1 Обзор основных концепций проектирования систем

В ЭТОЙ ГЛАВЕ

- ✓ Важность собеседований по проектированию систем
- ✓ Масштабирование сервисов
- ✓ Облачный хостинг или размещение на физических серверах

Собеседование по проектированию систем проходит в форме диалога между соискателем и экспертом, посвященного проектированию программной системы, обычно предоставляющей свою функциональность по сети. В начале интервью эксперт кратко и в общих чертах формулирует для кандидата задачу по проектированию конкретной программной системы. В зависимости от специфики системы база пользователей может включать нетехнических или технических пользователей.

Собеседования по проектированию систем проводятся на большинство позиций в области программной инженерии и архитектур и на руководящие инженерные должности. (В этой книге мы будем называть сотрудников, занимающих такие позиции, общим словом *инженеры*.) Помимо этого, интервью включает оценку навыков программирования и поведенческих/культурных качеств кандидата.

1.1. ОБСУЖДЕНИЕ КОМПРОМИССОВ

Следующие факторы подтверждают важность собеседований по проектированию систем и подготовки к ним в качестве соискателя и эксперта.

Проверка эффективности соискателя на собеседованиях по проектированию систем используется для оценки широты и глубины его знаний в области проектирования систем, а также его умения выстроить общение и обсуждать дизайн систем с другими инженерами. Это критический фактор для определения уровня вашей будущей должности. Важность способности проектировать и анализировать крупномасштабные системы возрастает по мере продвижения к верхним уровням инженерной иерархии. Соответственно, собеседования по проектированию систем получают больший вес при приеме на руководящие должности. Не пожалейте времени на подготовку к ним в роли как эксперта, так и соискателя — если вы планируете карьеру в технологической области, это время окупится.

У технологической отрасли есть одна уникальная особенность: инженеры обычно меняют компанию каждые несколько лет, в отличие от других отраслей, где сотрудник может оставаться в компании много лет и даже на протяжении всей карьеры. Это значит, что типичный инженер проходит собеседования по проектированию систем не один раз. Инженеры, работающие в престижных компаниях, проводят еще больше собеседований по проектированию систем в качестве эксперта. У вас как у соискателя есть меньше часа на то, чтобы произвести лучшее возможное впечатление, а кандидаты, которые являются вашими конкурентами, принадлежат к числу самых умных и мотивированных людей в мире.

Проектирование систем — скорее искусство, чем наука. Его суть не в достижении совершенства, а в принятии компромиссного варианта дизайна, который можно реализовать с имеющимися ресурсами и временем и который будет наиболее близок к текущим и возможным будущим требованиям. Все обсуждения систем в этой книге подразумевают всевозможные оценки и предположения; они не являются академически строгими, полными или научными. В книге могут упоминаться паттерны проектирования и архитектуры, но мы не будем формально описывать их принципы. За дополнительной информацией читателям следует обращаться к другим источникам.

Цель собеседования по проектированию систем не получить правильные ответы, а выявить, насколько соискатель способен обсуждать разные возможные решения и оценивать их компромиссы в отношении соответствия требованиям. Знание разных типов требований и распространенных систем, встречающихся в части 1, поможет вам проектировать систему, оценить возможные подходы и обсудить их сильные и слабые стороны.

1.2. СТОИТ ЛИ ЧИТАТЬ ЭТУ КНИГУ?

Открытая природа собеседований по проектированию систем существенно усложняет подготовку к ним, поскольку невозможно заранее узнать, как и что будет обсуждаться во время собеседования. Инженер или студент, который ищет в интернете материалы по собеседованиям по проектированию систем, найдет огромный объем контента, который сильно отличается по качеству и разнообразию рассматриваемых тем. Все это создает путаницу и усложняет обучение. Более того, до недавнего момента почти не было книг, посвященных теме собеседований по проектированию систем, — как сказал знаменитый французский поэт и романист XIX века Виктор Гюго, «идее, время которой пришло». То, что одна мысль приходит в одно время сразу многим, подтверждает ее актуальность.

В книге предлагается структурированный и упорядоченный подход, который поможет начать подготовку к собеседованию по проектированию систем или заполнить пробелы в знаниях и понимании материала, возникающие при изучении большого объема разрозненных материалов. Что не менее важно, она научит вас, как продемонстрировать в ходе собеседования свою инженерную зрелость и навыки общения, в частности умение ясно и лаконично формулировать мысли и показывать знания, а также задавать вопросы эксперту за короткое время (примерно 50 минут).

Собеседование по проектированию систем, как и любое другое, также проверяет навыки общения, сообразительность, умение задавать правильные вопросы и преодолевать страх перед возможной неудачей. Вы можете забыть упомянуть что-то, что рассчитывает услышать эксперт. На тему о том, является ли такой формат собеседования ограниченным, можно спорить до бесконечности. По нашему опыту, с повышением уровня должности инженеры проводят все больше времени на совещаниях, и для них важнейшими навыками в том числе являются сообразительность, умение задавать правильные вопросы, направлять обсуждение на самые важные и актуальные темы и лаконично формулировать свои мысли. В этой книге подчеркивается, что соискатель должен эффективно и кратко обрисовать свой опыт проектирования систем на собеседовании, занимающем менее часа, и вести беседу в нужном направлении, задавая эксперту правильные вопросы. Чтение этой книги, наряду с отработкой проектирования систем с другими инженерами, позволит вам развить знания и красноречие, необходимые для прохождения собеседований, и эффективно участвовать в проектировании систем в компании, к которой вы присоединяетесь. Книга также пригодится экспертам, проводящим собеседования по проектированию систем.

Соискатель, у которого навыки письменного общения развиты сильнее навыков устного, может забыть упомянуть важные моменты в ходе примерно 50-минутного собеседования. На таких интервью преимущество имеют инженеры с хорошо

развитыми навыками устного общения, а у тех, у кого эти навыки слабы, шансы получить предложение о работе ниже, даже если они обладают значительным опытом проектирования систем и вносили ценный вклад в проектирование в организациях, в которых они работали. Эта книга подготовит инженеров к подобным (и не только) трудностям на собеседованиях по проектированию систем; покажет, как организованно решать их, и научит их не бояться.

Если вы — разработчик, желающий расширить свои знания концепций проектирования систем, усовершенствовать свои навыки их обсуждения или просто ознакомиться с их подборкой и примерами обсуждений, продолжайте читать.

1.3. ОБЗОР КНИГИ

Книга разделена на две части. Часть 1 напоминает учебник. В ее главах рассматриваются темы, поднимаемые на собеседованиях по проектированию систем. Часть 2 состоит из обсуждений типичных вопросов, относящихся к концепциям из части 1, которые задают на собеседованиях; в ней также рассматриваются антипаттерны, часто встречающиеся заблуждения и ошибки. В этих обсуждениях мы также констатируем очевидное: никто не ждет от соискателя полного знания во всех областях. Скорее он должен уметь обосновать, что некоторые подходы обеспечивают лучшее соответствие требованиям при допущении определенных компромиссов. Например, вам не придется вычислять степень сокращения размера файлов, затраты процессорного времени и памяти, необходимых для сжатия файлов в формате Gzip, но вы должны уметь объяснить, что сжатие файла перед его отправкой приведет к снижению сетевого трафика, хотя и потребует дополнительных затрат процессорного времени и памяти на стороне отправителя и получателя.

Книга старается собрать воедино разные актуальные материалы по теме. Это позволит вам сформировать базу своих знаний или же выявить пробелы в ней, чтобы позже заняться изучением других материалов.

Оставшаяся часть этой главы — своего рода пролог к примеру проектирования системы, в котором упомянуты некоторые концепции из части 1. Мы будем рассматривать эти концепции в посвященных им главах на основе данного контекста.

1.4. ПРОЛОГ: КОРОТКО О МАСШТАБИРОВАНИИ СЕРВИСОВ СИСТЕМЫ

Начнем с краткого описания типичной исходной конфигурации приложения и общего подхода к масштабированию сервисов приложения по мере необходимости. Попутно перечислим многие термины и концепции, а также виды

сервисов, необходимые для технологической компании, которые более подробно рассмотрим в оставшейся части книги.

ОПРЕДЕЛЕНИЕ Масштабируемостью (scalability) сервиса называется возможность простого и экономичного изменения ресурсов, выделенных сервису, для адаптации к изменениям нагрузки. Это понятие относится как к увеличению, так и к снижению количества пользователей и/или запросов к системе. Тема более подробно рассматривается в главе 3.

1.4.1. Начало: малое исходное развертывание приложения

На волне популярности домашней выпечки создадим привлекательное клиентское приложение Beigel[1], в котором пользователи могут читать и писать свои посты о ближайших кафе-пекарнях.

Изначально приложение Beigel включает следующие компоненты:

- Пользовательские приложения. По сути, это три версии одного приложения для трех основных платформ:
  - Браузерное приложение. Это приложение на базе ReactJS, которое отправляет запросы к сервису времени выполнения, написанному на JavaScript. Чтобы сократить объем кода JavaScript, загружаемого пользователями, мы сжимаем его при помощи Brotli. Gzip — более традиционный и популярный вариант, но Brotli создает сжатые файлы меньшего размера.
  - Приложение для iOS, загружаемое на iOS-устройство пользователя.
  - Приложение для Android, также загружаемое на Android-устройство пользователя.
- Бэкенд-сервис без сохранения состояния, обслуживающий пользовательские приложения. Может быть написан на Go или на Java.
- База данных SQL, размещенная на одном облачном хосте.

В приложении два основных сервиса: фронтенд и бэкенд. Эти компоненты показаны на рис. 1.1. Как видно из диаграммы, пользовательские приложения представляют собой компоненты на стороне клиента, а сервисы и базы данных — компоненты на стороне сервера.

ПРИМЕЧАНИЕ В разделах 6.5.1 и 6.5.2 объясняется, для чего нужен фронтенд-сервис между браузером и бэкендом.

---

[1] От *англ.* bagel — бейгл, рогалик (вид выпечки). — *Примеч. ред.*

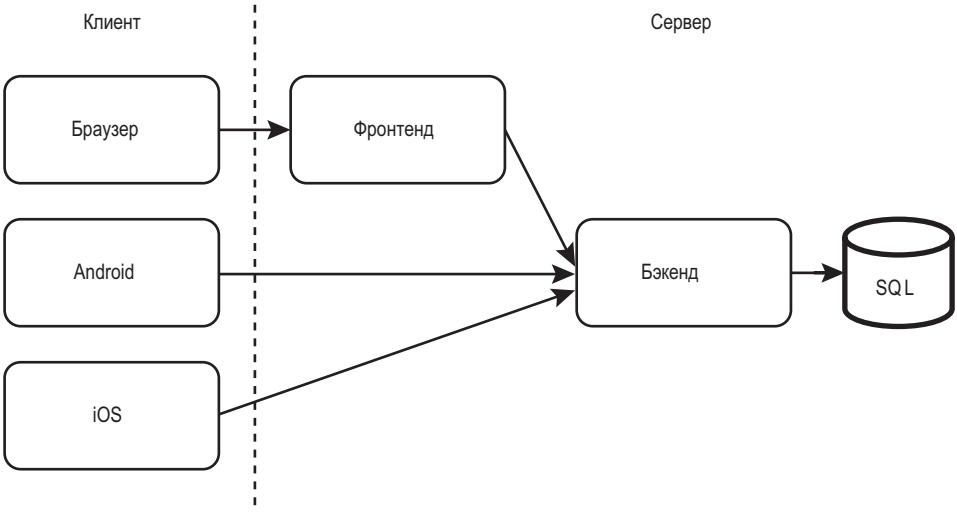

Рис. 1.1. Исходный дизайн системы приложения. Более подробное обоснование создания трех клиентских и двух серверных приложений (без учета приложения SQL/ базы данных) содержится в главе 6

При первом запуске сервиса количество пользователей будет небольшим и, как следствие, частота поступления запросов низкой. Для такой частоты может быть достаточно одного хоста. Мы настроим DNS для перенаправления всех запросов этому хосту.

Изначально два сервиса размещаются в одном датацентре, каждый на одном облачном хосте. (Облачный хостинг сравнивается с размещением на физическом оборудовании в следующем разделе.) Мы настраиваем DNS для перенаправления всех запросов из браузерного приложения на хост Node.js и от хоста Node.js и двух мобильных приложений на хост внутреннего сервиса.

1.4.2. Масштабирование с GeoDNS

Через несколько месяцев количество пользователей из Азии, Европы и Северной Америки, ежедневно проявляющих активность в Beigel, составляет сотни тысяч. В периоды пикового трафика фоновый сервис получает тысячи запросов в секунду, и мониторинговая система начинает отвечать кодом статуса 504 из-за тайм-аута. Система нуждается в масштабировании.

Мы своевременно заметили рост трафика и подготовились к нему. Наш сервис работает без сохранения состояния, как рекомендуют принятые лучшие практики, поэтому мы можем развернуть несколько одинаковых хостов бэкенда и разместить каждый хост в отдельном датацентре в своей части света. Обращаясь

к рис. 1.2, когда клиент выдает запрос к бэкенду через домен *beigel.com*, мы используем GeoDNS для направления клиента в ближайший к нему датацентр.

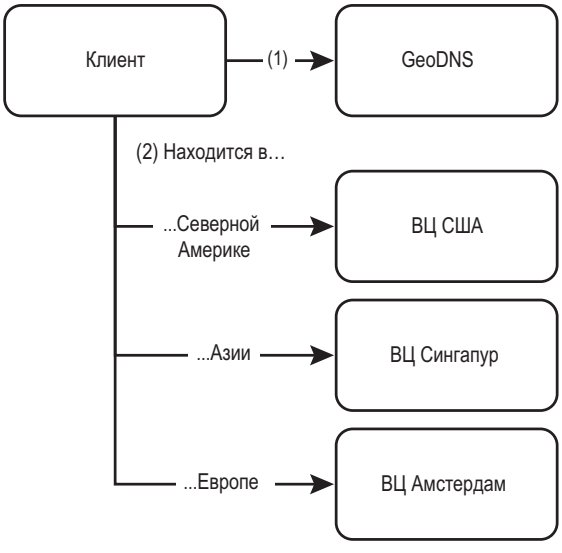

Рис. 1.2. Сервис может предоставляться в нескольких географически распределенных датацентрах. В зависимости от местонахождения клиента (определяемого по его IP-адресу) он получает IP-адрес хоста ближайшего центра данных, в который отправляет свои запросы. Клиент может кэшировать IP-адрес этого хоста

Если сервис обслуживает пользователей из определенной страны или географического региона в целом, он обычно размещается в близлежащем датацентре для минимизации задержки. Если же сервис обслуживает большую географически распределенную пользовательскую базу, его можно развернуть в нескольких датацентрах и использовать GeoDNS для возвращения пользователю IP-адреса сервиса, размещенного в ближайшем датацентре. Задача решается назначением нашему домену нескольких адресных записей для разных мест и IP-адреса по умолчанию для других мест. (*Адресная запись* — элемент конфигурации DNS, связывающий домен с IP-адресом.)

Когда клиент отправляет запрос серверу, GeoDNS получает информацию о местонахождении клиента по IP-адресу и назначает клиенту соответствующий IP-адрес хоста. В маловероятном, но возможном случае недоступности датацентра GeoDNS может вернуть IP-адрес сервиса в другом центре. Этот IP-адрес может кэшироваться на разных уровнях, включая провайдера (ISP) пользователя, уровни ОС и браузера.

1.4.3. Добавление сервиса кэширования

Согласно схеме на рис. 1.3, мы развертываем сервис кэширования Redis для обслуживания кэшированных запросов от пользовательских приложений. Мы выбрали несколько внутренних конечных точек с интенсивным трафиком, которые должны обслуживаться из кэша. Это дало немного времени на дальнейшую проработку системы, так как пользовательская база и нагрузка в виде запросов продолжили расти. Потребовались дополнительные усилия по масштабированию.

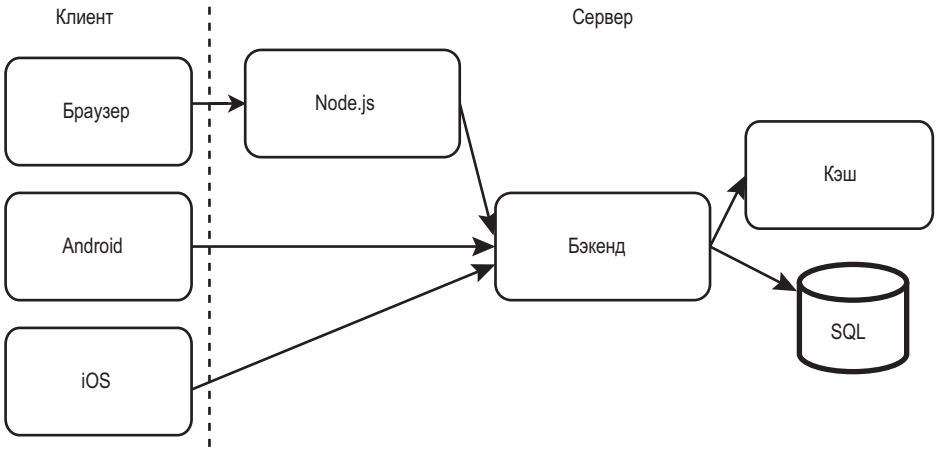

Рис. 1.3. Добавление кэширования к сервису. Некоторые внутренние конечные точки с интенсивным трафиком могут кэшироваться. Бэкенд запрашивает данные из БД при кэш-промахе или для баз данных/таблиц SQL, которые не были кэшированы

1.4.4. Сеть распространения контента

Наше приложение для браузера предоставляет статический контент/файлы, которые отображаются одинаково для всех пользователей и не зависят от пользовательского ввода: JavaScript, библиотеки CSS, графику и видео. Мы разместили эти файлы в репозитории исходного кода приложения, а пользователи загружали их из сервиса Node.js вместе с остальными частями приложения. Как показано на рис. 1.4, для размещения статичного контента было решено использовать стороннюю сеть распространения контента (CDN). Мы выбрали и обеспечили достаточный объем памяти в CDN для размещения файлов, отправили свои файлы в экземпляр CDN, переписали код для загрузки файлов по URL-адресам из CDN и удалили файлы из репозитория исходного кода.

На рис. 1.5 CDN хранит копии статических файлов в датацентрах по всему миру, чтобы пользователь загружал эти файлы из центра с наименьшей

задержкой — обычно самого близкого датацентра в географическом отношении, хотя другие датацентры могут оказаться быстрее, если ближайший центр находится под значительной нагрузкой или столкнулся с частичным сбоем.

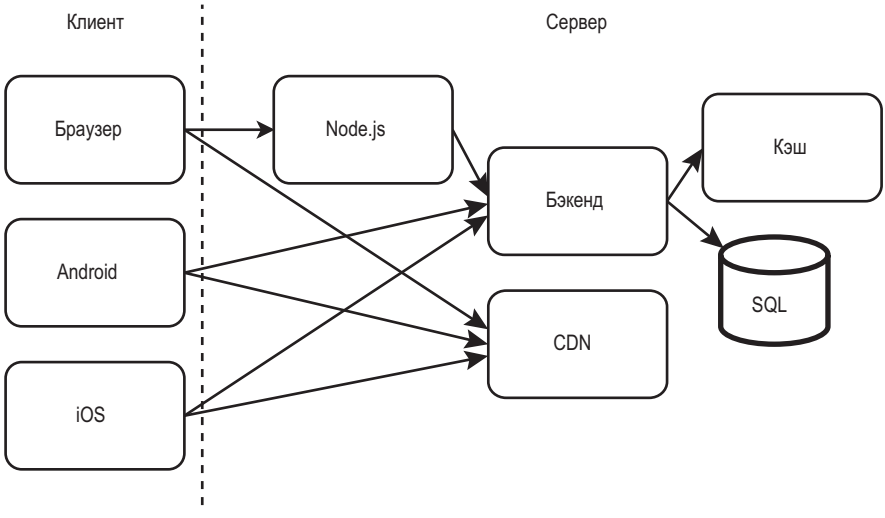

Рис. 1.4. Добавление CDN к сервису. Клиенты могут получать адреса CDN от бэкенда, или некоторые адреса CDN могут быть жестко закодированы в клиентах или сервисе Node.js

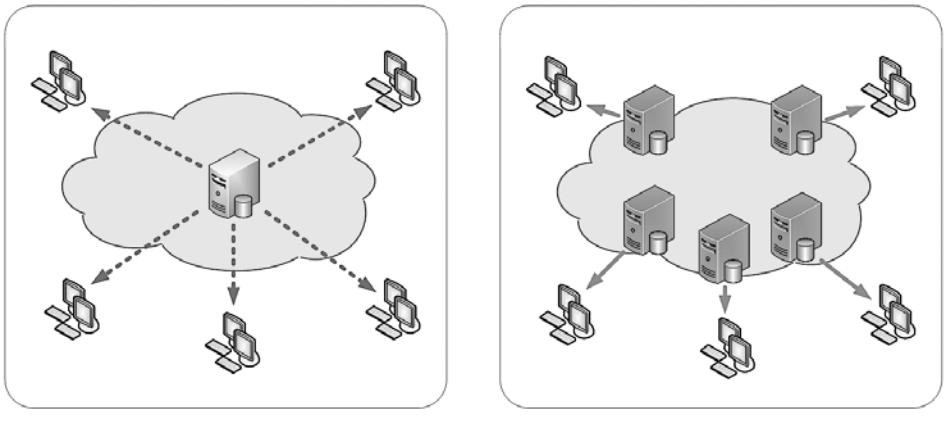

Рис. 1.5. На схеме слева все клиенты загружают данные с одного хоста. Справа клиенты загружают данные с разных хостов CDN. (Авторское право cc-by-sa https://creativecommons.org/licenses/by-sa/3.0/. Графика Kanoha, см. https://upload.wikimedia.org/wikipedia/commons/f/f9/NCDN_-_CDN.png)

Использование CDN снижает задержку и повышает пропускную способность, надежность и экономичность. (Все эти концепции рассматриваются в главе 3.)

При использовании CDN удельные затраты сокращаются, поскольку расходы на обслуживание, интеграцию и поддержку распределяются по большей нагрузке. Самые популярные CDN — CloudFlare, Rackspace и AWS CloudFront.

1.4.5. Коротко о горизонтальной масштабируемости и управлении кластерами, непрерывной интеграции и непрерывном развертывании

Наши сервисы — фронтенд и бэкенд — идемпотентны (некоторые преимущества идемпотентности описаны в разделах 4.6.1, 6.1.2 и 7.7); как следствие, они горизонтально масштабируемы, так что мы можем предоставить больше хостов для обработки увеличенной запросной нагрузки, не меняя исходный код, и развертывать фронтенд- и бэкенд-сервисы на этих хостах по мере надобности.

Над исходным кодом каждого из этих сервисов работают несколько инженеров. Инженеры ежедневно сохраняют новые коммиты. Мы изменяем существующие практики разработки и выпуска релизов на подходящие для поддержки большей команды и ускорения разработки, нанимая двух инженеров DevOps для разработки инфраструктуры управления большим кластером. Так как требования к масштабированию сервиса могут быстро измениться, необходима возможность легко изменять размеры кластера, развертывать сервисы и необходимые конфигурации на новых хостах и развертывать изменения кода на всех хостах кластера нашего сервиса. Мы можем экспериментировать с обширной пользовательской базой, развертывая разные версии кода или конфигурации на разных хостах. В этом разделе кратко рассматривается управление кластером для обеспечения горизонтального масштабирования и проведения экспериментов.

Непрерывная интеграция/непрерывное развертывание и инфраструктура как код

Чтобы выпускать релизы новой функциональности быстро и с минимальным риском ошибок, мы осуществляем непрерывную интеграцию и непрерывное развертывание (CI/CD) в Jenkins, а также средствами модульного и интеграционного тестирования. (Подробное обсуждение непрерывной интеграции/непрерывного развертывания выходит за рамки книги.) Docker используется для контейнеризации сервисов, Kubernetes (или Docker Swarm) — для управления кластером, включая масштабирование и балансировку нагрузки, а Ansible или Terraform — для управления конфигурацией разных сервисов, работающих на разных кластерах.

ПРИМЕЧАНИЕ Технология Mesos в целом считается устаревшей. Kubernetes — однозначный фаворит в этой области. Пара актуальных статей по теме: https://thenewstack.io/apache-mesos-narrowly-avoids-a-move-to-the-attic-for-now/ и https://www.datacenterknowledge.com/business/after-kubernetes-victory-its-former-rivals-change-tack.

Terraform позволяет инженеру по инфраструктуре создать единую конфигурацию, совместимую с разными облачными провайдерами. Конфигурация определяется на языке предметной области (DSL, Domain-Specific Language) Terraform и взаимодействует с облачными API для предоставления инфраструктуры. На практике конфигурация Terraform может содержать код, зависящий от поставщика; нужно постараться свести объем такого кода к минимуму. Как следствие, инфраструктура в меньшей степени привязана к конкретному поставщику.

Этот подход также известен под названием *«инфраструктура как код»*. Под этим термином понимается процесс управления и обеспечения работы датацентров с использованием машиночитаемых файлов определений вместо конфигурации физического оборудования или интерактивных средств настройки конфигурации (Wittig, Andreas; Wittig, Michael [2016]. Amazon Web Services in Action. Manning Publications. p. 93. ISBN 978-1-61729-288-0).

Постепенные развертывания и откаты

В этом разделе кратко рассматриваются постепенные развертывания и откаты, чтобы мы могли сравнить их с экспериментами в следующем разделе.

Развертывание сборки на продакшен можно проводить постепенно. Сборку можно развернуть на определенном проценте хостов, понаблюдать за их работой, а затем увеличить процент, постепенно доводя его до 100%. Например, можно последовательно развертывать сборку на 1, 5, 10, 25, 50, 75 и, наконец, 100% хостов. Развертывания можно вручную или автоматически откатить при обнаружении проблем:

- ошибок, пропущенных в ходе тестирования;
- сбоев;
- слишком высоких задержек или тайм-аутов;
- утечек памяти;
- повышенного потребления ресурсов (процессора, памяти, дискового пространства);
- увеличения оттока пользователей. Возможно, также стоит проанализировать постепенный отток пользователей — когда одни пользователи регистрируются и работают с приложением, а дргие перестают им пользоваться. Можно постепенно переводить все больший процент пользователей на новую сборку и изучать ее влияние на отток пользователей. Отток может возникать как из-за упомянутых факторов, так и из-за непредвиденных проблем, например из-за того, что изменения не понравились многим пользователям.

Например, в новой сборке задержка может превышать приемлемый уровень. Для решения этой проблемы можно воспользоваться комбинацией кэширования

и динамической маршрутизации. Сервис может заявлять задержку в 1 секунду. Когда клиент выдает запрос, направляемый к новой сборке, и происходит тайм-аут, клиент может читать результат из кэша или повторить запрос, который будет направлен хосту с более старой сборкой. Запросы и ответы следует сохранять в журнале, чтобы в дальнейшем провести диагностику тайм-аутов.

Можно настроить канал непрерывного развертывания, чтобы разбить рабочий кластер на несколько групп; инструментарий непрерывного развертывания будет определять подходящее количество хостов в каждой группе и закреплять хосты за разными группами. При увеличении размера кластера можно осуществлять переназначения и повторные развертывания.

Эксперименты

Когда вы вносите изменения в интерфейс при разработке новой функциональности (или ее удалении) или эстетические изменения в дизайн приложения, желательно развертывать их постепенно для группы пользователей, увеличивая размер этой группы, а не для всех пользователей сразу. Цель экспериментов — определение эффекта изменений в интерфейсе на поведение пользователя, в отличие от постепенного развертывания, направленного на изучение эффекта нового развертывания на эффективность приложения и отток пользователей. Стандартные методы проведения экспериментов — A/B-тестирование и многомерное тестирование (например, многорукий бандит). Эти темы выходят за рамки книги. За дополнительной информацией об A/B-тестировании обращайтесь по адресу https://www.optimizely.com/optimization-glossary/ab-testing/. Узнать подробнее о многомерном тестировании можно в книге «Experimentation for Engineers from A/B Testing to Bayesian Optimization»[1] Дэвида Свита (David Sweet) (Manning Publications, 2023), а метод многорукого бандита рассматривается в статье https://www.optimizely.com/ optimization-glossary/multi-armed-bandit/.

Эксперименты также проводятся для предоставления персонализированного опыта взаимодействия. Другое отличие экспериментов от постепенного развертывания/отката заключается в том, что в ходе экспериментов процент хостов, на которых выполняются разные сборки, часто оптимизируется при помощи средств включения/отключения функциональности, предназначенных для этой цели, тогда как при постепенном развертывании/откате механизм непрерывного развертывания используется для ручного или автоматического отката хостов к предыдущим сборкам при обнаружении проблем.

Непрерывное развертывание и эксперименты делают возможными короткие циклы обратной связи для новых развертываний и внедряемой функциональности.

---

[1] Свит Д. «Тюнинг систем: экспериментирование для инженеров от A/B-тестирования до байесовской оптимизации». СПб., издательство «Питер».

В веб-приложениях и приложениях бэкенда каждая версия функционала интерфейса (UX, user experience) обычно упаковывается в отдельную сборку. На части хостов будет работать другая сборка. С мобильными приложениями дело обычно обстоит иначе. Разные модификации UX часто кодируются в одной сборке, но каждый отдельный пользователь получает доступ только к определенному набору вариантов. Это делается по нескольким причинам:

- Развертывание мобильных приложений должно осуществляться через магазин приложений. Чтобы новая версия появилась на пользовательских устройствах, может понадобиться много часов. Быстрый откат развертывания невозможен.
- По сравнению с Wi-Fi мобильные данные медленнее, менее надежны и более дороги. Низкая скорость и ненадежность означают, что значительная часть контента должна предоставляться офлайн, то есть уже присутствовать в приложении. Тарифы на мобильную передачу данных во многих странах все еще высоки, при этом на нее могут устанавливаться лимиты и дополнительная плата за их превышение. Мы должны постараться оградить пользователей от этих затрат, в противном случае они будут реже использовать приложение, а то и вовсе удалят его. Чтобы проводить эксперименты и минимизировать использование данных загружаемых компонентов и медиа, мы просто включаем все эти компоненты и медиа в приложение и предоставляем доступ к нужному набору каждому конкретному пользователю.
- Мобильное приложение также может включать функциональность, которую часть пользователей никогда не будут применять. Например, в разделе 15.1 рассматриваются разные методы оплаты в приложениях. В мире существуют тысячи платежных решений. Приложение должно включать весь код и все SDK для всех решений, чтобы предоставить каждому пользователю небольшой набор платежных решений, доступных ему.

Как следствие, размер мобильного приложения вполне может превышать 100 Мбайт. Способы решения этой проблемы выходят за рамки книги; необходимо выдержать баланс, рассмотрев все компромиссы. Например, установка мобильного приложения YouTube очевидно не может включать множество видеороликов на YouTube.

1.4.6. Функциональная декомпозиция и централизация сквозных обязанностей

Целью функциональной декомпозиции является распределение разных функций по разным сервисам или хостам. Во многих сервисах присутствуют общие обязанности, которые могут выделяться в общие сервисы. В главе 6 обсуждаются причины, преимущества и компромиссы такого подхода.