Коротко: да, можно. Обе «альфы» можно держать на одном сервере и теоретически шарить один Groq-key и один Apify-key. Но сделай это аккуратно — вот как:

# Как поставить 2 системы на один сервер

* **Одна машина, два сервиса**: alpha1 и alpha2 как отдельные контейнеры (Docker Compose), общий Nginx/Traefik.
* **Разделение ресурсов**: выставь CPU/MEM limits на контейнеры, отдельные логи и каталоги данных.
* **Одна БД — два schemas** (или две БД) для изоляции.

# Шеринг Groq и Apify: что учесть

**Это можно**, но имей в виду 3 вещи: лимиты, учёт расходов, безопасность.

1. Лимиты и стабильность

* Повесь **общий rate-limiter**/очередь (Redis + bullmq / Celery) перед Groq/Apify вызовами.
* Введи **per-tenant квоты**: например, alpha1 ≤ 60% RPS, alpha2 ≤ 40%.
* Экспоненциальный **retry + jitter**, backoff на 429/5xx.

1. Учёт расходов

* Помечай все вызовы **метками**: x-tenant: alpha1|alpha2, job\_id, source.
* Для Groq: логируй **модель/токены**; для Apify: **actor runId/duration/dataset size**.
* Снимай ежедневные **cost-reports** по меткам → легко понять, кто съел бюджет.

1. Безопасность/бласт-радиус

* Ключи в **vault** (.env только при деплое), доступ контейнеру через env-vars с **read-only**.
* Минимизируй поверхность: отдельные **service accounts** в Apify, **projects/keys** в Groq, если доступно.
* Если останешься на **одном ключе для двоих**: обязательно пер-тенант rate-limit, чтобы один не положил второго.

# Рекомендация (практичная)

* **Эконом-вариант (можно сейчас):**
  + Один сервер → два сервиса → **один Groq key**, **один Apify token**.
  + Добавь: (а) общий rate-limiter, (б) пер-тенант квоты, (в) метки/логи для биллинга.
* **Идеально (когда пойдут объёмы):**
  + Для **Groq** — **два project-key** (по одному на альфу) в одной орг.
  + Для **Apify** — **два токена** или **два actor-token мета-неймспейса** (тоже в одной орг).
  + Так ты сохраняешь экономию (одна подписка/орг), но получаешь изоляцию и прозрачный биллинг.

# Мини-чеклист внедрения

1. Docker Compose: alpha1, alpha2, rate-limiter (Redis), nginx.
2. Env-переменные: GROQ\_API\_KEY, APIFY\_TOKEN поднимаются в оба сервиса через secrets.
3. Middleware-SDK: общий клиент, который:
   * проставляет метки {tenant, model, job\_id},
   * делает rate-limit/queue,
   * ведёт токен-и usage-лог.
4. Мониторинг: Grafana/Prometheus (RPS, 429, latency), алерты по квотам.
5. Политика отказа: если квота альфы выбита — мягкая деградация (очередь/пауза), не валить соседа.

# Ответы на твои «а можно…?»

* **Один сервер на двоих?** Да.
* **Один Groq и один Apify на двоих?** Да, с rate-limit, квотами и тегированием.
* **Экономия без двойной оплаты?** Да; позже лучше разнести на per-project keys внутри одной подписки для прозрачности и безопасности.

Если хочешь, на нашей сессии соберу тебе **готовый proxy-слой** (Node/Python) с квотами и логированием usage — это 15–25 минут поверх текущего кода и сразу даёт контроль над лимитами и затратами.

##### Вы сказали:

Да

###### ChatGPT сказал:

Думал на протяжении 4 секунд

Отлично. Тогда вот короткий план «в бой» для 2 альф на одном сервере с общим Groq+Apify:

# Шаги деплоя (коротко)

1. **Сервер**: Ubuntu 22.04+, Docker + Docker Compose, ufw (80/443/22).
2. **Структура**:

/srv/stack

├─ alpha1/

├─ alpha2/

├─ proxy/ # общий слой к Groq/Apify (rate-limit/квоты/логи)

├─ nginx/

├─ grafana/ prometheus/

└─ .env.secrets # хранить через docker secrets / sops/age

1. **Секреты**: положи GROQ\_API\_KEY и APIFY\_TOKEN в secrets; сервисам отдаём как env.
2. **Compose (ядро)**

version: "3.9"

services:

redis:

image: redis:7-alpine

command: ["redis-server","--appendonly","yes"]

volumes: ["./data/redis:/data"]

proxy:

build: ./proxy

environment:

- GROQ\_API\_KEY\_FILE=/run/secrets/groq\_key

- APIFY\_TOKEN\_FILE=/run/secrets/apify\_token

- TENANT\_QUOTAS=alpha1:60,alpha2:40 # проценты RPS

secrets: [groq\_key, apify\_token]

depends\_on: [redis]

alpha1:

build: ./alpha1

environment:

- TENANT=alpha1

- PROXY\_URL=http://proxy:8080

depends\_on: [proxy]

alpha2:

build: ./alpha2

environment:

- TENANT=alpha2

- PROXY\_URL=http://proxy:8080

depends\_on: [proxy]

nginx:

image: nginx:alpine

volumes:

- ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro

ports: ["80:80","443:443"]

depends\_on: [alpha1, alpha2]

secrets:

groq\_key:

file: ./secrets/groq\_key.txt

apify\_token:

file: ./secrets/apify\_token.txt

1. **Nginx (роутинг по путям/доменам)**

http {

upstream alpha1 { server alpha1:3000; }

upstream alpha2 { server alpha2:3000; }

server {

listen 80;

location /alpha1/ { proxy\_pass http://alpha1/; }

location /alpha2/ { proxy\_pass http://alpha2/; }

}

}

1. **Proxy-слой (Node/Python) — что делает**

* Принимает заявки POST /groq и POST /apify.
* Добавляет теги {tenant, model, job\_id}.
* **Rate-limit + очередь**: Redis + токены/лейки (per-tenant).
* **Биллинг-логи**: пишет usage (в т.ч. токены Groq, runId Apify) в лог/БД.
* **Backoff**: экспоненциальный retry + jitter на 429/5xx.

1. **Квоты/скорость**

* Поставь глобально: max RPS (например, 10–20 rps суммарно).
* Квоты: alpha1=60%, alpha2=40% (меняется в TENANT\_QUOTAS).
* Очередь: TTL задач, чтобы не копить ворох, если лимит упёрся.

1. **Мониторинг**

* Prometheus (метрики proxy: rps, p95 latency, 429, ошибки).
* Grafana: дашборды «per tenant» + алерты (квота >80%, 429 бурсты).

1. **Безопасность**

* Секреты только через Docker secrets / sops.
* Контейнерам — **read-only FS**, no-new-privileges, mem/cpu limits.
* Nginx → proxy только по внутренней сети; наружу — TLS (Caddy/Certbot).

1. **Проверка (чек-лист)**

* docker compose up -d → healthchecks ок.
* Тест curl:
  + POST /alpha1/api/test (внутренний вызов Groq через proxy).
  + POST /alpha2/api/test.
* Сымитируй нагрузку (5–10 rps) → убедись, что квоты соблюдаются, 429 нет, latency стабильна.

## Ответы на твои вопросы «сейчас»

* **Две альфы на одном сервере с одним Groq/APIFY** — да, со shared прокси-слоем и квотами.
* **Экономия** — да, оплата одна, но учёт расходов раздельный (по меткам tenant).
* **Расширение** — при росте просто добавишь второй ключ Groq/Apify и укажешь в proxy правило: tenant→key.

Хочешь — я подготовлю минимальный шаблон proxy (Node) с rate-limit и логированием + заготовки alpha1/alpha2 (Next.js/Express) — чтобы ты сразу поднял стенд и погнал тесты.

##### Вы сказали:

Да

###### ChatGPT сказал:

Красавчик. Ловишь готовый минимальный стенд: общий proxy с квотами+ретраями, две «альфы» (Next.js API) и docker-compose. Поднимешь — и можно сразу тестить shared Groq+Apify.

# Структура репо

/srv/stack

├─ docker-compose.yml

├─ secrets/

│ ├─ groq\_key.txt

│ └─ apify\_token.txt

├─ proxy/

│ ├─ Dockerfile

│ ├─ package.json

│ └─ index.js

├─ alpha1/ # Next.js (можно потом заменить на твой фронт)

│ ├─ Dockerfile

│ ├─ package.json

│ └─ pages/api/test.js

├─ alpha2/

│ ├─ Dockerfile

│ ├─ package.json

│ └─ pages/api/test.js

└─ nginx/

└─ nginx.conf

# docker-compose.yml

version: "3.9"

services:

redis:

image: redis:7-alpine

command: ["redis-server","--appendonly","yes"]

volumes: ["./data/redis:/data"]

proxy:

build: ./proxy

environment:

- REDIS\_URL=redis://redis:6379

- TENANT\_QUOTAS=alpha1:60,alpha2:40

- GROQ\_API\_KEY\_FILE=/run/secrets/groq\_key

- APIFY\_TOKEN\_FILE=/run/secrets/apify\_token

- GLOBAL\_RPS=15

- BURST=20

secrets: [groq\_key, apify\_token]

depends\_on: [redis]

alpha1:

build: ./alpha1

environment:

- TENANT=alpha1

- PROXY\_URL=http://proxy:8080

depends\_on: [proxy]

alpha2:

build: ./alpha2

environment:

- TENANT=alpha2

- PROXY\_URL=http://proxy:8080

depends\_on: [proxy]

nginx:

image: nginx:alpine

volumes: ["./nginx/nginx.conf:/etc/nginx/nginx.conf:ro"]

ports: ["80:80"]

depends\_on: [alpha1, alpha2]

secrets:

groq\_key:

file: ./secrets/groq\_key.txt

apify\_token:

file: ./secrets/apify\_token.txt

# proxy/Dockerfile

FROM node:20-alpine

WORKDIR /app

COPY package.json package-lock.json\* ./

RUN npm ci --omit=dev

COPY index.js .

EXPOSE 8080

CMD ["node","index.js"]

# proxy/package.json

{

"name": "shared-proxy",

"version": "1.0.0",

"dependencies": {

"axios": "^1.7.2",

"express": "^4.19.2",

"ioredis": "^5.3.2",

"pino": "^9.0.0",

"rate-limiter-flexible": "^2.5.3"

}

}

# proxy/index.js (RPS квоты per-tenant + ретраи + роуты /groq и /apify)

const express = require('express');

const axios = require('axios');

const Redis = require('ioredis');

const { RateLimiterRedis } = require('rate-limiter-flexible');

const pino = require('pino')();

const app = express();

app.use(express.json({ limit: '2mb' }));

// env

const REDIS\_URL = process.env.REDIS\_URL || 'redis://localhost:6379';

const GLOBAL\_RPS = Number(process.env.GLOBAL\_RPS || 10);

const BURST = Number(process.env.BURST || 15);

const quotas = (process.env.TENANT\_QUOTAS || 'alpha1:50,alpha2:50')

.split(',').map(s => s.trim()).reduce((acc, pair) => {

const [name, pct] = pair.split(':'); acc[name] = Number(pct) / 100; return acc;

}, {});

const groqKey = require('fs').readFileSync(process.env.GROQ\_API\_KEY\_FILE, 'utf8').trim();

const apifyToken = require('fs').readFileSync(process.env.APIFY\_TOKEN\_FILE, 'utf8').trim();

const redis = new Redis(REDIS\_URL);

// глобальный лимитер (burst)

const globalLimiter = new RateLimiterRedis({

storeClient: redis, keyPrefix: 'rl:global', points: BURST, duration: 1, execEvenly: true

});

// лимитеры на тенантов

const tenantLimiters = {};

Object.entries(quotas).forEach(([tenant, share]) => {

tenantLimiters[tenant] = new RateLimiterRedis({

storeClient: redis, keyPrefix: `rl:${tenant}`,

points: Math.max(1, Math.floor(GLOBAL\_RPS \* share)), duration: 1, execEvenly: true

});

});

async function withRateLimit(tenant, fn) {

if (!tenantLimiters[tenant]) throw new Error('unknown tenant');

await globalLimiter.consume('global');

await tenantLimiters[tenant].consume(tenant);

return fn();

}

// простой backoff с джиттером

const sleep = ms => new Promise(r => setTimeout(r, ms));

async function withRetry(task, tries = 4) {

let lastErr;

for (let i = 0; i < tries; i++) {

try { return await task(); }

catch (e) {

lastErr = e;

const status = e.response?.status;

if (status && status < 500 && status !== 429) break;

const delay = Math.min(2000, 200 \* (2 \*\* i)) + Math.floor(Math.random() \* 100);

await sleep(delay);

}

}

throw lastErr;

}

// GROQ proxy

app.post('/groq', async (req, res) => {

const { tenant, model = 'llama-3.1-70b-versatile', messages, temperature = 0.4 } = req.body || {};

if (!tenant || !Array.isArray(messages)) return res.status(400).json({ error: 'tenant & messages required' });

try {

const data = await withRateLimit(tenant, () =>

withRetry(async () => {

const r = await axios.post('https://api.groq.com/openai/v1/chat/completions', {

model, messages, temperature

}, {

headers: { Authorization: `Bearer ${groqKey}` },

timeout: 30000

});

return r.data;

})

);

pino.info({ t: tenant, model, usage: data.usage }, 'groq-ok');

res.json({ ok: true, data });

} catch (e) {

pino.error({ t: tenant, msg: e.message, status: e.response?.status, body: e.response?.data }, 'groq-fail');

res.status(502).json({ error: 'groq proxy failed', detail: e.response?.data || e.message });

}

});

// APIFY proxy (старт актёра)

app.post('/apify', async (req, res) => {

const { tenant, actorId, input } = req.body || {};

if (!tenant || !actorId) return res.status(400).json({ error: 'tenant & actorId required' });

try {

const data = await withRateLimit(tenant, () =>

withRetry(async () => {

const r = await axios.post(

`https://api.apify.com/v2/acts/${actorId}/runs?token=${apifyToken}`,

{ input }, { timeout: 30000 }

);

return r.data;

})

);

pino.info({ t: tenant, actorId, runId: data.data?.id }, 'apify-started');

res.json({ ok: true, data });

} catch (e) {

pino.error({ t: tenant, msg: e.message, status: e.response?.status, body: e.response?.data }, 'apify-fail');

res.status(502).json({ error: 'apify proxy failed', detail: e.response?.data || e.message });

}

});

app.get('/health', (req, res) => res.json({ ok: true }));

app.listen(8080, () => pino.info('proxy listening on 8080'));

# alpha1/Dockerfile (аналогично для alpha2)

FROM node:20-alpine

WORKDIR /app

RUN npm i -g next@14

COPY package.json package-lock.json\* ./

RUN npm ci --omit=dev

COPY pages ./pages

EXPOSE 3000

CMD ["npx","next","dev","-p","3000"]

# alpha1/package.json (аналогично в alpha2)

{

"name": "alpha1",

"version": "1.0.0",

"dependencies": {

"axios": "^1.7.2",

"next": "^14.2.5"

}

}

# alpha1/pages/api/test.js (аналогично в alpha2, поменяй только текст TENANT)

export default async function handler(req, res) {

try {

const tenant = process.env.TENANT || 'alpha1';

const proxy = process.env.PROXY\_URL || 'http://localhost:8080';

const r = await fetch(`${proxy}/groq`, {

method: 'POST',

headers: { 'content-type': 'application/json' },

body: JSON.stringify({

tenant,

model: 'llama-3.1-8b-instant',

messages: [

{ role: 'system', content: 'You are a concise assistant.'},

{ role: 'user', content: `Say hello from ${tenant} and give me a 1-sentence tip.`}

],

temperature: 0.5

})

});

const data = await r.json();

res.status(200).json({ ok: true, tenant, completion: data.data?.choices?.[0]?.message?.content || data });

} catch (e) {

res.status(500).json({ ok: false, error: e.message });

}

}

# nginx/nginx.conf

events {}

http {

upstream alpha1 { server alpha1:3000; }

upstream alpha2 { server alpha2:3000; }

server {

listen 80;

location /alpha1/ { proxy\_pass http://alpha1/; }

location /alpha2/ { proxy\_pass http://alpha2/; }

location /healthz { return 200 "ok\n"; add\_header Content-Type text/plain; }

}

}

# Быстрый запуск

# 1) ключи

mkdir -p /srv/stack/secrets

echo "YOUR\_GROQ\_API\_KEY" > /srv/stack/secrets/groq\_key.txt

echo "YOUR\_APIFY\_TOKEN" > /srv/stack/secrets/apify\_token.txt

# 2) поднять

cd /srv/stack

docker compose up -d --build

# 3) проверить

curl http://localhost/healthz

curl http://localhost/alpha1/api/test

curl http://localhost/alpha2/api/test

# Что уже есть «из коробки»

* **Shared Groq+Apify** через /groq и /apify.
* **Rate-limit** per-tenant + глобальный, **burst**, **retry** с backoff.
* **Квоты**: TENANT\_QUOTAS=alpha1:60,alpha2:40 — меняешь как надо.
* **Логи** (pino): usage, runId, ошибки — готово для Prometheus sidecar.
* **Два фронтовых сервиса** (alpha1/alpha2) с тестовым вызовом.

# Как дальше наращивать

* Подключить TLS (Caddy/Traefik) и домены для /alpha1 и /alpha2.
* В proxy добавить метрики /metrics (prom-client) и алерты.
* В альфах заменить pages/api/test.js на реальные эндпоинты парсера/UI.
* Если надо — добавить второй ключ Groq/Apify и роутинг tenant→key.

Если ок — идём дальше: прикручу пром-метрики и minimal UI-панель «Usage & Quotas» (Grafana/React) прямо поверх этого же стека.