Avoiding Vendor-Lockin in Cloud Monitoring using Generic Agent Templates*

Mathias Mormul, Pascal Hirmer, Christoph Stach, and Bernhard Mitschang

Institute for Parallel and Distributed Systems, University of Stuttgart,
Universitätsstraße 38, D-70569 Stuttgart, Germany
firstname.lastname@ipvs.uni-stuttgart.de

Abstract. Cloud computing passed the hype cycle long ago and firmly established itself as a future technology since then. However, to utilize the cloud optimally, and therefore, as cost-efficiently as possible, a continuous monitoring is key to prevent an over- or under-commissioning of resources. However, selecting a suitable monitoring solution is a challenging task. Monitoring agents that collect monitoring data are spread across the monitored IT environment. Therefore, the possibility of vendor lock-ins leads to a lack of flexibility when the cloud environment or the business needs change. To handle these challenges, we introduce *generic agent templates* that are applicable to many monitoring systems and support a replacement of monitoring systems. Solution-specific technical details of monitoring agents are abstracted from and system administrators only need to model generic agents, which can be transformed into solution-specific monitoring agents. The transformation logic required for this process is provided by domain experts to not further burden system administrators. Furthermore, we introduce an agent lifecycle to support the system administrator with the management and deployment of generic agents.

Keywords: Vendor Lock-in · Cloud Monitoring · Monitoring Agents · Genericity.

1 Introduction

Cloud computing passed the hype cycle long ago and firmly established itself as a future technology since then. Studies still predict a further increase of revenue of the worldwide public cloud services market by more than 15% in 2019, whereby Infrastructure-as-a-Service (IaaS) is the fastest growing segment [6]. The main advantages of cloud computing are the self-service based commissioning and decommissioning of resources (e. g., virtual machines) as needed, a flexible pay-per-use model, and seemingly infinite scalability to enable a perfectly fitted IT infrastructure for each company [19]. However, to utilize the cloud optimally, i. e., as cost-efficiently as possible, resource utilization must be known or analyzed

* This work is partially funded by the BMWi project IC4F (01MA17008G).

In: Abramowicz W. and Klein G. (Eds.) Business Information Systems. BIS 2020. Lecture Notes in Business Information Processing, vol 389. Springer, Cham, pp. 267–278, 2020.

to prevent an over- or under-commissioning of resources. As resource utilization may vary over time, a continuous monitoring of cloud resources is essential [13].

However, the flexibility provided by cloud computing introduces new challenges to the management and monitoring of the IT environment. Most current monitoring systems were originally not designed to monitor cloud environments with highly volatile virtual machines that may come and go in a matter of minutes, but rather for traditional hardware and slowly changing environments [27,1]. When business needs change, which leads to changes in the cloud environment, the use of a different monitoring system that better suits these business needs might be favorable. This challenge is amplified by the current state of IT departments, which are oftentimes Popovic2017 and lack resources for new technologies [3,21]. Hence, they have problems keeping up with state-of-the-art technologies and deploying them in the company. Replacing a monitoring system is a time-consuming, error-prone, and therefore, expensive task that cannot be handled by every IT department. In general, each monitoring system comes with its own monitoring agent (in the following, only called agent), a software process collecting monitoring data inputs and reporting them to a monitoring system [18]. Therefore, replacing a monitoring system leads to the replacement of all of its agents. Usually, the agents are written in different programming languages, which requires learning new syntax and semantics. Furthermore, features of the agents, e. g., aggregation or filtering, may differ from agent to agent. Lastly, the newly created agents need to be deployed onto the virtual machines, which oftentimes needs to be conducted manually.

We address these challenges by introducing generic agent templates to create an abstraction level for the modeling of agents. Agents are modeled only once in an abstract and generic way and domain experts provide the transformation logic required to transform these agents into executable agents for specific monitoring systems. Therefore, the complexity and time needed to replace a monitoring system are heavily reduced, which also reduces monetary losses. Furthermore, we introduce an extended lifecycle for agents to support the management and automatic deployment of agents to enable scalability, which is essential in large-scaled cloud environments.

The remainder of this paper is structured as follows: Section 2 introduces a motivating scenario and its requirements for cloud monitoring. In Sect. 3, we present the generic agent templates and the extended agent life cycle. Section 4 discusses related work and lastly, Sect. 5 contains the conclusion of this paper as well as future work.

2 Motivating Scenario

Modern cloud computing leads to a plethora of data points that need to be monitored. For illustration purposes, in this motivating scenario, we only consider the monitoring of the CPU of a virtual machine running a Linux server. In general, each IT infrastructure monitoring (ITIM) system provides preconfigured agents for the most common tasks. Ligus [18] defines an agent as *"a software*

process that continuously records data inputs and reports them to a monitoring system". The agent collects the CPU information from the host system and sends it to a monitoring server where the data are stored and can be accessed by the system administrator. In addition to the CPU measurement, metadata such as the host name and timestamp are sent as well. Oftentimes it is sensible to perform certain actions on the monitoring data before sending it to the server which requires changes to the configuration of the agent. Those actions fall into one of two categories: processing—actions performed on each data sample, i. e., each collected CPU load measurement—and aggregation—actions performed on a set of data samples. In our example, processing could be used to transform the timestamp (usually in UNIX time format) into a human-readable format. Aggregation could be used to aggregate multiple measurements for a specified time interval, e. g., 60 seconds, and compute the mean, max, or min value and only send this result to reduce network traffic.

This way, the system administrator can create arbitrary agent configurations that suit the business needs of the company. Based on these configurations, agents are deployed on each virtual machine. Now, business needs or the cloud environment may change that require an adaptation of the monitoring system. For example, Linux servers may be replaced by Windows servers. However, not every ITIM system supports Windows as well as Linux. Therefore, either an additional ITIM system must be added that supports Windows, which increases management complexity since two systems need to be managed, or the original system must be replaced by a system that supports both operating systems.

In both cases, the system administrator has to recreate the agent configurations for the new ITIM system. Furthermore, as mentioned in the previous section, agents differ in their programming language and functionalities, which the system administrator has to learn first. Lastly, the agents must be deployed again. This often occurs manually, which leads to scalability issues when considering large-scale cloud environments. Furthermore, changes at runtime to adapt to a dynamic environment and automatic deployment are not supported by most monitoring systems. All this results in a time-consuming process whereby the actual task, i. e., CPU monitoring, stays exactly the same.

Out of the above, we deduce following requirements:

*R*1: **Genericity:** Instead of modeling solution-specific agents, a generic way for modeling agents to abstract from implementation details is required. This way, system administrators only need to model an agent once. Domain experts provide the transformation logic required to transform generic agents into solution-specific agents. This way, system administrator can save time and money and focus on translating business needs into generic agents instead of learning technical details of monitoring systems.

*R*2: **Expressiveness:** The expressiveness of generic agents needs to support all common tasks of an agent, such as processing and aggregation of monitoring data. This guarantees that many monitoring systems can be supported by generic agents.

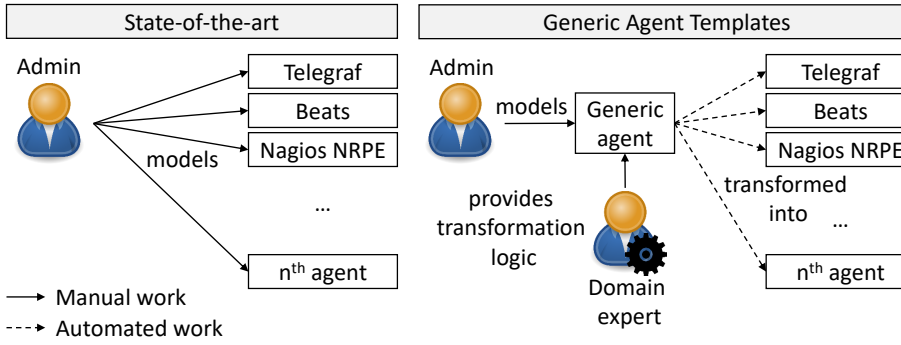

Fig. 1: Left: Admin has to model n agents; Right: Admin only has to model one agent

*R*3: **Automation:** With rising complexity in cloud environments, manual tasks must be minimized. Automation requires management in regard to the modeling and deployment of agents to unburden system administrators and enable them to focus on actual problems in the IT environment.

3 Generic Agent Templates

We introduce generic agent templates to fulfill the above-mentioned requirements $R1 - R3$. System administrators model agents in accordance to the business needs and decide which resources need to be monitored, and how often and what kind of actions are performed on the monitoring data (cf. our motivating scenario). As shown in Fig. 1 (left), currently, a system administrator has to model n different agents to support n monitoring systems, even if the underlying tasks, e. g., *collect CPU load, transform timestamp into human-readable format, calculate mean over one minute, send to database*, stay the same. This process is done manually and in each company separately.

By introducing generic agent templates (Fig. 1 (right), the system administrator only has to model this underlying task once in a generic way. However, those generic agents are non-executable and need to be transformed into a solution-specific agent that is used within the specific company. Because generic agents conform to a predefined schema, domain experts with detailed technical knowledge about the several solution-specific agents can provide the transformation logic to transform generic agents into executable agents for the desired monitoring system. The transformation logic can be shared across all companies and, therefore, only needs to be provided once per supported agent.

To further support system administrators, we introduce an extended lifecycle for agents, as shown in Fig. 2. On the left, the current, inadequate lifecycle is shown consisting of two manual tasks *Modeling* and *Manual Deployment*, both performed by system administrators. Each time an agent is modeled or changed,

the deployment onto each virtual machine must be repeated. To cope with the issues of this approach, we introduce an extended life cycle as shown in Fig. 2 (right), which comprises of the following four phases:

- *Phase 1 - Modeling:* The demand for genericity and usability requires changes in the modeling phase. We introduce *generic agent templates* to create an abstraction layer that is generic and supports an easy modeling of agents that can be used for multiple monitoring systems.
- *Phase 2 - Transformation:* Since the modeled agents are generic and non-executable, the generic agents are transformed into executable, solution-specific agents. To not further burden the system administrator, this task is automated since domain experts provide the transformation logic.
- *Phase 3 - Automatic Deployment:* Instead of deploying agents manually, we automate the deployment to tackle scalability issues in large-scale cloud environments using standards-compliant technologies.
- *Phase 4 - Adaptation:* We present an agent management to support adaptation of agents to the dynamically changing environment at runtime.

Phase 1 - Modeling: In general, agents are integrated into specific monitoring systems and cannot be used for different monitoring systems. There are a few exceptions, e. g., Nagios' NRPE agent, which, due to its wide distribution, is supported by many different monitoring systems. However, there are no agents that can be used in all monitoring systems. Also, each agent differs in its syntax, used programming language, and functionality. To enable genericity, we introduce the novel concept of *generic agent templates* to achieve an abstraction layer for the modeling of agents.

The basis of a generic agent template is the agent pipeline containing the four components *Input, Processor, Aggregator*, and *Output* nodes as shown in Fig. 3. This pipeline is based on the plugin-based architecture of agents, e.g., *Telegraf* [16] (TICK-Stack) or *Beats* [10] (ELK-Stack) and enables modeling a flexible and extendable agent due to its modular structure. With those components, a system administrator can model generic agents (exemplary configuration in Fig. 4) that can be transformed by the *Agent Mapper* to solution-specific agents. In the following, we describe the nodes and pipeline concept in detail.

- **Input Node** defines what metric the agent is collecting (e. g., *CPU load*). Besides meta data, such as *ID* and *name* of the node, for each input node, the sampling frequency can be set individually, e. g., *CPU load* is collected every second whereas *RAM load* is collected every ten seconds. Data from the Input Node can be sent to *Processor* and *Aggregator* nodes for further processing, or directly to an *Output Node*. A single agent can contain multiple Input Nodes.
- **Processor Node** is an optional node which contains functions that can be executed on single data samples. Examples are transformations of data (e. g., UNIX date to human readable date) or filtering (e. g., $if\ value < 200MB$). At the end, a data sample that passes a Processor Node may receive a tag

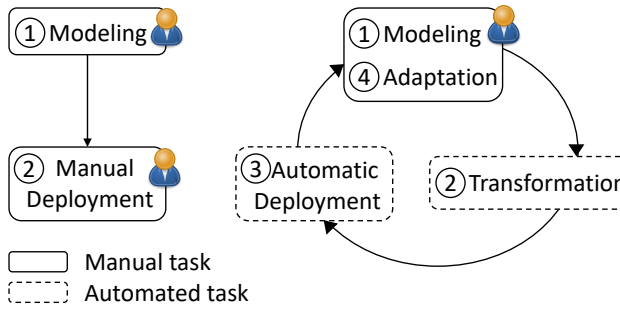

Fig. 2: Left: Current agent lifecycle; Right: New, extended agent lifecycle

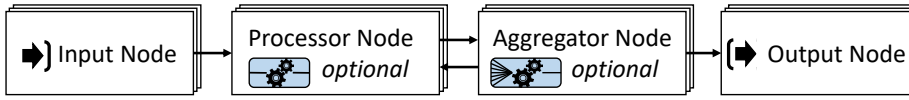

Fig. 3: Agent pipeline

that may be used for further routing inside the agent pipeline. Based on those tags, data are sent to Aggregator Nodes or Output Nodes.

- **Aggregator Node** is an optional node and acts similar to a Processor Node. The difference is that functions are not performed on single data samples, but on a set of data samples. Therefore, a window is defined, in which the Aggregator Node calculates statistics, e. g., calculate the mean CPU and RAM load over one minute. Again, tags can be added for further routing and data can be sent to Processor Nodes or Output Nodes.
- **Output Node** represents the endpoint of a pipeline and defines where data are sent to from an agent's perspective (e. g., to a database). Multiple Output Nodes can exist and can be chosen based on tags added to the data.

The result of the modeling process is a generic agent in form of a JSON document. We define a schema[1] for the node definitions using JSON schema. A small excerpt of this schema for the input node is shown in **??** 1.1. The *id* is used as an identifier within an agent template. The *type* denotes the type of input node referring to premodeled input types like CPU input or RAM input. The *config* contains configurable parameters of the input node, such as sampling frequency. Finally, *next* describes the next node within the pipeline. Based on this schema, we implemented a graphical, web-based modeling tool[2] to ease the modeling and alteration of agents without the need to understand all the specifics of a monitoring system.

[1] https://github.com/mormulms/agent-centric-monitoring/blob/master/generic-agent/mona-template-editor/src/assets/schema.json

[2] https://github.com/mormulms/agent-centric-monitoring/blob/master/generic-agent/mona-template-editor

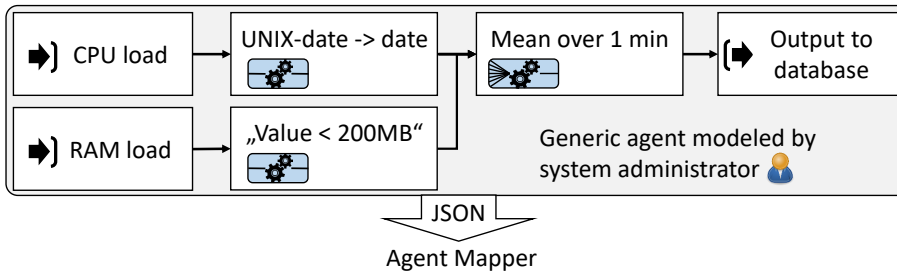

Fig. 4: Exemplary generic agent configuration sent to Agent Mapper

```
"GenericInput": {
   "type": "object",
   "properties": {
      "id": {
         "$ref": "#/definitions/InputId"
      },
      "type": {
         "$ref": "#/definitions/NodeType"
      },
      "config": {
         "$ref": "#/definitions/InputConfig"
      },
      "next": {
         "$ref": "#/definitions/NextArray"
      }
   },
   ...
}
```

Listing 1.1: Excerpt of the input node definition

Phase 2 - Transformation: The result of a modeled agent pipeline is a generic agent template. The *Agent Mapper* is the component that receives this generic agent template and transforms it into an executable agent for a specific monitoring system. Of course, for each supported monitoring system, the transformation logic must be implemented. Therefore, the complexity and variance of different monitoring systems is not diminished but rather shifted from the end user to a few domain experts who implement the transformation logic for specific monitoring agents. Those implementations must be shared publicly to gain an advantage. So far, in our current prototype, we only support transformations to Telegraf. To support further monitoring systems and agents in the future, we created an extendable, modular architecture for the Agent Mapper. Agents, such as Telegraf, already have the functionalities for processing and aggregating monitoring data and, therefore, are simple to transform to. However, agents that do not support

one or more of those functionalities require a more complicated transformation. The most basic agent always has a means to collect data and send it to the destination, i. e., the monitoring server. However, if a modeled generic agent contains functionality such as aggregation and the system administrator wants to transform this generic agent into a solution-specific agent that does not support this functionality, there are two options: (i) the system administrator receives a warning that the transformation is not possible and needs to select a different solution-specific agent, which the generic agent should be transformed into. The second option is the use of a Complex Event Processing (CEP) engine or similar engines. The agent sends its collected monitoring data to this engine. Then, the processing and/or aggregation nodes can be translated into CEP queries to perform the needed functionalities and forward the monitoring data back to the agent which further forwards it to the monitoring server. This greatly increases the impact of generic agents since a transformation to many existing monitoring agents is possible.

Phase 3 - Automatic Deployment: Especially in large-scaled scenarios with a large number of VMs, any manual task becomes cumbersome, error-prone, and simply does not scale. Therefore, an automatic deployment is essential [29,26]. There are many existing tools and platforms for automated deployment, such as *Docker* [7] or *Vagrant* [12]. A few monitoring systems like *Splunk* [24] support this feature as well. However, a standardized deployment framework is desirable, since technologies tend to disappear over time.

An established deployment standard is the Topology and Orchestration Specification for Cloud Applications (*TOSCA*) [20] of the Organization for the Advancement of Structured Information Standards (*OASIS*). TOSCA enables a two-step software deployment approach. First, a topology template is created, modeling the application, platform, and infrastructure components. Second, this topology is used to deploy the components in the corresponding infrastructure.

One implementation of the TOSCA standard is OpenTOSCA [28]. OpenTOSCA provides an eco system consisting of the TOSCA topology modeler Winery [8], the OpenTOSCA container that handles the actual software deployment based on the topology, and a self-service portal called Vinothek [4]. TOSCA and its implementation OpenTOSCA can be used in our approach for automated deployment of monitoring agents. Since TOSCA is a standard, it provides a high degree of applicability and is future-secure.

We modeled a topology template using OpenTOSCA to deploy a monitoring system and a template to deploy agents. If a new virtual machine is started, an agent is deployed on it to guarantee a monitoring from the beginning.

Phase 4 - Adaption:

According to Gartner [2], flexibility against a changing IT architecture is a key objective when investing in new monitoring systems. The modeling of agents is influenced by the current business needs and the status of the current cloud environment. However, both of these variables may change over time. In this case, starting a new agent life cycle is excessive when only minor changes to the agent are required. Instead, the user should be able to access the previously

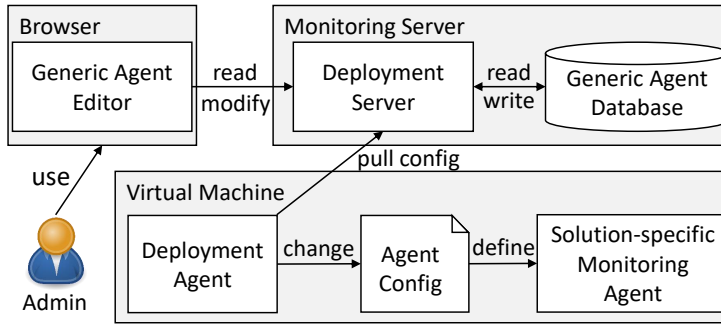

Fig. 5: Architecture for automatic adaptations at runtime

modeled generic agents and change them to fit the current needs. The changes are then propagated to all agents originating from this generic agent. To support this process, we implemented a prototype as shown in Fig. 5. The user models a new generic agent or changes an existing one via the web-based modeling tool, which is connected to the *Deployment Server* on the *Monitoring Server*. The deployment server is responsible for the management of the generic agents, which are stored in the *Generic Agent Database*. On the virtual machines, a *Deployment Agent* periodically requests the configuration (the transformed generic agent) for the agent and resolves differences between the retrieved configuration and the currently used one. In case of changed configurations or failures, the deployment agent automatically restarts the agent.

4 Related Work

The transformation of abstract models to a concrete executable implementation is a commonly used means in order to abstract from technical details. Consequently, domain users are able to create models on a high level of abstraction without requiring technical knowledge about their realization. The use of a single abstract model leads to genericity and reduces the threat of vendor lock-in, since it is not dependent on specific technologies. In the following, approaches are described that aim at a similar approach for transforming abstract models into concrete ones.

Falkenthal et al. [11] aim towards a generic approach to transform abstract pattern languages to concrete solution implementations. Since their approach is generic, they do not focus on a specific domain but rather discuss how such a transformation can be conducted in general. In this paper, we apply this approach to the concrete domain of monitoring by introducing a mapping of generic monitoring templates to concrete, executable implementations.

Eilam et al. [9] introduce an approach for model-based provisioning and management of applications. Through transformations, application topologies are mapped onto different levels of abstraction in order to finally create exe-

cutable implementations that can be deployed. However, Eilam et al. require a premodeling of concrete implementation artifacts, which is not necessary in our approach.

Furthermore, Eilam et al. introduce a combination of a model-based and workflow-based approach for the automated provisioning of the transformed applications. This approach creates a provisioning model based on workflow technology that can be used for automated deployment. In this paper, we also introduce an approach how the templates can be automatically deployed after transformation, which uses existing software deployment technologies instead of this heavy-weight workflow-based approach.

Similar approaches regarding the agent templates are introduced by Künzle et al. [17] and Cohn et al. [5] that use artifact-centric approaches. In these approaches, so-called business artifacts are created, i .e., abstract representations of software components with the goal of hiding technical details. These artifacts can be mapped onto concrete executable implementations, as shown by Sun et al. [25]. However, the business artifacts of Künzle et al. and Cohn et al. are vaguely described, i .e., only on a conceptual level, an example application is missing. In this paper, we introduce a concrete scenario our concepts can be applied to.

Reimann [22] introduces generic patterns for simulation workflows that are also mapped onto concrete executable implementations, in his approach onto workflow fragments provided in the Business Process Execution Language (BPEL) [30]. However, Reimann focuses exclusively on workflows. In contrast, we focus on agents and model their capabilities through tailored templates.

In our previous work [15], we introduced so-called Situation Templates, which represent abstract descriptions of situations to be recognized without the necessity to provide technical implementation details. These situation templates can be mapped onto various formats, for example, complex event processing queries. In this paper, we adapt this concept to the domain of cloud monitoring to make it suitable for modeling of agents.

5 Conclusion

The monitoring of complex cloud environments can lead to several challenges. Selecting an unsuitable monitoring system or evolving business needs may lead to a required replacement of the monitoring system. However, monitoring agents are spread across the IT environment and, oftentimes, can be tightly integrated into the monitoring system. Therefore, the replacement of the agents is a time-consuming task and the modeling of new agents requires technical expertise. For this, we introduce generic agent templates to unburden system administrators by creating an abstraction to the modeling of agents. System administrators model generic agents once and domain experts provide the transformation logic required to transform the generic agents to solution-specific agents. This way, generic agents can be transformed into several solution-specific agents without further additional work. Expressiveness is provided by the agent pipeline consisting

of Input, Processor, Aggregator, and Output Nodes. Of each node, multiple instances can be created so that system administrators can model arbitrary agent configurations. Furthermore, if solution-specific agents do not support some of those functionalities, e. g., processing, a transformation to CEP queries is also possible to even support agents with missing functionalities. Lastly, an extended life cycle supports management, automatic deployment, and adaptation of agents at runtime using state-of-the-art and standardized technologies. Transformations to solution-specific agents are automated via the Agent Mapper. The automatic deployment is enabled using the deployment standard TOSCA. Adaptations to generic agents at runtime are automatically propagated to all according agents.

In future work, we plan to validate our approach on multiple monitoring systems and extend the usage to the IoT domain by using the open-source IoT platform MBP [23,14]. Furthermore, we plan to apply the concept of generic templates to other parts of the monitoring system as well to further support a possible replacement and reduce the risk of vendor lock-ins. Similar to agent configurations, alerting rules are defined by system administrators to inform them about problems in the monitored IT environment. Therefore, analogous to generic agent templates, generic alerting rules may present similar benefits.

References

1. Alcaraz Calero, J.M., Gutiérrez Aguado, J.: Comparative Analysis of Architectures for Monitoring Cloud Computing Infrastructures. Future Generation Computer Systems **47**(C), 16–30 (2015). https://doi.org/10.1016/j.future.2014.12.008
2. Bhalla, V., Prasad, P.: Use This 4-Step Approach to Architect Your IT Monitoring Strategy. Research report, Gartner (7 2018), https://www.gartner.com/en/documents/3882275/use-this-4-step-approach-to-architect-your-it-monitoring
3. Bowen, D.A.: Challenges Archivists Encounter Adopting Cloud Storage for Digital Preservation. In: IKE '18 (2018)
4. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F.: Vinothek – A Self-Service Portal for TOSCA. In: ZEUS '14 (2014)
5. Cohn, D., Hull, R.: Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering **32**(3), 3–9 (2009)
6. Costello, K.: Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.3 Percent in 2019. Market analysis, Gartner (9 2019), https://www.gartner.com/en/newsroom/press-releases/2018-09-12-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2019
7. Docker, Inc.: Docker (2019), `https://www.docker.com/`
8. Eclipse: Winery (2019), `https://winery.readthedocs.io/en/latest/`
9. Eilam, T., Elder, M., Konstantinou, A.V., Snible, E.: Pattern-based composite application deployment. In: IM '11 (2011). https://doi.org/10.1109/INM.2011.5990694
10. Elasticsearch B.V.: Beats (2019), `https://www.elastic.co/beats/`
11. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F.: From pattern languages to solution implementations. In: PATTERNS '14 (2014)
12. HashiCorp: Vagrant (2019), `https://www.vagrantup.com/`

13. Hauser, C.B., Wesner, S.: Reviewing Cloud Monitoring: Towards Cloud Resource Profiling. In: CLOUD '18 (2018). https://doi.org/10.1109/CLOUD.2018.00093
14. Hirmer, P., Breitenbücher, U., Franco da Silva, A.C., Képes, K., Mitschang, B., Wieland, M.: Automating the Provisioning and Configuration of Devices in the Internet of Things. Complex Systems Informatics and Modeling Quarterly **9**, 28–43 (2016). https://doi.org/10.7250/csimq.2016-9.02
15. Hirmer, P., Wieland, M., Schwarz, H., Mitschang, B., Breitenbücher, U., Sáez, S.G., Leymann, F.: Situation Recognition and Handling Based on Executing Situation Templates and Situation-Aware Workflows. Computing **99**(2), 163–181 (2017). https://doi.org/10.1007/s00607-016-0522-9
16. InfluxData: Telegraf (2019), `https://www.influxdata.com/time-series-platform/telegraf/`
17. Künzle, V., Reichert, M.: PHILharmonicFlows: Towards a Framework for Object-Aware Process Management. Journal of Software Maintenance and Evolution: Research and Practice **23**(4), 205–244 (2011). https://doi.org/10.1002/smr.524
18. Ligus, S.: Effective Monitoring and Alerting. O'Reilly Media Inc. (2012)
19. Mell, P., Grance, T.: The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology. NIST Special Publication 800-145, NIST – National Institute of Standards and Technology, U.S. Department of Commerce (2011)
20. Palma, D., Spatzier, T.: Topology and Orchestration Specification for Cloud Applications Version 1.0. Oasis standard, OASIS (11 2013), http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html
21. Popovic, J.: All Things Are Digital In Business, But Finding Digital Talent Is A Tall Order. Press release, Robert Half Technology (11 2017), http://rh-us.mediaroom.com/2017-11-01-All-Things-Are-Digital-In-Business-But-Finding-Digital-Talent-Is-A-Tall-Order
22. Reimann, P., Schwarz, H., Mitschang, B.: A Pattern Approach to Conquer the Data Complexity in Simulation Workflow Design. In: OTM '14 (2014). https://doi.org/10.1007/978-3-662-45563-0_2
23. Franco da Silva, A.C., Hirmer, P., Schneider, J., Ulusal, S., Tavares Frigo, M.: MBP: Not Just an IoT Platform. In: PerCom Workshops) '20 (2020)
24. Splunk, Inc.: Splunk (2019), `https://www.splunk.com/`
25. Sun, Y., Su, J., Wu, B., Yang, J.: Modeling data for business processes. In: ICDE '14 (2014). https://doi.org/10.1109/ICDE.2014.6816722
26. Taherizadeh, S., Jones, A.C., Taylor, I., Zhao, Z., Stankovski, V.: Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review. Journal of Systems and Software **136**, 19–38 (2018). https://doi.org/10.1016/j.jss.2017.10.033
27. Trihinas, D., Pallis, G., Dikaiakos, M.D.: Monitoring Elastically Adaptive Multi-Cloud Services. IEEE Transactions on Cloud Computing **6**(3), 800–814 (2018). https://doi.org/10.1109/TCC.2015.2511760
28. University of Stuttgart: OpenTOSCA (2019), `https://www.opentosca.org/`
29. Ward, J.S., Barker, A.: Observing the clouds: a survey and taxonomy of cloud monitoring. Journal of Cloud Computing **3**, 24:1–24:30 (2014). https://doi.org/10.1186/s13677-014-0024-2
30. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall (2005)