Institute of Architecture of Application Systems

Modeling and Automated Deployment of Serverless Applications using TOSCA

Michael Wurster, Uwe Breitenbücher, Kálmán Képes,
Frank Leymann, and Vladimir Yussupov

Institute of Architecture of Application Systems
University of Stuttgart, Stuttgart, Germany
{wurster, breitenbuecher, kepes, leymann, yussupov}@informatik.uni-stuttgart.de

BibTeX :

```
@inproceedings{Wurster2018_ServerlessTOSCA,
  author    = {Michael Wurster, Uwe Breitenb\"{u}cher, K\'{a}lm\'{a}n K\'{e}pes,
               Frank Leymann, and Vladimir Yussupov},
  title     = {{Modeling and Automated Deployment of Serverless Applications
                using TOSCA}},
  booktitle = {Proceedings of the IEEE 11th International Conference on
               Service-Oriented Computing and Applications (SOCA)},
  year      = {2018},
  pages     = {73---80},
  doi       = {10.1109/SOCA.2018.00017},
  publisher = {IEEE Computer Society}
}
```


Universität Stuttgart
Germany

Modeling and Automated Deployment of Serverless Applications using TOSCA

Michael Wurster, Uwe Breitenbücher, Kálmán Képes, Frank Leymann, and Vladimir Yussupov
Institute of Architecture of Application Systems, University of Stuttgart, Germany
{wurster, breitenbuecher, kepes, leymann, yussupov}@informatik.uni-stuttgart.de

***Abstract*—The serverless computing paradigm brings multiple benefits to application developers who are interested in consuming computing resources as *services* without the need to manage physical capacities or limits. There are several deployment technologies and languages available suitable for deploying applications to a single cloud provider. However, for multi-cloud application deployments, multiple technologies have to be used and orchestrated. In addition, the event-driven nature of serverless computing imposes further requirements on modeling such application structures in order to automate their deployment. In this paper, we tackle these issues by introducing an event-driven deployment modeling approach using the standard Topology and Orchestration Specification for Cloud Applications (TOSCA) that fully employs the suggested standard lifecycle to provision and manage multi-cloud serverless applications. To show the feasibility of our approach, we extended the existing TOSCA-based ecosystem OpenTOSCA.**

***Index Terms*—Serverless, Multi-Cloud, Modeling, Automated Deployment, TOSCA**

I. Introduction

With the advent of the serverless computing paradigm, cloud application developers can focus more on application's business logic leaving the infrastructure-related duties to cloud providers [1], [2], [3]. The term *serverless* here stresses the fact that servers are irrelevant from a developer's point of view. This does not mean that servers are no longer needed. In the serverless computing model, management and scaling of required computing resources remains the responsibility of cloud providers [4]. Furthermore, serverless computing relies on a fine-grained, usage-based cost model where customers never pay for idle since only the actual amount of resources consumed by an application is charged. Frequently, the term serverless is linked with the *Function as a Service* (FaaS) cloud delivery model, which fits nicely into the idea of developing applications without worrying about the underlying infrastructure. The FaaS model allows developing and deploying custom server-side logic in the form of ephemeral and stateless functions employing an *event-driven programming model* [3]. The main difference from the *Platform as a Service* (PaaS) model is that with PaaS pre-purchased units of capacity are always running, which is not the case with FaaS. Such functions constitute event-centric *serverless architectures* that interact with a variety of fully managed cloud services, e.g., message queues, databases, payment, and identity services. Due to the increase use of serverless architectures, it is often the case that serverless applications have to be integrated with existing, traditional application stacks, e. g., deployed on *Infrastructure as a Service* (IaaS) offerings hosted in private cloud environments. For example, since FaaS is well-suited for infrequent but high workloads [4], [3], one valid use case is the migration of existing features, falling into this workload category, to event-driven, short-lived, and stateless functions. Multiple deployment technologies and cloud providers offer capabilities to deploy serverless applications. Often, such capabilities are tightly-coupled with the features, APIs, and other specifics of the chosen technology. In multi-cloud deployments, this results in an integration challenge of using multiple providers and technologies for a single deployment, which is complex, error-prone, and time-consuming. Furthermore, it is even more challenging when deployments into private and public cloud environments have to be combined where traditional and serverless technologies are exploited. Additional layers, in terms of deployment automation [5], have to deal with the problems of proper specification, coordination and orchestration of different infrastructure stacks and cloud services.

However, to fully benefit from employing serverless architectures, developers should be able to model the deployment of such multi-cloud application stacks independent of a specific technology. In this paper, we tackle this issue by introducing an approach that allows to model the automated deployment of multi-cloud serverless applications using the Topology and Orchestration Specification for Cloud Applications (TOSCA). As there are currently only partial solutions available, not working for multi-cloud application stacks or just for specific cloud providers, we demonstrate a TOSCA-based deployment modeling approach that is able to combine heterogeneous deployment technologies. We describe how serverless architectures can be modeled using the standard TOSCA modeling constructs to represent event sources, events, and functions. To support this, we analyze how event-driven behavior of application components can be expressed with respect to corresponding events and how deployment operations, required for configuring such behavior, can be described in TOSCA. On top of that, we show how to purely utilize the standard interfaces in order to install, configure, and run serverless applications. It has been shown that TOSCA is well suited to integrate heterogeneous technologies [6]. Therefore, being completely aligned to TOSCA, a standard compliant runtime is able to provision such modeled applications independent of the used deployment technologies. We validate the practical feasibility of our approach by showing how this can be realized using the OpenTOSCA ecosystem.

The remainder of this paper is structured as follows: Section II starts with a brief introduction into TOSCA. Section III and Section IV describe our approach of modeling serverless applications as well as aspects of automated deployment using TOSCA. Section V discusses the approach and highlights additional challenges, whereas Section VI validates our approach by presenting a prototypical implementation based on the OpenTOSCA ecosystem. Section VII presents related work, while Section VIII concludes and discusses future work.

II. TOSCA Fundamentals

In the following, we briefly cover the fundamentals of TOSCA [7], [8]. We simplify and skip all TOSCA details that are not important in our context. TOSCA is an open and vendor-neutral standard by OASIS that specifies a cloud modeling language (CML) [9]. TOSCA allows modelers to describe the structure and behavior of cloud-based services with the focus on portability and interoperability of the described application model. Such combination of structure and behavior information in TOSCA terminology is referred to as a *Service Template*. The application's structure, or its *topology*, can be represented as a directed graph with nodes describing components of the application and edges defining the relationships among them. In TOSCA terms, a structure of applications is defined in a form of a *Topology Template*, which comprises *Node Templates* and *Relationship Templates* as its core building blocks. Additionally, TOSCA provides a type system that is intended to specify common semantics and to simplify the reuse of modeled entities. By using types, modelers are able to define certain characteristics of nodes and relationships in corresponding *Node Types* or *Relationship Types*. Moreover, modelers can define interfaces for node and relationship types that allow provisioning engines to trigger lifecycle operations, e. g., a *create* operation that is responsible for installing the corresponding node or a *post_configure_target* operation used to trigger configuration actions required to establish a relationship. The actual logic performing specified operations is provided in a form of so-called *Implementation Artifacts* (IA), which can be simple Shell scripts or more complex applications implementing respective operations. *Node Type Implementation* or *Relationship Type Implementation* are then used to assign a given IA to the respective Node or Relationship Type. It is worth mentioning that TOSCA allows for one type to have multiple type implementations. For instance, creating a virtual machine, i. e., *create* operation of the lifecycle, might require different actions depending on the underlying hypervisor. Therefore, IAs can be specified in different Node Type Implementations to support handling the same lifecycle operations in different environments. In addition, there are *Deployment Artifacts* (DA) that implement the business functionality of a Node Template or Node Type, e. g., an online-shop application hosted on a web server. Moreover, TOSCA introduces constructs to describe non-functional system requirements in a form of *Policy Types* and *Policy Templates*. Attached to certain entities of the topology, policies can be utilized to fulfill additional requirements at different stages of the application's lifecycle.

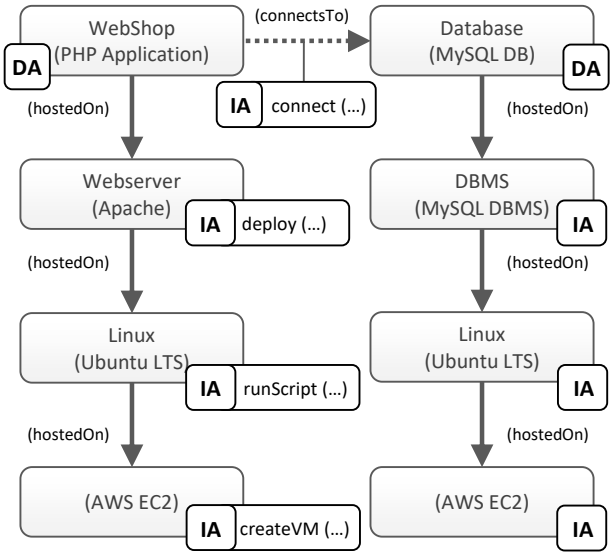

Figure 1. TOSCA Topology Template of a LAMP-based application [10].

Finally, the extensible nature of TOSCA allows introducing new or modifying existing constructs, which makes the modeling of cloud applications even more flexible.

One example [10] of an application topology relying on a standard TOSCA modeling approach is depicted in Fig. 1. The shown topology represents the structure of an e-commerce *PHP* application, which is hosted on *Apache Web Server*. In addition, a *MySQL* database is used for storing the application's state. All components are hosted on an *Ubuntu* operating system using two distinct instances of *AWS EC2* in order to operate the business logic and the database separately. Components of the depicted application, i. e., Node Templates, are represented as nodes of the graph. Every component is related to a certain Node Type, e. g., *WebShop* Node Template is of type *PHP Application*. Relationships among these nodes, i. e., Relationship Templates, are represented as directed edges, either of type *hostedOn* or *connectsTo*. Deployment Artifacts representing the application's business logic and the schema of the database are attached to the corresponding Node Templates. In a similar way, Implementation Artifacts are attached to the corresponding nodes or relationships and are assigned to respective lifecycle operations. For example, the *Linux* Node Template exposes a management operation *runScript()* to run arbitrary scripts on the operating system. The operation's logic is implemented by a respective IA and is referenced in the Topology Template. In the same manner, arbitrary management operations can be modeled using dedicated IAs.

The resulting topology, grouped together with attached artifacts and other metadata, represent a complete application ready for deployment, the so-called Service Template. In addition, TOSCA provides a packaging and export format to support the portability of applications. A so-called Cloud Service Archive (CSAR) groups together all required information including metafiles describing the contents of the archive.

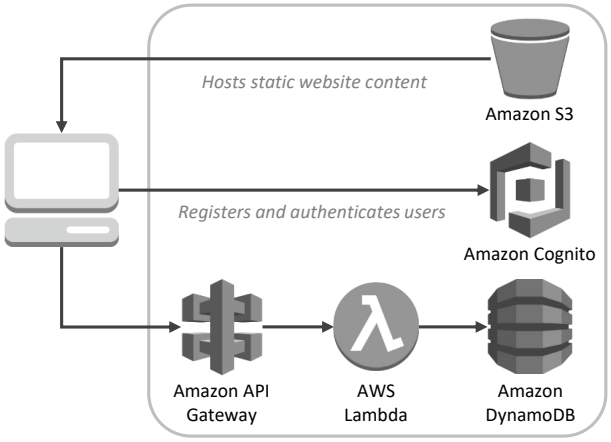

Figure 2. High-level serverless architecture use case [11].

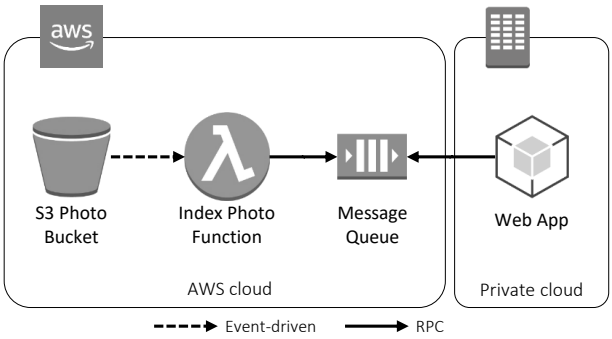

Figure 3. Multi-cloud serverless application use case.

III. Modeling Serverless Applications with TOSCA

In this section, we show how to use TOSCA standard constructs to model an automated deployment for serverless applications. The presented approach covers three important aspects of serverless application modeling: (i) modeling of function deployments, (ii) modeling of components that emit events, and (iii) modeling of an event flow. First of all, we introduce a motivating scenario that highlights challenging modeling and deployment aspects and with which we explain and demonstrate our approach in the following.

A. Motivating Scenario

With serverless computing model, one can fully focus on the application's business logic instead of worrying about managing and operating the underlying infrastructure. There is no need to worry about deploying or configuring servers since respective cloud providers do all the required work. Application developers can create serverless applications by combining different services provided and fully managed by a cloud provider. One typical use case for such architectures is to process traditional request and response workloads, for example, a web application that utilizes HTTP REST APIs as depicted in Fig. 2. For the sake of brevity, we chose Amazon Web Services (AWS) as cloud computing provider. In this example, a web application is envisioned where the respective static website content is hosted and served from AWS S3. Clients access the public URL of an S3 bucket with a web browser and download the required static website content, such as HTML, JavaScript, and CSS files, in order to run the application. For user registration and log in, the given web application relies on Cognito, an authentication service from Amazon. Next, through Amazon's API Gateway, the application can access arbitrary backend functionality implemented with AWS Lambda—Amazon's serverless data processing service. Using AWS Lambda, teams can develop and run any application logic as small pieces of code or rather event-driven functions that can respond to a variety of events and triggers. There are several event sources available that can be used to trigger such functions. For example, platforms provide messaging services, e. g., Amazon SQS, that enables processing of messages using specified functions. Furthermore, there are storage services, e. g., AWS S3 or Amazon DynamoDB, that emit events whenever clients interact with these services, for example, if entities are added, modified, or removed. Lastly, there are endpoint services, like Amazon's API Gateway, that allow turning HTTP client requests into events, or there are scheduling services to enable the invocation of functions at regular intervals. In turn, functions in AWS Lambda can use any managed service utilizing the respective Software Development Kit (SDK). In such scenario, the advantages of having a serverless architecture are reduced administrative burdens for infrastructure components and shortened time to market, since the platform takes care of it. Software development teams only have to configure the services together and upload the respective application code to AWS.

However, there are scenarios where application systems cannot be purely built serverless. For example, in scenarios with existing or even legacy applications that are already in place, which cannot be migrated to serverless due to capacity and time constraints. In such cases, a valid use case is to utilize serverless computing in order to extend existing application system with new features. Since serverless is well-suited for infrequent but high workloads [4], [3], new features can be implemented as event-driven, short-lived, and stateless functions using FaaS. Figure 3 depicts such scenario where a traditional application is hosted inside a private cloud environment, e. g., an OpenStack cloud environment, and connects to a message queue hosted and configured on AWS. On the left hand side of the figure, a simplified use case scenario is shown. In this scenario, an S3 bucket hosted on AWS is used to upload and store photos. Whenever a photo is uploaded to the S3 bucket, an event is emitted, which, in its turn, triggers a serverless processing task using AWS Lambda. This processing task, or function, extracts certain attributes of the photo, such as filename, size, and external URL, and publishes this information to a message queue. The traditional application can further process this information as soon as it gets notified by the queue.

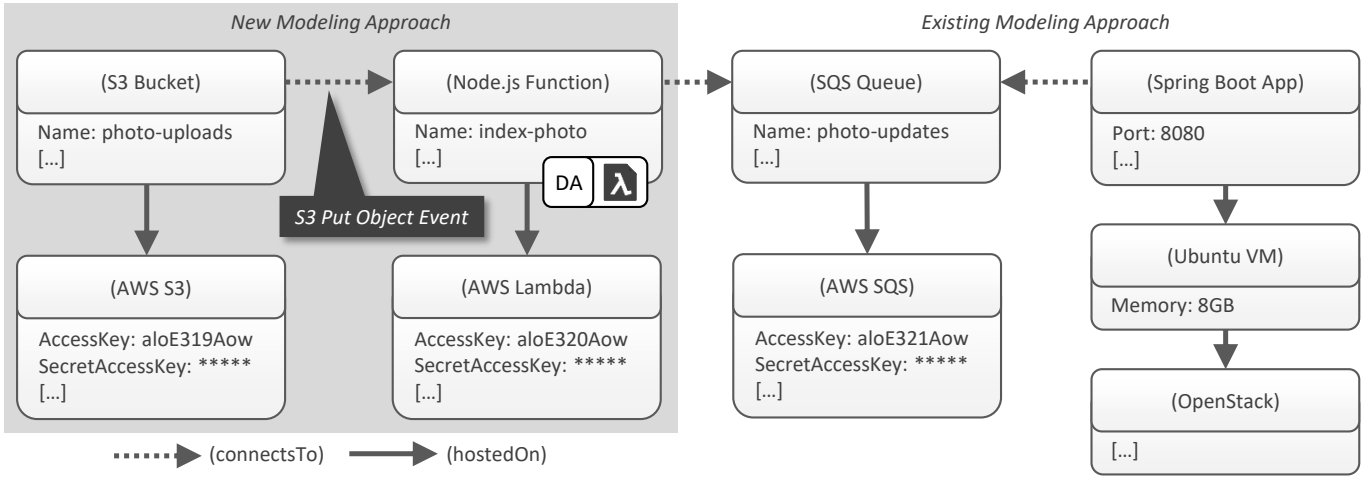

Figure 4. Motivating Scenario: Multi-cloud application topology combining serverless and traditional application provisioning.

In terms of deploying such a multi-cloud scenario, several things have to be considered and multiple technologies have to be used and orchestrated. First of all, the message queue has to be set up accordingly using the respective AWS API, whereas one can interact with the API using a command-line tool, a Software Development Kit (SDK) provided for several programming languages, or by directly using Amazon's RESTful HTTP API. Afterwards, the on-premise application can be installed and started in the private OpenStack environment. In this case, the application can be installed using automation tools, such as Chef [12], Puppet [13], or Ansible [14]. At this stage, the on-premise application is able to read messages from the queue. Thereafter, the S3 bucket has to be configured in such a way that files can be uploaded, i. e., by using one of Amazon's APIs. Next, the AWS Lambda function has to be setup and the respective function code has to be uploaded, which is able to push respective messages to the configured queue. Again, this deployment step can be automated using AWS APIs or using the Serverless Framework [15]. Finally, the event trigger for the function has to be declared. In our scenario, the connection between the S3 bucket and the function is made by specifying which type of event should trigger the function, for example, whenever a file has been uploaded to the bucket.

Using TOSCA, we can fully model such a multi-cloud serverless application independently of a certain deployment technology. Figure 4 shows the introduced motivating scenario as an application topology. We specify the application topology as a directed graph where nodes represent components and edges represent the relations among them (cf. Section II). On the right hand side of Fig. 4 we use an existing modeling approach using TOSCA to model a Java application that connects to Amazon's SQS service and is hosted on an Ubuntu virtual machine, which runs in an OpenStack environment. Former research and related work have shown the feasibility of modeling such application deployments [16], [17], [18], [10]. On the left, our modeling approach for serverless application deployment is depicted, which is explained in detail below.

B. Modeling Function Deployment

By employing a serverless architecture, arbitrary backend logic can be split up into several logical functions and deployed onto a FaaS platform. On what basis and granularity the business logic is split into functions is very much opinionated and not further discussed in this paper. In our motivating scenario, we only depict a single function that is hosted on AWS Lambda, Amazon's FaaS platform. Since functions are the deployment units in FaaS, we model a specific function as a Node Template where the respective function code can be supplied as a DA (cf. Fig. 4). Specific semantics, e. g., expression of configuration settings, can be modeled using a corresponding Node Type. For example, the actual name of the function or the required memory settings can be specified by respective property definitions inside such a Node Type. Furthermore, functions require an execution environment. They are running and hosted on a certain FaaS runtime, AWS Lambda in our example. In TOSCA, respective execution environments are usually modeled as separate Node Templates as well as Node Types. Like so, we model a *AWS Lambda* node specifying general, platform-specific semantics, e. g., authentication properties in order to access the cloud provider's API. As a result, functions can be related to these nodes using the normative TOSCA Relationship Type *hostedOn*.

C. Modeling Event-Emitting Components

The serverless paradigm exploits an event-driven programming model. Thus, there are cloud services that act as event sources in order to trigger functions running in FaaS (cf. Section III-A). Such services, e. g., Amazon's S3 object storage or SQS queueing service, are not intended to be installed. However, they most likely require proper configuration in order to be used. For example, in case of AWS S3 it is necessary to create and configure the specified bucket accordingly. In case of Amazon SQS, before a function or external application can use the queue, it has to be set up correctly. Therefore, similar to how we model an execution environment for a function, we

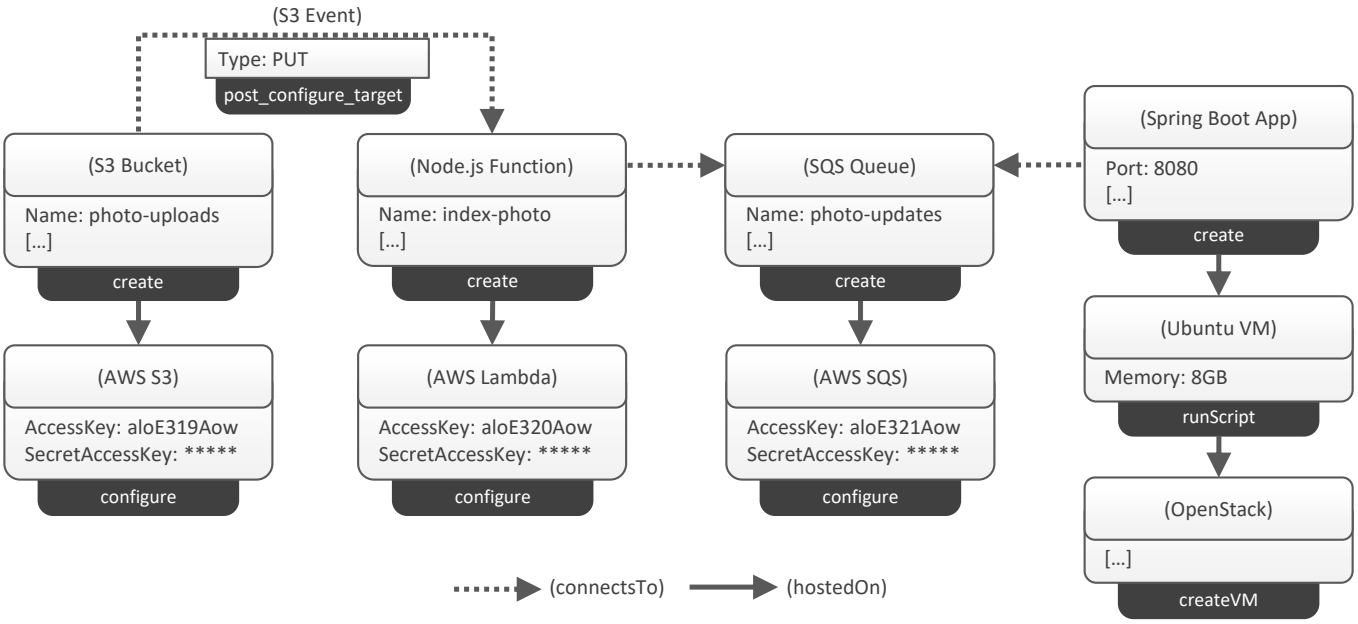

Figure 5. Full Topology Template utilizing and exposing standard lifecycle operations for the serverless application deployment.

express the semantics of a specific cloud service in a dedicated Node Type that provides all required information. Besides that, similar to functions, we express a specific configuration for a cloud service also as Node Template, e. g., a configuration for certain S3 buckets or SQS queues. Once the Node Templates are specified for an application deployment, the connection between a cloud service and the specific configuration is expressed using the normative TOSCA Relationship Type *hostedOn*.

D. Modeling Events and Event Flow

As mentioned previously, cloud services act as event sources on which other components can react on. Taking the motivating scenario as an example, we have a function that gets triggered whenever something happens with a certain AWS S3 bucket. To express this semantic between cloud services and functions, a special relation, or rather a kind of connection, is required. Therefore, we utilize the normative TOSCA Relationship Type *connectsTo* in order to specify such event connection. As there are different types of events, one can specify different Relationship Types that are derived from the normative type *connectsTo*. Examples include publish-subscribe events, e. g., when an object has been added to an object storage service or if a new entry has been added to a database table, or events reflecting HTTP requests and HTTP responses, e. g., when clients access a RESTful API that is implemented by an API Gateway service. A certain event flow is then modeled as a Relationship Template of such type between a corresponding cloud service configuration and a function. Furthermore, modeling restrictions can be expressed using the Relationship Type fields *valid_source_types* and *valid_target_types* such that only certain types can be used between nodes. As Relationship Types can define properties, we can use them to specify certain properties on these event types in order to configure specifically what kind of event will trigger a function. For example, in our AWS S3 scenario we defined a property *Type* on the event type *S3 Event* being able to specify that only *PUT* activities on the S3 bucket should trigger the function. Furthermore, TOSCA defines several normative operations that a Relationship Type may implement, such as *post_configure_source* or *post_configure_target*. These operations can be implemented by Implementation Artifacts in order to wire services with respective functions.

E. Modeling Limitations

In the serverless domain, there are several use cases that deal with events and functions. For example, one or more events trigger one function or one event triggers multiple functions executed in sequence or in parallel. Furthermore, the result of a function could trigger another function or even more complex, an event triggers a function and depending on the result different branches with different functions are triggered. In other words, ways are required to specify a workflow [19] based on functions. AWS, for example, provides “step function” primitives in order to specify a basic workflow [20]. Such workflows involve events and functions, whereas the interaction as well as how information can be passed between functions are modeled using a function graph.

Our presented approach does not support the ability to chain functions. Currently we focus only on the deployment of serverless applications including the configuration and wiring of certain cloud services with functions, based on one or more events. Furthermore, our focus of this work is more on the modeling of application deployments that exploit the serverless paradigm together with traditional application deployments. However, our approach provides a good basis for chained events, forming the fundamentals for further research.

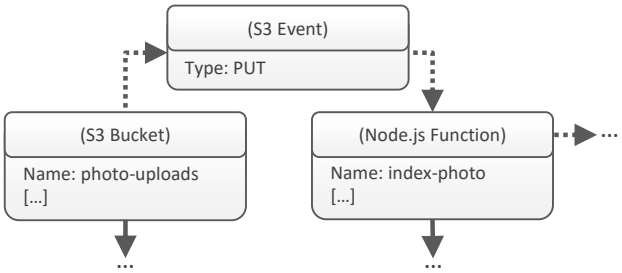

Figure 6. Alternative modeling approach for events and event flows.

IV. Automated Deployment

In terms of automating the deployment, our approach follows TOSCA's guideline to employ the recommended standard lifecycle, as suggested by the specification documentation [21], [8]. By following this recommendation, a TOSCA-compliant runtime can execute directly the resulting deployment model. Figure 5 shows the full Topology Template utilizing and exposing standard lifecycle operations for the serverless application deployment. TOSCA defines five normative operations a Node Type may have: *create*, *configure*, *start*, *stop*, and *delete*. Furthermore, TOSCA defines several normative operations (*configure* interface) on Relationship Types. Respective IAs may implement *post_configure_source* or *post_configure_target* operations in order to trigger certain actions. These operations are finally implemented by either Node Type Implementations or Relationship Type Implementations. For the sake of brevity, we only show one operation per node. For example, the implementation of the *create* operation on the *Node.js Function* node will setup the function metadata and upload the function code utilizing the cloud providers' programming interfaces. Following the standard lifecycle of TOSCA, a compliant runtime executes the operations in the following order: (1) configure "AWS S3", (2) create "S3 Bucket", (3) configure "AWS Lambda", (4) create "Node.js Function", and finally (5) configure "S3 Event" using the *post_configure_target* operation. Thus, using standard TOSCA constructs, the entire application can be provisioned automatically after modeling.

However, for the traditional part we still depict the notion that custom Management Operations were specified, e. g., as shown for the Ubuntu virtual machine as well as the OpenStack Node Type on the right hand side of the figure. The automated deployment can be realized, for example, by generating a deployment workflow fully automated. Breitenbücher et al. [10] presented an approach that is capable of deriving respective deployment steps by following the lifecycle and deployment order concept explained above. Thus, the deployment model is generated by the runtime itself. Another option is that the actual deployment steps are modeled imperatively by creating custom workflows that are executed during runtime [22]. For our example, we assume that an underlying TOSCA runtime is aware of how to deal with both, Ubuntu virtual machines and OpenStack execution environments, or can be easily extended in order to add such type awareness later onto the system.

V. Limitations and Discussion

In this section we discuss an alternative modeling approach for serverless applications, challenges with large application topologies in this context, and how to deal with event flows between different cloud execution environments.

A. Alternative Serverless Modeling Approach

In Fig. 6 we show an alternative approach how to model the serverless parts of an application deployment. In the alternative approach, respective events can be modeled similarly to how function deployments and components that emit events are represented, i. e., as Node Templates and Node Types. However, choosing the approach where events and the respective event flow are also modeled as nodes would unnecessarily clutter the application structure with elements that are redundant to express the same semantics. Especially in a graphical TOSCA modeling environment, such as Eclipse Winery [23], which allows the ability to create an application topology graphically, this approach will result in an unnecessarily overloaded user interface. Further, Moody [24], [25] also defines in his work about physics of visual notations that each visual symbol should have a single meaning in order to avoid *symbol overload*. Therefore, our assumption is that symbol overload is avoided by using edges as visual representation for event flows, which is similar to the semantics of connections and dependencies.

B. Challenges with Large Application Topologies

In aforementioned simplified use case example, only a deployment of one function is depicted. However, in real world scenarios more complex application deployments have to be modeled. With complex architectures the number of functions might be counted in hundreds. One possible solution is to introduce another level of abstraction, i. e., by separating logically related parts into different interconnected deployment models, or so-called Service Templates. A TOSCA runtime is then able to orchestrate these split parts. However, modeling of cross-dependencies among these parts needs to be further investigated. For example, additional constructs are required to model the event flow if one part of the application topology specifies a message queue and another part models a function that has to be triggered once a message is pushed to this queue. Further research can show how to model the configuration of cross-functional concerns, e. g., general DNS or special networking settings that are shared between application parts.

C. Multi-Cloud Event Flows

Another challenge that has not been addressed in this work is the modeling of event flows and the exchange of events between different serverless execution environments. For example, an AWS S3 event could trigger a function or serverless processing task on a privately hosted Apache OpenWhisk environment. In such a scenario, additional middleware or software components are required. Further research can derive the required software stacks implicitly from the deployment model in order to establish a connection between certain cloud provider services.

VI. Prototype and Application

To show the feasibility of our approach, we used and extended the existing TOSCA-based ecosystem OpenTOSCA. Thereby, we used the modeling tool Winery[1] in order to model the depicted example scenario and develop the required Node Types. Winery [23] is able to create the required modeling types according to the initial XML-based version of TOSCA [7], such as Node Types, Relationship Types, Deployment Artifacts, and Implementation Artifacts. Further, Winery comes with a topology modeling user interface in order to model a Service Template graphically based on created type definitions. The final application topology can then be exported as a CSAR. As TOSCA-compliant runtime, we used the OpenTOSCA Container[2], an open-source runtime to execute arbitrary TOSCA-based application deployments [26].

However, several extensions were required in order to fully support the depicted use case scenario, while all extensions have been merged into the stable branch of the individual repositories. Since Winery only supported the XML-based version of TOSCA, an extension was required so that interface and operation definitions on Relationship Types could be modeled according to TOSCA's Simple Profile [8], the latest YAML-based specification of the standard. We required this extension to define the depicted *configure* interface (cf. Fig. 5). With this addition, we were able to model the scenario used as example in the course of this paper. A similar extension was required for the OpenTOSCA Container to interpret the *configure* interface definitions correctly. The runtime was only aware of the standard lifecycle described by the older standard. Since the OpenTOSCA Container is a declarative runtime, it contains a component that derives the required deployment steps from a given application topology. This component is called *Plan Builder* [27], [10] and is able to derive and create a suitable BPEL [28] workflow, or so-called Build and Termination Plans, to execute and automate the deployment. Therefore, the extensions were primarily made in the Plan Builder component. With these additions, the Plan Builder is able to create a suitable workflow based on the specification of TOSCA's standard lifecycle of TOSCA (cf. Section IV).

Afterwards, we developed the respective IAs in order to implement the standard lifecycle interfaces and required operations (cf. Fig. 5). Using OpenTOSCA, these Implementation Artifacts are invoked using a component called *Management Bus* [29], [6]. OpenTOSCA supports several artifact types. The Management Bus provides an unified interface in order to invoke different kinds of implementations. For example, it is supported that IAs are implemented using plain Shell scripts, as web services, or even as Ansible Playbooks. As a result, the Build Plan generated by the Plan Builder specifies the order of operations to provision the application. During provisioning, or workflow execution, the operations are invoked by the Management Bus depending on the respective type.

[1]Eclipse Winery: https://github.com/eclipse/winery

[2]OpenTOSCA Container: https://github.com/OpenTOSCA/container

VII. Related Work

To the best of our knowledge, no published work suggest approaches to model event-driven serverless application deployments into multi-cloud environments using TOSCA. However, in the context of service-oriented architectures (SOA), Belli and Linschulte [30], [31] introduce an event-driven modeling approach for modeling behavior of web services for testing in real-time. Authors do not model in the context of automated application deployment, but propose a similar approach that relies on a directed graph, which uses nodes representing events and edges representing the flow of events. Laliwala and Chaudhary [32] present an event-driven service-oriented architecture (EDSOA) focused on modeling and automation of event-driven process chains. Authors use event calculus, a formal language for description of local events and time periods, to model event-driven business processes.

Multiple tools exist to support developers with "modeling for the cloud". Numerous existing deployment automation techniques and standards are based on the notion of *deployment models* [9], e. g., declarative or imperative deployment models [10]. The former describe deployment steps in a procedural manner and can be expressed, e. g., in the form of Shell scripts or Ansible Playbooks [14]. Declarative deployment models, in contrast, such as Chef [12] or Puppet [13], describe the desired result and a runtime drives the necessary deployment logic. Provider-specific modeling tools, such as AWS CloudFormation [33], focuses solely on their own platform. As a result, additional tooling is required for the coordination and deployment into multi-cloud environments. The Serverless Framework [15] provides the ability to deploy serverless applications to multiple cloud providers. It allows developers to specify platform-specific descriptions of functions and events. Once developers have decided on a platform, it is not intended to be changed any more. The resulting deployment model can only be used to deploy into a single cloud. Terraform [34], by HashiCorp, allows developers to deploy infrastructure resources, including multi-cloud provisioning scenarios, by specifying configuration files that are transformed into execution plans. Despite the focus on infrastructure-level resources, specification of serverless functions, e. g., AWS Lambda, is also possible. However, specification of event-driven aspects for serverless deployments is provider-specific.

None of the described methods fully supports modeling of multi-cloud serverless applications independently of certain deployment technologies. Moreover, deploying such architectures to hybrid clouds is non-trivial and has to be supported. TOSCA is an open, provider-agnostic standard that can be used to model and deploy cloud native applications [16] as well as complex application stacks combining multiple different technologies, execution runtimes, and cloud providers [17], [18]. In addition, TOSCA allows to model deployments of applications imperatively as well as declaratively [10]. Thus, the resulting topology models are flexible, portable and can easily be interchanged, e. g., to combine various technology stacks with different cloud provider offerings.

VIII. Conclusion and Future Work

In this work, we introduced an event-driven deployment modeling approach using TOSCA. The use of standard TOSCA modeling constructs to model the deployment of (i) functions, (ii) components that emit events, and (iii) events as well as event flows were presented and prototypically evaluated in the course of this paper. We showed how our approach can be used for multi-cloud application deployments where traditional software components—running in private cloud environments—can be extended using the serverless computing paradigm. Being fully compliant to TOSCA's standard lifecycle, we showed that a TOSCA-compliant runtime can automatically execute such modeled application deployments.

As future work, we aim to tackle the challenge of modeling multi-cloud event flows where events can be exchanged between different cloud environments. Further, we tackle the challenge to model more complex serverless application topologies where logical parts can be split up while considering shared configuration settings and cross-dependencies. Lastly, we aim to work on a full end to end deployment of certain serverless computing platforms that are modeled in topologies, such that Apache OpenWhisk is automatically deployed.

Acknowledgment

This work is partially funded by the BMWi projects *SePiA.Pro* (01MD16013F), *SmartOrchestra* (01MD16001F), and *IC4F* (01MA17008G).

References

[1] K. Fromm. (2012) Why The Future Of Software And Apps Is Serverless. [Online]. Available: https://readwrite.com/2012/10/15/why-the-future-of-software-and-apps-is-serverless

[2] M. Roberts. (2016) Serverless Architectures. [Online]. Available: http://martinfowler.com/articles/serverless.html

[3] Cloud Native Computing Foundation. (2018) CNCF Serverless Whitepaper v1.0. [Online]. Available: https://github.com/cncf/wg-serverless/tree/master/whitepapers/serverless-overview

[4] I. Baldini *et al.*, "Serverless Computing: Current Trends and Open Problems," in *Research Advances in Cloud Computing*. Springer Singapore, 2017, pp. 1–20.

[5] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems (USITS 2003)*. USENIX, Jun. 2003.

[6] J. Wettinger *et al.*, "Unified Invocation of Scripts and Services for Provisioning, Deployment, and Management of Cloud Applications Based on TOSCA," in *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014)*. SciTePress, Apr. 2014, pp. 559–568.

[7] OASIS, *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0*, Organization for the Advancement of Structured Information Standards (OASIS), 2013.

[8] ——, *TOSCA Simple Profile in YAML Version 1.1*, Organization for the Advancement of Structured Information Standards (OASIS), 2018.

[9] A. Bergmayr *et al.*, "A Systematic Review of Cloud Modeling Languages," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–38, Feb. 2018.

[10] U. Breitenbücher *et al.*, "Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA," in *International Conference on Cloud Engineering (IC2E 2014)*. IEEE, Mar. 2014, pp. 87–96.

[11] Amazon Web Services, Inc. (2018) Aws serverless web application workshop architecture. [Online]. Available: https://aws.amazon.com/de/serverless/build-a-web-app

[12] Opscode, Inc. (2018) Chef Official Site. [Online]. Available: http://www.opscode.com/chef

[13] Puppet Labs. (2018) Puppet Official Site. [Online]. Available: https://jujucharms.com

[14] Red Hat, Inc. (2018) Ansible Official Site. [Online]. Available: https://www.ansible.com

[15] Serverless, Inc. (2018) Serverless Framework. [Online]. Available: https://serverless.com/framework

[16] M. Wurster, U. Breitenbücher, M. Falkenthal, and F. Leymann, "Developing, Deploying, and Operating Twelve-Factor Applications with TOSCA," in *In Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services, Salzburg, Austria, December 4-6, 2017*. ACM, Dec. 2017, pp. 519–525.

[17] A. C. Franco da Silva *et al.*, "Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments," in *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER)*. SciTePress Digital Library, Jun. 2017, pp. 358–367.

[18] K. Saatkamp, U. Breitenbücher, F. Leymann, and M. Wurster, "Generic Driver Injection for Automated IoT Application Deployments," in *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services; Salzburg, Austria, December 4-6, 2017*. ACM, Dec. 2017, pp. 320–329.

[19] F. Leymann and D. Roller, *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000.

[20] Amazon Web Services, Inc. (2018) AWS Step Functions Official Site. [Online]. Available: https://aws.amazon.com/de/step-functions

[21] OASIS, *Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0*, Organization for the Advancement of Structured Information Standards (OASIS), 2013.

[22] C. Endres *et al.*, "Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications," in *Proceedings of the 9th International Conference on Pervasive Patterns and Applications (PATTERNS)*. Xpert Publishing Services, Feb. 2017, pp. 22–27.

[23] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann, "Winery – A Modeling Tool for TOSCA-based Cloud Applications," in *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*. Springer, Dec. 2013, pp. 700–704.

[24] D. L. Moody, "The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, Nov. 2009.

[25] ——, "The Physics of Notations: A Scientific Approach to Designing Visual Notations in Software Engineering," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, May 2010, pp. 485–486.

[26] T. Binz *et al.*, "OpenTOSCA – A Runtime for TOSCA-based Cloud Applications," in *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*. Springer, Dec. 2013, pp. 692–695.

[27] K. Képes, U. Breitenbücher, M. P. Fischer, F. Leymann, and M. Zimmermann, "Policy-Aware Provisioning Plan Generation for TOSCA-based Applications," in *Proceedings of the 11th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2017)*. Xpert Publishing Services, Sep. 2017, pp. 142–149.

[28] OASIS, *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, Organization for the Advancement of Structured Information Standards (OASIS), 2007.

[29] J. Wettinger, T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "Streamlining Cloud Management Automation by Unifying the Invocation of Scripts and Services Based on TOSCA," *International Journal of Organizational and Collective Intelligence (IJOCI), Volume 4, Issue 2*, pp. 45–63, Apr. 2014.

[30] F. Belli and M. Linschulte, "Event-Driven Modeling and Testing of Web Services," in *2008 32nd Annual IEEE International Computer Software and Applications Conference*, Jul. 2008.

[31] ——, "Event-Driven Modeling and Testing of Real-Time Web Services," *Service Oriented Computing and Applications*, vol. 4, no. 1, pp. 3–15, Mar. 2010.

[32] Z. Laliwala and S. Chaudhary, "Event-Driven Service-Oriented Architecture," in *2008 International Conference on Service Systems and Service Management*, Jun. 2008, pp. 1–6.

[33] Amazon Web Services, Inc. (2018) AWS CloudFormation Official Site. [Online]. Available: https://aws.amazon.com/de/cloudformation

[34] HashiCorp. (2014) Terraform. [Online]. Available: https://www.terraform.io