ENOS: a Holistic Framework for Conducting Scientific Evaluations of OpenStack

Ronan-Alexandre Cherrueau, Adrien Lebre, Dimitri Pertin, Anthony Simonet, Matthieu Simonin

ENOS: a Holistic Framework for Conducting Scientific Evaluations of OpenStack

Ronan-Alexandre Cherrueau , Adrien Lebre , Dimitri Pertin , Anthony Simonet , Matthieu Simonin

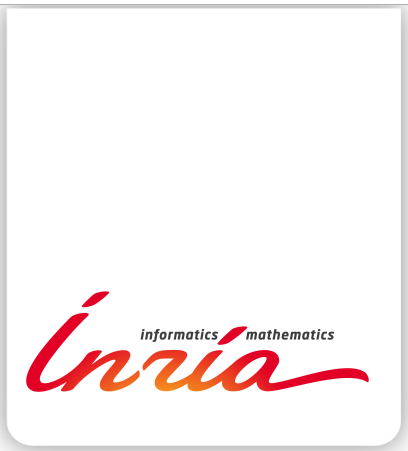
TECHNICAL REPORT

N° 485

November 2016

Project-Team Discovery

ISSN 0249-0803 ISRN INRIA/RT--485--FR+ENG



ENOS: a Holistic Framework for Conducting Scientific Evaluations of OpenStack

Ronan-Alexandre Cherrueau , Adrien Lebre , Dimitri Pertin ,
Anthony Simonet , Matthieu Simonin

Project-Team Discovery

Technical Report n° 485 — November 2016 — 12 pages

Abstract: By massively adopting OpenStack for operating small to large private and public clouds, the industry has made it one of the largest running software project. Driven by an incredibly vibrant community, OpenStack has now overgrown the Linux kernel. However, with success comes an increased complexity; facing technical and scientific challenges, developers are in great difficulty when testing the impact of individual changes on the performance of such a large codebase, which will likely slow down the evolution of OpenStack. In the light of the difficulties the OpenStack community is facing, we claim that it is time for our scientific community to join the effort and get involved in the development and the evolution of OpenStack, as it has been once done for Linux. However, diving into complex software such as OpenStack is tedious: reliable tools are necessary to ease the efforts of our community and make science as collaborative as possible.
In this spirit, we developed ENOS, an integrated framework that relies on container technologies for deploying and evaluating OpenStack on any testbed. ENOS allows researchers to easily express different configurations, enabling fine-grained investigations of OpenStack services. ENOS collects performance metrics at runtime and stores them for post-mortem analysis and sharing. The relevance of ENOS approach to reproducible research is illustrated by evaluating different OpenStack scenarios on the Grid'5000 testbed.

Key-words: Performance, Reproducibility, OpenStack, Cloud, Visualization, Control-plane, Data-plane

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

ENOS: un framework holistique pour l'évaluation d'OpenStack

Résumé : Openstack, poussé par les industriels pour la gestion de leur infrastructure virtualisée et animé par une communauté très active, a dépassé le développment du noyau linux. En même temps que le succès, OpenStack connaît également une complexité grandissante. Les développeurs sont désormais en difficulté lorsqu'il faut vérifier l'impact de changements dans une large base de code.

À la lumière de ces difficultés, nous pensons qu'il est temps pour la communauté scientifique de se joindre aux efforts et d'être impliquée dans le développement et les évolutions futures d'OpenStack comme cela a été le cas auparavant pour le noyau Linux.

Dans cet esprit, nous avons développé ENOS, un framework qui s'appuie sur la technologie des conteneurs pour déployer et évaluer OpenStack sur différentes plateformes. Avec ENOS, les chercheurs peuvent facilement exprimer différentes configurations permettant une analyse fine des services constituant OpenStack. ENOS collecte automatiquement des métriques de performance et les stocke pour des analyses *post-mortem*. La pertinance d'ENOS dans le cadre d'expérimentation reproductible est illustré via différents scénario sur la plateform Grid'5000.

Mots-clés : Performance, Reproductibilité, OpenStack, Coud, Visualisation, Control-plane, Data-plane

ENOS: a Holistic Framework for Conducting Scientific Evaluations of OpenStack

Ronan-Alexandre Cherrueau, Dimitri Pertin, Anthony Simonet, Adrien Lebre
Inria, Mines Nantes, LINA
Nantes, France
Email: firstname.lastname@inria.fr
,
Matthieu Simonin
Inria, IRISA
Rennes, France
Email: firstname.lastname@inria.fr

By massively adopting OpenStack for operating small to large private and public clouds, the industry has made it one of the largest running software project. Driven by an incredibly vibrant community, OpenStack has now overgrown the Linux kernel. However, with success comes an increased complexity; facing technical and scientific challenges, developers are in great difficulty when testing the impact of individual changes on the performance of such a large codebase, which will likely slow down the evolution of OpenStack. In the light of the difficulties the OpenStack community is facing, we claim that it is time for our scientific community to join the effort and get involved in the development and the evolution of OpenStack, as it has been once done for Linux. However, diving into complex software such as OpenStack is tedious: reliable tools are necessary to ease the efforts of our community and make science as collaborative as possible.

In this spirit, we developed ENOS, an integrated framework that relies on container technologies for deploying and evaluating OpenStack on any testbed. ENOS allows researchers to easily express different configurations, enabling fine-grained investigations of OpenStack services. ENOS collects performance metrics at runtime and stores them for post-mortem analysis and sharing. The relevance of ENOS approach to reproducible research is illustrated by evaluating different OpenStack scenarios on the Grid'5000 testbed.

I. Introduction

Although the adoption of Cloud Computing has been largely favored by public offers (Amazon EC2 and Microsoft Azure, to name a few), numerous private and public institutions have been contributing to the development of open-source projects in charge of delivering Cloud Computing management systems [**?**], [**?**], [**?**]. In addition to breaking vendor lock-in, these *operating systems* of Cloud Computing platforms enable administrators to deploy and operate private cloud offers, avoiding issues such as data-jurisdiction disputes, latency constraints, etc.

After more than six years of intensive effort, the OpenStack software suite has become the de facto open-source solution to operate, supervise and use a Cloud Computing infrastructure [**?**]. The OpenStack community gathers more than 500 organizations, including large groups such as Google, IBM and Intel. The software stack relies on tens of services with 6-month development-cycles.

Despite the current dynamicity of the whole ecosystem that makes it incredibly hard to keep up with, its adoption is still growing and the stack is now being used in a large variety of areas such as public administrations, e-commerce and science[1]. With the now undeniable success of OpenStack, we argue that it is time for the scientific community to get involved and contribute to the OpenStack software in the same way it has been once done for the Linux ecosystem, in particular in the HPC area. A major involvement of our community will enable OpenStack to cope better with ongoing changes in the Cloud Computing paradigm such as the Fog and Edge Computing proposals and the IoT application requirements. However, diving into the OpenStack ecosystem is a tedious task. The whole software stack represents more than 20 million lines of code including 2 million lines of Python code for core services alone.

To help developers and researchers identify major weaknesses of a complex system such as OpenStack and to facilitate the evaluation of proposed improvements, we designed ENOS[2]. ENOS is a free software framework that leverages container technologies and "off-the-shelf" benchmarks for automating reproducible evaluations of OpenStack in a flexible and extensible way. To the best

[1] See http://superuser.openstack.org/ for further information

[2] *Experimental eNvironment for OpenStack* – ENOS: https://github.com/BeyondTheClouds/enos

of our knowledge, ENOS is the first holistic approach for evaluating OpenStack. That is, it has been designed with the *Experimentation-as-Code* vision: every step of the experimentation workflow, from the configuration to the results gathering and analysis, can be automated [**?**].

Although several performance studies of OpenStack [**?**], [**?**], [**?**] have been achieved in the recent years, they all present weaknesses. First, they have been conducted by using ad-hoc approaches that prevent researchers to reproduce them. Second, the underlying complexity and the aforementioned velocity of the OpenStack ecosystem make these studies deprecated in less than a year. As a consequence, the results presented by these studies cannot determine whether specific revisions/extensions in the code provide significant benefits. Thanks to ENOS, researchers and developers alike can now evaluate the performance of distinct OpenStack deployment scenarios, and compare the collected results to both identify limitations and validate improvement proposals.

ENOS has been built on top of a containerized deployment model of OpenStack, where each OpenStack service is encapsulated in a dedicated container. This allows to easily express, deploy and evaluate different configurations enabling fine-grained investigations of every OpenStack service, including the latest versions available on the OpenStack trunk. Moreover, ENOS has been designed around pluggable mechanisms:

- The extensible deployment engine of ENOS allows to deploy OpenStack on various infrastructures, *e.g.*, testbed platforms such as Grid'5000 [**?**] and Chameleon [**?**], public or private cloud infrastructures such as Amazon EC2 [**?**], on OpenStack itself or on simpler systems such as Vagrant [**?**]).
- ENOS natively supports different types of benchmarking suites such as Rally [**?**] and Shaker [**?**] in addition to allowing the addition of customized ones. This enables ENOS end-users to conduct either control plane or data plane experiments. In the former case, understanding the performance of the controller nodes (*i.e.*, the nodes in charge of supervising the OpenStack infrastructure) is the objective of the experiments. In the latter case, the goal is to evaluate the performance from the application viewpoint (that is, understanding the performance an application can reach when it is executed on top of a particular OpenStack deployment). For both kinds of experiment, the way OpenStack is deployed in terms of configuration parameters and hardware topology has an impact on performance.

Finally, ENOS comes with generic visualization tools that provide different views using diagrams, plots and tables of gathered metrics. We claim it is another valuable feature of ENOS as it allows the achievements of explanatory and exploratory experiments. While in the first case, researchers know what data they should look for, identifying what information is relevant is more complicated when the behavior of the system is unknown or when side effects can occur in unexpected parts of the system. In this second case, providing a synthetic view of the gathered information makes the analysis and the identification of irregular patterns much easier. ENOS delivers such synthetic views either in real-time or *a posteriori*.

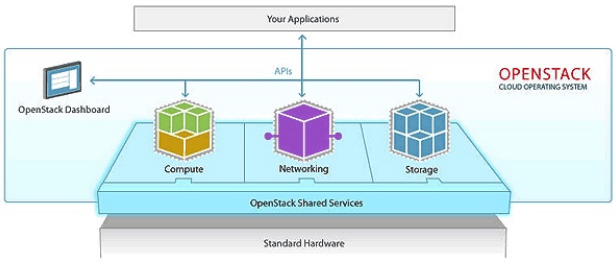


Fig. 1. OpenStack Overview

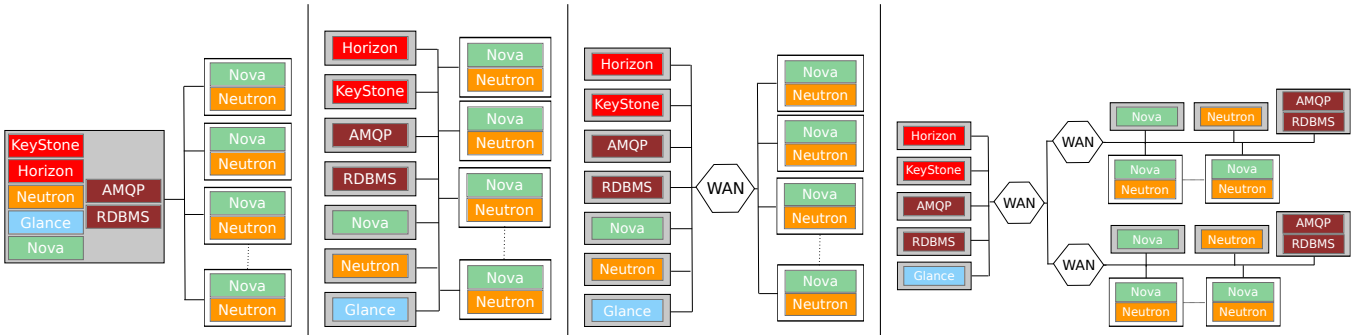
The remaining of this article is as follows. Section II presents the different technologies we used to build the ENOS framework. The framework itself is discussed in Section III. To illustrate the possibility offered by our framework, we discuss series of experiments that have been conducted thanks to ENOS in Section IV. Related works are presented in Section V. Finally Section VI concludes and discusses future research and development actions.

II. Background

In this section, we first give an overview of OpenStack, then we describe the technologies used to implement the ENOS framework. While the goal of the first part is to illustrate the richness of the OpenStack ecosystem and the different possibilities offered to deploy it, the second part may look rather technical. However, we believe it is more relevant to use already existing technologies than reinventing the wheel. The objective of this second part is to present these technologies we used as building blocks.

A. OpenStack

OpenStack [**?**] is an open-source project that aims to develop a complete Cloud Computing software stack. Figures 1 and 2 are well known from the OpenStack community. The first one presents the general vision of OpenStack with the three expected capabilities of IaaS platforms: Compute, Network and Storage. Applications at the top can request compute, network and storage resources through a high-level API. OpenStack components in the middle layer communicate through shared services. The second figure shows the historical



(a) single node, single location

(b) multi nodes, single location

(c) multi nodes, multi locations

(d) multi regions

Fig. 3. OpenStack deployment scenarios. Gray squares represent control nodes whereas white squares correspond to the compute nodes (*i.e.*, the nodes that host VMs).

core services of OpenStack. Briefly, *Keystone* provides service discovery and client authentication. *Horizon* provides a web-based user interface. *Nova* provides on-demand access to compute resources (*i.e.*, VMs). *Neutron* provides "network connectivity as a service" between interface devices (*e.g.*, vNICs) managed by other OpenStack services (*e.g.*, *Nova*). *Glance* provides services to discover, register, and retrieve VM images. Finally, *Swift* is a distributed object/blob store similar to Amazon S3. This architecture is comparable with the generic reference proposed by Moreno [**?**].

From the technical point of view, OpenStack is composed of two kinds of nodes: on the one hand, the compute/storage/network nodes are dedicated to deliver the XaaS capabilities, such as hosting VMs (*i.e.*, data plane); on the other hand, the control nodes are in charge of executing the OpenStack services (*i.e.*, control plane).

OpenStack services are organized following the *Shared Nothing* principle. Each instance of a service (*i.e.*, service worker) is exposed through an API accessible through a *Remote Procedure Call* (RPC) system implemented, on top of a messaging queue or via web services (REST). This enables a weak coupling between services and thus a large number of deployment possibilities, according to the size of the infrastructure and the capacity the cloud provider intends to offer. Nevertheless, we highlight that even if this organization of services respects the *Shared Nothing principle*, most services create and manipulate logical objects that are persisted in shared databases. While this enables service workers to easily collaborate, it also limits the deployment possibilities as each DB represents a single point of failure [**?**].

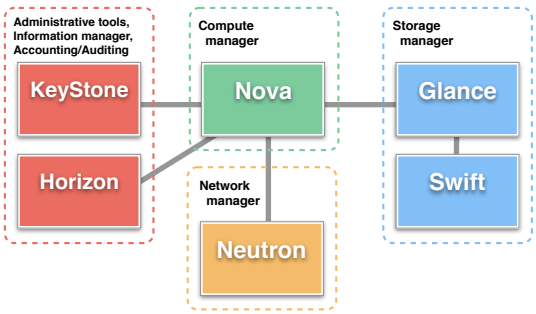


Fig. 2. OpenStack core-services

Figure 3 illustrates four deployment architectures for OpenStack services on a single site (Figures 3(a) and 3(b)) and on multiple sites linked through a WAN connection (Figures 3(c) and 3(d)), only with the essential core services (without *Cinder*, *Ironic*, *Heat*), for the sake of simplicity.

Figure 3(a) corresponds to a minimal OpenStack deployment: all services have been deployed on a dedicated controller node. Only the agents (`nova-compute` and `neutron-agent`) that are mandatory to interact with hypervisors on the compute nodes have been deployed.

In the second scenario, illustrated in Figure 3(b), each service has been deployed on a dedicated control node (there is no change for the compute nodes). While entirely feasible, this second scenario is rarely deployed. In most cases, a control node executes several services (*e.g.*, Keystone and Horizon are often deployed on the same node). Nevertheless, we highlight that for large scale infrastructures, isolating important services such as Nova and Neutron becomes mandatory. This enables the execution of several instances of sub-services such as the `nova-scheduler` on the node.

The third and fourth scenarios correspond to WAN-wide architectures. Although those are rarely deployed in production environments yet, industrial actors and telcos, in particular, are investigating to what extent current OpenStack mechanisms can handle Fog and Edge computing infrastructures [**?**]. In Figure 3(c), the control plane is deployed on one site of the infrastructure, and only compute nodes have been deployed to a remote

location.

Finally, Figure 3(d) corresponds to the guidelines presented on the OpenStack website to supervise a multisite infrastructure. This illustrates the current agitation around the architecture of OpenStack deployments and justifies an important feature of ENOS, namely the possibility of testing multiple OpenStack deployment topologies transparently.

B. Deployment of OpenStack

Due to the richness and complexity of the OpenStack ecosystem, making its deployment easy has always been an important topic. Among all the deployment solutions that are available, we chose to use Kolla [**?**]. Kolla provides production ready containers and deployment tools for operating OpenStack infrastructures. In Kolla, each OpenStack service is encapsulated with its dependencies in a dedicated container. Container images can be built on demand, stored and used during the deployment. Kolla features many default behaviors, allowing quick prototyping, but they are fully customizable: vanilla or modified versions of OpenStack can be installed, deployment topologies can be adapted to the user's needs and configuration of all the services can be changed. To perform remote actions such as deploying software components, Kolla uses the Ansible deployment engine [**?**]. Ansible gathers hosts on groups of machines on which specific tasks are applied. This group mechanism in play is very flexible and thus allows alternative deployment topologies to be specified. By default, Kolla defines two levels of groups. First on a per-service basis, and second on a logical basis. The former allows, for instance, single service to be isolated on a dedicated node. The latter are groups of OpenStack services based on typical deployment practices. An OpenStack deployment is at least composed of four logical groups: i) Control, for hosting the database, messaging middleware, and the various APIs, ii) Storage, for storing block devices, iii) Network, Neutron services and iv) Compute, for hosting the Nova Compute and Neutron agents.

C. Evaluation of OpenStack

Measuring the performance of a cloud infrastructure in a rigorous and comparable way is an important challenge for our community. The Standard Performance Evaluation Corporation has recently proposed the SPEC Cloud benchmark: *The benchmark suite's use is targeted at cloud providers, cloud consumers, hardware vendors, virtualization software vendors, application software vendors, and academic researchers*. Although the SPEC Cloud benchmark can be easily integrated into ENOS thanks to the pluggable approach (see Section III), the license issue of SPEC benchmark does not enable us to provide ENOS with it by default. Instead, ENOS comes with two open source benchmarks that are Rally and Shaker.

1) Rally: Rally is the official benchmark suite for OpenStack; it allows to test the control plane by injecting requests to running services using their corresponding Python clients. It can test a running cloud or deploy a cloud first, making it an all-in-one solution for both development and production testing. Rally executes scenarios that can be configured by JSON or YAML files. A scenario includes *Runner* options (*e.g.*, the number of times a request is performed or how many parallel threads are used to perform the requests), *Context* options (*e.g.*, how many users and tenants must be used for the injection) and scenario-specific configuration, *e.g.*, the number of VMs to boot, or the image file to use when creating an appliance). SLA options can also be provided. In this case, Rally will consider a scenario to fail if the requested SLA is not met. Execution times, failures, and SLA violations are collected and stored in a database. From this database, Rally can also generate HTML and JSON.

2) Shaker: Shaker is a framework for data plane testing of OpenStack. It currently targets synthetic benchmarks execution (for instance iperf3 [**?**], flent [**?**]) on top of instances. Shaker supports the definition and the deployment of different instances and network topologies. The possible scenarios include extensive evaluation of network capabilities of an OpenStack cloud.

D. Analysis of OpenStack

Analysis of OpenStack is mostly based on metrics generated during the experiment and relies on three components: metrics agents, metrics collector and metrics visualization. Those components are loosely coupled, allowing for alternatives to be plugged in when necessary. In the current implementation, metric agents are cAdvisor [**?**] and collectd [**?**]. They are responsible for sending metrics from hosts to the collector. Metrics can be enabled or disabled at will through the metrics agents configuration files. Metrics collector relies on the InfluxDB timeseries optimized database [**?**]. Visualization is enabled by Grafana [**?**], a dashboard composer. It allows to query multiple data sources and to display them in a Web browser. Dashboards can be saved and shared between users, increasing the reusability of user-made visualizations. Note that Grafana is suitable for both explanatory visualization (with predefined dashboards) and exploratory visualizations, as dashboard can be built interactively.

III. ENOS

Evaluating a complex appliance such as the OpenStack software suite can be divided into four logical phases. The first phase consists in getting raw resources;

```
resources:
  control: 1
  compute: 2
```

(a) General description

```
resources:
  paravance:
    control: 1
  econome:
    compute: 2
```

(b) Extended version for Grid'5000

Fig. 4. ENOS Resources Description Examples

the second one deploys and initializes the selected version of OpenStack over these resources; the third phase invokes the benchmarks to be executed; finally, the fourth phase consists in analyzing results of the evaluation. To help engineers/researchers in tackling all these phases, we developed ENOS[3], a holistic approach for the evaluation of OpenStack. After presenting the resource description language we used to configure ENOS, this section describes how each phase has been implemented, and in particular how by abstracting fundamental principles of each phase, ENOS can address performance evaluations for any infrastructure.

A. ENOS Description Language for Flexible Topologies

The description of the resources to acquire as well as the mapping of the different services on top of those resources is made with a YAML resource description language. In other words, ENOS comes with a dedicated language that describes what OpenStack service will be deployed on which resource. This language offers a very flexible mechanism that lets ENOS end-users specify and evaluate OpenStack performance over a large set of topologies. However, OpenStack is made of numerous services and writing this description is tedious. For this reason, ENOS reuses Kolla service groups (see II-B) to gather many OpenStack services under the same logical name, which drastically reduces the description size. For instance, the small description in Figure 4(a) describes a single-node deployment topology similar to the one in Figure 3(a). This description says: *"provide one resource for hosting control services and two others for hosting compute services"*.

In the context of ENOS, a resource is anything running a Docker daemon and that ENOS can SSH to. This could be a bare-metal machine, a virtual machine, or a container resource according to the testbed used for conducting the experiments.

Moreover, we emphasize that the language is resource provider dependent in order to handle infrastructure specificities. For instance, on Grid'5000 [**?**], the language has been extended to specify the name of physical clusters where resources should be acquired, as depicted in Figure 4(b). In this description, the paravance cluster (located in Rennes) will provide resources for control services and the econome cluster (located in Nantes) will provide resources for the compute nodes.

Last but not the least, it is noteworthy that more advanced deployment topologies can be defined by coupling resources with names of OpenStack services.

Isolating a service on a dedicated resource is as simple as adding its name to the description. For instance, adding `rabbitmq: 1` at the end of the description on Figure 4(a) tells ENOS to acquire a dedicated resource for the AMQP bus. Henceforth, the bus will no longer be part of the control resource but deployed on a separate resource at the deployment phase. Obviously, it is possible to do the same for the database, `nova-api`, `glance`, `neutron-server` ... and hence get a multi-nodes topology similar to the one presented in Figure 3(c).

Scaling a service simply requires increasing the number of resources allocated to this service into the description. For instance, increasing the value of `rabbitmq: 1` to `rabbitmq: 3` tells ENOS to acquire three dedicated resources for the AMQP bus. Henceforth, the deployment phase will deploy a cluster composed of three RabbitMQ.

These two characteristics of the language allow a very flexible mechanism to both isolate and scale services.

B. ENOS Workflow

In the following, we describe the four steps that are achieved by ENOS.

1) `enos up`: Getting Resources Phase: Calling `enos up` launches the first phase that acquires the resources necessary for the deployment of OpenStack. To get these resources, ENOS relies on the aforementioned description and the notion of *provider*. A provider implements how to get resources on a specific infrastructure and thus makes this job abstract to ENOS. With such mechanism, an operator can easily evaluate OpenStack over any kind of infrastructure by implementing the related provider. A provider can also be given by the support team of an infrastructure, independently of any particular OpenStack evaluation project. In other words for each testbed, an extended version of the ENOS DSL and a *provider* should be available. Currently, ENOS supports two kinds of infrastructures; the first one gets bare-metal resources from the Grid'5000 testbed [**?**]; the second one uses a VM based on Vagrant [**?**]. We motivate these two choices as follow: an ENOS end-user would go with bare-metal providers such as Grid'5000 for performance evaluation at various scales, and prefer the quick Vagrant deployment for testing a particular feature of one service. We emphasize that additional drivers for Amazon or any other system can be easily

[3]*Experimental eNvironment for OpenStack* – ENOS: https://github.com/BeyondTheClouds/enos

implemented, as it should correspond to less than 500 lines of Python code. By providing an OpenStack driver, it would also be possible to evaluate new OpenStack features on top of an existing OpenStack deployment.

The output of the first phase is a list of addresses which reference resources, together with the name of OpenStack services to deploy over each resource. This way, ENOS will be able to initiate a SSH connection to these resources during the next phase and deploy the requested OpenStack services.

2) `enos init`: Deploying and Initializing OpenStack Phase: The `init` phase deploys and initializes OpenStack with Kolla. Concretely, ENOS uses the list of resources and services provided by the previous phase and writes them into a file called the *inventory file*. Kolla then uses this file to deploy, in a containerized manner, OpenStack services onto the correct resources.

The Kolla tool runs each OpenStack service in an isolated container which presents a huge advantage for collecting metrics such as CPU, memory, and network utilization. Indeed, in addition to isolation, container technologies offer fine-grained resource management and monitoring capabilities [**?**]. This means that it is possible to collect the current resource usage and performance information, whatever the container runs through a standard API. This feature lets ENOS implement a generic metrics collection mechanism that stands for every OpenStack service.

Under the hood, ENOS relies on *cAdvisor* (see II-D) to implement this generic collection mechanism. Actually, ENOS deploys a monitoring stack that includes cAdvisor for CPU/memory/network usage and *collectd* for some service specific information such as the number and the type of requests performed on the database.

At the end of this phase, OpenStack has been deployed as defined by the ENOS configuration file. The next phase is the one that performs the execution of benchmarks.

3) `enos bench`: Running Performance Evaluation Phase: The `bench` phase runs benchmarks to stress the platform. By default, ENOS comes with Rally and Shaker frameworks. However, the ENOS abstractions allow end-users to plug any custom benchmarks.

a) Core benchmarking tools: ENOS unifies the description and execution of the workloads to run against a given OpenStack installation. A workload in ENOS is composed of generic scenarios that will be run in sequence. Given a set of parameters to apply to a generic scenario, the engine will run a concrete scenario. Workloads are described in dedicated files, as shown in Figure 5 where two Rally scenarios are included. In this workload definition, ENOS will run six concrete scenarios: four "boot and delete" which parameters are in the cartesian product of the top level arguments and two "boot and list", because local arguments shadow global ones. Workloads based on Shaker follow the same pattern.

```
rally:
    args:
      concurrency:
        - 25
        - 50
      times:
        - 10
        - 20
    scenarios:
      - name: boot and delete
        file: nova-boot-delete.yml
      - name: boot and list
        file: nova-boot-list.yml
        args:
          times:
            - 20
```

Fig. 5. Benchmark definition example

b) Custom benchmarking tools: To facilitate the extension of the benchmark phase with new benchmarking frameworks that won't fit the previous workload definition, ENOS exposes the list of resources together with the name of OpenStack services deployed over each resource. This way, one can easily develop an ah-doc solution that, using the resources list, deploys and runs a another benchmark framework.

4) `enos inspect`: Analysing the Evaluation Phase: The `inspect` phase generates all components needed for the analyses of the performance evaluation.

Metrics gathering is twofold. First, ENOS collects general metrics (CPU/memory usage, network utilization, opened sockets ...). Second, it is able to store specific statistics offered by the benchmarking suite used. The former relies on a set of agents whose role is to send metrics to a collector. The latter is specific to the benchmarking suite that is executed and occurs during the `inspect` phase. Similarly to the previous section, integrating custom benchmarking tools may require extending ENOS to retrieve the relevant reports.

ENOS allows general metrics to be observed in real-time during the experiment. Preconfigured dashboards are indeed accessible through a Web interface. ENOS's `inspect` gather a larger source of information since they include configuration files, logs of OpenStack services, all the collected metrics and all the reports generated by the benchmarking suite used. ENOS can then build a virtual machine image embedding all these data and tools to allow post-mortem exploration.

IV. Experiments

As explained in the previous sections, ENOS enables researchers to easily evaluate the performance of distinct OpenStack deployments in a reproducible manner. Thus, it can be used to compare the collected metrics in order to both identify limitations and validate proposals for improvement. We propose here two experiments that depict how ENOS can be used in these directions.

The first experiment compares control plane evaluations while the number of compute nodes scales up to 1,000. This kind of evaluation illustrates how operators and developers can use ENOS to identify limiting services at both coarse and fine grain (*e.g.*, general services such as RabbitMQ as well as sub-services such as `nova-conductor`) through the exploration of general metrics.

The second experiment shows that ENOS can be used to compare data-plane evaluations in order to help developers validate an improvement proposal with given performance metrics. In this experiment, ENOS is used to validate the introduction of a feature in Neutron by providing an explanatory visualization of the network traffic observed from compute nodes on two OpenStack deployments (one of which embeds the new feature).

Both experiments have been executed on the *paravance* cluster of the Grid'5000 [**?**] testbed. This cluster is composed of 72 nodes, each featuring two octa-core Intel Xeon E5-2630v3 CPUs @ 2.4 GHz, 128 GB of RAM, two 600 GB HDD and two Gigabit Ethernet NICs. OpenStack was based on the Mitaka release.

A. Control-plane performance study

In this section, we demonstrate how ENOS can be used to perform control-plane evaluations. This exploratory evaluation studies the effect of the number of compute nodes on the performance of OpenStack services in an idle state, *i.e.*, when there is no request from end-users/administrators. In other words, our goal is to analyze the resources that are consumed by the OpenStack services themselves.

For this experiment, we deploy an OpenStack cloud multiple times on Grid'5000 with ENOS and vary the number of compute nodes from 100 to 1,000 between two deployments. We use the "fake driver" capability of Nova to deploy 50 `nova-compute` containers per physical node, thus allowing to reach 1,000 *fake compute-nodes* with 20 physical machines. Note that two other physical machines were used to host Control and Neutron groups respectively. The fake driver is a hypervisor that does not bootstrap VMs but performs the same routine tasks to maintain the state of its local –fake– instances; thus, its use has no effect on the control-plane compared to an actual hypervisor. In addition to the compute nodes, the deployed cloud includes a control and a network node. The former hosts most OpenStack components such as the database and the monitoring services while the latter hosts the Neutron services.

Nb. of compute nodes	100	200	500	1,000
Nova Conductor	1.22	2.00	3.68	7.00
Neutron server	0.14	0.21	0.39	0.69
HAProxy	0.11	0.18	0.33	0.49
RabbitMQ	0.98	1.65	3.11	5.00
MariaDB	0.03	0.06	0.13	0.21

TABLE I
Average CPU usage of OpenStack services with varying the number of compute nodes (in number of cores).

Nb. of compute nodes	100	200	500	1,000
Nova Conductor	2.47	2.47	2.45	2.47
Neutron server	419	420	359	441
HAProxy	6.27	6.32	7.04	8.71
RabbitMQ	1,628	2,580	5202	11,520
MariaDB	502	546	570	594

TABLE II
Maximum RAM usage of OpenStack services with varying the number of compute nodes in megabytes.

Once the deployment is over, metrics are collected for one hour without performing any user or administrative request. ENOS enables us to individually inspect these metrics for each service. Table I and II present respectively the CPU and RAM consumption of representative services during this one-hour idle period. The observed CPU consumption is very small, except for the Nova Conductor service that interfaces all Nova services with the MariaDB database. This information is valuable for the OpenStack community as it clearly shows that there is room for improvement to reduce the consumption of the `nova-conductor` service (for 1,000 nodes, the current code requires the equivalent of 7 cores while the compute nodes are idle). For the RAM consumption, an important increase is observed for RabbitMQ, another supporting service that is heavily used for communications between services

Nb. of compute nodes	100	200	500	1,000
RabbitMQ	1.5	2.93	6.89	13.5
MariaDB	79	85	120	170

TABLE III
Maximum number of simultaneous open connections for OpenStack services with varying the number of compute nodes (thousands).

Nb. of compute nodes	100	200	500	1,000
SELECT	53	102	242	474
UPDATE	15	31	76	151

TABLE IV
Average number of SQL queries per second on MariaDB during the idle period

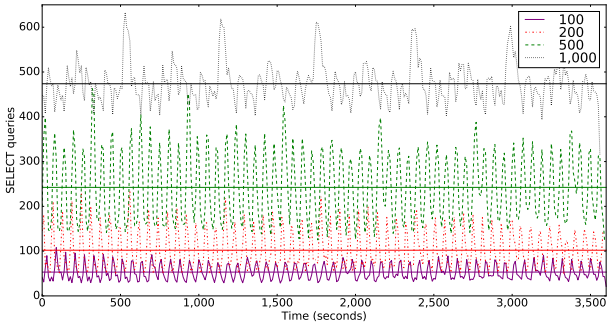


Fig. 6. Number of SQL queries per second executed by MariaDB with varying the number of compute nodes. Horizontal lines show the average for each series.

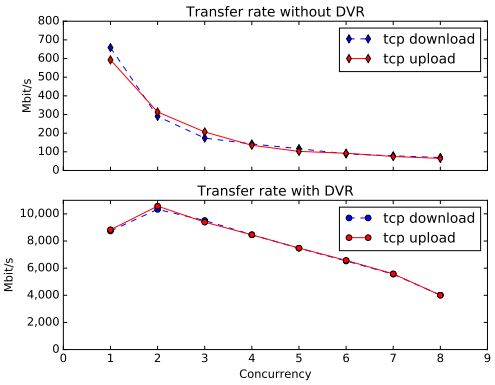


Fig. 7. Effect of DVR on the transfer rate observed by instances

like `nova-conductor` and MariaDB. Table III presents the maximum number of connections for RabbitMQ and MariaDB. It clearly shows that the increased RAM usage is linked to network usage: the number of open connections on the RabbitMQ container grows indeed at the same rate as memory usage. Moreover, the number of connections opened to RabbitMQ can be explained by the fact that, even in idle state, OpenStack is maintaining several permanent connections with each Nova and Neutron agents. This leads to the conclusion that RabbitMQ will be hard to scale beyond this limit without reviewing the communication patterns in use. To further explain this increase in resource usage, we export from ENOS the number of database queries performed each second by MariaDB. The average of SELECT and UPDATE queries are presented in Table IV, while the number of SELECT queries performed each second for the one-hour period is plotted on Figure 6. From the table, we observe that the average number of queries increases linearly with the number of nodes. More importantly, from the figure, we observe periodic spikes. These spikes are due to periodic tasks run by Nova services and Neutron agents. They are indeed reporting periodically their states in the database. UPDATE queries follow the same pattern but aren't plotted here. Note that the reporting interval is configurable and may be decreased in the configuration file at the cost of decreasing the consistency of the state stored in the database.

This evaluation demonstrates how OpenStack can be studied with ENOS as a black-box and how complex mechanisms involving multiple services can be explored.

B. Data-plane performance study with shaker

In this section, we illustrate ENOS's ability to conduct data-plane evaluations. This evaluation could have taken place some time ago when a new feature called Distributed Virtual Routing (DVR)[4] was introduced in Neutron. From a high-level perspective, it enables Neutron to distribute routing across a set of hosts instead of being centralized on a specific node. DVR removes a single point of failure and alleviates the load on the network. From the developer point of view, such an experiment is crucial as it validates intricate implementation choices that have a direct impact on application performance.

We deploy OpenStack using 3 physical machines hosting respectively the Control, Network and Compute group. For this experiment, we use the Shaker benchmark named *L3 East-West Dense*. This benchmark spawns pairs of instances on the same physical host. Paired instances are put in different tenant networks, forcing network traffic to be routed by the infrastructure. OpenStack was deployed using ENOS on Grid'5000 alternatively with and without DVR respectively. The latter uses a single `neutron-l3-agent` hosted on a dedicated node (the network node) whereas the former takes advantage of the distribution of `neutron-l3-agent` on all the compute nodes. In both cases, we are interested in the data transfer rate between instances in the same pair while increasing the number of simultaneous active pairs (the concurrency).

Figure 7 depicts the transfer rate observed by individual instances, while the concurrency increases. Inter networks traffic clearly shows better performance when enabling DVR. Figure 8 reports the network traffic observed by the network node. On the one hand, when DVR is disabled, we observe that the network node receives all inter-tenant data transfers since it acts as a gateway for all the tenant-networks. Note that for each concurrency value, three series of data transfer measurements were made, explaining the spikes seen on the graph. On the other hand, when DVR is enabled, no more traffic is seen on this host.

We underline that ENOS greatly eases this kind of study since it collects data from multiple sources. Here data from the application benchmark and general metrics on the network node were automatically gathered.

[4]https://blueprints.launchpad.net/neutron/+spec/neutron-ovs-dvr

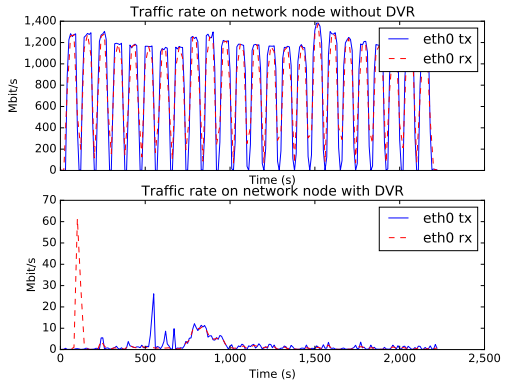


Fig. 8. Effect of DVR on the network traffic observed on the network node

V. Related Work

The evaluation of OpenStack can be achieved either by the control or the data plane side.

As previously highlighted, several control plane evaluations have been performed [**?**], [**?**]. However, they have been investigated using ad-hoc frameworks that prevent researchers to reproduce them. For instance, the authors of [**?**] reviewed the behavior of the Nova Scheduler using specific and deprecated tools [**?**] on Nova Computes.

Data plane evaluations have suffered from the same problem. For instance, different OpenStack network components were compared in [**?**] using deprecated tools named Rude and Crude [**?**]. Additionally, an ad-hoc framework based on perl and bash scripts has been proposed for network evaluation [**?**].

Many studies have investigated the challenges of evaluating complex infrastructures such as distributed systems [**?**] and IaaS clouds [**?**]. Four challenges can be extracted from these studies: the ease of experimenting, the replicability (replay an experiment in the same conditions), the reproducibility (experiments can be launched on different infrastructures), the control of the parameter space and the experiment scalability. By embracing the *Experimentation-as-Code* vision and by choosing a pluggable design, ENOS should be able to offer a sustainable method for evaluating OpenStack by tackling these four challenges. Although the current code base only integrate two benchmark suites, namely Rally [**?**] and Shaker [**?**], attractive tools such as PerfKit [**?**] and CloudBench [**?**] can be easily invoked to provide a large panel of synthetic and real-world workloads.

Among the different actions we know, Browbeat [**?**] is the only OpenStack project whose goals closely match those of ENOS. It provides indeed a set of Ansible playbooks to run workloads on OpenStack and to analyze the result metrics. The workloads are generated by Rally, Shaker or PerfKit Benchmarker [**?**] and the metric visualization is done by services such as collectd, grafana or ELK [**?**]. However, compared to ENOS, Browbeat requires that the operator sets a functional OpenStack with TripleO [**?**] (*i.e.*, OpenStack On OpenStack). TripleO is an OpenStack project to deploy two clouds. The first one is a deployment cloud (named *undercloud*), and is used to set up tunable workload *overclouds* on which Browbeat runs its benchmarks. This deployment phase adds a significant difficulty for researchers who desire to evaluate OpenStack releases. Moreover, it constraints the researcher to evaluate OpenStack on top of an OpenStack cloud whereas ENOS is testbed agnostic.

VI. Conclusion

With a community that gathers more than 5,000 people twice a year at the single location, the OpenStack software suite has become the de facto open-source solution to operate, supervise and use Cloud Computing infrastructures. While it has been mainly supported by key companies such as IBM, RedHat and more recently Google, we claim that distributed computing scientists should now join the effort to help the OpenStack consortium address the numerous technical and scientific challenges related to its scalability and reliability. Similarly to what our scientific community has been doing for Linux, the OpenStack software suite should benefit from scientific guidance. However, diving into OpenStack and understanding its intricate internal mechanisms is a tedious and sometimes too expensive task for researchers.

To allow academics, and more generally the OpenStack consortium to identify issues, propose countermeasures, and validate code improvements, we presented in this paper the ENOS framework. Thanks to container technologies and the use of "off-the-shelf" benchmarks, ENOS is the first holistic approach for evaluating OpenStack in a controlled and reproducible way. Two experiments illustrated how ENOS can address control-plane and data-plane evaluations. The first one focused on analyzing how an idle OpenStack behaves at different scales. This experiment helps in identifying services which will become bottlenecks (*e.g.*, RabbitMQ, `nova-conductor`) with a large number of compute nodes. The second evaluation shows how ENOS can be used to validate the performance of a specific feature, namely Distributed Virtual Routing. ENOS proved that enabling this new feature significantly improved the performance of inter-tenants network communications.

But the value of ENOS can be even bigger when integrated with a Continuous Integration system; in this case, ENOS can automatically execute performance evaluations of individual code contributions in order to prevent code that has a negative impact on performance to be merged into the upstream repository.

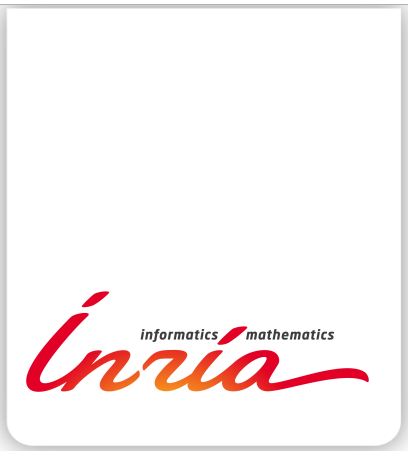
We emphasize this work enabled us to exchange with the OpenStack Foundation and take part in different discussions/working groups. As an example, we are using ENOS to conduct several experiments in the context of the OpenStack performance working group [**?**]. First, we are evaluating different message bus solutions that can replace the current RabbitMQ solution that does not scale well. Second, we are performing several experiments to identify network requirements in the case of WANwide infrastructures. Conducting these experiments is a first interest for telcos that target the deployment of Fog and Edge Computing infrastructures.

As mid-term actions, we plan to extend ENOS with the new OpenStack Profiler tool. This will help researchers investigating in details performance issues by analyzing the execution traces of any OpenStack functionality.

Finally, we would like to highlight that OpenStack is only one example of such large software projects that can benefit from the involvement of our scientific community. The approach pursued by ENOS can be easily extended to other complex software stacks. The only requirement is to get containerized versions of said software. This trend is expected to grow, looking for instance at the Docker Hub repository [**?**].

ACKNOWLEDGMENTS

Most of the materials presented in this article are available on the Discovery initiative website. Supported by the Inria Project Lab program, Discovery is an Open-Science Initiative aiming at implementing a fully decentralized IaaS manager: http://beyondtheclouds.github.io. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-0803