ShahnamehMap — سند ۴٫۸: برنامه عملکرد و مقیاس‌پذیری

نسخه: ۱٫۰

تاریخ: ۰۱/۰۹/۱۴۰۳

تهیه‌کننده: مدیر فنی (CTO)

وضعیت: فعال

---

۱. اهداف سطح سرویس (Service Level Objectives - SLOs)

SLOها مبنای سیستم است. این **قابلیت اطمینان (Reliability)** و **کارایی (Performance)** های ما تعهد داخلی برای تصمیم‌گیری‌های مهندسی و تجاری هستند.

هدف	شاخص سطح سرویس (SLI)	سطح سرویس (SLO)	توضیح

درخواست‌های موفق | **API Gateway (همه سرویس‌های REST)** | نسبت به کل درخواست‌ها. | **۹۹٫۵٪** در هر دوره (HTTP 2xx/3xx) | (در ماه Downtime ساعت ۳٫۵~) ۲۸ روزه. | ۰٫۵٪ خطای مجاز |

اتصالات | **Game Engine (اتصالات Real-time) سرویس** | پایدار (بدون قطعی غیرعمدی) نسبت به کل اتصالات. | WebSocket Real-time **۹۹٪** در هر دوره ۲۸ روزه. | تحمل ۱٪ قطعی برای سخت‌گیرانه). |

برای (P95) های اصلی** | زمان پاسخAPI (Latency) تأخیر** | endpoint`POST /api/characters`
• های حیاتی:
• `GET /api/campaigns` | P95 یعنی P95. | برای **ms زیر ۲۰۰** | ۹۵٪ درخواست‌ها در این زمان پاسخ می‌گیرند. |

زمان بین | **Game Action Latency) تأخیر عملیات بازی** | State توسط همه Action توسط کاربر و دریافت به‌روزرسانی ارسال یک برای تجربه بازی | P99. برای **ms زیر ۵۰۰** | (P99) بازیکنان اتاق روان حیاتی است. |

درصد زمان قابل | **Overall Uptime) در دسترس بودن کلی** | دسترسی بودن سرویس از دید کاربر نهایی. | **۹۹٪** در هر دوره ۳ ماهه. | معیار سطح بالای سلامت کسب‌وکار. |

، بودجه خطای ما در ٪99.5 SLO با **:(Error Budget) بودجه خطا**
ماه ٪۰,۵ (۴۳,۲ دقیقه) است. اگر در هفته اول ماه ٪۰,۴ خطا مصرف کنیم،
تیم موظف است توسعه فیچرهای جدید را متوقف کند و تمام تمرکز را
.بر بهبود پایداری بگذارد

(Capacity Planning) برنامه اندازه‌گیری و ظرفیت‌سنجی .۲ ##

(Key Performance معیارهای کلیدی عملکرد .۲,۱** ###
Metrics)**
:مانیتور می‌کنیم **Real-time** ما این متریک‌ها را به صورت

| متریک** | **هدف** | **ابزار مانیتورینگ** | **هشدار** |
(Alert)
API Gateway	-	به **(RPS) نرخ درخواست**
.افزایش ناگهانی > ٪۵۰	**Grafana** + **Prometheus**

| **CPU Utilization** (سرویس Game Engine) | < میانگین | برای ۸۰٪ < | Node Exporter + K8s Metrics | ۶۰٪، پیک < ۸۵٪ بیش از ۵ دقیقه. |

| **Memory Utilization** (سرویس Game Engine) | < ۷۰٪ | Node Exporter + K8s Metrics | > ۸۵٪. |

| **Redis INFO command** + Prometheus | نزدیک شدن به حد نظری هر نمونه (~۷۰۰). | - | **فعال WebSocket اتصالات** |

| **PostgreSQL Connections** | < ۸۰٪ از `max_connections` | **pg_stat_activity** + Prometheus | > ۹۰٪. |

| **Redis Latency** (P99) | < ۵ms | **Redis Monitoring** | > ۱۰ms. |

۲,۲. مدل ظرفیت (Capacity Model) و نقاط گلوگاه شناخته شده

ما بر اساس **کاربر فعال ماهانه (MAU)** و **کاربر فعال همزمان (CCU)** برنامه‌ریزی می‌کنیم.

:فرضیات مدل

* **WebSocket** اتصال ۱** ≈ **(CCU)** هر **کاربر فعال همزمان + **REST** درخواست ۵~ در دقیقه.

* **نسبت** CCU به MAU (۱:۲۰ ≈ **:پیک) CCU ≈ ۱۰،۰۰۰ یعنی ۲۰۰،۰۰۰ MAU).

گلوگاه شناخته‌شده**	**ظرفیت فعلی**	**(Resource) منبع**	راهکار **	**Scale-out** حد آستانه برای**	**(CCU برای ۱۰۰۰)	**مقیاس‌پذیری
CPU	**Game Engine (Node.js)** نمونه سرویس**	هر اتاق.	۱ نمونه = **State** و **حافظه** برای **(Node.js) **تک‌هسته ~۷۰۰ اتصال پایدار.	**۶۰۰ اتصال** در هر نمونه.	**مقیاس افقی:** اضافه کردن نمونه‌های جدید. **روتینگ مبتنی بر **RoomID** در **API** Gateway.
Redis (State بازی + Pub/Sub) **	پهنای باند شبکه** و **CPU > ۷۰٪** یا **Latency > ۵ms**.	۱. **تأخیر** در صورت فشار بالا.	نسخه فعلی: ۱ نمونه **Redis** ارتقا به کلاستر**

به RoomID شاردینگ اتاق‌ها** بر اساس** .۲
.(Redis Cluster)
چندین نمونه Redis. |

| **PostgreSQL** (تعداد** | Connection داده کاربر/کاراکتر)
| ۱ نمونه | همزمان** و **پرفورمنس کوئری‌های گزارش‌گیری.** |
| **Connection Pool > ۸۰٪** ۲۰۰ > کوئری‌های کند** یا **ms**. |
۱. **اتصال‌پول (Connection Pooling)** با PgBouncer.
۲.
۳.
برای کوئری‌های گزارش **(Read Replica)** خواندنی/نوشتنی**
شاردینگ** طولی‌مدت** (با Citus). |

| **API Gateway (Kong)** | **مقدار حافظه (RAM)** برای
کش و تعداد اتصالات همزمان. | ۲ نمونه پشت Load Balancer. |
مقیاس افقی** نمونه‌های** | **CPU > ۷۵٪**. Kong. |

| Real-time** (WebSocket). | ~۱۰۰ > **استفاده** | پهنای باند شبکه** (Network Bandwidth)** | ترافیک**
ارتقای پلن شبکه با ارائه‌دهنده کلاد | ۷۰٪**. |

---

۳. برنامه تست بار (Load Testing Plan)

.**هدف: **شناسایی محدودیت‌های سیستم قبل از مواجهه واقعی

۳,۱. محیط و ابزارها:

* **ابزار:** **k6** (به دلیل سادگی، اسکریپت‌نویسی با JS و یکپارچگی با Grafana).

* **محیط:** **Staging** که دقیقاً مشابه Production است (اما با نمونه‌های کوچک‌تر).

* **داده تست:** از **دیتابیس Production (anonimized)** برای تست واقعی‌تر استفاده می‌شود.

۳,۲. سناریوهای تست بار:

هر تست **۳ مرحله** دارد: **رَمپ آپ (Ramp-up)**، **حفظ پیک (Sustain)**، **رَمپ داون (Ramp-down)**.

سناریو	**هدف**	**پروفایل بار**	**معیارهای موفقیت**

| **LT-۱: تست | ثبت‌نام و ساخت کاراکتر انبوه** | ۵۰ VU → ۵۰۰ VU در ۵ دقیقه. حفظ ۵ دقیقه. | • نرخ خطای HTTP < ۱٪.
• P95 Latency < ۳۰۰ms.
• PostgreSQL CPU < ۸۰٪. |

| **LT-۲: تست سرویس | Real-time** شبیه‌سازی کاربران بازی پیچیده:** ایجاد ۲۰۰ "اتاق بازی" ** | Redis. | Game Engine و هر بازیکن هر ۱۰ ثانیه (CCU مجموعاً ۸۰۰) "مجازی، هر کدام ۴ "بازیکن می‌فرستد. | • نرخ قطعی WebSocket < ۰٫۵٪.
• Redis Latency < ۱۰ms.
• Game Engine CPU < ۷۰٪. | Action یک Game Action Latency (P99) < ۷۰۰ms.
• نمونه‌های

| **LT-۳: تست | API Gateway و مقیاس خودکار. | افزایش سریع از ۱۰۰ به ۱۵۰۰ VU در ۲ دقیقه. | • API Gateway Latency (P95) < ۲۵۰ms.
• Auto-scaling سرویس‌ها ظرف ۳ دقیقه فعال شود. | حفظ ۳ دقیقه. | پیک ترافیک (برگشت از کمپین مارکتینگ)**

| **LT-۴: استقامت (Endurance Test)** | شناسایی memory leak یا Degradation در طول زمان. | بار ثابت معادل ۵۰٪ از ظرفیت پیش‌بینی‌شده پیک برای **۲ ساعت**. | • هیچ‌گونه افزایش تدریجی Latency ۲۰٪ یا Memory Utilization بیش از. |

۳,۳. فرکانس اجرا:**

* **(اجباری) پس از هر تغییر عمده معماری.

* **(برای اطمینان از حفظ عملکرد) ماهانه.

* **(اجباری) قبل از رویدادهای بزرگ بازاریابی.

---

۴. برنامه مقیاس‌پذیری (Scalability Plan)**

۴,۱. مقیاس‌پذیری برنامه (Application Scaling)**

* **Auto-scaling** (استاتلس‌ها، Gateway، API - Stateless): افقی بر اساس CPU/Memory Utilization. scaling

* قانون K8s HPA:** اگر CPU > ۷۰٪ به مدت ۱ دقیقه، `scale up` اگر CPU < ۳۰٪ به مدت ۵ دقیقه. `scale down`

* **Auto-scaling** (استیت‌فول - Stateful - Game Engine): افقی با هوشمندی.

* چالش: هر نمونه، State اتاق‌های خاصی را در حافظه دارد.

راهکار:** استفاده از **شاخص** *
زمانی که میانگین اتصالات هر نمونه از .**`connections_per_pod` ۶۰۰ گذر کند، نمونه جدید ایجاد می‌شود. **اتاق‌های جدید** به نمونه‌های State شود، اتاق‌هایش بر اساس Fail کم‌بارتر هدایت می‌شوند. اگر نمونه‌ای .، به نمونه‌های دیگر مهاجرت می‌کنندRedis ذخیره‌شده در

۴,۲. مقیاس‌پذیری داده (Data Scaling)

* **مسیر تکامل PostgreSQL:**:

۱. **(هم‌اکنون) Connection Pooling با PgBouncer.

۲. **برای Read Replica** افزودن **(MAU > ۵۰،۰۰۰ در)** .کوئری‌های گزارش و تحلیل

۳. **بر (Sharding) شاردینگ** **(MAU > ۵۰۰،۰۰۰ در)** .**Citus** با استفاده از `user_id` اساس

* **مسیر تکامل Redis:**:

۱. **(هم‌اکنون)** یک نمونه قوی.

۲. **برای Redis Cluster** راه‌اندازی **(CCU > ۵،۰۰۰ در)** .توزیع داده و بار

* **ClickHouse:** به طور ذاتی برای بارهای تحلیلی سنگین مقیاس‌پذیر افقی است.

۴,۳. **مقیاس‌پذیری شبکه و CDN**

* **CDN:** استفاده از **Cloudflare** برای ارائه asset های استاتیک (تصاویر، JS، CSS). با رشد ترافیک، پلن‌های بالاتر خریداری می‌شود.

* **کلاد ارائه‌دهنده با شبکه پلن ارتقای :** پهنای باند.

---

۵. **برآورد هزینه رشد (Cost Projection)**

ما هزینه‌ها را بر اساس **مدل مصرف منابع در پلتفرم کلاد داخلی** (مثلاً دیجی‌کالا کلاد) برآورد می‌کنیم.

| سناریو کاربری** | **منابع مورد نیاز** | **هزینه ماهانه ** |
| **توضیح** | **تخمینی** |

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

پایه (سال اول) MAU: ۲۰K, CCU (۱پیک): ۴ • نمونه API/سایر ۲ • (هسته، ۲GB) • Game Engine (۲ هسته، ۲GB) • Redis (۲GB RAM) • PostgreSQL (۲GB) • ۲۰۰ GBپهنای باند: ۱۲-۸ میلیون تومان**	MVP

مقیاس متوسط (سال دوم) MAU: ۲۰۰K, CCU (۱۰پیک): ۱۵ • نمونه API/سایر • ۶ نمونه Game Engine • Redis Cluster (۳ Node) • PostgreSQL + Read Replica (۸GB) • ۲ TBپهنای باند: ۶۰-۴۰ میلیون تومان**	مرحله رشد سریع.

مقیاس بزرگ (سال سوم/بین‌المللی) MAU: ۱M, CCU (۵۰پیک): ۷۵ • نمونه API/سایر • ۲۰ نمونه Game Engine • Redis Cluster بزرگ • PostgreSQL Sharded (Citus) • ۱۰ TB+پهنای باند: ۳۵۰-۲۰۰ میلیون تومان** + هزینه کلاد بین‌المللی برای دیاسپورا.	پلتفرم پایدار و سودآور.

نکته:** این هزینه‌ها **فقط زیرساخت** است. هزینه‌های نیروی انسانی، مارکتینگ و سایر عملیات جداگانه است.

۶.** جمع‌بندی: آیا سیستم از پا درمی‌آید؟**

:بر اساس این برنامه، پاسخ **خیر** است — اگر و تنها اگر

۱. **ما هشدارها را به موقع شناسایی کند** مانیتورینگ فعال.

۲. **گلوگاه‌های جدید را قبل از بروز مشکل آشکار** تست‌های بار منظم کنند.

۳. **Auto-scaling Rules** به درستی پیکربندی شده باشند.

۴. **ظرفیت مالی برای ارتقای منابع در نقطه** (Inflection Point) وجود داشته باشد.

بزرگ‌ترین ریسک‌های مقیاس‌پذیری:

* **گلوگاه Stateful:** مشکل اصلی ما **مدیریت State** بازی Real-time است. راهکار شاردینگ Redis در مقیاس ده‌ها هزار کاربر همزمان است. راهکار شاردینگ Game Engine و معماری بر این چالش متمرکز است.

* **رشد نمایی:** هزینه CCU می‌تواند هزینه زیرساخت را به طور خطی افزایش دهد. **مدل درآمدی ما باید به گونه‌ای باشد که (Freemium) کاربر پولی، هزینه زیرساخت کاربران رایگان را پوشش دهد LTV.**

این سند نقشه راهی است که تضمین می‌کند **مهندسی همگام با رشد کسب‌وکار حرکت می‌کند**. ما نه تنها می‌دانیم سیستم امروز چگونه کار می‌کند، بلکه می‌دانیم فردا با دوبرابر شدن کاربران، چه اقداماتی باید انجام دهیم و این اقدامات چقدر هزینه خواهد داشت. این **قابلیت پیش‌بینی**، کلید رشد کنترل‌شده و بدون شوک است.