سلام مهندس! خوشحالم که پای کاری 😎 .

بشـین تا نقشـه راه جدید رو برات باز کنم. این بار یه مقدار فنی‌تر، دقیق‌تر و البته با دید بیزینسـی (Business-Oriented) به قضیه نگاه می‌کنیم. ما فقط کد نمی‌زنیم، داریم یه مشکل اساسی شرکت رو حل می‌کنیم:

هزینه زیرساخت و سرعت دسترسی به دانش.

ما می‌خوایم یه **موتور جستجوی مقیاس‌پذیر (Scalable)** بسازیم که بتونه روی سخت‌افزار محدود هم پرواز کنه.

🏢سناریوی تجاری(Business Case)

ببین، سـرورهای ما دارن زیر بار دیتای ویکی‌پدیا ناله می‌کنن. ابزارهای آماده مثل Elasticsearch عالین، اما رم (RAM) رو مثل آب می‌خورن.

هدف بیزینسـی ما اینه :یک موتور جسـتجوی سـفارشـی (Custom) داشـته باشـیم که برای دیتای متنی و گراف ویکی‌پدیا بهینه شده باشه. ما می‌خوایم هزینه سرورها رو ۳۰٪ کم کنیم و سرعت پاسخ‌دهی به کوئری‌های پیچیده سریع نگه داریم.

تو قراره "معمار سیستم" باشی. یعنی تست کنی، بنچمارک بگیری و بگی:

" آقای مدیر فنی، برای این حجم دیتا، فلان ساختار داده (Data Structure) بهترین بازدهی رو داره".

🏗️معماری و زمین بازی(Infrastructure)

اینجا همه چیز باید **Cloud-Native**و تمیز باشه. چرا داکر؟ چون می‌خوایم محیط تست ایزوله باشه. اگه یه الگوریتم رم سرور رو پر کرد و کرش کرد، کل سیستم نخوابه.

ما ۳ تا کامپوننت اصلی داریم:

۱. **دیتای مشـترک(Data Volume)** : ⚠️ نکته مهم: دیتا رو داخل کانتینر کپی نمی‌کنیم (این کار اشـتباهه و ایمیج رو سنگین می‌کنه). ما از **Docker Volume**استفاده می‌کنیم تا فایل‌های حجیم ویکی‌پدیا فقط یک بار روی دیسک باشن و کانتینرها (مثل فلش مموری) بهش وصل بشن و بخونن.

۲. **آزمایشگاه(Methods Container)** : این کانتینریه که کدهای تو; که میتونه پایتون، Go، یا C++ توش اجرا میشه. هر الگوریتم رو اینجا ایمپلیمنت می‌کنی.

۳. **ارکستراتور(The Orchestrator)** : این "مغز متفکر" سیستمه. یه سرویس مثلاً با **(FastAPI)**که:

- به کانتینر متد فرمان میده: "الآن الگوریتم X رو روی دیتای Y اجرا کن".
- زمان شروع و پایان رو ثبت می‌کنه.
- میزان مصرف رم و CPU کانتینر متد رو مانیتور می‌کنه با استفاده از Docker Stats API

🏃 فازهای عملیاتی (Sprints)

🏁 اسپرینت ۱: آماده‌سازی و پارس کردن (Ingestion)

هدف : تبدیل دیتای خام XML و SQL به فرمتی که ماشین بفهمه مثل JSON یا CSV

چالش : فایل XML ویکی‌پدیا وحشتناک بزرگه. اگه بخوای کلش رو یهو توی رم لود کنی، سیستم منفجر میشه! باید به صورت **Stream** بخونی.

منابع دیتا برای این اسپرینت:

- برای متن مقالات (سنگین‌ترین فایل):

enwiki-latest-pages-articles-multistream.xml.bz2

- برای متادیتای صفحات ID و Title :

enwiki-latest-page.sql.gz

🛡️ اسپرینت ۲: کارآگاه تکراری‌ها (Deduplication)

چالش : فرض کن میلیون‌ها URL داریم و می‌خوایم بفهمیم کدوم جدیده و کدوم قبلاً دیده شده.

راهنمایی: (Hint)

- ذخیره کردن تمام URL ها به صورت String توی یک لیست یا Set معمولی، رم سرور رو می‌ترکونه.
- دنبال روش‌هایی باش که "اثر انگشت (Hash) "دیتا رو نگه می‌دارن.
- یک قدم جلوتر: ساختارهایی هستن که **احتمالی (Probabilistic)** کار می‌کنن. یعنی با مصرف چند "بیت" حافظه، بهت میگن "این آیتم *احتمالاً* هست" یا "قطعاً نیست."

- *سؤال تحقیق :*چطور می‌تونیم با قبول کردن یک درصد خطای خیلی خیلی کم (مثلاً ۰.۱٪)، مصرف حافظه رو از ۱۰ گیگابایت به ۱۰۰ مگابایت برسونیم؟ کلیدواژه‌ها.(Hashing, Probabilistic Data Structures) :

🔍اسپرینت ۳: قلب موتور جستجو(Indexing & Compression)

اینجا کار به دو بخش تقسیم میشه. باید هر کدوم رو جدا تست کنی و بعد ترکیبشون کنی:

بخش اول: ایندکس‌گذاری(Fast Search)

- کاربر کلمه "History" رو سرچ می‌کنه. ما نباید کل متن‌ها رو بگردیم.باید "نمایه معکوس (Inverted Index) " بسازی.
- سوال: دیکشنری کلمات رو کجا نگه داریم؟ Hash Map یاTrie ؟

بخش دوم: فشرده‌سازی(Compression)

- لیست آیدی مقالاتی که کلمه "The" توشون هست، میلیونیه[... ,1, 5, 12, 20] !
- این لیست رو چطور فشرده کنیم؟ ذخیره خود عدد بهتره یا ذخیره "فاصله بین عددها(Delta Encoding) "؟
- از روش‌های کدگذاری بیت-محور مثل Variable Byte Code یا Huffman استفاده کن.

خروجی مقایسه‌ای:

۱. سرعت سرچ بدون فشرده‌سازی.

۲. سرعت سرچ با فشرده‌سازی آیا زمانی که صرف Decompress میشه، ارزش کاهش حجم رو داره؟

منبع دیتا برای تست:

- ایندکس آماده برای دسترسی سریع (جهت تست):

enwiki-latest-pages-articles-multistream-index.txt.bz2

📊اسپرینت ۴: رتبه‌بندی و سورت(Ranking & Sorting)

چالش :کاربر میگه "۱۰ مقاله طولانی‌تر" یا "۱۰ مقاله مرتبط‌تر.(Top-K Query) "

ما نیاز به ساختاری داریم که همیشه آیتم‌های "برتر" رو دم دست داشته باشه.

سناریوها:

۱. **Range Query**:مقالاتی که طولشون بین ۵۰۰ تا ۱۰۰۰ کلمه است."اینجا درخت‌ها (Trees) به کار میان

۲. **Top-K**:فقط ۵ تا مقاله اول رو بده .اینجا شاید هیپ‌ها (Heaps) بهتر باشن.

تحلیل تو:

- آیا برای پیدا کردنTop-10 ، لازمه کل آرایه رو Sortکنی؟ (قطعا نه! پس چه الگوریتمی بهتره؟).
- داده‌ها رو بر اساس "طول متن" یا "تعداد کلمات" مرتب کن.

اسپرینت ۵: کشف دانش(Graph Analysis)

چالش :ویکی‌پدیا یک شبکه است. ما می‌خوایم بدونیم "مهم‌ترین" صفحات کجان مثلPageRank

تسک :گراف لینک‌ها رو بساز.

- گره‌ها = (Nodes) مقالات.
- یال‌ها = (Edges) لینک‌های داخل متن.

تحلیل:

- پیدا کردن :(Strongly Connected Components (SCC کجای ویکی‌پدیاســـت که همه به هم لینک دادن؟ (حلقه‌های موضوعی).
- از الگوریتم‌های گراف کلاسیک استفاده کن ولی حواست به رم باشه (گراف خیلی بزرگه!).

منابع دیتا برای گراف:

- جدول لینک‌های بین صفحات (منبع اصلی گراف):

enwiki-latest-pagelinks.sql.gz

- جدول هدف لینک‌ها برای حل کردنRedirect ها برای :

enwiki-latest-linktarget.sql.gz

📦 دیتای مورد نیاز(The Gold Mine)

اینم لیست دقیق فایل‌هایی که باید بدی دست "کانتینر دیتا "از طریق Volume لازم نیست همش رو دانلود کنی، برای تست می‌تونی ۱۰,۰۰۰ خط اول رو بخونی، ولی کد نهاییت باید روی فایل کامل کار کنه.

لینک پایه:(Base URL)

برای فاز **توسعه و تست(Development)** ، ما استراتژی **"Subset"**رو پیش می‌گیریم. خوشبختانه ویکی‌پدیا فایل‌های XML رو تیکه تیکه (Chunk) کرده. ما برای شروع فقط **تیکه اول** رو برمی‌داریم که هم مقالات اصلی توشه، هم حجمش کمه (حدود ۳۰۰ مگابایت).

این لیست نهایی و **بهینه شده** برای شروع پروژه(Dev Environment) :

https://dumps.wikimedia.org/enwiki/latest/

کاربرد در پروژه (اسپرینت)	فایل پیشنهادی (نسخه سبک)	چرا این فایل؟
دیتای اصلی (متن مقالات) *(اسپرینت ۱ و ۳)*	https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles-multistream1.xml-p1p41242.bz2	**طلایی‌ترین فایل!** این فایل حدود **۲۹۵ مگابایت** حجم داره و شامل قدیمی‌ترین و مهم‌ترین مقالات (از آیدی ۱ تا ۴۱۲۴۲) میشه. برای تست پارسر و ایندکس عالیه.

کاربرد در پروژه (اسپرینت)	فایل پیشنهادی (نسخه سبک)	چرا این فایل؟
ایندکس مقالات *(اسپرینت ۳)*	https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles-multistream-index1.txt-p1p41242.bz2	ایندکسِ همون فایل بالایی هست. برای اینکه سریع آیدی‌ها رو پیدا کنی بدون اینکه کل XML رو بخونی.
گراف لینک‌ها (یال‌ها) *(اسپرینت ۵)*	https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pagelinks.sql.gz	**نکته فنی:** نسخه کوچیک نداره! ولی چون ساختارش SQL هست، توی کدت (Python/Go) بگو: "فقط ۱۰۰,۰۰۰ خط اول رو بخون". لازم نیست کل ۶ گیگ رو ایمپورت کنی.
اطلاعات صفحات (گره‌ها) *(اسپرینت ۱ و ۴)*	https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-page.sql.gz	اینم نسخه کوچیک نداره (۲.۳ گیگ). مثل بالا، به صورت **Stream** بخون و ۱۰,۰۰۰ رکورد اول رو بردار.

کاربرد در پروژه (اسپرینت)	فایل پیشنهادی (نسخه سبک)	چرا این فایل؟
نام‌های مجاز *(اسپرینت ۲ - فیلترها)*	https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-all-titles-in-ns0.gz	لیست تمام تیترهای مقالات اصلی. برای تست کردن Bloom Filter عالیه که ببینی آیا یک تیتر وجود داره یا نه.

🎯معیارهای ارزیابی(KPIs)

من به عنوانCTO ، آخر کار این ۳ تا چیز رو ازت می‌خوام:

۱. **صحت عملکرد:(Correctness)**

- خروجی‌ها باید استاندارد باشن مثلاًJSON
- داکر فایل‌ها (Dockerfile) باید "بیلد" بشن و بدون خطا بالا بیان.

۲. **تحلیل کارایی:(Performance Analysis)**

- فقط عدد نده. نمودار بده!
- مثلاً" *درخت B-Tree برای دیسک خوب بود ولی برای دیتای کم توی رم، Hash Map سریع‌تر بود".*
- مقایسه "زمان اجرا" vs "مصرف حافظه."

۳. **کیفیت کد و داکیومنت:**

- کد باید تمیز (Clean Code) باشه.
- گزارش نهاییت باید بگه: "پیشنهاد من برای معماری آینده شرکت این است که..." (با دلیل علمی).

خب مهندس کلی کار داریم! اگر سوالی بود در خدمتیم 🚀 😉 !