4.8 performance & scalibility plans

برنامه‌های عملکرد و مقیاس‌پذیری - ProDecks
نسخه: 1.0
تاریخ: 1403/11/16
وضعیت: MVP - برنامه‌ریزی شده

فصل ۱: مرور کلی

۱.۱. هدف این سند

در مواجهه با رشد کاربران ProDecks تعیین استراتژی‌ها و برنامه‌های لازم برای تضمین عملکرد بهینه و مقیاس‌پذیری سیستم و داده‌ها.

۱.۲. چالش‌های پیش‌رو

• رشد تصاعدی تعداد کاربران و داده‌ها
• افزایش انتظارات کاربران از زمان پاسخ‌گویی
• نیاز به دسترسی‌پذیری بالا (High Availability)
• حفظ هزینه‌های زیرساخت در سطح منطقی

۱.۳. اهداف کلیدی

• زمان پاسخ API: ۹۵٪ درخواست‌ها زیر ۲۰۰ms برای
• زمان لود صفحات: زیر ۳ ثانیه
• دسترسی‌پذیری (Uptime): ۹۹.۹٪
• پشتیبانی از ۱۰,۰۰۰ کاربر همزمان
• مقیاس‌پذیری خطی هزینه با رشد کاربران

فصل ۲: معیارهای عملکرد (Performance Metrics)

۲.۱. معیارهای کاربردی (Application Metrics)

• API Response Time:
- P50: < ۱۰۰ms
- P95: < ۲۰۰ms
- P99: < ۵۰۰ms

• Page Load Time:
- First Contentful Paint: < ۱.۵s

- Largest Contentful Paint: < ۲.۵s
- Time to Interactive: < ۳.۵s

• Transaction Rate:
- کارت ایجاد/آپدیت: ۵۰ تراکنش/ثانیه
- عملیات: ۱۰۰ تراکنش/ثانیه Drag & Drop

۲.۲. معیارهای زیرساختی (Infrastructure Metrics)

• CPU Utilization: حداکثر ۷۰٪
• Memory Utilization: حداکثر ۸۰٪
• Disk I/O: زیر ۱۰۰ms latency
• Network Throughput: مطابق با الگوی ترافیک

۲.۳. معیارهای پایگاه داده (Database Metrics)

• Query Execution Time:
- ساده: < ۱۰ms
- پیچیده: < ۱۰۰ms
• Connection Pool Utilization: حداکثر ۸۰٪
• Replication Lag: < ۵ ثانیه

۲.۴. معیارهای کسب‌وکار (Business Metrics)

• User Satisfaction Score: > ۴.۵/۵
• Task Completion Rate: > ۹۵٪
• System Availability: ۹۹.۹٪

فصل ۳: استراتژی‌های بهینه‌سازی عملکرد

۳.۱. بهینه‌سازی فرانت‌اند

۱. Bundling و Minification:
• ترکیب فایل‌های JavaScript و CSS
• فشرده‌سازی تصاویر
• استفاده از CDN برای فایل‌های استاتیک

۲. Lazy Loading:
• بارگذاری تصاویر هنگام نیاز
• تقسیم کد JavaScript
• به تعویق انداختن بارگذاری ماژول‌های غیرضروری

۳. Caching مرورگر:
• تنظیم مناسب Cache-Control headers
• استفاده از ETags

• Service Workers برای offline capability

۳.۲. بهینه‌سازی بک‌اند

:لایه برنامه Caching .۱

• برای داده‌های پرتکرار Redis

• session storage برای Memcached

• cache قابل API responses برای CDN

:۲. Query Optimization

• ایندکس‌گذاری بهینه

• کوئری‌های بهینه‌شده

• pagination استفاده از

:۳. Connection Pooling

• مدیریت کارآمد اتصالات دیتابیس

• connection leaks جلوگیری از

• بر اساس بار pool تنظیم اندازه

۳.۳. بهینه‌سازی پایگاه داده

:۱. Indexing Strategy

• ایندکس‌های ترکیبی برای کوئری‌های پرتکرار

• حذف ایندکس‌های غیرضروری

• covering indexes استفاده از

:۲. Partitioning

• بر اساس تاریخ ایجاد users جدول

• space_id بر اساس cards جدول

• پس از رسیدن به حد آستانه horizontal partitioning

:۳. Query Optimization

• برای تحلیل کوئری‌ها EXPLAIN استفاده از

• N+1 queries جلوگیری از

• برای گزارش‌ها materialized views استفاده از

فصل ۴: استراتژی‌های مقیاس‌پذیری

۴.۱. مقیاس‌پذیری عمودی (Vertical Scaling)

• فاز ۱ (MVP): سرورهای متوسط (۸GB RAM, ۴ vCPU)

• فاز ۲ (رشد اولیه): سرورهای بزرگ (۱۶GB RAM, ۸ vCPU)

• فاز ۳ (مقیاس بزرگ): سرورهای اختصاصی (۳۲GB RAM, ۱۶ vCPU)

۴.۲. مقیاس‌پذیری افقی (Horizontal Scaling)

۱. Web Tier:

• Load Balancer (NGINX/HAProxy)
• گروه سرورهای وب Auto-scaling
• طراحی برای Stateless ease of scaling

۲. Database Tier:

• Primary-Secondary Replication
• Read Replicas برای بار خواندنی
• Sharding در صورت نیاز

۳. Cache Tier:

• Redis Cluster
• Memcached با چندین نود

۴.۳. الگوهای معماری مقیاس‌پذیر

۱. Microservices (آینده):

• تفکیک سرویس‌های اصلی
• API Gateway
• Service Discovery

۲. Event-Driven Architecture:

• Message Queue (RabbitMQ/Kafka)
• Event Sourcing برای برخی موجودیت‌ها
• CQRS برای جداسازی read/write

۴.۴. Database Scaling Plan

• تا ۱۰,۰۰۰ کاربر: Single Master + ۲ Read Replicas
• ۱۰,۰۰۰ تا ۵۰,۰۰۰ کاربر: Master-Master + ۴ Read Replicas
• بیش از ۵۰,۰۰۰ کاربر: Sharding بر اساس space_id

فصل ۵: معماری مقیاس‌پذیر

۵.۱. معماری فعلی (MVP)

[CDN] → [Load Balancer] → [Web Servers] → [Database Master]
↓
[Database Replica]

۵.۲. معماری فاز ۲ (۱۰,۰۰۰ کاربر)

[CDN] → [Load Balancer] → [Web Servers] → [API Gateway] → [Microservices]
↓
[Message Queue] → [Background Workers]
↓

[Database Cluster]
(Master + ۳ Replicas)

معماری فاز ۳ (۵۰,۰۰۰ کاربر) .۵.۳

[Global CDN] → [Multiple Load Balancers] → [Multiple Regions]
↓
[Central API Gateway] → [Service Mesh] → [Microservices per Domain]
↓
[Distributed Cache] → [Sharded Databases] → [Data Lake]

فصل ۶: برنامه‌ریزی ظرفیت (Capacity Planning)

پیش‌بینی رشد .۶.۱

- ماه‌های ۱-۳: ۱,۰۰۰ کاربر فعال
- ماه‌های ۴-۶: ۵,۰۰۰ کاربر فعال
- ماه‌های ۷-۱۲: ۱۰,۰۰۰ کاربر فعال
- سال دوم: ۵۰,۰۰۰ کاربر فعال

نیازمندی‌های منابع .۶.۲

- برای ۱,۰۰۰ کاربر:
  - Web Servers: ۲ نود (۴GB RAM, ۲ vCPU)
  - Database: ۱ نود (۸GB RAM, ۴ vCPU)
  - Cache: ۱ نود Redis (۲GB RAM)

- برای ۱۰,۰۰۰ کاربر:
  - Web Servers: ۴-۶ نود (۸GB RAM, ۴ vCPU)
  - Database: Master + ۲ Replicas (۱۶GB RAM, ۸ vCPU)
  - Cache: Redis Cluster (۳ ۴نود، هرکدام GB)

- برای ۵۰,۰۰۰ کاربر:
  - Web Servers: ۱۰-۱۵ نود (۱۶GB RAM, ۸ vCPU)
  - Database: Sharded (۸ shards)
  - Cache: Redis Cluster (۶ ۸نود، هرکدام GB)

هزینه‌های زیرساخت پیش‌بینی شده .۶.۳

- ماه‌های ۱-۳: ۵ میلیون تومان ماهانه
- ماه‌های ۴-۶: ۱۵ میلیون تومان ماهانه
- ماه‌های ۷-۱۲: ۳۰ میلیون تومان ماهانه
- سال دوم: ۱۰۰ میلیون تومان ماهانه

فصل ۷: تست‌های عملکرد و مقیاس‌پذیری

۷.۱. انواع تست‌ها

۱. Load Testing:

- شبیه‌سازی کاربران واقعی
- افزایش تدریجی بار
- شناسایی bottlenecks

۲. Stress Testing:

- بار بیش از حد ظرفیت
- شناسایی نقاط شکست
- تعیین حد نهایی سیستم

۳. Endurance Testing:

- تست طولانی مدت (۲۴-۷۲ ساعت)
- شناسایی memory leaks
- بررسی stability

۴. Spike Testing:

- افزایش ناگهانی بار
- شبیه‌سازی رویدادهای خاص
- تست resilience

۷.۲. ابزارهای تست

- Load Testing: Apache JMeter, k6
- Monitoring: Prometheus, Grafana
- APM: New Relic, Datadog
- Database: pt-query-digest, Percona Toolkit

۷.۳. سناریوهای تست

سناریو ۱: ایجاد کارت

- کاربران همزمان: ۱۰۰ → ۵۰۰ → ۱,۰۰۰
- فرکانس: ۱۰ کارت/دقیقه به ازای هر کاربر
- معیارها: زمان پاسخ، throughput، error rate

سناریو ۲: Drag & Drop عملیات

- کاربران همزمان: ۵۰ → ۲۰۰ → ۵۰۰
- عملیات: جابجایی ۵ کارت/دقیقه به ازای هر کاربر
- معیارها: real-time performance، consistency

سناریو ۳: گزارش‌گیری

- کاربران همزمان: ۱۰ → ۵۰ → ۱۰۰
- گزارش‌ها: پیچیده با joins زیاد
- معیارها: زمان اجرای کوئری، مصرف منابع

فصل ۸: مانیتورینگ و هشدار

۸.۱. معیارهای مانیتورینگ

• Application Level:
- Response times
- Error rates
- Throughput
- Business transactions

• Infrastructure Level:
- CPU/Memory/Disk usage
- Network I/O
- Service availability

• Database Level:
- Query performance
- Connection counts
- Replication lag
- Lock contention

۸.۲. تنظیمات هشدار

• Critical (PagerDuty/SMS):
- Uptime < ۹۹٪
- Error rate > ۵٪
- Database down

• Warning (Email/Slack):
- Response time P95 > ۵۰۰ms
- CPU usage > ۸۰٪ به مدت ۵ دقیقه
- Disk usage > ۸۵٪

• Informational (Dashboard):
- Performance degradation
- Unusual patterns
- Capacity thresholds

۸.۳. dashboards

• Real-time Dashboard: وضعیت لحظه‌ای
• Business Dashboard: معیارهای کسب‌وکار
• Performance Dashboard: روند عملکرد
• Capacity Dashboard: مصرف منابع و trends

فصل ۹: برنامه مقیاس خودکار (Auto-scaling)

۹.۱. قوانین Auto-scaling

• Scale Out:
- CPU utilization > ۷۰٪ به مدت ۵ دقیقه
- Memory utilization > ۸۰٪ به مدت ۵ دقیقه
- Response time P95 > ۴۰۰ms به مدت ۱۰ دقیقه

• Scale In:
- CPU utilization < ۳۰٪ به مدت ۱۵ دقیقه
- Memory utilization < ۴۰٪ به مدت ۱۵ دقیقه

۹.۲. زمان‌بندی Scaling

• Scheduled Scaling:
- ساعت اوج مصرف (۹AM-۵PM): ظرفیت +۵۰٪
- ساعت شب (۱۲AM-۶AM): ظرفیت -۵۰٪
- آخر هفته: ظرفیت -۳۰٪

• Event-based Scaling:
- رویدادهای بازاریابی: +۱۰۰٪ ظرفیت
- انتشار feature جدید: +۵۰٪ ظرفیت

۹.۳. استراتژی Scaling

• Conservative: افزایش ۱ نود در هر scaling
• Aggressive: افزایش ۲-۳ نود در scaling events
• Predictive: استفاده از machine learning برای پیش‌بینی بار

فصل ۱۰: بهینه‌سازی هزینه

۱۰.۱. استراتژی‌های کاهش هزینه

۱. استفاده از Spot Instances:
• برای workloadهای قابل تحمل خطا
• صرفه‌جویی تا ۷۰٪
• همراه با on-demand instances برای stability

۲. Right-sizing:
• مانیتورینگ مداوم مصرف منابع
• تنظیم اندازه instances
• حذف منابع بلااستفاده

۳. Reserved Instances:
• برای workloadهای پایدار
• صرفه‌جویی تا ۴۰٪
• برنامه‌ریزی بلندمدت

۱۰.۲. معماری هزینه‌بهینه

- استفاده از CDN برای کاهش بار origin
- caching aggressive برای کاهش بار دیتابیس
- استفاده از object storage برای داده‌های قدیمی ارزان
- compression داده‌ها در انتقال و ذخیره‌سازی

۱۰.۳. مدیریت چرخه حیات داده

- انتقال داده‌های قدیمی به storage ارزان‌تر
- archiving پروژه‌های کامل شده
- حذف خودکار داده‌های موقت
- retention policies بر اساس ارزش داده

فصل ۱۱: Disaster Recovery و Business Continuity

۱۱.۱. استراتژی Backup

- Backup داده‌ها: هر ساعت incremental + روزانه
- Backup پیکربندی: versioning خودکار
- تست بازیابی: ماهانه

۱۱.۲. استراتژی Recovery

- Recovery Time Objective (RTO): ۴ ساعت
- Recovery Point Objective (RPO): ۱ ساعت
- Regional Failover: در صورت قطعی region اصلی

۱۱.۳. طراحی High Availability

- Multi-AZ deployment برای سرویس‌های حیاتی
- Load balancing across zones
- Automated failover mechanisms

فصل ۱۲: نقشه راه مقیاس‌پذیری

۱۲.۱. فاز ۱ (ماه‌های ۱-۳): پایه‌های مقیاس‌پذیری

- caching لایه پیاده‌سازی
- بهینه‌سازی کوئری‌های اصلی
- monitoring پایه تنظیم
- load اولیه تست

۱۲.۲. فاز ۲ (ماه‌های ۴-۶): مقیاس‌پذیری اولیه

• پیاده‌سازی read replicas
• auto-scaling برای web tier
• بهبود معماری برای statelessness
• تست‌های پیشرفته performance

۱۲.۳. فاز ۳ (ماه‌های ۷-۹): مقیاس‌پذیری پیشرفته

• تفکیک به microservices
• پیاده‌سازی message queue
• طراحی database sharding
• راه‌اندازی global CDN

۱۲.۴. فاز ۴ (ماه‌های ۱۰-۱۲): مقیاس‌پذیری enterprise

• multi-region deployment
• advanced caching strategies
• predictive auto-scaling
• cost optimization پیشرفته

فصل ۱۳: تیم و مسئولیت‌ها

۱۳.۱. نقش‌های کلیدی

• مسئول بهینه‌سازی :Performance Engineer
• مسئول زیرساخت و scaling :DevOps Engineer
• مسئول مقیاس‌پذیری دیتابیس :DBA
• مسئول reliability و monitoring :SRE

۱۳.۲. فرآیندها

• Weekly Performance Reviews
• جلسات ماهانه Capacity Planning
• Load Testing قبل از هر release اصلی
• Performance Budgeting برای featureهای جدید

۱۳.۳. آموزش و توسعه

• آموزش best practices مقیاس‌پذیری
• knowledge sharing sessions
• مشارکت در communities
• تحقیق در مورد تکنولوژی‌های جدید

فصل ۱۴: ریسک‌ها و mitigations

۱۴.۱. ریسک‌های فنی

• ریسک: Database becoming bottleneck
• Mitigation: Read replicas، caching، query optimization

• ریسک: Single point of failure
• Mitigation: Multi-AZ، failover automation

• ریسک: Cost overrun
• Mitigation: Cost monitoring، right-sizing، reserved instances

۱۴.۲. ریسک‌های عملیاتی

• ریسک: Lack of monitoring
• Mitigation: Comprehensive monitoring، alerting

• ریسک: Insufficient testing
• Mitigation: Regular load testing، chaos engineering

• ریسک: Knowledge silos
• Mitigation: Documentation، cross-training

۱۴.۳. ریسک‌های کسب‌وکار

• ریسک: Performance affecting user retention
• Mitigation: Performance SLAs، proactive monitoring

• ریسک: Inability to handle growth spikes
• Mitigation: Auto-scaling، capacity buffer

• ریسک: Competitive disadvantage
• Mitigation: Continuous performance improvement

فصل ۱۵: نتیجه‌گیری

یک رویکرد جامع و مرحله‌ای برای تضمین ProDecks برنامه عملکرد و مقیاس‌پذیری تجربه کاربری بهینه در تمام مراحل رشد کسب‌وکار ارائه می‌دهد. با ترکیب بهینه‌سازی‌های فنی، استراتژی‌های مقیاس‌پذیری هوشمند و فرآیندهای عملیاتی دقیق، ما آمادگی لازم برای رشد تصاعدی را خواهیم داشت.

- تمرکز ما نه تنها روی مقیاس‌پذیری فنی، بلکه روی مقیاس‌پذیری اقتصادی نیز هست. حفظ کیفیت سرویس در حالی که هزینه‌ها به صورت خطی با رشد افزایش می‌یابند.

ضمیمه‌ها

ضمیمه ۱: نتایج تست‌های load اولیه

معماری مقیاس‌پذیر diagrams :ضمیمه ۲

Capacity Planning Models :ضمیمه ۳

Alerting Configurations و Monitoring :ضمیمه ۴

:تهیه‌کنندگان

حامد کوهی (معمار فنی)

DevOps ProDecks تیم

مشاوران زیرساخت
