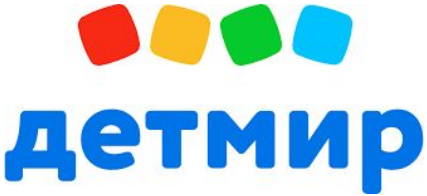
Дорога монолита в облако и обратно

Листеренко Ростислав
Архитектор

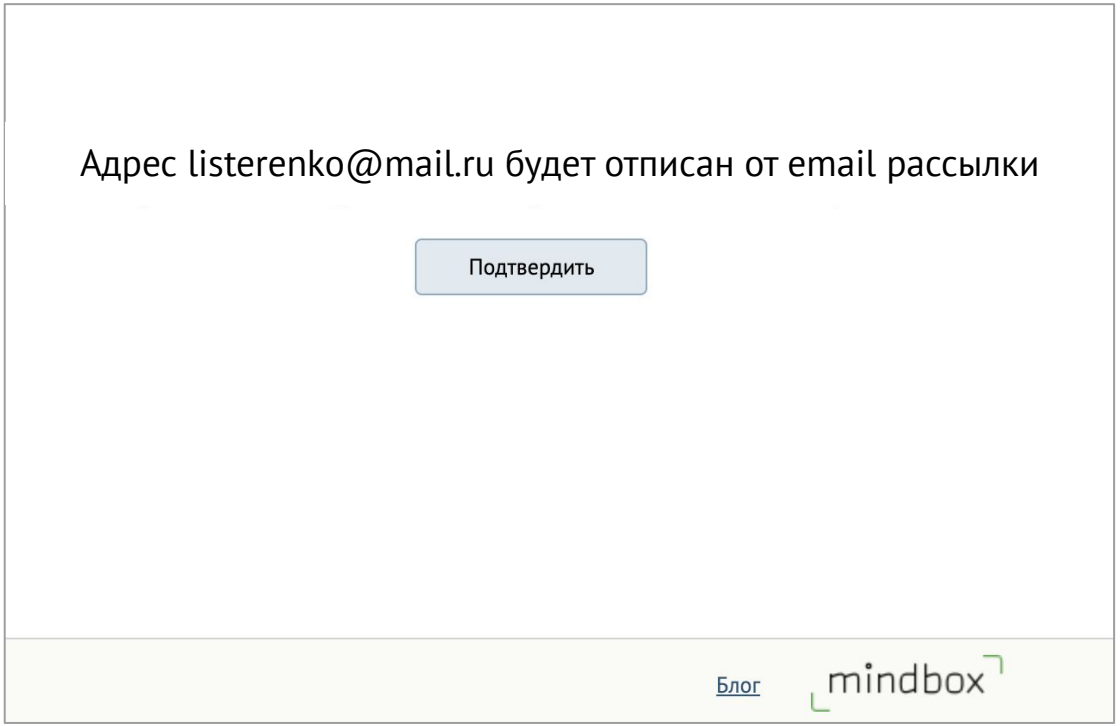
listerenko@mindbox.ru
fb.com/rostislav.listerenko

mindbox

Где можно столкнуться с нашей работой



… и самая узнаваемая ее часть



План доклада

1. Зачем переезжать?
2. Подготовка и планирование
3. Мониторинг
4. Препятствия и проблемы*
5. Опыт с прода
6. Итоги и выводы

*в том числе нерешенные

1 Зачем переезжать?

О платформе Mindbox

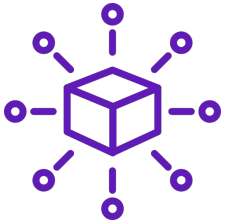
Глазами бизнеса:
SaaS для автоматизации маркетинга

- 500 компаний-клиентов
- SLA
  — Доступность 24/7 (> 500 000 rpm)
  — Скорость отправки персонализированных сообщений (> 100 млн в день)
- Программа лояльности, интеграции с кассами
- Виджеты на клиентские сайты

Глазами разработчика: 10 лет разработки

Монолит

- Инсталляция под каждого клиента
- 1.4 млн строк кода
- 350 таблиц в БД
- 3 приложения
- 70% — .NET Standard 2.0
  30% — .NET 4.8



30 микросервисов

- Мультитенатные
- 8 – .NET 4.8,
  22 – .NET Core 3.1

Глазами ops

- **Windows-сервера**
  - Ручное управление ресурсами для клиентов
  - 84 машины
  - 2022 ядра, ~7 Tib оперативной памяти

- **Неподъемный деплой**
  - 34 императивных шага (powershell и scriptcs)
  - ~4 часа в хорошую погоду

Схема с зависимостями

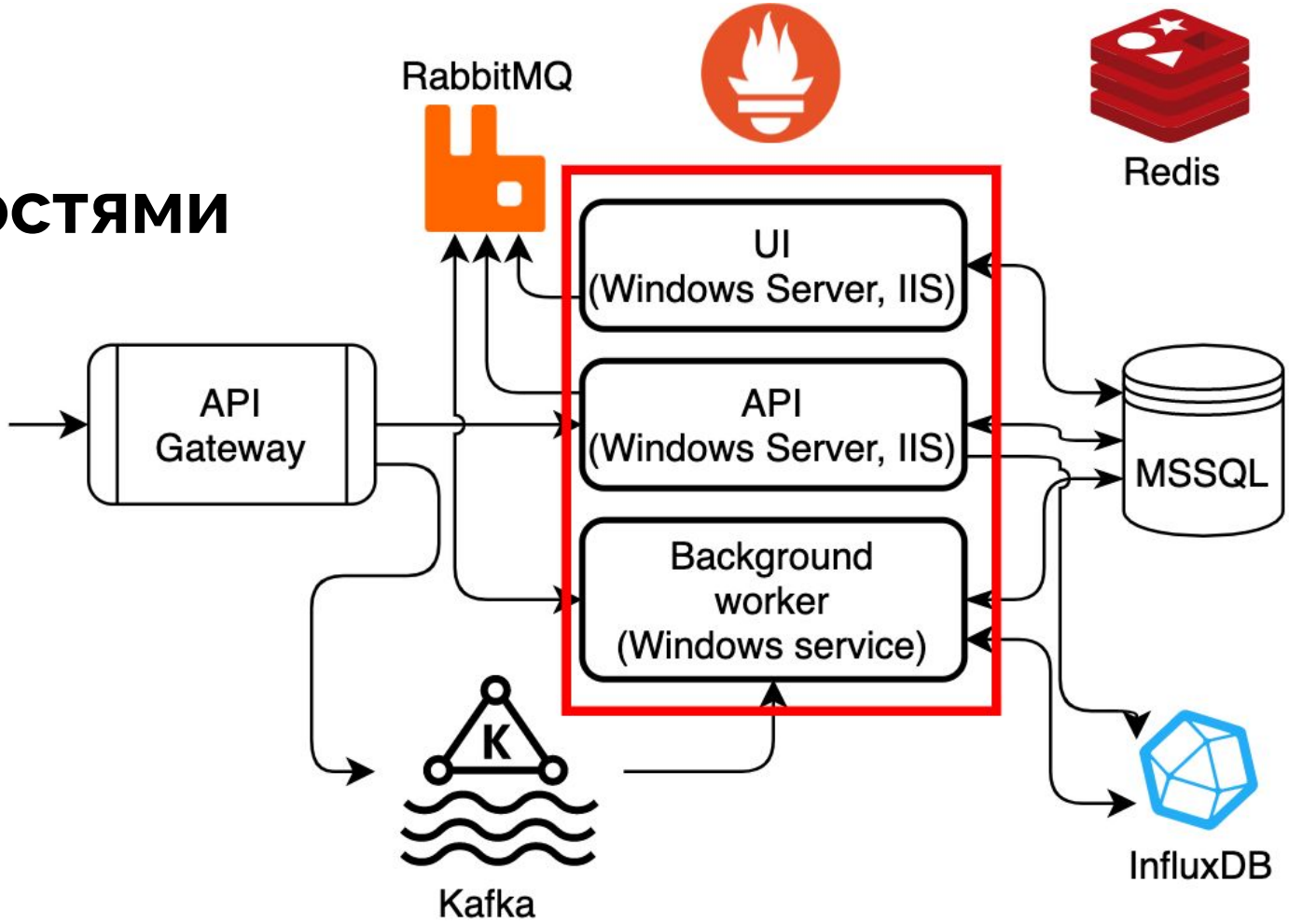
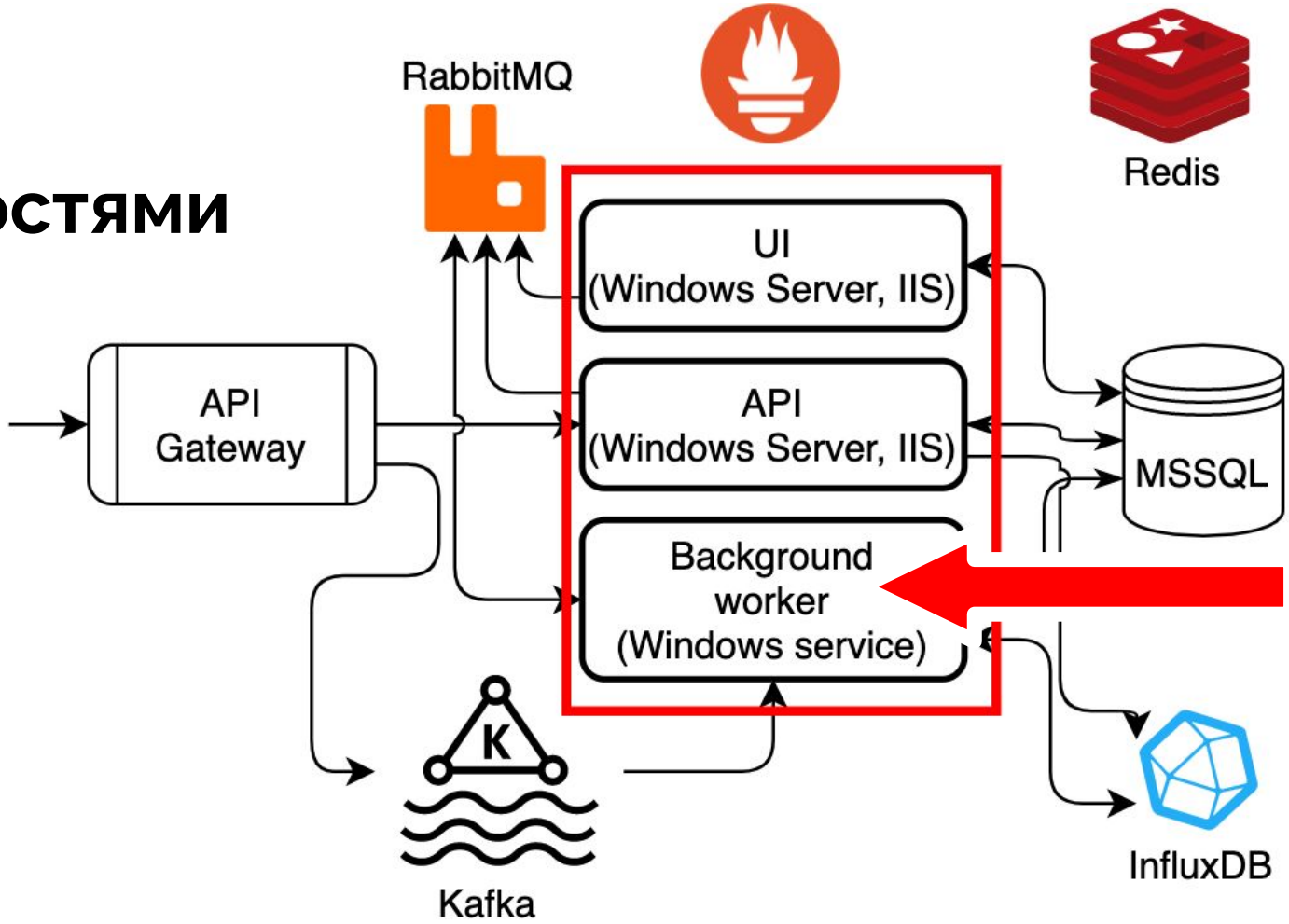


Схема с зависимостями



Что болело на проде

- Плохая утилизация ресурсов
  — Где-то простаивает
  — … а где-то не хватает
- Нужен человек-кубернетес
- Нет запаса на сезонный скачок нагрузки
- Человеческий фактор в настройке

Что болело у разработки

Сложно внедрять
новые инструменты
Например, Прометеус внедрялся несколько раз

Нельзя использовать
новые возможности языка
— Records и Async streams
— Nullable references
— Реализации интерфейсов по умолчанию (вообще никак)
— Оф. последняя версия языка для .NET Framework 4.8 - C# 7.3

Жалобы бизнеса

- Нарушается SLA при деплое
- Сложно прогнозировать закупки
- Хочется ускорить разработку
  — В том числе упростить найм

Подготовка и планирование

Как выбирали облако?

Azure/AWS/etc

Отпали по 152-ФЗ

Как выбирали облако?

Azure/AWS/etc

Отпали по 152-ФЗ

Yandex.Облако

- Свой технический стэк
(и можно влиять на роадмеп)
- Активно развивают свой Terraform-провайдер
- Интерконнект с нашим ЦОДом
- Были готовы к обсуждению SLA

Интерконнект

- **Две физических оптики (ЦОД – Y.C)**
  Чтобы одного лихого экскаватора не хватило
- **Есть фоллбэк и регулярные учения**
- **Ходим по внутренним адресам без интернета и VPN**
  — Экономим трафик
  — Выше производительность

Советы по планированию переезда

1. **Определите проблемы, которые хотите решить прежде всего**
   — Экономия на ресурсах
   — Экономия на обслуживании
   — Скорость разработки

2. **Спланируйте очередность переноса частей системы в облако**
   — И подумайте, как всё будет работать в промежуточных состояниях

3. **Проведите нагрузочные тесты**
   — Пригодится для планирования бюджета

Как начать работу?

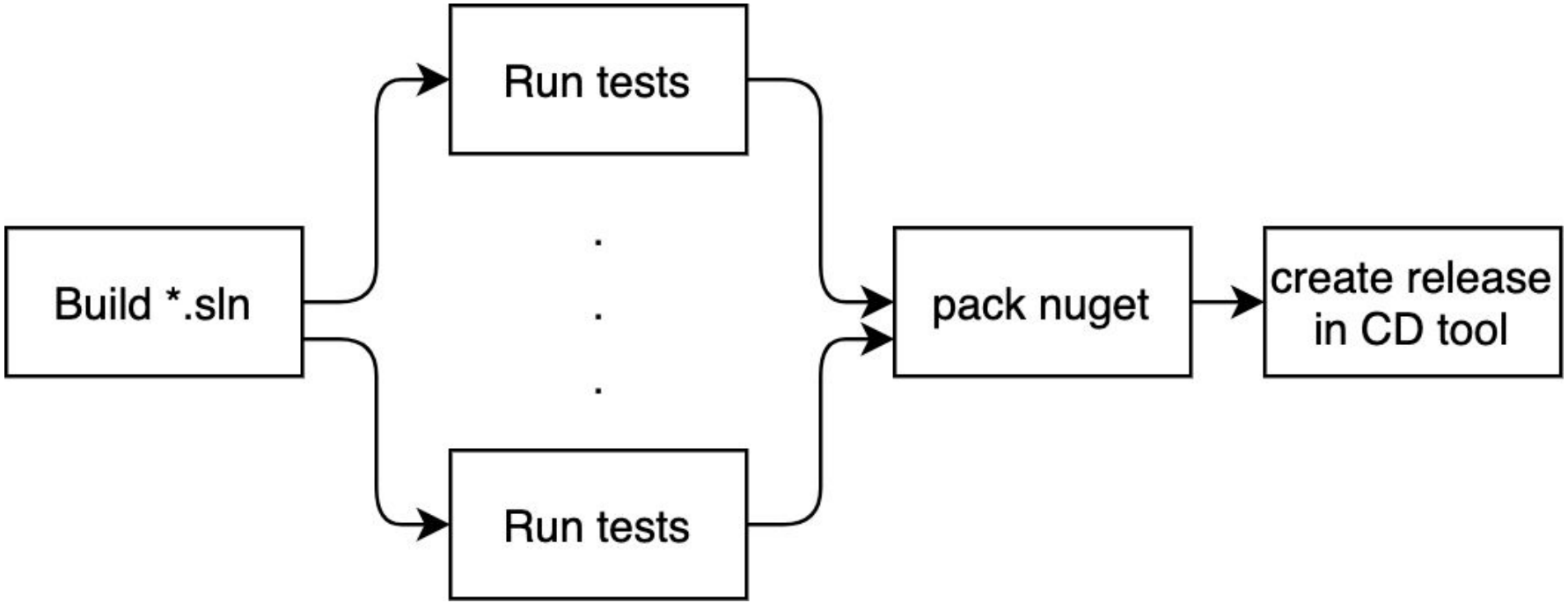
Источники оптимизма

- **~80% кодовой базы – .NET Standard 2.0**
- **Проверили наш фреймворк в облаке на других сервисах**
  - Планировщик задач
  - Мониторинг

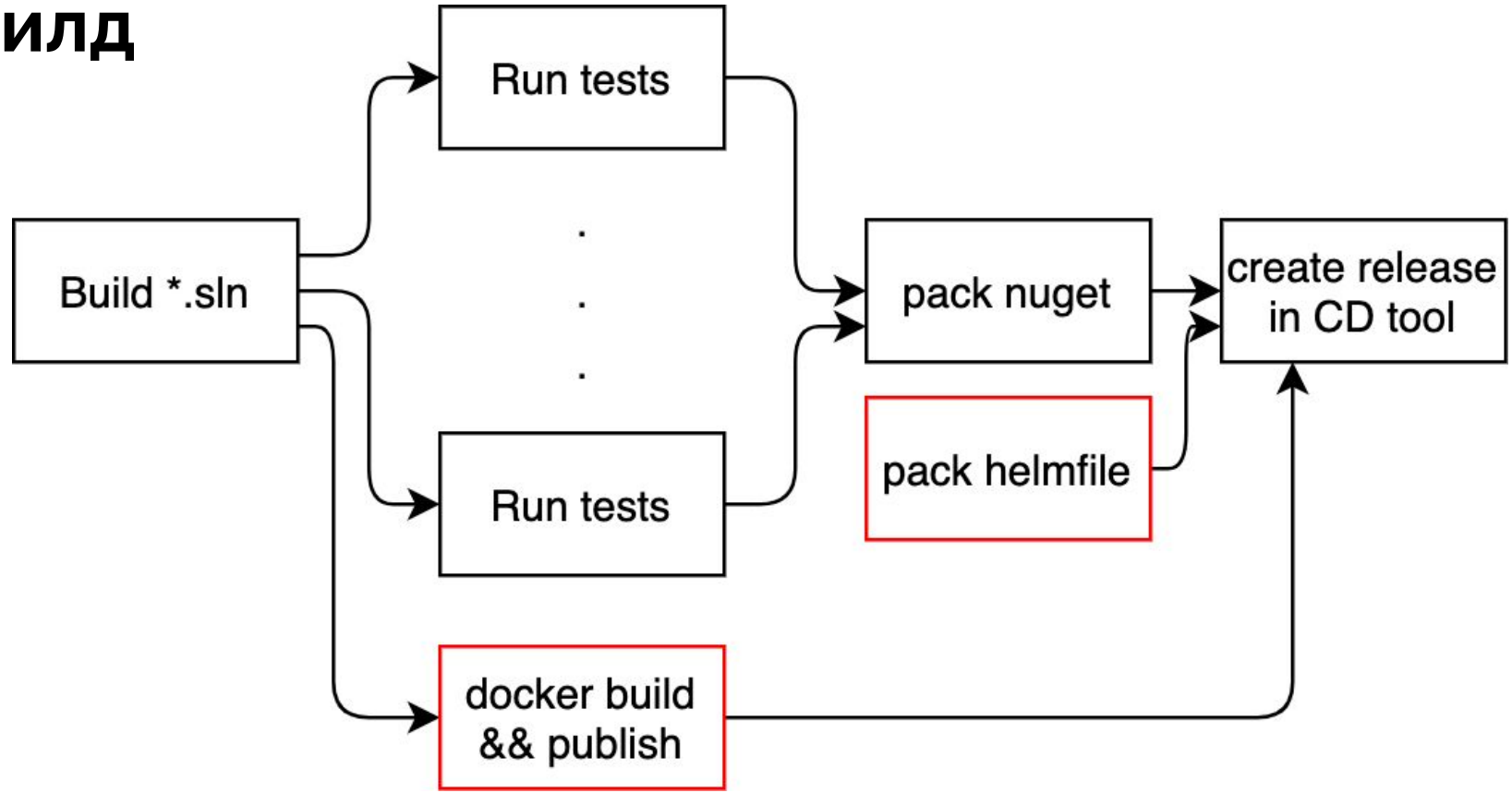
Декомпозировать

- Разделить нагрузку внутри приложения
- Запускать параллельно на **старой** и **новой** инфраструктуре
  — Общий пайплайн
- Майлстоун – очередной срез функциональности
  работает только в облаке

Билд


Run tests
Build *.sln
.
.
.
pack nuget
create release in CD tool
Run tests

Билд


Run tests
Build *.sln
pack nuget
create release in CD tool
Run tests
pack helmfile
docker build && publish

Деплой

Используем helm ***...но***

для описания
деплойментов

Деплой

Используем helm для описания деплойментов

...но

применяем kubectl apply 🤷

- Измеряли на 300 деплойментах
- **40 мин** создание релиза через helm **vs** **30 с** на шаблонизацию и применение всех манифестов
- Helm — для одного релиза в пайплайне

Советы по старту переноса

- Исследуйте возможность внедрения .NET Standard

  Все изменения сразу верифицируются работающим продуктом - меньше рискованных изменений за раз

- Начните с сервиса поменьше и набейте шишки на нём

  Чтобы приобрести экспертизу и проверить жизнью ваши внутренние компоненты

- Инвестируйте в пайплайн

  Чтобы была возможность быстро откатывать неудачные изменения

- Разбейте работу на минимально возможные куски

- Определитесь, как будете определять успешность изменения

3 Мониторинг

Кластер

- Prometheus operator
- Kubernetes-mixin
- Встроенный мониторинг Яндекс.Облака

  Но все равно настойчиво просим все нужное экспортировать в Прометеус

InfluxDB

- Отличия от Prometheus
  — Push vs pull
  — Event logging

- Данные используются из кода приложения
  Например, на их основе работает троттлинг

Инструментирование кода

Prometheus-net

Базовые метрики рантайма

Инструментирование кода

Prometheus-net

Базовые метрики рантайма

Mindbox.DiagnosticContext

- Доступна на гитхабе - https://github.com/mindbox-moscow/Mindbox.DiagnosticContext
- Собирает только то, что явно размечено
- Поддерживает вложенные метрики (и гарантирует, что не возникнут неучтенные фрагменты кода в размеченном участке)

Метрики приложения

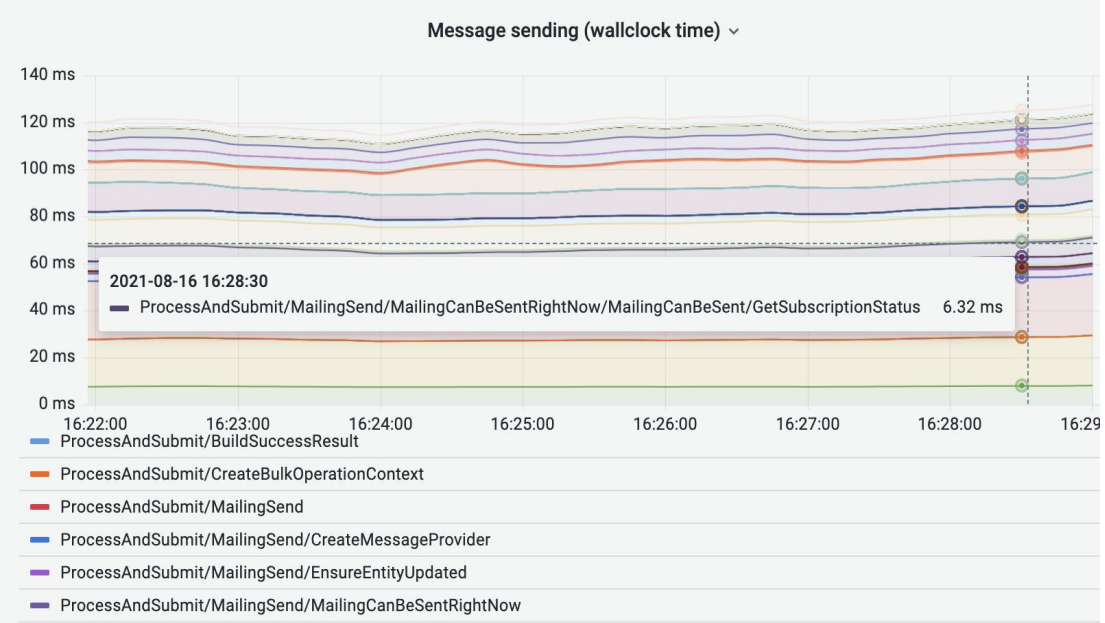
Diagnostic context – счетчики

```
using var dc = diagnosticContextFactory.CreateDiagnosticContext("SampleTransaction");
dc.Increment("sample_counter");
```

Метрики приложения

Время выполнения

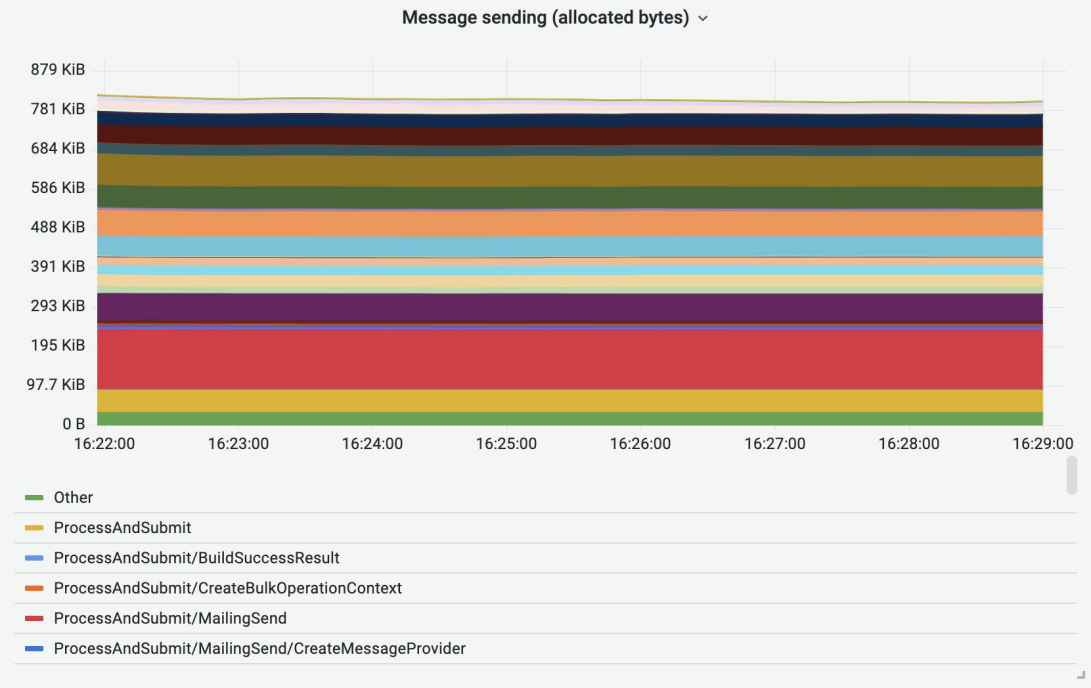
```
using (dc.Measure("ProcessMessage"))
{
  ProcessMessage();
}
```



Метрики приложения

Аллокации

```
using (dc.Measure("ProcessMessage"))
{
  ProcessMessage();
}
```



dotnet-monitor

- **Подключен как сайдкар-контейнер ко всем подам**
- **Автоматизированный ранбук**
- **Автодампы**
  - Экспериментально подобранные лимиты
    - Ограничен размер управляемого хипа,
      чтобы OOM падал до достижения лимитов контейнера
    - liveness-пробы достаточно мягкие, чтобы успевать снять дамп
  - Вызов по http в случае OOM

Советы по мониторингу

- **Поддерживайте прозрачность для остальной разработки**
  — Демо + Ранбуки
- **Дайте возможность быстро делать метрики, которых не хватает**
- **Не вводите в общую эскалацию алерты без ~~объявления войны~~ проверки и инструкций**
  Мы мариновали их в канале slack-а до готовности

Препятствия и проблемы

Проблемы самого кластера

- Дошли до размера 83 worker ноды, ~2000 ядер, ~3700-4000 подов
- Переполнение Prometheus
- Отказы metrics-server
  - Перестает работать HPA
- Нехватка ресурсов для мастер-нод
  - И вы можете даже не заметить этого
- Закончились виртуалки
  - Именно того типа, который был нужен нам, но всё же

Авторизация в БД

Windows Authentication – ❌

Kerberos кажется ненативным
и хрупким в поддержке

Авторизация в БД

Windows Authentication – ❌

Kerberos кажется ненативным и хрупким в поддержке

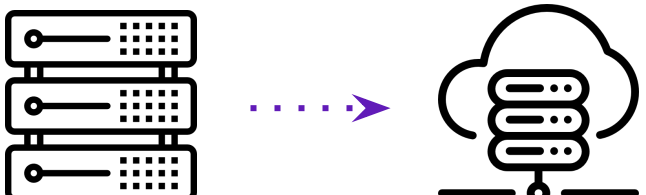
SQL Authentication

- **Server**
  - не дружит с Always On availability group
- **Contained** – ✅
  - требует настройки БД
  - не даёт доступ к ресурсам на сервере

Как мигрировать?

С болью — нужна эксклюзивная блокировка на базу целиком:

1. Разобрать AG
2. Перевести в Single-user Mode
   Будьте внимательны и не потеряйте нужный коннекшен
3. Переключить Containment type в Partial.
4. Собрать AG обратно
   Возможно, с нуля

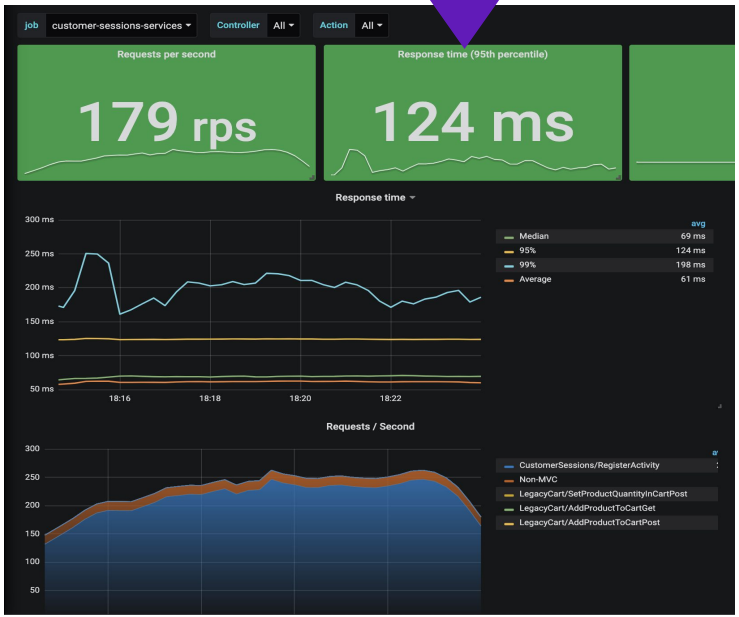


RestSharp (WebRequest) vs HttpClient

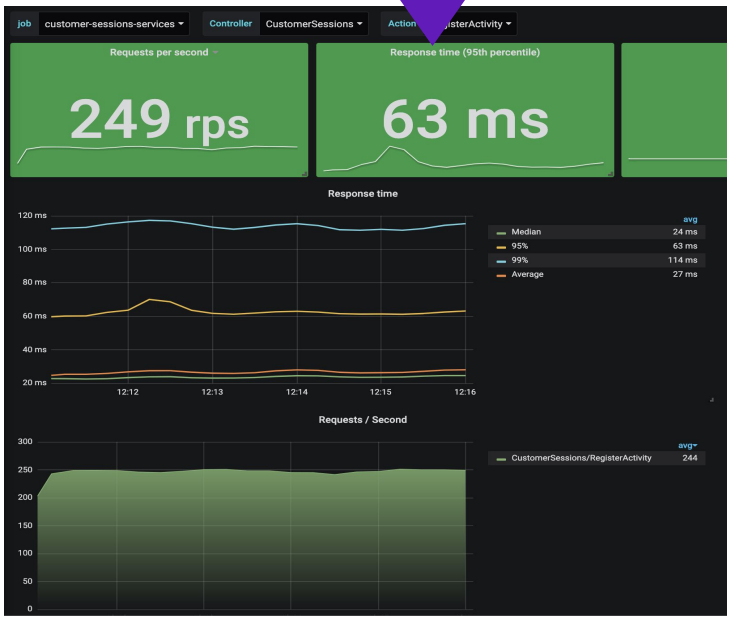
- Поступила жалоба на скорость рассылки
- Рассылка на ~400k получателей должна уходить за 10 мин
  Особенно на безлимитных ресурсах облака
- Почти не расходуется CPU
- Не срабатывает HPA
- 🤔

RestSharp vs HttpClient

RestSharp



HttpClient



Проблемы в кодовой базе

CultureInfo

- По умолчанию в ASP.NET Core – InvariantCulture
- Сломали внутреннюю интеграцию
- Решается настройкой образа

Проблемы в кодовой базе

Thread.Abort()

- Вообще нельзя использовать
- Не поддерживается в .NET Core
- Но почему-то у нас был 🤦
- Заменили на HealthCheck с управляемым состоянием

Проблемы в кодовой базе

BinaryFormatter

- Используется для кэширования
- Сериализованные значения отличаются между рантаймами
- Добавили суффикс с рантаймом в ключ
- А еще блокирует переход на .NET 6

Советы по борьбе с последствиями

- Подготовьте мониторинг

  Чтобы не обнаруживать проблемы из жалоб в поддержку

- Заложите ресурсы на правку внезапных багов

  Они точно будут

Опыт с прода

Управление ресурсами

Стремились ужать поды до минимума

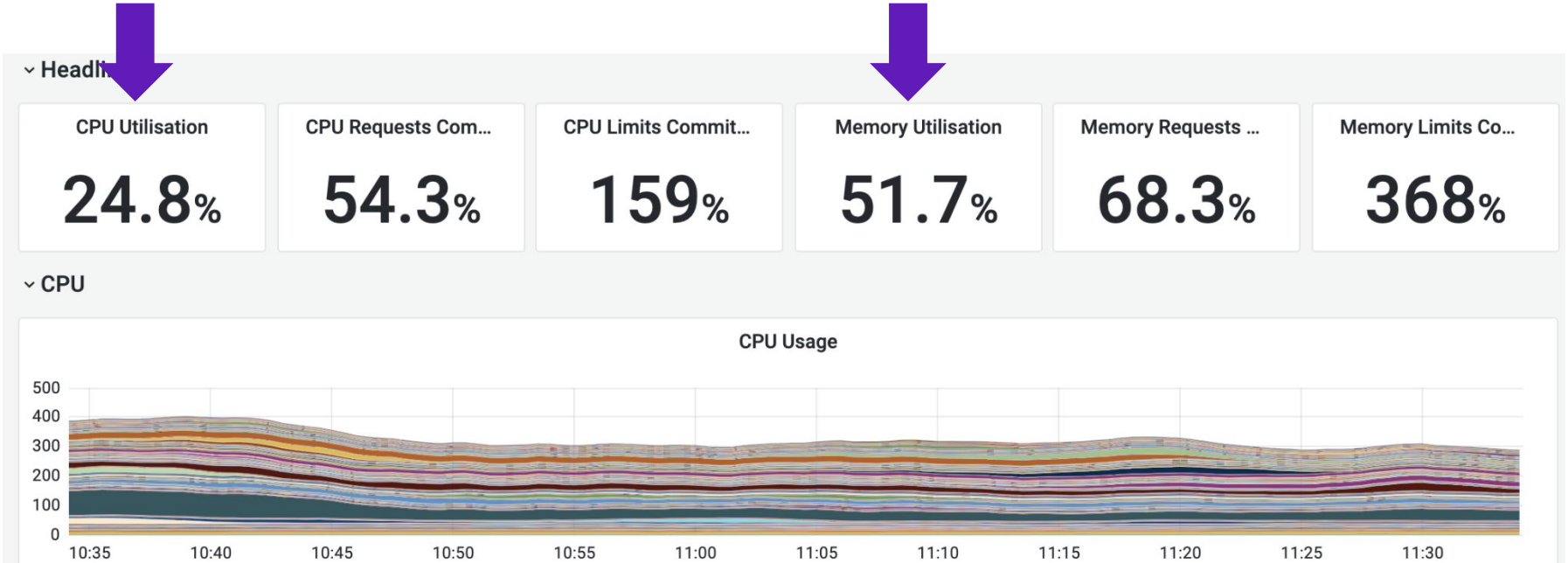
- ~20 ядер при первом запуске
- Ухудшается утилизация
- Есть риск перегрузить ноду
- Дошли до ~1 ядра на под

Управление ресурсами

Троттлинг мешает

- Приложение не может делать свою работу
- Падают таймауты на вызов внешних сервисов
- Пришлось убрать лимиты
- Ограничение потребления естественное и контролируется кодом

Проблемы с утилизацией



Но все равно стало лучше!

Сервис Рассылки

	Win-сервера	Kubernetes
CPU (ядра)	694	от 411 до 555
Память, Gb	1973	от 1140 до 1760

Как повысить утилизацию?

- Разделили клиентов по классам обслуживания
- Выделяем ресурсы в зависимости от размера базы
- Самые маленькие – живут в одном поде


DevCore / DevCore CDP to Kubernetes Migration
Last 2 days
Cores allocated by CDP k8s in YC
6500 ядер → 3000 ядер
Last * Max Mean
Allocated 3202 9434 5248
Requested 2218 8851 4161
Usage 934 1370 877
10000
9500
9000
8500
8000
7500
7000
6500
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
10/04 21:00
10/05 00:00
10/05 03:00
10/05 06:00
10/05 09:00
10/05 12:00
10/05 15:00
10/05 18:00
10/05 21:00
10/06 00:00
10/06 03:00
10/06 06:00
10/06 09:00
10/06 12:00
10/06 15:00
10/06 18:00

Влияние на остальную инфраструктуру

SQL — взрывной рост соединений

Влияние на остальную инфраструктуру

- **Influx**

  InfluxException: partial write: max-series-per-database limit exceeded

  - Пришлось переписать большую часть метрик

- **Kafka**
  - Параллелизм ограничен количеством партиций в топике
  - ...но нам хватило

Советы по управлению ресурсами

1. **Составьте карту системы со всеми сторонними компонентами**
2. **Сделайте прогноз, как перенос в облако на них повлияет**
3. **Проведите нагрузочные тесты**
   И отмасштабируйте заранее
4. **Не занижайте ресурсы приложения до тестов**
   — Можно получить OOMы
   — И троттлинг приложения по CPU

Что **не удалось** победить в облаке

Производительность CPU

- **На наших нагрузках разница** – 1.5-2 раза
  Измеряли нагрузочными тестами до миграции
- **Перед миграцией:** 2000 ядер
- **После**: 3000 ядер в облаке
  +500 ядер в локальном кластере

Производительность БД

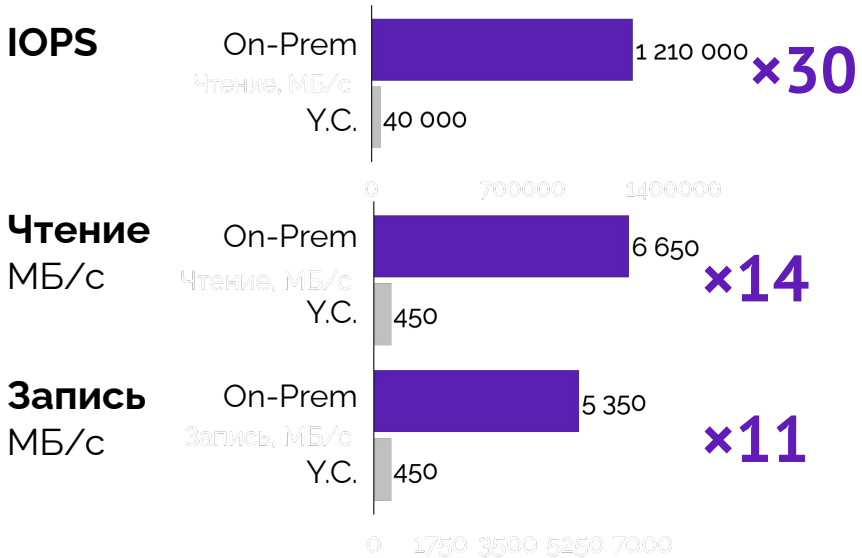
Ненулевой пинг

Каждый вызов
в транзакции
прибавляет

1-3 мс

Производительность БД

Медленные диски в облаке



Решение – локальный кластер

- С проверенным и работающим приложением не так страшно
- Используем сервис Флант
- На примере рассылок:
  — Яндекс - 90к/мин
  — On-premise - 160к/мин
- Для самых требовательных к производительности клиентов

Выводы и recap

Recap

- **Минимизируйте вносимые за раз изменения**
  Например, выделяйте какой-то узкий независимый фрагмент приложения и переносите его
- **Проинвестируйте в возможность быстрого отката неудачных экспериментов**
- **Определитесь с метриками**
  Например, потребление железа, скорость работы, etc
- **Заложите время на починки внезапных багов и проседаний производительности**
  У нас роадмеп съехал на 25% от плана
- **Настраивайте поды минимально возможного размера**
  с как можно менее волатильной нагрузкой
- **Подготовьте другие элементы инфраструктуры к росту нагрузки и обеспечьте их мониторинг**

Ростислав Листеренко

Архитектор, www.mindbox.ru

listerenko@mindbox.ru
fb.com/rostislav.listerenko

mindbox

Спасибо!