ISABEL: Infrastructure-Agnostic Benchmark Framework for Cloud-Native Platforms

Paulo Souza[a], Felipe Rubin[b], João Nascimento[c], Conrado Boeira[d], Ângelo Vieira[e], Rômulo Reis[f] and Tiago Ferreto[g]

School of Technology, Pontifical Catholic University of Rio Grande do Sul, Ipiranga Avenue, 6681 - Building 32, Porto Alegre, Brazil

Keywords: Cloud Computing, Cloud-Native Platforms, Cloud Services Evaluation, Benchmark Suite.

Abstract: The popularity of Cloud Computing has contributed to the growth of new business models, which have changed how companies develop and distribute their software. The reoccurring use of cloud resources called for a more modern and Cloud-Native approach in contrast to traditional monolithic architecture. While this approach has introduced better portability and use of resources, the abstraction of the infrastructure beneath it incited doubts about its reliability. Through the use of benchmarks, cloud providers began to provide quality assurances for their users in the form of service-level agreements (SLA). Although benchmarking allowed such assurances to be defined, the variety of offered services, each one of them requiring different tools, along with a nonexistent standard of input and output formats, has turned this in an arduous task. In order to promote better interoperability of benchmarking, we propose ISABEL, a benchmark suite for Cloud-Native platforms that standardizes the process of benchmarking using any existing benchmark tool.

1 INTRODUCTION

The widespread adoption of Cloud Computing by the industry has brought about new business models, which provides software development companies platforms to build, compile, and deploy their products while abstracting the infrastructure beneath it. Through these platforms, developers may request the provisioning of services instances from vast catalogs, according to their application's needs.

Due to the growth of cloud service providers, companies turn their decision based on their momentary requirements, such as demand for resources, security, cost management, and others (Garg et al., 2011).

While many cloud providers offer similar services and prices, recently, companies have shifted their decisions towards the quality of service assurances (Jin et al., 2013). Due to abstracting the infrastructure and also the configuration of service instances provided by these platforms, unexpected problems may occur in production environments, which may have unforeseeable outcomes for companies and their consumers.

The quality of service delivered by platforms is defined in contracts, such as service-level agreement (SLA), and must be agreed between providers and their consumers (Dillon et al., 2010). As the binding obligations described in these contracts must be fulfilled, benchmarking tools are used to ensure that the quality of service agreed upon is assured, since they can let the platform provider be aware of its boundaries.

Cloud providers can decide which assurances they can provide to their consumers by using the assistance of benchmark tools. Other than providers, developers may also benefit from benchmarking their applications and the services they rely on to better understand how the provided platform affects or benefits them and thus take the best course of action. (Binnig et al., 2009). While a wide variety of benchmarks exist for each service a platform may wish to provide, the use of different tools and their different config-

[a] https://orcid.org/0000-0003-4945-3329
[b] https://orcid.org/0000-0003-1612-078X
[c] https://orcid.org/0000-0002-2261-2791
[d] https://orcid.org/0000-0002-6519-9001
[e] https://orcid.org/0000-0002-1806-7241
[f] https://orcid.org/0000-0001-9949-3797
[g] https://orcid.org/0000-0001-8485-529X

Souza, P., Rubin, F., Nascimento, J., Boeira, C., Vieira, Â., Reis, R. and Ferreto, T.
ISABEL: Infrastructure-Agnostic Benchmark Framework for Cloud-Native Platforms.
DOI: 10.5220/0009581004820489
In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, pages 482-489
ISBN: 978-989-758-424-4

urations for both input and output formats for each benchmark tends to turn this whole operation into an onerous task for cloud providers.

This study proposes an approach to solve the current issue of interoperability between services and their benchmarking tools available in the market. Based on the current scenario, we present ISABEL: A benchmark suite for Cloud-Native platforms. ISABEL enables platforms to provide Benchmarking-as-a-Service (BaaS), which can be used by platform operators and developers alike.

Our contributions in this study are summarized as follows:

- **Unified Marketplace Solution:** of benchmark tools enabling operators to evaluate their services consistently.
- **Benchmark Accuracy Verification:** through a comparison of different tools by operators and benchmark developers alike.
- **Standardized Output Format:** based on a per-benchmark set of post-processing rules, which lessens the burden of processing raw output data for monitoring solutions.

The remainder of this paper is organized as follows: Section 2 provides the main concepts of cloud computing, Cloud-Native applications and platforms, and service reliability. Section 3 presents an overview of benchmarking cloud services, followed by ISABEL suite specification in section 4. Experimental results and then conclusion are provided in sections 5 and 6.

2 BACKGROUND

In this section, we provide background information on key-aspects discussed in the following sections of this paper.

2.1 Cloud Computing

The National Institute of Standards and Technology (NIST) (Mell et al., 2011) defines Cloud Computing as a model providing on-demand access to a configurable pool of shared resources (storage, networking, applications, and services) with minimum effort required for both provisioning and releasing.

Cloud Computing has changed how the software industry develops and distributes its products. It also allows companies to focus on developing their products, with no need for expertise outside their scope, leaving such to the cloud providers that specialize in those.

Other than reducing the need for expertise, there are compelling features that lead a business to adopt this technology. Elasticity enables resources to be scaled on a need-basis, thus reducing costs and eliminating the need for up-front payments, which are replaced by the model of pay-as-you-go. Resilience is also one of the main reasons for business' to adopt the cloud. As failures will eventually occur, having a fast response for such events is a must. By utilizing a service that assures a certain degree of resilience, organizations can reduce significant costs associated with responding to possible failures. This must be the primary goal for companies that rely on web-revenue (e.g., e-commerce, ads) since this downtime would be kept them from earning any revenue. Amazon's estimated cost for a one-hour downtime on Prime Day (most significant revenue day of the year) for its services is up to $100 million, as shown by (Chen et al., 2019).

Besides resilience, reliability must also be taken into consideration. Reliability is defined as the ability of an item to perform a required function, under stated conditions, for a stated time period (Bauer and Adams, 2012). The lack of knowledge regarding the infrastructure beneath, as well as security concerns of sharing the same resources with others, has presented itself as a hindrance over the years for significant companies to migrate their business-critical workloads (Sfondrini et al., 2018).

2.2 Cloud-Native Approach

As previously mentioned, the Cloud Computing paradigm has changed how the software is developed, delivered, and designed. With its development and the porting of applications from end-users' workstations to the cloud, applications started growing in complexity. This complexity eventually leads to increasing concerns regarding the monolithic development approach. One of them is the fact that, due to modules being highly coupled, compromising only one of them is enough to compromise the whole application. Also, if a single module needs scaling or fixing, the entire application must be interrupted.

To mitigate these concerns, a Cloud-Native approach appeared as an alternative. Cloud-Native applications are built specifically for the cloud; hence, they need to be scalable, fault-tolerant, easily upgradable, and secure (Gannon et al., 2017). To accomplish this, the development of new applications started following a microservices architecture where there are compartmentalization and decoupling of an application, allowing independent scalability (Namiot and Sneps-Sneppe, 2014). This methodology consists of

using minimal, loosely coupled services, each responsible for specific functions, as they improve resource utilization and also minimize waste. Another main characteristic of the Cloud-Native approach is the use of containers instead of virtual machines, due to reduced storage requirements, and faster booting time, which allows for improved elasticity and disposability (Cloud Native gives preference to provisioning new instances in case of failure, instead of trying to fix the failed instance).

The decomposition of monolithic applications into microservices improves independence among different software teams and achieves better resource usage and flexibility, as each component could be resized or replicated as the demand for resources changes.

Nonetheless, while the Cloud-Native approach has several benefits, it still has its pitfalls. As the microservices are distributed and may not reside on the same host, the infrastructure has a higher effect than on the monolithic architecture. Therefore they are more susceptible to losing performance due to problems such as latency, scheduling, among others. Consequently, assuring that the infrastructure beneath it is resilient and provides resources as expected, is highly required.

Developers deploy Cloud-Native applications on cloud platforms, where they gain access to personal spaces, allowing them to provision services (e.g., databases and caches) for use in their applications.

Provisioning VMs, aggregating logs, and scaling components are onerous tasks for cloud operators; hence, there are cloud platforms that seek to lessen this burden. Cloud Foundry is a platform intended for deploying and running applications, tasks, and services in the cloud. It seeks to minimize the efforts needed not only by operators but also from developers, enabling them to deploy and execute their applications reliably, even in an unreliable infrastructure (Winn, 2017). One of its components, BOSH, can interact with any cloud infrastructure, deploy software, monitor and heal VM's health, and also provision computational resources as needed.

The counterpart of BOSH VM management is Diego, responsible for handling containers. Diego allows running two types of applications: (i) Tasks that run once for a limited time, and (ii) Long-Running Process (LRP), which run continuously. After a task or LRP is pushed to Diego, it calls one of its internal components, the Auctioneer, to decide in which of the multiple isolated container environments the application should run. After an application starts running, it is necessary to keep track of logs and metrics of it. The component responsible for this is the Loggregator; it has agents on VMs that collect logs and metrics and forward them to temporary buffers (called Doppler). From there, the information can be used by specific programs called Nozzles that consume this data and process it.

3 RELATED WORK

To minimize potential risks caused by unknown variants, benchmarking tools can be used to compare the services each cloud provider has to offer. From the operator's perspective, data obtained from benchmarks provide insights on any problem on its infrastructure as well as its limits. With the analysis of this information, cloud providers can provide guarantees of their offering services to their users', in the form of Service-Level Agreements (SLAs) (Dillon et al., 2010), for example. Multiple Benchmark suites and frameworks have been developed for cloud applications and services. These suites are usually designed to provide testing for specific kinds of applications.

Kasture and Sanchez (Kasture and Sanchez, 2016) proposed a benchmark suite and evaluation methodology called *TailBench*. This tool focuses on providing a way to perform testing for latency-critical applications easily. It supports eight representative applications and contains workloads for them. The authors also validate their tool by testing it in a real system and simulations.

Ferdman et al. (Ferdman et al., 2012) introduced in his work the *CloudSuite benchmark suite*. This tool focus on testing applications that have workloads that scale-out to multiple machines in the datacenter. The applications tested included NoSQL databases, a Media Streaming server, and a Web Search service. The main objective of the work was to define the efficiency of the processors' micro-architecture for these workloads and to devise possible enhancements for it.

CloudCmp is a benchmark suite proposed by Li et al. (Li et al., 2010) with the primary objective of proposing a collection of tools to measure and evaluate the differences between cloud providers. The tool compares four of the major providers in terms of computing performance, scaling latency, storage response time, and cost per benchmark, among others.

Another benchmark suite proposed is the *DeathStarBench* (Gan et al., 2019). This benchmark includes six different applications (a social network, a media service, an e-commerce site, a banking system, and a drone control system). All of these were built using the concepts of microservices and are highly modular. The benchmark tool provides its tracing system to track requests between microservices. The au-

thors have used this tool to study the effects the applications have in the cloud system stack, from server design to the operating system used and programming framework.

Alhamazani et al. (Alhamazani et al., 2015) presented a framework for monitoring and benchmarking cloud applications called *CLAMBS* (Cross-Layer Multi-Cloud Application Monitoring and Benchmarking-as-a-Service Framework). He defines cloud platforms as being separated into three layers: Software-as-a-Service, Platform-as-a-Service, and Infrastructure-as-a-Service. The framework proposed by the authors allows for monitoring and benchmarking applications, and components on any of the three layers. Also, it provides the same capabilities for applications in multiple cloud provider environments.

One of the works most related to our contributions was proposed in (Chhetri et al., 2013). *Smart CloudBench* is a platform that offers benchmarking-as-a-service for major cloud providers. It automates multiple steps necessary for executing the testing of a service. The system allows the selection of the target cloud provider, one of its available benchmarking tools, and its respective workload for testing. It also collects results and generates a visualization of the metrics gathered. The authors have also implemented a prototype for Smart CloudBench that evaluated multiple cloud servers using the TPC-W benchmark.

The benchmark tools and suites previously described are capable of benchmarking different aspects of Cloud Computing, but mainly focus on benchmarking applications and cloud infrastructures and do not specifically target cloud services. Moreover, to the best of our knowledge, there is no framework for easily extending the available benchmarking tools on those solutions, as most of these tools are either inextensible or require changes to the codebase. Hence, we propose ISABEL, a benchmark framework for services that can abstract the implementation details needed for the integration of new tools and thus allowing it to be easily extensible.

4 ISABEL BENCHMARK SUITE

The adoption of benchmarking strategies for cloud services benefits both cloud providers and their clients. Through the analysis of different metrics obtained from benchmark tools, platform operators become capable of providing their clients with performance assurances. Conducting benchmarks on multiple services, such as databases and message handlers, entails the use of many different tools. With every additional tool, a new workflow must be adopted for running, gathering the output, and performing analysis over it.

Due to the unique configurations required to execute each tool (e.g., parameters and runtime dependencies), benchmarking becomes an onerous task for platform operators. Besides, problems regarding the interoperability between benchmark tools become more evident. Different tools might require different mechanisms for retrieving their output (e.g., reading from stdout and fetching from a database). Moreover, the lack of standard nomenclature for performance metrics leads to scenarios where different tools provide the same piece of information, but with different descriptions (e.g., ”AverageLatency” and ”AvgLatency”).

To lessen the burden of managing various benchmark tools, increasing their interoperability is essential. To this end, we propose ISABEL, a benchmarking suite framework for improving the evaluation of services on Cloud-Native platforms. The core motivation for ISABEL is to provide an internal service of Benchmarking-as-a-Service (BaaS) for cloud providers, enabling platform operators to benchmark their services, while also allowing benchmark developers to validate their tools against others. The proposed architecture is illustrated in Figure 1.

To deliver BaaS, ISABEL is designed as an extensible marketplace solution for ordering the benchmarking of any available service through one of its designated tools. From deploying service instances and containerized tools to retrieving benchmarking results, ISABEL is capable of performing a complete workflow through a single order.

When an order is received, a new service instance is provisioned accordingly; then, the desired benchmark tool is deployed as a container, along with all the required information for its execution, which follows next. Finally, the results are gathered by ISABEL and delivered to the client (i.e., who made the benchmarking request). A more detailed explanation of this procedure is presented further on.

While the responsibility of providing ISABEL with the means of deploying services instances fall upon platform operators, the execution of containerized tools by ISABEL is made available through one-time-only tool registration. The procedure for extending ISABEL with the registration of new benchmarks is illustrated in Figure 2.

The registration of new benchmarks requires above all else the definition of which service it supports. Hence, the service’s deployment workflow must be provided to ISABEL a priori. Benchmark developers can also query ISABEL regarding which ser-

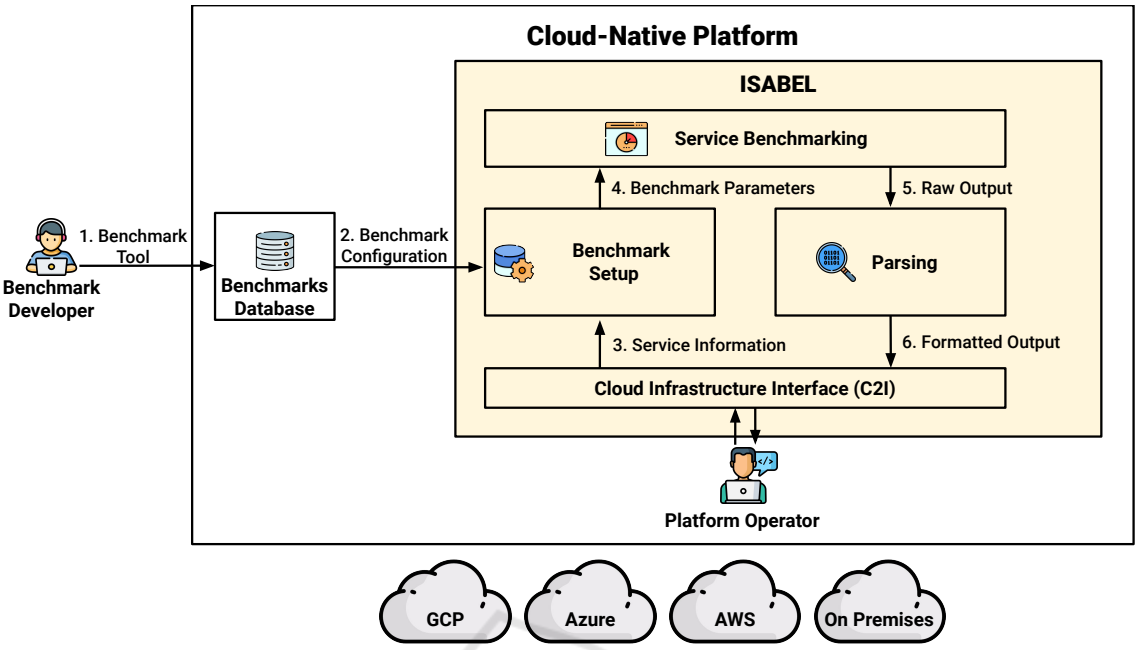

Figure 1: ISABEL architecture.

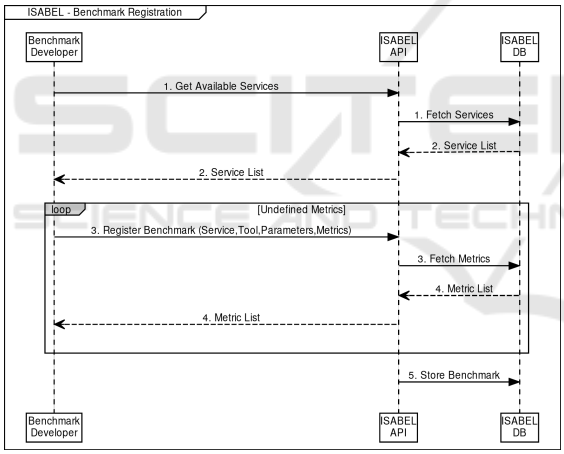

Figure 2: Procedure for the registration of a new benchmark.

vices are supported. In order to register a new benchmark, the following items must be provided:

- **Service:** The available target supported by the tool.
- **Tool:** A configuration file describing the containerized deployment and execution of the benchmark tool (e.g., a *Dockerfile*).
- **Parameters:** The information required for its execution, which will be provided to the container in the form of environment variables.
- **Metrics:** The performance information obtainable by executing the benchmark.

To promote interoperability, the registration of new benchmarks requires that their output follow a standard format, such as a key-value based *JSON*. Although the benchmarking tool's source-code might not be available for modification, the same could be achieved through the use of a wrapper for its execution, thus allowing the final output to be formatted. Additionally, to avoid misinterpretation of benchmarking results, ISABEL enforces the use of a metrics knowledge base (KB). Any metric obtained from a benchmark must be known to ISABEL at the moment of its registration. In the event one of its metrics is unknown, the registration fails. To solve this problem, each unknown metric must be either replaced with another equivalent or added to the metrics KB. Imposing a nomenclature for available metrics enables not only benchmark developers to compare their tools, but also facilitate performance analysis later on.

While our method of enforcing the registration of metrics improves the interoperability of benchmarking tools, it is not absolute; hence, one of its pitfalls is that it's prone to human error. If a benchmark developer must seek an equivalent entry or register a new one for one of its tool's metrics, nothing prevents him from missing the equivalent entry and registering a different one. In this case, there would be two entries consisting of the same information, but with different identifications. Even worse is the fact that even if these two entries were to be discovered later on, nothing could be done, as removing any of them would invalidate a registered benchmark. For these scenarios,

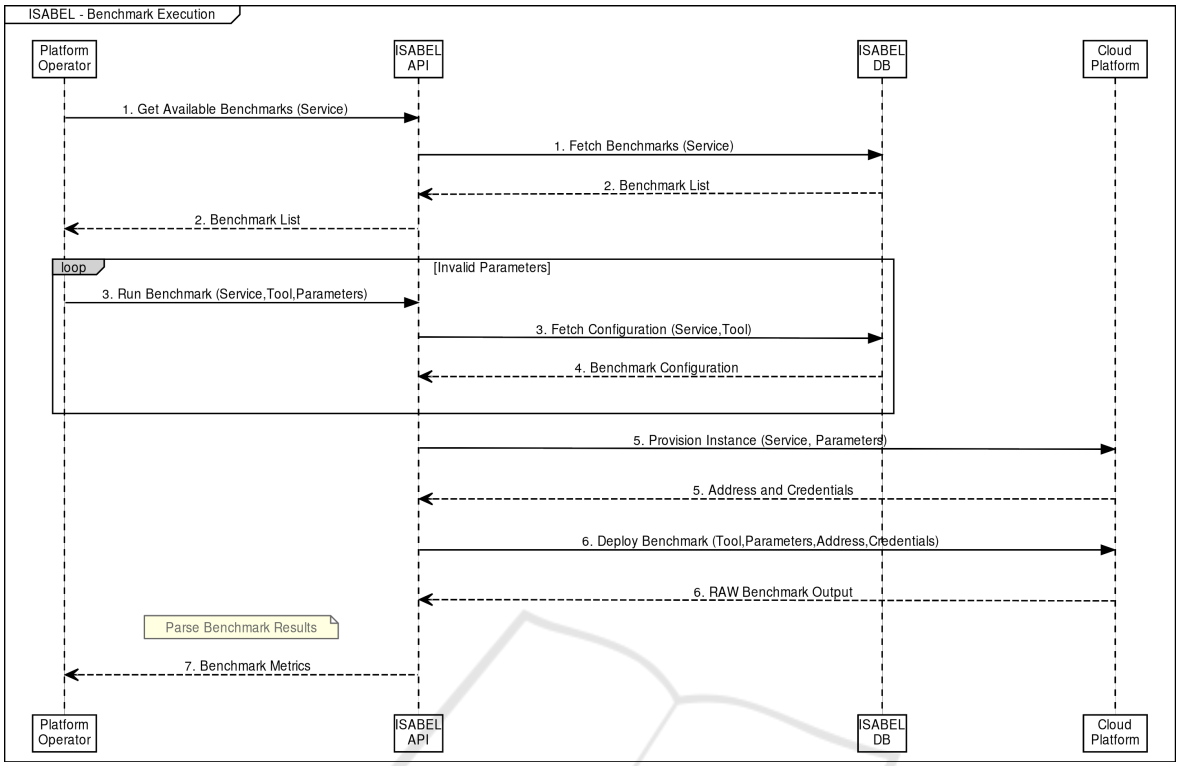

Figure 3: Procedure for executing the benchmark of a service.

we recommend that entries on the KB be able to reference others. When a problem of this kind is detected, simply changing the reference of all the affected entries to a single one would solve it, thus avoiding the re-registration of every affected benchmark.

When ISABEL successfully stores a benchmark on its database, the new benchmark's registration is complete, and it becomes available for execution. The execution procedure is illustrated on Figure 3.

In order to execute a benchmark, the following information must be provided to ISABEL: a desired benchmarking tool, its execution parameters, and a targeted service supported by the tool. Information regarding available services, benchmarking tools, and their execution parameters can be requested from ISABEL.

After receiving a benchmark request, ISABEL verifies if all the required information was provided correctly (e.g., all the execution parameters). If any inconsistency is detected on the provided information, ISABEL refuses the benchmark request. Once the information is validated, ISABEL provisions a new instance of the targeted service using the information provided by the platform operator for supporting this service. The benchmarking request may also include parameters regarding which service plan (i.e., resource configurations) to deploy. Supporting this feature relies on what capabilities the platform operator provided to ISABEL.

When the deployment is completed, ISABEL retrieves the instance's address and credentials, and deploys the containerized benchmarking tool with this information, along with the benchmarking request parameters, through environment variables. After the benchmark finishes, ISABEL gathers the output from the containerized tool and parses the obtained metrics according to the information provided during the benchmark's registration. Finally, the metrics are sent to the client as the result of the benchmarking request.

5 EXPERIMENTAL RESULTS

A Proof-of-Concept (PoC) of our proposed framework (ISABEL) was implemented for Pivotal Platform, a Cloud Foundry distribution. Pivotal Platform, formerly Pivotal Cloud Foundry (PCF), offers additional enterprise features, such as support, training, documentation, and technology certifications. Pivotal Platform obtains its PaaS capabilities through Pivotal Application Service (PAS), an application runtime. Platform operators can install additional software services (tiles) on Pivotal Platform and provide their users with different service-plans through PAS

Marketplace.

Experiments were conducted over Pivotal Cloud Foundry v2.6 and Pivotal Application Service v2.6 (PAS) runtime, deployed on a VMware vSphere v6.7 cluster. The cluster available resources consisted of two Dell PowerEdge R730 hosts, with VMware ESXi v6.7 bare-metal hypervisor, sharing a Dell Equallogic ps4100 storage through iSCSI. The hosts' hardware configuration is described in Table 1.

Table 1: Testbed hosts hardware specification.

Component	Specification
Processor	XEON E5-2650 v4 @ 2.20GHz (24/48)
Memory	192GB DDR4 @ 2400MHz
Storage	Seagate ST1000NM0023 1TB SAS @ 7200RPM x 10
Networking	1 Gigabit Ethernet

To validate ISABEL, we considered a scenario of benchmarking MySQL, a database service, using the benchmark tool Sysbench (Kopytov, 2012). To this end, we deployed ISABEL on PAS and created a Dockerfile to execute the SysBench benchmark tool (Kopytov, 2012). We also provided ISABEL with a workflow for enabling the deployment of Pivotal MySQL service instances. For our experiments, we selected three different MySQL service plans (resource configurations). These configurations are presented on Table 2. A performance analysis over the metrics obtained from the benchmark is illustrated in Figure 4.

We centered our analysis over the number of executed queries and average query latency. Our tests have shown that there is a relation between the number of CPU cores and the number of executed queries. Granted that MySQL database is a CPU-bound application (Gregg, 2019), increasing the number of CPU cores impacts the number of queries that can be processed. Therefore, as queries are processed faster, increasing the number of cores also decreases query latency.

While this assessment still holds, for there is a decrease of query latency as more CPU cores become available (small to large), the decreasing rate differs, even though the increase of CPU cores is equivalent

Table 2: VM configurations based on available CPU cores.

Configuration	CPU (Cores)	Memory (GB)	Disk (GB)
Small	1	16	16
Medium	4	16	128
Large	16	16	64

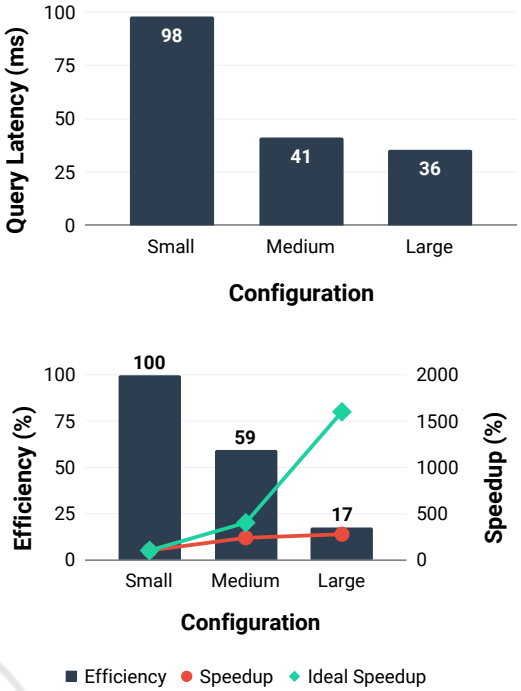

Figure 4: Number of executed queries (top) and average latency for each query (bottom).

(small to medium, then medium to large). Hence, we analyzed the speedup and efficiency of each service configuration. Whereas the speedup of an application that scales well is expected to increase at a similar rate to increase the number of cores, efficiency indicates how well an application uses the available resources. An application being stressed due to benchmarking may still not be able to use all available resources it was provided with, leaving them idle.

On a typical scenario of provisioning resources, where the pricing increases according to the selected configurations, we can deduce the following: Small is the best configuration in terms of costs since it achieves the best efficiency over the others (40.77% and 82.78%, respectively), fully utilizing the provisioned resources. Medium sacrifices some efficiency (40.77%) for an improvement in performance (136.93% from the small configuration); hence it provides the best cost-benefit. Large is the best when considering only the performance, as it provides an improvement above the others (275.46% and 38.53%, respectively).

6 CONCLUSIONS

With the Cloud-Native architecture approach of developing software as micro-services, applications are

capable of allocating only the necessary resources by scaling micro-services according to current workloads. However, this approach leaves the application's performances more dependent on the cloud services they interact with.

Platform operators use benchmarks to provide users with reliability assurances for their services. However, the more benchmarking tools are used, the more time-consuming their management becomes. Therefore, in this paper, we present ISABEL, a benchmark suite framework for Cloud-Native platforms. ISABEL lessens the burden of benchmarking multiple services by providing an extensible benchmark marketplace.

We describe ISABEL's architecture and the procedures of execution and registration of new benchmark tools. A proof-of-concept implementation on Pivotal Platform is presented, and metrics obtained by benchmarking a service are discussed. As future work, we intend to extend the capabilities of ISABEL to analyze metrics obtained from benchmarking services, providing a piece of more detailed performance information for platform operators.

ACKNOWLEDGEMENTS

This work was supported by the PDTI Program, funded by Dell Computadores do Brasil Ltda (Law 8.248 / 91).

REFERENCES

Alhamazani, K., Ranjan, R., Jayaraman, P. P., Mitra, K., Liu, C., Rabhi, F., Georgakopoulos, D., and Wang, L. (2015). Cross-layer multi-cloud real-time application qos monitoring and benchmarking as-a-service framework. *IEEE Transactions on Cloud Computing*, 7(1):48–61.

Bauer, E. and Adams, R. (2012). *Reliability and availability of cloud computing*. John Wiley & Sons.

Binnig, C., Kossmann, D., Kraska, T., and Loesing, S. (2009). How is the weather tomorrow?: Towards a benchmark for the cloud. In *Proceedings of the Second International Workshop on Testing Database Systems*, DBTest '09, pages 9:1–9:6, New York, NY, USA. ACM.

Chen, J., He, X., Lin, Q., Xu, Y., Zhang, H., Hao, D., Gao, F., Xu, Z., Dang, Y., and Zhang, D. (2019). An empirical investigation of incident triage for online service systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 111–120. IEEE.

Chhetri, M. B., Chichin, S., Vo, Q. B., and Kowalczyk, R. (2013). Smart cloudbench–automated performance benchmarking of the cloud. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 414–421. IEEE.

Dillon, T., Wu, C., and Chang, E. (2010). Cloud computing: Issues and challenges. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 27–33.

Ferdman, M., Adileh, A., Kocberber, O., Volos, S., Alisafaee, M., Jevdjic, D., Kaynak, C., Popescu, A. D., Ailamaki, A., and Falsafi, B. (2012). Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *Acm sigplan notices*, 47(4):37–48.

Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., Bruno, A., Hu, J., Ritchken, B., Jackson, B., et al. (2019). An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 3–18.

Gannon, D., Barga, R., and Sundaresan, N. (2017). Cloud-native applications. *IEEE Cloud Computing*, 4(5):16–21.

Garg, S. K., Versteeg, S., and Buyya, R. (2011). Smicloud: A framework for comparing and ranking cloud services. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 210–218.

Gregg, B. (2019). *BPF Performance Tools*. Addison-Wesley Professional, City.

Jin, S., Seol, J., and Maeng, S. (2013). Towards assurance of availability in virtualized cloud system. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 192–193.

Kasture, H. and Sanchez, D. (2016). Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–10. IEEE.

Kopytov, A. (2012). Sysbench manual. *MySQL AB*, pages 2–3.

Li, A., Yang, X., Kandula, S., and Zhang, M. (2010). Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14.

Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

Namiot, D. and Sneps-Sneppe, M. (2014). On micro-services architecture. *International Journal of Open Information Technologies*, 2(9):24–27.

Sfondrini, N., Motta, G., and Longo, A. (2018). Public cloud adoption in multinational companies: A survey. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 177–184.

Winn, D. C. (2017). *Cloud Foundry: The Definitive Guide: Develop, Deploy, and Scale*. " O'Reilly Media, Inc.".