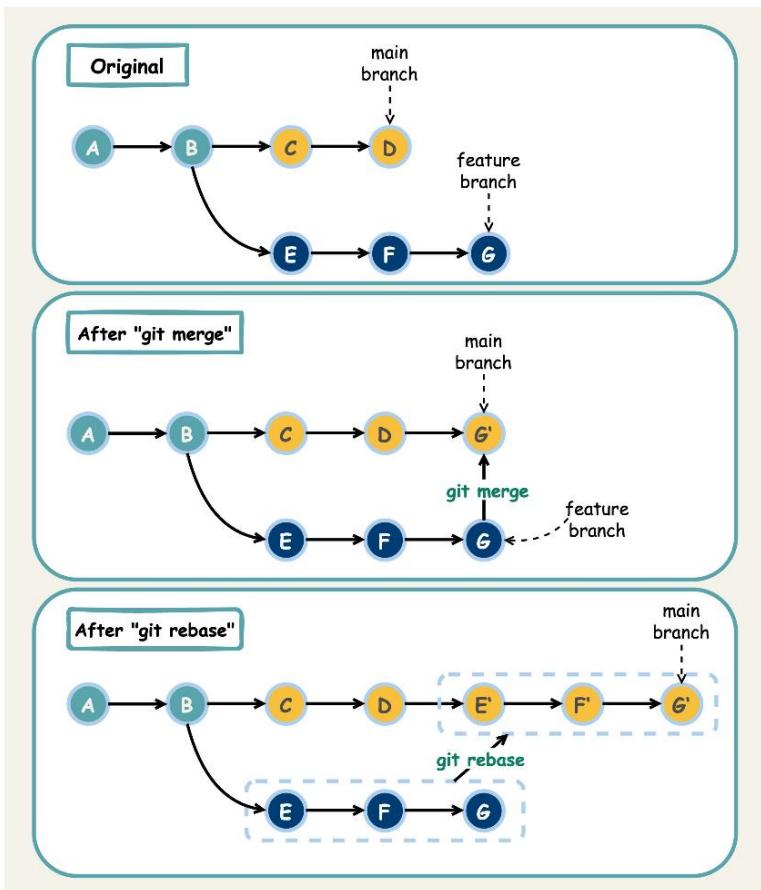
هر توسعه‌دهنده یک کپی محلی از مخزن اصلی را نگه می‌دارد و ویرایش و commit را روی کپی محلی انجام می‌دهد.

اگر commit بسیار سریع است؛ زیرا این عملیات با Remote repository تعامل ندارد. Remote repository خراب شود، فایل‌ها را می‌توان از مخزن‌های محلی بازیابی کرد.

## تفاوت Git rebase و Git merge

### Git Merge vs. Git Rebase

 blog.bytebytego.com



هنگامی که تغییرات را از یک branch گیت به دیگر branch merge می‌کنیم، می‌توانیم از 'git merge' یا 'git rebase' استفاده کنیم. نمودار بالا نحوه عملکرد این دو دستور را نشان می‌دهد.

### Git merge

این یک commit جدید 'G در شاخه اصلی ایجاد می‌کند. 'G تاریخچه هر دو شاخه main و feature را به هم متصل می‌کند. Git merge مخرب نیست. نه شاخه اصلی (main) و نه شاخه feature تغییر نمی‌کنند.

### Git rebase

در واقع Git rebase تاریخچه شاخه feature را به سر شاخه main منتقل می‌کند. برای هر کامیت در شاخه feature، کامیت‌های جدید 'E، 'F و 'G ایجاد می‌کند. مزیت Git rebase این است که یک تاریخچه commit کاملاً خطی دارد. اگر «قانون طلایی Git rebase» رعایت نشود، rebase می‌تواند خطرناک باشد.

### :Git rebase قانون طلایی

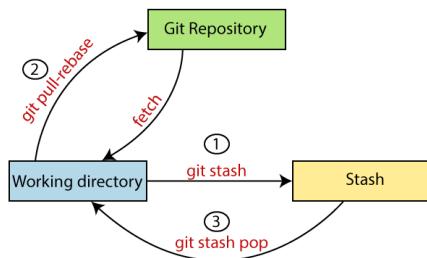
هرگز از آن روی public branch استفاده نکنید!

### بررسی دستورات ویژه در git

یک دستور بسیار مفید در گیت است که به شما اجازه می‌دهد تغییرات فعلی خود را در یک فضای موقت ذخیره کنید، بدون اینکه آنها را commit کنید. این کار زمانی مفید است که:

کار نیمه‌کارهای دارید و می‌خواهید به شاخه دیگری بروید: می‌توانید تغییرات فعلی خود را stash کنید و بعد از انجام کار در شاخه دیگر، تغییرات stash شده را به شاخه قبلی خود برگردانید.

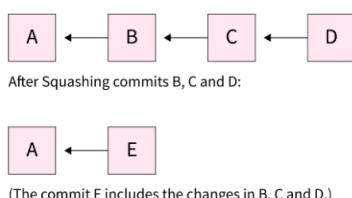
می‌خواهید یک تغییر آزمایشی را انجام دهید: می‌توانید تغییرات فعلی خود را stash کنید، تغییر آزمایشی خود را انجام دهید و اگر نتیجه دلخواه نبود، تغییرات stash شده را برگردانید.



می‌خواهید یک وضعیت تمیز در **repository** داشته باشید: قبل از انجام یک کار مهم یا قبل از ایجاد یک commit جدید، می‌توانید تغییرات کوچک و بی‌اهمیت را stash کنید تا وضعیت کاری شما تمیزتر شود.

## Git Squash

Git Squashing یک ویژگی Git است که به توسعه‌دهنده اجازه می‌دهد تا commit متواتی را در یک commit (که به عنوان commit پایه شناخته می‌شود) merge کند. این به توسعه‌دهنگان اجازه می‌دهد تا درخت پروژه Git را تمیز نگه دارند. اصطلاح Squash برای توصیف فرآیند فشرده‌سازی چندین commit قبلی در یک commit استفاده می‌شود. این استراتژی زمانی مفید است که می‌خواهید تغییرات یک branch را به عنوان یک واحد merge کنید و در صورت نیاز، ردیابی و پیدا کردن تغییرات را آسان‌تر کنید. معمولاً در ترکیب با Merge یا Rebase استفاده می‌شود.



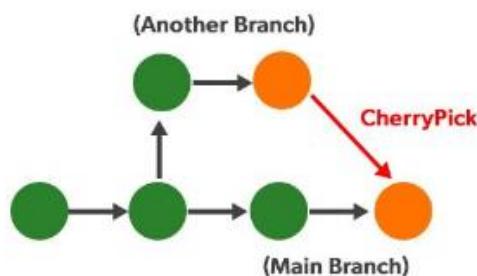
## git cherry-pick

بررسی انتخاب یک commit از یک branch در git به معنای انتخاب یک commit از یک branch و اعمال آن در branch دیگر است. این در تضاد با روش‌های دیگری مانند merge و rebase است که معمولاً بسیاری از commit‌ها را در branch دیگری اعمال می‌کنند. git cherry-pick درست مانند rebasing یک مفهوم پیشرفته و همچنین یک دستور قدرتمند است. عمدتاً اگر نمی‌خواهید کل شاخه را merge کنید ولی برخی از commit‌ها را می‌خواهید merge کنید مورد استفاده قرار می‌گیرد.

### چه زمانی از cherry-pick استفاده کنیم؟

فرض کنید یک توسعه‌دهنده نتواند تشخیص دهد که در حال حاضر در کدام branch است و به اشتباه به جای commit به شاخه main، به یک branch دیگر commit می‌کند. حال برای رفع این اشتباه ابتدا باید git show را اجرا کند، سپس commit را ذخیره کند main branch را بررسی کند، یک اصلاحیه را در آنجا اعمال کند و با همان پیام قبلی commit را کند. اما همه اینها را می‌توان به طور خودکار با استفاده از یک دستور یعنی cherry-pick کرد.

انجام داد.



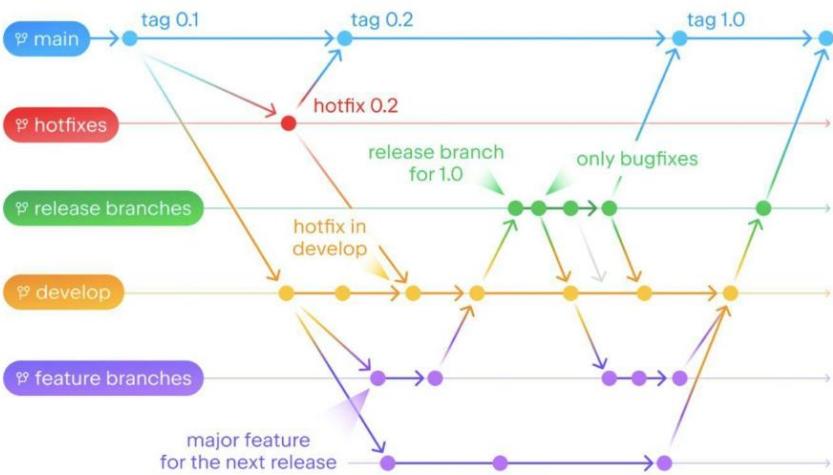
### انواع استراتژی‌های branch

- شاخه main فقط برای کدهای production است.

- شاخه develop برای کد در حال development است.
- شاخه های ویژگی (feature branches) از شاخه develop ساخته می شوند.
- شاخه های رفع مشکل (hotfix branches) از شاخه main ساخته می شوند.
- شاخه های انتشار (release branches) از شاخه develop ساخته می شوند.

## Git flow

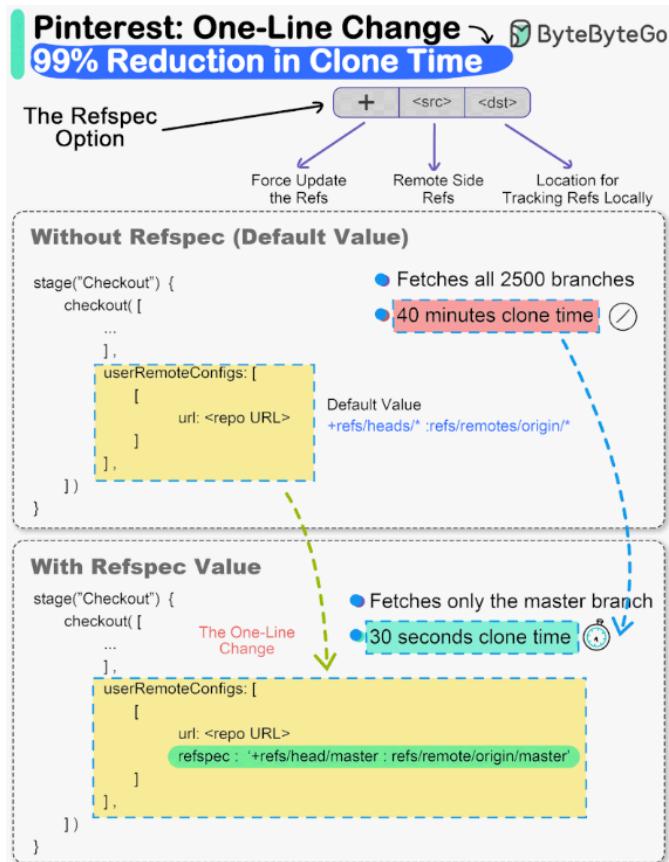
Img source: <https://blog.jetbrains.com/space/2023/04/18/space-git-flow/>



تیم ها اغلب روش های مختلفی برای مدیریت کد خود دارند، مثل Git flow، شاخه های ویژگی (feature branches) و توسعه trunk-based development (Git flow) که روشن های مشابه محبوب ترین هستند. این تصویر نحوه کار آن ها را نشان می دهد.

## یک خط تغییر، زمان clone را تا ۹۹ درصد کاهش داد!

همان طور که ممکن است کلیشهای به نظر برسد، تغییرات کوچک قطعاً می توانند تأثیر زیادی داشته باشند.



تیم بهره‌وری مهندسی در Pinterest این موضوع را به صورت مستقیم تجربه کرده است. آن‌ها یک تغییر کوچک در pipeline مربوط به سامانه Jenkins برای codebase Pinboard ایجاد کردند و این تغییر زمان clone را از ۴۰ دقیقه به حیرت‌انگیز ۳۰ ثانیه کاهش داد. برای درک بهتر، Pinboard قدیمی‌ترین و بزرگ‌ترین monorepo در Pinterest است. چند واقعیت در مورد آن به شرح ذیل است:

- commit ۳۵۰ هزار
- ۲۰ گیگابایت حجم هنگام Clone کامل
- ۶۰ هزار دستور git pull در هر روز کاری

Clone کردن monorepo‌هایی که حجم کد و تاریخچه زیادی دارند، کار زمانبری است. این همان اتفاقی بود که برای Pinboard رخ می‌داد.

build pipeline (نوشته شده با Groovy) با مرحله‌ای به نام Checkout شروع می‌شد که در آن، مخزن repository برای مراحل build و تست Clone می‌شد.

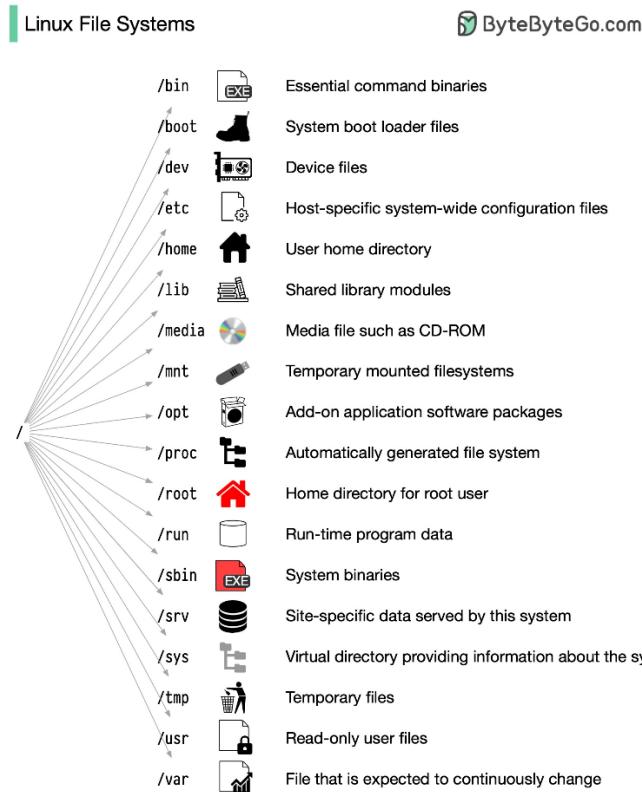
گزینه‌های Clone روی Commit کم‌عمق<sup>۱</sup>، بدون دریافت تگ‌ها و فقط دریافت ۵۰ عدد از commit آخر تنظیم شده بودند. اما یک بهینه‌سازی حیاتی را ازدست‌داده بود. مرحله Checkout از گزینه refspec گیت استفاده نمی‌کرد. این بدان معنا بود که گیت به طور مؤثر تمام refs‌ها را برای هر build دریافت می‌کرد. برای Pinboard، این به معنای دریافت بیش از ۲۵۰۰ تعداد branch بود.

### پس راه حل چه بود؟

تیم به سادگی گزینه refspec را اضافه کرد و مشخص کرد که به کدام refspec اهمیت می‌دهند. در این مورد، master branch مربوط به یک شاخه کار کند و زمان کل ساخت این تغییر واحد به Git clone اجازه داد تا فقط با یک شاخه کار کند و زمان کل ساخت monorepo را به میزان قابل توجهی کاهش دهد.

# Linux بررسی

## سیستم فایل لینوکس به زبان ساده



سیستم فایل لینوکس قبلًاً شبیه به یک شهر بی‌نظم بود که هر کسی هر کجا دلش می‌خواست خانه می‌ساخت. اما در سال ۱۹۹۴، استاندارد سلسله‌مراتب سیستم فایل (FHS<sup>۱</sup>) برای برقراری نظم در سیستم فایل لینوکس معرفی شد.

با اجرای استانداردی مانند FHS، نرم افزارها می توانند از چیدمان یکنواخت در توزیع های مختلف لینوکس اطمینان حاصل کند. با این حال، همه توزیع های لینوکس به طور کامل به این استاندارد پایبند نیستند. آنها اغلب عناصر منحصر به فرد خود را گنجانده یا نیازهای خاصی را برآورده می کنند. برای تسلط بر این استاندارد، می توانید با کاوش و کنجکاوی شروع کنید. از دستورات مانند "cd" برای پیمایش و "ls" برای لیست کردن محتویات فهرست (directory) استفاده کنید. سیستم فایل را به صورت یک درخت تصور کنید که از ریشه (/) شروع می شود. با گذشت زمان، برای شما به امری عادی تبدیل می شود و شما را به یک مدیر لینوکس ماهر تبدیل می کند. همین طور تصویر بالا در مورد فولدرهای اصلی لینوکس توضیح می دهد.

## ۱۸ دستور پر کاربرد لینوکس که باید بدانید

دستورات لینوکس دستورالعمل‌هایی برای تعامل با سیستم‌عامل هستند. آنها به مدیریت فایل‌ها، فهرست‌ها، فرایندهای سیستم و بسیاری از جنبه‌های دیگر سیستم کمک می‌کنند. برای اینکه به طور مؤثر و کارآمد در سیستم‌های مبتنی بر لینوکس پیمایش و نگهداری کنید، باید با این دستورات آشنا شوید.

این نمودار زیر دستورات محبوب لینوکس را نشان می‌دهد:

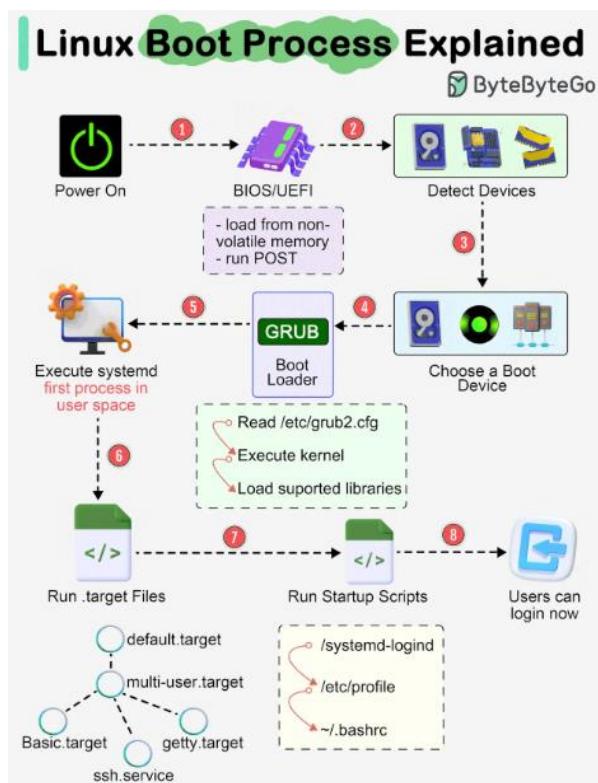
18 Most-Used Linux Commands You Should Know  blog.bytebytogo.com

<b>ls</b> List files and directories	<b>cd</b> Change current directory	<b>mkdir</b> Create new directory
<b>rm</b> Remove files or directories	<b>mv</b> Move or rename files or directories	<b>chmod</b> Change file or directory permission
<b>cp</b> Copy files or directories	<b>find</b> Search for files or directories	<b>grep</b> Search for a pattern in files
<b>vi</b> Edit files using text editor	<b>cat</b> Display the content of files	<b>tar</b> Manipulate tarball archive files
<b>ps</b> Display process information	<b>kill</b> Terminate process by sending a signal	<b>top</b> Display process and resource usage
<b>ifconfig</b> Configure network interfaces	<b>ping</b> Test network connectivity between hosts	<b>du</b> Estimate file space usage

- ls - لیست کردن فایل‌ها و فهرست‌ها •
- cd - تغییر دایرکتوری فعلی •
- mkdir - ایجاد یک فولدر جدید •
- rm - حذف فایل‌ها یا فولدرها •
- cp - کپی کردن فایل‌ها یا فولدرها •
- mv - جابه‌جایی یا تغییر نام فایل‌ها •
- chmod - تغییر مجوزهای فایل یا فولدر •
- grep - جستجو برای یک الگو در فایل‌ها •
- find - جستجو برای فایل‌ها و فولدرها •
- tar - مدیریت فایل‌های بایگانی tarball •
- vi - ویرایش فایل‌ها با استفاده از ویرایشگرهای متن •
- cat - نمایش محتوای فایل‌ها •
- top - نمایش فرایندها و میزان مصرف منابع •
- ps - نمایش اطلاعات فرایندها •
- kill - خاتمه‌دادن به یک فرایند با ارسال سیگنال •
- du - برآورد فضای استفاده شده توسط فایل‌ها •
- ifconfig - پیکربندی رابطهای شبکه •
- ping - تست اتصال شبکه بین میزبان‌ها •

## فرایند بوت لینوکس به زبان ساده

تقریباً همه مهندسان نرم افزار قبلاً از لینوکس استفاده کرده‌اند، اما تنها عده کمی می‌دانند فرایند بوت آن چگونه کار می‌کند. باید نگاهی به آن بیندازیم. این نمودار زیر مراحل را نشان می‌دهد:



۱. هنگامی که برق را روشن می‌کنیم، بایوس (سیستم ورودی/خروجی پایه) یا سیستم عامل UEFI (رابط میان افزار توسعه‌پذیر یکپارچه) از حافظه غیر فرار بارگیری شده و تست خودکار هنگام روشن شدن قسمت POST را اجرا می‌کند.

۲. بایوس/UEFI دستگاه‌های متصل به سیستم، از جمله CPU، رم و حافظه ذخیره‌سازی را شناسایی می‌کند.
۳. یک دستگاه راهاندازی را برای بوت کردن سیستم عامل انتخاب کنید. این می‌تواند هارددیسک، سرور شبکه یا CD رام باشد.
۴. بایوس/UEFI بوت لودر (GRUB) را اجرا می‌کند که منوی برای انتخاب سیستم عامل یا توابع هسته ارائه می‌دهد.
۵. پس از اینکه هسته آماده شد، اکنون به فضای کاربر (user space) سوئیچ می‌کنیم. هسته systemd را به عنوان اولین فرایند فضای کاربر راهاندازی می‌کند که فرایندها و سرویس‌ها را مدیریت می‌کند، تمام سخت‌افزارهای باقیمانده را بررسی می‌کند، سیستم‌های فایل را سوار می‌کند و یک محیط دسکتاپ را اجرا می‌کند.
۶. systemd به طور پیش‌فرض واحد پیش‌فرض (default.target) را هنگام بوت شدن سیستم فعال می‌کند. سایر واحدهای تجزیه و تحلیل نیز اجرا می‌شوند.
۷. سیستم مجموعه‌ای از اسکریپت‌های راهاندازی را اجرا می‌کند و محیط را پیکربندی می‌کند.
۸. یک پنجره ورود به سیستم به کاربران نمایش داده می‌شود. حالا سیستم آماده است.

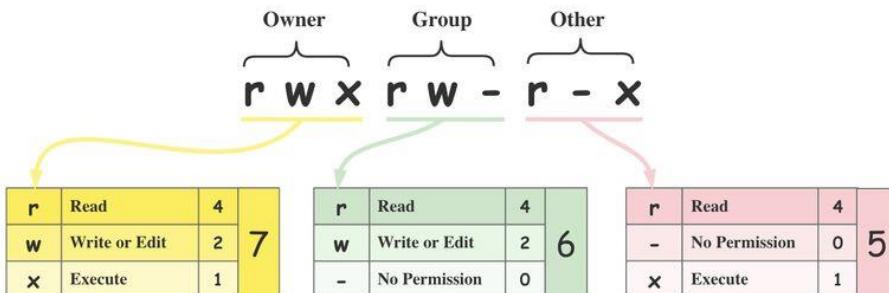
## مجوزهای فایل‌های لینوکس

برای درک مجوزهای فایل لینوکس، باید مفاهیم مالکیت<sup>۱</sup> و مجوز<sup>۲</sup> را درک کنیم.

### Linux File Permissions

 blog.bytebytogo.com

Binary	Octal	String Representation	Permissions
000	0 (0+0+0)	---	No Permission
001	1 (0+0+1)	--x	Execute
010	2 (0+2+0)	-w-	Write
011	3 (0+2+1)	-wx	Write + Execute
100	4 (4+0+0)	r--	Read
101	5 (4+0+1)	r-x	Read + Execute
110	6 (4+2+0)	r-w-	Read + Write
111	7 (4+2+1)	rwx	Read + Write + Execute



مالکیت:

هر فایل یا دایرکتوری دارای سه نوع سطح مالکیت است:

Ownership<sup>۱</sup>  
Permission<sup>۲</sup>

- **مالک<sup>۱</sup>**: مالک، کاربری است که فایل یا دایرکتوری را ایجاد کرده است.
- **گروه**: یک گروه می‌تواند کاربران متعددی داشته باشد. همه کاربران گروه مجوزهای یکسانی برای دسترسی به فایل یا دایرکتوری دارند.
- **سایر<sup>۲</sup>**: سایر به معنای کاربرانی است که مالک یا عضو گروه نیستند.

**مجوز:**

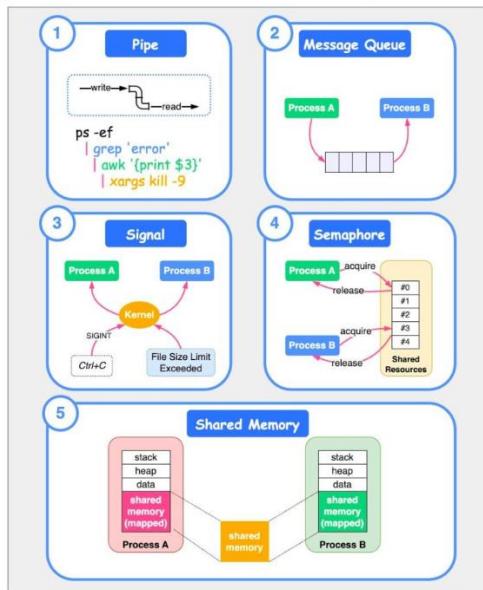
فقط سه نوع مجوز برای یک فایل یا دایرکتوری وجود دارد:

- **r - Read**: مجوز خواندن به کاربر اجازه می‌دهد تا یک فایل را بخواند.
- **w - Write**: مجوز نوشتن به کاربر اجازه می‌دهد تا محتوای فایل را تغییر دهد.
- **x - Execute**: مجوز اجرا به یک فایل اجازه می‌دهد تا اجرا شود.

## ارتباط بین فرایندها IPC

ارتباط بین فرایندها<sup>۱</sup> (IPC) به روش‌هایی اشاره دارد که به فرایندها/پراسس‌ها اجازه می‌دهد داده‌ها را تبادل کرده و اعمال را هماهنگ کنند. در اینجا، به بررسی مهم‌ترین مکانیزم‌های IPC در سیستم‌عامل‌های شبیه یونیکس مانند لینوکس می‌پردازیم.

### 5 Inter-Process Communications [blog.bytebytogo.com](http://blog.bytebytogo.com)



### Pipes

لوله‌ها ورودی و خروجی دو یا چند process را به هم متصل می‌کنند و یک pipeline برای جریان داده ایجاد می‌کنند. به عنوان مثال، اسکریپت‌های shell به طور معمول با استفاده از عملگر pipe، دستورات را به هم زنجیر می‌کنند:

`cat /var/log/syslog | grep 'error' | less`

### صف‌های پیام (Message Queues)

صف‌های پیام با امکان تبادل داده بین process‌ها به شکل پیام‌رسانی، امکان ارتباط ناهم‌زمان را فراهم می‌کنند. پیام‌های نوشته شده در یک صف به ترتیب first-in, first-out یا به صور

خلاصه (FIFO) مدیریت می‌شوند. به عنوان مثال، یک سرور ممکن است کارها را به صفحه‌ای ارسال کند که<sup>۱</sup> worker daemons آن‌ها را از صفحه خارج کرده و به طور مستقل پردازش می‌کنند.

### سیگنال‌ها (Signals)

سیگنال‌ها یک سیستم هشداردهنده را فراهم می‌کنند که به process‌ها اجازه می‌دهد تا فوراً از رویدادهای مهم مانند خاتمه‌ی اجباری مطلع شوند. به عنوان مثال، SIGKILL روشی مطمئن برای پایان دادن به برنامه‌های بی‌پاسخ ارائه می‌دهد.

### سمافورها (Semaphores)

سمافورها با محدود کردن استفاده هم‌زمان به همگام‌سازی دسترسی به منابعی که باید بین process‌ها به اشتراک گذاشته شوند، کمک می‌کنند. آن‌ها به جلوگیری از وضعیت رقابتی<sup>۲</sup> ناشی از خواندن و نوشتمن هم‌زمان process‌ها به منابعی مانند فایل‌های داده‌ی مشترک کمک می‌کنند.

### حافظه‌ی مشترک (Shared Memory)

حافظه‌ی مشترک دسترسی مستقیم به مناطق حافظه‌ی مشترک را فراهم می‌کند، بنابراین چندین فرایند می‌توانند بدون کپی کردن، داده‌ها را به طور کارآمد خوانده و اصلاح کنند. یک نمونه‌ی کاربردی، برنامه‌ای است که یک فریم تصویر بزرگ را دست‌کاری می‌کند، جایی که چندین فرایند می‌توانند به طور هم‌زمان به بخش‌های مختلف تصویر دسترسی پیدا کرده و آن‌ها را پردازش کنند.

همین‌طور این مکانیزم‌های IPC گاهی اوقات برای امکان برقراری ارتباط کارآمد بین فرایندها ترکیب می‌شوند.

<sup>۱</sup> در سیستم‌عامل‌های کامپیوتری چندوظیفه‌ای، دیمون یک برنامه کامپیوتری است که به جای اینکه تحت کنترل مستقیم یک کاربر تعاملی باشد، به عنوان یک فرآیند پس زمینه اجرا می‌شود.

<sup>۲</sup> race conditions

## برگه تقلب تجزیه لگ

نمودار زیر لیست ۶ دستور برتر برای تجزیه لگ را نمایش می‌دهد.

### Top 6 Log Parsing Commands



 <b>GREP</b>	<b>\$grep &lt;pattern&gt; file.log</b> GREP searches any given input files, selecting lines that match one or more patterns	1 find file names that match <code>\$grep -l "bytebybytego" *.log</code> 2 case insensitive word match <code>\$grep -wi "bytebybytego" test.log</code> 3 show line numbers <code>\$grep -n "bytebybytego" test.log</code>	4 invert matches <code>\$grep -v "bytebybytego" test.log</code> 5 take patterns from a file <code>\$grep -f pattern.txt test.log</code> 6 search recursively in a dir <code>\$grep -R "bytebybytego"/home</code>
 <b>CUT</b>	<b>\$cut -d"," -f 3 file.log</b> CUT cuts out selected portions of each line from each file and writes them to the standard output.	1 cut first 3 bytes <code>\$cut -b 1,2,3 file.log</code> 2 select 2nd column delimited by a space <code>\$cut -d " " -f 2 test.log</code>	3 specify characters position <code>\$cut -c 1-8 test.log</code>
 <b>SED</b>	<b>\$sed s/&lt;regex&gt;/&lt;replace&gt;/g</b> SED reads the specified files, modifying the input as specified by a list of commands.	1 substitute a string <code>\$sed s/bytebybytego/go/g test.log</code> 2 replace the 2nd occurrence <code>\$sed s/bytebybytego/go/2 test.log</code> 3 replace string on the 4th line <code>\$sed '4 s/bytebybytego/go/' test.log</code>	4 replace string on a range of lines <code>\$sed '2-4 s/bytebybytego/go/' test.log</code> 5 delete a line <code>\$sed '4d' test.log</code>
 <b>AWK</b>	<b>\$awk{print \$4} test.log</b> AWK scans each input file for lines that match any of a set of patterns.	1 print matched lines <code>\$awk '/bytebybytego/ {print}' test.log</code> 2 split a line into fields <code>\$awk '{print \\$1,\\$3}' test.log</code> 3 print lines 2 to 7 <code>\$awk 'NR==2, NR==7 {print NR, \\$0}' test.log</code>	4 print lines with more than 10 characters <code>\$awk 'length(\\$0)&gt;10' test.log</code> 5 find a string in a column <code>\$awk '{ if(\\$4=="byte" print \\$0; }' test.log</code>
 <b>SORT</b>	<b>\$sort test.log</b> SORT sorts text and binary files by lines.	1 output to a file <code>\$sort -o output.txt input.txt</code> 2 sort in reverse order <code>\$sort -r test.log</code> 3 sort numerically <code>\$sort -n test.log</code>	4 sort based on the 3rd column <code>\$sort -k 3n test.log</code> 5 check if a file is ordered <code>\$sort -c test.log</code> 6 sort and remove duplicates <code>\$sort -u test.log</code>
 <b>UNIQ</b>	<b>\$uniq test.log</b> UNIQ reads the specified input file comparing adjacent lines, and writes a copy of each unique input line to the output file.	1 tell how many times a line is repeated <code>\$uniq -c test.log</code> 2 print repeated lines <code>\$uniq -d test.log</code> 3 print unique lines <code>\$uniq -u test.log</code>	4 skip the first two fields <code>\$uniq -f 2 test.log</code> 5 compare case-insensitive <code>\$uniq -i test.log</code>

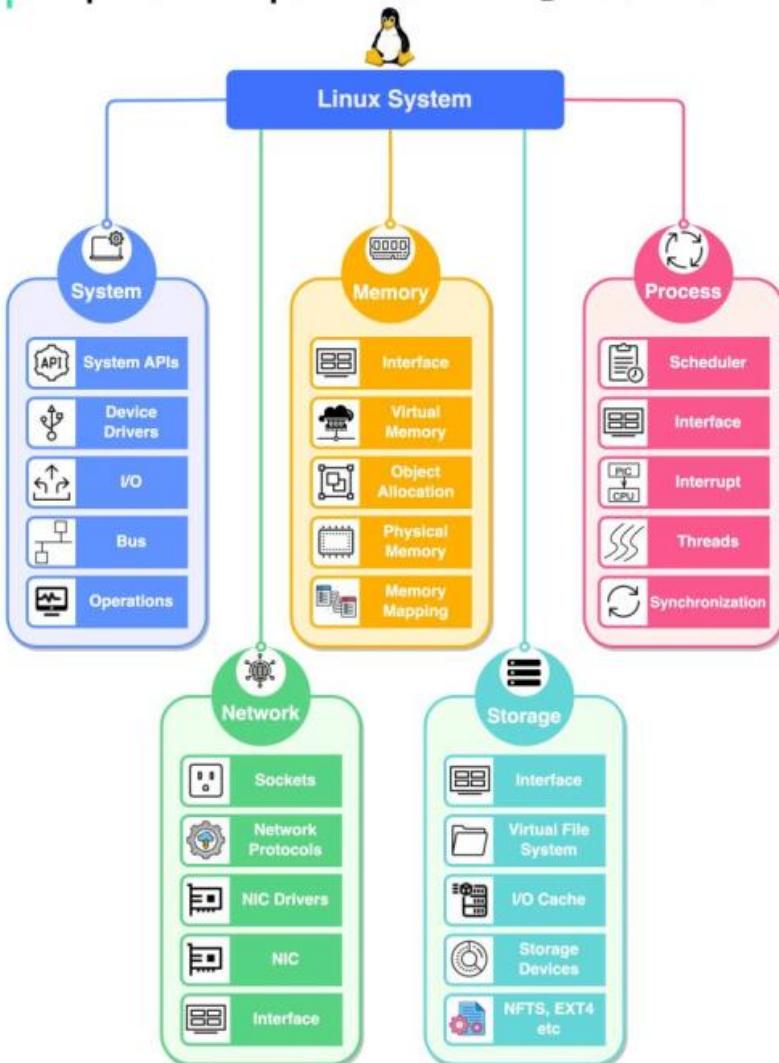
۱. **grep** هر فایل ورودی داده شده را جستجو می کند و خطوطی را که با یک یا چند الگو مطابقت دارند را انتخاب می کند.
۲. **cut** بخش های انتخاب شده از هر خط در هر فایل را جدا کرده و آنها را در خروجی و به صورت استاندارد می نویسد.
۳. **sed** فایل های مشخص شده را می خواند و ورودی را مطابق با لیستی از دستورات اصلاح می کند.
۴. **awk** هر فایل ورودی را برای خطوطی که با یک از مجموعه الگوها مطابقت دارند اسکن می کند.
۵. **sort** فایل های متنی و باینری را بر اساس خطوط مرتب می کند.
۶. **uniq** فایل ورودی مشخص شده را می خواند، خطوط مجاور را مقایسه می کند و یک کپی از هر خط ورودی منحصر به فرد را به فایل خروجی می نویسد.

این دستورات اغلب برای یافتن سریع اطلاعات مفید از فایل های لاغ به صورت ترکیبی استفاده می شوند. برای مثال، دستورات زیر timestamps (ستون ۲) را لیست می کنند که یک خطای استثنا (Exception) برای سرویس xxService رخداده است.

```
grep "xxService" service.log | grep "Exception" | cut -d" " -f2
```

## ۵ اصل مهم در لینوکس

### 5 important components of Linux [blog.bytebytogo.com](http://blog.bytebytogo.com)



• سیستم

در مؤلفه سیستم، باید ماثولهایی مانند API‌های سیستم، درایورهای دستگاه، ورودی/خروجی، گذرگاهها و غیره را بیاموزیم.

• حافظه

در مدیریت حافظه، باید ماثولهایی مانند حافظه فیزیکی، حافظه مجازی، نگاشتهای حافظه، تخصیص اشیاء و غیره را بیاموزیم.

• process

در مدیریت process، باید ماثولهایی مانند process scheduling، thread ها، وقفه‌ها، همگام‌سازی و غیره را بیاموزیم.

• شبکه

در مؤلفه شبکه، باید ماثولهای مهمی مانند پروتکل‌های شبکه، سوکت‌ها، درایورهای NIC و غیره را بیاموزیم.

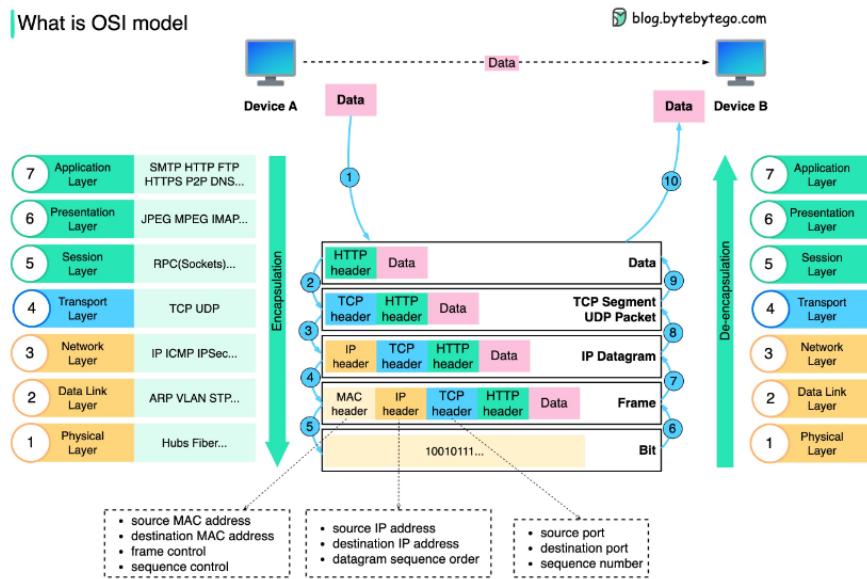
• ذخیره‌سازی

در مدیریت ذخیره‌سازی سیستم، باید ماثولهایی مانند file system، I/O cache، دستگاه‌های ذخیره‌سازی مختلف، پیاده‌سازی‌های سیستم فایل و غیره را بیاموزیم.

# بررسی مباحث شبکه

## آشنایی با TCP/IP و مدل OSI

داده‌ها چگونه از طریق شبکه ارسال می‌شوند؟ چرا به این‌همه لایه در مدل OSI نیاز داریم؟  
شکل زیر نحوه کپسوله‌سازی و غیر کپسوله‌سازی داده هنگام انتقال از طریق شبکه را نشان می‌دهد.



مرحله ۱: هنگامی که دستگاه A داده را از طریق پروتکل HTTP از طریق شبکه به دستگاه B ارسال می‌کند، ابتدا یک هدر HTTP در لایه application به آن اضافه می‌شود.  
مرحله ۲: سپس یک هدر TCP یا UDP به داده اضافه می‌شود. در لایه انتقال (transport) به بخش‌های TCP کپسوله می‌شود. هدر حاوی پورت مبدأ، پورت مقصد و شماره دنباله است.

مرحله ۳: سپس قطعات با یک هدر IP در لایه شبکه کپسوله می‌شوند. هدر IP حاوی آدرس های IP مبدأ/مقصد است.

مرحله ۴: به دیتاگرام IP یک هدر MAC در لایه پیوند داده (data link) اضافه می‌شود که حاوی آدرس های MAC مبدأ/مقصد است.

مرحله ۵: فریم های کپسوله شده به لایه فیزیکی (physical layer) ارسال شده و به صورت بیت های باینری از طریق شبکه ارسال می‌شوند.

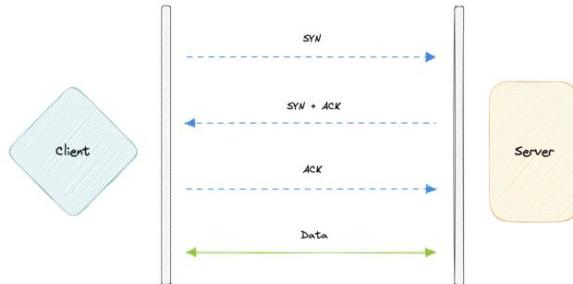
مراحل ۶ تا ۱۰: هنگامی که دستگاه B بیت‌ها را از شبکه دریافت می‌کند، فرایند دیکپسوله سازی را انجام می‌دهد که فرآیندی معکوس فرایند کپسوله سازی است. هدرها به صورت لایه به لایه حذف می‌شوند و در نهایت دستگاه B می‌تواند داده را بخواند.

ما به لایه‌ها در مدل شبکه نیاز داریم؛ زیرا هر لایه روی مسئولیت‌های خاص خود تمرکز می‌کند. هر لایه می‌تواند برای دستورالعمل‌های پردازش روی هدرها تکیه کند و نیازی به دانستن معنای داده از لایه آخر ندارد.

## تفاوت UDP و TCP

### TCP

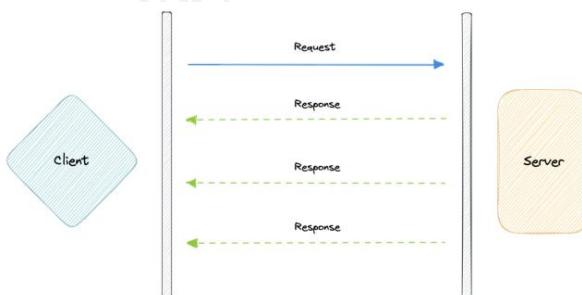
TCP<sup>۱</sup> مبتنی بر اتصال است ، به این معنی که پس از برقراری ارتباط، داده‌ها می‌توانند در هر دو جهت منتقل شوند. TCP دارای سیستم‌های داخلی برای بررسی خطاهای خطاها و تضمین اینکه داده‌ها به ترتیب ارسال شده تحويل داده می‌شوند که آن را به پروتکل مناسبی برای انتقال اطلاعات مانند تصاویر ثابت، فایل‌های داده و صفحات وب تبدیل می‌کند.



اما در حالی که TCP به طور ذاتی قابل اعتماد است، مکانیسم‌های بازخورد آن نیز منجر به سریال بزرگتر می‌شود که به معنای استفاده بیشتر از پهنای باند موجود در شبکه است.

### UDP

UDP<sup>۱</sup> یک پروتکل اینترنتی ساده‌تر و بدون اتصال است که در آن به خدمات بررسی خطای بازیابی نیازی نیست. با UDP، هیچ هزینه‌ای برای باز کردن یک اتصال<sup>۲</sup>، حفظ اتصال یا خاتمه اتصال وجود ندارد. داده‌ها به طور مداوم برای گیرنده ارسال می‌شود، چه آنها آن را دریافت کنند یا نه.



این مدل ارتباطی تا حد زیادی برای ارتباطات بلاذرنگ مانند پخش یا انتقال شبکه چندپخشی ترجیح داده می‌شود. زمانی که به کمترین تأخیر نیاز داریم باید از UDP روی TCP استفاده کنیم، در زمانی که دیر رسیدن داده‌ها بدتر از ازدستدادن داده‌ها هستند.

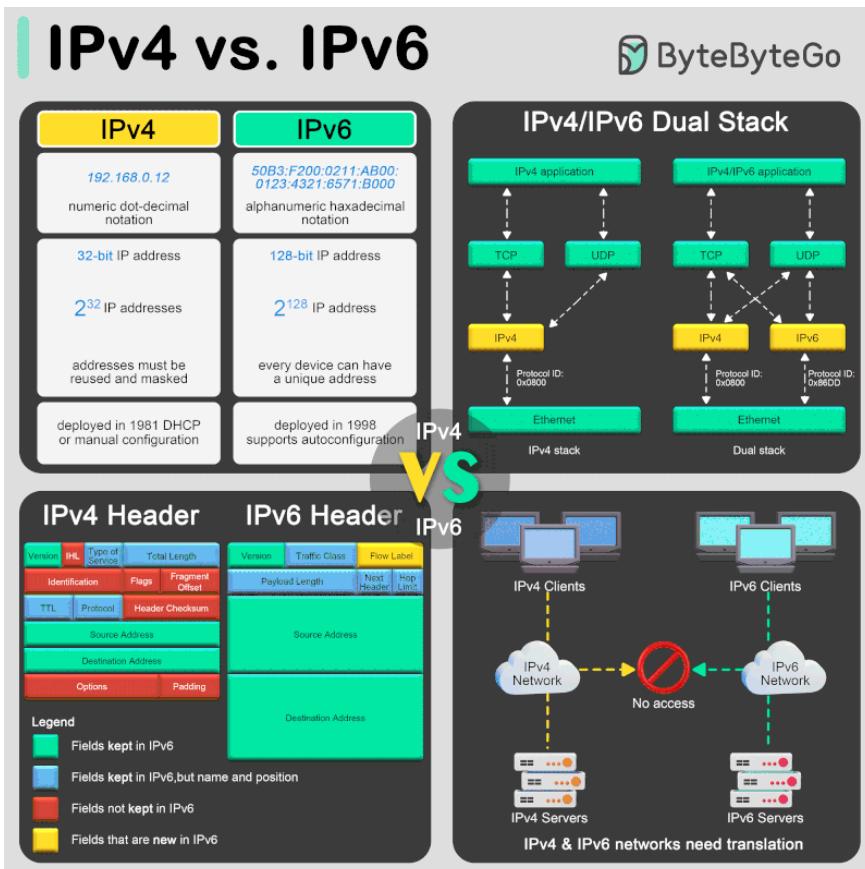
TCP یک پروتکل مبتنی بر اتصال<sup>۱</sup> است، در حالی که UDP یک پروتکل بدون اتصال<sup>۲</sup> است. یکی از تفاوت‌های کلیدی بین TCP و UDP سرعت است، زیرا TCP کندتر از UDP است. به طور کلی، UDP پروتکلی بسیار سریع‌تر، ساده‌تر و کارآمدتر است، اما باز ارسال بسته‌های Dاده از دست رفته تنها با TCP امکان‌پذیر است.

TCP تحويل مرتب داده‌ها را از کاربر به سرور (و بالعکس) ارائه می‌دهد، در حالی که UDP به ارتباطات انتهای‌به‌نهای اختصاص ندارد و آمادگی گیرنده را نیز بررسی نمی‌کند.

UDP	TCP	ویژگی
بدون نیاز به برقراری ارتباط	نیازمند برقراری ارتباط	برقراری ارتباط
عدم تضمین تحويل داده	امکان تضمین تحويل داده	تضمین تحويل
عدم بازارسال بسته‌های ازدست رفته	امکان بازارسال بسته‌های ازدست رفته	بازارسال
Serious	کندتر از UDP	سرعت
پشتیبانی می‌کند	پشتیبانی نمی‌کند	پخش همگانی
پخش ویدئو، DNS، VoIP و غیره	HTTPS، HTTP، SMTP، POP، FTP	موارد استفاده

## IPv4 در مقابل IPv6، تفاوت‌ها چیست؟

گذار از پروتکل اینترنت نسخه ۴ (IPv4) به پروتکل اینترنت نسخه ۶ (IPv6) عمدتاً به دلیل نیاز به آدرس‌های اینترنتی بیشتر، در کنار تمایل به ساده سازی برخی جنبه‌های مدیریت شبکه هدایت می‌شود.



- قالب و طول

IPv4 از یک قالب آدرس ۳۲ بیتی استفاده می‌کند که معمولاً به صورت چهار عدد دهدهی جدا شده با نقطه نمایش داده می‌شود (مثلاً ۱۹۲.۱۶۸.۰.۱۲). قالب ۳۲ بیتی تقریباً به ۴.۳

میلیارد آدرس منحصر به فرد اجازه می‌دهد، عددی که به دلیل انفجار دستگاه‌های متصل به اینترنت به سرعت در حال تکمیل شدن است.

در مقابل، IPv6 از یک قالب آدرس ۱۲۸ بیتی استفاده می‌کند که با هشت گروه از چهار رقم هگزادسیمال جدا شده با کولون (:) (مثال ۵۰B3:F200:0211:AB00:0123:4321:6571:B000) نشان داده می‌شود. این روش امکان آدرس دهی بسیار بیشتری را فراهم می‌کند و اطمینان می‌دهد که رشد اتصالات به اینترنت بدون وقفه و مشکل ادامه یابد.

### Header •

Header‌ها در پروتکل IPv4 پیچیده‌تر هستند و شامل فیلد‌هایی مانند طول header، نوع سرویس، طول کل، شناسایی، پرچم‌ها، جبران قطعه، زمان عمر (TTL)، پروتکل، checksum، header، آدرس‌های IP مبدأ و مقصد و گزینه‌ها می‌شود.

Header‌های موجود در IPv6 برای ساده‌تر و کارآمدتر بودن طراحی شده‌اند. اندازه Header‌ها ثابت و حدود ۴۰ بایت است و شامل فیلد‌های کمتر استفاده شده در Header‌های الحقیقی و اختیاری است. فیلد‌های اصلی شامل نسخه، کلاس ترافیک، برچسب مسیر<sup>۱</sup>، طول payload، هدر بعدی، محدودیت hop و آدرس‌های مبدأ و مقصد هستند. این ساده‌سازی به بهبود سرعت پردازش packet کمک می‌کند.

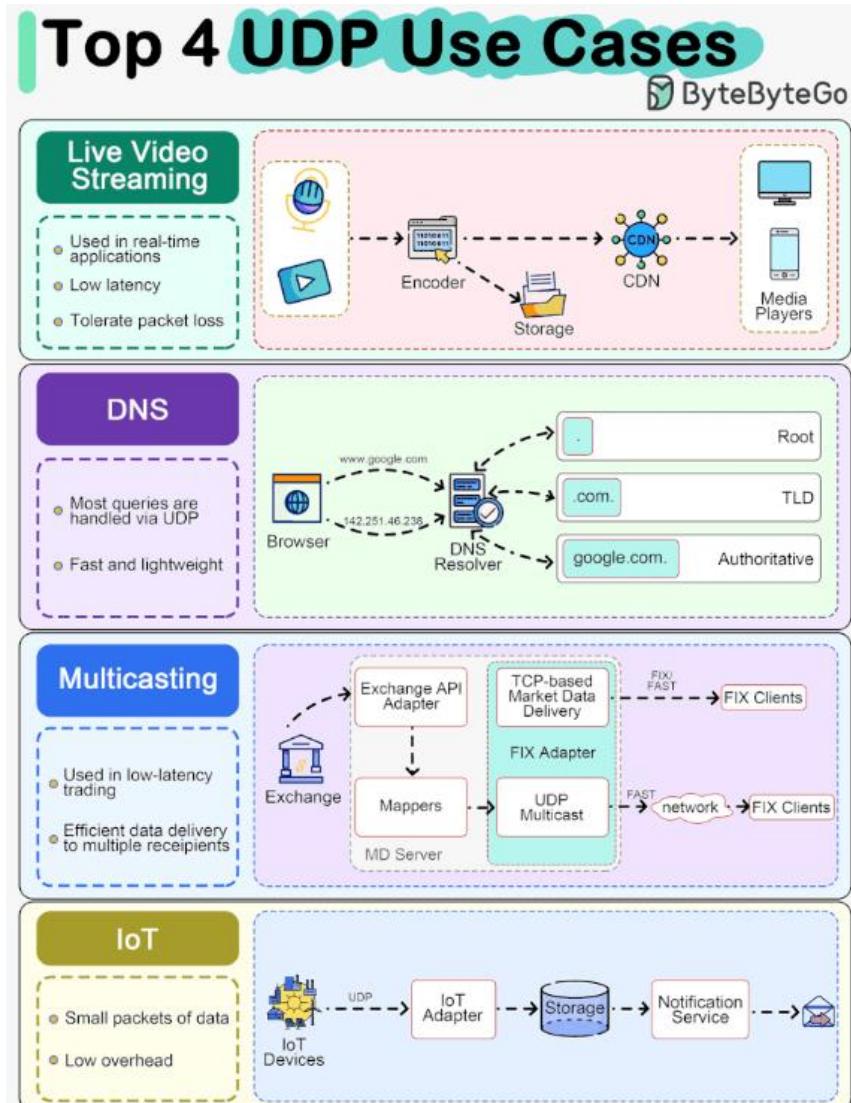
### • ترجمه‌ی داده بین IPv4 و IPv6

با انتقال اینترنت از IPv4 به IPv6، مکانیزم‌هایی برای هم‌زیستی این پروتکل‌ها ضروری شده است:

Dual Stack – این تکنیک شامل اجرای هم‌زمان IPv4 و IPv6 روی دستگاه‌های مشابهی در شبکه است. این امر به ارتباط یکپارچه در هر دو پروتکل، بسته به دردسترس بودن و سازگاری آدرس مقصد آن‌ها اجازه این تبدیل را می‌دهد. Dual Stack یکی از بهترین روش‌های برای انتقال روان از IPv4 به IPv6 در نظر گرفته می‌شود.

## ۴ مورد از محبوب‌ترین سناریوهای کاربردی برای UDP

UDP به دلیل سادگی، سرعت و سریار کم آن نسبت به پروتکل‌های دیگر مانند TCP، در معماری‌های مختلف نرم‌افزاری مورد استفاده قرار می‌گیرد.



### استریم ویدئوی زنده

بسیاری از برنامه‌های ارتباط تصویری و کنفرانس صوتی به دلیل سربار کمتر و توانایی تحمل از دست رفتن بسته‌های داده از UDP استفاده می‌کنند. ارتباطات به صورت بلاذرنگ و با کاهش تأخیر UDP نسبت به TCP بهره‌مند می‌شوند.

### DNS

در خواسته‌های سرویس نام دامنه (DNS) معمولاً برای ماهیت سریع و سبک بودن از UDP استفاده می‌کنند. اگرچه DNS می‌تواند برای پاسخ‌های بزرگ یا انتقال منطقه از TCP نیز استفاده کند، اما اکثر پرسش‌ها از طریق UDP مدیریت می‌شوند.

### چند پخشی<sup>۱</sup> داده‌های بازار

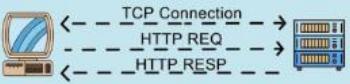
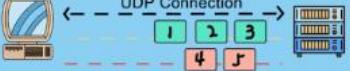
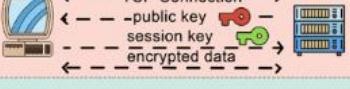
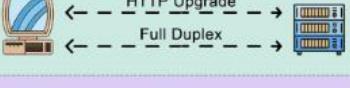
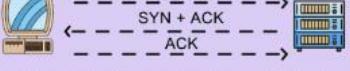
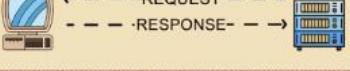
در معاملات با تأخیر کم، UDP برای ارسال کارآمد داده‌های بازار به طور همزمان به چندین گیرنده مورد استفاده قرار می‌گیرد.

### ایترنت اشیاء

UDP اغلب در دستگاه‌های ایترنت اشیاء برای برقراری ارتباط و ارسال بسته‌های کوچک داده بین دستگاه‌ها استفاده می‌شود.

## ۸ پروتکل محبوب در شبکه

پروتکل‌های شبکه روش‌های استانداردی برای انتقال داده بین دو رایانه در یک شبکه هستند.

Protocol	How does It Work?	Use Cases
HTTP	 <p>TCP Connection HTTP REQ HTTP RESP</p>	 <p>Web Browsing</p>
HTTP/3 (QUIC)	 <p>UDP Connection 1 2 3 4 5</p>	 <p>IoT Virtual Reality</p>
HTTPS	 <p>TCP Connection public key session key encrypted data</p>	 <p>Web Browsing</p>
WebSocket	 <p>HTTP Upgrade Full Duplex</p>	 <p>Live Chat Real-Time Data Transmission</p>
TCP	 <p>SYN SYN + ACK ACK</p>	 <p>Web Browsing Email Protocols</p>
UDP	 <p>REQUEST RESPONSE</p>	 <p>Video Conferencing</p>
SMTP	 <p>sender      SMTP Server      receiver</p>	 <p>Sending/Receiving Emails</p>
FTP	 <p>Control Channel Data Channel</p>	 <p>Upload/Download Files</p>

### ۱. پروتکل انتقال ابر متن (HTTP)

HTTP پروتکلی برای دریافت منابع مانند اسناد HTML است. این اساس هرگونه تبادل داده در وب است و یک پروتکل مبتنی بر سرویس client-server است.

**HTTP/3.**

HTTP/3 نسخه بازنگری اصلی بعدی HTTP است. این نسخه بر روی QUIC اجرا می‌شود که یک پروتکل حمل و نقل جدید که برای استفاده از اینترنت با حجم بالای موبایل طراحی شده است. این پروتکل به جای TCP از UDP استفاده می‌کند که امکان پاسخگویی سریع‌تر صفحات وب را فراهم می‌کند. برنامه‌های واقعیت مجازی برای ریندر جزئیات پیچیده یک صحنه مجازی به پهنانی باند زیادی می‌کند. برنامه‌های پیش‌بینی QUIC با HTTP/3 بهره‌مند خواهند شد.

**HTTPS (HyperText Transfer Protocol) .۳**

HTTP پروتکل را گسترش می‌دهد و از رمزگذاری برای برقراری ارتباط امن استفاده می‌کند.

**WebSocket .۴**

WebSocket پروتکلی است که ارتباطات دو طرفه کامل را بر روی TCP ارائه می‌دهد. سرویس‌های کلاینت برای دریافت به روزرسانی‌های لحظه‌ای از سرویس‌های بک‌اند اقدام به ارتباطات WebSocket می‌کنند. برخلاف REST که همیشه داده‌ها را pull می‌کند، WebSocket امکان push داده‌ها را فراهم می‌کند. برنامه‌هایی مانند بازی‌های آنلاین، تجارت سهام و برنامه‌های پیام‌رسانی از WebSocket برای ارتباطات لحظه‌ای استفاده می‌کنند.

**TCP (Transmission Control Protocol) .۵**

TCP برای ارسال بسته‌ها در سراسر اینترنت و اطمینان از انتقال موفقیت‌آمیز داده و پیام در شبکه‌ها طراحی شده است. بسیاری از پروتکل‌های لایه کاربردی بر روی TCP ساخته می‌شوند.

**UDP (Transmission Control Protocol) .۶**

UDP معمولاً packet‌ها را بدون برقراری اتصال اولیه به طور مستقیم به یک رایانه هدف ارسال می‌کند. UDP به طور رایج در ارتباطات حساس به زمان استفاده می‌شود، جایی که گاهی اوقات حذف بسته‌ها بهتر از انتظار است. ترافیک صدا و تصویر اغلب با استفاده از این پروتکل ارسال می‌شود.

**SMTP (Simple Mail Transfer Protocol) .۷**

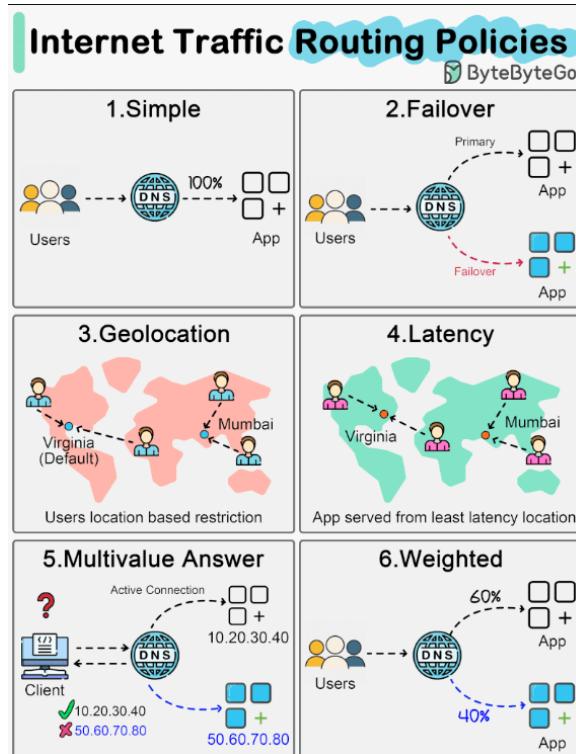
SMTP یک پروتکل استاندارد برای انتقال ایمیل الکترونیکی از یک کاربر به کاربر دیگر است.

**FTP (File Transfer Protocol) .۸**

FTP برای انتقال فایل‌های رایانه‌ای بین سرویس کلاینت و سرور استفاده می‌شود. این پروتکل دارای اتصالات جداگانه برای کانال کنترل و کانال داده است.

## مسیریابی ترافیک اینترنت

خط مشی های مسیریابی ترافیک اینترنت (DNS policies) نقش اساسی در مدیریت و هدایت کارآمد ترافیک شبکه ایفا می کنند. بیایید انواع مختلف سیاست ها را بررسی کنیم:



۱. ساده (Simple): کل ترافیک را بر اساس یک پرس و جوی / کوئری استاندارد DNS

بدون هیچ شرایط یا الزامات خاصی به یک endpoint هدایت می کند.

۲. جایگزین (Failover): ترافیک را به یک endpoint اصلی هدایت می کند، اما در

صورت عدم دسترسی به endpoint اصلی، به طور خودکار به یک endpoint ثانویه سوئیچ می کند.

۳. موقعیت جغرافیایی (Geolocation): ترافیک را بر اساس موقعیت جغرافیایی

درخواست کننده توزیع می کند تا محتوای محلی یا سرویس هایی را ارائه دهد.

۴. تأخیر (Latency): ترافیک را به endpoint ای که کمترین تأخیر را برای

درخواست‌کننده فراهم می‌کند هدایت می‌کند و با زمان پاسخگویی سریع‌تر، تجربه کاربری را بهبود می‌بخشد.

۵. پاسخ چند مقداری (Multivalue Answer): به پرس‌وجوهای DNS با چندین آدرس IP پاسخ می‌دهد و به کلاینت اجازه می‌دهد تا یک endpoint خاصی را

انتخاب کند. با این حال، نباید آن را جایگزینی برای توزیع‌کننده بار در نظر گرفت.

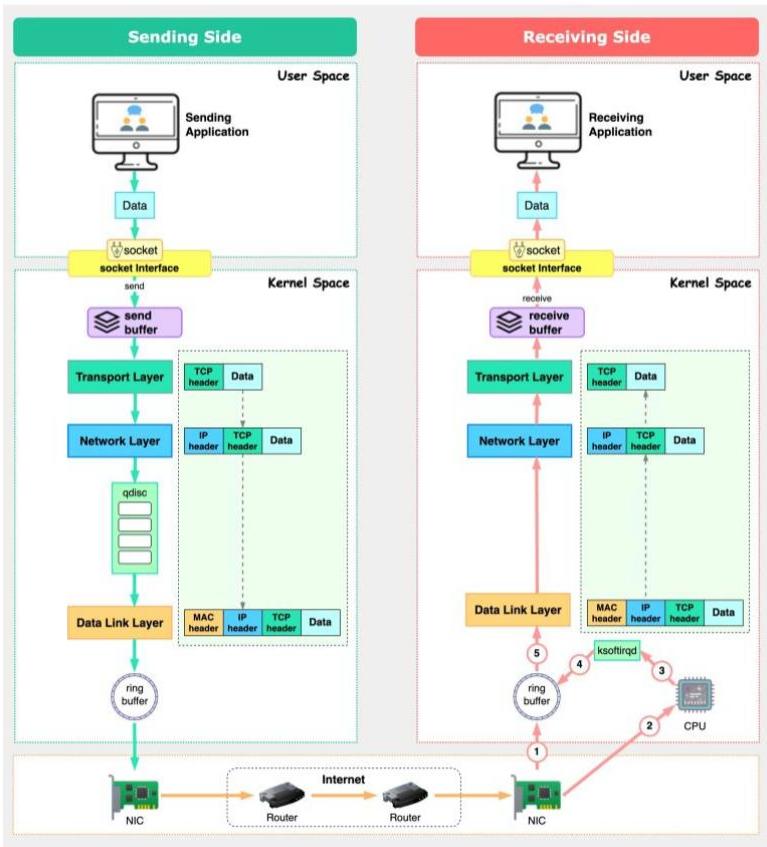
۶. خطمشی مسیریابی وزنی (Weighted Routing Policy): ترافیک را با وزن‌های اختصاص‌داده شده در چندین endpoint توزیع می‌کند و امکان توزیع متناسب

ترافیک بر اساس این وزن‌ها را فراهم می‌کند.

### داده‌ها چگونه بین برنامه‌ها منتقل می‌شوند؟

نمودار زیر نشان می‌دهد که چگونه یک سرور داده را به سرور دیگری ارسال می‌کند. فرض کنید یک برنامه چت که در فضای کاربری اجرا می‌شود، یک پیام چت ارسال می‌کند. پیام به بافر ارسال در فضای کرنل ارسال می‌شود. سپس داده‌ها از طریق پشته شبکه عبور می‌کنند و با یک هدر TCP، یک هدر IP و یک هدر MAC بسته‌بندی می‌شوند. داده‌ها همچنین از طریق qdisc<sup>1</sup> (Queueing Disciplines) برای کنترل جریان و مسیر حرکت NIC (Network Interface) خود، عبور می‌کنند. سپس داده‌ها از طریق یک بافر حلقوی به NIC (Network Interface) ارسال می‌شوند. داده‌ها از طریق NIC به اینترنت ارسال می‌شوند. پس از پرش‌های Card متعدد بین روترهای و سوئیچ‌ها، داده‌ها به NIC سرور گیرنده می‌رسند.

<sup>1</sup> مخفف queueing discipline است و در واقع یک مکانیزم هسته لینوکس برای مدیریت ترافیک شبکه است. به عبارت ساده‌تر، qdisc تعیین می‌کند که بسته‌های داده‌ای چگونه در صفحه انتظار قرار بگیرند و با چه ترتیبی ارسال شوند. این مکانیزم به شما اجازه می‌دهد تا ترافیک شبکه را شکل‌دهی، اولویت‌بندی و کنترل کنید.



NIC در سرور گیرنده داده‌ها را در بافر حلقوی قرار می‌دهد و یک وقفه سخت<sup>۱</sup> به CPU ارسال می‌کند CPU. یک وقفه نرم<sup>۲</sup> ارسال می‌کند تا ksoftirqd<sup>۳</sup> داده‌ها را از بافر حلقوی دریافت کند. سپس داده‌ها از طریق لایه پیوند داده، لایه شبکه و لایه انتقال باز می‌شوند. درنهایت، داده‌ها (پیام چت) به فضای کاربر کپی می‌شوند و به برنامه چت در سمت گیرنده می‌رسند.

hard interrupt<sup>۱</sup>

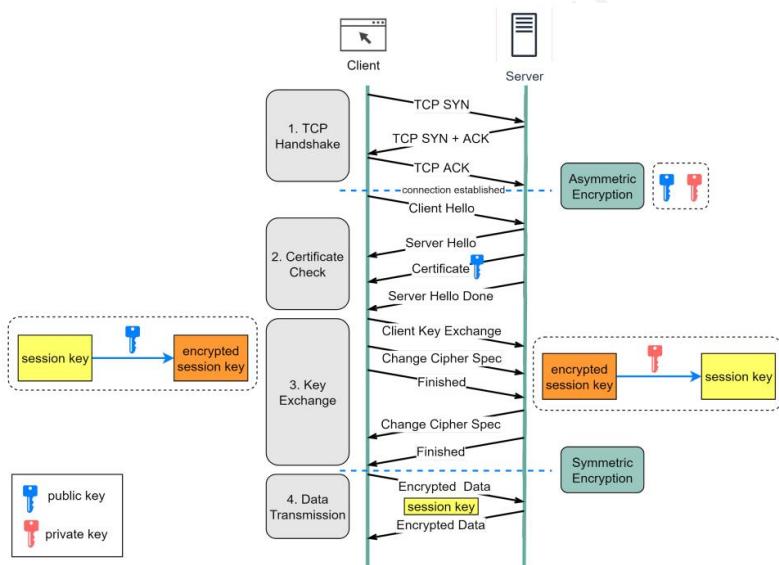
soft interrupt<sup>۲</sup>

ksoftirqd<sup>۳</sup> یک فرآیند هسته‌ای در لینوکس است که برای مدیریت بار سنگین وقفه‌های نرم‌افزاری طراحی شده است. وقفه‌های نرم‌افزاری، برخلاف وقفه‌های سخت‌افزاری که توسط سخت‌افزار ایجاد می‌شوند، توسط خود هسته تولید می‌شوند.

## بررسی امنیت در وب

### چگونه HTTPS کار می کند؟

پروتکل انتقال هایپرтекست امن (HTTPS<sup>1</sup>) یک گسترش از پروتکل HTTP است. HTTPS داده ها را با استفاده از امنیت لایه انتقال (TLS) به صورت رمزگاری شده منتقل می کند. اگر داده های آنلاین دزدیده شوند، همه چیزی که دزد دریافت می کند کد دودویی غیرقابل فهم است.



### چگونه داده ها رمزگاری و رمزگشایی می شوند؟

مرحله ۱ - کلاینت (مرورگر) و سرور یک اتصال TCP برقرار می کنند.  
 مرحله ۲ - کلاینت یک "hello client" به سرور می فرستد. پیام شامل مجموعه ای از الگوریتم های رمزگاری ضروری (مجموعه های رمز) و آخرین نسخه TLS که می تواند

Hypertext Transfer Protocol Secure <sup>1</sup>

پشتیبانی کند، است. سرور با یک "hello server" پاسخ می‌دهد تا مرورگر بداند آیا می‌تواند الگوریتم‌ها و نسخه TLS را پشتیبانی کند یا خیر. سپس سرور گواهینامه SSL را به کلاینت ارسال می‌کند. گواهینامه شامل کلید عمومی، نام میزبان، تاریخ‌های انقضا وغیره است. کلاینت گواهینامه را تأیید می‌کند.

مرحله ۳ - پس از تأیید گواهینامه SSL، کلاینت یک session key را تولید می‌کند و آن را با استفاده از کلید عمومی رمزنگاری می‌کند. سرور session key رمزنگاری شده را دریافت کرده و با کلید خصوصی آن را رمزگشایی می‌کند.

مرحله ۴ - حالا که هر دوی کلاینت و سرور همان session key را (رمزنگاری متقارن) در اختیار دارند، داده‌های رمزنگاری شده در یک کانال دو طرفه امن منتقل می‌شوند.

چرا HTTPS در طول انتقال داده‌ها به رمزنگاری متقارن تغییر می‌کند؟

دو دلیل اصلی وجود دارد:

امنیت: رمزنگاری نامتقارن تنها در یک جهت عمل می‌کند. این بدان معناست که اگر سرور بخواهد داده‌های رمزنگاری شده را به کلاینت برگرداند، هر کسی می‌تواند داده‌ها را با استفاده از کلید عمومی رمزگشایی کند.

منابع سرور: رمزنگاری نامتقارن مقدار زیادی بار محاسباتی اضافه می‌کند. این روش برای انتقال داده‌ها در جلسات طولانی<sup>۱</sup> مناسب نیست.

بار عملکردی که HTTPS نسبت به HTTP اضافه می‌کند چقدر است؟

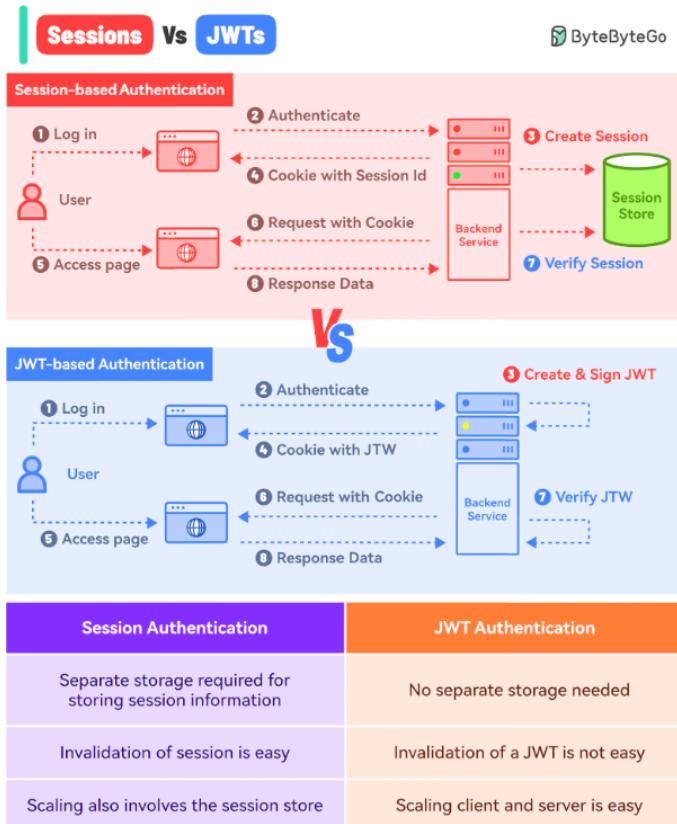
<sup>۱</sup> long sessions

پاسخ به سؤال شما در مورد بار عملکردی که HTTPS نسبت به HTTP اضافه می‌کند، به چند فاکتور بستگی دارد، اما معمولاً تفاوت قلبل توجهی در عملکرد وجود دارد. رمزنگاری و رمزگشایی داده‌ها در HTTPS به منابع CPU بیشتری نیاز دارد و ممکن است تأخیر کمی در زمان بارگذاری صفحه ایجاد کند. با این حال، با پیشرفت‌های اخیر در الگوریتم‌های رمزنگاری و بهبود در ساخت‌افزار، این تفاوت‌ها کمتر محسوس هستند.

به طور معمول، می‌توان انتظار داشت که HTTPS حدود ۱۰٪ بیشتر بار عملکردی نسبت به HTTP اضافه کند، اما این افزایش بار معمولاً به خاطر مزایای امنیتی که HTTPS فراهم می‌کند، قابل قبول است. افزایش امنیت، حفاظت از داده‌های حساس و جلوگیری از حملات مرد میانی (Man-in-the-Middle) ارزش این بار اضافی را دارد.

## تفاوت بین تأیید اعتبار مبتنی بر JWT و Session چیست؟

این متن به طور خلاصه هر دو رویکرد را توضیح می‌دهد:



### تأیید اعتبار مبتنی بر Session

در این روش، اطلاعات مربوط به جلسه را در یک پایگاهداده یا ذخیره‌ساز جلسه ذخیره می‌کنید و یک شناسه جلسه (Session ID) را به کاربر تحویل می‌دهید. این روش را مانند مسافری تصور کنید که فقط شناسه بلیت پرواز خود را دریافت می‌کند در حالی که تمام جزئیات دیگر در پایگاهداده ایرلайн ذخیره شده است.

نحوه عملکرد آن به شرح زیر است:

۱. کاربر درخواست ورود به سیستم را ارسال می‌کند و برنامه frontend درخواست را به سرور backend ارسال می‌کند.
۲. کاربر با استفاده از یک کلید مخفی یک Session ایجاد می‌کند و داده‌ها را در ذخیره‌ساز جلسه ذخیره می‌کند.
۳. سرور یک کوکی با Session Id منحصر به فرد برای کاربر به کلاینت ارسال می‌کند.
۴. کاربر درخواست جدیدی ایجاد می‌کند و مرورگر شناسه Session را همراه با درخواست ارسال می‌کند.
۵. سرور با استفاده از شناسه Session، کاربر را تأیید اعتبار می‌کند.

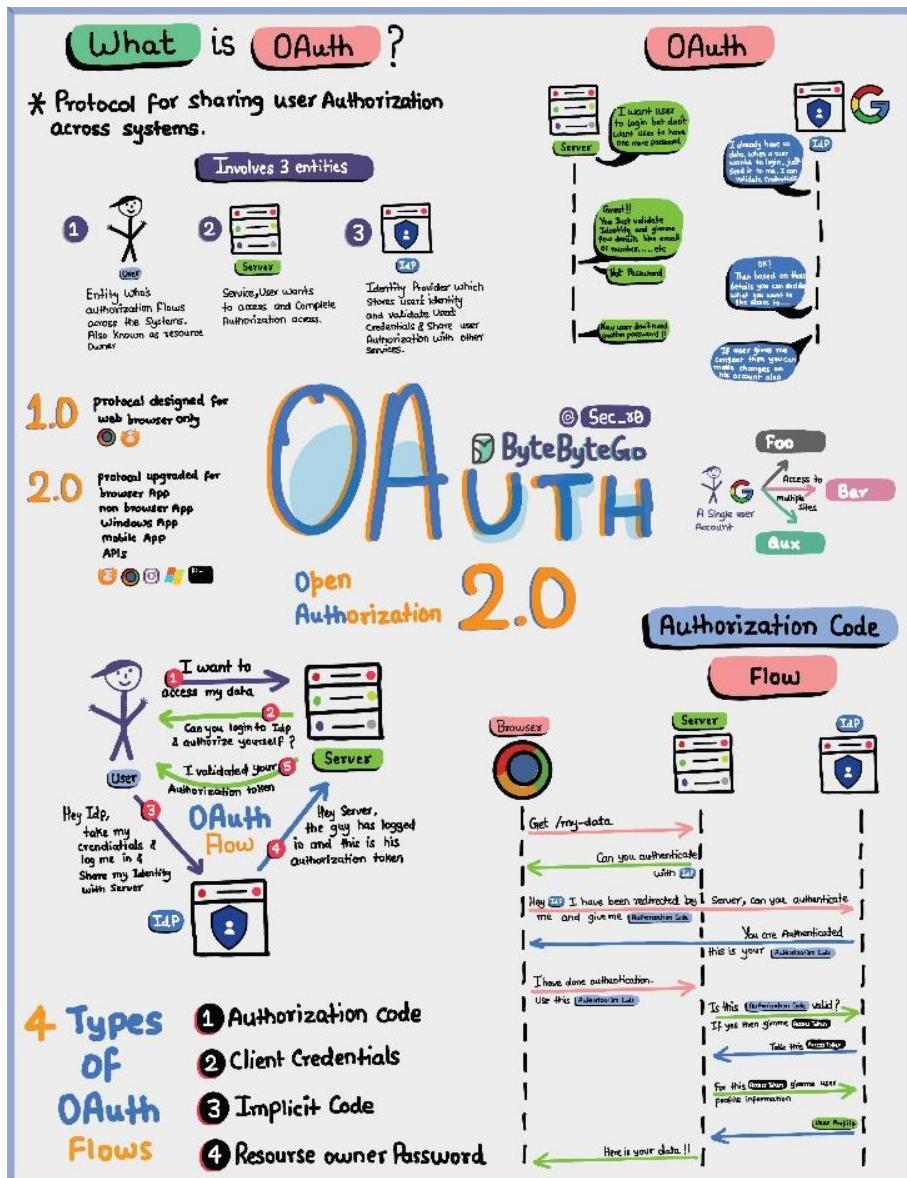
### تأیید اعتبار مبتنی بر JWT

در روش مبتنی بر JWT، اطلاعات مربوط به Session را در ذخیره‌ساز Session ذخیره نمی‌کنید. در واقع کل اطلاعات در داخل Token موجود است. این را مانند دریافت بلیط هوایپما به همراه تمام جزئیات موجود روی بلیط اما به صورت کدگذاری شده است را تصور کنید. نحوه عملکرد آن به شرح زیر است:

۱. کاربر درخواست ورود به سیستم ارسال می‌کند و درخواست به سرور backend می‌رود.
۲. سرور اعتبارنامه‌ها را تأیید می‌کند و یک JWT صادر می‌کند. JWT با استفاده از یک کلید خصوصی امضا می‌شود و هیچ ذخیره‌ساز Session درگیر نیست.
۳. JWT به کلاینت منتقل می‌شود، چه به صورت کوکی و چه در بدنه به عنوان پاسخ. هر دو رویکرد مزایا و معایب خود را دارند، اما ما روش کوکی را انتخاب کرده‌ایم.
۴. برای هر درخواست بعدی، مرورگر کوکی حاوی JWT را ارسال می‌کند.
۵. سرور با استفاده از کلید خصوصی مخفی، JWT را تأیید می‌کند و اطلاعات کاربر را استخراج می‌کند.

## احراز هویت و دسترسی امن با OAuth 2.0

OAuth 2.0 یک چارچوب قدرتمند و امن است که به برنامه‌های مختلف اجازه می‌دهد تا بدون به اشتراک گذاشتن اعتبار حساس، به طور ایمن از طرف کاربران با یکدیگر تعامل داشته باشد.



اجزای درگیر در OAuth عبارت‌اند از کاربر، سرور و ارائه‌دهنده هویت (IDP<sup>۱</sup>).

### توکن OAuth چه کاری می‌تواند انجام دهد؟

هنگامی که از OAuth استفاده می‌کنید، یک توکن OAuth دریافت می‌کنید که نشان‌دهنده هویت<sup>۲</sup> و مجوزهای<sup>۳</sup> شما است. این توکن می‌تواند چند کار مهم را انجام دهد:

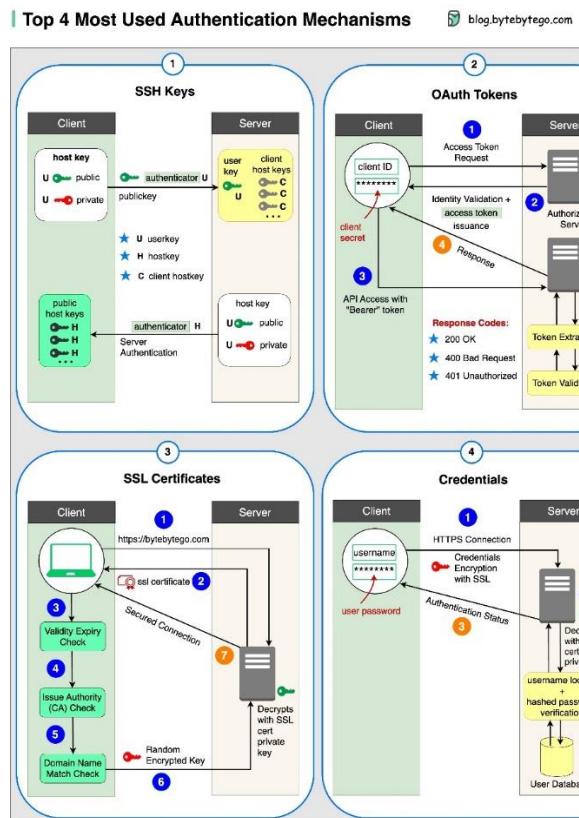
- **ورود یکپارچه (SSO):** با استفاده از یک توکن OAuth، می‌توانید فقط با یک ورود به سیستم، وارد چندین سرویس یا برنامه شوید و فرایند را آسان‌تر و امن‌تر کنید.
- **مجوزدهی در سراسر سیستم‌ها:** توکن OAuth به شما امکان می‌دهد مجوز یا حقوق دسترسی خود را در سیستم‌های مختلف به اشتراک بگذارید، بنابراین مجبور نیستید در همه‌جا جداگانه وارد شوید.
- **دسترسی به پروفایل کاربر:** برنامه‌هایی که دارای توکن OAuth هستند می‌توانند به بخش‌های خاصی از پروفایل کاربری شما که مجاز می‌دانید دسترسی داشته باشند، اما همه چیز را نمی‌بینند.

<sup>۱</sup> Identity Provider

<sup>۲</sup> identity

<sup>۳</sup> permissions

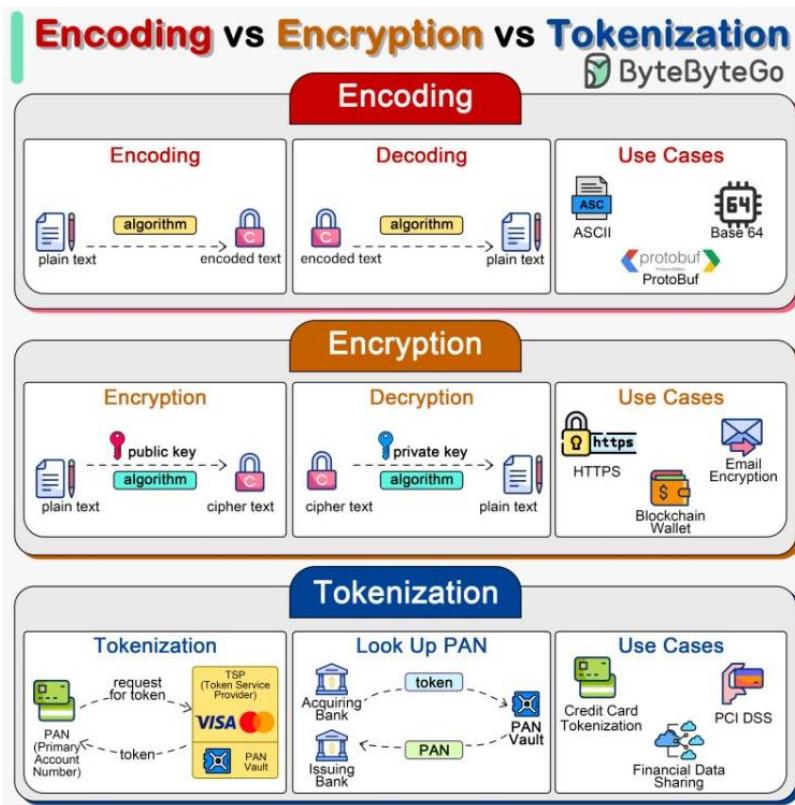
## ۴ روش برای مکانیزم‌های احراز هویت



- کلیدهای SSH: از کلیدهای رمزگاری برای دسترسی امن به سیستم‌های remote و سرورها استفاده می‌شود.
- توکن‌های OAuth: توکن‌هایی که دسترسی محدودی به داده‌های کاربر را در برنامه‌های شخص ثالث ارائه می‌دهند.
- گواهینامه‌های SSL: گواهینامه‌های دیجیتال ارتباط امن و رمزگذاری شده بین سرورها و کلاینت‌ها را تضمین می‌کنند.
- اعتبارنامه‌ها<sup>۱</sup>: اطلاعات احراز هویت کاربر برای تأیید و اعطای دسترسی به سیستم‌ها و سرویس‌های مختلف استفاده می‌شود.

## رمزگذاری در مقابل رمزگذاری در مقابل توکن سازی

رمزگذاری، رمزگذاری و توکن سازی سه فرایند مجزا هستند که داده‌ها را به روش‌های مختلف برای اهداف گوناگون از جمله انتقال داده، امنیت و انطباق با قوانین، مدیریت می‌کنند. در طراحی سیستم، انتخاب رویکرد مناسب برای رسیدگی به اطلاعات حساس بسیار مهم است.



### رمزگذاری (Encoding)

رمزگذاری، داده‌ها را با استفاده از یک طرح برگشت‌پذیر به فرمت دیگری تبدیل می‌کند. به عنوان مثال، رمزگذاری Base64 داده‌های باینری را به کاراکترهای ASCII تبدیل می‌کند و به سیستم‌های مبتنی بر متن اجازه پردازش می‌دهد.

هدف از رمزگذاری، تأمین امنیت داده‌ها نیست. داده‌های رمزگذاری شده را می‌توان به‌راحتی و بدون نیاز به کلید با استفاده از همان طرح، رمزگشایی کرد.

### رمزنگاری (Encryption)

رمزنگاری، داده‌ها را با استفاده از الگوریتم‌ها و کلیدها به یک فرمت امن تبدیل می‌کند. رمزنگاری متقارن از یک کلید برای هر دو فرایند رمزگذاری و رمزگشایی استفاده می‌کند، در حالی که رمزنگاری نامتقارن از یک کلید عمومی برای رمزگذاری و یک کلید خصوصی برای رمزگشایی استفاده می‌کند.

رمزنگاری با تبدیل داده‌های قابل خواندن (متن ساده) به فرمت غیر قابل خواندن (متن رمزگذاری شده) از محروم‌گی داده‌ها محافظت می‌کند. فقط کسانی که کلید صحیح را دارند می‌توانند داده‌های اصلی را رمزگشایی کرده و به آن‌ها دسترسی پیدا کنند.

### نشانه‌گذاری (توکن‌سازی – Tokenization)

نشانه‌گذاری، داده‌های حساس را با نگهدارنده‌های غیرحساس به نام نشانه (توکن) جایگزین می‌کند. نگاشت بین داده‌های اصلی و token به طور امن در یک مخزن توکن (token vault) ذخیره می‌شود. توکن‌ها را می‌توان در سیستم‌های مختلف بدون فاش کردن داده‌های اصلی استفاده کرد و بدین ترتیب خطر لورفتن داده‌ها را کاهش داد.

Tokenization اغلب برای محافظت از اطلاعات کارت اعتباری، شماره‌های شناسایی شخصی و سایر داده‌های حساس استفاده می‌شود. این روش ایمن است؛ زیرا توکن‌ها هیچ بخشی از داده‌های اصلی را در خود جای نداده و قابل مهندسی معکوس نیستند. Tokenization به ویژه برای انطباق با مقرراتی مانند PCI DSS<sup>۱</sup> مفید است.

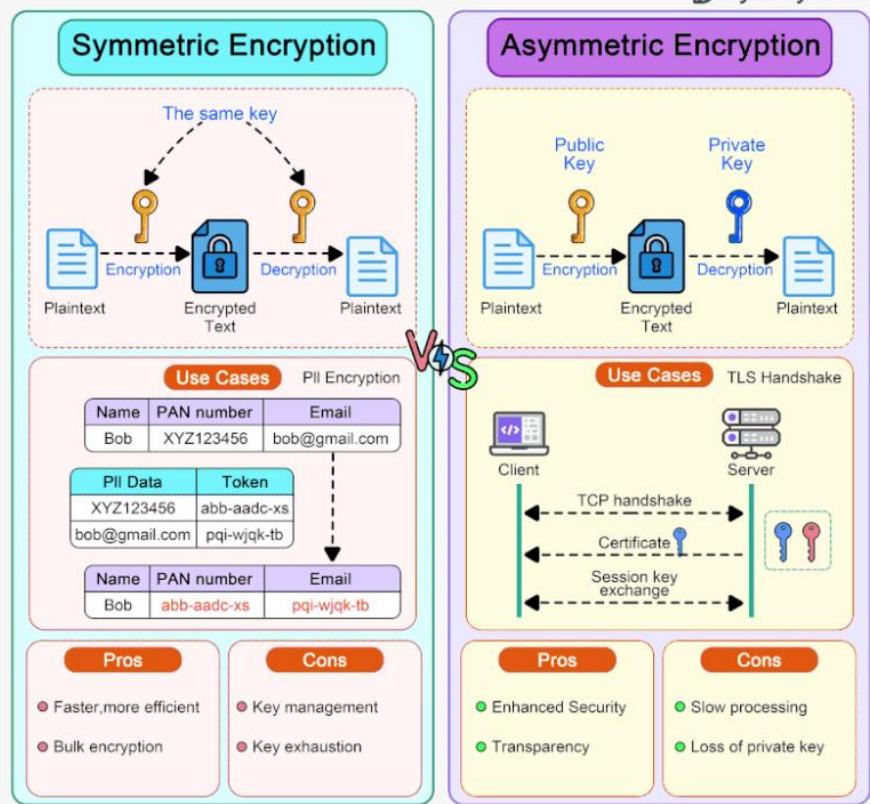
<sup>۱</sup> استاندارد امنیت داده صنعت پرداخت کارت (Payment Card Industry Data Security Standard) یک استاندارد امنیت اطلاعات است برای سازمانهایی که تراکنش‌های کارت‌های اعتباری پردازش می‌کنند.

## رمزنگاری متقارن در مقابل رمزنگاری نامتقارن

رمزنگاری متقارن و رمزنگاری نامتقارن دو نوع تکنیک رمزنگاری هستند که برای ایمن سازی داده ها و ارتباطات استفاده می شوند، اما در روش های رمزنگاری و رمزگشایی با هم تفاوت دارند.

### Symmetric vs Asymmetric Encryption

ByteByteGo



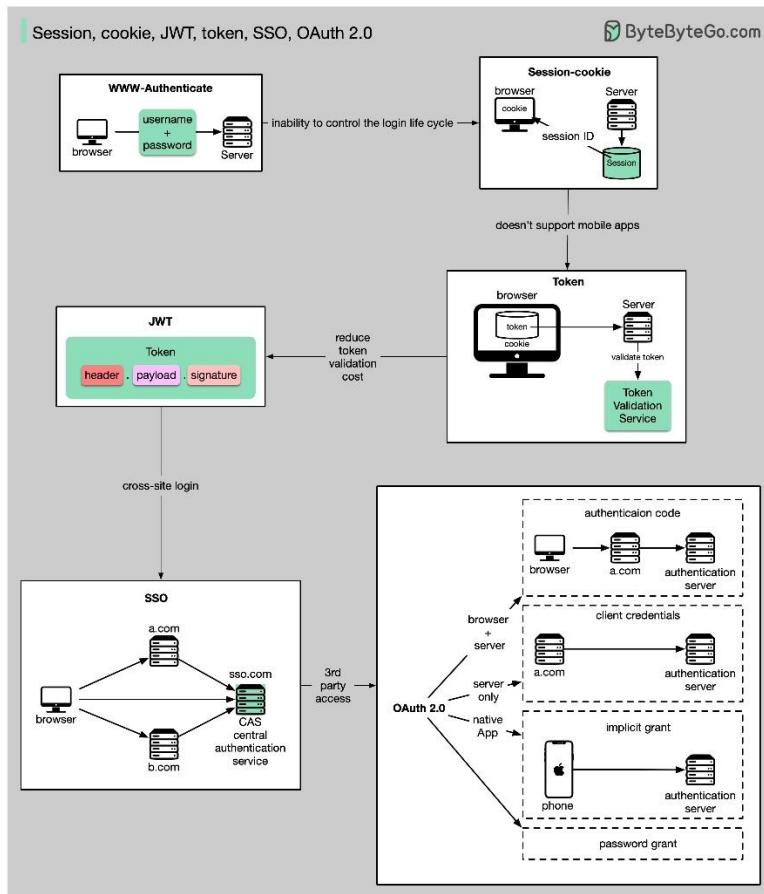
در رمزنگاری متقارن، از یک کلید واحد برای رمزنگاری و رمزگشایی داده ها استفاده می شود. این روش سریع تر است و می تواند برای رمزنگاری/رمزگشایی داده های حجمی استفاده شود. به عنوان مثال، می توانیم از آن برای رمزنگاری مقادیر زیادی از داده های PII (اطلاعات شخصی

قابل شناسایی<sup>۱</sup>) استفاده کنیم. این روش در مدیریت کلید چالش‌هایی ایجاد می‌کند؛ زیرا فرستنده و گیرنده کلید یکسانی را به اشتراک می‌گذارند.

رمزنگاری نامتقارن از یک جفت کلید استفاده می‌کند: یک کلید عمومی و یک کلید خصوصی. کلید عمومی آزادانه توزیع می‌شود و برای رمزگشایی داده‌ها استفاده می‌شود، در حالی که کلید خصوصی مخفی نگه داشته می‌شود و برای رمزگشایی داده‌ها استفاده می‌شود. این روش امن‌تر از رمزنگاری متقارن است؛ زیرا کلید خصوصی هرگز به اشتراک گذاشته نمی‌شود. با این حال، رمزنگاری نامتقارن به دلیل پیچیدگی تولید کلید و محاسبات ریاضی کندر است. به عنوان مثال، HTTPS از رمزنگاری نامتقارن برای تبادل کلیدهای session در طول دست‌دادن<sup>۲</sup> استفاده می‌کند، و پس از آن، HTTPS از رمزنگاری متقارن برای ارتباطات بعدی استفاده می‌کند.

## OAuth 2.0 و SSO، توکن، JWT و Session کوکی

این اصطلاحات همگی به مدیریت هویت کاربر<sup>۱</sup> مرتبط هستند. هنگامی که وارد یک وبسایت می‌شوید، اعلام می‌کنید که چه کسی هستید (شناسایی<sup>۲</sup>). هویت شما تأیید می‌شود (احراز هویت<sup>۳</sup>) و مجوزهای لازم به شما اعطا می‌شود (مجوز<sup>۴</sup>). راه حل‌های زیادی درگذشته پیشنهاد شده است و این لیست همچنان درحال توسعه است.



user identity management<sup>۱</sup>  
identification<sup>۲</sup>  
authentication<sup>۳</sup>  
authorization<sup>۴</sup>

در اینجا درک من از مدیریت هویت کاربر، از ساده به پیچیده، آورده شده است:

- **WWW-Authenticate**: ابتدایی‌ترین روش است. مرورگر از شما نام کاربری و رمز عبور را درخواست می‌کند. در نتیجه عدم توانایی در کنترل چرخه عمر نحوه ورود کاربر، امروزه به‌ندرت استفاده می‌شود.
- **session-cookie**: کنترل دقیق‌تری بر چرخه عمر ورود دارد. سرور، ذخیره‌سازی session را حفظ می‌کند و مرورگر شناسه session را نگه می‌دارد. یک کوکی معمولاً فقط با مرورگرها کار می‌کند و برای برنامه‌های موبایل مناسب نیست.
- **توکن**: برای رفع مشکل سازگاری قابل استفاده است. سرویس‌گیرنده توکن را به سرور ارسال می‌کند و سرور توکن را اعتبارسنجی می‌کند. نکته منفی این است که توکن نیاز به رمزگذاری و رمزگشایی دارد که ممکن است زمان بر باشد.
- **JWT<sup>1</sup>**: یک روش استاندارد برای نمایش توکن‌ها است. این‌ها اطلاعات قابل تأیید و اعتمادی هستند که به صورت دیجیتالی امضا شده‌اند. از آنجایی که JWT حاوی امضا<sup>2</sup> است، نیازی به ذخیره اطلاعات session در سمت سرور نیست.
- **SSO<sup>3</sup>**: با استفاده از SSO، می‌توانید فقط یکبار وارد/login شوید و سپس به چندین وب‌سایت وارد/login شوید. از CAS<sup>4</sup> (سرویس احراز هویت مرکزی) برای نگهداری اطلاعات بین سایتی استفاده می‌کند.
- **OAuth 2.0**: با استفاده از OAuth 2.0، می‌توانید به یک وب‌سایت اجازه دهید تا به اطلاعات شما در یک وب‌سایت دیگر دسترسی داشته باشد.

JSON Web Token<sup>1</sup>  
signature<sup>2</sup>

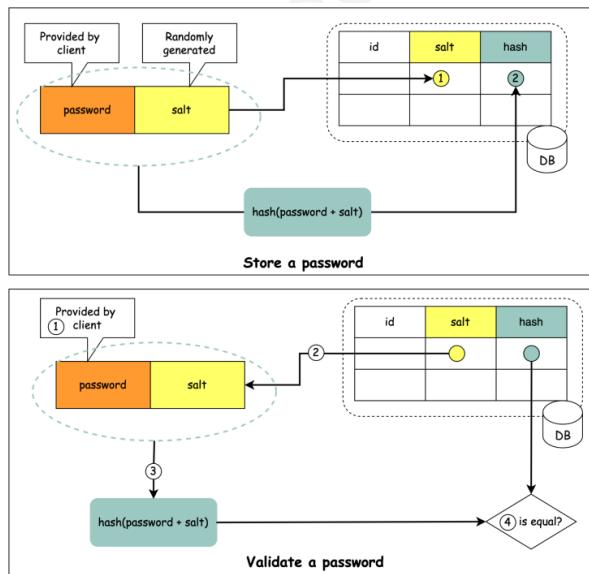
Single Sign-On<sup>3</sup>  
central authentication service<sup>4</sup>

## ذخیره‌سازی رمزها

چگونه رمزهای عبور را به صورت ایمن در پایگاه‌داده ذخیره کنیم؟  
باید نگاهی بیندازیم.

چیزهایی که باید انجام داد

- ذخیره رمزهای عبور به صورت متن ساده ایده خوبی نیست؛ زیرا هر کسی که به داخل دسترسی داشته باشد می‌تواند آن‌ها را ببیند.
  - ذخیره‌سازی هش رمزها به صورت مستقیم کافی نیست؛ زیرا در معرض حملات پیش محاسبه، مانند جدول‌های رنگین‌کمان<sup>۱</sup>، قرار دارد.
  - برای مقابله با حملات پیش محاسبه، ما رمزها را salt می‌کنیم.
- salt چیست؟ بر اساس دستورالعمل‌های OWASP، «یک رشته منحصر به فرد و تصادفی است که در فرایند هش کردن به هر رمز افزوده می‌شود».



### چگونه یک رمز و salt را ذخیره کنیم؟

۱. salt قرار نیست مخفی باشد و می‌توان آن را به صورت متن ساده در پایگاهداده ذخیره کرد. از آن برای اطمینان از منحصر به فرد بودن نتیجه هش هر رمز استفاده می‌شود.
۲. رمز عبور می‌تواند در پایگاهداده با فرمتی مانند مقدار زیر ذخیره شود:  
$$\text{hash}(\text{password} + \text{salt})$$

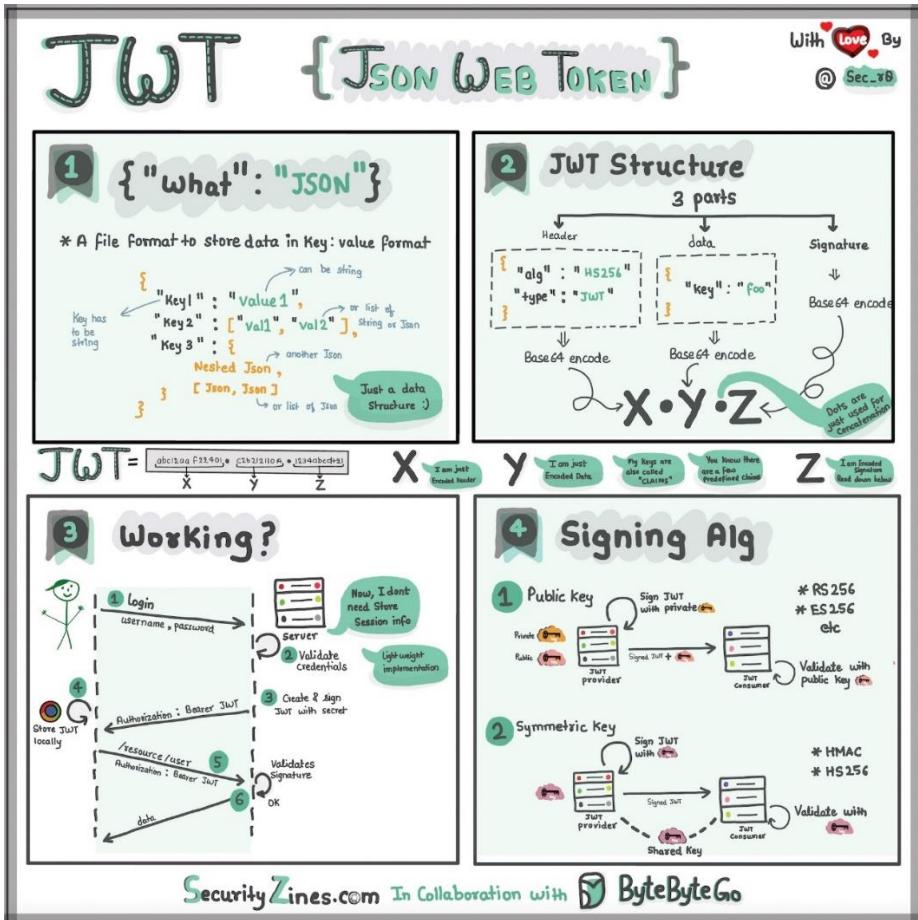
### چگونه یک رمز عبور را تأیید کنیم؟

- برای تأیید یک رمز عبور، می‌توان از فرایند زیر استفاده کرد:
۱. کاربر رمز عبور را وارد می‌کند.
  ۲. سیستم salt متناظر را از پایگاهداده بازیابی می‌کند.
  ۳. سیستم salt را به رمز عبور اضافه کرده و آن را هش می‌کند. باید ارزش هش شده را H1 بنامیم.
  ۴. سیستم H1 و H2 را، جایی که H2 هش ذخیره شده در پایگاهداده است، مقایسه می‌کند. اگر آن‌ها یکسان باشند، رمز عبور معتبر است.

## توکن امن وب جیسون (JWT) به زبان ساده

فرض کنید یک جعبه مخصوص به اسم JWT دارد. داخل این جعبه سه بخش وجود دارد:

- .signature و payload .header



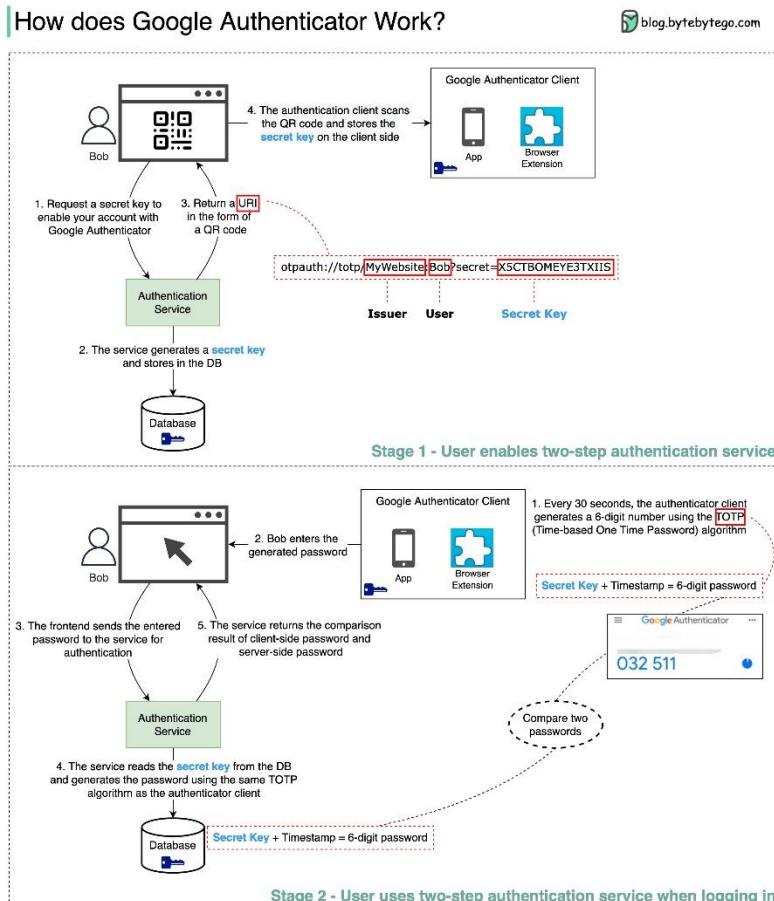
مانند label خارج از محدوده است و بیان می کند که این محدوده از چه نوعی است و چطور امن شده. معمولاً با فرمتی به اسم JSON نوشته می شود و کار با آسان و به سادگی استفاده از آکولاد {} و دونقطه : هست.

payload، مثل پیام یا اطلاعات واقعی که می‌خواهید بفرستید، عمل می‌کند. می‌تواند اسم، سن، یا هر اطلاعات دیگری که می‌خواهید به اشتراک بگذارید را شامل شود. این مورد نیز با فرمت JSON نوشته شده تا به سادگی قابل فهم و استفاده باشد. حالا امضا<sup>۱</sup> چیزی است که JWT را امن‌تر می‌کند و مانند یک مهر مخصوص است که فقط فرستنده‌ی آن می‌داند که چطور می‌تواند آن را بسازد. امضا با یک کد مخفی، شبیه رمز عبور، ساخته می‌شود. این امضا تضمین می‌کند که هیچ‌کس نمی‌تواند بدون اینکه فرستنده متوجه شود، محتویات JWT را دست‌کاری کند.

وقتی می‌خواهید JWT را به یک سرور ارسال کنید، header، payload و signature را داخل بسته می‌گذارید و سپس آن را به سمت سرور ارسال می‌کنید. سرور می‌تواند به راحتی header و payload را بخواند تا متوجه شود که شما چه کسی هستید و می‌خواهید چه کاری را انجام دهید.

## تأیید دو مرحله‌ای Google Authenticator کار می‌کند؟

احراز هویت دو مرحله‌ای گوگل (Google Authenticator) یک نرم افزاری است که از یک سرویس تأیید دو مرحله‌ای استفاده می‌کند. شکل زیر جزئیات آن را نشان می‌دهد.



دو مرحله اساسی وجود دارد:

- مرحله ۱ - کاربر تأیید دو مرحله‌ای گوگل را فعال می‌کند.
- مرحله ۲ - کاربر برای ورود به حساب و کارهای دیگر از احراز هویت<sup>۱</sup> استفاده می‌کند.

بیایید به این مراحل نگاه کنیم.

## مرحله ۱

مراحل ۱ و ۲: آقای باب صفحه وب را برای فعال کردن تأیید دو مرحله‌ای باز می‌کند. بخش کاربری یک کلید مخفی درخواست می‌کند. سرویس احراز هویت، secret key را برای باب تولید می‌کند و سپس در پایگاهداده ذخیره می‌کند.

مرحله ۳: سرویس احراز هویت (authenticator) یک URI به بخش کاربری برمی‌گرداند. این URI از یک صادرکننده کلید، نام کاربری، و secret key تشکیل شده است. این URI به شکل یک QR code روی صفحه وب نمایش داده می‌شود.

مرحله ۴: حالا باب از Google Authenticator برای اسکن کردن کد QR تولید شده استفاده می‌کند. authenticator در QR code ذخیره می‌شود.

## مرحله ۲

مراحل ۱ و ۲: باب می‌خواهد با تأیید دو مرحله‌ای گوگل وارد یک سایت بشود. برای این کار به رمز عبور نیاز دارد. هر ۳۰ ثانیه، Google Authenticator یک رمز عبور ۶ رقمی با استفاده از الگوریتم TOTP (رمز عبور یکبار مصرف مبتنی بر زمان) تولید می‌کند. باب از این رمز عبور برای ورود به این سایت استفاده می‌کند.

مراحل ۳ و ۴: بخش frontend رمز عبوری را که باب وارد کرده است را برای احراز هویت به بخش سرور می‌فرستد. سرویس authenticator که secret key را از پایگاهداده می‌خواند و یک رمز عبور ۶ رقمی با استفاده از همان الگوریتم TOTP که کلاینت استفاده کرده است را تولید می‌کند.

مرحله ۵: سرویس احراز سنجش دو رمز عبور تولید شده توسط کلاینت و سرور را مقایسه می‌کند و نتیجه مقایسه را به بخش frontend برمی‌گرداند. باب فقط در صورتی می‌تواند فرایند ورود را ادامه بدهد که این دو رمز عبور با هم مطابقت داشته باشند.

آیا این مکانیسم احراز هویت امن است؟

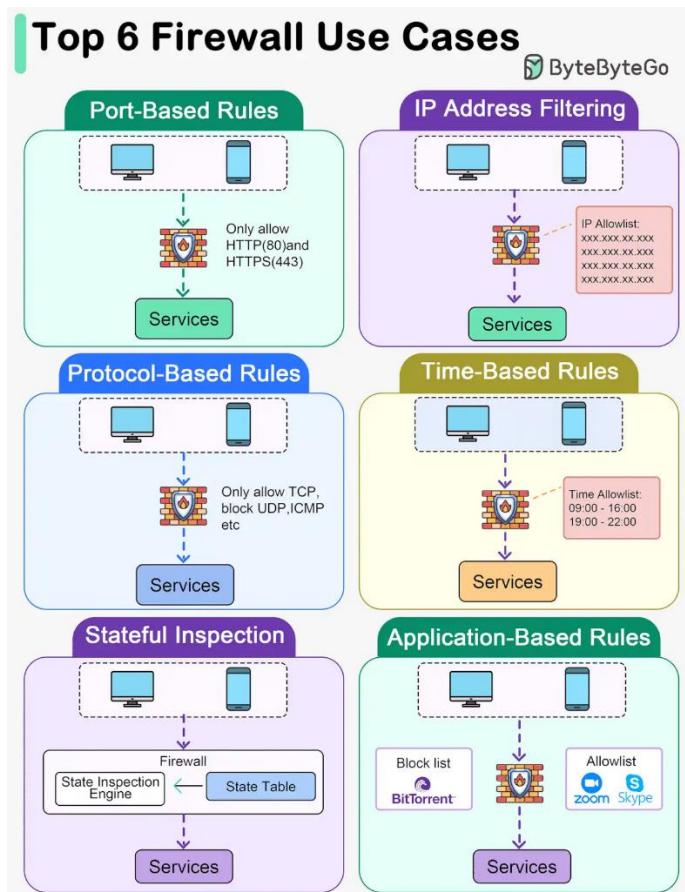
آیا کلید secret key افراد دیگر قابل دستیابی است؟

ما باید مطمئن شویم که secret key با استفاده از HTTPS منتقل می‌شود. کلاینت و پایگاهداده secret key را ذخیره می‌کنند و ما باید مطمئن شویم که کلیدهای secret رمزگذاری شده‌اند.

آیا رمز عبور ۶ رقمی توسط هکرها قابل حدس زدن است؟

خیر. رمز دارای ۶ رقم است، بنابراین رمز عبور ایجاد شده دارای ۱ میلیون ترکیب بالقوه است. به علاوه، رمز عبور هر ۳۰ ثانیه تغییر می‌کند. اگر هکرها بخواهند رمز عبور را در ۳۰ ثانیه حدس بزنند، باید ۳۰۰۰۰ ترکیب مختلف را در ثانیه وارد کنند.

## ۶ مورد از مهم‌ترین کاربردهای فایروال



### قواعد مبتنی بر پورت

قوانين فایروال<sup>۱</sup> می‌توانند بر اساس پورت‌های خاص برای مجاز کردن یا مسدودکردن ترافیک تنظیم بشوند. برای مثال، فقط به ترافیک روی پورت‌های ۸۰ (HTTP) و ۴۴۳ (HTTPS) برای مرورگر وب را اجازه بدهید.

Firewall rules<sup>۱</sup>

### فیلترینگ بر اساس آدرس IP

قوانين می‌توانند برای مجاز کردن یا ردکردن ترافیک بر اساس آدرس IP مبدأ یا مقصد پیکربندی شوند. این می‌تواند شامل لیست مجازی از آدرس‌های IP قابل اعتماد یا لیست غیرمجاز آدرس‌های IP مخرب شناخته شود.

### قوانين مبتنی بر پروتکل

فایروال‌ها را می‌توان برای اجازه‌دادن یا مسدودکردن ترافیک بر اساس پروتکل‌های خاص شبکه مانند TCP، UDP، ICMP و غیره پیکربندی کرد. به عنوان مثال، اجازه می‌دهد تنها ترافیک TCP در پورت ۲۲ (SSH) مجاز باشد.

### قوانين مبتنی بر زمان

فایروال‌ها را می‌توان برای اجرای قوانین بر اساس زمان یا برنامه‌های خاص پیکربندی کرد. این مورد می‌تواند برای تنظیم قوانین دسترسی مختلف در طول ساعات کاری در مقابل بعد از ساعت کاری مفید باشد.

### بازرسی Stateful

بازرسی Stateful: فایروال‌های Stateful وضعیت اتصالات فعلی را نظارت می‌کنند و ترافیک را تنها در صورتی که با یک اتصال ثابت مطابقت داشته باشد، مانع از دسترسی غیرمجاز از خارج می‌شود.

### قوانين مبتنی بر اپلیکیشن

برخی فایروال‌ها کنترل سطح برنامه را با اجازه‌دادن یا مسدودکردن ترافیک بر اساس برنامه‌ها یا سرویس‌های خاص ارائه می‌دهند. به عنوان مثال، اجازه یا محدودکردن دسترسی به برنامه‌های خاص مانند اسکایپ، BitTorrent و غیره.

## هر آنچه در مورد Cross-Site Scripting (XSS) نیاز دارد بدانید

حمله XSS یک آسیب‌پذیری رایج است که زمانی رخ می‌دهد که اسکریپت‌های مخرب، اغلب از طریق فیلدهای ورودی، به صفحات وب تزریق شوند. برای درک عمیق‌تر نحوه‌ی پدیدارشدن این آسیب‌پذیری با مدیریت نادرست ورودی کاربر و سپس بازگشت آن به کاربر، به نمودار زیر مراجعه کنید که سیستم‌ها را در برابر سوءاستفاده آسیب‌پذیر می‌کند.



درک تمایز بین XSS بازتابی (Reflective) و ذخیره‌شده (Stored) بسیار مهم است. XSS بازتابی شامل اجرای فوری اسکریپت تزریق شده می‌شود، درحالی که XSS ذخیره‌شده به مرور زمان باقی می‌ماند و تهدیدات بلندمدتی را ایجاد می‌کند. برای مقایسه‌ی جامع این بردارهای حمله، به نمودارها مراجعه کنید.

این سناریو را تصور کنید: یک هکر حیله‌گر از XSS برای برداشت مخفیانه‌ی اعتبارنامه‌های کاربر، مانند کوکی‌ها، از مرورگر آن‌ها سوءاستفاده می‌کند که به طور بالقوه منجر به دسترسی غیرمجاز و نقض داده‌ها می‌شود. این یک واقعیت وحشتناک است.

در تصویر زیر به استراتژی‌های مؤثر برای کاهش آسیب‌پذیری می‌پردازد و شما را قادر می‌سازد تا سیستم‌های خود را در برابر حملات XSS تقویت کنید. در این از اعتبارسنجی ورودی و کدگذاری خروجی گرفته تا اجرای سیاست‌های سخت‌گیرانه‌ی امنیت محتوا (CSP) شما را تحت پوشش قرار داده شده است.

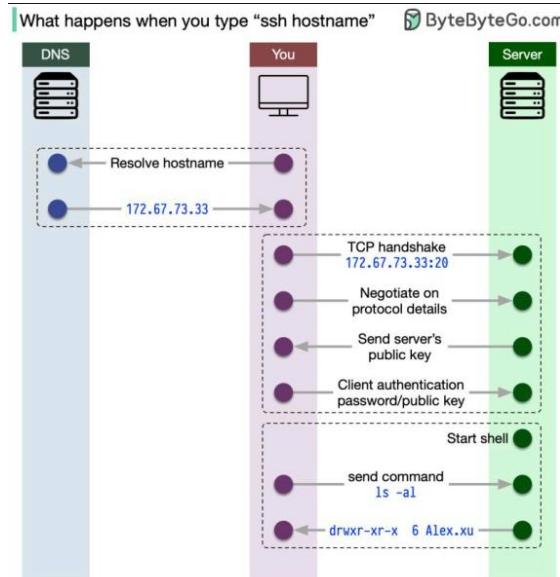
### چه اتفاقی می‌افتد وقتی ssh hostname را تایپ می‌کنید؟

در دهه ۱۹۹۰، Secure Shell برای ارائه یک جایگزین امن برای Telnet برای دسترسی و مدیریت سیستم‌های remote توسعه یافت. استفاده از SSH یک روش عالی برای ایجاد ارتباط امن بین کلاینت و سرور است؛ زیرا از یک پروتکل امن استفاده می‌کند.

هنگامی که "ssh hostname" را تایپ می‌کنید، موارد زیر اتفاق می‌افتد:

- Hostname resolution: تبدیل نام میزبان به آدرس IP با استفاده از DNS یا فایل local میزبان در حالت
- آغازگر SSH کلایнт: اتصال به سرور SSH به صورت remote

- TCP handshake: ایجاد یک اتصال قابل اعتماد.
- Protocol negotiation: توافق بر روی نسخه پروتکل SSH و الگوریتم‌های رمزنگاری.
- تبادل کلید: تولید یک کلید مخفی مشترک به صورت امن.
- احراز هویت<sup>۱</sup> سرور: تأیید کلید عمومی سرور.
- احراز هویت کاربر: احراز هویت با استفاده از رمز عبور، کلید عمومی یا روش دیگر.
- ایجاد Session: ایجاد یک Session SSH رمزنگاری شده و دسترسی به سیستم از راه دور.

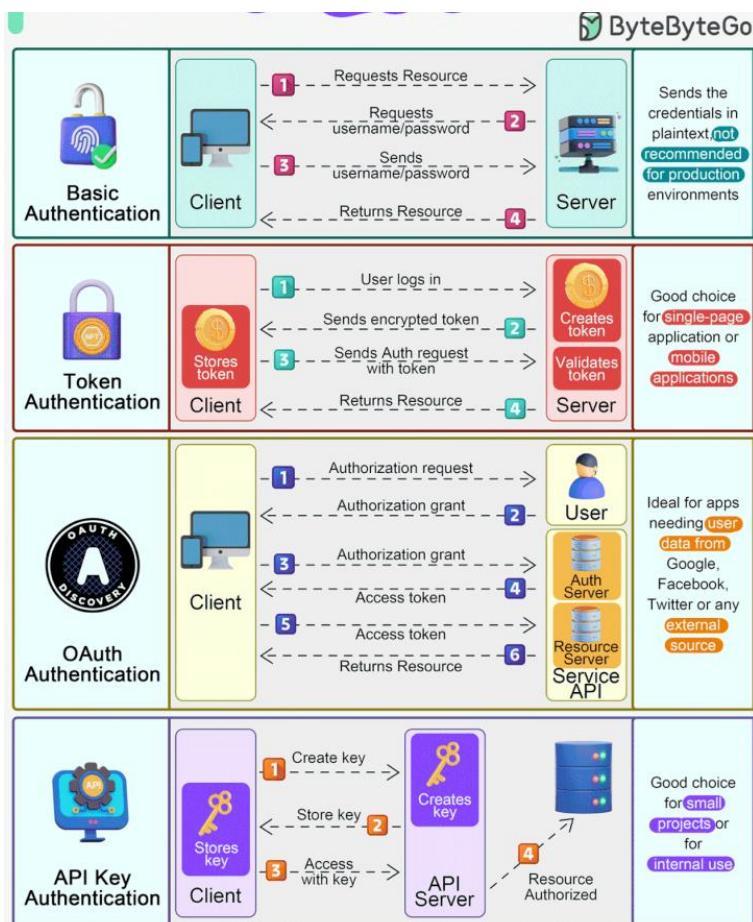


حتماً از احراز هویت مبتنی بر کلید با SSH برای امنیت بهتر استفاده کنید و فایل‌های پیکربندی و گزینه‌های SSH را برای سفارشی‌سازی تجربه خود بیاموزید. از بهترین روش‌ها و توصیه‌های امنیتی برای اطمینان از یک تجربه دسترسی remote امن و کارآمد پیروی کنید.

## روش‌های REST API Authentication

احراز هویت در REST API‌های به عنوان یک مورد حیاتی عمل می‌کند و اطمینان حاصل می‌کند که تنها کاربران یا برنامه‌های معجاز به منابع API دسترسی پیدا می‌کنند.

برخی از روش‌های محبوب احراز هویت برای REST API‌های عبارت‌اند از:



### ۱. Basic Authentication

در گیر ارسال نام کاربری و رمز عبور با هر درخواست می‌شود، اما بدون رمزگذاری می‌تواند نامن باشد.

چه زمانی استفاده شود:

برای برنامه‌های ساده مناسب است که امنیت و رمزگذاری اولویت اصلی نیست یا زمانی که از طریق اتصالات امن استفاده می‌شود.

## ۲. **Token Authentication**

از توکن‌های تولید شده مانند JSON Web Tokens (JWT) استفاده می‌کند که بین کلاینت و سرور مبادله می‌شوند و امنیت بیشتری را بدون ارسال اطلاعات ورود به سیستم با هر درخواست ارائه می‌دهند.

### چه زمانی استفاده شود:

برای سیستم‌های امن‌تر و مقیاس‌پذیرتر ایده‌آل است، به خصوص زمانی که جلوگیری از ارسال اطلاعات ورود به سیستم با هر درخواست اولویت دارد.

## ۳. **OAuth Authentication**

با صدور توکن‌های دسترسی پس از احراز هویت کاربر، امکان دسترسی محدود شخص ثالث به منابع کاربر را بدون افشای اطلاعات ورود به سیستم فراهم می‌کند.

### چه زمانی استفاده شود:

برای سناریوهایی که نیاز به کنترل دسترسی به منابع کاربر توسط برنامه‌ها یا سرویس‌های شخص ثالث دارند، ایده‌آل است.

## ۴. **API Key Authentication**

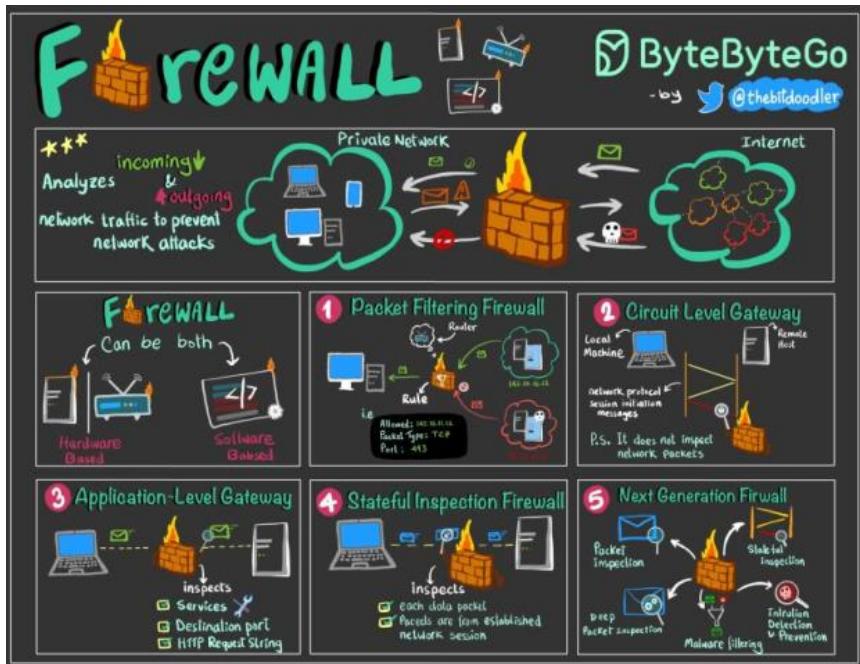
کلیدهای منحصر به‌فردی را به کاربران یا برنامه‌ها اختصاص می‌دهد که در هدرها یا پارامترها ارسال می‌شوند؛ در حالی که ساده است، ممکن است فاقد ویژگی‌های امنیتی روش‌های مبتنی بر توکن یا OAuth باشد.

### چه زمانی استفاده شود:

برای کنترل دسترسی ساده در محیط‌های کمتر حساس یا برای اعطای دسترسی به برخی عملکردها بدون نیاز به مجوزهای خاص کاربر مناسب است.

## فایروال به زبان ساده

فایروال یک سیستم امنیتی شبکه است که ترافیک شبکه را کنترل و فیلتر می‌کند و مانند یک نگهبان بین یک شبکه خصوصی و اینترنت عمومی عمل می‌کند.



آنها در دو دسته اصلی وجود دارند:

نرم افزاری: روی دستگاههای فردی برای محافظت نصب می‌شود

سخت افزاری: دستگاههای مستقل که کل شبکه را محافظت می‌کنند

فایروال‌ها انواع مختلفی دارند که هر کدام برای نیازهای امنیتی خاصی طراحی شده‌اند:

۱. **Packet Filtering Firewalls**: بسته‌های داده را بررسی می‌کنند و بر اساس مبدأ، مقصد

یا پروتکل‌ها آنها را می‌پذیرند یا رد می‌کنند.

۲. **Circuit-level Gateways**: ارتباط TCP بین بسته‌ها را برای تعیین مشروعیت session

ناظارت می‌کنند.

۳. Application-level Gateways (فایروال‌های پروکسی): ترافیک ورودی بین شبکه شما و منبع ترافیک را فیلتر می‌کنند و یک سپر محافظتی در برابر شبکه‌های غیرقابل اعتماد ارائه می‌دهند.

۴. Stateful Inspection Firewalls: اتصالات فعال را برای تعیین اینکه کدام بسته‌ها را اجازه دهنند ردیابی می‌کنند و در زمینه جایگاه آن‌ها در یک جریان داده تجزیه و تحلیل می‌کنند.

۵. Next-Generation Firewalls (NGFWs): فایروال‌های پیشرفته‌ای که روش‌های سنتی را با قابلیت‌هایی مانند سیستم‌های جلوگیری از نفوذ، تجزیه و تحلیل عمیق بسته‌ها و آگاهی از برنامه‌ها ادغام می‌کنند.

## الگوریتم‌های کاربردی در سیستم‌های توزیع شده

### الگوریتم‌هایی مهم در طراحی سیستم

این الگوریتم‌ها نه تنها برای مصاحبه مفید هستند، بلکه برای هر مهندس نرم افزاری درک آن‌ها مفید است.

Algorithm	How it Works	Priority	Use Cases
Geohash		★★★★★	Location based service
Quadtree		★★★★★	Location based service
Consistent Hashing		★★★★★	Balance the load within a cluster of services
Leaky bucket		★★★★★	Rate limiter
Token bucket		★★★★★	Rate limiter
Trie		★★★★★	Search autocomplete
Rsync		★★★☆☆	File transfers
Raft/Paxos		★★★☆☆	Consensus algorithms

یک نکته مهم این است که در مصاحبه طراحی سیستم، درک «چگونگی استفاده از این الگوریتم‌ها در سیستم‌های دنیای واقعی» به طور کلی مهم‌تر از جزئیات پیاده‌سازی آن‌ها است.

Bloomfilter		★★★☆☆	Eliminate costly lookups
Merkle tree		★★★☆☆	Identify inconsistencies between nodes
HyperLogLog		★☆☆☆☆	Count unique values fast
Count-min sketch		★☆☆☆☆	Estimate frequencies of items
Hierarchical timing wheels		★☆☆☆☆	Job scheduler
Operational transformation		★☆☆☆☆	Collaborative editing

ستاره‌ها در نمودار به چه معنا هستند؟

رتبه‌بندی الگوریتم‌ها بر اساس اهمیت به طور عینی بسیار دشوار است.

پنج ستاره: بسیار مهم. سعی کنید نحوه عملکرد و دلیل آن را درک کنید.

سه ستاره: تا حدودی مهم است. ممکن است لازم نباشد جزئیات پیاده‌سازی را بدانید.

یک ستاره: پیش‌رفته. برای کاندیداهای ارشد خوب است.

نکته: الگوریتم‌های مثل **merkle tree – leaky bucket – consistent hashing** –

**Trie – bloom filter – token bucket** در کتاب « طراحی سیستم‌های نرم‌افزاری ۱ »

که در انتهای همین کتاب معرفی شده است به صورت مفصل توضیح داده شده است.

برای الگوریتم [Count-min sketch](#) ، [Rsync](#)، [Raft/Paxos](#) و [Hyperloglog](#) ،

به لینک‌های مربوطه [Operational transformation](#)، [Hierarchical Timing Wheels](#)

مراجعه کنید.

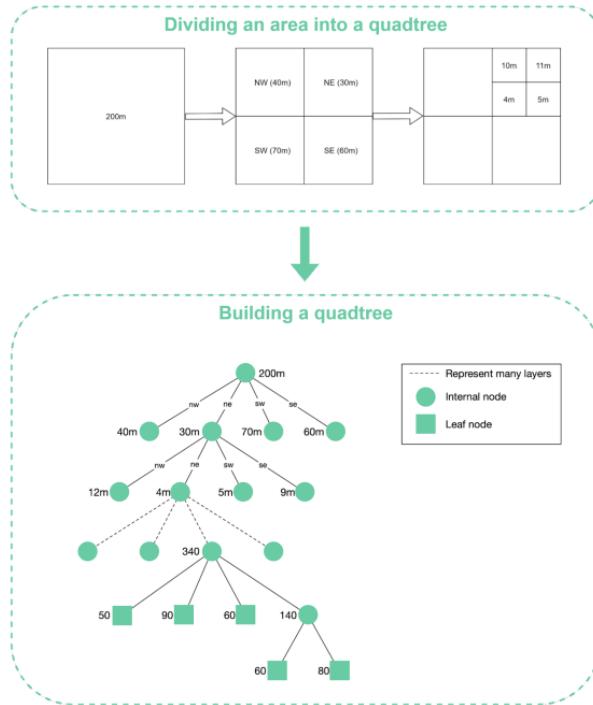
سایر موارد مانند quadtree و geohash در همین کتاب معرفی شده است.

## Quadtree

در این پست، بیایید ساختار داده دیگری را برای یافتن رستوران‌های نزدیک در Yelp یا Google Maps بررسی کنیم.

یک ساختار داده است که معمولاً برای تقسیم یک فضای دوبعدی با تقسیم مجدد آن به چهار ربع (grids) تا زمانی که محتویات gridها معیارهای خاصی را برآورده کنند، استفاده می‌شود (نمودار اول را ببینید).

### | Quadtree



یک ساختار داده درون حافظه<sup>۱</sup> است و یک راه حل پایگاه داده‌ای نیست. در هر سرور LBS<sup>2</sup> (سرویس مبتنی بر مکان) اجرا می‌شود و ساختار داده در زمان راهاندازی سرور

in-memory<sup>۱</sup>

Location-Based Service<sup>2</sup>

ساخته می‌شود. نمودار دوم فرایند ساخت Quadtree را با جزئیات بیشتری توضیح می‌دهد. گره ریشه کل نقشه جهان را نمایش می‌دهد. گره ریشه به طور مجدد به ۴ ربع شکسته می‌شود تا زمانی که گره‌ای با بیش از ۱۰۰ نمونه باقی نماند.

### چگونه با Quadtree مربوط به کسب و کارهای نزدیک را پیدا کنیم؟

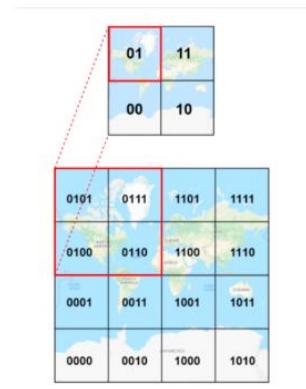
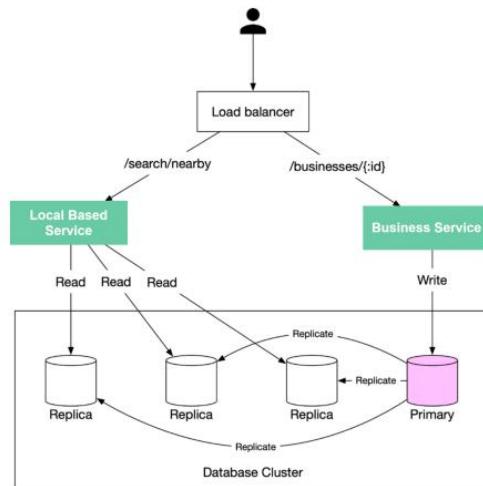
- Quadtree را در حافظه بسازید.
- پس از ساخت Quadtree، جستجو را از ریشه شروع کنید و درخت را طی کنید، تا گره برگی که مبدأ جستجو در آن قرار دارد را پیدا کنید.
- اگر آن گره، برگی از ۱۰۰ کسب و کار دارد، گره را برگردانید. در غیر این صورت، کسب و کارها را از همسایگان آن اضافه کنید تا تعداد کافی کسب و کار برگردانده شود.

### بهروزرسانی سرور LBS و بازسازی Quadtree

- ساخت یک Quadtree در حافظه با ۲۰۰ میلیون کسب و کار در زمان راه اندازی سرور ممکن است چند دقیقه طول بکشد.
- در حالی که Quadtree در حال ساخته شدن است، سرور نمی‌تواند ترافیک را پردازش کند.
- بنابراین، باید یک نسخه جدید از سرور را به صورت تدریجی در یک زیرمجموعه کوچک از سرورها در هر زمان اجرا کنیم. این از آفلاین شدن بخش بزرگی از خوشه سرور<sup>۱</sup> و ایجاد اختلال در سرویس جلوگیری می‌کند.

## چگونه رستوران‌های نزدیک را به کمک Geohash پیدا می‌کنیم؟

در مورد<sup>۱</sup> Yelp در اینجا جزئیات طراحی پشت‌صخنه را مشاهده می‌کنید. دو سرویس کلیدی وجود دارند (مانند شکل زیر):



<sup>۱</sup> Yelp (به فارسی: یلپ) یک شرکت چند ملیتی آمریکایی است که مقر آن در سان فرانسیسکو کالیفرنیا است. این سرویس بر پایه وبسایت Yelp.com و اپلیکیشن موبایل Yelp، دیدگاه‌های دیگران در مورد کسب و کارهای محلی مانند رستوران‌ها، کافه‌ها و... را پردازش و در اختیار کاربران دیگر قرار می‌دهد. همچنین، این شرکت به مدیران کسب و کارها روش برخوردار با دیدگاه‌های آنلاین و روش‌های پاسخ به آنها را آموزش می‌دهد.

- سرویس کسب و کار.
- اضافه کردن / حذف کردن / ویرایش اطلاعات رستوران.
- مشتریان می‌توانند جزئیات رستوران را مشاهده کنند.
- سرویس مبتنی بر محل (LBS<sup>۱</sup>).
- با درنظر گرفتن ناحیه و مکان، لیستی از رستوران‌های نزدیک را برمی‌گرداند.

چگونه اطلاعات مربوط به مکان رستوران‌ها در پایگاهداده ذخیره می‌شوند تا LBS بتواند به طور کارآمد رستوران‌های نزدیک را برگرداند؟

اطلاعات مکانی (عرض و طول جغرافیایی) رستوران‌ها را در پایگاهداده ذخیره کنیم؟ در صورتی که بخواهیم فاصله شما تا هر رستوران را محاسبه کنیم، کوثری‌های پایگاهداده بسیار ناکارآمد خواهند بود.

یک راه برای افزایش سرعت جستجو، استفاده از الگوریتم **geohash** است. ابتدا، کره زمین را به چهار ناحیه تقسیم می‌کنیم با توجه به خط استوا و خط طول اولیه آن داریم:

- بازه عرض جغرافیایی [۹۰°، ۰°] با عدد ۰ نشان داده می‌شود
- بازه عرض جغرافیایی [۰°، ۹۰°] با عدد ۱ نشان داده می‌شود
- بازه طول جغرافیایی [-۱۸۰°، ۰°] با عدد ۰ نشان داده می‌شود
- بازه طول جغرافیایی [۰°، ۱۸۰°] با عدد ۱ نشان داده می‌شود

سپس هر گره را به چهار گره کوچک‌تر تقسیم می‌کنیم. هر گره می‌تواند با جایه‌جایی بین بیت طول جغرافیایی و بیت عرض جغرافیایی نشان داده شود.

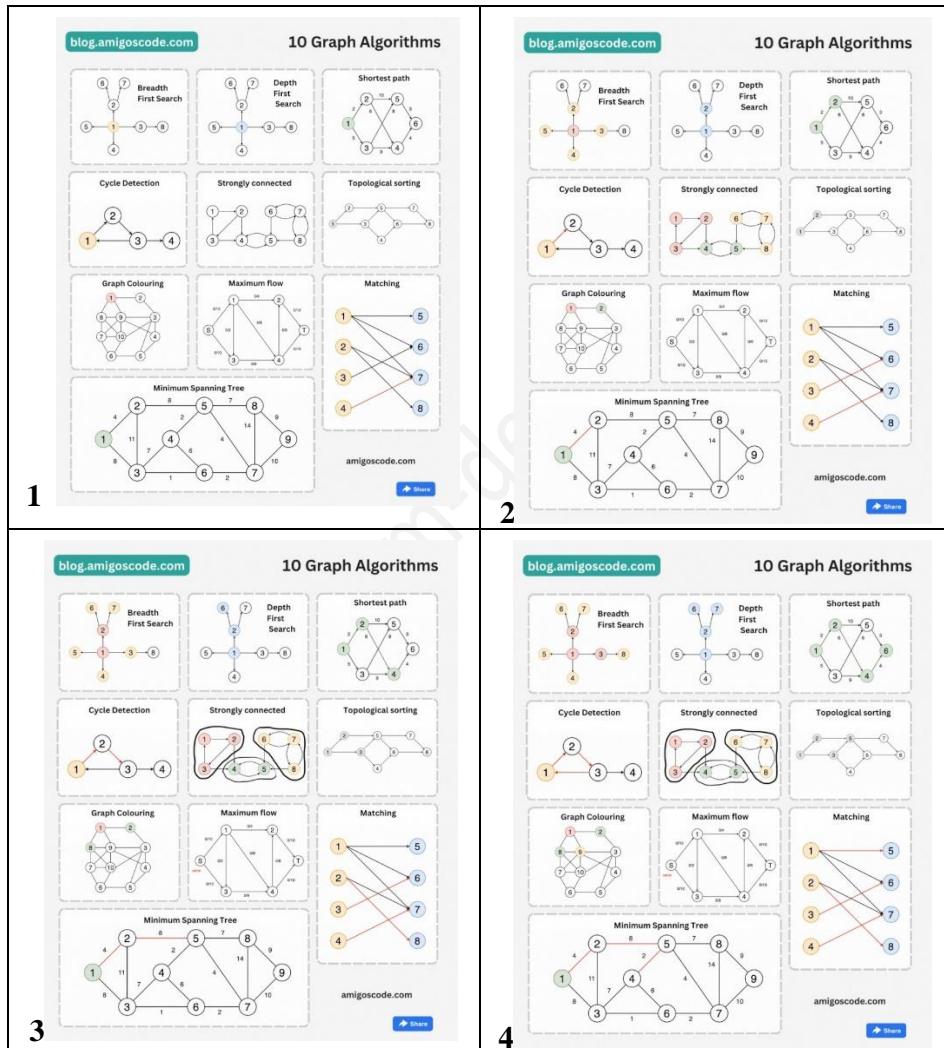
بنابراین، وقتی می‌خواهید رستوران‌های نزدیک در بلوک مشخص شده با رنگ قرمز را جستجو کنید، می‌توانید SQL مشابه زیر را بنویسید:

'SELECT \* FROM geohash\_index WHERE geohash LIKE '01%

جغرافیای کد (Geohash) برخی محدودیت‌ها دارد. ممکن است در یک بلوک (مانند مرکز شهر نیویورک) بسیاری از رستوران‌ها وجود داشته باشد، اما در بلوک دیگر (مانند اقیانوس) هیچ رستورانی وجود نداشته باشد؛ بنابراین، الگوریتم‌های پیچیده‌تر دیگری برای بهینه‌سازی فرایند وجود دارند. اگر علاقه‌مند به جزئیات هستید، می‌توانید در این موارد بیشتر مطالعه کنید.

## ۱۰. الگوریتم گراف که باید بدانید

الگوریتم‌های گراف رویه‌های محاسباتی هستند که برای تحلیل و دستکاری ساختارهای گراف طراحی شده‌اند و کارهایی مانند مسیریابی، تحلیل اتصالات و بهینه‌سازی را در کاربردهای مختلفی مانند مسیریابی شبکه و تحلیل شبکه‌های اجتماعی تسهیل می‌کنند.



<p><b>blog.amigoscode.com</b></p> <p><b>10 Graph Algorithms</b></p> <p>5</p>	<p><b>blog.amigoscode.com</b></p> <p><b>10 Graph Algorithms</b></p> <p>6</p>
<p><b>blog.amigoscode.com</b></p> <p><b>10 Graph Algorithms</b></p> <p>7</p>	<p><b>blog.amigoscode.com</b></p> <p><b>10 Graph Algorithms</b></p> <p>8</p>

۱. جستجوی اول سطح (Breadth-First Search): این الگوریتم یک گراف را سطح به سطح طی می‌کند و تمام همسایگان یک گره را قبل از رفتن به سطح بعدی بررسی می‌کند.

۲. جستجوی عمق اول (**Depth-First Search**): تا جایی که ممکن است در امتداد هر شاخه پیش می‌رود قبل از اینکه backtracking<sup>۱</sup> صورت پذیرد، این الگوریتم اغلب با استفاده از روش‌های بازگشتی (recursion) اجرا می‌شود.
۳. کوتاه‌ترین مسیر (**Shortest Path**): کارآمدترین مسیر بین دو گره را از نظر مجموع وزن لبه‌ها پیدا می‌کند.
۴. شناسایی دوره (**Cycle Detection**): وجود دوره (حلقه) در گراف را شناسایی می‌کند که برای تشخیص وابستگی‌ها و اجتناب از حلقه‌های بینهایت ضروری است.
۵. درخت پوشای کمینه (**Minimum Spanning Tree**): زیرمجموعه‌ای از لبه‌ها را پیدا می‌کند که همه رئوس را با حداقل وزن کل لبه‌ها را متصل می‌کند و یک درخت را تشکیل می‌دهد.
۶. اجزای قویاً همبند (**Strongly Connected Components**): یک گراف جهت‌دار را به زیرگراف‌های جهت‌دار قویاً مرتبط تقسیم می‌کند، جایی که هر رأس از هر رأس دیگری قابل دسترسی باشد.
۷. توپولوژیکی مرتب‌سازی (**Topological Sorting**): رئوس یک گراف بدون دور جهت‌دار را به گونه‌ای مرتب می‌کند که برای هر یال جهت‌دار، رأس مقصد بعد از رأس مبدأ می‌آید.
۸. رنگ‌آمیزی گراف (**Graph Colouring**): رنگ‌هایی را به رئوس یک گراف اختصاص می‌دهد به گونه‌ای که هیچ دو رأس مجاوری رنگ یکسانی نداشته باشند که اغلب در زمان‌بندی و تخصیص منابع استفاده می‌شود.

<sup>۱</sup> به عبارت ساده، backtracking مانند امتحان کردن مسیرهای مختلف است. هنگامی که به یک بن‌بست می‌رسید، به آخرین انتخاب خود باز می‌گردید (برمی‌گردید) و مسیر دیگری را امتحان می‌کنید. در این مقاله، اصول اولیه‌ی backtracking، نحوی عملکرد آن و اینکه چگونه می‌تواند به حل انواع مشکلات چالش‌برانگیز کمک کند را بررسی می‌کنیم. این الگوریتم روشی برای پیدا کردن مسیر درست در میان انتخاب‌های پیچیده است.

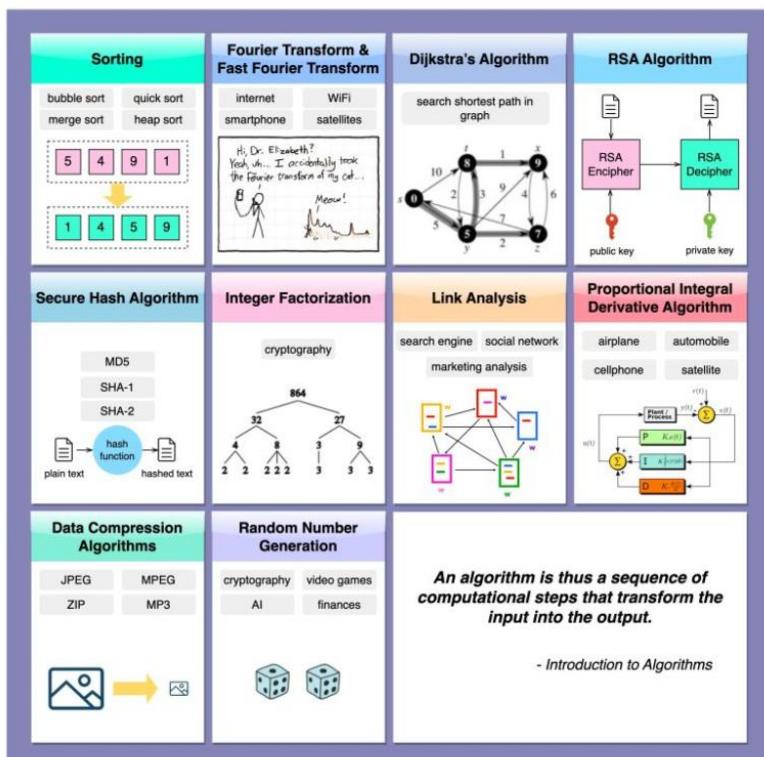
۹. جریان ماکزیمم (Maximum Flow): حداقل مقدار جریانی را تعیین می کند که می تواند از یک منبع تعیین شده به یک مقصد تعیین شده در یک شبکه جریان ارسال شود.

۱۰. جفت یابی (Matching): یالهایی را در یک گراف شناسایی می کند که به گونه ای هیچ دو یالی رأس مشترکی نداشته باشند که اغلب در تطابق گراف دو بخشی یا مسائل تخصیص استفاده می شود.

## ۱۰ الگوریتم مهم در دنیای ما

این الگوریتم ها در موتورهای جستجوی اینترنتی، شبکه های اجتماعی، وای فای، تلفن های همراه و حتی ماهواره ها استفاده می شوند.

### The 10 Algorithms That Dominate Our World blog.bytebytogo.com

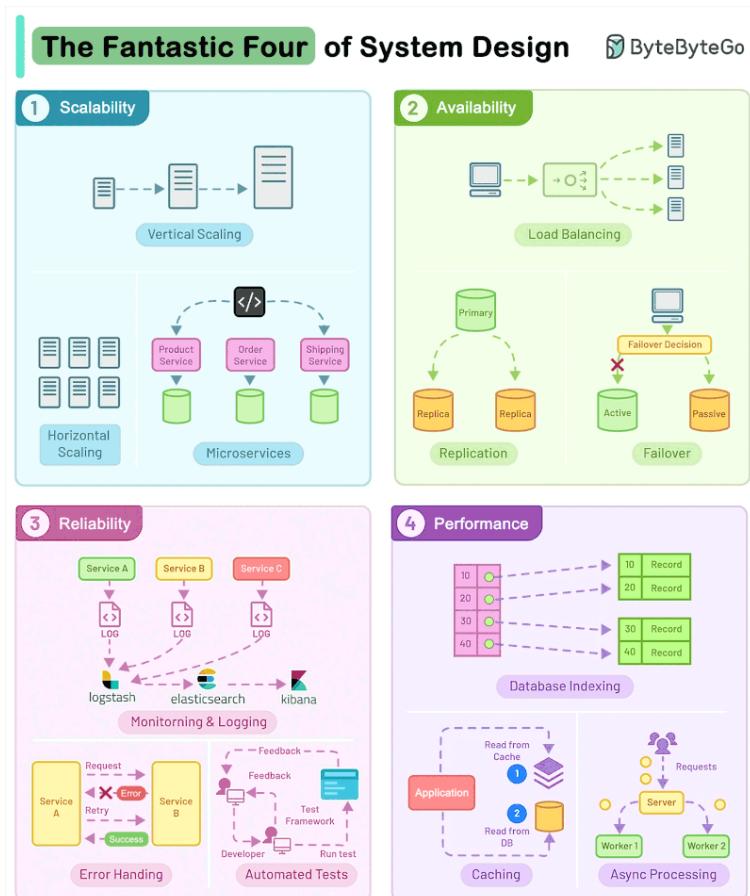


۱. مرتب‌سازی (Sorting)
۲. تبدیل فوریه و تبدیل فوریه سریع (Fourier Transform and Fast Fourier Transform)
۳. الگوریتم دایکسترا (Dijkstra's algorithm)
۴. الگوریتم RSA
۵. الگوریتم Secure Hash
۶. تحلیل لینک (Link Analysis)
۷. الگوریتم PID (Proportional Integral Derivative Algorithm)
۸. الگوریتم‌های فشرده‌سازی داده (Data compression algorithms)
۹. تولید‌کننده اعداد تصادفی (Random Number Generation)

# بررسی طراحی سیستم‌ها در دنیای واقعی

## چهار شگفت‌انگیز طراحی سیستم چه چیزهایی هستند؟

در طراحی سیستم، قابلیت مقیاس‌پذیری (Availability)، درسترس‌بودن (Scalability)، قابلیت اطمینان (Reliability) و کارایی (Performance) که به اختصار SARP نامیده می‌شوند، به عنوان «چهار شگفت‌انگیز» شناخته می‌شوند. زیرا آن‌ها ارکان اصلی یک سیستم نرم‌افزاری خوب و موفق هستند.



در اینجا به توضیح تک‌تک این مفاهیم به همراه جزئیات بیشتر می‌پردازیم:

### ۱. قابلیت مقیاس‌پذیری (Scalability):

این ویژگی تضمین می‌کند که اپلیکیشن شما بتواند با افزایش تعداد کاربران یا حجم کاری، بدون افت قابل توجه عملکرد، سازگار شود.

روش‌های پیاده‌سازی:

- مقیاس‌پذیری افقی (Horizontal Scaling): با افزودن سرورهای بیشتر، بار توزیع می‌شود.
- مقیاس‌پذیری عمودی (Vertical Scaling): ارتقای سرورهای موجود با منابع بیشتر (CPU، حافظه).
- تقسیم‌بندی داده‌ها در چندین پایگاه‌داده برای بهبود عملکرد. (Database Sharding)

### ۲. دردسترس‌بودن (Availability):

این ویژگی تضمین می‌کند که اپلیکیشن شما برای کاربران در دسترس و عملیاتی باشد.

روش‌های پیاده‌سازی:

- افزونگی (Redundancy): تکرار اجزای حیاتی (سرورها، پایگاه‌داده‌ها) برای مدیریت خرابی‌ها.
- توزیع بار (Load Balancing): توزیع درخواست‌های ورودی در سرورهای مختلف برای جلوگیری از اضافه‌بار.
- تحمل خطا (Fault Tolerance): طراحی سیستم‌هایی که با خطاها به درستی برخورد کرده و به سرعت بازیابی شوند.

### ۳. قابلیت اطمینان (Reliability):

این ویژگی بر توانایی سیستم در ارائه نتایج سازگار و دقیق در طول زمان تمرکز دارد.

روش‌های پیاده‌سازی:

- تست واحد (Unit Testing): آزمایش کامل واحدهای مستقل کد.

- تست ادغام (Integration Testing): تست چگونگی کارکرد اجزای مختلف با هم.
- مدیریت خطا (Error Handling): پیاده سازی مکانیزم هایی برای برخورد درست با شرایط غیرمنتظره.
- نظارت (Monitoring): نظارت مداوم بر سلامت و عملکرد سیستم.

#### ۴. کارایی (Performance)

این ویژگی به چگونگی کارآمدی استفاده هی سیستم از منابع برای تکمیل وظایف در یک بازه زمانی مطلوب اشاره دارد.

روش های پیاده سازی:

- بهینه سازی کد (Code Optimization): نوشتن کد کارآمد و با ساختار مناسب.
- کش (Caching): ذخیره داده های پر کاربرد برای کاهش بار روی پایگاه داده.
- شبکه های توزیع محتوا (CDN): تحویل محتوای استاتیک (تصاویر، ویدئوها) از سرور های توزیع شده جغرافیایی برای بهبود زمان پاسخ.

با تمرکز بر این چهار جنبه در مرحله طراحی، توسعه دهنده ها می توانند سیستم های نرم افزاری قوی و کاربر پسند بسازند که قادر به رسیدگی به تقاضاهای دنیای واقعی باشند.

### چگونه یک وب سایت را برای پشتیبانی از میلیون ها کاربر مقیاس پذیر کنیم؟

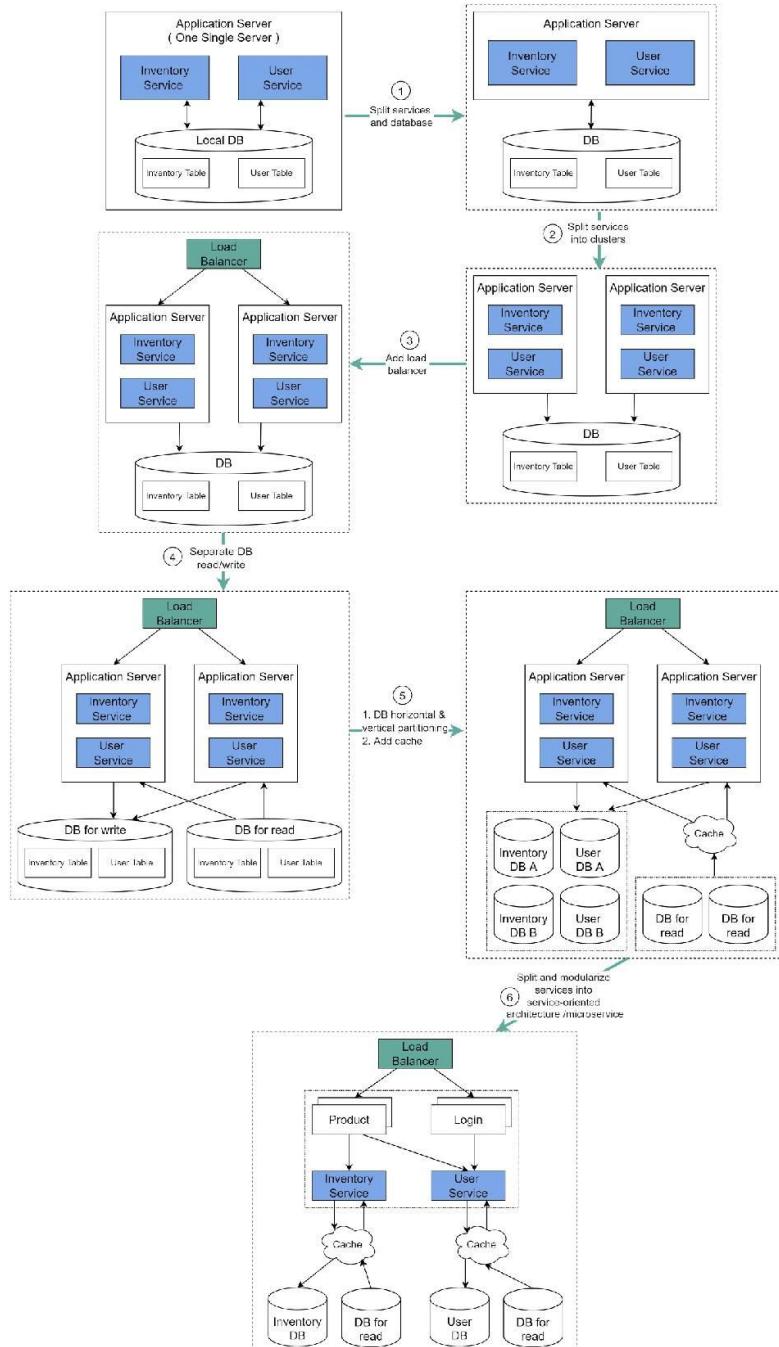
ما این را مرحله به مرحله توضیح خواهیم داد.

نمودار زیر تکامل یک وب سایت تجارت الکترونیکی ساده را نشان می دهد. از طراحی **monolithic** در یک سرور واحد به معناری سرویس گرا / میکروسرویس تغییر می کند.

فرض کنید دو سرویس داریم: سرویس موجودی (مدیریت توصیف محصول و موجودی) و سرویس کاربر (اطلاعات کاربر، ثبت نام، ورود و غیره را مدیریت می کند).

### How to Scale a Website Step-by-Step?

ByteByteGo



**مرحله ۱** - با رشد پایگاه کاربری، یک سرور اپلیکیشن واحد دیگر نمی‌تواند ترافیک را مدیریت کند. سرور برنامه و سرور پایگاهداده را در دو سرور جداگانه قرار می‌دهیم.

**مرحله ۲** - کسب و کار همچنان رو به رشد است و یک سرور برنامه واحد دیگر کافی نیست. بنابراین یک خوشهای<sup>۱</sup> از سرورهای برنامه را مستقر می‌کنیم.

**مرحله ۳** - اکنون درخواست‌های ورودی باید به چندین سرور برنامه مسیریابی شوند، چگونه می‌توانیم اطمینان حاصل کنیم که هر سرور برنامه بار یکسانی دریافت می‌کند؟ توزیع کننده بار<sup>۲</sup> این مورد را به خوبی مدیریت می‌کند.

**مرحله ۴** - با ادامه رشد کسب و کار، پایگاهداده ممکن است به گلوگاه تبدیل شود. برای رفع این مشکل، خواندن و نوشتمن را به نحوی جدا می‌کنیم که درخواست‌های خواندن مکرر به تکثیر کننده‌های خواندن<sup>۳</sup> ارسال شوند. با این تنظیم، پهناهی باند برای نوشتمن در پایگاهداده به طرز چشمگیری افزایش می‌یابد.

**مرحله ۵** - فرض کنید کسب و کار همچنان رو به رشد است. یک پایگاهداده واحد دیگر نمی‌تواند بار جدول موجودی دارایی‌ها<sup>۴</sup> و جدول کاربر<sup>۵</sup> را مدیریت کند. چند گزینه داریم:

cluster<sup>۱</sup>

load balancer<sup>۲</sup>

read replicas<sup>۳</sup>

inventory table<sup>۴</sup>

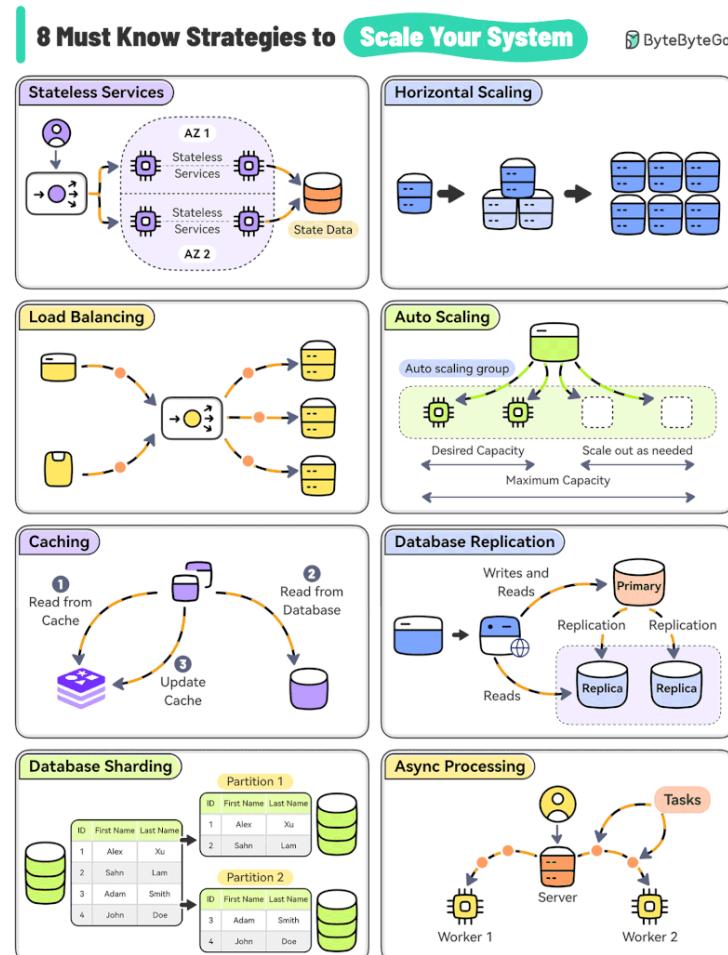
user table<sup>۵</sup>

۱. پارتيشن عمودی یا افزودن قدرت بیشتر (CPU، RAM و غیره) به سرور پایگاه‌داده. محدودیت سختی دارد.
۲. پارتيشن افقی با افزودن سرورهای پایگاه‌داده بیشتر.
۳. افزودن یک لایه حافظه کش برای تخلیه درخواست‌های خواندن.

مرحله ۶ - اکنون می‌توانیم کارکردها را به سرویس‌های مختلف تقسیم کنیم. معماری به صورت سرویس-گرا / میکروسرویس تبدیل می‌شود.

## راز موفقیت آمازون، نتفلیکس و اوبر چیست؟

آنها به طرز فوق العاده‌ای در مقیاس‌بندی سیستم‌های خود هر زمان که نیاز باشد، عمل می‌کنند. در اینجا ۸ استراتژی ضروری برای مقیاس‌بندی سیستم شما آورده شده است:



### ۱. سرویس‌های Stateless

سرویس‌های Stateless را طراحی کنید؛ زیرا آنها به داده‌های خاص سرور وابسته نیستند و مقیاس‌بندی آنها آسان‌تر است.

## ۲. مقیاس‌بندی افقی (Horizontal Scaling)

سرورهای بیشتری اضافه کنید تا حجم کاری قابل اشتراک باشد.

### Load Balancing .۳

از توزیع کننده بار برای توزیع یکنواخت درخواست‌های ورودی در سرورهای مختلف استفاده کنید.

### Auto Scaling .۴

برای تنظیم منابع بر اساس ترافیک لحظه‌ای، خط‌مشی‌های مقیاس‌بندی خودکار را پیاده‌سازی کنید.

### Caching .۵

برای کاهش بار روی پایگاه‌داده و مدیریت درخواست‌های تکراری در مقیاس، از کش‌کردن استفاده کنید.

### Database Replication .۶

برای مقیاس‌بندی عملیات خواندن و در عین حال بهبود افزونگی، داده‌ها را در چندین گره تکثیر کنید.

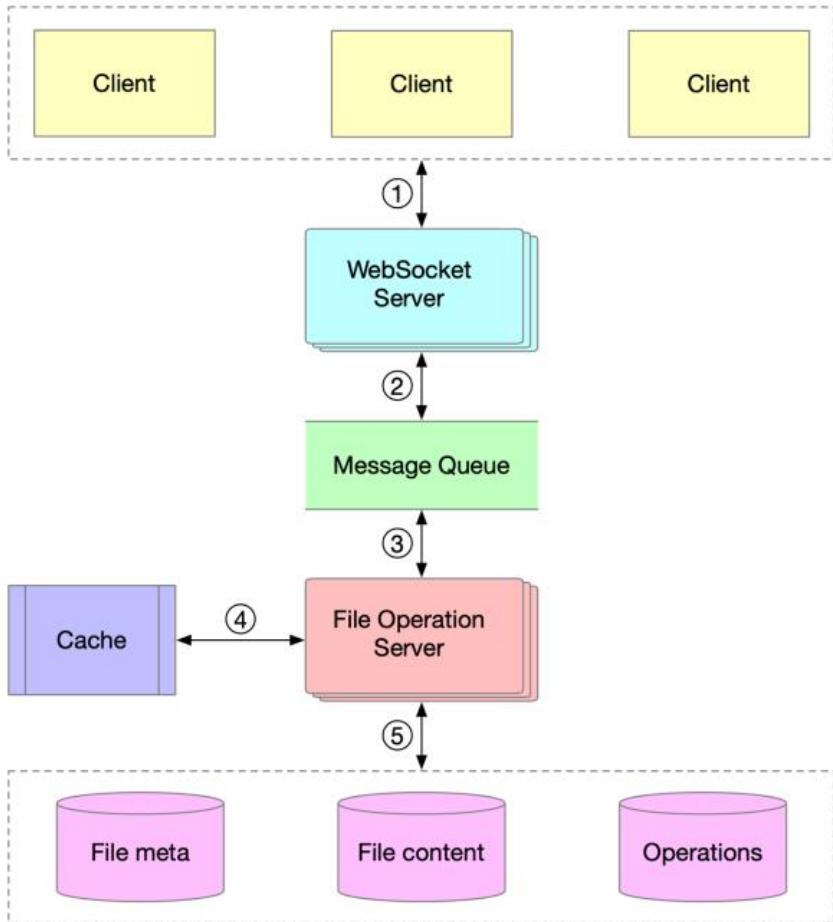
### Database Sharding .۷

داده‌ها را در چندین نمونه توزیع کنید تا هم نوشتن و هم خواندن مقیاس‌بندی شوند.

### Async Processing .۸

با استفاده از پردازش ناهم‌زمان، کارهای زمان‌بر و پر مصرف را به کارگران پس‌زمینه منتقل کنید تا درخواست‌های جدید را به صورت مقیاس‌پذیر مدیریت کنید.

## Google Docs طراحی



۱. کاربرها عملیات ویرایش سند را به سرور WebSocket ارسال می‌کنند.
۲. ارتباط بلادرنگ<sup>۱</sup> توسط سرور WebSocket مدیریت می‌شود.
۳. عملیات اسناد در صفحه پیام‌ها ذخیره می‌شود.

<sup>۱</sup> real-time

۴. سرور عملیات فایل، عملیات تولید شده توسط کاربرها را مورد استفاده قرار

می‌دهد و با استفاده از الگوریتم‌های همکاری<sup>۱</sup>، عملیات تبدیل شده را تولید

می‌کند.

۵. سه نوع داده ذخیره می‌شود: متادیتای فایل، محتوای فایل و عملیات.

یکی از بزرگ‌ترین چالش‌ها، حل تعارضات به صورت بلادرنگ است. الگوریتم‌های رایج

عبارت‌اند از:

- تبدیل عملیاتی<sup>۲</sup> (OT)
- همگام‌سازی تفاضلی<sup>۳</sup> (DS)
- نوع داده تکثیر شده بدون تعارض<sup>۴</sup> (CRDT)

بر اساس صفحه ویکی‌پدیای آن، Google Docs از OT استفاده می‌کند و CRDT یک حوزه تحقیقاتی فعال برای ویرایش همزمان بلادرنگ است.

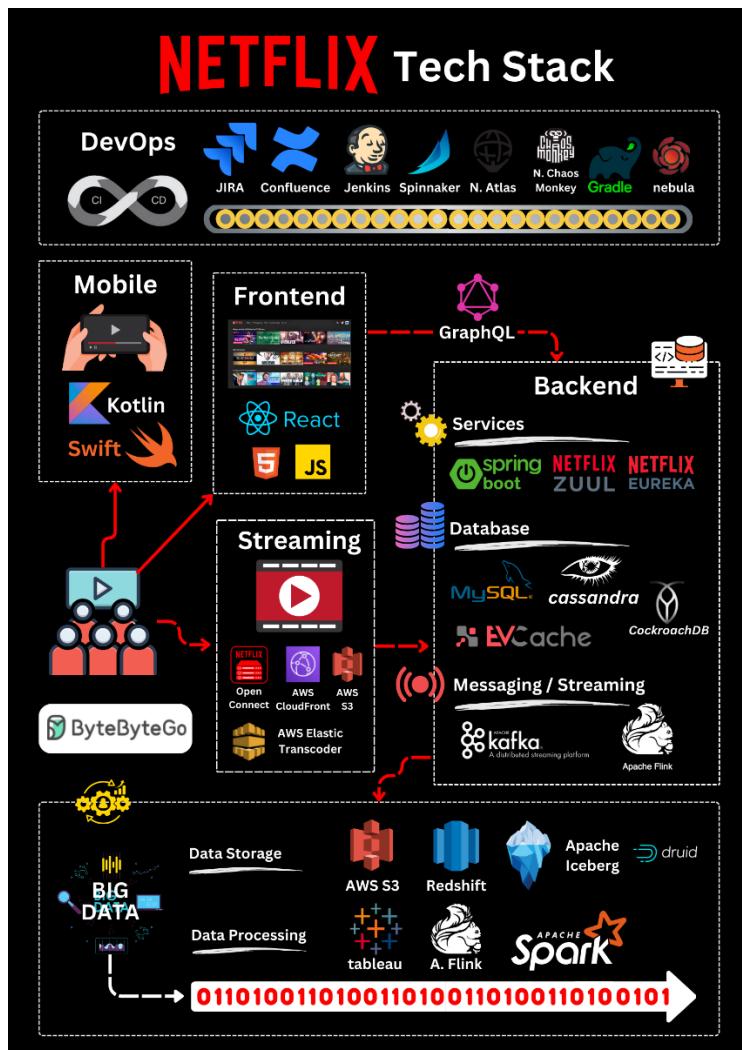
۱ collaboration algorithms

۲ Operational transformation

۳ Differential Synchronization

۴ Conflict-free replicated data type

## تکنولوژی مورد استفاده‌ی تفليكس

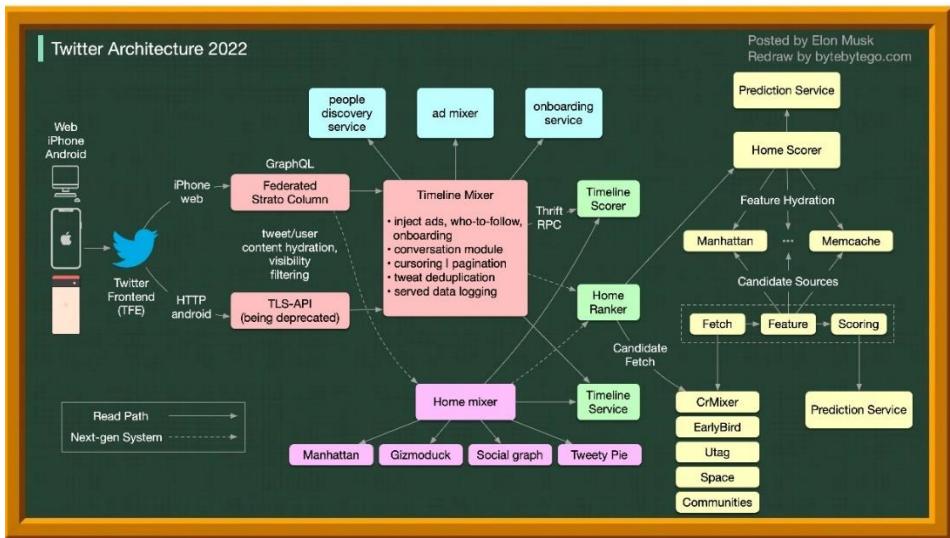


این متن بر اساس تحقیقات انجام شده روی بسیاری از بلاگ‌های مهندسی تفليكس و پروژه‌های متناز تهیه شده است.

- **موبایل و وب:** نتفلیکس برای ساخت اپلیکیشن‌های موبایلی سراغ Kotlin و Swift رفته است. برای وب اپلیکیشن خود نیز از React استفاده می‌کند.
- **ارتباط فرانت‌اند/سرور:** نتفلیکس از GraphQL استفاده می‌کند.
- **سرویس‌های بک‌اند:** نتفلیکس به Eureka، ZUUL، Spring Boot و Frimورک دیگر تکنولوژی‌ها تکیه می‌کند.
- **پایگاه‌های داده:** نتفلیکس از CockroachDB، Cassandra، EV cache و دیگر پایگاه‌های داده استفاده می‌کند.
- **پیام‌رسانی/استریمینگ:** نتفلیکس برای اهداف پیام‌رسانی و استریمینگ از Apache Kafka و Flink استفاده می‌کند.
- **ذخیره‌سازی ویدئو:** نتفلیکس از S3 و Open Connect برای ذخیره‌سازی ویدئو استفاده می‌کند.
- **پردازش داده:** نتفلیکس برای پردازش داده از Flink و Spark استفاده می‌کند که سپس با استفاده از Tableau به صورت بصری نمایش داده می‌شوند. برای پردازش اطلاعات انبار داده ساختاریافته به کار می‌رود.
- **CI/CD:** نتفلیکس از ابزارهای مختلفی نظیر JIRA، Confluence، Spinnaker، Chaos Monkey، Jenkins، Gradle و Atlas موارد دیگر برای فرایندهای CI/CD استفاده می‌کند.

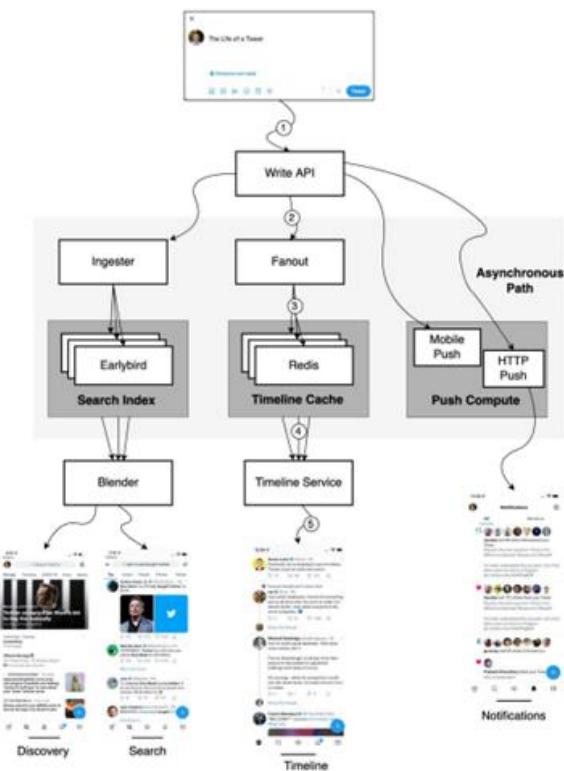
## ۲۰۲۲ معماری توییتر

این معماری واقعی توییتر است. توسط ایلان ماسک ارسال شده و توسط ما برای خوانایی بهتر دوباره ترسیم شده است.



## معماری توییتر در ۲۰۱۳

این پست خلاصه‌ای از سخنرانی فنی است که توسط توییتر در سال ۲۰۱۳ ارائه شده است. بیایید نگاهی بیندازیم.



زنگی یک توییت:

۱. یک توییت از طریق API نوشتن وارد می‌شود.
۲. API نوشتن درخواست را به سرویس Fanout هدایت می‌کند.
۳. سرویس Fanout پردازش‌های زیادی انجام می‌دهد و آنها را در کش Redis ذخیره می‌کند.

۴. از سرویس Timeline برای یافتن سرور Redis که تایم‌لاین home روی آن قرار دارد، استفاده می‌شود.
۵. یک کاربر تایم‌لاین home خود را از طریق سرویس Timeline دریافت می‌کند.

### جستجو و کشف

- Ingester: توییت‌ها را با علامت‌گذاری و توکن‌سازی می‌کند تا بتوان داده‌ها را فهرست‌بندی کرد.
- Earlybird: فهرست جستجو را ذخیره می‌کند.
- Blender: تایم‌لاین‌های جستجو و کشف را ایجاد می‌کند.

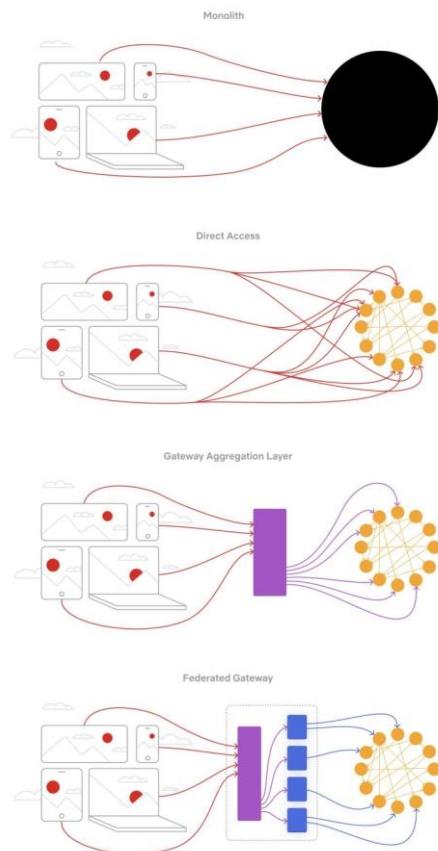
### PushCompute

- HTTP push
- Mobile push

## تکامل غیرمعمول معماری API نتفلیکس

معماری API نتفلیکس چهار مرحله اصلی را پشت سر گذاشته است:

Evolution of an API Architecture



- **یکپارچه (Monolith):** برنامه به صورت یکپارچه بسته‌بندی و مستقر می‌شود، مانند یک فایل JAR جاوا یا برنامه‌های مبتنی زبان برنامه‌نویسی Rails. اکثر استارت‌آپ‌ها با معماری یکپارچه شروع می‌کنند.

- دسترسی مستقیم (**Direct Access**): در این معماری، یک برنامه سمت کلاینت می‌تواند مستقیماً به میکروسرویس‌ها درخواست ارسال کند. با وجود صدها یا حتی هزاران میکروسرویس، در معرض قراردادن همه آن‌ها برای کاربران ایده‌آل نیست.
- لایه تجمعی دروازه (**Gateway Aggregation Layer**): برخی از سناریوهای استفاده ممکن است شامل چندین سرویس باشد، به یک‌لایه تجمعی دروازه نیاز داریم. تصور کنید برنامه نتفلیکس برای رِندر کردن رابط کاربری به ۳ API نیاز دارد (فیلم، تولید، استعداد). لایه تجمعی دروازه این امکان را فراهم می‌کند.
- دروازه فدرال (**Federated Gateway**): با افزایش تعداد توسعه‌دهنده‌گان و پیچیدگی دامنه، توسعه لایه تجمعی API دشوارتر می‌شود. فدرال کردن GraphQL به نتفلیکس اجازه می‌دهد تا یک دروازه GraphQL واحد راهاندازی کند که داده‌ها را از تمام API‌های دیگر دریافت می‌کند.

### نکته مصاحبه طراحی سیستم

یک نکته حرفه‌ای برای موفقیت در مصاحبه طراحی سیستم، خواندن و بلاگ مهندسی شرکتی است که در آن مصاحبه می‌شوید. می‌توانید درک خوبی از اینکه چه فناوری را استفاده می‌کنند، چرا این فناوری بر دیگران ترجیح داده شده است و یادگیری اینکه چه مسائلی برای مهندسان مهم است را داشته باشید.



...

Interview pro-tip: To those interviewing for our engineering roles - checkout some of these key blog posts that can help you understand our architecture and prepare for the System Design rounds. 1/5



11:36 AM · Oct 27, 2021 · Twitter Web App

59 Retweets 5 Quote Tweets 222 Likes

به عنوان مثال، اینجا ۴ پست و بلاگی است که مهندسی توییتر آن را توصیه می‌کند:

۱. زیرساخت پشت توییتر: مقیاس‌پذیری

۲. شناسایی و مصرف داده‌های تحلیلی در توییتر

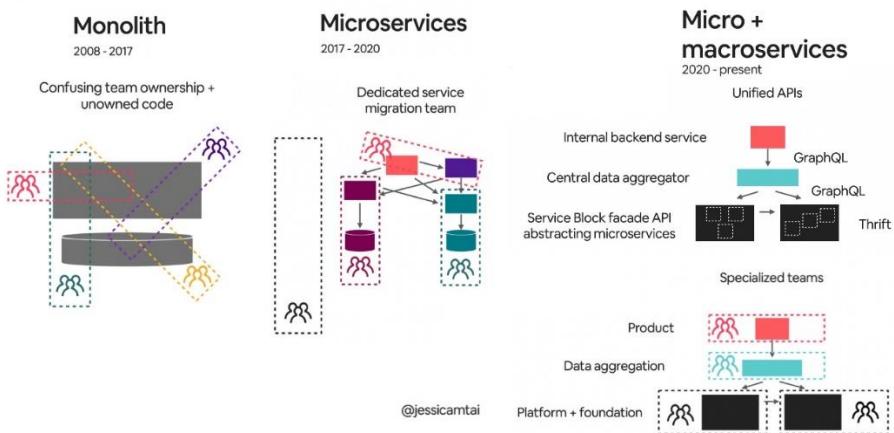
۳. علت آزمایش محصول در توییتر

۴. تجربیات توییتر: بررسی‌های فنی

## تحول معماری میکروسرویس Airbnb طی ۱۵ سال گذشته

معماری میکروسرویس Airbnb سه مرحله اصلی را پشت سر گذاشته است.

### Airbnb's Microservice Architecture



### مونولیت (۲۰۰۸ تا ۲۰۱۷)

Airbnb به عنوان یک بازار ساده برای میزبانان و مهمنان کار خود را آغاز کرد. این با یک اپلیکیشن Ruby on Rails ساخته شده است و دارای یک سیستم یکپارچه<sup>۱</sup> است.

### چه چالش‌هایی وجود دارد؟

- مالکیت گیج‌کننده تیم + کدی که مالک مشخصی ندارد.
- استقرار آهسته

### میکروسرویس‌ها (۲۰۱۷ تا ۲۰۲۰)

<sup>1</sup> monolith

میکروسرویس‌ها با هدف حل کردن این چالش‌ها به وجود آمدند. در معماری میکروسرویس، سرویس‌های کلیدی شامل موارد زیر می‌شوند:

- سرویس دریافت اطلاعات.
- سرویس داده مربوط به Business logic.
- سرویس جریان کاری<sup>۱</sup> نوشتن.
- سرویس تلفیق رابط کاربری.
- هر سرویس یک تیم مالک داشت.

### چه چالش‌هایی وجود دارد؟

مدیریت صدها سرویس و وابستگی‌ها برای انسان‌ها دشوار بود.

### میکروسرویس + ماکرو (۲۰۲۰ تاکنون)

این چیزی است که Airbnb در حال حاضر روی آن کار می‌کند. مدل ترکیبی میکروسرویس و ماکروسرویس بر روی یکپارچه‌سازی API‌ها تمرکز دارد.

## Microrepo در مقابل Monorepo

کدام‌یک بهتر است؟ چرا شرکت‌های مختلف گزینه‌های متفاوتی را انتخاب می‌کنند؟

	Monorepo	Microrepo
Company	     	    
Collaboration	 <p>Services work under the same repository.</p>	 <p>Service owners work under separate repositories.</p>
Dependency	 <p>Service share the same dependency.</p>	 <p>Service choose their own dependency.</p>
Scalability	 <p>Services share the same standard.</p>	 <p>Services set their own standard.</p>
Tooling	   	   

Monorepo چیز جدیدی نیست؛ لینوکس و ویندوز هر دو با استفاده از ساخته شده‌اند. برای بهبود قابلیت مقیاس‌پذیری و سرعت ساخت، گوگل مجموعه ابزار اختصاصی

داخلی خود را برای مقیاس‌پذیری سریع‌تر و استانداردهای سخت‌گیرانه کیفیت کد برای حفظ ثبات آن توسعه داد.

آمازون و نتفلیکس سفیران اصلی فلسفه میکروسرویس هستند. این رویکرد به طور طبیعی کد سرویس را به مخازن<sup>۱</sup> جداگانه تقسیم می‌کند. این روش سریع‌تر مقیاس‌پذیر می‌شود، اما می‌تواند بعداً به مشکلات منجر شود.

در Monorepo، هر سرویس یک پوشه است و هر پوشه دارای یک پیکربندی BUILD و کنترل مجوز OWNERS است. هر عضو سرویس مسئول پوشه خود است.

از طرف دیگر، در Microrepo، هر سرویس مسئول repository خود است، با پیکربندی ساخت و مجوزهایی که به طور معمول برای کل repository تنظیم شده‌اند.

در Monorepo، وابستگی‌ها در همه‌ی قسمت‌های کد منبع<sup>۲</sup>، صرف‌نظر از کسب‌وکار شما به اشتراک گذاشته می‌شوند، بنابراین زمانی که ارتقای نسخه‌ای وجود دارد، هر کد منبع نسخه‌ی خود را ارتقا می‌دهد.

در Microrepo، وابستگی‌ها در هر مخزن<sup>۳</sup> کنترل می‌شوند. نظام‌مندی برنامه بر اساس برنامه‌های زمانی خود تصمیم می‌گیرند نسخه‌های خود را چه زمانی ارتقا دهند.

<sup>۱</sup> repositories

<sup>۲</sup> codebase

<sup>۳</sup> repository

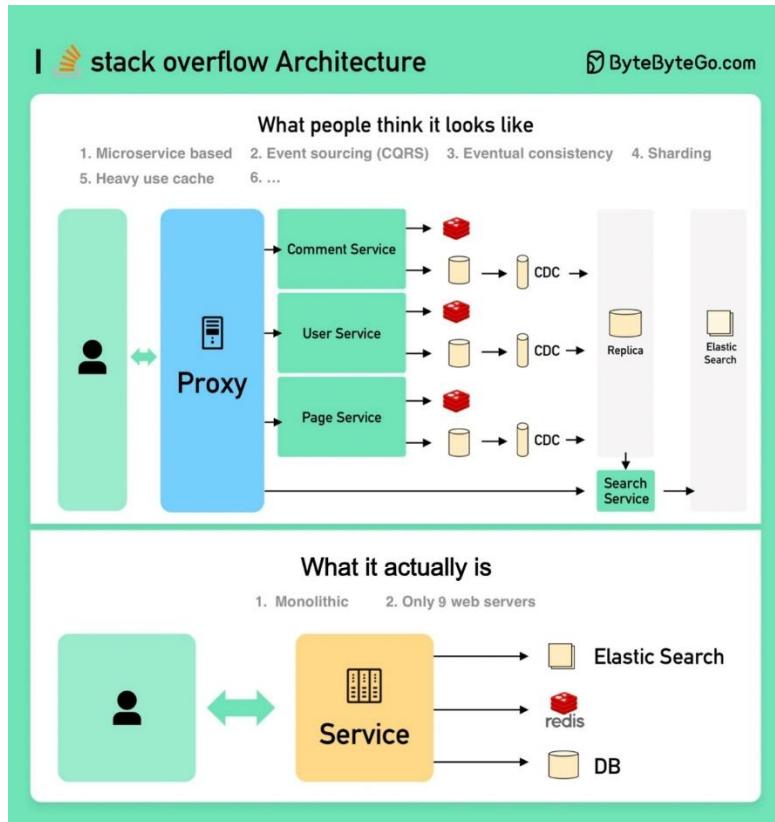
استانداردهایی برای بررسی و تحلیل شدن کیفیت خود دارد. فرایند بررسی کد در گوگل به طور معروفی برای تعیین سطح بالای کیفیت شناخته شده است و اطمینان از یک استاندارد باکیفیت منسجم برای Monorepo، صرف نظر از کسب‌وکار را تضمین می‌کند.

Microrepo می‌تواند استاندارد خود را تعیین کند یا با ترکیب بهترین شیوه‌ها، یک استاندارد مشترک را اتخاذ کند. این می‌تواند برای کسب‌وکاری که نیازمند پارامتر مقیاس‌پذیری سریع تر باشد مناسب باشد هر چند ممکن است کیفیت کد کمی متفاوت باشد. مهندسان گوگل Bazel را ساختند و مهندسان فیس‌بوک، Buck را ساخت. ابزارهای مترباز دیگری از جمله Nx و موارد دیگر در دسترس هستند. Lerna

در طول سال‌ها، Microrepo از ابزارهای بیشتری پشتیبانی کرده است، از جمله Maven و NPM برای جاوا، CMake و NodeJS برای C++ و غیره. Gradle

## چگونه وبسایت Stack Overflow را طراحی می‌کنید؟

اگر پاسخ شما سرورهای<sup>۱</sup> on-premise<sup>۲</sup> و monolith (در تصویر زیر) باشد، به احتمال زیاد در مصاحبه رد خواهد شد، اما این همان چیزی است که در واقعیت ساخته شده است!



تصویر مردم در مورد اینکه وبسایت Stack Overflow چگونه باید باشد چیست؟

مصاحبه‌کننده احتمالاً انتظار چیزی شبیه به قسمت بالای تصویر را دارد.

<sup>۱</sup> داخلی

<sup>۲</sup> یکپارچه

- از میکروسرویس برای تجزیه سیستم به اجزای کوچک استفاده می‌شود.
- هر سرویس پایگاهداده‌ی خود را دارد تا می‌توانید از کش استفاده کنید.
- سرویس‌ها shard شده هستند.
- سرویس‌ها به صورت ناهم‌زمان از طریق صف پیام با یکدیگر صحبت می‌کنند.
- این سرویس با استفاده از Event Sourcing با CQRS اجرا می‌شود.
- به رخ‌کشیدن دانش در سیستم‌های توزیع‌شده مانند یکپارچگی تدریجی، تئوری CAP و غیره.

اما آنچه در واقعیت است.

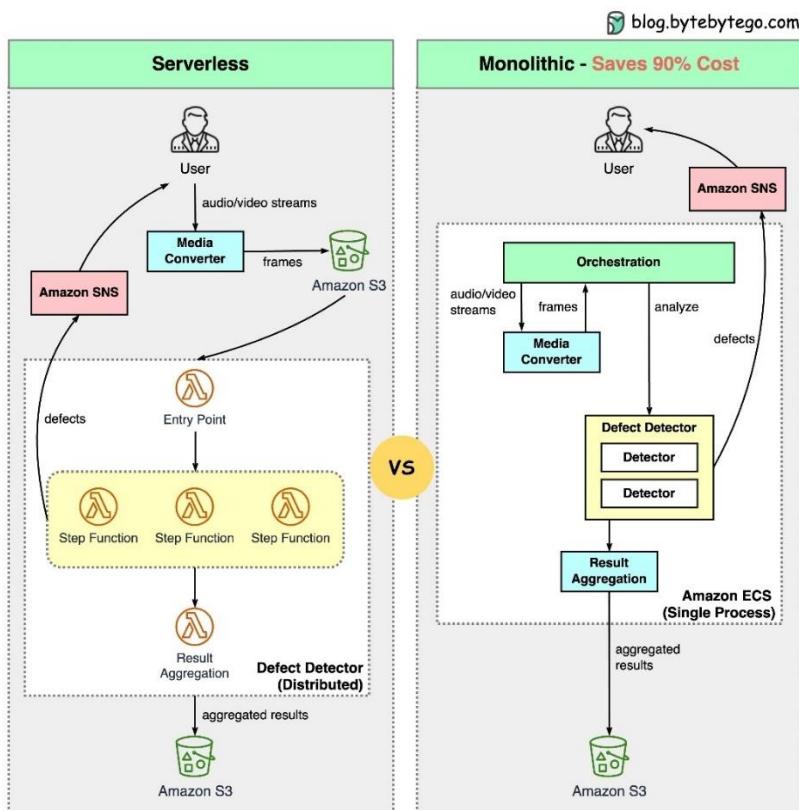
کل ترافیک را تنها با ۹ سرور وب داخلی<sup>۱</sup> مدیریت می‌کند و معماری آن monolith است! همچنین سرورهای اختصاصی خود را دارد و روی ابر اجرا نمی‌شود.

این برخلاف تمام باورهای رایج ما در این روزهاست.

## چرا مانیتورینگ ویدئوی آمازون از Monolithic به Serverless تغییر یافت؟

آمازون چگونه می‌تواند حدود ۹۰ درصد در هزینه‌های خود را صرفه‌جویی کند؟  
نمودار زیر مقایسه معماری قبل و بعد از این مهاجرت را نشان می‌دهد.

### Amazon Prime Video monitoring - From Serverless to Monolithic



سرویس مانیتورینگ ویدئوی پرایم آمازون چیست؟

سرویس Prime Video نیاز به مانیتورینگ بر کیفیت هزاران استریم<sup>۱</sup> زنده دارد. ابزار مانیتورینگ به طور خودکار استریم‌ها را به صورت لحظه‌ای تجزیه و تحلیل می‌کند و مشکلات کیفیتی مانند خرابی بلوك، فریز شدن ویدئو و مشکلات همگام‌سازی را شناسایی می‌کند. این فرایند برای رضایت کاربر بسیار مهم است.

در اینجا سه مرحله اصلی وجود دارد: تبدیل کننده<sup>۲</sup> media، آشکارساز خطأ<sup>۳</sup> و نوتیفیکیشن لحظه‌ای.

### مشکل معماری قدیمی چیست؟

معماری قدیمی مبتنی بر Amazon Lambda بود که برای ساخت سریع سرویس‌ها مناسب بود. با این حال، هنگام اجرای معماري در مقیاس بالا، از نظر هزینه مقرنون به صرفه نبود. دو مورد از پرهزینه‌ترین عملیات عبارت‌اند از:

۱. گردش کار ارکستراسیون<sup>۴</sup> توابع گام AWS<sup>۵</sup>، کاربران را بر اساس انتقال حالت<sup>۶</sup> شارژ می‌کند و ارکستراسیون هر ثانیه چندین انتقال حالت را انجام می‌دهد.
۲. انتقال داده بین اجزای توزیع شده – این داده‌های میانی در Amazon S3 ذخیره می‌شوند تا مرحله بعدی بتواند آن‌ها را دانلود کند. دانلود داده با حجم بالا می‌تواند پرهزینه باشد.

- معماری یکپارچه<sup>۷</sup> حدود ۹۰ درصد هزینه را صرفه‌جویی می‌کند

stream<sup>۸</sup>

media converter<sup>۹</sup>

defect detector<sup>۱۰</sup>

Orchestration<sup>۱۱</sup>

Step Functions<sup>۱۲</sup>

State Transitions<sup>۱۳</sup>

Monolithic<sup>۱۴</sup>

معماری یکپارچه برای رفع مشکلات مربوط به هزینه طراحی شده است. همچنان سه جزء وجود دارد، اما تبدیل کننده media و آشکارساز خطا در یک فرایند/پروسس مشابه مستقر می‌شوند و هزینه انتقال داده را از طریق شبکه را صرفه‌جویی می‌کنند. جای تعجب نیست که این رویکرد به تغییر معماری استقرار منجر به ۹۰ درصد صرفه‌جویی در هزینه شد!

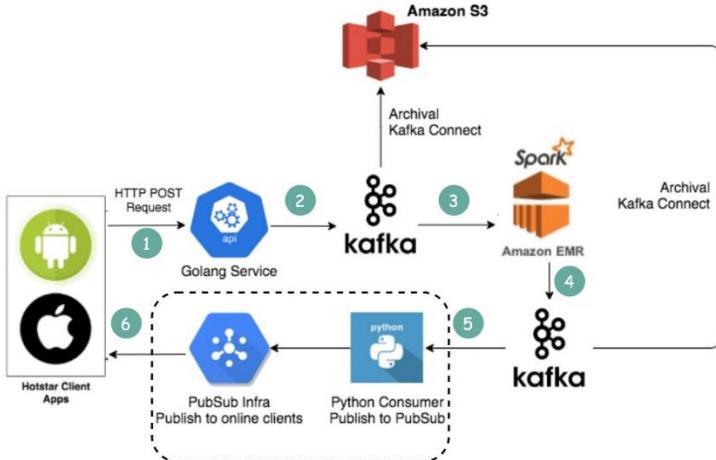
این یک مطالعه موردنی جالب و منحصربه‌فرد است؛ زیرا میکروسرویس‌ها به یک انتخاب رایج و مدروز در صنعت فناوری تبدیل شده‌اند. خوب است که می‌بینیم بحث‌های بیشتری در مورد تکامل معماری و بحث‌های صادقانه‌تر در مورد جوانب مثبت و منفی آن داریم. تجزیه اجزای یک سیستم یکپارچه به میکروسرویس‌های توزیع شده با هزینه همراه است.

### رهبران آمازون در این باره چه گفتند؟

آقای Werner Vogels، مدیر ارشد فناوری آمازون: «ساختن سیستم‌های نرم‌افزاری قابل تکامل یک استراتژی است، نه یک مذهب. بازنگری در معماری مورداستفاده خود با ذهنی باز یک مسئله ضروری است.»

Adrian Cockcroft، معاون سابق بخش پایداری آمازون: «تیم پرایم ویدئو مسیری را دنبال کردند که من آن را Serverless First می‌نامم. من طرف‌دار فقط Serverless نیستم.»

## چگونه Disney Hotstar ۵ میلیارد ایموجی را در طول یک تورنمنت ثبت می‌کند؟



۱. کاربران ایموجی‌ها را از طریق درخواست‌های استاندارد HTTP ارسال می‌کنند.

سرویس Golang را می‌توان به عنوان یک وب سرور معمولی در نظر گرفت. زبان

برنامه‌نویسی Golang به دلیل پشتیبانی خوب از همزمانی<sup>۱</sup> انتخاب شده است.

۲. از آنجایی که حجم نوشتمن (write volume) بسیار بالا است، از Kafka که یک صفحه مگابایت) هستند.

۳. داده‌های ایموجی توسط یک سرویس پردازش استریم<sup>۲</sup> به نام Spark جمع‌آوری می‌شوند. این سرویس داده‌ها را هر ۲ ثانیه (قابل تنظیم) جمع‌آوری می‌کند. بین

concurrency<sup>۱</sup>

message queue<sup>۲</sup>

streaming processing<sup>۳</sup>

فاصله زمانی و منابع محاسباتی موردنیاز، یک مبادله<sup>۱</sup> وجود دارد. فاصله زمانی کوتاه‌تر به معنای تحویل سریع‌تر ایموجی‌ها به سایر کاربران است، اما به منابع محاسباتی بیشتری نیز نیاز دارد.

۴. داده‌های جمع‌آوری شده در یک Kafka دیگر نوشته می‌شوند.

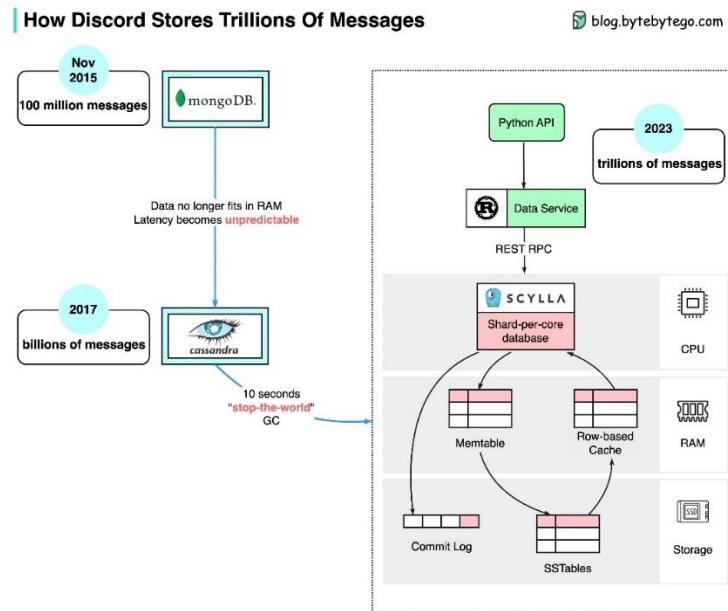
۵. مصرف‌کنندگان PubSub داده‌های ایموجی جمع‌آوری شده را از Kafka دریافت می‌کنند.

۶. ایموجی‌ها از طریق زیرساخت PubSub به صورت بلاذرنگ به سایر کاربران تحویل داده می‌شوند. زیرساخت PubSub موضوعی بسیار جالب است. شرکت Hotstar پروتکل‌های زیر را در نظر گرفت: NATS، Socketio، MQTT و gRPC و در نهایت MQTT را انتخاب کرد.

همچنین یک طراحی مشابه توسط لینکدین که یک میلیون لايك در ثانیه را استریم می‌کند، اتخاذ شده است.

## چگونه Discord تریلیون‌ها پیام را ذخیره می‌کند.

نمودار زیر تکامل ذخیره‌سازی پیام در Discord را نشان می‌دهد:



### MongoDB → Cassandra → ScyllaDB

در سال ۲۰۱۵، اولین نسخه Discord روی یک MongoDB replica ساخته شد. در حدود نوامبر ۲۰۱۵، ۱۰۰ میلیون پیام را ذخیره کرد و مقدار RAM موجود دیگر نمی‌توانست داده‌ها را نگه دارد و هیچ فضایی برای ایندکس باقی نمانده بوده. تأخیر<sup>۱</sup> غیرقابل‌پیش‌بینی شده بود و نیاز به انتقال ذخیره‌سازی پیام به یک پایگاه‌داده دیگر بود. در نتیجه Cassandra به عنوان گزینه بهتر انتخاب شد.

در سال ۲۰۱۷ دارای ۱۲ گره Cassandra بود و میلیارد‌ها پیام را ذخیره می‌کرد.

در ابتدای سال ۲۰۲۲ دارای ۱۷۷ گره با تریلیون‌ها پیام بود. در این مرحله، تأخیر غیرقابل‌پیش‌بینی بود و اجرای عملیات نگهداری بسیار پرهزینه شد.

دلایل مختلفی برای این مشکل وجود دارد:

• برای ساختار داده داخلی از درخت LSM استفاده می‌کند. خواندن داده‌ها نسبت به نوشتن آن‌ها پرهزینه‌تر هستند. در سروری با صدها کاربر، ممکن است خواندن‌های همزمان زیادی وجود داشته باشد که منجر به نقاط بحرانی یا در اصطلاح hotspots می‌شود.

• نگهداری خوش‌ها، مانند فشرده‌سازی‌های SSTable ها، بر کارایی سیستم تأثیر می‌گذارد.

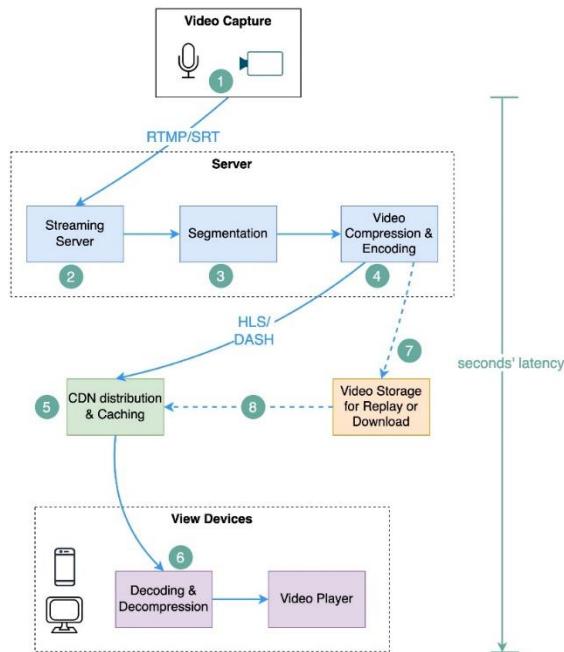
توقف‌های مربوط به Garbage collection باعث افزایش قابل توجه تأخیر می‌شود.

ScyllaDB یک پایگاه‌داده سازگار با Cassandra است که با C++ نوشته شده است. Discord معماری خود را برای داشتن یک API یکپارچه، یک سرویس داده‌ای نوشته‌شده با Rust و ذخیره‌سازی مبتنی بر ScyllaDB بازطراحی کرد. تأخیر خواندن p99 در ScyllaDB برابر با ۱۵ میلی‌ثانیه در مقایسه با ۴۰-۱۲۵ میلی‌ثانیه در Cassandra است. تأخیر نوشتن p99 برابر با ۵ میلی‌ثانیه در مقایسه با ۵-۷۰ میلی‌ثانیه در Cassandra است.

## چگونه پخش زنده ویدئویی در یوتیوب، TikTok live یا Twitch کار می‌کند؟

پخش زنده با پخش معمولی متفاوت است؛ زیرا محتوای ویدئویی به صورت بلاذرنگ از طریق اینترنت ارسال می‌شود که معمولاً<sup>۱</sup> با تأخیر چند ثانیه‌ای همراه است. نمودار زیر توضیح می‌دهد که چه اتفاقی در پشت‌صحنه برای این امر رخ می‌دهد.

### How does Live Streaming Work? [blog.bytebytego.com](http://blog.bytebytego.com)



مرحله ۱: داده‌های خام ویدئو توسط میکروفون و دوربین ضبط می‌شوند. داده‌ها به سمت سرور ارسال می‌شوند.

مرحله ۲: داده‌های ویدئویی فشرده و کدگذاری می‌شوند. برای مثال، با استفاده از الگوریتم‌های فشرده‌سازی، پس‌زنینه و سایر عناصر ویدئو را جدا می‌کند. پس از فشرده‌سازی،

<sup>1</sup> latency

ویدئو با استانداردهای مانند H.264 کدگذاری می‌شود. حجم داده‌های ویدیویی پس از این مرحله بسیار کوچک‌تر می‌شود.

مرحله ۳: داده‌های کدگذاری شده به بخش‌های کوچک‌تر، معمولاً<sup>۱</sup> به طول ثانیه تقسیم می‌شوند، بنابراین زمان دانلود یا استریم به میزان قابل توجهی کاهش می‌یابد.

مرحله ۴: داده‌های تقسیم شده به سرور استریم ارسال می‌شوند. سرور استریم نیاز به پشتیبانی از دستگاه‌های مختلف و شرایط شبکه دارد. این به «پخش استریم با نرخ بیت تطبیقی<sup>۲</sup>» معروف است. این بدان معناست که ما نیاز به تولید چندین فایل با نرخ بیت متفاوت در مراحل ۲ و ۳ داریم.

مرحله ۵: داده‌های پخش زنده توسط سرورهای لبه<sup>۳</sup> پشتیبانی شده توسط CDN (شبکه توزیع محتوا) پخش می‌شوند. میلیون‌ها کاربر می‌توانند ویدئو را از یک سرور لبه نزدیک تماشا کنند. CDN به طور قابل توجهی تأخیر انتقال داده را کاهش می‌دهد.

مرحله ۶: دستگاه‌های نمایش‌دهنده داده‌های ویدئویی را رمزگشایی و فشرده‌سازی می‌کنند و ویدئو را در یک پخش‌کننده ویدئویی پخش می‌کنند.

مراحل ۷ و ۸: اگر ویدئو نیاز به ذخیره برای بازپخش داشته باشد، داده‌های کدگذاری شده به یک سرور ذخیره‌سازی ارسال می‌شوند و بینندگان می‌توانند بعداً درخواست بازپخش از آن داشته باشند.

پروتکل‌های استاندارد برای پخش زنده شامل موارد زیر هستند:

- Macromedia RTMP (پروتکل پیام‌رسانی بلادرنگ<sup>۱</sup>): این پروتکل در ابتدا توسط برای انتقال داده بین یک پخش‌کننده فلش و یک سرور توسعه داده شد. اکنون برای پخش داده‌های ویدئویی از طریق اینترنت استفاده می‌شود. توجه داشته باشید که برنامه‌های ویدئوکنفرانس مانند اسکایپ از پروتکل<sup>۲</sup> RTC برای تأخیر کمتر استفاده می‌کنند.
- HLS (پخش زنده<sup>۳</sup> HTTP): این پروتکل به کدگذاری H.264 یا H.265 نیاز دارد. دستگاه‌های اپل فقط فرمت HLS را قبول می‌کنند.
- DASH<sup>4</sup>: در عمل DASH از دستگاه‌های اپل پشتیبانی نمی‌کند.
- هر دو HLS و DASH از پخش استریم با نرخ بیت تطبیقی پشتیبانی می‌کنند.

Real-Time Messaging Protocol <sup>۱</sup>

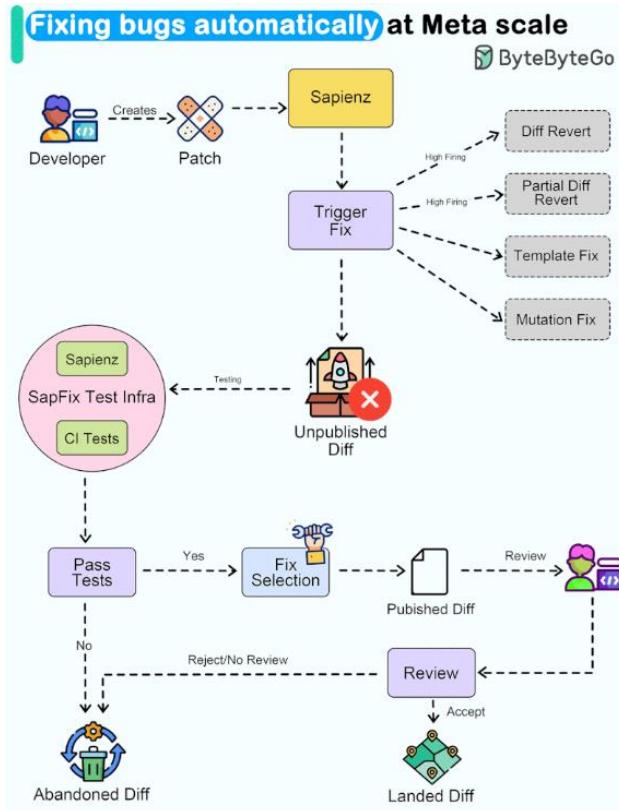
Real-Time Communication <sup>۲</sup>

Real-Time Communication <sup>۳</sup>

Dynamic Adaptive Streaming over HTTP <sup>۴</sup>

## چگونه یک سیستم به طور خودکار باگ‌ها را برای ما شناسایی و رفع کند؟

شرکت Meta مقاله‌ای در مورد نحوه خودکارسازی کامل فرایند رفع باگ در مقیاس فیسبوک منتشر کرد. هدف ابزاری به نام SapFix، ساده‌سازی فرایند اشکال‌زدایی با تولید خودکار رفع باگ برای موارد خاص است.



## چقدر موفق بوده است؟

در اینجا برخی جزئیاتی که در دسترس قرار گرفته است، آورده شده است: این ابزار در شش برنامه کلیدی از مجموعه برنامه‌های فیسبوک (فیسبوک، مسنجر، اینستاگرام، FBLite و Workplace) استفاده می‌شود. هر برنامه شامل دهها میلیون خط کد است.

- در یک دوره آزمایشی ۹۰ روزه، ۱۶۵ وصله<sup>۱</sup> برای ۵۷ خرابی ایجاد کرد.
  - میانگین زمان بین تشخیص باگ تا ارسال رفع باگ برای تأیید انسان، ۶۹ دقیقه بود.
- در اینجا نحوه عملکرد SapFix آورده شده است:
۱. توسعه‌دهندگان با استفاده از (Phabricator CI سیستم فیس‌بوک) تغییرات را برای بررسی ارسال می‌کنند.
  ۲. SapFix موارد تست مناسب را از Sapienz (سیستم طراحی خودکار سناریوهای تست فیس‌بوک) انتخاب می‌کند و آن‌ها را روی تغییرات ارسالی برای بررسی اجرا می‌کند.
  ۳. هنگامی که SapFix خرابی ناشی از تغییرات را تشخیص می‌دهد، سعی می‌کند رفع باگ‌های بالقوه‌ای را ایجاد کند. چهار نوع رفع باگ وجود دارد: الگو<sup>۲</sup>، جهش<sup>۳</sup>، بازگشت کامل<sup>۴</sup> و بازگشت جزئی<sup>۵</sup>
  ۴. برای ایجاد رفع باگ، SapFix روی نسخه‌های patch شده تست اجرا می‌کند و بررسی می‌کند که چه چیزی کار می‌کند. این فرایند را مانند حل یک پازل با امتحان‌کردن قطعه‌های مختلف تصور کنید.
  ۵. پس از تست شدن patch‌ها، SapFix یک وصله/patch کاندید را انتخاب می‌کند و آن را از طریق Phabricator برای بررسی به یک بررسی‌کننده انسانی ارسال می‌کند.
  ۶. بررسی‌کننده اصلی، توسعه‌دهنده‌ای است که تغییری را ایجاد کرده که باعث خرابی شده است. این توسعه‌دهنده اغلب بهترین درک فنی از مسئله را دارد. همچنین سایر مهندسان نیز در جریان تغییرات پیشنهادی قرار می‌گیرند.
  ۷. توسعه‌دهنده می‌تواند وصله پیشنهادی توسط SapFix را بپذیرد. با این حال، توسعه‌دهنده همچنین می‌تواند رفع باگ را رد کرده و آن را کنار بگذارد.

---

patches<sup>۱</sup>

template<sup>۲</sup>

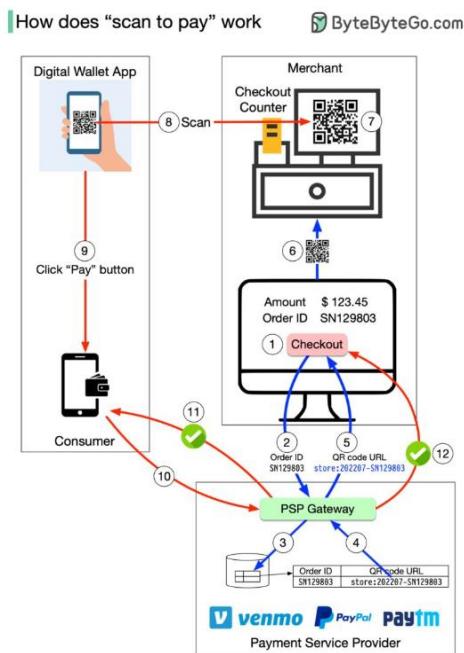
mutation<sup>۳</sup>

full revert<sup>۴</sup>

partial revert<sup>۵</sup>

## چگونه با اسکن کردن کد QR از کیف پول دیجیتال پرداخت کنیم؟

برای درک فرایند درگیر، باید فرایند «اسکن برای پرداخت» را به دو زیر فرایند تقسیم کنیم:



- فروشگاه کد QR تولید می‌کند و آن را روی صفحه‌نمایش می‌دهد.
- مشتری کد QR را اسکن می‌کند و پرداخت را انجام می‌دهد.

در اینجا مراحل تولید کد QR آورده شده است:

۱. هنگامی که می‌خواهید برای خرید خود هزینه پرداخت کنید، صندوقدار تمام کالاها را جمع می‌کند و مبلغ کل قابل پرداخت را محاسبه می‌کند، به عنوان مثال که هزینه برابر ۱۲۳.۴۵ دلار می‌باشد. برای این سفارش یک شناسه‌ی سفارش با کد SN129803 اختصاص می‌یابد. صندوقدار روی دکمه‌ی «پرداخت» کلیک می‌کند.

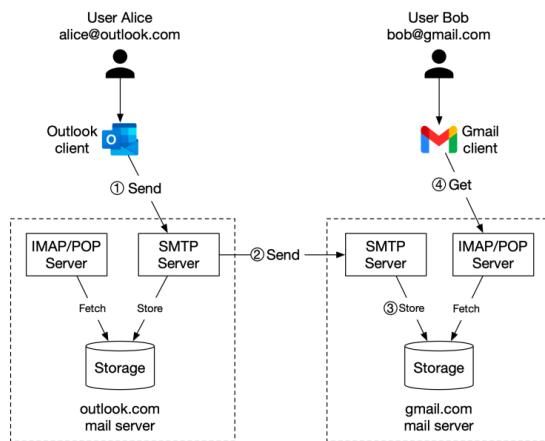
۲. رایانه‌ی صندوقدار، شناسه‌ی سفارش و مبلغ را به PSP (ارائه‌دهنده‌ی خدمات پرداخت) ارسال می‌کند.
۳. PSP این اطلاعات را در پایگاه‌داده ذخیره می‌کند و یک URL برای کد QR تولید می‌کند.
۴. سرویس درگاه پرداخت PSP، URL کد QR را می‌خواند.
۵. درگاه پرداخت، URL کد QR را به رایانه‌ی فروشگاه برمی‌گرداند.
۶. رایانه‌ی فروشگاه، URL (یا تصویر) کد QR را به صندوق پرداخت ارسال می‌کند.
۷. صندوق پرداخت، کد QR را نمایش می‌دهد.

این ۷ مرحله در کمتر از یک ثانیه تکمیل می‌شود. حالا نوبت مشتری است که با اسکن کردن کد QR از طریق کیف پول دیجیتال خود، پرداخت را انجام دهد:

۱. مشتری برنامه‌ی کیف پول دیجیتال خود را برای اسکن کردن کد QR باز می‌کند.
۲. پس از تأیید صحت مبلغ، مشتری روی دکمه‌ی «پرداخت» کلیک می‌کند.
۳. برنامه‌ی کیف پول دیجیتال، PSP را مطلع می‌کند که مشتری برای کد QR موردنظر، پرداخت را انجام داده است.
۴. درگاه پرداخت PSP، این کد QR را به عنوان پرداخت شده علامت‌گذاری می‌کند و یک پیام موفقیت را به برنامه‌ی کیف پول دیجیتال مشتری برمی‌گرداند.
۵. درگاه پرداخت PSP، فروشگاه را از پرداخت مشتری برای کد QR موردنظر مطلع می‌کند.

## سؤال مصاحبه: طراحی Gmail

یک تصویر بیش از هزار کلمه ارزش دارد. در این پست، نگاهی خواهیم انداخت که چه اتفاقی می‌افتد وقتی آلیس ایمیلی را برای باب ارسال می‌کند.



۱- آلیس به کلاینت Outlook خود وارد می‌شود، یک ایمیل می‌نویسد و دکمه "ارسال" را می‌زند. ایمیل به سرور ایمیل Outlook ارسال می‌شود. پروتکل ارتباطی بین کلاینت Outlook و سرور ایمیل، SMTP است.

۲- سرور ایمیل Outlook از DNS (در نمودار نشان داده نشده) آدرس سرور SMTP گیرنده را پرس و جو می‌کند. در این مورد، سرور SMTP گوگل است. سپس، ایمیل را به سرور ایمیل Gmail منتقل می‌کند. پروتکل ارتباطی بین سرورهای ایمیل، SMTP است.

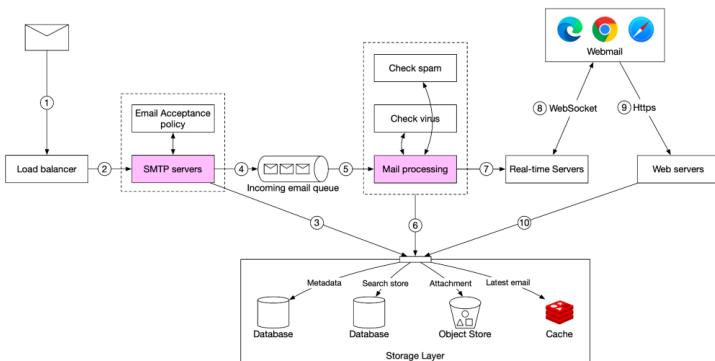
۳- سرور Gmail ایمیل را ذخیره می‌کند و آن را برای باب به عنوان گیرنده در دسترس قرار می‌دهد.

۴- کلاینت Gmail ایمیل‌های جدید را از طریق سرور IMAP/POP هنگامی که آقای باب به Gmail وارد می‌شود را دریافت می‌کند.

لطفاً توجه داشته باشید که این یک طراحی بسیار ساده شده است. امیدوارم این علاقه و کنجکاوی شما را برانگیزد.

## مسیر دریافت ایمیل

نمودار زیر جریان دریافت ایمیل را نشان می‌دهد.

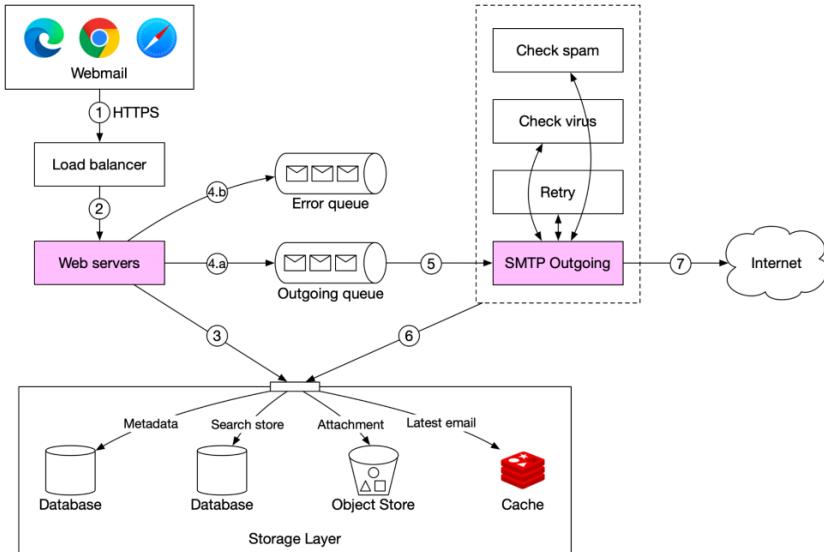


۱. ایمیل‌های ورودی در متعادل‌کننده بار SMTP وارد می‌شوند.
۲. متعادل‌کننده بار، ترافیک را بین سرورهای SMTP توزیع می‌کند. سیاست‌پذیرش ایمیل می‌تواند پیکربندی و در سطح اتصال SMTP اعمال شود. برای مثال، ایمیل‌های نامعتبر برگردانده می‌شوند تا از پردازش ایمیل غیرضروری جلوگیری شود.
۳. اگر پیوست یک ایمیل خیلی بزرگ باشد که نتوان آن را در صفحه قرارداد، می‌توانیم آن را در فضای ذخیره‌سازی پیوست‌ها در (S3) قرار دهیم.
۴. ایمیل‌ها در صفحه ایمیل‌های ورودی قرار می‌گیرند. صفحه workerها پردازش ایمیل را از سرورهای SMTP جدا می‌کند تا بتوان آنها را به طور مستقل مقیاس‌پذیر کرد. علاوه بر این، صفحه عنوان بافر عمل می‌کند در صورتی که حجم ایمیل افزایش یابد.
۵. workerها پردازش ایمیل مسئول انجام بسیاری از وظایف هستند، از جمله فیلتر کردن ایمیل‌های spam، متوقف کردن ویروس‌ها و غیره. مراحل بعدی فرض می‌کنند یک ایمیل از اعتبار سنجی عبور کرده است.

۶. ایمیل در فضای ذخیره‌سازی ایمیل، کش و فضای ذخیره‌سازی اشیاء<sup>۱</sup> ذخیره می‌شود.
۷. اگر گیرنده در حال حاضر آنلاین است، ایمیل به سرورهای بلادرنگ ارسال می‌شود.
۸. سرورهای بلادرنگ، سرورهای WebSocket هستند که به کاربران امکان می‌دهند تا ایمیل‌های جدید را در زمان بلادرنگ دریافت کنند.
۹. برای کاربران آفلاین، ایمیل‌ها در لایه ذخیره‌سازی، ذخیره می‌شوند. زمانی که یک کاربر دوباره آنلاین می‌شود، کلاینت وب می‌پرسد از طریق RESTful API به سرورهای وب متصل می‌شود.
۱۰. سرورهای وب ایمیل‌های جدید را از لایه ذخیره‌سازی می‌کشند و آنها را به کلاینت بر می‌گردانند.

## مسیر ارسال ایمیل

در این پست، به مسیر ارسال ایمیل نگاه دقیق‌تری خواهیم داشت.



۱. یک کاربر ایمیلی را در وب‌میل می‌نویسد و دکمه «ارسال» را می‌زند. درخواست به توزیع‌کننده بار ارسال می‌شود.
۲. توزیع‌کننده بار اطمینان حاصل می‌کند که درخواست‌های ارسال ایمیل از محدودیت نرخ تجاوز نمی‌کند و ترافیک را به وب سرورها هدایت می‌کند.
۳. وب سرورها مسئول موارد زیر هستند:
  - ۳/۱ اعتبارسنجی اولیه ایمیل. هر ایمیل وارد شده در برابر قواعد از پیش تعیین شده مانند محدودیت اندازه ایمیل بررسی می‌شود.
  - ۳/۲ بررسی اینکه آیا دامنه آدرس ایمیل گیرنده با فرستنده یکسان است یا خیر. اگر یکسان باشد، داده‌های ایمیل مستقیماً در ذخیره‌سازی، کش و object store وارد

می‌شوند. گیرنده می‌تواند ایمیل را مستقیماً از طریق RESTful API دریافت کند.  
نیازی به رفتن به مرحله ۴ نیست.

#### ۴. صفحه پیام‌ها

۴/۱. اگر اعتبارسنجی اولیه ایمیل موفقیت‌آمیز باشد، داده‌های ایمیل به صفحه خروجی منتقل می‌شوند.

۴/۲. اگر اعتبارسنجی اولیه ایمیل ناموفق باشد، ایمیل در صفحه خطای قرار می‌گیرد.

۵. SMTP worker‌های خروجی رویدادها را از صفحه خروجی می‌کشند و اطمینان حاصل می‌کنند که ایمیل‌ها عاری از هرزنامه و ویروس هستند.

۶. ایمیل خروجی در "پوشه ارسال شده" در لایه ذخیره‌سازی ذخیره می‌شود.

۷. worker‌های مربوط به SMTP خروجی ایمیل را به سرور ایمیل گیرنده ارسال می‌کنند.  
هر پیام در صفحه خروجی شامل تمام متادیتای مورد نیاز برای ایجاد یک ایمیل است. یک صفحه پیام توزیع شده یک جزء حیاتی است که پردازش ایمیل غیرهمگام را امکان‌پذیر می‌کند. با جدایکردن worker‌های SMTP خروجی از وب سرورها، می‌توانیم worker‌های SMTP خروجی را به طور مستقل مقیاس‌پذیر کنیم.

ما اندازه صفحه خروجی را از نزدیک نظارت می‌کنیم. اگر تعداد زیادی ایمیل در صفحه گیرکرده

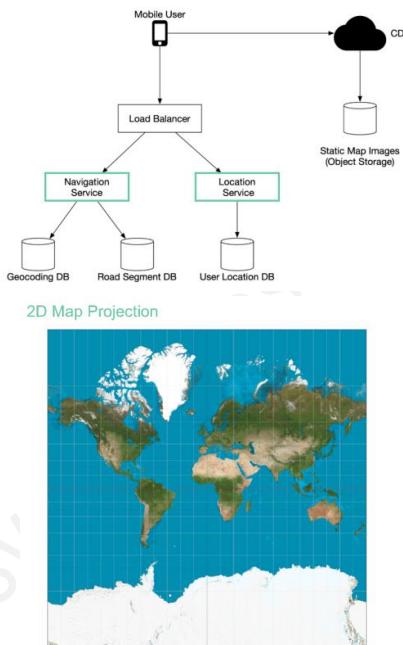
باشند، باید علت مشکل را تجزیه و تحلیل کنیم. درنتیجه احتمالات زیر وجود دارد:

- سرور ایمیل گیرنده در دسترس نیست. در این صورت، باید ارسال ایمیل را در زمان دیگری دوباره امتحان کنیم. استراتژی Exponential backoff ممکن است یک استراتژی تلاش مجدد خوب باشد.

- مصرف کننده‌های کافی برای ارسال ایمیل‌ها وجود ندارد. در این صورت، ممکن است نیاز به مصرف کننده‌های بیشتر برای کاهش زمان پردازش داشته باشیم.

## سؤال مصاحبه: طراحی Google Maps

گوگل پروژه Google Maps را در سال ۲۰۰۵ آغاز کرد. تا ماه مارس ۲۰۲۱ نرم افزار Google Maps یک میلیارد کاربر فعال روزانه داشت و ۹۹ درصد از جهان را در ۲۰۰ کشور پوشش می‌داد. اگرچه Google Maps یک سیستم بسیار پیچیده است، اما می‌توانیم آن را به ۳ مؤلفه سطح بالا تقسیم کنیم. در این متن بیایید نگاهی به چگونگی طراحی یک Google Maps ساده‌شده بیندازیم.



### سرویس موقعیت مکانی

سرویس موقعیت مکانی مسئول ثبت به روزرسانی موقعیت مکانی کاربر است. کلاینت‌های Google Maps هر چند ثانیه یکبار موقعیت مکانی را به روزرسانی می‌کنند. داده‌های موقعیت مکانی کاربر در موارد زیر مورد استفاده قرار می‌گیرند:

- تشخیص جاده‌های جدید و جاده‌های تازه بسته شده یا جاده‌های تغییر داده شده.
- بهبود دقت نقشه با گذشت زمان.
- استفاده به عنوان ورودی برای تحلیل داده‌های ترافیک به صورت بلادرنگ و زنده.

### رِندر کردن نقشه<sup>۱</sup>

نقشه جهان به یک تصویر نقشه ۲ بعدی بزرگ تصویر شده است. این نقشه به قطعات کوچک تصویری به نام "کاشی" (tile) تقسیم شده است (به زیر نگاه کنید). کاشی‌ها ثابت هستند و اغلب تغییر نمی‌کنند. یک راه کارآمد برای سرویس دادن فایل‌های کاشی ثابت، استفاده از CDN پشتیبانی شده توسط ذخیره‌سازی ابری مانند S3 است. کاربران می‌توانند tile‌های موردنیاز را از CDN نزدیک برای ترکیب یک نقشه بارگیری کنند.

اما اگر کاربر روی کلاینت در حال بزرگنمایی و حرکت در نقطه دید نقشه برای کاوش محیط اطراف خود باشد، چه اتفاقی می‌افتد؟

یک راه کارآمد این است که از قبل بلوک‌های نقشه با سطوح بزرگنمایی مختلف را محاسبه کنیم و در صورت نیاز تصاویر را بارگیری کنیم.

### سرویس مسیریابی<sup>۲</sup>

این مؤلفه مسئول یافتن یک مسیر نسبتاً سریع از نقطه A به نقطه B است. برای کمک به محاسبه مسیر، از دو سرویس زیر استفاده می‌کند:

۱. سرویس گُدینگ جغرافیایی (Geocoding): آدرس داده شده را به یک جفت عرض و طول جغرافیایی تبدیل می‌کند.

۲. سرویس برنامه‌ریز مسیردهی (Route Planner): این سرویس سه کار مختلف را به ترتیب زیر انجام می‌دهد:

- محاسبه K-برتر که بیانگر کوتاه‌ترین مسیر بین A و B است.
- محاسبه تخمین زمان برای هر مسیر بر اساس ترافیک فعلی و داده‌هایی که در گذشته تحلیل شده است.
- رتبه‌بندی مسیرها بر اساس پیش‌بینی زمان و فیلترهای کاربر. به عنوان مثال، کاربر نمی‌خواهد از عوارضی عبور کند.

## رندر کردن نقشه Google Maps

در این متن به رندر کردن نقشه (Map Rendering) نگاهی خواهیم انداخت. کاشی‌های<sup>۱</sup> از پیش محاسبه شده یکی از مفاهیم بنیادی در رندر کردن نقشه، کاشی‌کاری (tiling) است. به جای رندر کردن کل نقشه به صورت یک تصویر بزرگ سفارشی، جهان به کاشی‌های کوچک‌تر تقسیم می‌شود. کلاینت فقط کاشی‌های مرتبط با منطقه‌ای را که کاربر در آن قرار دارد دانلود می‌کند و آنها را مانند یک کاشی‌کاری برای نمایش به هم می‌چسباند. کاشی‌ها در سطوح بزرگ‌نمایی مختلف از پیش محاسبه می‌شوند. Google Maps از ۲۱ سطح بزرگ‌نمایی استفاده می‌کند.

به عنوان مثال، در سطح بزرگ‌نمایی ۰ کل نقشه توسط یک کاشی تک با اندازه  $256 \times 256$  پیکسل نمایش داده می‌شود. سپس در سطح بزرگ‌نمایی ۱، تعداد کاشی‌های نقشه در هر دو جهت شمال-جنوب و شرق-غرب دو برابر می‌شود، در حالی‌که هر کاشی در اندازه  $256 \times 256$  پیکسل باقی می‌ماند. بنابراین در سطح بزرگ‌نمایی ۱ حدود ۴ کاشی داریم و کل تصویر در سطح بزرگ‌نمایی ۱ اندازه  $512 \times 512$  پیکسل دارد. با هر افزایش سطح، کل مجموعه کاشی‌ها ۴ برابر تعداد پیکسل‌های سطح قبلی را دارد. افزایش تعداد پیکسل‌ها، سطح جزئیات بیشتری را در اختیار کاربر قرار می‌دهد. این امر به کلاینت امکان می‌دهد نقشه را در بهترین سطوح جزئیات بسته به سطح بزرگ‌نمایی کلاینت رندر کند، بدون اینکه پهنای باند زیادی برای دانلود کاشی‌هایی با جزئیات بیش از حد مصرف شود. این موضوع به ویژه هنگام بارگیری تصاویر از کلاینت‌های موبایل اهمیت دارد.

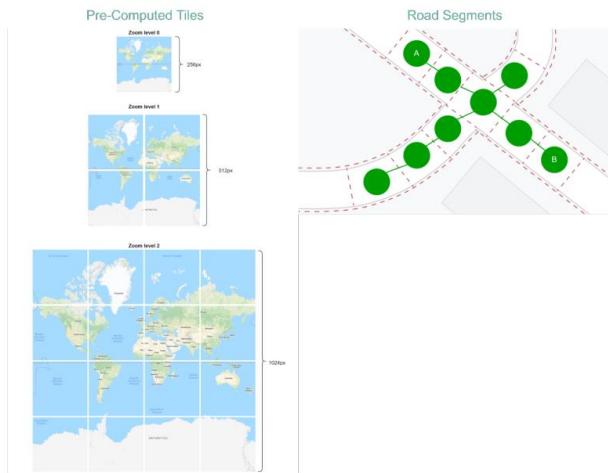
## بخش‌های جاده و خیابان

اکنون که نقشه‌های عظیم را به کاشی‌های کوچک‌تر تبدیل کرده‌ایم، نیاز داریم یک ساختار داده برای جاده‌ها نیز تعریف کنیم. ما جهان جاده‌ها را به بلوک‌های کوچک تقسیم می‌کنیم.

به این بلوک‌ها، بخش‌های جاده می‌گوییم. هر بخش جاده شامل چندین جاده، تقاطع‌ها و سایر متادیتاهای<sup>۱</sup> دیگر است.

ما بخش‌های نزدیک به هم را در آبر بخش‌ها<sup>۲</sup> گروه‌بندی می‌کنیم. این فرایند می‌تواند به طور مکرر برای رسیدن به سطح پوشش موردنیاز اعمال شود. سپس بخش‌های جاده را به یک ساختار داده تبدیل می‌کنیم که الگوریتم‌های ناوبری می‌توانند از آن استفاده کنند. رویکرد معمول این است که نقشه را به یک گراف تبدیل کنیم، جایی که گره‌ها بخش‌های جاده هستند و دو گره اگر بخش‌های جاده مربوطه همسایه‌های قابل دسترس باشند، به هم متصل می‌شوند. به این ترتیب، یافتن مسیر بین دو مکان به یک مسئله کوتاه‌ترین مسیر تبدیل می‌شود که می‌توانیم از الگوریتم‌های Dijkstra یا  $A^*$  در آن بهره ببریم.

| Google Maps - Map Rendering



metadata<sup>۱</sup>

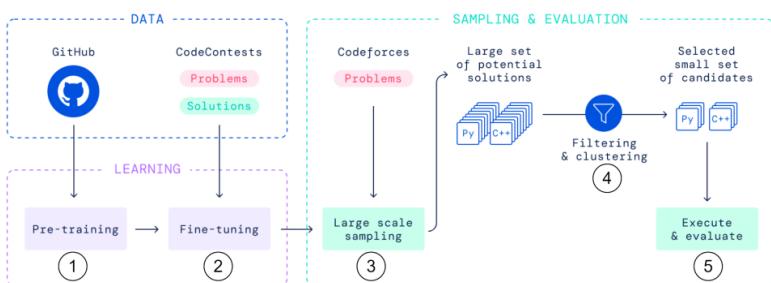
super segments<sup>۲</sup>

## موتور کدنویسی هوش مصنوعی

شرکت deepMind می‌گوید که موتور کدنویسی جدید هوش مصنوعی آن (آلفا کد) به اندازه یک برنامه‌نویس متوسط خوب است.

ربات هوش مصنوعی در ۱۰ مسابقه برنامه‌نویسی Codeforces شرکت کرد و رتبه ۵۴.۳٪ را کسب کرد. این بدان معناست که نمره آن از نیمی از شرکت‌کنندگان انسانی بالاتر بود. اگر به نمره آن در ۶ ماه گذشته نگاه کنیم، آلفاکد در رتبه ۲۸٪ قرار می‌گیرد.

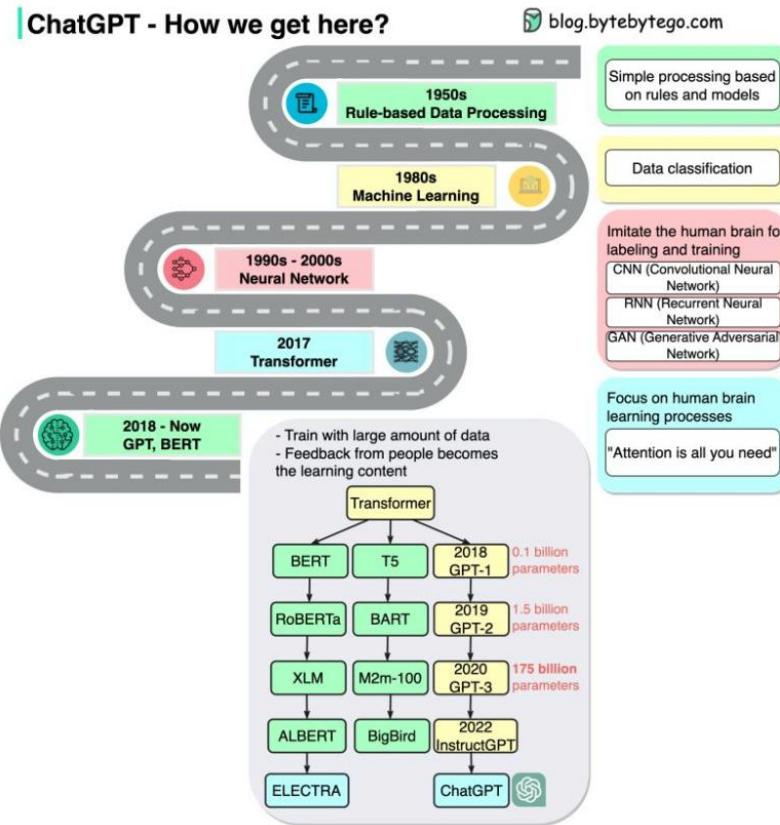
نمودار زیر نحوه کار ربات هوش مصنوعی را توضیح می‌دهد:



۱. مدل‌های ترانسفورمر را روی کدهای گیت‌هاب پیش آموزش دهید.
۲. مدل‌ها را روی مجموعه‌داده نسبتاً کوچک برنامه‌نویسی رقابتی تنظیم کنید.
۳. در زمان ارزیابی، مقدار زیادی از راه حل‌ها را برای هر مسئله ایجاد کنید.
۴. راه حل‌ها را فیلتر، خوشبندی و رتبه‌بندی مجدد کنید تا به یک مجموعه کوچک از برنامه‌های کاندیدا (حداکثر ۱۰ مورد) برسید، سپس برای ارزیابی‌های بیشتر ارسال کنید.
۵. برنامه‌های کاندیدا را در برابر موارد آزمایشی اجرا کنید، عملکرد را ارزیابی کنید و بهترین گزینه را انتخاب کنید.

## جدول زمانی - ChatGPT

اینجا یک تصویر بالرزش هزار کلمه است. به نظر می‌رسد ChatGPT از آسمان به زمین آمده است. ما نمی‌دانستیم که این بنا بر اساس دهه‌ها تحقیق ساخته شده است. نمودار زیر نشان می‌دهد که چگونه به اینجا رسیده‌ایم.



• ۱۹۵۰ دهه

در این مرحله، مردم هنوز از مدل‌های ابتدایی مبتنی بر قوانین استفاده می‌کردند. – دهه ۱۹۸۰ از دهه ۱۹۸۰، یادگیری ماشین شروع به کار کرد و برای طبقه‌بندی استفاده شد. این آموزش بر روی محدوده کمی از داده‌ها انجام شد.

• ۲۰۰۰ دهه ۱۹۹۰ –

از دهه ۱۹۹۰، شبکه‌های عصبی شروع به تقلید از مغز انسان برای برچسب‌زن و آموزش کردن. به طور کلی ۳ نوع وجود دارد:

- CNN (شبکه عصبی کانولوشنال): اغلب در کارهای مربوط به بصری استفاده می‌شود.
  - RNN (شبکه عصبی مکرر): مفید در وظایف زبان طبیعی
  - GAN (شبکه متخاصل مولد): متشکل از دو شبکه (Discriminative و Generative).
- این یک مدل generative است که می‌تواند تصاویر بدیع و شبیه به هم تولید کند.

۲۰۱۷ •

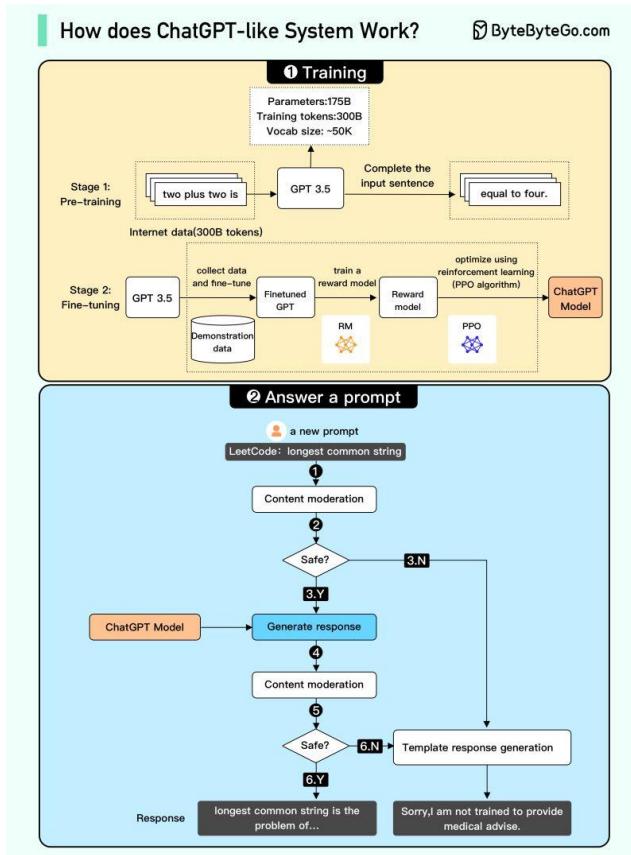
«Attention» تنها چیزی است که نیاز دارید» در واقع Attention نشان‌دهنده پایه و اساس هوش مصنوعی مولد است. مدل transformer زمان آموزش را با استفاده از روش‌های موازی‌سازی بسیار کوتاه می‌کند.

۲۰۱۸ • - اکنون

در این مرحله با توجه به پیشرفت عمده مدل transformer ، شاهد آموزش مدل‌های مختلف بر روی حجم عظیمی از داده‌ها هستیم. نمایش انسانی به محتوای یادگیری مدل تبدیل می‌شود. بسیاری از مدل‌های هوش مصنوعی را دیده‌ایم که می‌توانند مقالات، اخبار، استناد فنی و حتی کد بنویسن. این موارد نیز ارزش تجاری زیادی دارد و یک طوفان جهانی را به راه می‌اندازد.

## چگونه سیستم‌های شبیه به ChatGPT کار می‌کنند؟

از آنجایی که OpenAI تمام جزئیات را ارائه نکرده است، ممکن است برخی از بخش‌های نمودار نادرست باشند.



ما سعی کرده‌ایم در نمودار بالا نحوه عملکرد این سیستم‌ها را توضیح دهیم. این فرایند را می‌توان به دو بخش تقسیم کرد:

۱. برای آموزش یک مدل ChatGPT، دو مرحله وجود دارد:

• پیش - آموزش (Pre-training) •

در این مرحله، یک مدل GPT (ترانسفورماتور فقط رمزگشا<sup>۱</sup>) را روی حجم عظیمی از داده‌های اینترنت آموخته می‌دهیم. هدف این است که مدلی را آموخته دهیم که بتواند کلمات بعدی را با توجه به یک جمله به روشی دستور زبان درست و از نظر معنایی مشابه با داده‌های اینترنت را پیش‌بینی کند. پس از مرحله پیش آموخته این مدل می‌تواند جملات داده شده را تکمیل کند، اما قادر به پاسخ‌دادن به سؤالات نیست.

### • Fine-tuning •

این مرحله یک فرایند ۳ مرحله‌ای است که مدل پیش آموخته دیده را به یک مدل پرسش‌پاسخ ChatGPT تبدیل می‌کند:

- ۱) جمع‌آوری داده‌های آموخته‌شی (سؤالات و پاسخ‌ها) و Fine-tune کردن دقیق مدل پیش آموخته دیده بر اساس این داده‌ها. مدل یک سؤال را به عنوان ورودی دریافت می‌کند و یاد می‌گیرد پاسخی مشابه با داده‌های آموخته‌شی تولید کند.
- ۲) جمع‌آوری داده‌های بیشتر (سؤال، چندین پاسخ) و آموخته یک مدل پاداش برای رتبه‌بندی این پاسخ‌ها از مرتبط‌ترین تا کم ارتباطی ترین.
- ۳) استفاده از یادگیری تقویتی<sup>۲</sup> (بهینه‌سازی PPO)<sup>۳</sup> برای Fine-tune دقیق مدل به‌گونه‌ای که پاسخ‌های مدل دقیق‌تر شوند.

### ۲. پاسخ به یک دستور (Prompt):

<sup>۱</sup> decoder-only transformer

<sup>۲</sup> reinforcement learning

<sup>۳</sup> PPO optimization

**گام ۱:** کاربر کل سؤال را وارد می‌کند، به عنوان مثال: «نحوه عملکرد یک الگوریتم طبقه‌بندی را توضیح دهید».

**گام ۲:** سؤال به بخش تغییر محتوا<sup>۱</sup> فرستاده می‌شود. این بخش تضمین می‌کند که سؤال، دستورالعمل‌های ایمنی را نقض نمی‌کند و سؤالات نامناسب را فیلتر می‌کند.

**گام‌های ۳-۴:** اگر ورودی از تغییر محتوا عبور کند، به مدل chatGPT فرستاده می‌شود. اگر ورودی از تغییر محتوا عبور نکند؛ بنابراین مستقیماً به قسمتی به نام template response generation می‌رود.

**گام‌های ۵-۶:** پس از اینکه مدل پاسخ را تولید کرد، دوباره به بخش تغییر محتوا فرستاده می‌شود. این کار تضمین می‌کند که پاسخ تولید شده ایمن، بی‌خطر، بدون سوگیری و غیره باشد.

**گام ۷:** اگر ورودی از تغییر محتوا عبور کند درنتیجه به کاربر نمایش داده می‌شود. اگر ورودی از تغییر محتوا عبور نکند، به تولید پاسخ قالبی می‌رود و یک پاسخ قالبی به کاربر نشان می‌دهد.

## مدیریت یک خرابی گسترده

این یک داستان واقعی درباره مدیریت یک خرابی گسترده است که توسط مهندسان ارشد در Discord Sahn Lam نوشته شده است. حدود ۱۰ سال پیش، شاهد بزرگترین اشکالات رابط کاربری در طول دوران حرفه‌ای خود بودم. ساعت ۹ شب یک جمیعه بود. من در تیمی مسئول یکی از بزرگترین بازی‌های اجتماعی آن زمان بودم. این بازی حدود ۳۰ میلیون کاربر فعال روزانه داشت. اتفاقاً قبل از خاموش کردن رایانه برای شب، به داشبورد عملیاتی نگاهی انداختم. هر خط در داشبورد برابر صفر بود. در همان لحظه، تلفنی از رئیسم دریافت کردم. او گفت که بازی کاملاً از کارافتاده است. پس در یک حالت اورژانسی کامل بودیم. همه چیز متوقف شده بود. هر یک از نمونه‌های AWS متوقف شده بود. نمونه‌های HA proxy، سرورهای وب PHP، پایگاه‌های داده MySQL، گره‌های Memcache، همه چیز. ۵۰ نفر ۱۰ ساعت طول کشید تا همه چیز را دوباره راهاندازی کنند. این کار شگفت‌انگیزی بود. این خود داستانی برای روز دیگری است. ما از نرم‌افزار مدیریت ابر استفاده می‌کردیم تا پیاده‌سازی AWS را مدیریت کنیم. این قبل از اینکه زیرساخت به عنوان کد یک مسئله باشد بود. این در ابتدای محاسبات ابری بود و ما به قدری بزرگ بودیم که AWS باید از قبل هشدار می‌داد قبل از اینکه ما مقیاس‌پذیری را افزایش دهیم. چه اتفاقی افتاده بود؟ ارائه‌دهنده نرم‌افزار مدیریت ابر آن هفته یک اشکال را در جریان تأیید خود وارد کرده بود. هنگام خاتمه‌دادن به یک زیرمجموعه از گره‌ها در رابط کاربری، به درستی در جعبه تأیید فهرست گره‌های مورد خاتمه را نشان می‌داد، اما در زیرلایه همه چیز را خاتمه داد. کمی قبل از ساعت ۹ شب در آن شب سرنوشت‌ساز، یکی از SRE‌های ما درخواست روتین ما را برای خاتمه دادن به یک پول Memcache استفاده نشده انجام داد. تنها می‌توانم تصور کنم چه وحشتی و چه مکالمه‌ای در پی داشت. چه نوع ساختار کدی می‌توانست به این اشکال فاجعه‌بار اجازه سر باز کردن را بدهد؟ ما فقط می‌توانستیم حدس بزنیم. هرگز توضیح کاملی دریافت نکردیم. در طول دوران حرفه‌ای خود با چه برخی از بزرگ‌ترین اشکالات نرم‌افزاری رو برو شده‌اید؟ ما فقط می‌توانستیم حدس بزنیم که چه ساختار کدی به این اشکال فاجعه‌بار اجازه ظهور داده است. هرگز توضیح کامل آن را دریافت نکردیم.

## چگونه شناسه‌های منحصر به فرد تولید کنید؟

در این پست، ما به بررسی نیازمندی‌های رایج برای شناسه‌هایی که در شبکه‌های اجتماعی مانند فیسبوک، توییتر و لینکدین استفاده می‌شوند، خواهیم پرداخت.

نیازمندی‌ها:

- منحصر به فرد جهانی<sup>۱</sup>
- تقریباً مرتب شده بر اساس زمان
- فقط مقادیر عددی
- ۶۴ بیتی
- بسیار مقیاس‌پذیر، کم تأخیر

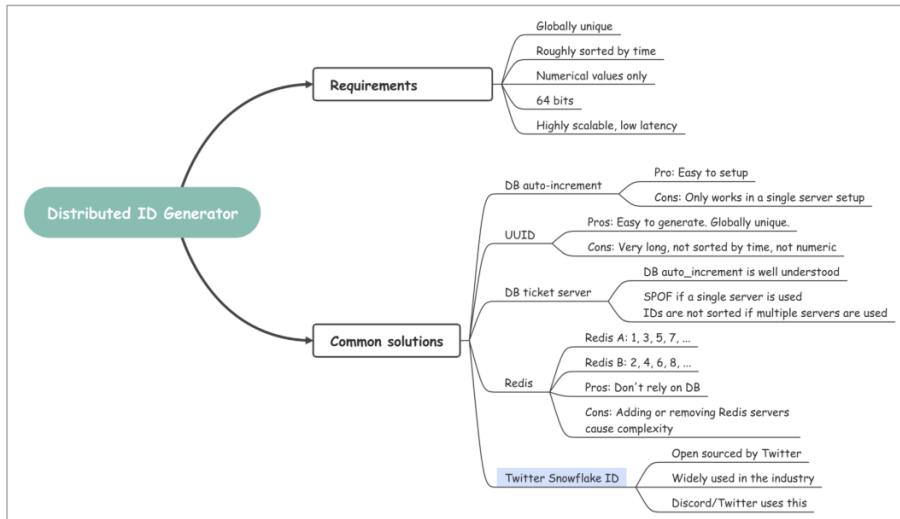
جزئیات پیاده‌سازی الگوریتم‌ها به صورت آنلاین یافت می‌شوند، بنابراین ما در اینجا به جزئیات نخواهیم پرداخت.

### Unique ID Generator

 blog.bytebytogo.com

Reasons	
Globally unique	If IDs are not globally unique, there could be collisions.
Roughly sorted by time	So user IDs, post IDs can be sorted by time without fetching additional info
Numerical values only	Naturally sortable by time
64 bits	$2^{32} = \sim 4 \text{ billion}$ $\rightarrow$ not enough IDs. $2^{64}$ is big enough $2^{128}$ wastes space and is too long
Highly scalable, low latency	Ability to generate a lot of IDs per second in low latency fashion is critical.

<sup>۱</sup>Globally unique



توضیح: این مقاله بر اساس سخنرانی فنی ارائه شده توسط توییتر در سال ۲۰۱۳ (https://bit.ly/3vNfjRp) است.

## چگونه از crawl آدرس‌های تکراری در ابعاد گوگل جلوگیری کنیم؟

گزینه ۱: استفاده از ساختار داده Set برای بررسی اینکه آیا یک URL قبلاً وجود دارد یا خیر. Set سریع است، اما از نظر فضایی کارآمد نیست.

گزینه ۲: ذخیره URL‌ها در یک پایگاه‌داده و بررسی اینکه آیا یک URL جدید در پایگاه‌داده وجود دارد. این روش ممکن است به شیوه مناسب کار کند؛ اما بار روی پایگاه‌داده بسیار بالا خواهد بود.

گزینه ۳: فیلتر بلوم.<sup>۱</sup> این گزینه ترجیح داده می‌شود. فیلتر بلوم توسط برتون هاوارد بلوم در سال ۱۹۷۰ پیشنهاد شد. این یک ساختار داده احتمالی است که برای آزمایش اینکه آیا یک عنصر عضوی از یک مجموعه است استفاده می‌شود.

- خطأ: عنصر قطعاً در مجموعه نیست.
- درست: عنصر احتمالاً در مجموعه است.

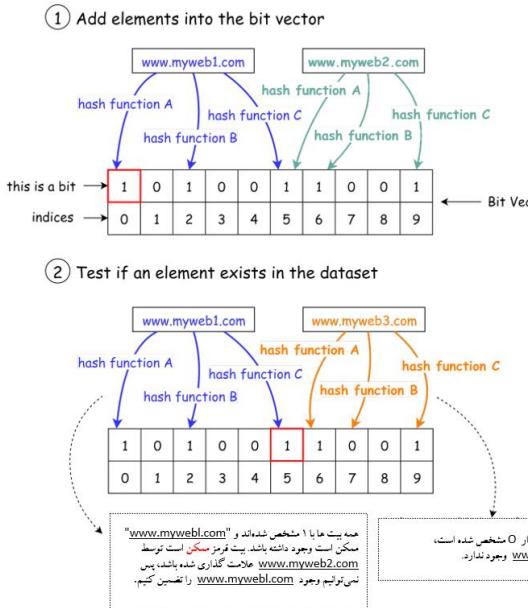
تطابق‌های مثبت کاذب<sup>۲</sup> امکان‌پذیر است، اما منفی‌های کاذب<sup>۳</sup> ممکن نیست. دیاگرام زیر نشان می‌دهد که فیلتر بلوم چگونه کار می‌کند. ساختار داده اصلی برای فیلتر بلوم از نوع بیت برداری<sup>۴</sup> است و هر بیت یک مقدار هش را نمایندگی می‌کند.

Bloom filter<sup>۱</sup>

False-positive<sup>۲</sup>

false negatives<sup>۳</sup>

Bit Vector<sup>۴</sup>



مرحله ۱: برای افزودن یک عنصر به فیلتر بلوم، آن را به سه تابع هش مختلف (A، B و C) تغذیه می‌کنیم و بیت‌ها را در موقعیت‌های نتیجه تنظیم می‌کنیم. توجه داشته باشید که هر دو علامت‌گذاری می‌کنند. درنتیجه امکان وقوع مثبت‌های کاذب ممکن است؛ زیرا یک بیت ممکن است توسط عنصر دیگری تنظیم شود.

مرحله ۲: هنگام آزمایش وجود یک رشته URL، همان توابع هش A، B و C به رشتہ URL اعمال می‌شوند. اگر هر سه بیت برابر ۱ باشند، آنگاه URL ممکن است در مجموعه‌داده‌ها وجود داشته باشد؛ اگر هر یک از بیت‌ها برابر ۰ باشد، آنگاه قطعاً URL در مجموعه‌داده‌ها وجود ندارد. انتخاب تابع هش مهم است. آنها باید به طور یکنواخت توزیع شده و سریع باشند. به عنوان مثال، Apache Spark و RedisBloom از روش<sup>۱</sup> Murmur استفاده می‌کنند و InfluxDB از xxhash استفاده می‌کند.

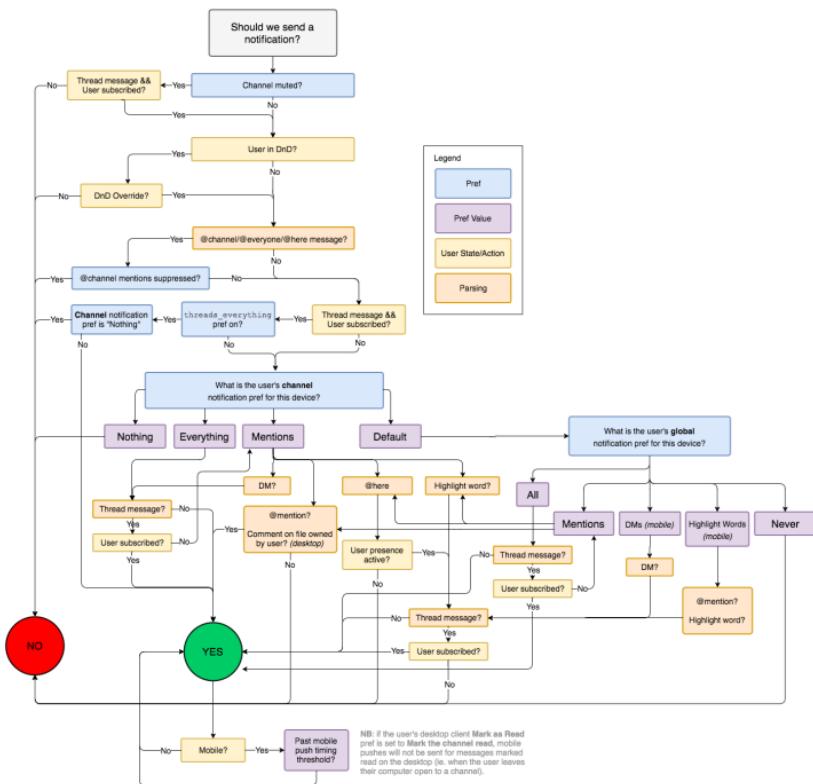
۱ index

۲ MurmurHash یک تابع هش غیر رمزنگاری مناسب برای جستجوی کلی مبتنی بر هش است.

## نحوه ارسال نوتیفیکیشن در slack

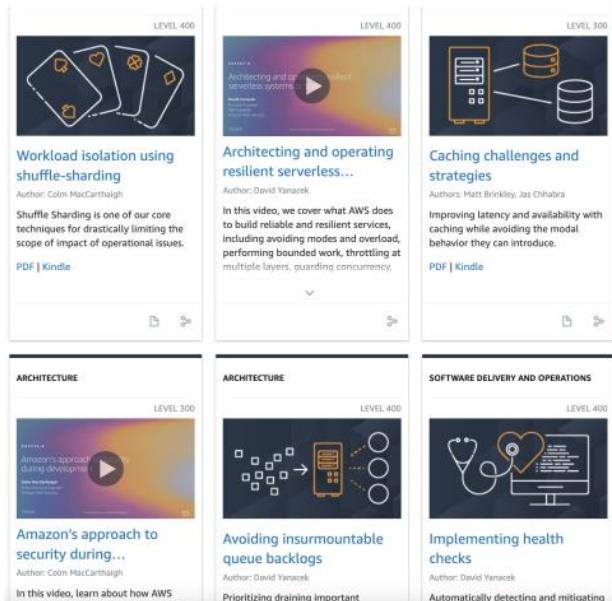
این مثال خوبی است که نشان می‌دهد چرا توسعه یک ویژگی ساده ممکن است بسیار طولانی‌تر از آنچه بسیاری از مردم فکر می‌کنند، طول بکشد. زمانی که ما طراحی خوبی داریم، کاربران ممکن است پیچیدگی‌ها را متوجه نشوند؛ زیرا به نظر می‌رسد که ویژگی به طور موردنظر کار می‌کند.

منبع تصویر: <https://slack.engineering/reducing-slacks-memory-footprint>



## چگونه آمازون نرم افزارهای خود را می سازد و اداره می کند؟

در سال ۲۰۱۹، آمازون کتابخانه سازندگان آمازون (The Amazon Builders' Library) را منتشر کرد. این کتابخانه شامل مقالات مبتنی بر معماری است که توضیح می دهد چگونه معماران آمازون، انتشارات و فناوری های آنها را اداره می کنند.



در اینجا برخی از موارد مورد بحث آمده است:

- ایمن سازی تلاش های مجدد با API های بی اثر<sup>۱</sup>
- تایم اوت ها<sup>۲</sup>، تلاش های مجدد<sup>۳</sup> و عقب گرد<sup>۴</sup> با جیتر<sup>۵</sup>

<sup>۱</sup> idempotent APIs

<sup>۲</sup> Timeouts

<sup>۳</sup> retries

<sup>۴</sup> Backoff – این روش زمان بین تلاش های مجدد بعدی را افزایش می دهد که بار را در backend یکنواخت و ثابت نگه می دارد.

<sup>۵</sup> Jitter – جیتر به تغییر در تاخیر زمانی بین بسته های داده در یک شبکه اشاره دارد که معمولاً با واحد میلی ثانیه اندازه گیری می شود. این یک معیار عملکرد ضروری در شبکه است که بر کیفیت برنامه های

- فراتر قوانین پنج ۹s: درس‌هایی از بالاترین سطوح دسترسی<sup>۱</sup> داده‌ها
- چالش‌ها و استراتژی‌های Caching
- تضمین امنیت بازگشت<sup>۲</sup> در هنگام استقرار<sup>۳</sup>
- سریع‌تر شدن با تحویل مداوم<sup>۴</sup>
- چالش‌ها با سیستم‌های توزیع شده
- رویکرد آمازون به استقرار با دردسترس بودن بالا

مانند بازی‌های ویدیوئی و تماس‌های تحت وب تأثیر می‌گذارد. جیتر ناشی از ازدحام شبکه، تغییر مسیر یا صفات نامناسب است.

highest available<sup>۱</sup>

rollback safety<sup>۲</sup>

deployments<sup>۳</sup>

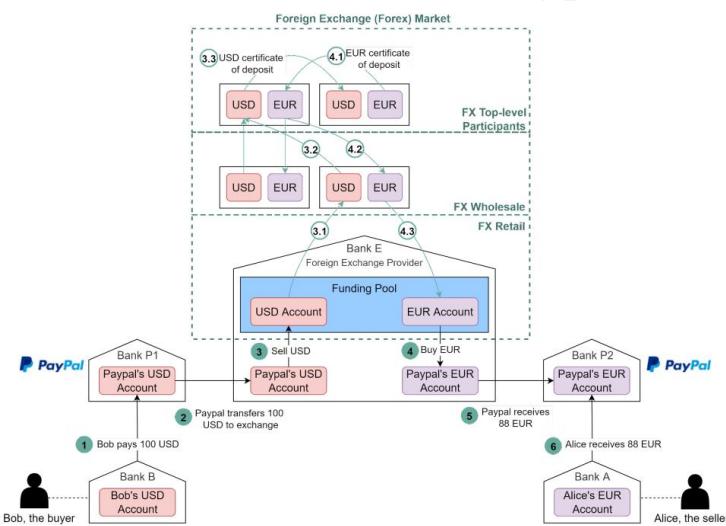
continuous delivery<sup>۴</sup>

# بررسی طراحی سیستم‌های مالی و بانکی

## مبادله خارجی در پرداخت

آیا تابه‌حال فکر کرده‌اید که چه اتفاقی در پشت صحنه رخ می‌دهد وقتی که شما با دلار آمریکا آنلاین پرداخت می‌کنید و فروشنده از اروپا یورو دریافت می‌کند؟ این فرایند مبادله خارجی نامیده می‌شود.

فرض کنید آقای اسمیت (خریدار) باید ۱۰۰ دلار آمریکا به آلیس (فروشنده) پرداخت کند و آلیس فقط می‌تواند یورو دریافت کند. نمودار زیر این فرایند را نشان می‌دهد.



- اسمیت ۱۰۰ دلار آمریکا را از طریق یک ارائه دهنده پرداخت شخص ثالث ارسال می‌کند. در مثال ما، این پرداخت از طریق PayPal است. پول از حساب بانکی اسمیت (بانک B) به حساب PayPal در بانک P1 منتقل می‌شود.
- PayPal باید دلار آمریکا را به یورو تبدیل کند. از ارائه‌دهنده مبادله خارجی (بانک E) استفاده می‌کند. پی‌پال ۱۰۰ دلار آمریکا را به حساب USD خود در بانک E ارسال می‌کند.

۳. ۱۰۰ دلار آمریکا به پول مخزن بانک E فروخته می‌شود.
۴. مخزن تأمین مالی بانک E در ازای ۱۰۰ دلار آمریکا، ۸۸ یورو ارائه می‌دهد. پول به حساب یورو Paypal در بانک E واریز می‌شود.
۵. حساب یورو Paypal در بانک P2، ۸۸ یورو دریافت می‌کند.
۶. ۸۸ یورو به حساب یورو آلیس در بانک A پرداخت می‌شود.
- اکنون نگاهی دقیق‌تر به بازار مبادلات خارجی (فارکس) بیندازیم. این بازار دارای ۳ لایه است:
- بازار خردفروشی. مخزن‌های تأمین مالی بخشی از بازار خردفروشی هستند. برای بهبود کارایی، Paypal معمولاً مقداری ارز خارجی را به صورت پیش‌پرداخت خریداری می‌کند.
  - بازار عمدفروشی. کسب‌وکار عمدفروشی از بانک‌های سرمایه‌گذاری، بانک‌های تجاری و ارائه‌دهنگان مبادله خارجی تشکیل شده است. معمولاً سفارش‌های انباسته شده از بازار خردفروشی را مدیریت می‌کند.
  - شرکت‌کنندگان سطح بالا. آنها بانک‌های تجاری چندملیتی هستند که تعداد زیادی گواهی سپرده از کشورهای مختلف را در اختیار دارند. آنها این گواهینامه‌ها را برای تجارت مبادله خارجی مبادله می‌کنند.
- وقتی مخزن تأمین مالی بانک E به یورو بیشتری نیاز دارد، به بازار عمدفروشی می‌رود تا دلار آمریکا را بفروشد و یورو بخرد. وقتی بازار عمدفروشی سفارش‌های کافی را انباسته کرد، آنگاه به شرکت‌کنندگان سطح بالا می‌رود. مراحل ۳.۳-۳.۱ و ۴.۳-۴.۱ نحوه کار را توضیح می‌دهند.

## مطابقت سفارش‌های خریدوفروش

سهام بالا و پایین می‌رود. آیا می‌دانید چه ساختاری از داده‌ها برای مطابقت کارا با سفارش‌های خریدوفروش به کار گرفته می‌شود؟

بورس‌های اوراق بهادر از ساختاری از داده‌ها به نام «سفارش کتاب‌ها<sup>۱</sup>» استفاده می‌کنند. یک سفارش کتاب، فهرستی الکترونیکی از سفارش‌های خریدوفروش است که بر اساس سطوح قیمتی سازماندهی شده است. دارای یک کتاب خرید و یک کتاب فروش است که هر طرف کتاب حاوی تعدادی سطوح قیمتی است و هر سطح قیمتی حاوی فهرستی از سفارش‌ها به صورت first in first out است.

		Price	Quantity	
	depth of ask	100.13	100   200	← price levels
		100.12	600   900	
		100.11	900   700   400	
Sell book	best ask	100.10	200   400   1100   100	
Buy book	best bid	100.08	500   600   900	
		100.07	100   700	
		100.06	1100   400   300   200	
	depth of bid	100.05	500   100	

$$\text{Buy 2700 shares: } 2700 - 200 - 400 - 1100 - 100 - 900 = 0$$

این تصویر مثالی از سطوح قیمتی و مقدار صف‌بندی شده در هر سطح قیمتی است. بنابراین، وقتی در نمودار یک سفارش بازار برای خرید ۲۷۰۰ سهم قرار می‌دهید چه اتفاقی می‌افتد؟

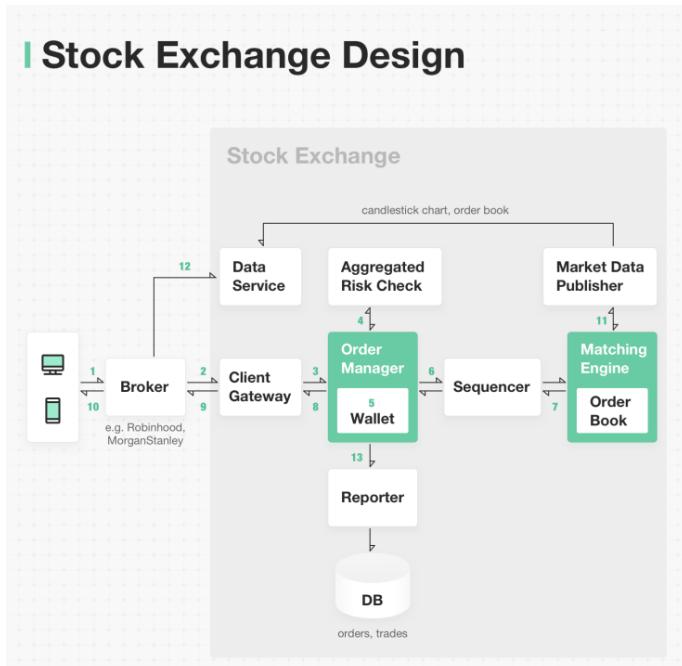
- سفارش خرید با همه سفارش‌های فروش در قیمت ۱۰۰.۱۰ مطابقت دارد و با اولین سفارش در قیمت ۱۰۰.۱۱ (به رنگ صورتی روشن نشان داده شده) مطابقت دارد.

- اکنون به دلیل سفارش خرید بزرگ که اولین سطح قیمتی در order book را «بلغید<sup>۱</sup>»، بهترین قیمت فروش از ۱۰۰.۱۰ به ۱۰۰.۱۱ افزایش یافت.
- بنابراین، وقتی بازار صعودی است، مردم تمایل دارند سهام بخرند و قیمت بالا می‌رود.

- یک ساختار داده کارا برای یک order book باید شرایط زیر را برآورده کند:
- زمان جستجوی ثابت. عملیات شامل موارد زیر است: دریافت حجم در یک سطح قیمتی یا بین سطوح قیمتی، بهترین پیشنهاد خرید/فروش را دریافت کنید.
  - عملیات سریع افزودن/لغو/اجرا/بهروزرسانی، ترجیحاً پیچیدگی زمانی (O(1)). این عملیات شامل موارد زیر است: سفارش جدیدی قرار دهید، یک سفارش را لغو کنید و یک سفارش را مطابقت دهید.

## طراحی بورس اوراق بهادر

بازار سهام اخیراً متلاطم بوده است. در اینجا مختصراً در مورد طراحی یک سامانه بورس اوراق بهادر صحبت می‌کنیم.



مرحله ۱: مشتری سفارشی را از طریق برنامه وب یا موبایل یک کارگزاری درخواست می‌کند.

مرحله ۲: نرمافزار کارگزاری، سفارش را به بورس ارسال می‌کند.

مرحله ۳: درگاه مشتری بورس عملیاتی مانند اعتبارسنجی، محدودیت نرخ، احراز هویت، عادی‌سازی و غیره را انجام می‌دهد و سفارش را به مدیر سفارش ارسال می‌کند.

مرحله ۴: مدیر سفارش بررسی‌های خطر را بر اساس قوانین تعیین شده توسط مدیر خطر انجام می‌دهد.

مرحله ۵: پس از تأیید بررسی‌های خطر، مدیر سفارش بررسی می‌کند که آیا در کیف پول، مانده کافی وجود دارد یا خیر.

مرحله ۶-۷: سفارش به موتور مطابقت ارسال می‌شود. اگر مطابق‌تی پیدا شود، موتور مطابقت نتیجه اجرا را ارسال می‌کند. هم سفارش و هم نتایج اجرا باید ابتدا مرتب شوند تا مطابقت‌ها تضمین شود.

مرحله ۸ - ۱۰: نتیجه اجرا به کل مسیر بازگشت به مشتری انتقال داده می‌شود.

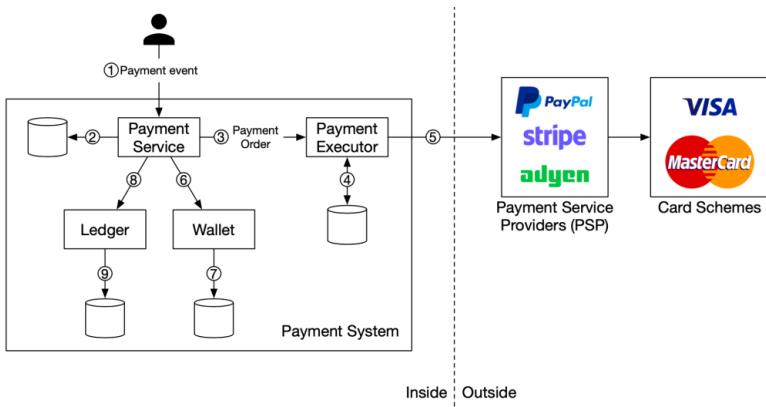
مرحله ۱۲-۱۱: داده‌های بازار (از جمله نمودار میله‌ای و order book) برای تلفیق به سرویس داده ارسال می‌شوند. کارگزاران برای دریافت داده‌های بازار از سرویس داده پرس‌وجو می‌کنند.

مرحله ۱۳: گزارشگر همه فیلد‌های گزارش‌دهی ضروری (مثلاً client\_id، قیمت، مقدار، remaining\_quantity، filled\_quantity، order\_type) را تنظیم می‌کند و داده‌ها را برای ماندگاری در پایگاه‌داده می‌نویسد.

یک بورس اوراق بهادار به تأخیر زمانی فوق العاده پایینی نیاز دارد. در حالی که بیشتر برنامه‌های وب با تأخیر زمانی صدها میلی‌ثانیه‌ای مشکلی ندارند، یک بورس اوراق بهادار به تأخیر زمانی میکرو‌ثانیه‌ای نیاز دارد.

## طراحی یک سیستم پرداخت

اینجا نحوه حرکت پول را زمانی که دکمه خرید را در آمازون یا هر یک از وبسایت‌های خرید مورد علاقه خود کلیک می‌کنید، توضیح می‌دهیم.

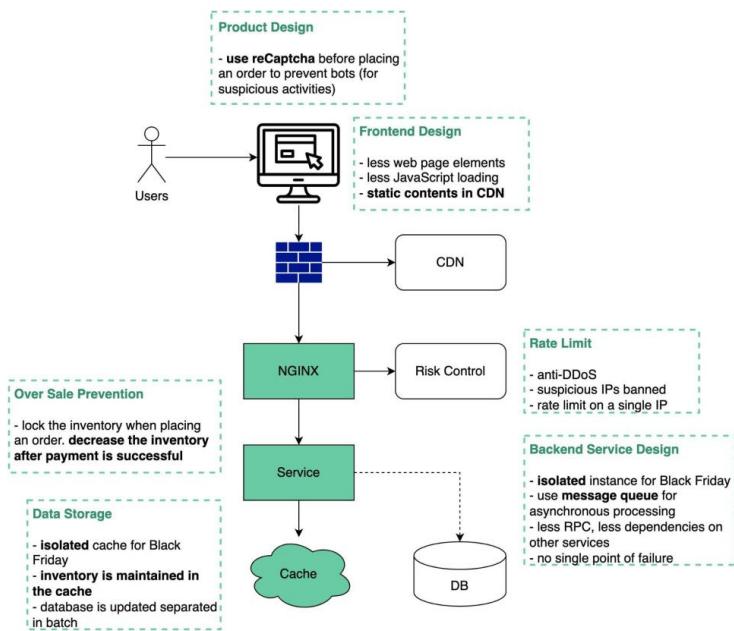


۱. وقتی یک کاربر دکمه «خرید» را کلیک می‌کند، یک رویداد پرداخت ایجاد می‌شود و به سرویس پرداخت ارسال می‌شود.
۲. سرویس پرداخت رویداد پرداخت را در پایگاهداده ذخیره می‌کند.
۳. گاهی اوقات یک رویداد پرداخت واحد ممکن است حاوی چندین سفارش پرداخت باشد. برای مثال، ممکن است محصولات را از چندین فروشنده در یک فرایند خرید انتخاب کنید. سرویس پرداخت برای هر سفارش پرداخت، مجری پرداخت را فراخوانی می‌کند.
۴. مجری پرداخت، سفارش پرداخت را در پایگاهداده ذخیره می‌کند.
۵. مجری پرداخت یک PSP خارجی را برای تکمیل پرداخت کارت اعتباری فراخوانی می‌کند.

۶. پس از اینکه مجری پرداخت به طور موقتی آمیزی پرداخت را اجرا کرد، سرویس پرداخت کیف پول را به روزرسانی می‌کند تا ثبت کند که یک فروشنده معین چه مقدار پول دارد.
۷. سرور کیف پول، اطلاعات مانده به روزرسانی شده را در پایگاهداده ذخیره می‌کند.
۸. پس از اینکه سرویس کیف پول به طور موقتی آمیزی اطلاعات مانده فروشنده را به روزرسانی کرد، سرویس پرداخت دفترکل را برای به روزرسانی آن فراخوانی می‌کند.
۹. سرویس دفترکل، اطلاعات جدید دفترکل را به پایگاهداده ضمیمه می‌کند.
۱۰. هر شب PSP یا بانک‌ها فایل‌های تسویه را برای مشتریان خود ارسال می‌کنند. فایل تسویه شامل مانده‌حساب بانکی، به همراه همه تراکنش‌هایی که در طول روز در این حساب بانکی رخداده است را دارد.

## طراحی یک سیستم فروش/حراج

جمعه سیاه در راه است. طراحی یک سیستم با قابلیت دسترسی همزمان فوق العاده بالا در دسترس بودن بالا و پاسخگویی سریع نیاز به درنظر گرفتن بسیاری از جنبه‌ها همه راههای ممکن از جلو دارد. برای جزئیات به تصویر زیر مراجعه کنید:



اصول طراحی:

۱. حتی مقداری کم هم گاهی اوقات زیاد است - عناصر کمتر در صفحه وب، کوئری داده‌های کمتری روی پایگاهداده، درخواست‌های وب کمتر، وابستگی‌های سیستمی کمتر و ...
۲. مسیر بحرانی کوتاه - پرش‌های میان سرویس‌ها یا ادغام در یک سرویس
۳. غیرهمزمانی - از صفحه‌ای پیام برای مدیریت نرخ بالای تراکنش در هر ثانیه استفاده کنید.

۴. ایزوله کردن - محتواهای پویا و ایستا را از یکدیگر جدا کنید، فرایندها و پایگاهداده‌ها را برای موارد حتی نادر از یکدیگر جدا کنید.
۵. فروش بیش از حد کاری نامناسب است بهخصوص زمانی که احتمال کاهش موجودی وجود دارد.
۶. تجربه کاربر مهم است. ما قطعاً نمی‌خواهیم به کاربران بگوییم که سفارش‌های خود را با موفقیت قرار داده‌اند؛ اما بعداً به آنها بگوییم که در واقع کالایی در دسترس نیست.

## چگونه یک بورس اوراق بهادار مدرن به تأخیرهای میکروثانیه دست می‌یابد؟

اصول اصلی به این شرح است:

- زمان کمتری برای هر کار
- تعداد جهش‌های شبکه‌ای کمتری
- استفاده کمتر از دیسک
- تعداد کمتری کار روی مسیر بحرانی

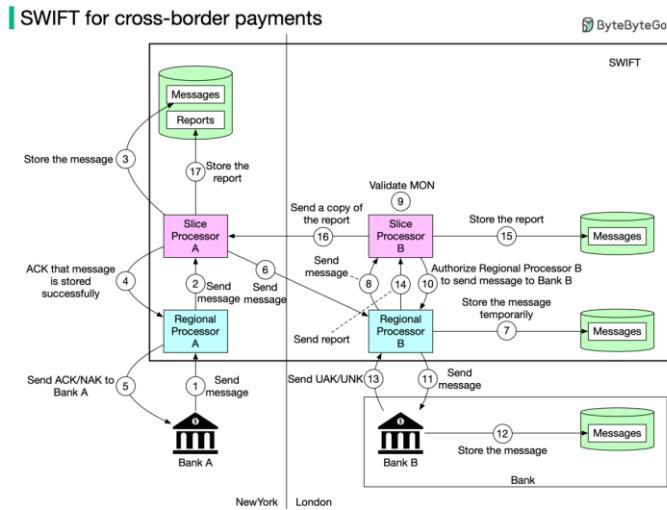
مسیر بحرانی برای بورس اوراق بهادار به شرح زیر است:

- شروع: سفارشی به مدیر سفارش وارد می‌شود
  - بررسی‌های خطر اجباری
  - سفارش مطابقت دارد و اجرای آن ارسال می‌شود
  - پایان: اجرای سفارش از مدیر سفارش خارج می‌شود
- با استی سایر کارها غیر بحرانی از مسیر بحرانی حذف شوند.
- ما طراحی را همان‌طور که در نمودار نشان‌داده شده، گرد هم آوردیم:
- همه مؤلفه‌ها را در یک سرور غول‌پیکر واحد مستقر کنید (بدون کاتئنیر)
  - از حافظه مشترک به عنوان یک اتوبوس رویداد برای برقراری ارتباط بین مؤلفه‌ها استفاده کنید، بدون دیسک سخت

- مؤلفه‌های کلیدی مانند مدیر سفارش و موتور مطابقت روی مسیر بحرانی تکرشته‌ای هستند و هر کدام به یک CPU سنجاق می‌شوند تا هیچ پرش رشته‌ای و هیچ قفلی وجود نداشته باشد
- حلقه برنامه تکرشته‌ای، کارها را یکی پس از دیگری به ترتیب اجرا می‌کند
- سایر مؤلفه‌ها به اتوبوس رویداد گوش می‌دهند و بر اساس آن واکنش نشان می‌دهند

## شبکه پرداخت SWIFT

احتمالاً در مورد SWIFT شنیدهاید. SWIFT چیست؟ چه نقشی در پرداخت‌های فرامرزی ایفا می‌کند؟ می‌توانید پاسخ این سوالات را در اینجا پیدا کنید.



انجمان جهانی ارتباطات مالی بین‌بانکی<sup>۱</sup> (SWIFT) اصلی‌ترین سیستم امن ارسال پیام است که بانک‌های جهان را به هم متصل می‌کند.

این سیستم مستقر در بلژیک، توسط بانک‌های عضو آن اداره می‌شود و میلیون‌ها پیام پرداخت را در روز مدیریت می‌کند. نمودار زیر نشان می‌دهد که چگونه پیام‌های پرداخت از بانک A (در نیویورک) به بانک B (در لندن) ارسال می‌شوند.

مرحله ۱: بانک A پیامی با جزئیات انتقال را به پردازشگر منطقه‌ای A در نیویورک ارسال می‌کند. مقصد بانک B است.

مرحله ۲: پردازشگر منطقه‌ای، فرمت آن را تأیید و سپس آن را به پردازشگر برش A ارسال می‌کند. پردازشگر منطقه‌ای مسئول اعتبارسنجی پیام ورودی و صفات پیام‌های خروجی است. پردازشگر مقطعي<sup>۲</sup> مسئول ذخیره و هدایت امن پیام‌ها است.

<sup>۱</sup> Society for Worldwide Interbank Financial Telecommunication

<sup>۲</sup> Slice Processor

مرحله ۳: پردازشگر مقطعی A، پیام را ذخیره می‌کند.

مرحله ۴: پردازشگر مقطعی A، به پردازشگر منطقه‌ای A اطلاع می‌دهد که پیام ذخیره شده است.

مرحله ۵: پردازشگر منطقه‌ای A تأییدیه ACK/NAK را به بانک A ارسال می‌کند. ACK به این

معنی است که پیام به بانک B ارسال خواهد شد. NAK به این معنی است که پیام به بانک B ارسال  
نخواهد شد.

مرحله ۶: پردازشگر مقطعی A، پیام را به پردازشگر منطقه‌ای B در لندن ارسال می‌کند.

مرحله ۷: پردازشگر منطقه‌ای B پیام را موقتاً ذخیره می‌کند.

مرحله ۸: پردازشگر منطقه‌ای B یک شناسه یکتا MON (شماره خروجی پیام) به پیام اختصاص  
می‌دهد و آن را به پردازشگر مقطعی B ارسال می‌کند.

مرحله ۹: پردازشگر مقطعی B شناسه MON را تأیید می‌کند.

مرحله ۱۰: پردازشگر مقطعی B به پردازشگر منطقه‌ای B اجازه می‌دهد پیام را به بانک B ارسال  
کند.

مرحله ۱۱: پردازشگر منطقه‌ای B پیام را به بانک B ارسال می‌کند.

مرحله ۱۲: بانک B پیام را دریافت و ذخیره می‌کند.

مرحله ۱۳: بانک B تأییدیه UAK/UNK را به پردازشگر منطقه‌ای B ارسال می‌کند. UAK (تأییدیه  
ثبت کاربر) به این معنی است که بانک B پیام را بدون خطا دریافت کرده است؛ UNK (تأییدیه  
منفی کاربر) به این معنی است که بانک B خطای checksum را دریافت کرده است.

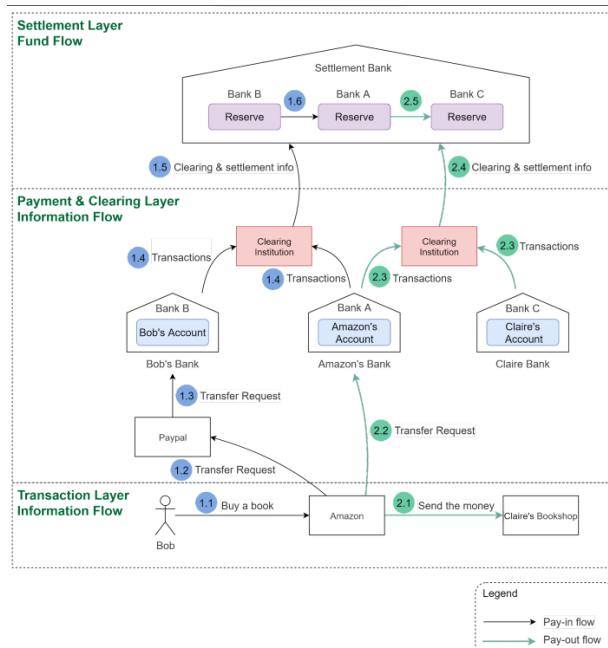
مرحله ۱۴: پردازشگر منطقه‌ای B بر اساس پاسخ بانک B، گزارشی تهیه و آن را به پردازشگر مقطعی  
B ارسال می‌کند.

مرحله ۱۵: پردازشگر مقطعی B گزارش را ذخیره می‌کند.

مرحله ۱۶ - ۱۷: پردازشگر مقطعی B یک نسخه از گزارش را به پردازشگر مقطعی A ارسال می‌کند.  
پردازشگر مقطعی A گزارش را ذخیره می‌کند.

## فرایند جابه‌جایی پول در نرم‌افزارهای بانکی

این تصویر بیش از هزار کلمه ارزش دارد. این اتفاقی است که زمانی که شما یک محصول را با استفاده از paypal/کارت‌بانکی می‌خرید، در پشت صحنه رخ می‌دهد.



برای درک این موضوع، ما نیاز داریم دو مفهوم را بررسی کنیم: فرایند حسابرسی<sup>۱</sup> و تصفیه<sup>۲</sup>. حسابرسی فرایندی است که محاسبه می‌کند چه کسی باید به چه کسی چقدر پول پرداخت کند؛ در حالی که تصفیه فرایندی است که در آن پول واقعی بین ذخایر در بانک تسویه جابه‌جا می‌شود. فرض کنید باب می‌خواهد یک کتاب را از فروشگاه آنلاین در آمازون بخرد. مسیر پرداخت (باب به آمازون پول می‌پردازد):

۱/۱. باب یک کتاب را در آمازون با استفاده از پی پال می‌خرد.

۱/۲. آمازون یک درخواست انتقال وجهه به پی پال صادر می‌کند.

<sup>۱</sup> clearing

<sup>۲</sup> settlement

۱/۳. از آنجایی که نشان پرداخت کارت بدهی باب در پی پال ذخیره شده است، پی پال می‌تواند از طرف باب، پول را به حساب بانکی آمازون در بانک A منتقل کند.

۱/۴. هر دو بانک A و بانک B گزارش‌های تراکنش را به مؤسسه حسابرسی ارسال می‌کنند. این کار تراکنش‌هایی را که نیاز به تصفیه دارند را کاهش می‌دهد. فرض کنید در پایان روز بانک A به بانک B ۱۰۰ دلار بدهکار است و بانک B به بانک A دقیقاً ۵۰۰ دلار بدهکار است. هنگامی که آنها تصفیه می‌کنند، موقعیت خالص این است که بانک B به بانک A ۴۰۰ دلار پرداخت می‌کند.

۱/۵. مؤسسه حسابرسی اطلاعات حسابرسی و تصفیه را به بانک جهت تصفیه حساب ارسال می‌کند. هر دو بانک A و بانک B از قبل وجودی را در بانک جهت تصفیه به عنوان ذخایر پولی سپرده گذاشته‌اند، بنابراین جابه‌جایی پول واقعی بین دو حساب ذخیره در حساب بانکی مخصوص تصفیه انجام می‌شود.

مسیر پرداخت (آمازون پول را به فروشنده: که خانمی به نام کلر است، می‌پردازد):  
۲/۱. آمازون به فروشنده (خانم کلر) اطلاع می‌دهد که به زودی به او پرداخت خواهد شد.

۲/۲. آمازون یک درخواست انتقال وجهه از بانک خودش (بانک A) به بانک فروشنده (بانک C) صادر می‌کند. در اینجا هر دو بانک تراکنش‌ها را ثبت می‌کنند، اما هیچ پول واقعی جابه‌جا نمی‌شود.

۲/۳. هر دو بانک A و بانک C گزارش‌های تراکنش را به مؤسسه حسابرسی ارسال می‌کنند.

۲/۴. مؤسسه حسابرسی اطلاعات حسابرسی و تصفیه را به بانک جهت تصفیه ارسال می‌کند. پول از ذخیره بانک A به ذخیره بانک C منتقل می‌شود.

توجه کنید که ما سه لایه داریم:

- لایه تراکنش: جایی که خریدهای آنلاین انجام می‌شود.
- لایه پرداخت و حسابرسی: جایی که دستورالعمل‌های پرداخت و خالص تراکنش‌ها انجام می‌شود.
- لایه تصفیه: جایی که جابه‌جایی پول واقعی انجام می‌شود.

دولایه اول مسیر اطلاعات نامیده می‌شوند و لایه تصفیه مسیر صندوق نامیده می‌شود.

شما می‌توانید بینید که مسیر اطلاعات و مسیر صندوق از هم جدا هستند. در مسیر اطلاعات، پول از یک حساب بانکی کسر و به حساب بانکی دیگری اضافه می‌شود، اما جابه‌جایی واقعی پول در بانک جهت تصفیه در پایان روز انجام می‌شود.

به دلیل ماهیت غیرهمگام مسیر اطلاعات و جریان صندوق، تطابق و هماهنگی<sup>۱</sup> برای یکپارچگی داده‌ها<sup>۲</sup> در سیستم‌ها در برای این مسیرها بسیار مهم است.

زمانی که باب می‌خواهد یک کتاب را در بازار هند بخرد، جایی که باب دلار آمریکا پرداخت می‌کند؛ اما فروشنده فقط می‌تواند روپیه هند دریافت کند، اوضاع جالب‌تر می‌شود.

asynchronous<sup>۱</sup>

data consistency<sup>۲</sup>

## تطبیق و سازگاری در پرداخت بانکی ۱

پست قبلی درباره مشکلات تسویه حساب باعث شد تا سؤال‌های جالب زیادی به وجود آید. یکی از موردهایی که مشکلات بیشتری را ممکن است وقتی با پردازنده‌های پرداخت میانجی کار می‌کنیم، برخورد کنیم، مطرح شود و راه حلی ممکن برای آن ارائه داده شود این است که:

۱. مشکل ارزهای خارجی: وقتی که یک فروشگاه معروف در سطح جهان (مثل آمازون) عملیاتی را انجام می‌دهید با این مشکل بسیار زیاد مواجه خواهد شد. برای برگشت به مثالی که در Paypal مطرح کردیم - اگر معامله در یک ارز متفاوت از ارز استاندارد Paypal که معمولاً دلار است، انجام شود، این کار یک لایه دیگر جهت پردازش ایجاد می‌کند که در آن ابتدا معامله در آن ارز دریافت می‌شود و سپس به ارزی که Paypal استفاده می‌کند تبدیل می‌شود. نیاز است که یک راه قابل اطمینان برای تسویه تبدیل ارز ارائه شود. البته این کار به این واقعیت کمک نمی‌کند که هر ارائه‌دهنده خدمات پرداخت به صورت متفاوتی این مسئله را مدیریت می‌کند.

۲. ارائه‌دهنده‌گان خدمات پرداخت فقط واسطه هستند. هر خرید، ۴ رویداد را برای یک شرکت به وجود می‌آورد. خرید از طریق Paypal (که در آن هم زمان و بعد ارزی نقش مهمی بازی می‌کنند) یک جفت خطوط برداشت/واریز برای معامله ایجاد می‌کند و معمولاً چند روز بعد، یک جفت دیگر وقتی پول از Paypal به حساب بانکی منتقل می‌شود (که در اینجا ممکن است تفاوت ارزی دیگری برای تسویه وجود داشته باشد اگر به عنوان مثال، خرید اولیه با واحد پولی ژاپن (JPY) انجام شده باشد، Paypal آن را با دلار آمریکا تنظیم می‌کند حتی اگر حساب بانکی شما بر اساس یورو باشد). در نتیجه نیاز است که یک راه برای تسویه همه این‌ها وجود داشته باشد.

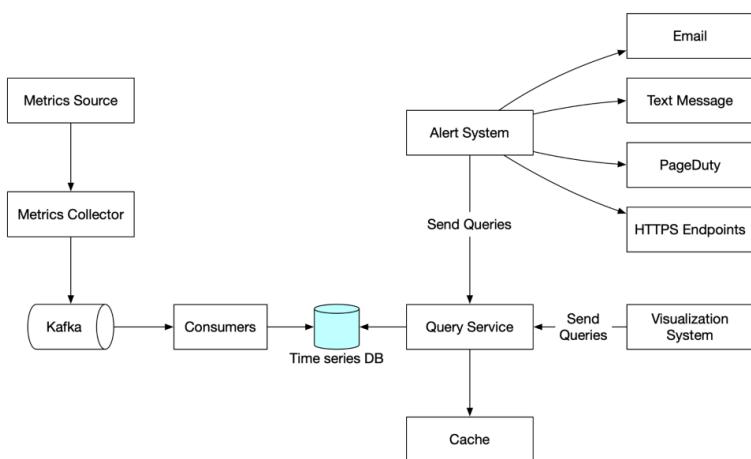
۳. برخی از مشکلات نیز در طرف خریدار بروز می‌کنند که بسیار وابسته به پلتفرم هستند. یک مثال از این مشکلات، تراکنش سایه‌ای<sup>۱</sup> در Paypal است: اگر شما دو موردی که بین آن‌ها یک هفته فاصله زمان باشد را از طریق Paypal خریداری کنید، ابتدا پول را از حساب بانکی شما برای تراکنش A کسر خواهد کرد. اگر در زمان تراکنش B، تراکنش A به طور کامل انجام نشده یا لغو شده باشد، ممکن است در حالتی قرار بگیریم که Paypal از پول تراکنش A برای پرداخت جزئی تراکنش B استفاده کند که منجر به خروج تنها مقداری از تراکنش B از حساب بانکی خواهد شد.

در عمل، این کار معمولاً<sup>۲</sup> به شکل زیر به نظر می‌رسد:

۱. فروشگاهی که از آن خرید آنلاین می‌کنید یک شماره سفارش به خرید شما اختصاص می‌دهد.
۲. شماره سفارش به پلتفرم پرداخت منتقل می‌شود.
۳. ارائه‌دهنده خدمات پرداخت، شناسه داخلی<sup>۳</sup> دیگری را ایجاد می‌کند که در تراکنش‌های بعدی سیستم انتقال می‌یابد.
۴. شناسه پرداخت هنگامی که پرداخت خود را در حساب بانکی دریافت می‌کنید (یا پرداخت‌کننده پرداخت‌های فردی را به صورت یکجا در سیستم خود ترکیب می‌کند که می‌تواند در سیستم پرداخت‌کننده بازگشت داده شود) استفاده می‌شود.
۵. بهتر است پرداخت‌کننده شما و فروشگاه آنلاینی که از آن خرید می‌کنید یک سیستم API مجتمع شده داشته باشند و از ابزاری برای (به طور امیدوارانه به صورت

خودکار) ایجاد فاکتورها استفاده کند. این کار معمولاً شماره سفارش را از فروشگاه آنلاین انتقال می‌دهد (حلقه را بسته می‌کند) و گاهی اوقات شناسه پرداخت را برای همخوانی با شناسه فاکتور مطابقت می‌دهد که پس از می‌توانید آن را با حساب داشته‌های دریافتی/پرداختی خود مطابقت دهید.

### بررسی پایگاهداده مناسب برای سروی جمع‌آوری متريک‌ها



تعداد زیادی از سیستم‌های ذخیره‌سازی موجود هستند که برای داده‌های زمانی<sup>۱</sup> بهینه شده‌اند. بهینه‌سازی امکان استفاده از تعداد کمتری سرور را برای مدیریت همان حجم داده به ما می‌دهد. بسیاری از این پایگاهداده‌ها نیز کوئری‌های سفارشی دارند که به صورت ویژه برای تحلیل داده‌های زمانی طراحی شده‌اند و استفاده از آن‌ها آسان‌تر از SQL است. برخی حتی ویژگی‌هایی برای مدیریت نگهداری داده و تجمعیع داده ارائه می‌دهند. چندین نمونه از پایگاهداده‌های سری زمانی<sup>۲</sup> عبارت‌اند از:

<sup>۱</sup> time-series

<sup>۲</sup> time-series databases

OpenTSDB یک پایگاهداده زمانی توزیع شده<sup>۱</sup> است، اما چون بر پایه Hadoop و HBase ساخته شده است، درنتیجه اجرای یک خوش Hadoop/HBase پیچیدگی اضافی ایجاد می‌کند. Twitter از MetricsDB استفاده می‌کند و Amazon Timestream را به عنوان یک پایگاهداده زمانی ارائه می‌دهد. بر اساس DB-engines، دو پایگاهداده زمانی محبوب InfluxDB و Prometheus هستند که برای ذخیره‌سازی حجم بزرگی از داده‌های زمانی طراحی شده‌اند و می‌توانند تحلیل‌های بلادرنگ را به سرعت انجام دهند. هر دو از یک حافظه کش داخلی<sup>۲</sup> و ذخیره‌سازی روی دیسک استفاده می‌کنند و هر دو به خوبی در مورد ماندگاری داده‌ها و کارایی بالا عمل می‌کنند. بر اساس بنچمارک، یک InfluxDB با ۸ هسته و ۳۲ گیگابایت RAM می‌تواند بیش از ۲۵۰,۰۰۰ نوشتمن در ثانیه را انجام دهد.

چون time-series databases یک پایگاهداده ویژه است، انتظار نمی‌رود که در یک مصاحبه، قسمت‌های داخلی آن را متوجه شویم، مگر اینکه به طور صریح در رزومه شما ذکر شده باشد. برای هدف مصاحبه، مهم است که متوجه شویم که داده‌های متريک زمانی هستند و می‌توانیم از پایگاهداده‌های زمانی مانند InfluxDB برای ذخیره‌سازی آن‌ها استفاده کنیم. یکی دیگر از ویژگی‌های پایگاهداده زمانی قوی، توانایی تجمعی و تحلیل مقادیر زیادی از داده‌های زمانی بر اساس برچسب‌ها<sup>۳</sup> یا tag است. به عنوان مثال، InfluxDB برچسب‌ها را به عنوان شاخص‌هایی<sup>۴</sup> برای جستجوی سریع داده‌های زمانی استفاده می‌کند و بهترین شیوه را برای استفاده از برچسب‌ها را بدون بارگذاری بیش از حد داده‌ها ارائه می‌دهد. نکته کلیدی این است که مطمئن شویم هر برچسب دارای تعداد کمی از مقادیر ممکن است. این ویژگی برای نمایش داده‌ها بسیار حیاتی است و ساختن آن با یک پایگاهداده عمومی زمان و تلاش زیادی نیاز دارد.

<sup>۱</sup> distributed time-series database

<sup>۲</sup> in-memory cache

<sup>۳</sup> labels

<sup>۴</sup> indexes

## تطبیق و سازگاری در پرداخت بانکی ۲

تطبیق شاید پردردسرترین فرایند در یک سیستم پرداخت باشد. این فرایند مقایسه رکوردها در سیستم‌های مختلف برای اطمینان از تطابق مبالغ با یکدیگر است.

- برای مثال، اگر ۲۰۰ دلار برای خرید یک ساعت با Paypal پرداخت کنید:
  - وبسایت تجارت الکترونیکی باید رکوردهای درباره سفارش خرید به مبلغ ۲۰۰ دلار داشته باشد.
  - باید یک رکورد تراکنش ۲۰۰ دلاری در Paypal وجود داشته باشد (در نمودار با ۲ مشخص شده است).
  - دفترکل باید یک بدھی ۲۰۰ دلاری برای خریدار و یک اعتبار ۲۰۰ دلاری برای فروشنده را ثبت کند. این را حسابداری دو طرفه می‌نامند (جدول زیر را ببینید).  
بیایید نگاهی به برخی از نقاط دردنگ و چگونگی رفع آنها بیندازیم:

**مشکل ۱:** نرمال سازی داده<sup>۱</sup>. زمانی که رکوردها در سیستم‌های مختلف را مقایسه می‌کنیم، آنها در قالب‌های متفاوتی می‌آیند. برای مثال، برچسب زمانی<sup>۲</sup> می‌تواند در یک سیستم "۲۰۲۰/۰۱/۰۱" و در سیستم دیگر "۱ ژانویه ۲۰۲۲" باشد.

**راه حل ممکن:** می‌توانیم یک لایه اضافه کنیم تا قالب‌های مختلف را به یک قالب واحد تبدیل کند.

**مشکل ۲:** حجم داده عظیم.

<sup>۱</sup> Data normalization

<sup>۲</sup> timestamp

راه حل ممکن: می‌توانیم از تکنیک‌های پردازش داده بزرگ برای سرعت بخشیدن به مقایسه داده‌ها استفاده کنیم. اگر به تطبیق نزدیک به زمان بلاذرنگ نیاز داشته باشیم، از پلتفرم Flink streaming<sup>۱</sup> مانند استفاده می‌شود؛ در غیر این صورت پردازش دسته‌ای انتهای روز مانند Hadoop کافی است.

مشکل ۳: مسئله زمان برش<sup>۲</sup>. برای مثال، اگر ساعت ۰۰:۰۰:۰۰ را به عنوان زمان برش روزانه انتخاب کنیم، یک رکورد با برچسب زمانی ۰۵:۵۹:۲۳ در سیستم داخلی ثبت می‌شود اما ممکن است در سیستم خارجی (Paypal) با ۳۰:۰۰:۰۰ ثبت شده باشد که مربوط به روز بعدی است. در این صورت، نمی‌توانیم این رکورد را در رکوردهای امروز Paypal پیدا کنیم. این باعث عدم تطابق می‌شود.

راه حل ممکن: ما نیاز داریم این عدم تطابق را به عنوان یک "وقفه موقت" طبقه‌بندی کنیم و آن را بعداً در برابر رکوردهای روز بعد Paypal اجرا کنیم. اگر در رکوردهای روز بعد Paypal تطابقی پیدا کردیم، عدم تطابق برطرف می‌شود و نیازی به اقدام بیشتر نیست.

راه حل احتمالی: شما ممکن است استدلال کنید که اگر در سیستم مفهوم دقیقاً یک‌بار<sup>۳</sup> داشته باشیم، نباید هیچ عدم تطابقی وجود داشته باشد. اما واقعیت این است که جاهای زیادی وجود دارد که می‌تواند مسیر اشتباهی برود. داشتن یک سیستم تطبیق همیشه ضروری است.

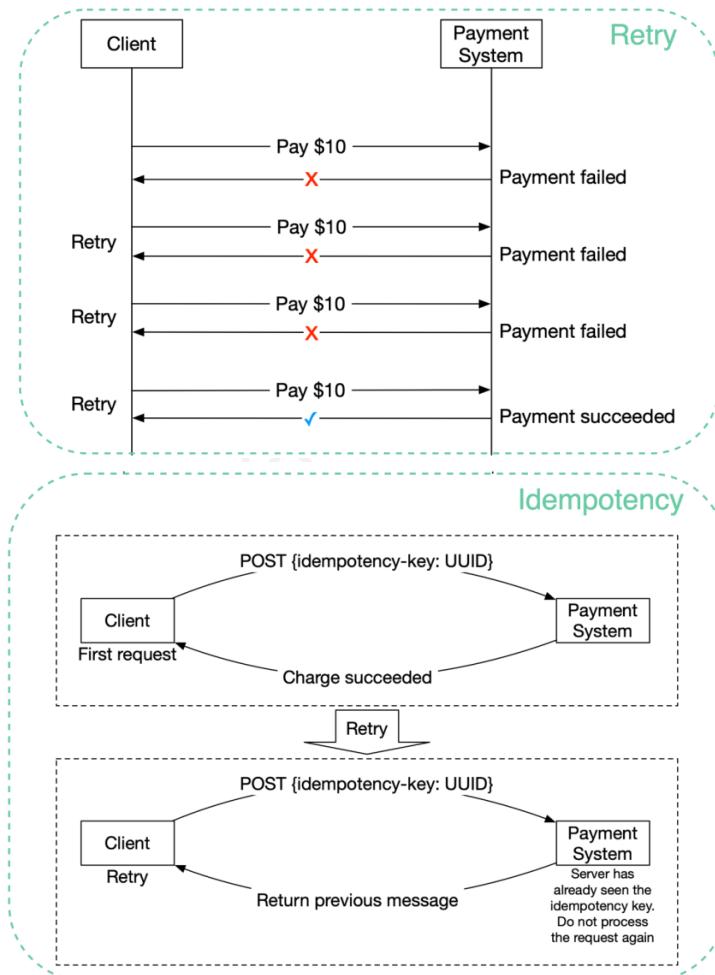
<sup>۱</sup> جریانی

<sup>۲</sup> Cut-off time issue

<sup>۳</sup> – در قسمت (حداکثر یک‌بار، حداقل یک‌بار و دقیقاً یک‌بار) توضیح داده شده است.

## جلوگیری از شارژ شدن دوباره

یکی از جدی‌ترین مشکلاتی که یک سیستم پرداخت می‌تواند داشته باشد، شارژ مجدد یک مشتری<sup>۱</sup> است. هنگام طراحی سیستم پرداخت، ضمانت اینکه سیستم پرداخت یک سفارش پرداخت را دقیقاً یک بار<sup>۲</sup> اجرا کند، بسیار مهم است.



double charge a customer<sup>۱</sup>  
exactly-once<sup>۲</sup>

در نگاه اول، تحویل دقیقاً یکبار بسیار دشوار به نظر می‌رسد، اما اگر مسئله را به دو بخش تقسیم کنیم، حل آن بسیار آسان‌تر است. از نظر ریاضی، یک عملیات دقیقاً یکبار اجرا می‌شود اگر:

۱. حداقل یکبار اجرا شود.

۲. در همان زمان، حداقل یکبار اجرا شود.

اکنون توضیح می‌دهیم چگونه با استفاده از مکانیزم تلاش مجدد<sup>۱</sup> و کنترل بی‌تأثیری<sup>۲</sup>، حداقل یکبار<sup>۳</sup> و حداقل یکبار<sup>۴</sup> را پیاده‌سازی کیم.

### تلاش مجدد – **retry**

گاهی اوقات به دلیل خطاهای شبکه یا مهلت زمانی<sup>۵</sup> تمام‌شدن درخواست‌ها، نیاز داریم یک تراکنش پرداخت را **retry** کنیم. تلاش مجدد یا در اصطلاح **retry**، تضمین حداقل یکبار را فراهم می‌کند. به عنوان مثال، همان‌طور که در شکل ۱۰ نشان داده شده است، کلاینت سعی در انجام یک پرداخت ۱۰ دلاری دارد، اما پرداخت به دلیل اتصال ضعیف شبکه مدواوماً شکست می‌خورد. با در نظر گرفتن اینکه شرایط شبکه ممکن است بهتر شود، کلاینت درخواست را دوباره تلاش می‌کند و این پرداخت در نهایت در چهارمین تلاش موفق می‌شود.

### بی‌تأثیری – **idempotency**

از دیدگاه یک API، بی‌تأثیری<sup>۶</sup> به این معنی است که کلاینت‌ها می‌توانند همان درخواست را چندین بار تکرار کنند و همان نتیجه را تولید کنند. برای ارتباط بین کلاینت‌ها (وب و اپلیکیشن‌های موبایل) و سرورها، معمولاً<sup>۷</sup> یک کلید بی‌تأثیری یک مقدار منحصر به فرد است

**retry**<sup>۸</sup>

**idempotency check**<sup>۹</sup>

**least once**<sup>۱۰</sup>

**most once**<sup>۱۱</sup>

**timeout**<sup>۱۲</sup>

**idempotency**<sup>۱۳</sup>

که توسط کلاینت‌ها ایجاد می‌شود و پس از مدت زمان معینی منقضی می‌شود. یک UUID معمولاً به عنوان کلید بی‌تأثیری استفاده می‌شود و توسط بسیاری از شرکت‌های تکنولوژی مانند PayPal و Stripe تووصیه می‌شود. برای انجام یک درخواست پرداخت بی‌تأثیر، یک کلید بی‌تأثیری به هدر HTTP اضافه می‌شود، به عنوان مثال:

.<idempotency-key: key\_value>

### امنیت در پرداخت بانکی

جدول زیر تکنیک‌هایی را که معمولاً در امنیت پرداخت بانکی استفاده می‌شوند را خلاصه می‌کند.

مشکل	راه حل
شنود کردن درخواست/پاسخ	از HTTPS استفاده کنید
دست کاری داده‌ها	نظرارت بر یکپارچگی و رمزگذاری را اعمال کنید
حمله مرد میانی ۱	از SSL و گواهی‌های احراز هویت استفاده کنید
از دست رفتن داده‌ها	تکثیر ۲ پایگاه‌داده در چندین منطقه
حمله DDoS توزیع شده	firewall و Rate limiting
سرقت کارت	توکن‌سازی به جای استفاده از شماره کارت واقعی، توکن‌ها ذخیره می‌شوند و برای پرداخت استفاده می‌شوند
انطباق با PCI	PCI DSS یک استاندارد امنیت اطلاعات برای سازمان‌هایی است که کارت‌های اعتباری معتبر را مدیریت می‌کنند.
کلاهبرداری و تقلب	تأیید آدرس، تأیید ارزش کارت (CVV)، تجزیه و تحلیل رفتار کاربر و غیره.

<sup>۱</sup> Man-in-the-middle attack

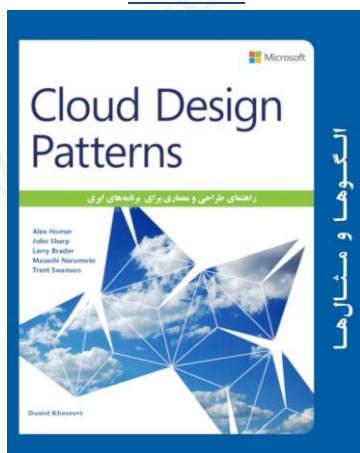
<sup>۲</sup> replication

<sup>۳</sup> Distributed denial-of-service attack (DDoS)

## ساير کتابها



طراحی سیستم‌های نرم‌افزاری ۱  
[لینک دانلود](#)



الگوهای طراحی ابری  
[لینک نسخه پیش از انتشار](#)