VYSOKÁ ŠKOLA POLYTECHNICKÁ JIHLAVA

Aplikovaná informatika

NASAZENÍ A PROVOZ SELF-HOSTED LLM OLLAMA NA PLATFORMĚ AWS

Bakalářská práce

Autor práce: Štěpán Konečný

Vedoucí práce: Ing. Jan Mittner, Ph.D.

Jihlava 2026

Vysoká škola polytechnická Jihlava

Tolstého 16, 586 01 Jihlava

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Autor práce:	**Štěpán Konečný**
Studijní program:	Aplikovaná informatika
Garant studijního programu:	Ing. Lenka Kuklišová Pavelková, Ph.D.
Název práce:	**Nasazení a provoz self-hosted LLM Ollama na platformě AWS**
Vedoucí práce:	Ing. Jan Mittner, Ph.D.
Cíl práce:	Práce se bude zabývat návrhem a implementací samostatně hostovaného velkého jazykového modelu (LLM) pomocí platformy Ollama, nasazeného v prostředí Amazon Web Services (AWS). Cílem je vytvořit bezpečné, výkonné a škálovatelné řešení, které umožní provoz vlastního jazykového modelu bez závislosti na komerčních cloudových API. Práce se bude zaměřovat na návrh architektury, která kombinuje výpočetní výkon GPU instancí AWS EC2 s moderními službami, jako jsou Elastic Load Balancer (ALB), API Gateway a VPC Link, čímž bude dosaženo bezpečného a efektivního přístupu k modelu prostřednictvím veřejného rozhraní.

Abstrakt

Bakalářská práce se zaměřuje na návrh a implementaci self-hosted řešení pro provoz velkého jazykového modelu (LLM - Large Language Model) pomocí platformy Ollama v cloudovém prostředí Amazon Web Services (AWS). Cílem je vytvořit bezpečnou, škálovatelnou a nákladově efektivní alternativu ke komerčním cloudovým API službám, která organizacím poskytne plnou kontrolu nad daty a infrastrukturou.

Teoretická část práce představuje základní koncepty velkých jazykových modelů, platformu Ollama pro jejich nasazení, klíčové služby AWS, kontejnerizaci pomocí Dockeru a GPU (Graphics Processing Unit - grafický procesor) computing. Praktická část popisuje návrh architektury kombinující EC2 GPU instance s moderními síťovými službami AWS včetně Network Load Balancer, VPC Link a API Gateway. Infrastruktura je definována pomocí Terraform pro automatizované a reprodukovatelné nasazení. Řešení využívá Docker kontejnerizaci s NVIDIA Container Toolkit pro přístup ke GPU a implementuje autentizaci pomocí API klíčů.

Testování prokázalo funkčnost systému s latencí 0,71 s pro krátké prompty až 4,46 s pro dlouhé prompty, throughput až 7 požadavků za sekundu při 10 souběžných požadavcích a time-to-first-token přibližně 981 ms. Analýza nákladů ukázala měsíční provozní náklady přibližně 796 USD pro On-Demand instanci. Práce také navrhuje strategie pro horizontální škálování pomocí Auto Scaling Groups a optimalizaci nákladů využitím Spot nebo Reserved instancí.

Klíčová slova

Large Language Models, LLM, Ollama, Amazon Web Services, AWS, EC2, GPU computing, Docker, kontejnerizace, API Gateway, Network Load Balancer, VPC Link, self-hosted, cloudové služby, škálovatelnost, Terraform, Infrastructure as Code

Prohlašuji, že předložená bakalářská práce je původní a zpracoval/a jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem v práci neporušil/a autorská práva (ve smyslu zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů, v platném znění, dále též „**AZ**“).

Byl/a jsem seznámen/a s tím, že na mou bakalářskou práci se plně vztahuje **AZ**, zejména § 60 (školní dílo).

Podle § 47b zákona o vysokých školách souhlasím se zveřejněním své práce podle Směrnice pro vedení, vypracování a zveřejňování závěrečných prací na VŠPJ, a to bez ohledu na výsledek obhajoby.

Beru na vědomí, že VŠPJ má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom/a toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem VŠPJ, která má právo ode mě požadovat přiměřený příspěvek na úhradu nákladů, vynaložených vysokou školou na vytvoření díla (až do jejich skutečné výše), z výdělku dosaženého v souvislosti s užitím díla či poskytnutím licence.

V Jihlavě dne 27. března 2026

…….

Podpis studenta/k

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

AI	Artificial Intelligence (umělá inteligence)
API	Application Programming Interface (rozhraní pro programování aplikací)
AWS	Amazon Web Services
CLI	Command Line Interface (rozhraní příkazové řádky)
CPU	Central Processing Unit (centrální procesor)
CUDA	Compute Unified Device Architecture
EBS	Elastic Block Store
EC2	Elastic Compute Cloud
GB	Gigabyte (gigabajt)
GPU	Graphics Processing Unit (grafický procesor)
GUI	Graphical User Interface (grafické uživatelské rozhraní)
HCL	HashiCorp Configuration Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Secure
IaC	Infrastructure as Code (infrastruktura jako kód)
JSON	JavaScript Object Notation
LLM	Large Language Model (velký jazykový model)
NLB	Network Load Balancer
NLP	Natural Language Processing (zpracování přirozeného jazyka)
RAM	Random Access Memory (operační paměť)
REST	Representational State Transfer
SDK	Software Development Kit
SSH	Secure Shell (zabezpečený vzdálený přístup)
SSD	Solid State Drive
TB	Terabyte
TCP	Transmission Control Protocol
URL	Uniform Resource Locator

VPC – Virtual Private Cloud (virtuální privátní cloud)

VRAM – Video Random Access Memory (video paměť)

Úvod

Velké jazykové modely představují jeden z nejvýznamnějších pokroků v oblasti umělé inteligence posledních let. Jejich schopnost generovat text, odpovídat na dotazy, programovat nebo analyzovat data je čím dál častěji využívána firmami v komerční i výzkumné sféře. Většina firem spoléhá na komerčně dostupné služby od firem jako OpenAI API, které s sebou přinášejí závislost na externím poskytovateli, potenciální bezpečnostní rizika při práci s citlivými daty a průběžné provozní náklady.

Cílem zpracované bakalářské práce je navrhnout a implementovat self-hosted řešení pro provoz velkého jazykového modelu pomocí platformy Ollama. Práce se zaměřuje na vytvoření API (Application Programming Interface - rozhraní pro programování aplikací) v cloudovém prostředí Amazon Web Services (AWS). Tímto způsobem lze získat plnou kontrolu nad daty, infrastrukturou a náklady, ale zůstane zachována flexibilita a škálovatelnost cloudového prostředí.

Práce se zaměřuje na vytvoření komplexní architektury, která kombinuje výpočetní výkon GPU instancí AWS EC2 s moderními síťovými a bezpečnostními službami. Výsledkem je bezpečné a výkonné API rozhraní, které umožňuje přístup k jazykovému modelu z libovolného zařízení prostřednictvím jednoduchých http (HyperText Transfer Protocol) požadavků s využitím autentizace pomocí API klíče.

Struktura práce je následující. Nejprve provedeme průzkum dostupných řešení pro provoz LLM a porovnáme výhody a nevýhody různých přístupů. V teoretické části představíme klíčové koncepty a technologie, které jsou základem celého řešení. Následuje návrh architektury systému a detailní popis implementace jednotlivých komponent. V praktické části provedeme měření výkonu, latence a provozních nákladů při různých scénářích použití. Závěr práce shrnuje dosažené výsledky a navrhuje možnosti dalšího vývoje řešení.

1 Průzkum dostupných řešení

Oblast umělé inteligence zaznamenává v posledních letech výrazný rozvoj. Firmy se intenzivně předhánějí ve vývoji nových AI aplikací a v implementaci umělé inteligence do různých oblastí svého fungování, například do zákaznické podpory.

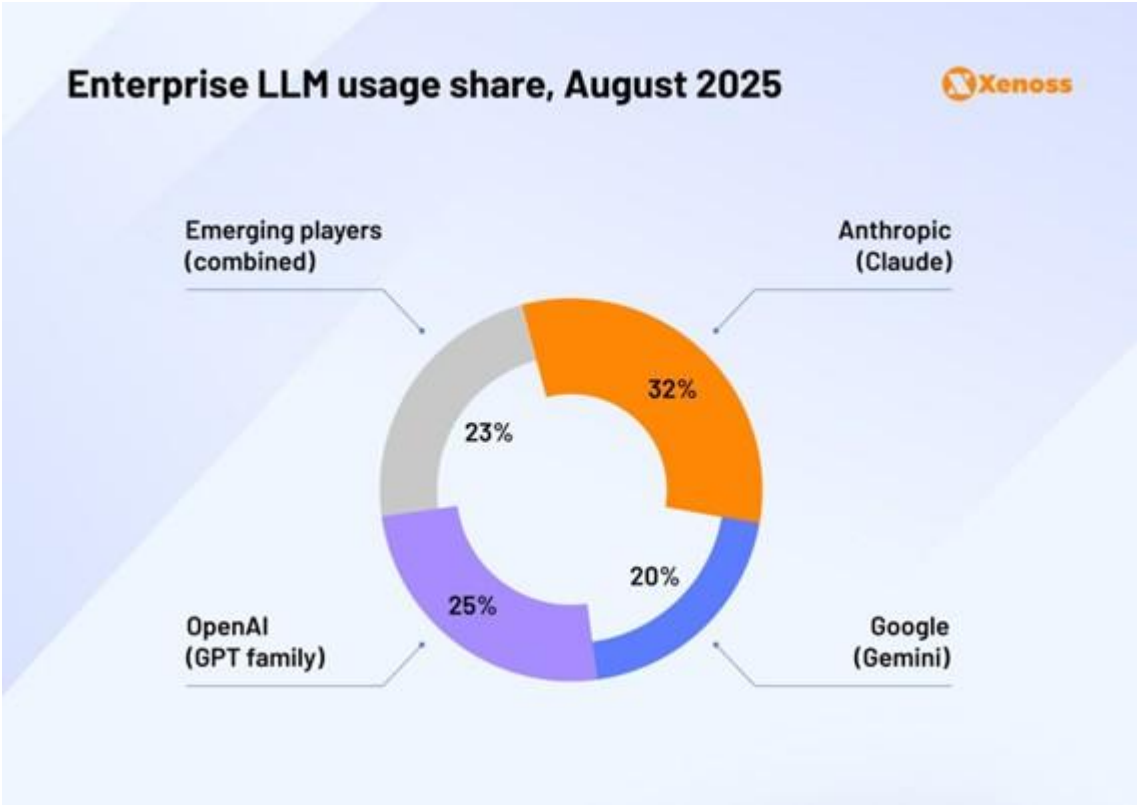
K vývoji aplikací firmy nejčastěji využívají aktuálně dostupných API platform služeb, ale vedle komerčních cloudových řešení existuje také alternativa v podobě self-hosted (lokálně provozované) LLM, které umožňují provoz modelů na vlastních serverech a přístup k LLM API bez závislosti na externím poskytovateli.

Volba mezi těmito přístupy závisí na konkrétních potřebách organizace, požadavcích na výkon, bezpečnost dat, nákladech a dalších faktorech. Každé řešení má své přednosti i slabiny, které je nutné pečlivě zvážit při rozhodování o výběru řešení.

1.1 Komerční cloudové služby

Komerční cloudové služby představují rychlé a pohodlné řešení pro nasazení AI aplikací bez nutnosti investovat do vlastní infrastruktury.

Současný trh je ovládán třemi největšími poskytovateli a jejich API platformy jsou nejčastěji využívány pro jakýkoliv vývoj AI řešení: OpenAI API, Anthropic API a Gemini API. Každá platforma nabízí svoje specifické výhody a zápory či omezení.



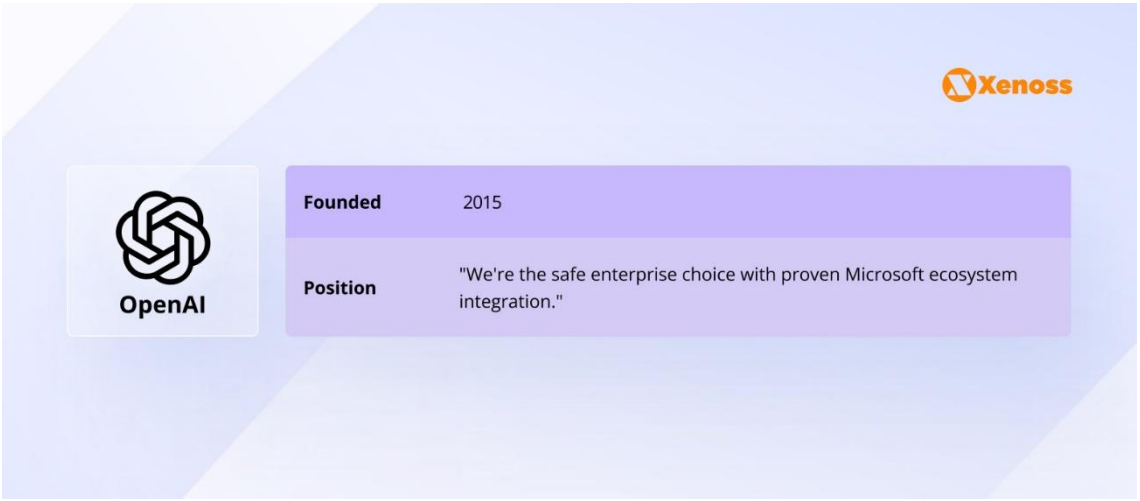
obr. 1: podíl LLM poskytovatelů na IT trhu
zdroj: Xenoss (2025)

1.1.1 OpenAI API

OpenAI je průkopníkem v oblasti komerčně dostupných LLM. Společnost byla založena v roce 2015 Samem Altmanem a Elonem Muskem (který v roce 2018 opustil board firmy) původně jako nezisková organizace. Uvedením produktu ChatGPT s platebním modelem pay-as-you-go vytvořila základní předpoklady pro masové rozšíření generativní AI a zásadně ovlivnila současný IT trh.

Aktuální řada modelů GPT (zejména GPT-4.1 a nově i GPT-5.1) je známá svojí všestranností v úlohách jako je uvažování, programování a obecná konverzace. API je dostupné prostřednictvím platformy Azure AI Foundry, což poskytuje výhodu pro organizace, které využívající ekosystém Microsoft. Mezi tzv. velkou trojkou má OpenAI pověst jako obecně nejlépe vyvážený model (Smith, 2025).

Cenový model je založen na tokenech, kde GPT-5 stojí 1,25 USD za 1 milion vstupních tokenů a 10 USD za 1 milion výstupních tokenů a řadí tím OpenAI jako jednoho z cenově dostupnějších providerů



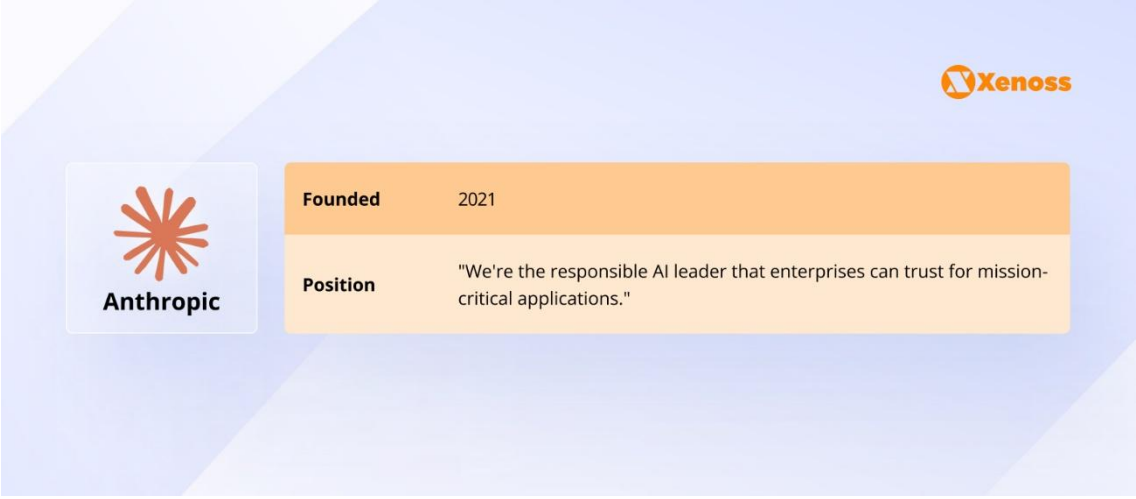
obr. 2: OpenAI logo a motto
zdroj: Xenoss (2025)

1.1.2 Anthropic Claude API

Anthropic byl založen bývalými výzkumníky z OpenAI, kteří se zaměřili na vývoj AI s důrazem na bezpečnost jako hlavní prioritu. Jejich modely Claude (zejména Claude 4 Opus a Sonnet) získaly ze začátku oblibu především u velkých podniků a korporací. Společnost přijala kontrakty od Pentagonu v ceně až 200 milionů USD od Pentagonu, což ukazuje důvěru v bezpečnost architektury modelů od Anthropic.

Postupem času ale obliba modelů Claude, zejména verze Sonnet zejména kvůli lepším výsledkům v programování předehnala OpenAI v podílu na trhu, ve kterém původně OpenAI mělo velký náskok.

Z hlediska cen je Claude Sonnet 4 a nově i Sonnet 4.5 nabízen za 3 USD za 1 milion vstupních tokenů a 15 USD za 1 milion výstupních tokenů, zatímco výkonnější Claude Opus 4.1 stojí 15 USD za vstup a 75 USD za výstup na 1 milion tokenů.



obr. 3: Anthropic logo a motto
Zdroj: Xenoss (2025)

1.1.3 Google Gemini API

Google Gemini vstoupil na trh v roce 2023 jako odpověď technologického giganta na dominanci OpenAI. Model byl od začátku navržen jako multimodální, což znamená, že byl vytvořen pro práci s textem, obrázky, videem a zvukem bez nutnosti dalších úprav.

Hlavní technickou výhodou je vysoké kontextové okno 1 000 000 tokenů. Gemini je integrován s ekosystémem Google Workspace a nabízí konkurenceschopnou cenu oproti Claude a GPT trh.

Pro organizace s předplatným Google Workspace Business nebo Enterprise jsou funkce Gemini zahrnuty přímo v rámci předplatného za ceny od 14,40 USD do 23,40 USD na uživatele měsíčně. Samostatné API pro Gemini 2.5 Pro je nabízeno za 1,25 USD za 1 milion vstupních tokenů a 10 USD za 1 milion výstupních tokenů.

obr. 4: Google logo a moto
Zdroj: Xenoss (2025)

1.1.4 Výhody a nevýhody komerčních API

Hlavní výhody komerčních cloudových API:

- Rychlé nasazení bez nutnosti investic do hardwaru a infrastruktury
- Automatické aktualizace modelů a zlepšování výkonu bez zásahu uživatele
- Škálovatelnost podle okamžité potřeby bez nutnosti předem plánovat kapacitu
- Rozsáhlá dokumentace, aktivní komunita a technická podpora

Hlavní nevýhody komerčních cloudových API:

- Závislost na externím poskytovateli a jeho podmínkách služby
- Omezená kontrola nad zpracováním citlivých dat, která jsou odesílána k externím službám
- Nepředvídatelné náklady při vysokém objemu dotazů, zejména u tokenového cenového modelu
- Možná latence při komunikaci přes internet
- Omezení rychlosti požadavků (rate limiting) a kvóty, které mohou bránit některým případům použití

tab. 1: porovnání cen a vlastností jednotlivých providerů

Poskytovatel	Model	Cena vstup (za 1M tokenů)	Cena výstup (za 1M tokenů)	Nejlepší pro úkoly, které vyžadují
OpenAI	GPT-4.1	$2.00	$8.00	Vysoký výkon, obecné úlohy
	GPT-4.1 mini	$0.40	$1.60	Vybalancovaný výkonem a cenou
	GPT-4.1 nano	$0.10	$0.40	Vysoký objem dat, krátkou odezvu

Anthropic	Claude 4 Opus	$15.00	$75.00	Komplexní uvažování, programování
	Claude 4 Sonnet	$3.00	$15.00	Vyvážený výkon, podnikové úlohy
	Claude 3 Haiku	$0.25	$1.25	Rychlost, vysoký objem dat
Google	Gemini 2.5 Pro	$1.25	$10.00	Pokročilou multimodálnost, dlouhý kontext
	Gemini 2.5 Flash	$0.15	$0.60	Rychlost, nákladovou efektivitu

Zdroj: Pangan (2025)

1.2 Self-hosted řešení

Self-hosted řešení představují alternativu ke komerčním cloudovým službám tím, že umožňují organizacím provozovat velké jazykové modely na vlastní infrastruktuře. Self-hosted přístup nabízí větší kontrolu nad daty, náklady a konfigurací, ale vyžaduje technické znalosti a investici do hardwaru.

1.2.1 Ollama

Ollama je open-source platforma navržená pro snadné spouštění velkých jazykových modelů lokálně. Hlavní výhodou je jednoduchost nasazení pomocí jediného příkazu, který automaticky stáhne a spustí vybraný model. Platforma podporuje širokou škálu populárních open-source modelů, včetně Llama, Mistral, Phi a dalších.

Ollama poskytuje RESTful API (REST - Representational State Transfer), což usnadňuje migraci existujících aplikací z komerčních služeb (Pangan, 2025). Automatická správa paměti GPU optimalizuje využití dostupných zdrojů. Platforma je dostupná pro Linux, macOS i Windows a nabízí knihovny pro různé programovací jazyky včetně Pythonu, JavaScriptu a Go.

Významnou výhodou je podpora Docker kontejnerizace, která zjednodušuje nasazení v cloudových prostředích (Monsur, 2025). Ollama umožňuje načítání a přepínání mezi více modely bez nutnosti restartování služby, což je užitečné pro aplikace vyžadující přístup k různým specializovaným modelům. Podle uživatelských zkušeností je Ollama stabilnější s menším množstvím dostupných modelů, což ji činí vhodnou pro nasazení velkých LLM modelů v produkčním prostředí.

1.2.2 LM Studio

LM Studio je desktopová aplikace s grafickým uživatelským rozhraním pro spouštění LLM modelů na osobních počítačích. Hlavním cílem je zpřístupnit velké jazykové modely netechnickým uživatelům prostřednictvím intuitivního GUI (Graphical User Interface). Na rozdíl od Ollama, která je zaměřena na příkazovou řádku, poskytuje LM Studio uživatelsky přívětivé rozhraní pro správu modelů, konfiguraci parametrů a testování promptů bez nutnosti práce s příkazovou řádkou.

1.2.3 Text Generation WebUI

Text Generation WebUI (také známé jako oobabooga) je komplexní webové rozhraní pro provoz a experimentování s velkými jazykovými modely. Na rozdíl od Ollama a LM Studio se zaměřuje na pokročilé funkce a možnosti konfigurace pro zkušené uživatele a výzkumníky.

Text Generation WebUI nabízí pokročilé funkce jako context length rozšíření, attention mechanismus, sampler settings a detailní kontrolu nad generovacím procesem.

Hlavní nevýhodou je složitější nastavení ve srovnání s Ollama a vyšší technické nároky na správu systému. Pro produkční nasazení také chybí enterprise funkce jako autentizace, rate limiting nebo monitoring, které jsou zásadní pro komerční použití.

Základní srovnání self-hosted platform je uvedeno v následující tabulce.

tab. 2: srovnání self-hosted platforem

Poskytovatel	Typ rozhraní	Hlavní výhody	Omezení
Ollama	CLI, API server	Jednoduché nasazení, Docker podpora, OpenAI kompatibilní API	Omezené pokročilé funkce
LM Studio	Desktop GUI	Intuitivní rozhraní, jednoduchá správa modelů	Pouze desktopové použití
Text Gen WebUI	Web GUI	Pokročilé funkce, systém pluginů, mnoho konfigurací	Složitější nastavení, chybí enterprise funkce

zdroj: vlastní zpracování

1.3 Srovnání přístupů

Výběr mezi komerčním cloudovým API a self-hosted řešením vyžaduje pečlivé zvážení několika klíčových faktorů, které ovlivňují provozní náklady, bezpečnost, flexibilitu a technickou komplexitu řešení.

1.3.1 Bezpečnost a ochrana dat

Komerční cloudová API vyžadují odesílání dat k externím poskytovatelům, což vytváří potenciální bezpečnostní rizika při práci s citlivými informacemi. Organizace v regulovaných odvětvích mohou čelit právním omezením při odesílání dat mimo vlastní infrastrukturu.

Self-hosted řešení nabízejí úplnou kontrolu nad daty, která nikdy neopouští interní síť organizace. To eliminuje rizika spojená s předáváním informací třetím stranám a umožňuje implementovat vlastní bezpečnostní politiky podle specifických potřeb.

Nevýhodou je plná odpovědnost za zabezpečení systému včetně aktualizací, monitoringu hrozeb a incident response, což vyžaduje specializované znalosti a nástroje.

1.3.2 Nákladová analýza

Komerční API využívají model pay-per-use, výhodný pro aplikace s nízkým provozem. Při vysokém objemu požadavků však mohou měsíční náklady rychle růst a stát se nepředvídatelnými.

Analýza reálných nákladů OpenAI API pro zpracování 10 milionů tokenů měsíčně činí přibližně 112,50 USD (GPT-4.1) nebo 90 USD (Claude 4 Sonnet).

Self-hosted řešení vyžaduje počáteční investici do GPU hardwaru. AWS EC2 g5.xlarge s NVIDIA A10G GPU stojí cca 730 USD měsíčně při nepřetržitém provozu, plus náklady na storage a další služby.

1.3.3 Flexibilita a kontrola

Komerční API nabízejí špičkové modely s pravidelnou aktualizací, ale omezují možnosti customizace. Rate limiting může omezit škálovatelnost aplikací s vysokým provozem. Self-hosted přístup umožňuje úplnou kontrolu nad výběrem modelu, konfigurací parametrů a možností fine-tuningu. Nevýhodou je nutnost manuálně sledovat vývoj a aktualizovat modely.

Z hlediska vendor lock-in je self-hosted přístup výhodnější díky nezávislosti na jediném poskytovateli a snadnému přepínání mezi open-source modely. Práce se zaměřuje na self-hosted řešení využívající Ollama v AWS, které kombinuje výhody kontroly nad daty se škálovatelností cloudových služeb.

2 Teoretická část

Teoretická část poskytuje teoretický základ pro pochopení technologií a konceptů využitých v praktické implementaci. Kapitola je rozdělena do pěti hlavních sekcí pokrývající velké jazykové modely, platformu Ollama, AWS služby, kontejnerizaci a GPU computing. Pochopení daných základů je nezbytné pro správný návrh a implementaci funkčního řešení.

2.1 Velké jazykové modely (LLM)

Velké jazykové modely představují revoluci v oblasti umělé inteligence a zpracování přirozeného jazyka. Jejich schopnost generovat text, odpovídat na dotazy a řešit komplexní úlohy je založena na pokročilých algoritmech a rozsáhlém tréninku na obrovském množství dat. Pro pochopení jejich fungování je důležité znát jejich historický vývoj, architekturu a dostupné modely.

2.1.1 Historie a vývoj LLM

Velké jazykové modely mají své kořeny v sémantice, kterou v roce 1883 definoval francouzský filolog Michel Bréal. Skutečný vývoj NLP (Natural Language Processing) začal po druhé světové válce s potřebou automatického překladu jazyků (Foote, 2023).

V roce 1966 vytvořil Joseph Weizenbaum program ELIZA, který dokázal identifikovat klíčová slova a odpovídat předprogramovanými odpověďmi. V osmdesátých letech začala IBM vyvíjet první malé jazykové modely pro predikci slov. Zásadní změnu přinesl vznik World Wide Webu v roce 1991, který poskytl přístup k obrovskému množství dat. V devadesátých letech se začaly používat statistické modely, které byly výrazně rychlejší než předchozí systémy založené na ručně psaných pravidlech (Foote, 2023).

GPU se staly klíčovým nástrojem pro trénování velkých modelů díky schopnosti zpracovávat data paralelně. V roce 2011 začalo být populární hluboké učení a do roku 2018 se jeho algoritmy používaly ve všech odvětvích (Foote, 2023). Koncem roku 2022 OpenAI vydalo ChatGPT, které dramaticky změnilo svět AI a ukázalo praktické možnosti generativní umělé inteligence.

2.1.2 Architektura transformerů

Moderní LLM jsou založeny na architektuře transformerů představené v roce 2017 v práci "Attention Is All You Need". Transformery revolucionizovaly zpracování jazyka tím, že dokážou zpracovávat všechna slova ve větě současně, na rozdíl od starších rekurentních neuronových sítí (RNN), které zpracovávaly text postupně slovo po slovu.

Architektura transformeru se skládá ze dvou hlavních komponent, enkodéru a dekodéru. Enkodér zpracovává vstupní text a transformuje ho do vektorové reprezentace, zatímco dekodér na základě vektorové reprezentace generuje výstupní text. Moderní LLM jako GPT používají pouze dekodér část, zatímco modely jako BERT používají pouze enkodér (TrueFoundry, 2024).

Klíčovou inovací je self-attention mechanismus, který umožňuje modelu vážit důležitost každého slova ve větě v porovnání ke všem ostatním slovům. Self-attention mechanismus pracuje s třemi vektory pro každé slovo. Query (dotaz), Key (klíč) a Value (hodnota), které společně určují, na která slova by se model měl zaměřit při zpracování konkrétního tokenu (TrueFoundry, 2024).

Positional encoding řeší problém pořadí slov. Protože transformery zpracovávají všechna slova současně, potřebují způsob, jak zakódovat informaci o pozici slova ve větě. Každému slovu je přidán unikátní poziční vektor, obvykle pomocí sinusoidálních funkcí.

Multi-head attention rozšiřuje self-attention o možnost zpracovávat text z různých perspektiv současně. Model rozdělí attention mechanismus do několika "hlav", z nichž každá se může zaměřit na jiné části. Například na syntaktické vztahy nebo sémantické podobnosti (TrueFoundry, 2024).

Architektura umožňuje zachytit dlouhodobé závislosti v textu a skvěle se škáluje na GPU, což vedlo k vývoji modelů s desítkami až stovkami miliard parametrů.

2.1.3 Populární open-source modely

Open-source modely nabízejí alternativu ke komerčním řešením a umožňují organizacím provozovat je na vlastní infrastruktuře. Llama od společnosti Meta je dostupná ve verzích od 7 miliard do 400 miliard parametrů. Llama 3.3 s 70 miliardami parametrů vyžaduje přibližně 43 GB paměti Mistral od francouzské společnosti Mistral AI je známý efektivitou. Například verze s 7B parametrů potřebuje pouze 4,1 GB paměti.

Gemma od Google nabízí velikosti od 1B do 27B parametrů, přičemž nejmenší verze potřebuje pouze 815 MB paměti. Mezi další významné modely patří Phi 4 od Microsoftu s 14B parametrů, DeepSeek-R1 dostupný v 7B i 671B variantách a QwQ s 32B parametrů (Ollama Documentation, 2025).

Pro provoz modelů platí obecné pravidlo, že menší modely s 7B parametrů vyžadují minimálně 8 GB RAM (Random Access Memory - operační paměť), střední modely s 13B potřebují 16 GB RAM a velké modely s 33B a více parametrů vyžadují 32 GB RAM nebo více.

2.2 Platforma Ollama

Ollama je open-source platforma zaměřená na zjednodušení nasazení a provozu velkých jazykových modelů. Hlavním cílem platformy je zpřístupnit LLM vývojářům a organizacím bez nutnosti hlubokých znalostí o infrastruktuře a správě modelů. Ollama nabízí jednoduché API, podporu pro Docker a širokou škálu open-source modelů.

2.2.1 Úvod do Ollama

Ollama je open-source platforma navržená pro snadné spouštění velkých jazykových modelů lokálně. Hlavní výhodou je jednoduchost nasazení. Jediný příkaz dokáže automaticky stáhnout a spustit vybraný model. Platforma je dostupná pro Linux, macOS i Windows a nabízí knihovny pro Python, JavaScript a Go.

Ollama poskytuje RESTful API kompatibilní s OpenAI API, což usnadňuje migraci existujících aplikací z komerčních služeb (Pangan, 2025). Významnou výhodou je podpora Docker kontejnerizace, která zjednodušuje nasazení v cloudových prostředích. Platforma automaticky spravuje paměť GPU a optimalizuje využití zdrojů. Umožňuje načítání a přepínání mezi modely bez restartování služby (Monsur, 2025).

2.2.2 Podporované modely

Ollama podporuje širokou škálu open-source modelů včetně Llama (verze 2, 3.1, 3.2, 3.3, 4), Mistral, Mixtral, Gemma, Phi, DeepSeek-R1 a QwQ. K dispozici jsou také multimodální modely jako Llama 3.2 Vision (11B a 90B), LLaVA (7B) a Moondream 2 (1,4B). Pro programování slouží Code Llama a pro embeddings embeddinggemma (Ollama Documentation, 2025).

2.2.3 API a komunikační rozhraní

Ollama poskytuje API běžící na portu 11434 s hlavními endpointy: /api/generate pro generování odpovědí, /api/chat pro konverzační režim s historií a /api/embeddings pro získání embeddings (LiteLLM Documentation, 2025).

API podporuje streaming odpovědí pro průběžné zobrazování textu, asynchronní operace a JSON režim pro strukturovaná data. Novější modely podporují tool calling pro volání externích funkcí (LiteLLM Documentation, 2025). Pro práci s API existují oficiální knihovny ollama-python a ollama-js, nebo lze využít LiteLLM pro jednotné rozhraní.

2.3 Amazon Web Services (AWS)

AWS je vedoucí poskytovatel cloudových služeb na globálním trhu. Platforma nabízí komplexní řešení pro výpočetní výkon, úložiště, síťové služby a mnoho dalších oblastí. Pro implementaci self-hosted LLM řešení jsou klíčové služby EC2 pro výpočetní kapacitu s GPU, VPC pro síťovou infrastrukturu a Network Load Balancer s API Gateway pro zpřístupnění služby.

2.3.1 Přehled AWS služeb

Amazon Web Services je nejrozsáhlejší cloudová platforma na světě, spuštěná v roce 2006. AWS nabízí více než 200 služeb z datových center po celém světě a je využívána miliony zákazníků včetně rychle rostoucích startupů, velkých podniků a vládních agentur pro snížení nákladů a zvýšení agility (AWS Documentation, 2025).

Hlavní výhodou AWS je široká nabídka služeb pokrývající výpočetní výkon, úložiště, databáze, síťové služby, strojové učení a bezpečnost. AWS disponuje největší a nejrozsáhlejší globální infrastrukturou, která zajišťuje škálovatelnost, výkon a spolehlivost aplikací. Pro organizace představuje platforma ideální prostředí pro nasazení self-hosted LLM řešení díky flexibilitě, bezpečnosti a dostupnosti specializovaných GPU instancí (AWS Documentation, 2025).

2.3.2 EC2 a GPU instance

Amazon Elastic Compute Cloud (EC2) je základní výpočetní služba AWS poskytující škálovatelnou výpočetní kapacitu v cloudu. EC2 umožňuje spouštět virtuální servery s různými konfiguracemi CPU, paměti, úložiště a síťové kapacity.

Pro provoz velkých jazykových modelů jsou klíčové GPU instance z rodiny G a P. Instance typu G jsou optimalizované pro graficky náročné aplikace a strojové učení. Instance g5.xlarge je

vybavena GPU NVIDIA A10G s 24 GB paměti, 4 virtuálními CPU a 16 GB RAM, což je vhodné pro středně velké LLM modely.

Pro náročnější úlohy slouží instance typu P s výkonnějšími GPU. Instance p3.2xlarge obsahuje GPU NVIDIA V100 s 16 GB paměti, zatímco novější p4d.24xlarge je vybavena osmi GPU A100 s celkovou pamětí 320 GB.

Cenový model nabízí několik možností. On-Demand instance se platí po hodinách bez závazků. Spot instance využívají nevyužitou kapacitu AWS se slevami až 90%, ale mohou být kdykoliv ukončeny. Reserved Instances nabízejí slevy až 75% výměnou za závazek na jeden nebo tři roky.

2.3.3 Síťové služby (VPC, NLB, API Gateway)

Amazon Virtual Private Cloud (VPC) umožňuje vytvořit izolovanou virtuální síť v rámci AWS cloudu s plnou kontrolou nad síťovým prostředím včetně výběru IP adres, vytváření podsítí a konfigurace směrovacích tabulek. Pomocí VPC lze vytvořit bezpečné privátní prostředí pro EC2 instance s LLM modely.

Network Load Balancer (NLB) distribuuje příchozí HTTP/HTTPS provoz mezi více cílů, jako jsou EC2 instance. NLB funguje na aplikační vrstvě a umožňuje pokročilé směrování. Pro LLM nasazení zajišťuje vysokou dostupnost a automatickou distribuci zátěže.

Amazon API Gateway je plně spravovaná služba pro vytváření, publikování a správu API. Poskytuje funkce jako autentizace, rate limiting, monitoring a transformace požadavků. Pro LLM služby umožňuje vytvořit veřejně přístupné REST API s bezpečným přístupem.

2.3.4 VPC Link

VPC Link je komponenta API Gateway, která umožňuje bezpečné propojení mezi veřejným API Gateway a privátními zdroji ve VPC. Bez VPC Link by bylo nutné zdroje ve VPC vystavit veřejnému internetu, což představuje bezpečnostní riziko.

VPC Link vytváří privátní spojení mezi API Gateway a Network Load Balancer v privátní podsíti VPC. Tím umožňuje přístup k interním službám bez jejich vystavení na internet. Pro LLM nasazení to znamená, že EC2 instance s Ollama mohou zůstat v privátní síti, zatímco API Gateway poskytuje kontrolovaný veřejný přístup.

2.3.5 Infrastructure as Code a Terraform

Infrastructure as Code (IaC) je přístup k správě infrastruktury pomocí kódu místo manuálních procesů. Terraform je open-source nástroj od společnosti HashiCorp, který umožňuje definovat cloudovou infrastrukturu deklarativním způsobem v konfiguračních souborech.

Terraform používá vlastní konfigurační jazyk HCL (HashiCorp Configuration Language) pro popis požadovaného stavu infrastruktury. Hlavní výhodou je možnost verzovat infrastrukturu v Git repozitáři, sdílet konfiguraci v týmu a automatizovat nasazení. Terraform podporuje širokou škálu poskytovatelů cloudových služeb včetně AWS, Azure a Google Cloud.

Pro účely implementace byla infrastruktura primárně nasazována pomocí AWS CLI pro detailní pochopení jednotlivých kroků. V repozitáři projektu je však k dispozici také Terraform

konfigurace jako alternativní způsob nasazení, která umožňuje automatizované a reprodukovatelné vytvoření celé infrastruktury jedním příkazem.

2.4 Kontejnerizace a Docker

Kontejnerizace představuje moderní přístup k nasazování a správě aplikací. Na rozdíl od tradičních virtuálních strojů nabízí kontejnery lehčí a efektivnější řešení s rychlejším startem a nižší režií. Docker se stal standardem pro kontejnerizaci a je široce podporován v cloudových prostředích včetně AWS.

2.4.1 Principy kontejnerizace

Kontejnerizace je forma virtualizace na úrovni operačního systému, která vytváří izolované virtuální jednotky zvané kontejnery. Na rozdíl od tradičních virtuálních strojů sdílejí kontejnery stejné jádro hostitelského operačního systému, ale jsou od sebe izolovány prostřednictvím privátních jmenných prostorů (GeeksforGeeks, 2025).

Hlavní výhodou kontejnerizace je absence režie spojené s virtualizací hardwaru. Kontejnery implementují izolaci procesů na úrovni operačního systému a v kontejnerech není nutné předem alokovat RAM. Ta je totiž alokována dynamicky, zatímco u virtuálních strojů je nutné paměť přidělit předem.

Kontejnery poskytují lepší využití zdrojů a mají rychlejší spouštění než virtuální stroje. Mohou běžet prakticky kdekoli. Na Linuxu, Windows i macOS, na virtuálních strojích nebo fyzických serverech. Kontejnery virtualizují CPU, paměť, úložiště a síťové zdroje na úrovni operačního systému.

2.4.2 NVIDIA Container Toolkit

NVIDIA Container Toolkit umožňuje přístup k GPU z Docker kontejnerů. Bez tohoto toolkitu by kontejnery neměly přístup k GPU akceleraci potřebné pro LLM modely.

Toolkit zahrnuje libnvidia-container pro konfiguraci GPU kontejnerů, NVIDIA Container Runtime pro integraci GPU podpory a NVIDIA Container Toolkit pro konfigurační nástroje (NVIDIA Documentation, 2025).

Instalace vyžaduje nejprve NVIDIA GPU ovladač, poté přidání repozitáře a instalaci balíčků. Po instalaci se nakonfiguruje Docker daemon příkazem nvidia-ctk runtime configure --runtime=docker a restartuje se Docker služba (NVIDIA Documentation, 2025).

Po konfiguraci lze kontejnery spouštět s --gpus all parametrem. Toolkit automaticky zpřístupní GPU zařízení a CUDA knihovny, což umožňuje Ollama využít plný výkon GPU pro inference modelů.

2.5 GPU computing

GPU computing je nezbytnou technologií pro efektivní provoz velkých jazykových modelů. Grafické procesory poskytují masivní paralelní výpočetní výkon, který je ideální pro operace

s maticemi a tenzory používané v neuronových sítích. Pochopení principů GPU computing a jejich optimalizace je klíčové pro dosažení dobrého výkonu a efektivity řešení.

2.5.1 CUDA a GPU akcelerace

CUDA (Compute Unified Device Architecture) je paralelní výpočetní platforma vyvinutá společností NVIDIA. CUDA umožňuje vývojářům využít výpočetní výkon GPU pro obecné výpočty, nejen pro grafiku (NVIDIA GPU Gems, 2025).

GPU se od CPU liší architekturou. CPU (Central Processing Unit - procesor) obsahuje několik výkonných jader pro sekvenční zpracování, GPU obsahuje tisíce menších jader pro paralelní zpracování. Architektura je ideální pro operace s maticemi a tenzory, které jsou základem neuronových sítí a velkých jazykových modelů.

CUDA poskytuje programovací rozhraní v C, C++ a Fortran. Pro Python existují knihovny jako CuPy. Moderní frameworky jako PyTorch a TensorFlow automaticky využívají CUDA akceleraci, pokud je dostupná GPU.

2.5.2 Výpočetní výkon pro AI/ML

GPU poskytují výrazně vyšší výpočetní výkon pro neuronové sítě ve srovnání s CPU. GPU NVIDIA A10G nabízí 31,2 TFLOPS pro single-precision výpočty a 125 TFLOPS s Tensor Cores pro mixed-precision výpočty.

Pro LLM je klíčová velikost paměti GPU. Model s několika miliardami parametrů vyžaduje desítky gigabytů paměti. GPU A10G s 24 GB dokáže provozovat modely do velikosti přibližně 20-25 miliard parametrů.

Kvantizace snižuje paměťové nároky redukcí přesnosti čísel reprezentujících váhy modelu. Místo 32-bitových float čísel se používají 16-bitové, 8-bitové nebo 4-bitové representace, což umožňuje provozovat větší modely s minimálním dopadem na kvalitu.

2.5.3 Optimalizace využití GPU

Ollama automaticky optimalizuje využití GPU dynamickou správou paměti a podporuje batch processing pro zpracování více požadavků současně. Pro modely, které se nevejdou do VRAM (Video Random Access Memory - paměť GPU), implementuje offloading mezi RAM a GPU.

AWS instance jako p4d.24xlarge s osmi GPU A100 umožňují distribuovat model nebo provozovat více modelů paralelně. Monitoring využití GPU je dostupný pomocí nástroje nvidia-smi, který zobrazuje využití GPU, spotřebu paměti a teplotu

3 Návrh architektury

Návrh architektury pro provoz self-hosted LLM v AWS vyžaduje pečlivé zvážení výpočetních zdrojů, síťové topologie a bezpečnostních mechanismů. Cílem je vytvořit škálovatelné a bezpečné řešení, které kombinuje výkon GPU instancí s flexibilitou cloudových služeb. Následující kapitola popisuje klíčové komponenty navržené architektury a zdůvodnění jejich výběru.

3.1 Architektura systému

Architektura systému definuje způsob propojení jednotlivých komponent a jejich vzájemnou komunikaci. Správný návrh architektury je zásadní pro zajištění bezpečnosti, výkonu a škálovatelnosti celého řešení. V sekci je představen diagram architektury a popis jednotlivých vrstev systému.

3.1.1 Diagram architektury

Navržená architektura kombinuje EC2 GPU instance s moderními síťovými službami AWS. Systém je navržen tak, aby EC2 instance s Ollama zůstaly v privátní síti bez přímého přístupu z internetu, zatímco veřejný přístup je kontrolován prostřednictvím API Gateway.

Hlavní komponenty tvoří EC2 instance s GPU v privátní podsíti VPC běžící Ollama v Docker kontejneru, Network Load Balancer pro směrování TCP (Transmission Control Protocol) provozu, VPC Link propojující API Gateway s privátními zdroji a API Gateway jako veřejný vstupní bod s autentizací pomocí API klíčů.

Navržená architektura poskytuje ochranu instancí v privátní síti, centrální správu přístupu přes API Gateway, směrování provozu pomocí Load Balanceru a škálovatelnost pro přidávání dalších instancí.

3.2 Výběr AWS služeb

Volba správných AWS služeb má přímý dopad na výkon, náklady a udržitelnost řešení. AWS nabízí širokou škálu služeb pokrývající výpočetní výkon, síťové funkce, úložiště a bezpečnost (No Fluff Jobs, 2025). Pro provoz LLM je klíčový výběr vhodné EC2 instance s GPU, která poskytne dostatečný výpočetní výkon při přijatelných nákladech.

3.2.1 Kritéria pro výběr EC2 instance

Výběr správného typu EC2 instance je klíčový pro výkon a náklady. Pro LLM jsou zásadní GPU instance z rodiny G optimalizované pro strojové učení (AWS EC2 Documentation, 2025).

Instance g5.xlarge byla vybrána jako primární varianta. Obsahuje GPU NVIDIA A10G s 24 GB paměti, 4 vCPU, 16 GB RAM a 250 GB NVMe SSD. Konfigurace g5.xlarge je vhodná pro modely do 20-25 miliard parametrů. Cena je přibližně 1,006 USD za hodinu pro On-Demand (No Fluff Jobs, 2025).

Pro větší modely lze využít g5.2xlarge s 32 GB RAM nebo g5.4xlarge s 64 GB RAM. Při výběru je třeba zvážit velikost modelu, očekávaný provoz a rozpočet. Cenové modely zahrnují On-Demand pro flexibilitu, Spot pro úspory až 90% a Reserved pro slevy až 75% při dlouhodobém provozu.

3.3 Síťová topologie a bezpečnost

Správná konfigurace síťové topologie a bezpečnostních mechanismů je zásadní pro ochranu dat a zabránění neoprávněnému přístupu. VPC umožňuje vytvořit izolovanou virtuální síť s plnou kontrolou nad síťovým prostředím. Bezpečnostní skupiny pak fungují jako první linie obrany proti nežádoucímu provozu a definují, jaký provoz je povolen do a z jednotlivých zdrojů.

3.3.1 Bezpečnostní skupiny (Security Groups)

Bezpečnostní skupiny fungují jako virtuální firewall kontrolující provoz pro AWS zdroje. Při vytváření VPC je automaticky vytvořena výchozí skupina, lze však vytvořit další s vlastními pravidly (AWS VPC Documentation, 2025).

Pro EC2 instance s Ollama je navržena skupina povolující TCP provoz (port 11434) pouze z Network Load Balanceru a SSH (Secure Shell - zabezpečený vzdálený přístup, port 22) z konkrétních IP adres pro správu. Odchozí provoz je povolen pro stahování modelů.

Pro Network Load Balancer je vytvořena samostatná skupina povolující TCP provoz pouze z VPC Link a odchozí provoz k EC2 instancím na portu 11434.

Bezpečnostní skupiny jsou stavové. Pokud je povolen příchozí provoz, odpovědi jsou automaticky povoleny (AWS VPC Documentation, 2025). Doporučuje se vytvořit minimální počet skupin a otevírat pouze nezbytné porty.

3.4 Mechanismy řízení přístupu

Řízení přístupu k API je klíčové pro zabezpečení řešení a kontrolu využití zdrojů. API Gateway poskytuje několik mechanismů autentizace a autorizace, přičemž API klíče představují jednoduchý a efektivní způsob identifikace klientů a sledování jejich využití. Kromě autentizace umožňují API klíče v kombinaci s usage plány implementovat různé úrovně služeb a limity pro jednotlivé uživatele.

3.4.1 API Key autentizace

API Gateway poskytuje autentizaci pomocí API klíčů pro kontrolu přístupu a sledování využití. API klíče jsou alfanumerické řetězce identifikující aplikaci volající API (AWS API Gateway Documentation, 2025).

Pro implementaci se nakonfigurují API metody tak, aby vyžadovaly API klíč aktivací volby "API key required". Po vytvoření klíče v konzoli se asociuje s usage plánem definujícím kvóty a throttling limity (AWS API Gateway Documentation, 2025).

Klienti posílají API klíč v HTTP hlavičce x-api-key. API Gateway validuje klíč před předáním požadavku. Neplatné klíče jsou zamítnuty s HTTP 403 chybou.

Usage plány umožňují definovat kvóty jako 10000 požadavků za měsíc a throttling například 100 požadavků za sekundu. Různé plány lze vytvořit pro různé úrovně služeb s různými limity a cenou.

4 Implementace

Implementační fáze převádí navržený koncept do funkčního řešení. Každý krok vyžaduje pečlivou konfiguraci a testování, aby bylo zajištěno správné fungování všech komponent.

Pro nasazení infrastruktury byla zvolena cesta pomocí AWS CLI, která umožňuje detailní pochopení jednotlivých kroků a komponent. V rámci projektu byla také vytvořena Terraform konfigurace dostupná v GitHub repozitáři, která poskytuje alternativní způsob automatizovaného nasazení celé infrastruktury. Terraform varianta je vhodná pro produkční prostředí, kde je požadována rychlá reprodukovatelnost a verzování infrastruktury.

Kapitola popisuje detailní postup manuálního nasazení pomocí AWS CLI od přípravy EC2 instance přes instalaci softwaru až po zpřístupnění API a testování funkčnosti.

4.1 Příprava EC2 instance s GPU

Prvním krokem implementace je vytvoření a konfigurace EC2 instance s GPU. Správná příprava instance je zásadní pro následující kroky instalace a zajišťuje, že systém bude mít přístup k potřebným hardwarovým zdrojům a bude správně nakonfigurován pro provoz LLM modelů.

4.1.1 Automatizace nasazení

Pro zjednodušení a zrychlení nasazení byl vytvořen automatizační bash skript setup-gpu.sh, který provádí všechny kroky instalace a konfigurace při prvním startu EC2 instance. Skript se spouští jako EC2 user_data a automaticky instaluje NVIDIA ovladače, Docker, NVIDIA Container Toolkit, spouští Ollama kontejner a stahuje model Llama 3.2 (3B parametrů). Následující sekce popisují jednotlivé kroky, které skript provádí. Pro manuální redeployment Ollama kontejneru je k dispozici skript deploy-ollama.sh.

4.1.2 Výběr typu instance

Pro nasazení Ollama byla vybrána instance g5.xlarge s GPU NVIDIA A10G. Instance se vytvoří v AWS konzoli v sekci EC2 s výběrem operačního systému Ubuntu Server 24.04 LTS, který poskytuje dobrou podporu pro Docker a NVIDIA ovladače. Při vytváření instance je důležité vybrat privátní podsíť ve VPC a přiřadit bezpečnostní skupinu nakonfigurovanou podle návrhu z kapitoly číslo 3.

Pro úložiště se doporučuje minimálně 100 GB EBS volume typu gp3 pro operační systém a Docker images. Instance se spouští s SSH klíčem pro bezpečný vzdálený přístup.

4.1.3 Konfigurace operačního systému

Po spuštění instance je nutné se připojit přes SSH a provést základní konfiguraci systému. Aktualizace systémových balíčků se provede příkazy sudo apt-get update a sudo apt-get upgrade. Nainstalují se potřebné nástroje jako curl, wget, git a build-essential.

4.1.4 Instalace NVIDIA driverů

NVIDIA doporučuje instalaci ovladačů pomocí package manageru dané distribuce (NVIDIA Documentation, 2025). Pro Ubuntu se přidá NVIDIA PPA repozitář příkazem sudo add-apt-repository ppa:graphics-drivers/ppa. Po aktualizaci seznamu balíčků se nainstaluje nejnovější doporučený ovladač příkazem sudo apt-get install nvidia-driver-535.

Po instalaci je nutné restartovat instanci. Funkčnost ovladače se ověří příkazem nvidia-smi, který zobrazí informace o GPU včetně využití, teploty a dostupné paměti.

4.2 Instalace Docker a NVIDIA Container Toolkit

Docker poskytuje kontejnerizaci aplikací a NVIDIA Container Toolkit umožňuje kontejnerům přístup k GPU. Kombinace obou nástrojů vytváří izolované a reprodukovatelné prostředí pro provoz Ollama.

4.2.1 Instalační proces Docker

Instalace Dockeru začíná přidáním oficiálního Docker repozitáře. Nejprve se nainstalují prerekvizity: sudo apt-get install ca-certificates curl gnupg. Přidá se Docker GPG klíč a repozitář. Následně se nainstaluje Docker Engine příkazem sudo apt-get install docker-ce docker-ce-cli containerd.io.

Pro možnost spouštění Docker příkazů bez sudo se aktuální uživatel přidá do docker skupiny příkazem sudo usermod -aG docker $USER. Funkčnost Dockeru se ověří spuštěním testovacího kontejneru docker run hello-world.

4.2.2 Konfigurace NVIDIA Container Toolkit

Instalace NVIDIA Container Toolkit umožňuje Docker kontejnerům využívat GPU. Proces instalace začíná přidáním NVIDIA Container Toolkit repozitáře (NVIDIA Documentation, 2025). Pro Ubuntu se použije příkaz pro konfiguraci produkčního repozitáře.

Po aktualizaci seznamu balíčků se nainstalují potřebné balíčky včetně nvidia-container-toolkit a libnvidia-container. Následuje konfigurace Docker runtime příkazem sudo nvidia-ctk runtime configure --runtime=docker, který upraví soubor /etc/docker/daemon.json.

Po restartu Docker služby příkazem sudo systemctl restart docker se funkčnost ověří spuštěním testovacího kontejneru s GPU: docker run --rm --gpus all nvidia/cuda:12.0-base nvidia-smi.

4.3 Nasazení Ollama v kontejneru

Ollama poskytuje oficiální Docker image, který zjednodušuje nasazení. Kontejnerizace zajišťuje konzistentní prostředí a snadnou správu.

4.3.1 Příprava Docker image

Oficiální Ollama image se stáhne z Docker Hub příkazem docker pull ollama/ollama. Image obsahuje veškerý potřebný software včetně Ollama serveru a podpory pro NVIDIA GPU.

Pro produkční použití se doporučuje vytvořit vlastní Dockerfile s dodatečnou konfigurací, například nastavením prostředí nebo přidáním monitorovacích nástrojů.

4.3.2 Konfigurace Ollama

Ollama se spouští v Docker kontejneru s parametrem --gpus all pro přístup k GPU. Port 11434 se mapuje na host systém pro přístup k API. Volume se vytvoří pro perzistentní uložení modelů: docker volume create ollama.

Kontejner se spustí příkazem s restartem při selhání: docker run -d --gpus all -v ollama:/root/.ollama -p 11434:11434 --name ollama --restart unless-stopped ollama/ollama.

4.3.3 Stažení a spuštění modelu

Po spuštění kontejneru se stáhne požadovaný model příkazem docker exec ollama ollama pull llama3.2. Model se automaticky uloží do volume a je připraven k použití. Funkčnost se ověří testovacím dotazem přes API nebo příkazem docker exec ollama ollama run llama3.2.

4.4 Konfigurace Network Load Balancer

Network Load Balancer směruje TCP provoz na EC2 instance a poskytuje health checking pro zajištění dostupnosti služby. NLB pracuje na síťové vrstvě (Layer 4) a je vyžadován pro VPC Link integrace s REST API Gateway.

4.4.1 Vytvoření Target Group

Target Group definuje cíle, kam NLB směruje požadavky. V AWS konzoli se vytvoří nová Target Group typu "Instance" s TCP protokolem na portu 11434. Jako health check se použije TCP protokol na portu 11434.

Do Target Group se registruje EC2 instance s Ollama. Health check interval se nastaví na 30 sekund s timeoutem 10 sekund a healthy threshold 3 úspěšné kontroly.

4.4.2 Nastavení health checks

Health checks zajišťují, že NLB směruje provoz pouze na zdravé instance. Ollama na portu 11434 odpovídá na TCP spojení, což je dostačující pro TCP health check. NLB pravidelně testuje dostupnost instance a při selhání přesměruje provoz na jiné zdravé instance.

4.4.3 Konfigurace load balanceru

Network Load Balancer se vytvoří v AWS konzoli jako internal (ne internet-facing), protože bude přístupný pouze přes VPC Link. NLB se umístí do stejné VPC jako EC2 instance, ale do jiné podsítě pro separaci vrstev.

Listener se nakonfiguruje na portu 11434 s TCP protokolem a směrováním na vytvořenou Target Group. Bezpečnostní skupina NLB povoluje příchozí TCP provoz na portu 11434 pouze z VPC Link.

4.5 Zpřístupnění přes API Gateway a VPC Link

API Gateway poskytuje veřejný endpoint s autentizací, zatímco VPC Link zajišťuje bezpečné propojení s privátními zdroji (AWS Blog, 2025).

4.5.1 Vytvoření VPC Link

VPC Link pro REST API se vytvoří v AWS konzoli v sekci API Gateway. Jako target se vybere Network Load Balancer vytvořený v předchozím kroku. VPC Link vytvoří VPC endpoint service a automaticky schválí připojení z API Gateway účtu (AWS Blog, 2025).

Proces vytvoření trvá několik minut. Status VPC Link musí být "Available" před pokračováním dalšími kroky.

4.5.2 Konfigurace API Gateway

V API Gateway se vytvoří nová REST API. Pro root resource se vytvoří metoda ANY, která přijímá všechny HTTP metody. Integration type se nastaví na "VPC Link" s výběrem vytvořeného VPC Link a endpoint URL směřující na NLB ve formátu http://nlb-internal-dns:11434/{proxy}.

Pro předávání všech path parametrů se v Method Execution nastaví Integration Request s mapping template pro proxy pass všech požadavků.

4.5.3 Integrace s NLB

API Gateway posílá požadavky přes VPC Link na NLB, který je distribuuje na EC2 instance. V konfiguraci integrace se nastaví timeout na 29 sekund (maximum pro API Gateway) (AWS API Gateway Documentation, 2025).

Custom headers se mohou přidat pro identifikaci požadavků přicházejících z API Gateway. NLB pak požadavky směruje na Ollama podle konfigurace Target Group.

4.6 Implementace autentizace a bezpečnosti

Bezpečnost API je zajištěna pomocí API klíčů a usage plánů, které kontrolují přístup a využití zdrojů.

4.6.1 API Key management

V API Gateway se vytvoří API klíče v sekci API Keys. Každý klíč má unikátní hodnotu a volitelný popis pro identifikaci klienta. Klíče mohou být automaticky generované nebo vlastní (AWS API Gateway Documentation, 2025).

Pro každou API metodu se aktivuje "API key required" v Method Request settings. Tím se zajistí, že požadavky bez platného API klíče budou odmítnuty.

4.6.2 Implementace IP whitelisting

Pro dodatečné zabezpečení se implementuje IP whitelisting pomocí Resource Policy v API Gateway. Policy definuje, které IP adresy mají povolený přístup k API. JSON policy obsahuje Allow statement s podmínkou pro povolené IP rozsahy.

Resource Policy se aplikuje na celé API a je vyhodnocována před API key autentizací. Požadavky z nepovolených IP adres jsou odmítnuty s HTTP 403 chybou.

4.7 Příklady použití API a testování

Po dokončení implementace je API připraveno k použití. Následující příklady ukazují praktické příklady volání API různými způsoby.

4.7.1 Dotazování přes cURL

Základní dotaz pomocí cURL obsahuje API klíč v hlavičce x-api-key. Příklad: curl -X POST https://api-id.execute-api.region.amazonaws.com/prod/api/generate -H "x-api-key: your-api-key" -H "Content-Type: application/json" -d '{"model": "llama3.2", "prompt": "Hello, world!"}'

Pro streaming odpovědí se přidá parametr stream: true. cURL pak zobrazuje data postupně jak jsou generována.

4.7.2 Python client

V repozitáři projektu je k dispozici Python klient examples/client.py, který demonstruje základní operace s API. Klient používá knihovnu requests a poskytuje tři hlavní funkce pro práci s Ollama API. Funkce generate(prompt) slouží pro generování textu z promptu, funkce chat(messages) umožňuje konverzaci s historií zpráv a funkce list_models() vrací výpis dostupných modelů na serveru.

4.7.3 Funkční testy

Skript tests/test_api.py provádí automatizované funkční testy API. Testování zahrnuje ověření konektivity k API endpointu, autentizaci pomocí API klíče s validací jak platného, tak neplatného klíče, generování textu s různými typy promptů a zpracování chybových stavů. Testy se spouštějí příkazem: python tests/test_api.py <API_URL> <API_KEY>.

4.7.4 Výkonnostní benchmark

Skript tests/benchmark.py měří výkonnostní charakteristiky API. Benchmark provádí komplexní testování zahrnující měření latence pro prompty různých délek (krátké, střední a dlouhé), time-to-first-token (TTFT) pro zjištění doby do první odpovědi, throughput při různém počtu souběžných požadavků (1, 2, 5 a 10) a rychlost generování tokenů. Výsledky benchmarku jsou použity v kapitole 5 pro vyhodnocení výkonu systému.

5 Testování a vyhodnocení

Po dokončení implementace je nezbytné provést důkladné testování a vyhodnocení řešení. Testování ověřuje funkčnost všech komponent, měří výkonnostní charakteristiky a identifikuje potenciální problémy. Kapitola popisuje různé testovací scénáře, analýzu výkonu a vyhodnocení provozních nákladů, které jsou klíčové pro posouzení efektivity navrženého řešení.

5.1 Testovací scénáře

Testovací scénáře pokrývají různé aspekty systému od základní funkčnosti přes bezpečnost až po zátěžové testy. Komplexní testování zajišťuje, že řešení je připraveno pro produkční nasazení a dokáže zvládnout očekávanou zátěž.

5.1.1 Funkční testování

Funkční testování ověřuje, že všechny komponenty systému fungují správně a komunikují mezi sebou. Testuje se správné předávání požadavků od API Gateway přes VPC Link a Network Load Balancer až k Ollama na EC2 instanci.

První test ověřuje základní konektivitu zasláním jednoduchého požadavku na API endpoint s platným API klíčem. Úspěšná odpověď potvrzuje, že celá integrace funguje. Druhý test ověřuje generování textu s různými modely a prompty pro kontrolu správného fungování Ollama.

Testují se také chybové stavy. Požadavek bez API klíče musí vrátit HTTP 403 chybu. Požadavek na neexistující model musí být korektně zpracován s příslušnou chybovou hláškou. Health check mechanismus NLB se testuje dočasným zastavením Ollama kontejneru a ověřením, že NLB přestane směrovat provoz na nefunkční instanci.

5.1.2 Bezpečnostní testování

Bezpečnostní testování ověřuje, že implementované bezpečnostní mechanismy chrání systém před neoprávněným přístupem. Testuje se autentizace pomocí API klíčů zasláním požadavků s neplatným klíčem, který musí být odmítnut.

IP whitelisting se testuje odesláním požadavků z IP adres mimo povolený rozsah. Požadavky musí být odmítnuty na úrovni API Gateway před dosažením backend systému. Testuje se také, že EC2 instance v privátní síti není dostupná z internetu a je přístupná pouze přes NLB.

5.2 Měření výkonu a latence

Měření výkonu poskytuje objektivní data o rychlosti a efektivitě systému. Latence a propustnost jsou klíčové metriky pro hodnocení uživatelské zkušenosti.

5.2.1 Metodika měření

Měření výkonu se provádělo pomocí Python skriptu, který odesílá požadavky na API endpoint a měří dobu odpovědi. Testy byly opakovány mnohokrát pro získání statisticky významných výsledků s výpočtem průměru a standardní odchylky.

Měřila se celková latence od odeslání požadavku po přijetí kompletní odpovědi. Pro streaming odpovědi byl zvlášť měřen time-to-first-token (TTFT), který udává dobu do přijetí prvního tokenu odpovědi. Throughput byl měřen při různém počtu souběžných požadavků (1, 2, 5, 10) pro zjištění kapacity systému.

Testy byly provedeny s modelem Llama 3.2 (3B parametrů) na EC2 instanci g5.xlarge s GPU NVIDIA A10G.

5.2.2 Výsledky pro různé typy dotazů

Latence výrazně závisí na délce vstupního promptu a generované odpovědi. Výsledky měření pro různé kategorie promptů:

tab. 3: Latence promptů různé délky

Typ promptu	Délka vstupu	Průměrná latence	Průměr tokenů	Tokeny	Std.dev.
Krátký	12 znaků	0,71 s	8 tokenů	12	0,09 s
Střední	113 znaků	1,37 s	115 tokenů	84	0,12 s
Dlouhý	790 znaků	4,46 s	644 tokenů	144	0,39 s

zdroj: vlastní zpracování

Time-to-first-token (TTFT) byl relativně konstantní napříč všemi typy promptů s průměrnou hodnotou 981 ms (minimum 912 ms). Hodnota zahrnuje zpracování promptu, inicializaci generování a network overhead.

Zajímavým poznatkem je, že rychlost generování tokenů roste s délkou odpovědi. Krátké odpovědi dosahují pouze ~12 tokenů za sekundu, zatímco dlouhé odpovědi dosahují ~144 tokenů za sekundu. To je způsobeno tím, že TTFT představuje konstantní overhead přibližně 1 sekundu, zatímco samotná generace tokenů je velmi rychlá.

Throughput systému byl měřen při různých úrovních souběžnosti:

tab. 4: Throughput (propustnost)

Souběžné požadavky	Požadavků za sekundu	Škálování
1	1,4	1,0x
2	2,8	2,0x
5	5,4	3,9x
10	7,0	5,0x

zdroj: vlastní zpracování

Throughput škáluje téměř lineárně do 2 souběžných požadavků (2,8 req/s = 2x baseline), poté sublineárně s klesající efektivitou. GPU zvládá batch processing více požadavků současně, ale s rostoucím počtem požadavků klesá efektivita využití zdrojů.

Nízká variance napříč měřeními (standardní odchylka 0,09-0,39 s) ukazuje na stabilní a předvídatelné chování systému.

5.3 Analýza provozních nákladů

Analýza nákladů je zásadní pro ekonomické zhodnocení řešení a porovnání s komerčními alternativami. Celkové náklady zahrnují výpočetní zdroje, síťové služby a úložiště.

5.3.1 Náklady na EC2 instance

Instance g5.xlarge s On-Demand pricing stojí přibližně 1,006 USD za hodinu, což při nepřetržitém provozu činí 730 USD měsíčně (No Fluff Jobs, 2025). Spot instance mohou snížit náklady až o 70% na přibližně 0,30 USD za hodinu, tedy 219 USD měsíčně, ale s rizikem přerušení.

Reserved Instance s jednoleté závazkem poskytuje slevu přibližně 30% na 0,70 USD za hodinu, tedy 511 USD měsíčně. Tříletý závazek nabízí slevy až 50%. EBS storage s 100 GB gp3 volume přidává přibližně 8 USD měsíčně.

5.3.2 Náklady na síťové služby

Network Load Balancer stojí 0,0225 USD za hodinu tedy přibližně 16,43 USD měsíčně plus 0,008 USD za LCU hodinu. API Gateway má cenu 3,50 USD za milion požadavků pro prvních 333 milionů požadavků měsíčně.

VPC Link nemá žádné dodatečné poplatky nad rámec nákladů na NLB. Data transfer out má cenu 0,09 USD za GB pro prvních 10 TB měsíčně. Pro aplikaci generující 1 GB dat denně to činí přibližně 2,70 USD měsíčně.

5.3.3 Celkové měsíční náklady

Celkové měsíční náklady pro On-Demand instanci: EC2 instance 730 USD, NLB 20 USD, API Gateway 35 USD pro 10 milionů požadavků, EBS storage 8 USD, data transfer 3 USD. Celkem přibližně 796 USD měsíčně.

Pro Spot instance se náklady snižují na přibližně 285 USD měsíčně. Pro srovnání, komerční řešení jako OpenAI API by pro 10 milionů tokenů měsíčně stálo přibližně 100-200 USD v závislosti na modelu, ale bez kontroly nad daty a infrastrukturou.

Self-hosted řešení se stává ekonomicky výhodným při vyšších objemech využití, kde náklady na tokeny u komerčních služeb rychle rostou, zatímco náklady na vlastní infrastrukturu zůstávají relativně konstantní.

5.3.4 Porovnání s komerčními službami a break-even bod

Pro vyhodnocení ekonomické výhodnosti je potřeba porovnat náklady vlastního řešení s komerčními API. Self-hosted řešení s On-Demand g5.xlarge instancí stojí 796 USD měsíčně.

Z komerčních služeb je nejdražší Claude 4 Sonnet s cenou 3 USD za milion vstupních tokenů a 15 USD za milion výstupních tokenů. Při typickém poměru vstup:výstup 1:3 vychází průměrná cena

(3 + 3×15)/4 = 12,75 USD za milion tokenů. Self-hosted řešení se začne vyplácet po 796/12,75 = 62 milionech tokenů měsíčně, což je asi 2 miliony tokenů denně.

Levnější je Gemini 2.5 Pro s cenou 1,25 USD za vstup a 10 USD za výstup. Průměrná cena je (1,25 + 3×10)/4 = 8,19 USD za milion tokenů. Break-even nastává při 97 milionech tokenů měsíčně (3,2 milionu denně).

Z naměřeného výkonu víme, že instance zvládne až 7 požadavků za sekundu s průměrně 256 tokeny na odpověď. To znamená maximálně 154 milionů tokenů denně při plném vytížení.

Vlastní řešení má smysl při větším provozu, zhruba nad 2-3 miliony tokenů denně. Pod touto hranicí jsou komerční API levnější.

6 Škálovatelnost a optimalizace

Po úspěšném nasazení a testování je důležité zvážit možnosti škálování a optimalizace řešení pro budoucí růst a efektivnější využití zdrojů. Kapitola se zaměřuje na strategie pro horizontální škálování systému, které umožní zvládnout rostoucí zátěž, a na metody optimalizace nákladů, které pomohou snížit provozní výdaje při zachování požadovaného výkonu.

6.1 Možnosti horizontálního škálování

Horizontální škálování přidáváním dalších instancí je preferovanou metodou pro zvýšení kapacity systému. Na rozdíl od vertikálního škálování, které má fyzická omezení, horizontální škálování teoreticky umožňuje neomezenou kapacitu. Implementace auto-scalingu zajišťuje automatické přizpůsobení kapacity aktuální zátěži.

6.1.1 Auto-scaling EC2 instancí

AWS Auto Scaling Groups umožňují automatické přidávání nebo odebírání EC2 instancí podle definovaných metrik. Vytvoří se Launch Template s konfigurací instance g5.xlarge včetně AMI s předinstalovaným Dockerem (kontejnerizační platforma), NVIDIA ovladači a Ollama kontejnerem.

Auto Scaling Group se nakonfiguruje s minimálním počtem instancí 1, požadovaným počtem 2 a maximálním počtem 5. Scaling policies se nastaví podle CPU utilizace na scale out při 70% využití CPU a scale in při 30% využití. Warm pool může být nakonfigurován pro rychlejší reakci na zvýšenou zátěž udržováním předpřipravených instancí.

Health checks z Network Load Balancer zajišťují, že nefunkční instance jsou automaticky nahrazeny. Instance distribution across availability zones poskytuje vysokou dostupnost i při výpadku jedné zóny.

6.1.2 Load balancing při více instancích

Network Load Balancer automaticky distribuuje provoz mezi všechny zdravé instance v Target Group. Round-robin algoritmus zajišťuje rovnoměrné rozložení zátěže. Connection draining umožňuje dokončit aktivní požadavky před ukončením instance během scale-in operace.

Cross-zone load balancing zajišťuje rovnoměrnou distribuci i mezi availability zones. Sticky sessions mohou být použity pro případy, kdy je nutné udržet uživatele na stejné instanci, ale pro bezstavové LLM API to není nutné.

6.1.3 Distribuce zátěže

Při škálování na více instancí je důležité monitorovat metriky jako request count per instance, target response time a unhealthy target count. CloudWatch alarmy upozorňují na problémy s distribucí zátěže.

GPU utilizace by měla být relativně vyrovnaná napříč instancemi. Pokud některé instance mají výrazně nižší využití, může to indikovat problém s health checks nebo networkingem. Target

group deregistration delay se nastaví na dostatečnou hodnotu pro dokončení dlouhých inference operací.

6.2 Optimalizace nákladů

Optimalizace nákladů je zásadní pro dlouhodobou udržitelnost řešení. AWS nabízí několik cenových modelů a strategií, které mohou výrazně snížit provozní výdaje při zachování požadované funkčnosti.

6.2.1 Spot instance vs On-Demand

Spot instance využívají nevyužitou kapacitu AWS s výraznými slevami až 90% oproti On-Demand cenám. Pro g5.xlarge klesá cena z 1,006 USD na přibližně 0,30 USD za hodinu. Spot instance mohou být kdykoliv ukončeny s dvouminutovým upozorněním, když AWS potřebuje kapacitu.

Pro LLM inference, která je bezstavová a snadno přenositelná, jsou Spot instance vhodnou volbou. Auto Scaling Group se nakonfiguruje s capacity-optimized allocation strategy, která vybírá instance types s nejnižší pravděpodobností přerušení. Kombinace On-Demand a Spot instancí v poměru 1:3 poskytuje dobrou rovnováhu mezi náklady a dostupností.

Spot instance interruption handling využívá CloudWatch Events pro detekci nadcházejícího ukončení. Instance se automaticky odregistruje z Target Group a nové požadavky jsou směrovány na zbývající instance.

6.2.2 Reserved instances

Reserved Instances nabízejí slevy až 75% výměnou za závazek na jeden nebo tři roky. Pro stabilní workload s předvídatelným využitím je to optimální volba. Jednoleté Reserved Instance pro g5.xlarge poskytuje slevu přibližně 30%, tříleté až 50%.

Convertible Reserved Instances nabízejí flexibilitu pro změnu instance typu během závazku s mírně nižší slevou. Standard Reserved Instances poskytují vyšší slevu, ale méně flexibility. Partial upfront nebo all upfront platba poskytuje dodatečné slevy.

Pro produkční nasazení se doporučuje kombinace Reserved Instances pro base capacity a Spot/On-Demand pro variable load. Cost Explorer a Compute Optimizer pomáhají analyzovat využití a doporučují optimální nákup Reserved Instances.

Závěr

Bakalářská práce se zaměřila na návrh a implementaci self-hosted řešení pro provoz velkého jazykového modelu pomocí platformy Ollama v cloudovém prostředí Amazon Web Services. Cílem bylo vytvořit bezpečnou, škálovatelnou a nákladově efektivní alternativu ke komerčním cloudovým API službám.

V teoretické části byly představeny základní koncepty velkých jazykových modelů, platforma Ollama a klíčové služby AWS potřebné pro implementaci řešení. Pozornost byla věnována také kontejnerizaci pomocí Dockeru a GPU computing.

Praktická část zahrnovala návrh a implementaci kompletní infrastruktury s Ollama v Docker kontejneru, Network Load Balancer pro distribuci zátěže a API Gateway s VPC Link pro bezpečný veřejný přístup. Autentizace pomocí API klíčů umožňuje kontrolu přístupu a sledování využití.

Testování prokázalo funkčnost systému s latencí 0,71 s pro krátké prompty, 1,37 s pro střední a 4,46 s pro dlouhé prompty. Throughput dosáhl 7 požadavků za sekundu při 10 souběžných požadavcích s time-to-first-token přibližně 981 ms.

Analýza nákladů ukázala měsíční náklady přibližně 796 USD pro On-Demand instanci, což je ekonomicky výhodné především při vysokém objemu dotazů.

Navržené řešení poskytuje úplnou kontrolu nad daty, eliminaci závislosti na externím poskytovateli, flexibilitu při výběru modelů a předvídatelné náklady. Možnosti dalšího rozvoje zahrnují advanced monitoring, CI/CD integraci a experimentování s různými modely a jejich optimalizací.

Self-hosted přístup představuje možnou alternativu ke komerčním službám, zejména pro organizace s vysokými požadavky na bezpečnost dat nebo s velkým objemem dotazů.

Seznam použité literatury

AWS Blog. Understanding VPC links in Amazon API Gateway private integrations [online]. 2021 [cit. 2025-08-10]. Dostupné z: https://aws.amazon.com/blogs/compute/understanding-vpc-links-in-amazon-api-gateway-private-integrations/

AWS Documentation. The difference between Docker images and containers [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://aws.amazon.com/compare/the-difference-between-docker-images-and-containers/

AWS Documentation. What is AWS? [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://aws.amazon.com/what-is-aws/

AWS EC2 Documentation. Amazon EC2 Instance Types [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://aws.amazon.com/ec2/instance-types/

CODEFINITY. GPU Computing: NVIDIA CUDA Explained [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://codefinity.com/blog/GPU-Computing:-NVIDIA-CUDA-Explained

Editorial Team. OpenAI vs. Anthropic vs. Google Gemini: The enterprise LLM platform guide. *Xenoss* [online]. 2025 [cit. 2025-11-20]. Dostupné z: https://xenoss.io/blog/openai-vs-anthropic-vs-google-gemini-enterprise-llm-platform-guide

ENGETO. Co je LLM - velké jazykové modely [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://engeto.cz/blog/ai/co-je-llm-velke-jazykove-modely/

FOOTE, Keith D. A Brief History of Large Language Models. Dataversity [online]. 2023 [cit. 2025-12-15]. Dostupné z: https://www.dataversity.net/articles/a-brief-history-of-large-language-models/

GeeksforGeeks. Containerization using Docker [online]. 2025 [cit. 2025-08-06]. Dostupné z: https://www.geeksforgeeks.org/blogs/containerization-using-docker/

LiteLLM Documentation. Ollama Provider Documentation [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://docs.litellm.ai/docs/providers/ollama

MONSUR. Ollama vs LM Studio: Which AI Platform is Best for Your Project? Medium [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://medium.com/ml-and-dl/ollama-vs-lm-studio-which-ai-platform-is-best-for-your-project-c58e384fa2e4

No Fluff Jobs. Co je AWS a jaká je práce AWS architekta? [online]. 2025 [cit. 2025-05-10]. Dostupné z: https://nofluffjobs.com/cs/log/pracujte-v-it/co-je-aws-a-jaka-je-prace-aws-architekta/

NVIDIA Documentation. Installing the NVIDIA Container Toolkit [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html

NVIDIA GPU Gems. GPU Computing [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing

Ollama Documentation. Ollama GitHub Repository [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://github.com/ollama/ollama

PANGAN, Kenneth. OpenAI API vs Anthropic API vs Gemini API: A practical guide for businesses in 2025. *Eesel* [online]. 2025 [cit. 2025-11-20]. Dostupné z: https://www.eesel.ai/blog/openai-api-vs-anthropic-api-vs-gemini-api

PLURALSIGHT. AWS vs Azure vs GCP: The Big 3 Cloud Providers Compared [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://www.pluralsight.com/resources/blog/cloud/aws-vs-azure-vs-gcp-the-big-3-cloud-providers-compared/

SKYWORK AI. Ollama Models List 2025: 100+ Models Compared [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://skywork.ai/blog/llm/ollama-models-list-2025-100-models-compared/

SMITH, Aron. ChatGPT vs Gemini vs Claude: How to Choose the Best AI for Your Needs in 2025. *Nutstudio* [online]. 2025 [cit. 2025-11-20]. Dostupné z: https://nutstudio.imyfone.com/llm-tips/chatgpt-vs-gemini-vs-claude/

TRUEFOUNDRY. Transformer Architecture in Large Language Models [online]. 2024 [cit. 2026-03-19]. Dostupné z: https://www.truefoundry.com/blog/transformer-architecture

TYAGI, Raghvendra. AWS Elastic Load Balancer: Overview and Types with Practical. Medium [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://medium.com/@Raghvendra_Tyagi/aws-elastic-load-balancer-overview-and-types-with-practical-c214cbdfdd9b

UBUNLOG. Umělá inteligence Ollama AI v terminálu [online]. 2025 [cit. 2025-12-15]. Dostupné z: https://cs.ubunlog.com/Um%C4%9Bl%C3%A1-inteligence-Ollama-AI-v-termin%C3%A1lu/

Přílohy

1. Zdrojové kódy projektu

GitHub repozitář obsahuje kompletní kód pro implementaci řešení. Najdete v něm Terraform konfiguraci pro automatické nasazení celé AWS infrastruktury a také bash skripty setup-gpu.sh a deploy-ollama.sh, které instalují a konfigurují EC2 instanci.

Pro práci s API je v repozitáři Python klient (client.py) a dva testovací skripty. Test_api.py pro funkční testy a benchmark.py pro měření výkonu. Součástí je také docker-compose.yml pro lokální testování a základní dokumentace.

Dostupné z: https://github.com/stepankonecny96/bakalarka

2. Videoukázka praktické implementace

Video ukazuje běžící řešení v AWS konzoli. Začíná kontrolou EC2 instance, kde je vidět její zdravý stav (healthy) a parametry konfigurace. Následuje zobrazení API Gateway a usage planu

Druhá část videa demonstruje spuštění testovacích skriptů. Nejdřív běží test_api.py pro ověření funkčnosti a pak benchmark.py pro měření výkonu. Video doplňuje výsledky testování popsané v kapitole 5.

Dostupné z: https://www.youtube.com/watch?v=02NNZdpyq_8