2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing

Application Deployment and Management in the Cloud

Marco Miglierina
DEIB, Politecnico di Milano
marco.miglierina@polimi.it

***Abstract*—Cloud computing is a revolutionary paradigm that allows to acquire infinite resources on demand and pay only for the actual use. Together with novel software development methodologies, such as *Agile* and *DevOps*, the cloud can really help companies in responding to market demand for continuous delivering innovative services, enabling the shift from a silo based release process, where customers, developers and system administrators are almost conflicting among each other, to a unified process, where the shared final objective is that of making the product available to the user.**

There are two main challenges that have to be faced when approaching the cloud. First of all, the heterogeneity of the offer. More and more cloud services are growing from different providers with no standard yet defined, driving companies towards the lock-in problem. Second, having the possibility to scale from 10s to 1000s of machines implies huge effort in IT management without smart automation systems.

This paper is an attempt to review some of the tools among the currently available ones that may help companies simplifying their application release process.

I. Introduction

In the last few years cloud computing increasingly earned the reputation of being the holy grail of application deployment. No doubt is a revolutionary paradigm that is changing the way companies, especially those which cannot afford big investments in hardware provisioning (SMEs), make business and deliver their product. However, whoever is approaching the cloud today will, first of all, find it difficult to choose among the heterogeneity of cloud providers offer and will face the well known problem of lock-in, which is certainly not what one would expect to deal with when looking for the elasticity and the flexibility that the cloud seems to promise. Moreover, having the possibility to scale from few tens to hundreds or even thousands of machines requires huge automation in managing deployment, maintenance, monitoring, scalability and whatever is implied by having to deal with complex systems. In plain terms, cloud computing by itself cannot certainly be the solution.

This paper is mainly an attempt to systematize the current state of the art on the matter based on the available literature and online content, without any expectation of being exhaustive since the author has little or no experience on actual usage of the tools and available material is mostly taken from the software official webpages or informal reviews.

We are first going to investigate the context in which companies are struggling to remain competitive (Section II). In Section III we will shortly describe what DevOps is and what is currently preventing fast software delivery. Then, we will go through a set of tools that, to the author's opinion, are worth being considered when dealing with application deployment and management on data centers or on the cloud (Section IV). Finally, we conclude by resuming different tools contributions in Section V.

II. From the Product to the Service Economy

According to [1], the 21st-century brought three main societal transformations. First of all, we are moving from a product economy to a service economy. Customers are not interested in the product itself anymore, rather they look at the services delivered together with the product. This is evident if we look at the mobile market, where the products that are succeeding are the ones that are pushing more in creating ecosystems of services together with their product (e.g., Apple iOS devices or Google Android devices). This transition is affecting software delivery as well, and cloud providers IaaS, PaaS and SaaS solutions are the direct response to this shift. Once, software companies were delivering their products and customers were responsible for the operations; today, as we will see, successful companies keyword is *continuous delivery*.

Second, the business environment is forcing companies to shift their focus from stability and efficiency to agility and innovation. If they do not keep up with today fast changing demand, by shortening work cycles and being creative, they can disappear in few years.

Third, the digital dimension is permeating every physical object and every human activity, making the IT companies playing a key role in the world economy.

Cloud computing is the direct response to the need for agility. IT companies can now focus on adding business value to their products, without thinking about hardware provisioning and management. However, Cloud by itself cannot solve the problem of speeding up the time to market.

Products must go from business requirements to the final delivery as fast as possible. Facebook, the second most visited website in the world, has a delivery process that allows two deployments per day. Flickr manages to deploy more than 10 times a day. At Etsy, they achieved the astonishing result of having more than 50 release per day. In this new context, what allows successful companies to remain competitive, managing to master complex system though releasing new services with such agility, is a new software development method named *DevOps* which is able to break the wall between developers and operations professionals.

III. DevOps

A big innovation in software development has been recently introduced by *Agile* methods [2]. Agile development allows to break a first wall between business stakeholders and developers by introducing close interaction and fast feedback among them. However, as highlighted in [3], when stepping back to the entire development to operations process from an enterprise

978-1-4799-8448-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SYNASC.2014.63

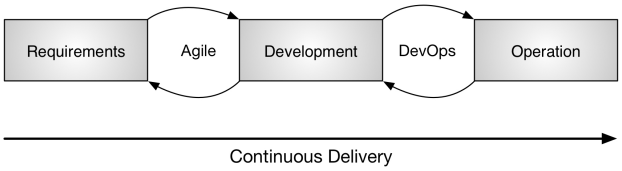

Fig. 1. Continuous delivery process

point of view, benefits brought by agile are often obscured. In [4], it is claimed that a cumbersome release process may turn agile development efforts back into a waterfall.

In simple words, *DevOps* is whatever is able to smooth the interaction between development and operations [3]. The first step is IT alignment, i.e., all parts involved must be aware that they are part of the same business process and they must have common objectives. Concretely, this means that the work lifecycle is extended to the final deployment by means of figures that are aware and work on the entire process and unifying tools.

Figure 1 shows where agile and devops give their contribution in the delivery process. Agile enables the alignment between business requirements and development, while DevOps can help bringing the developers and operations worlds closer. Even though DevOps experts stress a lot the fact that this new methodology is first of all a cultural change that must be pursued inside companies, it is certainly true that new tools and new methodologies that help automatize the entire release process are required. Many companies have been managing their infrastructure by means of custom *shell scripts* for years, and some of them still do it today. Shell scripts are, first of all, not *dev-compliant*. Developers usually do not know much about them, since is part of operations' competences, and therefore they break DevOps main requirement which is that of bridging the dev and ops worlds. Moreover, shell scripts have a series of downsides that prevent full automation:

- they are not idempotent, which means they cannot be run twice unless filling them with customized checks on what has already been done or commenting out part of it,
- if they are interrupted, for example for a network failure, all following dependent steps will fail,
- they are usually project dependent and not reusable,
- and as a consequence, they are hard to maintain.

Big enterprises, such as Google, probably solved this problem several years ago. Unfortunately, it is hard for companies to have Google-like IT departments and they must look for answers to all of these problems externally, among the available software. It turns out that, in recent years, there was a huge development in tools able to automate and simplify the release process especially in the open-source community. Moreover, the same big companies that were developing their supporting software in-house, started to use and even contribute to this kind of open-source projects (e.g., [5]–[7]). By means of these very tools, companies such as Twitter, Spotify, Facebook, Etsy and many other, managed to speed up their delivery process up to 50 times a day.

The main focus of this paper will be focused on ops tools that can help companies to do DevOps.

IV. Supporting tools

Among the existing tools, we tried to select the most famous or the most representative ones in their respective categories, giving short descriptions and identifying their role in the delivering process, with the objective of giving an overview and encourage the reader to deepen the matter of his interest.

A. Puppet

The first major open-source solution that addressed the issues raised in Section III was *Puppet* [8]. Puppet is a configuration and management tool that allows to express in a custom declarative language and using a model-based approach how the final system should be together with all interdependencies among components. Using a master client architecture it is then able to automatically configure resources and guarantee that the system remains consistent with the provided model.

Puppet is produced by PuppetLabs, funded by Luke Kanies in 2005. It was born as an open source project aiming to enable companies to exploit Google or Amazon-like automation. Today Puppet is the most famous DevOps tool and is used not only by SMEs, but also by big companies or foundations such as Wikimedia, Twitter, Paypal and even Google, which also financially supported the company with a considerable amount in 2011 [5].

The Puppet work-flow can be summarized by means of four steps:

- *Define* the desired state of the infrastructure configuration, using a declarative language, that is, no order has to be specified on how the configuration should take place, but only dependencies among components. The resulting definition could be represented as a graph of relations,
- *Simulate* the configuration to check the final state of the system after a change has been made, before actually perform it in the real system,
- *Enforce* the deployed desired state automatically,
- *Report* any difference between the initial state and the current state and any change that has been performed.

One of the key features of Puppet is reusability. On Puppet Forge, the online marketplace, the user can choose among a set of over 2000 pre-built modules. Alternatively, a new custom module can be built. Modules can than be reused on different machines with different operating systems. Moreover, modules can be combined into configuration stacks.

Puppet also offers the powerful feature of performing a dry-run of the changes the user want to do, before actually doing it. Launching the puppet command for updating the configuration in *noop* mode shows what would happen to the system, reporting back the changes.

The Puppet suggested configuration is a master client architecture, where configuration updates are pushed to the master and all clients will pull their specific configuration from. The user can also tell the master to push the clients configuration, once clients are properly configured. One can also decide to install it as a standalone solution, however this implies that configuration updates need to be distributed to all nodes by the user.

In 2011 Puppet 2.0 was released together with ability of provisioning in the cloud. Today, they managed to integrate several existing private and public cloud solutions, such as

VMWare, OpenStack, Amazon EC2 and Google Compute Engine.

Puppet offers both an open source and a commercial version. The commercial version offers a *Graphical User Interface*, that allows operations experts to visualize and monitor resources and to track errors faster than using the command line tool; *Supported Modules*, i.e., modules that have been tested by Puppetlabs engineers and are guaranteed to be reliable; user management and authentication; different levels of support services.

Several other configuration and management tools were born after Puppet, and have today become widespread, such as Chef [9] (e.g, used by Facebook), SaltStack [10] (e.g., used by Samsung), Ansible (e.g., used by Twitter) and CFEngine (e.g., used by Linkedin). A comparison among these tools is out of the scope of this paper, however, from available online literature [11], [12] there seem to be an eternal battle among which one is better, usually ending up with no winner since it seems to really depend on the different use cases. Each of them has its own peculiarity in the language and architectural style, however they tend to chase each other on the most-wanted features so that in the end they become hard to compare.

B. Vagrant

Another very famous and widespread tool able to speed up the release process and simplifying deployment on the cloud is *Vagrant* [13]. Vagrant is an open-source software for creating and configuring virtual development environments. One of the main issues when working in teams, across different development environments, and during quality assurance and finally in production, is the well known "works on my computer" problem. It is hard to guarantee that a software working on one PC it will successfully run on another one where there may be a different Linux Distribution, or a different version of some library and so on. Vagrant enable teams to develop and test their software on the exact same configuration that they will have on production, in a simple, reproducible and portable way.

By means of a configuration file named *Vagrantfile*, users specify one or more *boxes*, which are a sort of base images, and how these should be configured and by means of the command "vagrant up" each user will have its virtual development environment. The Vagrantfile, as well as any configuration script, must be kept under the same versioning system of the application under development. Boxes can be chosen from Vagrant Cloud[1], a publicly available catalog where everyone can share its own box. Boxes can be downloaded locally manually through the command "vagrant box add USERID/BOXID", or let Vagrant download it when you launch the machine for the first time.

Vagrant was initially born as a wrapper around VirtualBox, which enabled to manage a virtual machine from the command line. VirtualBox is still the default provider, but since version 1.1 it is not tied to it anymore and custom providers can be added. Besides VirtualBox, other official providers are: VMWare, Docker (see Section IV-F) and Hyper-V. Moreover, they provided as an example of custom provider implementation the Vagrant AWS plugin[2] which allows to use Vagrant to control and provision machines in EC2 and VPC.

Virtual environments are configured starting from the base box each time the user runs *vagrant up* and completely removed from his computer by running *vagrant destroy*, so that space is saved when he is not testing. Provisioning instructions must be versioned as well, so that all developers can contribute and always have the exact same environment. The provisioning mechanism can be shell scripts (which are highly discouraged except for simple provisioning according to what been said in Section III), files to be loaded, one of the previously cited configuration and management tools (Ansible, CFEngine, Chef, Puppet) or Docker.

Here we show a basic usage example to show how simple is to create a virtualized environment with Vagrant. Provided that Vagrant and VirtualBox are installed, by running the following three commands the user will have a virtual machine running and an ssh session open in it:

```
$ vagrant init hashicorp/precise32
$ vagrant up
$ vagrant ssh
```

hashicorp/precise is the id of an Ubuntu 12.04 LTS 32-bit image which is provided by Hashicorp, i.e., the company behind Vagrant. The current folder will be shared and synchronized with the virtual machine folder */vagrant* by default. The first command will create the following *Vagrantfile*:

```
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "hashicorp/precise32"
end
```

If we now change it with the following configuration:

```
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "hashicorp/precise32"
  config.vm.provision :shell, path: "bootstrap.sh"
  config.vm.network :forwarded_port, host: 4567, guest: 80
end
```

which tells Vagrant to provision the machine with a shell script named *bootstrap.sh* and to bind the guest port 80 to the host port 4567, and then we create and save as *bootstrap.sh* the following shell script:

```
#!/usr/bin/env bash
apt-get update
apt-get install -y apache2
rm -rf /var/www
ln -fs /vagrant /var/www
```

and finally we run

```
$ vagrant reload --provision
```

by pointing our browser to *http://127.0.0.1:4567* we will see the webpage showing the two files in the shared directory (Vagrantfile and bootstrap.sh) served by the virtual machine.

In order to use a different provider from VirtualBox, the user does not need to make any modification to the Vagrantfile (except for credentials where needed), and just run *vagrant up –provider=vmware_fusion* or *vagrant up –provider=aws*. In order to use custom providers such as *aws*, a plugin must be installed running, for example, *vagrant plugin install vagrant-aws*.

Last, Vagrant can also be integrated in a Maven[3] integration test work-flow by means of the related Maven plugin available

[1] Vagrant Cloud is available at https://vagrantcloud.com

[2] The Vagrant AWS plugin is available at https://github.com/mitchellh/vagrant-aws

[3] Maven is a build automation tool for Java projects. Official page at http://maven.apache.org.

online[4]. The plugin will first create the machine using Vagrant, run the tests, and finally destroying the machine.

C. Apache jclouds

Apache jclouds [14] is an open-source library for Java and Clojure that provides abstractions of several cloud providers APIs, allowing developers to create portable cloud provisioning software. It currently support 30 cloud providers [15]. The main abstractions, called *Views*, are:

- *ComputeService*, which simplifies the management of virtual machines,
- *BlobStore*, offering access to key-value stores,
- *LoadBalancerService*, currently in Beta version, allow to configure load balancers where they are offered by the cloud provider.

By using *Views* the developer is completely provider independent. However, he may still decide to use API-specific calls by *unwrapping* the view and exploit, for example the EC2 API which is currently supported by several providers, such as AWS, Eucalyptus and OpenStack. Finally, jclouds offers provider-specific calls if required.

Jclouds also provide handy instruments for unit testing, such as cloud simulation by means of stub connections.

The library does not perform deployment and management directly, however can be used as an abstraction layer for other deployment tools such as Apache Whirr (Section IV-D).

D. Apache Whirr

Apache Whirr is a set of java libraries for provisioning and deployment of distributed systems in a cloud neutral way exposing a common service api. The project started in 2007 as a set of bash scripts to run Hadoop on Amazon EC2. Afterwards, these scripts were ported to Python for extra features and enabling the usage of additional providers. In may 2010 the Apache Whirr Incubator was started. The Python version was discontinued and replaced by a java version, which uses Apache jclouds. In August 2011 Whirr became an Apache Top-Level Project.

The library offers smart defaults that allow easy deployment with default values for configuration, which can be easily overridden or constrained if necessary. Whirr main focus is providing an abstraction layer to define big data infrastructure based on Hadoop and instantiate those infrastructure on Clouds.

The first thing the user is supposed to do when using Whirr is to define the service roles that will compose the system. After that, the user should write a *ClusterActionHandler* for each role or reuse existing service handlers such as *puppet*, *zookeeper* or *hbase* that are provided together with the distribution. These handlers should define 2 different phases: the bootstrap phase, in which the user defines the scripts to be used for installing required dependencies on the nodes, and the configuration phase, in which the user specify firewall policies and configure the different instances based on their role name, orchestrating interdependency automatically and hiding configuration scripts. Last, the user should write the scripts (by default bash scripts) that are required to be run during the two phases. In order to run the deployment, the user can either do it programmatically in java or using the CLI together with configuration files.

[4]http://nicoulaj.github.io/vagrant-maven-plugin/.

Apache Whirr cannot be directly compared to other configuration management utilities such as Puppet since they do not overlap completely, but rather they can work together and complement each other. For example, Whirr can be used to deploy a Puppet infrastructure on the cloud and start it. Tools like Puppet are good at getting nodes to a target state and maintaining it, while it may be more challenging to use it to deploy distributed application requiring coordination during setup. Apache Whirr is instead a very handy library that allows to define coordination across instances in pre-defined stages. In [16], Whirr is, in fact, named a *Deployment Orchestrator Engine*.

As far as Vagrant is concerned, there is probably no direct integration among the the two. Vagrant allows to define new providers and provisioner by means of custom plugins, but the use of Ruby language is required. Therefore, the creation of a Whirr provider plugin for deploying on all clouds supported by Whirr as well as a Whirr provisioner plugin for configuring and manage them is currently challenging.

E. Brooklyn

Brooklyn is a framework for modeling, monitoring, and managing applications through autonomic blueprints [17]. Open-sourced in 2012 by CloudSoft Corporation, it recently entered the incubating process at Apache Software Foundation (May 2014).

Brooklyn addresses the same problem addressed by the tools described so far. However, its developers' claim is that all of the existing tools solve only part of the problem. For example, Puppet can, in a single command, provision a new cloud server then install and configure an application but it requires extra programming work to reconfigure the load balancer whenever the cluster is resized. Cloud providers may enforce autonomic procedures, such as Amazon autoscaling rules, but it is dependent on CloudWatch and policies that you can express are simple and the user may want to implement its own policies.

Brooklyn claims to differ from all the existing solutions, first of all, because it gathers provisioning, deployment, monitoring and complex autonomic management in a single tool. All of this can be integrated together with the code by means of blueprints expressed in YAML. The proposed analogy for this tool is that Brooklyn is to run-time what Maven is to build-time. Moreover, the tool can leverage all of the tools described so far (Puppet, Chef, jclouds, Whirr) and many other, especially for the deployment which is kept hidden to the user, and complement all of this with an application-oriented model which focuses on enforcing policies.

Another important advantage is that Brooklyn aims at being standards compliant. The only two standards available concerning the deployment and management on the cloud are TOSCA [18] and CAMP [19], which are currently under specification. Both specifications aim at enhancing the portability of Cloud applications across different Clouds. TOSCA is a specification for a language (YAML and XML) that should help users to describe how the system should be built, therefore a sort of standard for configuration and management tools languages, in a cloud agnostic way. CAMP is instead a specification on the artifacts and the API that a PaaS should support to manage the building, running, administration, monitoring and patching of applications in the cloud. Currently Brooklyn uses a YAML specification which complies with CAMP syntax

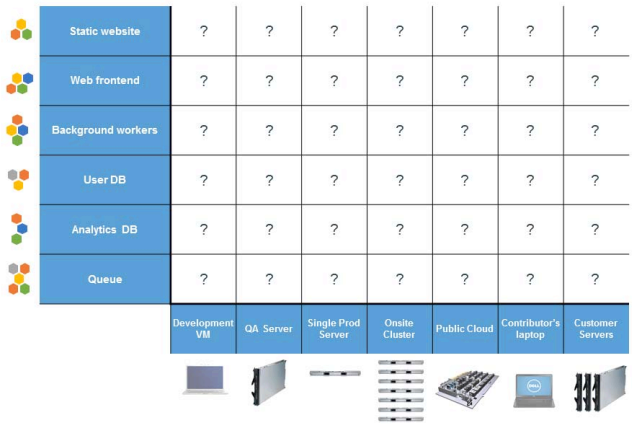

Fig. 2. The Matrix From Hell. Courtesy of Docker Inc.

and exposes many of the CAMP REST API endpoints, but they would like to expand the coverage. Moreover, they are working on supporting TOSCA specification.

F. Docker

So far we saw automation tools that can somehow help provisioning on existing and heterogeneous platforms. In March, 2013, the PaaS provider company dotCloud[5] released Docker, an open-source solution that tries to solve the problem they colloquially refer to as the *Matrix From Hell* (Figure 2), which well describes how many compatibility issues have to be faced today in order to have all software stacks run during the whole release process, in the complete opposite way we saw so far. Instead of automating and trying to make the deployment and management process as much transparent and multi-platform as possible, they proposed to solve the matrix from hell as cargo transport solved the same exact problem years ago, that is, by delivering goods in standards containers, so that, no matter what is inside, any means of transport always knows how to deal with it. After only 7 months from Docker release, the project was counting 5,000+ followers on Twitter, 6,700+ GitHub Stars, 800+ GitHub Forks, 175+ non-employee contributors, 650+ GitHub repositories with Dockerfiles, and 1600+ people attending meetups scheduled in New York, London, Paris, Nairobi, Hamburg, and San Francisco. In October, 2013, recognizing the momentum behind Docker, dotCloud officially changes its name to Docker, Inc., and announces a major shift to Docker-related mission and business model [20]. In June 2014, Docker 1.0 was released, even though several big companies such as Twitter and Ebay were already using it in production before the "ready-for-production" release and against Docker Inc. suggestion of not using it in production.

Behind Docker success is mainly the ability of offering the same advantages of delivering an entire VM with the application, where the developer can install all required dependencies, without the disadvantages of transferring GB of data for each release and without the overhead of a completely virtualized hosting machine. The core of all this magic comes from Linux containers interface (LXC[6]). LXC is an interface which allows Linux users to create and manages system or application containers. It is the same feature used for building Android or iPhone applications, where, even though each of them is running using the same kernel, they are completely sandboxed from each other. Each *Dockerized application* instance is therefore running on a sort of lightweight VM, with a complete copy of the file system.

Besides providing very lightweight images, Docker is very performant and efficient thanks to a smart mechanism for both storing caching information at runtime and saving different versions of images for subsequent builds, which is very similar to how revision control systems like git works. Starting from the base image, any subsequent change that requires to be saved is stored as a diff layer, containing only files or folder that were either changed, added or deleted. This mechanism makes any subsequent run of an already run build much faster than the first one, or pushing a new version of an image to a remote repository or to the production server very efficient.

Docker can only be installed on Linux hosts, however it can be easily installed on any Windows or Mac OS X provided that VirtualBox is installed. Docker installation will automatically setup a lightweight virtual machine and will automatically manage communication to the instance in a transparent way. Major cloud providers, such as AWS, provide Docker-ready images.

Configuration and management tools such as Puppet, are able to enhance Docker. Puppet, after installing the required module[7], can be used for both deploying a Docker platform and manage containers configuration. In [21], the author highlights an important shift that this new platform is bringing to the classic provisioning paradigm, which may apparently challenge standard configuration and management tools. Using containers as building blocks allows to push complete new images when launching new versions, which implies no configuration at runtime. The article clearly says that this is not actually a challenge but just a shift of this setup from runtime to build time. Ways of using Puppet for building Docker images against a Puppet master can be found online[8].

As we anticipated in Section IV-B, also Vagrant is integrated with Docker. By means of the docker provisioner, Vagrant can install Docker, pull Docker containers and configure certain containers to run on boot [22]. The docker provisioner for Vagrant is ideal for organizations that are using Docker as a means to distribute things like their application or services. Moreover, the docker vagrant provider allows for development environments to be backed by Docker containers rather than virtual machines.

Docker can also be used in combination with jclouds. jclouds-docker is a local cloud provider modelled on Docker that allow to use Docker as any other cloud provider.

As far as any integration of Docker with Apache Whirr, to the best of author's knowledge, no solution is currently available. However, given the runtime nature of Whirr, which is built for orchestrating the deployment, Apache Whirr is probably the most challenged tool, among the one seen so far, by the revolutionary paradigm brought by Docker.

Last, Brooklyn blueprints can be used to deploy applications to a Docker cloud by means of Clocker[9].

[5]https://www.dotcloud.com
[6]https://linuxcontainers.org

[7]https://forge.puppetlabs.com/garethr/docker
[8]https://github.com/lutter/puppet-docker
[9]https://github.com/brooklyncentral/clocker

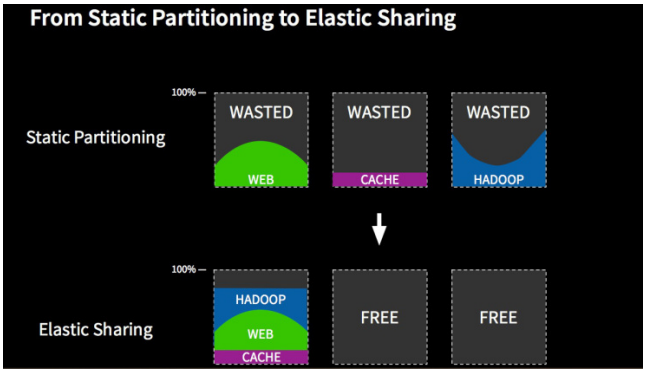

Fig. 3. Apache Mesos data center partitioning. Figure from [24]

G. Apache Mesos

So far, we saw different tools that can help deploying and managing applications either on standard servers or on virtual machines. Even though the process can be automated, deployment always targets machines independently, so that a company is forced to partition its data center ahead and assign each partition a service.

Around 2008, Ben Hindman, at the time a PhD Computer Scientist at UC Berkeley, was working on multi-core processors, looking for ways of spreading computing tasks across these chips as efficiently as possible. After he decided to work in a project with some of his friends that were working on software platforms that run across massive data centers, had the idea of trying to apply what he was studying on multi-core processors to entire data centers. "Sixty-four cores or 128 cores on a single chip looks a lot like 64 machines or 128 machines in a data center," he says. From this project Apache Mesos[10] came out [23], a server cluster management system able to make a data center look like a huge computer. The advantage is well depicted by Figure 3. Without "static" partitioning, datacenters resources are used much more efficiently. The idea behind Mesos, is actually not new. Google had been using a system called Borg since 10 years ago for managing their servers, and is now using a recently developed updated version, which looks much more similar to Mesos, called Omega [25], [26]. Both of these systems have been kept secret by the company, however, other companies such as Twitter and AirBnB can now manage their data centers in the same way thanks to Mesos.

Mesos technology is based on the same OS isolation mechanisms used by Docker (Section IV-F), that is, LXC. Thanks to isolation, Mesos is able to exploit a fine-grained sharing model at the level of tasks, and distribute tasks with a distributed scheduling mechanism managed by the framework.

The recently released version 0.20.0, introduced in Mesos native support for launching tasks that contain Docker images [27]. However, companies such as Twitter and eBay were already using Docker and Mesos together. [24] claims that, even though Mesos has been in production across a number of companies, the actual implementation remains highly complex and requires skilled engineers to implement a Mesos based solution. Mesosphere[11] seems to simplify Mesos and make it more usable. The Mesosphere company was actually founded by engineers who previously built the infrastructure at Twitter and AirBnB.

Configuration and management tools, such as Puppet, can still be used for managing Mesos nodes on a cluster[12]. Also Vagrant can be used to simplify the deployment of a Mesos infrastructure both on a VirtualBox for development or on the cloud[13].

As far as jclouds, Whirr and Brooklyn, no direct support for Mesos is currently available to the author's knowledge. However jclouds could be used to deploy Mesos nodes, and Brooklyn integrates with Docker by means of Clocker so it can be used to deploy applications on Mesos indirectly.

V. Conclusion

We overviewed several tools concerning deployment and management of applications on data centers or on the cloud. Considering Figure 1, we think all of these tools are able to speed up the process from left to right, i.e., from development to operation, no matter what is the scale of the faced use case. We saw most of these tools being used by big companies with huge infrastructures. However, SMEs and small teams can benefit as well since all these tools are available open-source and are becoming more and more usable, aiming at letting the user focus on its core business instead of bothering about how to make things work.

In Table I we tried to summarize the different contributions described in the paper.

To the author's opinion, all software developers and system administrator should at least be aware of the existence of all of the presented tools and have a general knowledge of what they do. This paper aimed at providing such an overview so to help choosing what tools are worth to be investigated for automating the deployment and management of own's use cases.

Acknowledgments

This work is partially funded by the EU commission in the FP7 programme through the MODAClouds project [30], contract number 318484.

References

[1] J. Sussna, "Cloud and devops: A marriage made in heaven," http://www.infoq.com/articles/cloud-and-devops, accessed: 2014-06-13.

[2] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," 2001. [Online]. Available: http://www.agilemanifesto.org/

[3] D. Edwards, "What is devops?" http://dev2ops.org/2010/02/what-is-devops/, accessed: 2014-06-15.

[4] A. Rendell, "Anti-pattern: The release vehicle," http://www.rendell.org/pebble/software/2010/02/09/1265756844985.html, accessed: 2014-06-15.

[5] C. Carling, "Google, vmware, and cisco throw money at puppet," http://www.wired.com/2011/11/puppet-series-c/, accessed: 2014-06-16.

[6] "Open source at twitter," https://engineering.twitter.com/opensource, accessed: 2014-09-12.

[7] "Netflix open source software center," http://netflix.github.io/#repo, accessed: 2014-09-12.

[8] PuppetLabs, "Puppet," http://puppetlabs.com.

[10] http://mesos.apache.org

[11] https://mesosphere.io

[12] Mesos Puppet Module is available at https://github.com/deric/puppet-mesos

[13] A ready-to-go example is available at https://github.com/everpeace/vagrant-mesos

TABLE I. SUMMARY TABLE

Name	Puppet	Vagrant	jclouds	Whirr	Brooklyn	Docker	Mesos
Role	Configuration and Management	Virtualization of Development Environment	Java Abstraction for Cloud API	Deployment Orchestrator Engine	Multi-cloud Deployment and Management Framework	Platform for Portable Distributed Applications	Cluster Manager
Contribution	Brings infrastructure as code, enable automation of deployment and configuration, runs are idempotent and modules are reusable	Allows to develop, perform QA and run in production on the same environment, simplifies deployment and provisioning of virtual machines	Brings abstraction from cloud-specific API	Brings cloud independent infrastructure as code, automates and orchestrate deployment	Brings cloud independent infrastructure as code, automates deployment and management through monitoring and adaptation policies as code, uses standards	Eliminates applications dependencies issues through isolation and containerization, dockerized application run on standardized environments, docker platform can be run either on bare metal, private or public cloud	Data centers are abstracted as if they were one only big computer, no per-service partitioning, more efficient use of resources
Supported providers	VMWare, OpenStack, Amazon EC2, Google Compute Engine, Eucalyptus, Azure (coming soon, recently announced)	VirtualBox, VMWare, Docker, Hyper-V as official providers. Any other provider through custom providers (e.g., AWS)	Amazon, Azure, GoGrid, Ninefold, OpenStack, Rackspace, vCloud and others	jclouds supported providers	jclouds supported providers	Mac OS X (virtualized), Ubuntu, Read Hat, Oracle Linux, CentOS, Debian, Gentoo, Google Cloud Platform, Rackspace Cloud, Amazon EC2, IBM Softlayer, Arch Linux, FrugalWare, Fedora, openSuSE, CRUX Linux, Microsoft Windows (virtualized)	Linux (64bit), Mac OS X (64bit)

[9] "Chef," http://www.getchef.com.

[10] SaltStack, "Salt," http://www.saltstack.com.

[11] P. Venezia, "Review: Puppet vs. chef vs. ansible vs. salt," http://www.infoworld.com/d/data-center/review-puppet-vs-chef-vs-ansible-vs-salt-231308, accessed: 2014-06-16.

[12] A. Sharp-Paul, "Puppet vs. chef - the battle wages on," http://www.scriptrock.com/blog/puppet-vs-chef-battle-wages, accessed: 2014-09-12.

[13] "Vagrant," http://www.vagrantup.com.

[14] "Apache jclouds," http://jclouds.apache.org.

[15] "Apache jclouds supported providers," http://jclouds.apache.org/reference/providers/.

[16] A. Bayer, "Cloudstack, jclouds and whirr!" http://www.slideshare.net/andrewbayer/cloudstack-jclouds-and-whirr, accessed: 2014-06-23.

[17] "Brooklyn incubator web site," http://brooklyn.incubator.apache.org, accessed: 2014-06-23.

[18] "Oasis topology and orchestration specification for cloud applications (tosca) tc," https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca, accessed: 2014-06-16.

[19] "Oasis cloud application management for platforms (camp) tc," https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=camp, accessed: 2014-06-16.

[20] B. Golub, "dotcloud, inc. is becoming docker, inc." http://blog.docker.com/2013/10/dotcloud-is-becoming-docker-inc/, accessed: 2014-09-14.

[21] D. Lutterkort, "Docker & puppet for application management and sanity," http://puppetlabs.com/blog/docker-and-puppet-for-application-management, accessed: 2014-09-15.

[22] "Docker provisioner," https://docs.vagrantup.com/v2/provisioning/docker.html, accessed: 2014-09-15.

[23] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[24] R. Miller, "Mesosphere grabs $10m in series a funding to transform server management," http://techcrunch.com/2014/06/09/mesosphere-grabs-10m-in-series-a-funding-to-transform-datacenters/, accessed: 2014-09-15.

[25] C. Metz, "Return of the borg: How twitter rebuilt googles secret weapon," http://www.wired.com/2013/03/google-borg-twitter-mesos/all/, accessed: 2014-09-15.

[26] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *SIGOPS European Conference on Computer Systems (EuroSys)*, Prague, Czech Republic, 2013, pp. 351–364. [Online]. Available: http://eurosys2013.tudos.org/wp-content/uploads/2013/paper/Schwarzkopf.pdf

[27] "Mesos 0.20.0 released," http://mesos.apache.org/blog/mesos-0-20-0-released/, accessed: 2014-09-15.

[28] M. Brittain, "Metrics-driven engineering," http://www.slideshare.net/mikebrittain/metricsdriven-engineering, accessed: 2014-07-11.

[29] P. Debois, "Patrick debois on the state of devops," http://www.infoq.com/interviews/debois-devops, accessed: 2014-07-11.

[30] "MODAClouds EU project," http://www.modaclouds.eu/.