Cloud Thinking

Модуль 6, урок 1

Сегодня мы поговорим о простоте предоставления облачных ресурсов для работы внутри большой компании.

В 2011 году в качестве ключевых критериев облака выделяли:

Облачные свойства

NIST, сентябрь 2011

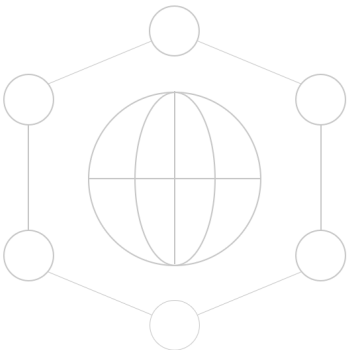
01 Самообслуживание по требованию (On-demand self-service)

02 Широкий сетевой доступ (Broad network access)

03 Объединение ресурсов в пулы (Resource pooling)

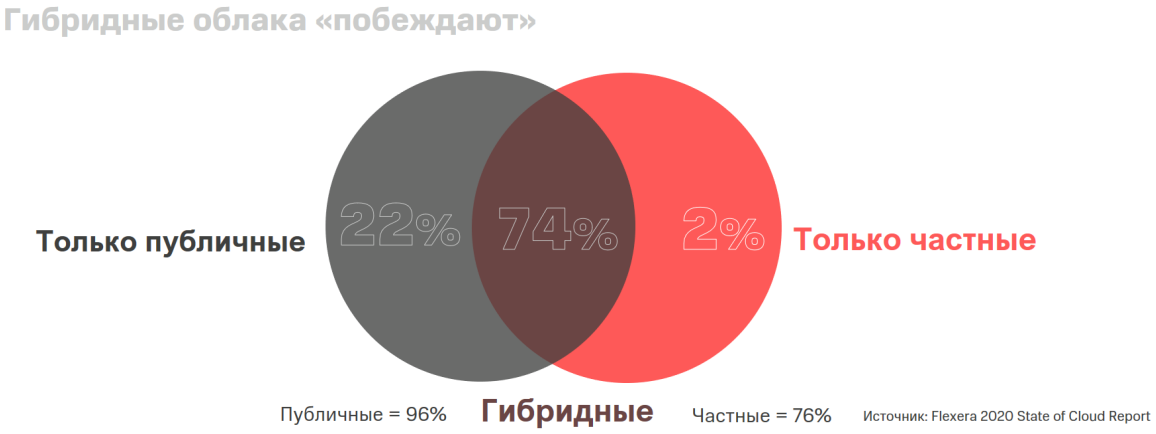
04 Мгновенная эластичность (Rapid elasticity)

05 Измеряемость (Measured service)



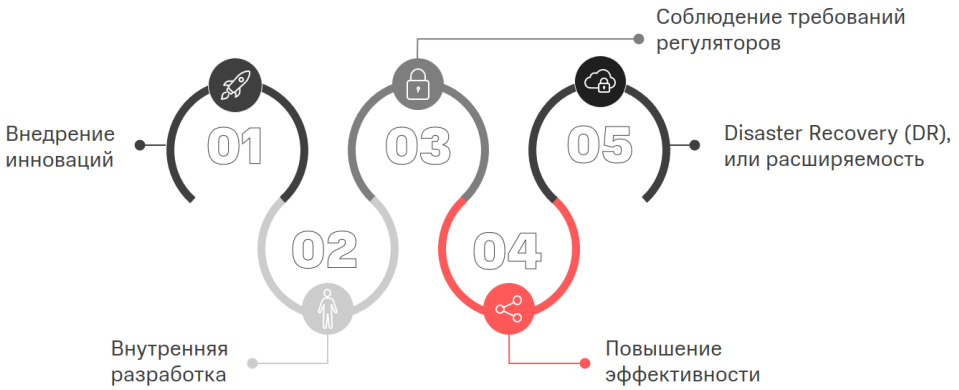
А как обстоят дела сегодня?

Согласно обзору компании Flexera от 2020 года, сейчас компании оперируют не только собственными ресурсами, но и берут их извне, то есть используют гибридные облака.



Неважно, это компании только с собственной инфраструктурой или компании, которые сразу родились в облаке, — все они используют разные облачные ресурсы: например, облака гиперскейлеров или локальных [сервис-провайдеров типа #CloudMTS](#).

Почему все стремятся в облака?



1. **Внедрение инноваций**

В последние несколько лет растет давление на бизнес, повышается конкуренция, особенно в сфере электронной коммерции, и все больше и больше сервисов необходимо предоставлять клиентам.

2. **Внутренняя разработка**

Компании даже из классического Enterprise, которые раньше использовали внешние сервисы, нанимают большой штат разработчиков. Причем речь идет буквально о тысячах разработчиков внутри, которым нужно предоставлять ресурсы, в том числе инфраструктурные. Как правило, эти ресурсы берутся из облаков.

3. **Соблюдение требований регуляторов**

Не стоит забывать и о требованиях регуляторов. Это могут быть как страновые требования, так и корпоративные стандарты (безопасности, ITIL и другие).

4. **Организация Disaster Recovery**

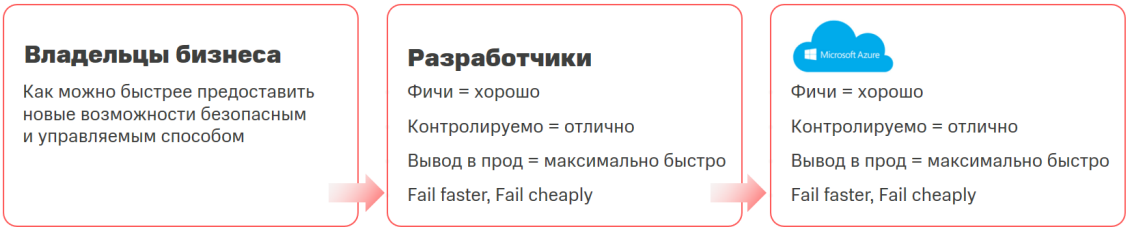
DR (восстановление после аварии) — довольно популярный сценарий использования облаков. Когда компании нужно обеспечить высокий уровень отказоустойчивости инфраструктуры, вместо самостоятельной реализации и удвоения объема ресурсов (а вместе с этим и CAPEX) она может обратиться к внешнему сервис-провайдеру за такой услугой.

5. **Повышение эффективности**

От облака ждут роста эффективности — высокой утилизации внутренних и внешних ресурсов (чтобы не платить за «воздух», то есть неиспользуемые ресурсы) и максимальной отдачи от их использования.

Бизнесу необходимо постоянное увеличение скорости

5 причин выбора мультиоблачной стратегии



Внедрение инноваций — один из ключевых драйверов популярности облаков со стороны бизнеса. Бизнесу нужны инновации → он ставит задачи разработчикам → разработчики обращают свое внимание на публичные облака, где можно быстро получить необходимые в данный момент ресурсы.

Однако разработчики не учитывают, как эти ресурсы в дальнейшем могут быть использованы, как переводить среды Test или Dev в продуктив, потому что эта задача находится в зоне ответственности отдела эксплуатации. Ведь разработчики в первую очередь сфокусированы на реализации фич в приложениях и не должны задумываться об инфраструктуре.

Разработчики реализуют бизнес-потребности. Чего они хотят?

- Мгновенного удовлетворения потребностей — получения ресурсов сразу по запросу.
- Повторяемых процессов: если что-то заказали один раз, должна быть возможность получить этот сервис в любой другой момент в необходимой конфигурации. Без автоматизации добиться этого невозможно.
- Не думать о безопасности, соответствии корпоративным стандартам и шаблонам — этой задачей должна заниматься группа эксплуатации или DevOps-команда, которая помогает разработчикам автоматизировать процесс доставки приложений.

О чем они не хотят думать?

Мгновенной удовлетворенности

Повторяемых процессов

Гибкости в определении, что им нужно

В ИТ, как правило, достаточно высокая доля адептов ITIL, хорошо известны процессы Change Management, есть процессы постановки заявок и многое другое. И это не очень хорошо ложится на текущие реалии с высокой степенью изменяемости, которые присутствуют в современной разработке.

Вспомним еще раз об облачных свойствах, а именно об измеряемости с точки зрения биллинга. Чтобы принимать решения, в какую сторону развиваться, какие облака использовать, требуется ли миграция, необходимо понимать, во сколько обходится эксплуатация ресурсов в разных облаках.

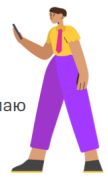
Мы все прошли через это

«Я слышал, ты можешь помочь развернуть новое окружение для тестирования моего нового сервиса»

«Конечно, приятель! У нас есть классный портал самообслуживания для этих целей!»

«Хм, но ничего из этого не подходит для моих задач...»

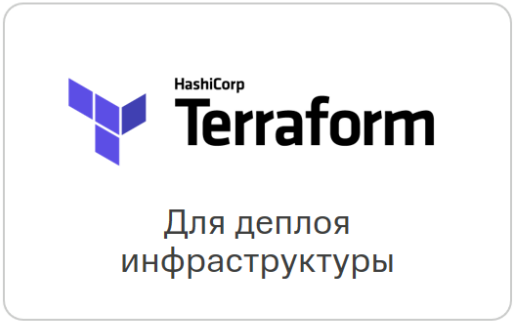
«Не дрейфь! Поставь заявку в ServiceDesk — и я все сделаю в течение 4–6 недель!»

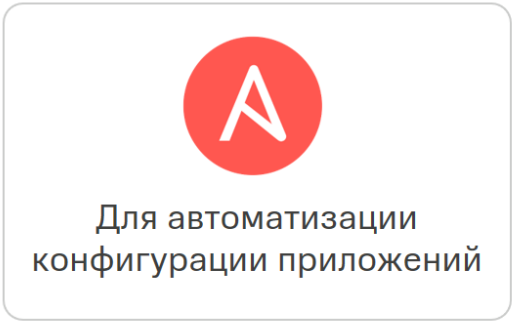


«Да ну, забудь...»

Облака (моно или гибридные) уже были реализованы в той или иной степени в разных организациях. В современных компаниях есть процесс заказа ресурсов (это может быть как принтер, так и доступ к приложению или развертывание нового сервера). Как правило, этот процесс инициируется через заявку в ServiceDesk и довольно жестко регламентируется: существуют процессы создания сервисов, их заказа, жизненного цикла и так далее.

Если какой-то группе разработки нужен новый сервис, его создание, изменение и конфигурация могут занимать продолжительное время. Все это заставляет разработчика смотреть на экосистемы публичных облаков, где он может быстро получить доступ к ресурсам, минуя корпоративные стандарты и ограничения. При этом там можно получить в свое распоряжение множество инструментов и использовать их для автоматизации выделения ресурсов. Одними из самых популярных инструментов являются Terraform и Ansible.





- Terraform служит для создания и заказа инфраструктуры.
- Ansible обычно используют в качестве инструмента автоматизации конфигурации приложений — для донастройки заказанного сервиса (установки БД, приложений) до состояния работающей системы.

Оба инструмента обладают следующими характеристиками:

01 Мультиоблачные возможности

02 Простая, легковесная архитектура

03 Просто писать шаблоны на основе YAML

04 Могут выполнять команды на удаленных ВМ

05 Можно использовать в CI-/CD-конвейерах

Зачем искать что-то еще?

- Оба инструмента обладают схожими возможностями: они определяют необходимую конфигурацию через код (как правило, YAML-файл), реализуют популярную схему IaC.
- У Terraform есть большое количество плагинов для работы с разными облаками.
- И Ansible, и Terraform могут работать без какого-либо сервера — можно сразу запустить на лэптопе.
- Их довольно просто использовать — порог входа в эти инструменты достаточно низок.
- Можно использовать в CI-/CD-конвейерах по части Continuous Delivery — например, когда разворачивается новая версия приложения на созданных перед этим ресурсах.

Это простые и удобные, но все же лимитированные инструменты быстрого конфигурирования. Их недостаточно для работы в больших компаниях со своими регламентами, стандартами, требованиями к управлению ресурсами, безопасности и т. д.

Все равно требуется отделение ресурсов от пользователей, ведь в случае использования этих инструментов, реализующих концепцию IaC, мы имеем админский доступ к инфраструктуре. Эти вещи необходимо разделить. Посередине должен быть некий брокер, который будет проксировать заказы к облакам и распределять их между различными облаками. Иными словами, пользователь должен через API или каталог обращаться к этому брокеру и получать ресурсы в том или ином облаке, причем логика выбора (где дешевле, быстрее, какое облако соответствует корпоративным стандартам и т. д.) должна оставаться за эксплуатацией.

Чего не хватает для полноценного облака?

1.Настоящей «кросс-облачности».
2.Возможности делегирования задач пользователю: скрипты запускают теперь уже не «админы виртуализации», но DevOps-команда.
3.Контроля ресурсов: пользователь/скрипт может завалить запросами все имеющиеся ресурсы.
4.Отчетности: кто, когда и что разворачивал/изменял.
5.Гранулярности: все должны быть «админами».
6.Управления (Governance): согласования, оповещения.
7.Выделения ресурсов на время (аренда).
8.Планирования потребления.
9.Инструмента для «обычного пользователя» – портала самообслуживания.
10.Интеграции с корпоративными системами .
11.Биллинга.
12.И многого другого...

Что хотелось бы получить в дополнение к этому удобству и гибкости:

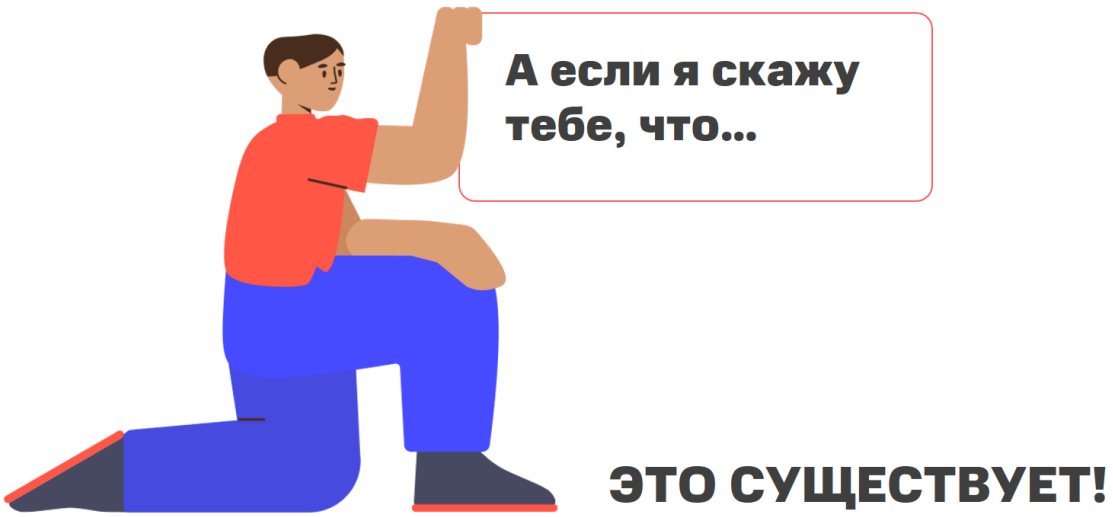
- Контроль ресурсов — для исключения риска «заспамить» инфраструктуру или получить большой счет от внешнего сервис-провайдера из-за бесконтрольного потребления ресурсов.
- Надзорность (governance) — соответствие требованиям корпоративных стандартов.
- Планирование роста объема ресурсов — с какой скоростью растет потребность в ресурсах со стороны команд разработки, как быстро необходимо наращивать внутреннее облако или как согласовывать этот рост с поставщиком, если речь идет о внешнем облаке.
- Биллинг — требуется максимальная эффективность использования внешних ресурсов.

Облако 2.0

- Интегрированное программное и аппаратное решение на основе гиперконвергентной архитектуры.
- Ориентировано на приложения, а не на ИТ.
- Проще разворачивать.
- Может конкурировать по стоимости с публичными облаками.
- Упрощена процедура перехода на новые версии.
- Предоставляют не только услуги IaaS, но и PaaS.
- Современные частные облака — это платформы с API.

В более современных требованиях к облаку (в дополнение к требованиям от 2011 года) появился фокус на приложения и API. Теперь большое значение имеют такие возможности, как:

- использование облаков для развертывания приложений;
- потребление готовых сервисов из облака (PaaS);
- заказ сервисов не только через пользовательские инструменты провайдера, но и через API — программная интеграция между различными компонентами облака, внутренней инфраструктурой и сервисами.



И это уже существует. Мы помним, что в компании из примера выше есть облако и ServiceDesk для заказа ресурсов, но он довольно жесткий, а все изменения в каталог проходят длительное согласование.

А нам бы хотелось получить тот самый брокер, который внутренняя команда эксплуатации могла бы использовать для организации доступа команд разработки к различным внутренним или внешним ресурсам.

Обратите внимание на схему. У нас есть ресурсы:

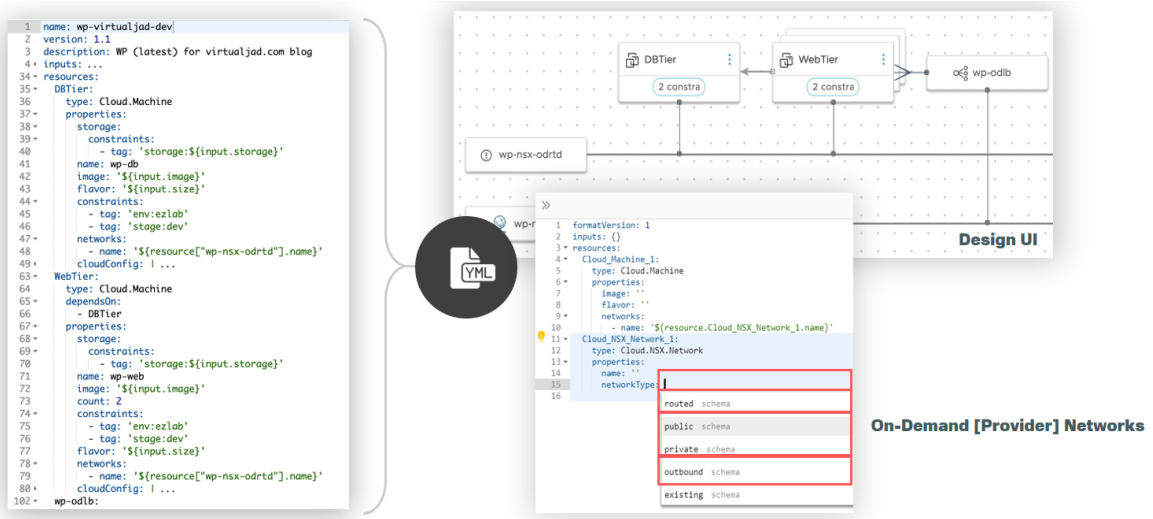
- центр обработки данных;
- периферия — небольшие серверные или другие небольшие участки инфраструктуры;
- облака локальных провайдеров, например #CloudMTS;
- ресурсы гиперскейлеров (AWS, Google Cloud, Alibaba Cloud, MS Azure).

Над всем этим находится уровень доступа, на базе которого можно обеспечить подключение к ресурсам через API или каталог. При этом вне зависимости от того, где именно размещаются ресурсы, желательно:

- использовать те самые удобные принципы для заказа через описание сервисов в виде кода;
- получать не только BM, но и контейнерные сервисы, кластеры, репозитории, PaaS-сервисы;
- иметь маркетплейс, откуда можно брать готовые сервисы и размещать в собственном каталоге.

Cloud Thinking

Инфраструктура как код



Ранее мы уже говорили о необходимости определения сервисов с помощью конфигурационных файлов. Допустим, есть классический сервис из уровня приложений, уровня базы данных и какой-то сети. Соответственно, нужно указать название сервиса, теги, которые будут использованы при логике развертывания в том или ином облаке — например, сервис можно отметить как «быстрый, «локальный» и т. д.

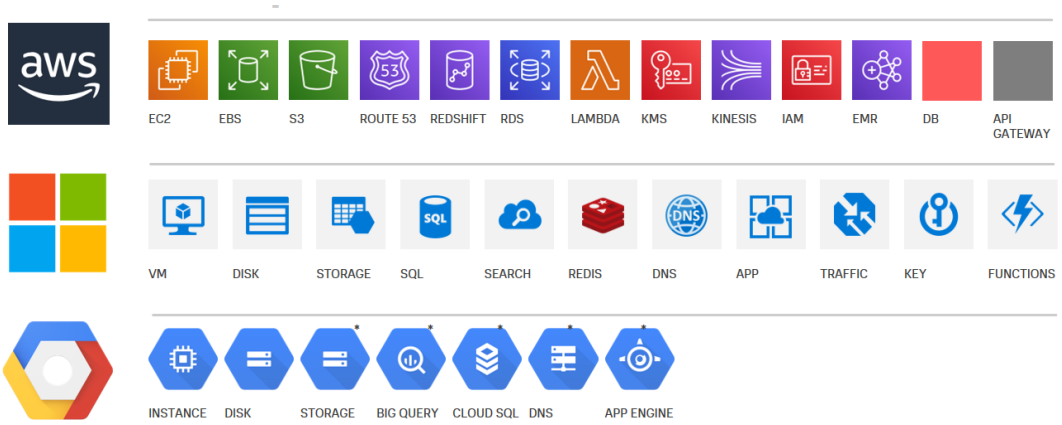
Описываем все, что нужно, в виде YAML-файла, который можно хранить во внешней системе версионирования (GitHub, GitLab) или прямо на лэптопе. Если возможности писать код нет (например, из-за незнания синтаксиса), можно сконфигурировать сервис визуально — это тоже удобно и позволяет быстрее научиться работать с системой.

Итак, мы описали, что нам нужно, и отправили брокеру. После того как брокер развернул ресурсы, возникает необходимость управления сервисом. Например, нужно внести какое-то изменение. Особенность такого более бережного отношения к сервису, когда это не просто какая-то игрушка или песочница разработки, но и целый процесс, который ведется от стадии разработки в продуктив, нам нужно вносить какие-то изменения и получать обратную связь.

Поэтому вопрос управления жизненным циклом сервиса очень важен, вплоть до своевременного удаления сервиса: представьте, если мы все время будем заказывать все новое и новое, рано или поздно контроль над ресурсами потеряется, а мы будем получать большие счета. Поэтому нужна возможность сворачивать ресурсы так же быстро и просто, как и разворачивать.

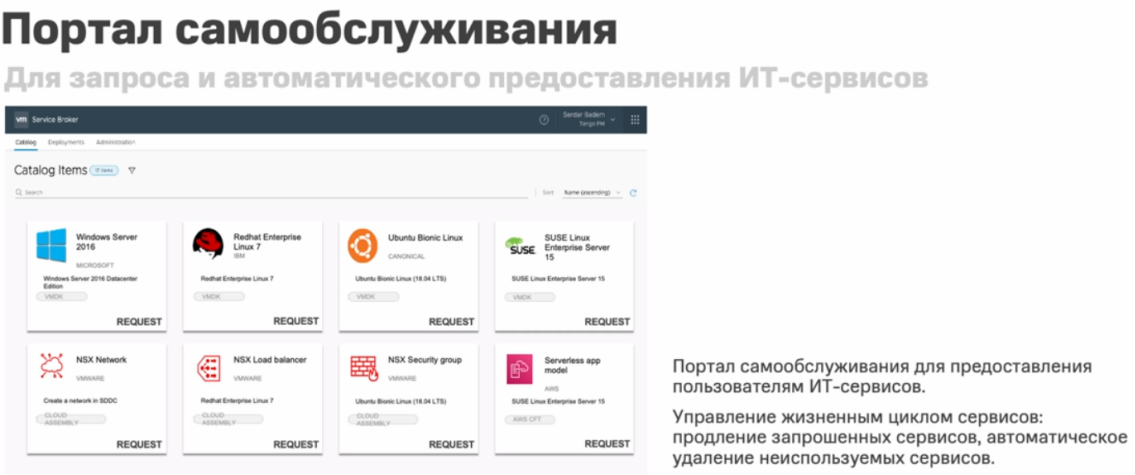
Удаление, оптимизация, повышение утилизации — это то, что реализуется через инфраструктурные конвейеры.

Ранее мы говорили о поддержке разных сервисов — локальных и во внешних облаках.
Во внешних облаках большая экосистема сервисов: не только виртуальные машины и контейнеры, но и готовые БД, лямбда-функции, сервисы авторизации, резервного копирования и так далее. Было бы здорово получать что-то готовое и переиспользовать это и во внутреннем облаке.



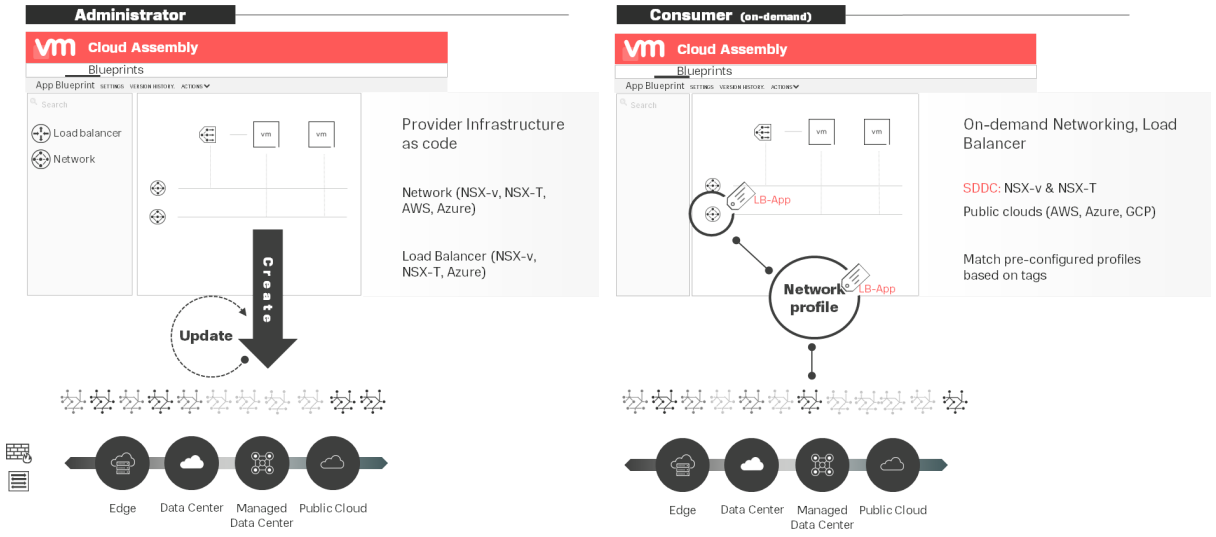
С помощью подобного брокера вы можете обратиться к готовому сервису, который для вас создал внешний провайдер, и опубликовать его во внутреннем каталоге без внесения значимых изменений или самостоятельного «изобретения» такого сервиса внутри.

Сейчас популярен заказ сервисов через API, то есть программное потребление, чтобы мы могли интегрировать это с существующими конвейерами разработки, или через консоль, если это удобно. Тем не менее классический заказ сервисов через каталог тоже имеет место.



Если разработчику нужна готовая ВМ, БД и он не хочет разбираться, как это внутри работает, — ему проще зайти в каталог, накликать нужную конфигурацию и получить желаемое.

Сервис-каталог — это то, что видит конечный пользователь.

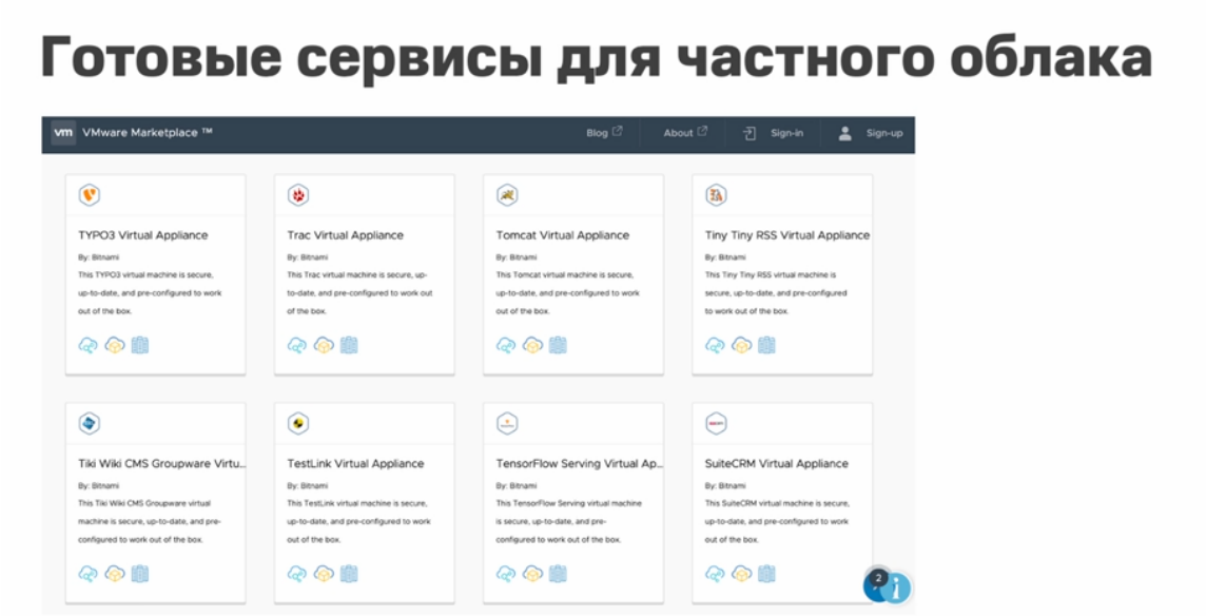


Вопросы сети, безопасности особенно актуальны для больших компаний, так как всем, что касается доступов, change management, правил файрвола, работы с физическим файрволом, занимается отдельная команда сетевиков-безопасников — и это довольно непросто встроить в непрерывный конвейер доставки сервисов и приложений.

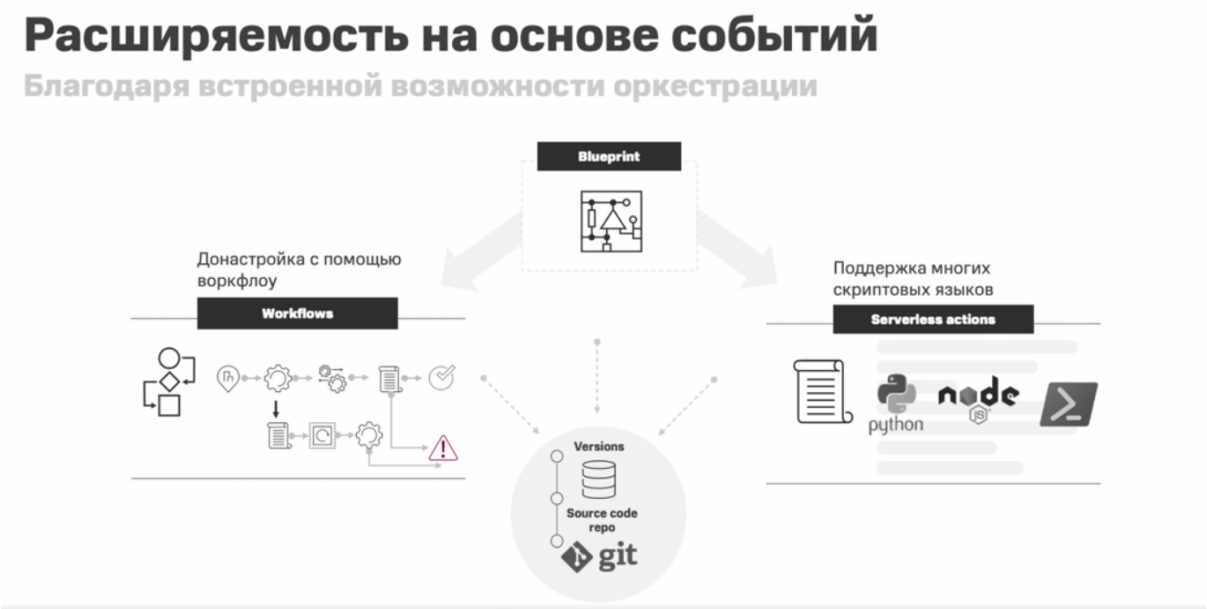
Как правило, на первом этапе все эти операции выполняются вручную. По мере трансформации, коммуникации с внешними командами удается привлечь их на свою сторону и встроить какие-то простые операции над сетью в общий конвейер. В идеале это программные сервисы (если используется SDN, которая позволяет исключить вмешательства в физическую сеть).

Подобный брокер должен уметь обращаться с сетевыми сервисами — балансировкой, файрволом, маршрутизацией и другими, чтобы наш сервис был максимально законченным и готовым к использованию разработчиком.

Также в нашем каталоге должны оказаться сервисы контейнеров — всё, что касается контейнеров Kubernetes (будь то ванильный K8s или на базе контейнеров от компаний Red Hat, VMware). Должна быть возможность получить их в виде готовых управляемых сервисов с минимальным погружением внутрь. Берем, используем, публикуем в каталоге, и вуаля — через процедуру заказа сервисов разработчик получает доступ к кластеру Kubernetes, скачивая файл конфигурации cubeconfig к себе на консоль и делая это без обращения напрямую к сервис-провайдеру (туда доступа у него нет, он может даже не знать, как работает K8s, зато знает, как работает контейнер).



Также было бы здорово иметь готовый маркетплейс, откуда можно брать сервисы для размещения во внутреннем каталоге. Например, такой маркетплейс есть у VMware: там можно скачать образы с готовыми приложениями внутри или helmchart'ы для запуска в среде K8s. Он позволяет максимально упростить запуск собственного каталога сервисов, не изобретая велосипед и используя готовые наработки.

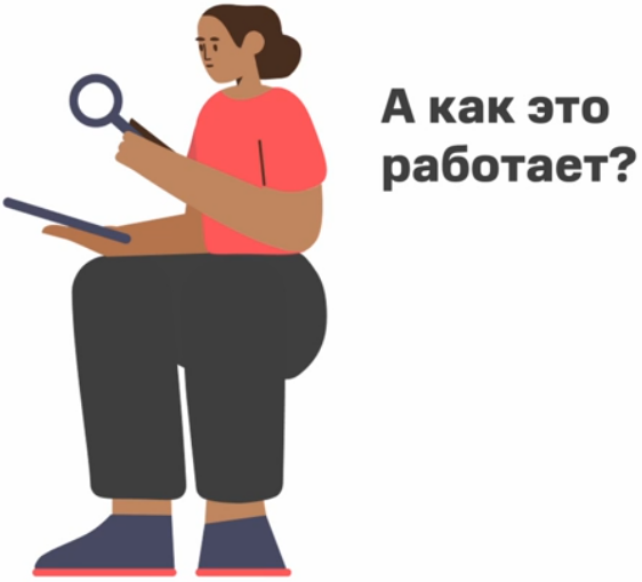


Если мы говорим о внутренней разработке, нам нужно не просто создать PaaS в вакууме: когда мы берем готовый элемент из маркетплейса, требуется максимально интегрировать его с существующими системами внутри компании, то есть довести его до готовности к использованию внутри конкретной компании. Ведь ни внешний сервис-провайдер, ни гиперскейлер не знают, что творится у вас внутри и какие корпоративные стандарты необходимо соблюсти.

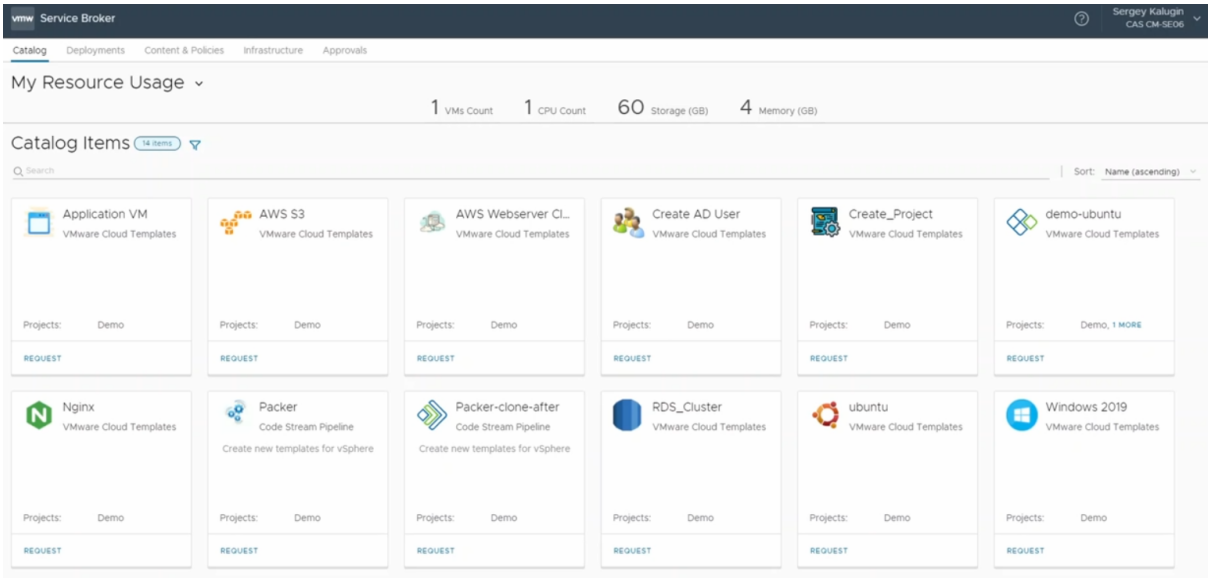
Все доработки — соблюдение требований к уровню доступности, интеграция с системами мониторинга или резервного копирования и пр. — необходимо выполнить самостоятельно. Когда вы берете готовый элемент из маркетплейса, он не сможет сразу заработать на вашей инфраструктуре. Потребуется его доработать и донастроить. Для этого важно, чтобы упомянутый выше брокер обладал возможностью расширения готового воркфлоу и мы могли бы кастомизировать его до требуемого состояния.

Например, когда запускается виртуальная машина, важно, чтобы она попала в ваш домен, чтобы там установился нужный агент мониторинга, чтобы в зависимости от уровня качества обслуживания она попала в подходящую политику резервного копирования.

Важно, чтобы все эти операции были выполнены не только в момент развертывания, но и при удалении. Ведь облако обладает некой цикличностью: там заказывают ресурсы и отдают их. Одним из показателей эффективности работы облака является процент оборачиваемости — сколько из ваших ВМ, какой процент кластеров создается и удаляется. Такая расширяемость, такая оркестрация — это тоже один из интересных функционалов, которые доступны в решении от компании VMware.



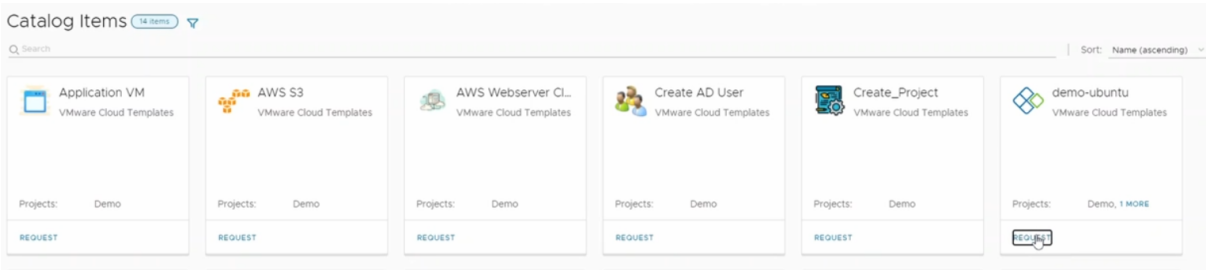
Посмотрим, как это работает на практике.
Перед нами каталог брокера, который является фронтендом для внутреннего пользователя — например, разработчика или менеджера проекта.



Каждая плашка — какой-то оркестрируемый сервис в одном из облаков, на которые мы подписаны. Или это может быть даже один сервис, в котором

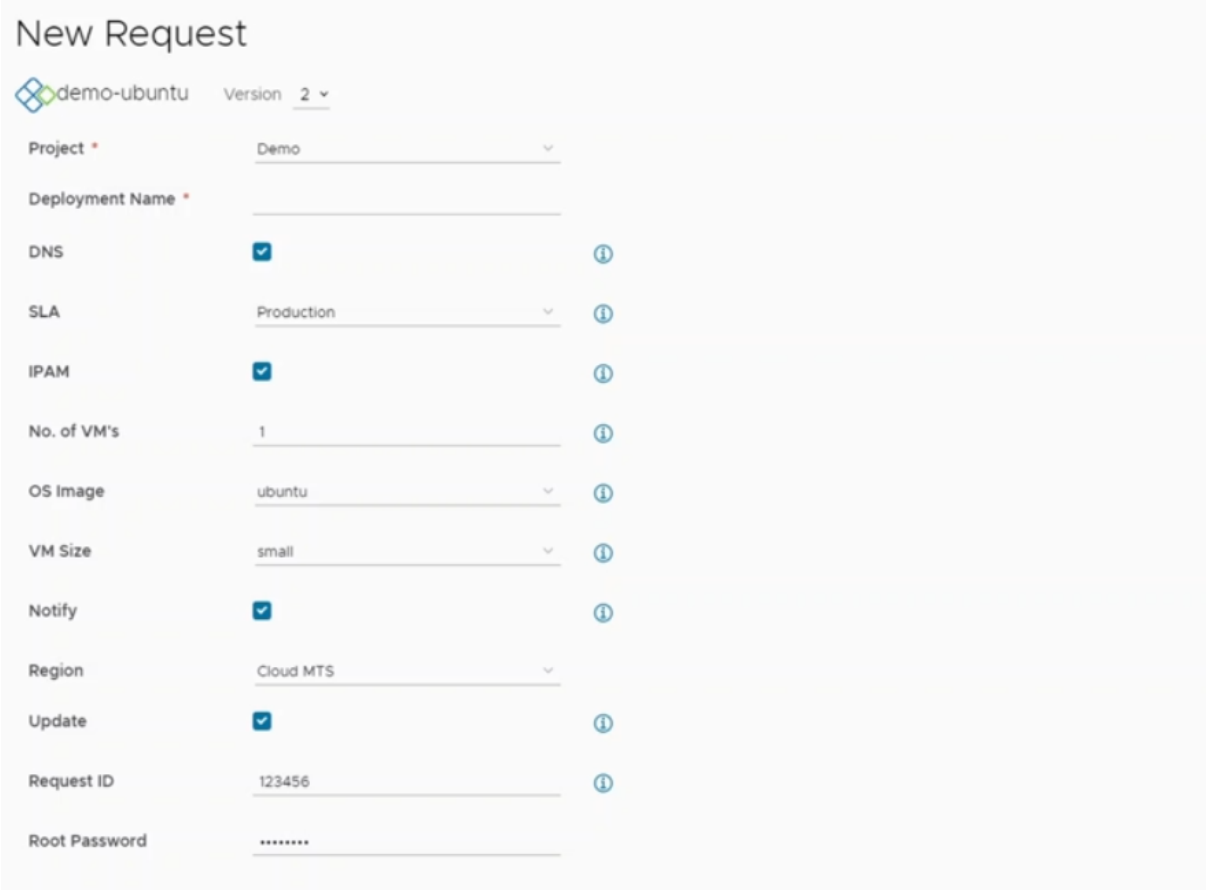
выбор конкретного облака является динамическим (в зависимости от требований сервис может быть приземлен на разные площадки).

Это могут быть ВМ, PaaS-сервисы (например, с базами данных), сервисы со специфичными разворачиваемыми приложениями или готовые сервисы из маркетплейса.



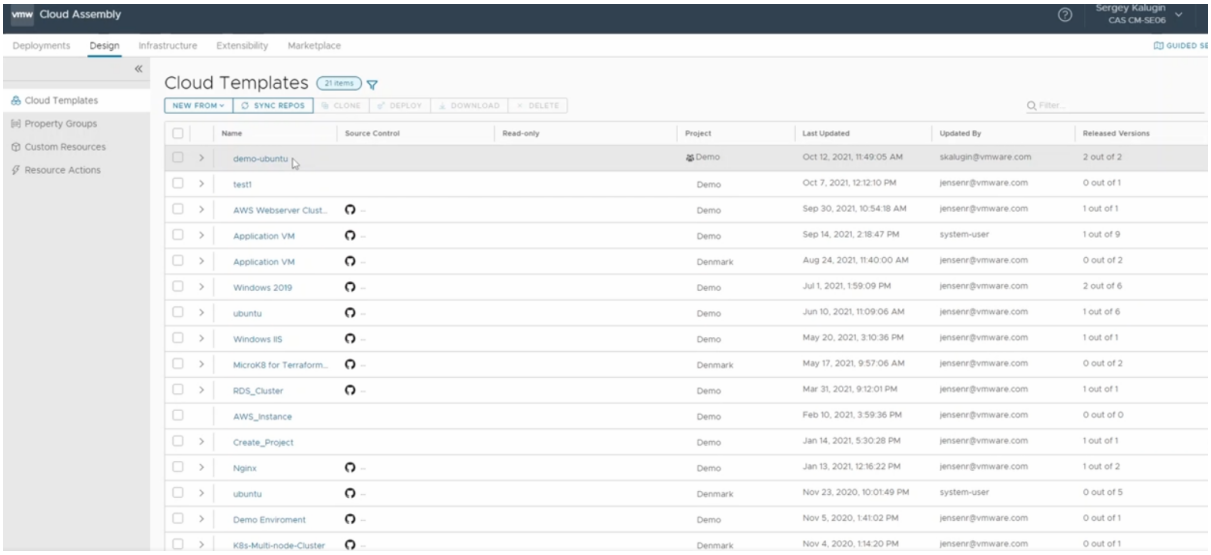
После авторизации нужно нажать кнопку Request и заполнить форму. Кроме названия можно указать:

- в каких системах сервис должен быть зарегистрирован;
- является ли он продуктивным или это сервис тестирования;
- количество элементов;
- регион или облачный провайдер, где будет размещена ВМ.

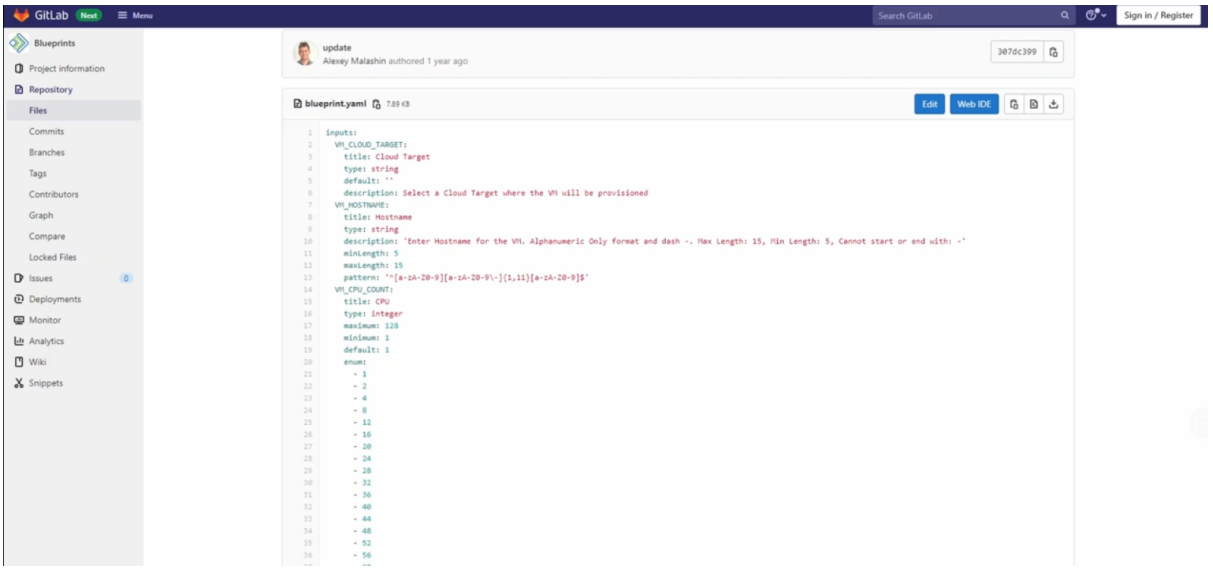


Посмотрим, как устроен наш брокер на бэкенде и что необходимо сделать, чтобы разместить сервис в облачном каталоге.

На вкладке Design представлены шаблоны (cloud templates) сервисов.

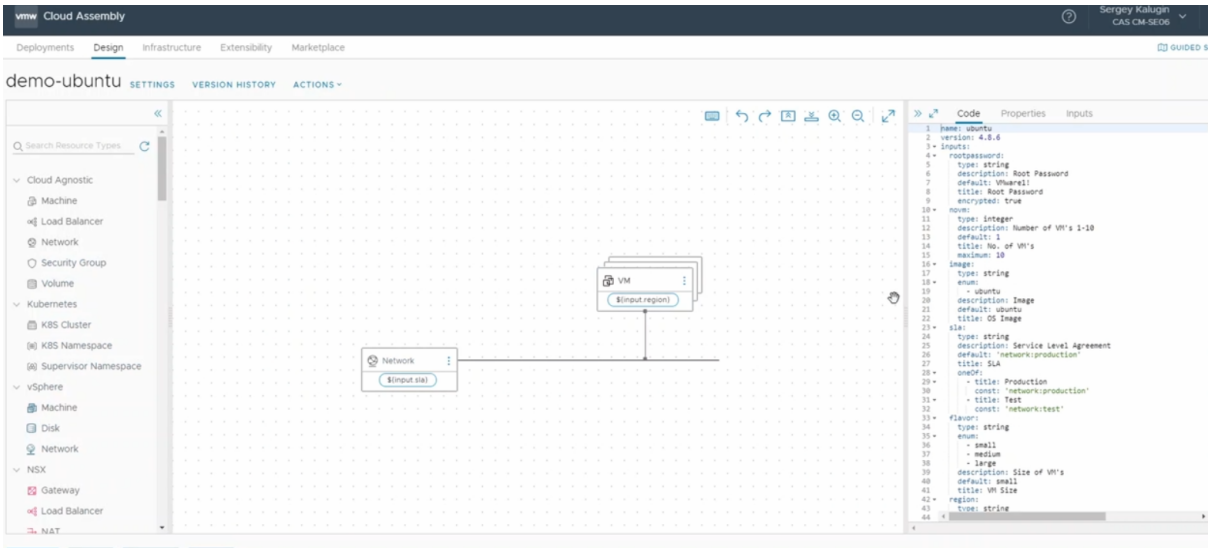


Обратите внимание: мы говорили, что придерживаемся принципов IaC, то есть эти шаблоны могут быть размещены в системе версионности (GitLab, GitHub) или на локальном десктопе — можно скопировать имеющуюся конфигурацию, занести ее в файл описания и получить готовый сервис, указав точку развертывания.

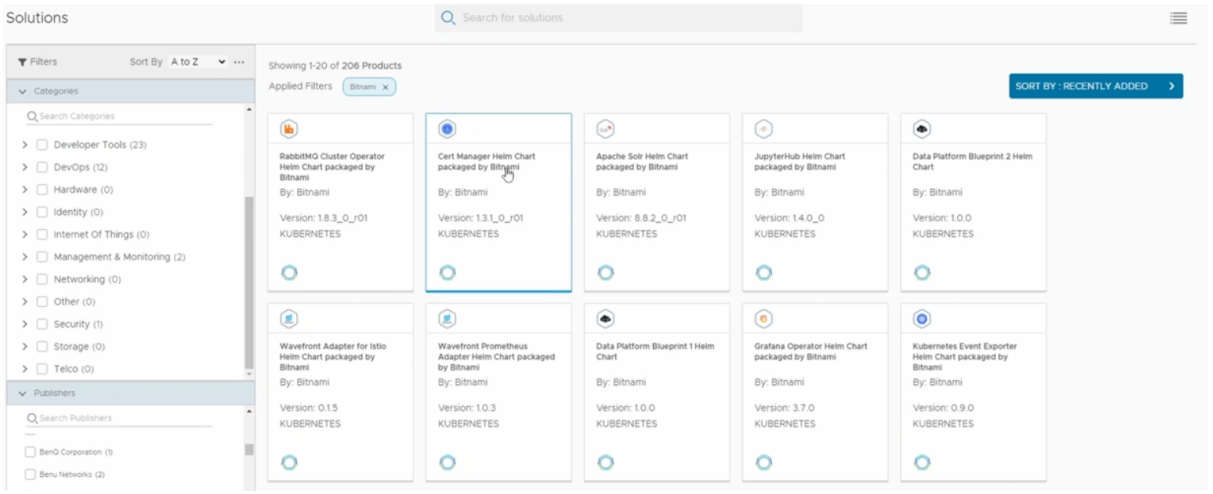


Если мы попытаемся его отредактировать, откроется похожий на Visio редактор с визуальными компонентами и компоненты в виде описания желаемой конфигурации.

Слева — элементы, из которых мы собираем сервис (это могут быть элементы, относящиеся к гипервизору, K8s, сетевые элементы вроде NAT, балансировщика или правил файрвола и т. д.). Выше мы уже говорили, что было бы здорово имплементировать сервисы, представленные в облачной инфраструктуре того или иного гиперскейлера — Amazon, Azure, Google Cloud. Они тоже здесь представлены.



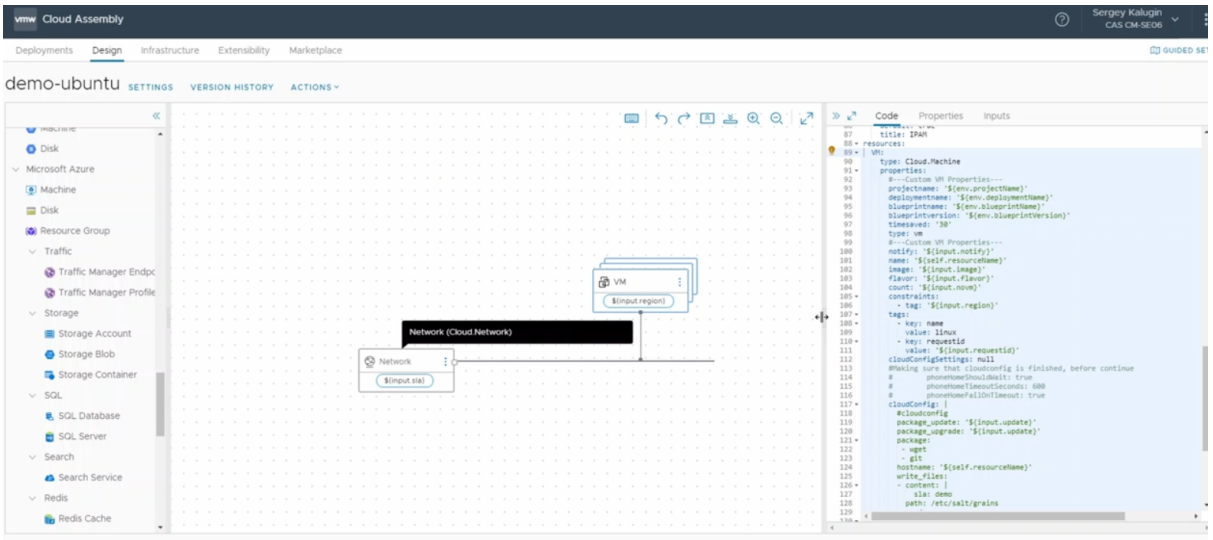
Также эти элементы есть в маркетплейсе, где представлены сотни готовых упакованных сервисов, — их можно скачать и разместить в нашем каталоге.



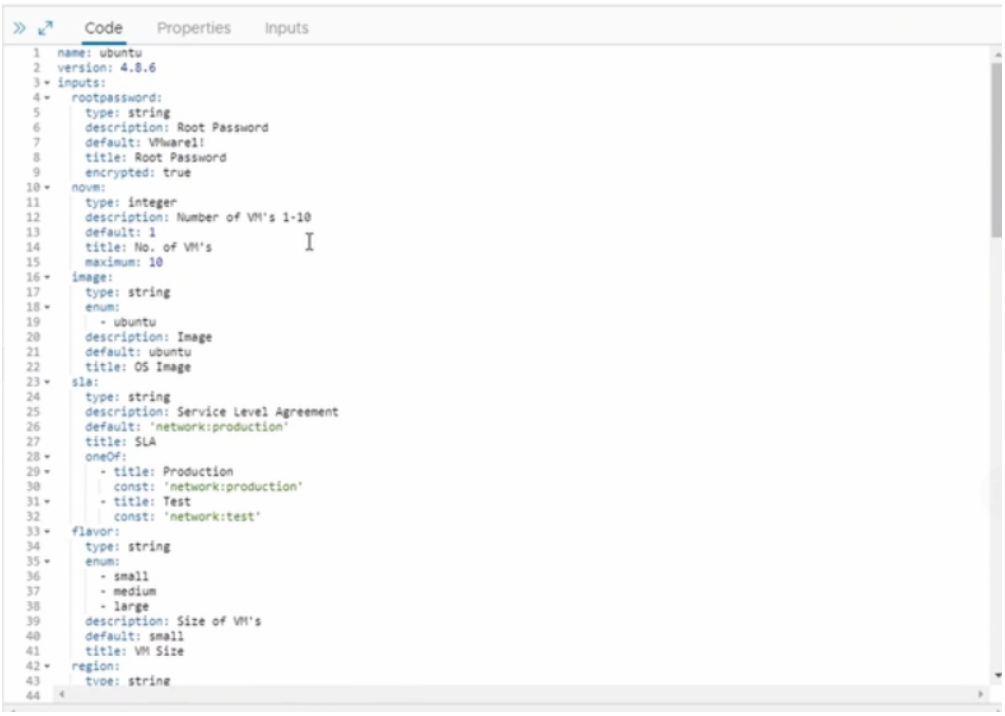
Все это позволяет быстро запустить в работу платформу автоматизации и предоставить разработчикам такую песочницу, где они смогут выбирать готовые элементы инфраструктуры и ИТ не придется создавать их самостоятельно.

Но важно не просто создать сервис на каких-то определенных версиях ПО, ведь оно быстро меняется: появляются новые версии баз данных, приложений, кластеров K8s и т. д. На создании и размещении сервиса в каталоге работа не заканчивается — его нужно дорабатывать, развивать, обеспечивать версионность. Маркетплейс хорош тем, что обеспечение этой версионности мы берем на себя. Все, что помещается в маркетплейс, уже протестировано и проверено на уязвимости.

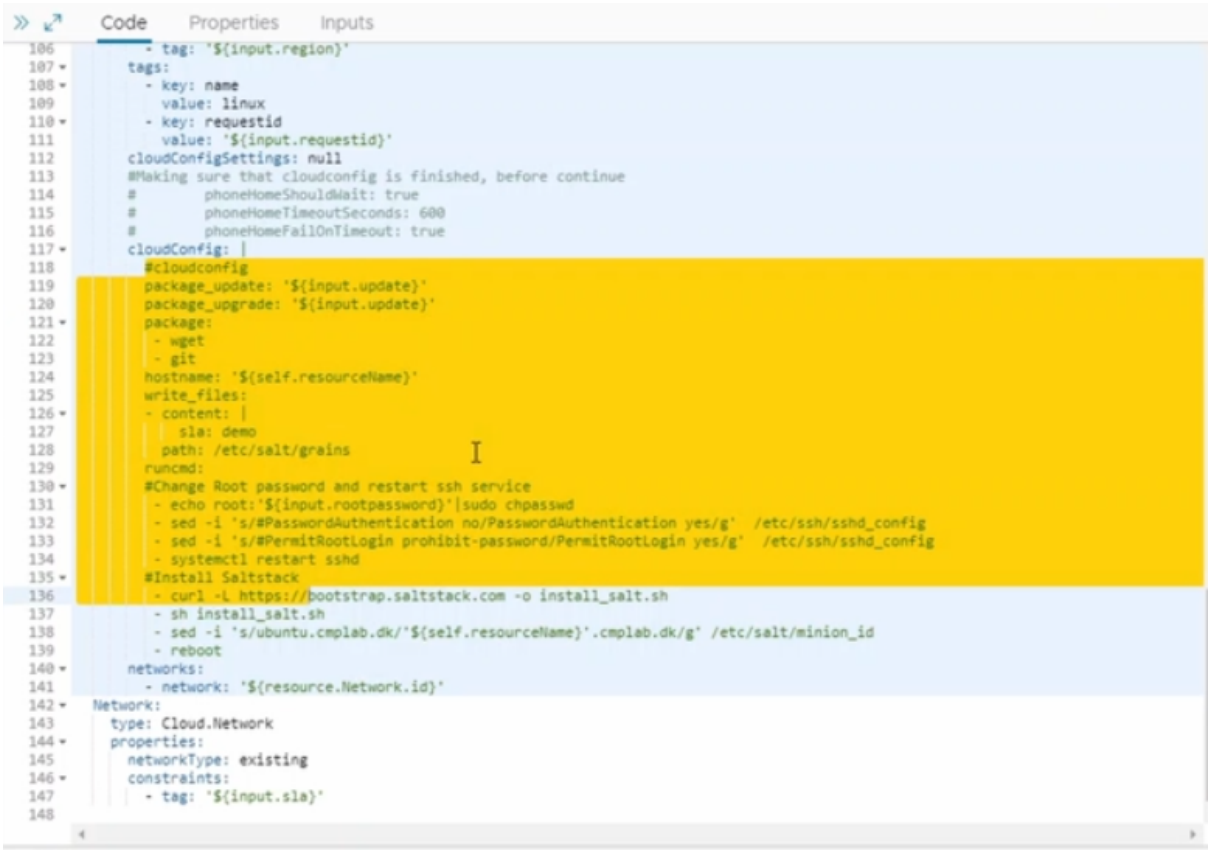
Вернемся к конфигурированию. Элементы можно методом drag-n-drop разместить на канве или доработать через конфигурирование YAML-файла.



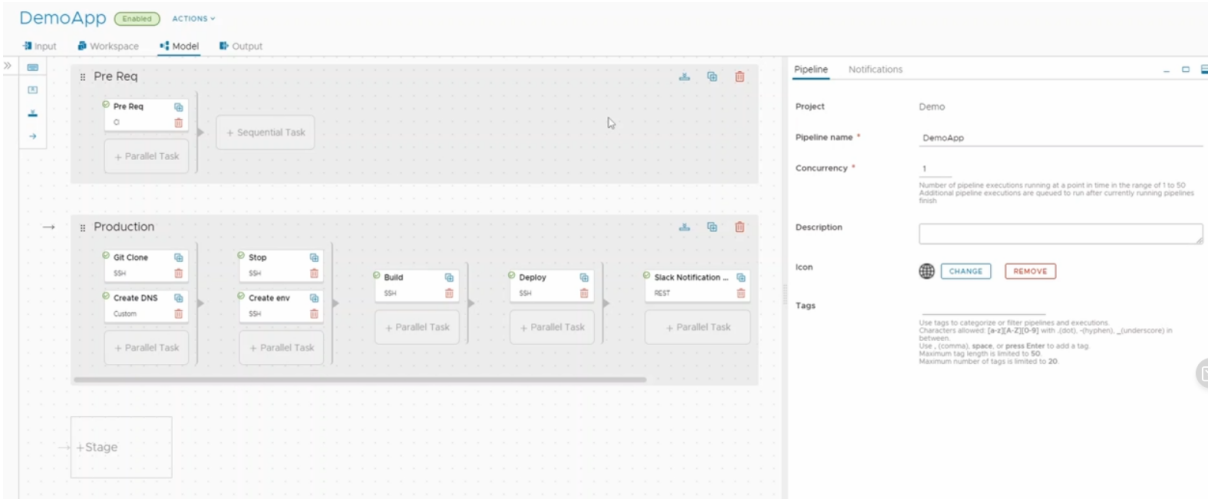
Файл конфигурации состоит из вводных параметров, которые должен заполнить разработчик (логин и пароль, место размещения и т. д.). Все это динамически влияет на логику развертывания сервиса на ресурсах того или иного провайдера и логику конфигурирования элемента.



Например, выделенный фрагмент относится к конфигурированию сервиса (ВМ) с точки зрения ОС:



Остановимся на организации пайплайнов (конвейеров) доставки. Наверняка вы уже знакомы с понятием CI/CD. Continuous Deployment с точки зрения инфраструктуры можно организовать на базе такого пайплайна, где все шаги оркестрации по доставке приложения на только что развернутую инфраструктуру выстраиваются в общий конвейер и исполняются.



- Pre Req — устанавливаем пререквизиты для работы конвейера.
- Production — через Git получаем описание (конфигурацию) сервиса, донастраиваем DNS, создаем Environment, разворачиваем из предварительно созданного каталога элемент сервиса, доустанавливаем приложение и, например, настраиваем уведомления в Telegram, Slack или другой мессенджер.

Такой конвейер позволяет нам еще сильнее ускорить доставку новых фич от разработчиков до продуктива.