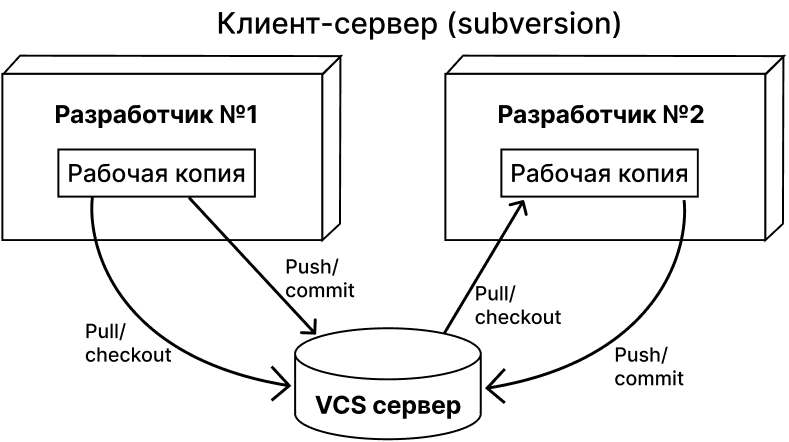
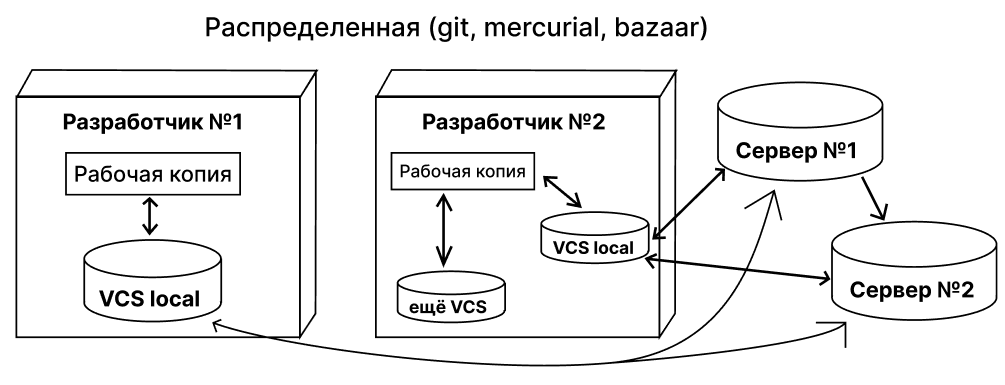
1. Системы управления версиями
2. git: centralized workflow
3. git: feature branch workflow
4. gitflow
5. Тестирование ПО. Классификация
6. Модульное тестирование
7. Многопоточность. Достоинства и недостатки
8. Многопоточность: shared state и его проблемы
9. Синхронизация в многопоточных программах
10. Многопоточность: Deadlock
11. Атомарные переменные
12. Unified Modelling Language
13. Внедрение зависимостей
14. Локализация и интернационализация
15. Model View Controller
16. Варианты MVC
17. Паттерны проектирования. Классификация
18. Паттерн одиночка
19. Паттерн фабричный метод
20. Паттерн абстрактная фабрика
21. Паттерн прототип
22. Паттерн строитель
23. Паттерн адаптер
24. Паттерн мост
25. Паттерн компоновщик
26. Паттерн декоратор
27. Паттерн фасад
28. Паттерн легковес (приспособленец)
29. Паттерн прокси
30. Паттерн цепочка обязанностей
31. Паттерн команда
32. Паттерн итератор
33. Паттерн посредник
34. Паттерн снимок
35. Паттерн наблюдатель
36. Паттерн состояние
37. Паттерн стратегия
38. Паттерн шаблонный метод
39. Паттерн посетитель
40. Виртуализация 1-го типа
41. Виртуализация 2-го типа
42. Контейнеры
43. Микросервисная архитектура
44. Infrastructure as Code
45. Брокеры сообщений
46. RabbitMQ: Fanout Exchange
47. RabbitMQ: Topic Exchange
48. RabbitMQ: Direct Exchange
49. RabbitMQ: Default Exchange
50. NoSQL
51. Теорема CAP
52. NoSQL: хранилища типа ключ-значение
53. NoSQL: документоориентированные СУБД
54. Continuous Integration / Continuous Delivery
55. **СИСТЕМЫ УПРАВЛЕНИЯ ВЕРСИЯМИ**

Version Control System (VCS) - система управления версиями (система контроля версий).

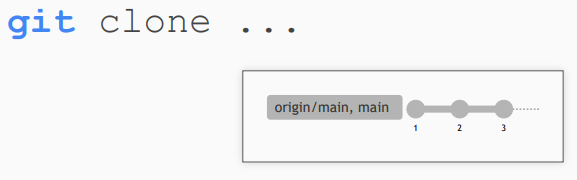
Варианты реализации:

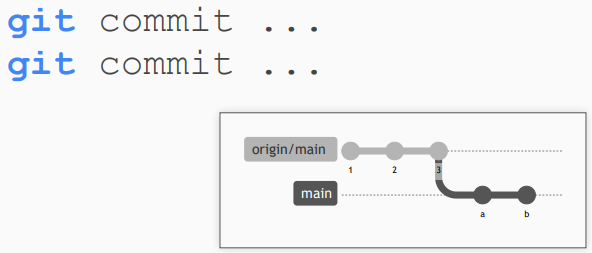
* 1. Репозиторий  
       
     
  2. Конфликты изменений – локальные VCS (пользователь один, конфликтов нет); блокировка файлов (используется в Visual SourceSafe, немного в Subversion); слияние (merge) измененных файлов (современный подход, работает автоматически, иногда требуется ручное решение конфликтов, Subversion, git, Mercurial)

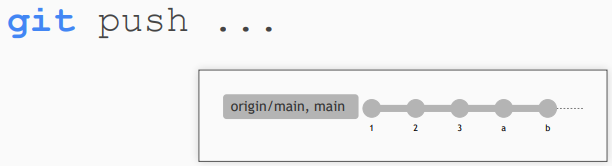
1. **git: centralized workflow**

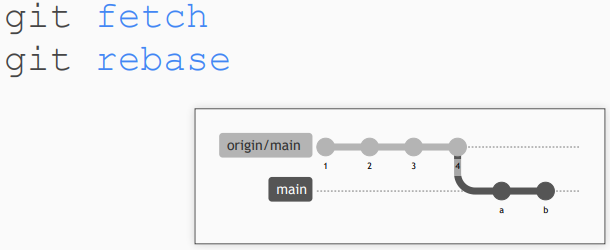
Основные проблемы: git весьма гибок и не предлагает стандартного процесса разработки.

Centralized workflow – простой, используется только 1 ветвь (master или main) для основной разработки, локальная ветвь master перебазируется на remote/master.





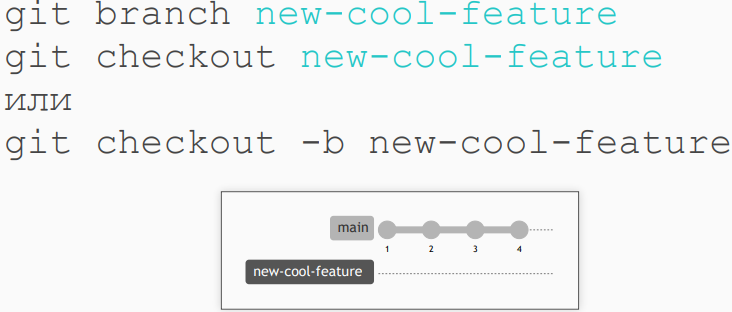


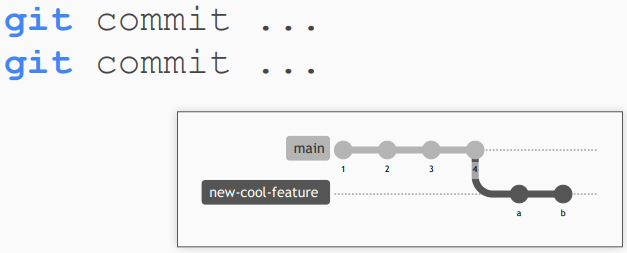


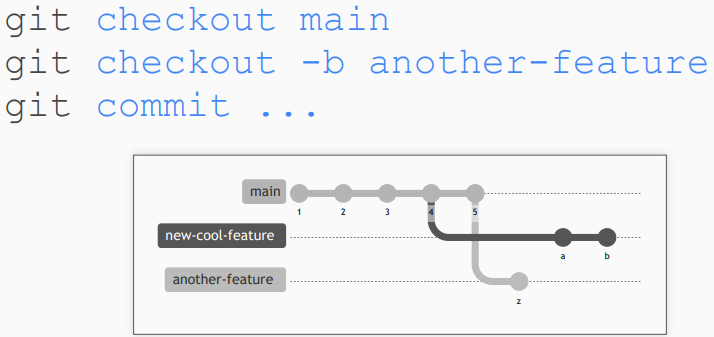
1. **git: feature branch workflow**

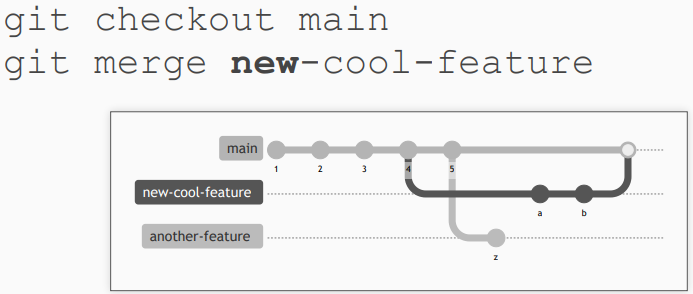
Основные проблемы: git весьма гибок и не предлагает стандартного процесса разработки.

Feature branch workflow – для каждого изменения создаем отдельную ветку, все изменения ведутся в ней, по завершению работы ветка сливается с основной. Слияние веток может быть ручным или с использованием pull request (если GitHub, bitbucket, GitLab и пр.)



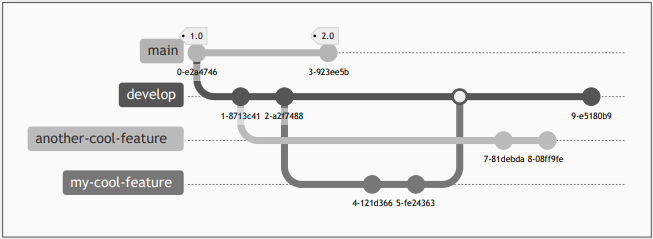




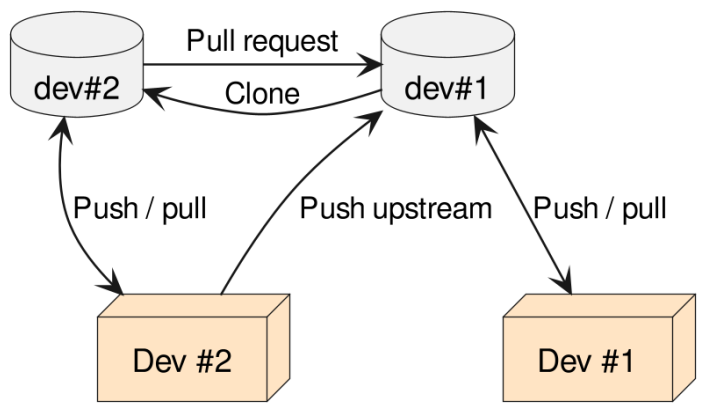
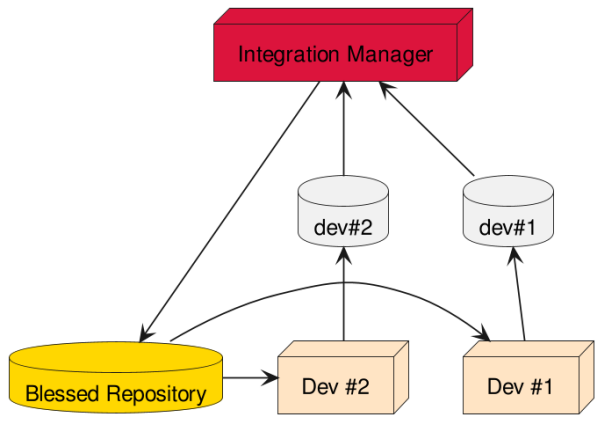


1. **gitflow**

Gitflow – более активное использование ветвей, есть дополнение к git (git-flow): git flow init, git flow feature start my-cool-feature, git flow feature finish my-cool-feature, 2 основные ветви main и develop.



Gerrit – похоже на Centralized Workflow + Code Review, есть расширение для git (git-review), у каждого коммита есть Change-Id, git push origin HEAD: master – запрещен, git push origin HEAD: refs/ for/ master.

Forking Workflow Integration Manager Workflow  
 

Идеальных workflow нет, можно комбинировать элементы любых техник, но выбранные правила должны быть общими для команды.

Основные принципы:

- Ветки должны cуществовать самостоятельно как можно меньше. Чем дольше ветка независима, тем сложнее её rebase или merge с основной.

- Минимизировать и упростить откаты. Процессы, где изменения тестируются до попадания в основную ветвь предпочтительнее.

- Соответствие расписанию релизов

- Не создавать излишних препятствий разработке

1. **ТЕСТИРОВАНИЕ ПО. КЛАССИФИКАЦИЯ**

Принципы: 1. Тестирование демонстрирует наличие дефектов, но не гарантирует их отсутствия. 2. Исчерпывающее тестирование невозможно. 3. Раннее тестирование. 4. Тестирование зависит от контекста.

*Классификация по запуску кода на исполнение:* Статическое тестирование — процесс тестирования, который проводится для верификации практически любого артефакта разработки: программного кода компонент, требований, системных спецификаций, функциональных спецификаций, документов проектирования и архитектуры программных систем и их компонентов. Динамическое тестирование — тестирование проводится на работающей системе, не может быть осуществлено без запуска программного кода приложения.

*Классификация по доступу к коду и архитектуре:* White box — полный доступ к коду проекта. Black box — метод не предполагает доступа (полного или частичного) к системе. Основывается на работе исключительно с внешним интерфейсом тестируемой системы. Gray box — промежуточный вариант.

*Классификация по доступу к коду и архитектуре:* Модульное — проводится для тестирования какого-либо одного логически выделенного и изолированного элемента (модуля) системы в коде. Проводится самими разработчиками, так как предполагает полный доступ к коду. Интеграционное — тестирование, направленное на проверку корректности взаимодействия нескольких модулей,

объединенных в единое целое. Системное — процесс тестирования системы, на котором проводится не только функциональное тестирование, но и оценка характеристик качества системы — ее устойчивости, надежности, безопасности и производительности. Приёмочное — проверяет соответствие системы потребностям, требованиям и бизнес-процессам пользователя.

*Классификация по степени автоматизации:* Ручное, Автоматизированное.

*Классификация по принципам работы с приложением:* Позитивное — тестирование, при котором используются только корректные данные. Негативное — тестирование приложения, при котором используются некорректные данные и выполняются некорректные операции.

*Классификация по уровню функционального тестирования:* Дымовое (smoke test) — тестирование, выполняемое на новой сборке, с целью подтверждения того, что программное обеспечение стартует и выполняет основные для бизнеса функции. Тестирование критического пути (critical path) — направлено для проверки функциональности, используемой обычными пользователями во время их повседневной деятельности.

1. **МОДУЛЬНОЕ ТЕСТИРОВАНИЕ**

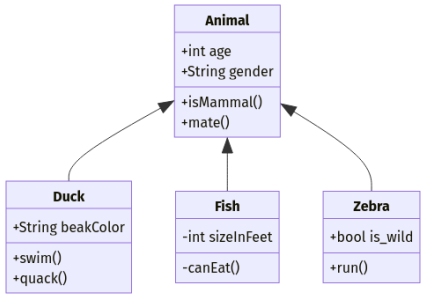
JUnit — фреймворк для модульного тестирования.

1. **МНОГОПОТОЧНОСТЬ. ДОСТОИНСТВА И НЕДОСТАТКИ**
2. **МНОГОПОТОЧНОСТЬ: SHARED STATE И ЕГО ПРОБЛЕМЫ**
3. **СИНХРОНИЗАЦИЯ В МНОГОПОТОЧНЫХ ПРОГРАММАХ**
4. **МНОГОПОТОЧНОСТЬ: DEADLOCK**
5. **АТОМАРНЫЕ ПЕРЕМЕННЫЕ**
6. **UNIFIED MODELLING LANGUAGE**

Unified Modelling Language — унифицированный язык моделирования (стандарт графического описания моделей, процессов и пр).

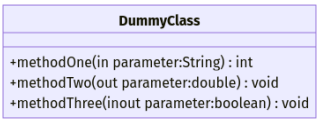
Классификация: 1. Структурные диаграммы (Structural diagrams): диаграмма классов (class diagram), диаграмма компонентов (component diagram), диаграмма композитной/составной структуры (composite structure diagram: диаграмма кооперации (collaboration diagram)), диаграмма развертывания (deployment diagram), диаграмма объектов (object diagram), диаграмма пакетов (package diagram), диаграмма профилей (profile diagram). 2. Диаграммы поведения (Behavioral diagrams): диаграмма деятельности (activity diagram), диаграмма коммуникации (communication diagram), диаграмма обзора взаимодействия (interaction overview diagram), диаграмма последовательности (sequence diagram), диаграмма состояний (state diagram), диаграмма синхронизации (timing diagram), диаграмма прецедентов/вариантов использования (use case diagram).

Недостатки: Неоправданно большой и сложный; Неточная семантика. Чуда не произошло и есть ряд неточностей, трактуемых всеми по-разному; "The code is the design". Код и есть проект. Только код отражает код; All models are wrong. Some are useful.



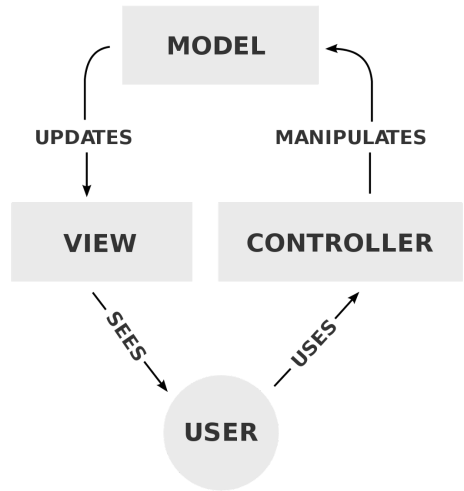
Диаграммы классов

Видимость: + public, - private, # protected, ~ package privave

Типы аргументов:  


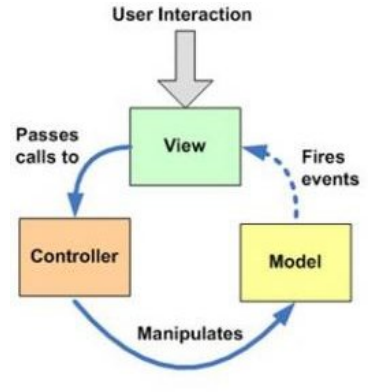
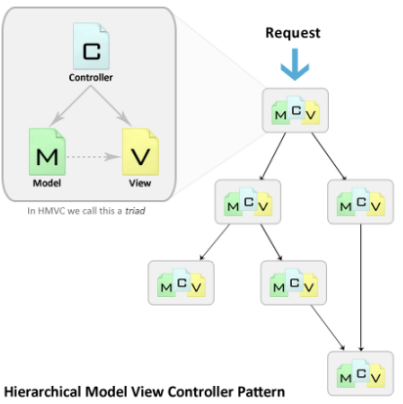
1. **ВНЕДРЕНИЕ ЗАВИСИМОСТЕЙ**
2. **ЛОКАЛИЗАЦИЯ И ИНТЕРНАЦИОНАЛИЗАЦИЯ**
3. **MODEL VIEW CONTROLLER**

Model-View-Controller — архитерктуный паттерн построения приложений (обычно приложений c UI). Model (модель) — основной элемент, содержит бизнес-логику приложения. Управляется контроллером. Может отправлять сообщения во View. View (вид) — отображает модель (или какую-то её часть для пользователя). Controller (контроллер) — реагирует на действия пользователя и передаёт изменения в модель.

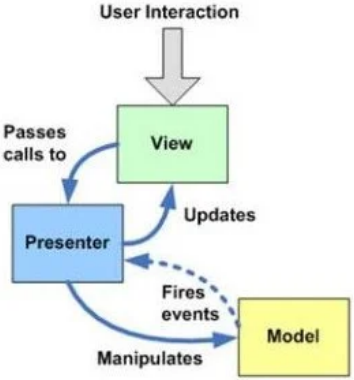
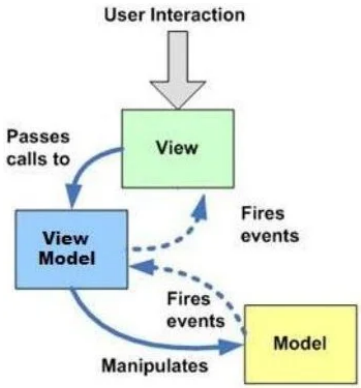


1. **ВАРИАНТЫ MVC**

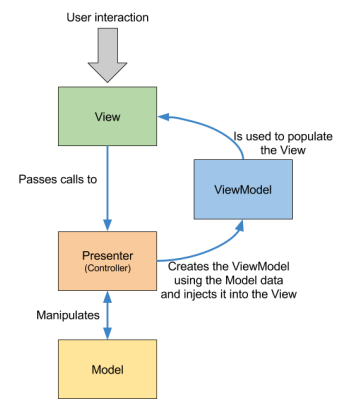
1979 г. – Model-View-Controller 1987 г. – Иерархический MVC (HMVC, PAC)

Model-View-Presenter Model-View-VIewModel (MVVM)

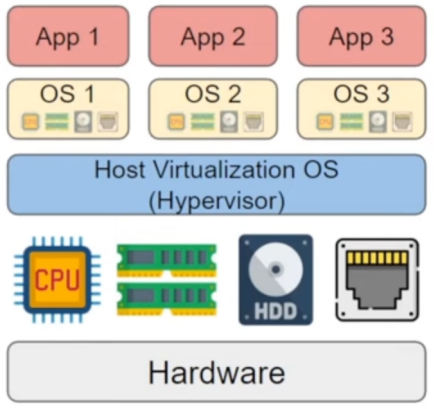
 

Model-View-Presenter-ViewModel



1. **ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ. КЛАССИФИКАЦИЯ**
2. **ПАТТЕРН ОДИНОЧКА**
3. **ПАТТЕРН ФАБРИЧНЫЙ МЕТОД**
4. **ПАТТЕРН АБСТРАКТНАЯ ФАБРИКА**
5. **ПАТТЕРН ПРОТОТИП**
6. **ПАТТЕРН СТРОИТЕЛЬ**
7. **ПАТТЕРН АДАПТЕР**
8. **ПАТТЕРН МОСТ**
9. **ПАТТЕРН КОМПОНОВЩИК**
10. **ПАТТЕРН ДЕКОРАТОР**
11. **ПАТТЕРН ФАСАД**
12. **ПАТТЕРН ЛЕГКОВЕС (ПРИСПОСОБЛЕНЕЦ)**
13. **ПАТТЕРН ПРОКСИ**
14. **ПАТТЕРН ЦЕПОЧКА ОБЯЗАННОСТЕЙ**
15. **ПАТТЕРН КОМАНДА**
16. **ПАТТЕРН ИТЕРАТОР**
17. **ПАТТЕРН ПОСРЕДНИК**
18. **ПАТТЕРН СНИМОК**
19. **ПАТТЕРН НАБЛЮДАТЕЛЬ**
20. **ПАТТЕРН СОСТОЯНИЕ**
21. **ПАТТЕРН СТРАТЕГИЯ**
22. **ПАТТЕРН ШАБЛОННЫЙ МЕТОД**
23. **ПАТТЕРН ПОСЕТИТЕЛЬ**
24. **ВИРТУАЛИЗАЦИЯ 1-ГО ТИПА**

Виртуализация — выделение ресурсов в изолированную среду.

****

Хост (Host) — основной компьютер / сервер, на котором и запущен гипервизор. Гость (Guest) - виртуальный компьютер. Чаще употребляется Guest OS - операционная система, запущенная в виртуальной машине. Гипервизор (hypervisor) - ПО занимающееся виртуализацией (эмуляция аппаратных ресурсов, изоляция гостевых ОС от хоста и друг от друга, распределение ресурсов). Паравиртуализация (paravirtualization) - техники подготовки гостевой ОС для работы в виртуальной среде.

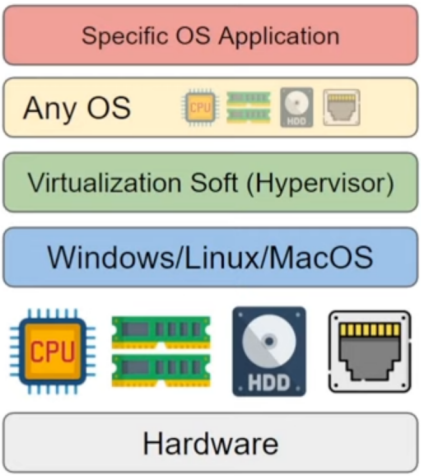
Hypervisor OS сильно урезана, как правило не имеет никакого GUI, только CLI. Но взамен предоставляется мощный web-интерфейс, desktop-приложение и API. Достоинства: высокая плотность виртуалок, миграция машин между серверами, есть бесплатный софт. Недостатки: высокая стоимость серверов, квалификация инженеров.

WMWare ESXi - одна из первых (с 2001 года), стандарт де-факто, есть бесплатная версия, но сильно урезанная (только один хост, ограничения на ресурсы и т.п.). Microsoft Hyper-V - требует серверную лицензию Windows, активно развивается, лучше для виртуализации Windows-машин. Xen - была платной, но после покупки компанией Citrix стала полностью бесплатной, не очень широко распространена. KVM - изначально Qumranet, затем Red Hat, открытый и бесплатный, на его основе построена масса других открытых и платных решений (oVirt, Red Hat Virtualization, Proxmox Virtual Enviroment).

Виртуализация позволила эффективнее использовать вычислительные ресурсы, упростила управление, резервное копирование, восстановление, усложнила мир.

1. **ВИРТУАЛИЗАЦИЯ 2-ГО ТИПА**

Виртуализация — выделение ресурсов в изолированную среду.

****

Хост (Host) — основной компьютер / сервер, на котором и запущен гипервизор. Гость (Guest) - виртуальный компьютер. Чаще употребляется Guest OS - операционная система, запущенная в виртуальной машине. Гипервизор (hypervisor) - ПО занимающееся виртуализацией (эмуляция аппаратных ресурсов, изоляция гостевых ОС от хоста и друг от друга, распределение ресурсов). Паравиртуализация (paravirtualization) - техники подготовки гостевой ОС для работы в виртуальной среде.

Достоинства: доступность, удобство, есть бесплатные варианты. Недостатки: неудобны для серверных решений, низкая плотность виртуальных машин.

VirtualBox - сейчас пренадлежит Oracle, бесплатный, работает под разными ОС (Windows, Linux, MacOS, Solaris). VMWare Workstation Pro - платное, но VMWare Workstation Player бесплатный, интеграция с VMWare ESXi, поддерживает Windows и Linux. Их много: Parallels, Microsoft Hyper-V, Xen, EMU и пр.

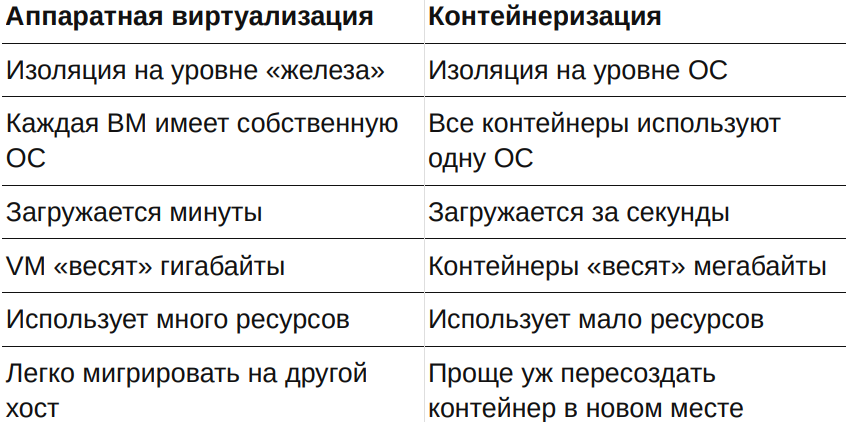
В виртуализации 2-го типа не забываем проверить в BIOS включена ли виртуализация (VT-x, VT-d, AMD-V), не забываем про Guest Additions для комфортной жизни (впрочем, гипервизор сам об этом напомнит), разные гипервизоры не очень ладят друг с другом: при включенном Microsoft Hyper-V может перестать работать VirtualBox и VMWare Workstation.

Виртуализация позволила эффективнее использовать вычислительные ресурсы, упростила управление, резервное копирование, восстановление, усложнила мир.

1. **КОНТЕЙНЕРЫ**

Контейнеризация — это вирутализация уровня операционной системы. Изолированные пространства называют контейнерами (docker, LXC, Podman), зоны (Solaris containers), виртуальные частные серверы (OpenVZ), разделы, виртуальные пространства, тюрьмы (jail) и т.д. Контейнер — это не полноценная ОС, он использует ресурсы ОС хоста.

Отличия от аппаратной виртуализации

****

Контейнеры стали популярны из-за скорости и простоты создания образов, лёгкости и универсальности, быстрого изменения и откатов, экономного использования ресурсов.

*Docker*. Недостатки: Хорошо работает под Linux, но есть проблемы с Windows и

MacOS; В случае Windows запускается отдельная VM (или WSL) с Linux и Docker Daemon; Не так просто конфигурировать / модифицировать существующие образы и контейнеры.

*Терминология*. image (образ) — шаблон контейнера (неизменяемый). container (контейнер) — это уже развернутое из образа и работающее приложение. docker daemon — резидентный процесс, который запущен на хост-машине постоянно. Он владеет всей инфраструктурой, а также предоставляет интерфейс взаимодействия с контейнерами, включающего создание и удаление, запуск и остановку. Ранее это dockerd, сейчас чаще речь идёт про containerd. docker client — CLI утилита docker для управления docker daemon.

Docker Registry - централизованное хранилище образов. Как правило речь идёт про Docker Hub, но можно создать своё частное хранилище. Работа через docker login / docker pull / docker push и пр.

Docker Volumes - создание тома при помощи Dockerfile VOLUME /var/lib/mysql; docker volume ls (имеющиеся в системе тома), docker volume create mysql-data(создание тома), docker volume create --driver dostorage --name my-volum (использование сторонних хранилищ при создании тома); docker inspect [container id|image id|volume id] (подробная информация об объекте); docker exec (запуск команды внутри контейнера); docker logs (логи приложения из контейнера).

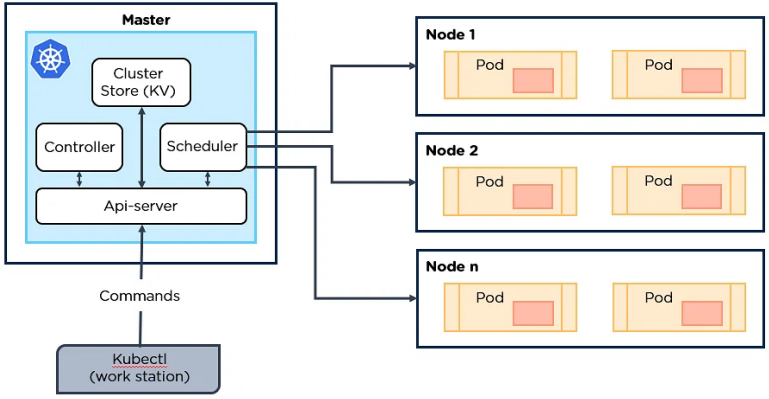
1. **МИКРОСЕРВИСНАЯ АРХИТЕКТУРА**

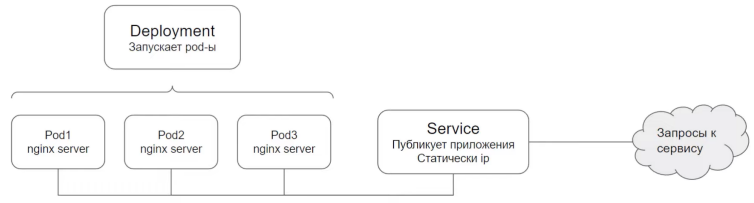
Достоинства: Простота и скорость разработки, Оптимальное масштабирование, Свобода выбора технологий (в т.ч. и хранилища), Небольшие команды разработки. Недостатки: Усложнение архитектуры, Изменение процесса разработки, Сложнее обеспечить безопасность, Сложнее обеспечить безопасность.

Проблемы: сложность управления системой, сложность масштабирования, сложность распределения нагрузки, сложность обеспечения отказоустойчивости.

Kubernetes (k8n, «кубер») — open source утилита оркестрации контейнеров. Это не одна программа, а фреймворк с большим набором технологий. Kubernetes помогает управлять контейнеризированными приложениями в любой среде: физические серверы, виртуальные машины, публичные облака. Kubernetes - это абстракция: переход от управления сетями, узлами и серверами к управлению сервисами. Kubernetes автоматизирует рутинные задачи: Отказоустойчивость (high availability), Отказоустойчивость (high availability), Аварийное восстановление (disaster recovery).

Архитектура:





1. **INFRASTRUCTURE AS CODE**

Infrastructure as Code — это процесс управления и провизионирования датацентров и серверов с помощью машиночитаемых файлов определений, созданный как альтернатива физическому конфигурированию оборудования и

оперируемым человеком инструментам.

Достоинства: скорость и уменьшение затрат, масштабируемость и стандартизация, восстановление в аварийных ситуациях, восстановление в аварийных ситуациях. Недостатки: всё становится сложнее, требуется высокая квалификация специлистов.

Vagrant - инструмент для создания и конфигурирования легковесных, повторяемых и переносимых окружений для разработки и пр., «из коробки» умеет управлять машинами VirtualBox, Hyper-V, контейнерами Docker. Но список расширяем плагинами, разрабатывается компанией HashiCorp, бесплатный. Vagrant - централизованное хранилище образов (только здесь box, а не image), можно иметь и свои репозитории образов, можно иметь и свои репозитории образов, возможно импорт из виртуальных машин в т.ч. и образов OVA/OVF.

Packer - автоматическое создание образов для vagrant, amazon, digital ocean,пр

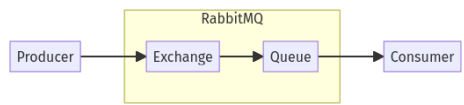
Terraform - решение для IaC (Infrastructure as Code), бесплатное, работает с AWS, Azure, Yandex Cloud, Google Cloud Platform, разрабатывается компанией HashiCorp, существует с 2014 года.

1. **БРОКЕРЫ СООБЩЕНИЙ**

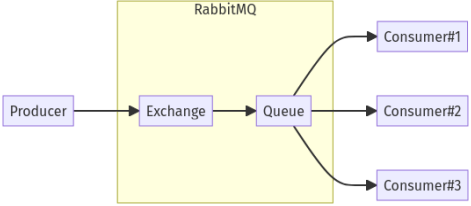
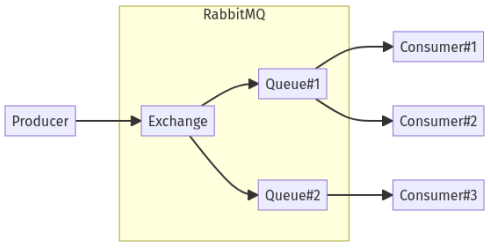
Брокер сообщений (message broker) — архитектурный паттерн или модуль для трансляции сообщений от отправителей к получателям. Позволяет решить многие проблемы микросервисной архитектуры: не нужен фиксированный API у микросервисов, распределение нагрузки, аварийные ситуации.

Примеры брокеров: Apache ActiveMQ, Apache Kafka, RabbitMQ, AmazonMQ, Microsoft Azure Service Bus и пр.

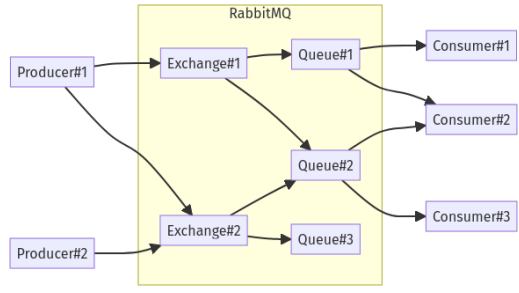
RabbitMQ - пишется с 2007, текущий владелец Pivotal Software, Open source и бесплатен, Написан на erlang, Клиентские библиотеки для популярных языков (Java, Python, Ruby, C#, JavaScript, Go, Swift), Поддержка протокола AMQP («Advanced Message Queuing Protocol»), но умеет и в STOMP, MQTT и многие другие.



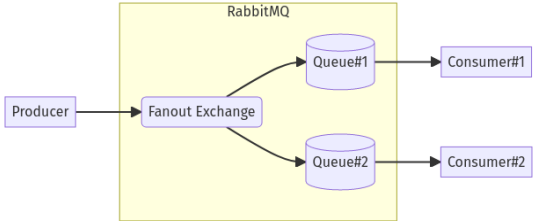
Producer — поставщик сообщений. Producer — поставщик сообщений. Exchange — «почтальон» обеспечивающий доставку в «почтовый ящик». Queue — очередь сообщений. Каналы позволяют использовать одно и то же соединение с RabbitMQ сервером для разных целей.

Сообщения многим: Разные очереди:  
 

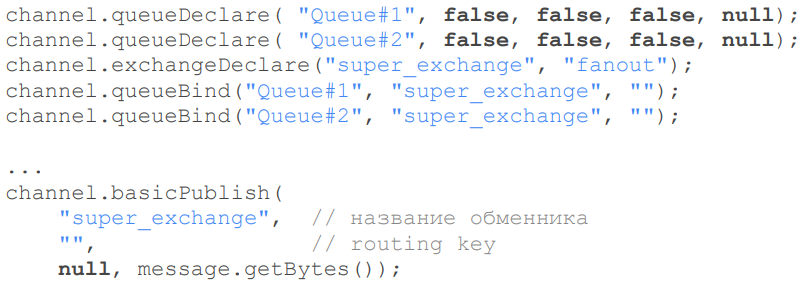
Несколько exchange:



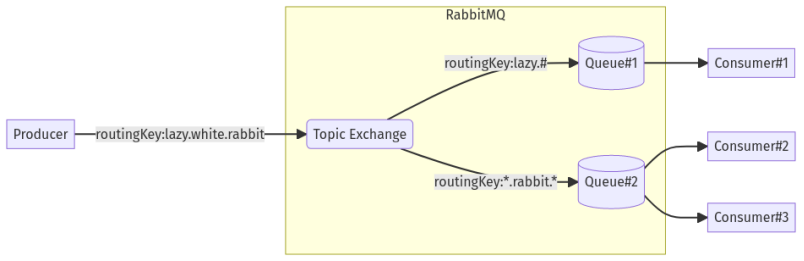
1. **RABBITMQ: FANOUT EXCHANGE**

****

Отправляет сообщения во все queue, привязанные к этому обменнику. Никакой маршрутизации. Важна только привязка exchange к нужным queue.

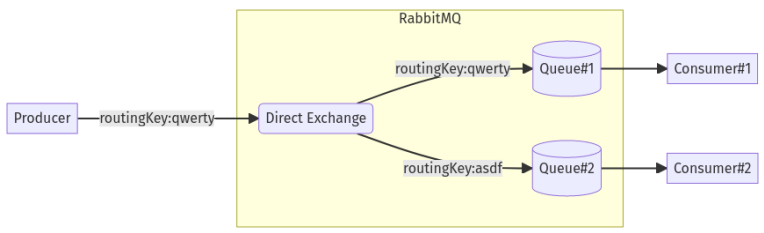


1. **RABBITMQ: TOPIC EXCHANGE**

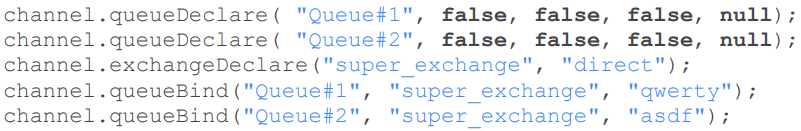
****

В сообщениях routing key - это набор слов, разделённых точками (как теги: lazy.white.rabbit). Привязки к очередям также имеют routing key такого же формата, но и wildcards: \* (одно любое слово), \*.white.\* (какое-то одно слово, потом white, потом ещё какое-то одно слово), # (сколько угодно слов), #.rabbit.# (неважно, какие слова есть, главное, чтобы было слово rabbit)

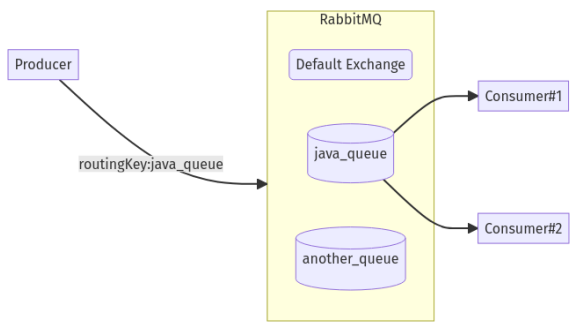
1. **RABBITMQ: DIRECT EXCHANGE**

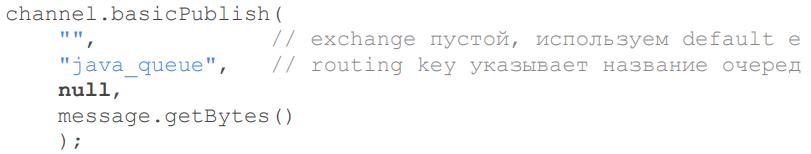
****

Очень похож на default exсhange, но маршрутизация идёт только по привязанным очередям, очередь выбирается не по названию (оно может быть любым), а по routingKey привязки.



1. **RABBITMQ: DEFAULT EXCHANGE**

****

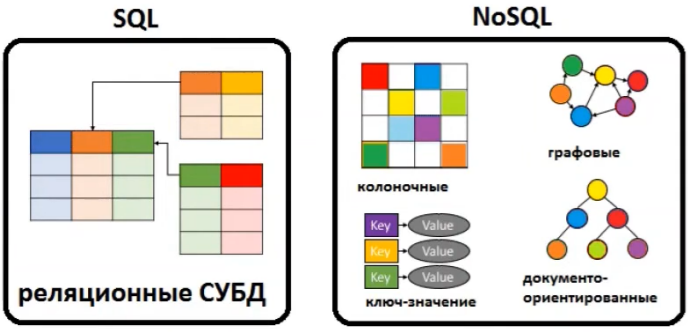
****

1. **NOSQL**

NoSQL (Not only SQL — не только SQL) термин объединяющий несколько разных подходов к раализации хранилищ баз данных, отличных от реляционных.

Причины появления NoSQL: потребность в распределённых СУБД, потребность в быстрой работе с данными, ряд стандартных задач можно моделировать проще, чем в РСУДБ. Особенности NoSQL БД: объект в БД — более сложная структура, чем строка в таблице, объект может не иметь чёткой структуры, мы не хотим использовать операции соединения (JOIN), хорошо масштабируется.

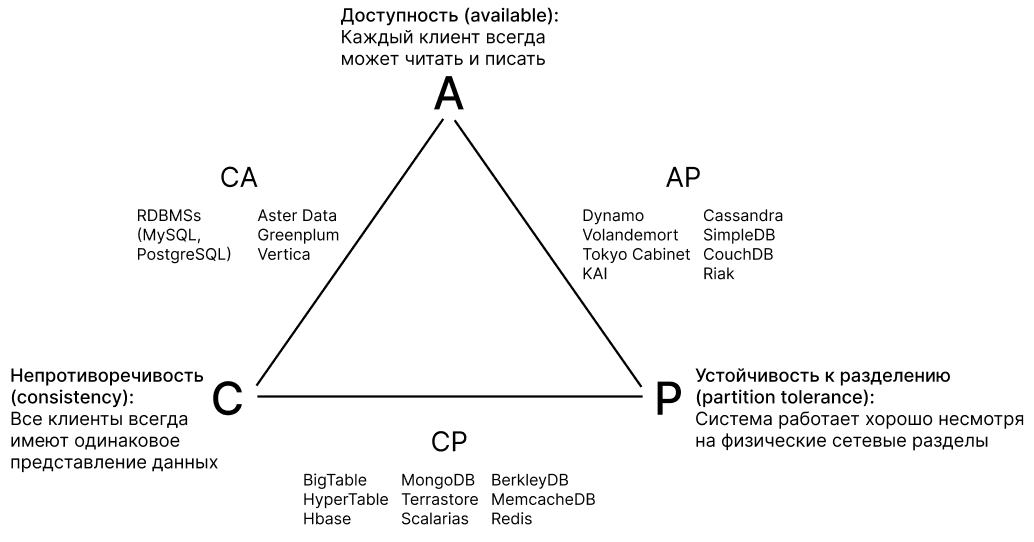
ACID → BASE: Базовая доступность (**B**asically **A**vailable) — каждый вопрос гарантированно завершается (в т.ч. и с ошибкой); Гибкое состояние (**S**oft state) — состояние системы может измениться со временем даже без добавления новых данных (для согласования данных); Согласованность в конечном счёте (**E**ventually consistent) — данные могут быть какое-то время несогласованы, но придут к согласованию через какое-то время.

****

В РСУБД и SQL плохо: не все данные хорошо «ложатся» в реляционную БД, производительность, зачастую излишняя сложность, затраты на преобразование объектов БД в объекты ПО, стоимость (Oracle, MS SQL и пр.) хотя хватает и бесплатных (MySQL/MariaDB, postgreSQL).

1. **ТЕОРЕМА CAP**

Доступность можно измерять по времени, прошедшему с посылки запроса до получения ответа. Какие-то системы могут позволить себе долго отвечать на запрос, другие должны это сделать за определённый промежуток времени. Свойство устойчивости к разделению напрямую зависит от того, каким образом классифицировать сложившуюся ситуацию: разделение или нет? У согласованности данных тоже могут быть разные степени.

****

Достуность сервиса (AP, Availability + Partition Tolerance) — система всегда доуступна, отвечает на все запросы, но данные могут быть не всегда актуальны. Например: система DNS. Целостность данных при разделении (CP, Consistency + Partition Tolerance) — данные всегда корректны, но при проблемах с сетью могут и не отвечать. Например: Hbase, Google bigtable, многие NoSQL БД. Целостность и доступность (AC, Availiability + Consistency) - данные всегда корректны и доступны, но всё очень плохо масштабируется. Пример: классические РСУБД.

В теории сетевые соединения надёжны, пропускная способность сети бесконечна, соединения безопасны. На практике это не так. Современное продолжение теоремы CAP - теорема PACELC: в случае разделения сетей (**P**artitioning) нужно выбирать между доступностью (**A**vailability) и целостностью (**C**onsistency), но иначе (**E**else), при нормальной работе системы, выбор стоит между задержкой (**L**atency) и целостностью (**C**onsistency).

1. **NOSQL: ХРАНИЛИЩА ТИПА КЛЮЧ-ЗНАЧЕНИЕ**
2. **NOSQL: ДОКУМЕНТООРИЕНТИРОВАННЫЕ СУБД**
3. **CONTINUOUS INTEGRATION / CONTINUOUS DELIVERY**