ARTICLE IN PRESS

Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Self-managing cloud-native applications: Design, implementation, and experience

Giovanni Toffetti *, Sandro Brunner, Martin Blöchlinger, Josef Spillner, Thomas Michael Bohnert

Zurich University of Applied Sciences, School of Engineering, Service Prototyping Lab (blog.zhaw.ch/icclab/), 8401 Winterthur, Switzerland

HIGHLIGHTS

- A definition of cloud-native applications and their desired characteristics.
- A distributed architecture for self-managing (micro) services.
- A report on our experiences and lessons learnt applying the proposed architecture to a legacy application brought to the cloud.

ARTICLE INFO

Article history:
Received 30 November 2015
Received in revised form
30 June 2016
Accepted 3 September 2016
Available online xxxx

Keywords:
Micro services
Cloud-native applications
Container-based applications
Distributed systems
Auto-scaling
Health-management

ABSTRACT

Running applications in the cloud efficiently requires much more than deploying software in virtual machines. Cloud applications have to be *continuously managed*: (1) to adjust their resources to the incoming load and (2) to face transient failures replicating and restarting components to provide resiliency on unreliable infrastructure. Continuous management *monitors* application and infrastructural metrics to provide automated and responsive reactions to failures (*health management*) and changing environmental conditions (*auto-scaling*) minimizing human intervention.

In the current practice, management functionalities are provided as infrastructural or third party services. In both cases they are external to the application deployment. We claim that this approach has intrinsic limits, namely that separating management functionalities from the application prevents them from naturally scaling with the application and requires additional management code and human intervention. Moreover, using infrastructure provider services for management functionalities results in vendor lock-in effectively preventing cloud applications to adapt and run on the most effective cloud for the job.

In this paper we discuss the main characteristics of cloud native applications, propose a novel architecture that enables scalable and resilient self-managing applications in the cloud, and relate on our experience in porting a legacy application to the cloud applying cloud-native principles.

1. Introduction

After a phase driven mainly by early adopters, cloud computing is now being embraced by most companies. Not only new applications are developed to be run in the cloud, but legacy workloads are increasingly being adapted and transformed to leverage the dominant cloud computing models. A suitable cloud application design was published previously by the authors [1] in the proceedings of the First International Workshop on Automated Incident Management in the Cloud (AIMC'15). With respect to that initial position paper, this article relates on our experience implementing the design we propose with a specific set of technologies and the evaluation of the non-functional behavior of the implementation with respect to scalability and resilience.

There are several advantages in embracing the cloud, but in essence they typically fall into two categories: either *operational* (flexibility/speed) or *economical* (costs) reasons. From the former perspective, cloud computing offers fast self-service provisioning and task automation through application programming interfaces (APIs) which allow to deploy and remove resources instantly, reduce wait time for provisioning development/test/production environments, enabling improved agility and time-to-market

* Corresponding author.
E-mail addresses: toff@zhaw.ch (G. Toffetti), brnr@zhaw.ch (S. Brunner), bloe@zhaw.ch (M. Blöchlinger), spio@zhaw.ch (J. Spillner), bohe@zhaw.ch (T.M. Bohnert).

http://dx.doi.org/10.1016/j.future.2016.09.002
0167-739X/

facing business changes. The bottom line is *increased productivity*. From the economical perspective, the *pay-per-use* model means that no upfront investment is needed for acquiring IT resources or for maintaining them, as companies pay only for allocated resources and subscribed services. Moreover, by handing off the responsibility of maintaining physical IT infrastructure, companies can avoid capital expenses (*capex*) in favor of usage-aligned operational expenses (*opex*) and can focus on development rather than operations support.

An extensive set of architectural patterns and best practices for cloud application development have been distilled, see for instance [2–4].

However, day-to-day cloud application development is still far from fully embracing these patterns. Most companies have just reached the point of adopting hardware virtualization (i.e., VMs). Innovation leaders have already moved on to successfully deploying newer, more productive patterns, like microservices, based on light-weight virtualization (i.e., containers).

On one hand, a pay-per-use model only brings cost savings with respect to a dedicated (statically sized) system solution if (1) an application has varying load over time and (2) the application provider is able to allocate the "right" amount of resources to it, avoiding both over-provisioning (paying for unneeded resources) and under-provisioning resulting in QoS degradation. On the other hand, years of cloud development experience have taught practitioners that commodity server hardware and network switches break often. Failure domains help isolate problems, but one should "plan for failure", striving to produce resilient applications on unreliable infrastructure, without compromising their elastic scalability.

In this article we relate on our experience in porting a legacy Web application to the cloud, adopting a novel design pattern for self-managing cloud native applications. This enables vendor independence and reduced costs with respect to relying on IaaS/PaaS and third party vendor services.

The main contributions of this article are: (1) a definition of cloud-native applications and their desired characteristics, (2) a distributed architecture for self-managing (micro) services, and (3) a report on our experiences and lessons learnt applying the proposed architecture to a legacy application brought to the cloud.

2. Cloud-native applications

Any application that runs on a cloud infrastructure is a "cloud application", but a "cloud-native application" (CNA from here on) is an application that has been *specifically designed* to run in a cloud environment.

2.1. CNA: definitions and requirements

We can derive the salient characteristics of CNA from the main aspects of the cloud computing paradigm. As defined in [5], there are five essential characteristics of cloud computing: *on-demand self service*, *broad network access*, *resource pooling*, *rapid elasticity* and *measured service*. In actual practice the *cloud infrastructure* is the *enabler* of these essential characteristics. Due to the economy of scale, infrastructure installations are large and typically built of *commodity hardware* so that failures are the norm rather than the exception [6]. Finally, cloud applications often rely on third-party services, as part of the application functionality, support (e.g., monitoring) or both. Third-party services might also fail or offer insufficient quality of service.

Given the considerations above, we can define the main requirements of CNA as:

- **Resilience**: CNA have to anticipate failures and fluctuation in quality of both cloud resources and third-party services needed to implement an application to remain available during outages. *Resource pooling* in the cloud implies that unexpected fluctuations of the infrastructure performance (e.g., noisy neighbor problem in multi-tenant systems) need to be expected and managed accordingly.
- **Elasticity**: CNA need to support adjusting their capacity by adding or removing resources to provide the required QoS in face of load variation avoiding over- and under-provisioning. In other terms, cloud-native applications should take full advantage of the cloud being a *measured service* offering *on-demand self-service* and *rapid elasticity*.

It should be clear that resilience is the first goal to be attained to achieve a functioning and available application in the cloud, while scalability deals with load variation and operational cost reduction. Resilience in the cloud is typically addressed using redundant resources. Formulating the trade-off between redundancy and operational cost reduction is a business decision.

The principles identified in the "12 factor app" methodology [7] focus not only on several aspects that impact on resiliency and scalability (e.g., dependencies, configuration in environment, backing services as attached resources, stateless processes, port-binding, concurrency via process model, disposability) of Web applications, but also the more general development and operations process (e.g., one codebase, build-release-run, dev/prod parity, administrative processes). Many of the best practices in current cloud development stem from these principles.

2.2. Current state of cloud development practice

Cloud computing is novel and economically more viable with respect to traditional enterprise-grade systems also because it relies on self-managed software automation (restarting components) rather than more expensive hardware redundancy to provide resilience and availability on top of commodity hardware [8]. However, many applications deployed in the cloud today are simply legacy applications that have been placed in VMs without changes of architecture or assumptions on the underlying infrastructure. Failing to adjust cost, performance and complexity expectations, and assuming the same reliability of resources and services in a traditional data center as in a public cloud can cost dearly, both in terms of technical failure and economical loss.

In order to achieve resilience and scalability, cloud applications have to be continuously *monitored*, analyzing their application-specific and infrastructural metrics to provide automated and responsive reactions to failures (*health management functionality*) and changing environmental conditions (*auto-scaling functionality*), minimizing human intervention.

The current state of the art in monitoring, health management, and scaling consists of one of the following options: (a) using services from the infrastructure provider (e.g., Amazon CloudWatch[1] and Auto Scaling[2] or Google Instance Group Manager[3]) with a default or a custom provided policy, (b) leveraging a third-party service (e.g., Rightscale,[4] New Relic[5]), (c) building an ad-hoc solution using available components (e.g., Netflix Scryer,[6] logstash[7]). Both

1 https://aws.amazon.com/cloudwatch.
2 https://aws.amazon.com/autoscaling.
3 https://cloud.google.com/compute/docs/autoscaler.
4 http://www.rightscale.com.
5 https://newrelic.com.
6 http://techblog.netflix.com/2013/11/scryer-netflixs-predictive-auto-scaling.html.
7 https://www.elastic.co/products/logstash.

Please cite this article in press as: G. Toffetti, et al., Self-managing cloud-native applications: Design, implementation, and experience, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.09.002