DOI: 10.15514/ISPRAS-2022-34(4)-15

Построение требований и архитектуры облачного оркестратора платформенных сервисов

Н.А. Лазарев, ORCID: 0000-0002-1008-1022 <lazarevn@ispras.ru>
О.Д. Борисенко, ORCID: 0000-0001-8297-5861 <al@somestuff.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Облачные технологии предоставляют пользователям простое и надежное масштабирование ресурсов, за счет чего они получили широкое распространение. На сегодняшний день особенно актуальна задача управления распределенными службами в облачной среде. Для этого используются специальные программы “оркестраторы”, которые реализуют функции управления жизненным циклом приложений. Однако существующие решения имеют множество ограничений и неприменимы в общем случае. Кроме того, не существует единого стандарта или протокола для взаимодействия с такими инструментами, из-за чего требуется адаптация программ для каждого частного случая. Основными задачами этой статьи являются выявление требований к оркестратору платформенного уровня облачных вычислений (PaaS), а также предложение подходов к построению гибкой архитектуры инструментов этого класса.

Ключевые слова: облачные вычисления; оркестрация; PaaS; распределенные системы; TOSCA; OCCI

Для цитирования: Лазарев Н.А., Борисенко О.Д. Построение требований и архитектуры облачного оркестратора платформенных сервисов. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 211-228 DOI: 10.15514/ISPRAS-2022-34(4)-15

Requirements and architecture design for cloud PaaS orchestrator

N.A. Lazarev, ORCID: 0000-0002-1008-1022 <lazarevn@ispras.ru>
O.D. Borisenko, ORCID: 0000-0001-8297-5861 <al@somestuff.ru>
Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Cloud technologies provide abilities for simple and reliable scaling of resources, due to which they have become widespread. The task of managing distributed services in a cloud environment is especially relevant today. Special programs are used for that purpose named “orchestrators” which implement the functions of lifecycle management for applications. However, the existing solutions have many limitations and are not applicable in the general case. Also there is no single standard or protocol for interaction with such tools which requires adaptation of programs for each particular case. The main objectives of this paper are to identify the requirements for a platform-level cloud computing (PaaS) orchestrator, as well as to propose flexible architecture patterns for such tools.

Keywords: cloud computing; orchestration; PaaS; distributed systems; TOSCA; OCCI

For citation: Lazarev N.A., Borisenko O.D. Requirements and architecture design for cloud PaaS orchestrator. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 211-228 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-15

1. Введение

Постоянное развитие технологий производства компьютерной техники приводит к её заметному удешевлению, что позволяет строить все большие вычислительные центры. В связи с этим получили толчок в развитии облачные вычисления – технология предоставления доступа к вычислительным ресурсам посредством сети Интернет, абстрагирующая пользователей от физических устройств. Согласно определению NIST [1], облачные вычисления подразделяются на три базовых уровня:

- инфраструктурный (IaaS) – предоставление виртуальных машин, блочных устройств и настройка сетевого взаимодействия;
- платформенный (PaaS) – размещение на ресурсах провайдера облачных услуг библиотек, сервисов и инструментов, поддерживаемых провайдером, для дальнейшего использования пользователем в своих программах без прямого управления нижележащей инфраструктурой;
- сервисный (SaaS) – обеспечение доступа пользователя к конечным приложениям, расположенным на ресурсах облачного провайдера.

Кроме того, облачные платформы подразделяются по признаку доступности ресурсов: публичные облака предоставляют услуги для любых пользователей, а приватные работают в рамках некоторой организации.

Для каждого уровня предоставления услуг, как правило, требуется специальное программное обеспечения для управления соответствующими ресурсами, все чаще называемое оркестраторами [2]. На сегодняшний день достаточно полно описан уровень IaaS[3]: определены предоставляемые ресурсы и способы предоставления доступа к ним. Для более высоких уровней существует множество реализаций, имеющих различные сферы применения и наборы интерфейсов, которые современные стандарты не покрывают. Эта статья посвящена проблематике разработки PaaS-оркестратора: определению актуальных задач, требуемого функционала и необходимых компонентов для применимости как для развертывания заранее подготовленных сервисов, так и описания пользователями сценариев развертывания собственного программного обеспечения, например, в процессе разработки.

Под PaaS-оркестратором здесь понимается программное обеспечение, обладающее следующей базовой функциональностью:

- развертывание ресурсов по запросу;
- абстрагирование пользователей от управления виртуальными машинами, сетями и дисками;
- возможность развертывания сервисов на виртуальных машинах без использования технологий контейнеризации;

С логической точки зрения, это программное обеспечение явным образом отделено от инфраструктурного слоя и использует доступные вызовы для получения инфраструктурных ресурсов облачного провайдера (виртуальных машин или контейнеров; программно-определяемых сетей и хранения). Сама же работа по управлению распределенными системами происходит уже не при помощи IaaS API облачного провайдера, а на уровне управления и настройки ОС и программных компонентов в экземплярах виртуальных машин или контейнеров.

Остальная статья организована следующим образом. Во втором разделе рассмотрены вычислительные задачи из различных сфер, рассмотрены основные преимущества контейнерных технологий, на основе чего сформулированы требования к PaaS-оркестратору. В третьем разделе проведен обзор существующих инструментов оркестрации, соответствующих рассматриваемой области, а также описаны применяемые технологии. Далее, в четвертом разделе предложена общая архитектура оркестратора исходя из сформулированных требований и рассмотренных решений. Наконец, в пятом разделе подведены итоги и сформулированы направления для дальнейшей работы.

2. Анализ применимости

Несмотря на наличие множества инструментов, называемых оркестраторами, на сегодняшний день не устоялось ни однозначного определения данного термина, ни даже функционала, который должен предоставлять тот или иной оркестратор. Из этого также вытекает отсутствие однозначных критериев сравнения и оценки таких инструментов. Зачастую каждый автор выбирает интересующий его набор критериев, который формируется на основе некоторых конкретных задач. Например, в сравнении TOSCA-оркестраторов [4], наиболее значимыми критериями сравнения являются поддерживаемая версия стандарта TOSCA и поддерживаемые облачные платформы, но наряду с ними оцениваются также год выпуска, поддерживаемые платформы (операционные системы), сложность установки и другие параметры, не отражающие функциональности инструментов. В другой статье [5], посвященной сравнению облачных оркестраторов, достаточно подробно рассматриваются аспекты, касающиеся работы оркестраторов в облачной среде, однако не рассматриваются способы обработки пользовательских данных и доступа к предоставляемым сервисам.

Одной из основных целей данной работы является формирование универсальных требований к инструментам оркестрации, которые позволили бы сравнивать их с точки зрения функциональности.

2.1 Актуальные задачи

На данный момент границы PaaS сильно размыты, так как набор услуг для работы конечных приложений сильно варьируется. Рассмотрим несколько сценариев использования оркестраторов в различных сферах.

С распространением удаленного обучения стремительно возрос спрос на обеспечение виртуальных учебных мест. Облачные платформы позволяют удовлетворить этот спрос различными способами: предоставляя сервисы для видеоконференций, обмена файлами, проверки заданий, либо предоставляя сами вычислительные ресурсы напрямую. Однако, существуют случаи, когда необходимо предоставить доступ конечных пользователей, например, студентов, непосредственно к некоторой платформе для получения навыков работы с ней. Примером может служить обучение работе с HPC-платформами или написание программ, ориентированных на работу с GPU. Помимо развертывания таких платформ, требуется также оркестрация инструментов взаимодействия с ними: от установки и конфигурации компиляторов до развертывания дополнительных сервисов.

Сегодня облачные вычисления применяются в естественных и гуманитарных науках[6], в том числе: физике и астрофизике, биоинформатике и медицине, социальных и других гуманитарных науках, науках о Земле и окружающей среде, химии (хемоинформатике).

В некоторых направлениях успели сложиться свои традиции и популярные наборы программных инструментов, в то время как другие начинали формироваться в последние несколько лет. Однако, в большинстве случаев справедливы следующие общие проблемы [7,8,9,10,11].

1) *Необходимость хранения и обработки больших объемов данных.* Современные задачи ориентированы на построение крупных и сложных моделей, в связи с чем возрастают требования на хранимые данные. Данные могут быть получены из открытых источников, собраны с помощью дополнительного оборудования, сгенерированы, а также получены от экспертов. Кроме того, в некоторых случаях требуется дополнительная предобработка данных: анонимизация и шифрование, устранение дубликатов и некорректных записей, а также другая подготовка к вычислениям. Таким образом, необходимо обеспечение как достаточного объема хранилища, так и возможность интеграции со сторонними программными и аппаратными средствами.
2) *Организация параллельных вычислений.* Другое следствие роста объемов данных – нелинейное повышение времени работы алгоритмов их обработки. В то время как

возможности вертикального масштабирования вычислительных ресурсов ограничены и требуют все больших затрат, современные платформы параллельной обработки данных на большом числе вычислительных узлов позволяют достигать повышения производительности программ без использования дорогостоящего оборудования. Кроме того, многие современные программные средства позволяют использовать графические ускорители для повышения эффективности программ. Хотя облачная среда позволяет автоматически создавать и настраивать виртуальные машины, особенно актуальна задача автоматизации настройки сложных распределенных систем.

3) *Массовое применение методов машинного обучения.* Наряду с задачами использования вычислительных ресурсов свои требования накладывает развитие методов обработки данных. Методы машинного обучения также являются ресурсоемкими и чувствительны к обеспечению горизонтального масштабирования. Эта область постоянно развивается, в связи с чем набор инструментов, используемых в ML-платформах постоянно растет, в связи с чем необходимо наличие механизма постоянного обновления, используемого в конкретном решении используемого инструментария. Наконец, использование машинного обучения и нейронных сетей тесно связано с построением нелинейного рабочего процесса, требующего постоянного обновления данных и взаимодействия с пользователями.
4) *Потребность в обмене и организации общего доступа к данным.* Современные задачи все чаще требуют участия множества специалистов из различных областей. В связи с этим требуются дополнительные службы, упрощающие взаимодействие между различными пользователями систем.
5) *Обработка ошибок.* Вместе с повышением сложности вычислительных систем все критичнее становится необходимость в автоматическом обнаружении и исправлении ошибок, а также сборе информации об ошибках. Для этих нужд необходима сквозная интеграция систем мониторинга и сбора логов, учитывающая специфику отдельных приложений.

Еще одной важной сферой для анализа задач оркестрации является коммерческая разработка приложений. С одной стороны, в современном бизнесе все чаще используют те же технологии, что и в науке, в том числе, машинное обучение и анализ больших данных. С другой стороны, в процессе коммерческого использования приложений критичны такие параметры, как время ответа пользователю и время недоступности сервиса, в то время как в науке и образовании этому уделяют меньше внимания. В связи с этим, например, задача масштабирования усложняется тем, что необходимо поддерживать постоянную доступность приложения для конечных пользователей. Также возникает явная необходимость обеспечения балансировки нагрузки, обеспечения корректного обновления системы.

2.2 Контейнерная оркестрация

Фактически, стандартным решением для построения коммерческих приложений стало использование контейнеров – легковесной виртуализации на уровне операционной системы и соответствующих оркестраторов, так как они обладают рядом преимуществ при необходимости обеспечения бесперебойной работы:

- легковесность контейнеров относительно виртуальных машин;
- образы контейнеров содержат все необходимое для запуска приложения;
- прозрачное версионирование приложения;
- абстракция от физических ресурсов и операционной системы при запуске;
- простое создание идентичных экземпляров приложения;
- предсказуемое поведение приложения при запуске.

Перечисленные свойства позволяют полностью изолировать процесс развертывания распределенных приложений от внутреннего устройства отдельных его компонент и крайне важны с точки зрения моделирования и оркестрации сложных систем.

Отдельно стоит отметить устоявшийся процесс разработки и развертывания прикладных систем при использовании контейнерных технологий:

1) независимая разработка отдельных компонентов системы;
2) написание файла, описывающего процесс создания образа для отдельных компонентов системы: Containerfile или Dockerfile;
3) создание и загрузка образа в приватный или публичный репозиторий контейнеров;
4) описание конфигураций запуска отдельных компонентов при развертывании всей системы, а также описание взаимодействия компонентов между собой;
5) развертывание всей системы;
6) обновление рабочей системы.

Для оркестрации контейнерных приложений существует множество инструментов, например, Docker Compose, Docker Swarm, Apache Mesos, Kubernetes. При этом выбор оркестратора зависит от масштаба системы [12].

На сегодняшний день наиболее популярным инструментом для оркестрации больших программных систем является Kubernetes. В контексте анализа задач оркестрации стоит отметить ключевой функционал данной платформы:

- **устойчивое развертывание**: формат описания сервисов в Kubernetes позволяет декларативно описывать желаемое состояние. Этот механизм позволяет контролируемо обновлять конфигурацию приложения и переводить на нее нагрузку только после того, как изменения полностью вступили в силу;
- **мониторинг**: контроль за работой сервисов как на уровне статуса всего контейнера, так и за счет выполнения указанных команд внутри контейнера;
- **масштабирование**: в процессе работы приложения Kubernetes позволяет в зависимости от актуальной нагрузки увеличивать или уменьшать количество контейнеров, соответствующих каждому сервису: автоматически или при помощи операторов;
- **конфигурация хранилища**: пользователь может настраивать как локальную файловую систему для своих сервисов, так и подключать сторонние хранилища, доступные по сети;
- **распределение нагрузки**: распределение трафика между контейнерами за счет анализа количества запросов по адресу каждого из контейнеров;
- **контроль за конфиденциальной информацией**: обеспечивается сохранность пользовательских и системных аутентификационных данных;

С другой стороны, использование контейнерной виртуализации накладывает ограничения на работу программ в части взаимодействия с аппаратной частью и хостовой операционной системой, а также имеют недостатки с точки зрения обеспечения безопасности. Кроме того, предполагаемый функционал работы PaaS-оркестратора шире, чем предполагаются при работе с контейнерными оркестраторами. Рассмотрим подробнее, какие типы платформ возникают при оркестрации уровня PaaS на основе приведенного обзора задач.

2.3 Определение функциональности оркестратора

Базовой функциональностью, предоставляемой разработчикам приложений, является среда сборки и выполнения программ. В данный уровень платформ входят компиляторы для различных языков программирования, настройка (возможно, виртуальных) окружений, обеспечение изоляции ресурсов. Отдельно стоит выделить настройку контейнерной среды исполнения, причем на данном уровне имеется в виду непосредственно инструменты для запуска контейнеров без дополнительного окружения. Используя такое окружение разработчики приложений (в том числе, ориентированных на запуск в контейнерном окружении) получают полноценную абстракцию как от физических ресурсов, так и от

операционных систем и необходимости установки дополнительных пакетов, имеют стабильное окружение для сборки и запуска собственных проектов. Аналогом данной абстракции является раздел FROM при описании образа контейнера в формате Dockerfile. Заметим, что обеспечение только этого уровня подразумевает необходимость обеспечения непосредственного доступа пользователей к виртуальным машинам с настроенным окружением посредством ssh или других способов.

Другим обязательным уровнем PaaS является обеспечение взаимодействия с различными инструментами хранения данных. Помимо подключения (виртуальных) блочных устройств к пользовательским виртуальным машинам, может потребоваться создание распределенной файловой системы или объектного хранилища, либо монтирование уже существующих хранилищ. За счет такой функциональности разработчики получают абстракцию для управления данными в различных форматах, сравнимую с подключением томов к контейнерам. При этом подразумевается лишь обеспечение доступа пользователей к данным инструментам по сети, в то время как вся логика взаимодействия с хранилищем остается вне оркестратора.

Наконец, наиболее сложным уровнем функциональности PaaS является обеспечение работы с различными службами, предоставляющими программные интерфейсы по сети. Частым примером для данного уровня является развертывание СУБД по запросу, хотя зачастую его относят к уровню хранилищ. Другими примерами для данного уровня могут являться: инструменты для сбора и обработки логов, мониторинга, веб-разработки, платформы параллельного выполнения программ и планировщики использования ресурсов, а также платформы оркестрации контейнеров. В отличие от контейнерного окружения, в зависимости от платформы, её конфигурации, а также зависимостей могут потребоваться дополнительные вычислительные ресурсы, установка дополнительных пакетов, настройка других инструментов и другие действия. Тем не менее, такой функционал обеспечивает уровень абстракции, аналогичный оркестрации контейнеров: пользователи PaaS-оркестратора получают полноценную возможность использовать сторонние платформы при разработке собственных инструментов.

При этом важно заметить, что аналогично уровню хранения данных, при развертывании высокоуровневых платформ задача оркестратора состоит в обеспечении доступа пользователя к тому или иному инструменту, в то время как любая бизнес-логика требует участия пользователя. Например, рассмотрим задачу предоставления реляционной СУБД по запросу. В простейшем случае эта задача подразумевает создание виртуальной машины достаточной мощности и подключения к ней блочного (виртуального) устройства достаточного объема, установки необходимых для запуска СУБД пакетов, настройки параметров виртуальной машины и самой СУБД, создания пользователя СУБД, а также настройки сетевого файервола для обеспечения доступа. Также, в зависимости от конфигурации всей системы, может потребоваться дополнительная настройка на уровне исполнения программ. В конечном итоге пользователь получает адрес СУБД, а также аутентификационные данные. Однако, в некоторых случаях может потребоваться создание связанных кластеров реляционных СУБД, состоящих из нескольких узлов. В этом случае необходимо обозначить границы возможностей оркестратора: любые манипуляции, для которых необходима информация о схеме базы данных может быть настроена только самим пользователем. Так, оркестратор может настроить кластер PostgreSQL с полной репликацией данных, однако для распределения данных между узлами необходимо указывать параметр распределения в каждой таблице базы данных, поэтому без участия пользователя это сделать невозможно.

Еще одним важным слоем является обеспечение масштабируемости распределенных систем. Использование распределенных систем может диктоваться как требованиями по соблюдению того или иного уровня соглашения о предоставлении услуг (SLA), требованиями к качеству и отзывчивости систем (QoS), а также тем, что для обеспечения

работоспособности программы не хватает физических ресурсов самих инфраструктурных узлов, и программа в принципе не может быть запущена на одном вычислительном узле. Все перечисленные причины к использованию распределенных систем не накладывают специфических требований к оркестратору, поскольку SLA, требования QoS относятся больше к бизнес-преимуществам компаний-провайдеров облачных услуг, однако явным образом добавляют требование на возможность настройки сложных с точки зрения сетевой топологии программных систем, которые требуют корректного распределения конфигурационных файлов, учитывающего порядок запуска каждого из компонентов и их взаимосвязи, а также контроль версий входящих в состав компонентов.

Контроль версий включает в себя множество вспомогательных задач: от учета версий пакетов, устанавливаемых на виртуальные машины, до отслеживания совместимости различных интерфейсов между собой. Так, при развертывании Apache Spark может потребоваться также настройка других инструментов, например, Apache Hadoop. При этом для Apache Spark версии 3.0.0 и выше возможна совместимость только с версиями Apache Hadoop не меньше 2.2.0. Кроме того, официальные сборки доступны только для ограниченного числа комбинаций версий данных платформ, в то время как при необходимости развертывания других потребуется дополнительно обеспечить сборку проектов. Также стоит отметить, что для работы данных платформ требуется настройка JVM определенной версии. В случае, если пользователю потребуется также JVM другой версии, необходима изоляция времени исполнения между платформой и пользовательскими программами. Таким образом, при оркестрации требуется поддерживать многоуровневый контроль совместимости различных компонентов.

Таким образом, на основе проведенного анализа, а также с учетом других работ, можно сформулировать следующий набор функциональных возможностей PaaS-оркестратора для покрытия актуальных задач:

1) механизм описания предоставляемых услуг, учитывающий уровни:
   a) предоставления среды исполнения;
   b) настройки хранилищ данных;
   c) предоставления программных интерфейсов по сети;
2) обобщение пакетных менеджеров уровня операционной системы с возможностью обновления программного обеспечения и контролем совместимости поставляемых пакетов;
3) управление конфигурационными файлами и их специализация под версии развертываемых программных систем, в том числе подстановка параметров;
4) сквозная доставка пользовательских данных;
5) интеграция систем мониторинга для обеспечения отказоустойчивости и масштабирования;
6) обеспечение доступа к развернутым сервисам посредством ssh или инструментов аутентификации в соответствии с уровнем предоставляемых услуг;
7) обеспечение балансировки нагрузки.

Кроме того, в зависимости от контекста использования оркестратора требуется обеспечение программного или пользовательского интерфейса с разграничением доступа к развернутым сервисам и действиям с ними. Также в современных оркестраторах требуется предусмотреть возможность мультиоблачного режима работы: в рамках построения гетерогенной гибридной облачной среды (при одновременном использовании публичных и приватных вычислительных ресурсов), при работе в рамках географически распределенной облачной среды, либо для унифицированного использования ресурсов независимых облачных провайдеров.

3. Обзор оркестраторов

В данном разделе рассматриваются существующие решения, относящиеся к оркестрации уровня PaaS. Так как основной целью работы является определение архитектуры и таксономии мультиоблачных оркестраторов, коммерческие разработки будут рассматриваться только с точки зрения предоставляемого функционала. В свою очередь, основной интерес для текущего исследования представляют открытые оркестраторы, позволяющие решать задачи, описанные во второй главе. Общее сравнение рассматриваемых оркестраторов приведено в табл. 1.

Табл. 1. Сравнение существующих оркестраторов
Tabl. 1. Existing orchestrators comparison

Критерий	**INDIGO-DC**	**yorc**	**Cloudify**	**Heroku**	**ElastiCluster**	**Michman**
Механизм описания платформ	TOSCA Simple Profile v1.0	TOSCA Simple Profile v1.2	DSL на основе TOSCA Simple Profile v1.3	Heroku Buildpack на основе git	Нет	Собственный язык в формате JSON
Управление пакетами	Задается в шаблонах	Задается в шаблонах	Задается в шаблонах	Обработка зависимостей	Задается в Ansible	Подключение репозиториев
Управление конфигурациями	Задается в шаблонах	Задается в шаблонах	Задается в шаблонах	Обработка переменных окружения	Задается в Ansible	Задается в описании сервисов
Доставка пользовательских данных	Задается в шаблонах	Задается в шаблонах	Задается в шаблонах	Нет	Задается в Ansible	Нет
Мониторинг	zabbix	consul	Плагины	Дополнения	Нет	consul
Доступ к ресурсам	Выделенный сервис	ssh	ssh, подключение к ldap	ssh	ssh	ssh
Балансировка нагрузки	Нет	Нет	Плагины	Встроенная маршрутизация	Нет	Нет

3.1 Формат описания предоставляемых услуг

В соответствии с предложенными критериями, в первую очередь необходимо рассмотреть возможность описания предоставляемых услуг. Хотя для конечных пользователей формат описания зачастую скрыт за счет использования какого-либо пользовательского интерфейса, используемое внутреннее представление оркестратора определяет как технологические возможности оркестратора, так и удобство его интеграции. Сегодня существует множество форматов и языков описания распределенных систем, имеющих различные области применимости, модель предоставления услуг и назначение[13].

Наибольший интерес среди них в рамках этой работы представляют независимые от облачной платформы языки и стандарты, позволяющие описать как топологию предоставляемых услуг уровня PaaS, так и процесс развертывания таких топологий. К таким можно отнести OCCI [14], mOSAIC [15], SOCCA [16], STRATOS [17], и другие, однако наиболее современным стандартом для описания топологий в облачной инфраструктуре любых уровней является TOSCA [18].

Основные сущности в данном стандарте делятся на два уровня: типы узлов и взаимоотношений, содержащие информацию о возможных компонентах топологий, а также различные шаблоны, которые конкретизируют описанные типы узлов и взаимосвязей и

описывают их объединение в топологии. TOSCA также определяет функционал инструментов, реализующих обработку документов, соответствующих стандарту:

- парсер (Parser): получает отдельные шаблоны и дополнительные “блоки” из одного или нескольких репозиториев, производит валидацию и нормализацию шаблонов;
- резолвер (Resolver): применяет входные данные для шаблонов, конвертирует нормализованные шаблоны в представление узлов, вызывает встроенные функции, а также вычисляет требования для узлов и создает граф взаимоотношений;
- оркестратор (Orchestrator): непрерывно создает или удаляет реализации для представления узлов, обновляет значения атрибутов узлов, обновляет результаты работы Resolver, а также может изменять представления узлов.

Отметим, что стандарт не ограничивает область применимости, в связи с чем не все TOSCA-оркестраторы могут быть использованы для управления ресурсами уровня PaaS. Поэтому, хотя многие инструменты заявляют о поддержке или реализации данного стандарта, в данной работе будут рассмотрены инструменты, получившие активное применение в научной и коммерческой среде.

- Indigo-DC [19] – облачная вычислительная платформа, предназначенная для научных сообществ и используемая в рамках европейского проекта Horizon 2020. В частности, в Indigo-DC реализуется управление ресурсами уровня PaaS;
- Yorc [20] – инструмент управления жизненным циклом приложений для работы в различных облачных платформах, а также в контейнерной среде Kubernetes, планировщике Slurm и при подключении к физическим серверам без виртуализации, который используется в рамках платформы Lexis [21];
- Cloudify [22] – мультиоблачная платформа оркестрации, ориентированная на автоматизацию развертывания программного обеспечения в процессе разработки. Использует собственный язык описания услуг, основанный на TOSCA.

Стандарт TOSCA позволяет связывать с различными сущностями специальные архивы, содержащие необходимые для развертывания скрипты, данные и прочие артефакты. Таким образом, настройка развертывания полностью лежит на разработчиках шаблонов топологий, в связи с чем снижается гибкость использования TOSCA-оркестраторов. Например, при необходимости добавления новой версии некоторого сервиса, пользователю необходимо либо самому добавить соответствующие артефакты к существующему описанию, либо запросить обновление у автора шаблона.

К минусам этого стандарта можно также отнести большое количество глав стандарта, находящихся в разработке, сложность использования, а также отсутствие соответствия описанных топологий конкретным ресурсам в облачной среде.

В большинстве публичных облачных платформ существуют свои внутренние форматы описания предоставляемых услуг, например, шаблоны Azure Resource Manager (ARM templates [23]), шаблоны AWS CloudFormation [24], шаблоны Google Cloud [25]. Ключевым недостатком такого подхода является непереносимость описаний между различными облачными платформами. С другой стороны, такие форматы ориентированы на конкретные возможности облачных платформ и позволяют более точно описывать детали развертывания требуемых услуг. Такой подход используют также некоторые открытые инструменты оркестрации: в то время как стандарт TOSCA ориентирован на описание наиболее широкого набора облачных услуг, в существующих инструментах может предоставляться ограниченный функционал, за счет чего снижается сложность описания услуг и использования оркестратора. Среди таких оркестраторов можно выделить следующие.

- Heroku [26] – публичная PaaS платформа с открытым исходным кодом, ориентированная на настройку среды разработки в мультиоблачной среде на основе контейнерной виртуализации. Позволяет предоставлять сценарии развертывания в любых форматах с использованием репозиториев git;

- ElastiCluster [27] – инструмент командной строки для автоматизации управления вычислительными кластерами в облачной среде, используемый в проекте ATLAS [28]. В качестве средства описания развертывания систем используется Ansible[29];
- Michman [30] – PaaS оркестратор с открытым исходным, ориентированный на предоставление вычислительных платформ для научных исследований. На данный момент используется собственный формат описания предоставляемых услуг в формате JSON в совокупности с ролями Ansible.

3.2 Управление пакетами

В некоторых случаях может быть необходимо использовать специфичные пакеты при развертывании пользовательских платформ. Например, это может потребоваться при использовании модифицированных в целях безопасности или изменения функционала библиотек. Также, в рамках приватных облачных платформ может возникнуть необходимость установки пакетов без доступа в интернет – для этого необходимо подключать локальные для этой платформы репозитории, либо зеркала. Также как при оркестрации контейнерных окружений, в рамках PaaS-оркестратора необходимо обеспечить подключение пользовательских репозиториев.

Коммерческие публичные решения зачастую скрывают детали развертывания предоставляемых услуг, либо используют публичные репозитории без возможности подключения собственных репозиториев. С другой стороны, при определении собственного шаблона в каждой платформе возможно определение скриптов установки и настройки системы, что позволяет при необходимости указать необходимый способ установки пакетов при развертывании услуги из этого шаблона. В этом случае пользовательские репозитории также должны быть публичными.

В облачных оркестраторах, основанных на стандарте TOSCA, управление установкой пакетов зачастую реализуется разработчиками шаблонов топологий предоставляемых услуг. В случае использования оркестратора в рамках приватной облачной платформы, в шаблонах могут использоваться репозитории, развернутые в этой платформе. Недостатком такого подхода является недостаточный контроль за конфликтами между пакетами. Например, при установке двух платформ, использующих JVM различных версий, может потребоваться дополнительная настройка среды исполнения, что может приводить к непредвиденным ошибкам на этапе развертывания.

В мультиоблачных PaaS-оркестраторах, использующих собственный формат описания, зачастую применяется один из описанных выше подходов в зависимости от того, кем описывается процесс развертывания.

3.3 Управление конфигурационными файлами

Для большинства служб уровня PaaS требуется указание пользователями параметров, применяемых в процессе разворачивания, например, подключаемые плагины, квоты на использование вычислительных ресурсов, параметры авторизации и другие. В зависимости от используемого представления и пользовательских интерфейсов, оркестраторы могут предоставлять гибкую обработку любых параметров, описанных для каждой услуги, либо обеспечивать обработку только для заранее определенных параметров.

3.4 Доставка пользовательских данных

При разработке приложений, ориентированных на обработку больших объемов данных, требуется подключать сторонние системы хранения данных помимо настройки среды исполнения. В то время как публичные облачные платформы обеспечивают сквозную интеграцию с собственными объектными хранилищами, при работе в приватных облаках

могут использоваться открытые и закрытые объектные хранилища, подключаемые блочные хранилища, а также внешние распределенные файловые системы.

На текущий момент реализации стандарта TOSCA позволяют явно описывать только подключение блочных устройств к пользовательским виртуальным машинам. Альтернативно, разработчики шаблонов могут использовать фиктивное определение служб для реализации подключения к другим типам хранилищ, однако в этом случае теряется основное использование стандарта – независимость от облачных платформ и других инструментов.

В свою очередь, инструменты оркестрации, использующие другие форматы описания ресурсов, могут предоставлять сквозную интеграцию с системами хранения данных и монтирование внешних файловых систем.

3.5 Обеспечение отказоустойчивости и масштабирования

В отличие от контейнерной среды, при работе с виртуальными машинами оркестратор не всегда имеет прямой доступ к управляемым ресурсам и требует настройки дополнительных систем мониторинга. В зависимости от целей, мониторинг может осуществляться как на уровне операционной системы – например, через сбор метрик использования ресурсов системы, так и через специализированные запросы к предоставляемым сервисам – через сетевые запросы к ним.

Таким образом, требование получения информации о способах проверки жизнеспособности и загруженности ресурсов подразумевают явное или неявное использование систем мониторинга, а также способы описания проверок в формате описания.

В оркестраторах также могут использоваться встроенные системы анализа состояния управляемыми ресурсами.

3.6 Доступ к ресурсам

Важным аспектом предоставления ресурсов по сети является обеспечение безопасного доступа к ним. В случае предоставления платформы для разработки ПО основным способом доступа является подключение к виртуальным машинам посредством SSH, поэтому зачастую задача сводится к добавлению пользовательских публичных ключей на виртуальные машины.

С другой стороны, при настройке более высокоуровневых платформ требуется обеспечить безопасную доставку паролей при инициализации платформ, что особенно критично при работе в публичных облачных платформах.

3.7 Балансировка нагрузки

Основным преимуществом облачных платформ является гибкость в настройке вычислительных ресурсов. Однако, при горизонтальном масштабировании методом создания идентичных экземпляров служб возникает задача корректного распределения нагрузки между экземплярами. Такая функциональность может решаться различными способами: за счет использования балансировки на инфраструктурном уровне, с использованием специальных встроенных плагинов, а также за счет подключения дополнительных служб в разворачиваемых платформах.

4. Основные компоненты оркестратора

В этом разделе инструменты оркестрации подробнее рассматриваются с точки зрения входящих в их состав служб для выявления общих составляющих, необходимых для работы оркестратора. Также здесь приводится обобщение входящих в оркестраторы компонент и предлагается унифицированная архитектура.

4.1 Архитектура существующих оркестраторов

4.1.1 Indigo-DC

Архитектура Indigo-DC основывается на концепции микросервисов. В состав оркестратора входят следующие компоненты:

- сервис оркестрации (Orchestrator Service) – центральный компонент системы, обеспечивающий развертывание и управление ресурсами;
- сервис аутентификации (IAM Service) – обеспечивает проверку прав пользователей;
- графический интерфейс (GUI) – обеспечивает доступ пользователей к имеющимся шаблонам TOSCA и сервису оркестрации;
- репозиторий шаблонов TOSCA (Repository) – хранит шаблоны TOSCA;
- сервис оркестрации (Orchestrator Service) – центральный компонент системы, обеспечивающий развертывание и управление ресурсами;
- инфраструктурный менеджер (Infrastructure Manager) – разворачивает инфраструктурные ресурсы в различных облаках;
- сервисы управления данными (Data Management Services) – набор сервисов, предоставляющих доступ к хранилищам данных в унифицированном виде.
- служба мониторинга (Monitoring) – обеспечивает сбор метрик из различных облаков и передает в унифицированном виде в сервис оркестрации;
- служба обработки политик (Brokering/Policy Service) – обеспечивает контроль за политиками использования ресурсов;
- служба управления QoS и SLA (QoS/SLA Service) – обеспечивает контроль за QoS и SLA как для отдельных пользователей, так и для всей системы в целом.

4.1.2 Yorc

Оркестратор Yorc основан на модели приложения без сохранения состояния с целью упрощения горизонтального масштабирования. Основной функционал оркестратора реализован в виде одного сервиса, который управляет ходом развертывания ресурсов. К нему также быть подключены дополнительные плагины. Для взаимодействия с оркестратором реализованы сервис, предоставляющий REST API, и интерфейс командной строки. Также для работы с оркестратором может быть использован инструмент Alien4Cloud[31].

Кроме того, в Yorc используются сторонние сервисы: Consul[32] для управления сервисами и мониторинга и Vault[33] для хранения конфиденциальных данных.

4.1.3 Cloudify

Cloudify реализует клиент-серверный подход: основным компонентом является Cloudify Manager, который взаимодействует с агентами на управляемых виртуальных машинах. В свою очередь, Cloudify Manager включает следующие сервисы:

- REST API;
- база данных;
- брокер сообщений;
- сервис управления развертыванием.

Помимо использования REST API напрямую, в Cloudify также реализованы графический интерфейс и интерфейс командной строки. Также предоставляется доступ к логам и мониторингу.

4.1.4 Heroku

В основе работы Heroku используется принцип подготовки образов, готовых к запуску. Для взаимодействия с инструментом предоставляется API, CLI, а также могут использоваться

файлы настройки непосредственно в публичных репозиториях. Кроме того, в состав оркестратора входят службы для маршрутизации трафика и других сетевых настроек, логирования и мониторинга разворачиваемых сервисов, а также системы аутентификации и контроля потребления ресурсов.

Система развертывания, встроенная в Heroku основана на технологии контейнерной виртуализации, однако подразумевает сборку образа самим оркестратором из базовых образов операционных систем. Также для сборки и развертывания сервисов в Heroku используются службы для:

- учета пакетных зависимостей;
- правил сборки программ;
- настройки переменных окружения;
- подключения внешних служб;
- контроля версий сервисов.

4.1.5 ElastiCluster

ElastiCluster является инструментом командной строки, упрощающий использование Ansible для развертывания распределенных систем в облачной среде. Разработчиками предоставляются готовые сценарии для создания кластеров Slurm, Spark и Hadoop. Инструмент позволяет настраивать подключение к различным облачным провайдерам уровня IaaS, отслеживать текущее состояние и масштабировать развернутые кластеры, подключать и настраивать различные распределенные файловые системы.

4.1.6 Michman

Michman также использует микросервисную архитектуру, основными компонентами которой являются сервисы REST и LAUNCHER. REST обеспечивает доступ пользователей к описаниям поддерживаемых оркестратором сервсисов, а также к управлению разворачиваемым кластерам. В LAUNCHER реализована основная логика управления развертывания кластеров. Michman использует Consul для мониторинга, Vault для хранения конфиденциальных данных и LogStash для доступа к логам запуска кластеров. Также в оркестраторе реализовано подключение пакетных репозиториев для обеспечения режима работы без доступа в Интернет.

4.2 Анализ и обобщение рассмотренных оркестраторов

Рассмотренные выше оркестраторы используют различные подходы в организации работы, так как рассчитаны на разные области применимости: от упрощения ручной настройки распределенных вычислительных кластеров до использования в публичных облаках. Оркестраторы, основанные на микросервисной архитектуре, могут быть явным образом разделены на компоненты, имеющие четкие границы применимости. Для монолитных инструментов выделение компонентов предполагает лишь логическое разделение. При этом некоторые компоненты находят реализацию в каждом инструменте в том или ином виде. Основные компоненты и их взаимодействие представлены на рис.1.

В первую очередь, в каждом рассмотренном инструменте используется некоторая абстракция управления вычислительными ресурсами инфраструктурного уровня, в связи с чем может быть выделен соответствующий компонент – IaaS manager. В качестве этого компонента может использоваться интерфейс некоторой облачной платформы, сторонний IaaS-оркестратор, либо встроенный программный компонент. Также в некоторых случаях в качестве альтернативы инфраструктурному уровню может использоваться контейнерная среда, либо физические узлы без виртуализации.

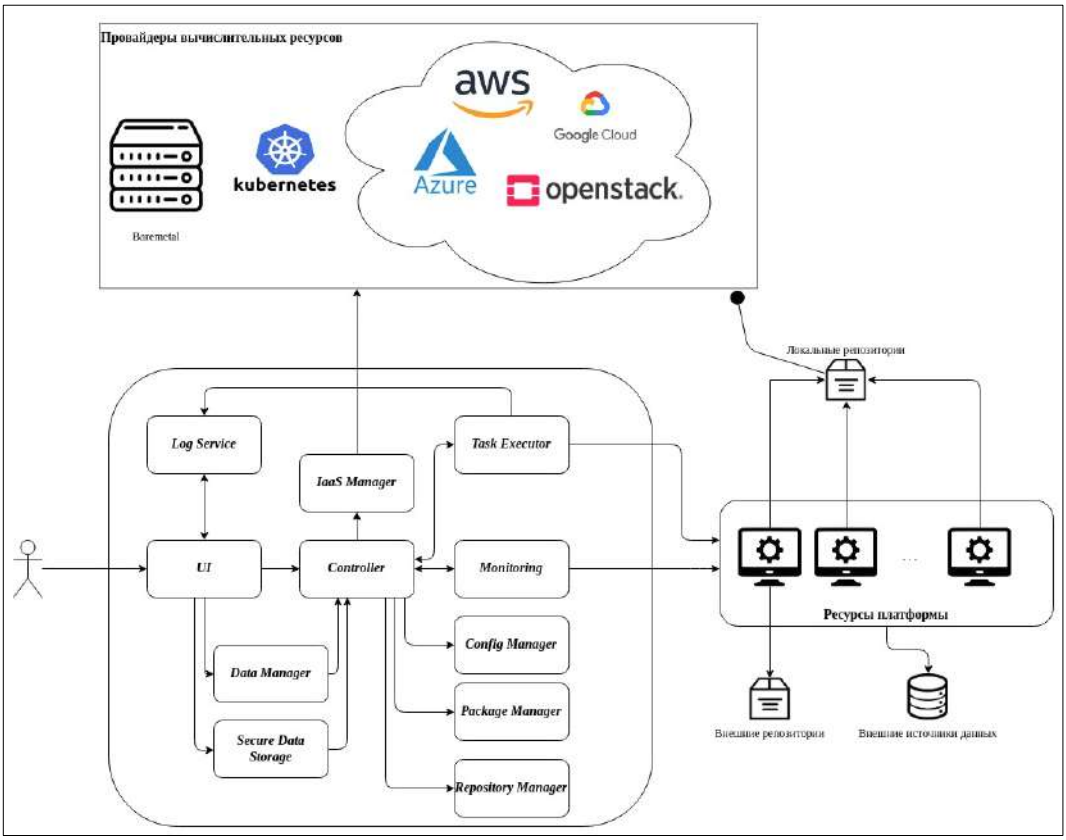

Рис. 1. Унифицированная архитектура PaaS оркестратора
Fig. 1. Unified PaaS orchestrator architecture

В инструментах, основанных на TOSCA, инфраструктурные ресурсы описываются наравне с остальными компонентами топологии платформы: для них существуют общие “нормативные” типы, которые могут быть уточнены для конкретного облачного провайдера, либо могут быть использованы собственные типы. Однако на сегодняшний день во всех рассматриваемых оркестраторах процесс создания экземпляров таких узлов реализован при помощи дополнительных типов узлов. Рассмотрим данный процесс на примере создания виртуальной машины, для которой в стандарте существует нормативный тип tosca.Nodes.Compute. В каждом оркестраторе созданы собственные типы узлов, выведенные из нормативного: “yorc.nodes.openstack.Compute”, “tosca.nodes.indigo.Compute”, “cloudify.nodes.Compute”. Далее в каждом из инструментов реализованы обработчики этих типов в виде функций или подключаемых плагинов для каждого из поддерживаемых облачных сред.

Оркестратор Heroku в большей степени ориентирован на работу в контейнерной среде, однако поддерживает работу с образами операционных систем для дальнейшей настройки на них платформ. ElastiCluster позволяет настраивать подключение к различным провайдерам, указывать образы и размеры виртуальных машин, которые будут в дальнейшем использоваться. Наконец, в Michman на текущий момент также настраивается подключение к облачной платформе при конфигурации оркестратора и выставляются параметры для создания виртуальных машин при создании кластеров. Далее в каждом оркестраторе реализовано создание по заданным параметрам вычислительных ресурсов, однако для пользователя эта информация скрыта, вместо этого пользователь оперирует конечными платформами.

Для обеспечения работы пользователя в рассмотренных инструментах предлагаются различные пользовательские интерфейсы – *UI*. В зависимости от области применимости оркестратора могут быть реализованы: графический интерфейс (GUI), интерфейс командной строки (CLI), а также может быть предоставлен REST API для программного использования. Кроме того, в ходе работы с UI в некоторых инструментах подразумевается также обеспечение аутентификации и доступа к описаниям поддерживаемых сервисов.

Также в большинстве инструментов в том или ином виде реализован доступ пользователей к службам мониторинга, необходимым для масштабирования и отказоустойчивости разворачиваемых сервисов. В общем случае для этого может быть выделен компонент *Monitoring*.

Для отслеживания процесса запуска разворачиваемых сервисов и получения дополнительной информации в случаях ошибок в оркестраторах обеспечивается доступ к логам. В большинстве рассмотренных инструментов хранение логов разделяется на два случая:

1) для доступа к логам развернутых сервисов предполагается развертывание дополнительных служб, либо предоставляется доступ к виртуальным машинам с сервисами.
2) информация о работе самого оркестратора и процессе запуска сервисов записывается локально, либо для этого явным образом выделен отдельный компонент – *Log Service*.

При развертывании приложений требуется настройка пакетов уровня операционной системы. В общем случае это может подразумевать две задачи:

1) Обобщение различных пакетных менеджеров (apt, yum, pip и другие) и разрешение конфликтов при установке пакетов, необходимых для различных сервисов;
2) Подключение к локальным пакетным репозиториям для установки пакетов без доступа в Интернет, а также для использования приватных пакетов.

В большинстве рассмотренных оркестраторов решение этих задач полностью контролируется разработчиками шаблонов и ролей разворачиваемых сервисов. С другой стороны, Heroku для подготовки образа к развертыванию описываются используемые в сервисе пакетные зависимости, доступные из публичных репозиториев. В оркестраторе Michman доступно подключение дополнительных либо локальных репозиториев. В общем случае оркестраторе для этих целей могут быть выделены компоненты для контроля пакетных зависимостей – *Package Manager*, и контроля репозиториев – *Repository Manager*.

Для запуска пользовательских сервисов необходима настройка конфигурационных файлов и переменных окружения. На данный момент явным образом этот уровень не выделен ни в одном инструменте оркестрации, хотя в Heroku используется механизм описания переменных окружения, доступных при развертывании приложений, а в Michman предложен механизм параметризации сервисов. В оркестраторах, основанных на TOSCA, обработка параметров запуска сервисов также возможна при создании шаблонов. Для подстановки пользовательских и системных значений в конфигурационные файлы может использоваться прямая модификация файлов, а также шаблоны файлов, например, в формате Jinja[34]. Заметим, что при совместном использовании нескольких сервисов могут быть модифицированы одни и те же конфигурационные файлы, что может приводить к конфликтам. Для корректной обработки пользовательских параметров и изоляции окружения различных сервисов предлагается выделить дополнительного компонента оркестратора – *Config Manager*.

Следующим важным аспектом работы оркестратора является хранение и обработка пользовательских данных: как необходимых для работы платформы, так и ключей и паролей для доступа к ресурсам. В связи с этим возникает необходимость добавления соответствующих компонентов: *Data Manager* и *Secure Data Storage*. В то время как основная задача Secure Data Storage – обеспечение безопасного хранения небольших объемов данных, Data Manager необходим для обеспечения доставки пользовательских архивов и

подключения внешних хранилищ и файловых систем. Для подключения внешних источников данных в Indigo-DC явным образом выделена служба обработки данных, в ElastiCluster реализовано подключение распределенных файловых систем, а в Heroku доступны дополнения. В свою очередь, в оркестраторах Yorc и Michman используется Vault – инструмент для безопасного хранения паролей, сертификатов, ключей доступа и других конфиденциальных данных.

Наконец, с повышением сложности управляемых оркестраторами сервисов между ними возникают зависимости разного рода. Кроме того, отдельные операции в процессе развертывания платформы требуют значительного времени. В связи с этим в большинстве оркестраторов явным образом выделены управляющие компоненты, которые могут быть обозначены в общем случае *Controller* и *Task Executor*. В то время как Controller необходим для координации всех компонентов оркестратора и обновления текущего статуса всех развернутых сервисов, Task Executor выполняет функции непосредственного выполнения отдельных команд на виртуальных машинах. При этом выполнение команд может выполняться в асинхронном режиме.

Отметим, что выделенные компоненты являются логическими составляющими, а конечная реализация может иметь как монолитную архитектуру, так и распределенную. Кроме того, предложенная архитектура может иметь дополнения в зависимости от формата использования. Например, в коммерческом использовании необходимо добавление компонентов, отвечающих за поддержку SLA, аудит пользовательских действий, а при работе в мультиоблачном режиме оркестратор должен поддерживать технологию единого входа (SSO). Однако в текущей работе основное внимание уделяется специфике организации развертывания распределенных платформ, в то время как такие дополнительные компоненты нацелены на обеспечение работы с пользователями.

5. Заключение

Современные учебные, научные и коммерческие задачи требуют не только больших вычислительных мощностей, но и сложного распределенного программного обеспечения. Платформы обработки больших данных могут требовать до тысяч вычислительных узлов, что делает невозможным ручное управление такими системами. В статье рассмотрены актуальные задачи, для которых используются облачные вычисления и современные способы описания распределенных платформенных сервисов. Исходя из этих задач, сформулирован набор требований к функциональности PaaS-оркетраторов.

На основе проведенного обзора существующих оркестраторов, предложена обобщенная архитектурная модель PaaS-оркестратора. Описанная архитектура позволяет реализовать развертывание распределенных вычислительных систем в устоявшемся при использовании контейнерной виртуализации формате:

- разработчики отдельных сервисов аналогично написанию Containerfile предоставляют информацию о том, как развертывать их программное обеспечение, включающую зависимости, требования к ресурсам, предоставляемые интерфейсы и скрипты развертывания;
- разработчики распределенных систем могут объединять доступные сервисы в распределенные топологии, предоставляя готовые к запуску шаблоны, аналогичные, например, декларативным описаниям сервисов в Kubernetes;
- конечные пользователи могут использовать готовые шаблоны, уточнять параметры в шаблонах, настраивать параметры масштабируемости отдельных компонент, а также управлять развернутыми сервисами в процессе работы.

Дальнейшая работа в этой области требует развития предложенной модели и построения полноценной таксономии облачных вычислений уровня PaaS. Также планируется применение предложенной архитектуры в оркестраторе Michman.

Кроме того, помимо предложенного набора требований к PaaS-оркестратору требуется разработка модельной задачи, которая могла бы позволить объективно оценивать такие инструменты.

Список литературы / References

[1] Mell P. and Grance T. The NIST definition of cloud computing, 2011. Available at: https://csrc.nist.gov/publications/detail/sp/800-145/final, accessed: 26.08.2022.

[2] Tomarchio O., Calcaterra D. & Modica G.D. Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks. Journal of Cloud Computing, vol. 9, 2020, article no. 49, 24 p.

[3] Dukaric R. and Juric M.B. Towards a unified taxonomy and architecture of cloud frameworks. Future Generation Computer Systems, vol. 29, issue 5, 2013, pp. 1196-1210.

[4] Luzar A., Stanovnik S., and Cankar M. Examination and comparison of TOSCA orchestration tools. Communications in Computer and Information Science, vol. 1269, 2020, pp. 247-259.

[5] Baur D., Seybold D. et al. Cloud orchestration features: Are tools fit for purpose? In Proc. of the IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), 2015, pp. 95-101.

[6] Ramon-Cortes C., Alvarez P. et al. A survey on the Distributed Computing stack. Computer Science Review vol. 42, 2021, article no. 100422, 22 p.

[7] Netto M.A.S., Calheiros R.N. et al. HPC cloud for scientific and business applications: taxonomy, vision, and research challenges. ACM Computing Surveys (CSUR), vol. 51, issue 1, 2018, article no. 8, pp. 1-29.

[8] Parodi A., Danovaro E. et al. LEXIS weather and climate large-scale pilot. Advances in Intelligent Systems and Computing, vol. 1194, 2020, pp. 267-277.

[9] Caballer M., Zala S. et al. Orchestrating complex application architectures in heterogeneous clouds. Journal of Grid Computing, vol. 16, issue 1, 2018, pp. 3-18.

[10] Yang X., Wallom D. et al. Cloud computing in e-Science: research challenges and opportunities. The Journal of Supercomputing, vol. 70, issue 1, 2014, pp. 408-464.

[11] Mathews S.M. Explainable artificial intelligence applications in NLP, biomedical, and malware classification: a literature review. Advances in Intelligent Systems and Computing, vol. 998, 2019, pp. 1269-1292.

[12] Mercl L. and Jakub P. The comparison of container orchestrators. Advances in Intelligent Systems and Computing, vol. 797, 2019, pp. 677-685.

[13] Nawaz F., Ahmad M., and Naeem K.J. Service description languages in cloud computing: state-of-the-art and research issues. Service Oriented Computing and Applications, vol. 13, issue 2, 2019, pp. 109-125.

[14] Metsch T., Papaspyrou A. et al. Open cloud computing interface–core. Open Grid Forum, OCCI-WG, Specification Document. Available at: https://ogf.org/documents/GFD.221.pdf, accessed: 26.08.2022.

[15] Moscato F., Aversa R. et al. An analysis of mOSAIC ontology for Cloud resources annotation. In Proc. of the Federated Conference on Computer Science and Information Systems (FedCSIS), 2011, pp. 973-980.

[16] Tsai W.-T., Sun X., and Balasooriya J. Service-Oriented Cloud Computing Architecture. In Proc. of the Seventh International Conference on Information Technology: New Generations, 2010, pp. 684-689.

[17] Pawluk P., Simmons B. et al. Introducing STRATOS: A Cloud Broker Service. In Proc. of the IEEE Fifth International Conference on Cloud Computing, 2012, pp. 891-898.

[18] TOSCA Version 2.0 Committee Specification Draft 04. Available at: http://docs.oasis-open.org/tosca/TOSCA/v2.0/TOSCA-v2.0.pdf, accessed: 26.08.2022.

[19] Salomoni D., Campos I. et al. INDIGO-DataCloud: A platform to facilitate seamless access to e-infrastructures. Journal of Grid Computing, vol. 16, issue 3, 2018, pp. 381-408.

[20] Ystia Suite. Available at: https://ystia.github.io/, accessed: 26.08.2022.

[21] Scionti A., Martinovic J. et al. HPC, Cloud and Big-Data Convergent Architectures: The LEXIS Approach. Advances in Intelligent Systems and Computing, vol 993, 2019, pp. 200-212.

[22] Cloudify web-site. Available at: https://cloudify.co/, accessed: 26.08.2022.

[23] Microsoft Azure documentation. Available at: https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/, accessed: 26.08.2022.

[24] AWS CloudFormation documentation. Available at: https://aws.amazon.com/cloudformation/resources/templates/, accessed: 26.08.2022.

[25] Google Cloud documentation. Available at: https://cloud.google.com/deployment-manager/docs/fundamentals, accessed: 26.08.2022.

[26] Heroku web-site. Available at: https://www.heroku.com/, accessed: 26.08.2022.

[27] ElastiCluster Repository. Available at: https://github.com/elasticluster/elasticluster, accessed: 26.08.2022.

[28] Haug S., Sciacca F.G. ATLAS computing on Swiss cloud SWITCHengines. Journal of Physics: Conference Series, vol. 898, issue. 5, 2017.
[29] Ansible web-site. Available at: https://www.ansible.com/, accessed: 26.08.2022
[30] Aksenova E., Lazarev N. et al. Michman: an Orchestrator to deploy distributed services in cloud environments. In Proc. of the 2020 Ivannikov ISPRAS Open Conference (ISPRAS), 2020, pp. 57-63.
[31] Alien4Cloud web-site. Available at: https://alien4cloud.github.io/, accessed: 26.08.2022.
[32] Consul web-site. Available at: https://www.consul.io/, accessed: 26.08.2022.
[33] Vault web-site. Available at: https://www.vaultproject.io/, accessed: 26.08.2022.
[34] Jinja web-site. Available at: https://palletsprojects.com/p/jinja/, accessed: 26.08.2022.

Информация об авторах / Information about authors

Никита Алексеевич ЛАЗАРЕВ – аспирант и стажёр-исследователь. Сфера научных интересов: облачные технологии, распределенные системы, разработка ПО.

Nikita LAZAREV is a postgraduate student and research intern. Research interests: cloud technologies, distributed systems 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 BORISENKO is a specialist and team leader at the Department of Information Systems. Research interests: cloud technologies and software development.