ShahnamehMap — سند ۴,۲: معماری سیستم

نسخه: ۱,۰

تاریخ: ۰۵/۰۸/۱۴۰۳

تهیه‌کننده: مدیر فنی (CTO)

وضعیت: تصویب شده

---

۱. **چکیده معماری**

معماری **ShahnamehMap** یک معماری **چندسرویسه مبتنی بر رویداد (Event-Driven Microservices)** با طراحی **ضدشکننده (Anti-Fragile)** است. هدف اصلی، **ایجاد یک پلتفرم پایدار، مقیاس‌پذیر و قابل نگهداری** است که بتواند از یک MVP کوچک به یک پلتفرم اجتماعی با هزاران کاربر همزمان رشد کند. تمرکز بر **جداسازی نگرانی‌ها (Separation of Concerns)** و **استقلال سرویس‌ها** است تا تیم‌های مختلف بتوانند به طور موازی روی بخش‌های مختلف کار کنند.

:اصول کلیدی

1. **سرویس‌های مستقل و قابل استقرار جداگانه**
2. **ذخیره‌سازی داده‌های مخصوص هر سرویس (Database per Service)**
3. **ارتباط غیرهمزمان (Async) از طریق صف پیام (Message Queue) برای عملیات زمان‌بر**
4. **به عنوان نقطه ورود واحد API Gateway**
5. **طراحی برای شکست (Design for Failure)**

---

۲. نمای کلی معماری (High-Level Architecture Diagram)

```
[ کاربران ] (مرورگر، اپ موبایل)
|
| HTTPS / WebSocket
```

v

[Cloudflare CDN & WAF] <-- امنیت لایه Edge، کش، تحویل محتوا

|

v

[API Gateway (Kong)] <-- نقطه ورود واحد، روتینگ، احراز هویت، ریت لیمیت

|

|----------|----------|----------|----------|

| | | |

v v v v

[Game سرویس] [User سرویس] [Auth سرویس] [Content سرویس]

(Node.js + Redis) (Node.js + PSQL) (Keycloak) (Node.js + Neo4j)

| | | |

| | | |

[Redis Pub/Sub] [PostgreSQL] [PostgreSQL] [Neo4j]

گراف) (Real-time بازی State) کاربران، توکن) (پروفایل، تنظیمات))
(دانش شاهنامه

|

|----------

| |

v v

[Analytics سرویس] [Notification سرویس]

(Node.js + ClickHouse) (Node.js)

| |

| |

[ClickHouse] [(RabbitMQ) صف پیام]

برای ایمیل/پوش) (لاگ‌ها، رویدادها))

```

---

**برای سناریوهای کلیدی (Data Flow) جریان داده .۳** ##
```

جریان: ورود کاربر و شروع یک بازی ۳,۱ ###

```mermaid

sequenceDiagram

کاربر User as شرکت کننده

مرورگر Browser as شرکت کننده

Gateway as API Gateway شرکتنده

Auth سرویس Auth as شرکتنده

User سرویس UserSvc as شرکتنده

Game سرویس GameSvc as شرکتنده

Redis as Redis شرکتنده

User->>Browser: وارد کردن نام کاربری/رمز

Browser->>Gateway: POST /auth/login

Gateway->>Auth: درخواست ورود

Auth->>Auth: اعتبارسنجی

Auth-->>Gateway: توکن JWT + اطلاعات پایه کاربر

Gateway-->>Browser: توکن JWT (در Cookie HttpOnly)
```

"کلیک روی "شروع بازی :User->>Browser

(با توکن) Browser->>Gateway: WS Upgrade

اعتبارسنجی توکن :Gateway->>Auth

تأیید :Auth-->>Gateway

کاربر WebSocket اتصال :Gateway->>GameSvc

Room Key ایجاد/پیوستن به :GameSvc->>Redis

اولیه بازی State تایید اتصال، ارسال :GameSvc-->>Browser

```

### ۳٫۲** جریان: ذخیره‌سازی یک کمپین کاربرساخته (Campaign Builder)**

```mermaid

sequenceDiagram

User as خالق شرکت کننده

Builder as ویرایشگر فرانت‌اند شرکتنده

Gateway as API Gateway شرکتنده

participant Content سرویس ContentSvc as شرکتنده

participant دیتابیس گراف Neo4j as شرکتنده

participant Queue as RabbitMQ شرکتنده

User->>Builder: کلیک "ذخیره"

Builder->>Gateway: POST /api/campaigns (با Payload JSON)

Gateway->>Auth: اعتبارسنجی توکن

Gateway->>ContentSvc: ارسال درخواست

ContentSvc->>ContentSvc: اعتبارسنجی ساختار داده

ContentSvc->>Neo4j: ذخیره نودهای کمپین، صحنه‌ها، ارتباطات

ContentSvc->>Queue: انتشار Event: CampaignSaved (Async)

ContentSvc-->>Gateway: پاسخ ۲۰۰ OK + ID کمپین

Gateway-->>Builder: تأیید ذخیره‌سازی

Note over Queue,Neo4j: پردازش‌های پس‌زمینه

برای پردازش Event ارسال :Queue->>NotificationSvc

آماده‌سازی اعلان (اختیاری) :NotificationSvc->>NotificationSvc

```

۴.** مرزبندی سرویس‌ها (Service Boundaries)**

سرویس**	**مسئولیت**	**تکنولوژی**	**دیتابیس**	**	**مقیاس‌پذیری**
API Gateway	روتینگ، Load Balancing، احراز اولیه، ریت لیمیت، کش درخواست‌های GET.	**Kong** (افقی**)	-	متن‌باز)
Authentication Service	احراز هویت (AuthN) و مدیریت کاربران، صدور توکن JWT، مدیریت سشن‌ها، مجوزها (AuthZ)	**Keycloak** (با حالت داخلی Keycloak)	**PostgreSQL**	(متن‌باز)	عمودی (با منابع بیشتر).
```

| **User Service** | مدیریت پروفایل کاربر، تنظیمات، کاراکترهای ذخیره‌شده، لیست دوستان. | **Node.js + Express** | **PostgreSQL** (جدول‌های user_profiles, user_characters) | Stateless. افقی |

| **Game Engine Service** | مدیریت State بازی Real-time، پردازش نوبت‌ها، اعمال قوانین، مدیریت اتاق‌ها. | **Node.js + Socket.io** | **Redis** (برای State اتاق‌ها، Pub/Sub) | **افقی بسیار مهم.** هر نمونه تعدادی اتاق را مدیریت می‌کند |

| **Content Service** | مدیریت محتوای هسته: نقشه، شخصیت‌ها، کمپین‌های رسمی و کاربری. | **Node.js + Express** | **MinIO** (برای فایل‌های آپلودشده) + **Neo4j** (گراف) | افقی |

| **Notification Service** | ارسال ایمیل، اعلان درون‌برنامه‌ای (Web Push). | **Node.js** | - | از صف می‌خواند) | افقی |

| **Analytics Service** | جمع‌آوری و پردازش رویدادها برای تحلیل. | **Node.js** | **ClickHouse** (برای ذخیره‌سازی و کوئری سریع) | افقی |

---

۵. انتخاب دیتابیس‌ها و دلیل (ADR - Architectural Decision Record)

ADR-001: استفاده از PostgreSQL و برای داده‌های تراکنشی و کاربر

* **وضعیت:** پذیرفته‌شده**

* **زمینه:** نیاز به ذخیره‌سازی داده‌های ساختاریافته، با ارتباطات پیچیده و نیاز به قابلیت اطمینان بالا (مانند اطلاعات کاربر، تراکنش‌های پرداخت).

* **تصمیم:** استفاده از **PostgreSQL**.

* **دلایل:**

1. **ACID-Compliant:** برای داده‌های حساس کاربر و مالی ضروری است.
2. **انعطاف‌پذیری:** پشتیبانی از JSONB در کنار اسکیماهای رابطه‌ای، نیاز ما به ذخیره تنظیمات پویای کاراکترها را به خوبی برآورده می‌کند.
3. **جامعه و ابزار:** جامعه بزرگ، ابزارهای مانیتورینگ و مدیریت **成熟**.

عواقب:** نیاز به مدیریت شاردینگ در صورت رشد بسیار زیاد جدول ** *
.کاربران (پس از ۱۰+ میلیون رکورد)

ADR-002: استفاده از Neo4j برای داده‌های هسته شاهنامه

وضعیت:** پذیرفته‌شده** *

زمانه:** داده‌های هسته ما (شخصیت‌ها، مکان‌ها، رویدادها) ذاتاً ** *
گرافی هستند. پرسش‌هایی مانند "تمام شخصیت‌های متولدشده در سیستان
.که با رستم جنگیدند" در یک دیتابیس رابطه‌ای پیچیده و کند است

.**Neo4j تصمیم:** استفاده از **دیتابیس گراف** *

:دلایل *

1. مدل طبیعی:** داده‌ها به صورت نود (موجودیت) و رابطه (ارتباط) **
.ذخیره می‌شوند که دقیقاً منطبق بر مدل ذهنی ما است

2. های عمیق و Joکارایی پرس‌وجوهای رابطه‌ای:** اجرای پرس‌و**
.(O(1)) چندمرحله‌ای به سرعت ثابت

3. ساخت ویژگی "کشف **:(Discoverability) قابلیت کشف**
.ساده می‌شود Cypher ارتباطات پنهان" در نقشه با کوئری‌های

دارد. عملیات **Cypher** عواقب:** تیم نیاز به یادگیری زبان** *
نیاز به Neo4j در (High Write Throughput) نوشتن حجم بالا
.تنظیمات خاص دارد

ADR-003: استفاده از Redis برای State بازی Real-time

وضعیت:** پذیرفته‌شده** *

* **زمینه:** State بسیار کم‌تأخیر (اتاق) باید با **اشتراکی** بین چندین نمونه از **(Low Latency)
.در دسترس باشد Game سرویس

* تصمیم:** استفاده از** Redis** به عنوان **حافظه موقت
.**Pub/Sub** و پشت (Shared Cache) اشتراکی

* **دلایل:**

1. **عملکرد بی‌نظیر در حافظه:** تأخیر زیر میلی‌ثانیه**.

2. **ساختمان داده‌های غنی:** استفاده از** Hash** برای ذخیره
یک اتاق، **Sorted Sets** برای مدیریت نوبت، State
یک WebSocket برای انتشار رویدادها به همه اتصالات **Pub/Sub**
.اتاق

به صورت Persistence قابلیت اطمینان:** پشتیبانی از** 3.
.اختیاری

در صورت عدم) ممکن است از بین برود Redis عواقب:** داده‌های** *
مهم بازی به طور دوره‌ای در State باید .(RDB/AOF پیکربندی صحیح
نیز پشتیبان‌گیری شود (PostgreSQL) دیتابیس پایدار
(Checkpointing).

ADR-004: استفاده از ClickHouse برای Analytics

وضعیت:** پذیرفته‌شده** *

زمینه:** نیاز به ذخیره و تحلیل میلیاردها رویداد کاربری (کلیک، ** *
.حرکت در بازی، خرید) با کوئری‌های جمع‌بندی سریع

.**ClickHouse** تصمیم:** استفاده از** *

:دلایل *

1. **OLAP:** بهینه‌شده برای** سرعت کوئری‌های تحلیلی بر روی
.داده‌های حجیم بی‌نظیر است

2. **فشرده‌سازی موثر:** کاهش چشمگیر هزینه ذخیره‌سازی رویدادها**.

3. **قابلیت مقیاس افقی**.

استاندارد. برای عملیات SQL عواقب:** عدم سازگاری کامل با** *
.(Append-Only الگوی) مناسب نیست Update/Delete

---

۶. API Design & Contracts

استراتژی کلی: REST برای CRUD، GraphQL برای Query های پیچیده، WebSocket برای Real-time.

* **REST API (برای مدیریت):** `/api/v1/campaigns` ، `/api/v1/characters`

* برای **OpenAPI Specification (Swagger)** از مستندسازی استفاده می‌شود.

* **GraphQL Endpoint:** `/graphql` (برای فرانت‌اند نقشه و داشبوردهای تحلیلی)

* دلیل:** جلوگیری از Over-fetching/Under-fetching داده‌های پیچیده گراف شاهنامه.

* **WebSocket:** `/ws` (برای اتصال بازی)

* **پیام‌های ساختاریافته:** پروتکل پیام سفارشی با فیلدهای JSON `type` و `payload`.

مثال: REST Endpoint برای ایجاد کاراکتر**

```
POST /api/v1/characters
Headers: Authorization: Bearer <JWT>
Body (application/json):
{
"name": "آرش کمانگیر",
"race": "ایرانی",
"class": "تیرانداز",
"attributes": {
"strength": 12,
"dexterity": 18,
"wisdom": 10
}
}
```

Response 201 Created:

{

"id": "char_abc123",

سایر فیلدها // ...

}

```

---

## **۷. امنیت (AuthN/AuthZ)**

### **احراز هویت (Authentication - AuthN):**

* **روش:** **OAuth 2.0 / OpenID Connect (OIDC)** با Identity Provider (IdP) به عنوان **Keycloak** استفاده از.

* **جریان:** **Authorization Code Flow with PKCE** برای برنامه‌های فرانت‌اند عمومی (SPA).
```

توکن‌ها کوتاه‌عمر .**(JSON Web Tokens)JWT** **:توکن** *
.(دسترسی: ۱۵ دقیقه، رفرش: ۷ روز) هستند

ذخیره‌سازی در فرانت:** توکن دسترسی در **حافظه** *
.**HttpOnly Secure Cookie** ، توکن رفرش در**(Memory)

مدیریت مجوز (Authorization - AuthZ):

* **روش:** **Role-Based Access Control (RBAC)** با
.سطح‌بندی دقیق

* **سطح‌های کلیدی:**

* `user`: کاربر عادی (می‌تواند بازی کند، کاراکتر بسازد).

* `creator`: کاربر تاییدشده (می‌تواند کمپین بسازد و منتشر کند).

* `teacher`: کاربر سازمانی (به پنل مدرسه و کمپین‌های آموزشی
.دسترسی دارد)

* `admin`: مدیر سیستم.

* **اعمال مجوز:** در سطح** (**API Gateway**
.و در سطح **هر سرویس داخلی** (برای کنترل دقیق‌تر) (JWT نقش در

:امنیت داده‌ها ###

تأیید هویت سرویس‌به‌سرویس:** استفاده از **توکن‌های سرویس** *
برای ارتباطات داخلی بین سرویس‌ها در یک شبکه خصوصی **(mTLS)
.(Kubernetes مثلاً در)

هش **bcrypt** رمزنگاری داده‌های حساس:** تمام گذرواژه‌ها با** *
(At Rest) می‌شوند. داده‌های حساس (مانند ایمیل) در حال استراحت
.رمزنگاری می‌شوند

و مانیتورینگ Observability .۸ ##

و **عیب‌یابی سریع **(Traceability) هدف: **قابلیت ردیابی کامل
.**(Quick Debugging)

:Observability سه ستون ###

:(با ساختار - Logs) لاگینگ 1.

.Node.js در **Pino** یا **Winston** **:فریمورک** *

.**JSON Structured Logs** **:قالب** *

سبک‌وزن، مناسب برای) **Loki** آگروگیشن:** ارسال به** *
.**Elastic Stack** متن) یا

Correlation ID اطلاعات همراه:** هر لاگ شامل** *
.است serviceName، userId، (ردیابی یک درخواست در همه سرویس‌ها)

(Metrics): متریک‌ها** 2.

.(ذخیره‌سازی و جمع‌آوری) **Prometheus** **:استاندارد** *

، نرخ خطا، تأخیرHTTP معیارهای کلیدی:** نرخ درخواست** *
فعال، WebSocket ، وضعیت سلامت سرویس‌ها، تعداد اتصالات(latency)
.RAM/CPU مصرف

.**Grafana** **:تصویرسازی** *

(Distributed Tracing): ردیابی توزیع‌شده 3.

.**Zipkin** یا **Jaeger** **:ابزار** *

هدف:** مشاهده مسیر یک درخواست کاربر (مثلاً "شروع بازی")** *
.و برگشت Redis، Game تا سرویس Gateway از

Health Checks & Alerting:

* هر سرویس یک endpoint `/health` دارد که وضعیتش را گزارش می‌دهد.

* **Alerting:** قوانین هشدار در Prometheus** تنظیم می‌شود (مثلاً اگر خطای ۵xx بیش از ۵٪ شد، Alertmanager** اعلان‌ها به Telegram/Email** ارسال می‌شوند. (از دسترس رفت Game یا اگر سرویس

---

۹. **برنامه مقیاس‌پذیری (Scalability Plan)**

افقی (Horizontal Scaling)**:

* **استاتeless سرویس‌ها (User, Content, API Gateway)**: افزودن نمونه‌های بیشتر پشت یک Load Balancer.

* **Stateful سرویس‌ها (Game Engine)**: چالش اصلی. استراتژی:

1. **اتاق‌بندی (Sharding by Room ID):** یک اتاق بازی خاص همیشه به یک نمونه خاص از سرویس Game هدایت می‌شود. این با روتینگ مبتنی بر اتاق در Gateway** انجام می‌پذیرد.

2. **Redis به عنوان Source of Truth:** State هر اتاق در Redis ذخیره می‌شود، بنابراین اگر یک نمونه Game از کار بیفتد، نمونه دیگری می‌تواند با خواندن State از Redis آن اتاق را 接管 (Take Over) کند.

3. **Auto-scaling:** بر اساس معیار تعداد اتصالات WebSocket در هر نمونه، نمونه‌های جدید سرویس Game فعال می‌شوند.

عمودی (Vertical Scaling):**

* در بلندمدت، شاردینگ (افزایش RAM/CPU) برای دیتابیس‌ها در ابتدا Redis و Neo4j و کلاستر برای PostgreSQL برای (Sharding).

مقیاس‌پذیری داده:**

* **PostgreSQL:** شاردینگ با استفاده از **Citus** یا **Vitess** زمانی که جدول کاربران از ۱۰ میلیون رکورد گذر کرد.

.به طور ذاتی مقیاس‌پذیر افقی است **ClickHouse:** *

مانند) Object Storage** فایل‌های آپلودشده:** استفاده از** *
.که به طور نامحدود مقیاس می‌پذیرد **(S3-Compatible یا MinIO

:(سال اول) Capacity Planning برنامه ###

.(CCU) حداکثر بار پیش‌بینی‌شده:** ۱۰،۰۰۰ کاربر همزمان** *

مورد نیاز:** هر نمونه می‌تواند Game تعداد نمونه‌های سرویس** *
پایدار را مدیریت کند. ← نیاز به **-۱۵ WebSocket ~۷۰۰-۵۰۰ اتصال
.**۲۰ نمونه

پهنای باند:** هر اتصال بازی ~۱۰-۵ کیلوبیت بر ثانیه داده رد و بدل ** *
.می‌کند. ← کل پهنای باند: ~۲۰۰-۱۰۰ مگابیت بر ثانیه

برای ۱ میلیارد رویداد ~ ۲۰۰ ClickHouse **:ذخیره‌سازی** *
.گیگابایت فضا نیاز دارد

DevOps استقرار و ۱۰. ##

* **Terraform** **(IaC):** زیرساخت به عنوان کد (DB، سرور، شبکه،) منابع کلاد.

* **کانتینری‌سازی:** **Docker**.

* **ارکستراسیون:** **Kubernetes (K8s)** (با استفاده از یک سرویس managed مانند **EKS** یا **GKE**).

* **CI/CD:** خط لوله **GitLab CI/CD** یا **GitHub Actions**.

* اسکن امنیتی → Build Docker Image → تست خودکار → استقرار در Stage → تست پذیرش → استقرار در محیط Production (Rolling Update).

* **مدیریت پیکربندی:** **Helm Charts** برای برنامه‌های K8s.

---

۱۱. **استراتژی Rollback و بازیابی از فاجعه (DR)**

* **Rollback:** در صورت وجود باگ در نسخه جدید، امکان بازگشت سریع به نسخه قبلی از طریق Helm/K8s وجود دارد.

:(Backup) پشتیبان‌گیری *

پشتیبان روزانه + پشتیبان لحظه‌ای پیوسته **:PostgreSQL** * (WAL).

.پشتیبان روزانه **:Neo4j** *

.فعال است (Versioning) نسخه‌بندی **:Object Storage** *

امکان راه‌اندازی کل **:(Disaster Recovery) راه‌اندازی مجدد** * دیگر با استفاده از اسکریپت‌های (Region) سیستم در یک منطقه با (RTO=4h) **و آخرین پشتیبان‌ها ظرف ۴** ساعت Terraform (RPO=1h). حداکثر از دست دادن داده‌های ۱ ساعته

(هاADR) تصمیم‌های معماری کلیدی .۱۲ ##

برای عملیات پس‌زمینه Event-Driven انتخاب :ADR-005 ###

Message به عنوان **RabbitMQ** تصمیم:** استفاده از** * برای عملیات غیرهمزمان (مانند ارسال ایمیل، پردازش رویدادهای Broker .تحلیل، به‌روزرسانی کش)

دلیل:** جداسازی سرویس‌ها و افزایش تحمل خطا. اگر سرویس** *
Notification از کار بیفتد، پیام‌ها در صف می‌مانند تا زمانی که مجدداً
فعال شود. همچنین امکان افزودن سرویس‌های جدید به راحتی با گوش دادن
.به رویدادها وجود دارد

**ADR-006: استفاده از API Gateway به جای Service Mesh
**(در ابتدا)

* تصمیم:** شروع با** Kong** به عنوان API Gateway و به
.(Istio مانند) کامل Service Mesh تعویق انداختن پیاده‌سازی یک

* دلیل:** در مراحل اولیه، پیچیدگی** Service Mesh (و منابع مورد
نیازهای روتینگ، احراز هویت و ریت لیمیت Kong .نیاز) توجیه‌پذیر نیست
مانند) را به خوبی برآورده می‌کند. در صورت نیاز به قابلیت‌های پیشرفته‌تر
، می‌توان بعداً به(بین سرویس‌های داخلی circuit breaker، retry
.مهاجرت کرد Service Mesh

**ADR-007: استفاده از WebSocket خام + پروتکل سفارشی به
**Socket.io Cluster جای

با استفاده از **WebSocket** تصمیم:** پیاده‌سازی مستقیم** *
، JSON-based و طراحی یک پروتکل سبک Node.js در `ws` کتابخانه
.(Adapter) با حالت کلاسر Socket.io به جای استفاده از

در حالت کلاسر برای توزیع پیام‌ها بین Socket.io **:دلیل** *
وابسته است. با حذف این (Redis مثل) Adapter نمونه‌ها، همچنان به یک
و Gateway لایه انتزاعی و کنترل مستقیم بر روتینگ پیام‌ها از طریق
عملکرد بهینه‌تر و قابل پیش‌بینی‌تری** خواهیم **، Pub/Sub Redis
داشت. این تصمیم مستلزم نوشتن کد مدیریت اتصال بیشتر است اما در
.بلندمدت کنترل بهتری می‌دهد

جمع‌بندی: کنترل هزینه آینده ##

این معماری با سرمایه‌گذاری اولیه بیشتر در **طراحی ماژولار** و
انتخاب ابزارهای مناسب برای مقیاس، هزینه‌های آینده را کنترل
:می‌کند

* **هزینه تغییر (Change Cost):** افزودن یک سرویس جدید (مثلاً "سیستم دوستی و چت") به دلیل مستقل بودن سرویس‌ها، آسان و کم‌ریسک است.

* **هزینه مقیاس (Scaling Cost):** هر بخش را (مثلاً فقط سرویس) می‌توان بر اساس نیازش به طور مستقل مقیاس داد (را برای رویداد خاصی مقیاس بالا برد Game).

* **هزینه خطا (Failure Cost):** شکست یک سرویس (مثلاً Content) به دلیل جداسازی، باعث از کار افتادن کل سیستم (مانند بازی Real-time) نمی‌شود.

این معماری تضمین می‌کند که **سیستم در رُشد نمی‌ترکد** و می‌تواند همراه با کسب‌وکار ShahnamehMap بلوغ یابد.