


8. Шаблоны облачной архитектуры

Что такое шаблон?

Шаблон - это метод проектирования (подход), который можно применить для получения ожидаемого результата. Шаблоны влияют на архитектуру приложения и нацелены на снижение рисков и стоимости, на защиту приложения от большей части проблем разработки. Шаблоны архитектуры, описанные ниже, полезны для создания облачных приложений.



Шаблоны облачных приложений

- Горизонтальное масштабирование вычислений
- Модель рабочего процесса, ориентированного на очередь
- Шаблон автоматического масштабирования
- Шаблон MapReduce
- Шаблон шардинга базы данных
- Многопрофильность
- Занятый сигнал
- Шаблон прерывания узла
- Колокатный паттерн
- Шаблон ключей камердинера
- CDN Pattern
- Многосайтовый шаблон развертывания



Данные шаблоны более
подробно описываются в
книге: Cloud Architecture
Patterns by Bill Wilder



1. Масштабируемость

Масштабируемость приложения - это показатель количества пользователей, которых приложение может одновременно эффективно поддерживать. Точка, в которой приложение не может эффективно обрабатывать дополнительных пользователей, является пределом его масштабируемости.



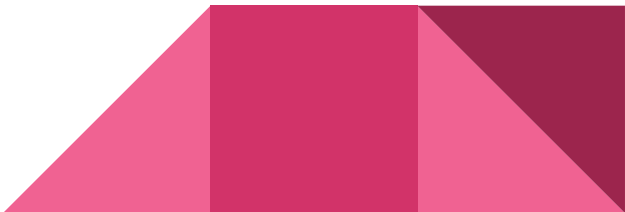
Виды масштабируемости

- 1) *Вертикальная масштабируемость*: увеличение общей емкости приложений за счет увеличения объема ресурсов в рамках существующих узлов;
- 2) *Горизонтальная масштабируемость*: увеличение общей пропускной способности приложений за счет добавления узлов.



Правила масштабируемости

Масштабирование ограничено используемыми возможностями доступного оборудования. Когда все узлы, поддерживающие определенную функцию, настроены одинаково - одни и те же аппаратные ресурсы, одна и та же операционная система, одно и то же программное обеспечение для конкретной функции - мы говорим, что эти узлы *однородны*.




Горизонтальная масштабируемость

Каждый дополнительный узел обычно добавляет эквивалентную емкость, такую как тот же объем памяти и тот же процессор. Приложения, разработанные для горизонтального масштабирования, обычно имеют узлы, выделенные для определенных функций. Горизонтальное масштабирование более эффективно с однородными узлами.




Термины масштабируемости

- *Одновременные пользователи*: количество пользователей с активностью в течение определенного промежутка времени (например, десять минут);
 - *Время ответа*: истекшее время между пользователем, инициирующим запрос (например, нажав кнопку), и получающим ответ в оба конца;
 - *Единица масштаба*: комбинация ресурсов, которые необходимо масштабировать вместе.
- 

Нехватка ресурсов

Проблемы с масштабируемостью являются проблемами нехватки *ресурсов*. Ограничивает масштабируемость не количество одновременно работающих пользователей, а конкурирующие требования к ограниченным ресурсам, таким как ЦП, память и пропускная способность сети. Это приводит к замедлению или блокировке некоторых пользователей. Они называются *узкими местами ресурсов*.



Разрешение конфликта ресурсов

- Улучшение алгоритма помогает обрабатывать больше транзакций в секунду;
- Улучшение оборудования.




Масштабируемость влияет на производительность, а эффективность влияет на масштабируемость. Два общих шаблона масштабирования - вертикальное и горизонтальное масштабирование. Вертикальное масштабирование, как правило, легче реализовать, хотя оно более ограничено, чем горизонтальное масштабирование. Собственные облачные приложения распределяют ресурсы горизонтально, а масштабируемость — это только одно преимущество.



2. Горизонтальное масштабирование вычислений

Ключом к эффективному использованию ресурсов являются автономные вычислительные узлы без сохранения состояния. Важное состояние может храниться вне узлов в облачном кэше или службе хранения, что для веб-уровня обычно выполняется с помощью файлов cookie.



3. Модель рабочего процесса, ориентированного на очередь

Этот важный шаблон для слабой связи сосредотачивается на асинхронной доставке командных запросов, отправляемых из пользовательского интерфейса в серверную службу для обработки. Этот шаблон является подмножеством шаблона CQRS.




Модель рабочего процесса, ориентированного на очередь

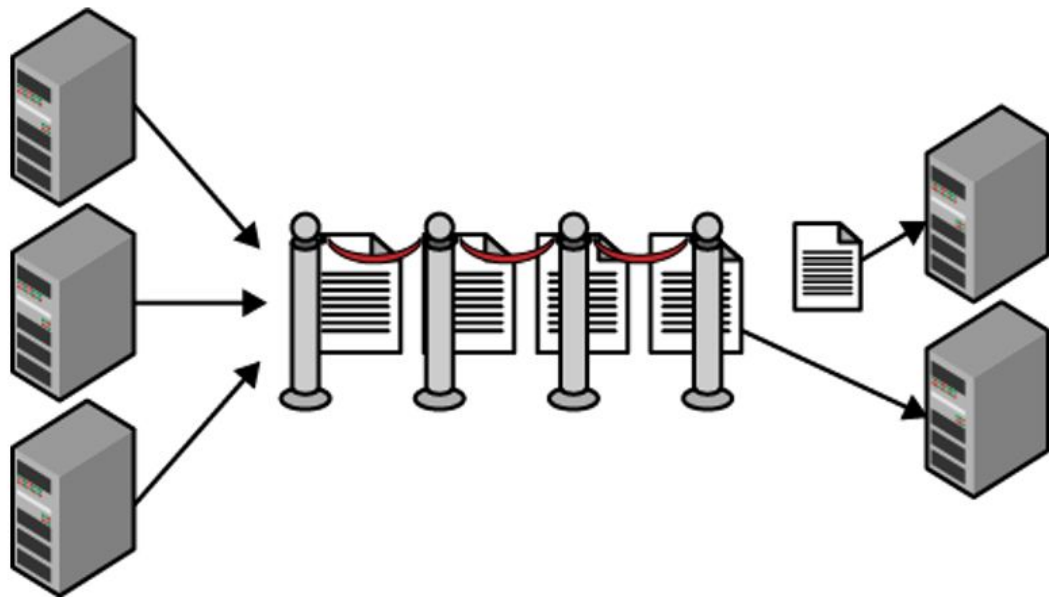
Шаблон используется для того, чтобы интерактивные пользователи могли обновлять веб-уровень без замедления веб-сервера. Это особенно полезно для обработки обновлений, которые требуют много времени, ресурсов или зависят от удаленных служб, которые могут быть не всегда доступны.



Ориентированный на очередь шаблон рабочего процесса используется в веб-приложениях для разделения связи между веб-уровнем (который реализует пользовательский интерфейс) и уровнем обслуживания (где происходит бизнес-обработка). Веб-уровень отправляет Команды для уровня обслуживания, где команда - это запрос на выполнение чего-либо. Примеры команд: создать новую учетную запись пользователя, добавить фотографию, обновить статус (например, в Twitter или Facebook), забронировать номер в отеле и отменить заказ.



Веб-уровень добавляет сообщения в очередь. Сервисный уровень удаляет и обрабатывает сообщения из очереди.



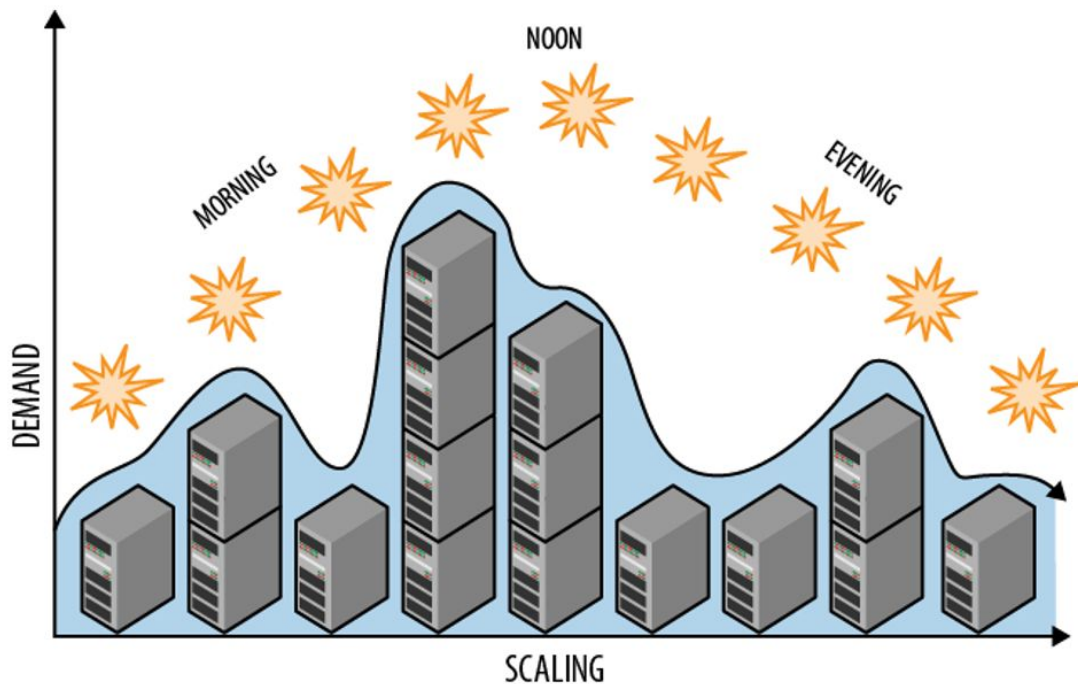
4. Шаблон автоматического масштабирования

Масштабирование вручную с помощью веб-инструмента управления вашего поставщика облачных вычислений помогает снизить ежемесячный счет за облачные вычисления, но автоматизация может лучше выполнять оптимизацию затрат с меньшими ручными затратами на рутинные задачи масштабирования. Это также может быть более динамичным. Автоматизация может реагировать на сигналы от самого облачного сервиса и реагировать на реальные потребности.

Вычислительные узлы являются наиболее распространенным ресурсом для масштабирования, чтобы обеспечить нужное количество узлов веб-сервера или службы. Автоматическое масштабирование может поддерживать правильные ресурсы на данный момент и может применяться к любому ресурсу, который может выиграть от автоматического масштабирования, например, к хранилищу данных и очередям. Этот шаблон ориентирован на автоматическое масштабирование вычислительных узлов.

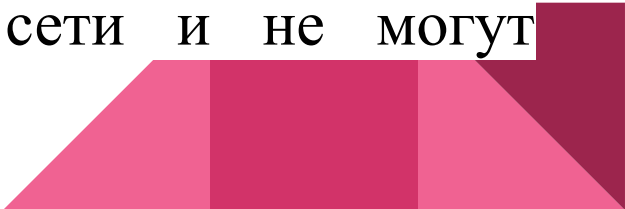


Запланированы упреждающие правила автоматического масштабирования, например, для добавления и освобождения ресурсов в течение дня по расписанию.



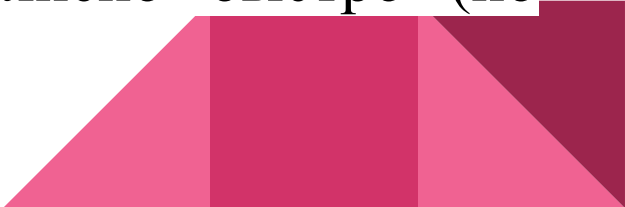
5. Теорема CAP Брюера

Гарантии для данных в распределенном приложении:

- *Последовательность*: означает, что каждый получает один и тот же ответ;
 - *Доступность*: означает, что клиенты имеют постоянный доступ (даже при частичном сбое системы);
 - *Допуск раздела*: означает правильную работу, даже если узлы в приложении отключены от сети и не могут обмениваться данными.
- 


6. Шаблон MapReduce

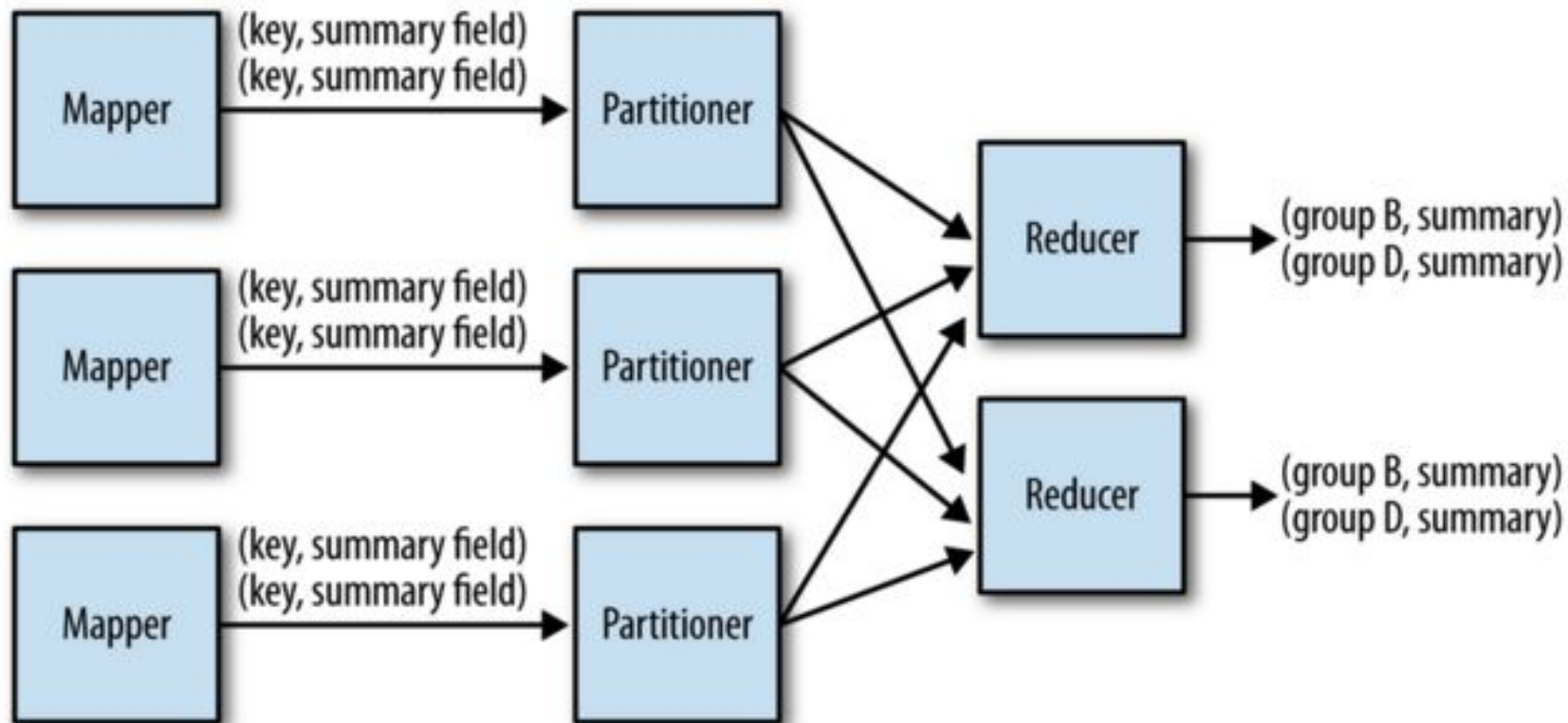
Подход к обработке данных, представляющий простую модель программирования для обработки высокопараллелизуемых наборов данных. Он реализован в виде кластера, в котором множество узлов работают параллельно над разными частями данных. При запуске задания MapReduce возникают большие накладные расходы, но после начала задание может быть выполнено быстро (по сравнению с обычными подходами).



Шаблон MapReduce


MapReduce требует написания двух функций: картографа и редуктора. Эти функции принимают данные как входные данные и затем возвращают преобразованные данные как выходные данные. Функции вызываются многократно с подмножествами данных, при этом выходные данные преобразователя агрегируются и затем отправляются в редуктор. MapReduce предназначен для пакетной обработки наборов данных.





Hadoop

Структура для применения карты и сокращения функций для произвольно больших наборов данных. Данные делятся на небольшие блоки, которые распределены по кластеру узлов данных. Типичный кластер Hadoop содержит от нескольких до сотен узлов данных. Каждый узел данных получает подмножество данных и применяет карту и сокращает функции к локально сохраненным данным в соответствии с инструкциями средства отслеживания заданий, которое координирует задания по всему кластеру.



Hadoop и MapReduce

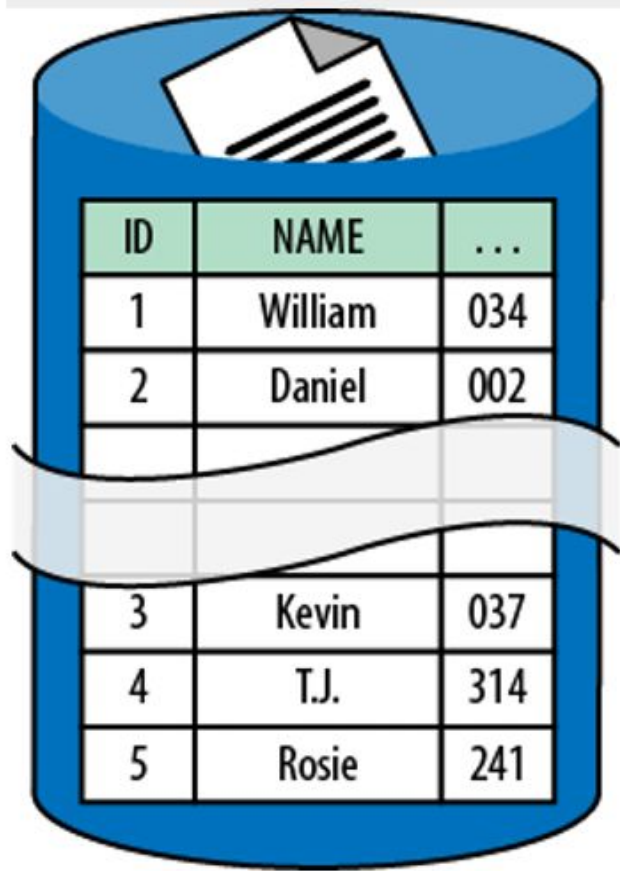
Использование MapReduce через службу облачной платформы Hadoop позволяет арендовать экземпляры за короткий промежуток времени. Поскольку кластер Hadoop может включать в себя множество вычислительных узлов, даже сотни, экономия может быть значительной. Это особенно удобно, когда данные хранятся в облачных сервисах хранения, которые хорошо интегрируются с Hadoop.



7. Шаблон шардинга базы данных

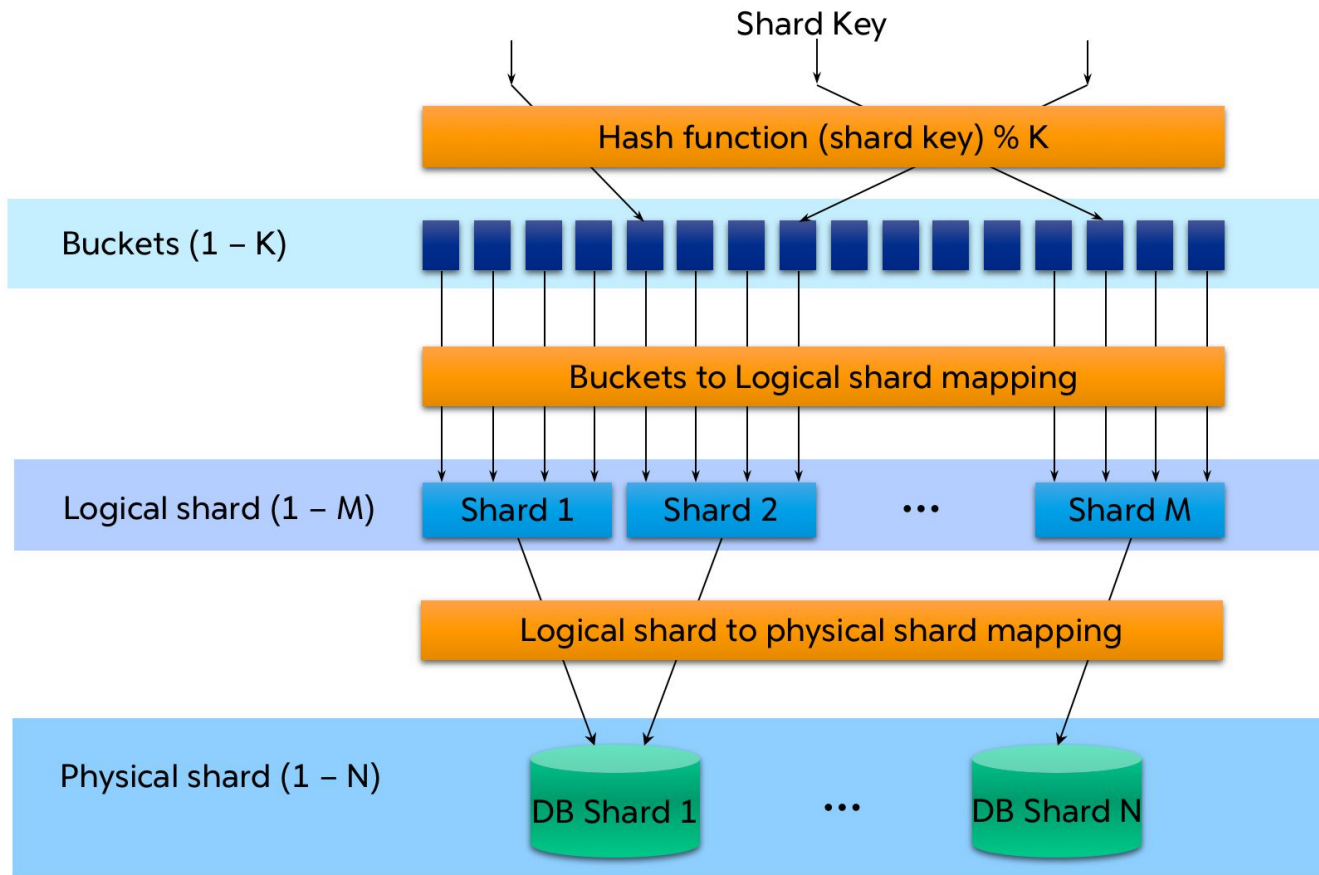
Разделить базу данных - значит начать с одной базы данных, а затем разделить ее данные на две или более баз данных (сегментов). Каждый шард имеет ту же схему базы данных, что и исходная база данных. Большая часть данных распределяется таким образом, что каждая строка отображается ровно в одном фрагменте. Объединенные данные из всех сегментов совпадают с данными из исходной базы данных.






ID	NAME	...
1	William	034
2	Daniel	002
3	Kevin	037
4	T.J.	314
5	Rosie	241

Строки данных распределены по осколкам, сохраняя при этом ту же структуру. Изображены два шарда: один шард содержит строки с ID = 1-2, другой - строки с ID = 3-5.

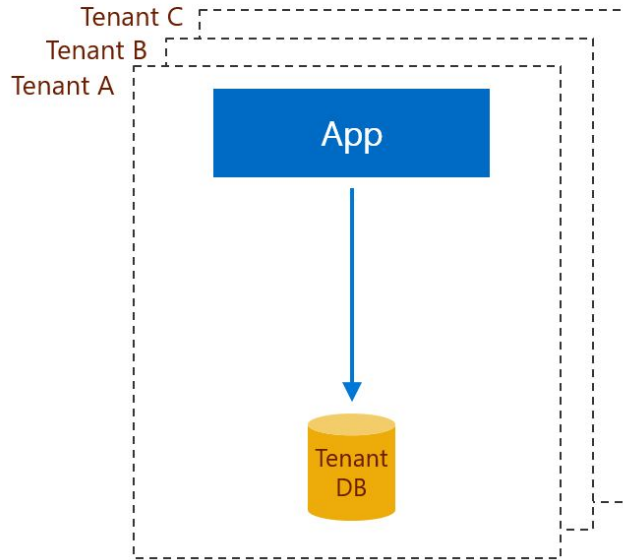


8. Многопрофильность

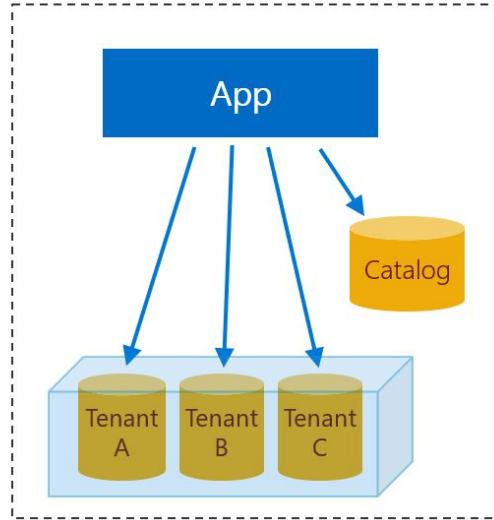
Это означает, что в системе работают несколько арендаторов. Обычно система представляет собой программное приложение, управляемое одной компанией-хостом, для использования другими компаниями-арендаторами. В каждой компании-арендаторе есть отдельные сотрудники, которые имеют доступ к программному обеспечению. Все сотрудники компании-арендатора могут быть подключены в приложении, тогда как другие арендаторы невидимы.



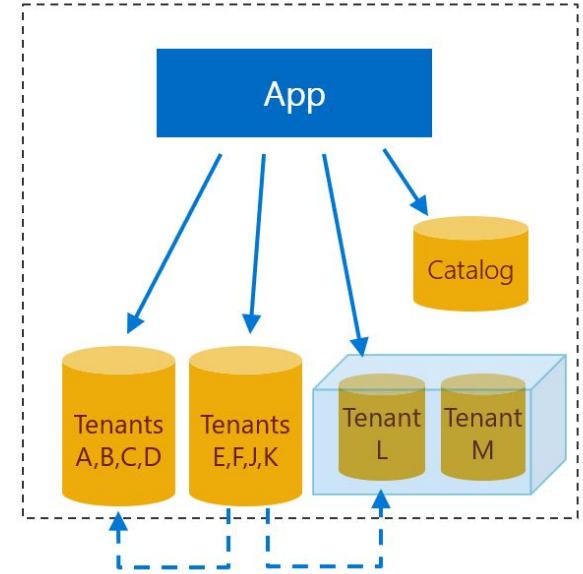
Standalone App



Database per Tenant



Sharded Multi-tenant




9. Занятый сигнал

Эта модель отражает точку зрения клиента, а не услуги. Клиент программно делает запрос службы, но служба отвечает сигналом занятости. Клиент отвечает за правильную интерпретацию сигнала занятости, за которым следует соответствующее количество попыток повторной попытки. Если сигналы занятости продолжаются во время повторных попыток, клиент рассматривает службу как недоступную.




Используйте шаблон занятости сигнала для обнаружения и обработки обычных переходных сбоев, которые возникают, когда ваше приложение (клиент в этих отношениях) получает доступ к облачной службе. Переходный отказ является кратковременным, отказ который не по вине клиента. Фактически, если клиент повторно отправляет идентичный запрос только через миллисекунды, он часто будет успешным. Временные сбои - это ожидаемые события, а не исключительные, похожие на телефонный звонок и получение сигнала занятости.



Результат переходных сбоев в занятых сигналах

Существует несколько причин сбоя запроса облачной службы: запрашивающая учетная запись слишком агрессивна, общий всплеск активности среди всех арендаторов или это может быть из-за аппаратного сбоя в облачной службе. В любом случае, служба активно управляет доступом к своим ресурсам, пытается сбалансировать взаимодействие между всеми арендаторами и даже переконфигурировать себя на лету в ответ на пики, изменения рабочей нагрузки и внутренние сбои оборудования.



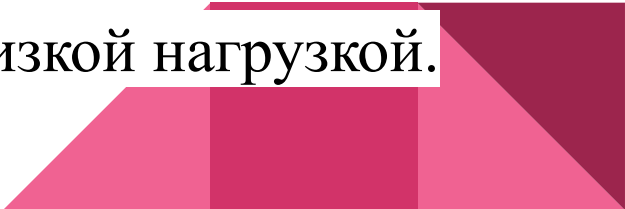
Регистрация и уменьшение занятых сигналов

Регистрация сигналов занятости может быть полезна для понимания шаблонов сбоев. Надежное отслеживание и обработка временных сбоев чрезвычайно важно в облаке из-за врожденных проблем при отладке и управлении распределенными облачными приложениями.



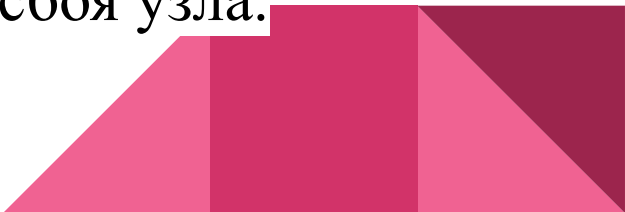
Обработка временных сбоев необходима для создания надежных облачных приложений. Используя шаблон сигнала занятости, ваше приложение может надлежащим образом обнаруживать и обрабатывать переходные сбои. Кроме того, ваш подход может быть настроен для пакетных или интерактивных пользовательских сценариев.

Может быть трудно проверить реакцию вашего приложения на переходные состояния отказа, если он работает на не облачном оборудовании или с нереально низкой нагрузкой.



10. Шаблон прерывания узла

Этот шаблон отражает перспективу кода приложения, работающего на узле (виртуальной машине), который отключен или неожиданно завершает работу из-за программных или аппаратных проблем. Приложение имеет три обязанности: подготовить приложение, чтобы минимизировать проблемы, возникающие при сбое узлов, корректно обрабатывать остановки узлов и восстанавливаться после сбоя узла.



Шаблон прерывания узла

Цель этого шаблона - дать возможность отдельным узлам выходить из строя, пока приложение в целом остается доступным. Следует рассмотреть несколько категорий сценариев отказа, но все они имеют общие характеристики для подготовки к отказу, обработки отключения узла (когда это возможно) и последующего восстановления.




Поддерживать достаточную емкость для отказа с помощью правила $N + 1$

Если для поддержки пользовательского спроса требуется N узлов, разверните $N + 1$ узлов. Один узел может выйти из строя или прерваться без воздействия на приложение. $N + 1$ правило должно рассматриваться и применяться независимо для каждого типа узла.




Обработка отключения узла

Мы можем ответственно справиться с отключением узла, но нет эквивалентной обработки для внезапного отказа узла, потому что узел просто останавливается в одно мгновение. Не все аппаратные сбои приводят к сбою узла: облачные платформы способны обнаруживать сигналы о том, что аппаратный сбой неизбежен, и могут превентивно инициировать контролируемое отключение для перемещения арендаторов на другое оборудование.



Обработка отключения узла

Мы можем ответственно справиться с отключением узла, но нет эквивалентной обработки для внезапного отказа узла, потому что узел просто останавливается в одно мгновение. Не все аппаратные сбои приводят к сбою узла: облачные платформы способны обнаруживать сигналы о том, что аппаратный сбой неизбежен, и могут превентивно инициировать контролируемое отключение для перемещения арендаторов на другое оборудование.

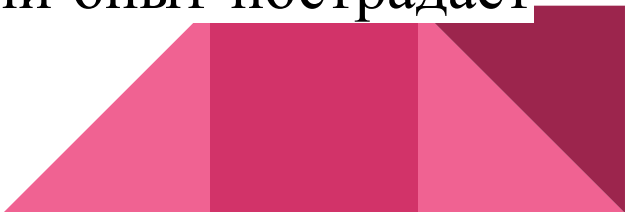


Восстановление после сбоя узла

Существуют аспекты восстановления после сбоя узла: поддержание положительного опыта пользователя в процессе сбоя и возобновление любой незавершенной работы, которая была прервана.




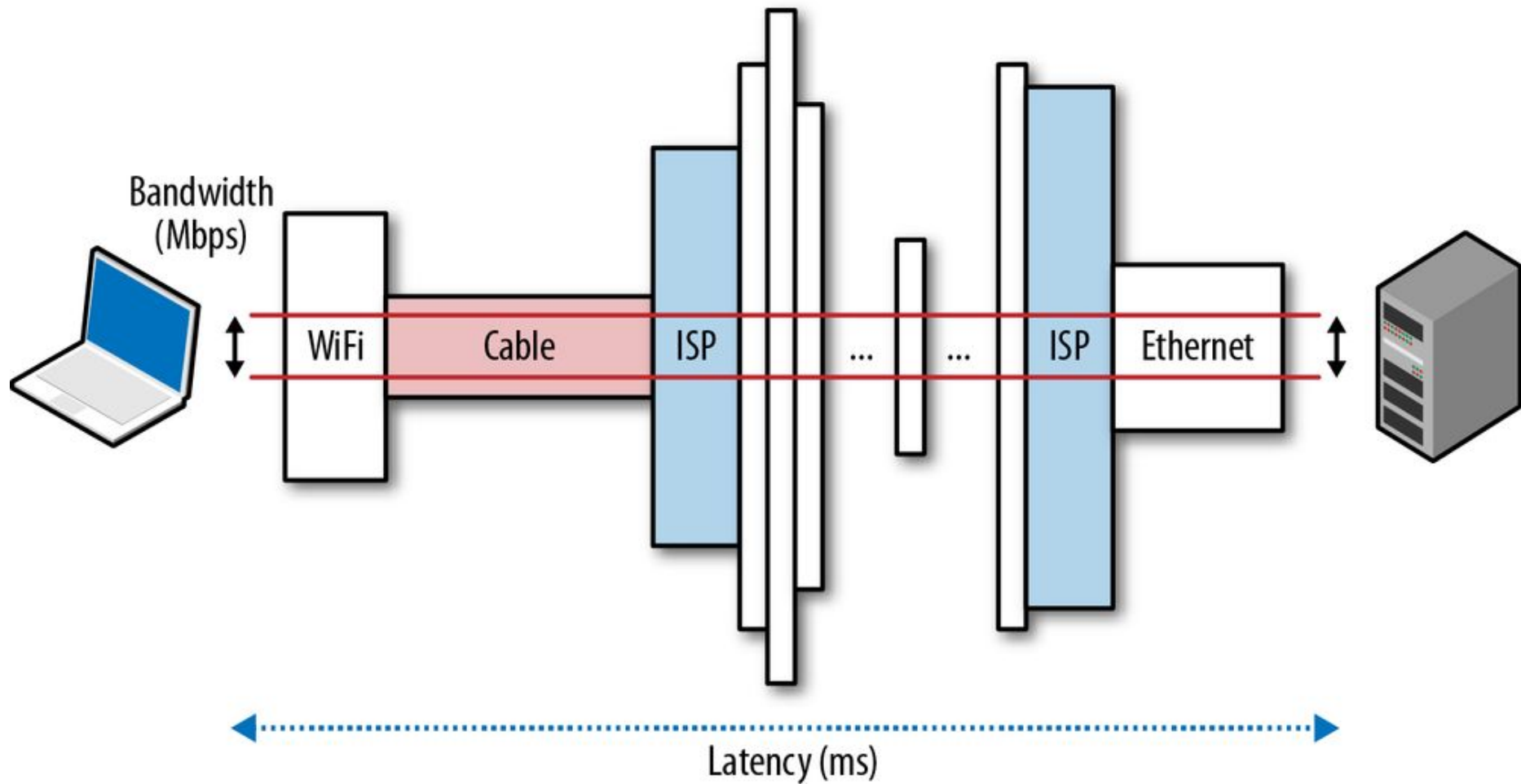
Отказ в облаке является обычным явлением, хотя время простоя редко встречается в облачных приложениях. Использование шаблона отказа узла помогает вашему приложению подготовиться, изящно обрабатывать и восстанавливаться после случайных сбоев и сбоев вычислительных узлов, на которых оно работает. Многие сценарии обрабатываются с одинаковой реализацией. Облачные приложения, которые не учитывают сценарии сбоев узлов, будут ненадежными: пользовательский опыт пострадает и данные могут быть потеряны.



11. Задержка в сети

Время, необходимое для передачи данных по сети, называется задержкой в сети. Сетевая задержка замедляет наше приложение. В то время как отдельные сетевые устройства, такие как маршрутизаторы, коммутаторы, точки беспроводного доступа и сетевые карты, имеют собственные задержки, этот пример объединяет их воедино в общий вид изображения - общая задержка, с которой сталкиваются данные, путешествующие по сети.





Колокатный паттерн

Этот базовый шаблон направлен на предотвращение ненужных задержек в сети. Связь между узлами быстрее, когда узлы находятся близко друг к другу. Расстояние добавляет задержку сети. В облаке «близко друг к другу» означает один и тот же центр обработки данных (иногда даже ближе, например, в одной стойке).




Колокатный паттерн

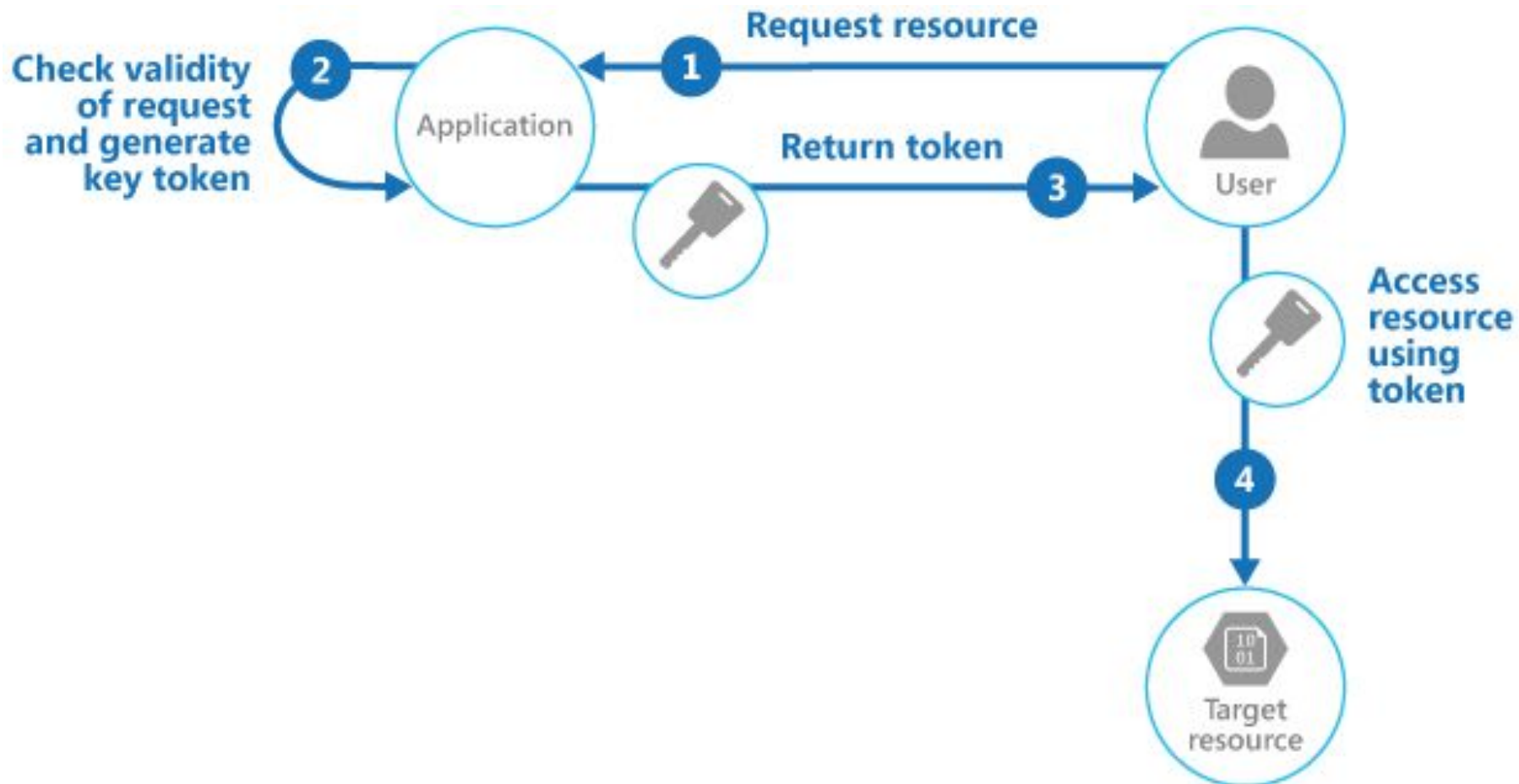
Самый простой способ начать работу в облаке - это разместить узлы, обычно все в одном центре обработки данных. Это подходит для многих приложений и должно быть обычной конфигурацией. Отклоняться только по уважительной причине и избегать ошибок случайного развертывания в нескольких центрах обработки данных, в том числе для хранения оперативных данных.



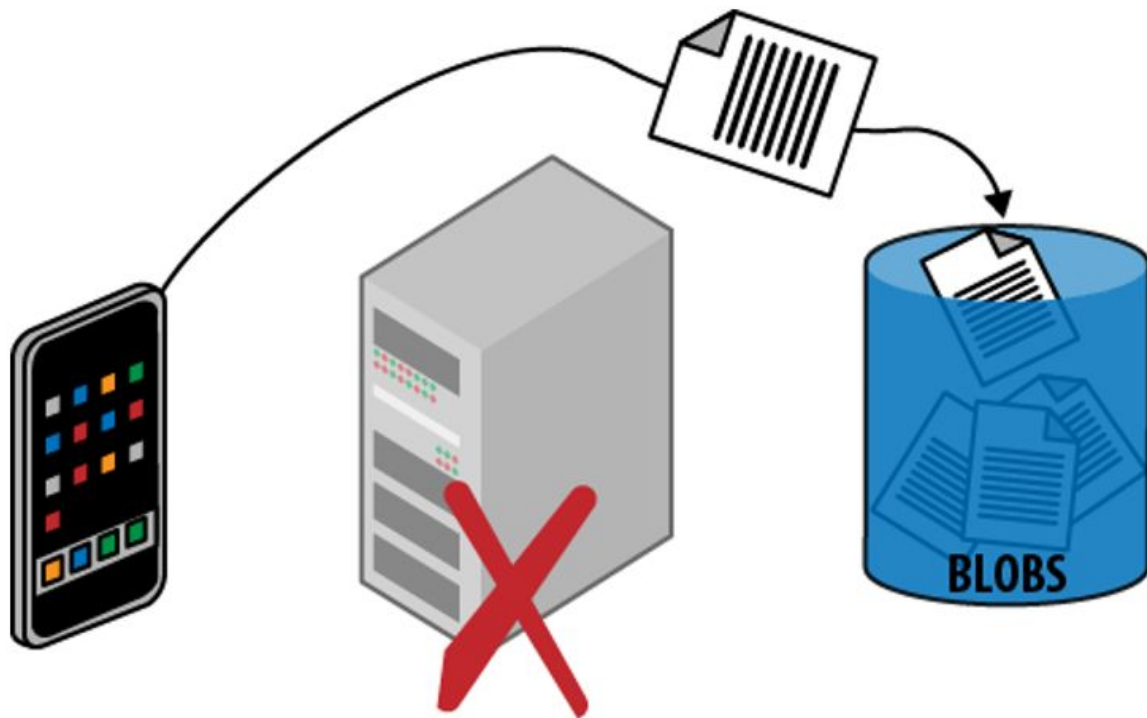
12. Шаблон ключей камердинера

Доступ к облачным сервисам хранения данных является привилегированной операцией. Как правило, вы разрешаете только доверенным подсистемам в вашем приложении иметь беспрепятственный доступ к вашим учетным записям облачного хранилища, где доверенные подсистемы работают полностью под вашим контролем на сервере. Этот шаблон ориентирован на безопасное использование хранилища больших двоичных объектов, а также на чтение и запись данных с максимальной эффективностью.





Данные загружаются эффективно, напрямую с клиента в облачное хранилище без прохождения через веб-уровень.



Использование этого шаблона следует рассматривать везде, где его можно безопасно применять. Возможность управлять временным доступом для чтения и записи делает этот шаблон широко используемым. Наиболее распространенным проблемным вариантом использования будет загрузка непосредственно из веб-браузера, но чтение хорошо поддерживается, и запись из более гибких клиентов, таких как мобильные приложения, также хорошо поддерживается.



13. CDN Pattern

Цель состоит в том, чтобы ускорить доставку контента приложения пользователям. Контент - это все, что может храниться в файле, например изображения, видео и документы. Сеть доставки контента (CDN) - это сервис, который функционирует как глобально распределенный кеш. CDN хранит копии файлов приложений во многих местах по всему миру. Когда эти места находятся близко к пользователям, контент не должен перемещаться так далеко, чтобы доставляться, поэтому он будет доставляться быстрее, улучшая пользовательский опыт.

14. Многосайтовый шаблон развертывания

Развертывание в нескольких центрах обработки данных помогает снизить задержку в сети за счет маршрутизации клиента до ближайшего центра обработки данных, что улучшает взаимодействие с пользователем. Это также обеспечивает семена решения, которое может обрабатывать аварийное переключение между центрами обработки данных и повышать доступность.

