Быстрый запуск микросервиса

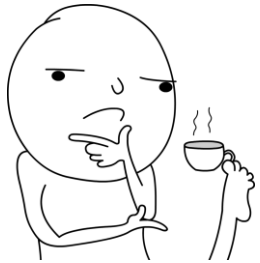
Познакомимся?

Муляр Николай
@proxeter

Разработчик на Go
Организатор Go SPB

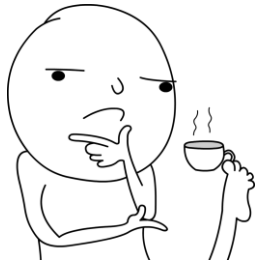
Senior Go Developer AirPush

Зачем мы собрались?



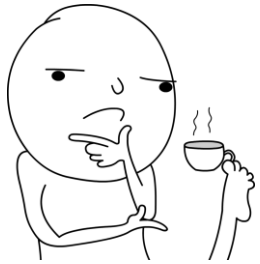
1. Поделиться с Вами опытом создания шаблона микросервиса

Зачем мы собрались?



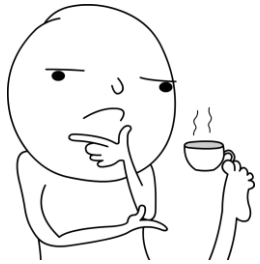
1. Поделиться с Вами опытом создания шаблона микросервиса
2. Рассказать о проблемах, которые привели нас к нему

Зачем мы собрались?



1. Поделиться с Вами опытом создания шаблона микросервиса
2. Рассказать о проблемах, которые привели нас к нему
3. Уберечь Вас от повторения ошибок

Зачем мы собрались?



1. Поделиться с Вами опытом создания шаблона микросервиса
2. Рассказать о проблемах, которые привели нас к нему
3. Уберечь Вас от повторения ошибок
4. Представить разработанный шаблон

Микросервис. Что за зверь?

1. Выполняет одну задачу

Микросервис. Что за зверь?

1. Выполняет одну задачу
2. Содержит минимальную кодовую базу

Микросервис. Что за зверь?

1. Выполняет одну задачу
2. Содержит минимальную кодовую базу
3. Прост для тестирования

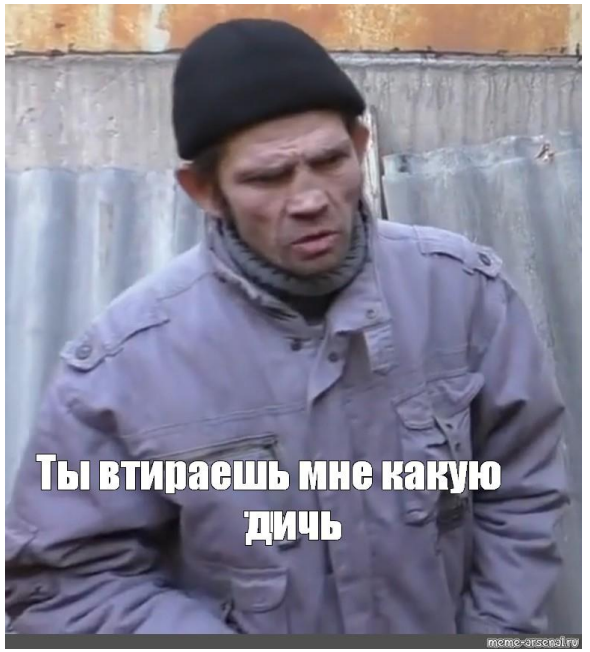
Микросервис. Что за зверь?

1. Выполняет одну задачу
2. Содержит минимальную кодовую базу
3. Прост для тестирования
4. Быстро и легко развертывается

Микросервис. Что за зверь?

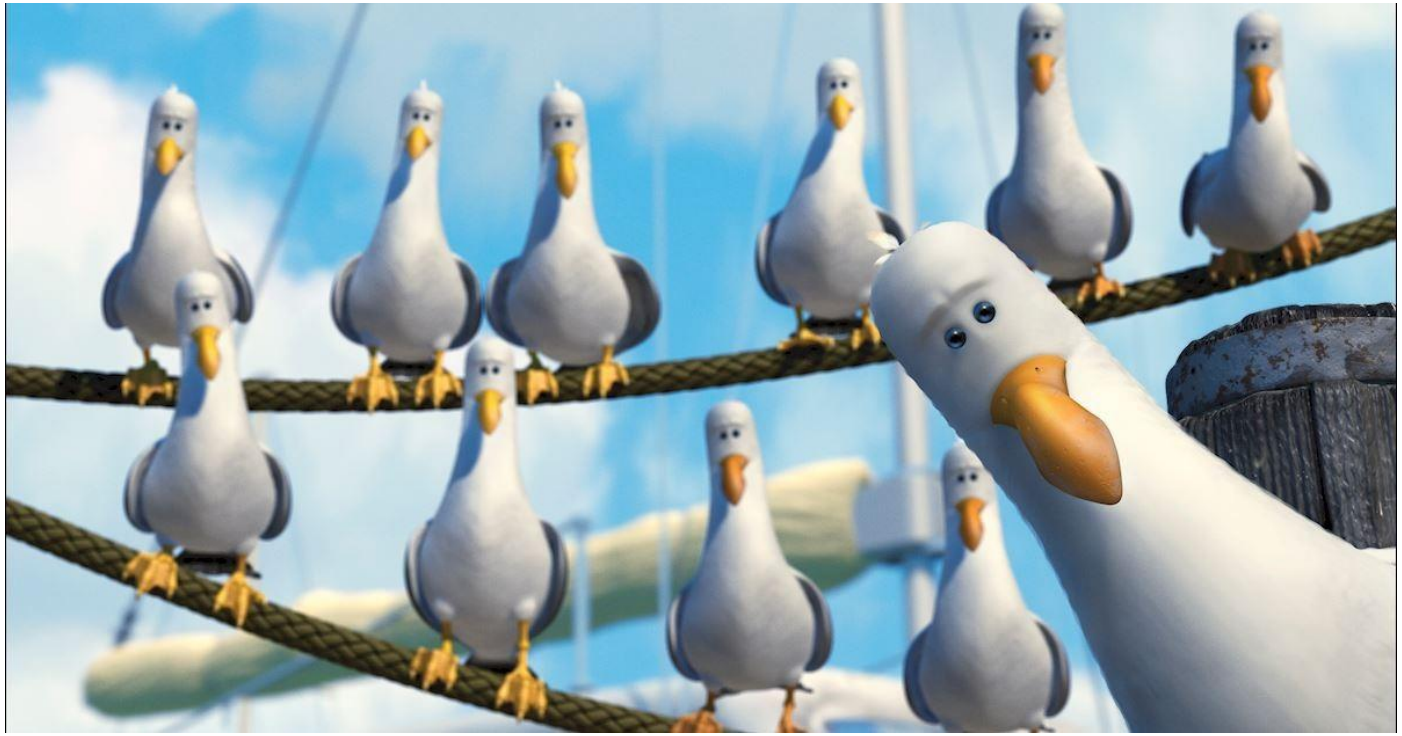
1. Выполняет одну задачу
2. Содержит минимальную кодовую базу
3. Прост для тестирования
4. Быстро и легко развертывается
5. "Ленивый программист" рад работе с таким сервисом

Микросервис. Что за зверь?



1. Выполняет одну задачу
2. Содержит минимальную кодовую базу
3. Прост для тестирования
4. Быстро и легко развертывается
5. "Ленивый программист" рад работе с таким сервисом

Давайте посмотрим



proxeter@proxeter ~/Documents/go/github.com/proxeter ✔ 1005 23:30:03

Важная ремарка

- **Все**, о чем будет рассказано дальше, реализовано в шаблоне

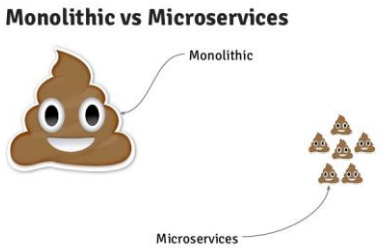
Важная ремарка

- **Все**, о чем будет рассказано дальше, реализовано в шаблоне
- Это можно пощупать и запустить у себя

Важная ремарка

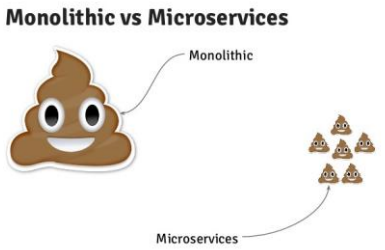
- **Все**, о чем будет рассказано дальше, реализовано в шаблоне
- Это можно пощупать и запустить у себя
- Шаблон используется для Go приложения, но основные идеи **применимы** для **любого** языка

Из чего состоит микросервис?



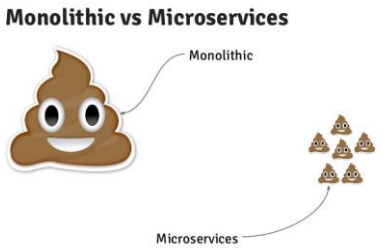
1. Бизнес-логика

Из чего состоит микросервис?



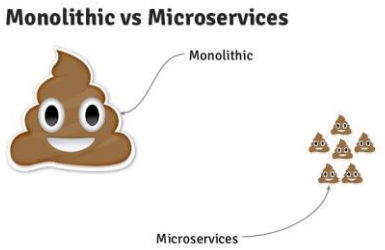
1. Бизнес-логика
2. Окружение для локальной отладки

Из чего состоит микросервис?



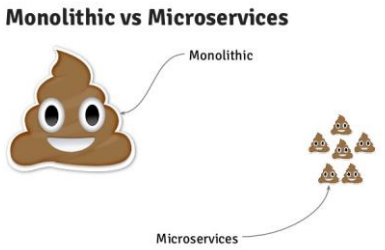
1. Бизнес-логика
2. Окружение для локальной отладки
3. Сборка итогового артефакта

Из чего состоит микросервис?



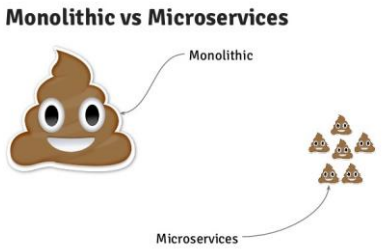
1. Бизнес-логика
2. Окружение для локальной отладки
3. Сборка итогового артефакта
4. Инструменты для деплоя

Из чего состоит микросервис?



1. Бизнес-логика
2. Окружение для локальной отладки
3. Сборка итогового артефакта
4. Инструменты для деплоя
5. Инструменты для мониторинга состояния

Из чего состоит микросервис?



1. Бизнес-логика
2. Окружение для локальной отладки
3. Сборка итогового артефакта
4. Инструменты для деплоя
5. Инструменты для мониторинга состояния
6. Инструменты для мониторинга производительности

Окружение для локальной отладки. Проблемы

1. Не каждый проект имел локальное окружение

Окружение для локальной отладки. Проблемы

1. Не каждый проект имел локальное окружение
2. А если имел, то документация по запуску локального окружения занимала несколько страниц

Окружение для локальной отладки. Проблемы

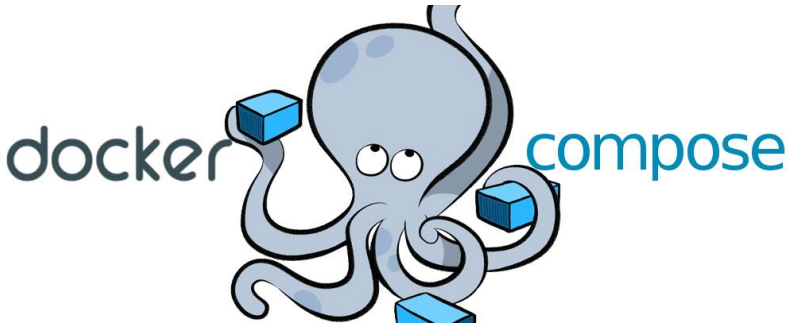
1. Не каждый проект имел локальное окружение
2. А если имел, то документация по запуску локального окружения занимала несколько страниц
3. Чаще всего оно не работало

Окружение для локальной отладки. Решаем проблемы

1. Docker

Окружение для локальной отладки. Решаем проблемы

1. Docker
2. Docker-compose



Окружение для локальной отладки. Решаем проблемы

1. Несколько окружений

Окружение для локальной отладки. Решаем проблемы

1. Несколько окружений:
   - Обвязка -> все зависимые сервисы, кроме самого приложения (binding)

Окружение для локальной отладки. Решаем проблемы

docker-compose -f deployments/docker/docker-compose.yml

up –d **consul mysql hazelcast kafka**

Окружение для локальной отладки. Решаем проблемы

1. Несколько окружений:
   - Обвязка -> все зависимые сервисы, кроме самого приложения (binding)
   - Полное -> обвязка с самим приложением (full)

Окружение для локальной отладки. Решаем проблемы

docker-compose -f deployments/docker/docker-compose.yml up –d consul mysql hazelcast kafka

docker-compose -f deployments/docker/docker-compose.yml

up -d --build **${NAME}**

Окружение для локальной отладки. Однообразность

1. Все, что может быть использовано другими людьми - в makefile, bat, powershell, ...

Окружение для локальной отладки. Однообразность

1. Все, что может быть использовано другими людьми - в makefile, bat, powershell, ...
2. Сначала цель, затем поведение

docker_env_start_binding

tar_build: build tar

Окружение для локальной отладки. Однообразность

1. Все, что может быть использовано другими людьми - в makefile, bat, powershell, ...
2. Сначала цель, затем поведение
3. Выводить отладочную информацию

```
clean:
@echo "> Cleaning build cache and binaries"
```

Окружение для локальной отладки. Однообразность

1. Все, что может быть использовано другими людьми - в makefile, bat, powershell, ...
2. Сначала цель, затем поведение
3. Выводить отладочную информацию
4. Использовать "условные" цели

Окружение для локальной отладки. Однообразность

```
docker_elk_up:
@read -p "Run 'ELK' [y/n]: " ans; \
if [ $$ans = y ]; then \
docker-compose -f deployments/docker/docker-compose.yml
up –d elasticsearch kibana apm-server; \
fi
```

Окружение для локальной отладки. Еще важного

1. Не запускать ресурсоемкие зависимости, если они не нужны

Окружение для локальной отладки. Еще важного

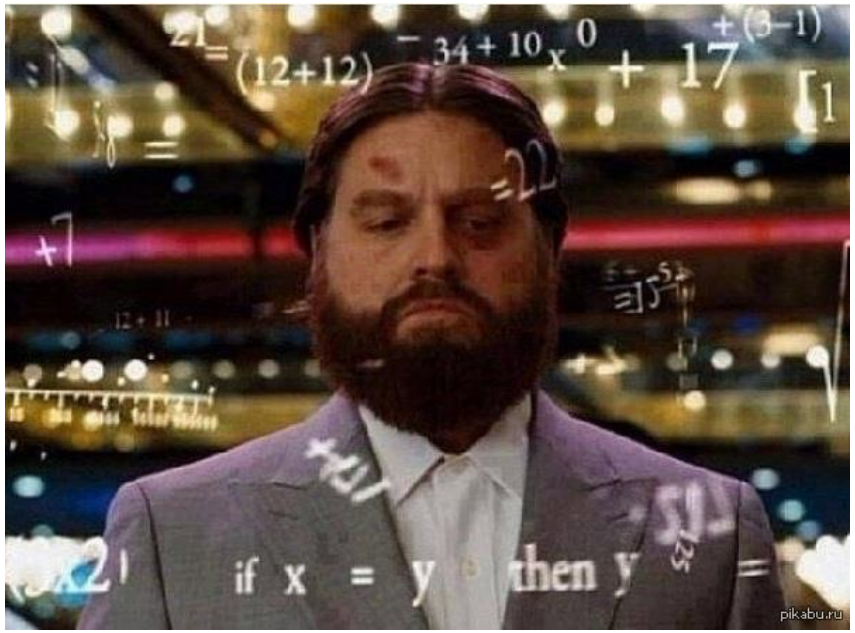
1. Не запускать ресурсоемкие зависимости, если они не нужны
2. Приложение должно взаимодействовать с окружением, если оно запущено

Окружение для локальной отладки. Еще важного

1. Не запускать ресурсоемкие зависимости, если они не нужны
2. Приложение должно взаимодействовать с окружением, если оно запущено
3. Иметь цель для выключения ресурсоемких зависимостей

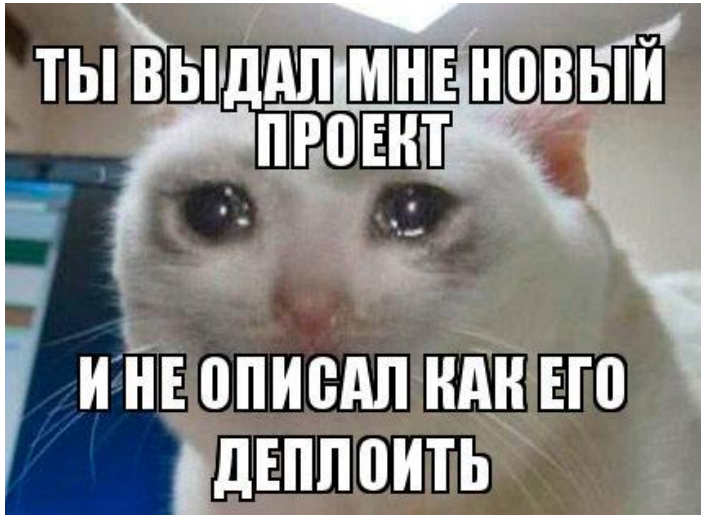
Сборка итогового артефакта. Проблемы

1. Каждый проект собирается "как может"



Сборка итогового артефакта. Проблемы

1. Каждый проект собирается "как может"
2. Деплой проекта затягивается командой DevOps из-за необходимости писать скрипты под каждый проект



Сборка итогового артефакта. Проблемы

1. Каждый проект собирается "как может"
2. Деплой проекта затягивается командой DevOps из-за необходимости писать скрипты под каждый проект
3. Но это работает в отличии от локального окружения

Сборка итогового артефакта. Исправляем проблемы

1. Использовать git tag для версионирования

Сборка итогового артефакта. Исправляем проблемы

1. Использовать git tag для версионирования
2. Единое имя "таргета" для всех проектов

Помните **tar_build**?

Сборка итогового артефакта. Исправляем проблемы

```
tar:
rm -rf tars build
mkdir -p build build/configs tars
cp bin/${NAME} build/${NAME}
cp init/systemd.service build/${NAME}.service
sed -i -e 's/{{ NAME }}/${NAME}/g' build/${NAME}.service
sed -i -e 's/{{ VERSION }}/${VERSION}/g' build/${NAME}.service
sed -i -e 's/{{ LOG_LEVEL }}/${LOG_LEVEL}/g' build/${NAME}.service
sed -i -e 's/{{ ENVIRONMENT }}/${ENVIRONMENT}/g' build/${NAME}.service
cp ./configs/${NAME}/${ENVIRONMENT}.yml build/configs/
tar -cvzf tars/${NAME}.tar.gz -C build/ .
rm -rf bin build
```

Сборка итогового артефакта. Исправляем проблемы

```
tar:
rm -rf tars build
mkdir -p build build/configs tars
cp bin/${NAME} build/${NAME}
cp init/systemd.service build/${NAME}.service
sed -i -e 's/{{ NAME }}/${NAME}/g' build/${NAME}.service
sed -i -e 's/{{ VERSION }}/${VERSION}/g' build/${NAME}.service
sed -i -e 's/{{ LOG_LEVEL }}/${LOG_LEVEL}/g' build/${NAME}.service
sed -i -e 's/{{ ENVIRONMENT }}/${ENVIRONMENT}/g' build/${NAME}.service
cp ./configs/${NAME}/${ENVIRONMENT}.yml build/configs/
tar -cvzf tars/${NAME}.tar.gz -C build/ .
rm -rf bin build
```

Сборка итогового артефакта. Исправляем проблемы

```
tar:
rm -rf tars build
mkdir -p build build/configs tars
cp bin/${NAME} build/${NAME}
cp init/systemd.service build/${NAME}.service
sed -i -e 's/{{ NAME }}/${NAME}/g' build/${NAME}.service
sed -i -e 's/{{ VERSION }}/${VERSION}/g' build/${NAME}.service
sed -i -e 's/{{ LOG_LEVEL }}/${LOG_LEVEL}/g' build/${NAME}.service
sed -i -e 's/{{ ENVIRONMENT }}/${ENVIRONMENT}/g' build/${NAME}.service
cp ./configs/${NAME}/${ENVIRONMENT}.yml build/configs/
tar -cvzf tars/${NAME}.tar.gz -C build/ .
rm -rf bin build
```

Сборка итогового артефакта. Последнее

1. Использовать git tag для версионирования
2. Единое имя "таргета" для всех проектов
3. **Сохранять** собранный артефакт, чтобы можно было **быстро откатиться**

Деплой. Проблемы

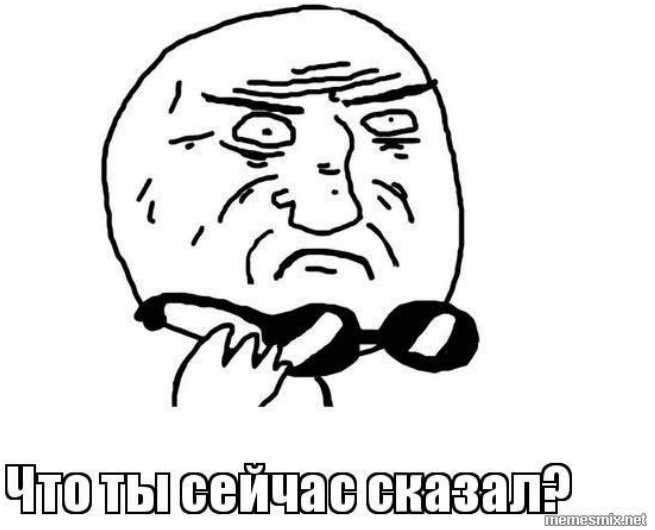
1. DevOps **всегда** перегружены

Деплой. Проблемы

1. DevOps **всегда** перегружены
2. Окружений может быть несколько

Деплой. Проблемы

1. DevOps **всегда** перегружены
2. Окружений может быть несколько
3. Иногда нужно отлаживать на **проде**



Деплой. Решаем проблемы

1. Использовать шаблон скриптов деплоя:

Мы что, зря создавали **таргеты**?

Деплой. Решаем проблемы

1. Использовать шаблон скриптов деплоя:
    - GitLab CI

Деплой. Решаем проблемы

1. Использовать шаблон скриптов деплоя:
    - GitLab CI
    - Jenkinsfile

Деплой. Решаем проблемы

1. Использовать шаблон скриптов деплоя:
    - GitLab CI
    - Jenkinsfile

```
stages:
  - build
  - lint
  - deploy
  - docker
```

Деплой. Решаем проблемы

1. Использовать шаблон скриптов деплоя:
    - GitLab CI
    - Jenkinsfile

```
stages:
  - build
  - lint
  - deploy
  - docker
```

Деплой. Решаем проблемы

1. Использовать шаблон скриптов деплоя:
    - GitLab CI
    - Jenkinsfile

```
stages:
  - build
  - lint
  - deploy
  - docker
```

Деплой. Решаем проблемы

1. Использовать шаблон скриптов деплоя
2. Деплой в различные окружения решен шаблоном

Деплой. Решаем проблемы

1. Использовать шаблон скриптов деплоя
2. Деплой в различные окружения решен шаблоном
   - Но мы еще вернемся к окружениям, не переживайте

Деплой. Решаем проблемы

1. Использовать шаблон скриптов деплоя
2. Деплой в различные окружения решен шаблоном
   - Но мы еще вернемся к окружениям, не переживайте
3. Закладывать работу в прод и дебаг режиме при сборке

ExecStart=/opt/{{ NAME }}/{{ NAME }} -e {{ ENVIRONMENT }} -v {{ VERSION }}

Деплой. Решаем проблемы

1. Использовать шаблон скриптов деплоя
2. Деплой в различные окружения решен шаблоном
   - Но мы еще вернемся к окружениям, не переживайте
3. Закладывать работу в прод и дебаг режиме при сборке

#**ExecStart**=/usr/bin/**dlv** --listen=:2345 --headless=true --api-version=2

exec /opt/{{ NAME }}/{{ NAME }} -e {{ ENVIRONMENT }} -v {{ VERSION }}

И снова ремарка

1. Дальше говорим не о **проблемах**

И снова ремарка

1. Дальше говорим не о **проблемах**
2. Говорим о **технологиях** и почему выбраны они

Деплой. Окружения

1. Железная машина

Деплой. Окружения

1. Железная машина
   - Быстродействие

Деплой. Окружения

1. Железная машина
   - Быстродействие
   - Больше DevOps специалистов, предпочитающих этот способ

Деплой. Окружения

1. Железная машина
   - Быстродействие
   - Больше DevOps специалистов, предпочитающих этот способ
   - Между приложением и железом - ядро

Деплой. Окружения

1. Железная машина
   - Быстродействие
   - Больше DevOps специалистов, предпочитающих этот способ
   - Между приложением и железом - ядро
2. Docker

Деплой. Окружения

1. Железная машина
   - Быстродействие
   - Больше DevOps специалистов, предпочитающих этот способ
   - Между приложением и железом - ядро
2. Docker
   - "Готово". Благодаря локальному окружению

Деплой. Окружения

1. Железная машина
   - Быстродействие
   - Больше DevOps специалистов, предпочитающих этот способ
   - Между приложением и железом - ядро
2. Docker
   - "Готово". Благодаря локальному окружению
3. K8S

Деплой. Окружения

1. Железная машина
   - Быстродействие
   - Больше DevOps специалистов, предпочитающих этот способ
   - Между приложением и железом - ядро
2. Docker
   - "Готово". Благодаря локальному окружению
3. K8S
   - Все хотят K8S

Деплой. Окружения

1. Железная машина
   - Быстродействие
   - Больше DevOps специалистов, предпочитающих этот способ
   - Между приложением и железом - ядро
2. Docker
   - "Готово". Благодаря локальному окружению
3. K8S
   - Все хотят K8S
   - Мы **тоже** хотим

Инструменты для мониторинга. Проблемы

1. "В моем коде точно нет ошибок!"

Инструменты для мониторинга. Проблемы

1. "В моем коде точно нет ошибок!"
2. "Да всего одну строчку поменял"

Инструменты для мониторинга. Проблемы

1. "В моем коде точно нет ошибок!"
2. "Да всего одну строчку поменял"
3. "Я закончил писать этот код в 4 ночи"

Инструменты для мониторинга. Проблемы

1. "В моем коде точно нет ошибок!"
2. "Да всего одну строчку поменял"
3. "Я закончил писать этот код в 4 ночи"
4. "Не совсем понимаю как, но оно работает"

Инструменты для мониторинга. Проблемы

1. "В моем коде точно нет ошибок!"
2. "Да всего одну строчку поменял"
3. "Я закончил писать этот код в 4 ночи"
4. "Не совсем понимаю как, но оно работает"
5. "Зуб даю, работает!"

Инструменты для мониторинга. Решаем проблему

1. Логи:

- Собираем: ElasticSearch

Инструменты для мониторинга. Решаем проблему

1. Логи:

- Собираем: ElasticSearch
- Доставляем: Fluentd

Инструменты для мониторинга. Решаем проблему

1. Логи:

- Собираем: ElasticSearch
- Доставляем: Fluentd
- Визуализируем: Kibana

Инструменты для мониторинга. Решаем проблему

1. Логи:

- Собираем: ElasticSearch
- Доставляем: Fluentd
- Визуализируем: Kibana

2. Метрики

- Собираем: Prometheus

Инструменты для мониторинга. Решаем проблему

1. Логи:

- Собираем: ElasticSearch
- Доставляем: Fluentd
- Визуализируем: Kibana

2. Метрики

- Собираем: Prometheus
- Визуализируем: Grafana

Инструменты для мониторинга. Решаем проблему

1. Логи:

- Собираем: ElasticSearch
- Доставляем: Fluentd
- Визуализируем: Kibana

2. Метрики

- Собираем: Prometheus
- Визуализируем: Grafana

3. Трейсинг -> ElasticSearch APM

Контакты

Муляр Николай

@proxeter

Шаблон

github.com/proxeter/go-service-template

Спасибо за внимание!