Слайд 1. Liveness и Readiness Probes

Что проверяет Kubernetes:

- Контейнер жив или завис
- Контейнер готов обслуживать трафик
- Приложение действительно работает, а не просто «в контейнере что-то запущено»

Зачем это нужно:
Ошибки приложения должны приводить к правильным действиям: перезапуск или исключение из балансировки.

Слайд 2. Проблема без проверок

Ситуация:
Контейнер завис → процесс внутри работает, но не отвечает.
Kubernetes без probe считает его здоровым.

Последствия для сервиса:

- Трафик продолжает идти в нерабочий экземпляр
- Нет автоматического восстановления
- Ошибки на стороне пользователей
- Дебаг занимает больше времени

Слайд 3. Что такое Liveness Probe

Назначение:
Определяет, нужно ли перезапустить контейнер.

Критерий:
Проверка не проходит несколько раз → kubelet перезапускает контейнер.

Когда применять:

- Возможны «зависания» или deadlock в приложении
- Нужен автоматический recovery
- Есть endpoint для здоровья или команда проверки

Слайд 4. Что такое Readiness Probe

Назначение:
Определяет, можно ли направлять трафик в контейнер.

Критерий:
Если probe не проходит → pod остаётся Running, но исключается из endpoint-листа сервиса.

Когда применять:

- Приложение стартует дольше контейнера
- Нужны миграции перед запуском
- Временами приложение недоступно, но его не нужно перезапускать

Слайд 5. Поведение Kubernetes

Liveness FAIL:

- Контейнер перезапускается
- Счётчик RESTARTS увеличивается
- Pod остаётся в статусе Running

Readiness FAIL:

- Под не получает трафик
- Сервис исключает Pod из списка адресов
- Никаких перезапусков

Слайд 6. Виды проверок

HTTP проверка (httpGet)
Подходит для большинства web-приложений. Ожидается код 200.

TCP проверка (tcpSocket)
Проверяет, открыт ли порт. Минимальный вариант.

Exec проверка (exec)
Запускает команду внутри контейнера.
Подходит, когда приложение не HTTP-сервис.

Слайд 7. Ключевые параметры

- **initialDelaySeconds** — задержка перед первой проверкой
- **periodSeconds** — как часто проверять
- **timeoutSeconds** — максимальное время ответа
- **failureThreshold** — после скольких ошибок реагировать
- **successThreshold** — сколько успешных проверок считать контейнер снова healthy

Применение:

- Liveness → медленные значения, чтобы избежать ложных рестартов
- Readiness → более чувствительные значения

Слайд 8. Пример, как это выглядит в YAML

```yaml
livenessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 5
  periodSeconds: 10

readinessProbe:
  httpGet:
    path: /ready
    port: 8080
  initialDelaySeconds: 2
  periodSeconds: 3
```

🟦 Слайд 9. Демонстрация: нерабочий контейнер

Шаги:

1. Запустить Deployment без probe
2. Проверить pod: он всегда Running
3. Сломать приложение внутри контейнера
4. Убедиться: Kubernetes не замечает проблему

Вывод: без probe контейнер = «всё хорошо», даже если он фактически сломан.

Слайд 10. Демонстрация Liveness

Идея:
Показать перезапуск при ошибке.

Шаги:

1. Добавить liveness probe
2. Искуственно сломать endpoint
3. Посмотреть `kubectl describe pod`
4. Убедиться: контейнер перезапущен
5. Проверить — счётчик RESTARTS вырос

🟦 Слайд 11. Демонстрация Readiness

Идея:
Показать, как Pod перестаёт получать трафик.

Шаги:

1. Добавить readiness probe
2. Вызывать ошибку проверки
3. Проверить `kubectl get endpoints`
4. Pod исчезнет из списка, но не будет перезапущен

Слайд 12. Итог: разница в одной таблице

Что проверяет	Liveness	Readiness
Контейнер запущен?	✓	✓
Готов обслуживать трафик?	✗	✓
Ошибка → перезапуск	✓	✗
Ошибка → убрать из сервисов	✗	✓
Когда использовать	Зависания	Долгий старт, временная недоступность

Слайд 13. Практическое задание

Задача 1:
Добавить liveness и readiness в Deployment.

Задача 2:
Симулировать ошибку liveness и увидеть перезапуск.

Задача 3:
Симулировать ошибку readiness и убедиться, что Pod пропадает из endpoints.

Задача 4:
Вернуть приложение в рабочее состояние.