xored

Continuous Integration Delivery Deployment

Павел Нуждин

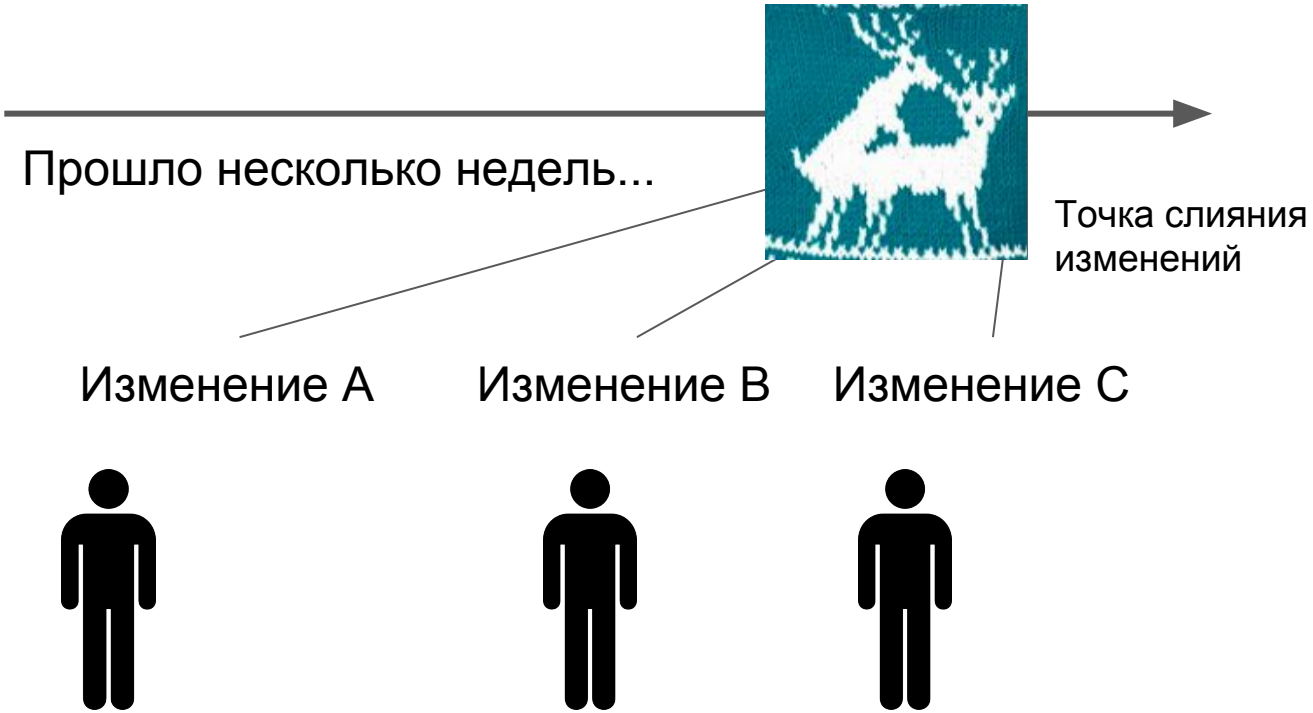
Technical Leader @ Xored


Как дела?
Всё сделали! Осталось проверить...
и развернуть!

План

- Continuous Integration:
  - Базовый процесс и pipelines.
  - Способы решения типичных проблем.
  - Метрики и советы.
- Continuous Delivery:
  - Окружение. Виды окружений. Создание окружений.
  - Базовый процесс.
  - Советы.
- Continuous Deployment.
- Плюсы и минусы CI/CD.
- Инструменты.
- Что почитать и посмотреть.
- Заключение.

Интеграция как событие

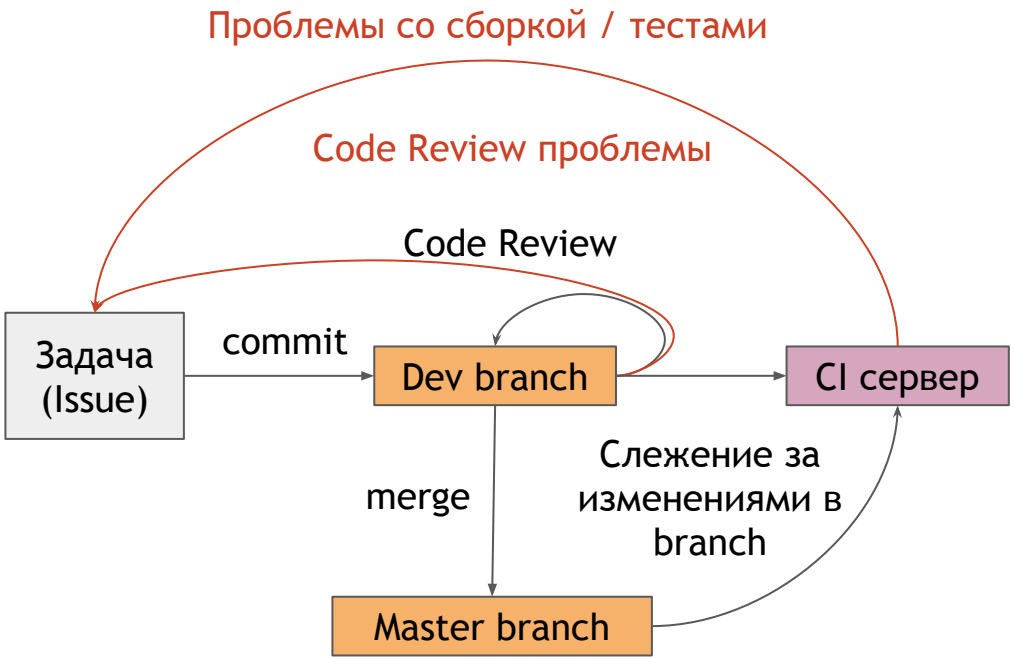


Continuous Integration (CI)

Практика регулярного слияния изменений в общий branch (trunk, master):

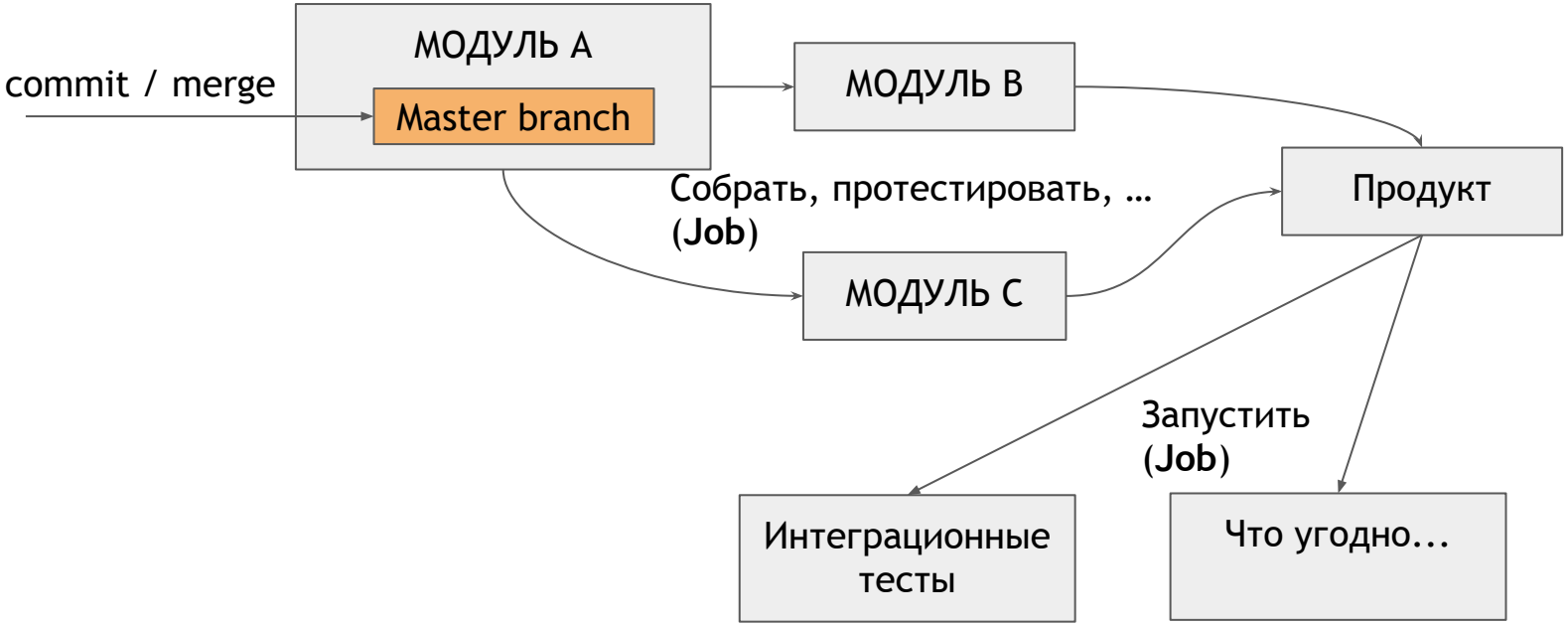
- Цель – смягчить интеграцию изменений.
- Несколько раз в день изменения сливаются в общий branch.
- Изменения тестируются юнит и интеграционными тестами перед слиянием.
- Тесты выполняются автоматически специальными CI серверами.
- Позволяет избежать выделения общей фазы интеграции изменений.

Базовый CI процесс



1. Компиляция в CI workspace.
2. Выполнение юнит и интеграционных тестов.
3. Сбор метрик.
4. Оповещение заинтересованных лиц.
5. Сохранение артефактов сборки.
6. Инициирование других сборок и задач.

CI pipelines




WAITING
TOO LONG
memegenerator.net

Sanity / Smoke тесты

Тесты покрывающие лишь небольшой класс ошибок:

- Позволяют проверить наиболее простые ошибки, как правило имеющие наибольшее значение.
- Намного быстрее обычных тестов.
- Предшествуют более глубокому тестированию.

Пример:

1. Открыть приложение.
2. Убедиться, что все UI элементы на месте, нет ошибок в логах.


TOO MUCH!
IT'S TOO MUCH!
memegenerator.net

Feature Flags

Техника интегрирования незаконченных или не готовых к выпуску изменений:

- Блоки кода оборачиваются в оператор ветвления, зависящий от значения feature flag (on/off).
- Значения могут быть изменены в работающем ПО, удаленно или в конфигурации.
- Могут быть использованы для постепенного ввода новой функциональности.

CI: Метрики

- Время компиляции (часы, минуты).
- Время выполнения тестов (часы, минуты).
- Время инспектирования кода (часы, минуты).
- Общее время сборки (часы, минуты).

CI: Метрики

- Отношение количества успешных сборок ко всем (процент).
- Покрытие тестами (процент).
- Результаты инспектирования кода (статический анализ):
  - Ошибки и предупреждения о нарушении стиля и правил (число).
  - Цикломатическая сложность (avg, max, число).
  - Объем дублированного кода (процент).
  - Объем кода (LOC, число).

CI: Советы

- Скрипты для сборки держите в репозитории.
- Гарантируйте доступность зависимостей.
- Пишите как можно больше качественных тестов.
- Запретите делать commit прямо в trunk/master.
- Стремитесь к сборке после каждого изменения. Не получается?
  => пишите sanity/smoke тесты и собирайте ночами.
- Следите за состоянием сборок.
- Следите за скоростью и актуальностью CI процесса.
- Не злоупотребляйте feature flags.

Continuous Delivery

Практика развертывания приложения в требуемых окружениях:

- Цель – код должен быть всегда готов к развертыванию.
- Базируется на налаженном Continuous Integration процессе.

CD: Окружение

Компьютерная система, состоящая из:

- Аппаратного обеспечения.
- Сетевой инфраструктуры.
- Операционной системы.
- Установленных программ и библиотек.

CD: Виды окружений

- Local / Development – машина разработчика.
- Integration – CI workspace.
- Testing/QA – окружение для команды QA-инженеров.
- Staging – окружение, максимально близкое к боевому.
- Production – окружение, обслуживающее реальных пользователей.

CD: Как создать окружение?

- ~~По-памяти~~.
- ~~По-инструкции / документации~~.
- Подложить артефакты в существующее окружение.
- Автоматически:
  - На обычной машине.
  - На виртуальной машине.
  - В виде Linux-контейнеров.

Vagrant: Окружение в виртуальной машине

Vagrantfile

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
  config.vm.provision :shell, path: "scripts/provision.sh"
  config.vm.network :forwarded_port, guest: 8000, host: 8000
end
```

server.js

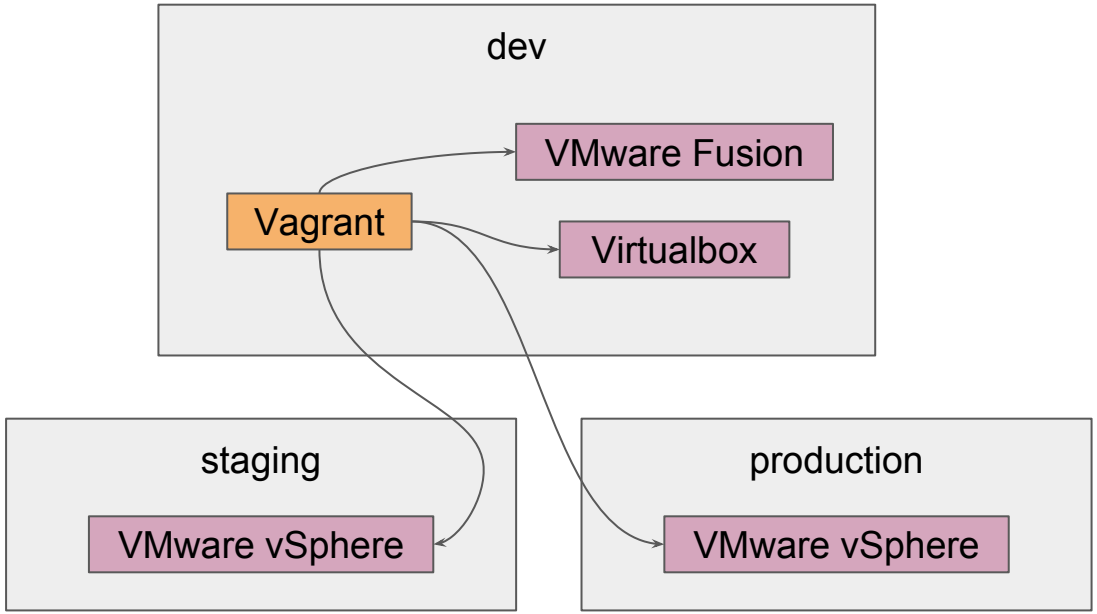
```
const http = require('http');
http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
}).listen(8000, '', () => {
  console.log('Server running');
});
```

scripts/provision.sh

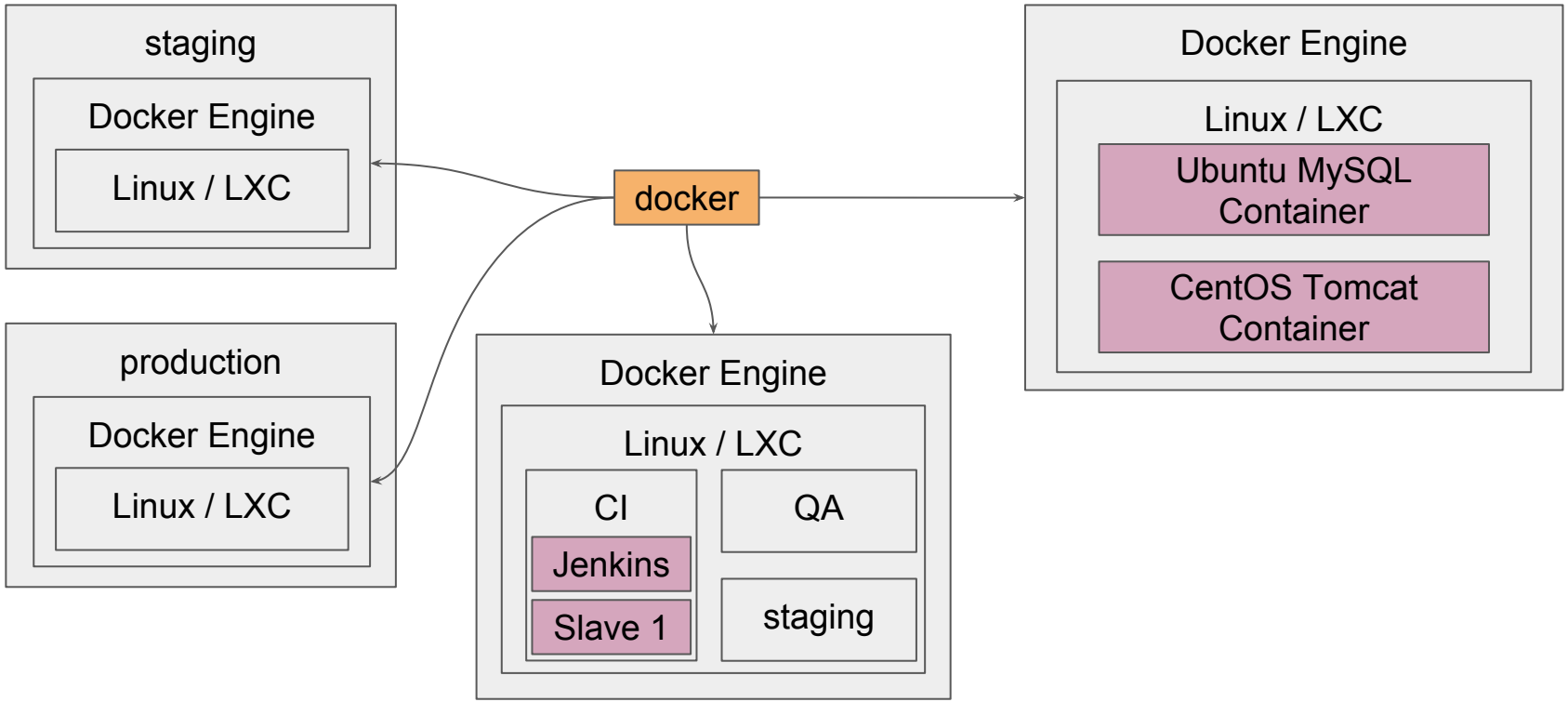
```
#!/usr/bin/env bash

wget https://nodejs.org/dist/v4.5.0/node-v4.5.0-linux-x64.tar.xz
tar xf node-v4.5.0-linux-x64.tar.xz
#...
nodemon /vagrant/server.js
```

```
$ vagrant up
```

Vagrant: Окружение в виртуальной машине



Docker: Окружение из Linux-контейнеров



Docker: Окружение из Linux-контейнеров

server.js

```
const app = require('./app');
const MongoClient = require('mongodb').MongoClient,

MongoClient.connect(process.env.MONGO_URI, (err, db) => {
  if (err) {
    console.error(err);
  } else {
    const server = app.listen(process.env.PORT, () => {
      app.set('server', server);
      app.set('db', db);
    });
  }
});
```

app.js

```
const express = require('express');

const app = express();
module.exports = app;

app.get('/users', (req, res) => {
  const db = app.get('db');
  const users = db.collection('users');
  users.find({}).toArray((err, result) => {
    if (err) {
      return res.status(500).json( { success:
false, reason: err.message });
    }
    res.send({ success: true, users: result });
  });
});
```

Docker: Окружение из Linux-контейнеров

docker-compose.yml

```
version: '2'
services:
  web:
    build: .
    links:
      - mongo
    ports:
      - "8000:8000"
    environment:
      - PORT=8000
      - MONGO_URI="mongodb://mongo:27017/demo"
  mongo:
    image: mongo
    volumes:
      - "./data:/data/db"
```

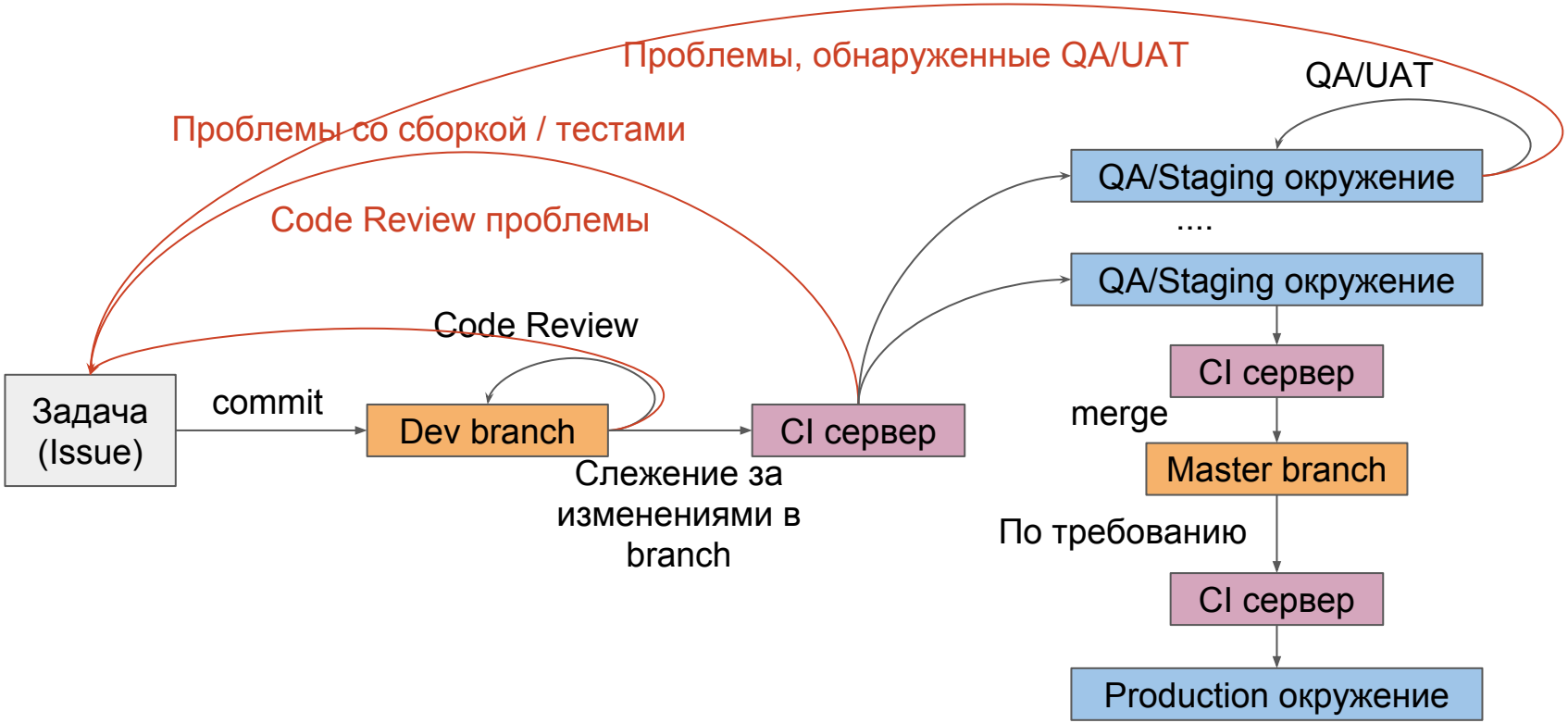
Dockerfile

```
FROM ubuntu:14.04

RUN apt-get update -yq && apt-get upgrade -yq && \
    apt-get install -yq curl git ssh sshpass
RUN apt-get -q -y install nodejs npm build-essential

# npm install

EXPOSE $PORT
CMD [ "node", "/src/app.js" ]
```

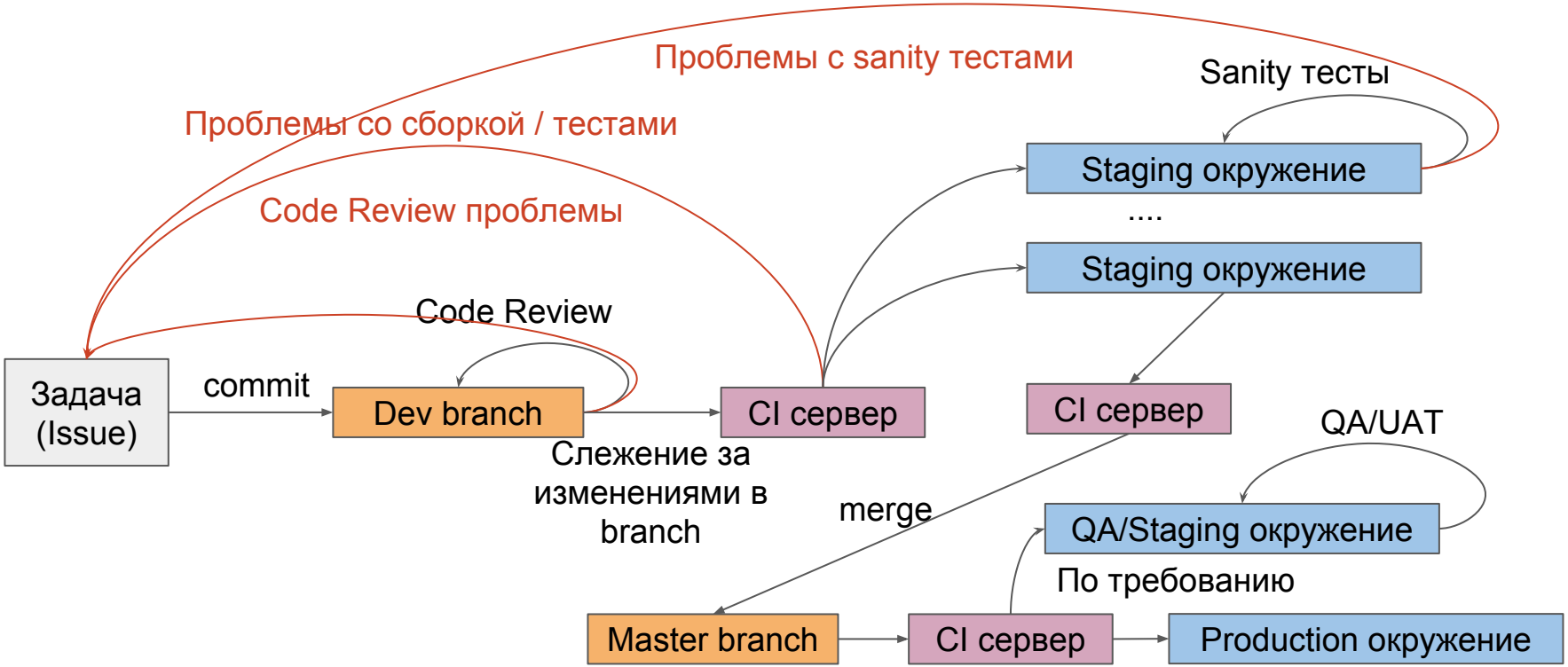
```
docker-compose up -d
```

```
$ docker-compose -f docker-compose.yml -f production.yml up -d
```

Базовый CD процесс



Чуть более реальный CD процесс



CD: Советы

- Создавайте скрипты миграции данных и конфигураций.
- Полностью автоматизируйте развертывание.
- Держите скрипты развертывания в репозитории.
- Сохраняйте артефакты для развертывания.
- Стремитесь к воспроизводимым окружениям.
- Либо ускоряйте процесс проверки решения QA-инженерами и приемочными тестами либо измените правила слияния изменений в общий branch.

Continuous Deployment

Практика автоматического развертывания приложения в Production:

- Цель – ускорить доставку обновлений реальным пользователям.
- Базируется на налаженном Continuous Delivery процессе.
- Как только изменения одобряются QA и UAT, они автоматически сливаются в общую ветку и развертываются в Production.

Плюсы и минусы CI/CD

+ Быстрое выявление ошибок, которые, как правило, исправляются автором изменений => повышение качества продукта.
+ Всегда доступна работающая версия для демонстрации.
+ Прозрачный процесс получения релизной версии.
+ Снижение рисков для бизнеса.
- Вводятся новые сущности и процессы, которыми нужно управлять, а также:
  - Покупать и поддерживать сервера для сборки.
  - Нанимать людей, поддерживающих сервера и процессы.
- CI/CD сущности и процессы - точки синхронизации команд => поломка может заблокировать многое.

Инструменты

- CI-сервера:


Jenkins Bamboo TeamCity cruisecontrol


Travis CI drone.io

- Feature Flags:


LaunchDarkly

- Статический анализ:

sonarqube 
checkstyle JSLint
The JavaScript Code Quality Tool

Инструменты

- Конфигурирование:

- Создание и управление LXC-окружениями:

- Создание и управление виртуальными окружениями:

Что почитать и посмотреть?

- http://martinfowler.com/articles/continuousIntegration.html.
- https://blog.assembla.com/AssemblaBlog/tabid/12618/bid/92411/Continuous-Delivery-vs-Continuous-Deployment-vs-Continuous-Integration-Wait-huh.aspx.
- Continuous Integration. Improving Software Quality and Reducing Risk / Paul Duvall, Steve Matyas, and Andrew Glover.
- Jenkins Continuous Integration Cookbook. Second Edition / Alang Mark Berg.
- https://www.docker.com/use-cases/cicd.

Заключение: Тест Джоэла Спольски

- ✅ Пользуетесь ли вы системой контроля версий?
- ✅ Можете ли вы собрать продукт за один шаг?
- ✅ Есть ли у вас ~~ежедневные~~ билды на каждый коммит?
- ❑ Есть ли у вас баг-трекер?
- ✅ Исправляете ли вы ошибки перед написанием нового кода?
- ❑ Есть ли у вас актуальный план работ?
- ❑ Есть ли у вас спецификация?
- ❑ Комфортные ли условия работы программистов?
- ❑ Используете ли вы самое современное оборудование
- ✅ Есть ли у вас QA инженеры?
- ❑ Пишут ли кандидаты на работу код во время собеседования?
- ❑ Проводите ли вы коридорное UX тестирование?