System Architecture Document

۱. چکیده معماری

۱.۱ رویکرد معماری

معماری نوبت‌نو بر پایه **Microservices** با **Event-Driven Architecture** طراحی شده است. این انتخاب به دلایل زیر انجام شده:

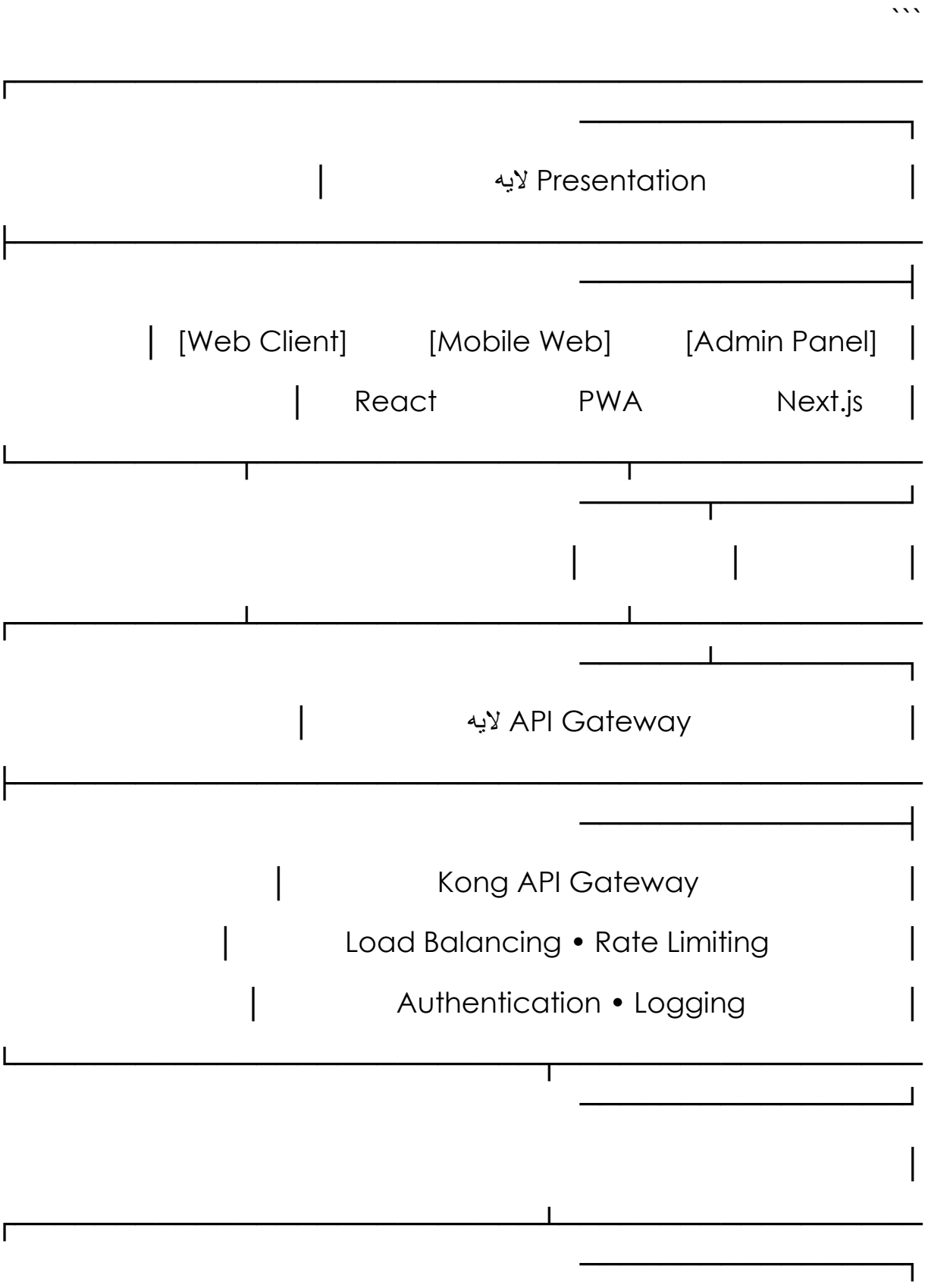
- **انعطاف‌پذیری بالا:** امکان توسعه مستقل سرویس‌ها
- **مقیاس‌پذیری انتخابی:** مقیاس هر سرویس بر اساس نیاز
- **تحمل خطا:** خرابی یک سرویس کل سیستم را از کار نمی‌اندازد
- **قابلیت نگهداشت:** کدبیس‌های کوچک‌تر و مستقل

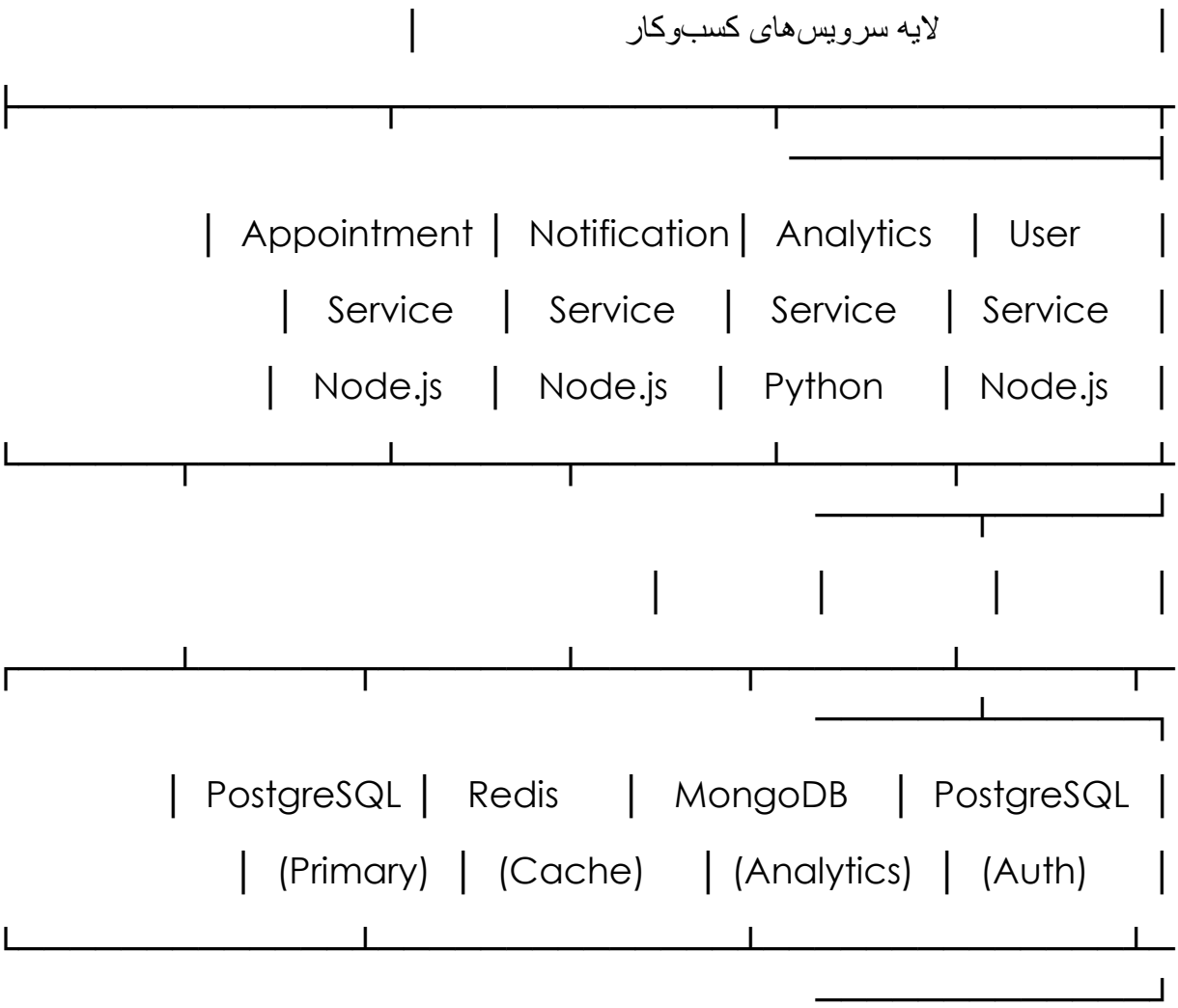
۱.۲ محدودیت‌های اصلی

- **زمان پاسخ:** زیر ۲ ثانیه برای عملیات کاربری
- **همزمانی:** پشتیبانی از ۱۰۰ کاربر همزمان در هر مطب
- **دسترسی:** ۹۹٫۵٪ uptime در ماه
- **امنیت:** رمزنگاری end-to-end برای داده‌های حساس

---

۲. دیاگرام معماری کلی


لایه Presentation
[Web Client] [Mobile Web] [Admin Panel]
React PWA Next.js
لایه API Gateway
Kong API Gateway
Load Balancing • Rate Limiting
Authentication • Logging




```

---

## **۳. جریان داده**

### **۳٫۱. جریان ثبت نوبت جدید**

```

بیمار → منشی (تلفنی/حضوری) .۱

(ثبت اطلاعات) Frontend → منشی .۲

۳. Frontend → API Gateway (POST /api/appointments)

۴. API Gateway → Appointment Service

۵. Appointment Service:

اعتبارسنجی داده‌ها -

PostgreSQL ذخیره در -

"Event "AppointmentCreated انتشار -

(ارسال پیامک تأیید) Event Bus → Notification Service .۶

(ثبت در سیستم تحلیل) Event Bus → Analytics Service .۷

منشی → بیمار → Frontend → پاسخ .۸

```

**جریان مشاهده وضعیت لحظه‌ای .۳/۲** ###

```

(مشاهده وضعیت) Frontend → بیمار .۱

۲. Frontend → API Gateway (GET /api/status/{clinicId})

۳. API Gateway → Cache Layer (بررسی Redis)

بازگشت داده → Cache Hit اگر .۴

۵. اگر Cache Miss:

(دریافت داده تازه) Appointment Service -

(ثانیه 30 :TTL) Redis ذخیره در -

بازگشت داده -

۶. Frontend → نمایش به بیمار

```

---

## ۴. **مرزبندی سرویس‌ها**

### ۴٫۱. Appointment Service (سرویس نوبت‌دهی)**

**مسئولیت:** مدیریت کامل نوبت‌ها

**تکنولوژی:** Node.js + Express + TypeScript

**دیتابیس:** PostgreSQL

**APIهای اصلی:**

- `POST /appointments` - ثبت نوبت جدید
- `GET /appointments/{id}` - دریافت نوبت
- `PUT /appointments/{id}` - به‌روزرسانی نوبت
- `DELETE /appointments/{id}` - حذف نوبت
- `GET /clinics/{id}/status` - وضعیت لحظه‌ای مطب

**محدودیت‌های Context Boundary:**

- فقط با User Service برای احراز هویت تعامل دارد
- داده‌های مالی را مدیریت نمی‌کند
```

مسئول ارسال پیامک نیست -

۴٫۲. Notification Service (سرویس اطلاع‌رسانی)

مسئولیت:** ارسال تمامی اطلاع‌رسانی‌ها**

تکنولوژی:** Node.js + Bull Queue**

دیتابیس:** Redis (برای Queue)**

سرویس‌های خارجی:** سرویس پیامک ایرانی**

انواع اطلاع‌رسانی:**

۱. **پیامک:** برای بیماران (تأیید، یادآوری، تأخیر)**

۲. **ایمیل:** برای گزارش‌های پزشکان**

۳. **Push Notification:** (اپلیکیشن موبایل) برای آینده

۴٫۳. User Service (سرویس کاربران)

مسئولیت:** مدیریت کاربران و احراز هویت**

تکنولوژی:** Node.js + JWT**

دیتابیس:** PostgreSQL**

ویژگی‌ها:**

- ثبت‌نام و ورود
- مدیریت نقش‌ها (پزشک، منشی، ادمین)
- مدیریت دسترسی (RBAC)
- Session Management

۴٫۴. Analytics Service (سرویس تحلیل)

مسئولیت:** جمع‌آوری و تحلیل داده‌ها**

تکنولوژی:** Python + FastAPI**

دیتابیس:** MongoDB (برای داده‌های تحلیلی)**

داده‌های تحلیل‌شده:**

- الگوی مراجعه بیماران

- زمان‌های اوج شلوغی

- عملکرد منشی‌ها

- رضایت بیماران

۴٫۵. API Gateway

مسئولیت:** نقطه ورود واحد به سیستم**

تکنولوژی:** Kong

وظایف:**

- مسیریابی درخواست‌ها

- Rate Limiting

- احراز هویت مقدماتی

- Logging مرکزی

- Load Balancing

---

۵. انتخاب دیتابیس‌ها

۵.۱. PostgreSQL (دیتابیس اصلی)

موارد استفاده: Appointment Service, User Service

دلایل انتخاب:

1. **ACID Compliance:** برای داده‌های تراکنشی حیاتی
2. **پشتیبانی از JSON:** برای ذخیره داده‌های نیمه‌ساختاریافته
3. **امکان Replication و Sharding:** مقیاس‌پذیری
4. **منابع آموزشی و پشتیبانی فراوان:** جامعه بزرگ

طرح‌بندی نمونه (Appointments Table):

```sql
CREATE TABLE appointments (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
clinic_id UUID NOT NULL REFERENCES clinics(id),
patient_id UUID NOT NULL REFERENCES patients(id),
appointment_time TIMESTAMP NOT NULL,
status VARCHAR(20) NOT NULL CHECK (status IN ('pending', 'in-progress', 'completed', 'cancelled')),
estimated_duration INTEGER, -- in minutes
```

actual_duration INTEGER,

created_at TIMESTAMP DEFAULT NOW(),

updated_at TIMESTAMP DEFAULT NOW()

);

CREATE INDEX idx_appointments_clinic_date ON appointments(clinic_id, appointment_time);

CREATE INDEX idx_appointments_status ON appointments(status) WHERE status = 'pending';

```

### **۵٫۲. Redis (کش و صف)**

موارد استفاده:** کش وضعیت لحظه‌ای، صف پیامک**

**:دلایل انتخاب**

1. **سرعت بالا:** دسترسی زیر میلی‌ثانیه**

2. **ساختار داده‌ای متنوع:** Strings, Hashes, Lists, Sets

3. **قابلیت TTL:** انقضای خودکار داده‌های کش

4. **پایداری:** امکان persistence

**:استفاده‌های اصلی**

- کش وضعیت نوبت‌ها (TTL: 30 ثانیه)

- صف ارسال پیامک‌ها
```

- Session Storage

- Rate Limiting

۵٫۳. MongoDB (دیتابیس تحلیلی)

موارد استفاده: Analytics Service

دلایل انتخاب:

1. **طرح‌بندی پویا:** برای داده‌های تحلیلی متنوع
2. **پرس‌وجوهای پیچیده:** Aggregation Pipeline
3. **مقیاس‌پذیری افقی:** Sharding آسان
4. **ذخیره داده‌های حجیم:** مناسب برای لاگ‌ها

---

۶. API Design

۶٫۱. استانداردهای API

- **قالب:** RESTful با JSON
- **نسخه‌بندی:** در URL (`/api/v1/...`)
- **احراز هویت:** Bearer Token (JWT)
- **خطاها:** HTTP Status Codes + پیغام فارسی استاندارد

۶٫۲. APIهای اصلی

Appointment API (v1):

```yaml
POST /api/v1/appointments:
description: ثبت نوبت جدید
request:
clinic_id: UUID
patient_phone: string (11 digit)
appointment_time: ISO timestamp
notes: string (optional)
response:
201: Appointment created
400: Invalid input
401: Unauthorized

GET /api/v1/clinics/{clinicId}/status:
description: دریافت وضعیت لحظه‌ای مطب
response:
200: { current: number, queue: [], estimated_times: {} }
404: Clinic not found

PUT /api/v1/appointments/{id}/status:
```

description: تغییر وضعیت نوبت

request:

status: 'in-progress' | 'completed' | 'cancelled'

response:

200: Status updated

404: Appointment not found

```

#### **Notification API (v1):**

```yaml

POST /api/v1/notifications/sms:

description: ارسال پیامک

request:

phone: string

template: 'confirmation' | 'reminder' | 'delay'

variables: object

response:

202: Notification queued

GET /api/v1/notifications/status/{messageId}:

description: بررسی وضعیت ارسال

```
```

۶٫۳. Event-Driven APIs (Async)

Event Schema:

```typescript
interface BaseEvent {
event_id: string;
event_type: string;
timestamp: string;
source: string;
version: string;
}

interface AppointmentCreatedEvent extends BaseEvent {
event_type: 'appointment.created';
data: {
appointment_id: string;
clinic_id: string;
patient_phone: string;
appointment_time: string;
};
}
```

Event Consumers:

- Notification Service: برای ارسال پیامک

- Analytics Service: برای ثبت آماری

- Audit Service: برای ثبت تاریخچه

---

۷. امنیت (Authentication & Authorization)

۷٫۱. Authentication (احراز هویت)

الگو: JWT (JSON Web Tokens)

جریان:

```
۱. کاربر credentials می‌فرستد
۲. User Service اعتبارسنجی می‌کند
۳. JWT Token (اعتبار: ۲۴ ساعت) تولید می‌شود
۴. Token در Header ارسال می‌شود
۵. API Gateway token را validate می‌کند
```

Token Structure:

```json
{
"sub": "user_id",
"clinic_id": "clinic_uuid",
"role": "doctor | secretary | admin",
"permissions": ["appointments:create", "appointments:read"],
"iat": 1625097600,
"exp": 1625184000
}
```

۷٫۲. Authorization (مجوزدهی)

الگو: Role-Based Access Control (RBAC) + Attribute-Based

نقش‌ها و دسترسی‌ها:

- **پزشک:**
(همه) appointments:read -
(وضعیت) appointments:update -
reports:read -
settings:manage -

- **منشی:**

- appointments:create

- appointments:read (محدود)

- appointments:update (محدود)

- patients:manage

:بیمار -

- status:read (فقط خودش)

- appointments:cancel (فقط خودش)

امنیت داده‌ها ۷٫۳. ###

۱. **رمزنگاری در حال انتقال:** TLS 1.3

۲. **رمزنگاری در حالت ذخیره:** AES-256 برای داده‌های حساس

۳. **نمایش جزئی اطلاعات در لاگ‌ها:** ماسک‌کردن

۴. **اعتبارسنجی ورودی:** Validation سمت سرور

5 . **Rate Limiting:** جلوگیری از حملات Brute-force

---

۸. Observability (قابلیت مشاهده)

۸٫۱. Logging (ثبت رویداد)

ساختار لاگ‌ها:

```json
{
"timestamp": "2024-01-15T10:30:00Z",
"level": "INFO",
"service": "appointment-service",
"correlation_id": "corr-123",
"user_id": "user-456",
"clinic_id": "clinic-789",
"message": "Appointment created successfully",
"metadata": {
"appointment_id": "app-123",
"duration_ms": 120
}
}
```

سطوح لاگ:

- **ERROR:** خطاهای بحرانی (پیامک نشود)
- **WARN:** هشدارها (نیاز به توجه)
- **INFO:** اطلاعات عملیاتی
- **DEBUG:** اطلاعات دیباگ (فقط توسعه)

۸٫۲. Metrics (معیارها)

معیارهای کلیدی:

1. **Application Metrics:**

- `http_requests_total` (تعداد درخواست‌ها)

- `http_request_duration_seconds` (زمان پاسخ)

- `appointments_created_total` (نوبت‌های ثبت‌شده)

- `notifications_sent_total` (پیامک‌های ارسال‌شده)

2. **System Metrics:**

- CPU/Memory Usage

- Database Connection Pool

- Cache Hit Rate

- Queue Length

ابزارها: Prometheus برای جمع‌آوری، Grafana برای نمایش

۸٫۳. Tracing (ردیابی)

الگو: Distributed Tracing با OpenTelemetry

اطلاعات ردیابی:

- Trace ID (درخواست سطح بالا)

- Span ID (هر عملیات)

- Parent Span ID (ارتباط سلسله‌مراتبی)

- Timing Information

:مزایا

- شناسایی bottlenecks

- عیب‌یابی خطاها در معماری توزیع‌شده

- تحلیل عملکرد end-to-end

۸٫۴. Alerting (هشدارها)

:کانال‌های هشدار

- Slack (تیم توسعه)

- SMS (تیم عملیاتی)

- Email (مدیریت)

:هشدارهای حیاتی

1. **Uptime** فوراً زیر ٪۹۹:**

2. **۵xx خطاهای** در ۵ دقیقه بیش از ٪۱:**

3. **پاسخ API** در ۱۰ دقیقه بالای ۵ ثانیه:**

4. **حجم غیرعادی درخواست:** در ۲ دقیقه**

---

۹. برنامه مقیاس‌پذیری

۹٫۱. مقیاس‌پذیری عمودی (Vertical Scaling)

مرحله ۱ (شروع):

- سرورهای کوچک (۲ CPU, 4GB RAM)
- تک نمونه از هر سرویس
- Single PostgreSQL instance

مرحله ۲ (رشد اولیه):

- ارتقاء سرورها (۴ CPU, 8GB RAM)
- اضافه شدن Read Replicas برای PostgreSQL
- کش Redis جداگانه

۹٫۲. مقیاس‌پذیری افقی (Horizontal Scaling)

استراتژی:

۱. **Appointment Service:** مقیاس بر اساس تعداد مطب‌ها

۲. **Notification Service:** مقیاس بر اساس حجم پیامک

۳. **API Gateway:** مقیاس بر اساس ترافیک ورودی

الگوی مقیاس خودکار (Auto-scaling):

```yaml
metrics:
- type: cpu
```

target: 70%

- type: memory

target: 80%

- type: http_requests

target: 1000/minute

scaling:

min_replicas: 2

max_replicas: 10

cooldown: 300 seconds

```

### **۹/۳. مقیاس‌پذیری دیتابیس**

#### **PostgreSQL Scaling:**

۱. **Read Replicas:** برای عملیات خواندن

۲. **Connection Pooling:** PgBouncer

۳. **Sharding:** بر اساس `clinic_id`

۴. **Partitioning:** بر اساس تاریخ نوبت

#### **Redis Scaling:**

۱. **Redis Cluster:** برای توزیع داده
```

۲. **Replication:** Master-Slave

3 . **Persistence:** RDB + AOF

۹٫۴. برنامه ظرفیت (Capacity Planning)

فاز	تعداد مطب	کاربران همزمان	درخواست/دقیقه	پیکنمود معماری
فاز ۱**	۱۰۰	۱٬۰۰۰	۱۰۰	تک سرور، تک دیتابیس**
فاز ۲**	۵۰۰	۵٬۰۰۰	۵۰۰	میکروسرویس‌ها، کش**
فاز ۳**	۲٬۰۰۰	۲۰٬۰۰۰	۲٬۰۰۰	کلاستر، شاردینگ**
فاز ۴**	۱۰٬۰۰۰	۱۰۰٬۰۰۰	۱۰٬۰۰۰	چند منطقه، ** CDN

---

۱۰. تصمیم‌های معماری (ADRs)

۱۰/۱. ADR-001: انتخاب میکروسرویس به جای مونولیت

تاریخ: ۱۵/۰۱/۱۴۰۳**

وضعیت: پذیرفته شده**

زمینه: ساختار کلی سیستم**

تصمیم: استفاده از معماری میکروسرویس**

عواقب:

- توسعه موازی و مستقل ✅

- مقیاس‌پذیری انتخابی ✅

- تحمل خطا بهتر ✅

- پیچیدگی عملیاتی بیشتر ❌

- تاخیر شبکه بین سرویس‌ها ❌

۱۰/۲. ADR-002: استفاده از Event-Driven Architecture

تاریخ: ۲۰/۰۱/۱۴۰۳**

وضعیت: پذیرفته شده**

زمینه: ارتباط بین سرویس‌ها**

تصمیم: استفاده از الگوی Event-Driven**

عواقب:

- عدم وابستگی (Loose Coupling) ✅

- قابلیت توسعه آسان ✅

- بازیابی از خطا (Retry) ✅

- پیچیدگی در debugging ❌

- دشوار Events تضمین ترتیب ❌

۱۰/۳. ADR-003: انتخاب PostgreSQL به جای NoSQL

تاریخ: ۲۵/۰۱/۱۴۰۳**

وضعیت: پذیرفته شده**

زمینه:** دیتابیس اصلی**

برای داده‌های تراکنشی PostgreSQL **:تصمیم**

:عواقب

(ACID) تضمین یکپارچگی داده ✅ -

تراکنش‌های پیچیده ✅ -

جامعه بزرگ و پشتیبانی ✅ -

مقیاس‌پذیری افقی سخت‌تر ❌ -

نیاز به طرح‌بندی از پیش تعریف‌شده ❌ -

۱۰/۴. ADR-004: استفاده از JWT به جای Session-based Auth

تاریخ:** ۰۱/۰۲/۱۴۰۳**

وضعیت:** پذیرفته شده**

زمینه:** احراز هویت**

تصمیم:** استفاده از JSON Web Tokens**

:عواقب

بودن سرور Stateless ✅ -

مقیاس‌پذیری آسان ✅ -

مناسب برای میکروسرویس ✅ -

عدم امکان باطل کردن Token ❌ -

حجم بیشتر داده در هر درخواست ❌ -

۱۰/۵. ADR-005: استفاده از Kong به جای Nginx برای API Gateway

تاریخ:** ۰۵/۰۲/۱۴۰۳**

وضعیت:** پذیرفته شده**

زمینه:** مدیریت ترافیک ورودی**

تصمیم:** Kong API Gateway**

عواقب:**

- ✓ Plugin-based Architecture

- ✓ Observability داخلی

- ✓ مدیریت متمرکز

- ✗ منحنی یادگیری

- ✗ منابع سخت‌افزاری بیشتر

---

۱۱. استراتژی استقرار (Deployment)

۱۱/۱. محیط‌های استقرار

۱. **Development:** برای توسعه‌دهندگان

۲. **Staging:** شبیه‌سازی تولید

3 . **Production:** محیط واقعی

۱۱٫۲. استراتژی Deployment

الگو: Blue-Green Deployment

مزایا:

- صفر Downtime
- امکان Rollback سریع
- آسان A/B تست

فرآیند:

```
۱. استقرار نسخه جدید در محیط Green
۲. تست کامل در Green
۳. تغییر مسیر ترافیک از Blue به Green
۴. مانیتورینگ دقیق
۵. بازگشت به Blue در صورت مشکل:
```

۱۱٫۳. CI/CD Pipeline

```
Git Push → Build → Test → Dockerize →
Push to Registry → Deploy to Staging →
Automated Tests → Manual Approval →
Deploy to Production → Smoke Tests → Monitor
```

```

**:ابزارها**

- **CI/CD:** GitHub Actions

- **Container Registry:** Docker Hub (خصوصی)

- **Orchestration:** Kubernetes

- **Monitoring:** بعد از deploy

---

## ۱۲. **بازیابی از فاجعه (Disaster Recovery)**

### ۱۲٫۱. **استراتژی Backup**

**:فرکانس**

- **دیتابیس:** روزانه کامل + هر ۱ ساعت افزایشی**

- **فایل‌ها:** روزانه**

- **کانفیگ:** با هر تغییر**

**:نگهداری**

- روز اخیر روزانه ۷

- هفته اخیر هفتگی ۴

- ماه اخیر ماهانه ۱۲
```

۱۲٫۲. Recovery Point Objective (RPO)

- داده‌های حیاتی:** ۱ ساعت**

- داده‌های غیرحیاتی:** ۲۴ ساعت**

۱۲٫۳. Recovery Time Objective (RTO)

- سیستم حیاتی:** ۴ ساعت**

- سیستم غیرحیاتی:** ۲۴ ساعت**

۱۲٫۴. DR Plan

۱. **شناسایی:** سیستم‌های حیاتی اولویت‌بندی شوند**

۲. **Backup:** بازیابی از آخرین Backup سالم

۳. **راه‌اندازی DR Site:** استقرار در**

۴. **تست:** تست کامل عملکرد**

۵. **بازگشت به Primary Site:**

---

۱۳. ملاحظات هزینه

**۱۳٫۱. هزینه‌های سخت‌افزاری/ابر

ماه اول (۱۰۰ مطب):

سرورها: ۲ میلیون تومان -

دیتابیس: ۱ میلیون تومان -

ذخیره‌سازی: ۵۰۰ هزار تومان -

شبکه: ۵۰۰ هزار تومان -

جمع:** ۴ میلیون تومان ماهانه**

:ماه ششم (۵۰۰ مطب)

سرورها: ۸ میلیون تومان -

دیتابیس: ۳ میلیون تومان -

ذخیره‌سازی: ۱ میلیون تومان -

شبکه: ۱ میلیون تومان -

جمع:** ۱۳ میلیون تومان ماهانه**

بهینه‌سازی هزینه .۱۳/۲ ###

۱. **Auto-scaling:** کاهش هزینه در ساعات کم‌ترافیک

۲. **Spot Instances:** برای سرویس‌های غیرحیاتی

۳. **Caching:** کاهش بار دیتابیس

4 . **Compression:** کاهش حجم انتقال داده

ریسک‌های معماری .۱۴ ##

۱۴٫۱. ریسک‌های فنی

۱. **Latency** بین سرویس‌ها:** طراحی با در نظر گرفتن تاخیر

۲. **Data Consistency:** الگوهای مناسب مثل Saga Pattern

3 . **Service Discovery:** استفاده از Consul یا مشابه

4 . **Configuration Management:** سیستم متمرکز کانفیگ

۱۴٫۲. ریسک‌های عملیاتی

۱. **Monitoring Complexity:** ابزارهای یکپارچه

۲. **Deployment Complexity:** اتوماسیون کامل

3 . **Team Skills:** آموزش تیم

4 . **Vendor Lock-in:** استفاده از استانداردهای باز

۱۴٫۳. ریسک‌های کسب‌وکار

۱. **Over-engineering:** شروع ساده، توسعه تدریجی

2 . **Cost Overrun:** مانیتورینگ هزینه

3 . **Time to Market:** اولویت‌بندی ویژگی‌ها

4 . **Scalability Limits:** طراحی مقیاس‌پذیر از ابتدا

---

۱۵. نقشه راه فنی

۱۵.۱. فاز ۱: پایه (ماه ۱-۳)

- میکروسرویس‌های اصلی
- API Gateway پایه
- PostgreSQL دیتابیس
- سیستم لاگ‌گیری ساده

۱۵.۲. فاز ۲: مقیاس‌پذیری (ماه ۴-۶)

- Redis برای کش
- Read Replicas
- Auto-scaling
- Monitoring پیشرفته

۱۵.۳. فاز ۳: پیشرفته (ماه ۷-۱۲)

- Kubernetes
- Service Mesh
- Advanced Caching
- Multi-region

۱۵.۴. فاز ۴: بهینه‌سازی (ماه ۱۳-۱۸)

- Machine Learning برای پیش‌بینی

- Edge Computing
- Real-time Analytics
- Cost Optimization

جمع‌بندی ۱۶. ##

اصول طراحی کلیدی ۱۶/۱. ###

۱. **Loose Coupling:** سرویس‌ها مستقل باشند

۲. **High Cohesion:** هر سرویس یک مسئولیت مشخص

3. **Design for Failure:** assume failure

4. **Automate Everything:** از Infrastructure as Code

مزایای معماری انتخاب‌شده ۱۶/۲. ###

۱. **مقیاس‌پذیری:** رشد بدون بازنویسی

۲. **قابلیت نگهداشت:** توسعه و تست آسان

3. **تحمل خطا:** خرابی محدود می‌ماند

4. **انعطاف‌پذیری:** تغییر تکنولوژی‌ها آسان

چالش‌های پیش‌رو ۱۶/۳. ###

۱. **پیچیدگی عملیاتی:** نیاز به تیم قوی DevOps

یادگیری تیم:** آموزش معماری توزیع‌شده** .۲

اتصال شبکه:** وابستگی به کیفیت شبکه** . 3

اشکال‌زدایی:** نیاز به ابزارهای پیشرفته** . 4

معیارهای موفقیت معماری .۱۶٫۴ ###

کارایی:** زمان پاسخ زیر ۲ ثانیه** .۱

uptime قابلیت اطمینان:** ۹۹٫۵٪** .۲

مقیاس‌پذیری:** پشتیبانی از ۱۰۰۰٪ رشد** . 3

زیر ۱ ساعت deploy قابلیت نگهداشت:** زمان** . 4

این معماری به گونه‌ای طراحی شده که هزینه‌های آینده کنترل شود و سیستم در برابر رشد سریع **
**.مقاوم باشد. هر تصمیم معماری با توجه به مقیاس‌پذیری، هزینه، و قابلیت نگهداشت گرفته شده است