ТЕСТОВОЕ ЗАДАНИЕ НА СТАЖИРОВКУ

Выполнитель : Галлаб А.М

1 Вопросы :

- Чем git pull отличается от git fetch ?

Они обе команды Git, и их используют для получения изменения из удаленного репозитория, но они делают это по-разному:

git fetch - загружает изменения из удаленного репозитория в локальный, но не изменяет локальные ветки. Он скачивает все новые коммиты, файлы и другие объекты из удаленного репозитория и сохраняет их в локальном репозитории, но в специальных ветках, называемых fetch-heads. Эти ветки отслеживают состояние удаленных репозиториев, но не сливаются с рабочими ветками. Мы должны явно выполнить git merge или git rebase для объединения этих изменений с вашей локальной веткой.

git pull - это на самом деле сокращение для git fetch затем git merge. То есть, он загружает изменения из удаленного репозитория и автоматически сливает их в текущую локальную ветку. Если слияние происходит без конфликтов, это происходит незаметно. Если возникают конфликты, Git отметит их, и нужно будет вручную разрешить конфликты перед завершением слияния.

- Какие модели ветвления git существуют?

1. **Модель Gitflow:**

Описание: Это достаточно сложная, но хорошо структурированная модель, предназначенная для крупных проектов с несколькими разработчиками. Она использует отдельные ветки для разработки, выпуска и горячих исправлений (хотфиксов).

- **Ветки:**
  - main (или master): Основная ветка, содержащая стабильный, релизный код.
  - develop: Ветка разработки, в которую сливаются все фичи.
  - feature/*: Ветки для новых фич (функционала). Создаются из develop и сливаются обратно в develop.
  - release/*: Ветки для подготовки релизов. Создаются из develop, в них вносятся небольшие исправления и обновления перед релизом. Затем сливаются в main и develop.
  - hotfix/*: Ветки для исправления критических ошибок в релизной версии. Создаются из main, исправляют ошибку, затем сливаются в main и develop.

2. **GitHub Flow**:

Описание: Более простая и гибкая модель, идеально подходящая для проектов с частыми развертываниями.

- **Ветки:**
  - main (или master): Основная ветка, всегда отражает рабочую версию.
  - feature/*: Ветки для новых фич. Создаются из main, а после завершения сливаются обратно в main.

3. Gitlab Flow:

- **Описание:** Подобна GitHub Flow, но добавляет поддержку веток для релизов (release/*) и среды (environment/*).
- **Ветки:**
  - main (или master): Основная ветка.
  - feature/*: Ветки для новых фич.
  - release/*: Ветки для подготовки релизов, используются для тестирования в промежуточной среде.
  - environment/*: Ветки для разных сред (staging, production и т.д.)

4. Trunk-Based Development:

- **Описание:** Фокусируется на минимальном использовании веток. Разработчики интегрируют свои изменения в основную ветку (main или trunk) несколько раз в день.
- **Ветки:** В основном используется только main, короткие ветки для небольших задач могут создаваться, но быстро сливаются обратно в main.

- Чем в linux отличается soft link от hard link? Какое поведение будет при удалении файла оригинала? Можно ли создать hardlink на директорию?

Характеристика	**Soft Link**	**Hard Link**
Тип	Символическая ссылка	Жесткая ссылка
Хранение данных	Только путь	Нет данных, только i-node
Удаление оригинала	Ссылка становится битой	Файл остается, если существуют другие hard link-и
i-node	Разный для ссылки и оригинала	Один и тот же для всех ссылок
Создание	ln -s	ln
Ссылки на директории	Да	Нет

- Нельзя создать hard link на директорию.
- **Пример:**

```
touch my_file.txt
ln -s my_file.txt my_file_softlink.txt  # Soft link
ln my_file.txt my_file_hardlink.txt     # Hard link
```

Теперь удалим my_file.txt:

- my_file_softlink.txt станет битой ссылкой.
- my_file_hardlink.txt будет по-прежнему работать, потому что это жесткая ссылка, и данные файла все еще существуют. Только после удаления my_file_hardlink.txt данные будут удалены.

- Как проверить сетевую доступность между двумя linux машинами?

1. Использовать команду ping. Успешный пинг указывает на доступность на уровне сети.

```
ping <IP-адрес или имя хоста>
```

2. Netcat (nc) -z указывает на режим сканирования — nc не будет пытаться установить соединение, а только проверит доступность порта. -v обеспечивает более подробный вывод.

```
nc -zv <IP-адрес или имя хоста> <номер порта>
```

- Чем контейнер отличается от виртуальной машины?

Контейнеры и виртуальные машины (ВМ) — это две разные технологии виртуализации, которые позволяют запускать несколько изолированных сред на одном физическом сервере. Однако они отличаются по своей архитектуре и производительности:

Характеристика	**Виртуальная Машина (ВМ)**	**Контейнер**
Изоляция	Высокая	Низкая (но достаточная)
Гипервизор	Требуется	Не требуется
Ядро	Виртуальное	Разделяемое (хост-системы)
Размер	Большой	Маленький
Время запуска	Медленное	Быстрое
Потребление ресурсов	Высокое	Низкое
Переносимость	Высокая	Высокая (с использованием Docker и т.п.)
Зависимость от ОС хоста	Низкая	Высокая

- На каких компонентах Linux основана контейнеризации в Docker?

Namespaces (Пространства имен): Namespaces изолируют различные аспекты системы, предоставляя каждому контейнеру свою собственную, виртуальную, версию этих аспектов.

Control groups (cgroups): Cgroups предоставляют механизм для ограничения и мониторинга использования ресурсов контейнерами. Они позволяют задавать лимиты на использование CPU, памяти, дискового пространства и других ресурсов, предотвращая перегрузку системы одним контейнером.

Union file systems (Объединенные файловые системы): Docker использует union file systems (например, AUFS, btrfs, OverlayFS, Device Mapper) для эффективного управления файловой системой контейнера. Они позволяют создавать слоистую файловую систему, где изменения, внесенные в контейнер, хранятся в отдельных слоях, что обеспечивает экономию дискового пространства и ускоряет создание и запуск контейнеров. OverlayFS — наиболее распространенный сейчас механизм.

Libcontainer (или containerd): Это библиотека, которая реализует низкоуровневый доступ к namespaces и cgroups. Docker использует её (или containerd, который её заменяет) для создания и управления контейнерами.

- **Опишите составные части архитектуры Docker**

Архитектура Docker состоит из нескольких взаимосвязанных компонентов, которые работают вместе, чтобы обеспечить контейнеризацию и управление образами и контейнерами. Основные компоненты:

1. Docker Client (Клиент Docker):

Функция: Это интерфейс командной строки (CLI) или API, с помощью которого пользователи взаимодействуют с Docker. Используется клиент для создания, запуска, остановки, удаления контейнеров и управления образами.

Взаимодействие: Клиент взаимодействует с Docker daemon (демони) через REST API.

2. Docker Daemon (Демон Docker):

Функция: Это долго работающий процесс на хост-машине, который отвечает за выполнение команд, отправленных клиентом. Он управляет образами, контейнерами, сетями и хранилищами. Он является сердцевиной Docker.

Взаимодействие: Слушает на Unix сокете (по умолчанию) или TCP порте (если конфигурировано) для приема команд от Docker Client.

3. Docker Images (Образы Docker):

Функция: Это неизменяемые (immutable) шаблоны для создания контейнеров. Образ содержит все необходимые файлы, библиотеки, зависимости и инструкции для запуска приложения. Они построены слой за слоем, что обеспечивает эффективность хранения и обмена.

Хранение: Хранятся в Docker registry (реестре).

4. Docker Containers (Контейнеры Docker):

Функция: Это экземпляры образов Docker, запущенные на хост-машине. Каждый контейнер представляет собой изолированное окружение для запуска приложения.

Взаимодействие: Запускаются Docker daemon по запросу от Docker Client и используют ресурсы хоста.

5. Docker Registry (Реестр Docker):

Функция: Это централизованное хранилище для Docker образов. Самый известный реестр — Docker Hub.

Взаимодействие: Docker daemon использует registry для загрузки и загрузки образов.

6. Docker Hub:

Функция: Это публичный облачный реестр Docker, предоставляемый Docker, Inc. Он содержит огромное количество общедоступных образов.

Схема взаимодействия:

1. Пользователь использует docker client для отправки команд.
2. docker client отправляет команды через REST API к docker daemon.
3. docker daemon обрабатывает команды и управляет docker images и docker containers, используя возможности ядра Linux (namespaces, cgroups и т.д.).
4. docker daemon взаимодействует с docker registry (например, Docker Hub) для загрузки и загрузки образов.

- **Как устроен образ контейнера?**

Устройство образа можно представить следующим образом:

1. Слоистая архитектура (Layered Architecture):

Образ Docker состоит из нескольких слоёв (layers). Каждый слой содержит определенные изменения относительно предыдущего слоя. Это ключевая особенность, обеспечивающая эффективность:

- **Эффективность хранения:** Если несколько образов используют одни и те же базовые слои, эти слои хранятся только один раз на диске. Это значительно экономит место.
- **Быстрый запуск:** При запуске контейнера Docker использует только необходимые слои, а не копирует весь образ каждый раз. Это значительно ускоряет запуск контейнеров.
- **Изменения:** Изменения, вносимые при создании нового образа, создают новый слой поверх существующих. Это сохраняет неизменяемость нижних слоёв, позволяя повторно использовать их в других образах.

2. Файлы и инструкции:

Каждый слой содержит файлы и инструкции, определённые в Dockerfile. Dockerfile — это текстовый файл, содержащий инструкции для сборки образа. Инструкции определяют, какие команды выполняются для создания каждого слоя:

- FROM: Указывает базовый образ, на котором строится текущий.
- RUN: Выполняет команду в текущем слое.
- COPY: Копирует файлы из хост-машины в образ.
- ADD: Похоже на COPY, но может также скачивать файлы из URL.
- CMD: Устанавливает команду, которая будет выполнена при запуске контейнера.

- ENTRYPOINT: Устанавливает исполняемый файл, который будет запущен при запуске контейнера.
- EXPOSE: Определяет порты, которые будут доступны извне контейнера.
- ENV: Устанавливает переменные окружения.
- WORKDIR: Устанавливает рабочую директорию.
- USER: Устанавливает пользователя, от имени которого будут выполняться команды.
- и другие.

3. Механизм UnionFS:

Docker использует UnionFS (Объединенная файловая система) для реализации слоистой архитектуры. UnionFS позволяет объединять несколько слоев файловой системы в единое виртуальное пространство. В Docker часто используется OverlayFS, но могут применяться и другие реализации.

4. Manifest (манифест):

Это JSON-файл, который описывает образ: его имя, теги, архитектуру, и указатели на слои. Это метаданные, которые позволяют Docker понять, как собрать и запустить образ.

- **Почему вместо "COPY . . / RUN npm install" рекомендуют делать "COPY package.json / RUN npm install / COPY . ."**

Рекомендация использовать COPY package.json / RUN npm install / COPY . . вместо COPY . . / RUN npm install в Dockerfile связана с оптимизацией процесса сборки и кеширования слоёв.

Разделение команды COPY и RUN npm install на два отдельных этапа позволяет Docker эффективно использовать кеширование слоёв. Это значительно ускоряет процесс сборки образов и делает его более предсказуемым, особенно в больших проектах с множеством зависимостей. Если package.json не изменяется, npm install выполняется только один раз, и это значительно экономит время. Это ключевой принцип оптимизации Dockerfile для повышения скорости и эффективности сборки.

Что такое под в Kubernetes? Могут ли два контейнера внутри одного пода слушать один и тот же порт?

В Kubernetes, под — это базовая единица развертывания. Это группа одного или нескольких контейнеров, которые работают вместе и разделяют некоторые ресурсы, такие как IP-адрес, объём хранения и сеть. Можно представить под как логическую машину, содержащую один или несколько процессов (контейнеров). Под может содержать один или несколько контейнеров, которые тесно взаимодействуют друг с другом. Каждый под получает собственный IP-адрес и имя хоста, что позволяет контейнерам внутри пода общаться друг с другом по сети. Поды могут использовать общие тома (volumes) для обмена данными между контейнерами. Контейнеры в одном под находятся в одной сети. Как и контейнеры, поды обычно считаются неизменяемыми. Для обновления под заменяется новым.

Два контейнера внутри одного пода *не могут* слушать один и тот же порт. Это связано с тем, что у каждого пода есть только один IP-адрес и порт – это ограничение со стороны сети. Если два контейнера попытаются слушать один и тот же порт, возникнет конфликт, и один из контейнеров не сможет работать корректно.

- **Какие виды JOIN знаете и чем они отличаются?**

Тип JOIN	**Левая таблица**	**Правая таблица**	**Результат**
INNER JOIN	Только совпадения	Только совпадения	Только строки с совпадениями в обеих таблицах
LEFT (OUTER) JOIN	Все строки	Только совпадения	Все строки из левой таблицы, совпадения из правой
RIGHT (OUTER) JOIN	Только совпадения	Все строки	Все строки из правой таблицы, совпадения из левой
FULL (OUTER) JOIN	Все строки	Все строки	Все строки из обеих таблиц

- Что такое having в SQL запросе? Чем отличается от where?

WHERE:

- **Функция:** Фильтрует строки *перед* группировкой (если используется GROUP BY). Он применяется к отдельным строкам таблицы.

HAVING:

- **Функция:** Фильтрует группы строк *после* группировки (если используется GROUP BY). Он применяется к агрегированным результатам.

WHERE фильтрует *отдельные строки*, а HAVING фильтрует *группы строк*, которые были созданы с помощью GROUP BY. HAVING не может использовать столбцы, которые не участвуют в GROUP BY