ПРОЕКТИРОВАНИЕ РАСПРЕДЕЛЕННЫХ СИСТЕМ АВТОМАТИЧЕСКОГО ОБНОВЛЕНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Плетнев Андрей Владимирович

Независимый исследователь,

Директор ТОО «SimCo Soft»,

Руководитель группы разработки ТОО «OneBill»,

Республика Казахстан, г. Алматы.

DESIGNING DISTRIBUTED AUTOMATIC SOFTWARE UPDATE SYSTEM

Pletnev Andrey Vladimirovich

Independent researcher,

CEO «SimCo Soft» LLP,

Team lead «OneBill» LLP,

Republic of Kazakhstan, Almaty city.

Аннотация. Данная статья предлагает вниманию читателя практическое руководство по проектированию и разработке распределенной клиент-серверной системы, предназначенной для выполнения автоматического обновления программного обеспечения на примере нетривиальной функциональности. Для сравнения приводятся примеры существующих систем и решений, применяемых для выполнения обновления программного обеспечения и даются обоснования их функциональной непригодности в приведенных обстоятельствах.

Abstract. This article offers the reader a practical guide to the design and development of a distributed client-server system designed to perform automatic software updates using the example of non-trivial functionality. For comparison, examples of existing systems and solutions used to perform software updates are given and justifications for their functional unsuitability in the given circumstances are provided.

Ключевые слова: обновление, backend, frontend, информационные системы, клиент-сервер, git, DevOps, CI\CD, deploy.

Keywords: update, backend, frontend, information systems, client-server, git, DevOps, CI\CD, deploy.

Введение

Любой специалист, так или иначе связанный с производством программного обеспечения, понимает, что процесс регулярного выпуска обновлений разработанного им программного обеспечения является неотъемлемой частью жизненного цикла [2] любого программного продукта. Далеко ходить за примерами не нужно: все мы регулярно получаем обновления программных продуктов, которыми пользуемся каждый день: операционные системы, офисные пакеты, инструменты разработчика, системные утилиты, мобильные приложения и многое другое.

На заре компьютерной эры, в 70е-90е годы прошлого столетия, доставка обновлений программных продуктов до конечных потребителей осуществлялась по обычной почте или курьерскими службами в конвертах и бандеролях – сначала на перфокартах или перфолентах, а позже на магнитных лентах и магнитных гибких дисках. Одними из последних типов носителей информации, использовавшихся для доставки программного обеспечения и обновлений для него подобным способом, были оптические компакт-диски.

С появлением и повсеместным распространением сети Интернет многие компании-разработчики программного обеспечения начали публиковать на своих сайтах свои программные продукты и обновления для них, позволяя своим пользователям самостоятельно скачивать, устанавливать или выполнять обновление установленных ранее программ.

На этом технический прогресс не остановился. Вскоре начали появляться программные продукты, которые стали самостоятельно выполнять проверку на наличие обновлений и предлагать пользователю выполнить их загрузку и установку, показывая список исправлений и улучшений в новой версии. В последнее время, в особенности на мобильных платформах (iOS и Android), обновление программ и вовсе выполняется в фоновом режиме без участия пользователя. Механизм такой автоматизации весьма прост: программа с определенной периодичностью отправляет через сеть Интернет запрос на *Сервер обновлений* компании-разработчика, в котором передает идентификатор продукта и номер текущей версии. В случае наличия новой версии, *Сервер обновлений* в ответном сообщении передает текстовую информацию со списком исправлений и улучшений для

отображения пользователю, а также список файлов для закидывания и замещения на компьютере или ином устройстве пользователя.

Примером наиболее тривиальной функциональности могут послужить упомянутые выше системы обновления приложений для мобильных платформ, где выполняется простая сверка номеров версий и отдача пакетной сборки приложения в виде одного файла.

Другим методом доставки обновлений в производственную среду является *Непрерывная Интеграция и Непрерывная Доставка* (CI\CD). Этот метод включает в себя применение набора принципов и практик, которые позволяют разработчикам чаще и надежнее развертывать изменения программного обеспечения [5]. Данный набор практик очень хорошо подходит для систем с одной производственной средой – когда компания-разработчик занимается развитием продукта для одного клиента, либо сама эксплуатирует собственный продукт. Скажу больше: этот набор практик подходит и для наших задач тоже, вот только использовать его мы будем немного иначе – поставки обновлений будут осуществляться не напрямую в производственную среду, а в специальный каталог на нашем *Сервере обновлений* (см. ниже).

В ряде случаев функциональности описанных систем доставки обновлений бывает недостаточно. Как быть, когда структура Ваших продуктов и сеть распределения их доставки более сложная? Для большей ясности приведу пару примеров упомянутой сложной структуры:

Пример 1. Ваша команда ведет разработку программного обеспечения, например, для сети банкоматов или платежных терминалов. Каждая поставка обновлений включает в себя множество файлов. Сеть разделена на регионы или филиальные представительства, в каждом из которых разные настройки и разный состав файлов, поставляемых для обновления.

Пример 2. Имеется информационная система, установленная у десяти Ваших клиентов. У каждого из этих клиентов могут стоять разные подвиды Вашего программного продукта (например: «minimal», «pro», «unlimited»), либо Вы выполняете лицензирование Ваших клиентов с разной степенью ограниченности функционала, чем определяется состав файлов, поставляемых для обновления, либо Ваш *Продукт* поставляется под разные операционные системы и платформы. Графический интерфейс пользователя информационной системы адаптирован отдельно под каждого *Клиента*. Инфраструктура информационной системы не ограничивается условно одним корпоративным сервером, где развернут *backend* и откуда выполняется раздача *frontend* в браузеры корпоративных пользователей – кроме этого, в корпоративной сети клиента работает некоторое количество разнообразных программных модулей, для которых также нужно выполнять обновления.

Как в таких случаях организовать систему автоматизированной доставки обновлений? В данной статье я предлагаю читателю создать собственный централизованный сервис автоматизированной доставки и распределения обновлений программного обеспечения до конечного потребителя, функциональные возможности которого выходят за рамки упомянутой выше тривиальной функциональности. Материал статьи будет полезен системным архитекторам, проектным менеджерам и, конечно же, разработчикам программного обеспечения, компании которых занимаются разработкой и сопровождением информационных систем.

Инфраструктура распределенной системы автоматического обновления программного обеспечения

За основу для построения нашей распределенной системы обновлений мы возьмем приведенный выше **Пример 2**, как более интересный с точки зрения наибольшей функциональной полноты. В приведенном примере четко прослеживается взаимосвязь цепочки узлов в схеме производства и распределения обновлений программного обеспечения: производство → загрузка на центральный Сервер обновлений → раздача обновлений на корпоративные серверы клиентов с центрального Сервера обновлений → раздача обновлений с корпоративного сервера Клиента на точки обновления Клиента.

На основании этого мы можем построить схему взаимодействия сервисов в распределенной системе обновления программного обеспечения. Данная схема представлена на рисунке 1.

Рисунок 1. «Схема взаимодействия сервисов обновления программного обеспечения»

Внимательный читатель, наверное, обратил внимание, что представленная схема показывает последовательную цепочку из узлов типа клиент-сервер [3]. Идея состоит в том, чтобы последовательно выполнять передачу обновляемых файлов от звена к звену, пошагово делегируя основную задачу Системы – выполнить доставку обновлений программного обеспечения до конечного потребителя, но контроль за выполнением всего процесса осуществлять централизованно.

Первым узлом в нашей цепочке является процесс производства обновлений. Все изменения в коде своих продуктов, как известно, компания-разработчик хранит в *git-репозиториях*. Специалист компании-разработчика, ответственный за поставки и развертывание программного обеспечения в производственную среду (*DevOps*), применяя методы и практики *CI/CD*, выполняет настройку доставки изменений из ветки *Master* *git-репозитория* проекта в каталог «*CI/CD directory*» Сервера обновлений (*Main Update Server*) так, как будто это и есть производственная среда. Далее Администратор Сервера обновлений, через Панель управления (*Administrator Frontend*), формирует из полученных файлов Релиз программного продукта.

После того как Релиз программного Продукта будет сформирован и опубликован, он становится доступным для получения Клиентами (*Clients*) – администратор информационной системы Клиента получает уведомление о выходе новой версии Продукта и может, в зависимости от настроек своей Информационной системы, получить это обновление в ручном, либо в автоматическом режиме. За этот процесс отвечает Сервис Обновления (*Update Service*), установленный на корпоративном сервере Клиента (*Corporate Information System Server*) и являющийся частью Информационной системы Клиента. После того, как Сервис Обновления на Корпоративном сервере Клиента получит все файлы Релиза, такой Релиз становится доступен для распределения по Точкам обновления (*Update Points*) внутри корпоративной сети Клиента. Контроль и мониторинг этого процесса предоставляется Администратору Информационной системы Клиента. В своей системе обновления Вы можете добавить эту функцию также и Администратору Центрального Сервера обновлений.

Следует понимать, что представленная схема доставки и распределения обновлений программного обеспечения способна одновременно поддерживать работу с десятками программных Продуктов для сотен и тысяч Клиентов. Для этого в каталогах «*CI/CD directory*» и «*File Storage*» просто нужно создавать подкаталоги – отдельно для каждого нового программного Продукта.

Центральный Сервер обновлений

Как видно на схеме, представленной на рисунке 1, Центральный Сервер обновлений (*Main Update Server*) является классическим веб-сервисом. Функционально он представляет собой управляемый репозиторий файлов для всех проектов компании разработчика и отвечает за распределение доставки этих файлов Клиентам. Веб-сервис Сервера обновлений можно реализовать как в виде монолита, так и в виде двух отдельных микро-сервисов [4]: один будет имплементировать API-методы для Клиентов Сервера обновлений (*Clients*), второй – методы для Панели управления Сервером обновлений (*Administrator Frontend*). Давайте рассмотрим минимально необходимый список методов нашего веб-сервиса.

API-Методы для Клиентов Сервера обновлений

1. Авторизация Клиента;
2. Обработка запроса Клиента о наличии новой версии (нового Релиза) программного продукта;
3. Обработка запроса на получение файла;
4. Обработка запроса с подтверждением о получении Клиентом указанного файла.

API-Методы для Панели управления Сервером обновлений

1. Авторизация пользователя (Администратора) панели управления Сервером обновлений;

2. Работа с учетными записями пользователей панели управления *Сервером обновлений*: получение списка, добавление и изменение записей, блокирование и разблокирование пользователей;
3. Работа со справочником *Продуктов*: получение списка, добавление и изменение записи;
4. Работа со справочником подвидов *Продуктов*: получение списка, добавление и изменение записи;
5. Работа со справочником файловых групп: получение списка, добавление и изменение записи;
6. Работа со справочником файлового репозитория: получение списка файлов, добавление и изменение параметров файлов, отслеживание изменений в каталоге «*CI\CD directory*» и регистрация новых файлов в таблице справочника файлового репозитория;
7. Привязка файлов к подвидам *Продуктов*: получение списка файлов, входящих в состав указанного подвида *Продукта*, добавление файла в этот список, удаление файла из списка;
8. Работа со справочником *Клиентов*: получение списка, добавление и изменение записи, определение доступа *Клиента* к *Продуктам*;
9. Управление *Релизами Продуктов*: Создание и публикация *Релиза* на основании изменений в каталоге «*CI\CD directory*»;
10. Получение отчета о ходе выполнения обновления. Просмотр истории обновлений.

Для реализации API-методов для сервисов рекомендую придерживаться набора правил по организации обмена данными между компонентами системы REST. Это позволит повысить производительность и масштабируемость сервисов, а также упростить унифицированный интерфейс между ними [6].

На основании предоставленного списка необходимых методов и функций мы можем построить обобщенную схему базы данных для нашего *Сервера обновлений*:

Рисунок 2. «Схема базы данных Центрального Сервера обновлений»

Давайте рассмотрим назначение таблиц в представленной на рисунке 1 схеме:

Products – Справочник программных *Продуктов*. Хранит наименования и описания *Продуктов* компании разработчика, файлы которых хранятся в файловом репозитории *Сервера обновлений*.

ProductVersion – Справочник подвидов программных *Продуктов*. Добавьте такой справочник в Вашу базу данных, если Ваши *Продукты* имеют подвиды с разной степенью ограниченности функционала, либо поддерживаются сборки для разных операционных систем и/или платформ.

Clients – Справочник *Клиентов* компании разработчика, которым выполняется поставка обновлений программного обеспечения. По сути, данная таблица хранит учетные записи Клиентов, обращающихся на Центральный Сервер обновлений за проверкой наличия и за получением обновлений.

ClientsProducts – Таблица для хранения связей *Продуктов* и *Клиентов*. Определяет, обновления каких подвидов *Продуктов* будут доступны указанному *Клиенту*.

FileGroups – Справочник типов файлов. Позволяет Администратору *Сервера обновлений* группировать файлы репозитория по видам. Например: *графика, конфигурационные файлы, файлы интернационализации, бинарные и исполняемые файлы* и т.д.

FileRepository – Справочник файлов, входящих в файловый репозиторий *Сервера обновлений*. Минимально рекомендуемый набор атрибутов для данной таблицы:

- **id** – идентификатор файла. Автоинкрементное поле;
- **product_id** – идентификатор продукта, к которому относится файл;
- **filegroup_id** – идентификатор типа файла;
- **filename** – имя файла;
- **cicd_path** – путь к файлу в каталоге «*CI\CD directory*». Указывает *Серверу Обновлений* путь к новой версии файла в каталоге «*CI\CD directory*»;
- **server_path** – путь к файлу на Сервере обновлений. Указывает *Серверу Обновлений* путь к файлу в файловом репозитории (каталог «*File Storage*»), где следует брать файл для отправки *Клиенту*;
- **client_path** – путь к файлу на клиенте. Указывает *Клиентскому модулю обновлений*, куда следует поместить файл, полученный в процессе обновления.

ProductVersionFiles – Таблица, определяющая состав файлов, входящих в состав подвида *Продукта*.

ProductReleases – Таблица, хранящая информацию о *Релизах Продуктов*:

- номер версии *Релиза*;
- информация о *Релизе* для пользователя в формате «что нового»;
- признак «*опубликован*». Определяет, доступен ли сформированный *Релиз* для получения *Клиентами*.

FileVersion – Таблица, хранящая информацию о версии файла, вошедшего в *Релиз*. Для бинарных исполняемых файлов, поддерживающих включение номер версии в секцию строковых ресурсов, будет полезно написать функцию, которая будет извлекать этот номер и записывать его в эту таблицу. Для остальных типов файлов вместо номера версии следует вычислять и записывать hash файла. Алгоритм, вычисляющий hash для файлов Вы можете выбрать любой, будь то MD5, CRC32, SHA или любой другой на Ваше усмотрение.

FileStates – Справочник имен состояний процесса получения файла *Клиентом*. Хранит текстовые наименования состояний. Например: «*Ожидает получения*», «*Выполняется обновление*», «*Файл получен*», «*Ошибка*».

ClientFileVersionState – Таблица, хранящая состояния передачи файлов, получаемые в ходе выполнения процесса обновления файлов *Клиента*.

Следует отметить, что схема базы данных, представленная на рисунке 2, состав ее таблиц и связей между ними, а также состав полей в ее таблицах, даны для общего понимания примерной структуры данных *Сервера обновлений*. В Вашей реализации *Сервера обновлений* эта структура может быть другой – все зависит от задач, которые будет решать Ваш *Сервер обновлений*.

Панель управления Центральным Сервером обновлений

Панель управления является рабочим местом Администратора *Сервера обновлений*. Администратором сервера является сотрудник компании-разработчика, отвечающий за поставку обновлений программного обеспечения конечным потребителям (*Клиентам*). К функциям Панели управления Центральным *Сервером обновлений* можно отнести:

1. Управление учетными записями пользователей Панели управления: создание, изменение, блокирование\разблокирование;
2. Управление учетными записями *Клиентов*: создание, изменение, блокирование\разблокирование;
3. Управление продуктами компании-разработчика: создание, изменение, назначение доступа *Клиентов* к *Продуктам*;
4. Управление файловыми репозиториями сервера: регистрация новых файлов, блокирование \ разблокирование существующих файлов;
5. Создание и публикация *Релизов*;
6. Просмотр статистики процесса получения *Клиентами* файлов в ходе выполнения обновления;
7. Управление и настройка системы автоматической рассылки уведомлений *Клиентам*: бюллетеней и анонсов к релизам.

Теперь кратко затронем вопросы инициализации файловых репозиториях *Сервера обновлений* и внутренних процессов создания *Релиза* для программного продукта. К процессу инициализации файловых репозиториях *Сервера обновлений* можно отнести:

1. При создании нового *Продукта* Администратором *Сервера обновлений* в таблицу **Products** базы данных добавляется запись с информацией об этом *Продукте*. Идентификатор новой записи (значение поля **id**) используется в качестве имени для создания подкаталогов в каталогах «*CI\CD directory*» и «*File Storage*» (см. рисунок 1);

2. DevOps компании-разработчика настраивает инструменты git-репозитория на доставку файлов *Продукта* в созданный подкаталог каталога «*CI/CD directory*» на *Сервере обновлений* и выполняет первичную доставку файлов;

3. Администратор *Сервера обновлений*, используя Панель управления в разделе Файлы Продукта получает список незарегистрированных файлов и выполняет их регистрацию в системе (записи об этом добавляются в таблицу ***FileRepository***):

- Указывает тип файла;
- Указывает путь к файлу в файловом хранилище «*File Storage*» *Сервера обновлений*;
- Указывает путь к файлу на стороне *Клиента* – там, где он должен будет располагаться в производственной среде;
- Путь к файлу в каталоге «*CI/CD directory*», идентификатор проекта и имя файла система должна прописывать автоматически.

Процесс создания *Релиза* на *Сервере обновлений* во многом схож с процессом первичной инициализации файловых репозитория, за исключением того, что *Продукт* уже создан – остается только управлять его файлами:

1. DevOps компании-разработчика выполняет доставку файлов из git-репозитория на *Сервер обновлений* в подкаталог *Продукта* в каталоге «*CI/CD directory*» с замещением файлов;

2. При наличии в новой поставке незарегистрированных файлов, Администратор *Сервера обновлений* выполняет регистрацию таких файлов, как было описано выше;

3. Администратор *Сервера обновлений*, используя Панель управления в разделе Релизы Продукта создает новый *Релиз*. При этом:

- в таблицу ***ProductReleases*** добавляется запись с информацией о *Релизе*;
- на основании данных таблицы ***FileVersion*** по предыдущим *Релизам*, осуществляется выбор файлов подкаталога *Продукта* в «*CI/CD directory*»;
- в подкаталоге *Продукта* в «*File Storage*» создается новый подкаталог для файлов создаваемого *Релиза* и в него копируются файлы, отобранные на предыдущем шаге;
- в таблицу ***FileVersion*** заносятся записи о каждом из новых файлов *Релиза*;

4. Администратор *Сервера обновлений* открывает доступ *Клиентам* на получение файлов созданного *Релиза*, выставив ему признак «опубликован».

Корпоративный сервер Клиента

Наиболее выгодным местом развертывания клиентского *Сервиса обновлений* будет корпоративный сервер *Клиента* – там же, где развернута поставляемая Вами информационная система.

Как видно по схеме, показанной на рисунке 1, *Сервис обновлений* (*Update Service*) состоит из двух частей:

1. Клиент Центрального *Сервера обновлений*. Эта часть *Сервиса обновлений* должна имплементировать следующий функционал:

- Подключение на Центральный *Сервер обновлений* и Авторизация на нем;
- Запрос на наличие новых версий (*Релизов*);
- Сохранение информации о получаемых *Релизах* в базу данных (см. таблица ***Releases*** на рисунке 3), загрузка списка файлов из полученного *Релиза* в базу данных (см. таблица ***ReleaseFiles*** на рисунке 3);
- Загрузка файлов по списку из полученного *Релиза* и сохранение их в файловое хранилище (*File Storage*);
- Отправка запроса с подтверждением о получении файла на Центральный *Сервер обновлений*.

2. Сервер для *Точек обновления*. Имплементирует практически тот же функционал API, что и Центральный *Сервер обновлений* (см. выше: «API-Методы для Клиентов Сервера обновлений»).

Кроме этого, на схеме, представленной на рисунке 1, видно, что *Сервис обновлений* работает совместно с базой данных. Вы можете завести как отдельную базу данных, так и использовать существующую, которую использует информационная система, добавив в нее все необходимые таблицы. Использование отдельной базы данных хорошо вписывается в концепцию микросервисной архитектуры [4], но потребует имплементации в *Сервисе обновлений* дополнительного API для взаимодействия с Информационной системой – в частности для реализации функционала контроля и мониторинга для Администратора Информационной системы. Вариант с использованием уже имеющейся базы данных Информационной системы можно рассматривать с точки зрения упрощения реализации функционала контроля и мониторинга для Администратора Информационной системы. На рисунке 3 представлена минимально необходимая структура базы данных для *Сервиса обновлений*.

Рисунок 3. «Схема базы данных сервиса обновлений на сервере корпоративной информационной системы»

Назначение таблиц в представленной схеме:

Releases – Хранит список информации о *Релизах*, полученных с Центрального Сервера обновлений. Структурно и функционально подобна одноименной таблице в базе данных Центрального Сервера обновлений;

Points – Справочник *Точек обновления* в корпоративной инфраструктуре Клиента. Подробнее о *Точках обновления* будет рассказано далее;

UploadStates – Справочник имен состояний загрузки файлов с Центрального Сервера обновлений на корпоративный сервер Клиента. Хранит текстовые названия состояний. Например: «в очереди», «загружается», «загружен», «ошибка»;

FileStates – Справочник имен состояний процесса получения файла *Точками обновления*. Хранит текстовые наименования состояний. Например: «Ожидает получения», «Выполняется обновление», «Файл получен», «Ошибка»;

ReleaseFiles – Список файлов *Релиза* загружаемых с Центрального Сервера обновлений и хранимых на корпоративном сервере Клиента для их последующей доставки на *Точки обновления*;

PointFilesUpdateState – Таблица, хранящая состояния передачи файлов, получаемые в ходе выполнения процесса обновления *Точек обновления*.

Как и в случае со схемой базы данных Центрального Сервера обновлений, схема базы данных, представленная на рисунке 3, состав ее таблиц и связей между ними, а также состав полей в ее таблицах, даны для общего понимания примерной структуры данных *Сервиса обновлений* для корпоративного сервера Клиента. В Вашей реализации *Сервиса обновлений* эта структура может быть другой – все зависит от задач, которые будет решать Ваш *Сервис обновлений*. Например, если Клиент использует несколько Ваших *Продуктов* одновременно, добавьте в эту схему данных таблицу **Products** аналогично схеме данных Центрального Сервера обновлений.

Помимо этого, Вам потребуется выполнить доработку *Frontend* Информационной системы, а именно той ее части, которая предназначена для Администратора Информационной системы:

1. В Центр уведомлений Информационной системы (если таковой имеется) добавить раздел с уведомлениями об обновлениях;
2. В системные настройки добавить опции, разрешающие\запрещающие автоматическую загрузку и установку всех обновлений;
3. Добавить Управление учетными записями *Точек обновления* (по необходимости);
4. Добавить просмотр истории *Релизов*, получаемых с Центрального Сервера обновлений, с возможностью ручного запуска загрузки;

5. Добавить просмотр статуса и хода выполнения процесса загрузки файлов *Релиза* с Центрального Сервера обновлений и процесса обновления *Точек обновления*;

Точка обновления

Под *Точками обновления* в контексте данной статьи я подразумеваю аппаратные устройства (серверы, рабочие станции, системы самообслуживания, планшетные компьютеры, телевизионные табло, системы сбора данных и многое другое) с установленным на них программным обеспечением, которое требуется время от времени обновлять. Эти устройства являются последним звеном в нашей цепочке распределенной системы автоматического обновления программного обеспечения. За получение обновления и выполнение процесса замены программного обеспечения на упомянутых устройствах отвечает *Клиентский модуль*, который устанавливается на каждой *Точке обновления* и работает как фоновый процесс, с установленной периодичностью запрашивая наличие обновлений на Корпоративном сервере Информационной системы *Клиента*. Функционально *Клиентский модуль Точки обновления* близок к Клиентской части *Сервиса обновлений* на стороне Корпоративного сервера *Клиента* – в его задачи входит:

1. Подключение на Корпоративный *Сервис обновлений* и Авторизация на нем;
2. Запрос на наличие новых версий (*Релизов*) для данной *Точки обновления*;
3. Загрузка файлов по списку из полученного *Релиза* во временную папку на *Точке обновления*;
4. Замена файлов из полученного *Релиза* на *Точке обновления*, перезапуск модулей, если потребуется;
5. Отправка запроса с подтверждением о получении файла на Корпоративный *Сервис обновлений*.

Заключение

На основе представленных в этой статье рекомендаций Вы сможете самостоятельно построить собственную распределенную систему обновления программного обеспечения. Вовсе не обязательно слепо следовать приведенным в статье примерам – они даны для общего понимания проблематики одной конкретно поставленной задачи. Представляйте приведенные примеры как набор блоков конструктора Lego, с помощью которых Вы сможете построить собственную систему под Ваши задачи.

В заключении хочу дать несколько рекомендаций касательно построения систем обновления программного обеспечения:

1. Все модули, отвечающие за прием обновлений в описанной инфраструктуре, должны уметь обновлять сами себя;
2. Добавьте в Вашу систему возможность передачи на *Точку обновления* сценариев, которые должны выполняться в процессе обновления: сценарии командной строки, SQL-скрипты и пр.;
3. Если Вы только начинаете разработку Информационной системы и системы автоматического обновления для нее, но еще не определились в каком технологическом стеке вести разработку, рекомендую мою статью [1].

Список литературы:

1. Плетнев А.В. Выбор технологического стека для it-проекта // Интернаука: электрон. научн. журн. 2021. № 36(212). URL: <https://internauka.org/journal/science/internauka/212> (дата обращения: 11.11.2021).
2. Карпович Е.Е. Жизненный цикл программного обеспечения. Лабораторный практикум. – М.: ИД «МИСиС», 2016.
3. Клиент – сервер / [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Клиент_—_сервер (дата обращения: 02.09.2021).
4. Ньюмен Сэм. Создание микросервисов. – СПб.: ИД «Питер», 2016.
5. Что такое CI/CD? Разбираемся с непрерывной интеграцией и непрерывной поставкой. Блог компании OTUS. / [Электронный ресурс]. URL: <https://habr.com/ru/company/otus/blog/515078/> (дата обращения: 12.11.2021).
6. REST / [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/REST> (дата обращения: 10.09.2021).

References:

1. Pletnev A.V. Choice of tech stack for IT project. // Internauka: scientific Internet-journal. 2021. № 36(212). URL: <https://internauka.org/journal/science/internauka/212> (date of access: 11.11.2021). (In Russian).
2. Karpovich E.E. The life cycle of software. Laboratory workshop. – M.: «MISiS», 2016. (In Russian).
3. Client – Server – online encyclopedia. URL: https://ru.wikipedia.org/wiki/Клиент_—_сервер (date of access: 02.09.2021). (In Russian).
4. Newman Sam. Building Microservices. – St.P.: «Piter», 2016. (In Russian).
5. What is CI/CD? Dealing with continuous integration and continuous delivery. OTUS company blog. Online blog. URL: <https://habr.com/ru/company/otus/blog/515078/> (date of access: 12.11.2021).
6. REST – online encyclopedia. URL: <https://ru.wikipedia.org/wiki/REST> (date of access: 10.09.2021). (In Russian).