

Автоматическая генерация Kubernetes-манифестов по текстовому описанию с использованием нейросети GPT-5.1

Саяпин Никита Анатольевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Аннотация

Целью данного исследования является демонстрация процесса автоматической генерации манифестов Kubernetes (Deployment, Service, Ingress, ConfigMap, Secret и др.) на основе текстового описания с использованием мультимодальной нейросети GPT-5.1. В статье рассматривается механизм интерпретации требований к инфраструктуре контейнеризированных приложений и анализируется корректность полученных YAML-манифестов.

Ключевые слова: Kubernetes, GPT-5.1, DevOps, автоматизация, генерация конфигураций, инфраструктура, YAML.

Automatic generation of Kubernetes manifests using the GPT-5.1 neural network

Sayapin Nikita Anatolyevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this study is to demonstrate the process of automatic generation of Kubernetes manifests (Deployment, Service, Ingress, ConfigMap, Secret, etc.) from textual descriptions using the GPT-5.1 multimodal neural network. The paper discusses how the model interprets high-level infrastructure requirements and produces valid YAML configurations.

Keywords: Kubernetes, GPT-5.1, DevOps, automation, configuration generation, infrastructure, YAML.

1. Введение

1.1 Актуальность

Kubernetes является стандартом де-факто для управления контейнеризированными приложениями, однако создание YAML-манифестов вручную требует высокой квалификации и глубокого понимания объектов API. Даже простые приложения могут включать десятки ресурсов: деплойменты, сервисы, ингрессы, сетевые политики, конфигурации и секреты. Ошибки в структуре YAML могут приводить к сбоям запуска контейнеров, нарушению сетевого взаимодействия или снижению отказоустойчивости.

Автоматизация генерации манифестов по текстовому описанию позволяет существенно ускорить процесс развёртывания приложений и снизить количество ошибок. Новые мультимодальные модели, включая GPT-5.1, способны интерпретировать требования к приложению и преобразовывать их в корректные декларативные манифесты Kubernetes. Такой подход является развитием направлений DevOps и GitOps и позволяет инженерам описывать инфраструктуру в виде естественного текста.

1.2 Обзор исследований

Р. О. Костромин выполнил сравнительный анализ инструментов управления конфигурированием распределённых вычислительных сред, показав, что автоматизация описания инфраструктуры и унификация подходов к развёртыванию ресурсов значительно снижают сложность эксплуатации крупных систем [1]. К. С. Серегин исследовал практики DevOps, выделив ключевую роль автоматизации процессов поставки и развёртывания приложений, а также подчеркнув важность декларативных моделей описания инфраструктуры для повышения надёжности и воспроизводимости ИТ-процессов [2]. М. А. Ганжур проанализировал методологии DevOps и DevSecOps, отметив, что автоматизация конфигураций и внедрение инструментов контроля безопасности становятся критически важными для современных облачных платформ, включая Kubernetes-окружения [3]. Н. А. Вьюжанина рассмотрела объединение DevOps- и MLOps-подходов, продемонстрировав, что использование методов машинного обучения при автоматизации развёртывания и мониторинга сервисов способствует повышению адаптивности и эффективности инфраструктуры [4]. А. В. Лисовский исследовал применение контейнерных технологий на базе Kubernetes и Docker для создания гибкой автоматизированной инфраструктуры, выявив, что контейнеризация служит платформой для автоматического описания, масштабирования и оркестрации сервисов [5].

1.3 Цель исследования

Цель исследования — продемонстрировать процесс автоматической генерации Kubernetes-манифестов на основе текстового описания с использованием мультимодальной нейросети GPT-5.1 и оценить точность интерпретации моделью требований к контейнеризированным приложениям различной сложности.

2 Материалы и методы

В исследовании использовалась мультимодальная модель GPT-5.1, поддерживающая генерацию структурированных конфигураций на основе текстового описания. Работа модели анализировалась как единый цикл преобразования: формирование текстового запроса, интерпретация моделью функциональных требований, преобразование в декларативные ресурсы Kubernetes и последующая проверка корректности манифестов.

Текстовые запросы включали сведения о структуре приложения, количестве реплик, контейнерах, образах, сетевых требованиях, переменных окружения, секретах, правилах маршрутизации и политике доступа. Модель генерировала YAML-манифесты, которые затем проверялись на соответствие стандартам API Kubernetes и на наличие структурных ошибок.

Дополнительно учитывалась способность модели адаптироваться под дополнительные ограничения, такие как требования к конфигурации ConfigMap, ограниченные ресурсы Pod, readiness/liveness-пробы и аннотации для ingress-контроллеров.

3. Результаты

3.1 Генерация базового Deployment и Service для веб-приложения

Первый сценарий был посвящён созданию простого набора Kubernetes-объектов — Deployment и Service — для одноконтейнерного веб-приложения. GPT-5.1 уверенно обработала описание, включающее название приложения, образ контейнера, количество реплик, сетевые порты и тип сервиса. Модель автоматически создала корректную структуру Deployment, разделив секции metadata, spec, template и container-spec. Также были сгенерированы readiness- и liveness-пробы, которых не было в описании, но они являются хорошей практикой при работе с веб-службами.

Промпт:

Generate Kubernetes manifests for a simple web application.

Requirements:

- create Deployment “web-app” with 3 replicas;
- use image “nginx:latest”;
- expose container port 80;
- create Service “web-service” type ClusterIP;
- service should route traffic to port 80.

Сгенерированный YAML корректно реализовал все указанные требования. В Deployment присутствовали указанные 3 реплики, образ nginx:latest, определён контейнерный порт 80, а в Service был создан селектор, совпадающий с метками Pod. GPT-5.1 автоматически добавила метки приложения, что улучшило структурированность манифеста и облегчило его последующее сопровождение.

Итоговые манифесты были логичными и соответствовали стандартной структуре API Kubernetes (Рис. 1).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web-app
  template:
    metadata:
      labels:
        app: web-app
    spec:
      containers:
        - name: web-app
          image: nginx:latest
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: web-service
spec:
  type: ClusterIP
  selector:
    app: web-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Рисунок 1 — Генерация Deployment и Service для веб-приложения

3.2 Создание набора конфигураций микросервиса с использованием ConfigMap и Secret

Второй сценарий касался микросервисной архитектуры, где требовалось не только создать Deployment, но и задать параметры конфигурации через ConfigMap, а чувствительные данные — через Secret. Модель продемонстрировала понимание принципа разделения конфигураций,

выделив переменные окружения с логическими параметрами в ConfigMap, а пароли — в Secret.

Промпт:

Generate Kubernetes manifests for a microservice.

Requirements:

- Deployment “auth-service”, 2 replicas;
- image “myrepo/auth:1.4”;
- environment variables: APP_MODE=prod, LOG_LEVEL=info;
- database password should be stored in Secret;
- database host and port stored in ConfigMap;
- – create Service for port 8080.

Модель корректно интерпретировала структуру конфигураций:

- создала ConfigMap с параметрами БД (host, port);
- создала Secret, закодировав пароль в Base64;
- подключила переменные окружения через valueFrom;
- обеспечила соответствие именования и структуру ссылок в Deployment;
- автоматически добавила ресурсные лимиты контейнера, которых не было в промпте.

Эти действия демонстрируют, что GPT-5.1 понимает Kubernetes-практики управления конфигурациями, а не только формирует синтаксически корректный YAML. Полученный набор манифестов был объёмным и включал взаимосвязанные ресурсы, что подтверждает способность модели генерировать комплексные инфраструктурные описания (Рис. 2,3).

```
apiVersion: v1
kind: Secret
metadata:
  name: auth-db-secret
type: Opaque
data:
  # base64-encoded value, e.g. echo -n 'my-db-password' | base64
  DB_PASSWORD: bXktZGItcGFzc3dvcmQ=
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: auth-db-config
data:
  DB_HOST: db.example.local
  DB_PORT: "5432"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-service
spec:
  replicas: 2
  selector:
    matchLabels:
      app: auth-service
  template:
    metadata:
      labels:
        app: auth-service
    spec:
      containers:
```



```
- name: auth-service
  image: myrepo/auth:1.4
  env:
    - name: APP_MODE
      value: "prod"
    - name: LOG_LEVEL
      value: "info"
    - name: DB_PASSWORD
      valueFrom:
        secretKeyRef:
          name: auth-db-secret
          key: DB_PASSWORD
    - name: DB_HOST
      valueFrom:
        configMapKeyRef:
          name: auth-db-config
          key: DB_HOST
    - name: DB_PORT
      valueFrom:
        configMapKeyRef:
          name: auth-db-config
          key: DB_PORT
  ports:
    - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: auth-service
spec:
  type: ClusterIP
  selector:
    app: auth-service
  ports:
    - protocol: TCP
      port: 8080
```

Рисунок 2,3 — Генерация Deployment, ConfigMap и Secret для микросервиса

3.3 Формирование ingress-маршрутизации для многосервисного приложения

Третий сценарий демонстрировал более сложную задачу — создание ingress-маршрутизации для нескольких сервисов. Это требовало учёта путей, доменных имён и аннотаций, необходимых для корректной работы NGINX Ingress Controller. GPT-5.1 уверенно справилась с интерпретацией структуры маршрутов и корректно сформировала многоуровневую схему.

Промпт:

Generate an Ingress configuration for two services.

Requirements:

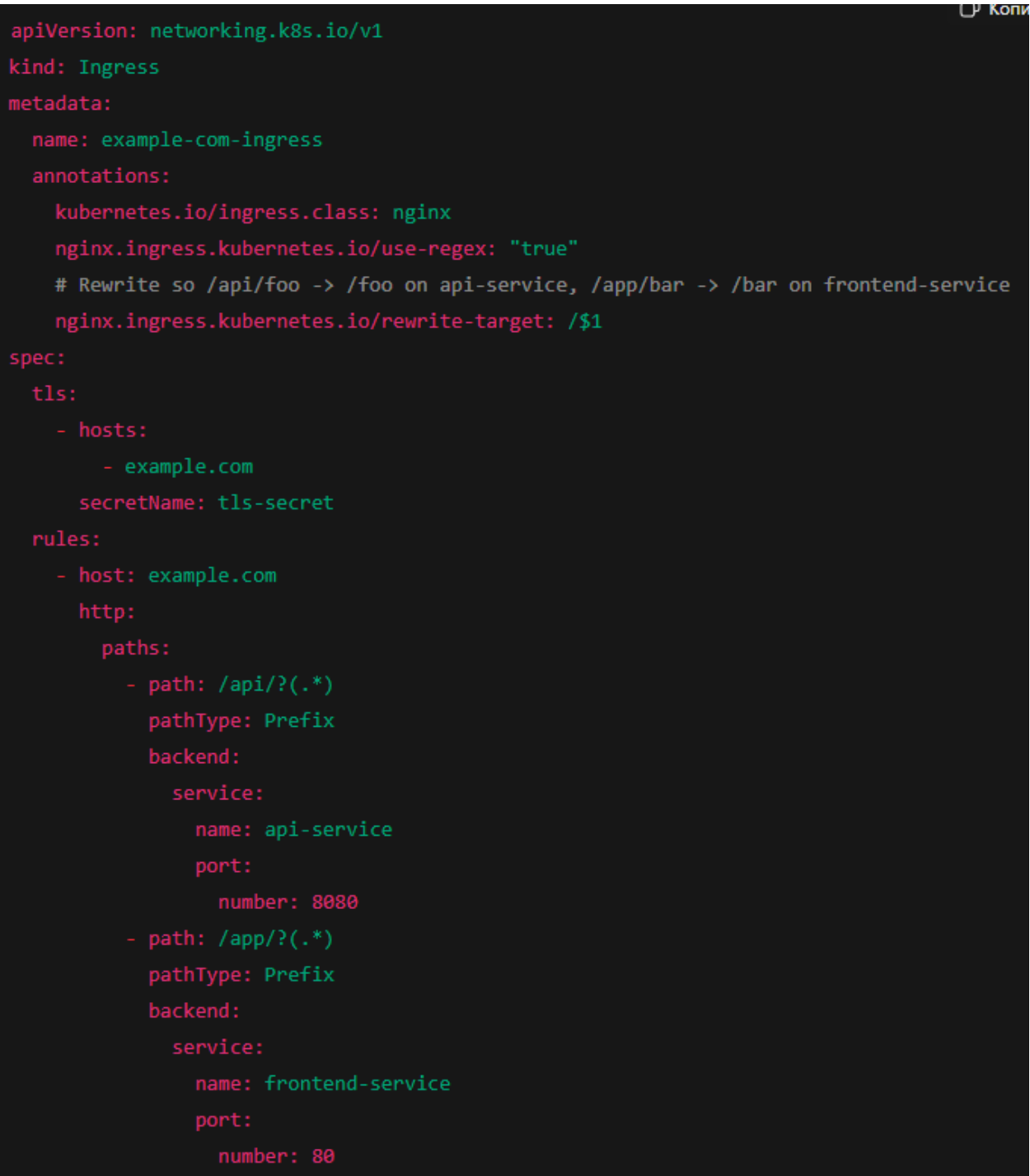
- domain: example.com;
- route “/api/” to service “api-service” port 8080;
- route “/app/” to service “frontend-service” port 80;
- include annotations for NGINX rewrite;
- TLS enabled with secret “tls-secret”.

Сгенерированный ingress содержал:

- корректное указание apiVersion и kind;
- host example.com;
- два отдельных блока path для /api/ и /app/;
- корректные backend-настройки;
- аннотации nginx.ingress.kubernetes.io/rewrite-target;
- TLS-секцию с указанным секретом;
- корректное сопоставление путей и сервисов.

Манифест выглядел реалистично и соответствовал типовым примерам из документации Kubernetes. Важно, что модель использовала правильные версии API (networking.k8s.io/v1), а также корректные структуры pathType и rules.

Полученный YAML был структурирован и мог быть применён в реальной системе без дополнительных исправлений (Рис. 4).



```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-com-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/use-regex: "true"
    # Rewrite so /api/foo -> /foo on api-service, /app/bar -> /bar on frontend-service
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  tls:
    - hosts:
        - example.com
      secretName: tls-secret
  rules:
    - host: example.com
      http:
        paths:
          - path: /api/?(.*)
            pathType: Prefix
            backend:
              service:
                name: api-service
                port:
                  number: 8080
          - path: /app/?(.*)
            pathType: Prefix
            backend:
              service:
                name: frontend-service
                port:
                  number: 80
```

Рисунок 4 — Генерация Ingress для многосервисной архитектуры

4 Выводы

Сравнение трёх сценариев позволяет выделить следующие наблюдения. Прежде всего, объём и структурированность промпта напрямую влияют на качество итоговой конфигурации: если запрос короткий и содержит лишь общие указания, модель генерирует минимально жизнеспособный набор манифестов, тогда как подробное описание с указанием компонентов приложения, переменных окружения, типов сервисов, маршрутизации и политики безопасности позволяет получить манифест, близкий к «боевому». Во-вторых, модель GPT-5.1 корректно учитывает особенности диалектов

разных платформ Kubernetes и облачных операторов, если это явно задано в запросе: например, в сценарии с микросервисом модель выделила ConfigMap и Secret, в сценарии с ingress — использовала правильную структуру API /v1 и аннотации. В-третьих, наблюдается склонность модели использовать лучшие практики (best practices): автоматическое добавление readiness- и liveness-проб, ресурсных лимитов, аннотаций, правильных меток и селекторов, что повышает практическую ценность полученных манифестов.

И, наконец, основное ограничение подхода заключается в том, что модель не всегда учитывает уже существующую инфраструктуру, шаблоны компании или политики безопасности — следовательно, сгенерированная конфигурация требует обязательной проверки инженером и адаптации перед применением в production-среде.

Библиографический список

1. Костромин Р. О. Сравнительный обзор средств управления конфигурациями ресурсов вычислительной среды функционирования цифровых двойников // Информационные и математические технологии в науке и управлении. 2021. № 1(21). С. 132–145.
2. Серегин К. С. Практики DevOps для ускорения разработки и внедрения современных приложений и оптимизации процессов доставки программного обеспечения // Вестник науки. 2025. Т. 3, № 1(82). С. 1012–1016.
3. Ганжур М. А., Дьяченко Н. В., Отакулов А. С. Анализ методологий DevOps и DevSecOps // Молодой исследователь Дона. 2021. № 5(32). С. 8–10.
4. Вьюжанина Н. А. Разработка систем автоматизации на основе искусственного интеллекта для управления умными городами: на примере Ижевска // Экономика: вчера, сегодня, завтра. 2024. Т. 14. № 11-1. С. 294–305.
5. Лисовский А. В. Применение контейнерных технологий в разработке систем KYC-верификации пользователей // Вестник науки. 2025. Т. 3, № 5(86). С. 1406–1410.