**BC/NW 2021№ 1 (37):4.1**

**ИССЛЕДОВАНИЕ СПОСОБОВ СОКРАЩЕНИЯ ВРЕМЕНИ РАЗВЕРТЫВАНИЯ И ВОССТАНОВЛЕНИЯ ПОСЛЕ СБОЯ ВИРТУАЛЬНЫХ СТЕНДОВ В ОБЛАКЕ БАНКА**

**Абросимов Л.И., Петренко А.С.**

Введение

В настоящее время информационные системы банков используют приватные и публичные облака.

Приватное облако используется внутренними клиентами банков (разработчиками и службами эксплуатации банка, а также дочерних организаций) для размещения виртуальных машин и виртуальных стендов на их базе в целях организации разработки, тестирования и промышленной эксплуатации программного обеспечения.

Публичное облако используется для предоставления вычислительных услуг внешним клиентам, в том числе компаниям, входящим в экосистему банка.

Внешние и внутренние клиенты заказывают услуги и сервисы в приватном и публичном облаках с помощью графического интерфейса или через облачный программный интерфейс приложения - Application Programming Interface (API). В результате они хотят получать эти услуги и сервисы за как можно меньшее время в виде готовых виртуальных стендов с настроенным и запущенным программным обеспечением.

На первоначальном этапе множество таких услуг и сервисов подготавливается для клиентов в ручном режиме. После заказа облачной услуги или сервиса, заявка отправляется в автоматизированную систему учёта, где распределяется между исполнителями, которые берут эти заявки в работу, и подготавливают виртуальные стенды в соответствии с пожеланиями заказчика.

Ручная обработка и исполнение заявок приводят к длительному ожиданию клиентом заказанной услуги или сервиса из-за загруженности исполнителей и ограниченных человеческих возможностей.

Для обеспечения основных признаков облачных вычислений, таких как «самообслуживание по требованию», «быстрая эластичность и «масштабируемость», а также для повышения скорости развёртывания и восстановления виртуальных стендов, увеличения количества обрабатываемых заявок и снижения количество ошибок, возникающих в процессе развёртывания из-за человеческого фактора — предпринимаются действия, направленные на автоматизацию предоставления услуг и сервисов в облаках банка.

Наиболее эффективным является подход, при котором развёртывание виртуальных стендов осуществляется посредством взаимодействия с облачными API по заранее подготовленным сценариям, хранящимся в системе контроля версий. Такой подход называется Everything as Code (EaC). Он позволяет создавать виртуальные стенды в облаке с заданными параметрами в полностью автоматическом режиме.

1. Постановка задачи исследования

Облачные вычисления — парадигма для предоставления возможности сетевого доступа к масштабируемому и эластичному пулу разделяемых физических или виртуальных ресурсов, обеспечивающая самообслуживание и администрирование по требованию. [9]

Облачные вычисления, описанные в стандарте [2] обеспечивают:

— широкий сетевой доступ - характеристика, где физические и виртуальные ресурсы доступны по сети и доступ к ним осуществляется через стандартные механизмы, которые продвигают использование разнородных платформ клиентов.

— измеримое обслуживание - характеристика, где измеренное предоставление служб облачных вычислений таково, что его использование можно отслеживать, управлять им, отчитываться и по нему можно выставить счёт.

—мультиаренда - характеристика, где физические или виртуальные ресурсы распределены таким образом, при котором несколько арендаторов и их вычисления и данные изолированы друг от друга и недоступны друг другу.

—пул ресурсов - характеристика, при которой физические или виртуальные ресурсы службы облачных вычислений поставщика могут быть объединены для обслуживания одного или более потребителей службы облачных вычислений.

Виртуальные машины - фундаментальные компоненты виртуализации, имеют множество характеристик как у физических серверов. Как и реальный сервер, виртуальная машина поддерживает операционную систему и настроена с набором ресурсов, к которым приложения, запущенные на виртуальной машине, могут запрашивать доступ.

В отличие от физического сервера, на котором одновременно работает только одна операционная система и работает несколько чаще всего связанных приложений, виртуальные машины могут работать одновременно на одном физическом сервере, и эти виртуальные машины также могут запускать множество различных операционных систем, поддерживающих множество несвязанных приложений. Кроме того, в отличие от физического сервера, виртуальная машина на самом деле представляет собой не более чем набор файлов, которые описывают и составляют виртуальный сервер.

Основными файлами, составляющими виртуальную машину, являются файл конфигурации и файлы виртуального диска. Файл конфигурации описывает ресурсы, которые может использовать виртуальная машина: он перечисляет виртуальное оборудование, составляющее эту конкретную виртуальную машину.

Виртуальные устройства, с которыми работают виртуальные машины, являются стандартными устройствами, они одинаковы для каждой виртуальной машины, что делает их переносимыми на различные аппаратные платформы, решения виртуализации или решения поставщиков [22].

Рис. 1. Различия виртуальных машин и контейнеров

Контейнер - единый образ операционной системы, объединяющий набор приложений и их зависимостей, отделённых от хостовой машины. На одной хостовой машине под управлением может работать несколько таких контейнеров. Контейнеры можно разделить на два типа:

1) Уровень операционной системы: вся операционная система работает в изолированном пространстве на хост-машине, используя одно и то же ядро, что и хост-машина.

2) Уровень приложения: приложение или услуга и минимум процессов, необходимых для этого приложения, выполняются в изолированном пространстве на хост-машине.

Контейнеризация отличается от традиционных технологий виртуализации и предлагает множество преимуществ по сравнению с ними:

– контейнеры легковесней по сравнению с традиционными виртуальными машинами;

– в отличие от контейнеров, виртуальным машинам требуются уровни эмуляции (программные или аппаратные), которые потребляют больше ресурсов и добавляют дополнительные накладные расходы;

– контейнеры уровня ОС разделяют ресурсы с хостовой машиной с помощью пространств имён (namespaces) и контрольных групп (cgroups);

– из-за лёгкости контейнеров на каждом хосте может быть запущено больше контейнеров, чем виртуальных машин;

– запуск контейнера происходит почти мгновенно по сравнению с более медленным процессом загрузки виртуальных машин;

– контейнеры легко переносимы и могут надёжно регенерировать системную среду с необходимыми программными пакетами, независимо от базовой операционной системы [15].

Контейнеры не являются виртуальными машинами, но они достаточно близки по функциональности и широко используются в разработке приложений и бизнес-вычислениях. Как и виртуальные машины, контейнеры предлагают независимый от платформы пакет для объединения, доставки и развёртывания приложений. В отличие от виртуальных машин, контейнеры абстрагируются на уровне операционной системы, а не на уровне оборудования. Это означает, что контейнеры могут объединять несколько рабочих нагрузок в одном приёмнике, обычно поддерживая одно приложение.

Контейнеры обеспечивают лучшую переносимость и меньшие накладные расходы вычислительных ресурсов, чем виртуальные машины. Но у них есть некоторые ограничения.

Модель контейнера, поскольку она абстрагируется на уровне операционной системы, имеет ограничение, заключающееся в том, что все рабочие нагрузки в контейнере должны запускать одну и ту же операционную систему или ядро операционной системы.

Изоляция рабочих нагрузок в контейнерах не такая надёжная, как у виртуальных машин, запущенных на гипервизорах. Во многих случаях контейнеры могут предложить более быстрое развёртывание и меньшие накладные расходы, чем виртуальные машины. Благодаря этим свойствам контейнеры широко используются для для создания окружений разработки [22].

Цели автоматизированного управления сервером

Использование инфраструктуры как кода для управления конфигурацией сервера должно привести к следующему [18]:

– Новый сервер можно полностью подготовить по запросу за несколько минут.

– Новый сервер можно полностью подготовить без участия человека, например, в ответ на события.

– Когда изменение конфигурации сервера определено, оно применяется к серверам без участия человека.

– Каждое изменение применяется ко всем серверам, к которым оно относится, и отражается на всех новых серверах, подготовленных после того, как изменение было сделано.

– Процессы предоставления и применения изменений к серверам повторяемы, согласованы, самодокументированны и прозрачны.

– Легко и безопасно вносить изменения в процессы, используемые для подготовки серверов, и изменять их конфигурацию.

– Автоматические тесты запускаются каждый раз, когда вносятся изменения в определение конфигурации сервера и в любой процесс, участвующий в настройке и изменении серверов.

– Изменения в конфигурации и изменения в процессах, которые выполняют задачи в инфраструктуре, версионируются и применяются к различным средам, чтобы поддерживать стратегии контролируемого тестирования и поэтапного выпуска.

Эффективность

Эффективность можно измерить отношением полезной работы, выполняемой машиной или процессом, к общему количеству затраченной на это энергии. Когда дело доходит до развёртывания приложений и управления ими, многие из доступных инструментов и процессов (например, сценарии BASH, обновления APT или императивное управление конфигурацией) несколько неэффективны. [11].

Стоимость эксплуатации сервера зависит от энергопотребления, требований к охлаждению, площади центра обработки данных и чистой вычислительной мощности. После того, как сервер поставлен в стойку и включён, счётчик буквально начинает работать. Любое время простоя процессора - это потеря денег.

Kubernetes предоставляет инструменты, которые автоматизируют распределение приложений по кластеру машин, обеспечивая более высокий уровень использования, чем это возможно с традиционными инструментами

Дальнейшее повышение эффективности связано с тем, что тестовая среда разработчика может быть быстро и дёшево создана как набор контейнеров, работающих в личном представлении общего кластера Kubernetes (с помощью функции, называемой пространствами имён). Уменьшение общего количества используемых машин, в свою очередь, повышает эффективность каждой системы: поскольку используется больше ресурсов (ЦП, ОЗУ и т.д.) на каждой отдельной машине. Общая стоимость каждого контейнера становится намного ниже.

Приватное и публичное облака банка предоставляют все, перечисленные в стандарте [2] категории служб облачных вычислений для использования внешним и внутренним клиентам. При этом не все сервисы, предоставляемые облаками банка. соответствуют в полной мере основным признакам облачных вычислений, описанных в стандарте, из-за недостаточной автоматизации развёртывания сервисов. В первую очередь это оказывает влияние на реализацию возможностей самообслуживания по требованию, а также снижает эластичность и масштабируемость предоставляемых услуг.

Услуги предоставляются как правило с помощью традиционных средств виртуализации на базе гипервизоров первого рода, таких как VMware vSphere и OpenStack. В то время, как средства виртуализации уровня ОС с использованием оркестраторов контейнеров, обещают более высокую производительность, в том чиле более высокую скорость развёртывания стендов, высокую надёжность, автоматическое восстановление в случае сбоев и многие другие вышеописанные преимущества.

Новые подходы к управлению конфигурациями в облачных средах в сочетании с традиционными ВМ и контейнерами также способны повысить надёжность, скорость развёртывания и восстановления после сбоев. Особенно в случае использования декларативного подхода к управлению конфигурациями.

Вышеописанные рекомендации описывают общие принципы и лучшие практики развёртывания ПО в облачных средах, не предоставляя готовых решений для развёртывания стендов в облаках банка.

Поэтому необходимо:

• исследовать применимость рассмотренных принципов, а также способы применений их применения в облачной инфраструктуре банка.

• разработать, опробовать и сравнить методики развёртывания в облаках банка.

• разработать и протестировать программное обеспечение для развёртывания виртуальных стендов в условиях облачной инфраструктуры банка.

2. Анализ методов и средств тестирования

Объект исследования

Исследование может проводиться на примере различных автоматизированных систем, эксплуатируемых в виртуальных средах, к которым предъявляются требования по времени развёртывания.

В представленной работе объектом исследования является программное обеспечение Функциональная подсистема Репозиторий образов контейнеров (ФП РОК), которая работает в составе автоматизированной системы/

Автоматизированная система Фонд программ и документации (АС ФПД).

ФП РОК предназначена для хранения, передачи и обработки образов Docker- и OCI- контейнеров, используемых в банке для различных целей.

Также система осуществляет проксирование, кэширование и обеспечивает безопасный доступ к образам контейнеров, расположенным во внешних репозиториях в интернете. В целях обеспечения целостности хранимой информации, в системе ведётся подсчёт контрольных сумм содержимого образов контейнеров. Система позволяет производить аутентификацию и авторизацию пользователей при работе с образами контейнеров.

В ближайшей перспективе к функциональности ФП РОК будет добавлена функция хранения Helm-чартов с подписанием артефактов ЭЦП по ГОСТ.

Архитектура ФП РОК

Функциональная подсистема Репозиторий образов контейнеров (ФП РОК)представляет собой Nexus Repository Manager OSS 3.x, установленный за реверс-прокси на базе сервера Nginx. ФП РОК, подключается к службе каталогов Microsoft Active Directory по протоколу LDAP для осуществления аутентификации пользователей.

Служебные данные Nexus Repository Manager хранятся во внешней базе данных, подключаемой в облаке как сервис. Пользовательские данные (образы docker-контейнеров) хранятся на внешнем хранилище, подключаемом в облаке как сервис.

Архитектура ФП РОК представлена на рисунке 2

Рис. 2. Архитектура ФП РОК

По умолчанию в Nexus Repository Manager настраиваются следующие docker-репозитории:

– hosted-репозиторий snapshot;

– hosted-репозиторий release;

– прокси-репозиторий registry.access.redhat.com;

– прокси-репозиторий registry.redhat.io.

Для проксирования внешних репозиториев RedHat Container Catalog используется подключение по протоколу HTTPS через шлюз прикладного уровня (отсутствует на тестовом стенде), который проверяет передаваемые данные на соответствие требованиям информационной безопасности.

Доступ к каждому репозиторию в Nexus Repository Manager осуществляется с помощью выделенного TCP-порта.

Nginx осуществляет переадресацию внешних запросов к виртуальным директориям в ФП РОК на соответствующие TCP-порты репозиториев в Nexus Repository Manager OSS.

Рис. 3. Упрощённая в целях тестирования архитектура ФП РОК

В целях тестирования конфигурация ФП РОК была упрощена:

– вместо внешнего сервера LDAP используются внутренние учётные записи Nexus;

– вместо внешней БД используется Orient DB встроенная в Nexus;

– вместо внешнего хранилища, используется локальный диск виртуальной машины или ноды Kubernetes.

Упрощённая архитектура ФП РОК представлена на рисунке 3 .

Средства автоматизации

Для развёртывания ФП РОК выбраны следующие средства автоматизации в соответствии с требованиями инструментального стандарта DevOps банка:

– сервер автоматизации Jenkins со скриптовым языком Groovy DSL, позволяющий вести запись журналов выполнения операций снабжаемых

метками времени, используемых для мониторинга;

– система управления конфигурациями Ansible;

– утилита Terraform для настройки виртуальных стендов;

– пакетный менеджер Helm для развёртывания релизов в среде Kubernetes

или OpenShift.

Состав тестового стенда

Тестовый стенд состоит из следующих виртуальным машин выделенных для:

– сервера Jenkins;

– сервера Nexus Repository Manager OSS 3;

– сервер Nginx.

Описание вычислительных ресурсов виртуальных машин тестового стенда представлено в таблице 1 .

Схема создания виртуальных машин в облаке банка и подов в Kubernetes представлена на рисунке 4.

Рис. 4. Схема создания виртуальных машин в облаке и подов в Kubernetes

Описание ВМ с сервером автоматизации Jenkins

Виртуальная машина для сервера автоматизации Jenkins создаётся и настраивается один раз вручную, после чего используется для запуска скриптов и последующего мониторинга их выполнения.

Используется Jenkins версии 2.263 с плагинами:

– Ansible plugin версии 1.1;

– Credentials plugin версии 2.3.14;

– Git plugin версии 4.5.0;

– Pipeline plugin версии 2.6.

На виртуальную машину также установлены:

– распределенная система хранения версий Git версии 2.17;

– система управления конфигурациями Anisble версии 2.9 с коллекцией

модулей community.general;

– утилита Terraform версии 0.14;

– демон Docker версии 19.03;

– пакетный менеджер Helm версии 3.4;

– сервер etcd версии 3.4.

Утилизация ресурсов

Конфигурация вычислительных ресурсов подбирается таким образом, чтобы выдерживать высокие нагрузки, связанные с бизнес-функционалом ФП РОК. Чтобы исключить влияние нехватки памяти или производительности процессоров во время проведения ручных или автоматизированных тестов развёртывания, в процессе ручного развёртывания виртуальных стендов, производится замер утилизации ресурсов с помощью утилиты htop.

Утилизация ресурсов в процессе развёртывания в пике не превышает 50%.

Для ПЗУ используются стандартные ресурсы облака на базе HDD. Производительность подключаемой дисковой подсистемы постоянна во всех случаях для всех виртуальных машин и контейнеров.

Средства мониторинга

Для проведения каждого исследования записывается автоматизированный скрипт-конвейер (pipeline) на Groovy DSL для запуска в качестве задания Jenkins.

Конвейер состоит из этапов (stage), например: «Установка зависимостей», «Создание виртуальных машин», «Установка Nexus» и т.д.

Этапы состоят из шагов (step), например, этап «Создание виртуальных

машин» состоит из шагов: «Создать виртуальные машины» и «Ожидать запуск виртуальных машин».

На каждом шаге с помощью Jenkins Ansible plugin запускается Ansibleплейбук (playbook) для настройки виртуальных машин или приложений.

На основании записанных журналов определяется время, затраченное на выполнение шагов и этапов, как разность между окончанием и началом выполнения каждого шага или этапа.

В процессе отладки скриптов осуществляются тестовые развёртывания стендов с целью подтверждения корректности и оптимальности работы скриптов. После чего проводится повторный запуск скриптов развёртывания, проводится анализ результатов, и делается вывод о достижении целей исследования.

3. Методика тестирования с использованием API облака

Настройки Terraform для подготовки виртуальных машин

Взаимодействие с API облака, для создания виртуальных машин, осуществляется с помощью инструмента Terraform с использованием соответствующего плагина для взаимодействия с API облака банка.

Описание конфигураций виртуальных машин выполнено на языке Terraform (Terraform Language) [30], размещено в блоках ресурсов (Resource Blocks),и находится в файлах-шаблонах nexus.tf.j2 и nginx.tf.j2 в директории ter\_templates.

Шаблоны с конфигурациями ресурсов выполнены на языке шаблонов jinja2 [13], чтобы можно было динамически изменять источник для создания дисков виртуальных машин.

Если виртуальные машины создаются и настраиваются с нуля, то в шаблон подставляется идентификатор образа диска с чистой операционной системой для последующей настройки.

Если виртуальные машины создаются на основе заранее подготовленных снимков, тогда в шаблон подставляется идентификатор снимка предварительно настроенного виртуального диска. Данные о состоянии виртуальных машин сохраняются в виде плана Terraform в хранилище etcd5, запущенном на сервере Jenkins.

Настройки для сохранения плана содержаться в файле-шаблоне backend.tf.j2. Информация об используемом провайдере и бэкэнде (backend) для сохранения данных о состоянии виртуальных машин содержится в файлешаблоне providers.tf.j2.

В файле-шаблоне snapshot.tf.j2 содержится шаблон ресурса для создания снимка диска виртуальной машины. Имя снимка и идентификатор диска подставляются динамически во время выполнения выполнения сценария по подготовке снимка.

Файл-шаблон hosts.yml.j2 предназначен для сохранения ip-адресов и идентификаторов дисков виртуальных машин, полученных в результате применения плана Terraform, и создания inventory-файла в yaml формате для последующего использования в Ansible сценариях.

Описание Ansible-плейбуков

Плейбуки (playbook) Ansible содержать описание желаемого состояния

настраиваемой системы в YAML-формате10

Плейбук CreateVMs.yml содержит описание задач для создания виртуальных машин.

Предварительные задачи (pre\_tasks) предназначены для установки инструмента Terraform из репозиториев вендора Hashicorp.

10https://docs.ansible.com/ansible/latest/user\_guide/playbooks\_intro.html

Основные задачи (tasks) предназначены для:

– загрузки списка снимков дисков в облаке (если таковые существуют);

– подготовки шаблонов Terraform;

– инициализацию и подготовку плана Terraform;

– применение плана Terraform.

Заключительные задачи (post\_tasks) сохраняют ip-адреса и идентификаторы дисков в Ansible inventory файл для последующего обращения к созданным виртуальным машинам.Плейбук WaitVMs.yml содержит основные задачи, осуществляющие ожидание запуска виртуальных машин, подготовленных с помощью Terraform.

Заключительные задачи добавляют в информацию из шаблонов Nexus и Nginx в Ansible inventory.

Плейбук NexusOSS3.yml содержит задачи и роли для установки и настройки Nexus Repository Manager 3.

Предварительные задачи осуществляют установку зависимостей (openjdk);

создание группы, пользователя, проверку их идентификаторов; загрузку переменных, необходимых для настройки S3.

После чего запускаются Ansible-роли, осуществляющие настройку S3

и Nexus.

Плейбук Nginx.yml состоит из двух плеев (play) и содержит задачи для

установки и настройки nginx.

Плей «Install nginx» содержит задачи по установке nginx с помощью

пакетного менеджера операционной системы.

Плей «Configure nginx» содержит задачи для копирования ключа и сертификата SSL, подготовки шаблона основного конфигурационного файла

nginx.conf и перезапуска сервера nginx с новыми настройками.

Плейбук NginxTests.yml содержит задачи для проверки доступности сервера Nginx. В задачах осуществляется циклический опрос TCP-портов, заданных в файле vhosts.yml в переменной listen, для хоста nginx из Ansible

inventory файла. Если хост nginx не ответит на заданных портах в течении

определённого промежутка времени, то выполнение плейбука будет прервано с сообщением об ошибке.

Плейбук NexusTests.yml содержит задачи для проверки доступности dockerрепозиториев. В задачах осуществляется циклический опрос TCP-портов, заданных в файле nexus\_templates\main.yml в переменных http\_port, для хоста nexus из Ansible inventory файла. Если хост nexus не ответит на заданных портах в течении определённого промежутка времени, то выполнение плейбука будет прервано с сообщением об ошибке.

Плейбук FprokTests.yml содержит предварительные задачи для установки docker-демона и необходимых зависимостей из репозитория поставщика операционной системы.

В задачах осуществляется проверка функциональности ФП РОК:

– аутентификация и авторизация в ФП РОК;

– загрузка образа rhel-minimal из прокси-репозитория

registry.access.redhat.com;

– выгрузка образа rhel-minimal в репозиторий snapshot;

– выгрузка образа rhel-minimal в репозиторий release.

Если хотя бы одна из операций завершится с ошибкой, то выполнение плейбука будет прервано с сообщением об ошибке.

Плейбук TakeSnapshots.yml состоит из двух плеев.

Плей «Stop NexusOSS3 correctly» отвечает за остановку демона nexus, чтобы все операции чтения-записи в Nexus Repository Manager были завершены корректно перед получением снимка диска.

Плей «Take snapshots» в цикле вызывает набор задач из task\snapshots.yml последовательно для диска каждого хоста, описанного в Ansible inventory файле.

В файле snapshots.yml содержаться задачи для:

– подготовки шаблона snapshots.tf.j

– подготовки плана terraform;

– применения плана terraform.

Плейбук LookForSnapshots содержит задачи для получения списка снимков дисков из облака банка, и создания файла-списка в формате json.

Плейбук CheckSnapshot.yml содержит задачи для чтения файла-списка

снимков дисков, и проверки каждого элемента списка на соответствие заданному хосту и идентификатору диска хоста.

Плейбук RemoveVMs.yml содержит задачи для удаления виртуальных машин, описанных в плане terraform, с целью освобождения ресурсов в облаке банка или подготовки тестового стенда к повторному развёртыванию ФП РОК.

4 Алгоритмы проведения тестирования с использованием API облака

Развёртывание и настройка виртуальных машин с нуля с последовательным выполнением шагов

Автоматизированное развёртывание ФП РОК с использованием API облака банка с помощью инструмента Terraform состоит из следующих шагов:

1) Установка необходимых ansible-ролей из файла requirements.yml.

2) Запуск плейбука CreateVMs.yml для создания виртуальных машин согласно настройкам заданным в шаблонах terraform.

3) Запуск плейбука WaitVMs.yml для проверяющего готовность виртуальных машин, созданных на предыдущем шаге.

4) Запуск плейбука NexusOSS3.yml для установки и настройки Nexus Repository Manager 3 в соответствии с настройками, заданными в шаблонах nexus.

5) Запуск плейбука Nginx.yml для установки и настройки сервера Nginx в

соответствии с настройками, заданными в шаблонах nginx.

6) Запуск плейбука NginxTests.yml для проверки работоспособности и правильности выполненных настроек севера Nginx.

7) Запуск плейбука NexusTests.yml для проверки работоспособности и правильности выполненных настроек Nexus Repository Manager 3.

8) Запуск плейбука FprokTests.yml для проверки работоспособности и правильности настройки ФП РОК.

9) Последовательный запуск плейбуков TakeSnapshots.yml, LookForSnapshots.yml,

CheckSnapshots.yml, если был задан соответствующий параметр задачи Jenkins, для получения снимков дисков настроенных виртуальных машин.

10) Запуск плейбука RemoveVMs.yml, если был задан соответствующий параметр задачи Jenkins, для удаления виртуальных машин и освобождения вычислительных ресурсов в облаке банка.

Блок-схема алгоритма развёртывания и настройки виртуальных машин с нуля с последовательным выполнением шагов представлена на рисунке 5.

Алгоритм реализован на Groovy DSL в AutodeployVmsFromScratch.Jenkinsfile (помечен тегом v1.0.0 в репозитории).

Рис. 5. Блок-схема алгоритма развёртывания и настройки виртуальных машин с нуля с последовательным выполнением шагов

Развёртывание и настройка виртуальных машин с нуля с параллельным выполнением ключевых шагов

Автоматизированное развёртывание ФП РОК с использованием API облака банка с помощью инструмента Terraform, с одновременным выполнением некоторых шагов для сокращения общего времени развёртывания, состоит из следующих шагов:

1) Установка необходимых ansible-ролей из файла requirements.yml.

2) Запуск плейбука CreateVMs.yml для создания виртуальных машин согласно настройкам заданным в шаблонах terraform

3) Запуск плейбука WaitVMs.yml для проверяющего готовность виртуальных машин, созданных на предыдущем шаге.

4) Одновременный запуск плейбуков:

– NexusOSS3.yml для установки и настройки Nexus Repository Manager в соответствии с настройками, заданными в шаблонах nexus;

– и Nginx.yml для установки и настройки сервера Nginx в соответствии с настройками, заданными в шаблонах nginx

5) Запуск плейбука NginxTests.yml для проверки работоспособности и правильности выполненных настроек севера Nginx.

6) Запуск плейбука NexusTests.yml для проверки работоспособности и правильности выполненных настроек Nexus Repository Manager 3.

7) Запуск плейбука FprokTests.yml для проверки работоспособности и правильности настройки ФП РОК.

8) Последовательный запуск плейбуков TakeSnapshots.yml, LookForSnapshots.yml, CheckSnapshots.yml, если был задан соответствующий параметр задачи Jenkins, для получения снимков дисков настроенных виртуальных машин

9) Запуск плейбука RemoveVMs.yml, если был задан соответствующий па раметр задачи Jenkins, для удаления виртуальных машин и освобождения вычислительных ресурсов в облаке банка. Блок-схема алгоритма развёртывания и настройки виртуальных машин с нуля с параллельным выполнением ключевых шагов представлена на рисунке 6. Алгоритм реализован на Groovy DSL в AutodeployVmsFromScratch.Jenkinsfile в ветке (помечен тегом v1.1.0 в репозитории).

Рис. 6. Блок-схема алгоритма развёртывания и настройки виртуальных машин с нуля с параллельным выполнением ключевых шагов

Автоматизированное развёртывание с использованием снимков дисков виртуальных машин

Автоматизированное развёртывание ФП РОК с использованием API облака банка с помощью инструмента Terraform и предварительно подготовленных снимков дисков виртуальных машин состоит из следующих шагов:

1) Установка необходимых ansible-ролей из файла requirements.yml.

2) Запуск плейбука LookForSnapshot.yml для поиска снимков дисков подготовленных виртуальных машин для Nexus и Nginx.

3) Запуск плейбука CreateVMs.yml для создания виртуальных машин из снимков, согласно настройкам заданным в шаблонах terraform.

4) Запуск плейбука WaitVMs.yml для проверяющего готовность виртуальных машин, созданных на предыдущем шаге.

5) Запуск плейбука NexusOSS3.yml для перенастройки Nexus Repository Manager 3 без переустановки приложения в соответствии с настройками, заданными в шаблонах nexus, если был задан соответствующий параметр задачи Jenkins.

6) Запуск плейбука Nginx.yml для перенастройки сервера Nginx в соответствии с настройками, заданными в шаблонах nginx т.к. ip-адрес и номера портов сервера Nexus могли измениться.

7) Запуск плейбука NginxTests.yml для проверки работоспособности и пра вильности выполненных настроек севера Nginx.

8) Запуск плейбука NexusTests.yml для проверки работоспособности и правильности выполненных настроек Nexus Repository Manager.

9) Запуск плейбука FprokTests.yml для проверки работоспособности и правильности настройки ФП РОК.

10) Запуск плейбука RemoveVMs.yml, если был задан соответствующий параметр задачи Jenkins, для удаления виртуальных машин и освобождения вычислительных ресурсов в облаке банка.

Блок-схема алгоритма развёртывания и настройки виртуальных машин с использованием снимков дисков виртуальных машин на рисунке 7.

Алгоритм реализован на Groovy DSL в AutodeployVmsFromSnapshot.Jenkinsfile. 2

Рис. 7. Блок-схема алгоритма развёртывания и настройки виртуальных машин с использованием снимков дисков виртуальных машин

5. Методика тестирования с использованием Kubernetes API и пакетного менеджера Helm

Данный вид тестирования производится с использованием виртуализации уровня ОС, а именно: технологий docker- и OCI- контейнеров.

Развёртывание ФП РОК осуществляется с помощью Helm-чартов [10] в проект на кластере Kubernetes, запущенном в облаке банка. Шаблоны объектов Kubernetes в Helm-чартах описываются на языке Gotemplate11

Для развёртывания ФП РОК с помощью docker-контейнеров не требуется предварительная подготовка образов контейнеров, аналогичная подготовке снимков дисков виртуальных машин, т.к. производители Nexus Repository Manager и Nginx предлагают готовые образы контейнеров с предустановленным ПО с настройками по-умолчанию, которые нужно только сконфигурировать в соответствии с требованиями банка и потребностями пользователей.

Количество настраиваемых параметров также было снижено до минимально необходимого относительно ansible-роли nexus3-oss12: оставлено конфигурирование Docker-репозиториев, внутренних учётных записей Nexus и параметров подключения к серверам LDAP (не используется при тестовом развёртывании).

Установка и настройка Nexus и Nginx происходит одновременно. Проверка готовности Nexus и Nginx производится встроенными средствами Kubernetes - readiness probes13. [14]

Описание Ansible-плейбуков

Плейбук Helm.yml Плейбук Helm.yml содержит задачи по установке helm с помощью пакетного менеджера операционной системы.

Плейбук Fprok.yml Плейбук Fprok.yml содержит предзадачи которые:

– устанавливают необходимы зависимости для Ansible-модуля community.kubernetes.k8s;

– подготавливают файл kubeconfig с необходимыми настройками для доступа Helm к проекту в кластере Kubernetes;

– подготавливают объекты типа Secret и устанавливают их в соответствующую проектную область в кластере Kubernetes.

Задачу, которая запускает установку релиза fprok из Helm-чарта fprok с помощью Ansible-модуля community.kubernetes.helm, который в загружает значения параметров Helm-чарта из файлов, расположенных в директории fprok\_configs.

Плейбук FprokTests.yml Плейбук FprokTests.yml содержит предварительные задачи для получения IP-адреса сервиса nginx типа LoadBalancer, а также для установки dockerдемона и необходимых зависимостей из репозитория поставщика операционной системы.

В задачах осуществляется проверка функциональности ФП РОК:

– аутентификация и авторизация в ФП РОК;

– загрузка образа rhel-minimal из прокси-репозитория registry.access.redhat.com;

– выгрузка образа rhel-minimal в репозиторий snapshot;

– выгрузка образа rhel-minimal в репозиторий release. Если хотя бы одна из операций завершится с ошибкой, то выполнение плейбука будет прервано с сообщением об ошибке.

Плейбук CleanupNamespace.yml Плейбук CleanupNamespace.yml содержит задачи для очистки проектной области:

– подготовка файла kubeconfig с необходимыми настройками для доступа Helm к проекту в кластере Kubernetes;

– деинсталляция Helm-чарта fprok;

– удаление всех объектов типа Secret, Configmap, Ingress, IngressClass, Deployment, Service, а также других типов объектов.

Алгоритм проведения тестирования с использованием Kubernetes API и пакетного менеджера Helm

Автоматизированное развёртывание ФП РОК с использованием Kubernetes API и пакетного менеджера Helm состоит из следующих шагов:

1) Запуск плейбука CleanupNamespace.yml, если был задан соответствующий параметр задачи Jenkins, для предварительной очистки проектной области.

2) Установка необходимых модулей Ansible указанных в файле requirements.yml.

3) Запуск плейбука Helm.yml для установки пакетного менеджера Helm на сервер Jenkins.

4) Запуск плейбука Fprok.yml для установки релиза fprok с помощью пакетного менеджера Helm с заданными параметрами.

5) Запуск плейбука FprokTests.yml для проверки работоспособности и правильности настройки ФП РОК.

6) Запуск плейбука CleanupNamespace.yml, если был задан соответствующий параметр задачи Jenkins, для очистки проектной области и освобождения вычислительных ресурсов в облаке банка.

Блок-схема алгоритма развёртывания и настройки виртуальных машин с использованием снимков дисков виртуальных машин на рисунке 8.

Алгоритм реализован на Groovy DSL в AutodeployInKube.Jenkinsfile.

Рис.8. Блок-схема алгоритма развёртывания и настройки с использованием Kubernetes API и пакетного менеджера Helm 86

Алгоритм реализован на Groovy DSL в AutodeployInKube.Jenkinsfile.

6. Исследование способов и методов развертывания ФП РОК

Разработка скриптов тестирования

Для проведения тестирования разрабатываются скрипты тестирования. Скрипты тестирования разрабатываются с использованием ПО Jenkins 2.263, Anisble 2.9, Terraform 0.14, Docker 19.03, Helm 3.4, Kubernetes 1.19.

Были выбраны наиболее часто встречающиеся сценарии развёртывания, используемые приватном и публичном облаках банка.

В процессе тестирования на сервера Jenkins непрерывно ведётся запись журналов выполнения всех операций, для получения данных о процессе тестирования. В журнале, с точностью до 1 секунды, фиксируется время запуска и завершения работы каждого Ansible плейбука, время начала и завершения выполнения каждой задачи в плейбуках, время начала и завершения выполнения других операций, успешность завершения задач.

Все сценарии тестирования запускаются по 10 раз с одинаковыми параметрами в одинаковых условиях для проверки повторяемости и исключения воздействия случайных внешних факторов.

Для каждой интересующей нас операции по журналам определяется точный интервал (время начала и завершения операции), вычисляется среднее время для всех запусков сценария, строятся диаграммы на основании полученных данных для каждого сценария, полученные данные сравниваются между собой и с нормативами для ручного развёртывания, и делаются выводы. Критерии успешности тестирования

Тест считается выполненным успешно в случае, если перед проведением тестирования были очищены проектные области, установлены все необходимые инструменты и их зависимости, загружены Ansible-модули и роли.

В результате проведения теста записаны журналы выполнения операций, сохранены результаты выполнения каждой операции, каждого плейбука и каждой задачи.

А также выполняются следующие условия:

– Nexus прослушивает заданные TCP порты;

– Nginx прослушивает заданные TCP порты;

– образ контейнера rhel-minimal скачивается из прокси-репозитория registry.access.redhat.com на сервер Jenkins;

– доступна загрузка образа в hosted-репозиторий snapshot;

– доступна загрузка образа в hosted-репозиторий release;

Результаты тестов развёртывания

Ключевыми этапами считаются этапы связанные непосредственно с установкой и настройкой программного обеспечения Nexus Repository Manager и Nginx.

Результаты автоматизированного развёртывания с использованием API

Результаты автоматизированного развёртывания с использованием API представлены в таблице 2 и на рисунке 9

не производился. Время на очистку проектной области после развёртывания - не учитывается

. Соотношение времени ключевых этапов и этапов тестирования в процессе развёртывания представлено на рисунке 10.

Средняя длительность этапов развёртывания в порядке убывания:

1) установка и настройка Nexus OSS 3 - 7 минут 27 секунд;

2) одновременное создание виртуальных машин для Nexus и Nginx - 1 минута 27 секунд;

3) установка и настройка Nginx - 34 секунды;

4) тестирование ФП РОК - 26 секунд;

5) прочие этапы - 19 секунд.

Рис. 9. Результаты автоматизированного развёртывания с использованием API

Рис. 10. Соотношение времени ключевые этапы и этапов тестирования автоматизированного развёртывания с использованием API

Среднее время автоматизированного развёртывания с использованием API составило 10 минут 13 секунд. Из них:

– среднее время выполнения ключевых этапов 9 минут 28 секунд;

– среднее время тестирования 39 секунд.

Исходя из выше представленной информации, в первую очередь необходимо поработать над оптимизацией установки Nexus и Nginx, а также созданием виртуальных машин

Результаты автоматизированного развёртывания с использованием API с параллельным выполнением ключевых шагов

В целях сокращения времени развёртывания было использовано параллельное (одновременное) развёртывание Nexus и Nginx на соответствующие виртуальные машины с использованием Jenkins pipeline parallel stages14 [12].

Результаты автоматизированного развёртывания с использованием API с параллельным выполнением шагов по установке Nexus и Nginx представлены в таблице 3 и на рисунке 11.

После развёртывания захват снимков также не производился, и время, затраченное на очистку проектной области, не учитывается.

Соотношение времени ключевых этапов и этапов тестирования в процессе развёртывания представлено на рисунке 12.

Рис. 11. Результаты автоматизированного развёртывания с использованием API с параллельным выполнением шагов

Средняя длительность этапов развёртывания в порядке убывания:

1) одновременная установка и настройка Nexus и Nginx - 7 минут 32 секунд;

2) одновременное создание виртуальных машин для Nexus и Nginx - 1 минута 38 секунд;

3) тестирование ФП РОК - 43 секунды;

4) прочие этапы - 6 секунд.

Рис. 12. Соотношение времени ключевые этапы и этапов тестирования автоматизированного развёртывания с использованием API с параллельным

выполнением шагов

Среднее время автоматизированного развёртывания с использованием

API с параллельным выполнением составило 9 минут 58 секунд. Из них:

– среднее время выполнения ключевых этапов 9 минут 10 секунд;

– среднее время тестирования 43 секунды.

Параллельное выполнение шагов по установке и настройке Nexus и Nginx дало 14 секундный выигрыш в общем времени развёртывания относительно последовательного выполнения шагов, снизив время развёртывания всего лишь 2,4%. При этом среднее время выполнения ключевых этапов снизилось на 18 секунд (3,4%), а среднее время тестирования выросло на 4 секунды (10,3%), т.к. между запуском демона Nexus Repository Manager и началом тестирования не было этапа установки Nginx, который Nexus использовал для загрузки компонентов. Соответственно Nexus Repository Manager дольше обрабатывал тестовые запросы.

Параллельное выполнение шагов по установке Nexus и Nginx не дало существенного снижения времени развёртывания, т.к. основное время затрачивается на установку Nexus Repository Manager, который является ограничивающим фактором для ускорения развёртывания

Этот способ подойдёт для развёртывания систем, где несколько компонентов устанавливаются и настраиваются за примерно одинаковое время.

Результаты автоматизированного развёртывания с использованием снимков дисков виртуальных машин

Чтобы ускорить развёртывание ФП РОК, было принято решение разделить его на две стадии:

1) однократная предварительная подготовка снимков дисков виртуальных машин с установленными приложениями Nexus и Nginx;

2) последующее многократное развёртывание приложений из предварительно подготовленных снимков.

Предварительная подготовка снимков дисков виртуальных машин осуществляется в результате развёртывания согласно одному из алгоритмов,

Перенастройка Nginx необходима, потому что меняется IP-адрес виртуальной машины с Nexus Repository Manager процессе развёртывания с использованием снимков представлено на рисунке 12.

Рис. 13. Результаты автоматизированного развёртывания с использованием снимков дисков виртуальных машин.

Результаты автоматизированного развёртывания с использованием предварительно подготовленных снимков дисков виртуальных машин представлены в таблице 4 и на рисунке 13. Соотношение времени ключевых этапов и этапов тестирования в процессе развёртывания с использованием снимков представлено на рисунке 14.

Рис. 14. Соотношение времени ключевых этапов и этапов тестирования автоматизированного развёртывания с использованием снимков дисков виртуальных машин

Средняя длительность этапов развёртывания в порядке убывания:

1) одновременное создание виртуальных машин из преднастроенных снимков дисков - 1 минута 45 секунд;

2) тестирование Nexus - 39 секунд;

3) тестирование ФП РОК - 25 секунд;

4) перенастройка Nginx - 24 секунды;

5) прочие этапы - 24 секунды.

Среднее время автоматизированного развёртывания с использованием снимков дисков виртуальных машин составило 3 минуты 37 секунд. Из них:

– среднее время выполнения ключевых этапов 2 минуты 9 секунд;

– среднее время выполнения этапов тестирования 1 минута 12 секунд.

Развёртывание с помощью снимков дисков виртуальных машин дало выигрыш в 6 минут 36 секунд (64,6%) относительно развёртывания с нуля,. При этом среднее время выполнения ключевых этапов сократилось на 7 минут 19 секунд (77,3%). А среднее время тестирования увеличилось на 34 секунды (87,1%), что, впрочем, не смогло испортить положительный эффект от использования предварительно настроенных снимков дисков виртуальных машин.

Таким образом, использование предварительно настроенных снимков дисков виртуальных машин дало существенный выигрыш (64,6%) во времени развёртывания ФП РОК.

Но такой подход обладает недостатками:

– Nexus разворачивается с настройками по умолчанию, что подходит не всем заказчикам/потребителям;

– требуется предварительная настройка и подготовка снимков дисков виртуальных машин.

Результаты автоматизированного развёртывания с использованием снимков дисков виртуальных машин с последующей перенастройкой Nexus

Для удовлетворения потребностей пользователей, которым требуется индивидуальная настройка ФП РОК, после развёртывания виртуальных машин из предварительно подготовленных снимков дисков производится дополнительная перенастройка Nexus.

Результаты автоматизированного развёртывания с использованием снимков дисков виртуальных машин с последующей перенастройкой Nexus представлены в таблице 5.

Соотношение времени ключевых этапов и этапов тестирования автоматизированного развёртывания с использованием снимков дисков виртуальных машин с последующей перенастройкой Nexus представлено на рисунке 16.

Средняя длительность этапов развёртывания в порядке убывания:

Рис. 15. Результаты автоматизированного развёртывания с использованием снимков дисков виртуальных машин с последующей перенастройкой Nexu

Рис. 16. Соотношение времени ключевых этапов и этапов тестирования автоматизированного развёртывания с использованием снимков дисков виртуальных машин с последующей перенастройкой Nexus

1) перенастройка Nexus - 4 минуты 16 секунд;

2) одновременное создание виртуальных машин из преднастроенных снимков дисков - 1 минута 43 секунды;

3) тестирование ФП РОК - 26 секунд;

4) прочие этапы - 25 секунд;

5) перенастройка Nginx - 22 секунды.

Среднее время автоматизированного развёртывания с использованием

снимков дисков виртуальных машин с последующей перенастройкой Nexus

составило 7 минут 13 секунд. Из них:

– среднее время выполнения ключевых этапов 6 минут 22 секунды;

– среднее время выполнения этапов тестирования 38 секунд.

Развёртывание с помощью снимков дисков виртуальных машин с последующей перенастройкой Nexus:

– относительно развёртывания с нуля дало выигрыш в 3 минуты 0 секунд (29,3%) , описанного в подразделе 2.9.1; при этом среднее время выполнения ключевых этапов сократилось на 3 минуты 6 секунд

(32,8%), и почти никак не повлияло на среднее время тестирования (сократилось на 1 секунду или 2,1%);

– относительно развёртывания с нуля с параллельным выполнением шагов дало выигрыш в 2 минуты 45 секунд (27,6%) при этом среднее время ключевых этапов сократилось на 2 минуты 48 секунд (30,6%), и среднее время тестирования сократилось на 4 секунды (9,2%).

Результаты автоматизированного развёртывания с использованием Kubernetes API и пакетного менеджера Hel

Рис. 17. Результаты автоматизированного развёртывания с использованием Kubernetes API и пакетного менеджера Helm

Результаты авто API и пакетного менеджера Helm представлены в таблице 6 и на рисунке 17. Соотношение времени ключевых этапов и этапов тестирования автоматизированного развёртывания с использованием Kubernetes API и пакетного менеджера Helm представлено на рисунке 18. матизированного развёртывания с использованием Kubernetes 99

Рис. 18. Соотношение времени ключевых этапов и этапов тестирования автоматизированного развёртывания с использованием Kubernetes API и пакетного менеджера Helm

Средняя длительность этапов развёртывания в порядке убывания:

1) установка Helm-чарта с компонентами ФП РОК, включающая установку и настройку Nexus и Nginx - 1 минута 32 секунды;

2) тестирование ФП РОК - 20 секунд;

3) прочие этапы - 18 секунд.

Среднее время автоматизированного развёртывания с использованием Kubernetes API и пакетного менеджера Helm составило 2 минуты 10 секунд. Из них: – среднее время выполнения ключевых этапов 1 минута 32 секунды; – среднее время выполнения тестирования 20 секунд. Развёртывание с использованием Kubernetes API и пакетного менеджера Helm дало выигрыш: 101 – относительно развёртывания с нуля в 8 минут 3 секунды (78,8%), при этом среднее время выполнения ключевых этапов сократилось на 7 минут 56 секунд (83,8%), а этапов тестирования - на 19 секунд (48,1%); – относительно развёртывания с помощью снимков дисков виртуальных машин 2 минуты 5 секунд (57,6%),; при этом среднее время выполнения ключевых этапов сократилось на 37 секунд (28,7%), а этапов тестирования - на 52 секунды (72,3%); – относительно развёртывания с помощью снимков дисков виртуальных машин с последующей перенастройкой Nexus 5 минут 3 секунды (70,1%), описанного в разделе 3.3.4; при этом среднее время выполнения ключевых этапов сократилось на 4 минуты 50 секунд (75,9%), а этапов тестирования - на 18 секунд (47,4%).

3.4. Анализ результатов развёртывания различными способами Протестированные способы позволяют сократить до нескольких минут время развёртывания ФП РОК, по сравнению с одним нормо-часом ручного развёртывания, с поправкой на квалификацию администратора облачных сервисов, выполняющего установку ФП РОК, предусмотренного в карточке АС Service Manager. Для наглядности сравнения представлены две диаграммы на рисунке 19 и на рисунке 20. Использованные обозначения для различных видов развёртываний на диаграммах: Scratch - развёртывание и настройка виртуальных машин с нуля с последовательным выполнением шагов. Parallel - развёртывание и настройка виртуальных машин с нуля параллельным выполнением ключевых шагов. Snapshot - развёртывание с использованием снимков дисков виртуальных машин Reconfigure - развёртывание с использованием снимков дисков виртуальных машин с последующей перенастройкой Nexus Repository Manager. Kube - развёртывание с использованием Kubernetes API и пакетного менеджера Helm.

Рис. 19. Сравнение продолжительностей развёртывания целиком, а также ключевых этапов развёртывания и продолжительности тестирования

Диаграмма на рисунке 19 показывает, что наиболее выгодными с точки зрения полного времени развёртывания и длительности ключевых этапов являются варианты (в порядке возрастания времени развёртывания):

1) Kube;

2) Snapshot;

3) Reconfigure.

Диаграмма на рисунке 20 показывает, что наиболее выгодными с точки зрения суммарной длительности ключевых этапов и этапов тестирования, также являются варианты (в порядке возрастания времени развёртывания):

1) Kube;

2) Snapshot;

3) Reconfigure.

При этом Kube является наиболее предпочтительным вариантом по следующим причинам:

– обладает временем развёртывания меньшим, чем у остальных вариантов, включая вариант Snapshot;

– обладает недоступной для варианта Snapshot гибкостью настроек, почти как у варианта Reconfigure;

Рис. 20 Сравнение суммарной длительности ключевых этапов и этапов тестирования для каждого развёртывания, без учёта прочих этапов развёртывания

– не требует предварительной подготовки снимков дисков виртуальных машин.

В случае, если использование виртуализации на уровне ОС, как в варианте Kube, недоступно или нежелательно по какой-либо причине, в таком случае наиболее выгодно использовать варианты Snapshot или Reconfigure, в зависимости от потребностей в индивидуальной настройке Nexus Repository Manager.

Заключение

В результате выполнения исследований разработаны авторские методики развёртывания виртуальных стендов в облаке банка, с помощью которых были доказаны преимущества автоматизированного развёртывания, с использованием подхода EaC посредством взаимодействия с облачными API банка, а также преимущества виртуализации уровня операционной системы над традиционной виртуализацией.

Для этого автором были разработаны и отлажены различные сценарии развёртывания виртуальных стендов ФП РОК, в том числе предложены новые для банка подходы к развёртыванию стендов автоматизированных систем и их функциональных подсистем в облаках банка с использованием утилиты Terraform и пакетного менеджера Helm. А также проведено сравнение результатов их использования.

Разработанные программные средства позволили:

1) значительно сократить время подготовки виртуальных стендов АС и ФП, с одного нормо-часа при ручном развертывании:

– до 2 минут 10 секунд при автоматизированном развёртывании в Kubernetes с помощью пакетного менеджера Helm ;

– до 3 минут 37 секунд при автоматизированном развёртывании с использованием снимков дисков виртуальных машин ;

– до 9 минут 58 секунд при автоматизированном развёртывании с использованием API облака и инструмента Terraform

2) сократить время восстановления после сбоя виртуальных стендов в облаке банка

3) сократить количество ошибок в процессе подготовки, связанных с человеческим фактором;

4) освободить администраторов облачных сервисов от рутиной работы по подготовке однотипных стендов ФП РОК, которые теперь могут быть задействованы для автоматизации развёртывания виртуальных стендов других систем и сервисов в облаках банка.

Новизна исследования определяется разработкой авторских методик развёртывания стендов автоматизированных систем и их функциональных подсистем в облаках банка с использованием передового подхода EaC, утилиты Terraform и пакетного менеджера Helm.

Разработанные способы развёртывания ФП РОК могут быть применены для увеличения скорости развёртывания и восстановления стендов других автоматизированных систем банка, а также их функциональных подсистем с учётом их особенностей, для обеспечения основных признаков облачных вычислений, таких как «самообслуживание по требованию», «быстрая эластичность и масштабируемость»

Использование новых подходов к развёртыванию, позволит снизить время ожидания обработки заявок и количество ошибок при создании стендов, увеличить количество обрабатываемых заявок на создание стендов, и положительно повлияет на удовлетворенность клиентов облачных сервисов банка.

Список использованных источников и литературы

1. Батура Т., Мурзин Ф., Семич Д. Облачные технологии: основные модели, приложения, концепции и тендеции развития // Программные продукты и системы. — 2014. — DOI: 10.15827/0236- 235X. 107.064-072. — URL: http://www.swsys.ru/index.php? page=article&id=3862&lang=.

2. ГОСТ ISO/IEC 17788-2016 Информационные технологии (ИТ). Облачные вычисления. Общие положения и терминология. — Межгосударственный совет по стандартизации, метрологии и сертификации СНГ.

3. Aiello B., Sachs L. Configuration Management Best Practices Practical Methods that Work in the Real World. — Addison-Wesley, 2011.

4. Ansible Documentation. — URL: https://docs.ansible.com/ ansible/latest/index.html.

5. Brikman Y. Terraform: Up and Running. — 2nd edition. — O’Reilly Media, 2019.

6. Docker Documentation. — URL: https://docs.docker.com/.

7. Everything as Code: The future of ops tools. — URL: https://www. hashicorp.com/resources/everything- as- code- thefuture-of-ops-tools.

8. Hamdaqa M., Tahvildari L. Advances in Computers. Т. 86. — Elsevier, 01.2012. — ISBN 978-0-12-396535-6.

9. Handbook of Cloud Computing / под ред. B. Furht, A. Escalante. — Springer, 2010.

10. Helm Documentation. — URL: https://helm.sh/docs/.

11. Hightower K., Burns B., Beda J. Kubernetes: Up and Running / под ред. A. Rufino. — O’Reilly, 2017. 109

12. Jenkins User Documentation. — URL: https://www.jenkins.io/ doc/.

13. Jinja2 Documentation. — URL: https://jinja.palletsprojects. com/en/2.11.x/.

14. Kubernetes Documentation. — URL: https://kubernetes.io/ docs/.

15. Kumaran S. S. Practical LXC and LXD: Linux Containers for Virtualization and Orchestration / под ред. S. McIntyre. — Apress, 2017. — DOI: 10. 1007/978-1-4842-3024-4\_1.

16. Medina O., Schumann E. DevOps for SharePoint With Packer, Terraform, Ansible, and Vagrant. — Apress, 2018.

17. Morabito R., Kjallman J. ¨ , Komu M. IEEE International Conference on Cloud Engineering // Hypervisors vs. Lightweight Virtualization: A Performance Comparison. — Tempe, AZ, USA : IEEE, 2015. — URL: https:// ieeexplore.ieee.org/abstract/document/7092949/.

18. Morris K. Infrastructure as Code: managing servers in the cloud. — O’Reilly Media, 2016.

19. Nginx Documentation. — URL: http://nginx.org/en/docs/.

20. Nickoloff J. Docker in Action / под ред. C. Kane. — Manning, 2016.

21. Podman Documentation. — URL: http://docs.podman.io/.

22. Portnoy M. Virtualization essentials / под ред. J. Lefevere. — 2nd edition. — John Wiley & Sons, Inc, 2016.

23. POSIX. — URL: https://ru.wikipedia.org/wiki/POSIX.

24. RedHat Documentation. — URL: https://access.redhat.com/ documentation/. 25. REST. — URL: https://ru.wikipedia.org/wiki/REST. 110

26. Saito H., Lee H.-C. C., Wu C.-Y. DevOps with Kubernetes: Accelerating software delivery with container orchestrators. — Packt, 2017. — ISBN 978-1-78839-664-6.

27. Smith R. Docker Orchestration / под ред. V. D. Smet. — Packt, 2017. — ISBN 978-1-78712-212-3.

28. Sonatype Nexus Repository Manager 3 Documentation. — URL: https: //help.sonatype.com/repomanager3.

29. Special Publication 800-145 The NIST Definition of Cloud Computing. — National Institute of Standards, Technology U.S. Department of Commerce.

30. Terraform Language Documentation. — URL: https://www.terraform. io/docs/configuration/index.html.

31. Watson R. Comparing Leading Cloud-Native Application Platforms: Pivotal Cloud Foundry and Red Hat OpenShift : тех. отч. / Garthner. — 06.2017. — G00319528.

32. Контрольная группа (Linux). — URL: https://ru.wikipedia. org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0% BB%D1%8C%D0%BD%D0%B0%D1%8F\_%D0%B3%D1%80%D1%83%D0% BF%D0%BF%D0%B0\_(Linux).

33. Пространство имён (Linux). — URL: https://ru.wikipedia. org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0% B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE\_%D0%B8%D0%BC%D1% 91%D0%BD\_(Linux).