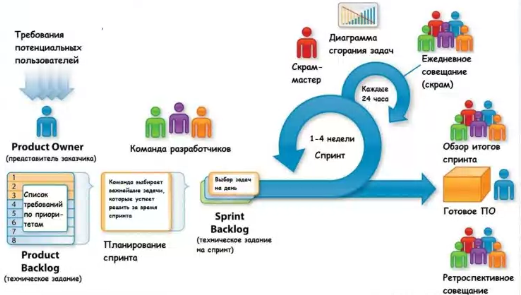
Каскадные модели разработки

Waterfall – каждый этап один за другим – нельзя вернуться назад

Этапы циркулируют только после завершения последнего этапа



Гибкие модели разработки



SCRUM – в конце спринта имеется готовый продукт.

Гибкое изменения требования.

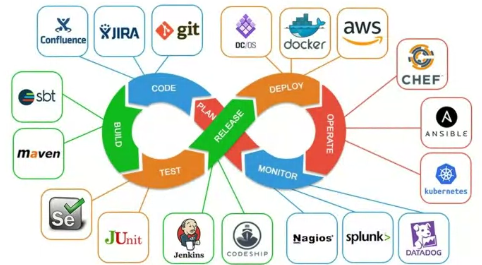
Быстрое реагирование на изменение трендов.

Бюджет не фиксирован.

Основные этапа разработки ПО

1. Анализ
2. Разработка
3. Сборка
4. Тестирование
5. Развертывание
6. Релиз (деплой)
7. Сопровождение
8. Мониторинг

Понятие DevOps



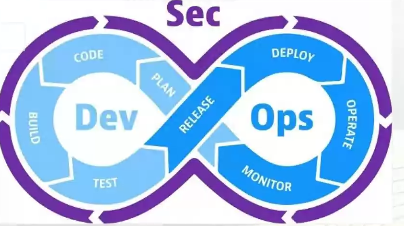
DevOps

Бесперебойная поставка ПО с помощью непрерывной интеграции рабочих процессов

Ускорение этапов разработки

Снижение ошибок, связанных с человеческим фактором

Развитие DevOps – DevSecOps



Интегрирует безопасность в процессы разработки и автоматизации

Концепции составляющие DevOps

Инфраструктура как код

Методология, которая предлагает использовать архитектурный подход к управлению инфраструктурными ресурсами, а именно автоматизировать создание и управление инфраструктурными компонентами с помощью программного кода.

Основная идея заключается в том, что вся инфраструктура, такая как виртуальные машины, сетевые настройки, хранилища данных и другие ресурсы, описывается в виде кода, который может быть управляем исходными файлами, хранящимися в системе контроля версий.\

CI/CD

Методология разработки программного обеспечения, целью которой является автоматизация и ускорение процесса разработки, тестирования и доставки программного продукта.

Цель CI/CD состоит в том, чтобы обеспечить быстрое время от идеи до развертывания новых функций или исправлений ошибок, с учетом меньшей вероятности ошибок и конфликтов, а также поддержания высокого уровня качества кода и процесса разработки.

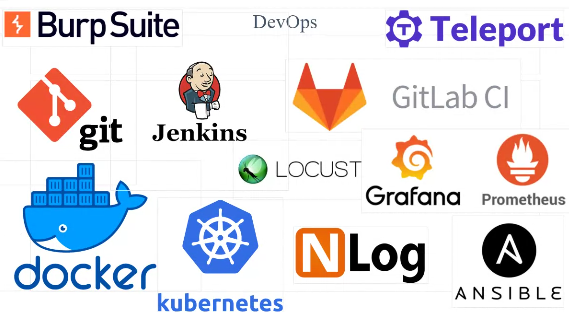
Концепция CI/CD

Состав

- Непрерывная интеграция (Continuous Integration)

- Непрерывная доставка (Continuous Delivery) (Автоматическое)

- Непрерывное развертывание (Continuous Deployment) (Ручное)



Лекция № 2

Система контроля версий Git.

Работа с GitLab и GitHub

Система контроля версий – это специальное программное обеспечение, которое используется для управления изменениями в файловой системе, отслеживание и контроля версий документов или кода программы. Она позволяет разработчикам и команде проекта работать на файлами вместе, отслеживать изменения, управлять их версиями и восстанавливаться к предыдущим состояниям проекта.

Преимущества применения системы контроля версий

1. История изменений
2. Контроль версий
3. Коллективная работа
4. Ветвление и слияние
5. Отслеживание ощибок

Распределенная VS Централизованная системы контроля версий

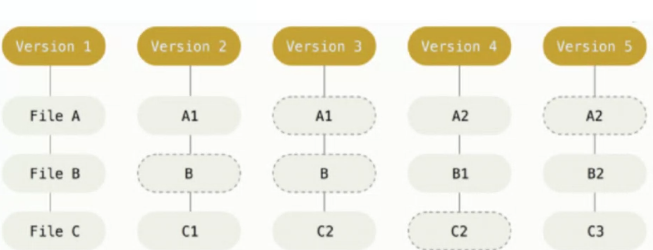
Архитектура: Централизованная система контроля версий имеет единую центральную базу данных, где хранятся все файлы и изменения. В распределенной системе каждый пользователь имеет собственную копию репозитория, включая всю историю изменений.

Работа в офлайн-режиме: Распределенная система позволяет работать в офлайн-режиме, так как каждый пользователь имеет полную копию репозитория. В централизованной системе требуется подключение к центральному серверу для выполнения большинства операций.

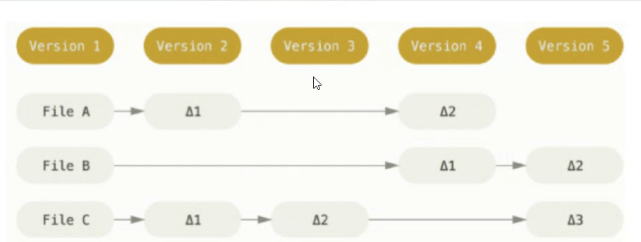
Управление правами доступа: В централизованной системе управление правами доступа к репозиторию обычно происходит на уровне центрального сервера. В распределенной системе каждый пользователь может выполнять операции с собственной копией репозитория, что позволяет более гибко управлять правами доступа.

Скорость работы: Распределенная система обычно быстрее выполняет операции, так как все данные хранятся локально у каждого пользователя. В централизованной системе операции происходят через удаленное соединение, что может быть медленнее при большом количестве пользователей.

Система контроля версий с механизмом снимков (например, Git, Mercurial) хранят все версии файлов и директорий в форме полных копий (снимков) состояния проекта в определенный момент времени. Когда вы делаете коммит, система фиксирует все измененные файлы и создаст новый снимок, который включает в себя полные копии всех измененных файлов. Это позволяет быстро переключаться между разными версиями проекта и осуществлять обход старых коммитов.



Система контроля версий со списком изменений (например, Subversion) хранят только разницу (delta) между последующими версиями файлов. Когда вы делаете коммит, система сохраняет только изменения, сделанные B файлах, относительно предыдущего коммита, в виде списка изменений. На самом деле, система часто хранит несколько версий файла, чтобы быстрее обрабатывать запросы, основанные на списке изменений. При переключении на предыдущие версии проекта, СКВ применяет все изменения последовательно, чтобы восстановить запрошенную версию проекта.



Коммит

Коммит (commit) представляет собой операцию, при которой изменения в файловой системе, внесенные программистом, сохраняются в репозитории проекта. Коммит фиксирует изменения в файле или наборе файлов, а также добавляет комментарии или описания к этим изменениям. Каждый коммит имеет уникальный идентификатор, который позволяет отслеживать историю изменений и восстанавливать предыдущие версии файлов. Коммиты также позволяют работать с несколькими программистами над одним проектом, объединяя изменения, внесенные каждым из них.

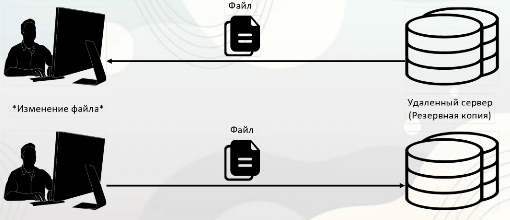


Гит – главная платформа, меркуриал – используется на азиатском рынке, сабверсион – для удобной документации, перфорс – для игр используется в основном

Система контроля версий Git

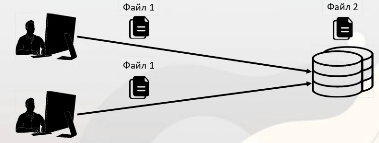
Git - это распределенная система управления версиями, которая используется для отслеживания изменений в коде программного обеспечения. Git позволяет разработчикам работать над проектом одновременно и совместно, сохраняя историю изменений и легко управляя версиями. Git также обеспечивает возможность ветвления и слияния кода, а также гибкую работу с удаленными репозиториями.



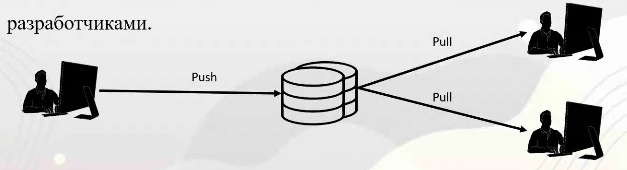


Merging – это процесс объединения различных версий файлов.

Merging может проводиться автоматически или в ручном режиме.



Git-хостинги – это платформы, которые представляют инфраструктуру для хранения и управления репозиториями Git. Они позволяют разработчикам работать с Git репозиториями удаленно, управлять версиями, отслеживать изменения и сотрудничать с другими разработчиками.



GitLab – это хостинг-платформа для хранения и управления кодом. Она предоставляет функциональность, аналогичную GitHub, но с открытым исходным кодом. GitLab может быть развернут на собственных серверах, что позволяет организациям иметь полный контроль над своими репозиториями. Он также предлагает широкий набор инструментов для управления проектами.

Лекция № 3

Docker часть № 1

Docker – это открытая платформа для разработки, доставки и запуска приложений. Он позволяет упаковать приложение и его зависимости в контейнер, который может быть запущен на любой совместимой с Docker системе. Контейнеры Docker предлагают независимую от операционной системы среду выполнения

Изолированная среда

Docker обеспечивает изолированное окружение, в котором приложение может работать независимо от других приложений и зависимостей на хостовой системе. Это позволяет избежать конфликтов и снижает вероятность неправильной работы приложения. Поэтому разработчики могут не задумываться, в каком окружении будет работать их приложение, а инженеры по эксплуатации – единообразно запускать приложения и меньше заботиться о системных зависимостях.

Управление зависимостями

Docker позволяет включать все зависимости приложения, включая операционную систему, библиотеки и другие компоненты, в контейнер. Это упрощает управления зависимостями и обеспечивает согласованность окружения между разработчиками и на разных этапах разработки и может снизить проблемы, связанные с развертыванием приложения на разных серверах.

Переносимость

Docker позволяет легко перемещать контейнеры с приложением между различными средами разработки, тестирования и продакшн. Это упрощает процесс развертывания и масштабирования приложений.

Dockerfile – это текстовый файл, содержащий инструкции для автоматического создания Docker-образа. Он описывает шаги, которые необходимо выполнить для создания контейнера Docker, включая установку и настройку приложений, копирование файлов и настройку сетей. Dockerfile позволяет разработчикам создавать повторяемые и легко воспроизводимые среды разработки и выполнения приложений, где каждый шаг указан явно и может быть автоматически выполнен при создании контейнера Docker.

Контейнеры (Containers) – это изолированные и легковесные среды, в которых запускаются приложения и их зависимости. Они представляют собой запускаемые экземпляры образов Docker, содержащих все необходимое для работы приложения. Их можно создавать, запускать, останавливать и удалять. Также можно подключать к контейнеру хранилище, объединять контейнеры одной или несколькими сетями и общаться с контейнерами, используя Docker API или командную строку.

1 контейнер = 1 приложение

Образ (Image) – это шаблон контейнеров. В образе описывается, что должно быть установлено в контейнере (например код, среда выполнения, библиотеки, переменные окружения и другие зависимости) и какие действия нужно выполнить при старте контейнера.

Используя 1 образ можно создавать много контейнеров

Кластер Docker – это совокупность компьютеров или серверов, которые объединены вместе для работы с контейнерами Docker. Кластер Docker использует контейнеры для запуска и управления приложениями на нескольких компьютерах одновременно.

Кластер обеспечивает масштабируемость

Docker Compose – это инструмент, которые позволяет запускать и управлять несколькими контейнерами Docker одновременно. Он позволяет описывать структуру и конфигурацию вашего приложения в файле YAML, что делает его очень простым для понимания и использования. Docker Compose позволяет запускать и остановить все ваши контейнеры одной командой, а также связывать их между собой, чтобы они могли взаимодействовать друг с другом.

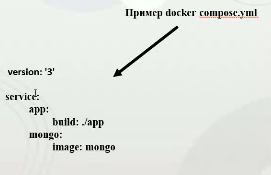
Docker демон – это основной компонент в системе Docker, которые управляет выполнением и контролем контейнеров. Он работает в фоновом режиме и управлять ресурсами, созданием и удалением контейнеров, а также назначением им ресурсов

Docker клиент – это инструмент командной строки, который позволяет управлять контейнерами Docker (через него пользователь взаимодействует с демоном). Позволяет выполнять различные операции с контейнерами, такие как создание, запуск, остановка, удаление и мониторинг, используя простые команды в командной строке.

Docker-compose

Docker Compose - это инструмент, который позволяет запускать и управлять несколькими контейнерами Docker одновременно. Он позволяет описывать структуру конфигурацию вашего приложения в файле YAML, что делает его очень простым для понимания и использования. Docker Compose позволяет запускать и остановить все ваши контейнеры одной командой, а также связывать их между собой, чтобы они могли взаимодействовать друг с другом.

Декларативное программирование – это парадигма, при которой описывается желаемый результат, без составления детального алгоритма его получения.



YAML (YAML Ain't Markup Language) - это удобный формат для хранения и передачи данных в текстовой форме. Он используется для конфигурационных файлов, данных, и другой информации, которую читает и записывает компьютерное программное обеспечение. YAML напоминает человекочитаемый текст и позволяет организовать данные в виде списков, словарей и других структур для удобства чтения и редактирования как людьми, так и компьютерами. Важно соблюдение табуляции(!!!)

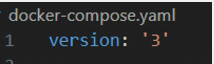


Преимущества Docker compose

* Все необходимые контейперы запускаются и выключаются одной командой (docker-compose up и docker-compose down);
* Автоматическое создание необходимых образов на основании Dockerfile каждого приложения
* Автоматическое создание изолированной сети для взаимодействия контейнеров
* С помощью DNS возможно взаимодействие между контейнерами, используя имя.

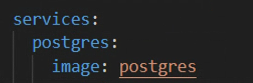
Docker compose компоненты: version

Version объявляет версию docker-compose синтаксиса

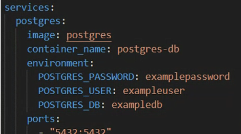


Services – определяет все контейнеры, которые вы хотите запустить с помощью docker-compose. Каждый сервис имеет имя и определенные настройки.

Image - Можно использовать существующие Docker-образы из Docker Hub или указать путь к своему Docker-образу.

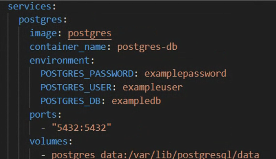


Можно настроить различные параметры для каждого сервиса, такие как переменные окружения, порты, названия контейнера и тд. Переменные окружения в Docker Compose позволяют определить и передать конфигурационные параметры в контейнеры



Depends\_on - можно указать зависимости между сервисами, чтобы гарантировать, что один сервис запускается только после запуска другого.

Volumes - Можно указать тома, чтобы сохранить данные между запусками контейнеров. Все, что находится в volumes сохраняется на диске после удаления контейнера. Остальные данные хранятся в памяти.



docker-compose up - Запуск приложения в соответствии с конфигурацией из файла docker-compose.yml. Если файл называется иначе, добавляется ключ docker-compose -f <имя файла>

docker-compose down - Остановка и удаление всех контейнеров, сетей и томов, созданных docker-compose up.

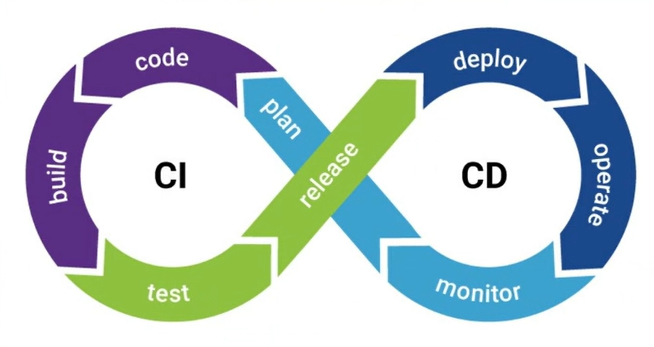
docker-compose ps - Отображение статуса контейнеров, запущенных с помощью Docker Compose.

docker-compose logs - Просмотр логов контейнеров.

docker-compose build - Построение образов, определенных в docker-compose.yml.

docker-compose exec <service-name> <command> - Выполнение команды внутри контейнера.

CI/CD



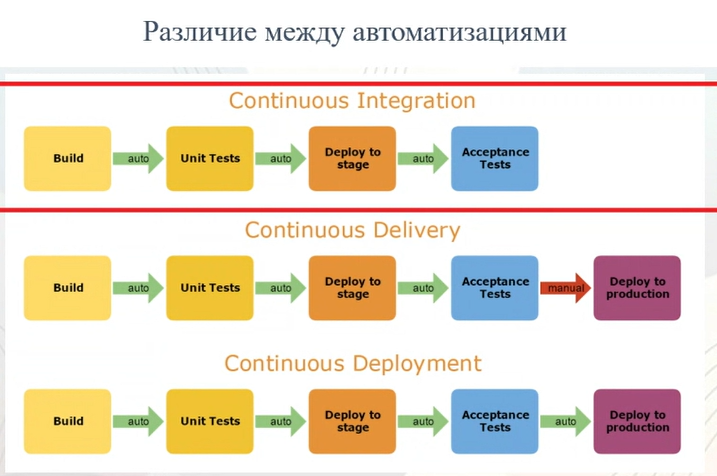
CI/CD представляет собой методологию разработки, которая стремится автоматизировать каждый шаг процесса разработки приложений.

Основная цель CI/CD - упростить и ускорить процесс интеграции кода от разработчиков и его развертывания. CI/CD помогает повысить качество и надежность приложений, минимизируя возможные ошибки во время интеграции.

Один из основных принципов CI/CD - регулярное и частое интегрирование нового кода для предотвращения конфликтов и сбоев в работе приложения.

CI/CD также предлагает автоматическую сборку И тестирование приложений, чтобы разработчики могли быстро получать обратную связь об их работе.

С помощью CI/CD команды разработчиков могут быстро выявить и исправить ошибки в коде, еще до того, как они попадут в продакшн.

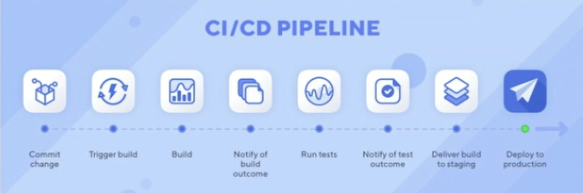


Continuous Integration (непрерывная интеграция) – это практика, заключающаяся в частом интегрировании кода разработчиков в общую базу. При использовании СІ команда разработки регулярно коммитит свои изменения в общий репозиторий, а затем система автоматически собирает и проверяет код на наличие ошибок и конфликтов. Это позволяет выявлять проблемы в коде на ранних этапах разработки и снижать время на объединение изменений OT разных разработчиков.

Continuous Delivery (непрерывная доставка) - это практика, которая позволяет автоматизировать процесс доставки приложений в производственную среду. При использовании CD каждый успешный билд проходит через набор тестов, автоматическую сборку, a затем деплоится B среду тестирования, предпринимательскую среду и, в конечном счете, приложение деплоится в продакшн в ручном формате. Это позволяет команде разработки быстро и надежно доставлять продукт пользователям.

Continuous Deployment (непрерывное развертывание) - это практика, которая позволяет автоматизировать процесс доставки приложений B производственную среду. При использовании CD каждый успешный билд проходит через набор тестов, автоматическую сборку, а затем деплоится в среду тестирования, предпринимательскую среду и, в конечном счете, приложение деплоится в продакшн в автоматическом режиме, т.е. разворачивается само. Это позволяет команде разработки быстро и надежно доставлять пользователям продукт.

Пайнлайн - это последовательность шагов или процессов, которые выполняются для достижения определенного результата.



Одна из основных целей пайплайна работы и повышение эффективности. - устранение ручной

Пайплайн может использоваться в различных отраслях, таких как ІТ, производство, маркетинг и т.д.

В ІТ пайплайн может включать этапы разработки, тестирования, развертывания и обновления программного обеспечения.

Для создания пайплайнов в ІТ можно использовать различные инструменты, такие как Jenkins, GitLab CI/CD, AWS CodePipeline и другие.