مقایسه معماری‌های مونولیتیک و میکروسرویس

خلاصه (Summary)

- این سند یک راهنمای جامع، عملی و پروژه‌محور است برای درک عمیق دو الگوی پرکاربرد معماری نرم‌افزار: «مونولیتیک» و «میکروسرویس».

- اگر تیم کوچک هستید، دامنه محدود دارید، و سرعت تحویل برایتان مهم است، مونولیت (یا «مونولیت ماژولار») معمولاً انتخاب بهتری است.

- اگر چند تیم موازی دارید، دامنه بزرگ و ناهمگن دارید، نیاز به مقیاس‌پذیری مستقل، استقلال چرخه انتشار، و جداسازی مرزهای بیزنسی دارید، میکروسرویس با درایت (و نه شتاب‌زده) مناسب‌تر است.

- بزرگ‌ترین ریسک میکروسرویس برای تیم‌های کوچک: «مونولیت توزیع‌شده» (Distributed Monolith) — پیچیدگی دو برابر بدون مزیت واقعی.

تعریف و تفاوت‌های ساختاری

-مونولیتیک :(Monolithic) کل اپلیکیشنAPI ، منطق دامنه، رندر، جاب‌ها، در یک کُدبیس و یک پکیج/دیپلوی واحد. پایگاه‌داده معمولاً یکجا/اشتراکی است. تست و دیپلوی ساده‌تر؛ مرزبندی لایه‌ها داخل همان کدبیس انجام می‌شود.

-مونولیت ماژولار: همان مونولیت اما با مرزبندی صریحِ پکیج‌ها/ماژول‌ها (لایه دامنه‌ها، آداپترها، سرویس‌ها) برای جلوگیری از درهم‌تنیدگی. بهترین سکوی پرش به میکروسرویس.

-میکروسرویس :(Microservices) سیستم از چندین سرویس مستقل تشکیل می‌شود. هر سرویس یک دامنه باریک را پوشش می‌دهد، دیتابیسِ خودش را دارد(Database per Service) ، با دیگر سرویس‌ها از طریق/API پیام (Sync/Async) ارتباط برقرار می‌کند و چرخه انتشار مستقل دارد.

-تفاوت‌های کلیدی ساختاری:

- -مرزها: در مونولیت «مرز درون کد» است؛ در میکروسرویس «مرز در سطح اجرا/شبکه.«
- -دیتابیس: در مونولیت معمولاً یک DB مشترک وجود دارد؛ در میکروسرویس DB هر سرویس جدا است و منبعِ Shared« «DBبین سرویس‌ها.
- -ارتباطات: مونولیت — فراخوانی درون‌فرآیندی؛ میکروسرویس — شبکه/(HTTP/gRPC) پیام‌رسان.(

• -استقرار: مونولیت — یک آرتیفکت و یک کانتینر/سرویس؛ میکروسرویس — چندین سرویس، ارکستریشن.(Kubernetes)

• -تیم‌ها: مونولیت — یک تیم یا چند تیم روی یک ریپو؛ میکروسرویس — تیم‌های «کراس فانکشنال» هرکدام مالک یک سرویس.

احراز هویت و امنیت(AuthN/AuthZ, Security)

- مونولیت:

• -لایه Auth متمرکز داخل همان اپ. الگوهای رایج: سشن/کوکی امن (HttpOnly, SameSite, Secure) یا JWT. CSRF برای کوکی ضروری.

برای RBAC/ABAC • -درون همان کُدبیس پیاده‌سازی و Audit Log مشترک.

• -محافظت ازAdmin Panel ، Rate Limit و فایروال سطح Nginx/Reverse Proxy ساده‌تر.

- میکروسرویس:

• - درگاه API (API Gateway یا) «Identity Provider (IdP)» مانند (Keycloak/Auth0 نقطه ورود. استاندارد OAuth2.1/OIDCبرایAuthN.

• - توکن‌های امضاشده (JWT/JWS) با Scope/Role برای «انتشار» هویت به سرویس‌ها؛ احراز اصالت امضا با JWKS و چرخش کلید.(Key Rotation)

• - ارتباط سرویس-به-سرویس mTLS (Service Mesh :مثل Istio/Linkerd) + Authorization در مرز هر سرویس.

• - اصل :Zero-Trust هر درخواست باید احراز هویت و مجوز داشته باشد؛ به «شبکه داخلی» اعتماد ضمنی نباید کرد.

• - سکرتها (Secrets) در/Secret Manager کوبرِنتسSecret ؛ عدم هاردکدینگ. حداقل سطح دسترسی برای DB/Broker/Cache.

• - لاگ ممیزی متمرکز(SIEM) ، WAF، Rate Limit سراسری، و محافظت DDoS در لایه Gatewayگیتوی.

• -نکته پایتونی: در FastAPI می‌توانOAuth2PasswordBearer/OIDC Client ، PyJWT/JOSEو dependencyهای امنیتی را در Gateway و سرویس‌ها پیاده کنید.

مقیاس‌پذیری (Scalability)

-مونولیت:

• -مقیاس عمودی (Vertical) ساده CPU/RAM :بیشتر. مقیاس افقی (Replica) ممکن است با «حالت‌مند بودن سشن/فایل» دردسر ایجاد کند) نیاز به.(Redis/Shared Storage

کشینگ (Redis) Caching • -و جداسازی خواندن/نوشتن در (Read Replicas) DB می‌تواند ظرفیت را بالا ببرد.

-میکروسرویس:

• -مقیاس افقیِ مستقل برای هر سرویس متناسب با بار. مثال: «سرویس رزرو» را ۱۰ برابر کنید بدون افزایش «سرویس گزارش‌گیری.«

• -امکان انتخاب فناوری بهینه برای هر دامنه — (Polyglot Persistence) ولی مراقب هزینه عملیاتی باید باشیم.

• -ارتباط آسنکرون) پیام‌محور (Kafka/RabbitMQ :باعث دِکاپلینگ و جذب پیک می‌شود؛ نیازمند idempotency و DLQ.

• -الگوهای معماری Saga :برای تراکنش‌های چندسرویسی، Outboxبرای تضمین تحویل، Circuit Breaker/Retry/Timeoutبرای تاب‌آوری.

۴ هزینه‌ها (پیاده‌سازی، نگهداری، توسعه

-مونولیت ـــ هزینه ورود پایین‌تر:

• -زیرساخت: یک ریپو/سرویس، یک پایپلاین، Composeیا یکVM ؛ ارزان‌تر و سریع‌تر.

• -توسعه onboard :ساده‌تر، دیباگ و تست کم‌هزینه‌تر، تصمیمات معماری کمتر.

• -ریسک: با رشد زیاد، couplingکد و «زمان «Build/Deploy زیاد می‌شود؛ اما با ماژولارکردن قابل مدیریت است.

-میکروسرویس ـــ هزینه ثابت بالاتر:

• -زیرساختK8s :، Service Mesh، Observability Stack (Prometheus/Grafana/Loki/Tempo/Jaeger)، API Gateway، Registry، Secrets، CI/CDچندگانه.

• -تیم: نیاز بهDevOps/Platform Engineering ، مانیتورینگ و امنیت جدی‌تر، قراردادهای بین‌سرویسی، مدیریت نسخه/سازگاری.

• -توسعه: تست‌های یکپارچه/قراردادی پیچیده‌تر، دیباگ توزیع‌شده، مدیریت داده و تراکنش‌های بین‌سرویس.

پیچیدگی، تست، استقرار، خطایابی، وابستگی‌ها

-پیچیدگی: میکروسرویس پیچیدگی «زیرساخت» و «هماهنگی بین سرویس‌ها» را اضافه می‌کند؛ اگر ارزش بیزنسیِ استقلال/مقیاس نیاز نباشد، صرفاً سربار است.

-تست:

• -مونولیت: پوشش‌دهی واحد/یکپارچه ساده‌تر. تست End-to-End یک سیستم واحد.

• -میکروسرویس: اضافه‌شدن«Contract Testing (Consumer-Driven)» ، تست‌های همگرایی، تست‌های مقاومتی شبکه، تستِ روی داده‌های نهاییِ ایونت‌محور.

-استقرار:

• -مونولیت Blue/Green :یا Rolling ساده. ساده‌سازی با.Docker+Compose/Gunicorn/Nginx

• -میکروسرویس: کاناری Progressive Delivery/باFeature Flags ، Helm/Kustomize، .ArgoCD/FluxCD

-خطایابی/مشاهده‌پذیری:

• -مونولیت: لاگ و متریک متمرکز، Traceساده.

• -میکروسرویس Trace :توزیع‌شده(OpenTelemetry→Jaeger/Tempo) ، Correlation-ID، Centralized Logging (ELK/Loki)، متریک‌ها.(Prometheus)

-وابستگی‌ها/نسخه‌ها:

• -میکروسرویس: «سازگاری رو به عقب (Backward Compatibility) «و نسخه‌بندی قراردادها (SemVer) حیاتی است. از «Shared DB Schema»پرهیز کنیم.

مدیریت داده و تراکنش

-مونولیت: تراکنش ACID چندجدولی ساده) یک.(DB

-میکروسرویس: پرهیز از PC2؛ به‌جایش الگوی(Saga (Choreography/Orchestration ، Eventual Consistency، طراحی«Idempotent Command/Handler» ، Outbox/Inboxبرای تضمین تحویل و جلوگیری از دو بار اعمال.

ضدالگوها(Anti-Patterns)

Distributed Monolith: -چند سرویس اما به‌شدت به هم کوپل؛ همه باید با هم منتشر شوند.

Nanoservices: -شکستن بیهوده دامنه به سرویس‌های بسیار ریز با ارزش افزوده ناچیز و سربار.massive

Shared Database Between Services: -ضدالگو؛ شکستن استقلال سرویس‌ها و قفل‌شدن نسخه‌ها.

چارچوب تصمیم‌گیری(Decision Framework)

-چک‌لیست سریع: اگر ≤۳ مورد زیر «بله» است، باید به میکروسرویس فکر کنیم؛ در غیر این صورت «مونولیت ماژولار» شروع خوبی است:

- -چند تیم هم‌زمان روی دامنه‌های مستقل کار می‌کنند.
- -سرویس‌ها نیاز به مقیاس‌پذیری مستقل دارند (پروفایل بار متفاوت).
- -چرخه‌های انتشار مستقل برای هر دامنه ارزش بیزنسی دارد.
- -مرزهای دامنه (Bounded Context) روشن و پایدارند.
- -بودجه/تخصص DevOps برای K8s/Observability/Security موجود است.

-نمره‌دهی ساده (۰ تا ۵) برای هر معیار: اندازه دامنه، رشد مورد انتظار، تنوع فناوری، استقلال تیم‌ها، نیاز به زمان تحویل مستقل، حساسیت امنیتی/قانونی. اگر میانگین <۳ شد ⟵ میکروسرویس.

الگوهای پیاده‌سازی با) FastAPI پیشنهادی(

-مونولیت ماژولار (پروژه تک‌ریپو):

- • apps/ – auth, users, orders, payments, reporting (هر کدام پکیج مجزا)
- • core/ – config, db, security, celery, utils
- • api/ – v1/routers.py (شکستن روت‌ها به ماژول‌ها)
- • tests/ – واحد/یکپارچه
- • infra/ – docker-compose.yml, nginx.conf

• -مزیت: سرعت بالا، هماهنگی ساده، دیباگ سریع. مناسب شروع و تیم کوچک.

-میکروسرویس (چندریپو یا مونو-ریپو):

- • gateway/ (Traefik/Kong/Nginx + OIDC)
- • services/
- ├– auth-service (FastAPI + PostgreSQL + JWT/JWKS)
- ├– reservation-service (FastAPI + PostgreSQL + Kafka/RabbitMQ)
- ├– billing-service (FastAPI + PostgreSQL + پرداخت/وب‌هوک)
- └– reporting-service (FastAPI + Read-Model/CQRS)
- • platform/ – Helm charts, K8s manifests, GitOps (ArgoCD), Observability stack
- • قراردادها: OpenAPI/AsyncAPI برای API/رویدادها + Pact برای Contract Testing.

احراز هویت: سناریوی نمونه توکنی(Gateway → Services)

• -کاربر در Gateway لاگین می‌کند (OIDC/OAuth2). Access Token (JWT) با Claims (sub, roles, scopes) دریافت می‌شود.

• -کلاینت در هر درخواست Authorization: Bearer <token> می‌فرستد.

• -سرویس‌ها با کلید عمومی (JWKS) امضای توکن را اعتبارسنجی و Claims را برای RBAC بررسی می‌کنند.

• -برای سرویس-به-سرویس client_credentials: یا mTLS با SPIFFE/SPIRE. توکن با audience مخصوص سرویس مقصد صادر می‌شود.

• -چرخش کلید (kid) و انقضا (exp) الزامی؛ استفاده از Cache برای JWKS با بک‌آف.

نمونه ساختار کانفیگ/دیپلوی

- Docker Compose (شروع سریع مونولیت):
- • web (uvicorn/gunicorn) + db (PostgreSQL) + redis + nginx + celery/worker/beat.
- Kubernetes (میکروسرویس):

• -هر سرویس یکDeployment + Service ؛ Ingress Controller (NGINX/Traefik)؛ Secret/ConfigMap ؛
HPA; Service Mesh؛ Observability باPrometheus/Grafana/Tempo/Loki ؛ Jaeger/Tempo برای.Trace

تست و تضمین کیفیت

- • Pyramid: واحد (fast), یکپارچه (db/broker یا Testcontainer) ، قراردادی(Pact), E2E (کم و هدفمند).

- • Chaos/Resilience: تست قطع موقتی یک سرویس، افزایش تاخیر شبکه، Circuit-Breaker/Retry/Timeout.

- • Data Contract: اسکیماهای رویداد/پیام نسخه‌دار؛ Validation سمت مصرف‌کننده.

برنامه مهاجرت مونولیت ⟶ میکروسرویس (گام‌به‌گام)

۱- (ماژولارسازی مونولیت و روشن‌کردن مرزهای دامنه.(Bounded Context)

۲- (استخراج «اولویت‌های داغ» (رزرو/پرداخت/گزارش) با KPI روشن.(Latency/Throughput/Change Frequency)

۳- (معرفی Gateway API و Auth مرکزی؛ سخت‌گیری روی.Observability

۴- (استخراج سرویسِ هدف با «استراتژی استرانگلر:(Strangler Fig)

مسیرهای جدید ⟶ سرویس جدید، مسیرهای قدیمی هنوز از مونولیت.

۵- (داده: الگوی Outbox برای مهاجرت تدریجی؛ همگام‌سازی رویدادی؛ قطع وابستگی به.Shared Schema

۶- (حکمرانی: نسخه‌بندی قراردادها، Backward Compatibility، چرخه انتشار مستقل.

۷- (بهینه‌سازی هزینه: اندازه سرویس‌ها را منطقی نگه دارید؛ از nanoservice بپرهیزید.

کاربرد در یک پروژه برای مثال : رزرو سرویس خودرو

-شروع عملی پیشنهادی: «مونولیت ماژولار FastAPI» برای .Auth/Users/Reservation/Payments/Reporting
(Compose + Nginx + Postgres + Redis + Celery)

-وقتی رزرو/پرداخت بار بالایی گرفت: استخراج reservation-service و billing-service با DB جدا، GatewayباOIDC ،
Queueبرای ارسال/SMS ایمیل/وب‌هوک. گزارش‌گیری را Read Model جدا.(CQRS)

-ملاحظات امنیتی شعب/گارانتیClaim :های نقش/شعبه درJWT ؛ سیاست‌های مجوز در Gateway و سرویس‌ها.

مزایا/معایب خلاصه

-مونولیت:

- -مزایا: سادگی، هزینه پایین، دیباگ سریع، تحویل سریع.MVP
- -معایب: رشد کُدبیس Build/Deploy → کند، coupling، مقیاس مستقلِ دامنه‌ها دشوار است.

-میکروسرویس:

- -مزایا: مقیاس‌پذیری/استقلال سرویس، انعطاف فناوری، انتشار مستقل، انزوا/تاب‌آوری.
- -معایب: هزینه ثابت بالا، پیچیدگی شبکه/امنیت/مشاهده‌پذیری/تست، نیاز به تیم باتجربه.

چک‌لیست نهایی انتخاب

✓ -تیم کوچک، بودجه محدود، ددلاین نزدیک ⟶ مونولیت ماژولار.

✓ -نیاز به مقیاس مستقل/انتشار مستقل/تیم‌های متعدد ⟶ میکروسرویس) با.(Gateway+Observability

ضمیمه A اسکلت پوشه‌ها — مونولیت ماژولار (FastAPI)

```
project/
```

- ├─ api/
- │ └─ v1/ (routers: auth.py, users.py, orders.py, ...)
- ├─ apps/
- │ ├─ auth/ (schemas.py, service.py, repository.py, routes.py)
- │ ├─ users/ ...
- │ └─ payments/ ...
- ├─ core/ (config.py, db.py, security.py, celery_conf.py)
- ├─ tests/
- ├─ infra/ (docker-compose.yml, nginx.conf)
- └─ main.py

ضمیمه B اسکلت پوشه‌ها — میکروسرویس (نمونه

```
repo/
```

- ├─ gateway/
- ├─ services/
- │ ├─ auth-service/
- │ ├─ reservation-service/
- │ ├─ billing-service/
- │ └─ reporting-service/
- └─ platform/ (k8s manifests, helm, gitops)

ضمیمه C اسنیپت‌های نمونه(FastAPI, Security)

-نمونه وریفی توکن در سرویس‌ها:(FastAPI)

```
from fastapi import Depends, HTTPException, status, FastAPI
```

```
from jose import jwt, JWTError
```

```
import httpx
```

```
JWKS_URL = "https://idp.example.com/.well-known/jwks.json"
```

- AUDIENCE = "reservation-service"
- ISSUER = "https://idp.example.com/"
- app = FastAPI()
- async def verify_token(authorization: str = Depends(...)):

-استخراج و وریفای JWT با کش‌کردن + JWKS بررسی + iss/aud/exp/nbf نقش‌ها/اسکوپ‌ها

- ...

-سپس dependency verify_token را روی روت‌ها اعمال کنید.

جمع‌بندی و توصیه نهایی

-برای اکثر تیم‌های کوچک/متوسط: «مونولیت ماژولار» بهترین نقطه شروع است. با مرزبندی دامنه‌ها، تست خوب، و observabilityاولیه، به‌محض نیاز می‌توانیم یک دامنه پرریسک/پربار را به میکروسرویس تبدیل کنیم.

-اگر میکروسرویس می‌خواهیم، ابتدا Platform/Observability/Gateway/IdP را درست کنیم؛ سپس هر سرویس را با قرارداد روشن، دیتابیس جدا، و تست قراردادی بسازیم؛ و از Shared DB و nanoservice دوری کنیم.