

# DevSecOps и технологии контейнеризации

Представление программы

# Цель обучения

---

- Освоить современные практики непрерывной интеграции и доставки нового кода (Continuous Integration / Continuous Delivery, CI/CD) на примере технологий GitLab/GitFlow и Docker. Участники разберутся с принципами CI/CD, научатся создавать пайплайны на примере Gitlab CI, освоят методы и инструменты контейнеризации на примере Docker, научатся строить цепочку поставки, включающую все основные стадии - сборка, тестирование, упаковка, сканирование.

# Чему мы научимся

---

- Работать в git и выстраивать gitflow
- Разберемся с принципами CI/CD и ознакомимся с лучшими практиками
- Научимся создавать пайплайны в gitlab CI
- Освоим методы и инструменты контейнеризации на примере docker
- Научимся строить цепочку поставки, включающую все основные стадии – сборка, тестирование, упаковка, сканирование
- Разберемся с SAST и DAST, научимся находить уязвимости
- Ознакомимся с принципами безопасной разработки
- Разберемся с безопасностью рабочей нагрузки и побегами из контейнера
- И многое другое

# Почему решили сделать эту программу

---

- Тема «горячая», но мы ее выбрали не поэтому
- Необходимость давать актуальные навыки студентам.
  - Навыки работы с git, навыки DevOps – входят в базовый набор требований современного специалиста
- Востребовано индустрией
  - Студенты сталкиваются с этим при попытках трудоустройства
- Нет в базовых программах
  - Студенты выходят без знания правильной работы git, docker, CI и т.п.
- Новые технологии дают новые вызовы с точки зрения безопасности
  - Важно знать, как их использовать на благо и делать приложения защищенными

# DevOps/DevSecOps обучение

---



## **Летняя школа**

Проходит в июле или в августе. Основы DevOps



## **Зимняя школа**

Январь или февраль. Более сложные темы



## **Программа ДПО**

Осень 2022

# Кому может быть полезно



## Разработчику

чтобы научиться настраивать процесс сборки, тестирования и разворачивания приложения



## Системному администратору

автоматизировать рутинные операции, оптимизировать нагрузки



## QA-инженеру

научиться строить автоматизированные тестовые среды, настраивать предпродакшн тестирование

# Кому может быть полезно

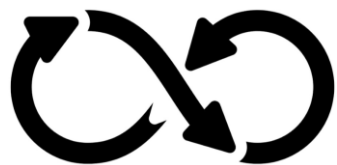
---



## **Специалисту по безопасности**

при внедрении CI/CD в организации, часть его экспертизы так же может быть автоматизирована.

должен научиться взаимодействовать с разработчиками и тестировщиками (DevSecOps)



## **DevOps инженер**

сейчас отдельная профессия

# Представление тем

---

- Введение в DevOps/DevSecOps
- Инструменты совместной работы
- Технологии контейнеризации
- Обеспечение безопасности контейнеров
- Автоматизация цепочки поставки
- Безопасность рабочей нагрузки
- Технологии оркестрации (самостоятельная работа)



# Дополнительно

---

- Группа в telegram
  - основные оповещения и канал взаимодействия
- Страница программы
  - <https://git.miem.hse.ru/vbashun/devsecops2022>
  - Информация будет обновляться
- Трансляции
  - Youtube

Начнем!

# DevSecOps и технологии контейнеризации

DevOps, DevSecOps

# DevOps – Development и Operations

- Есть несколько стадий, которые проходит любой код от разработки до пользователя



# DevOps – Development и Operations

- Основные стадии
  - разработка (кодирование),
  - сборка,
  - тестирование,
  - упаковка,
  - выпуск релиза,
  - настройка инфраструктуры,
  - внедрение на продуктовый сервер,
  - мониторинг

# DevOps – Development и Operations

- Две большие группы занимаются следующими вопросами
- Development
  - Разработка приложения
  - Тестирование, выпуск релизов
- Operations
  - Развертывание приложения
  - Поддержка и мониторинг



# DevOps – Development и Operations

- Этот цикл повторяется



# DevOps – Development и Operations

- Каждый продукт проходит цикл
  - Идея – Разработка (Кодинг) - Сборка - Тестирование - Упаковка – Развертывание
- Кроме того, постоянная доработка
  - Новые функции, новые версии продукта
  - Исправление багов
- Каждое изменение запускает цикл заново
- Без автоматизации процессов быстрые циклы невозможны
- DevOps – это про то, как сделать совместную работу разных команд эффективнее



# DevOps – Development и Operations

- В частности, внедрение DevOps призвано
  - Ускорить прохождение всех этапов жизненного цикла ПО, обеспечить эффективное взаимодействие разных членов команды
  - Ускорить получение обратной связи и исправление ошибок
  - Обеспечить возможность быстрого отката в случае ошибок
  - Ускорить процесс обновлений ПО
  - Упростить процессы оптимизации нагрузок и восстановления после сбоев (косвенно)
- Все это происходит за счет автоматизации всех процессов

# DevOps – Development и Operations

- Идея автоматизировать рутинные операции не нова
- Что же изменилось, откуда всплеск интереса к DevOps?
- За последние 15 лет появилось много технологий, которые существенно изменили многое, в том числе, подход к разработке и внедрению приложений
  - Посмотрим некоторые из них



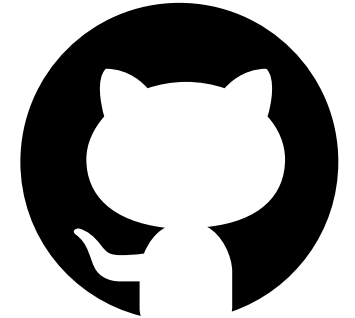
# Git (2005)

- Распределенная система управления версиями
- Системы контроля версий были и раньше, просто git оказался лучшим
  - в плане работы из командной строки,
  - работы с ветками,
  - разрешения конфликтов,
  - синхронизации,
  - работы оффлайн и т.п.



# Git

- Постепенно появлялись публичные и частные репозитории
- Например github – основное хранилище opensource решений
- Были и другие
  - GitLab, Bitbucket, etc.
- Больше чем просто репозитории



# Технологии виртуализации

- Аппаратная поддержка виртуализации
- Появление гипервизоров
  - Специальные приложения, которые позволяли запускать на одной машине несколько виртуальных платформ
  - Открепление сервера от железа, работа в виртуальном окружении
- IBM LPAR, VMware, Hyper-V, Xen, KVM, Bhyve
- Платформы управления облачными ресурсами

# Публичные облака

- Далее естественно появились публичные облачные платформы
  - Amazon Web Services (2006),
  - Google Cloud (с 2008),
  - Azure (2010)
  - и т.п.

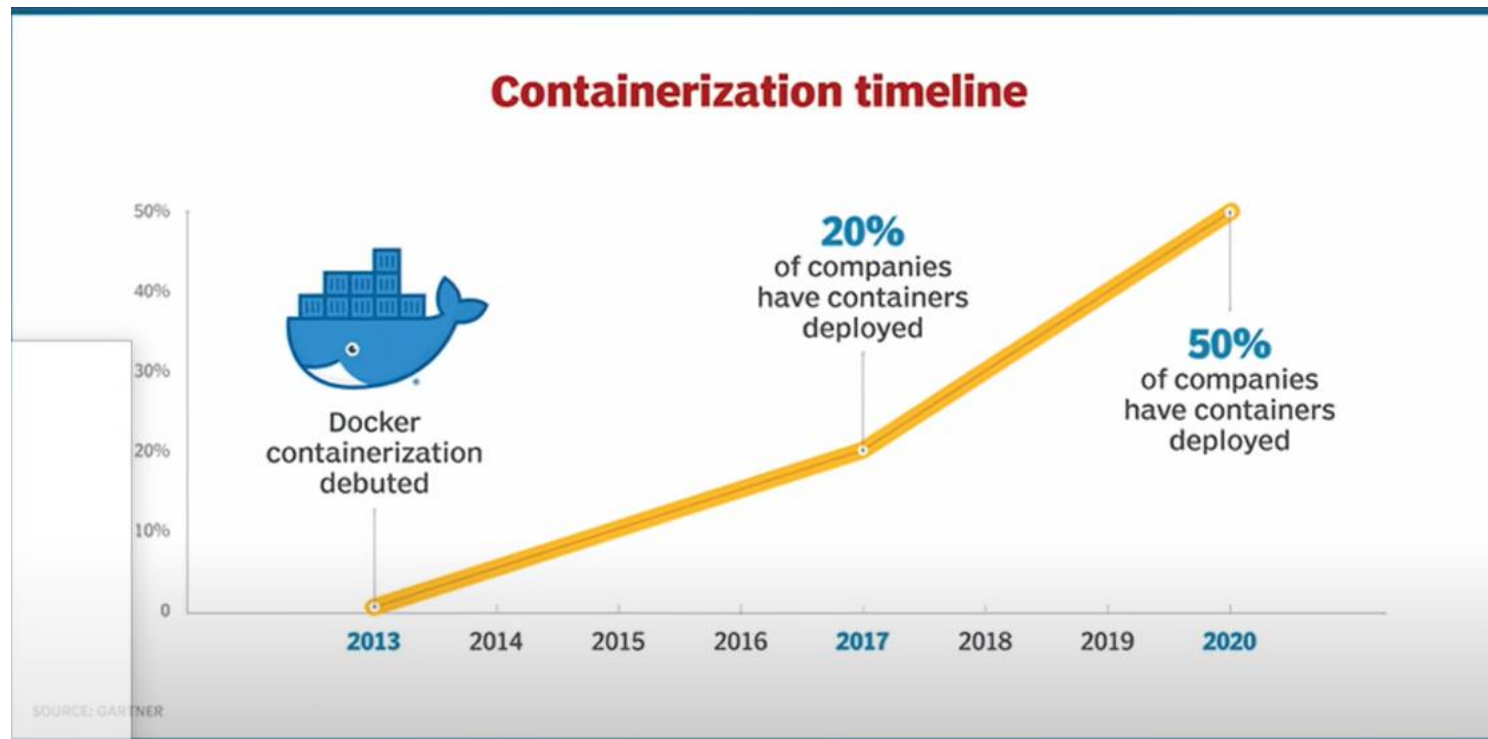


# Контейнерная виртуализация

- Контейнер это не вполне виртуальная машина
  - Об этом будет подробно рассказано завтра
- Были разные решения
  - FreeBSD Jail (2000), Virtuozzo Containers (2000), Solaris Containers (2005), Linux-VServer[en], OpenVZ (2005), LXC (2008), iCore Virtual Accounts (2008)
- Но потом появился Docker (2013)
  - и достаточно быстро стал наиболее используемым приложением для контейнеризации
  - (и завоевал мир)

# Контейнерная виртуализация

- Сейчас использование docker распространилось очень широко
  - Работа с docker - must-have навык





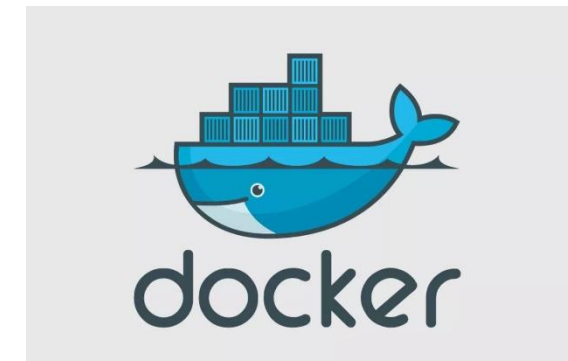
# Контейнерная виртуализация

- Почему это так популярно?
- Рассмотрим аналогию с появлением стандартных морских контейнеров
- Использование стандартных морских контейнеров позволяет в отдельных случаях снизить затраты на перевозку материалов и изделий практически вдвое.



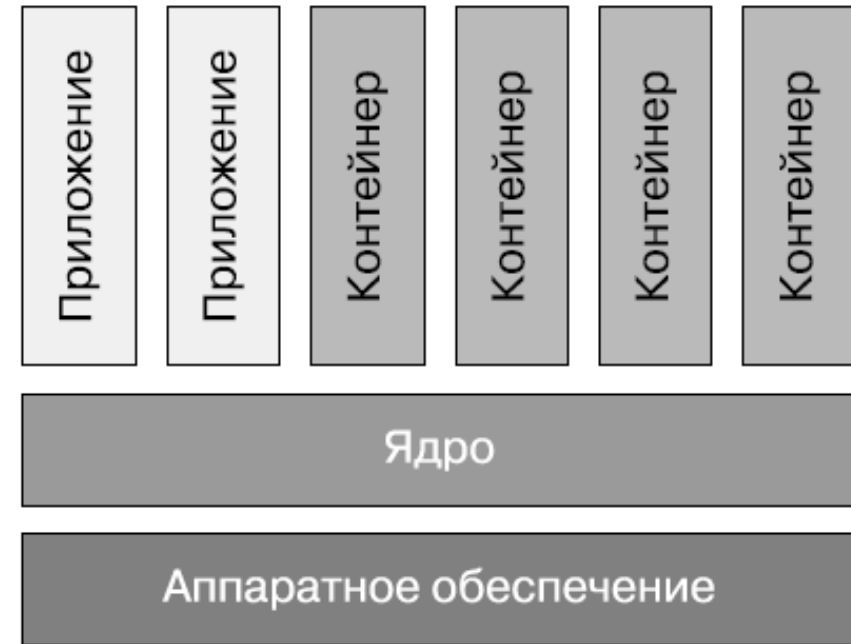
# Контейнерная виртуализация

- Использование стандартных контейнеров для упаковки программных решений имеет схожие плюсы
  - Поддержка на разных платформах
  - Возможность написать dockerfile и запустить его в любом месте, получив гарантированный повторяемый результат
  - Можно упаковать свое приложение в стандартный docker контейнер и запустить его сколько угодно раз
  - Можно развернуть, например, в docker compose



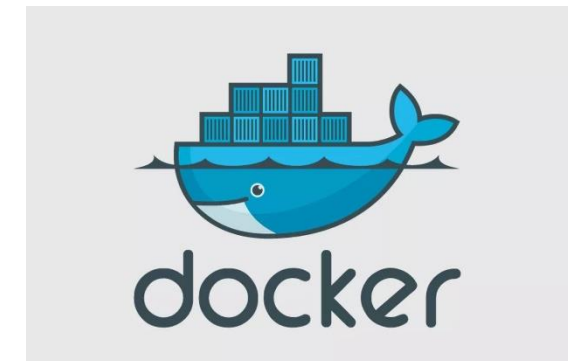
# Контейнеры vs виртуальные машины

- Виртуальные машины – диспетчер (гипервизор) выделяет ресурсы для гостевых ОС (у каждой – свое ядро)
- Контейнеры – делят одно ядро с хостом

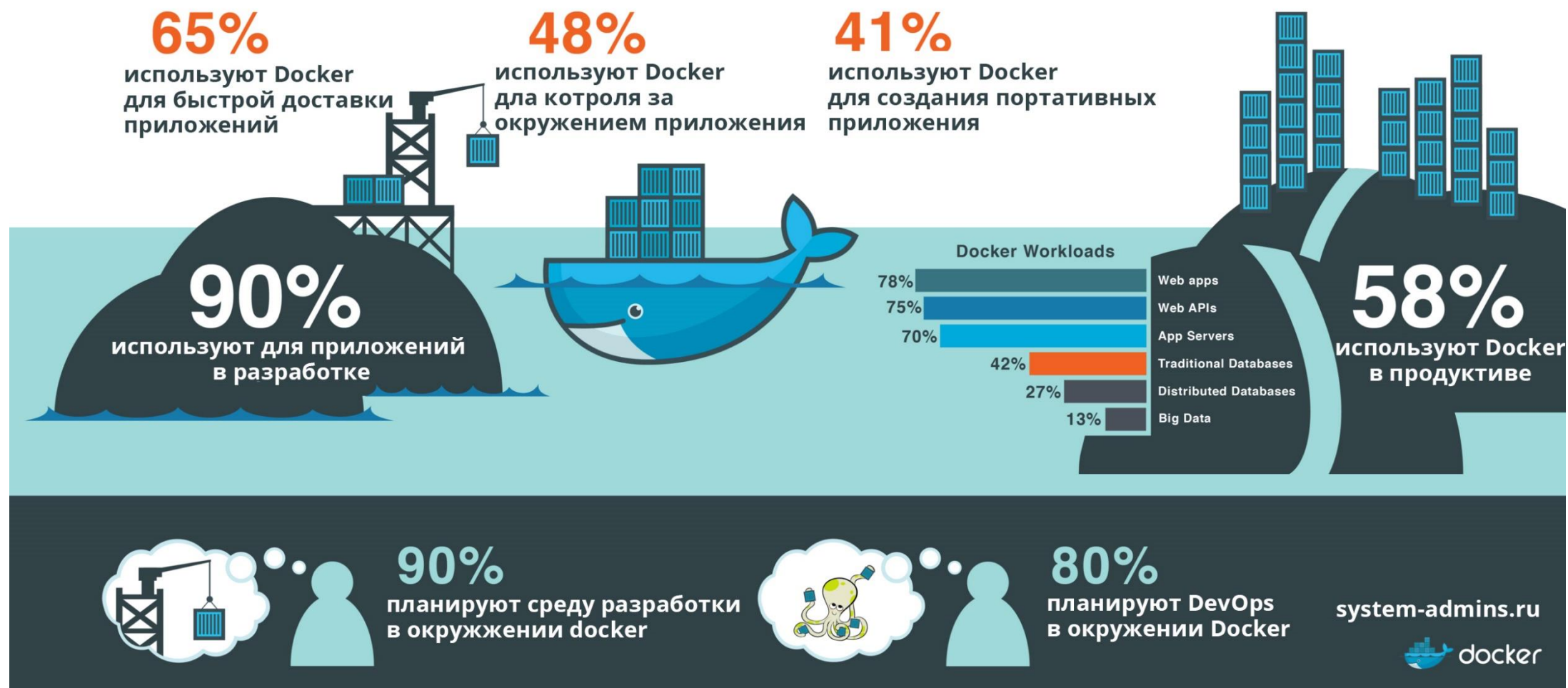


# Контейнерная виртуализация

- Использование стандартных контейнеров для упаковки программных решений имеет схожие плюсы
  - Поддержка на разных платформах
  - Возможность написать dockerfile и запустить его в любом месте, получив гарантированный повторяемый результат
  - Можно упаковать свое приложение в стандартный docker контейнер и запустить его сколько угодно раз
  - Можно развернуть, например, в docker compose



# Контейнерная виртуализация



# Оркестрация

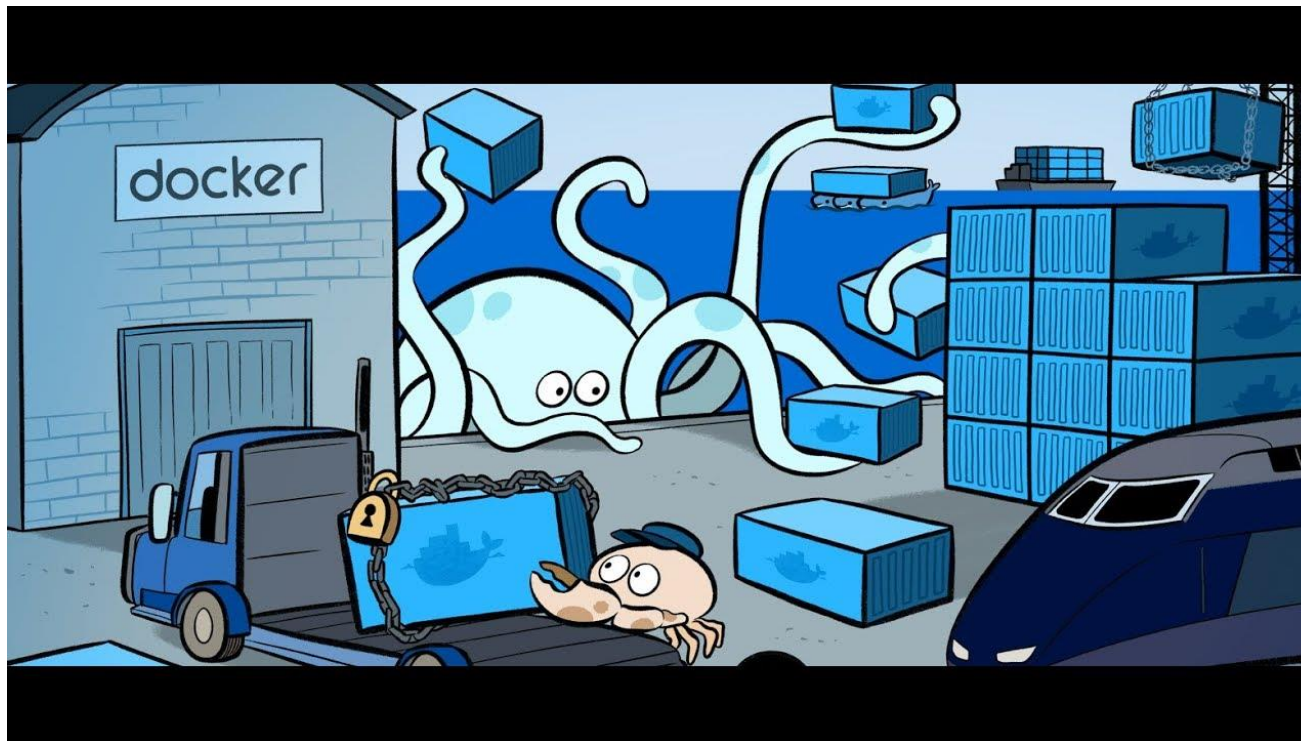
- Управление большим количеством контейнеров
  - На самом деле большим – сотнями, тысячами
  - Контроль их состояния, выделение ресурса, перезапуск, балансировка, развертывание дополнительных серверов (масштабирование) и т.п.
- Для этого так же есть специализированные решения
- Прежде всего - Kubernetes





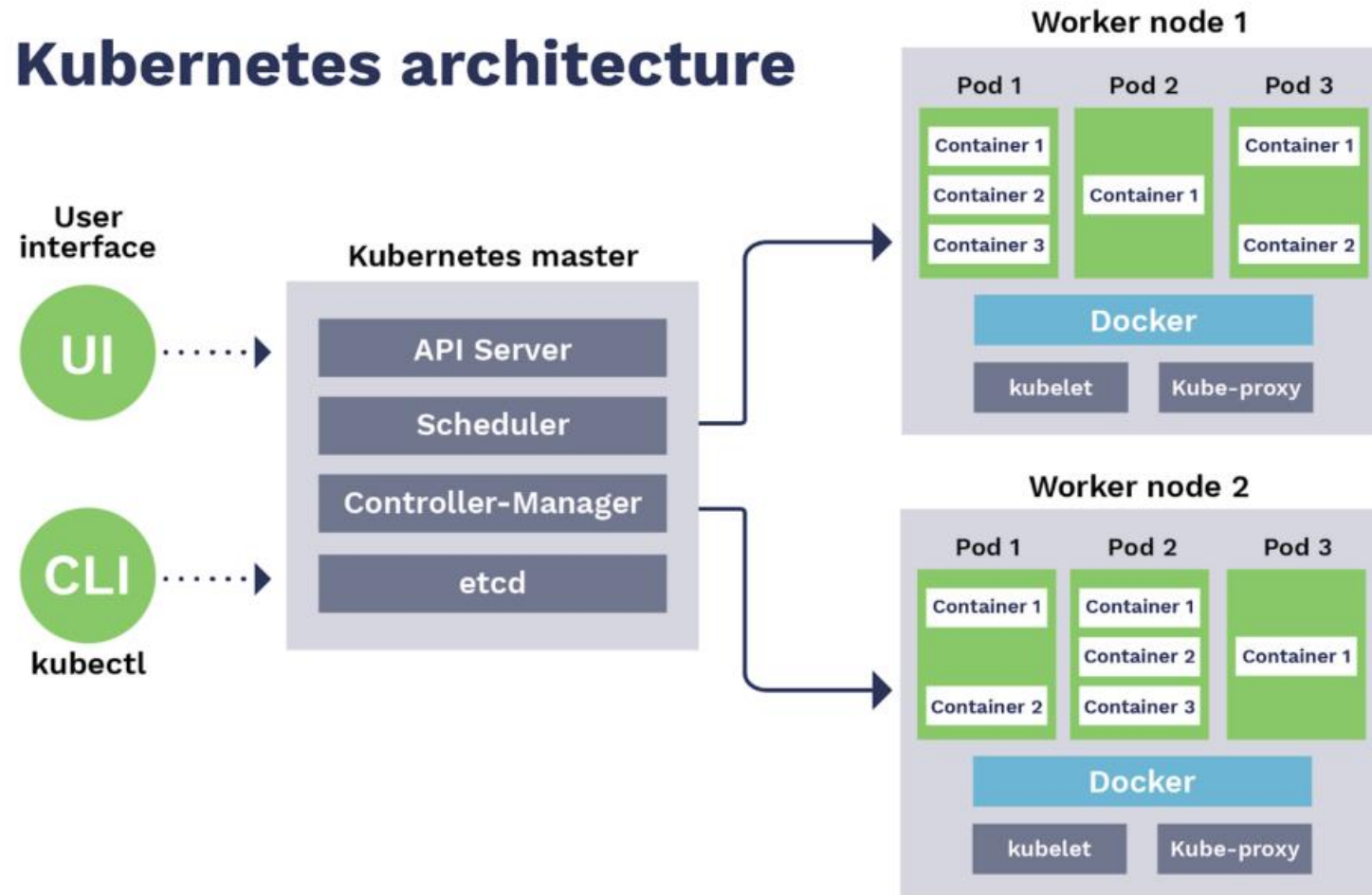
# Оркестрация

- Стандартная схема работы
  - Разрабатываем приложение
  - Упаковываем его в docker контейнер
  - Разворачиваем в Kubernetes (частный, или в публичное облако)



# Оркестрация

## Kubernetes architecture

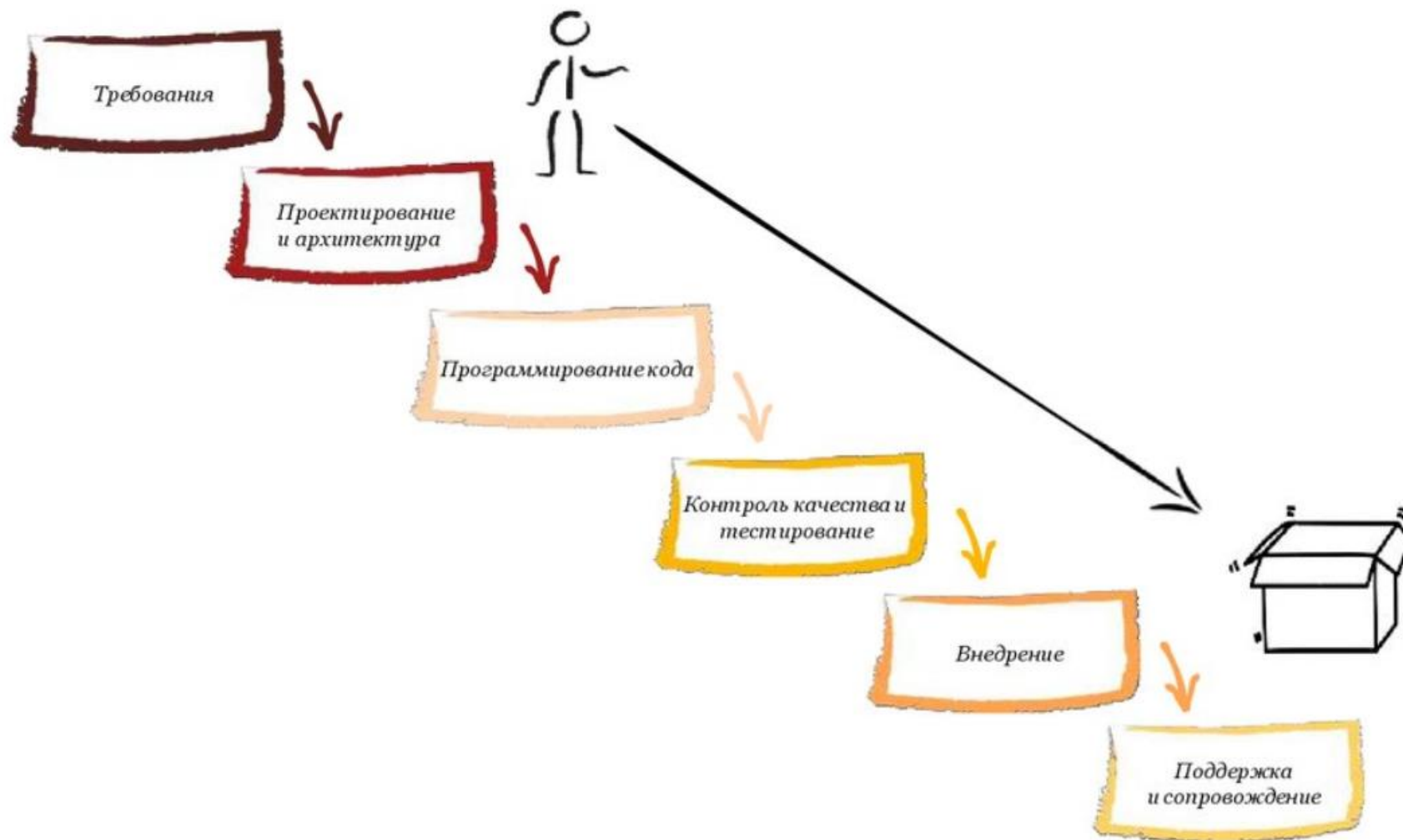




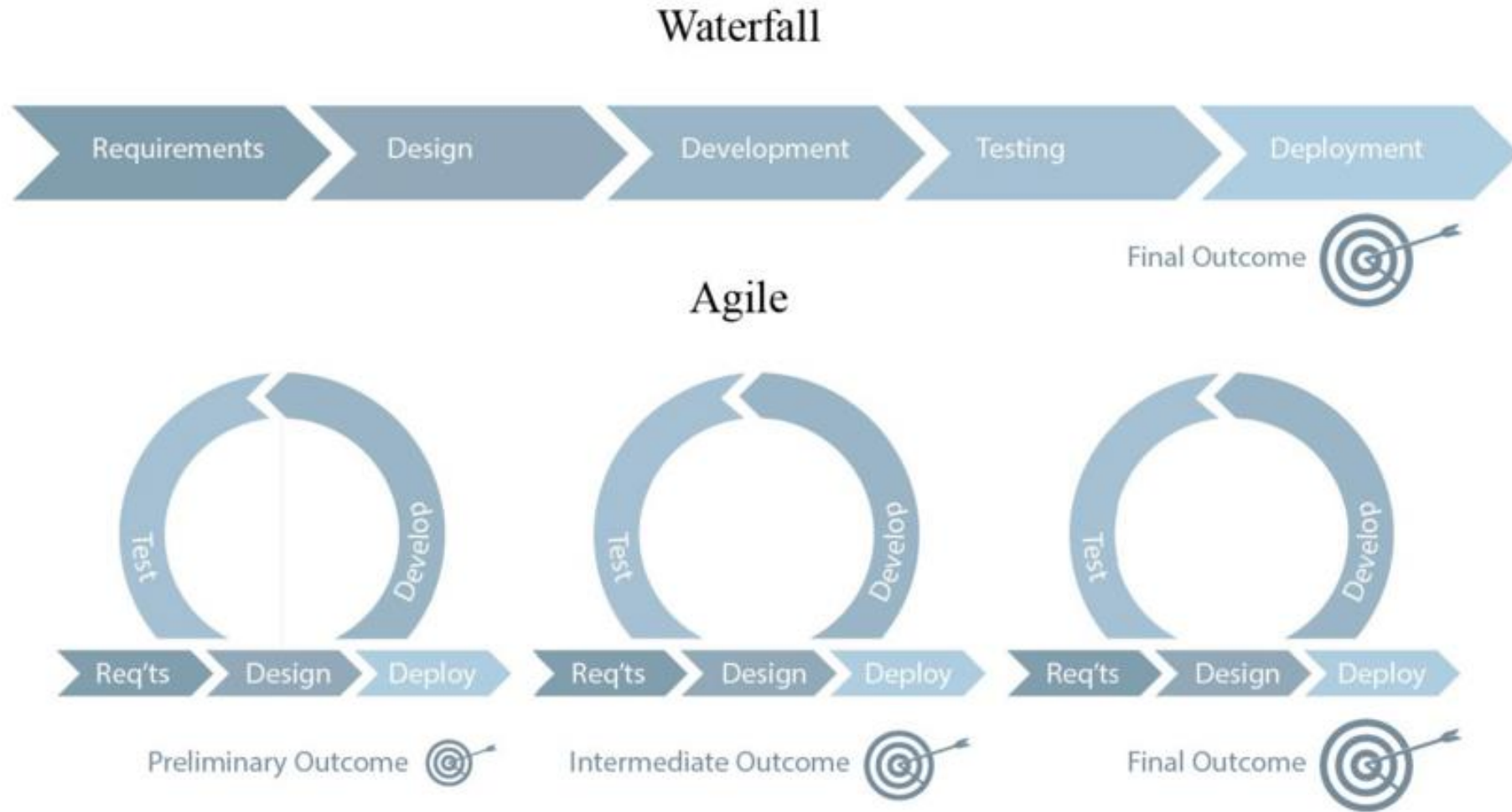
# Жизненный цикл ПО

- Помимо технологий, серьезные изменения произошли и в методологии разработки, в частности, это касается жизненного цикла разработки ПО
  - Скорее, развитие одного делало возможным изменение другого
- Известны классические стадии разработки
  - Анализ требований – проектирование – разработка – тестирование – внедрение
  - Модель «waterfall»

# Каскадный метод

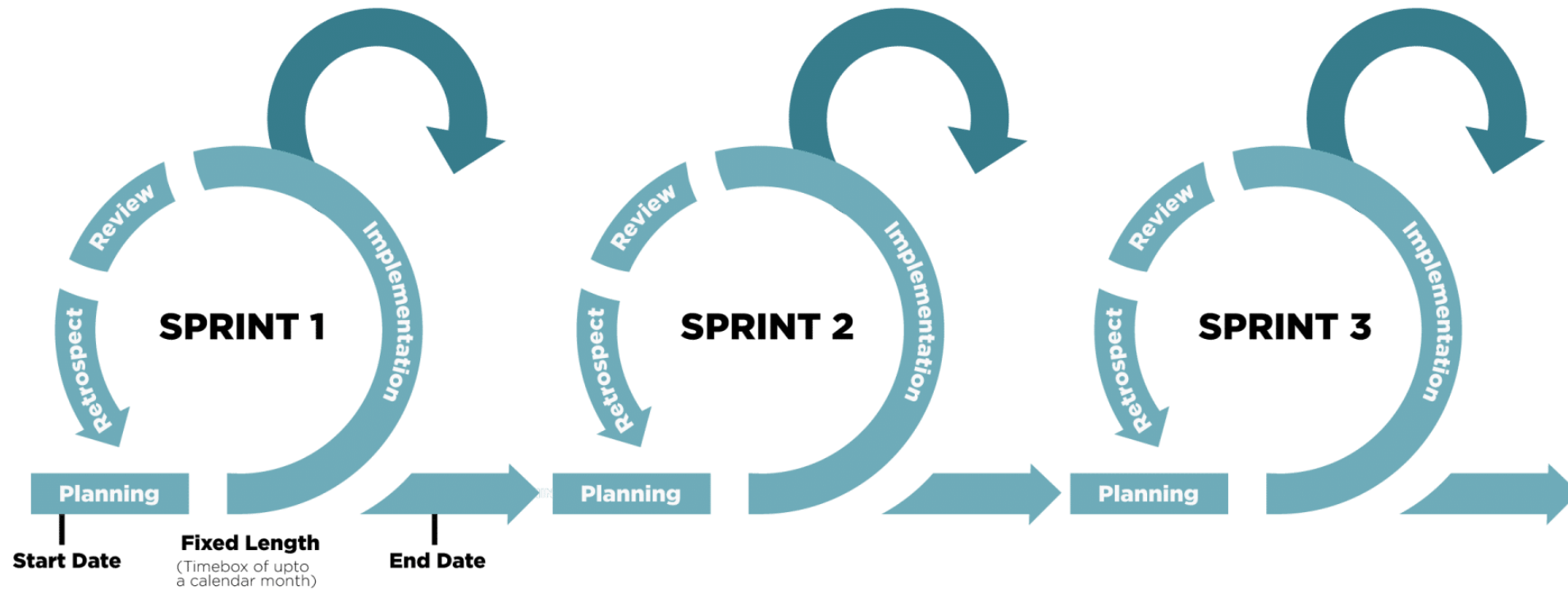


# Гибкая методология разработки



# Гибкая методология разработки

- При разработке используются спринты – короткие периоды, на выходе которых отдается готовый результат



# Гибкая методология разработки

- Такой подход, например, внедрен в проектную модель обучения
  - Вся методология работы в проектах построена по модели Agile
- Есть еще такие связанные термины такие как Scrum, Kanban
- Главное отличие – такая методология позволяет часто выпускать обновление продукта
  - Сокращается «time-to-market»
  - И это влечет за собой изменение всей организации работы

# Гибкая методология разработки

- Частая выкатка обновлений требует изменения всей структуры работы
  - Другой жизненный цикл ПО
  - Изменение культуры разработки и тестирования
    - Внедрение автоматизированного тестирования, налаживание тесного взаимодействия между development и operations командами
  - Внедрение средств автоматизации при всех стадиях создания приложения
- И мы постепенно подходим к continuous integration, continuous delivery

# Автоматизация DevOps

- При частой выкатке обновлений, невозможно продолжать делать все операции вручную
  - Главное это не нужно
- Несколько тезисов
  - Лень и изобретательность – двигатель прогресса
    - Зачем повторять рутинные операции, если их можно делать автоматически
  - Все, что может быть формализовано, может быть автоматизировано
    - Процедуры сборки, упаковки, доставки на сервер, тестирования, достаточно хорошо формализуются

# Автоматизация DevOps

- Важный момент – корень доверия
  - Или корень правильных исходников, настроек и т.п.
  - Что именно должно быть собрано/оттестировано/доставлено
- Иными словами, «где именно лежит правильная версия»
  - Это нужно знать, чтобы знать что именно и когда разворачивать в продакшн, как откатываться, если все упало и т.п.
- И тут на помощь приходит система контроля версий
  - При условии правильного выстраивания workflow



# Автоматизация DevOps

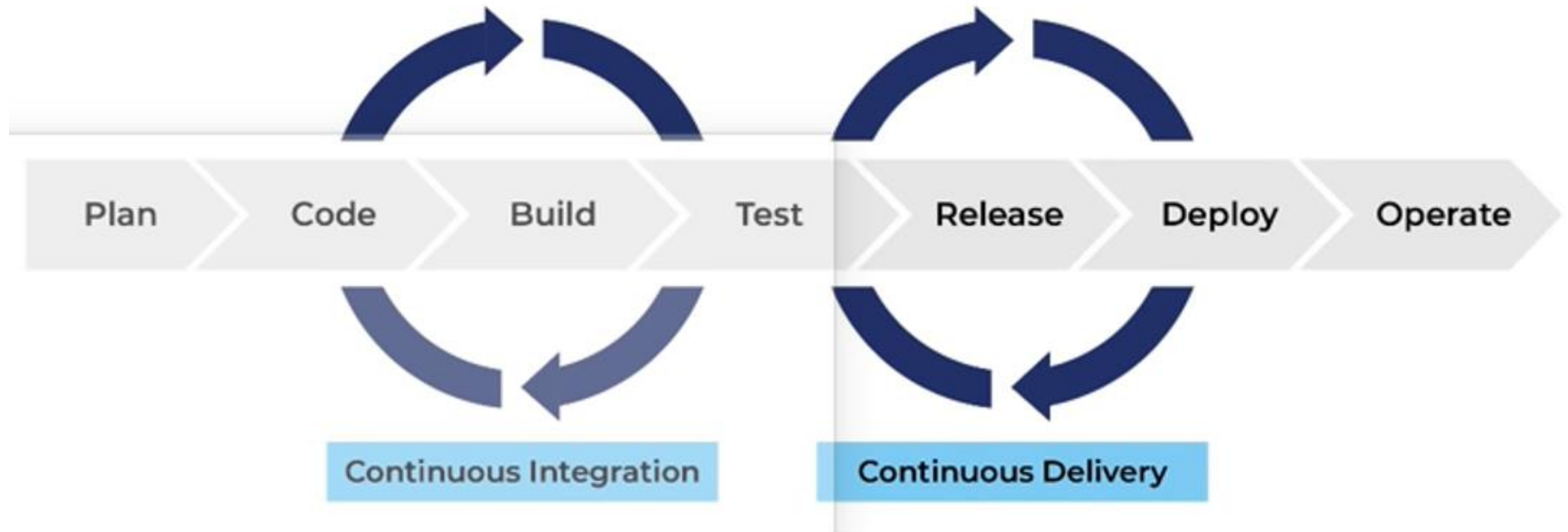
- Все кусочки пазла сошлись
  - Новая методология разработки
    - Необходимость регулярно и быстро выкатывать приложение
  - Код, который лежит в репозитории – корень доверия
    - Код, который лежит в предыдущих коммитах – предыдущие версии приложения
  - Форма передачи исходников – стандартный контейнер
  - Кластер или публичное облако – берут на себя решение задачи поддержания работоспособности
- Осталось только настроить скрипты

# CI/CD

- Скрипты – это собственно автоматизация всех рутинных процессов
  - Тут нет ничего нового, они были всегда
  - Но теперь они стали более унифицированы, появились технические средства поддержки такой автоматизации, стандартные контейнеры и т.п.
- Все это называется непрерывная интеграция и непрерывное внедрение (доставка)
  - Про них мы более подробно поговорим позже
- Это скорее не какой-то конкретный продукт, и идеология конвейера при прохождении стандартных стадий жизни кода и внедрения его в продакшн

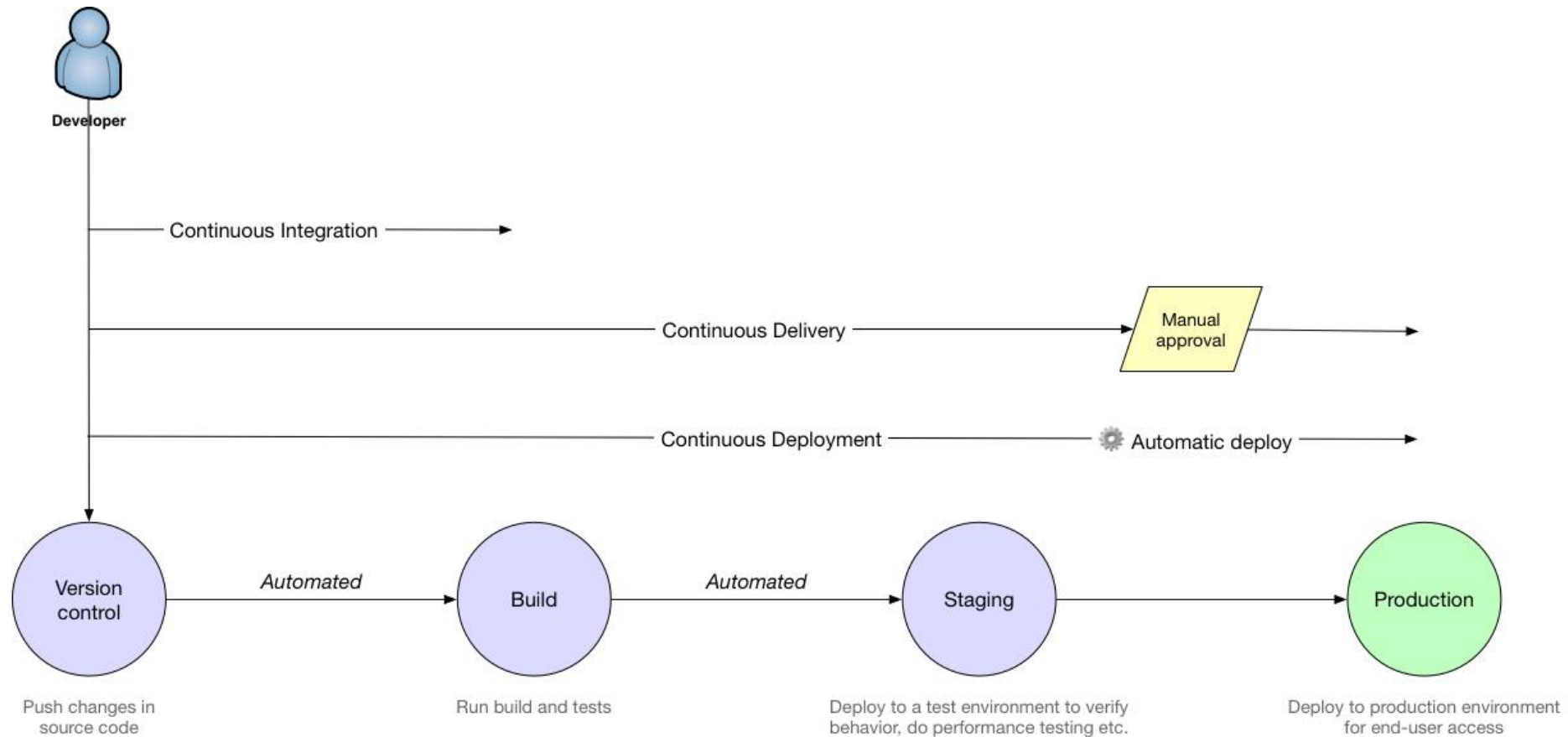
# CI/CD

- CI/CD – это собственно continuous integration (непрерывная интеграция) и continuous delivery (непрерывная доставка)



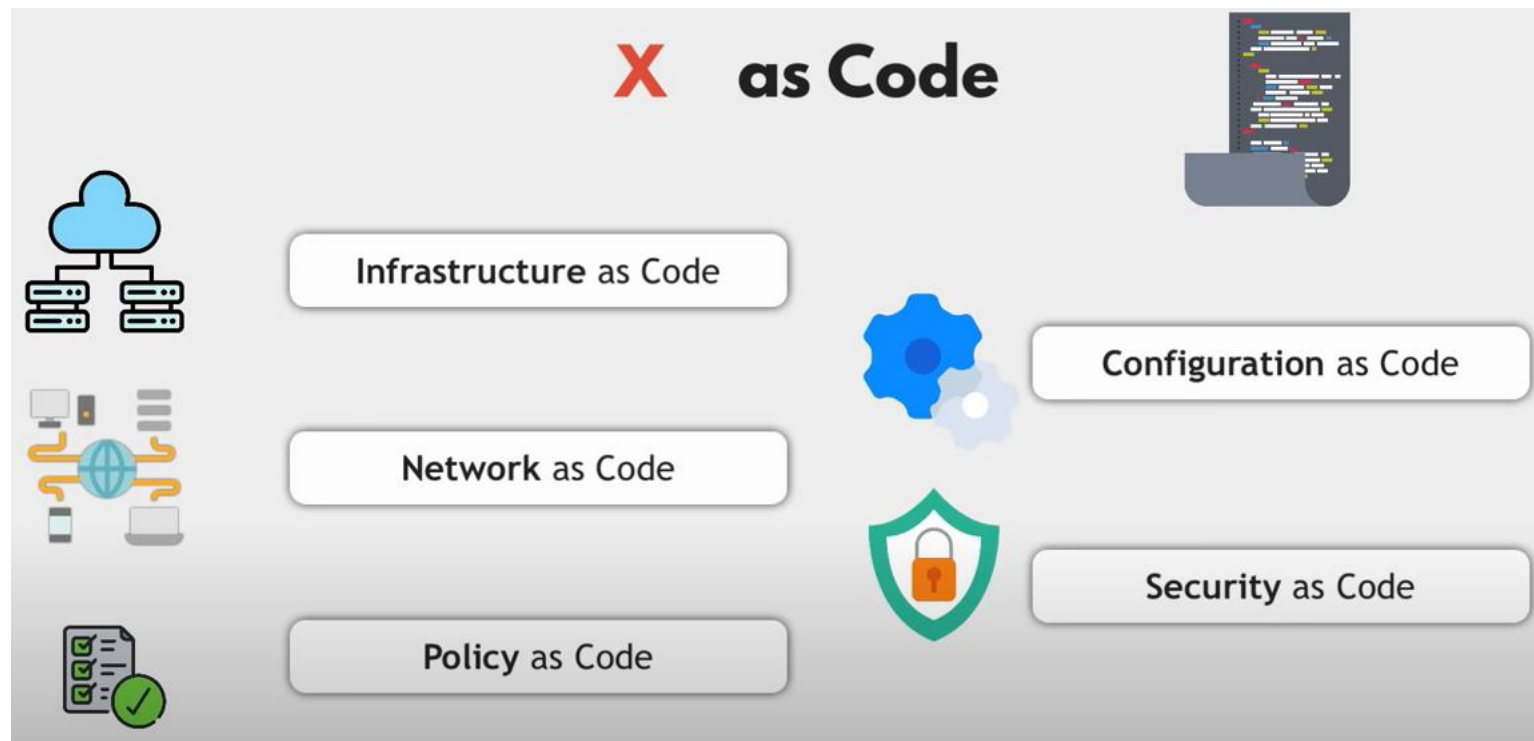
# CI/CD

- Continuous delivery / continuous deployment



# X as Code

- Все что можно формализовать – можно автоматизировать
- В git можно хранить не только исходники



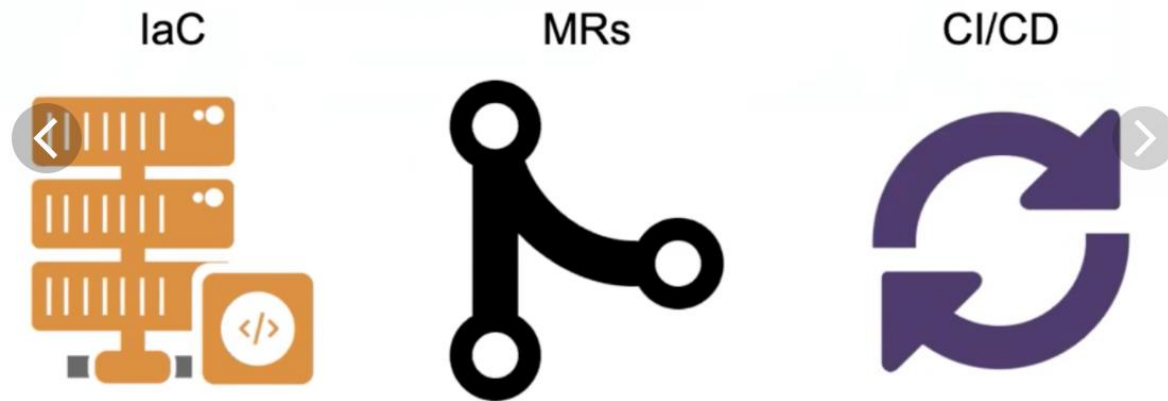
# IaC

- Сейчас очень активно развивается тема Infrastructure as Code
  - Решения вопросов управления инфраструктурой через конфигурационные файлы
  - Terraform для подготовки инфраструктуры
  - Ansible для управления конфигурацией
- Все это так же интегрируется с CI/CD



# GitOps

- Новый термин GitOps
  - **GitOps** = IaC + MRs + CI/CD
- IaC - Infrastructure as code
- MR – Merge requests
- Continuous integration and deployment



# DevSecOps

- Расширяет DevOps, фокусируется на на вопросах безопасности
  - Т.е. интегрирует вопросы безопасности в цепочки поставки CI/CD
- Задача – применять соответствующие механизмы и инструменты безопасности на каждом этапе жизненного цикла ПО, и автоматизировать этот процесс
- Это включает в себя
  - Обеспечение безопасности контейнеров
  - Безопасность рабочей нагрузки
  - Анализ безопасности (SAST, DAST и т.п.)
  - IaC
  - И др.



# Летняя школа DevOps и CI/CD

GitLab runners & job execution
































































# GitLab это не только сервер для git

- Контроль версий (исходников, конфигурации, скриптов развертывания)
- Хранилище кода
- Поддержка запросов на слияние (merge requests)
  - Поддержка gitflow
- Поддержка code reviews
- Разрешение/предупреждение конфликтов
- Управление доступом
- И много другое

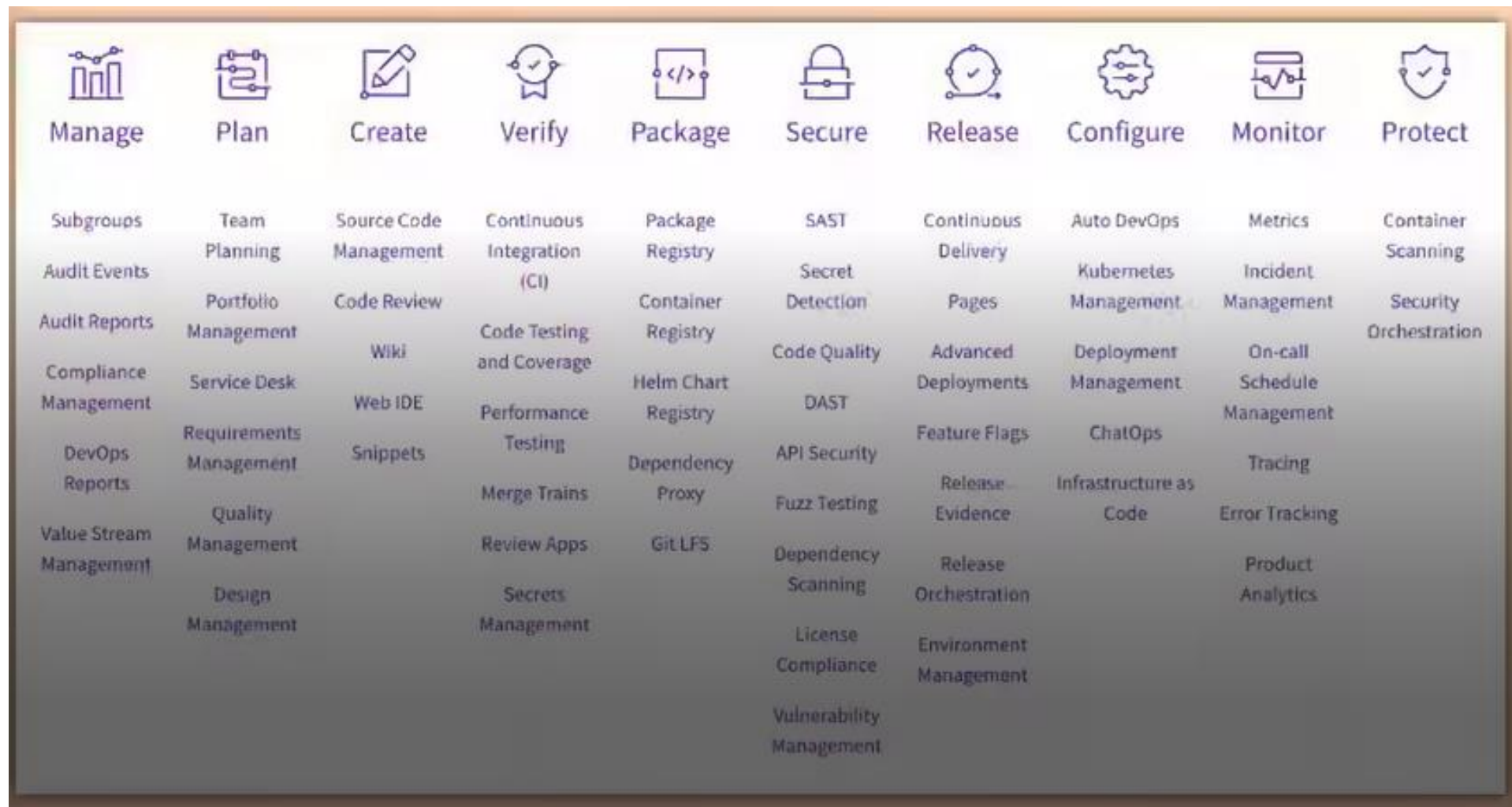
# GitLab это не только сервер для git

- Инструменты, позволяющие проводить тестирование, сборку и т.п.
- Встроенный реестр пакетов
- Встроенные средства безопасности (такие как SAST, DAST)
- Интегрированное решение – DevOps platform

# DevOps platform

| <br>Manage | <br>Plan | <br>Create | <br>Verify | <br>Package | <br>Release | <br>Configure | <br>Monitor | <br>Secure |
|---|---|---|---|--|--|--|--|---|
|            |          |            |            |             |             |               |             |            |
|            |          |            |            |             |             |               |             |            |
|            |          |            |            |             |             |               |             |            |
|           |         |           |           |            |            |              |            |           |
|          |        |          |          |           |           |             |           |          |
|          |        |          |          |           |           |             |           |          |

# DevOps platform



# GitLab

- Варианты развертывания
  - SaaS – gitlab.com
  - Self-managed (как git.miem.hse.ru)

| type of difference              | GitLab SaaS  | GitLab self-managed                                  |
|---------------------------------|--|--|
| Infrastructure                  | GitLab manages HA Architecture, and instance-level backups, recovery, and upgrades | manage your own, anywhere                            |
| Instance wide settings          | same for all users   | custom   |
| Access controls                 | customer is group owner  | customer is admin                                    |
| Features availability, such as: | <a href="#">SAML SSO</a> is Premium  | <a href="#">SAML</a> or <a href="#">LDAP</a> is Core |
| Log information and auditing *  | no access, but Support or Security can answer questions                            | unrestricted access                                  |
| Reporting, DevOps adoption      | Group and project level DevOps adoption reports                                    | Usage trends, instance-level DevOps adoption reports |

# GitLab architecture

- Рассмотрим, как устроен GitLab с точки зрения выполнения заданий
- Мы помним, что GitLab это комплексное приложение с множеством функций, сейчас интересует именно job execution
- Вне зависимости от типа установки (облачная или self-managed), у GitLab будут следующие элементы:
  - Сервер GitLab
  - Специальные хосты для выполнения заданий – раннеры (runners)

# GitLab server

- Главный компонент
- На нем хранится вся конфигурация
  - В том числе, конфигурация pipeline
- Он управляет запуском заданий – ставит их на выполнение
- На сервере хранятся результаты выполнения заданий
- Однако сам он задания не выполняет, а передает их специальному узлу – раннеру
  - Если точнее, раннер сам забирает готовые задачи по мере их появления



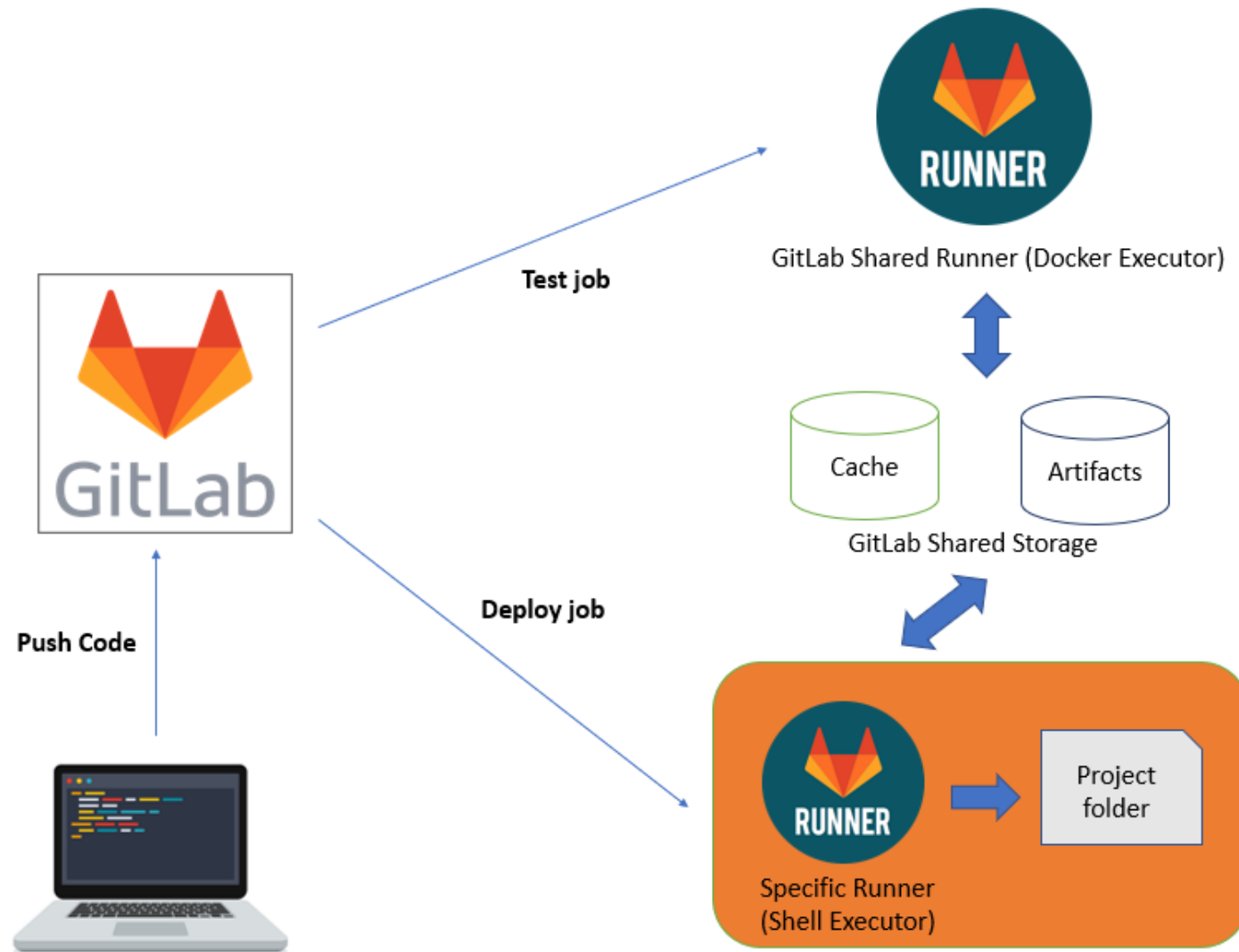
# GitLab runner

- Специальное приложение, которое работает совместно с GitLab CI для выполнения заданий
- Запускается и берет задание в работу в нужный момент
- Можно настроить необходимое количество раннеров в настройках GitLab
- Могут быть разные типы раннеров под разные задачи
- Это достаточно простое приложение, которое может быть установлено на разные ОС
  - Linux, Windows, Mac

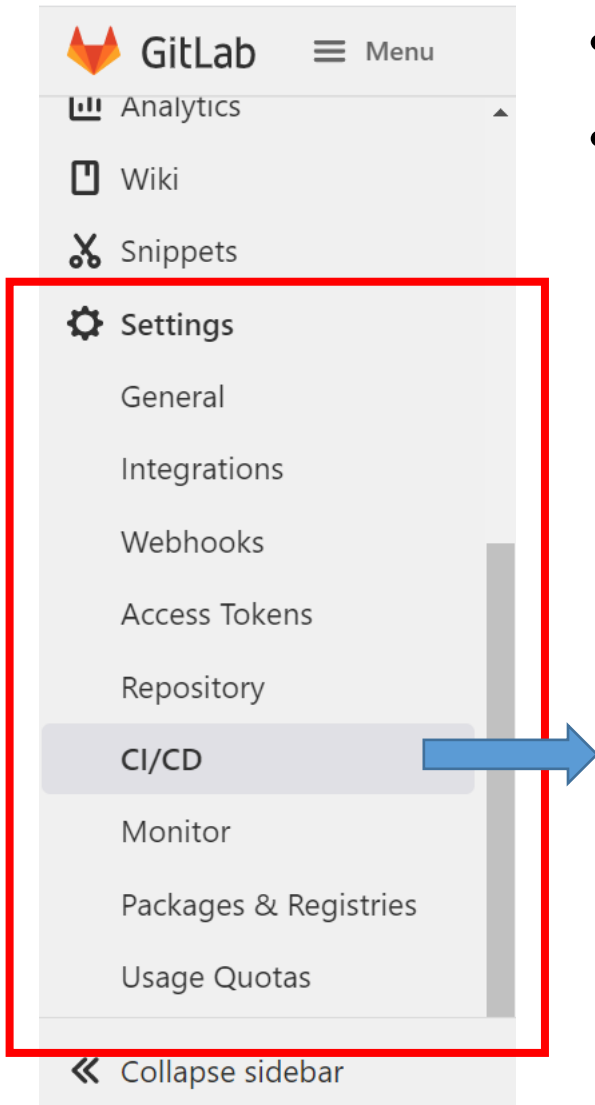
# GitLab runner

- Есть несколько видов раннеров с точки зрения доступности
- Shared runner
  - Доступен из любого проекта данного сервера GitLab
  - Доступен всем пользователям данного GitLab на конкурентной основе
  - Настраивается для экземпляра GitLab в целом
- Specific runner
  - Доступен для данного проекта только
- Group runner
  - Доступен для всех проектов конкретной группы

# GitLab runner



# GitLab runner



- Как узнать, какие раннеры доступны для проекта?
- Раздел Settings => CI/CD => Runners

## Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. [How do I configure runners?](#)

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine. Runners are either:

- **active** - Available to run jobs.
- **paused** - Not available to run jobs.

## Specific runners

These runners are specific to this project.

Set up a specific Runner for a project

1. [Install GitLab Runner and ensure it's running.](#)
2. Register the runner with this URL:

`https://git.miem.hse.ru/`

## Shared runners

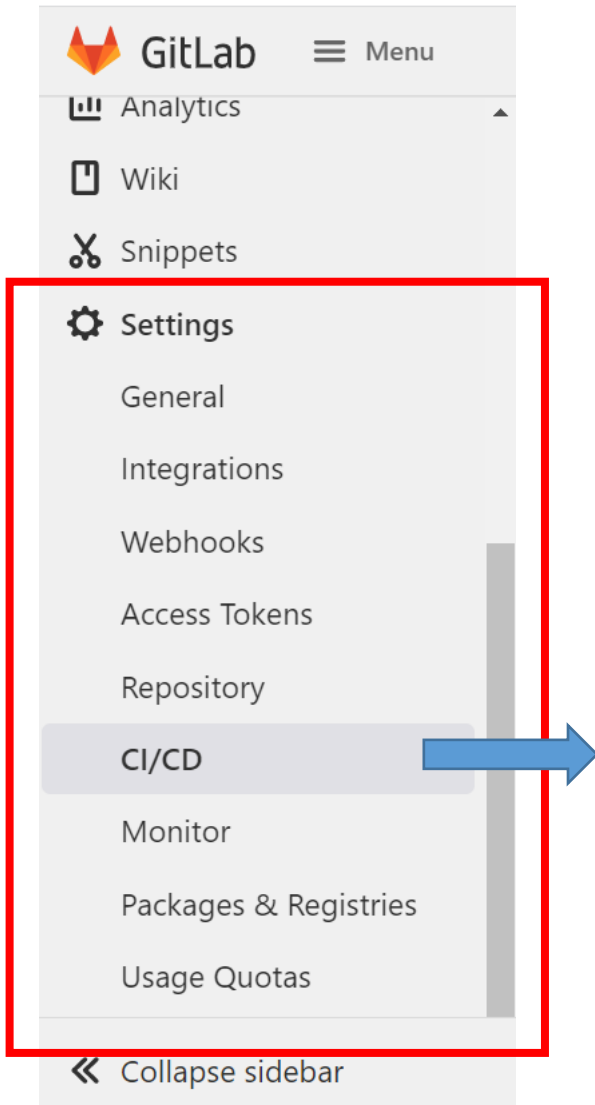
These runners are shared across this GitLab instance.

The same shared runner executes code from multiple projects, unless you configure autoscaling with [MaxBuilds](#) set to 1 (which it is on GitLab.com).

Enable shared runners for this project



# Shared runners



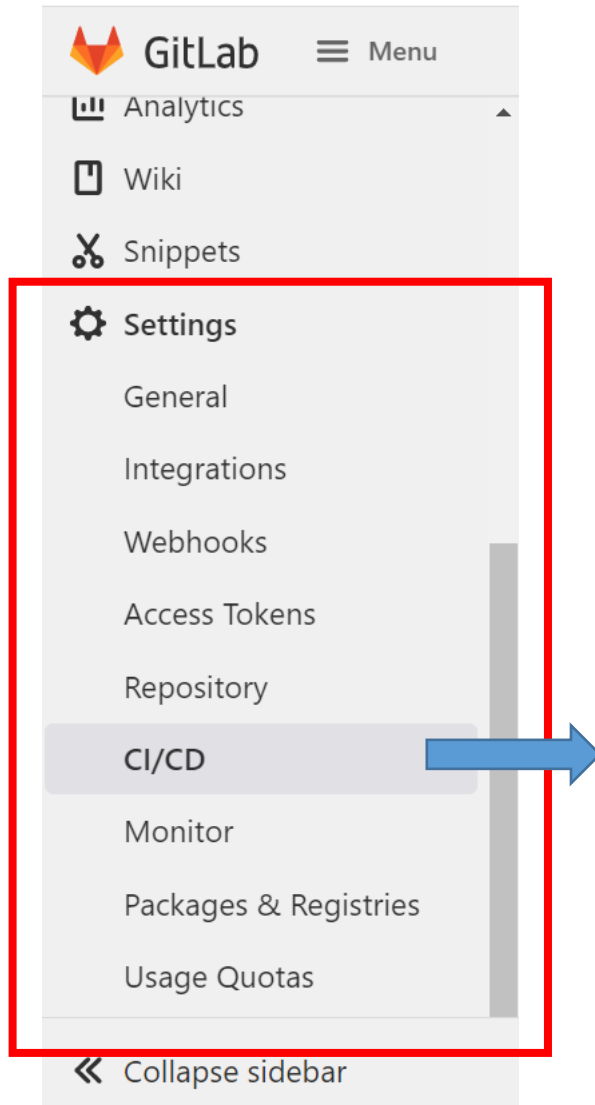
Раздел Settings => CI/CD => Runners

## Available shared runners: 4

- #73 (ZAM4rVjE)  
shared-runner-2-2  
[docker](#)
- #55 (9ZzsxiHx) 🔒  
shared cluster  
[docker](#)
- #60 (1qKTCpzJ) 🔒  
shared-cluster2  
[video-lab-miem](#)
- #72 (6ZBc681U)  
shared-runner-1-2  
[docker](#)

- Их настраивает администратор GitLab, но можно использовать
- Доступные раннеры видны в секции available shared runners

# Specific runners



Раздел Settings => CI/CD => Runners

## Available specific runners


|   |   |
|---|---|
| <div><div>—</div><div>#94 (KPPnACsW)</div><div>🔒</div></div> <div>vbox-ubuntu-docker</div> <div><div>docker-linux</div><div>my</div></div>  | <div><div>✎</div><div>⏸</div><div>Remove runner</div></div> |
| <div><div>●</div><div>#93 (q6qR52MA)</div><div>🔒</div></div> <div>my-windows-runner-2</div> <div><div>my</div><div>shell-runner</div></div> | <div><div>✎</div><div>⏸</div><div>Remove runner</div></div> |
| <div><div>●</div><div>#91 (C8a9KJPf)</div><div>🔒</div></div> <div>my-win-runner</div> <div><div>docker-run</div><div>my</div></div>         | <div><div>✎</div><div>⏸</div><div>Remove runner</div></div> |

- Можно настроить для конкретного проекта
- Достаточно установить приложение-агент на доступный хост
- Мы позже посмотрим, как установить и настроить раннер для своего проекта

# GitLab runner

- Runner это отдельные машины, их нужно настраивать и добавлять в настройках
- Смотрим, какие раннеры нам сейчас доступны
  - Надо зайти в раздел Settings -> CI/CD
  - Выбираем нужный (по тегу)
- В настройках задания надо указать тег нужного раннера

```
7  ∨ job:
8    stage: build
9    ∨ tags:
10   - docker
11  ∨ script:
12    # provide a shell script as argument for this keyword.
13    - echo "Hello World"
14
```



## Available shared runners: 4

● #55 (9ZzsxiHx) 🔒

shared cluster

docker

docker\_cache\_3

docker\_miem

● #12 (7dY1o7Pa) 🔒

shared runner 1

docker

docker\_cache\_1

docker\_itsoft

● #60 (1qKTCpzJ) 🔒

shared-cluster2

video-lab-miem

# Практика – установка своего runner

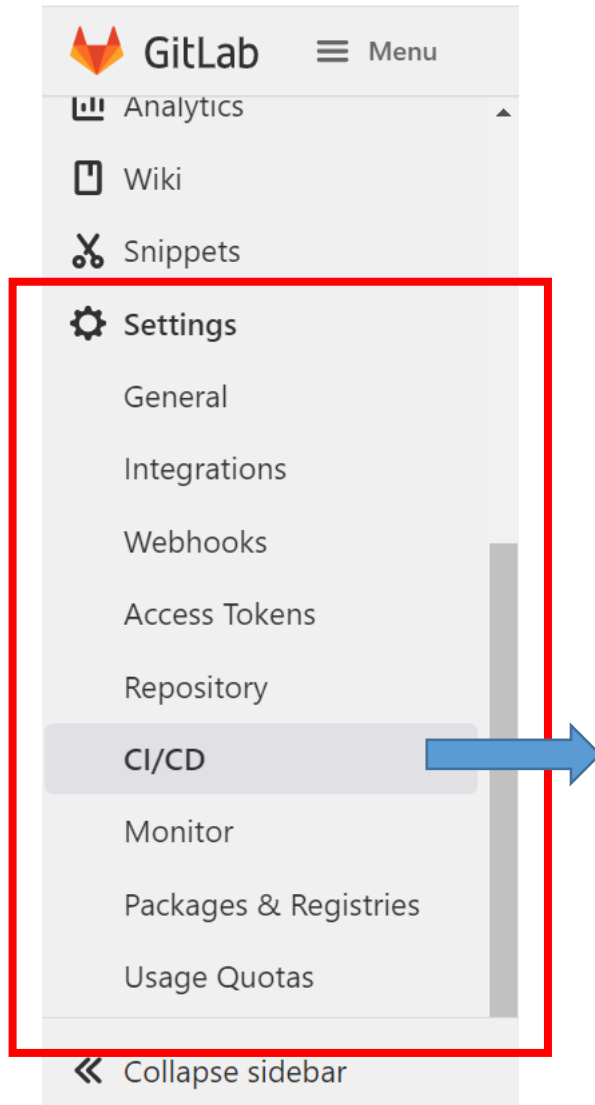
- Сейчас базовые раннеры – на основе ОС Linux
  - И чаще всего, это раннеры типа docker
- Но вообще, может быть установлен на разных ОС
  - Windows, Mac
- Установим и настроим свой раннер, и зарегистрируем его в проекте
  - Мы сделаем раннер в docker, т.е. у вас уже должен стоять docker
  - Мы сделаем раннер для группы



# Регистрация своего runner

- Поскольку мы работаем на gitlab.com, выключим использование shared runners
  - Чтобы не пришлось указывать реквизиты карты
  - Использование раннеров на платформе GitLab может быть платным
  - Поэтому мы используем свой раннер
- Если делать собственную установку gitlab и свои shared runners, ограничений нет
- Раннер можно зарегистрировать для конкретного проекта или для группы проектов

# Регистрация своего runner



Раздел Settings => CI/CD => Runners

## Available specific runners

|   |   |
|---|---|
| <div><div>—</div><div>#94 (KPPnACsW)</div><div>🔒</div></div> <div>vbox-ubuntu-docker</div> <div><div>docker-linux</div><div>my</div></div>  | <div><div>✎</div><div>⏸</div><div>Remove runner</div></div> |
| <div><div>●</div><div>#93 (q6qR52MA)</div><div>🔒</div></div> <div>my-windows-runner-2</div> <div><div>my</div><div>shell-runner</div></div> | <div><div>✎</div><div>⏸</div><div>Remove runner</div></div> |
| <div><div>●</div><div>#91 (C8a9KJPf)</div><div>🔒</div></div> <div>my-win-runner</div> <div><div>docker-run</div><div>my</div></div>         | <div><div>✎</div><div>⏸</div><div>Remove runner</div></div> |

- Если все прошло успешно, мы увидим новый раннер в настройках проекта

Спасибо