РАЗРАБОТКА ИНФРАСТРУКТУРЫ АВТОМАТИЧЕСКОГО РАЗВЕРТЫВАНИЯ, ТЕСТИРОВАНИЯ И ОБСЛУЖИВАНИЯ СЕРВЕРНЫХ КОМПОНЕНТОВ ИНФОРМАЦИОННЫХ СИСТЕМ

Содержание

1 Введение

В настоящее время все чаще внедряются сложные механизмы для облегчения различных процессов проектирования и производства в компаниях различного масштаба. Из-за достижения высоких технологий в различных областях промышленности очень сильно возросли требования к программному продукту. В связи с этим, реализация программ стала гораздо сложнее и требует больших усилий.

Создавая программный продукт, работающий на удаленном компьютере - сервере и взаимодействующий с множеством клиентов, необходимо использовать множество средств, помогающих реализовывать, поддерживать и развивать разрабатываемый продукт. Из таких средств можно выделить различного рода программные библиотеки, программные и виртуальные среды, набор утилит для нужд программной реализации, ее отладки, тестов и запуска приложения.

Во все времена первостепенным средством для разработки являлась компьютерная операционная система ОС. Именно она содержала минимальный набор средств для создания продукта. Таким образом, имея только консоль и даже консольный текстовый редактор, можно было создавать программы. Но с развитием технологий требования усложнялись, подходы программирования становились сложнее и эффективнее. В наши дни недостаточно "пустой"операционной системы, чтобы реализовать сложное приложение. Программы наделили сложным графическим интерфейсом, они стали способны работать удаленно и использовать общие пользовательские ресурсы вместо личный локальных. Изменились и подходы к созданию таких приложений. Перед началом разработки необходимо оценить важность работы, назвать и рассмотреть основные стадии ее реализации от начала развития до внедрения. Сегодня проблемы развития продукта решают различные методологии и практики разработки софта. Среди них известны такие, как

- Водопадная методика
- Спиралевидная методика
- Инкрементальная методика
- V-Model
- Agile
- и другие

Для всех методик можно назвать общие этапы разработки ПО, и как минимум это тестирование и внедрение продукта. Если в процессе написания кода и сборки проекта используются известные среды разработки, то при выполнении вышеупомянутых этапов может не существовать подходящего решения для быстрой отладки, проверки работоспособности написанной программеы вскупе с другими компонентами системы. В данном случае необходимо реализовать собственные решения для эффективного тестирования.

2 Модель

Мультитенантность — элемент архитектуры программного обеспечения, где единый экземпляр приложения, запущенного на сервере, обслуживает множество организаций-клиентов. Мультитенантность противопоставляется архитектуре из множественных экземпляров (англ. multiinstance), где для каждой организации-клиента создаются отдельные программные экземпляры. В мультиарендной архитектуре программные приложения работают одновременно с несколькими конфигурациями и наборами данных нескольких организаций, а каждая организация-клиент работает со своим экземпляром виртуального приложения, видя только свою конфигурацию и свой набор данных.

2.1 Схема инфраструктуры

Для каждого разрабатываемого приложения известны сценарии его использвания и способы предоставления услуг. В данном случае, финальным результатом будет являться вебприложение типа SaaS[1]. В качестве точки входа выступает сслыка в приложение для каждого клиента очередной организации. После этого каждому пользователю доступна площадка, в которой он может хранить различные файлы-модели, полученные в САПР приложениях, включая любую доументацию. На уровне организации доступна единая область памяти, способная быть ограниченной для любого члена организации.

В итоге получаем информационную систему с возможностью изолированно обслуживать пользователей из разных организацией (т.е. независимых подписчиков SaaS) в рамках одного сервиса. Основным здесь является соблюдение изолированности подписчиков друг от друга. Действительно, клиенты не обрадуются, если данные, которые они хранят в SaaS приложении, будут доступны при поиске для других клиентов. Это явное нарушение изолированности.

Для реализации можно взять один сервер, одну операционную систему, одну базу данных. Механизм изоляции будет осуществляться с помощью особого хранения данных в таблицах БД и записью файлов в общую файловую сисему, а так же обработкой и перенаправлением web-запросов от клиентов на микросервисы.

Для достижения поставленных целей можно рассмотреть слудующую архитектуру сервера:

ДНС-сервер
Прокси-сервер
Backend
БД
Файловое хранилище
Средства инфраструктуры

Каждый уровенень выполняет определенную роль. В совокупности с развитыми средствами автоматизации инфрастуктуры они реализуют целостную структуру информационной системы приложения. Далее будут рассмотрена каждый из них.

[1]SaaS (англ. software as a service - программное обеспечение как услуга) — одна из форм облачных вычислений, модель обслуживания, при которой подписчикам предоставляется готовое прикладное программное обеспечение, полностью обслуживаемое провайдером. Поставщик в этой модели самостоятельно управляет приложением, предоставляя заказчикам доступ к функциям с клиентских устройств, как правило через мобильное приложение или веб-браузер.

2.2 Уровень ДНС-сервера

Доменное имя — основа для имени сайта, с помощью которой можно попасть на его ресурс.
Доменная зона — совокупность доменных имён определённого уровня, входящих в конкретный домен.
Делегирование — это передача контроля над частью доменной зоны другой ответственной стороне.

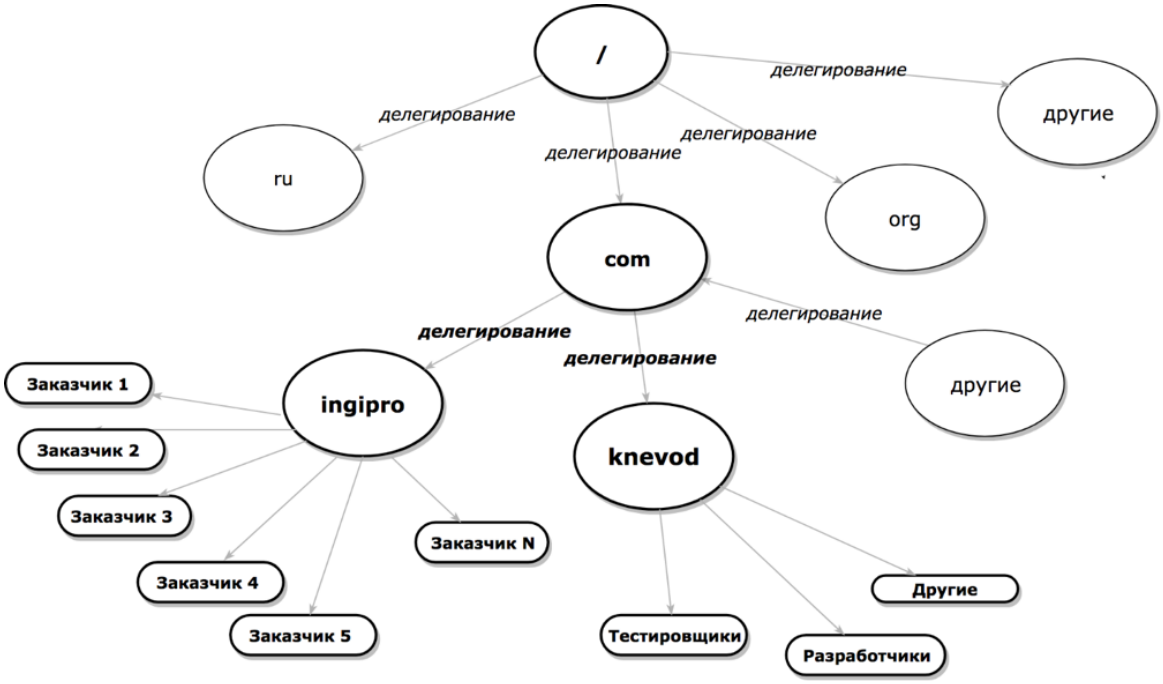

Рис. 1: Иерархия доменных имен

ДНС[2] сервер в данной ситуации играет важную роль. На уровне доменных имен происходит разделенный досуп к инфрмационной системе. Это самый верхний уровень, который помогает реализовать множество точек входа в систему. Для того, чтобы добавить еще одного польователя, необходимо будет добавить в доменную зону новое имя, выделяемое для заказчика.

Так, например, если информационная система доступна по имени ingipro.com, то дабавив имя new-company в зону и получив адрес **new-company.ingipro.com**, мы можем назначить клиенту этот адрес. Остальной механизм валидации осуществляется на следующих уровнях.

В данном проекте используется Bind в качестве ДНС-сервера.

2.3 Уровень Прокси-сервера

На роль Прокси-сервера[3] возлагается много задач. Среди них:

- Обработка запрашиваемого адреса с целью перенаправления на другой адрес

[2]DNS (англ. Domain Name System «система доменных имён») — компьютерная распределённая система для получения информации о доменах. Чаще всего используется для получения IP-адреса по имени хоста (компьютера или устройства), получения информации о маршрутизации почты, обслуживающих узлах для протоколов в домене.

[3]Прокси-сервер (от англ. proxy — «представитель», «уполномоченный») — промежуточный сервер в компьютерных сетях, выполняющий роль посредника между пользователем и целевым сервером, позволяющий клиентам как выполнять косвенные запросы (принимая и передавая их через прокси-сервер) к другим сетевым службам, так и получать ответы.

- Фильтрация и перенаправление запросов на уровень Backend
- Выдача специальных ресурсов для запрашиваемого доменного уровня
- Выдача статических ресурсов на сторону клиента

На этом уровне Веб-сервер[4] Nginx, выступающий в роли Прокси-сервера, принимает HTTP-запросы от клиентов, перенаправляет их на необходимые микросервисы, которые в свою очередь реализуют основную логику работы разрабатываемого приложения. Далее возвращает в ответ HTTP-ответ с необходимым содержимым для отображения графического дизайна и данных в браузере клиента.

2.4 Уровень Backend

Backend[5] является основной логической частью веб-приложения. Его архитектура бывает монолитной и микросервисной. В текущем проекте реализуется микросервисная архитекрура, состоящая из отдельных приложений, выполняющих специальные задачи. Так, например, существует микросервис, отвечающий за авторизацию пользователя в системе; микросервис, отвечающий за передачу файловых данных клиенту; микросервис, в основную задачу которого входит хранение множества атрибутов клиента; микросервис, отвечающий за отрисовку графических данных и так далее...

Каждый микросервис работает на специально выделенном сетевом порту сервера. В зависимости от поступившего от клиента HTTP-запроса происходит перенаправление на конкретный микросервис через сетевой порт. Во время работы Backend оперирует базой даных и файлами, располагающихся на жестком диске. Так проиходит обмен между уровнями инфраструктуры.

2.5 Уровень Базы данных

Огромное количество информации пользователей хранится в базе данных. Для ее управления используется СУБД[6] PostgreSQL. Через нее микросервисы получают данные пользователей, хранящиеся в таблицах. Для достижения принципа мультитенантности реализуется следующий подход: пусть выделенная экосистема для клинета, называется new-company (по имени домена). Экосистема[7] в БД представляется двумя таблицами `attr_new-company`, `entity_new-company` и одной последовательностью `entity_seq_new-company`. Каждая заводимая экосистема хранит три объекта базы данных, которые относятся только к ней. Таким образом изолируются данные клиентов на уровне БД.

2.6 Уровень файлового хранилища

Уровень файлового хранилища представляет из себя часть файловой системы, находящейся на локальном или удаленном диске. Тут хранятся гораздо более тяжелые данные мультимедийного характера: документы, изображения, чертежи, аудио и видео записи. Область файлового хранилища закрыта от системных пользователей на уровне ОС и доступ туда имеют только программы, относящиеся к сервису. Как было сказано выше, любые данные

[4]Веб-сервер — сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными.

[5]Backend — программно-аппаратная часть сервиса. Набор приложений сервиса, работающих на стороне сервера.

[6]СУБД — Система управления базой данных

[7]Изолированное информационное пространство клиента для работы с сервисами приложения. Любые данные хранимые в экосистеме, связаны только с ней и клиентом на всех уровнях модели сервера.

Экосистемы имеют отношение только к ней самой. Поэтому в способ хранения информации Экосистемы выглядит слудующим образом:

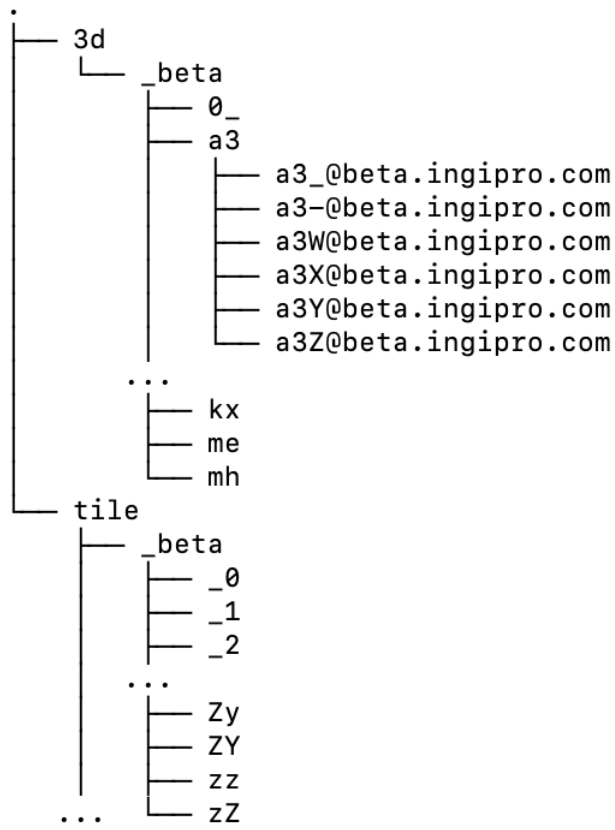

Рис. 2: Структура файлового хранилища

На (рис. ??) показан пример для экосистемы под названием `_beta`. Таким структурированным образом хранится информация любых документов, изображений и моделей систем САПР в виде отдельных тайлов.

2.7 Средства инфраструктуры

Все вышеописанные уровни модели сервера реализуют определенные задачи. Все вместе они формируют программную систему, которая предлагается как конечный продукт. За реализацию каждого уровня отвечает определенный разрабочик или команда разработчиков. Они используют собственные средства, обычно не совместимые с продуктами других уровней. Это разные языки программирования, сетевые интерфейсты, логика работы приложения, конфигурационные файлы. Часто для внедрения новой версии, например backend, требуется внесения изменений в веб-сервере или DNS-сервере. Это несет некоторые трудности, так как требует работы нескольких команд или людей.

Для решения подобных проблем выступают различные сервисы непреревной интеграции и доставки. Они оперируют специальными агентами, которые выполняют конкретные задачи при замене версий. Главной целью средств инфраструктуры выступает автоматизация процессов обновления, останова и запуска. Это всегда уникальные решения, которые адаптированы для текущей системы.

3 Автоматизация процессов

Под автоматизацией процессов будем понимать набор действий по сборке программного продукта (backend, frontend), действий по управлению сервисами (останов, запуск, перезапуск), действий по работе с конфигурационными файлами используемых программ и действий по обновлению версий микросервисов, сайта и прочее.

3.1 Автоматизация сборки микросервисов

Много действий по разработке ПО связано с постоянными изменениями в коде, его дополнении и тестировании. Поэтому очень часто приходится пересобирать из исходного кода часть программного продукта. Особенно часто обращает на себя внимаение Backend с его микросервисами.

Для достижения цели конечного продукта используется 2 сервера: отладочный и производственный. Цели отладочного сводятся непосредственно к разработке ПО, ее отладке и тестированию. Финальную версию ПО выкладывают на производственный сервер, где клиенты пользуются сервисом, пока на отладочном происходит разработка новой версии программного продукта.

Ниже приводится автоматический алгоритм, используемый в текущем проекте:

1. Проверка наличия и синтаксиса конфигурационного файла.
2. Обновление Исходного кода из удаленного репозитория.
   (a) Проверка текущей ветки;
   (b) Обновления копии локального репозитория из удаленной ветки.
3. Обновление и модификация UPP[8].
   (a) Сброс локального репозитория до первоначального состояния;
   (b) Обновление из официального репозитория;
   (c) Наложение патчей, подготовленных программистами микросервисов в отдельной локальной ветке.
4. Выполнение сборки микросервисов.
   (a) Проверка разрешения на пересборку конкретного микросервиса.
   (b) Выполнение сборки проекта микросервиса с помощью встроенного во фреймворк UPP инструмента umk. При сборке указываются необходимые для сборки каждого микросервиса флаги и ключи сборки, конечный путь для новых собранных бинарных файлов.
   (c) Установка метки статуса сборки в скрытый временный файл;
   (d) Сохранение старой версии микросервиса в файл в постфиксом '.old' в директорию с микросервисом.
5. Вызов модуля тестирования микросервисов
   (a) Проверка статуса сборки;
   (b) Вызов модуля UNIT тестов;

[8]U ++ — это кроссплатформенная среда быстрой разработки приложений на C ++, ориентированная на производительность труда программистов. Он включает в себя набор библиотек (GUI, SQL и т.д.) И интегрированную среду разработки.

(c) Вызов модуля регрессионного тестирования.

6. Корректное завершение процессов.

   (a) Проверка статусов тестирования;

   (b) Проверка разрешения на перезапуск микросервиса;

   (c) Чтение из конфигурационного файла переменной, содержащей максимальное время ожидания программного завершения микросервиса;

   (d) Отправка процессу программного сигнала завершения SIGINT и старт таймера;

   (e) Ожидание завершения. Отправка SIGKILL, если процесс не успел завершиться за указанный интервал времени;

   (f) Проверка закрытия сетевых портов;

   (g) Установка метки статуса сокета, на котором работал микросервис;

7. Запуск микросервисов и проверка их состояния.

   (a) Проверка статуса сетевого порта;

   (b) Запуск и демонизация[9] микросервиса;

   (c) Ожидание некоторого времени и проверка состояния процесса.

3.2 Автоматизация перезапуска микросервисов

Для реализации перезапуска микросервисов потребовался отдельный скрипт, который отвечает за передачу сигнала прерывания в микросервис и обработку реакции останавливаемой программы на сигнал. Общий алгоритм останова и перезапуска микросервиса в автоматическом режиме следующий:

1. Проверка разрешения на перезапуск, отмеченным в конфигурационном файле.

2. Проверка наличия рабочих процессов микросервиса в системе.

3. Отправка сигнала SIGINT на программное завершение.

4. Ожидание корректного завершения микросервиса и всех его дочерних процессов в течение указанного в конфигурационном файле времени. Если процесс не завершился самостоятельно, производится отправка сигнала принудительного завершения SIGKILL.

5. Ожидание закрытия сетевого порта микросервиса. Если порт не закрывается в течение указанного времени (10 секунд), то автосборка отражает это состояние в параметре статуса порта.

6. Проверка состояние сетевого порта по флагу. Если порт не был закрыт, то процесс автосборки завершается с определенным кодом.

Алгоритм ручного перезапуска вызывается в режиме '-restart' и аналогичен автоматическому. Алгоритмы останова и запуска требуют каждый проверку порта, для этого и введен параметр состояния порта. Это необходимо для гарантированного завершения микросервиса. Реализованный механизм останова и запуска микросервиса достаточно гибкий, так как включает в себя несколько проверок. Это позволяет максимально корректно завершить работу микросервиса, в каком бы он состоянии не находился.

[9] Демонизацией процесса называют запуск программы для работы в специальном фоновом режиме даже если пользователь выйдет из системы.

3.3 Автоматизация замены версии микросервисов

Процесс замены версий на отладочном и производственном серверах отличается. Любая сборка проекта происходит на отладочном сервере. В данном случае, как и в случае с автоматизацией сборки и перезапуска микросервисов, учавствует специльно реализованная программа. На отладочном сервере для замены версии микросервисов можно воспользоваться этой утилитой для сборки. Так она перезапишет локальные версии программ. С производственным сервером немного сложнее, так как нужно локально собрать ПО и доставить его. Алгоритм можно описать так:

1. Проверка наличия и синтаксиса конфигурационного файла автосборки
2. Проверка существования бинарных файлов и формирование их списка
3. Установление SSH соединения с боевым сервером
4. Остановка микросервисов, используя тот же алгоритм автосборки
5. Создание бэкапа каждого микросервиса
6. Перезапись старых бинарных файлов новыми
7. Запуск микросервисов
8. Разрыв SSH соединения

На производственном сервере существует две версии одновременно работающих Backend: Beta[10] и Production[11]. Рис. ?? показывает нагладное отношение объектов и последовательность обновления микросервисов.

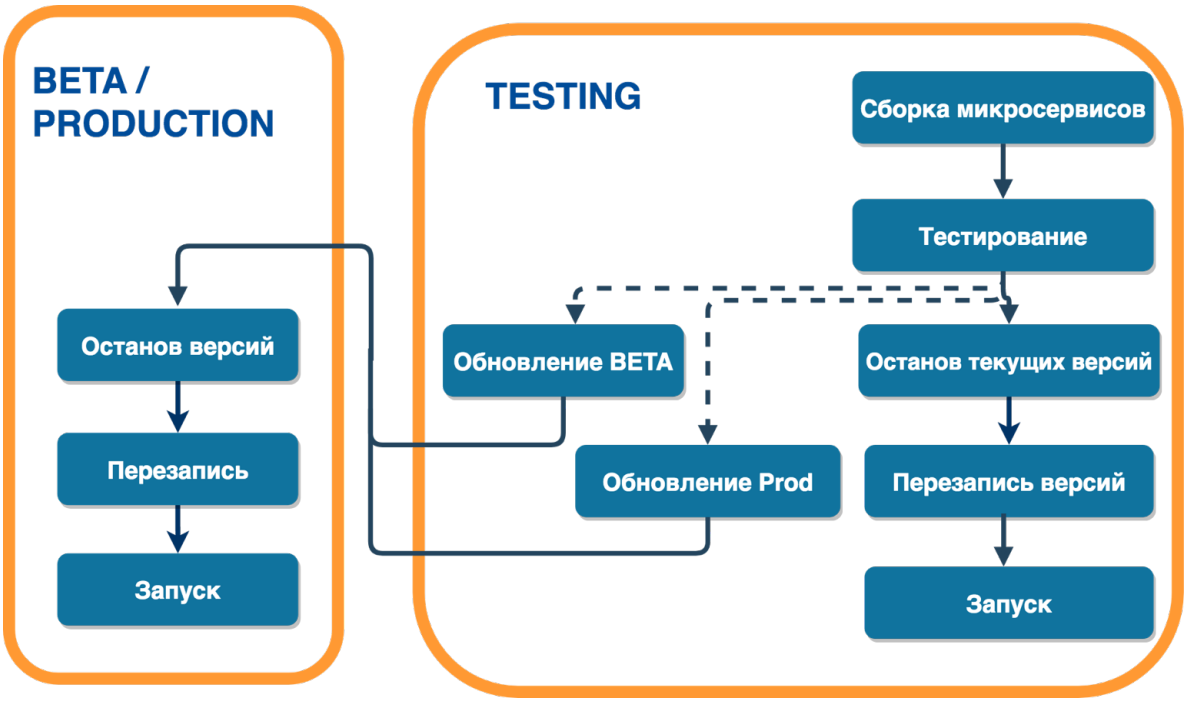

Рис. 3: Схема замены версий на серверах

[10]Бета-версия продукта. Используется для предрелиза продукта. В очередной раз команда убеждается, что все работает корректно.

[11]Конечый вариант продукта, доступный для использования клиентам.

3.4 Автоматизация создания доменного имени

Доменное имя - это самый высокоуровневый атрибут экосистемы клиента. Именно он является визуальной составляющей точки входа в систему, т.е. клиент его видет самым первым в строке браузера. Более того, доменное имя является основной характеристикой экосистемы. Все низлежащие уровни модели сервера оперируют этим именем, а на уровне ДНС-сервера обепечивается "открытие ворот"в систему для клиента.

Необходимой задачей является добавление нового доменного имени или удаление старого. Домен второго уровня, например, `ingipro.com` является основным для сервиса. Домены третьего уровня являются уникальными для каждой компании, например, `beta.ingipro.com` или `new-company.ingipro.com`. Вот эти домены и предстоит добавлять и удалять из доменной зоны, чтобы открыть или закрыть доступ к сервису для клинета.

Чтобы вручную это сделать, необходимо выполнить слудующие действия:

1. Развернуть и настроить ДНС-сервер.
2. Добавить доменное имя в файл зоны.
3. Инкриментировать идинтификационный номер файла доменной зоны.
4. Сохранить изменения.
5. Выполнить команду ДНС-сервера на чтение нового файла доменной зоны.
6. Проверить доступность нового доменного имени.

В жизни это окажется рутинной работой, утомительной и неэффективной, а так же велика вероятность ошибиться. Поэтому для этой цели реализуется специальный сценарий, реализующий вышеперечисленные действия, причем с дополнительными проверками синтакисиса и кодов завершения команд.

3.5 Автоматизации регистрации нового пользователя

Новый пользователь - клиент, представляет собой некую организацию. Создание точки входа сопровождается созданием экосистемы, в рамках которой клиент будет работать. Регистрация экосистемы происходит с помощью API[12], реализованного командой разработчиком микросервисов.

В процессе регистрации пользователя, а именно в процессе создания экосистемы, учавствуют все уровни архитектуры сервера. Первое — это ДНС-сервер. С помошью полученной выше утилиты мы регистрируем доменное имя. Второе — база данных PostgreSQL. Роль базы данных состоит в хранении данных в структурированном табличном виде. Третья составляющая — это веб-сервер Nginx. Четвертый компонент это один из микросервисов. Он отвечает за авторизацию пользователей в системе, а также за контроль доступа к запрашиваемым данным любого типа и их изменение.

Чтобы зарегистрировать новую экосистему, необходимо создать суперпользователя для экосистемы, далее получить ключ для этого пользователя, а затем уникальный одноразовый номер, для создания узла. Каждый HTTP-запрос формируется с помощью утилиты curl, содержащей в качестве аргументов синтаксис API. Затем он отправляется через сокет на порт, на котором работает веб-сервер. Nginx обрабатывает запрос и передает данные в соответствующий микросервис. Этот микросервис выполняет требуемый запрос веб-сервера используя базу данных, и передает данные веб-серверу, а он клиенту. Так мы получаем метаданные данные. В итоге имеет зарегистрированную экосистему с пользователями для клиента.

[12]API (программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

Задачей автоматизации стоит мгновенное создание экосистемы для организации, а также ее изменение, если необходимо добавить или удалить пользователей. При этом необходимо реализовать удобное взаимодействие с процессом регистрации при помощи JSON-файла и файлом с данными экосистемы. Для этих целей реализован скрипт на языке Bash. Входным параметром для него является файл экосистемы, названный по доменному имени в формате RFC 1033. Выходными данными является файл формата JSON, содержащий ключ пользователя root, название и список всех пользователей экосистемы с собственными ключами.

Алгоритм работы утилиты регистрации экосистемы следующий:

1. Проверка наличия файла экосистемы, имеющего название организации.
2. Проверка наличия выходного файла JSON с ключами пользователей.
3. Если такого файла нет, происходит создание экосистемы с нуля.
4. Если уже имеется конечный JSON файл, тогда происходит добавление или удаление пользователей, в зависимости от того, как мы меняем входной файл экосистемы.
5. Вызов скрипта управления файлом доменной зоны с ключем добавления домена. Если домен не существует, то он добавляется в файл зоны DNS-сервера.
6. Вывод полученных уникальных ключей пользователей для авторизации в системе Ingipro.