Evaluating the Resolution of AERs to Establish Diverse Application Environments

Johannes Wettinger, Vasilios Andrikopoulos, and Frank Leymann

Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstr. 38, Stuttgart, Germany
{wettinger,andrikopoulos,leymann}@iaas.uni-stuttgart.de

This little report complements the paper entitled *Enabling DevOps Collaboration and Continuous Delivery Using Diverse Application Environments*[1], which we published and presented at the CoopIS 2015 conference. In this report we present an evaluation, which is based on the Web shop application we introduced in the original paper as shown in Figure 1. Initially, we expressed the application environment requirements (AERs) attached to different parts of the application topology as AER predicates, bundled by an AER attachment map:

$$\begin{aligned}
req_{front,all} &= P^{im,include}_{WebServer}(\text{`Web Server'})\\
req_{front,dev} &= P^{im,include}_{Git}(\text{`Git'}) \wedge P^{im,equals}_{Grunt}(\text{`Grunt', `version', `0.4.5'})\\
req_{back,all} &= P^{im,eqGr}_{PHP}(\text{`PHP', `version', `5.5'}) \wedge P^{im,include}_{WebServer}(\text{`Web Server'})\\
req_{back,dev} &= P^{im,include}_{Git}(\text{`Git'}) \wedge P^{im,eqGr}_{PHPUnit}(\text{`PHPUnit', `version', `4.3'})\\
&\quad \wedge P^{im,equals}_{PhpStorm}(\text{`PhpStorm', `version', `8.0'})\\
req_{back,prod} &= P^{ev,include}_{Ubuntu}(\text{`Ubuntu OS'})\\
req_{db,all} &= P^{im,eqGr}_{MySQL}(\text{`MySQL', `version', `5.7'})
\end{aligned}$$

$$map(n,e) = \begin{cases}
req_{front,all} \wedge req_{front,dev} & \text{if } \mathit{label}(n) = \text{`Front-end'} \wedge e = \mathit{dev},\\
req_{front,all} & \text{if } \mathit{label}(n) = \text{`Front-end'} \wedge e = \mathit{prod},\\
req_{back,all} \wedge req_{back,dev} & \text{if } \mathit{label}(n) = \text{`Back-end'} \wedge e = \mathit{dev},\\
req_{back,all} \wedge req_{back,prod} & \text{if } \mathit{label}(n) = \text{`Back-end'} \wedge e = \mathit{prod},\\
req_{db,all} & \text{if } \mathit{label}(n) = \text{`Product DB'} \wedge e = \mathit{dev},\\
req_{db,all} & \text{if } \mathit{label}(n) = \text{`Product DB'} \wedge e = \mathit{prod},\\
true & \text{otherwise.}
\end{cases}$$

The upper part of the application environment topology shown in Fig. 2 represents the (unresolved) α-topology. Application-specific AERs are attached to the α-topology using the *map* function, which is defined in the original paper. In addition to these

[1] Wettinger et al.: *Enabling DevOps Collaboration and Continuous Delivery Using Diverse Application Environments,* Proceedings of the 23rd International Conference on Cooperative Information Systems (CoopIS), Springer, 2015.

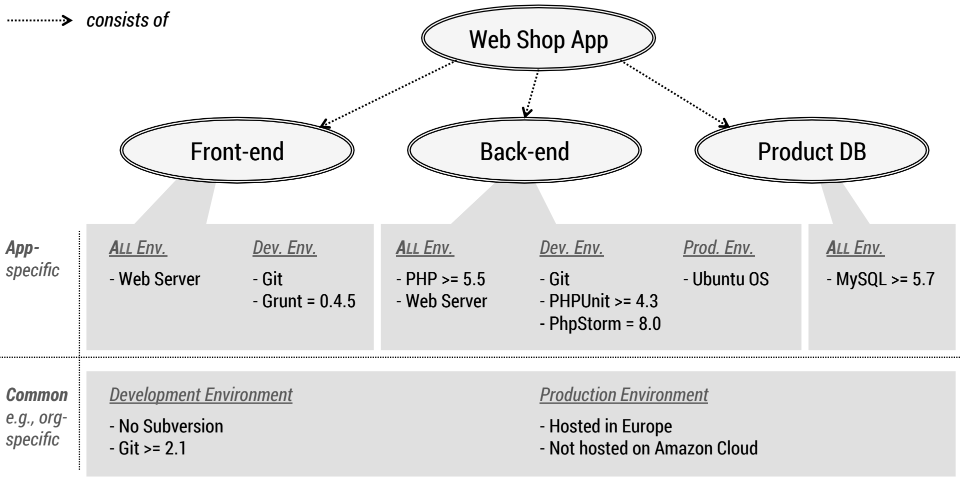

Fig. 1. Application environment requirements for the Web shop application

AERs, there are common, application-agnostic AERs that also need to be considered when deriving resolved topologies:

$$req_{common,dev} = P^{im,exclude}_{Subversion}(\text{‘Subversion’}) \wedge P^{im,eqGr}_{GitVer}(\text{‘Git’, ‘version’, ‘2.1’})$$
$$req_{common,prod} = P^{ev,exclude}_{Amazon}(\text{‘Amazon Cloud’}) \wedge P^{ev,equals}_{Europe}(\text{‘*’, ‘region’, ‘Europe’})$$

We proceeded by generating a γ-topology utilizing our knowledge base, the application-specific AERs, and the common AERs. From it, we derived two sets of viable topologies: the set $\mathcal{T}_{dev}$ contains topologies that are potentially viable for development environments, based on the corresponding AERs, i.e., the predicates $req_{*,all}$ and $req_{*,dev}$. Likewise, the set $\mathcal{T}_{prod}$ contains topologies viable for production environments. Next, we considered *environment preferences,* i.e., environment-specific constraints to identify suitable topologies based on the sets of potentially viable topologies. For this purpose, we defined a constraint for each kind of target environment. For a given topology T the following constraint c_{dev}, for example, specifies that all parts of the application (front-end, back-end, and database) must be hosted on a single machine, namely the developer machine:

$$\begin{aligned} c_{dev} = {} & \text{‘Developer Machine’} \in T \wedge \exists \text{path} \in T : \text{‘Front-end’} \xrightarrow{*} \text{‘Developer Machine’} \\ & \wedge \exists \text{path} \in T : \text{‘Back-end’} \xrightarrow{*} \text{‘Developer Machine’} \\ & \wedge \exists \text{path} \in T : \text{‘Product DB’} \xrightarrow{*} \text{‘Developer Machine’} \end{aligned}$$

This is to minimize resource usage for development environments. Similarly, the constraint c_{prod} can be defined to specify that each part of the application must

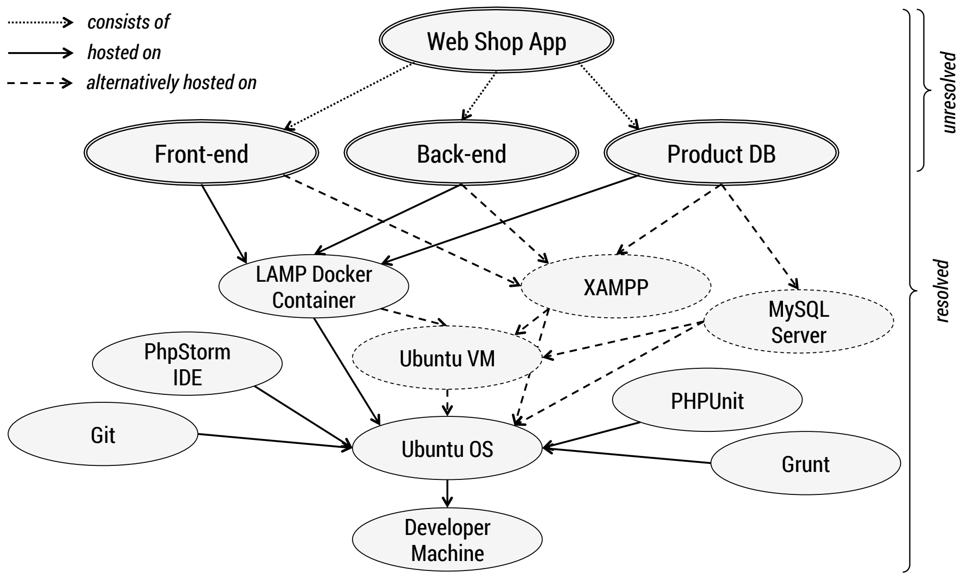

Fig. 2. Suitable topology alternatives for development environment (simplified)

be hosted on a separate, dedicated machine such as a virtual server at a Cloud provider. This is essential for production environments to avoid a single point of failure. Further constraints could be defined, e.g., to specify that all virtual servers must be located at a single Cloud provider to make application instances responsive by avoiding additional latency. By applying the filter function σ (defined in the original paper) in conjunction with such constraints, we generated a set of suitable topology alternatives. A simplified subset of development environment topologies resulting from $\cup_{T \in \mathcal{T}_{dev}} \sigma(T, c_{dev})$, is outlined in Fig. 2. Likewise, Fig. 3 shows a simplified subset of suitable production environment topologies resulting from $\cup_{T \in \mathcal{T}_{prod}} \sigma(T, c_{prod})$. We can then define utility functions to rank a filtered set of topologies to find the most suitable alternative. As an example, we used the following function for development environments in case we consider the topology with the least number of nodes (implying minimum resource usage) as the most suitable one:

$$utility_{dev}(T) = |nodes(T)|, \text{ where } T \in \bigcup_{T' \in \mathcal{T}_{dev}} \sigma(T', c_{dev})$$

That is, the function returns the number of nodes in a specific topology. By iterating over all topology alternatives and applying the function to each of them, they can be ranked. Consequently, the topology alternative with the least number returned by $utility_{dev}$ was selected as the most suitable topology alternative. In addition to the case study based on our formal concepts discussed in this section, we implemented a

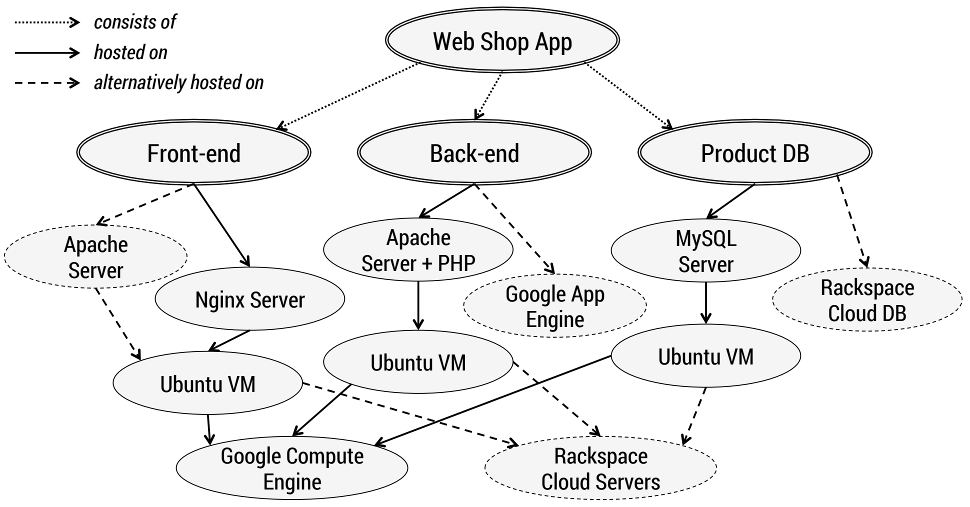

Fig. 3. Suitable topology alternatives for production environment (simplified)

prototype based on Neo4j, a popular graph database. A simplified and interactive demo is publicly available as Neo4j gist[2]; the sources are shared on GitHub[3].

AERs are not limited in their usage to the technical level of building suitable application environments. They can further be utilized as a common language to express infrastructure, middleware, and tooling-related requirements that are maintained and considered by both developers and operations personnel. Since many of today's applications are implemented following agile development processes and principles, requirements are usually changing frequently as the application evolves constantly and seamlessly. As an example, migrating the product database of the Web shop application from MySQL to MongoDB impacts development, test, and production environments. AERs are as such the means to maintain these requirements and to keep them consistent across environments.

[2] Neo4j gist: `http://gist.neo4j.org/?aaf3561ae0fc34063707`
[3] GitHub gist: `https://gist.github.com/jojow/42ea44ca273d631a9bff`