مقیاس‌پذیری سرویس کوتاه‌کننده لینک

این سند به بررسی راهکارهای مقیاس‌پذیر کردن سرویس کوتاه‌کننده URL که با FastAPI و PostgreSQL پیاده‌سازی شده است، می‌پردازد. هدف، حفظ عملکرد مناسب سرویس در شرایط بار بالا، جلوگیری از ایجاد نقاط تک‌خرابی (Single Point of Failure) و امکان گسترش افقی سیستم روی چندین سرور است.

جداسازی لاگ‌گیری سنگین از مسیر اصلی درخواست‌ها

هنگامی که حجم لاگ‌گیری برای هر درخواست افزایش یابد و نیاز باشد اطلاعات (مانند IP کاربر، زمان دسترسی و کد کوتاه) به صورت لحظه‌ای به یک سرویس جانبی یا دیتابیس جداگانه ارسال شود، اجرای این عملیات به صورت Synchronous می‌تواند زمان پاسخ‌دهی درخواست‌های اصلی (به‌ویژه ریدایرکت) را به شدت افزایش دهد.

برای جلوگیری از این مشکل، پیشنهاد می‌شود لاگ‌گیری را از مسیر بحرانی درخواست جدا کنیم. راهکار پیشنهادی استفاده از یک **Message Queue** مانند Kafka یا RabbitMQ است. در Middleware FastAPI، اطلاعات لاگ به صورت غیرهمزمان (Asynchronous) به یک تاپیک در کافکا ارسال می‌شود. این کار باعث می‌شود پاسخ به کاربر (HTTP 302 Status Code) بدون انتظار برای ذخیره لاگ برگردانده شود.

در سمت دیگر، Workerهای Celery (با Redis به عنوان Broker) پیام‌ها را از صف دریافت کرده و پردازش می‌کنند. این Workerها می‌توانند داده‌ها را تجمیع کرده و در یک دیتابیس تحلیلی جداگانه (مثلاً PostgreSQL شاردشده یا ClickHouse) ذخیره کنند. برای کاهش بار بیشتر، می‌توان لاگ‌ها را به صورت دسته‌ای (Batch) ارسال کرد؛ مثلاً هر ۱۰۰ رکورد یا هر ۵۰۰ میلی‌ثانیه یک بار.

در صورت بروز مشکل در صف (Back-Pressure)، با استفاده از Circuit Breaker می‌توان لاگ‌های غیرحیاتی را به طور موقت دور انداخت تا از تأثیر منفی بر سرویس اصلی جلوگیری شود. این رویکرد زمان پاسخ‌دهی عملیات اصلی را زیر ۱۰۰ میلی‌ثانیه نگه می‌دارد و با اصول Observability (مانند Tracing با OpenTelemetry) نیز سازگار است.

آماده‌سازی سیستم برای استقرار چند-نمونه (Multi-Instance)

اگر قرار باشد سرویس روی چندین سرور به صورت همزمان اجرا شود، باید از حالت تک‌نمونه (Single-Instance) به معماری توزیع‌شده تغییر کنیم.

ابتدا یک **load balancer** (مانند NGINX) در مقابل نمونه‌های FastAPI قرار می‌گیرد تا ترافیک را بین آن‌ها تقسیم کند. در مرحله بعد، تمام وابستگی‌های دارای حالت (Stateful) باید **External** شوند:

- **دیتابیس**: به جای PostgreSQL محلی، از یک کلاستر مدیریت‌شده با قابلیت Replication و Read Replica استفاده می‌شود. برای مقیاس‌پذیری بهتر نوشتن، می‌توان داده‌ها را بر اساس Hash کد کوتاه شارد کرد.
- **کش**: Redis به صورت یک سرویس خارجی (ترجیحاً Redis Cluster) برای کش کردن ریدایرکت‌های پرتکرار و شمارش بازدیدها به کار گرفته می‌شود تا از مشکلات هم‌زمانی جلوگیری شود.
- **صف‌ها و لاگ‌گیری**: Kafka یا RabbitMQ نیز به صورت کلاستر مستقر می‌شوند و Worker‌های Celery به طور مستقل مقیاس‌پذیر خواهند بود.

برای تولید کد کوتاه بدون تداخل (Collision) در محیط توزیع‌شده، می‌توان از یک **Key Generation Service** با هماهنگی Zookeeper استفاده کرد یا کلیدها را به صورت پیش‌تولید در Redis ذخیره نمود.

ریسک‌های مهم شامل موارد زیر است:

- احتمال ایجاد کدهای تکراری → با استفاده از عملیات اتمیک در Redis (مانند INCR) یا قفل Zookeeper مدیریت می‌شود.
- خرابی یک نمونه از دیتابیس یا Redis → با Replication و Auto-Failover برطرف می‌گردد.
- افزایش تأخیر شبکه → با مانیتورینگ Tracing و در صورت نیاز استفاده از gRPC قابل کاهش است.

مقابله با ترافیک سنگین ناشی از کمپین تبلیغاتی

در شرایطی که یک کمپین تبلیغاتی باعث ورود چندین هزار درخواست در ثانیه شود، باید ترکیبی از تصمیم‌های موجود و اقدامات جدید را به کار بگیریم تا سرویس از دسترس خارج نشود.

در طراحی فعلی از **Connection Pooling** در SQLAlchemy و **Index** روی `short_code` و افزایش اتمیک تعداد بازدیدها استفاده شده است که پایه خوبی برای عملکرد فراهم می‌کند. برای تحمل بار بسیار بالا:

- **لایه کش قوی**: با Redis برای کدهای پربازدید (Hot Keys) اضافه می‌شود. طبق اصل Pareto، معمولاً ۲۰٪ کدها ۸۰٪ ترافیک را تشکیل می‌دهند؛ کش کردن این موارد می‌تواند بار دیتابیس را به شدت کاهش دهد.
- **Rate Limiting**: با استفاده از `slowapi` در Middleware اعمال می‌شود تا از سوءاستفاده جلوگیری کند و در صورت لزوم IPهای کمپین را Whitelist کنیم.
- **CDN**: (مانند Cloudflare) برای کش کردن پاسخ‌های 302 در لبه شبکه به کار گرفته می‌شود تا بخش زیادی از ترافیک از سرورهای اصلی عبور نکند.
- **مانیتورینگ پیشرفته**: با Prometheus، Grafana و Tracing با Jaeger برای شناسایی سریع گلوگاه‌ها و تنظیم آلارم (مثلاً وقتی اتصالات دیتابیس بیش از ۸۰٪ ظرفیت Pool شود).
- در شرایط بحرانی، **Graceful Degradation** اعمال می‌شود: مثلاً در صورت فشار زیاد روی دیتابیس،

آمار بازدید از کش قدیمی ارائه شود.

منابع الهام‌بخش

- System Design : Scalable URL shortener service like TinyURL
- a simple link shortener with FastAPI