

# Непрерывное развертывание .Net Core приложений и инфраструктуры в Linux среде

Ветчинкин Кирилл

<https://www.facebook.com/k.vetchinkin>

system-komkon@yandex.ru



О себе

12

Месяцев с  
.net Core на  
Prod

6

Месяцев с  
ОС Linux на  
Prod

4

Проекта на  
.net Core+Linux

# План доклада

- Рассмотрим проблемы разработки и поставки
- Рассмотрим способы их решения
- Поговорим про Windows/Linux
- Практические примеры(тестовая среда, создание, настройка, деплой и т.п.)

# Проблемы разработки и поставки ПО



На моем компьютере это работает





# Единый тестовый стенд



# Ручные операции/инструкции для Ops



Создать и настроить 20 машин



Через неделю будет готово



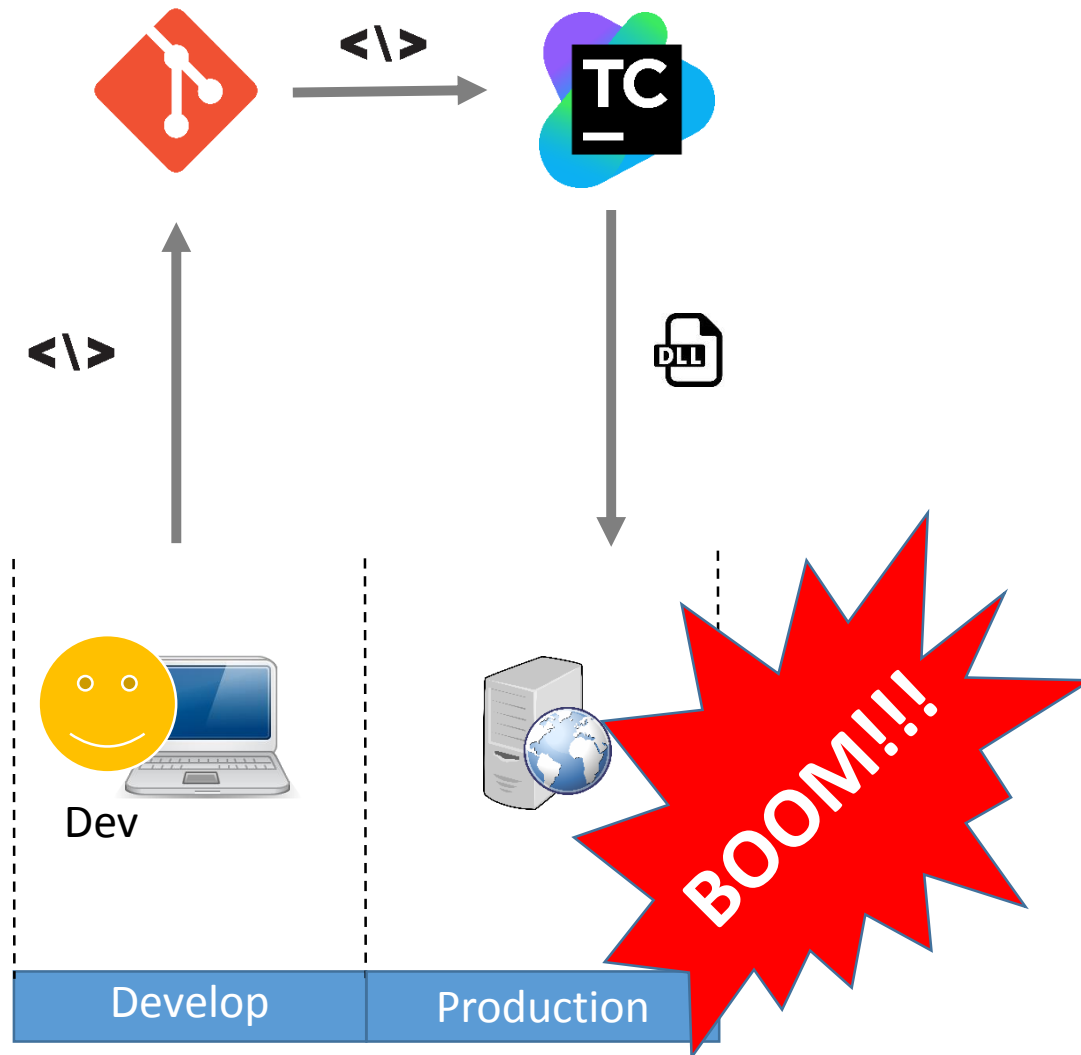
Создать их идентичными и не ошибиться



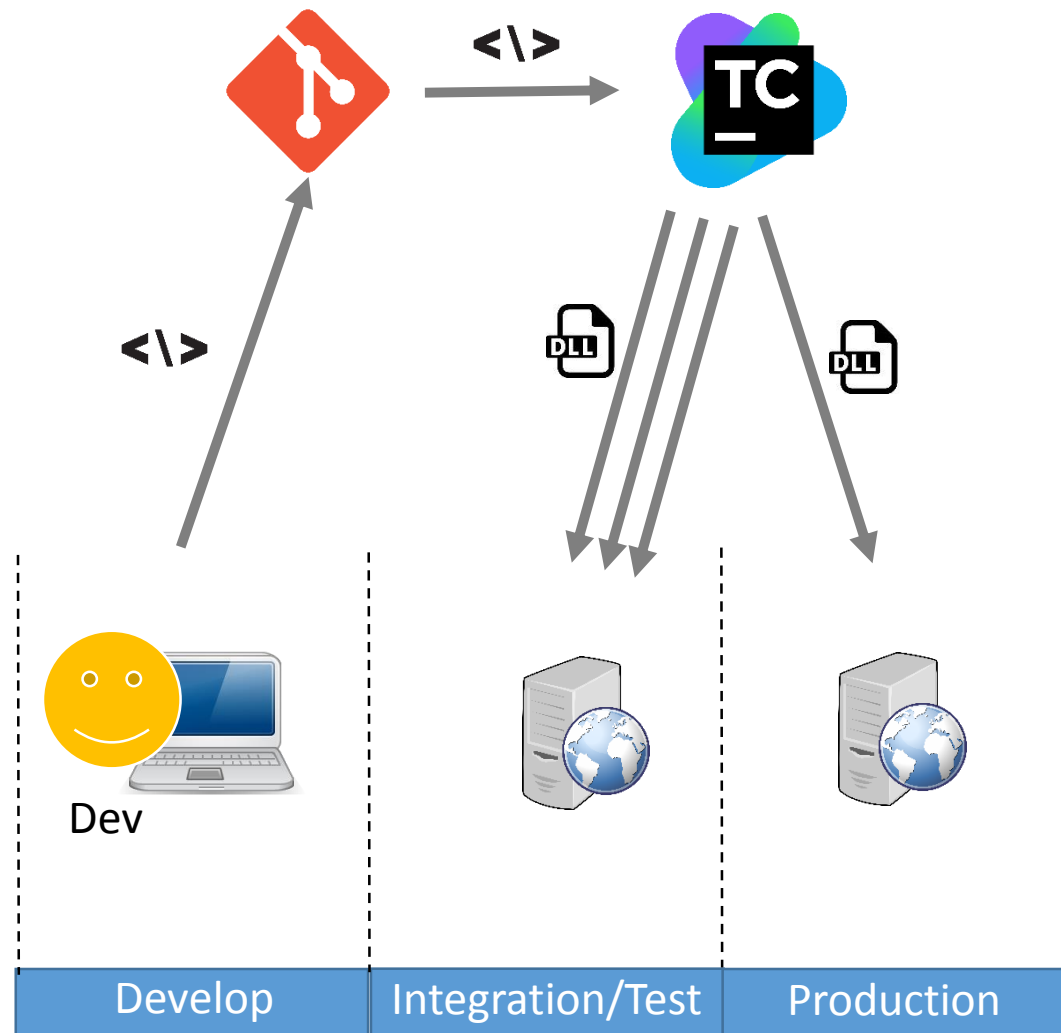
**UNREAL**

Давайте решим все  
эти проблемы!

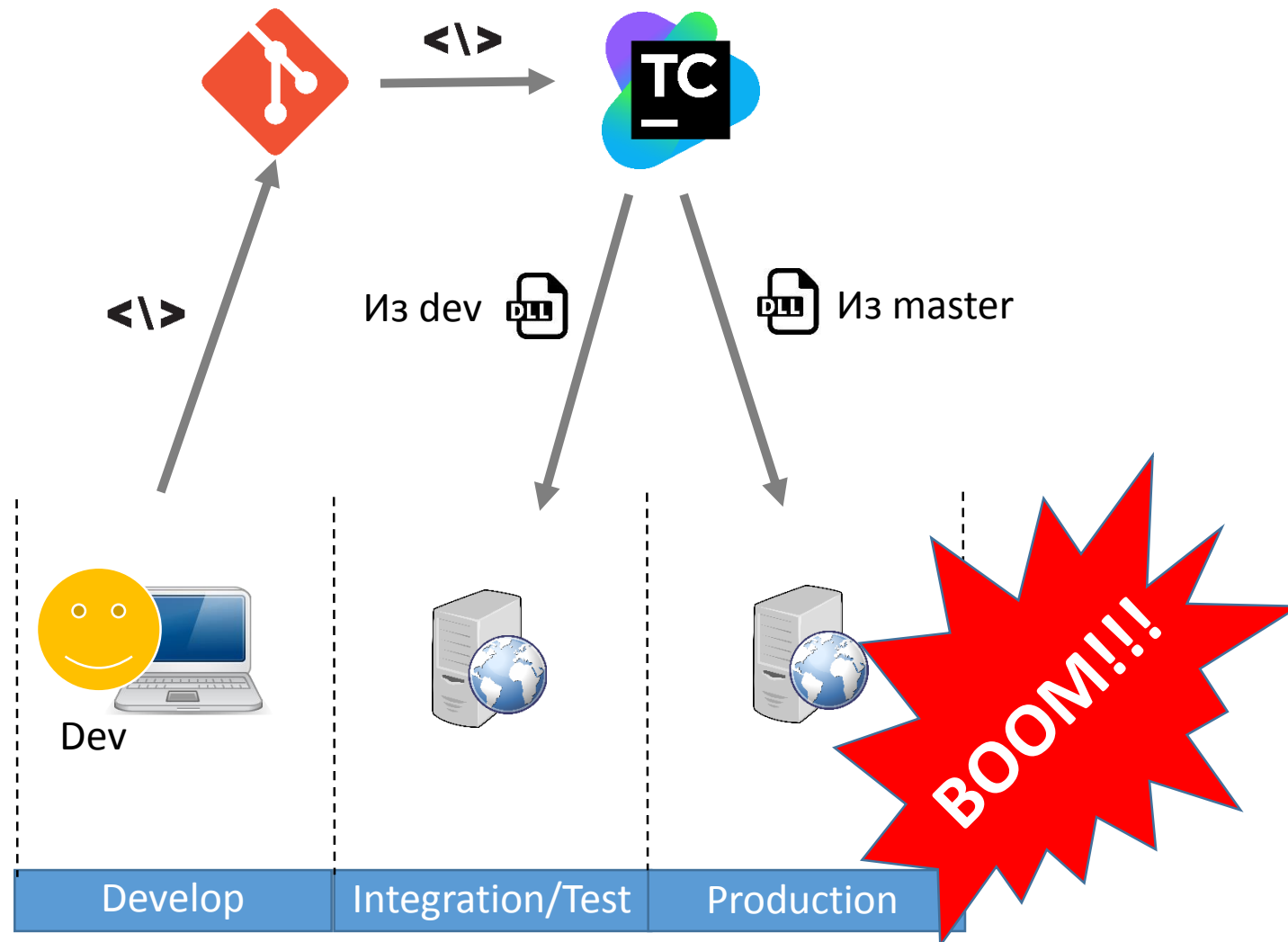
# На моем компьютере это работает



# Решение – добавляем CI

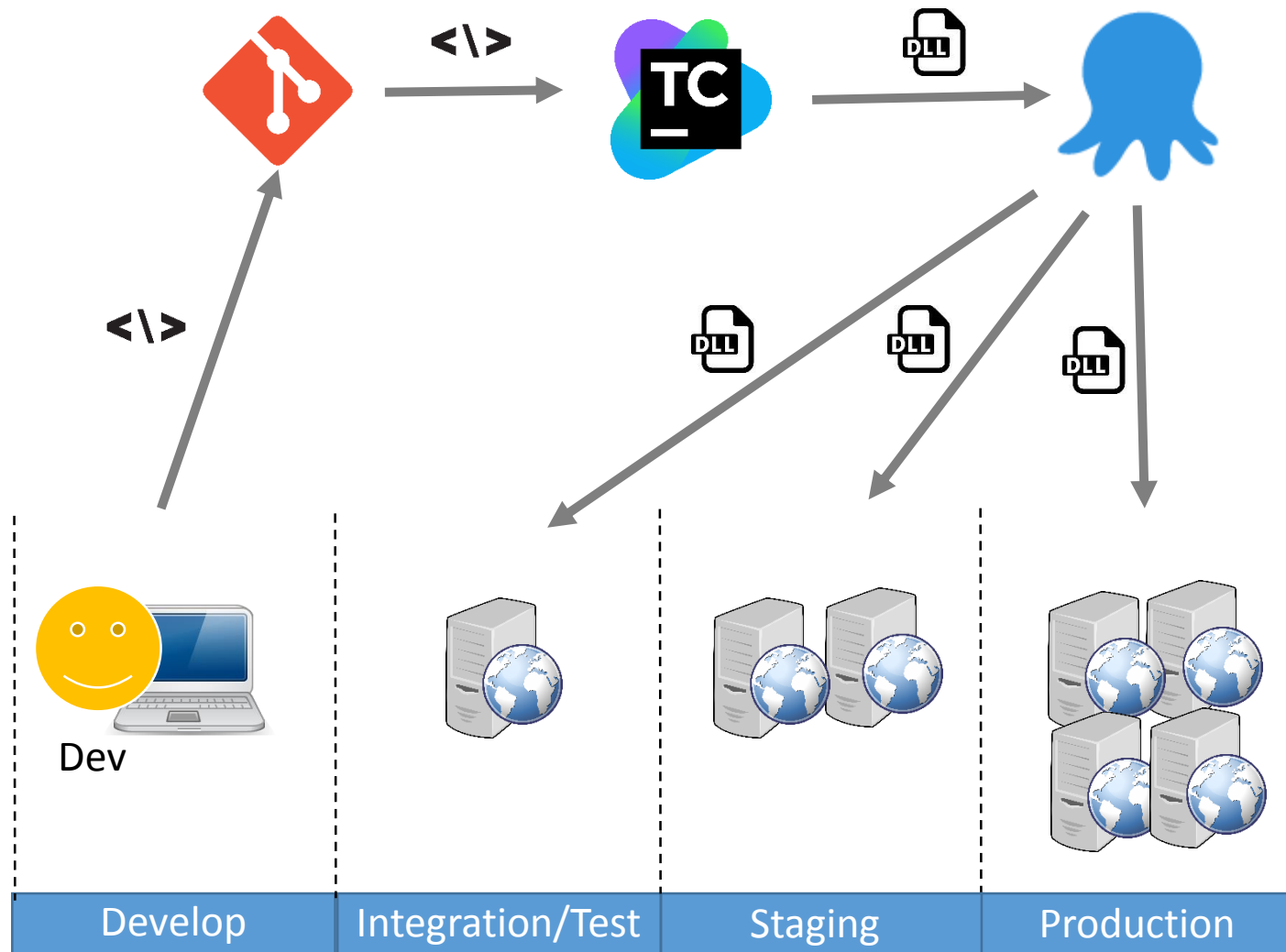


# Пересборка для каждой среды

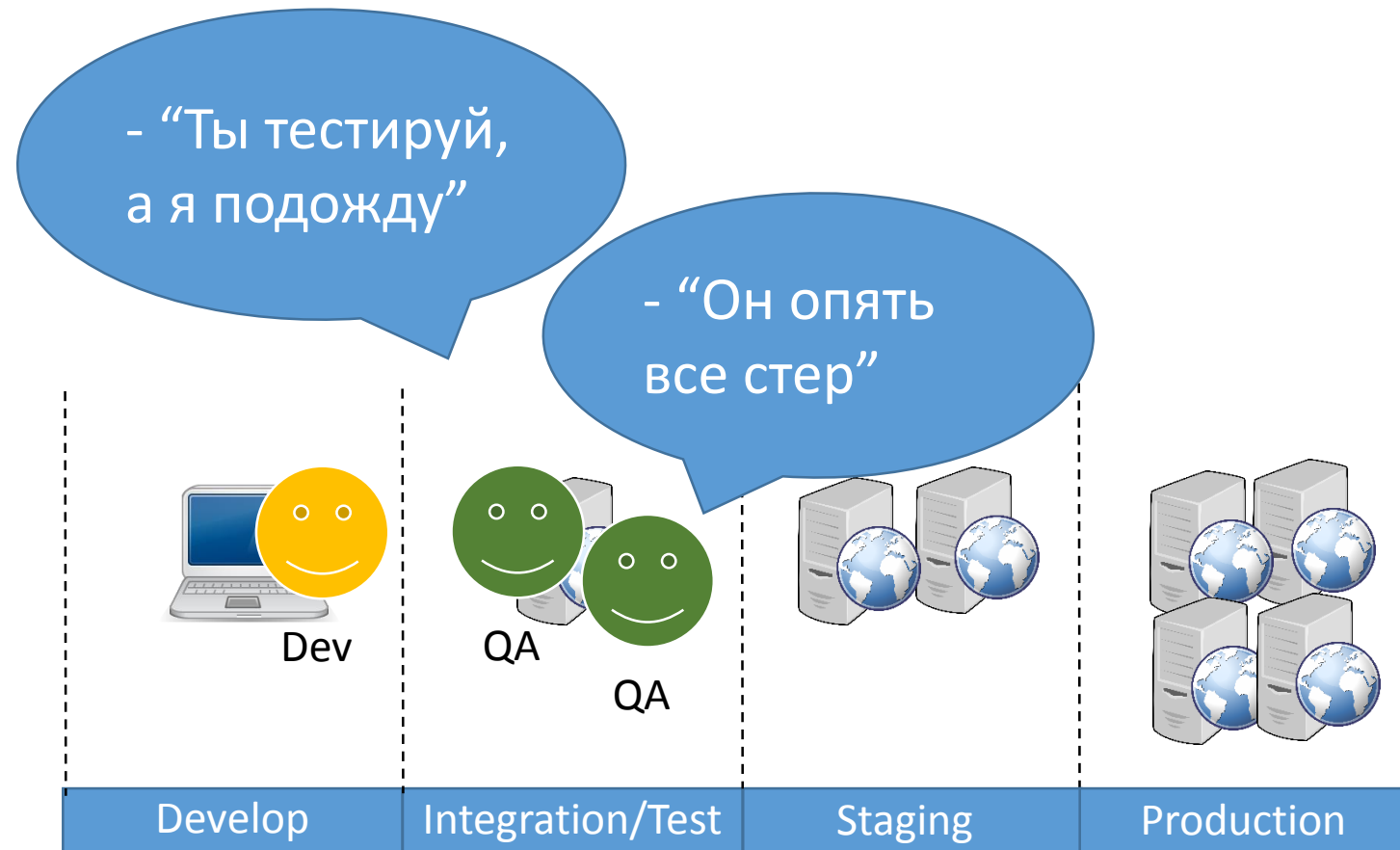




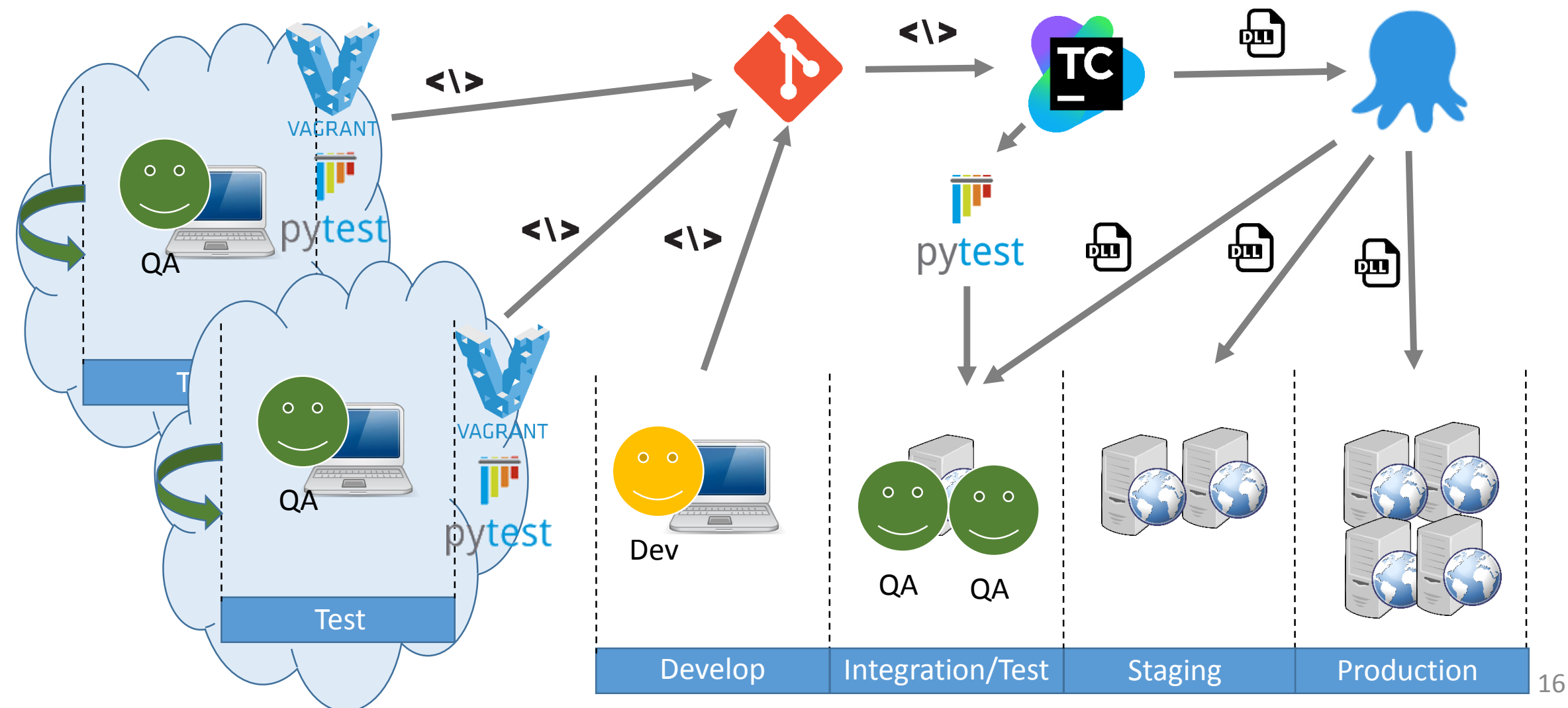
# Решение – добавляем CD



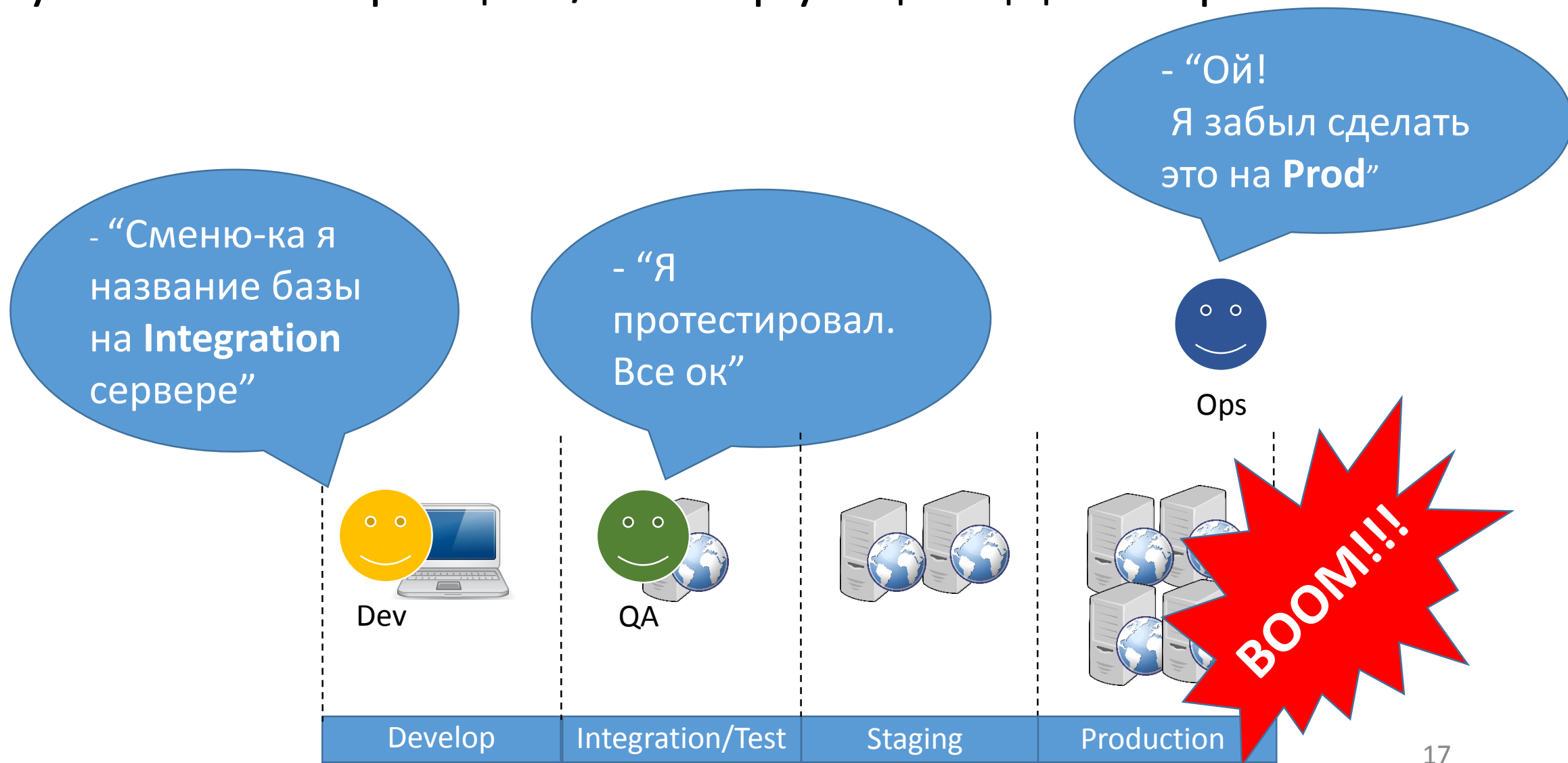
# Единый тестовый стенд



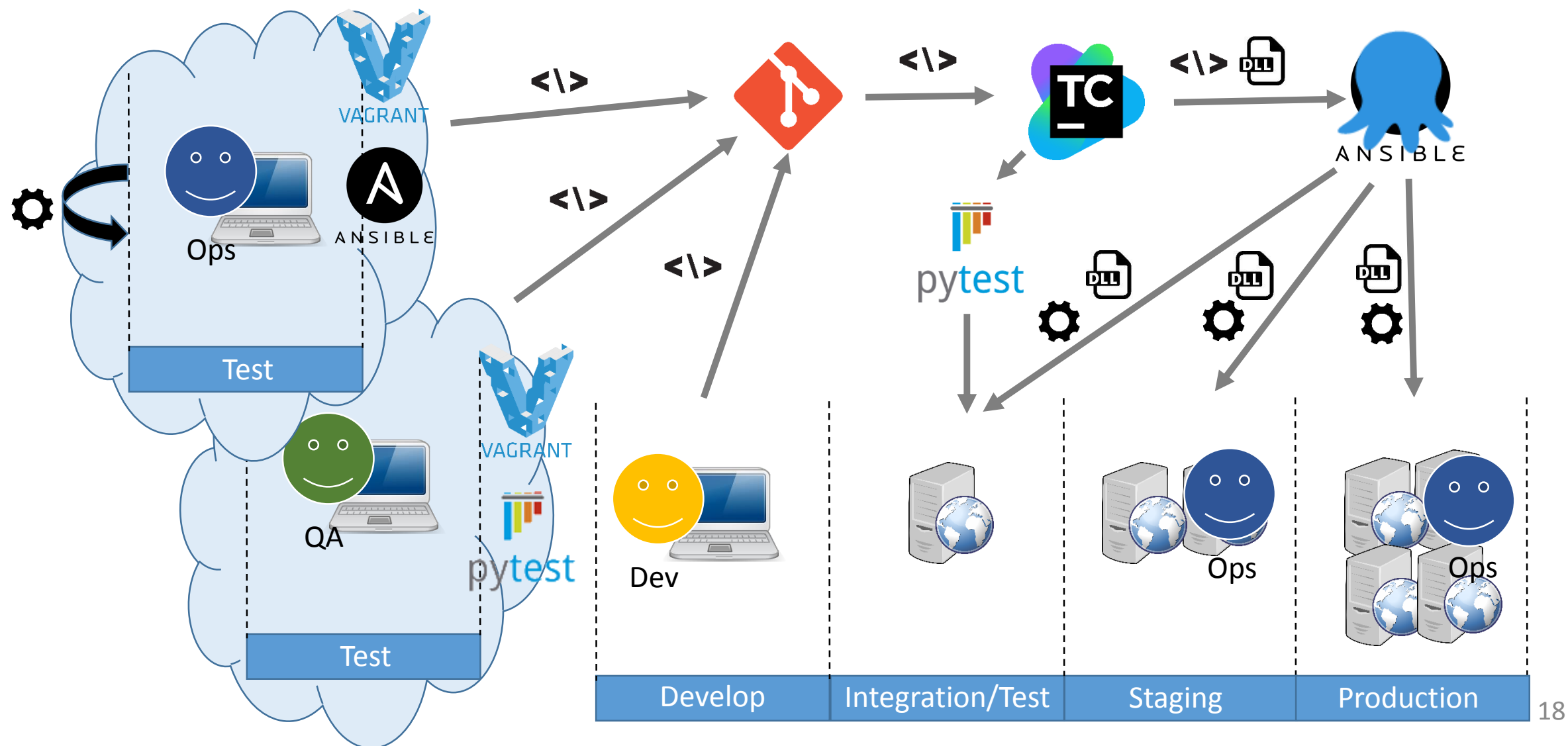
# Решение - локальные тестовые среды



# Ручные операции/инструкции для Ops

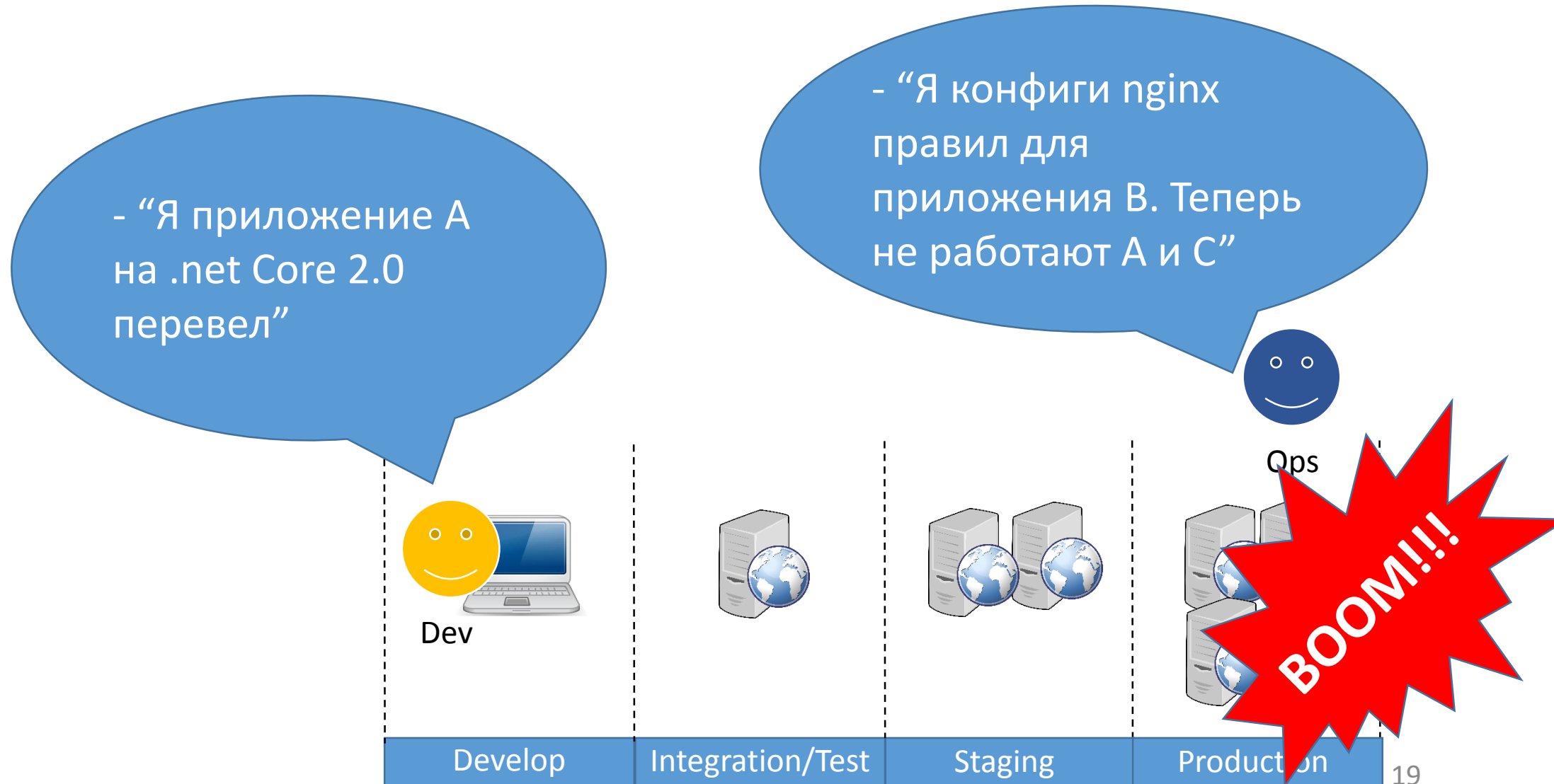


# Решение - добавляем Infrastructure as code

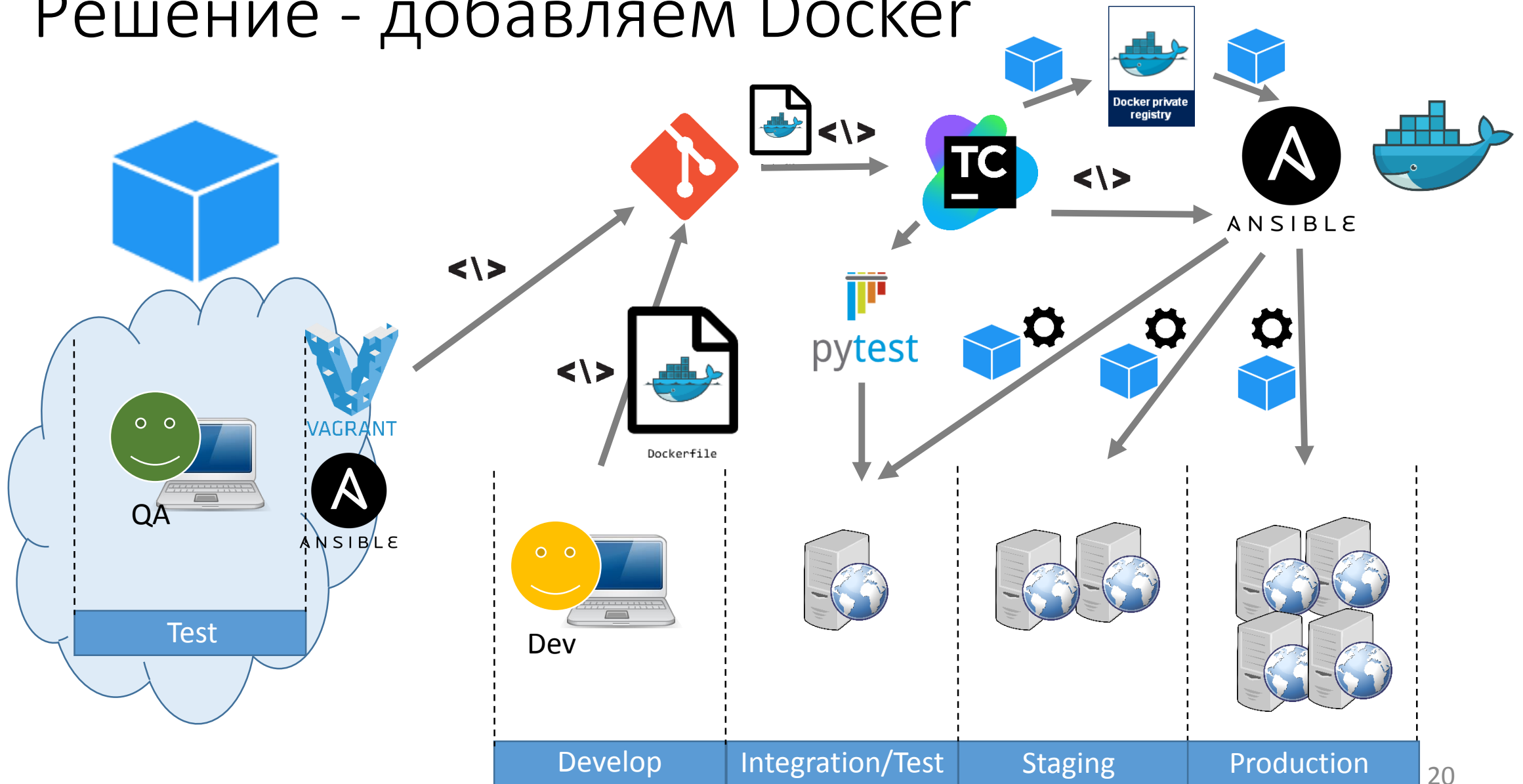




# Общее окружение для всех приложений



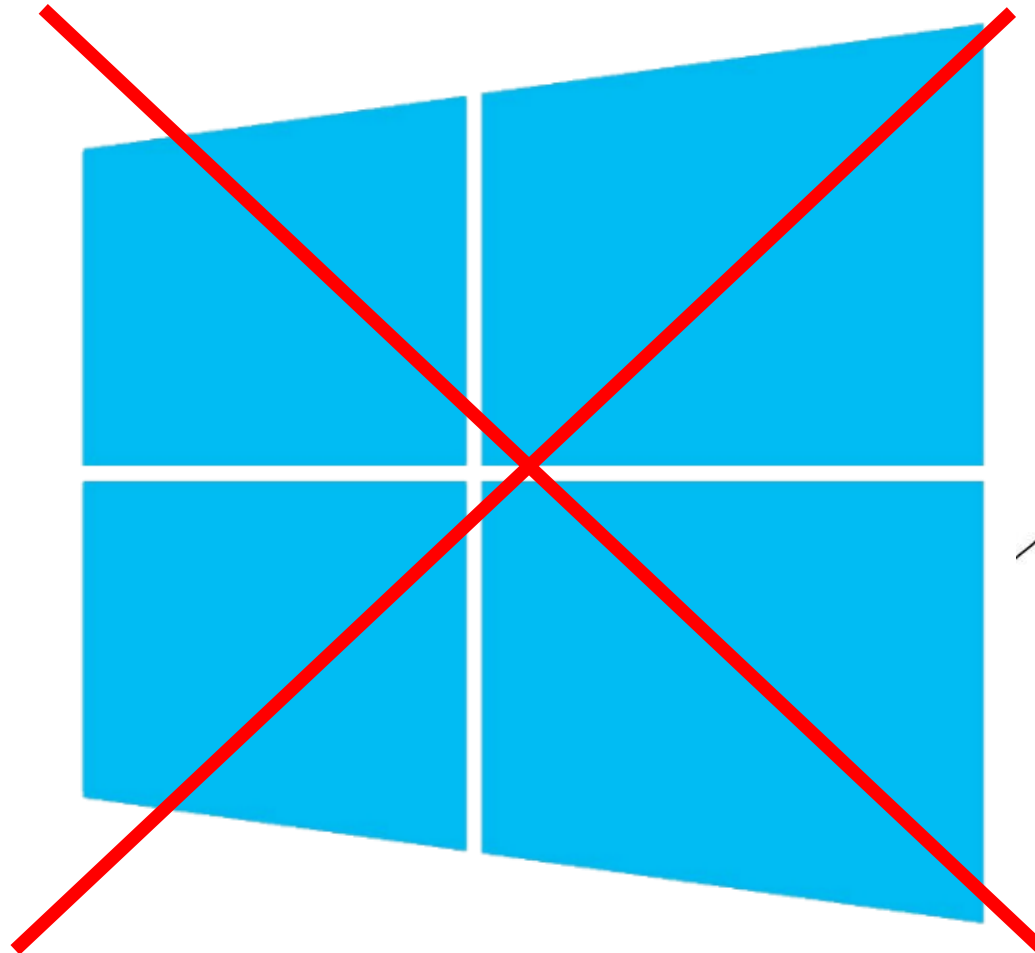
# Решение - добавляем Docker



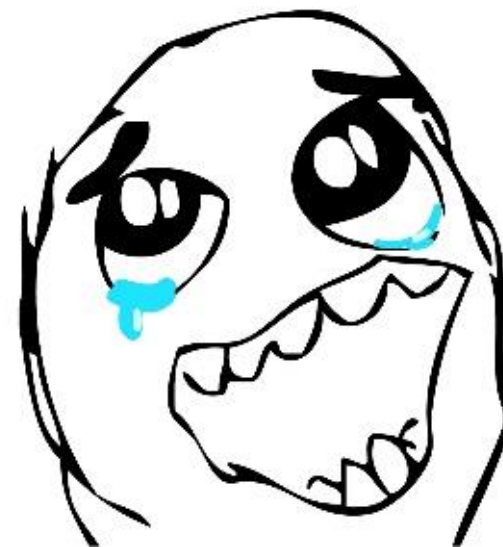
# Как это реализовать на Windows?

1. Купить Windows Server 2014(**323 848 руб.**)
2. Купить MS Sql Server 2014 (**374 502 руб.**)
3. Купить Azure подписку для VM на Windows  
(**х3 по сравнению с Linux**)

# Купили? – теперь попробуйте настроить

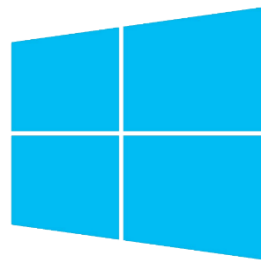


# Как это реализовать на Linux?





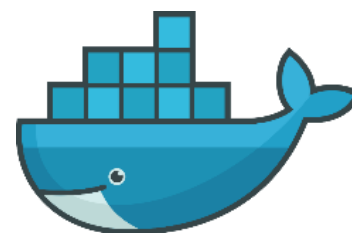
# Переход на Linux



0\$



0\$

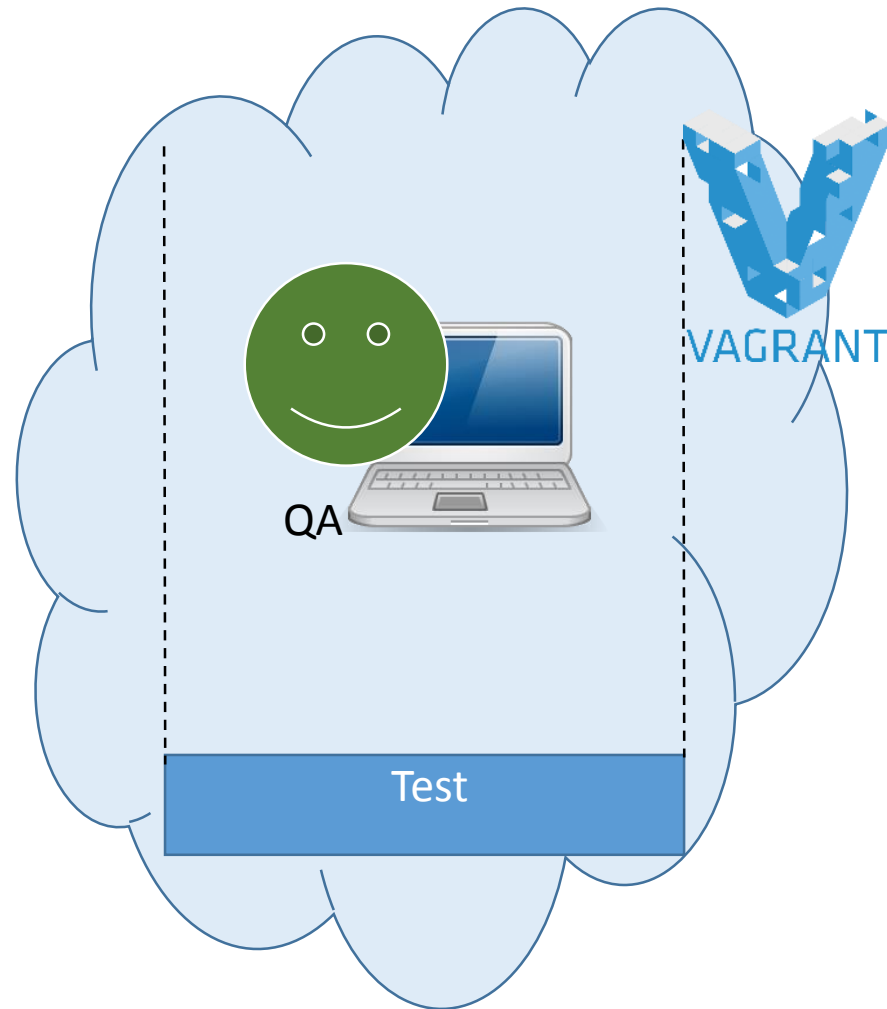


0\$

И нам показалось, что это очень круто



# Тестовая среда



# Что мы от нее ожидаем

- Убивается/создается «с кнопки»
- Вся конфигурация в Git
- Каждый тестировщик работает независимо со своей средой
- Настроена теми же скриптами что и Integration/Staging/Production(а значит равна им)

# Технологии





# Что делает Vagrant, Virtual Box



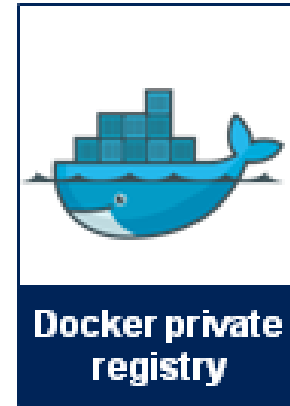
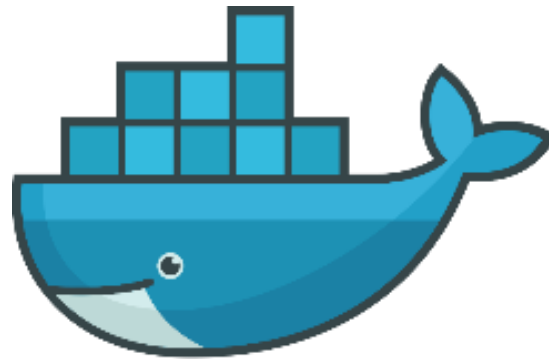
- Создает минимально настроенную виртуальную машину
- По сути это просто конфиг для VM

# Что делает Ansible



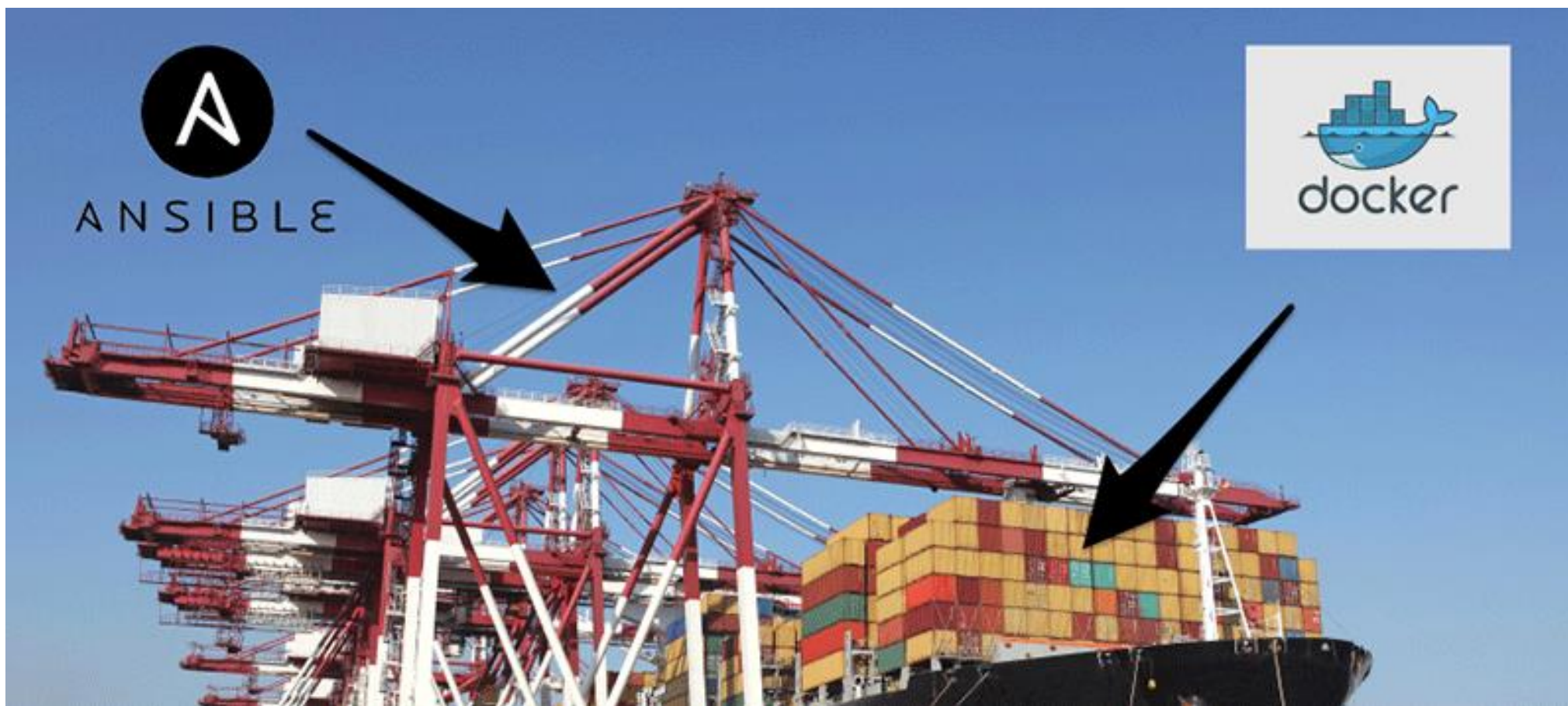
- Настройка ОС(ufw, mc и т.п.)
- Установка Docker, Docker Swarm
- Pull Docker Images
- Run/Stop/Remove Docker containers

# Что делает Docker

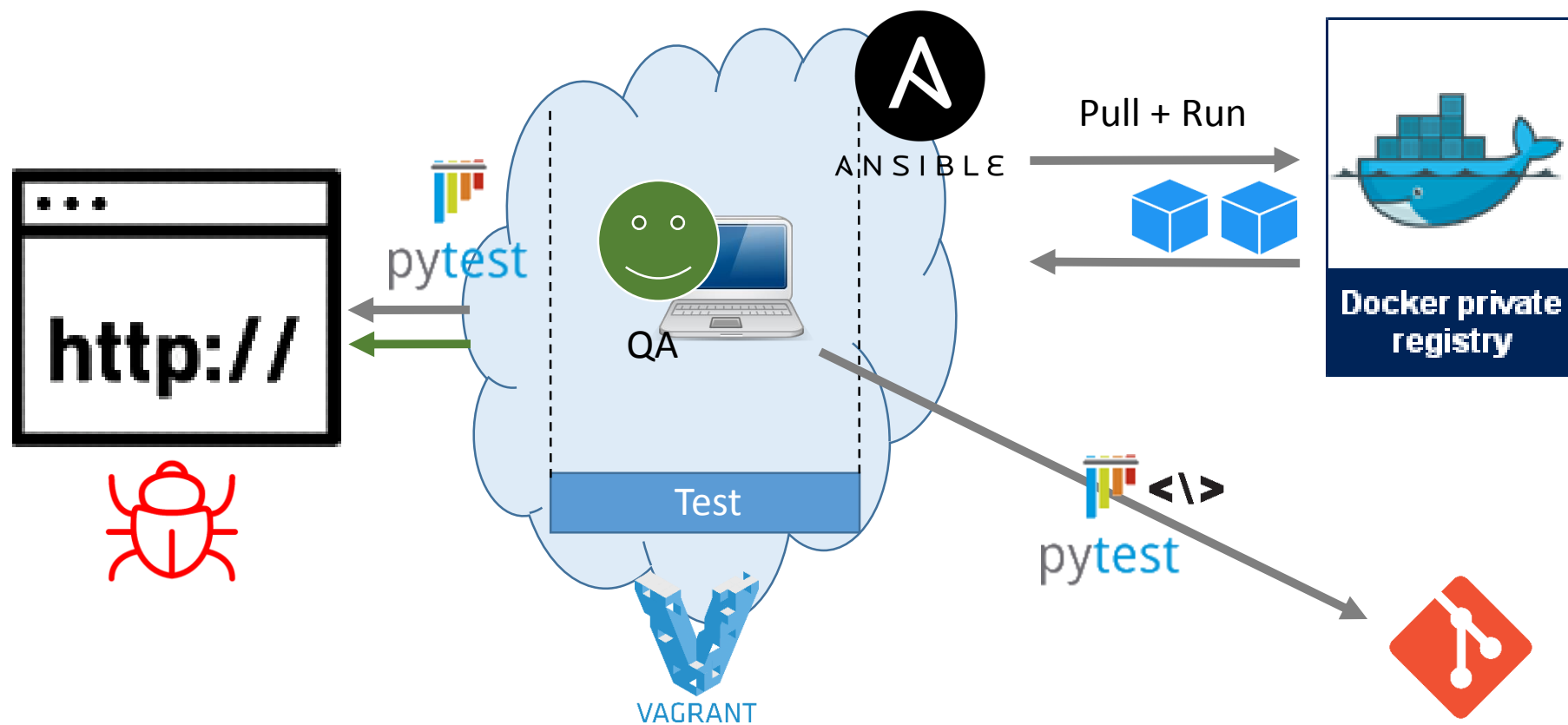


- Удобная доставка обновлений
- Изолированная работа приложений

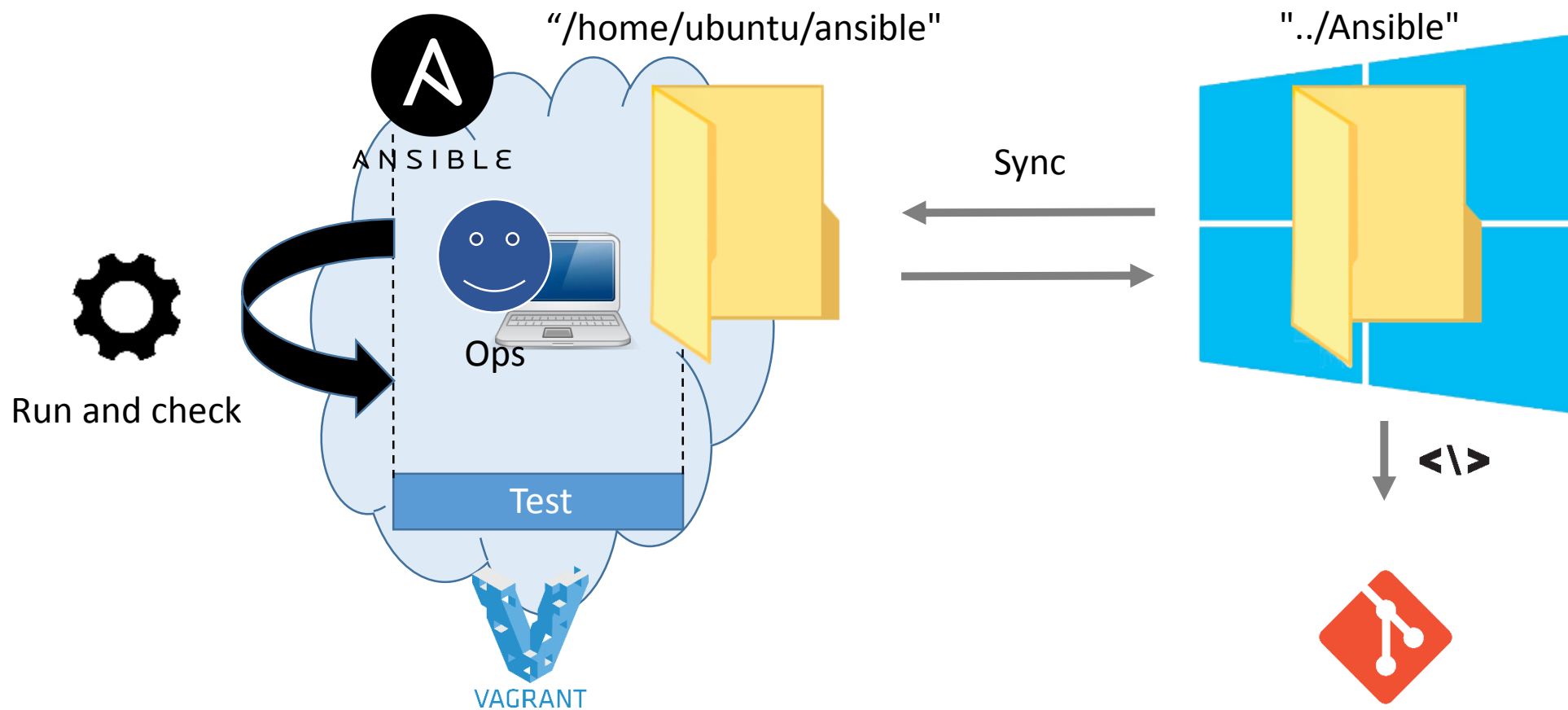
# Связь Ansible и Docker



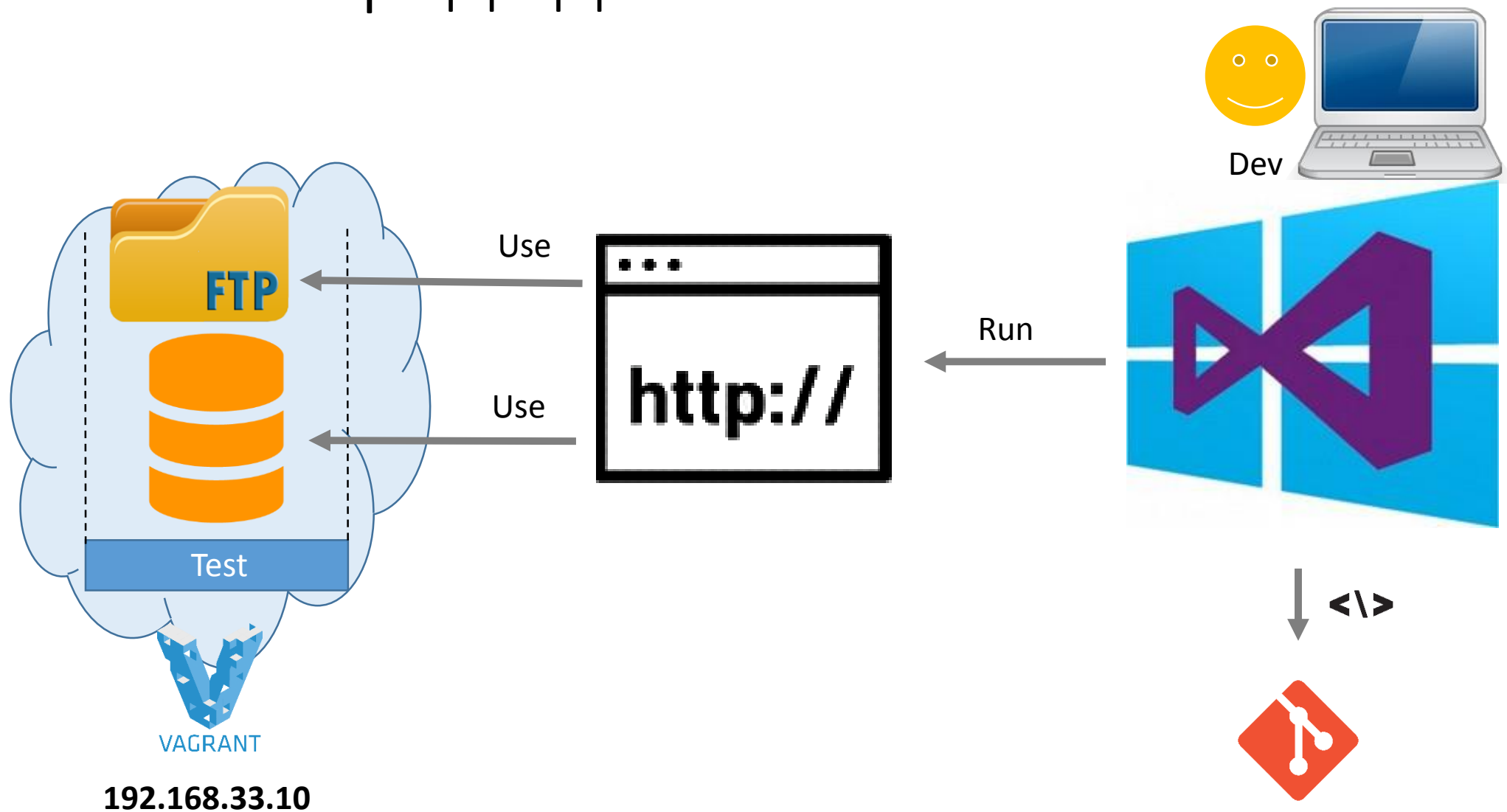
# Тестовая среда для QA



# Тестовая среда для Ops

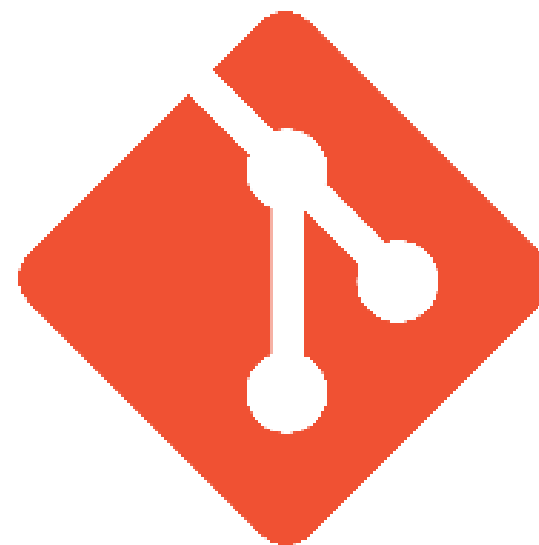
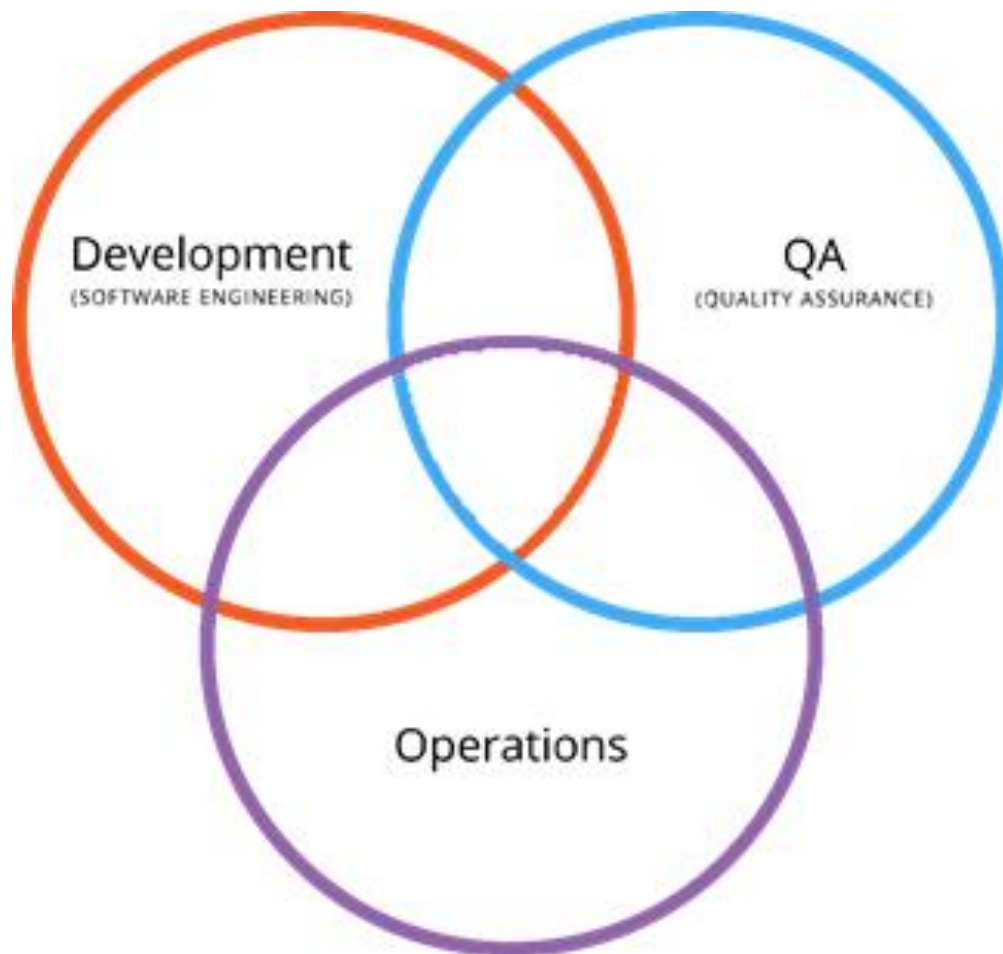


# Тестовая среда для Dev





# Все участники работают ради общей цели



# Команды Vagrant

- **“vagrant up”** – создать машину
- **“vagrant destroy -f”** – убить машину

# Команды Ansible

- **“ansible-playbook –i inventories/test main.yml “**  
– настроить машину
- **“ansible-playbook –i inventories/test deploy.yml**  
– deploy Docker контейнеров

# Демо – создание тестовой среды

<https://youtu.be/OzcJ7tb3nFE>

Демо – настройка и деплой на тест

<https://youtu.be/HVxQOX6wCYE>

# Vagrant - что внутри

Ubuntu 16.04



```
Vagrant.configure("2") do |config|  
  #box  
  config.vm.box = "ubuntu/xenial64"
```

Static IP



```
#network  
config.vm.network "private_network", ip: "192.168.33.10"
```

Ports



```
#forwarded ports  
config.vm.network "forwarded_port", guest: 80, host: 80  
config.vm.network "forwarded_port", guest: 5432, host: 5432  
config.vm.network "forwarded_port", guest: 20, host: 20  
config.vm.network "forwarded_port", guest: 21, host: 2121  
config.vm.network "forwarded_port", guest: 990, host: 990  
for i in 40000..40005  
  config.vm.network :forwarded_port, guest: i, host: i  
end
```

Apps



```
#apps  
config.vm.provision "shell", inline: "apt-get update -y"  
config.vm.provision "shell", inline: "apt-get install -y mc"  
config.vm.provision "shell", inline: "apt-get install -y expect"  
config.vm.provision "shell", inline: "apt-get install -y software-properties-common"  
config.vm.provision "shell", inline: "apt-add-repository -y ppa:ansible/ansible"  
config.vm.provision "shell", inline: "apt-get update -y"  
config.vm.provision "shell", inline: "apt-get install -y ansible"
```

Sync folder Ansible



```
#ansible  
config.vm.provision "shell", inline: "rm -rf /etc/ansible/*"  
config.vm.synced_folder "../Ansible", "/home/ubuntu/ansible"
```

```
#message  
config.vm.post_up_message = "Connection settings: 192.168.33.10 | ubuntu/ubuntu"  
end
```

# Ansible – что внутри

## Среды

 integration

 production

 stage

 test

```
[app-servers]
```

```
appserver1
```

```
appserver2
```

```
[db-servers]
```

```
dbserver
```





```
[ftp-servers]
```

```
ftpserver
```



# Ansible – что внутри

## Роли

-  app\_server
-  common
-  db\_server
-  docker
-  elk\_server
-  ftp\_server
-  post\_deploy
-  self

```
---  
- name: Update apt cache  
  apt:  
    update_cache: yes  
    cache_valid_time: 3600
```

# Docker – что внутри

Базовый образ

→  
`# Build runtime image  
FROM microsoft/aspnetcore:1.1.0  
RUN apt-get update -y && apt-get install mc -y && \  
apt-get install dos2unix -y && apt-get clean`

Приложение

→  
`WORKDIR /tools  
COPY tools/. .  
RUN chmod +x run.sh  
RUN dos2unix /tools/run.sh`

Пульс

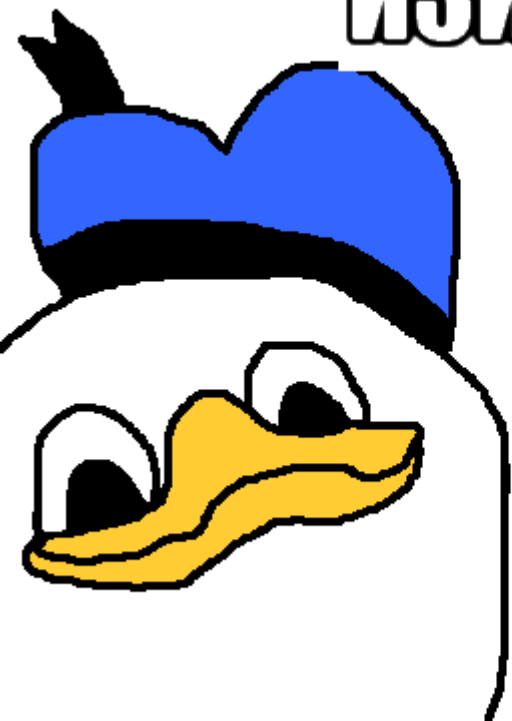
→  
`WORKDIR /netcore  
COPY --from=build-env /netcore/src/Atol.Billing.Api/Atol.Billing.Api/out .  
  
HEALTHCHECK --interval=30s --timeout=3s \  
CMD curl -f http://localhost:8500/swagger/ui/index.html || exit 1`

Скрипт запуска

→  
`ENTRYPOINT [ "/tools/run.sh" ]`

# Deploy на Integration/Staging/Production

ИЗМ



```
ansible-playbook -i inventories/integration main.yml
```

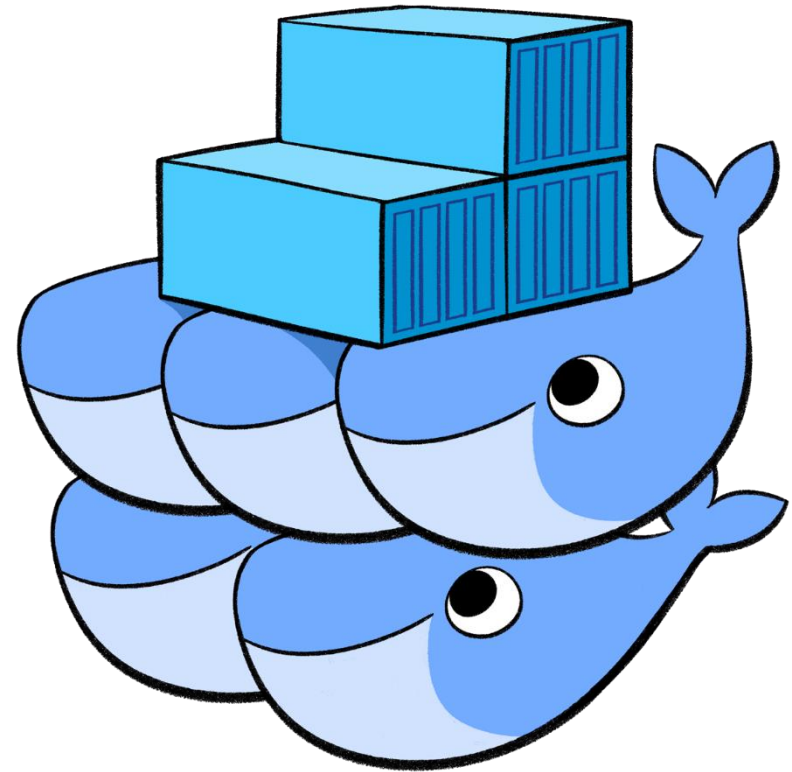
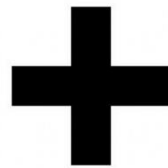
```
ansible-playbook -i inventories/stage main.yml
```

```
ansible-playbook -i inventories/production main.yml
```

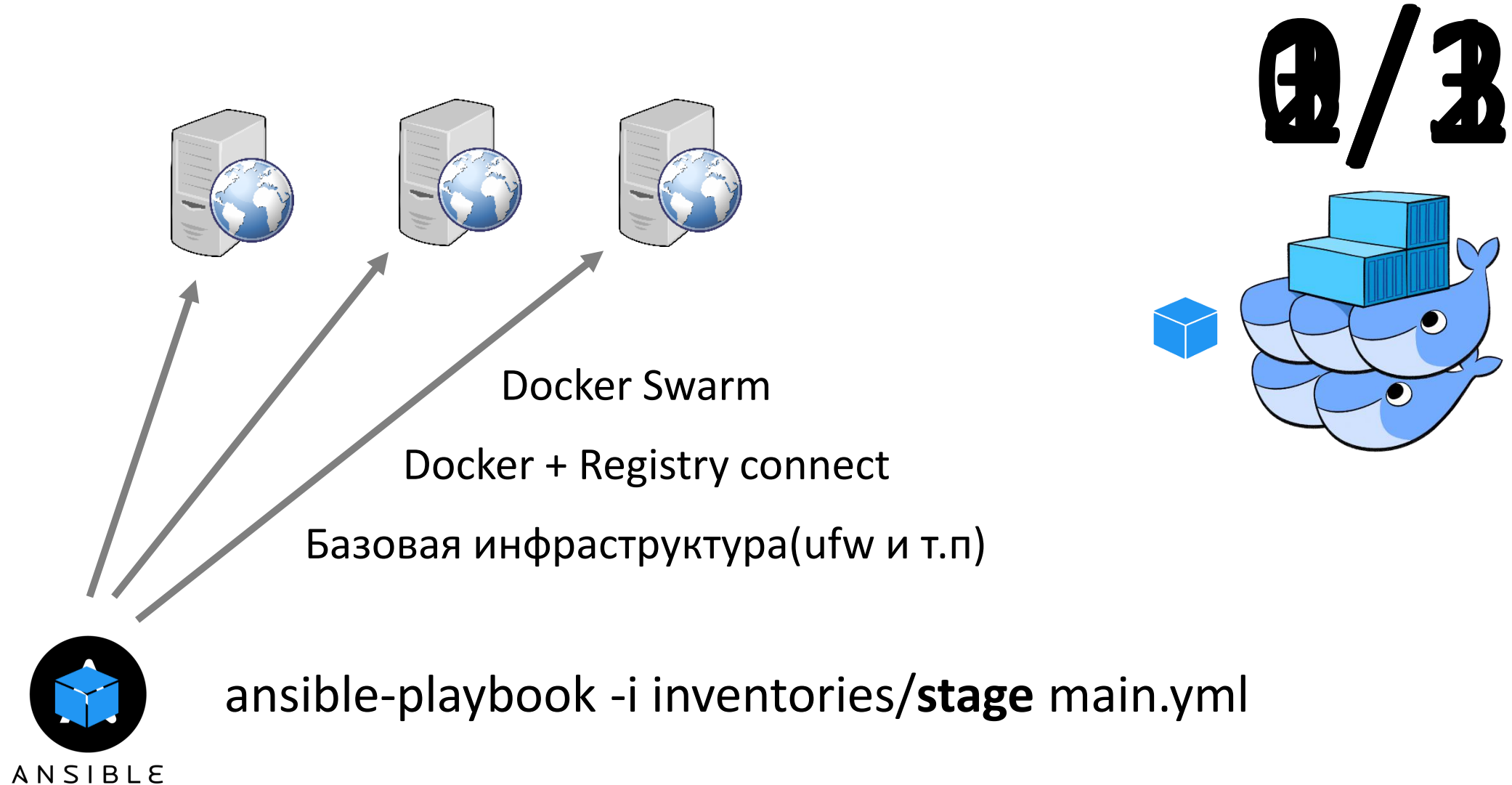
...

# Docker Swarm

```
HEALTHCHECK  
--interval=30s \  
--timeout=3s \  
CMD curl -f  
http://localhost:8500/swagger  
/ui/index.html || exit 1
```



# Docker Swarm + Ansible



# Демо – CI/CD инфраструктуры

<https://youtu.be/-iGMScCJNac>

# Демо – CI/CD Docker

<https://youtu.be/wfXzTvL15kg>

# ИТОГИ



- На моем компьютере это работает



- Пересборка для каждой среды



- Единый тестовый стенд



- Ручные операции/инструкции для Ops



- Общее окружение для всех приложений



# Выводы

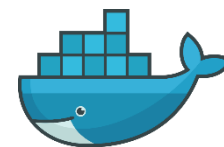
# Контроль

- Приложение – код
- Схема БД – код
- Авто тесты – код
- Инфраструктура – код
- CI/CD – код(ТС\*)



# Скорость

- Интеграция/деплой при каждом коммите
- Любое количество сред и машин
- Настройка машины – 5-7 минут
- Деплой – 1-3 минуты



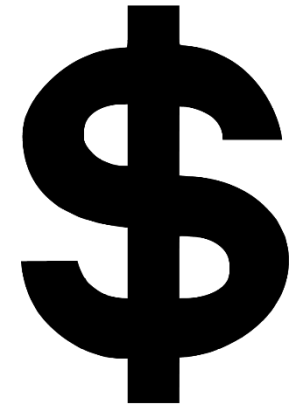
# Уверенность

- Частый деплой
- Частое создание/убийство сред
- Тестирование инфраструктуры
- Тестирование конвейера
- Тестирование кода
- Повторяемость
- Все среды одинаковы



# Выгода

- Linux, PostgreSQL = **0\$**
- Azure Linux VM дешевле Windows VM в **3 раза**
- 10 минут работы машины дешевле 2 дней работы Ops руками в **96 раз**



# .Net Core на ОС Linux

*«.Net Core на ОС Linux – это полностью контролируемый, быстрый, надежный процесс разработки и поставки ПО и в  $\infty$  раз дешевле чем на Windows»*

# Вопросы и ответы

**Ветчинкин Кирилл**

**<https://www.facebook.com/k.vetchinkin>**

**system-komkon@yandex.ru**