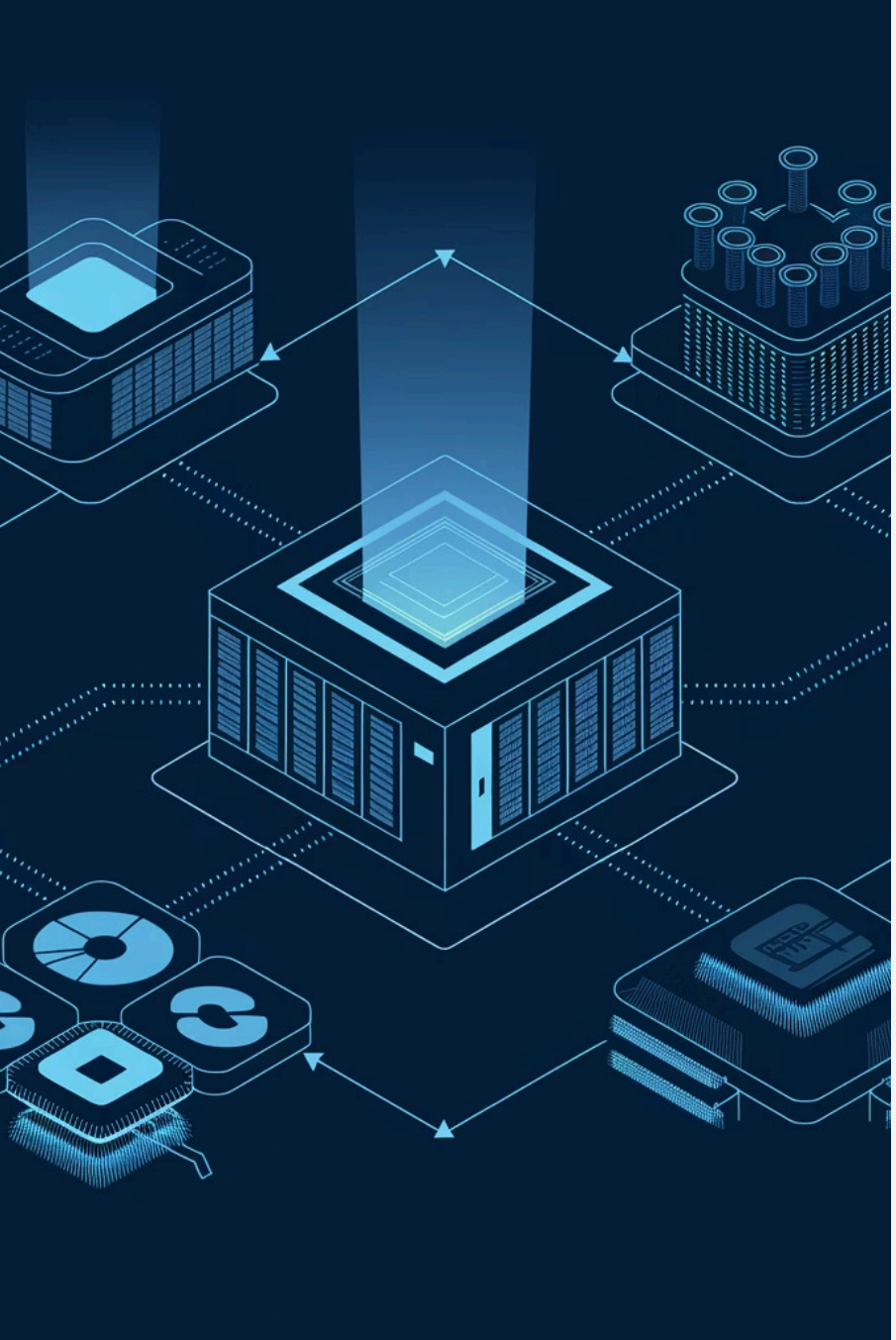


Введение в Triton Inference Server

NVIDIA Triton Inference Server - это современная платформа для развёртывания моделей машинного обучения в продакшене, отвечающая всем требованиям промышленного уровня. Triton предоставляет единый интерфейс для работы с моделями, независимо от того, в каком формате они были обучены: будь то модели из TensorFlow, PyTorch, ONNX или оптимизированные с помощью TensorRT.

Платформа Triton обеспечивает оптимизацию инференса, унифицированный интерфейс, масштабируемость, а также встроенные механизмы для мониторинга и отладки.

И **по Иван Копылов**

История и эволюция серверов инференса

1

Ранние подходы

На заре эпохи машинного обучения модели разворачивались в виде отдельных серверных приложений, зачастую жестко привязанных к используемым фреймворкам. Такие решения страдали от недостаточной универсальности и ограниченных возможностей оптимизации.

2

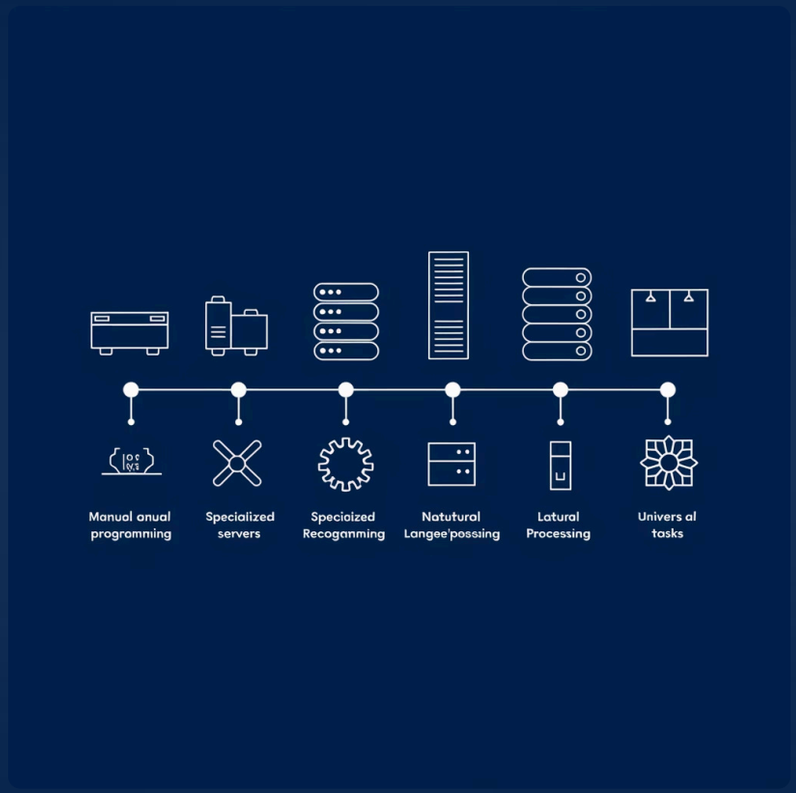
Специализированные серверы

Появились решения, такие как TensorRT Inference Server, TensorFlow Serving и ONNX Runtime, ориентированные на конкретные фреймворки и предлагающие значительные улучшения в производительности.

3

Универсальное решение

NVIDIA разработала Triton Inference Server, объединяющий лучшие практики и обеспечивающий поддержку множества фреймворков, пакетный инференс и гибкие механизмы управления ресурсами.



Поддерживаемые фреймворки и форматы моделей

TensorFlow

Triton полностью поддерживает модели, обученные с использованием TensorFlow, включая оптимизированные версии, экспортированные в формате SavedModel.

PyTorch

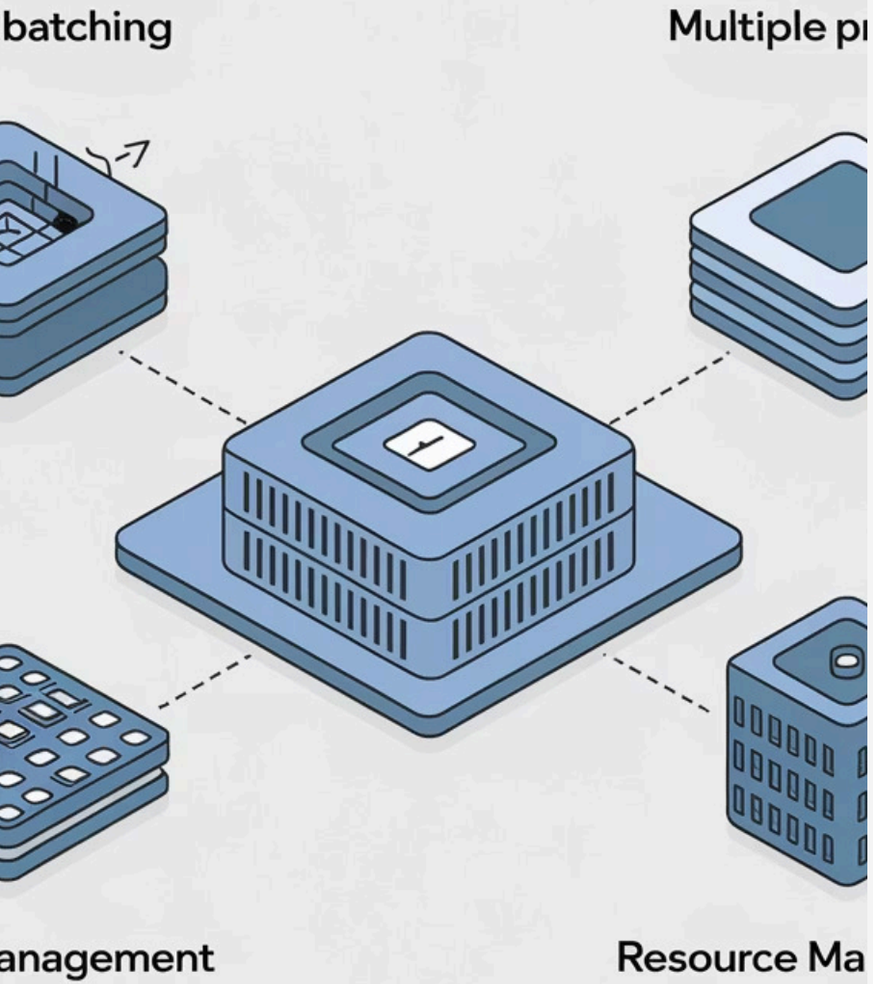
Модели, разработанные в PyTorch, могут быть экспортированы в формат TorchScript или ONNX и развернуты через Triton.

ONNX

Triton поддерживает модели ONNX, оптимизируя граф вычислений для высокой производительности.

TensorRT

Triton позволяет использовать модели, оптимизированные с помощью TensorRT, что обеспечивает минимальные задержки и максимальную производительность.



Основные возможности Triton Inference Server

1 Пакетирование запросов

Поддержка динамического и статического пакетирования запросов позволяет объединять несколько инференс-запросов в один пакет для повышения пропускной способности.

2 Поддержка нескольких протоколов

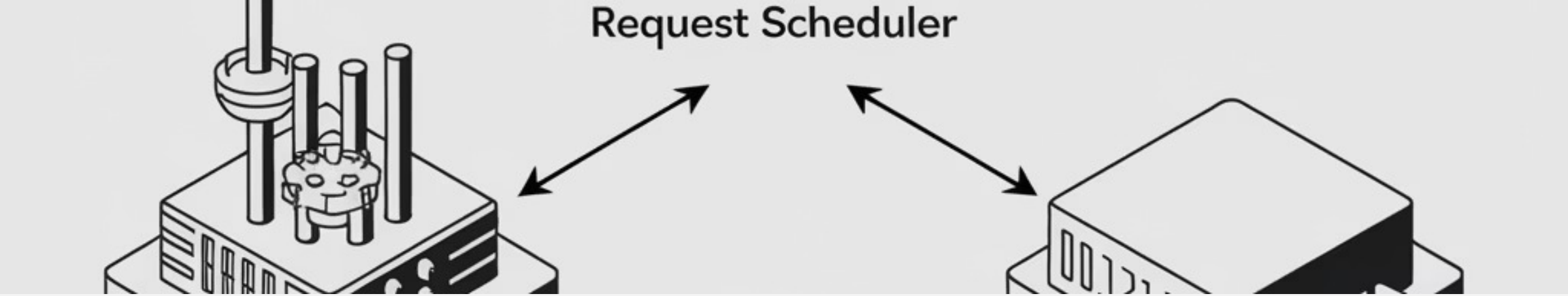
Triton предоставляет REST и gRPC API для взаимодействия, что облегчает интеграцию с различными системами.

3 Управление версиями моделей

Возможность развертывания нескольких версий одной модели позволяет проводить A/B-тестирование и плавное обновление.

4 Динамическое управление ресурсами

Автоматическое распределение нагрузки между GPU и CPU обеспечивает высокую производительность даже при пиковых нагрузках.



Архитектурные особенности Triton

Менеджер моделей

Отвечает за загрузку, обновление и управление жизненным циклом моделей, а также за поддержку нескольких версий одной модели.

Планировщик запросов

Организует очереди инференс-запросов, осуществляет пакетирование и распределяет нагрузку между вычислительными ресурсами с учетом приоритетов.

Интерфейс API

Единый слой, предоставляющий REST и gRPC API для отправки запросов и получения результатов, абстрагируя детали внутренней реализации.

Модуль мониторинга

Встроенные механизмы сбора метрик и логирования позволяют отслеживать задержки, пропускную способность и использование ресурсов в режиме реального времени.

Преимущества использования Triton в промышленности

Повышенная производительность

Динамическое пакетирование, аппаратное ускорение и эффективное распределение нагрузки обеспечивают высокую пропускную способность и минимальные задержки.

Универсальность

Поддержка множества форматов и единый API упрощают интеграцию Triton в существующую инфраструктуру.

Надёжность

Управление версиями, мониторинг и интеграция с оркестраторами обеспечивают стабильную работу и быстрое восстановление при сбоях.

Экономическая эффективность

Оптимальное использование ресурсов и унифицированная платформа снижают затраты на инфраструктуру и поддержку.

Формулы для оценки эффективности инференса

1 Средняя задержка

Average Latency = (Σ Latency_i) / N, где N – общее число запросов.

2 Пропускная способность

Throughput = Количество обработанных запросов / Время работы (секунды)

3 Утилизация ресурсов

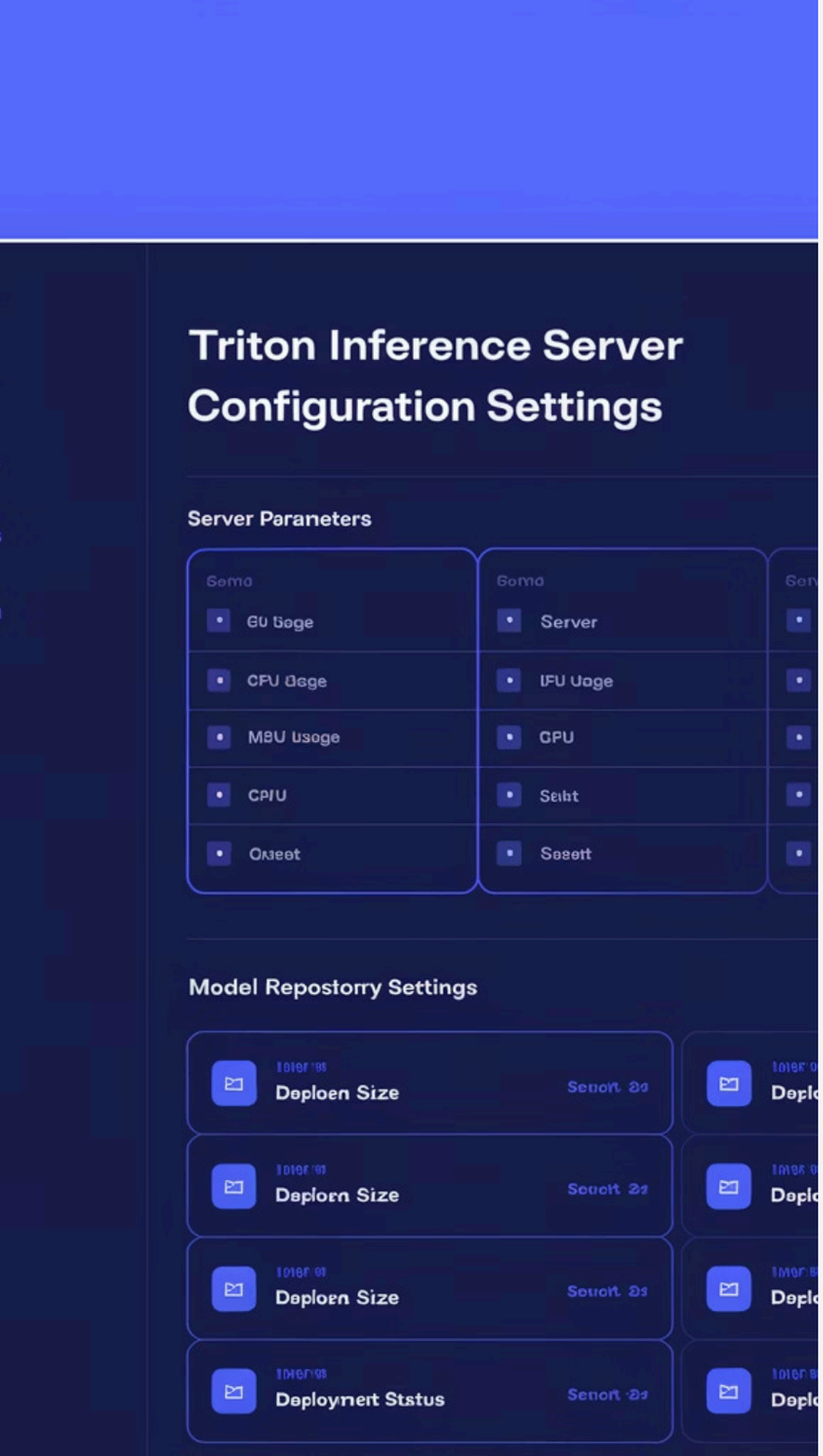
Utilization = (Время активной работы устройства / Общее время работы) × 100%

Установка и запуск Triton в контейнере Docker

```
docker pull nvcr.io/nvidia/tritonserver:23.02-py3
docker run --gpus=all --rm -p8000:8000 -p8001:8001 -p8002:8002 \
-v /path/to/model_repository:/models \
nvcr.io/nvidia/tritonserver:23.02-py3 tritonserver \
--model-repository=/models
```

В этом примере:

- Параметр --gpus=all обеспечивает использование всех доступных GPU.
- Порты 8000, 8001 и 8002 открыты для REST, gRPC и метрик соответственно.
- Локальная директория /path/to/model_repository монтируется как репозиторий моделей.



Конфигурация серверных параметров Triton

1 **--model-repository**

Путь к директории с моделями.

2 **--strict-model-config=false**

Позволяет серверу автоматически генерировать конфигурации для моделей без файла config.pbtxt.

3 **--allow-soft-placement=true**

Включает возможность динамического распределения запросов между устройствами.

Пример запуска с дополнительными параметрами:

```
tritonserver --model-repository=/models \
--strict-model-config=false \
--allow-soft-placement=true
```

Настройка конфигурационных файлов моделей

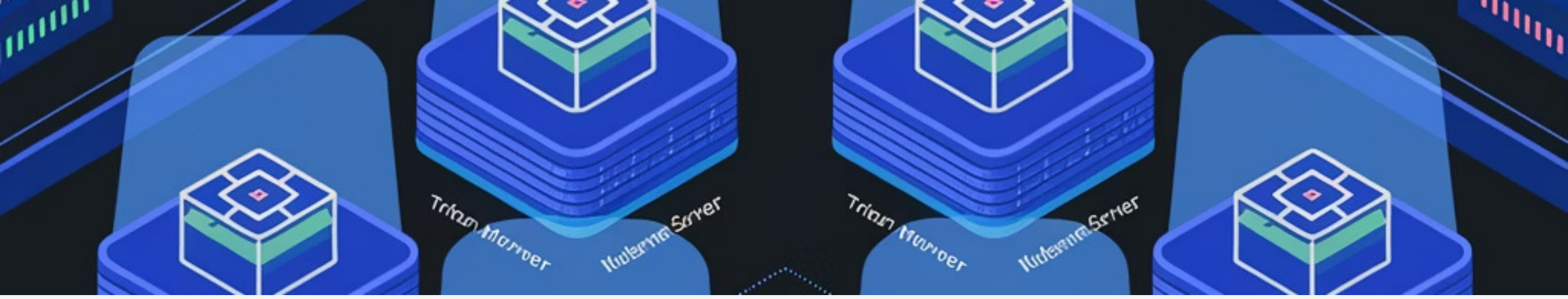
Конфигурационные файлы (например, config.pbtxt) определяют параметры инференса для каждой модели: входные и выходные данные, максимальный размер пакета, динамическое пакетирование и прочее.

```
name: "image_classification_model"
platform: "onnxruntime_onnx"
max_batch_size: 64
input [
 {
   name: "input"
   data_type: TYPE_FP32
   dims: [ 3, 224, 224 ]
 }
]
output [
 {
   name: "output"
   data_type: TYPE_FP32
   dims: [ 1000 ]
 }
]
dynamic_batching {
 preferred_batch_size: [ 16, 32 ]
 max_queue_delay_microseconds: 10000
}
```

Развертывание с Docker Compose

Docker Compose позволяет объединить Triton Inference Server с другими сервисами в одном файле конфигурации.

```
version: "3.8"
services:
  triton:
    image: nvcr.io/nvidia/tritonserver:23.02-py3
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
    ports:
      - "8000:8000"
      - "8001:8001"
      - "8002:8002"
    volumes:
      - ./model_repository:/models
    command: >
      tritonserver --model-repository=/models
  prometheus:
    image: prom/prometheus:latest
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
```



Развертывание в Kubernetes

Kubernetes обеспечивает автоматическое масштабирование и управление ресурсами. Для развертывания Triton в Kubernetes создаются манифесты Deployment и Service.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: triton-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: triton
  template:
    metadata:
      labels:
        app: triton
    spec:
      containers:
      - name: triton
        image: nvcr.io/nvidia/tritonserver:23.02-py3
        args: ["--model-repository=/models"]
        resources:
          limits:
            nvidia.com/gpu: 1
        ports:
        - containerPort: 8000
        - containerPort: 8001
        - containerPort: 8002
        volumeMounts:
        - name: model-repo
          mountPath: /models
      volumes:
      - name: model-repo
        hostPath:
          path: /mnt/models
```

Мониторинг и сбор метрик Triton

1. **Средняя задержка**

   Отслеживание времени обработки запросов позволяет оценить производительность системы.

2. **Пропускная способность**

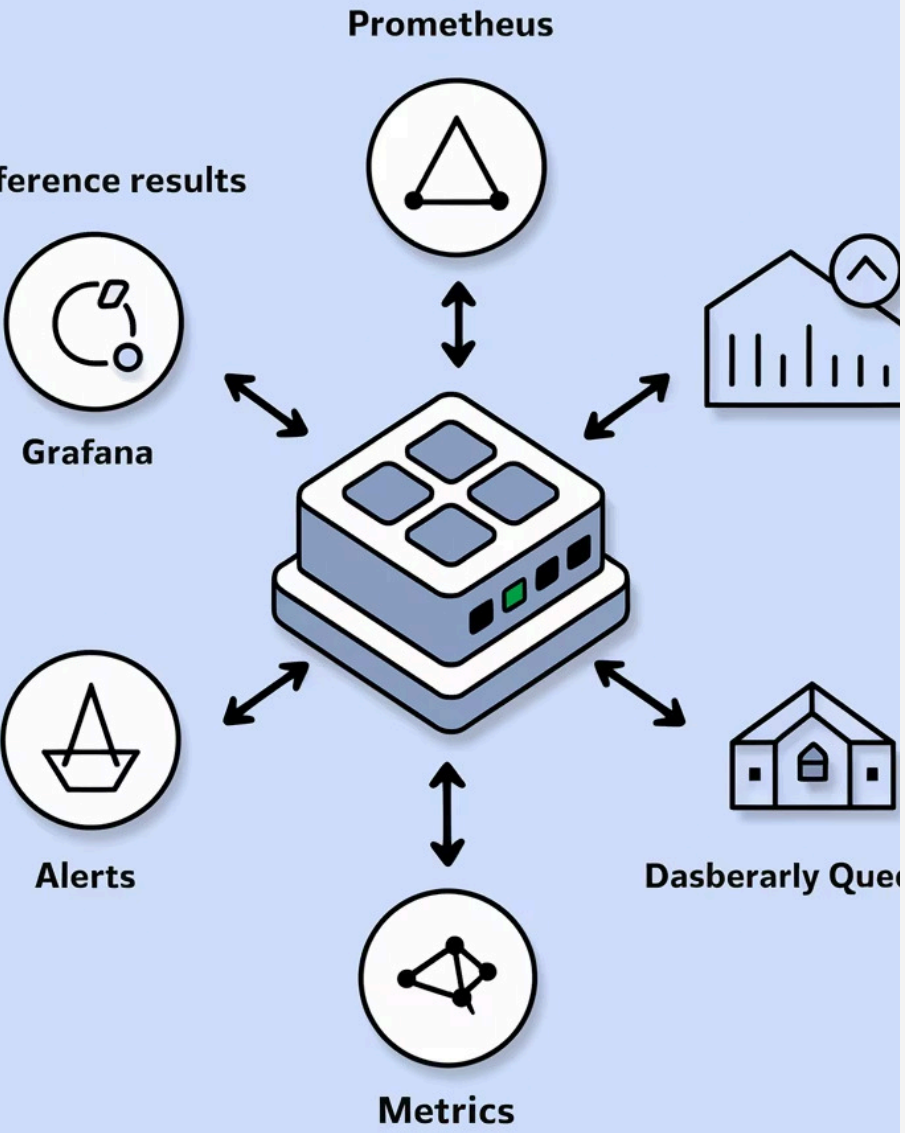
   Измерение количества обработанных запросов в единицу времени помогает оценить эффективность сервера.

3. **Утилизация GPU и CPU**

   Мониторинг использования вычислительных ресурсов позволяет оптимизировать нагрузку.

4. **Состояние очереди запросов**

   Анализ очереди помогает выявить узкие места и оптимизировать пакетирование.



Интеграция с Prometheus и Grafana

Triton позволяет экспортировать метрики, которые затем собираются Prometheus и отображаются в удобных дашбордах Grafana.

```
triton_inference_server_inference_count{model_name="image_classification_model"}
```

Этот запрос возвращает общее количество инференс-запросов для заданной модели, что позволяет отслеживать нагрузку и эффективность сервера.

Управление ресурсами и автошкалирование

Динамическое масштабирование

Автоматическое добавление реплик Triton в Kubernetes при увеличении нагрузки позволяет гибко реагировать на изменения трафика.

Настройка лимитов

Задание лимитов на использование CPU, памяти и GPU позволяет избежать перегрузки инфраструктуры и обеспечить стабильную работу.

Формула для расчёта загрузки GPU:

```
GPU Utilization (%) = (Время работы GPU под нагрузкой / Общее время) × 100%
```

Практические сценарии использования Triton

Компьютерное зрение

Triton позволяет обслуживать модели распознавания объектов и классификации изображений, обученные с использованием TensorFlow или PyTorch, с оптимизацией через TensorRT.

Обработка естественного языка

Быстрый инференс моделей NLP для систем онлайн-поддержки и чат-ботов, с возможностью плавного обновления версий и A/B-тестирования.

Рекомендательные системы

Обслуживание сложных рекомендательных моделей с высокой пропускной способностью для e-commerce платформ.

Обеспечение безопасности Triton Inference Server

1 Принцип наименьших привилегий

Запуск процессов от имени непривилегированных пользователей для минимизации рисков.

2 Безопасная конфигурация

Использование минимальных базовых образов и регулярное обновление контейнеров.

3 Сканирование уязвимостей

Регулярное автоматизированное сканирование с помощью инструментов, таких как Trivy или Clair.

4 Мониторинг безопасности

Централизованное логирование и анализ аномалий в поведении сервера.

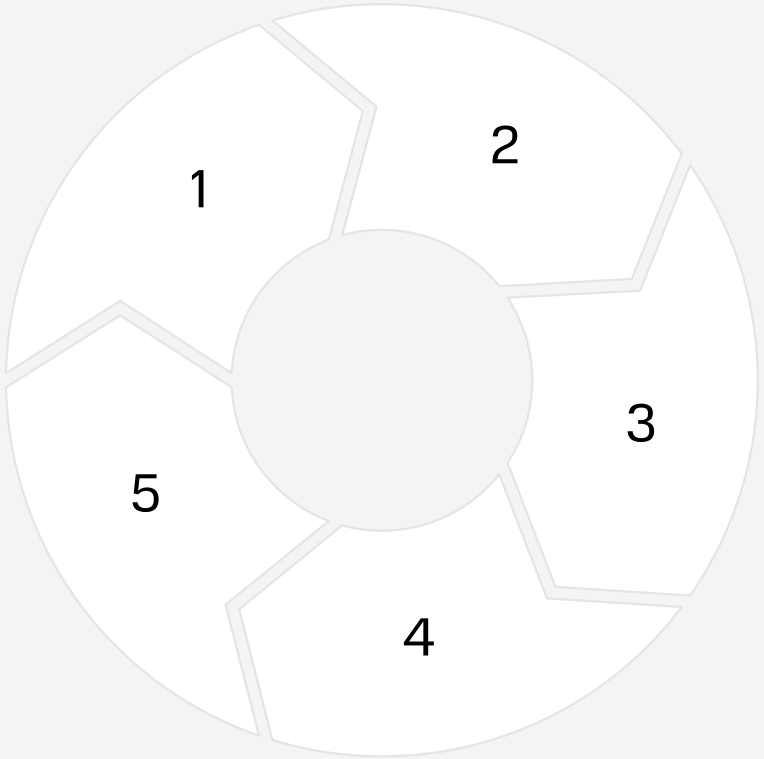
Управление жизненным циклом моделей

Разработка

Создание и обучение моделей в различных фреймворках.

Обновление

Внедрение улучшений и новых версий моделей.



Тестирование

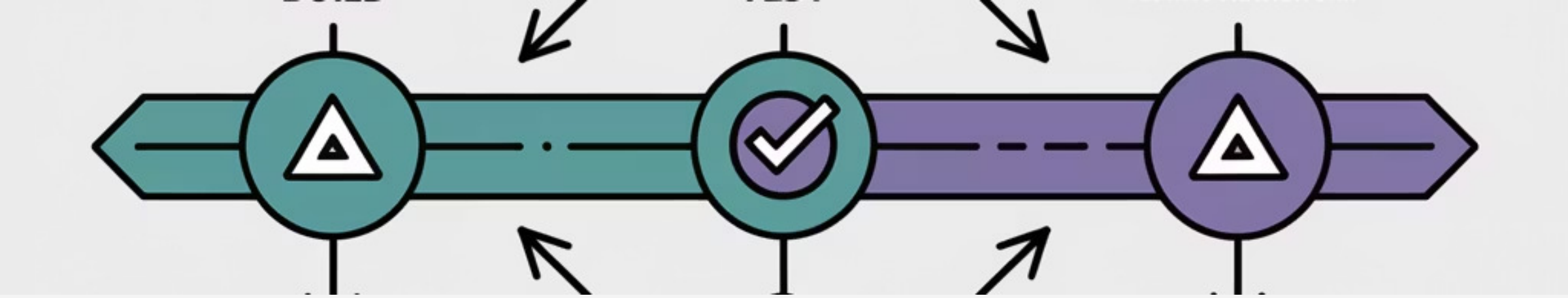
Проверка производительности и точности моделей.

Развертывание

Интеграция моделей в Triton Inference Server.

Мониторинг

Отслеживание производительности и сбор метрик.



CI/CD интеграция для Triton

Интеграция Triton Inference Server в процессы CI/CD позволяет автоматизировать сборку, тестирование и развертывание обновлений моделей.

```
stages:
- build
- test
- deploy

build:
  stage: build
  script:
    - docker build --no-cache -t registry.example.com/triton_model:$CI_COMMIT_SHA .
    - docker push registry.example.com/triton_model:$CI_COMMIT_SHA

test:
  stage: test
  script:
    - docker run --rm registry.example.com/triton_model:$CI_COMMIT_SHA pytest --maxfail=1 --disable-warnings -q

deploy:
  stage: deploy
  script:
    - docker pull registry.example.com/triton_model:$CI_COMMIT_SHA
    - docker tag registry.example.com/triton_model:$CI_COMMIT_SHA registry.example.com/triton_model:latest
    - docker push registry.example.com/triton_model:latest
```

Заключение

NVIDIA Triton Inference Server представляет собой мощное решение для развёртывания и эксплуатации моделей машинного обучения в промышленных условиях. Платформа обеспечивает:

1. **Универсальность**

   Поддержка различных форматов и фреймворков моделей.

2. **Производительность**

   Оптимизация инференса и динамическое управление ресурсами.

3. **Надёжность**

   Отказоустойчивость и интеграция с системами оркестрации.

4. **Экономичность**

   Оптимальное использование ресурсов и снижение затрат на поддержку.