1. Что такое вычислительная система? Коллектив вычислителей? Классификация Флинна, примеры. SMP/NUMA системы, их отличия. Живучие ВС/ВС со структурной избыточностью – особенности и отличия.

- **Вычислительная система** — это совокупность аппаратных и программных средств, которые совместно выполняют вычислительные задачи. Пример: ПК или кластер серверов.
- **Коллектив вычислителей** — это несколько вычислительных устройств, которые работают вместе для выполнения задачи. Например, серверный кластер.
- **Классификация Флинна**:
    - **SISD (Single Instruction Stream Single Data stream)**: Однопроцессорная система, выполняющая одну инструкцию за раз (например, обычный ПК).
    - **SIMD (Single Instruction stream Multiple Data streams)**: Одна инструкция выполняется над несколькими данными (например, векторные процессоры).
    - **MISD (Multiple Instruction streams Single Data stream)**: Множественные потоки инструкций для одного потока данных (редко используется).
    - **MIMD (Multiple Instruction streams Multiple Data streams)**: Несколько процессоров выполняют разные инструкции над разными данными (например, многопроцессорные системы).
- **SMP (Symmetric Multiprocessing)** — система с несколькими процессорами, работающими в симметричном режиме, имеющие общий доступ к памяти.
- **NUMA (Non-Uniform Memory Access)** — система с несколькими процессорами, но память разделена на несколько регионов, доступ к которым имеет разное время отклика в зависимости от расположения процессора.
- **Отличия SMP и NUMA**: SMP имеет единый адрес памяти для всех процессоров, NUMA — более сложная архитектура с неодинаковыми временными задержками доступа к памяти.
- **Живучие ВС (Fault-Tolerant Systems)** — системы, которые могут продолжать работать при выходе из строя части компонентов. Например, серверы с резервированием.
- **ВС со структурной избыточностью** — системы, обеспечивающие избыточность для повышения надежности, например, кластеры с несколькими узлами.

2. Файловая система. Журналируемая/не журналируемая система, чем отличаются. Как файл представлен в ФС. Какую информацию о файле хранит ФС? Что хранится в /proc, /sys, /dev?

- **Журналируемая файловая система** — сохраняет изменения в журнале, что позволяет восстанавливать систему после сбоя (например, ext4, XFS).
- **Не журналируемая файловая система** — не сохраняет промежуточные изменения, восстановление после сбоя сложнее (например, FAT).
- **Файл в ФС** представлен как набор данных, идентифицируемых через имя и метаданные (права, владелец, время создания).
- **Информация о файле**: Размер, время создания, права доступа, расположение блоков данных.
- **/proc** — виртуальная файловая система, предоставляющая информацию о процессах, системных данных (например, /proc/cpuinfo).
- **/sys** — виртуальная файловая система, которая позволяет взаимодействовать с ядром ОС и конфигурировать устройства.
- **/dev** — директория, содержащая файлы устройств (например, /dev/sda — жесткий диск).

3. Лимиты процесса, какие есть, как можно изменить/посмотреть? Что содержит в себе процесс? Какую информацию о нем хранит ОС? Что размещается в памяти процесса?

- Лимиты: использование процессора, памяти, количество открытых файлов и т. д. Можно посмотреть с помощью команды ulimit или prlimit.
- Процесс включает в себя инструкции, данные, стек и регистры. ОС хранит информацию в структуре task_struct (например, состояние процесса, PID, память).
- В памяти процесса размещаются: код, данные (переменные), стек и куча (динамическое выделение памяти).

4. Что означает mode в утилизации процессора/cpu time? (User, system, etc…)

- **User**: Время, потраченное на выполнение пользовательского кода.
- **System**: Время, потраченное на выполнение системных вызовов ядра.
- **Idle**: Время, когда процессор не занят.
- **I/O Wait**: Время, когда процесс ожидает завершения операции ввода/вывода.

5. Отличие контейнера от виртуальной машины? Что происходит при запуске контейнера?

- **Контейнер** использует ядро хостовой ОС, а виртуальная машина запускает собственное ядро. Контейнеры быстрее и легче, чем виртуальные машины.
- При запуске контейнера создается изолированная среда с доступом к нужным ресурсам и библиотекам хостовой ОС.

6. Что такое docker image? Из чего состоит, что содержит? Можно ли собрать образ без dockerfile?

- **Docker Image** — это шаблон для создания контейнеров. Содержит файловую систему, библиотеки, настройки, зависимости.
- Без Dockerfile можно создать образ вручную, но это неудобно и не рекомендуется.
- Базовые образы создаются официальными командами, например, FROM ubuntu.

7. За счет чего происходит изоляция контейнера? Что можно изолировать? Что и от чего изолируется?

- Изоляция происходит через механизмы cgroups и namespaces.
- Можно изолировать: файловую систему, процессы, сеть, пользователей, устройства.
- Контейнеры изолированы от хостовой ОС и других контейнеров.

8. Как происходит остановка контейнера? Что такое сигналы? В чем разница между сигналы, особенности? Какой именно сигнал будет отправлен контейнеру и можно ли этим управлять?

- Контейнер останавливается с помощью сигнала SIGTERM. Если контейнер не завершится в течение времени ожидания, отправляется SIGKILL.
- **SIGTERM** — запрос на завершение работы. **SIGKILL** — немедленное завершение.

9. Зачем нужен docker compose? Какие задачи решает? Что предоставляет? Как работает изоляция сети?

- **Docker Compose** используется для управления многоконтейнерными приложениями. Предоставляет возможность запускать, останавливать и настраивать несколько контейнеров с помощью одного конфигурационного файла.
- Изоляция сети осуществляется через создание отдельной сети для контейнеров.

10. Какие сущности ansible за что отвечают? (inventory, playbook, etc…). Из чего состоит роль? Чем отличается от playbook-а. Что такое handler? Когда запускается? Можно ли управлять временем его запуска? Что такое идемпотентность?

Ansible — это система автоматизации и управления конфигурациями, которая использует декларативный подход для описания состояния системы.

Сущности Ansible:

- **Inventory**: Это файл или динамическая сущность, в которой описываются хосты (сервера), на которых будут выполняться задачи. В inventory указываются IP-адреса или доменные имена хостов, группы хостов и дополнительные переменные.
- **Playbook**: Это набор сценариев (плейбуков) в формате YAML, в которых описаны задачи для исполнения на хостах. Playbook состоит из списка **плей** (один или несколько), где каждый плей описывает, какие задачи (tasks) нужно выполнить на группе хостов. Playbook обычно используется для выполнения сложных, многошаговых процессов.
- **Task**: Одиночная задача, которая выполняет конкретное действие, например, установку пакета, запуск сервиса или изменение конфигурации.
- **Role**: Роль — это способ структурирования playbook, организующий код в модули (директории). Каждая роль включает в себя задачи, шаблоны, переменные, файлы и т. д. Роли помогают повторно использовать код и обеспечивают лучшую организацию, делая его более удобным для повторного использования.
    - Роль включает в себя:
        - **Tasks** (Задачи)
        - **Handlers** (Обработчики)
        - **Templates** (Шаблоны)
        - **Files** (Файлы)
        - **Defaults** (Переменные по умолчанию)
        - **Vars** (Переменные)
- **Handler**: Это специальная задача, которая запускается в ответ на изменение состояния системы. Обработчики используются для того, чтобы выполнить определённые действия только в том случае, если произошли изменения, например, перезапуск сервиса после изменения конфигурации. Handler запускается только в конце выполнения playbook после выполнения всех задач.
- **Когда запускается handler?** Он запускается только если одна из задач была изменена, и его вызов был явным образом определен через ключевое слово notify.
- **Можно ли управлять временем запуска handler?** Да, можно использовать условные выражения для определения, когда будет выполняться handler, но

его выполнение всегда будет происходить в конце текущего playbook (после выполнения всех задач).

Идемпотентность: Это свойство операционной системы или инструмента, при котором многократное выполнение одной и той же операции не приводит к изменению состояния системы, если состояние уже соответствует требуемому. В контексте Ansible это означает, что если задача уже выполнена и не требует изменений, то Ansible не будет повторно её выполнять, предотвращая ненужные изменения. Идемпотентность является одной из ключевых характеристик Ansible, что делает его удобным для автоматического управления конфигурациями.

Конечно, продолжим с 11-го вопроса:

11. Какие виды распределения нагрузки вам известны? Примеры, отличия, преимущества и недостатки? На каких уровнях модели OSI какие способы распределения нагрузки существуют?

- **Распределение нагрузки (load balancing)** — это процесс распределения входящего трафика на несколько серверов для улучшения производительности, отказоустойчивости и масштабируемости.
- **Виды распределения нагрузки**:
    - **DNS балансировка**: Распределяет запросы по разным серверам на уровне DNS. Преимущества: простота. Недостатки: не всегда эффективна для сложных приложений.
    - **Логическая балансировка**: Распределение на уровне приложения. Например, использование алгоритмов (Round-robin, Least Connections, IP Hash и др.).
    - **Сетевой балансировщик (Layer 4)**: Работает на уровне TCP/UDP, распределяя запросы по серверам на основе их IP-адреса и порта. Преимущества: быстрая обработка. Недостатки: не поддерживает сложную логику маршрутизации.
    - **Балансировка на уровне приложения (Layer 7)**: Работает на уровне HTTP/HTTPS, анализирует запросы и решает, на какой сервер направить трафик. Преимущества: высокая гибкость, возможность использовать более сложные алгоритмы. Недостатки: может быть более медленным.
- **Примеры**:
    - **Round Robin**: Равномерное распределение запросов по серверам.
    - **Least Connections**: Направляет запросы на сервер с наименьшим количеством активных соединений.
    - **IP Hash**: Направляет запросы с одного IP-адреса на один и тот же сервер.

12. Как можно обеспечить отказоустойчивость балансировщиков? Как при распределении нагрузки гарантировать, что пользователь всегда будет попадать на один и тот же backend сервер?

- Для обеспечения отказоустойчивости балансировщиков можно использовать:
    - **Группы резервирования (Failover)**: Если один балансировщик выходит из строя, запросы перенаправляются на другой.

    - **Health checks**: Регулярные проверки состояния серверов, чтобы направить трафик только на работоспособные.

- **Гарантированное направление трафика на тот же сервер (Sticky sessions или session affinity)**:

    - Это достигается с помощью cookies или на основе IP-хеша, чтобы пользователь всегда попадал на один и тот же сервер в рамках сессии.

13. Из каких компонентов состоит kubernetes? Как связаны между собой Deployment, Container, Pod, ReplicaSet? Что такое Pod? Из чего состоит? Requests/Limits. Probes. Для чего применяются, как определяются, на что влияют.

- **Kubernetes** состоит из нескольких компонентов:

    - **Node**: Рабочая единица Kubernetes, где выполняются контейнеры.
    - **Pod**: Это наименьшая единица развертывания в Kubernetes, содержащая один или несколько контейнеров, которые работают на одном Node и разделяют ресурсы.
    - **ReplicaSet**: Управляет количеством реплик Pods, чтобы обеспечить необходимое количество экземпляров приложения.
    - **Deployment**: Более высокоуровневая сущность, которая управляет развертыванием и обновлением ReplicaSet.

- **Requests**: Минимальное количество ресурсов, которое контейнер должен получить (например, CPU и память).
- **Limits**: Максимальное количество ресурсов, которое контейнер может потребить.
- **Probes**: Проверки жизнеспособности (liveness) и готовности (readiness) контейнера. Liveness проверяет, жив ли контейнер, а readiness проверяет, готов ли он принимать трафик.

14. Хранение данных в K8S. Хранение данных в docker compose. Как обеспечить сохранность данных между перезапусками? Политики доступа к хранилищу в k8s? Что такое CSI? В чем особенность StatefulSet в сравнении с Deployment?

- В **Kubernetes** данные можно хранить через:

    - **Volumes**: Абстракция для хранения данных, которая может быть прикреплена к Pod.
    - **Persistent Volumes (PV)** и **Persistent Volume Claims (PVC)**: Позволяют отделить жизненный цикл хранилища от жизненного цикла Pod.

- В **Docker Compose** для сохранности данных используется volumes, которые сохраняют данные между перезапусками контейнеров.
- **CSI (Container Storage Interface)**: Стандарт для подключения различных типов хранилищ к контейнерам.
- **StatefulSet** используется для развертывания состояния, например, для баз данных, в отличие от Deployment, который используется для бесстатичных приложений. StatefulSet сохраняет порядок развертывания и гарантирует, что при перезапуске контейнеры получат тот же идентификатор и хранилище.

15. Какие виды Service бывают? Для чего предназначены? Когда начнет поступать трафик через Service на вновь созданные Pod-ы? Какие условия должны выполняться? Какую задачу решает CNI? Какие технологии использует в работе?

- В **Kubernetes** есть следующие виды Service:
    - **ClusterIP**: Доступ к сервису внутри кластера.
    - **NodePort**: Доступ к сервису через каждый узел кластера по конкретному порту.
    - **LoadBalancer**: Доступ через внешний балансировщик нагрузки (обычно используется в облачных средах).
    - **Headless Service**: Без IP-адреса, используется для сервисов, требующих прямого доступа к каждому экземпляру Pod.
- Трафик через Service начнет поступать на новые Pods, как только они станут "Ready" (по проверкам readiness).
- **CNI (Container Network Interface)** решает задачи сетевой интеграции контейнеров, обеспечивая связь между ними и управляя сетевыми ресурсами. В Kubernetes используется несколько реализаций CNI (например, Calico, Flannel).

16. Что такое SLA/SLO/SLI ? Как между собой связаны? Какие варианты мониторинга приложений/серверов вам известны? По какой схеме получения метрик работает Prometheus? Какие компоненты содержит стек Prometheus?

- **SLA (Service Level Agreement)** — соглашение об уровне сервиса, описывает минимальные гарантии.
- **SLO (Service Level Objective)** — целевой показатель уровня сервиса, то, чего хотят достичь.
- **SLI (Service Level Indicator)** — показатель уровня сервиса, измеряет фактические показатели.
- **Мониторинг приложений и серверов** может быть реализован через:
    - **Prometheus** — система мониторинга и сбора метрик.
    - **Grafana** — для визуализации данных.
- **Prometheus** работает на основе метрик, которые собираются через **pull**-модель (сервисы регистрируются как "targets" в Prometheus, и Prometheus "запрашивает" метрики).
- Стек Prometheus состоит из:
    - **Prometheus server**: Сбор и хранение метрик.
    - **Alertmanager**: Управление уведомлениями.
    - **Grafana**: Визуализация данных.

17. Как обеспечить согласованность работы экземпляров распределенного приложения? Какие условия должны выполняться? Какие протоколы или приложения позволяют этого достичь?

- Для обеспечения согласованности требуется использование распределенных транзакций или алгоритмов согласования. Протоколы:

- **Paxos** и **Raft**: Используются для достижения консенсуса в распределенных системах.
- **Two-Phase Commit (2PC)**: Используется для согласования транзакций.
- **Event Sourcing**: Моделирует приложение через события, что позволяет отслеживать и восстанавливать состояния.

18. Системы хранения данных. Какие распределенные файловые системы известны? Какие типы дисков и интерфейсы? Для чего это нужно?

- Известны следующие распределенные файловые системы:
  - **HDFS** (Hadoop Distributed File System)
  - **Ceph**
  - **GlusterFS**
- Типы дисков:
  - **HDD** — механические жесткие диски.
  - **SSD** — твердотельные диски.
- Интерфейсы:
  - **SATA**, **SAS**, **NVMe** — интерфейсы для подключения дисков.

19. Виды RAID - плюсы, минусы, ограничения? (1, 0, 5, 6 и их комбинации). ZFS, LVM – что это, из чего состоит, что предоставляет? Какие ограничения имеет?

- **RAID 1** — зеркалирование, высокая отказоустойчивость, но низкая производительность.
- **RAID 0** — чередование, высокая производительность, но без отказоустойчивости.
- **RAID 5** — чередование с избыточностью, хорошая отказоустойчивость.
- **RAID 6** — как RAID 5, но с двумя дисками избыточности.
- **ZFS** — файловая система с высокой производительностью, поддерживает дедупликацию и снэпшоты.
- **LVM** — система управления логическими томами, позволяет динамически управлять хранилищем.

20. HDFS – из чего состоит, какую информацию хранят компоненты системы, как хранится файл? Lizard FS – из чего состоит, какую информацию хранят компоненты системы, как хранится файл?

- **HDFS** состоит из **NameNode** (управляет метаданными) и **DataNode** (хранит данные).
- Файл разбивается на блоки, которые сохраняются на различных DataNode.
- **LizardFS** — распределенная файловая система, аналогичная HDFS, с использованием блоков для хранения данных и центрального узла для метаданных.

21. CEPH – из чего состоит, какую информацию хранят компоненты системы, как хранится файл? Что такое CRUSH в Ceph? Placement group? Что такое S3 совместимое хранилище? Как с ним работать?

- **Ceph** состоит из следующих компонентов:
    - **Monitor (MON)**: Отвечает за хранение и распространение информации о состоянии кластера (например, какие OSD активны, какие данные на каких устройствах).
    - **Object Storage Daemon (OSD)**: Компонент, который фактически хранит данные в кластере. Это процесс, который управляет устройствами хранения и непосредственно занимается операциями чтения и записи.
    - **Metadata Server (MDS)**: Служит для хранения метаданных файловой системы CephFS. Это необходимо для работы с данными, размещенными на файловой системе Ceph.
    - **Ceph Manager**: Управляет мониторингом, отчётами, настройками и также обрабатывает метаданные для кластеров.
- Файл в Ceph хранится как объект, который разбивается на несколько блоков и распределяется по различным OSD, чтобы обеспечить отказоустойчивость и масштабируемость.
- **CRUSH** (Controlled Replication Under Scalable Hashing) — это алгоритм, который в Ceph используется для вычисления, где будет храниться объект на основе его хэшированного значения. CRUSH решает, как и где должны размещаться данные, и позволяет эффективно распределять их по кластеру.
- **Placement group** (PG) — это группа объектов, которые управляются одним или несколькими OSD и служат для распределения данных по OSD на основе алгоритма CRUSH. Каждый объект в кластере Ceph может быть размещён в одной или нескольких PG.
- **S3-совместимое хранилище** — это тип объектного хранилища, которое использует API, совместимый с AWS S3, позволяя взаимодействовать с данными с помощью стандартных инструментов, таких как s3cmd, или через сторонние приложения, использующие S3-протокол. Ceph предоставляет S3-совместимый интерфейс через модуль **Rados Gateway (RGW)**.

Для работы с S3-совместимым хранилищем в Ceph можно использовать такие инструменты, как aws-cli, или библиотеку для Python boto3, для взаимодействия с объектами хранилища через API, поддерживающий такие операции, как загрузка, удаление и скачивание объектов.