DOI: 10.15514/ISPRAS-2022-34(5)-9

Исследование методов построения облачных платформенных сервисов и реализаций стандарта TOSCA

[1,2] *А.А. Борисова, ORCID: 0000-0003-4558-7872 <aaborisova_4@edu.hse.ru>*
[2] *О.Д. Борисенко, ORCID: 0000-0001-8297-5861 <al@somestuff.ru>*

[1] *Национальный исследовательский университет «Высшая школа экономики», 101000, Россия, г. Москва, ул. Мясницкая, д. 20*
[2] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*

Аннотация. В статье рассматриваются и сравниваются различные инструменты автоматизации управления ресурсами в облаке. Изменения в архитектуре программного обеспечения и подходов к разработке требуют автоматизации процессов управления развертывания и дальнейшего сопровождения по в разных средах. В разд. 2 представлен подробный обзор инструментов с примерами конфигураций, а также разбор релевантных статей, рассматривающих различные инструменты автоматизации и эффективность их внедрения. В разд. 3 представлен проект решения по объединению оркестраторов, разработанных в ИСП РАН, для получения инструмента с функционалом, которого нет у конкурентов.

Ключевые слова: оркестрация; развертывание ПО; TOSCA

Для цитирования: Борисова А.А., Борисенко О.Д. Исследование методов построения облачных платформенных сервисов и реализаций стандарта TOSCA. Труды ИСП РАН, том 34, вып. 5, 2022 г., стр. 143-162. DOI: 10.15514/ISPRAS-2022-34(5)-9

Благодарности: Работа выполнена при финансовой поддержке Министерства науки и высшего образования Российской Федерации (соглашение №075-15-2022-294 от 15 апреля 2022)

Research of Construction Methods for Cloud Services and Overview of the Implementations TOSCA Standard

[1,2] *A.A. Borisova, ORCID: 0000-0003-4558-7872 <aaborisova_4@edu.hse.ru>*
[2] *O.D. Borisenko, ORCID: 0000-0001-8297-5861 <al@somestuff.ru>*

[1] *HSE University, 20, Myasnitskaya st., Moscow, 101000 Russia*
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*

Abstract. This paper overview and compares various tools for automating resource management in the cloud. Changes in software architecture and development approaches require automation of deployment management processes and further maintenance of software in different environments. Chapter 2 provides a detailed overview of the tools with sample configurations, as well as a breakdown of relevant articles that look at various automation tools and the effectiveness of their implementation. Chapter 3 presents a draft solution for combining orchestrators developed at ISP RAS to obtain a tool with functionality that competitors do not have.

Keywords: Orchestration; Software Deployment; TOSCA

For citation: Borisova A.A., Borisenko O.D. Research of Construction Methods for Cloud Services and Overview of the Implementations TOSCA Standard. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 5, 2022. pp. 143-162 (in Russian). DOI: 10.15514/ISPRAS-2022-34(5)-9

Acknowledgements. This work was supported by the Ministry of Science and Higher Education of the Russian Federation, agreement No. 075-15-2022-294 dated 15 April 2022.

1. Введение

В индустриальной разработке программного обеспечения наблюдается переход от монолитных архитектур к микросервисным. Микросервисом называют сервис, у которого определена роль и набор процессов, которые он обрабатывает. Преимущество по сравнению с монолитными архитектурами состоит в том, что управление и масштабирование микросервиса может производиться независимо от других микросервисов системы. При этом сервисы можно объединить в группы по специфике использования (например: базы данных, служебные сервисы, сервисы приложений и пр.). Для каждой группы микросервисов жизненный цикл управления – разный. Микросервисы общаются между собой по протоколам передачи данных через программные интерфейсы приложений или передачу сообщений. Архитектура разрабатываемого ПО влияет на то, как будет организованы разработка, поставка и сопровождение ПО, обслуживание инфраструктуры, и организация команды. Переход от монолитной архитектуры к микросервисной влечет за собой переход подходов: обслуживания баз данных; обслуживания инфраструктуры; мониторинга; организации процесса разработки.

Внедрение техник автоматизации процессов разработки (DevOps – development and operations) дает значительный прирост скорости поставки ПО, делает итерации разработки короче и увеличивает частоту поставки. К тому же уменьшается количество отказов системы. Это достигается за счет того, что каждый микросервис поставляется в виде контейнера. Контейнер — это единица поставляемого ПО, содержащая в себе все необходимые для его работы библиотеки и зависимости.

В стандартный жизненный цикл монолитного приложения входит *ручное*:

- развертывание;
- обновление;
- удаление;
- размещение;
- создание резервной копии (англ. backup);
- восстановление из сохраненной копии.

Для управления микросервисным системами "ручной" подход (и даже автоматизация отдельных задач) неприменим, т.к. количество сервисов для одной системы может измеряться тысячами. Поэтому ЖЦ был расширен и автоматизирован. Для микросервисов продвинутый ЖЦ позволяет:

- автоматическое управление стандартным ЖЦ ПО;
- автоматическое масштабирование;
- автоматический возврат ПО к предыдущей версии (или "откат изменений" - англ. rollback);
- развертывание изменений для части сервисов, а не на все сразу (canary и blue-green развертывания).

К тому же, были разработаны подходы автоматического управления, которые способны реагировать на изменения в микросервисах и по необходимости менять шаг жизненного цикла (напр. автоматическое восстановление при возникновении ошибок во время развертывания). Для эффективного управления сложными системами необходимо повышать прозрачность и просматриваемость (observability) системы. Необходимо отслеживать очаги

аномальной работы, падений (иногда отказ одного сервиса тянет за собой отказ связанных сервисов). Все ошибки записываются, отслеживаются и измеряются метриками - этот процесс называется мониторингом системы. Также вводится понятие состояние сервиса и предоставляются хранилища состояний, для отслеживания состояния сервиса и системы в целом в конкретный период времени. Примеры состояний: запущен, в ожидании, прерван. Не всегда отображается действительное состояние сервиса, он может быть запущен, но может не отвечать на определенного рода запросы. Для более детальной информации используются обработчики событий, которые посылают запросы на проверку "здоровья" (работоспособности) сервиса. Все это дает четкое представление о работе системы. Как итог, видны критические сервисы, и сервисы, на которые идет большая/меньшая нагрузка. Все это позволяет сделать систему в целом более управляемой, позволяет быстрее реагировать на изменение потребления ресурсов системы (напр. автоматическое масштабирование при высоком потреблении ресурсов) [3].

Стоит отметить, что для работы распределенной системы с микросервисной архитектурой, необходимо поддерживать рабочее состояние сети и обеспечивать связывание с другими микросервисами и устаревшими системами по сети (используя различные протоколы и сервисы обмена сообщениями).

Рассмотрим все вышеописанное на примере. Предположим, что для продукта была выбрана микросервисная архитектура. При этом происходят следующие изменения: команда продукта делится на множество мелких подкоманд, которые отвечают каждая за свой сервис. Развертывание микросервисов может происходить независимо друг от друга и с частотой менее одного дня. Само развертывание микросервиса длится не долго. Управление микросервисом происходит автоматически. Микросервис можно развернуть на тестовой среде и протестировать изменения перед развертыванием в промышленную среду. При этом для микросервисов доступно продвинутое управление ЖЦ ПО. Устройство системы позволяет быстро реагировать на изменения рынка (разрабатывать и поставлять обновления ПО как можно быстрее), и получать на выходе успешный и конкурентоспособный продукт.

Однако реализация перехода с монолитных на микросервисные архитектуры оказалась проблематичной в направлениях создания инфраструктуры, а также развертывания и управления сразу множеством сервисов. Благодаря появлению контейнеров и инструментов оркестрации проблемы были решены.

Для конфигурации и управления монолитным ПО используются инструменты автоматизации стандартных задач ЖЦ ПО. Но для микросервисов этого недостаточно, поэтому используются инструменты оркестрации, которые позволяют автоматизировать управление набором отдельных автоматизированных задач. Термин «оркестрация» не имеет зафиксированного определения и границы сильно размыты. Однако можно описать набор задач, которые должен решать оркестратор. А именно: автоматизировать набор задач по развертыванию, координации и управлению инфраструктурой, ПО и сервисами. Процесс оркестрации зависит от того, как вообще организована поставка ПО для конкретного продукта. Включение оркестрации в процессы позволяет ощутимо для компании и не ощутимо для пользователя поставлять обновления, менять схему взаимодействия сервисов, исправлять перебои и пр. Гибкая разработка ПО напрямую связана с гибким управлением и поставкой. Это позволяет наиболее эффективно использовать ресурсы. С появлением оркестрации появилась возможность делать безопасные Canary, и Blue-Green релизы. Производить частичное тестирование новых функций ПО до того, как масштабировать изменения на всех пользователей. Платформы оркестрации позволяют осуществлять более надежную и стабильную работу системы в целом, а также осуществлять поиск возможных ошибок на ранних этапах и быстрое обслуживание системы, без увеличения количества сотрудников.

Контейнеры более легковесные и управляемые, ресурсы контейнера изолированы от других частей системы. Контейнерам не нужно настраивать и развертывать уровень

инфраструктуры, в нем уже содержится инфраструктура в виде слоя базового образа. Инфраструктурой называют часть сервисного ПО, которая необходима для работы разрабатываемого ПО. Процесс оркестрации контейнеров и виртуальных машин отличается по этапам и построению. Для контейнеров используется платформы оркестрации Docker Swarm и Kubernetes. На начальном этапе пользователь уже имеет «собранный» вручную контейнер, обслуживаемый контейнерным движком.

Настройки сборки прописываются при запуске сборки или конфигурируются на следующем этапе. Далее пользователь пишет конфигурацию сервиса (или группы сервисов). В Kubernetes есть API конфигурации, который абстрактно описывает элементы инфраструктуры. По API для контейнеров можно понять, какую роль выполняет контейнер (напр. рабочая нагрузка, БД, балансировщик и т.д.). Также в конфигурации описано поведение контейнера при отказах, заданы связи с другими контейнерами. Для более сложных функций работы контейнеров друг с другом используются специальные настраиваемые элементы - «операторы».

В отличие от контейнеров, ПО не имеет под собой инфраструктуры. Создание инфраструктуры выполняется не линейно. Перед запуском сценария развертывания может выполниться подготовительный шаг, а после - недостающие параметры подставляются в конфигурацию и запускается скрипт донастройки. Это объясняется тем, что виртуализация на базе гипервизора требует развертывания на аппаратной части, в то время как виртуализация контейнеров выполняет развертывание на уровне операционной системы. Таким образом, при развертывании виртуальных машин на базе гипервизора необходимо развернуть уровень инфраструктуры и настроить его (этот этап называют Infrastructure Provisioning), затем собрать и развернуть на инфраструктуре сервисы (этот этап называют Configuration management). Для каждого уровня свои сценарии развертывания. Связи и отношения между сервисами также заданы в сценарии развертывания (поэтому это не так прозрачно, как у контейнеров). Инструменты конфигурации очень гибкие и работают на высоком уровне абстракции, поэтому никак не ограничивают роль, которую может выполнять сервис, что делает процесс настройки намного более сложным, нежели настройка контейнеров. При наличии таких сценариев можно начинать использовать Оркестратор, который позволит создавать сценарии управления группой сервисов. Далее подключаются сервисы мониторинга и переход на этап обслуживания и мониторинга.

На практике распределенные приложения требуют решений и для обслуживания виртуальных машин, и для обслуживания контейнеров. Для контейнеров стандартом стал Kubernetes – как наиболее мощная и развивающаяся платформа. А для виртуальных машин оркестрация остается сложно реализуемой, потому что уровень виртуализации ближе к аппаратной части. Поэтому оркестраторы поставляются для каждого провайдера по отдельности, либо процесс оркестрации реализуется инструментами конфигурации в полуавтоматическом режиме. Задача построения мультиоблачного оркестратора с поддержкой нескольких инструментов конфигурации остается актуальной.

На пути реализации такого оркестратора был создан стандарт TOSCA [2], позволяющий выйти на необходимый уровень абстракции. Стандартом описаны сущности облачных вычислений без привязки к инструменту или провайдеру. На базе стандарта стало возможно реализовать мультиоблачный оркестратор.

Еще одной актуальной проблемой является измерение эффективности внедрения оркестратора.

2. Обзор решений

В обзоре технологий рассмотрены различные инструменты оркестрации. Сравнение инструментов возможно, при учете следующих факторов: уровень виртуализации, уровень доступа к ресурсам, поддержка одного/нескольких облачных провайдеров, возможность

взаимодействия с инструментами конфигурации, графический интерфейс построения топологий и пр. Эти факторы необходимо учитывать, т.к. некорректно сравнивать оркестраторы различных уровней доступа к ресурсам или же сравнивать производительность инструмента конфигурации и оркестратора контейнеров. Поэтому технологии разбиты на 4 категории, в каждой из которых описан обзор и предоставлено сравнение инструментов.

2.1 Инструменты, позволяющие осуществлять оркестрацию

2.1.1 Terraform

Terraform – инструмент, осуществляющий оркестрацию для различных облачных провайдеров (более 30) уровня IaaS на этапе настройки и развертывания [4]. Для некоторых провайдеров доступен PaaS уровень. Поддерживается создание, удаление и обновление единиц развертывания. Единицей развертывания является модуль, поддерживаемый для данного провайдера. Конфигурация задается через Terraform configuration – шаблон описания конфигурации для каждого провайдера на языке Terraform. Не поддерживаются функции масштабирования, мониторинга, обслуживания ЖЦ ПО (например, автоматическое восстановление в случае отказа).

Terraform предоставляет часть функций бесплатно. Функции Terraform: удаленное хранилище состояний, возможность удаленных запусков, соединение с инструментами контроля версий (Github, Gitlab, Bitbucket, Azure Devops). В индустриальной версии Terraform есть возможность поддержки собственного облачного сервиса Terraform. При этом облако можно разделить между организациями, у которых будет отдельный счет для оплаты ресурсов и собственное рабочее пространство. Внутри каждой организации ресурсы можно разделить и ограничить между командами разработки и пользователями. Terraform можно самостоятельно расширять, добавляя поддержку необходимого провайдера. Однако официальные модули крупных облачных провайдеров обновляются со значительной задержкой. Инструмент позволяет описывать зависимости между модулями. Но при развертывании и удалении проверяется только порядок, а не совместимость и наличие циклов.

Terraform подходит для простых задач по подготовке инфраструктуры для различных облачных провайдеров. На листинге 1 представлен пример сценария развертывания с использованием Terraform. Оркестрация сервисов PaaS уровня менее полезна, потому что инструмент не поддерживает продвинутых функций оркестрации. Инструмент не предоставляет доступ к своим собственным PaaS сервисам, а лишь управляет сервисами провайдеров. К тому же язык Terraform требует изучения и не позволяет использовать внутри себя вставки с вызовом кода. Terraform позволяет описать топологию, однако шаблоны описания не универсальны и для каждого провайдера необходимо создавать сценарий.

Terraform не предоставляет ресурсы по запросу, но позволяет управлять этими ресурсами.

```
#### INSTANCE DB ####

# Create instance
#
resource "openstack_compute_instance_v2" "db" {
  name        = "front01"
  image_name  = var.image
  flavor_name = var.flavor_db
  key_pair    = openstack_compute_keypair_v2.user_key.name
  user_data   = file("scripts/first-boot.sh")
  network {
    port = openstack_networking_port_v2.db.id
  }
}

# Create network port
resource "openstack_networking_port_v2" "db" {
  name           = "port-instance-db"
  network_id     = openstack_networking_network_v2.generic.id
  admin_state_up = true
  security_group_ids = [
    openstack_compute_secgroup_v2.ssh.id,
    openstack_compute_secgroup_v2.db.id,
  ]
  fixed_ip {
    subnet_id = openstack_networking_subnet_v2.http.id
  }
}

# Create floating ip
resource "openstack_networking_floatingip_v2" "db" {
  pool = var.external_network
}

# Attach floating ip to instance
resource "openstack_compute_floatingip_associate_v2" "db" {
  floating_ip = openstack_networking_floatingip_v2.db.address
  instance_id = openstack_compute_instance_v2.db.id
}

#### VOLUME MANAGEMENT ####

# Create volume
resource "openstack_blockstorage_volume_v2" "db" {
  name = "volume-db"
  size = var.volume_db
}

# Attach volume to instance instance db
resource "openstack_compute_volume_attach_v2" "db" {
  instance_id = openstack_compute_instance_v2.db.id
  volume_id   = openstack_blockstorage_volume_v2.db.id
}
```

Листинг 1. Пример сценария развертывания ресурсов Openstack в Terraform
Listing 1. Example of a Terraform deployment scenario for OpenStack

2.1.2 Configuration management software

Инструменты, такие как Ansible, Chef, Puppet, имеют схожий функционал [5, 6, 7].

Все они поддерживают несколько облачных провайдеров, для которых описаны модули, с которыми они работают. В сценариях можно описывать не только развертывание, но и любую конфигурацию. В сценарии можно описать развертывания различных модулей различных провайдеров. Сценарии не являются шаблонами, не предназначены для пере

использования между провайдерами. Инструментами конфигурации поддерживаются все уровни доступа к ресурсам. Инструменты конфигурации позволяют осуществлять процессы оркестрации, однако написание вручную шаблонов и вызов функций не дают им преимущество по сравнению с оркестраторами.

Рассмотрим один из инструментов конфигурации.

Ansible

Ansible – мощный инструмент настройки для автоматизации управления вычислительными ресурсами в облаке. Благодаря декларативному описанию на предметно-ориентированном языке в формате YAML специалисты могут легко настраивать, развертывать и контролировать инфраструктуры, приложения, сети и множество других сущностей. Файлы описания конфигурации называются Playbook, и они состоят из списка задач для исполнения, будем называть их «сценариями Ansible». На листинге 2 представлен пример сценария развертывания с использованием Ansible.

Однако использование этого инструмента требует понимания правил построения Ansible сценариев, а также знания точного API провайдера конкретного облачного провайдера. Это вызывает проблемы при использовании, когда конфигурация приложения уже выполнена, и требуется развернуть приложение с использованием услуг другого облачного провайдера. Это то, что отличает его от шаблонов унифицированного абстрактного описания, которые не зависят от конкретной технологии или поставщика услуг.

```
- name: Create OpenStack component openstack cluster
  hosts: localhost
  tasks:
  - name: Create OpenStack component server
    os_server:
      config_drive: false
      name: server_master
      flavor: '{{ id_3387 }}'
      image: '{{ name_8282 }}'
    register: server_master_server
  - set_fact:
      server_master_server_list: '{{ server_master_server_list
      | default([])
        }} + [ "{{ item.id }}" ]'
    loop: '{{ server_master_server.results | flatten(levels=1)  }}'
    when: item.id  is defined
  - set_fact:
      server_master_server_list:
        server_master_server_ids: '{{ server_master_server_list }}'
    when: server_master_server_list is defined
  - lineinfile:
      path: '{{ playbook_dir }}/id_vars_example.yaml'
      line: 'server_master_server_delete: {{ server_master_server.id }}'
    when: server_master_server.id is defined
  - lineinfile:
      path: '{{ playbook_dir }}/id_vars_example.yaml'
      line: '{{ server_master_server_list | to_nice_yaml }}'
    when: server_master_server_list is defined
  - fail:
```



```
      msg: Variable server_master_server is undefined! So it will not be
deleted
    when:      server_master_server_list      is      undefined      and
server_master_server.id
      is undefined
    ignore_errors: true
```

Листинг 2. Пример сценария развертывания ресурсов Openstack в Ansible
Listing 2. Example of a Ansible deployment scenario for OpenStack

Ansible Galaxy представляет собой цифровую библиотеку открытых исходных кодов, которая содержит более 27 тысяч сценариев конфигурации сервисов, оформленных в виде ролей Ansible [8]. Далеко не все они работают, и большая часть из них требует дополнительной подготовки перед использованием.

Ansible используется множеством компаний для сопровождения их сервисов, а Galaxy стал местом для обмена лучшими практиками. Число людей, входящих в это сообщество, превосходит 299 тысяч человек.

2.2 Оркестраторы с поддержкой одного облачного провайдера

2.2.1 Cloud Formation

Cloud Formation – оркестратор облачного провайдера Amazon AWS уровня IaaS [9]. Поддерживает создание, удаление и обновление единиц развертывания. Единицей развертывания является Stack – набор ресурсов (напр. балансировщик, виртуальные машины, хранилища данных). Конфигурация задается через AWS templates в формате YAML/JSON на языке Cloud Formation template language. Шаблоны можно проверять на корректность. Можно задавать связи типа «Depends on» внутри шаблона. Поддерживается функция управления ЖЦ ПО сине-зеленое развертывание, сохранение резервной копии и восстановление из нее, автоматическое масштабирование. Пример сценария развертывания приведен на листинге 3.

Также, по шаблону можно создать "безсервисную" конфигурацию и получить развертывание уровня FaaS (функция как сервис). Шаблоны покрывают все ресурсов провайдера, в том числе сервисы мониторинга и сопровождения ЖЦ ПО. Провайдер Amazon имеет встроенный инструмент графического задания топологий формата CloudFormation. CloudFormation это полноценный оркестратор, покрывающий все функции для одного облачного провайдера Amazon AWS.

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",

  "Description"     :     "AWS     CloudFormation     Sample     Template
EC2InstanceWithSecurityGroupSample:  Create  an  Amazon  EC2  instance
running the Amazon Linux AMI. The AMI is chosen based on the region in
which the stack is run. This example creates an EC2 security group for
the instance to give you SSH access. **WARNING** This template creates
an Amazon EC2 instance. You will be billed for the AWS resources used if
you create a stack from this template.",

  "Parameters" : {
    "KeyName": {
      "Description" : "Name of an existing EC2 KeyPair to enable
       SSH access to the instance",
      "Type": "AWS::EC2::KeyPair::KeyName",
      "ConstraintDescription" : "must be the name of an existing
       EC2 KeyPair."
```

```
    },

    "InstanceType" : {
      "Description" : "WebServer EC2 instance type",
      "Type" : "String",
      "Default" : "t2.small",
      "AllowedValues" : [ "t1.micro", "t2.nano", "t2.micro", "t2.small",
"t2.medium",    "t2.large",    "m1.small",    "m1.medium",    "m1.large",
"m1.xlarge",   "m2.xlarge",   "m2.2xlarge",   "m2.4xlarge",   "m3.medium",
"m3.large",    "m3.xlarge",    "m3.2xlarge",    "m4.large",    "m4.xlarge",
"m4.2xlarge",  "m4.4xlarge",  "m4.10xlarge",  "c1.medium",  "c1.xlarge",
"c3.large",   "c3.xlarge",   "c3.2xlarge",   "c3.4xlarge",   "c3.8xlarge",
"c4.large",   "c4.xlarge",   "c4.2xlarge",   "c4.4xlarge",   "c4.8xlarge",
"g2.2xlarge",   "g2.8xlarge",   "r3.large",   "r3.xlarge",   "r3.2xlarge",
"r3.4xlarge",  "r3.8xlarge",  "i2.xlarge",  "i2.2xlarge",  "i2.4xlarge",
"i2.8xlarge",  "d2.xlarge",  "d2.2xlarge",  "d2.4xlarge",  "d2.8xlarge",
"hi1.4xlarge",     "hs1.8xlarge",     "cr1.8xlarge",     "cc2.8xlarge",
"cg1.4xlarge"]
,
      "ConstraintDescription" : "must be a valid EC2 instance type."
    },

    "SSHLocation" : {
      "Description" : "The IP address range that can be used to SSH to
the EC2 instances",
      "Type": "String",
      "MinLength": "9",
      "MaxLength": "18",
      "Default": "0.0.0.0/0",
      "AllowedPattern":
"(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})/(\\d{1,2})",
      "ConstraintDescription": "must be a valid IP CIDR range of the form
x.x.x.x/x."
   }
 }
```

Листинг 3. Пример сценария развертывания ресурсов Amazon в CloudFormation
Listing 3. Example of a CloudFormation deployment scenario for Amazon

2.2.2 Heat Orchestrator

Heat – инструмент, осуществляющий оркестрацию для облачного провайдера OpenStack уровня IaaS [10]. Поддерживает создание, удаление и обновление единиц развертывания. Единицей развертывания является Stack – набор ресурсов (напр. балансировщик нагрузки, виртуальные машины, хранилища данных). Конфигурация задается через Heat Orchastration Templates – шаблоны TOSCA ненормативных типов Openstack. Пример сценария развертывания показан на листинге 4.

Продвинутые функции управления жизненным циклом ПО не поддерживаются. Имеется Amazon AWS Query API для поддержки создания сопоставимых ресурсов Cloud Formation. Провайдер OpenStack имеет встроенный инструмент графического задания топологий формата Heat.

```
heat_template_version: 2015-04-30

description: Simple template to deploy a single compute instance
```



```
parameters:
  key_name:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance
  image_id:
    type: string
    label: Image ID
    description: Image to be used for compute instance
  instance_type:
    type: string
    label: Instance Type
    description: Type of instance (flavor) to be used

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: { get_param: key_name }
      image: { get_param: image_id }
      flavor: { get_param: instance_type }
```

Рис. 4. Пример сценария развертывания ресурсов OpenStack в Heat Orchestrator
Fig. 4. Example of a Heat deployment scenario for Openstack

2.2.3 Michman

Michman (разработка ИСП РАН) - это оркестратор уровня PaaS провайдера OpenStack [11, 12]. Отличие оркестратора в том, что сервисы предоставляются по запросу и уровень инфраструктуры настраивается автоматически. Сейчас реализована поддержка следующих сервисов: Apache Spark, Apache Hadoop, Apache Ignite, Apache Cassandra, ClickHouse, CouchDB, CVAT, ElasticSearch with OpenDistro tools, Jupyter, Jupyterhub, Kubernetes, Nextcloud, NFS-Server, Slurm, PostgreSQL, Redis. Есть возможность добавления поддержки собственного сервиса. Michman поддерживает создание и удаление единиц развертывания, а также масштабирование сервиса. Единицей развертывания является сервис, который пользователь выбирает и настраивает через графический или программный интерфейс. Michman использует Ansible, как инструмент конфигурации. Для развёртывания некоторого сервиса пользователю необходимо задать выбрать сервис, а затем выполнить одну команду для развертывания. Michman автоматически подберет параметры IaaS и обработает зависимости между сервисами. Результатом программы является сервис, развернутый в облаке Openstack. Не поддерживаются функции, мониторинга и продвинутого обслуживания ЖЦ ПО (например, автоматическое восстановление в случае отказа). Нет графического редактора топологий.

2.3 Оркестраторы нескольких облачных провайдеров

2.3.1 Cloudify

Cloudify – мультиоблачный оркестратор уровня IaaS и контейнеров .[13]. Расширяем при помощи добавления плагинов. На данный момент плагины реализованы для интеграции с

инструментами конфигураций, развертывания IaaS ресурсов и осуществления оркестрации. Cloudify поддерживает создание, удаление и обновление единиц развертывания а также любые операции, которые поддерживают инструменты Terraform, Ansible, Helm (Kubernetes), Docker. Единицей развертывания является Blueprint template – топология, описанная на похожем на стандарт TOSCA предметно-ориентированном языке в формате YAML файла. Пример сценария развертывания приведен на листинге 5.

Шаблоны покрывают ресурсы и операции ЖЦ ПО, реализованные в соответствующих плагинах. Поддерживает интеграцию с CI/CD инструментами Jenkins, Github Actions, CircleCI. Cloudify имеет графический инструмент создания топологий, что делает порог вхождения в Cloudify dsl ниже и позволяет визуально отобразить топологию и связи между элементами.

```
tosca_definitions_version: cloudify_dsl_1_3

imports:
  - https://cloudify.co/spec/cloudify/6.3.0/types.yaml
  - plugin:cloudify-openstack-plugin?version= >=3.2.2
  - plugin:cloudify-ansible-plugin
  - plugin:cloudify-utilities-plugin?version= >=1.22.1
  - includes/hello-world-ansible.yaml

inputs:
  <parameters_here>

dsl_definitions:

  openstack_config: &openstack_config
    auth_url: { get_secret: openstack_auth_url }
    region_name: { get_input: region }
    project_name: { get_secret: openstack_tenant_name }
    username: { get_secret: openstack_username }
    password: { get_secret: openstack_password }
    user_domain_name: { get_input: user_domain_name }
    project_domain_name: { get_input: project_domain_name }

node_templates:

  vm:
    type: cloudify.nodes.openstack.Server
    properties:
      client_config: *openstack_config
      agent_config:
        install_method: none
        key: { get_attribute: [agent_key, private_key_export] }
        user: { get_input: agent_user }
      resource_config:
        name: vm
        image_id: { get_input: image }
        flavor_id: { get_input: flavor }
        user_data: { get_attribute: [ cloud_init, cloud_config ] }
      use_public_ip: true
```



```
    relationships:
      - type: cloudify.relationships.openstack.server_connected_to_port
        target: port
      - type: cloudify.relationships.depends_on
        target: cloud_init
```

Рис. 5. Пример сценария развертывания ресурсов Openstack в Cloudify
Fig. 5. Example of a Cloudify deployment scenario for Openstack

2.3.2 xOpera Orchestrator

xOpera – мультиоблачный оркестратор уровня IaaS провайдера Openstack, FaaS провайдеров Amazon, Azure, Google Cloud Platform и SaaS [14]. В xOpera реализована поддержка соединения между облаками (запрос можно последовательно обрабатывать на ресурсах разных провайдеров). xOpera поддерживает создание, удаление единиц развертывания (обновление происходит как удаление и развертывание новых). Единицей развертывания является TOSCA template – топология, описанная по стандарту TOSCA в формате YAML файла. Шаблоны содержат описания в ненормативных типах, зависящих от провайдера. На каждый ресурс необходимо предоставить артефакты со сценариями создания и удаления на Ansible. xOpera поддерживает интеграцию только с инструментом конфигурации Ansible. Параметры из шаблона TOSCA подставляются в сценарий при запуске оркестратором сценария развертывания Ansible. И шаблоны, и сценарии остаются провайдер-зависимыми. Нельзя добавлять поддержку нового инструмента конфигурации, но можно описать свои ненормативные типы ля добавления нового провайдера.

2.3.3 Clouni

Clouni (разработка ИСП РАН) - это TOSCA мультиоблачный оркестратор уровня IaaS провайдеров OpenStack, Amazon, Kubernetes [15]. Основная особенность этого инструмента - использование нормативных типов TOSCA для их автоматического преобразования в сценарии для поддерживаемых облачных провайдеров. Clouni поддерживает создание и удаление единиц развертывания. Единицей развертывания является TOSCA template – топология, описанная по стандарту TOSCA в формате YAML файла. Инструменты конфигурации для разных провайдеров могут отличаться. Например, для Kubernetes провайдеров используются Kubernetes манифесты, для OpenStack провайдеров - сценарии Ansible. Для развёртывания некоторой инфраструктуры пользователю необходимо описать эту инфраструктуру в TOSCA шаблоне с использованием нормативных типов (параметры, описанные в стандарте TOSCA, которые интуитивно понятны, например, ЦП, размер памяти и т. д.). А затем выполнить одну команду для создания сценария развёртывания. Результатом программы является готовый к развертыванию сценарий или манифест. Еще одна особенность заключается в том, что любой может добавлять поддержку облачных провайдеров, создав определение ненормативных TOSCA типов ресурсов облачного провайдера и схему сопоставления с нормативными типами.

В настоящее время Clouni реализовано несколько нормативных типов TOSCA. А именно преобразование Compute, Network и Port в Ansible Playbooks для провайдеров OpenStack и только Compute в Amazon и в Kubernetes Manifest для контейнерной платформы Kubernetes. Сценарии Ansible генерируются для создания и удаления инфраструктуры. Манифесты Kubernetes следует использовать с командами kubectl apply/delete.

2.3.4 Alien4Cloud

Alien4Cloud - это инструмент моделирования TOSCA с открытым исходным кодом [16]. Его основное отличие состоит в том, что он позволяет разворачивать сервисные кластеры из веб-интерфейса за счёт подключаемых плагинов. Каждый плагин трансформирует запросы и

ответы Alien4Cloud и некоторого TOSCA сервиса, что позволяет использовать внешний TOSCA сервис из веб-интерфейса Alien4Cloud. Кроме того, инструмент предоставляет возможность добавлять собственные расширяющие плагины с определенной логикой (например, дополнительный оркестратор TOSCA). Alien4cloud не меняет вводимые данные и позволяет использовать собственный нормативный тип TOSCA. Реализована функция мониторинга.

Yorc оркестратор - официальный мультиоблачный оркестратор TOSCA для Alien4cloud уровня IaaS провайдеров Amazon, OpenStack, Google Cloud Platform, Kubernetes [17]. Также, Yorc предоставляет Slurm как сервис (PaaS уровень). Yorc использует Alien4cloud типы TOSCA на основе TOSCA Simple YAML Profile v.1.2. Надлежащая документация по обоим инструментам четко описывает, как использовать инструменты вместе. Единицей развертывания является архив, содержащий TOSCA template. Продвинутые функции управления жизненным циклом ПО не поддерживаются. К сожалению, примеры процессов оркестрации из документации нельзя оценить, потому что образцы не обновлялись долгое время и не работали. Однако же удалось подключить Alien4cloud к Openstack и запустить развертывание виртуальной машины, которое оказалось не успешным. По анализу логов, можно убедиться, что Alien4cloud использует Terraform как инструмент развертывание и подставляет готовые сценарии развертывание для заданных ненормативных типов.

2.4 Оркестраторы контейнеров

2.4.1 Docker Swarm

Swarm это специальный режим работы сервиса Docker. Swarm не предоставляет ресурсы по запросу, он только управляет контейнерами. Поддерживает развертывание, удаление, обновление и масштабирование ресурсов. Единицей развертывания является стек - набор сервисов-контейнеров. Конфигурация развертывания задается через конфигурационный файл docker compose. Swarm использует декларативный подход и одновременно выступает в роли инструмента оркестрации контейнеров. Мониторинг сервисов не производится, но можно получить состояние сервиса по запросу. При обновлении стека можно указать порядок обновления сервисов, количество контейнеров, которые можно обновлять одновременно, указать стратегию в случае сбоя (есть функция отката обновлений в случае сбоя). Функции Swarm можно расширять при помощи подключения плагинов. Графический интерфейс не поддерживается.

2.4.2 Kubernetes

Наиболее функциональной платформой оркестрации контейнеров на данный момент является Kubernetes. Kubernetes выступает как провайдер, позволяя предоставлять ресурсы на всех уровнях IaaS, PaaS, SaaS [18]. А также как платформа оркестрации, которая управляет ресурсами. Поддерживает все функции обслуживания ЖЦ ПО, в том числе масштабирование (как горизонтальное, так и вертикальное), а также мониторинг ресурсов и сервисов. Конфигурация развертывания задается через Kubernetes manifest. Kubernetes использует декларативный подход и одновременно выступает в роли инструмента конфигурации и оркестрации. Шаг со сборкой сервиса не требуется (т.к. оркестратор оперирует контейнерами, в которых уже содержаться собранный библиотеки и зависимости). Поэтому Kubernetes позволяет настроить конфигурацию работы между сервисами и полностью описать работу сервиса в различных случаях, например в различных состояниях. В каждый момент времени Kubernetes отслеживает состояние сервиса и принимает действие, например, в случае отказа. Состояния отслеживаются при помощи механизма "проверки работоспособности" сервиса (англ. Healthcheck). Kubernetes Можно интегрировать с

различными инструментами конфигурации, интеграции и поставки ПО. Минусом является отсутствие программ для графического редактирования топологий.

2.5 Сравнение Cloudify и Terraform

В статье [19] сравнивают производительность развертывания двух инструментов – Terraform (инструмент автоматизации) и Cloudify (оркестратор на базе стандарта TOSCA), а также рассматривают функции оркестраторов: OpenStack Heat, CloudFormation, и Cloud Assembly. Многие инструменты описаны поверхностно, авторы дают представление о функциональности, но не позволяют сделать сравнительный анализ и выявить сильные и слабые стороны инструментов.

Производительность сравнивают только для Cloudify и Terraform. Для эксперимента авторы предложили развернуть инфраструктуру (IaaS) и приложение Wordpress на ней (PaaS), т. к. Wordpress популярный вариант приложения. Было решено развернуть облачное приложение у 3-х различных провайдеров (Aws, Google cloud platform and Azure) под это требование из рассмотренных в статье оркестраторов подходят только Cloudify и Terraform.

На хост с оркестраторами были установлены сервисы мониторинга, которые следили за состоянием во время развертывания и удаления инфраструктуры. Авторы статьи использовали похожие конфигурации провайдеров. Однако, стоит заметить, что Terraform не оркестратор, и в данном случае выступает, как инструмент конфигурации. В то время как Cloudify - оркестратор, главным преимуществом которого является не работа "на скорость", а работа с несколькими облачными провайдерами и инструментами конфигурации. То есть само по себе сопоставление не является благоразумным. Авторы выбрали Ansible, как инструмент конфигурации для эксперимента, но не упомянули об этом. Стоит заметить, что Cloudify может работать и с Terraform и причины выбора Ansible авторы не называют. То есть в случае с Cloudify, неявно используется инструмент конфигурации, а Terraform работает напрямую с провайдером и очевидно, что результаты будут в худшую сторону для Cloudify. Эти результаты подтверждены графиками, но не отражают действительных преимуществ оркестратора. В заключении авторы делают упор на высокую производительность инструмента конфигурации Terraform и значительные преимущества в производительности. Однако эксперимент был проведен только в части развертывания и удаления сервисов. У Cloudify есть значительные преимущества - унифицированные шаблоны топологий, графический интерфейс для отображения топологии, возможность полномасштабной оркестрации (Terraform осуществляет оркестрацию только на IaaS уровне).

Авторы дают ссылку на шаблоны развертывания, которые они используют. По ним видно, какую проблему пытается решить стандарт TOSCA.

1) Оба инструмента развертывают одну и ту же инфраструктуру.
2) Для использования Terraform необходимо выучить синтаксис языка инструмента, а для использования Cloudify с несколькими провайдерами и инструментами конфигурации - достаточно научиться использовать только TOSCA.

С другой стороны, Cloudify использует свою интерпретацию стандарта TOSCA - TOSCA DSL. И поддерживает только ненормативные типы, которые добавляются в виде плагинов, что лишает преимуществ инструмент по сравнению с другими TOSCA Оркестраторами.

2.6 ToolKit Deployment Manager

В статье [20] рассматривается инструмент автоматизации развертывания (оркестратор) для различных дистрибутивов облачного провайдера Openstack на уровне IaaS.

Реализованный оркестратор должен отвечать требованиям по координации, автоматизации, подготовке и мониторингу ресурсов. Инструмент называется "ToolKit Deployment Manager" и состоит из набора блоков-контроллеров, которые отвечают за управлением (развертыванием и удалением): Сетью, Платформой, ПО, Сервисами.

Инструмент был реализован и проведено исследование эффективности оркестратора, которое наглядно показывает пользу. Процесс оркестрации проводился при помощи сценариев Развертывания Ansible. В качестве метрик для измерения возможности развертывания взяты:

- "среднее время развертывания" – время от вызова скрипта развертывания до перевода состояния как "запущено";
- "надежность развертывания" – отношение успешных развертываний к общему количеству запусков;
- "количество шагов развертывания" – общее количество шагов, требующееся для выполнения развертывания;
- "параметры, введенные во время развертывания" – столько раз пользователю необходимо вручную ввести параметры во время выполнения развертывания;
- "сумма шагов развертывания" – сумма количества шагов развертывания и количества вмешательств пользователя.
- "возможность развертывания" – итоговое количество усилий для успешного развертывания ((1-"Надежность развертывания") + 1) * "Сумма шагов развертывания".

Эти метрики действительно показывают эффективность внедрения инструментов оркестрации, т.к. они учитывают время, которое пользователь бы затратил без использования оркестратора. Также, необходимо добавить метрики, которые показывают количество артефактов развертывания, с которыми приходится работать. Т.к. версионирование, учет и написание новых сценариев занимает большое количество времени, а оркестратор помогает автоматизировать эти задачи и уменьшить количество сценариев.

Результаты, полученные в статье, демонстрируют сокращение количества ошибок развертывания и потраченного времени, что говорит об эффективном внедрении оркестратора.

3 Исследование и построение решения задачи

3.1 Сравнительный анализ методов построения облачных платформенных сервисов

Построение сервисов для виртуализации на базе гипервизора и на базе ОС (контейнеры) отличается. Для развертывания контейнеров не требуется развертывание уровня инфраструктуры. Kubernetes фактически стал стандартом в оркестрации и поставке сервисов уровня PaaS для контейнеров. Вопрос построения сервисов, развернутых на виртуальных машинах, остается открытым. Это связано с тем, что только провайдеры решают, предоставлять открытые интерфейсы для управления ресурсами в облаке или оставлять оркестрацию за собой. В виду этого, модули, которые поддерживаются инструментами конфигурации ограничены в возможностях. Например, ни Terraform, ни Ansible не могут управлять ресурсами в облаке Amazon также гибко, как и Cloud Formation. Именно поэтому, новейшие оркестраторы используют унифицированное описание топологий по стандарту TOSCA, которое помогает абстрактно описывать ресурсы и ЖЦ ПО, не привязываясь к реализациям провайдера. Кроме того, использование стандарта помогает настроить сообщение и работу между облачными провайдерами, а это безусловное преимущество TOSCA-оркестраторов.

Для развертывания виртуальных машин в облаке активно используются инструменты конфигурации, а также Terraform, однако это инструменты конфигурации можно считать устаревшими, ведь они не отвечают уровню, который необходим для оркестрации и масштабирования сложных микросервисных систем.

В табл. 1 видно, что оркестраторы в основном не предоставляют ресурсы, а лишь управляют ими (ведь это задача провайдера). Однако, на самом деле возможно построение такого оркестратора, который бы предоставлял платформенные сервисы по запросу. Примером реализации является Michman. Сложность построения кроется в том, что Michman "под капотом" развертывает и настраивает уровень инфраструктуры, который необходим для работы сервиса. Подобные возможности есть только у оркестраторов-провайдеров, таких как Kubernetes. Слабой стороной Michman является то, что нельзя запросить любую топологию, необходимую пользователю. Эту проблему можно решить, если представить уровень инфраструктуры в Michman через TOSCA. Такое представление уже реализовано в оркестраторе уровня инфраструктуры - Clouni.

Еще из данных табл. 1 можно понять, что гибкое и продвинутое управление ЖЦ ПО является основным преимуществом систем оркестрации. Эти функции также необходимо реализовать в эффективном оркестраторе.

Табл. 1. Таблица сравнения систем Оркестрации
Tab. 1. Competitive analysis of orchestration systems

Инструмент	Совместимость с провайдером	Возможность управления уровнем предоставления ресурсов	Уровень предоставления ресурсов	Имеется встроенный редактор топологий	Совместимость с инструментами конфигурации
Инструменты, позволяющие осуществлять оркестрацию					
Terraform	OpenStack, AWS, Azure, GCP, Kubernetes, etc	IaaS, PaaS	-	Нет	-
Ansible	OpenStack, AWS, Azure, GCP, Kubernetes, etc	IaaS, PaaS, SaaS	-	Нет	-
Оркестраторы с поддержкой одного облачного провайдера					
Cloud Formation	AWS	IaaS, PaaS, SaaS, FaaS	IaaS, PaaS, SaaS, FaaS	да	
Heat Orchestrator	OpenStack, AWS	IaaS	-	Нет	
Michman	OpenStack	PaaS	PaaS	Нет	Ansible
Оркестраторы нескольких облачных провайдеров					
xOpera	OpenStack, AWS, Azure, GCP, Kubernetes	IaaS, SaaS, FaaS	-	Нет	Ansible
Cloudify	OpenStack, AWS, Azure, GCP, Kubernetes, etc	IaaS, PaaS	-	Да	Ansible, Terraform, etc
Alien4Cloud + Yorc	OpenStack, AWS, GCP, Kubernetes, etc	IaaS	-	Да	Terraform
Clouni	OpenStack, AWS, Kubernetes	IaaS	-	Нет	Ansible

Оркестраторы контейнеров					
Swarm	Есть возможность поставки при помощи других провайдеров	PaaS	-	Нет	Ansible, Terraform, etc
Kubernetes	Выступает как провайдер + есть возможность поставки при помощи других провайдеров	IaaS, PaaS	IaaS, PaaS	Нет	Ansible, Terraform, etc

Табл. 1. Продолжение 1
Table 1. Continuation 1

Инструмент	**Автоматический мониторинг**	**Валидация зависимостей**	**Использует ли TOSCA?**	**Автоматическое отслеживание состояния единиц развертывания**	**Возможность автоматического создания резервной копии и восстановления из нее**
Инструменты, позволяющие осуществлять оркестрацию					
Terraform	Нет	Нет	Нет	Нет	Да
Ansible	Нет	Нет	Нет	Нет	Да (не авт.-е)
Оркестраторы с поддержкой одного облачного провайдера					
Cloud Formation	Да	Нет	Нет	Да	Да
Heat Orchestrator	Нет	Нет	Да	Нет	Нет
Michman	Нет	Да	Планируется переход	Да	Нет
Оркестраторы нескольких облачных провайдеров					
xOpera	Нет	Нет	Да	Нет	
Cloudify	Да	Нет	Да	Да	
Alien4Cloud + Yorc	Да	Нет	Да	Да	
Clouni	Нет	Да	Да	Нет	
Оркестраторы контейнеров					
Swarm	Нет	Нет	Нет	Да	Нет
Kubernetes	Да	Нет	Нет	Да	Да

Табл. 1. Продолжение 2
Table 1. Continuation 2

Инструмент	**Возможность автоматического возврата ПО к предыдущей версии**	**Возможность автоматического масштабирования**	**Возможность развертывание изменений для части сервисов**
Инструменты, позволяющие осуществлять оркестрацию			
Terraform	Нет	Нет	Нет
Ansible	Да (не авт.-е)	Да (не авт.-е)	Да (не авт.-е)
Оркестраторы с поддержкой одного облачного провайдера			
Cloud Formation	Да	Да	Да
Heat Orchestrator	Нет	Нет	Нет
Michman	Нет	Да	Нет
Оркестраторы нескольких облачных провайдеров			
xOpera	Нет	Нет	Нет
Cloudify	Да	Да	Да
Alien4Cloud + Yorc	Нет	Нет	Нет
Clouni	Нет	Нет	Нет
Оркестраторы контейнеров			
Swarm	Да	Да	Нет
Kubernetes	Да	Да	Да

3.2 Проект решения

Далее будет описано решение по созданию единого оркестратора и будет описан требуемый функционал, но реализация не будет осуществлена в рамках работы.

Сравнительный анализ позволил выявить сильные и слабые стороны инструментов и функции, которые необходимо реализовать в оркестраторах, разработанных в ИСП РАН.

К тому же становится понятно, что не существует полнофункционального TOSCA оркестратора, в котором были бы полностью реализованы функции автоматизации.

Michman предоставляет сервисы PaaS по запросу как услугу. Однако для развертывания сервиса Michman использует заранее подготовленные скрипты развертывания инфраструктуры, которую нельзя дополнить и гибко настроить. Michman может развернуть один мастер узел и неограниченное количество узлов с рабочей нагрузкой в кластере. Во время удаления сервиса, Michman удаляет кластер целиком по причине того, что развертывание инфраструктуры и сервисов сильно связаны в инструменте через конфигурационные переменные. Этап создания инфраструктуры и этап развертывания

сервисов необходимо разделить. Тогда можно будет масштабировать инфраструктуру и сервисы не зависимо. Также необходимо реализовать поэтапное и раздельное развертывание и удаление.

Clouni позволяет унифицировано описывать единожды TOSCA template и развертывать инфраструктуру для нескольких облачных провайдеров. Однако в Clouni не реализована возможность предоставлять сервисы по запросу.

Выгодное решение – соединить оба оркестратора для совместной и эффективной работы.

Проектируемый TOSCA оркестратор должен принимать на вход унифицированный шаблон описания топологии инфраструктуры и сервисов. На выходе пользователь должен получить уже развернутую инфраструктуру или сервис в выбранном облачном провайдере с возможностью дальнейшей оркестрации и отслеживания статуса.

Функции оркестратора:

- графическое моделирование топологии;
- создание, удаление, обновление топологии;
- создание, удаление, обновление инфраструктуры для заданной топологии;
- отслеживание состояния единицы развертывания для заданной топологии;
- добавление нового провайдера;
- добавление добавление поддержки нового сервиса;
- валидация топологии и зависимостей в ней;
- автоматическое создание резервной копии единицы развертывания и восстановление из копии;
- автоматический возврат единицы развертывания к предыдущей версии;
- автоматическое масштабирование единицы развертывания;
- развертывание изменений на часть единиц развертывания.

Новый оркестратор будет ориентирован прежде всего на OpenStack и Ansible, однако в процессе проектирования необходимо закладывать возможность добавления поддерживаемых провайдеров и инструментов конфигурации.

В конечной реализации должны использоваться OpenStak-специфичные типы. Состояние сервисов должно быть реализовано в соответствии с дополнением к стандарту TOSCA Instance model [21]. Механизм подстановки параметров, необходимых для получения сценария для заданного инструмента конфигурации должен быть реализован через TOSCA substitute. После реализации этих требований можно будет реализовать механизм мониторинга

4 Заключение

В ходе описанной работы были решены следующие задачи:

- исследованы отличия оркестрации виртуальных машин с сервисами от оркестрации контейнеров;
- исследованы инструменты оркестрации контейнеров;
- исследованы инструменты конфигурации;
- исследованы механизмы оркестрации крупных облачных провайдеров;
- исследованы оркестраторы на базе унифицированного стандарта TOSCA;
- исследованы оркестраторы, разработанные в ИСП РАН;
- проведен сравнительный анализ методов построения облачных платформенных сервисов;
- предложен проект решения по объединению работы оркестраторов ИСП РАН.

Исходя из результатов работы были выявлены функциональные требования, необходимые для реализации оркестратора. В работе описан проект решения по объединению работы оркестраторов, разрабатываемых в ИСП РАН. Проект одобрен к реализации.

Реализация оркестратора, расширяемого путем добавления поддержки новых провайдеров и инструментов конфигурации, возможна при помощи стандарта TOSCA. Также, были проанализированы метрики эффективности внедрения инструментов оркестрации, которые можно будет использовать в дальнейшем.

Список литературы / References

[1] NIST Special Publication 500-332. The NIST Cloud Federation Reference Architecture. Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-332.pdf.
[2] Topology and Orchestration Specification for Cloud Applications Version 1.0. OASIS Standard, 2013. Available at: http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html.
[3] Bilgin I. Multi-Runtime Microservices Architecture. 2022. Available at: https://www.infoq.com/articles/multi-runtime-microservice-architecture/.
[4] Terraform About the Docs. 2022. Available at: https://www.terraform.io/docs.
[5] Red Hat Ansible Automation Platform. 2022. Available at: https://www.ansible.com/.
[6] Chef Documentation. 2022. Available at: https://docs.chef.io/.
[7] Puppet. 2022. Available at: https://puppet.com/.
[8] Ansible Galaxy. 2022. Available at: https://galaxy.ansible.com/.
[9] AWS CloudFormation. 2022. Available at: https://aws.amazon.com/ru/cloudformation/.
[10] Heat documentation. 2022. Available at: https://docs.openstack.org/heat/latest/.
[11] Michman. 2022. Available at: https://github.com/ispras/michman.
[12] Openstack. 2022. Available at: https://www.openstack.org/.
[13] Cloudify. 2022. Available at: https://github.com/cloudify-cosmo.
[14] X-opera. 2022. Available at: https://github.com/xlab-si/xopera-opera.
[15] Clouni TOSCA orchestrator. 2022. Available at: https://github.com/ispras/clouni.
[16] Alien4Cloud. 2022. Available at: http://alien4cloud.github.io.
[17] Yorc. 2022. Available at: https://github.com/ystia/yorc.
[18] Kubernetes Documentation. 2022. Available at: https://kubernetes.io/docs/home/.
[19] de Carvalho L.R., de Araujo A.P.F. Performance Comparison of Terraform and Cloudify as Multicloud Orchestrators. In Proc. of the 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), 2020, pp. 380-389.
[20] Qadeer A., Malik A. W. et al. Virtual Infrastructure Orchestration for Cloud Service Deployment, The Computer Journal, vol. 63, issue 1, 2020, pp. 295-307.
[21] Instance Model for TOSCA Version 1.0. Available at: http://docs.oasis-open.org/tosca/TOSCA-Instance-Model/v1.0/TOSCA-Instance-Model-v1.0.html. 2013.

Информация об авторах / Information about authors

Александра Андреевна БОРИСОВА – студентка магистратуры НИУ ВШЭ и старший лаборант отдела информационных систем ИСП РАН. Сфера научных интересов: облачные технологии и разработка ПО.

Alexandra Andreevna BORISOVA – student at the Higher School of Economics, research assistant of the Department of Information System of the Institute for System Programming of the RAS since 2020. Research interests: cloud technologies and software development.

Олег Дмитриевич БОРИСЕНКО – научный сотрудник отдела информационных систем и руководитель команды облачных технологий. Сфера научных интересов: облачные технологии и разработка ПО.

Oleg Dmitrievich BORISENKO is a specialist and team leader of the Department of Information System. Research interests: cloud technologies and software development.