Load2SRE: от нагрузки к доступности, не утратив производительность

Кирилл Юрков
SRE Team Lead

Отправная точка - нагрузочный тестировщик

Среда обитания: тестовый стенд

Основные задачи:

- скрипты
- запуски
- документация тестов
- проверка требований от заказчика
- заглушки
- автоматизация

Нагрузочный тестировщик - инженер производительности

Что меняется?

- Помимо проверки требований - локализация проблем производительности
- Тюнинг производительности
- Умная автоматизация
- Инструментарий

Заглянем в прод

- Приносят баги с прода (редко)
- Испытания на проде (совсем редко)

Инженер производительности

Среда обитания: тестовый стенд и иногда прод

Основные задачи:

- инструментарий
- скрипты и запуски
- документация тестов
- проверка требований от заказчика
- локализация проблем
- заглушки
- умная автоматизация
- иногда проблемы прода
- совсем редко испытания на проде

инженер производительности - SRE

Что меняется?

1. Помимо проверки требований - **создаем** эти **требования**
2. **??**

SRE

Среда обитания: ??

Основные задачи:

- ??

Чем занимается SRE?

- **Reliable**
- **Stability**
- **Performance**

Производительность

1. Нужно ли нагрузочное тестирование в SRE?
2. Как оценивать capacity и почему в нём есть слово planning?
3. Помимо продуктов вам предстоит оценивать производительность всей инфраструктуры

Стабильность

1. Incident management (см. ITIL)
   a. Реагируем на текущие проблемы
   b. Предотвращаем новые проблемы
2. Внедряем практики
   a. Deployment/Runtime requirement

Прикладные требования к сервису:

1. Настроены endpoint'ы для Readiness/ Liveness probes
2. Минимально развертывание сервиса осуществляется в 2 инстанса
3. Умеет обрабатывать SIGTERM (закрываются сетевые соединения, сохраняются данные, завершаются любые работы);
4. Не хранит состояние и умеет запускаться в нескольких экземплярах (позволяет горизонтально масштабироваться и обеспечивать отказоустойчивость);
5. Пишет лог в stdout;
6. Выставлены целочисленные значения CPU requests (>=1)
   a. Позволяет избегать троттлинга по CPU при голодании Ноды по ресурсам
   b. JVM | CPU requests =< 1 приводит к использованию SerialGC
7. Используется JDK 11+ версии

Проверяем через валидацию манифестов - kyverno

Надежный

1. Паттерны
   a. Retryier
   b. Burst/Rate Limiter
   c. Circuit breaker
   d. Balancer
   e. Graceful degradation/fallback
2. Нужны ли тут какие то тесты?
   a. Chaos engineering
   b. Испытания в проде :)

Chaos engineering

Самый простой тест надежности и показательный:

1. Запускаем тест
2. Разрушаем 1 инстанс нашего приложения - мы используем chaos-mesh
3. Проверяем что нефункциональные требования выполняются

Ворота в прод

1. Подключаемся к проблемам, предотвращаем новые
2. Следим за ёмкостью
3. Проводим испытания
4. Создаем требования и контролируем их

SRE

Среда обитания: продуктовый стенд и иногда тестовые стенды

Основные задачи:

- создаем требования
- локализация проблем **прода**
- стабильность
- производительность
- надежность
- ?

Доступность

- Это время
  - SLI/SLO/SLA - Error Budget

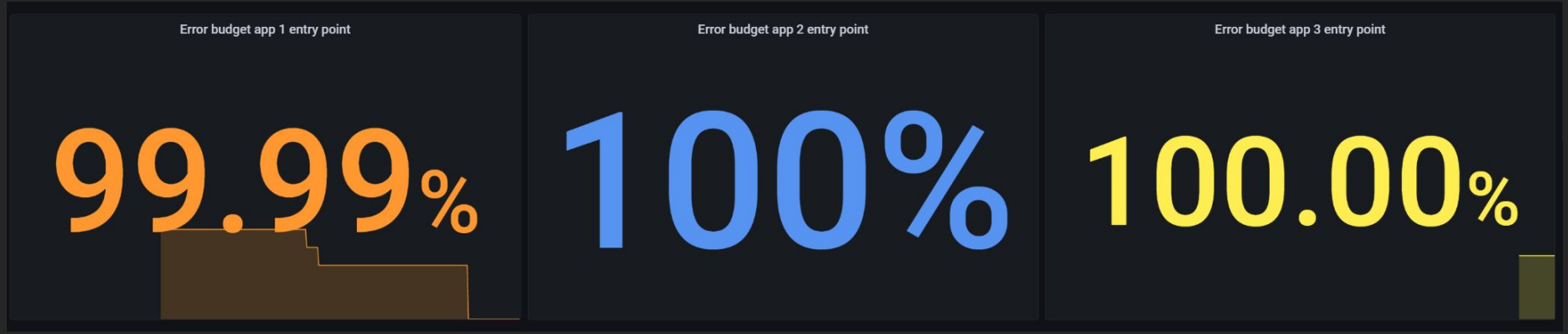
Error budget на коленке

1. Недоступность входных точек в бизнес процессы
2. Недоступность главного процесс, который приносит прибыль и каждого его этапа
3. Потраченный бюджет == приоритезация задач отказоустойчивости

Бюджет входных точек

Мы тратим бюджет когда:

- Отвечаем дольше чем X времени на N% запросов
- Отвечает определенной ошибкой на Y% запросов



Бюджет на аудиторию

Надежных стрельб!

Ссылочки:

1. ITIL Incident managment - https://www.itil-docs.com/blogs/incident-management/incident-management-process
2. Валидация манифестов - https://kyverno.io
3. Chaos mesh - https://chaos-mesh.org
4. Классное коммьюнити про перформанс - https://t.me/qa_load
5. Мой telegram для всех вопросов на которые не успею ответить - https://t.me/login40k