ДОКЛАДЫ РОССИЙСКОЙ АКАДЕМИИ НАУК. МАТЕМАТИКА, ИНФОРМАТИКА, ПРОЦЕССЫ УПРАВЛЕНИЯ, 2024, том 515, с. 105–113

ИНФОРМАТИКА

УДК 004.9

TOMMANO – УПРАВЛЕНИЕ ВИРТУАЛИЗОВАННЫМИ СЕТЕВЫМИ ФУНКЦИЯМИ В ОБЛАЧНОЙ СРЕДЕ НА ОСНОВЕ СТАНДАРТА TOSCA

© 2024 Р. К. Столяров[1,*], В. В. Швецова[1,], О. Д. Борисенко[1,***]**

Представлено академиком РАН А.И. Аветисяном
Поступило 25.10.2023 г.
После доработки 15.01.2024 г.
Принято к публикации 29.01.2024 г.

С момента своего дебюта в 2012 г. концепция виртуализации сетевых функций (NFV) значительно эволюционировала и получила широкое распространение. Технология NFV позволяет упростить настройку сетевых функций и снизить затраты на обработку трафика за счет использования программных модулей, работающих на виртуальных машинах, запускаемых на стандартном серверном оборудовании, вместо физических проприетарных сетевых устройств. Однако развертывание виртуализованных сетевых функций (таких как брандмауэр, NAT, спам-фильтр) в виде программных компонентов, управление их жизненным циклом, изменение конфигураций этих компонентов и ручная настройка маршрутизации между ними по-прежнему являются трудозатратными операциями. Описанная проблема существует из-за огромного количества различных компонентов сетевой инфраструктуры и из-за различий в функциональности выбранного программного обеспечения, сетевых операционных систем и облачных платформ. В частности, проблема актуальна для платформы анализа биомедицинских данных Научного центра мирового уровня Сеченовского университета.

В этой статье нами описывается созданный для решения данной проблемы фреймворк TOMMANO, который позволяет автоматизировать развертывание виртуализированных сетевых функций на виртуальных машинах в произвольных облачных средах. Принцип его работы основан на преобразовании декларативных шаблонов OASIS TOSCA [5, 6] в нотации, соответствующей стандарту ETSI MANO [2] для NFV, в нормативные шаблоны TOSCA и наборы скриптов Ansible. Используя эти выходные данные, TOSCA-оркестратор может развернуть приложение, использующее виртуализированные сетевые функции, в любой поддерживаемой им облачной среде.

Кроме того, в статье приводится пример использования данного фреймворка для автоматического развертывания некоторого набора сетевых функций. В этом примере Cumulus VX используется в качестве провайдерской операционной системы для сетевых функций, Clouni используется в качестве TOSCA-оркестратора, Openstack используется в качестве облачного провайдера.

Разработанный фреймворк TOMMANO получил свидетельство о государственной регистрации программы для ЭВМ № 2023682112 от 23.10.2023.

Ключевые слова: облачные вычисления, сервисные цепочки (SFC), NFV, TOSCA, сетевая автоматизация, автоматизация развертывания

DOI: 10.31857/S2686954324010169, **EDN:** ZSSASN

1. ВВЕДЕНИЕ

Облачные вычисления – это технология которая обеспечивает доступ по требованию к цифровым ресурсам по сети. Они могут представлять собой:

- приложения
- физические и виртуальные серверы
- средства разработки
- хранилища данных
- сетевые функции

Описанные ресурсы размещаются в удаленном центре обработки данных, управляемом облачным провайдером.

Провайдеры могут предоставлять вычислительные ресурсы, такие как виртуальные процессоры, сети и дисковые хранилища, без какого-либо дополнительного программного обеспечения. Такая модель называется IaaS – инфраструктура как услуга. Предоставление программных платформ поверх IaaS для пользовательских приложений реализуются моделью PaaS – платформа как услуга.

[1]*Институт системного программирования им. В.П. Иванникова Российской академии наук, Москва, Россия*

[*]*E-mail: sadimer@ispras.ru*

[**]*E-mail: shvetcova@ispras.ru*

[***]*E-mail: borisenko@ispras.ru*

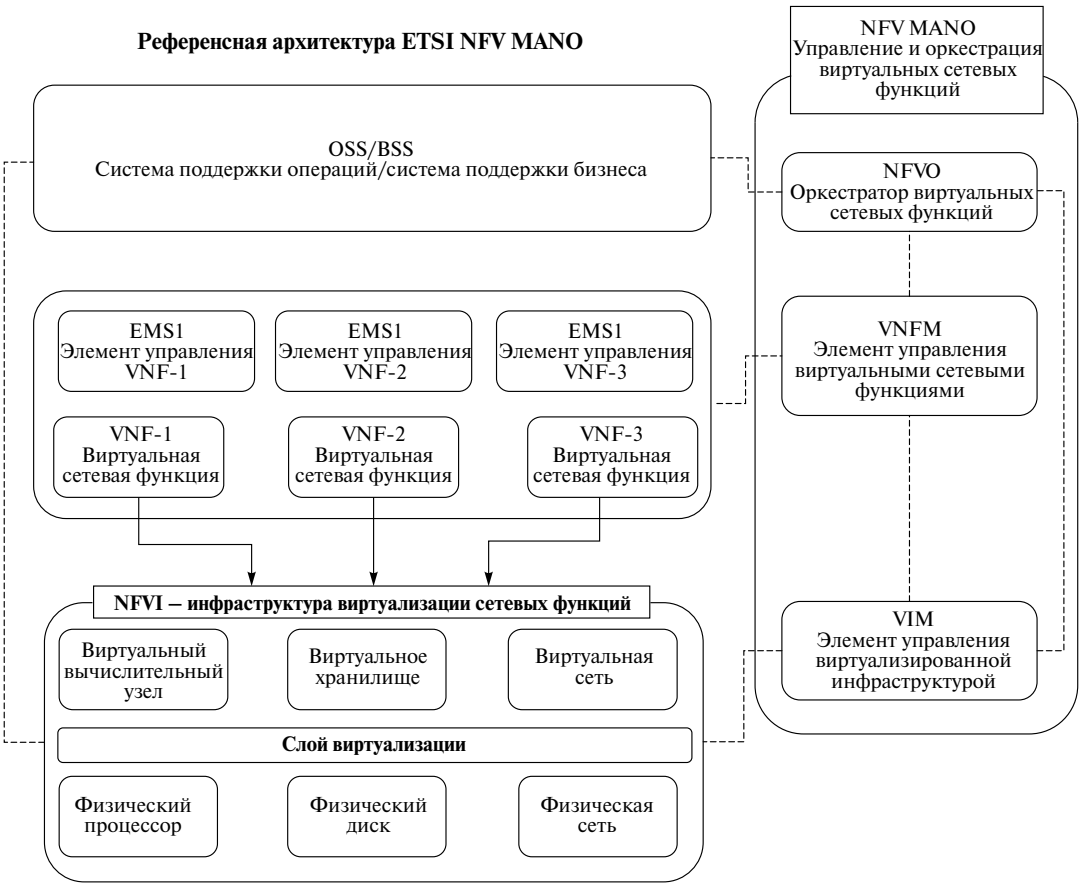

Рис. 1. Референсная архитектура ETSI MANO.

Из описания следует, что виртуальные сети, маршрутизаторы и коммутаторы являются частью модели IaaS, а, например, средства балансировки нагрузки, брандмауэры, DNS и DHCP-серверы являются частью модели PaaS.

SDN

Программно-конфигурируемые сети (SDN) – это подход к построению сети, в которой плоскость управления сетью отделена от плоскости данных и реализуется с помощью специального программного обеспечения. Сети в облачных платформах обычно организуются с использованием SDN.

NFV

Сетевой функцией (NF) называется базовый блок сетевой инфраструктуры с различными внешними интерфейсами и функциональным поведением.

Виртуализация сетевых функций (NFV) [1] – это принцип отделения сетевых функций (таких как брандмауэры, балансировщики нагрузки, IPS/IDS) от оборудования, на котором они выполняются.

Виртуализированная сетевая функция (VNF) – это реализация NF, которая может быть развернута на нескольких виртуальных машинах (BM), имитирующих отдельные сетевые устройства при помощи специального программного обеспечения.

Сетевые функции могут быть объединены в сетевые службы (NS). Они, как правило, включают в себя графы пересылки NF (NFFG и VNFFG) – графы, описывающие потоки трафика между сетевыми функциями. А также пути сетевой переадресации (NFP) – упорядоченные цепочки точек подключения к NF с политиками классификации трафика для этих цепочек.

ETSI MANO [2] (рис. 1) стандартизирует описание сетевых служб и виртуализированных сетевых функций. Стандарт описывает компоненты, входящие в структуры NS и VNF, связи между ними (включая организацию сервисных цепочек через VNFFG), интерфейсы операций

жизненного цикла, параметры масштабирования, параметры состояний, подход к организации систем оркестрации NS и VNF.

Для кого это актуально?

Пользователи облака могут задаться вопросом, зачем им использовать технологию NFV, изначально предназначенную для использования телекоммуникационными провайдерами, в облачных средах.

Преимущества NFV могут быть очевидны не сразу, однако предоставление NFV в качестве услуги облачным провайдером может косвенно повысить производительность и масштабируемость пользовательских распределенных приложений, а также усилить сетевую безопасность.

Например, в рамках предоставления NFV как услуги клиентам могут стать доступны такие функции, как балансировщик нагрузки и брандмауэр. Балансировщик нагрузки распределяет входящий трафик между несколькими серверами, снижая нагрузку на каждый из серверов. Службы брандмауэра защищают от угроз безопасности и сбоев, обеспечивая бесперебойную и надежную работу кластера и зависимых от него приложений. Таким образом, описанные службы могут помочь повысить производительность и безопасность кластера, не требуя от пользователя самостоятельного управления и обслуживания этих сетевых функций. В подтверждение сказанного, описанный функционал крайне необходим для организации безопасного сетевого взаимодействия и балансировки нагрузки при работе с нейросетевыми моделями в продуктовом сегменте платформы анализа биомедицинских данных Научного центра мирового уровня Сеченовского университета.

Кроме того, использование SDN и NFV может помочь облачным провайдерам оптимизировать использование своих ресурсов, таких как процессорное время, используемое для обработки сетевых пакетов, и пропускная способность сети. Это прямым образом приводит к повышению производительности и снижению затрат для их клиентов.

Именно поэтому надежность, доступность, скорость и удобство развертывания компонентов архитектуры NFV очень важны для пользователей.

Архитектура большинства облачных платформ по умолчанию подразумевает наличие SDN. В то же время NFV предоставляется лишь небольшим числом облачных провайдеров, и их реализации имеет существенные ограничения, что делает данную статью, описывающую реализацию фреймворка для платформонезависимого развертывания NFV, еще более актуальной.

TOSCA

TOSCA [5, 6] – это стандарт OASIS, целью которого является стандартизация описания облачных приложений, их зависимостей, возможных операций по управлению этими приложениями и используемой инфраструктуры. Основной формой представления топологии облачного приложения является граф с различными типами узлов и отношений.

Кроме того, стандарт TOSCA описывает вспомогательные сущности, такие как:

- property – определяемый пользователем параметр узла или отношения;
- attribute – параметр узла или отношения, который заполняется автоматически во время развертывания (например, IP-адрес);
- requirement – параметр узла, который связывает его с другим узлом и инициализирует отношение;
- capability – параметр, описывающий функции узла, которые он может предоставить другому узлу, сформировав с ним отношение;
- datatype – пользовательская типизированная структура данных;
- interface – параметр, описывающий операции, изменяющие состояние узла (создание, запуск, остановка). Содержит три ключевых поля:
  - inputs – входные параметры используемого скрипта или вызываемой функции;
  - implementation – скрипт или функция, реализующая операцию;
  - outputs – параметры, которые передаются в атрибуты после выполнения скрипта или функции.

Для описания компонентов в стандарте TOSCA используются две основные сущности: типы и шаблоны. Типы описывают возможные конфигурации отдельных элементов облачного приложения (аналогично определениям классов в ООП). Шаблоны описывают топологию конкретного приложения (аналогично экземплярам классов в ООП).

Существуют стандартизированные типы, называемые нормативными, которые представляют собой общие компоненты облачных приложений, такие как серверы, базы данных, сети, маршрутизаторы и т.д. Эти типы обеспечивают стандартизированный способ описания характеристик и поведения этих компонентов и могут использоваться в качестве основы для определения пользовательских типов, специфичных для конкретного приложения или варианта использования (аналогично механизму наследования из ООП).

Например, нормативные типы TOSCA включают в себя тип tosca.nodes.Compute для описания типовой конфигурации серверов, tosca.nodes.Database для представления баз данных и tosca.nodes.LoadBalancer для описания служб балансировщика нагрузки. Эти типы определяют свойства и возможности этих компонентов (например, операционная система и число ядер для сервера), а также требования к другим узлам (характеристики емкости диска для базы данных).

Используя нормативные типы TOSCA, разработчики могут определять компоненты и зависимости своих облачных приложений стандартизированным способом, переносимым на различные облачные платформы.

Важно отметить, что язык TOSCA может использоваться для автоматизации развертывания облачных приложений и управления ими, поскольку шаблоны TOSCA могут транслироваться в инструменты развертывания инфраструктуры и конфигурации ПО (такие как Ansible и Terraform) либо использоваться как метаязык для описания порядка операций, использующих данные инструменты.

TOSCA-оркестратор [7] – это система, предназначенная для настройки и развертывания целевого приложения в соответствии с декларативным описанием, соответствующим стандарту TOSCA в формате YAML, а также для мониторинга его состояния и управления жизненным циклом.

Описание NFV с использованием языка TOSCA

Концепция NFV подразумевает, что виртуализированная сетевая функция (VNF) может быть представлена как набор некоторых сущностей (VDU, swImage, virtualCompute, VL, CP), зависимостей и отношений между ними. Нетрудно заметить, что использование стандарта TOSCA для описания сетевой инфраструктуры в нотации NFV является удачным решением.

Основные сущности архитектуры NFV могут быть сопоставлены с компонентами облачной среды следующим образом:

• Virtual Network Function (VNF) – это совокупность программ, реализующих сетевую функцию, и инфраструктурных ресурсов облачной платформы, на которых они запущены (виртуальные сети, порты, машины, диски);

• Virtual Deployment Unit (VDU) – это виртуальная машина, на которой запущены программы, реализующие сетевую функцию;

• Virtual Link (VL) – это виртуальная сеть для обмена данными между виртуальными машинами;

• Connection Point (CP) – порт для подключения к виртуальной сети. Внутренний CP используется для подключения между VDU, а внешний CP используется для подключения к сторонним устройствам облачной среды или для выхода в глобальный Интернет.

Описываемый в данной статье фреймворк основан на этом отображении и удобстве представления описанных в нем элементов с использованием нормативных типов TOSCA. Он позволяет развертывать приложения, содержащие сетевые службы, соответствующие архитектуре NFV в соответствии с их декларативным описанием. Это избавляет пользователя от необходимости вручную настраивать сетевую связность компонент и конфигурацию программного обеспечения VNF. Кроме того, представленное решение позволяет не зависеть от конкретного облачного провайдера [8–9] (OpenStack, AWS, GCP и др.). Это возможно потому, что некоторые TOSCA-оркестраторы (например, Clouni [10–11]) поддерживают развертывание нормативных шаблонов в различных облачных средах.

Связанные работы

Наиболее известным описанием архитектуры NFV на языке TOSCA является набор нормативных типов, называемый TOSCA Simple Profile for Network Functions Virtualisation [12]. Этот набор охватывает множество объектов, описанных в NFV IFA 011 [1], включая свойства VnfConfigurableProperties, групповые типы VNF и маршрутизацию трафика внутри них с помощью графов пересылки VNF (VNFFG).

Однако его создание было направлено исключительно на обеспечение совместимости с TOSCA.

Задача развертывания шаблонов топологии, состоящих из описанных типов, полностью перекладывается на разработчиков TOSCA-оркестраторов. Из-за этого многие описанные типы совершенно не приспособлены для развертывания в облачных средах. Это привело к тому, что каждый разработчик оркестратора для NFV сталкивается с выбором – расширять эти типы или использовать свои собственные.

Примеры систем оркестрации, использующих расширенные типы:

• Tacker [13–14] – это инструмент оркестрации NFV со встроенной системой управления VNF. Он предназначен для развертывания и управления виртуализированными сетевыми функциями (VNF) в облачной среде Openstack.

Основные характеристики:

○ поддерживает развертывание, отслеживание состояний, модификацию VNF в режиме реального времени;

○ использует напрямую внутреннюю сетевую инфраструктуру облака на базе OpenStack. Это эффективнее, чем организовывать собственную инфраструктуру поверх предоставляемой.

Основные недостатки:

○ зависит от облачной платформы – развертывание возможно только в OpenStack;

○ может быть установлен только непосредственно на узлы облака, что делает невозможным использование в публичных облаках;

○ требует установки сторонних сервисов, таких как barbican для генерации ключей и networking-sfc для пересылки трафика в виртуальных сетях;

• mini-nfv [15] – это инструмент локальной оркестрации NFV с использованием mininet. Его основное предназначение – тестирование шаблонов топологии NFV. Как и Tacker, он поддерживает развертывание и отслеживание состояний VNF, а также перенаправление трафика с помощью VNFFG. Однако с его помощью сложно протестировать работу отдельных VNF из-за отсутствия виртуализации (mininet не имеет возможности эмулировать некоторые функции, встроенные в сетевые операционные системы) и изоляции от внешних сетей. Предлагаемое нами решение не имеет таких недостатков.

Пример системы оркестрации, использующей собственные типы – Openbaton [16]. Openbaton – это независимая платформа оркестрации, которая позволяет развертывать компоненты инфраструктуры NFV в соответствии со стандартом ETSI MANO. Аналогично с Tacker, его разработчики заявляют о поддержке развертывания только на базе инфраструктуры OpenStack. В качестве альтернативного способа описания топологий Openbaton поддерживает шаблоны TOSCA. Однако типы, которые он использует, имеют существенные расхождения со стандартом OASIS, что делает такой язык описания не стандартизированным диалектом.

2. РЕАЛИЗАЦИЯ

Разработанный фреймворк [17] представляет собой консольное приложение, которое принимает на вход шаблоны, содержащие нормативные типы TOSCA и типы, описанные в нотации NFV.

Ключевым его компонентом является транслятор, который преобразует узлы и отношения типов, соответствующих нотации NFV, в узлы и отношения нормативных типов TOSCA и универсальные сценарии Ansible для настройки параметров VNF. Сценарии добавляются к операциям узла или отношения соответствующего нормативного типа TOSCA с необходимыми параметрами и используются TOSCA оркестратором при развертывании.

Возможность настройки виртуализированных сетевых функций с использованием различных программных компонентов и операционных систем поддерживается за счет генерации сценариев Ansible по универсальным шаблонам. Поддержка различных оркестраторов обеспечивается возможностью изменения файла конфигурации транслятора и файла конфигурации оркестратора. Поддержка развертывания топологий, включающих элементы NFV, в различных облачных средах достигается благодаря возможностям выбранного TOSCA оркестратора.

Описание элементов NFV с помощью TOSCA

Описание типов TOSCA (листинг 1), соответствующих спецификации NFV:

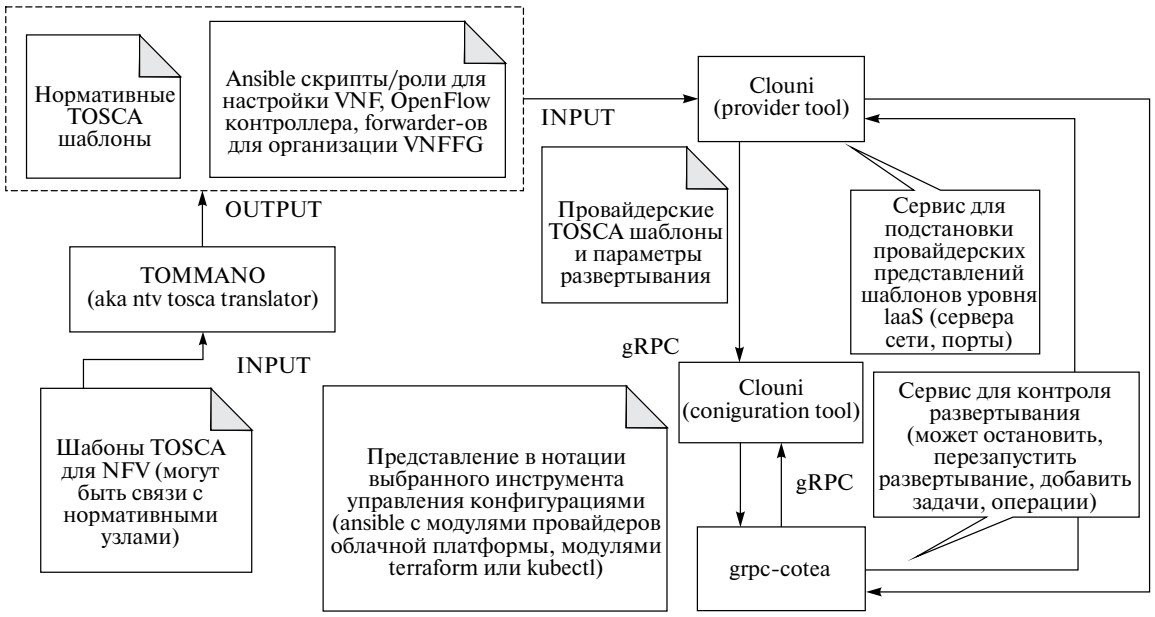

Рис. 2. Полная схема работы фреймворка.

• nfv.nodes.VDU, описывает объект VDU. Этот тип имеет три связанных datatype: virtualComputeDesc, который описывает аппаратные характеристики VDU, такие как число ядер, ОЗУ и т.д.; VirtualStorageDesc, который описывает характеристики системы хранения данных; и swImageDesc, который описывает образ операционной системы для VNF;

• nfv.nodes.Cpd, описывает объект CP. Существует два типа, которые расширяют (наследуются от) nfv.nodes.Cpd: nfv.nodes.VnfExtCpd, который представляет внешнюю точку подключения, и nfv.nodes.VDUCpd, который представляет внутреннюю точку подключения;

• nfv.nodes.VnfVirtualLinkDesc, описывает объект VL. Этот тип имеет два связанных datatype: ConnectivityType, который описывает тип соединения (например, Ethernet, MPLS, ODU2, IPV4, IPV6) и VirtualLinkDescFlavour, который описывает требования к соединению (например, скорость передачи данных, политики QoS). С ним также связан опциональный datatype CidrData, который предоставляет информацию об адресах и масках подсетей. Если этот datatype не используется в шаблоне, IP-адрес выбирается случайным образом из пула незанятых адресов локальной сети;

• nfv.nodes.VNFD, является основным типом, описывающим VNF и связывающим все остальные типы вместе в единый дескриптор VNF. Тип для каждой сетевой функции представляет собой тип, расширяющий nfv.nodes.VNFD (например, nfv.nodes.VNFD.DNS). Эти типы отличаются структурой связанных атрибутов внутри datatype VnfInfoModifiableAttributes, которые могут либо описывать параметры конфигурации конкретной VNF в виде вложенных datatype, либо содержать дополнительный скрипт для настройки VNF, если универсального Ansible-сценария недостаточно. Список этих типов может быть расширен в процессе написания приложения с использованием фреймворка, описанного в статье.

Транслятор

Алгоритм транслятора состоит из следующих действий:

• чтение из файла шаблона топологии TOSCA в нотации NFV, дополнение его определениями типов узлов и отношений;

• валидация шаблона и разделение на основные логические части;

• расширение набора properties, attributes и requirements в типах, производных от других типов;

• обработка встроенных функций, таких как get_property, get_input и т.д., и подстановка вычисляемых значений;

• замена узлов и отношений TOSCA, описанных во входном шаблоне, узлами и отношениями нормативных типов путем преобразования их requirements, properties и attributes в соответствии с правилами сопоставления (листинг 2);

```
    routing:
     type: nfv.nodes.VNFD.Routing
     properties:
       vnfdId: vnfd_0
       vnfProvider: clouni
       vnfProductName: Routing
       vnfSoftwareVersion: v1.0
       modifiableAttributes:
         extension:
           rules:
             - destAddr: { get_property: [ port_0, ip_address ] }
             - destAddr: { get_property: [ port_1, ip_address ] }
           routes:
             - destCidr: default
               gateway: { get_property: [ internal_v1, cidrData,
gatewayIp ] }
               dev: swp1
     requirements:
      - vnfExtCpd: ext_gateway_cpd
      - vdu: router_vdu
```

Листинг 1. Шаблон TOSCA с использованием сущности из стандарта NFV.

```
nfv.nodes.{node name}:
   properties:
      {datatype name}.{...}.{property name}:
        - type: {normative node name}
          parameter: {datatype name}.{...}.{property/attribute/re-
quirement name}
          format: {python format changer}
   attributes:
      {datatype name}.{...}.{attribute name}:
        - type: {normative node name}
          parameter: {datatype name}.{...}.{property/attribute/re-
quirement name}
          format: {python format changer}
   requirements:
      {datatype name}.{...}.{requirement name}:
       - type: {normative node name}
         parameter: {datatype name}.{...}.{property/attribute/re-
quirement name}
         format: {python format changer}
         node_name: {check/rename/not change}
```

Листинг 2. Правила сопоставления.

• случайная генерация IP-адресов и названий сетей/подсетей, если это необходимо (согласно стандарту ETSI MANO для NFV, они не указываются в дескрипторах);

• установление соответствия между IP-адресами и портами в узлах типов tosca.nodes.network.Network, tosca.nodes.Compute и tosca.nodes.network.Port;

• сопоставление целевых узлов с их конфигурационными скриптами Ansible и добавление их к операциям соответствующих интерфейсов.

Важно отметить, что процесс трансляции не нарушает отношения с узлами типов, отличных от NFV.

После применения описанного алгоритма шаблон готов к развертыванию с помощью TOSCA-оркестратора.

Если исключить конфигурацию портов, сетей, вычислительных узлов и рассмотреть только конфигурацию программного обеспечения, реализующего виртуализированную сетевую функцию, то после сопоставления вместо узла в листинге 1 будет создан узел, описанный в листинге 3.

```
  software_for_router_vdu:
    interfaces:
      Standard:
        create:
          implementation: Routing.yaml
          inputs:
            Routing_routes:
            - destCidr: default
              dev: swp1
              gateway: 192.168.2.1
            Routing_rules:
            - destAddr: 192.168.2.10
            - destAddr: 192.168.2.11
    requirements:
    - host: router_vdu
    type: tosca.nodes.SoftwareComponent
```

Листинг 3. Нормативный TOSCA-шаблон после трансляции.

Процесс развертывания

Схема на рис. 2 демонстрирует процесс развертывания сетевой службы с использованием описываемого фреймворка и TOSCA-оркестратора Clouni. Clouni получает от TOMMANO шаблоны в нормативной нотации и скрипты Ansible, связанные с интерфейсами операций узлов, описанных в этих шаблонах. Затем Clouni преобразует нормативные шаблоны в шаблоны провайдера, обрабатывает граф зависимостей между узлами шаблона и генерирует сценарии Ansible, используя специализированные модули провайдера (или модули конфигурации Terraform или kubectl) для взаимодействия с API соответствующего облачного провайдера. Сгенерированные скрипты Ansible запускаются параллельно при помощи grpc-cotea, инструмента для программно-контролируемого запуска Ansible.

Если узел, полученный путем обхода графа шаблона, принадлежит IaaS-части приложения, Clouni автоматически генерирует сценарии Ansible для его развертывания. В противном случае выполняется сценарий, связанный с интерфейсом выполняемой операции, который был сгенерирован TOMMANO по одному из универсальных Ansible-сценариев. Этот механизм позволяет использовать шаблонные скрипты для развертывания программного обеспечения VNF.

Демонстрационный пример

Для демонстрации функциональности описываемого фреймворка с его помощью было развернуто приложение, автоматически настраивающее виртуализированные сетевые функции (DNS, DHCP, брандмауэр, NAT, DPI, маршрутизацию, анализ трафика на различных уровнях) с использованием TOSCA-оркестратора Clouni

[10–11] и Cumulus VX [18] в качестве операционной системы VDU (рис. 3).

Cumulus VX была выбрана в качестве операционной системы для VDU из-за широкой функциональности этой системы, возможности установки сторонних программных пакетов для расширения набора поддерживаемых сетевых функций (такие системы, как OpenWRT и VyOS [19–20], не имеют такой функциональности) и возможности использовать стандартные linux-интерфейсы в качестве портов коммутатора (такие системы, как SoNiC и PicOS [21–22], используют свое собственное представление интерфейсов, что затрудняет их использование в облачной среде).

Более того, Cumulus VX [18] представляет собой демоверсию операционной системы для whitebox сетевых устройств. Это позволяет использовать фреймворк, описанный в этой статье, для создания прототипов приложений с набором сетевых служб, развертываемых на физических узлах.

Для брандмауэра, NAT, маршрутизации и DHCP использовались инструменты, предоставляемые Cumulus VX, для DNS использовался bind9 [23], для DPI использовалось расширение для iptables под названием nDPI [24], в качестве анализаторов трафика использовались tshark и ntopng [25]. Для удобства и скорости развертывания был создан образ Cumulus VX с предустановленными службами для всех описанных VNF, однако развертывание “с нуля” также возможно.

3. ЗАКЛЮЧЕНИЕ

Комбинация описанного приложения и TOSCA-оркестратора является решением поставленной проблемы, поскольку полностью устраняет необходимость вручную настраивать инфраструктуру в облачной среде и ПО сетевых функций, развертываемых на этой инфраструктуре. Вместо этого используется общепринятый декларативный стандарт описания сетевых функций и системы для автоматизации развертывания облачных инфраструктур и приложений.

Планы на будущее

Расширение набора поддерживаемых функций в реализации на базе маршрутизатора с Cumulus VX за счет добавления следующих компонентов: IPS/IDS, QoS, VPN, балансировка нагрузки.

Эксперименты с заголовками IPv6 для организации сервисных цепочек.

Добавление TOMMANO в каталог сервисов системы оркестрации Michman [26].

Эксперименты с собственным SDN-контроллером и туннелированием с использованием протокола GENEVE.

ИСТОЧНИК ФИНАНСИРОВАНИЯ

Данная работа выполнена при поддержке Министерства науки и высшего образования Российской Федерации, соглашение № 075-15-2022-294 от 15 апреля 2022 г.

СПИСОК ЛИТЕРАТУРЫ

1. ETSI GS NFV-IFA 011 Network Functions Virtualisation (NFV) Specification. Available at: https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/011/02.01.01_60/gs_nfv-ifa-011v020101p.pdf, accessed: 31.08.2023.
2. ETSI GS NFV-MAN 001 Network Functions Virtualisation (NFV), Management and Orchestration. Available at: https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man-001v010101p.pdf, accessed: 31.08.2023.
3. *Bouten N., Boutaba R., Gorricho J., Mijumbi R., Serrat J., Turck F.D.* Network Function Virtualization: State-of-the-Art and Research Challenges // IEEE Communications Surveys and Tutorials. 2016. Vol. 18. P. 236–262.
4. *Kaur K., Mangat V., Saluja K.* A review on Virtualized Infrastructure Managers with management and orchestration features in NFV architecture // Computer Networks. 2022. Vol. 217. 109281. DOI: 10.1016/j.comnet.2022.109281
5. OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA). Available at: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.html, accessed: 31.08.2023.
6. *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. 2022/ Vol. 34. I. 5. P. 143–162 (in Russ.). doi: 10.15514/ISPRAS-2022-34(5)-9
7. *Lazarev N.A., Borisenko O.D.* Requirements and architecture design for cloud PaaS orchestrator // Trudy ISP RAN / Proc. ISP RAS. 2022. Vol. 34. I. 4. P. 211–228 (in Russ.). doi: 10.15514/ISPRAS2022-34(4)-15
8. Amazon Web Services. Available at: https://aws.amazon.com/, accessed: 31.08.2023.
9. Open Source Cloud Computing Infrastructure – OpenStack. Available at: https://www.openstack.org/, accessed: 31.08.2023.
10. *Shvetcova V., Borisenko O., Polischuk M.* Domain-Specific Language for Infrastructure as Code // 2019

Ivannikov Memorial Workshop (IVMEM), Velikiy Novgorod, Russia, 2019. P. 39–45. doi: 10.1109/IVMEM.2019.00012
11. *Shvetcova V., Borisenko O., Polischuk M.* Using Ansible as Part of TOSCA Orchestrator // 2020 Ivannikov Ispras Open Conference (ISPRAS), Moscow, Russia, 2020. P. 109–114. doi: 10.1109/ISPRAS51486.2020.00023
12. OASIS TOSCA Simple Profile for Network Functions Virtualization (NFV). Available at: http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html, accessed: 31.08.2023.
13. *Simar A.* NFV Orchestration using OpenStack. Master's thesis/ Computer Science Dept., University of Victoria, 2017.
14. *Chen J., Chen Y., Tsai S.-C., Lin Y.-B.* Implementing NFV system with OpenStack // 2017 IEEE Conference on Dependable and Secure Computing. Taipei, Taiwan, 2017. P. 188–194. doi: 10.1109/DESEC.2017.8073806
15. *Castillo-Lema J., Venâncio Neto A., Oliveira de F., Takeo Kofuji S.* Mininet-NFV: Evolving Mininet with OASIS TOSCA NVF profiles Towards Reproducible NFV Prototyping // 2019 IEEE Conference on Network Softwarization (NetSoft). Paris, France, 2019. P. 506–512. doi: 10.1109/NETSOFT.2019.8806686
16. Open Baton: an open source reference implementation of the ETSI Network Function Virtualization MANO specification. Available at: https://openbaton.github.io/cases.html, accessed: 31.08.2023.
17. TOMMANO source code. Available at: https://github.com/sadimer/tommano, accessed: 31.08.2023.
18. Cumulus Linux User Guide. Available at: https://docs.nvidia.com/networking-ethernet-software/cumulus-linux-54/, accessed: 31.08.2023.
19. VyOS – Open source router and firewall platform. Available at: https://vyos.net/, accessed: 31.08.2023.
20. OpenWrt Project. Available at: https://openwrt.org/, accessed: 31.08.2023.
21. Microsoft Azure, Software for Open Networking in the Cloud. Available at: https://sonic-net.github.io/SONiC/, accessed: 31.08.2023.
22. PicOS: Disaggregated NOS for White Box Switches. Available at: https://www.pica8.com/picos-software/, accessed: 31.08.2023.
23. Bind9 – DNS server. Available at: https://www.isc.org/bind/, accessed: 31.08.2023.
24. DPI for linux as an extension of iptables. Available at: https://devel.aanet.ru/ndpi/, accessed: 31.08.2023.
25. ntopng – High-Speed Web-based Traffic Analysis and Flow Collection. Available at: https://www.ntop.org/products/traffic-analysis/ntop/, accessed: 31.08.2023.
26. *Aksenova E., Lazarev N., Badalyan D., Borisenko O., Pastukhov R.* Michman: an Orchestrator to deploy distributed services in cloud environments // 2020 Ivannikov Ispras Open Conference (ISPRAS). Moscow, Russia, 2020. P. 57–63. doi: 10.1109/ISPRAS51486.2020.00015

TOMMANO – VIRTUALISED NETWORK FUNCTIONS MANAGEMENT IN CLOUD ENVIRONMENT BASED ON THE TOSCA STANDARD

© 2024 R. K. Stolyarov[a], V. V. Shvetcova[a], O. D. Borisenko[a,*]**

[a] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, Moscow, 109004, Russia*

Presented by Academician of the RAS A.I. Avetisyan

Since 2012 **NFV (Network Functions Virtualisation)** technology has evolved significantly and became widespread. Before the advent of this technology, proprietary network devices had to be used to process traffic. **NFV** technology allows you to simplify the configuration of network functions and reduce the cost of traffic processing by using software modules running on completely standard datacenter servers (in virtual machines). However, deploying and maintaining **virtualised network functions** (such as firewall, NAT, spam filter, access speed restriction) in the form of software components, changing the configurations of these components, and manually configuring traffic routing are still complicated operations. The problems described exist due to the huge number of network infrastructure components and differences in the functionality of chosen software, network operating systems and cloud platforms. In particular, the problem is relevant for the biomedical data analysis platform of the world-class Scientific Center of Sechenov University.

In this article, we propose a solution to this problem by creating a framework TOMMANO that allows you to automate the deployment of virtualised network functions on virtual machines in cloud environments. It converts **OASIS TOSCA** [5][6] declarative templates in notation corresponding to the ETSI MANO [2] for NFV standard into normative **TOSCA** templates and sets of **Ansible** scripts. Using these outputs an application containing virtualised network functions can be deployed by the **TOSCA** orchestrator in any cloud environment it supports. The developed TOMMANO framework received a certificate of state registration of the computer program No. 2023682112 dated 10.23.2023.

In addition, this article provides an example of using this framework for the automatic deployment of network functions. In this solution Cumulus VX is used as the provider operating system of network functions. Clouni is used as an orchestrator. Openstack is used as a cloud provider.

Keywords: cloud computing, service function chaining, NFV, TOSCA, network automation, deployment automation