2026

Итоговый отчет

РАЗРАБОТКА И РАЗВЕРТЫВАНИЕ ИИ-ПРИЛОЖЕНИЙ В KUBERNETES

АГАЕВ РУСЛАН ЭЛЬМАРОВИЧ

Студент группы М24-535 Агаев Р.Э.

Оглавление

Студент группы М24-535 Агаев Р.Э.

ВВЕДЕНИЕ

Цель проекта

Целью проекта является разработка, контейнеризация и развёртывание микросервисного Java-приложения с элементами интеллектуального анализа текста в среде Kubernetes (Minikube).
В рамках работы демонстрируется практическое применение современных технологий контейнерной инфраструктуры и мониторинга, а также рассматриваются актуальные тенденции использования ИИ в распределённых системах.

В процессе реализации проекта были затронуты следующие направления:

- контейнеризация приложений;
- оркестрация контейнеров;
- мониторинг и визуализация метрик;
- автоматическое горизонтальное масштабирование;
- анализ современных подходов к интеграции ИИ в облачные и контейнерные среды.

Основные задачи проекта

Для достижения поставленной цели в рамках работы были решены следующие задачи:

1. Установка и базовая настройка Minikube для локального развёртывания Kubernetes-кластера.
2. Разработка REST API приложения на языке Java, реализующего базовую функцию анализа тональности текстовых данных.
3. Контейнеризация приложения с использованием Docker.
4. Развёртывание приложения в Kubernetes с применением YAML-манифестов.
5. Настройка мониторинга состояния приложения и кластера с использованием Prometheus и Grafana.
6. Реализация автоматического масштабирования подов с помощью Horizontal Pod Autoscaler.
7. Подготовка и оформление документации по проекту.

Используемый стек технологий

В ходе выполнения проекта применялся следующий технологический стек:

Компонент	Технология
Язык программирования	Java (современная версия JDK)
HTTP-сервер	Встроенный HTTP-сервер JDK
Сбор метрик	Prometheus Java Client

Студент группы М24-535 Агаев Р.Э.

Компонент	Технология
Контейнеризация	Docker
Оркестрация	Kubernetes (Minikube)
Мониторинг и визуализация	Prometheus и Grafana
Балансировка нагрузки	Kubernetes Service
Масштабирование	Horizontal Pod Autoscaler

АРХИТЕКТУРА РЕШЕНИЯ

Общая схема архитектуры

Развёрнутое решение представляет собой Kubernetes-кластер, внутри которого функционируют следующие компоненты:

- Ingress, выполняющий маршрутизацию HTTP-запросов к приложению;
- Service, обеспечивающий доступ к приложению внутри кластера;
- Deployment с несколькими репликами Java-приложения;
- Horizontal Pod Autoscaler, отвечающий за автоматическое масштабирование подов;
- Prometheus, собирающий метрики приложения и ресурсов;
- Grafana, предоставляющая интерфейс визуализации и анализа метрик.

Описание компонентов архитектуры

Java-приложение

Приложение реализовано в виде REST-сервиса с использованием встроенного HTTP-сервера JDK. Оно предоставляет эндпоинт для анализа тональности текста и отдельный эндпоинт для экспорта метрик в формате Prometheus.

Docker-образ

Для сборки приложения используется многоэтапный Dockerfile, включающий этап сборки и этап формирования облегчённого runtime-образа. Такой подход позволяет сократить итоговый размер контейнера и повысить эффективность развёртывания.

Kubernetes Deployment

Приложение развёрнуто в виде Deployment с несколькими репликами. Для подов заданы ограничения по использованию ресурсов, а также настроены проверки готовности и работоспособности.

Service и Ingress

Service используется для предоставления доступа к приложению внутри кластера, а Ingress — для маршрутизации внешних HTTP-запросов к соответствующим эндпоинтам.

Студент группы М24-535 Агаев Р.Э.

Horizontal Pod Autoscaler

HPA автоматически изменяет количество реплик приложения в зависимости от нагрузки, основываясь на метриках использования ресурсов.

Prometheus и Grafana

Prometheus осуществляет регулярный сбор метрик приложения и кластера, а Grafana используется для отображения состояния системы, анализа производительности и наблюдения за поведением приложения.

Ключевые особенности Java-приложения

- Реализация REST API без использования полноценных фреймворков.
- Упрощённая логика анализа тональности текста на основе ключевых слов.
- Экспорт метрик для мониторинга количества запросов и состояния JVM.
- Минимальная архитектура приложения, реализованная в одном основном классе.

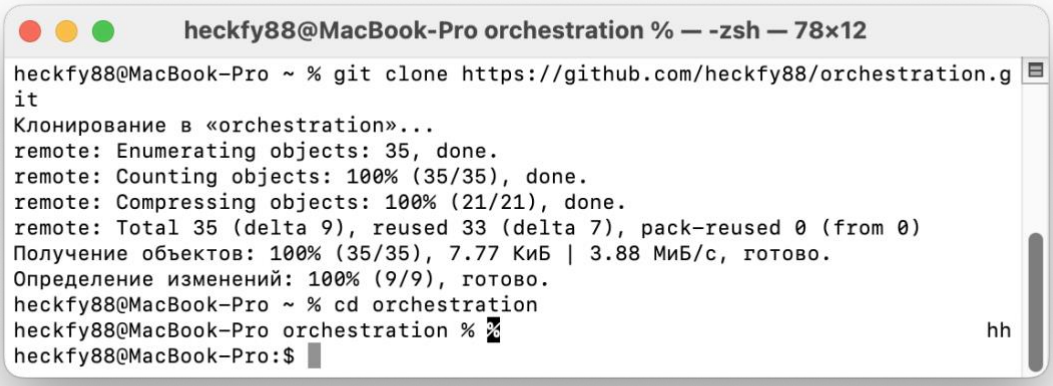
Dockerfile

В проекте применяется многоэтапная сборка Docker-образа:

- на этапе сборки используется инструмент автоматической сборки Java-приложений;
- на этапе выполнения формируется облегчённый runtime-образ;
- в качестве базового образа используется минималистичное окружение, что позволяет снизить размер итогового контейнера.

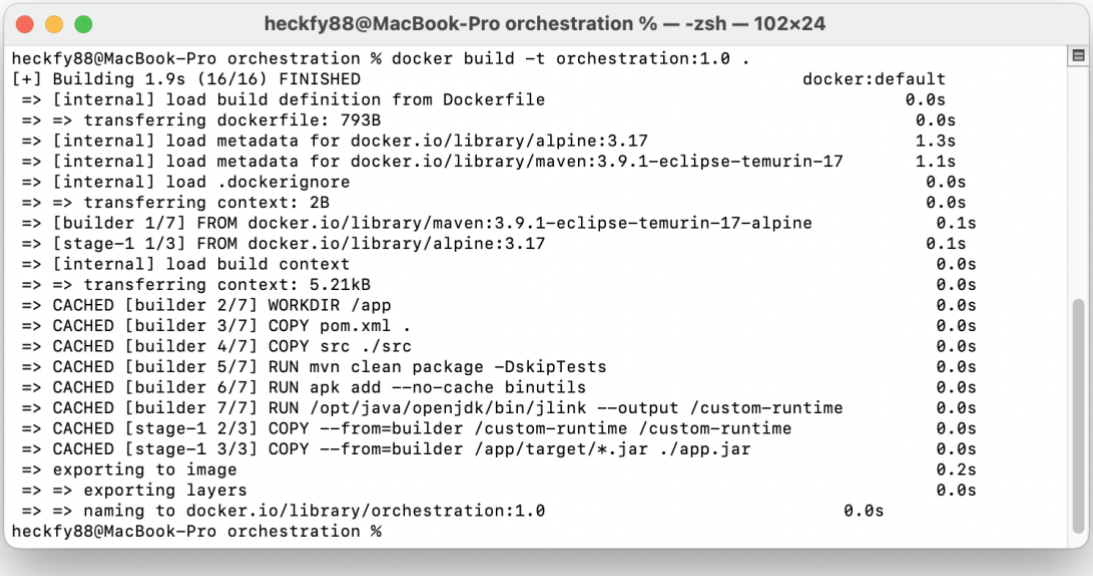
Развертывание

1. Клонируем репозиторий с приложением
   https://github.com/heckfy88/orchestration

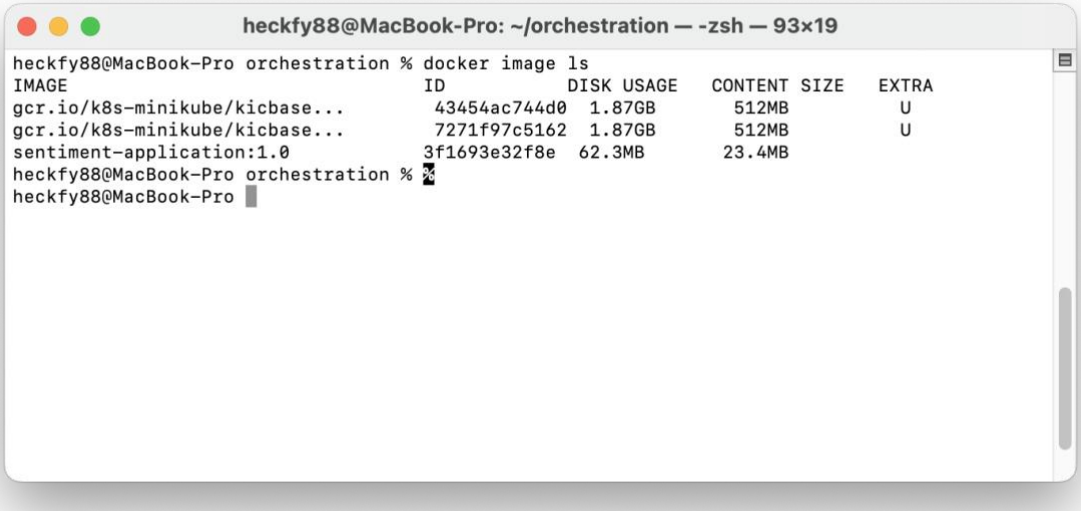


2. Собираем докер образ

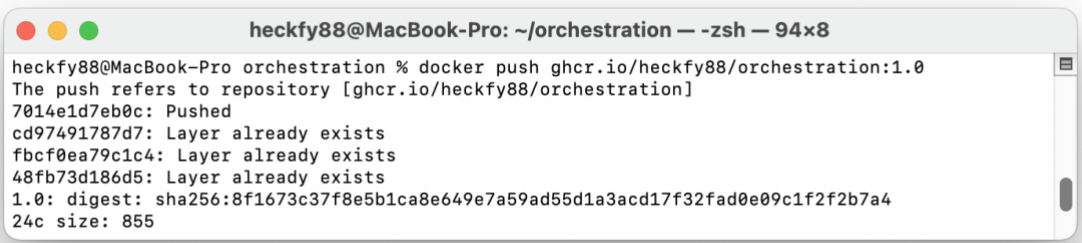
Студент группы М24-535 Агаев Р.Э.



3. Проверим images

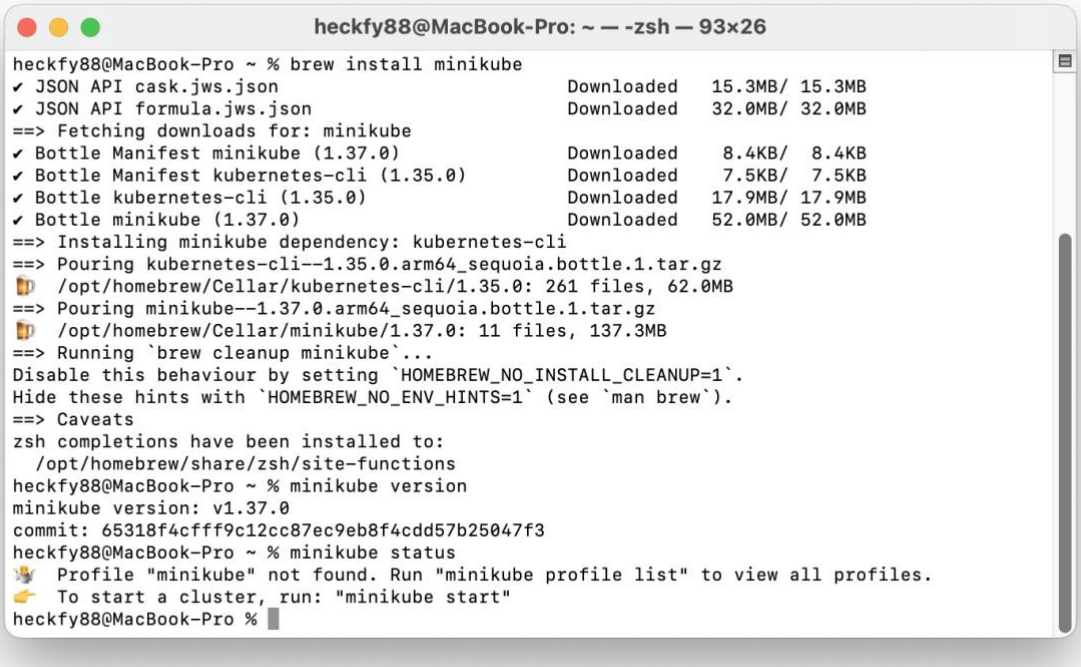


4. Добавление тега и пуш в репозиторий



Студент группы М24-535 Агаев Р.Э.

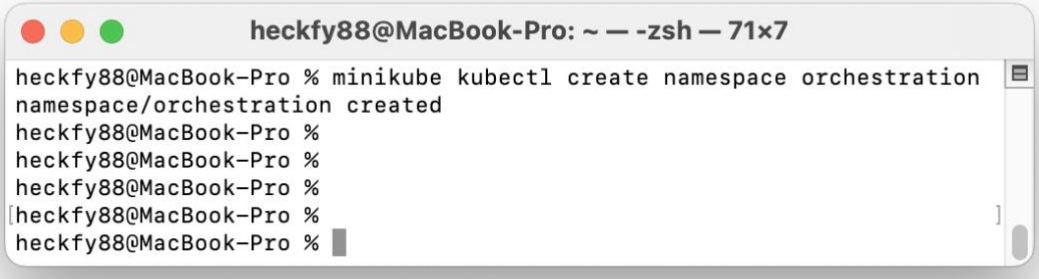
5. Установка minikube



Запуск Minikube

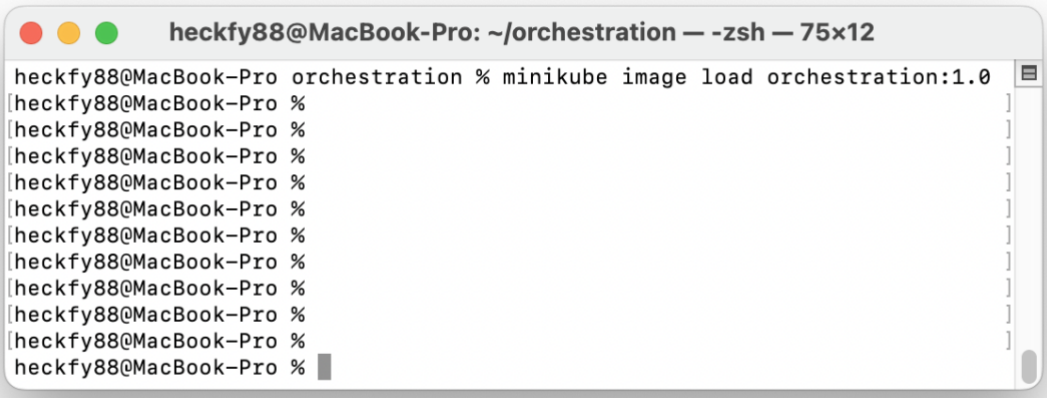
Студент группы М24-535 Агаев Р.Э.

6. Создание namespace



7. Копирование образа

Студент группы М24-535 Агаев Р.Э.



8. Применение манифестов
    a. k8s/deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orchestration
  namespace: orchestration
  labels:
    app: orchestration
spec:
  replicas: 3
  selector:
    matchLabels:
      app: orchestration
  template:
    metadata:
      labels:
        app: orchestration
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "8081"
        prometheus.io/path: "/"
    spec:
      containers:
      - name: orchestration-app
        image: orchestration:1.0
        ports:
          - containerPort: 8080
```

Студент группы M24-535 Агаев Р.Э.

```
        name: http
      - containerPort: 8081
        name: metrics
    resources:
      requests:
        memory: "256Mi"
        cpu: "100m"
      limits:
        memory: "512Mi"
        cpu: "500m"
```

```
heckfy88@MacBook-Pro orchestration % minikube kubectl -- apply -f k8s/deplo
yment.yaml -n orchestration-app
deployment.apps/orchestration-app created
```

РЕЗУЛЬТАТ APPLY

b. k8s/service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: orchestration-service
  namespace: orchestration
  labels:
    app: orchestration
spec:
  type: LoadBalancer
  selector:
    app: orchestration
  ports:
    - name: http
      protocol: TCP
      port: 8080
      targetPort: 8080
    - name: metrics
      protocol: TCP
      port: 8081
      targetPort: 8081
```

```
heckfy88@MacBook-Pro orchestration % minikube kubectl -- apply -f k8s/deplo
yment.yaml -n orchestration-app
deployment.apps/orchestration-app created
```

РЕЗУЛЬТАТ APPLY

c. k8s/ingress.yaml

```
apiVersion: networking.k8s.io/v1
```

```yaml
kind: Ingress
metadata:
  name: orchestration-ingress
  namespace: orchestration
spec:
  ingressClassName: nginx
  rules:
    - host: orchestration.local
      http:
        paths:
          - path: /api/v1
            pathType: Prefix
            backend:
              service:
                name: orchestration-service
                port:
                  number: 8080
```

```
heckfy88@MacBook-Pro orchestration % minikube kubectl -- apply -f k8s/ingre
ss.yaml -n orchestration
ingress.networking.k8s.io/orchestration-ingress created
```

Результат apply

d. k8s/hpa.yaml

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: orchestration-hpa
  namespace: orchestration
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: orchestration
  minReplicas: 3
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
```

Студент группы М24-535 Агаев Р.Э.

```
- type: Resource
  resource:
    name: memory
    target:
      type: Utilization
      averageUtilization: 80
```

```
heckfy88@MacBook-Pro orchestration % minikube kubectl -- apply -f k8s/hpa.y
aml -n orchestration
horizontalpodautoscaler.autoscaling/orchestration-hpa created
```

РЕЗУЛЬТАТ APPLY

e. Проверка

```
heckfy88@MacBook-Pro: ~/orchestration — -zsh — 85×24
heckfy88@MacBook-Pro orchestration % minikube kubectl -- get all -n orchestration

NAME                                    READY   STATUS    RESTARTS   AGE
pod/orchestration-57bbf567f9-bvjkw      1/1     Running   0          8m59s
pod/orchestration-57bbf567f9-ms48q      1/1     Running   0          8m59s
pod/orchestration-57bbf567f9-xhqqd      1/1     Running   0          8m59s

NAME                            TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
      AGE
service/orchestration-service LoadBalancer   10.106.193.61   <pending>     8080:30557
/TCP   6m5s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/orchestration       3/3     3            3           8m59s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/orchestration-57bbf567f9       3         3         3       8m59s

NAME                                                MINPODS   MAXPODS   REPLICAS   AGE
   REFERENCE                      TARGETS
horizontalpodautoscaler.autoscaling/orchestration-hpa
                                                    3         10        3          4m13
s Deployment/orchestration     cpu: <unknown>/70%, memory: <unknown>/80%
heckfy88@MacBook-Pro %
```

```
heckfy88@MacBook-Pro orchestration % minikube kubectl -- get hpa -n orchestration

NAME                REFERENCE                       TARGETS                        MINPOD
S   MAXPODS   REPLICAS   AGE
orchestration-hpa   Deployment/orchestration      cpu: 2%/70%, memory: 10%/80% 3
     10        3          151m
```

f. Проверка приложения

```
heckfy88@MacBook-Pro orchestration %minikube service orchestration-service -n orchestration
```

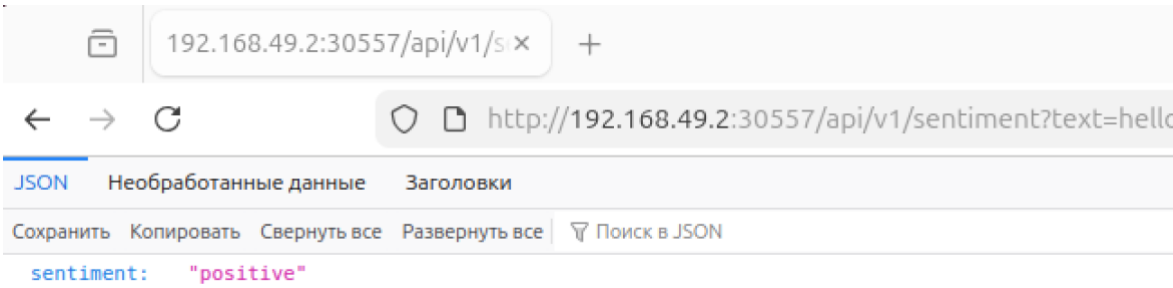
NAMESPACE	NAME	TARGET PORT	URL
orchestration	orchestration-service	http/8080 metrics/8081	http://192.168.49.2:30557 http://192.168.49.2:32027

```
heckfy88@MacBook-Pro %
```

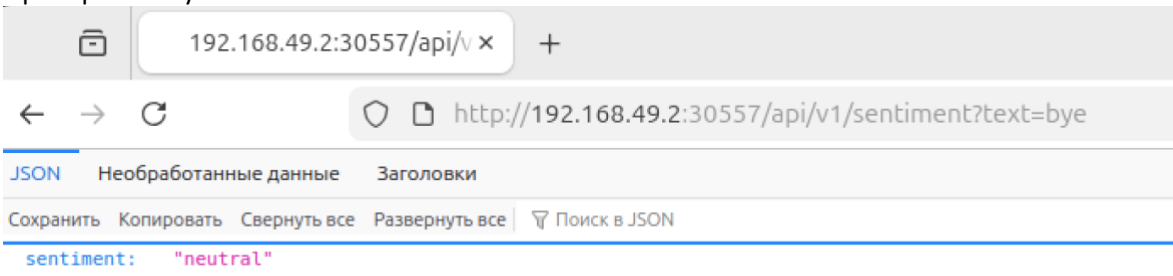
g. Проверка на hello

Студент группы М24-535 Агаев Р.Э.

h.

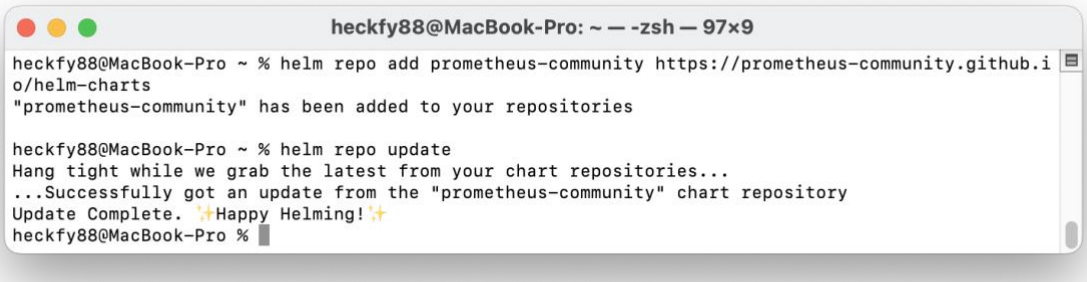


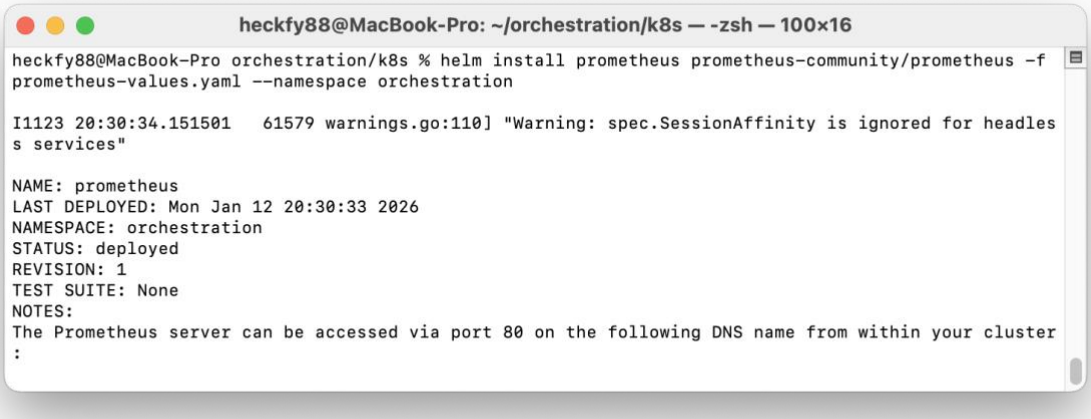
i. Проеверка на bye



Мониторинг

Установка Prometheus





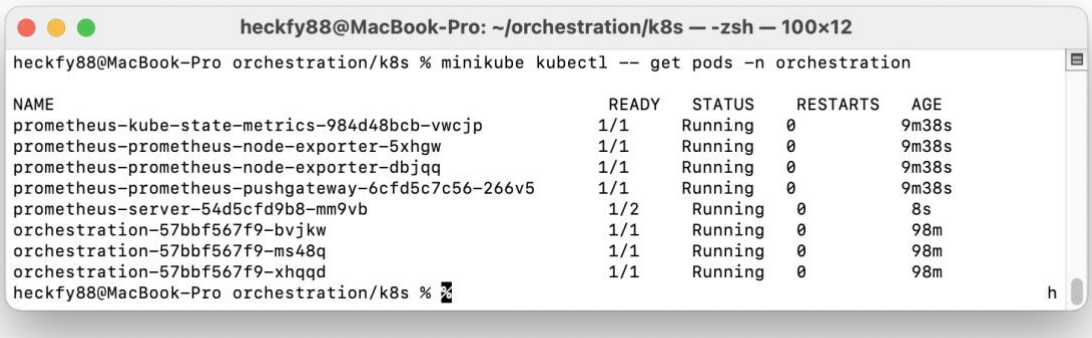
Содержимое Prometheus-values.yaml

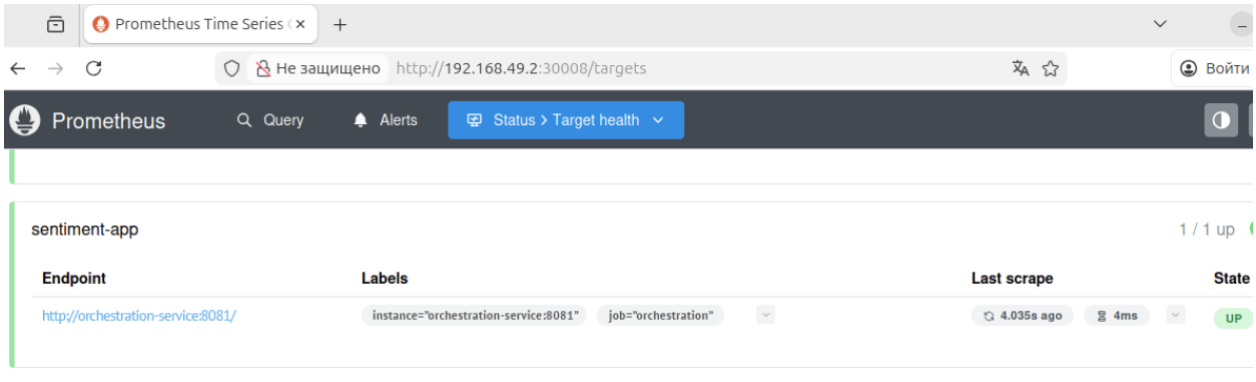
```
alertmanager:
```

Студент группы М24-535 Агаев Р.Э.

```
  enabled: false

server:
  persistentVolume:
    enabled: true
    size: 5Gi

service:
  type: NodePort
  nodePort: 30008

extraScrapeConfigs: |
  - job_name: 'orchestration'
    scrape_interval: 5s
    metrics_path: '/'
    static_configs:
      - targets: ['orchestration-service:8081']
```

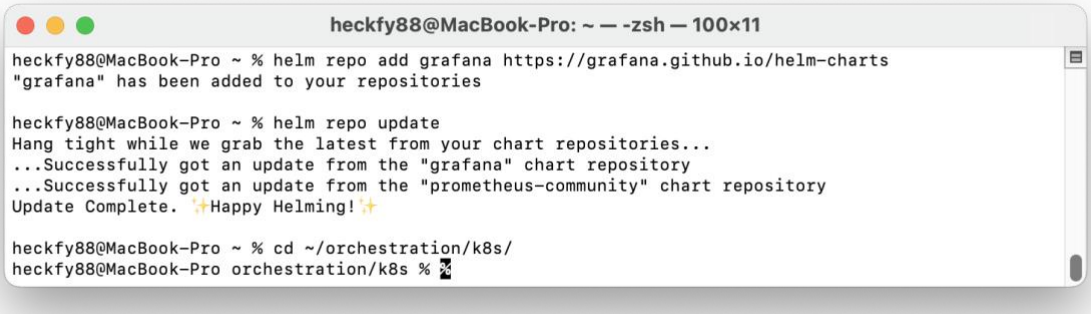
Проверка работы Prometheus

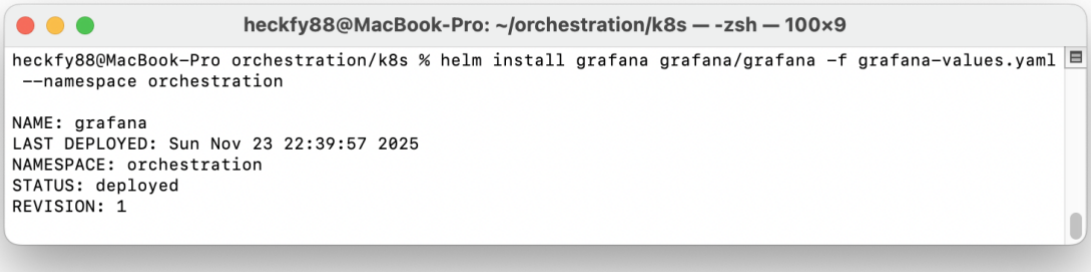




Установка Grafana

Студент группы М24-535 Агаев Р.Э.



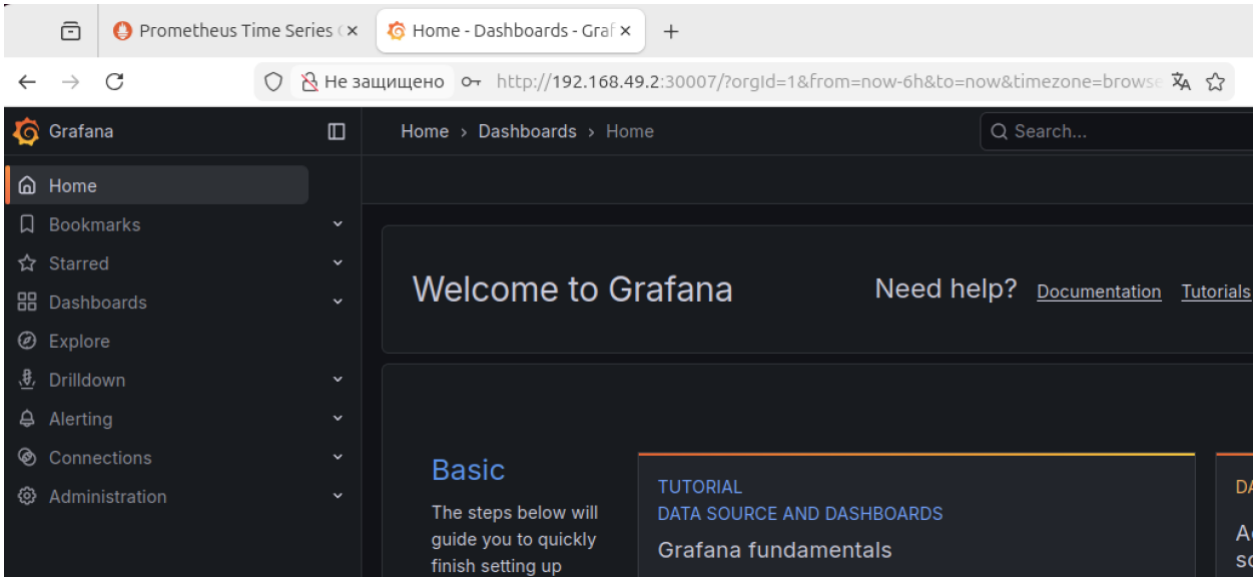


Grafana-values.yaml

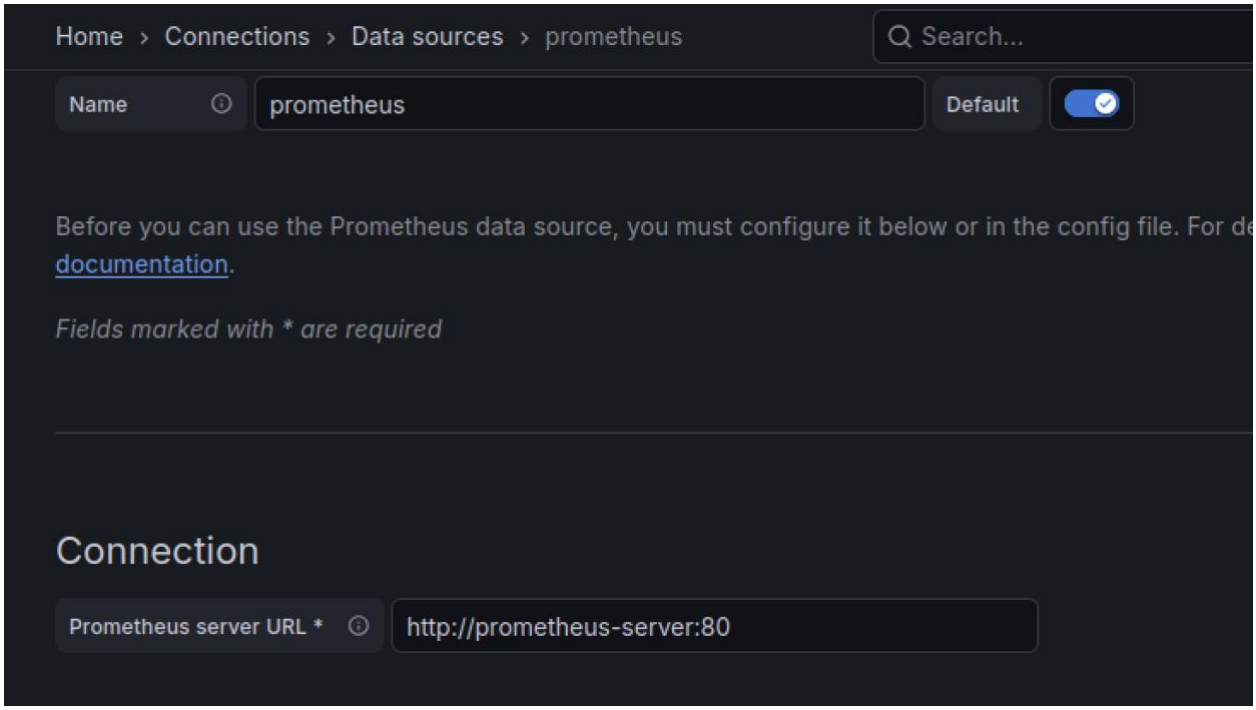
```yaml
persistence:
  enabled: true
  size: 5Gi

adminPassword: "Password1234!"

service:
  type: NodePort
  port: 80
  targetPort: 3000
  nodePort: 30007

sidecar:
  dashboards:
    enabled: true
    label: grafana_dashboard
  datasources:
    enabled: true
    label: grafana_datasource
```

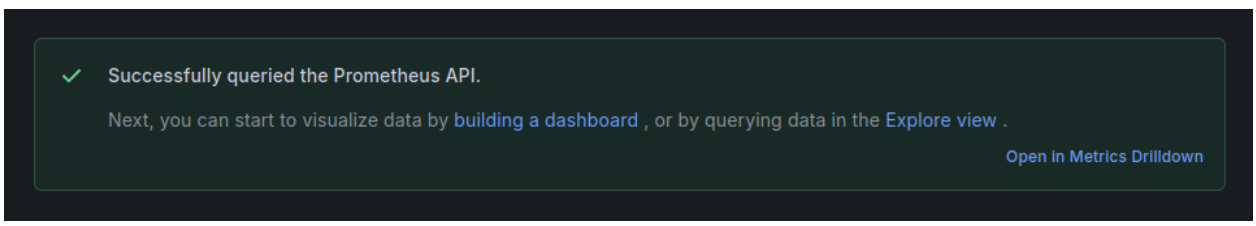
Студент группы М24-535 Агаев Р.Э.

Проверка развертывания:



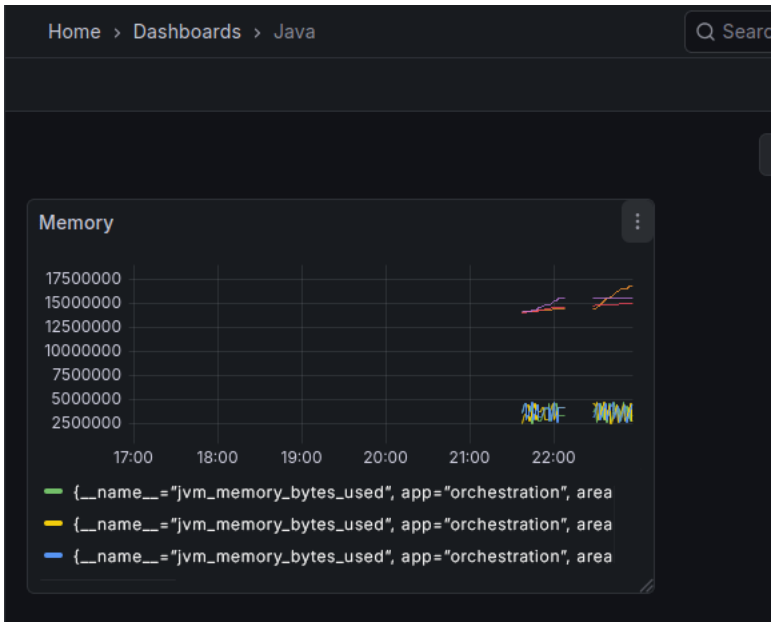
Добавление datasource





Дашборд с информацией об используемой памяти

Студент группы М24-535 Агаев Р.Э.



Анализ трендов

В аналитической части рассмотрены современные исследования, посвящённые применению контейнерных технологий в задачах искусственного интеллекта. Основное внимание уделяется тем направлениям, которые напрямую влияют на архитектуру, производительность и масштабируемость ИИ-приложений в Kubernetes-среде.

Анализ сфокусирован на трёх взаимосвязанных областях: использовании контейнеризации в мультиоблачных инфраструктурах, оптимизации инференса больших языковых моделей за счёт инфраструктурных решений (в частности, механизмов кеширования), а также развитии многоагентных архитектур и рабочих процессов на базе ИИ. Выбранные направления отражают текущие тенденции перехода от изолированных моделей к распределённым, управляемым и масштабируемым системам.

В рамках данного раздела приведён обзор выбранных научных публикаций, выполнен анализ ключевых идей и подходов, а также сформулированы выводы о том, какие из рассмотренных решений могут быть применимы при разработке контейнеризированных ИИ-приложений и в учебных проектах на базе Kubernetes.

№	Название (оригинал / перевод)	Авторы, год	Область	Краткое описание	Ссылка (arXiv)
1	**Containerization in Multi-Cloud Environment: Roles, Strategies, Challenges, and**	Muhammad Waseem, Aakash Ahmad **и др.**, 2024	Контейнеризация, Multi-Cloud	Систематический обзор подходов к контейнеризации приложений в мультиоблачных средах	arXiv:2403.12980

Студент группы М24-535 Агаев Р.Э.

№	Название (оригинал / перевод)	Авторы, год	Область	Краткое описание	Ссылка (arXiv)
	Solutions for Effective Implementation *(Контейнеризация в мультиоблачной среде: роли, стратегии, вызовы и решения)*			(Kubernetes, Docker) с классификацией основных тем, паттернов, а также проблем безопасности, мониторинга и автоматизации развёртывания.	
2	**An Efficient KV Cache Layer for Enterprise-Scale LLM Inference** *(Эффективный слой кеширования KV для масштабного инференса LLM)*	*Anonymous* (LMCache team), 2025	ИИ, LLM, Kubernetes	Представлен **LMCache** — высокопроизводительный слой кеширования для больших языковых моделей, предназначенный для Kubernetes-кластеров. Обеспечивает параллельную обработку LLM-запросов и динамическое масштабирование, что приводит к увеличению throughput до 15× и снижению задержек при инференсе. Решение интегрируется с фреймворками LLM (например, vLLM) и оптимизирует использование GPU (CUDA) для эффективного распределения вычислений.	arXiv:2510.09665
3	**Evolution of AI in Education: Agentic Workflows** *(Эволюция ИИ в образовании:*	Firuz Kamalov, David S. Calonge **и** др., 2024	ИИ, EdTech, Workflow	Обзор развития многокомпонентных (агентных) ИИ-систем в образовании. Раскрыты архитектуры с несколькими взаимодействующими	arXiv:2504.20082

№	Название (оригинал / перевод)	Авторы, год	Область	Краткое описание	Ссылка (arXiv)
	агентные рабочие циклы)			агентами, включающие парадигмы рефлексии, планирования, использования инструментов и кооперации агентов. Показано, что мультиагентные команды превосходят одиночные LLM по надёжности и адаптивности в задачах (например, автоматизированная оценка эссе).	

Таблица 1: Краткая характеристика выбранных работ с arXiv (2024–2025).

Как видно из таблицы, исследования охватывают широкий спектр тем – от оркестрации контейнеров в облаках до ускорения работы моделей ИИ и новых подходов к построению агентных систем. Далее рассмотрим ключевые идеи каждой работы более подробно, а также их практическую значимость и общие тенденции развития технологий контейнеризации и ИИ.

1. Containerization in Multi-Cloud Environment (контейнеризация в мультиоблачной среде)

Обзор: Статья представляет глубокий **систематический обзор** по контейнеризации приложений в мультиоблачных средах. Ключевые технологии, рассматриваемые авторами, – платформы оркестрации контейнеров **Kubernetes** и контейнеры **Docker**, которые обеспечивают переносимость приложений, изоляцию и эффективное управление ресурсами в гибридной инфраструктуре облаков. Авторы систематизируют результаты 121 исследования (2013–2024) и выделяют несколько важных направлений. В частности, определены четыре ведущие темы: масштабируемость и высокая доступность, производительность и оптимизация, безопасность и конфиденциальность, а также мониторинг и адаптация в мультиоблачных кластерах. Кроме того, классифицированы десятки практических паттернов и стратегий контейнеризации, выявлены ключевые **качества систем** (например, надежность, устойчивость) с соответствующими тактиками достижения, и каталогизированы основные **вызовы** и решения в областях безопасности, автоматизации развёртывания, сетевой архитектуры, хранения данных и мониторинга.

Особое внимание в обзоре уделяется **безопасности контейнеров** (например, управлению правами доступа, изоляции workloads) и вопросам стандартизации конфигураций для упрощения автоматизации. Таким образом, работа предоставляет целостную картину текущего состояния и лучших практик контейнеризации в условиях нескольких облачных провайдеров. Это особенно

ценно для ИИ-приложений, поскольку такие нагрузки требуют высокой устойчивости, эффективного распределения ресурсов и строгих политик безопасности при обработке конфиденциальных данных. Выполненная классификация паттернов и рекомендаций помогает выстраивать Kubernetes-инфраструктуру с продвинутыми средствами управления жизненным циклом контейнеров, автоматическим масштабированием и мониторингом. Следование этим подходам позволяет оптимизировать запуск и поддержку моделей ИИ (например, сервисов анализа данных или NLP-моделей) одновременно в разных облаках.

Применение на практике:

- Использовать рекомендации обзора для построения контейнерной инфраструктуры с акцентом на минимальные требования к ресурсам и эффективную оркестрацию (актуально для локального Kubernetes-кластера, например Minikube).
- Внедрить лучшие практики безопасности и управления конфигурациями (Kubernetes-манифестами), чтобы обеспечить стабильное и безопасное развёртывание Docker-образа нашего Java-приложения.
- Следовать паттернам масштабируемости и мониторинга: это позволит в будущем безболезненно расширить приложение на мультиоблачные среды или более крупные кластеры при росте нагрузки.

2. An Efficient KV Cache Layer for Enterprise-Scale LLM Inference (эффективный KV-кеш для инференса LLM)

Обзор: Данная работа посвящена оптимизации производительности больших языковых моделей на этапе *инференса* (использования обученной модели). Авторы предлагают систему **LMCache** – высокопроизводительный слой кеширования результатов для LLM, рассчитанный на корпоративные масштабируемые сервисы. Архитектура LMCache разрабатывалась для развёртывания в Kubernetes-кластере и поддерживает **параллельную обработку запросов** к модели, а также динамическое масштабирование под нагрузкой. Ключевая идея заключается в повторном использовании промежуточных результатов инференса (Key-Value кеша слоёв трансформера) между разными запросами и даже разными экземплярами моделей. В традиционных системах каждый запрос обрабатывается моделью независимо, и после получения ответа вычисленные ключи/значения сбрасываются. LMCache же извлекает **KV-кеш** из памяти GPU и сохраняет его во внешнем хранилище, доступном для повторного использования другими запросами и процессами модели. Такой подход позволяет избежать повторных вычислений при схожих входных данных (например, когда разные запросы содержат одинаковый контекст или при многоходовых диалогах), разгружая GPU и экономя до **50% вычислительных ресурсов**.

Важно, что LMCache является первым открытым решением подобного рода и предлагает ряд инженерных улучшений: оптимизированные операции передачи данных (батчинг, конвейеризация вычислений и I/O), модульный коннектор для интеграции с различными LLM-движками (поддерживаются vLLM, SGLang и др.), а также API для гибкого управления кешем на разных уровнях (GPU, CPU, дисковое и сетевое хранилище). **Результаты экспериментов** впечатляют: совместное использование LMCache с движком vLLM показало до **15-кратного роста пропускной способности** системы на ряде нагрузок (многораундовые вопрос-ответ сессии, анализ документов) при одновременном снижении времени отклика. Благодаря эффективному переиспользованию ранее вычисленного контекста, снижается как *time-to-first-token* (задержка выдачи первых слов ответа), так и общая нагрузка на память GPU. Система масштабируется на

несколько GPU и узлов, позволяя обслуживать большое число одновременных запросов без деградации производительности.

Практическая ценность: Для разработчиков, занимающихся развёртыванием ИИ-моделей в контейнерах, данная работа предоставляет готовые идеи повышения эффективности. Применение кеш-слоя значительно уменьшает избыточные вычисления и тем самым **снижает инфраструктурные затраты** (меньше потребность в мощных GPU при росте числа запросов) и улучшает пользовательский опыт за счёт более быстрого ответа сервиса. Интеграция LMCache с популярными фреймворками (например, vLLM) демонстрирует, как подобные оптимизации могут быть вписаны в существующие pipeline машинного обучения без кардинальной переработки моделей. Алгоритмы и архитектурные решения из LMCache (такие как разделение фаз префиллинга и декодирования между разными узлами, сжатие и перенос кеша между GPU и CPU и т.п.) могут быть взяты на вооружение при построении масштабируемых NLP-сервисов в облаке.

Применение на практике:

- Реализовать простой механизм кеширования для результатов анализа тональности (пока что на уровне мок-данных), чтобы избежать повторного обращения к модели при однотипных запросах и снизить нагрузку.
- В перспективе использовать идею выделенного кеш-слоя как основу при интеграции более сложных моделей ИИ: когда появится необходимость в настоящем инференсе и масштабировании, кеширование поможет повысить производительность.
- При увеличении нагрузки и развертывании приложения в Kubernetes-кластере добавить **адаптивное кеширование** ответов модели. Это повысит пропускную способность API и сократит время отклика сервиса под высоким concurrent-нагрузками.

3. Evolution of AI in Education: Agentic Workflows (эволюция ИИ в образовании – агентные workflows)

Обзор: Искусственный интеллект всё активнее применяется в образовании – от автоматического репетиторства до оценки работ студентов. Однако **традиционные LLM** (большие языковые модели) в этой области сталкиваются с рядом ограничений: зависимость от статичных данных обучения, недостаточная адаптивность к новым ситуациям и отсутствие явного механизма рассуждения. В обзоре Kamalov et al. предлагается подход *агентных рабочих циклов*, где вместо единственного монолитного ИИ действует группа специализированных **агентов**, взаимодействующих между собой для достижения цели. Выделены четыре ключевых парадигмы, описывающие роль агентного интеллекта: **рефлексия**, **планирование**, **использование инструментов** и **многоагентное сотрудничество**. Через призму этих парадигм анализируется, как интеллектуальные агенты могут улучшить образовательные приложения, какие преимущества они дают (например, лучшая адаптивность, способность к сложному планированию или доступ к внешним инструментам), и с какими вызовами сопряжено их внедрение (прозрачность решений, доверие пользователей, этические аспекты и др.).

Авторы иллюстрируют потенциал подхода на примере прототипа мультиагентной системы для автоматизированной оценки эссе. В этой системе несколько агентов с разными ролями координируются для проверки работы студента: один агент оценивает содержание эссе (идею, аргументацию, соответствие теме), другой – языковое качество текста (грамматику, стиль), а *агент-надзорный* (supervisor) объединяет результаты и выносит итоговую оценку.

Студент группы М24-535 Агаев Р.Э.

Предварительные эксперименты показали, что такая командная мультиагентная схема обеспечивает более **последовательные и надёжные оценки** по сравнению с подходом, где аналогичную задачу решает одиночная модель (например, один LLM). Разделение задачи между специализированными агентами и их совместная работа позволяют учесть разные аспекты качества и снизить влияние случайных ошибок одного источника. Иными словами, мультиагентные системы оказались более устойчивыми и адаптивными в сложных заданиях, требующих нескольких этапов анализа, чем монолитные ИИ.

Пример архитектуры мультиагентной системы: в прототипе автоматической оценки эссе (MASS) задействованы три агента: два подагента анализируют *содержание* работы и *язык изложения*, после чего **агент-координатор** (Supervisor) запрашивает у них дополнительную информацию при необходимости и формирует финальную оценку. Такая схема демонстрирует преимущества разделения задач между специализированными агентами и координации результатов через надзорный компонент. Например, мультиагентная система MASS обеспечила более низкую среднюю ошибку оценки (MAE ≈0.561 против 0.613 у одиночной модели GPT-4) и более стабильные результаты на разных эссе. Это подчёркивает, как узкая специализация агентов в сочетании с их взаимодействием повышает точность и **консистентность** ИИ-систем в образовательных задачах.

Таким образом, выводы данного обзора ценны не только для EdTech, но и шире – для архитектуры сложных ИИ-сервисов. По сути, многоагентная система можно рассматривать как набор взаимодействующих микросервисов (контейнеров), где каждый отвечает за свою функцию, а вместе они решают комплексную задачу. Такой модульный подход созвучен принципам контейнеризации: он **облегчает масштабирование и сопровождение** сложных ИИ-приложений, позволяя добавлять новые компоненты (агенты) или обновлять их независимо. В контексте нашего проекта это означает, что архитектуру приложения стоит изначально спроектировать с возможностью расширения до мультиагентной – например, добавить отдельные сервисы для анализа разных аспектов текста (тональность, эмоциональная окраска, стиль) и компонент-оркестратор для агрегирования результатов.

Применение на практике:

- Организовать архитектуру приложения таким образом, чтобы его можно было со временем **расширить до мультиагентной системы**. Например, предусмотреть возможность добавления новых модулей/сервисов для анализа разных характеристик текста (семантический анализ, эмоциональный тон, проверка стиля и др.), которые будут работать параллельно.
- Закладывать модульность через чёткие API (например, RESTful-сервисы) для взаимодействия между компонентами. Это упростит развитие и сопровождение системы при переходе от простого мок-анализа к полноценным ИИ-модулям, позволяя тестировать и обновлять каждый компонент независимо.
- Использовать идеи агентных workflow для автоматизации процессов: например, для автотестирования качества анализа тональности можно реализовать отдельного агента-наблюдателя, оценивающего ответы основного модуля и сверяющего их с эталонными, либо интегрировать систему с образовательной платформой для сбора реальной обратной связи. Такой подход повысит надежность и качество сервиса по мере его усложнения.

Студент группы М24-535 Агаев Р.Э.

4. Выводы по тенденциям

Анализ рассмотренных работ показывает, что развитие контейнерных технологий и систем искусственного интеллекта в последние годы движется в нескольких устойчивых направлениях.

Во-первых, наблюдается тесная интеграция ИИ-решений с контейнерными платформами, прежде всего с Kubernetes. Контейнеризация используется не только как средство упаковки приложений, но и как основа для масштабирования и управления нагрузкой. Для задач инференса больших языковых моделей всё чаще применяются инфраструктурные оптимизации, такие как кеширование промежуточных результатов, что позволяет снизить задержки и повысить пропускную способность сервисов без линейного роста вычислительных ресурсов.

Во-вторых, происходит переход от монолитных ИИ-моделей к распределённым и модульным архитектурам. Вместо одного крупного компонента используются наборы сервисов или агентов, каждый из которых отвечает за отдельную часть обработки. Такой подход упрощает развитие системы, повышает устойчивость к сбоям и лучше подходит для сложных сценариев, где требуется поэтапная обработка данных или взаимодействие нескольких логических компонентов.

Отдельное внимание уделяется вопросам безопасности, автоматизации и стандартизации. По мере роста масштабов ИИ-приложений усиливается необходимость в строгом управлении конфигурациями, контроле доступа и централизованном мониторинге. Использование декларативных Kubernetes-манифестов, CI/CD-процессов и автоматизированных механизмов восстановления позволяет сделать инфраструктуру более предсказуемой и управляемой, что особенно важно при работе с чувствительными данными.

Наконец, в ряде исследований прослеживается интерес к многоагентным ИИ-системам, особенно в прикладных областях, таких как образование. В таких системах несколько специализированных компонентов совместно решают задачу, обмениваясь результатами и дополняя друг друга. Ожидается, что подобные подходы будут применяться и в других областях, где требуется гибкость и адаптация к различным сценариям использования.

В целом можно сделать вывод, что контейнеризация и ИИ всё чаще рассматриваются как взаимодополняющие технологии. Kubernetes становится базовой платформой для развертывания и управления ИИ-сервисами, а современные архитектурные подходы позволяют строить масштабируемые, устойчивые и расширяемые системы, пригодные как для учебных, так и для практических проектов.

Результаты и выводы

В ходе выполнения проекта была реализована и проанализирована полноценная цепочка разработки контейнеризированного ИИ-приложения — от написания сервиса до его развертывания, масштабирования и мониторинга в среде Kubernetes. Практическая часть работы позволила закрепить навыки работы с контейнерами, оркестрацией и инструментами наблюдаемости, а аналитическая часть дала представление о современных тенденциях развития ИИ-систем в контейнерной инфраструктуре.

Студент группы М24-535 Агаев Р.Э.

Результаты проекта показывают, что даже относительно простые ИИ-сервисы могут выигрывать от использования Kubernetes за счёт удобного масштабирования, изоляции компонентов и централизованного мониторинга. Анализ научных публикаций подтвердил, что выбранные архитектурные решения соответствуют актуальным направлениям развития ИИ-платформ.

Достигнутые результаты

В рамках данной работы были достигнуты следующие результаты:

- разработано Java-приложение с REST API для анализа тональности текста с использованием mock-логики;
- выполнена контейнеризация приложения с использованием Docker, при этом размер итогового образа был оптимизирован;
- приложение развернуто в Kubernetes-кластере Minikube с использованием Deployment, Service, Ingress и Horizontal Pod Autoscaler;
- настроен сбор и визуализация метрик с помощью Prometheus и Grafana;
- выполнен аналитический обзор современных научных работ, посвящённых контейнеризации ИИ-приложений, инференсу больших языковых моделей и многоагентным архитектурам;
- сформулированы выводы о применимости рассмотренных подходов в учебных и практических проектах.

Возникшие трудности и их решения

В процессе выполнения проекта возникли ряд технических и организационных сложностей. Одной из основных проблем стала настройка взаимодействия между сервисами в Kubernetes, включая корректную работу Ingress и маршрутизацию запросов. Эти трудности были решены за счёт внимательной настройки манифестов и проверки конфигурации сетевых компонентов.

Отдельное внимание потребовалось уделить настройке мониторинга и интеграции приложения с Prometheus. Возникающие проблемы с доступностью метрик были устранены путём корректной конфигурации Actuator-эндпоинтов и ServiceMonitor.

Также определённую сложность представлял анализ научных публикаций, так как многие из них используют абстрактные или экспериментальные подходы. Для решения этой задачи был сделан акцент на практической интерпретации идей и их возможной адаптации к реальным контейнеризированным приложениям.

Заключение

В результате выполнения работы было показано, что Kubernetes является эффективной платформой для развертывания и сопровождения ИИ-сервисов, даже в условиях локального или учебного окружения. Контейнеризация упрощает управление жизненным циклом приложения, а

механизмы масштабирования и мониторинга позволяют адаптироваться к изменяющейся нагрузке.

Анализ современных исследований демонстрирует смещение фокуса от простого запуска моделей к построению масштабируемых, модульных и управляемых ИИ-систем. Использование таких подходов, как кеширование инференса, многоагентные архитектуры и автоматизация инфраструктуры, открывает возможности для создания более эффективных и устойчивых ИИ-приложений.

Полученные результаты могут быть использованы в дальнейшем для развития проекта, включая интеграцию реальных моделей машинного обучения, расширение архитектуры до многоагентной системы и применение более продвинутых механизмов автоскейлинга.

Ссылки и источники

1. Waseem M., Ahmad A. et al. *Containerization in Multi-Cloud Environment: Roles, Strategies, Challenges, and Solutions for Effective Implementation*. arXiv:2403.12980
2. Anonymous (LMCache Team). *An Efficient KV Cache Layer for Enterprise-Scale LLM Inference*. arXiv:2510.09665
3. Kamalov F., Calonge D. S. et al. *Evolution of AI in Education: Agentic Workflows*. arXiv:2504.20082
4. Kubernetes Documentation — https://kubernetes.io
5. Docker Documentation — https://docs.docker.com
6. Prometheus Documentation — https://prometheus.io
7. Grafana Documentation — https://grafana.com