An Agile Container-based Approach to TaaS

Pedro Verdugo , Joaquín Salvachúa , Gabriel Huecas

Abstract—**Current cloud deployment scenarios imply a need for fast testing of user oriented software in diverse, heterogeneous and often unknown hardware and network environments, making it difficult to ensure optimal or reproducible in-site testing. The current paper proposes the use of container based lightweight virtualization with a ready-to-run, just-in-time deployment strategy in order to minimize time and resources needed for streamlined multicomponent prototyping in PaaS systems. To that end, we will study a specific case of use consisting of providing end users with pre-tested custom prepackaged and preconfigured software, guaranteeing the viability of the aforementioned custom software, the syntactical integrity of the provided deployment system, the availability of needed dependencies as well as the sanity check of the already deployed and running software. From an architectural standpoint, by using standard, common use deployment packages as Chef or Puppet hosted in parallellizable workloads over ready-to-run Docker images, we can minimize the time required for full-deployment multicomponent systems testing and validation, as well as wrap the commonly provided features via a user-accessible RESTful API. The proposed infrastructure is currently available and freely accessible as part of the FIWARE EU initiative, and is open to third party collaboration and extension from a FOSS perspective.**

1. Introduction

In the last few years the idea of leveraging cloud technologies for performance and functional testing has been revisited in numerous occasions, and mostly all authors concur in the economic, organizational and security-oriented advantages of a cloud-based testing platform [1], [2], [3], [4].

In the present paper we will propose our implementation for one of such functional, goal-oriented cloud-based testing platform. To begin with, we will define some of the most prominent terms used along this study, as are the Testing as a Service and Lightweigth Virtualization paradigms. As remarked above, there is a current interest and constant advancements in the cloud testing field, so we will analyze the most relevant and prominent contributions to our field of study. Once a global view is attained, we will rationally detect and categorize the requirements for our own case of study, modeling a working solution. Then we will detail a functional implementation of the previously analyzed solution, including full details on the software stacks used and code links where relevant. Later, a validation of the presented work will be made available, showing the obtained data in comparison with similar implementations. Finally, an objective assessment of the goals attained will be presented, along with detected improvements and future works.

1.1. Testing as a Service

Traditional Cloud deployment models [5] can be briefly defined as follow:

- *Software as a Service* (SaaS): Provides software packages ready to be deployed in a cloud platform. - *Platform as a Service* (PaaS): Provides a prebuilt software environment, commonly an OS with libraries and optional dependencies. - *Infrastructure as a Service* (IaaS): Provides bare metal or virtualized hardware on demand.

But for our case of study, a recently adopted model [6] of special interest is that of **Testing as a Service** (TaaS), being defined as automated software testing offered via a cloud-based service.

There are several well known advantages derived from adopting a TaaS perspective for Cloud testing [1], the most important of which we can enumerate as follows:

- **Scalable testing environment**: We can elastically adapt the number and power of hardware resources to the testing requirements.

- **Cost reduction**: By reusing testing environment hardware and resources we can minimize the number of locked infrastructure usages for a fixed time.

- **Utility-based service models**: Customize the testing services to those required by the tenant utilities.

- **On-demand testing services**: Our testing services can be fully available all the time.

As illustrated on Harikrishna [3], the usual capabilities for a TaaS system are organized as follows:

- *TaaS process management*: Understood as the control of the testing workflow.

- *QoS requirement management*: Delimiting the required parameters for a given QoS target.

- *Test environment service*: Dedicated to enabling and offering virtualized environments for testing.

- *Testing solution service*: Packages of standardized test models and methods, known as solutions.

- *Testing simulation service*: Offers capabilities for simulation of external environments.

- *On-demand test service*: Provides a continuous execution environment for tests.

- *Tracking and monitor service*: Responsible for accounting and monitoring of test results and behaviours.

1.2. Lightweight Virtualization

Lightweight virtualization techniques have been studied for a number of years [7], but have not seen popularity until the recent coming of the container-based VM [8] system.

This approach proposes the use of the linux kernel defined namespaces and control groups as VM-like objects, and presents several advantages related to full-stack virtualization [9], [10]:

- *Performance*: Comparatively similar to running native processes, classical VMs need hardware virtualization support to achieve near results.

- *Resource Utilization*: The sharing of the same kernel and modules, as well as optionally memory spaces, allows for a smaller container footprint as compared to a full VM.

- *Functionality*: The previously mentioned points allow for containers to almost completely substitute their common OS app counterparts, simplifying version updates and management.

There have been recent concerns with the security of the container-hosted processes, but at the time of writing the advances on control group security modules (GRSEC) and in-memory namespace management seems to have rendered said concerns obsolete [11].

2. Related Work

The field of cloud automated testing is nowadays mainly focused on the validation of the external main cloud platform components, as stated with initiatives like Open Cloud Lab [4]. Also to note, structure proposals like Progress Cloud Test Framework tend to integrate the corresponding testing suites in the framework itself [12]. This idea has lately been partially set aside to allow for more heterogeneous test loads, as shown in the FCHTS proposal [13].

From the virtualization standpoint, the main concern for VM deployment seems to have been with scheduling strategies [14], but not with the chosen VM technology itself [15]. An exception can be made for the Kuo proposal [16], which bases its deployment on the OpenStack framework. Touching very near our field of interest, the CUTEi testbed environment [17] proposes a container-based testbed solution, alas oriented to network simulation issues.

There have also been proposed solutions purporting intelligent test case generation and adaptation to variable applications, as is the case with [18], but without an extended implementation overview it is hard to measure the performance and success rate of said proposals.

Another promising and currently popular line of work seems to be the application of these techniques to mobile device app testing, specifically the Android platform [19] [20] [21] [16].

3. Case of Study: Definition

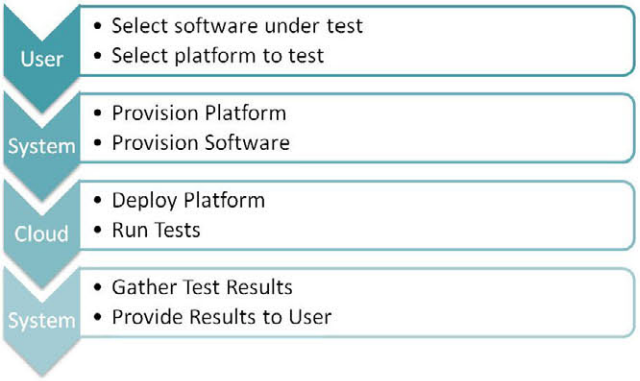

Figure 1. Testing Process Steps

To define the applicable uses of our system, we will begin by detecting and defining usage requirements. In the FIWARE environment, users are given a choice of base infrastructures for their cloud deployments, and are also allowed to customize the aforementioned deployment by installing prepackaged software. This prepackaged software is commonly subject to short cycles of development in a agile development fashion.

The need arises then to constantly test new available packages as well as new versions and bugfixes of preexisting packages for all the provided infrastructures.

The system currently under study proposes the automatization of said tests in a fast, resource oriented, on-demand, always available environment. To reach these objectives, we will define a main workflow, comprising the previously shown TaaS capabilities as related to our given scenario.

3.1. Deployment Artifact Validation

We will define a *deployment artifact* as the set of instructions and states necessary for the installation and configuration of a given software package. The main goal for our system will therefore be to successfully instantiate a deployment artifact in a given OS environment.

There are several variables to consider that will be of interest for the monitoring of our deployment, that we will detail as follows:

- **Operating System**: Deployment artifacts will have to be guaranteed to work at least in current Ubuntu and RedHat/CentOs releases.

- **Provisioning Software**: Chef, Puppet and optionally Murano deployment artifacts will have to be supported.

- **Dependency Management**: All necessary package dependencies will have to be provisioned and installed.

- **Syntax Check**: All provided deployment artifacts will have to comply with the chosen provisioning software syntax.

- **Deployment Check**: All provided deployment artifacts will have to complete the deployment process without errors.

- **Sanity Check**: Optionally, a simple check of the correct installation and configuration of the software packages will be executed.

Following the issues presented above, we can tentatively define the main workflow for our testing process, as illustrated in Figure 1.

4. Case of Study: System Implementation

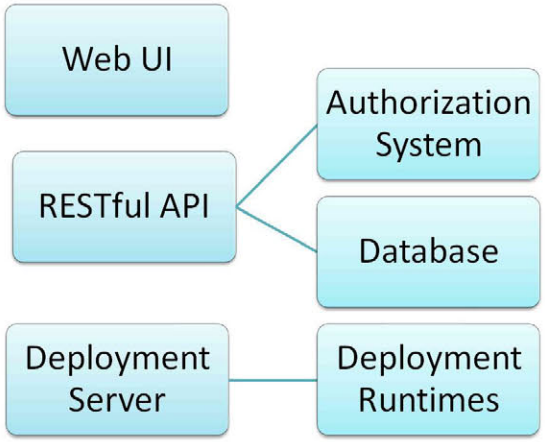

Figure 2. System Components

In order to cover the previously defined use cases, a reference system architecture has been implemented. Figure 2 gives a simplified overview of the current system structure, which we will study in detail in the following paragraphs.

With the idea of universal application and *libre* usage, our software stack will be exclusively composed of **Free Open Source Software** (FOSS), and is fully released under the *Apache License Version 2.0*.

4.1. Web UI

The User interface is written on javascript via the Backbone.js 1.1.0 framework and can work standalone as a completely browser based solution. For javascript dependency management, the usage of Require.js 2.2.0 enables fully AMD-compliant asynchronous module definition and on-demand load.

This UI is based on the single-page app pattern, and is oriented to usage by technical personnel, so the interface is extremely simplified and designed for a fast, workflow-oriented user experience. Thanks to the AMD structure and framework of choice, local and network resource usage are extremely low.

Figure 3 shows the current state of the *Validator UI*.

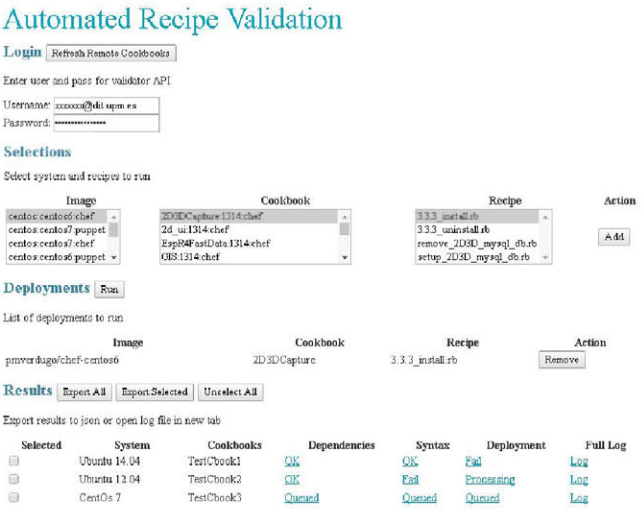

Figure 3. WebUI

4.2. RESTful API

The main component of our system is a RESTful API based on RESTfulFramework 3.3.3 and hosted in a Django server version 1.8.3. All code for the API is written for the Python 2.7.11 interpreter, and is compliant with the OpenStack style guides and the pep8 standard.

The API is the only external system interface, and is testable and fully documented in the *APIary* service.

4.3. Database

The database objects usages and management are wrapped in Django models, which will have an important role in the application workflow:

- Storage of available system images.
- Storage of remote software packages repositories and versions.
- Storage of usage history and monitoring.

4.4. Authentication API

The user authentication and authorization services will be externally provided by the FIWARE Identity Manager, an extension from the OpenStack Keystone authorization framework. A custom *Django Authorization Plugin* has been adapted to warrant compliance with the authorization schema and is available at the application *github* repository.

4.5. Deployment Service

The image deployment service is based on a Docker server level 1.23 API, and is responsible for the generation of images, instantiation of said images in flexible containers and execution, configuration and monitoring of said containers. A simple on-demand scheduler has been implemented as a rational assessment of the observed low number of concurrent users and jobs.

4.6. Hardware Environment

TABLE 1. HARDWARE ENVIRONMENT

Role	*CPUs*	*Ram*	*Network*	*OS*
API Server	2	2048	1000	Ubuntu 14.04
Deployment Server	4	4096	1000	Ubuntu 14.04
Web Server	2	2048	100	Windows 7 SP1

As seen in Table 1, the hardware requirements for our implementation are quite low. The separation of functionalities in different machines can be avoided when full integration is required, as is the case with the validation measurement infrastructure illustrated in the following section.

5. Case of Study: Metrics and Validation

TABLE 2. IMAGE CATALOG

OS	*Version*	*Provisioner*	*Generation (s)*	*Deployment (s)*
Ubuntu	12.04	Chef	1582	2
Ubuntu	14.04	Chef	1800	4
CentOs	6	Chef	1201	1
CentOs	7	Chef	1404	2
Ubuntu	12.04	Puppet	1209	2
Ubuntu	14.04	Puppet	1383	3
CentOs	6	Puppet	796	1
CentOs	7	Puppet	995	1

In this section we will define several metrics to enable a comparison with alternative deployment techniques.

To begin with, Table 2 shows the OS and provisioning engine currently supported by our system, along with the mean times for 3 runs of image generation and deployment to achieve results with a standard deviation under 10The image generation process is based on the *github* provided dockerfiles, and includes software updating and installation of the chosen provisioning environment. As we can observe, Ubuntu 14.04 is the bulkiest OS to install, centos6 being the lightest. As for provisioning systems, the download of ChefDK during the generation process is quite costly, mainly compared to the usage of system-provided Puppet-client packages. A special mention can be made about the very fast image deployment times, negligible in all cases.

The main concern with our system performance will be determined by the obtained speedup relative to preexisting virtualization systems. To the end of measuring this relative performance, we will test our system by running deployment tests for all the 47 available software packages in FIWARE-Lab.

We will define the observed times as follows:

- t_d: Elapsed time until software dependencies are installed.

- t_s: Elapsed time until deployment system syntax is checked.

- t_D: Elapsed time until software is deployed.

- t_T: Total system deployment time.

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \quad (1)$$

All given times are obtained by calculating the mean of 5 consecutive deployments, obtaining a standard deviation of the results under 10% ($\overline{x}$ with N=5 in the well-known formula for standard deviation given in equation 1) and supposing all times following a normal distribution, with the API and virtualization engine running in the same physical host machine and with hardware characteristics similar to those of the previously mentioned *Deployment Server*.

The performance tests will run over a pre-deployed system VM, as the image generation times are previously detailed and expected to be constant, and will include several deployment strategies:

- Local OpenStack Nova-based KVM machine.
- Local Vagrant-provisioned VirtualBox machine.
- Local Docker-based container.

As illustrated in Table 3, we can notice a considerable speedup by moving to lightweight virtualization environments. Dependency install times, have been observed to be the most variable factor in the total deployment time, mainly due to the random delay introduced by network usage, but are vastly exceeded by the i/o writes incurred during the main deployment phase. The multilayered, network-oriented infrastructure of OpenStack seem to be the reason for the slow response times obtained, which is totally avoidable in a single machine environment, but nonetheless serves as a representative picture of a real world scenario.

A graphic comparison of the mean times for all deployments is given in Figure 4, where we can finally extract visual conclusions from our experiment, namely that a mean of a 34.87% speed gain can be achieved by using container based virtualization as opposed to OpenStack-based VMs in test deployment environments. For local VirtualBox deployments, this number is reduced to a mean of 26.22%, also significant in the authors' opinion.

6. Conclusions and Future Work

Foremost, the authors would like to note that all sources for the presented software infrastructure are hosted via an open access *github* repository at http://www.github.org/ging/fiware-validator, available for free usage, peer review and collaboration.

6.1. Summary

In the opinion of the authors, the system as presented covers sufficiently the initial fixed requirements. A brief validation of the main system characteristics can be summarized in the following list:

- *TaaS process management*: The testing workflow process is unified in a single component.

- *QoS requirement management*: The system covers the QoS requirements for the currently given workloads.

TABLE 3. RUNTIMES FOR AVAILABLE FIWARE-LAB DEPLOYMENTS

Software under Test	Virtualization Engine (values in seconds) KVM				VirtualBox				Docker			
t_d	t_s	t_D	t_T	t_d	t_s	t_D	t_T	t_d	t_s	t_D	t_T	
2D3DCapture	753	320	4772	5845	467	221	4327	5015	330	166	2663	3159
2d ui	45	24	349	418	49	23	375	447	33	16	259	308
EspR4FastData	79	51	867	997	123	49	966	1138	76	31	651	758
GIS	425	163	3359	3947	407	202	3420	4029	258	144	2598	3000
IoTBroker	148	92	1608	1848	172	91	1410	1673	114	62	1035	1211
MRCoAP	41	25	366	432	40	19	331	390	37	16	283	336
RealVirtualInteractionGE	322	152	2873	3347	345	165	2476	2986	252	127	2267	2646
Stream oriented kurento	372	199	3691	4262	374	232	3381	3987	337	171	3003	3511
augmentedreality	50	29	449	528	63	35	518	616	37	23	424	484
beatest	386	206	3253	3845	399	213	3504	4116	294	165	2574	3033
cdvideoanalysis	83	47	668	798	71	31	534	636	43	30	420	493
cepheus	571	363	5384	6318	538	297	4610	5445	380	160	3125	3665
ckan	278	125	2530	2933	247	137	2289	2673	198	107	1725	2030
cloud portal	607	231	4494	5332	440	233	4017	4690	448	207	3517	4172
cloud rendering	191	103	1754	2048	196	117	1701	2014	135	69	1200	1404
content-based-security	456	220	3920	4596	456	236	4148	4840	295	189	3100	3584
cosmos	383	186	3305	3874	435	221	3216	3872	249	136	2584	2969
flod-enabler	157	72	1289	1518	151	97	1259	1507	161	76	1301	1538
interface designer	50	22	337	409	37	16	273	326	33	14	293	340
iotDiscovery	101	54	891	1046	80	33	672	785	55	27	568	650
keyrock	315	153	2713	3181	279	133	2284	2696	213	110	1892	2215
kiara	64	31	544	639	58	26	482	566	38	17	346	401
kurento	418	199	3530	4147	367	225	3713	4305	309	143	2619	3071
lightweightsematiccomposition	182	89	1517	1788	185	78	1281	1544	128	55	1068	1251
mahout	34	22	281	337	33	22	336	391	32	12	256	300
marketplace-ri	415	233	3903	4551	682	274	5208	6164	557	183	3492	4232
me-querybroker	55	28	473	556	62	22	424	508	31	19	314	364
metadatapreproc	79	42	716	837	56	30	564	650	57	33	563	653
orion	169	86	1460	1715	182	113	1746	2041	152	54	1234	1440
orion-dbcluster	272	151	2304	2727	156	73	1404	1633	151	82	1486	1719
poi dp	90	51	836	977	99	40	736	875	81	34	712	827
prrs	515	309	4370	5194	420	228	4539	5187	413	248	3627	4288
registry-ri	175	75	1338	1588	126	56	1196	1378	91	51	982	1124
repository-ri	433	235	3883	4551	406	192	3228	3826	392	157	2657	3206
rpcdds	33	18	304	355	33	15	277	325	30	19	282	331
semanticas	592	287	4465	5344	508	261	4328	5097	353	157	3190	3700
sls	622	271	4215	5108	461	235	4170	4866	398	185	3181	3764
sls-securityprobe	509	226	4037	4772	315	162	3204	3681	354	173	3016	3543
sopeco	164	66	1224	1454	172	87	1399	1658	127	74	1253	1454
spagobi	363	200	3495	4058	405	176	3421	4002	373	187	3160	3720
synch-fives	603	260	4843	5706	574	246	4563	5383	415	234	3728	4377
synchronization	74	46	699	819	64	31	523	618	62	28	473	563
virtualcharacters	56	28	445	529	48	26	425	499	33	15	269	317
webtundra	81	35	670	786	54	27	440	521	42	23	358	423
wilma	165	86	1540	1791	115	56	1019	1190	129	55	1053	1237
wirecloud	224	119	2271	2614	185	98	1881	2164	213	92	1570	1875
wstore	315	204	3210	3729	392	204	3217	3813	365	143	2665	3173
xml3d	366	240	3379	3985	309	168	2715	3192	180	117	1880	2177

- *Test environment service*: The lightweight virtualized environments implemented as Docker images are available on-the-fly for immediate deployment.

- *Testing solution service*: Given the context of deployment artifact testing, the system implements the most common provisioning solutions usages.

- *Testing simulation service*: The simulation of testing environments is limited by the availability of Docker containers, but sufficiently covers the presented workloads.

- *On-demand test service*: The system depends on a minimal number of external services and is constantly available for usage.

- *Tracking and monitor service*: The unification of data collection in a single database simplifies the tracking and monitoring of system usages.

6.2. Future Work

To fully cover the previously listed TaaS capabilities, the system as presented can be significantly improved in the following fields:

- Increase the available variables for external environments simulation.

- Increase the number of standard testing solutions supported.

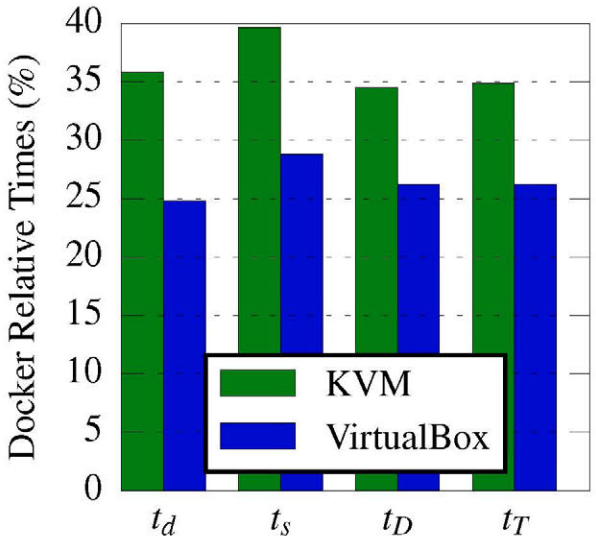

Figure 4. Runtimes Comparison

7. Acknowledgements

The current work is partially founded by the FIWARE (https://www.fiware.org/) European Union ICT FP7 package (http://cordis.europa.eu/fp7/ict/)

References

[1] W. Jun and F. Meng, "Software Testing Based on Cloud Computing," in *2011 International Conference on Internet Computing Information Services (ICICIS)*, Sep. 2011, pp. 176–178.

[2] P. Zhenlong, O. Y. Zhonghui, and H. Youlan, "The Application and Development of Software Testing in Cloud Computing Environment," in *2012 International Conference on Computer Science Service System (CSSS)*, Aug. 2012, pp. 450–454.

[3] P. Harikrishna and A. Amuthan, "A survey of testing as a service in cloud computing," in *2016 International Conference on Computer Communication and Informatics (ICCCI)*, Jan. 2016, pp. 1–5.

[4] C. J. Li and H. J. Shih, "A cloud testing platform and its methods based on essential cloud characteristics," in *2015 International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 1, Jul. 2015, pp. 163–169.

[5] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, "NIST cloud computing reference architecture," *NIST special publication*, vol. 500, no. 2011, p. 292, 2011. [Online]. Available: http://www.disa-apps.com/Services/DoD-Cloud-Broker/~/media/Files/DISA/Services/Cloud-Broker/nist-cloud-ref-architecture.pdf

[6] J. Gao, X. Bai, W. T. Tsai, and T. Uehara, "Testing as a Service (TaaS) on Clouds," in *2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, Mar. 2013, pp. 212–223.

[7] S. J. Vaughan-nichols, "New Approach to Virtualization Is a Lightweight," *Computer*, vol. 39, no. 11, pp. 12–14, Nov. 2006.

[8] C. Pahl, "Containerization and the PaaS Cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, May 2015.

[9] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," in *2015 IEEE International Conference on Cloud Engineering (IC2E)*, Mar. 2015, pp. 386–393.

[10] A. Sonone, A. Soni, S. Nathan, and U. Bellur, "On Exploiting Page Sharing in a Virtualised Environment - An Empirical Study of Virtualization Versus Lightweight Containers," in *2015 IEEE 8th International Conference on Cloud Computing*, Jun. 2015, pp. 49–56.

[11] J. C. Wang, W. F. Cheng, H. C. Chen, and H. L. Chien, "Benefit of construct information security environment based on lightweight virtualization technology," in *2015 International Carnahan Conference on Security Technology (ICCST)*, Sep. 2015, pp. 1–4.

[12] G. N. Iyer, J. Pasimuthu, and R. Loganathan, "PCTF: An Integrated, Extensible Cloud Test Framework for Testing Cloud Platforms and Applications," in *2013 13th International Conference on Quality Software*, Jul. 2013, pp. 135–138.

[13] S. J. Hsieh, S. M. Yuan, G. H. Luo, and H. W. Chen, "A flexible public cloud based testing service for heterogeneous testing targets," in *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*, Sep. 2014, pp. 1–3.

[14] Y. H. Tung, C. C. Lin, and H. L. Shan, "Test as a Service: A Framework for Web Security TaaS Service in Cloud Environment," in *2014 IEEE 8th International Symposium on Service Oriented System Engineering (SOSE)*, Apr. 2014, pp. 212–217.

[15] Y. Zheng, L. Cai, S. Huang, and Z. Wang, "VM scheduling strategies based on artificial intelligence in Cloud Testing," in *2014 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Jun. 2014, pp. 1–7.

[16] J. Y. Kuo, C. H. Liu, and W. T. Yu, "The Study of Cloud-Based Testing Platform for Android," in *2015 IEEE International Conference on Mobile Services*, Jun. 2015, pp. 197–201.

[17] H. Asaeda, R. Li, and N. Choi, "Container-based unified testbed for information-centric networking," *IEEE Network*, vol. 28, no. 6, pp. 60–66, Nov. 2014.

[18] R. Nasiri and S. Hosseini, "A case study for a novel framework for cloud testing," in *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)*, Sep. 2014, pp. 1–5.

[19] A. Malini, N. Venkatesh, K. Sundarakantham, and S. Mercyshalinie, "Mobile application testing on smart devices using MTAAS framework in cloud," in *2014 International Conference on Computer and Communications Technologies (ICCCT)*, Dec. 2014, pp. 1–5.

[20] L. Murugesan and P. Balasubramanian, "Cloud based mobile application testing," in *2014 IEEE/ACIS 13th International Conference on Computer and Information Science (ICIS)*, Jun. 2014, pp. 287–289.

[21] C. M. Prathibhan, A. Malini, N. Venkatesh, and K. Sundarakantham, "An automated testing framework for testing Android mobile applications in the cloud," in *2014 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, May 2014, pp. 1216–1219.