



Agent Smith — Multi-Model Government AI Platform

 Agent Smith: Суверенная мульти-модельная AI-платформа для государственного сектора

— это гибкая, безопасная и масштабируемая AI-инфраструктура, предназначенная для использования в органах государственного управления.

Платформа объединяет **on-prem решения** и **облачные API ведущих мировых LLM-провайдеров**, включая:

- OpenAI (GPT-4 Turbo)
- Anthropic (Claude 3 Opus)
- Google (Gemini Pro) -
- Groq API
- Meta ( LLaMA 4, вышла 6 апреля 2025)
- DeepSeek V3, QwenMax, KazLM (open-source модели)

Архитектура

User → LLM Router → [GPT-4 | Claude | Gemini | LLaMA4 | KazLM | Qwen] → Tools → GovChain

LLM Router — маршрутизирует запросы на нужную модель (по задаче, стоимости, политике).

- **Skills/Agents** — микросервисы в Kubernetes (аналитика, документы, чат, CV, Blockchain).
- **GovChain Layer** — DAO-блокчейн, где фиксируются решения с PKI-подписями.


Безопасность - Все данные внутри: MinIO, Postgres, Redis, Milvus.

- Только выбранные запросы уходят в API, остальное — через локальные модели (LLaMA4, KazLM).
- Все результаты валидируются и хэшируются через **GovChain**.

Hugging Face интеграция


- Используем `huggingface_hub` для загрузки LLaMA4, DeepSeek, Yi, Zephyr, Qwen.

- DoLoRA моделей — через MLflow и HF CLI/API.
- Возможность использовать HF Spaces или Enterprise Hub.

 **Агентные сценарии - Chat & Summarization -
Стенограммы совещаний - Юридическая
классификация - Q&A по нормативным актам -
Blockchain ops - CV-анализ фото/видео**

Технологии

- vLLM, MLflow, LangChain, CrewAI
- Hyperledger Besu, OpenZeppelin Governor
- Tyk, Keycloak, Prometheus, n8n, Langfuse

 **Бизнес-эффект - 30+ госорганов, 50+ кейсов -
Агент = 1–10 млн тг годовой экономии**

- Суверенность, масштабируемость, юр. сила

Лицензия и партнёрство

Поддержка: Bitmagic × QOSI × Astana Hub

Лицензия: MIT | CC-BY-4.0 Контакты: bitmagic.kz | qosi.kz`

 **Презентация (текстовая) для инвестора и
МЦРИАП**

Миссия проекта

Agent Smith — это суверенная AI-инфраструктура для Казахстана, объединяющая лучшие модели мира и полный локальный контроль. Мы создаём нейтральный многоагентный оркестратор, который:

- может использовать LLaMA 4, KazLM и Mistral локально,
 - вызывать GPT-4o, Claude 3.7, Gemini 2.5 при необходимости,
 - сохраняет юридическую силу через DAO-блокчейн (GovChain).
-



Почему сейчас?

- **LLaMA 4 только вышла** (6 апреля 2025) и уже превосходит GPT-3.5 и Gemini Pro в reasoning.
 - В Казахстане — более **50 подтверждённых кейсов госИИ**, но **нет инфраструктуры**.
 - Агентно-модельный стек снижает стоимость внедрения на 60–80% по сравнению с bespoke-системами.
-



Что мы строим

- **LLM-orchestrator** с policy-роутингом запросов.
 - Многоагентная система (NLP, CV, Blockchain, Summarization, Finance).
 - Архитектура «единого роевого интеллекта» на базе open-source и API.
 - DAO-реестр решений, подписанных ЭЦП и зафиксированных в блокчейне.
-



Преимущества для МЦРИАП

- Полный on-prem стек (KazLM, LLaMA, Besu).
 - Возможность масштабирования без vendor lock-in.
 - Стандартизация всего цикла: от запроса до зафиксированного решения.
 - Валидация решений на уровне DAO голосования (юридическая сила).
 - Готовность к eGov интеграциям (SSO, ЭЦП, сервисы).
-



Команда и статус

- Архитектура готова.
 - MLOps пайплайн работает (MLflow + vLLM + HF Hub).
 - Поддержка Astana Hub, QOSI, АО НИТ (пилоты).
 - Ожидаем подтверждение MVP от 2–3 ведомств.
-



Дорожная карта

- Q2 2025** — деплой RoAI ядра, агентов NLP + DAO.
- Q3 2025** — подключение LLaMA4, CV-агенты, Hugging Face workflow.
- Q4 2025** — полная DAO-платформа с мульти-модельной инфраструктурой и бизнес-моделями.

Обзор для технически подкованных

Мы строим **многоагентную AI-платформу “под ключ”**, сразу развернутую в private-cloud (Kubernetes). Ниже — что именно «под капотом» и почему так.

Слой	Что там происходит	Зачем это нужно
API-edge	<i>Tyk Gateway</i> — HTTPS-вход, JWT/OAuth, rate-limits.	Один контракт для внешних клиентов (НПКИ, бэкенды, чат-UI).
Оркестратор	<i>CrewAI</i> -координатор (LangChain-friendly). Делит задачу на под-таски, управляет “узкими” агентами.	Чёткая маршрутизация без «монолитных» промптов.
Агенты-сервисы	Python-микросервисы: <i>finance-agent</i> , <i>calendar-agent</i> , <i>rag-agent</i> , <i>cv-agent</i> , <i>blockchain-agent</i> ... общаются по gRPC/REST.	Каждый можно масштабировать/обновлять независимо.
Inference	<i>vLLM</i> на GPU-нодах. Модели тянем из Hugging Face Hub через <i>huggingface_hub</i> SDK. Поддерживаем mix: <i>KazLM-13B</i> , <i>Mistral-7B</i> , <i>Whisper-large</i> .	Локальный быстрый генератор с streaming + возможность легко менять/дообучать модели.
Retrieval	<i>Milvus 2.x</i> (vector DB) + HF <i>sentence-transformers</i> .	RAG: добавляем свежий контекст, минимизируем галлюцинации.
State & cache	<i>PostgreSQL</i> (метаданные, логика), <i>Redis</i> (сессии/кеш), <i>MinIO</i> (S3 для моделей, бэкапов).	Чёткое разделение: быстрый in-memory, транзакционная БД, дешёвое объектное.
MLOps	<i>MLflow</i> + <i>H2O LLM Studio</i> для fine-tuning. Чекпоинты пушим обратно в HF (приватный реп).	Версионирование, A/B, откат моделей одной командой.
Observability	<i>Prometheus</i> + <i>Grafana</i> (кластер, GPU, бизнес-метрики), <i>Langfuse</i> (LLM-tracing), <i>Loki</i> (логи).	Полный срез: от latency токена-стрима до стоимости одного запроса.

Слой	Что там происходит	Зачем это нужно
Automation	<i>n8n</i> self-hosted — 400+ интеграций (G-Suite, CRM, почта).	Быстрые no-code workflow без нового кода агентов.
Web3 слой	MVP: Moralis API → EVM сети. Prod: Hyperledger Besu (PoA) + OpenZeppelin Governor.	Хэшируем решения/ протоколы, DAO-голосования, аудит неизменности.

Поток запроса

1. Клиент шлёт `POST /api/query` → Тук валидирует токен.
2. Coordinator создаёт план: *“извлечь контекст из Milvus, вызвать LLM, сформировать отчёт, при необходимости записать хэш в chain”*.
3. Агенты выполняют параллельно; все вызовы LLM проходят через vLLM (streaming).
4. Langfuse логирует каждый шаг; Prometheus фиксирует latency, GPU usage.
5. Ответ идёт обратно через Тук; если требовалась blockchain-фиксация — Moralis/Besu возвращает tx-hash.

Dev-/Ops-поток

- **CI/CD:** GitHub Actions → build Docker → push → ArgoCD раскатывает.
- **Fine-tune:** MLflow experiment → `transformers.Trainer` → LoRA → `mlflow.register_model` → `hf_hub_upload` → vLLM hot-reload.
- **Scale:** HPA по QPS или GPU utilization; новые агенты — Helm chart + route в Тук.

Почему это “полная стек-пушка”

- **Модульность** — можно выкинуть/добавить агента без даунтайма.
- **Data-gravity внутри:** все базы on-prem, данные не уходят наружу.
- **HF Hub** — моментальный доступ к SOTA-весам и публикация своих моделей без собственного registry.
- **Гибрид Web2/Web3** — один и тот же API пишет как в Postgres, так и в блокчейн, если нужна неизменность.
- **Наблюдаемость с первого дня** — Langfuse + Grafana показывают, где модель тупит или жжёт бюджет.

Итог: это **частное облако с роем доменных LLM-агентов**, которое можно продать как SaaS, поставить on-prem государству или использовать внутри компании, где конфиденциальность и масштабируемость важнее, чем зависимость от внешних OpenAI-endpoint'ов.

пошаговые инструкции на техническом уровне, «для тех, кто всё будет делать руками». Этот мануал ориентирован на системного инженера/DevOps/ML-специалиста, который **разворачивает** инфраструктуру и компоненты в Kubernetes, настраивает взаимодействие с Hugging Face (HF), а также обеспечивает работу всего стека.

1. Подготовка окружения

1. Аппаратное обеспечение

- Минимум 3 узла (виртуалки или физические серверы).
 - **1 GPU**-узел (например, RTX 3090, 24 GB VRAM) для vLLM.
 - **2 CPU**-узла (по 8–16 vCPU, 32+ GB RAM) для баз данных, сервисов, агентов.
- Сетевое окружение: внутренний адрес для K8s, публичный/Ingress для API.
- Объем дисков:
 - для Milvus (векторная БД) и MinIO (S3-хранилище) выделить быстрый SSD (например, 100–500 GB).
 - для PostgreSQL тоже желательно SSD, 50–200 GB.

2. Установка Kubernetes

- Можно **kubeadm** или использовать облачный сервис (AWS EKS, GKE, OpenStack, etc.).
- На **GPU**-ноде установить нужные драйверы NVIDIA, включить `nvidia-device-plugin` DaemonSet.
- Проверить, что `kubectl get nodes` видит все ноды, и GPU видна как `nvidia.com/gpu`.

3. Установка базовых инструментов (локально у себя, не в кластере):

- **Helm** (v3+), **kubectl**, **huggingface-cli** (опционально), **Python** 3.9+ с `pip` (для HF SDK).
- **ArgoCD** (или Flux) — если хотим GitOps-подход. Иначе можно вручную делать `helm install`.

(На этом этапе у вас есть готовый K8s-кластер с минимумом add-ons и GPU-драйвером.)

2. Деплой инфраструктурных сервисов

Все следующие компоненты обычно деплоятся как **Helm** chart'ы (либо yaml-манифесты). Можете найти готовые публичные чарт-репозитории (например, bitnami, official, etc.).

1. MinIO (S3)

- `helm repo add minio https://charts.min.io/`
- `helm install myminio minio/minio --set ...`
- Конфигурируйте PersistentVolume для хранения.
- Запомните MINIO_ACCESS_KEY и MINIO_SECRET_KEY.

2. PostgreSQL

- `helm repo add bitnami https://charts.bitnami.com/bitnami`
- `helm install mypg bitnami/postgresql --set ...`
- Настраиваем Storage (PVC), указать пароль.

3. Redis (если нужна)

- `helm install myredis bitnami/redis --set ...`
- Настраиваем для кэша, сессий, pub/sub.

4. Milvus (векторная БД)

- `helm repo add milvus https://milvus-io.github.io/milvus-helm/`
- `helm install mymilvus milvus/milvus --set ...`
- Указываем тип (standalone или cluster), PVC, CPU/GPU.

5. Tyk Gateway

- `helm repo add tyk https://helm.tyk.io/public/helm/charts/`
- `helm install mytyk tyk/tyk-gateway --set ...`
- Возможно, нужен Tyk dashboard, Redis для хранения ключей.
- Настраиваем Ingress, чтобы Tyk слушал 80/443.

6. Prometheus + Grafana

- Либо `helm install kube-prometheus-stack prometheus-community/kube-prometheus-stack`,
либо использовать `prometheus-operator`, `grafana` отдельно.
- Включить nodeExporter, GPU Exporter, etc.

7. n8n (No-code automation)

- Можно `helm` или `docker-compose` → K8s.
- Указать env: `N8N_BASIC_AUTH_ACTIVE=true` + логин/пароль.

8. MLflow

- Может не иметь официального helm, но есть community charts.
- Запускаем как Deployment + Service.
- Указываем место хранения (MinIO или local PVC).

9. Langfuse

- self-host helm/yaml. [langfuse/langfuse GitHub](https://github.com/langfuse/langfuse).
- Использует Postgres для БД, нужно URL.

(Теперь у нас готова база “скелет” — MinIO, Postgres, Tyk, Milvus, Redis, MLflow, etc. Все сервисы доступные внутри K8s.)

3. Деплой vLLM (GPU инференс)

1. **vLLM** обычно не имеет готового helm. Можно:

- Создать Docker image (Dockerfile) с vLLM и нужными requirements (PyTorch, HF, ctransformers).
- Запустить Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vllm-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vllm
  template:
    metadata:
      labels:
        app: vllm
    spec:
      nodeSelector:
        nvidia.com/gpu.present: "true"
      containers:
        - name: vllm
          image: myregistry/vllm-gpu:latest
          command: ["vllm", "serve"]
          args: ["--model", "/models/llama2-7b", "--port", "8000", "--tensor-parallel-size", "1"]
          resources:
            limits:
              nvidia.com/gpu: 1
          volumeMounts:
            - name: model-volume
              mountPath: /models
      volumes:
        - name: model-volume
          persistentVolumeClaim:
            claimName: modelpvc
```


2. Модели (с Hugging Face) можно загрузить:

- (A) на этапе сборки Docker (RUN `git lfs install && git clone https://huggingface.co/org/model`).
- (B) Или `initContainer/entrypoint: huggingface-cli login + snapshot_download(...)` .
- (C) Или держать в MinIO, смонтировать.

3. Сервис

- Нужно `Service vllm-service type=ClusterIP`, чтобы агенты могли `http://vllm-service:8000` .

4. Настройка Hugging Face SDK

1. Вариант локальный:

- `pip install huggingface_hub`
- ИСПОЛЬЗОВАТЬ `from huggingface_hub import snapshot_download` .
- `snapshot_download(repo_id="meta-llama/Llama-2-7b-chat-hf", revision="main", local_dir="/models/llama2-7b")` .
- На уровне K8s — это можно сделать в `initContainer`:

```
initContainers:
- name: hf-downloader
  image: python:3.9
  command: ["sh", "-c"]
  args:
    - >-
      pip install huggingface_hub &&
      huggingface-cli login --token $HF_TOKEN &&
      python -c "from huggingface_hub import snapshot_download;
snapshot_download(repo_id='meta-llama/Llama-2-7b-chat-hf',
local_dir='/models/llama2-7b')"
  volumeMounts:
    - name: model-volume
      mountPath: /models
  env:
    - name: HF_TOKEN
      valueFrom:
        secretKeyRef:
          name: huggingface-secret
          key: hf_token
```

2. Хранение локально:

- Модель однажды скачана → сохраняется на PVC. При перезапуске не надо повторно тянуть.

3. LoRA/дообучение:

- MLflow pipeline, после обучения → `huggingface-cli upload my-lora --repo`
- В проде agent / initContainer скачивает LoRA и base model, в runtime их сливает.

5. Настройка GitOps (опционально)

Если используем **ArgoCD**:

1. Создаём repo manifest: `agent-finance/Chart.yaml`, `agent-finance/values.yaml`, ...
2. ArgoCD Application, указываем repo URL.
3. При изменении Helm chart, ArgoCD автоматически делает sync.

Или можно вручную `helm install / helm upgrade`.

6. Реализация агентов (пример FinanceAgent)

Создаём Python сервис:

```
finance-agent/  
├── Dockerfile  
├── requirements.txt  
└── app.py
```

app.py (упрощённо)

```
from flask import Flask, request  
import requests  
  
app = Flask(__name__)  
  
@app.route("/finance/query", methods=["POST"])  
def handle_finance():  
    data = request.json  
    # parse question, maybe "что потратил на рекламу?"  
    # logic: get data from Postgres or external bank API
```

```
# or call vLLM with a specific prompt
answer = "В этом месяце 150k тенге на рекламу."
return {"answer": answer}

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

Dockerfile

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

Helm chart (пример)

```
# finance-agent/Chart.yaml
apiVersion: v2
name: finance-agent
version: 0.1.0

# finance-agent/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: finance-agent
spec:
  replicas: 1
  selector:
    matchLabels:
      app: finance-agent
  template:
    metadata:
      labels:
        app: finance-agent
    spec:
      containers:
        - name: finance-agent
          image: myregistry/finance-agent:latest
          ports:
            - containerPort: 8080
```

Сервис

```
kind: Service
apiVersion: v1
metadata:
  name: finance-agent-svc
spec:
  selector:
    app: finance-agent
  ports:
    - port: 80
      targetPort: 8080
      name: http
```

Route via Tyk

- В Tyk Dashboard (или yaml), создаём API definition:
 - `listen_path: /api/finance/` → `upstream http://finance-agent-svc/`
- Тык → `POST /api/finance/query`.

(По такому же шаблону делаем “calendar-agent”, “blockchain-agent” и т. д., меняя логику.)

7. Пример вызова vLLM из агента

Если хотим внутри `finance-agent` вызвать LLM:

```
def call_llm(prompt):
    url = "http://vllm-service:8000/generate"
    payload = {"prompt": prompt, "max_tokens": 512}
    resp = requests.post(url, json=payload)
    return resp.json()["text"]
```

(Формат зависит от vLLM REST API. Может быть `/generate_stream`.)

Таким образом, **агент** может сам управлять промптом, добавлять контекст, etc.

8. Подключение Milvus (RAG agent пример)

1. Установить PyMilvus: `pip install pymilvus`
2. Хранить эмбединги (берём HF embeddings, ex. `from sentence_transformers import SentenceTransformer`).
3. `rag-agent` :

```
# rag_agent.py
from milvus import Milvus, DataType
from sentence_transformers import SentenceTransformer

milvus = Milvus(uri="tcp://mymilvus-milvus.default.svc.cluster.local:19530")
embed_model = SentenceTransformer("sentence-transformers/multi-qa-mpnet-base-dot-v1")

def embed_text(text):
    return embed_model.encode(text).tolist()

def search(query):
    vec = embed_text(query)
    # milvus collection search ...
    # retrieve top-k
```

9. Взаимодействие с Hugging Face (подробней)

1. Секреты:
 - Создать K8s secret `huggingface-secret` с `hf_token`.
 - `kubectl create secret generic huggingface-secret --from-literal=hf_token=<YOUR_TOKEN>`
2. Подтягивание модели (runtime)
 - Либо `initContainer`, либо `bash script`, например:

```
HF_TOKEN=$(cat /secrets/hf_token)
huggingface-cli login --token $HF_TOKEN
huggingface-cli repo clone <model_repo> .
# or snapshot_download(...)
```

3. Обновление модели
 - `MLflow job` → `Python script` → `HF Trainer` → `trainer.train()` → `output` → `huggingface-cli upload new-lora-lm`.
 - Per agent rollout. Possibly use `Helm upgrade` to trigger re-pull of new weights.

10. Тестирование и отладка

1. Проверка Tyk

- `curl -H "Authorization: Bearer <key>" http://tyk-gw/api/finance/query -d '{"question": "Мои траты?"}'`
- Должны получить JSON ответ.

2. Проверка vLLM

- `curl -XPOST "http://vllm-service:8000/generate" -d '{"prompt": "Hello world"}'` .

3. Prometheus

- `kubectl port-forward svc/prometheus 9090:9090` .
- Смотрим GPU usage, CPU usage.

4. Langfuse

- В код агентов вставляем trace (Langfuse Python SDK).
- Смотрим UI, видим цепочки.

5. Milvus

- `pymilvus` in python REPL. Insert embeddings. Search test.

11. Готовность к продакшн

1. Security:

- mTLS внутри кластера, Keycloak for user auth.
- Tyk OIDC / API tokens.
- Secrets (DB, HF token) в K8s SealedSecret/HashiCorp Vault.

2. Scalability:

- HorizontalPodAutoscaler (HPA) на agents/vLLM.
- Dist. vLLM or multiple GPU pods if big concurrency.

3. Backups

- Postgres, MinIO, Milvus store — либо Velero (k8s backups)
- Scripts with `pg_dump` , `mc mirror` (for minio).

4. Logging:

- EFK (Elastic / Fluentd / Kibana) or Loki + Grafana.

5. Disaster recovery

- Setup multi-zone cluster if possible.

12. Итог (тезисно)

- **Шаги:**
 1. Поднять K8s (3 ноды),
 2. Установить MinIO, Postgres, Redis, Milvus, Tyk, MLflow, Langfuse, n8n,
 3. vLLM (GPU) Deployment,
 4. HF SDK/CLI для моделей,
 5. Писать Python агенты (с Pydantic/FastAPI/Flask),
 6. Helm для каждого, подключить Tyk routes,
 7. Тест, логирование, Prom/Grafana.
- **Это «суповой набор»:** много компонентов, но все open-source, гибко собираются.
- **Hugging Face** — ключевой канал для получения/публикации LLM/эмбеддингов.
- **Продукт:** разом агентный сервис, MLOps pipeline, blockchain-friendly, 100% on-prem.

Таким образом, «для тех, кто руками всё будет ставить», этот мануал даёт общее представление: **что, как и в каком порядке** монтируем в кластере, где какие Helm-чарты брать, как прописываем initContainers, как вызываем HF SDK, как строим Python-агентов и подключаем их к Tyk. Далее уже каждый шаг можно детализировать по своей спецификации (конкретные values.yaml, volumes, securityContext, etc.), но основные направления именно так.