**Но зачем нам балансировка нагрузки, если у нас только один сервер? Верно - для масштабирования**

**Масштабирование — это способ увеличения пропускной способности** и отказоустойчивости системы путем добавления дополнительных ресурсов или оптимизации существующих. Но масштабирование не всегда тривиальная задача. Оно касается не только увеличения количества серверов или улучшения их характеристик, но и решения сложных вопросов, связанных с обеспечением надежной работы системы в новых условиях.

Представьте, что у вас есть небольшой интернет-магазин. В начале он легко обходится одним сервером, который обрабатывает все запросы и хранит всю информацию. Но со временем магазин становится популярным, количество клиентов и, соответственно, запросов к серверу возрастает. В этот момент начинаются проблемы: сайт замедляется, появляются сбои, клиенты уходят.

**Варианты масштабирования**

- **Горизонтальное масштабирование**: Это как размножение вашего единственного сервера интернет-магазина. Каждый новый сервер занимается той же функцией, что и оригинальный. Пользовательским запросам теперь уделяется больше ресурсов, но проблема в том, как эффективно распределять нагрузку между серверами и как синхронизировать данные между ними. Обычно все приходят к этому варианту, так как вертикальное масштабирование имеет свои ограничения.
- **Вертикальное масштабирование**: Это как замена двигателя в вашем единственном сервере на более мощный. Вы повышаете производительность, добавляя ресурсы (CPU, RAM и т.д.) на существующую машину. Однако этот метод имеет свои ограничения в виде аппаратных возможностей сервера - пока что в реальном мире нет бесконечных мощностей :)

**Проблемы**

Масштабирование сложных систем представляет особую проблематику. Вернемся к примеру интернет-магазина: если у вас есть несколько серверов, как гарантировать, что клиент, добавивший товар в корзину на одном сервере, увидит его в корзине, когда вернется на сайт и попадет на другой сервер? Тут уже не обойтись без механизмов синхронизации и балансировки нагрузки.

В случае еще более сложных систем, например социальных сетей, проблемы масштабирования становятся еще более непростыми. Разделить такую систему на независимые части почти невозможно, так как данные и функциональность сильно переплетены (например, один и тот же пользователь может иметь друзей в разных "частях" системы).

Вот почему масштабирование — это не просто увеличение количества ресурсов, это искусство обеспечения качественной работы системы при увеличении нагрузки и сложности. Понимание особенностей вашего приложения и данных, которые оно обрабатывает, позволяет выбрать наиболее подходящие методы масштабирования.

**Горизонтальное масштабирование** — это техника увеличения мощности и пропускной способности системы путем добавления дополнительных экземпляров серверов или других компонентов. Вместо усиления одного сервера (как в вертикальном масштабировании), вы добавляете больше машин с аналогичной конфигурацией. Давайте рассмотрим, как это работает на разных уровнях.

**На уровне сети идёт работа балансировщиков**

Балансировщики нагрузки являются ключевыми компонентами при горизонтальном масштабировании. Они принимают входящие сетевые запросы и распределяют их между доступными серверами на основе различных алгоритмов (например, Round Robin, Least Connections и т.д.). Сессионное привязывание и другие технологии могут использоваться для обеспечения консистентности данных между запросами.

**На уровне уже серверов приложений идёт обработка запросов от балансировщика на определённую копию приложения.**

Горизонтальное масштабирование в основном включает в себя добавление дополнительных экземпляров (или "копий") вашего приложения на новых серверах, чтобы распределить нагрузку. Я попробую описать процесс на примерах:

**Пример с фронтендом:**

1. **Изначальная настройка**: У вас есть один сервер, на котором работает ваш фронтенд-сервис. Пользователи обращаются к этому серверу по определенному IP-адресу или доменному имени.

2. **Добавление серверов**: Когда нагрузка увеличивается, вы добавляете еще один или несколько серверов. На этих серверах запускается точно такое же фронтенд-приложение.

3. **Синхронизация кода**: Весь код и ресурсы фронтенда должны быть идентичны на всех серверах. Обычно это делается с помощью систем автоматического развертывания, таких как Kubernetes, Docker Swarm или просто скриптов автоматизации.

4. **Тестирование**: Перед тем, как направить на новые серверы реальный трафик, проводится тестирование, чтобы удостовериться, что все серверы функционируют одинаково.

5. **Обновление и мониторинг**: Все серверы должны быть постоянно обновлены и мониториться для удостоверения, что они функционируют идентично.

**Пример с бэкендом:**

1. **Изначальная настройка**: У вас есть один сервер, на котором работает ваш бэкенд-сервис, например, REST API.

2. **Добавление серверов**: Когда нагрузка увеличивается, вы добавляете дополнительные серверы, на которых работает копия вашего бэкенд-приложения.

3. **Синхронизация кода и данных**: Весь код и возможно, некоторые данные (если они не хранятся в централизованной БД или хранилище), должны быть идентичны на всех серверах.

4. **Тестирование**: Проведение тестов для подтверждения, что все серверы обрабатывают запросы корректно.

5. **Обновление и мониторинг**: Все серверы должны быть постоянно обновлены и мониториться для удостоверения, что они функционируют идентично.

Обратите внимание, что для того, чтобы горизонтально масштабировать сервисы, их архитектура должна быть "stateless", то есть не зависеть от локального состояния сервера. Это позволяет каждой копии приложения обрабатывать любой запрос независимо от других копий.

Ключевой момент здесь в том, что каждая "копия" приложения на новом сервере работает автономно и идентично оригинальному экземпляру. Новые серверы добавляются или убираются динамически в зависимости от нагрузки, и это обычно управляется с помощью систем оркестрации, таких как Kubernetes.