Cloud Test Suite - CERN IT Documentation

CERN IT-DI-EFP

Feb 10, 2020

Cloud Test Suite Documentation

THE SUITE HAS BEEN MOVED OVER TO . THIS DOCUMENTATION IS NO LONGER UPDATED

This tool is intended to be used to test and validate commercial cloud services across the stack for research and education environments. This Test-Suite is being used as a validation tool for commercial cloud services procurement in OCRE (Open Clouds for Research Environments) project sponsored by the European Commission.

More information at: http://ocre-project.eu.

Please find the the repository here: https://github.com/cern-it-efp/OCRE-Testsuite

CHAPTER 1

1. Getting started

Please follow the steps below in order to deploy tests in a cloud provider:

Refer to section "Using Docker" to use the Docker image we provide to avoid dealing with required packages and dependencies.

1.1 1.1 Dependencies

This test-suite requires some packages to work properly and these must be installed by yourself directly. Please see below.

1.1.1 Terraform

Terraform is the tool that creates the VMs that will later become a Kubernetes cluster. The test-suite makes use of it so download and install on your machine. In some cases, providers are not fully supported by Terraform, however they might provide plugins to bridge this gap. In such cases, please refer to the documentation of the provider to download the plugin. Once downloaded, this must be placed at *~/.terraform.d/plugins* and execution permissions must be given to it ($+x$).

1.1.2 Kubernetes client

In order to manage the Kubernetes cluster locally instead of using the master node, install on your machine.

1.1.3 Python

Python version 3 is required. The following python packages are required:

- pyyaml
- jsonschema

- kubernetes
- requests

Please install them with pip3.

1.2 1.2 SSH key

A ssh key pair is needed to establish connections to the VMs to be created later. Therefore, you must create (or import) this key on your provider beforehand and place the private key at *~/.ssh/id_rsa*. Note errors may occur if your key doesn't have the right permissions. Set these to the right value using the following command:

```
$ chmod 600 path/to/key
```

1.3 1.3 Security groups

The following ports have to be opened:

Port	Protocol	Functionality
22	TCP	SSH
6443	TCP	Kubernetes API
10250	TCP	API which allows node access
8472	UDP	Flannel overlay network, k8s pods communication

1.4 1.4 Networking and IPs

Some providers do not allocate public IPs to the VMs but use NAT. Hence the VM can be reached from outside but that IP is not really residing on the VM. This causes conflicts when creating the Kubernetes cluster. If one wants to run the Test-Suite on a provider of this case, then the suite must be launched from within the network the nodes will be connected to, this is a private network. In other words, **a VM will have to be created first manually** and the Test Suite will have to be triggered from there.

1.5 1.5 Download and preparation

1.5.1 Cloning repository

Please clone the repository as follows and cd into it:

```
$ git clone https://github.com/cern-it-efp/OCRE-Testsuite.git
$ cd OCRE-Testsuite
```

1.5.2 Configuration

While completing this task, please refer to in order to complete it successfully as some parts are provider specific and differ from one provider to another.

You will find in the root of the cloned repository a folder named *configurations*. That folder must containing the following files:

`testsCatalog.yaml (required)`

Refer to the section "Test Catalog" to learn how to fill this file.

`configs.yaml (required)`

[**NOTE**: For running on Azure, AWS, GCP, OpenStack, CloudStack and Exoscale refer to the section "Main clouds" below. In those cases, only configs.yaml and testsCatalog.yaml are needed.]

Its variables:

Name	Explanation / Values
providerName	Name of the provider for Terraform. (required)
providerInstanceName	Compute instance name for Terraform. This is provider specific. (required)
pathToKey	Path to the location of your private key (required)
flavor	Flavor to be used for the main cluster. This has to be specified as a key-value pair according to the provider. (required)
openUser	User to be used in case the provider doesn't allow root ssh. If not specified, root will be used for ssh connections.
dockerCE	Version of docker-ce to be installed. Leave empty for latest.
dockerEngine	Version of docker-engine to be installed. Leave empty for latest.
kubernetes	Version of Kubernetes to be installed. Leave empty for latest.

Note that it's possible to choose between "Docker Community Edition" and "Docker Engine" (older Docker packages). However it's **highly recommended** to leave these variables empty to create a cluster with the latest stack.

The file also contains a section named *costCalculation*. Refer to the section "Cost of run calculation" to understand how to fill that part.

`credentials`

This file must contains .tf (HCL) code for authentication that goes on the provider definition section of a Terraform configuration file (i.e AWS) In case this file is empty, the TS assumes an external authentication method: like env variables (i.e OpenStack) or CLI (i.e Azure). Note that if you aim to use external authentication but you need something inside the provider section of the Terraform configuration file (i.e AWS region), this file is the place to define that.

`instanceDefinition (required)`

In this file one should write all the key-pair values that would be written on the body of an instance definition resource on Terraform, according to the cloud one wants to test. Please refer to the documentation of the cloud provider to

check which pairs you need to specify. In any case, you can run the Test-Suite (next steps) and if there is any missing pair a message will be shown in the terminal telling you which ones these are. This is how you must specify each pair:

```
<YOUR_PROVIDER'S_STRING_FOR_A_KEY> = "<VALUE_GIVEN_FOR_THAT_KEY>"
```

An example:

```
display_name = "NAME_PH"
template = "Linux CentOS 7.5 64-bit"
key_pair = "k_cl"
security_groups = ["kgroup"]
disk_size = 50
zone = "ch-gva-2"
```

One of the properties specified on the block that defines a compute node (VM) is the flavor or machine type. This property must not be specified on instanceDefinition but on configs.yaml's flavor.

Please pay attention in this section to the name for the instance, which will be set by the Test-Suite containing:

- The string "kubenode"
- A string indicating the purpose of the cluster to which the VM belongs
- A random, 4 character string to avoid DNS issues
- An integer. 0 would be the master node, 1+ would be the slaves

To achieve this, your instance definition must contain the 'NAME_PH' placeholder. When specifying the name for the instance, please follow this structure:

```
<YOUR_PROVIDER'S_STRING_FOR_NAME> = "NAME_PH"
```

Now, let's assume your provider's string for the instance name is "display_name", then you should write:

```
display_name = "NAME_PH"
```

As an example let's assume the suite comes up with the name "kubenode-hpcTest-aws-0", Then it would switch that name with the NAME_PH placeholder:

```
display_name = "kubenode-hpcTest-aws-0"
```

[**NOTE 1**: This will be taken as a whole block and placed directly on a .tf file]
[**NOTE 2**: Clouds that don't support resource creation with Terraform or k8saaS can't currently be tested with this Test-Suite]

`Dependencies`

This file takes also HCL code. There are providers for which dependencies are required, for example Azure: Terraform can't create a VM if there is no NIC for it. Then this is the file to define those dependencies needed by the VMs.

1.5.3 Main clouds: additional support

Writing Terraform files is not needed when running the suite on Azure, AWS, GCP, OpenStack, CloudStack and Exoscale. In those cases the suite will create itself the Terraform files on the fly according to the configuration provided. Find below these lines details on how to run the suite on these providers:

`Azure`

(Find the example files at *examples/azure*. It is also possible to use AKS to provision the cluster, for this refer to section "Using existing clusters".)

Install az CLI and configure credentials with 'az login'.

Variables for configs.yaml:

Name	Explanation / Values
providerName	It's value must be "azurerm". (required)
pathToKey	Path to the location of your private key (required)
flavor	Flavor to be used for the main cluster.
openUser	User to be used for ssh connections.
dockerCE	Version of docker-ce to be installed. Leave empty for latest.
dockerEngine	Version of docker-engine to be installed. Leave empty for latest.
kubernetes	Version of Kubernetes to be installed. Leave empty for latest.
location	The region in which to create the compute instances. (required)
subscriptionId	ID of the subscription. (required)
resourceGroup-Name	Specifies the name of the Resource Group in which the Virtual Machine should exist. (required)
pubSSH	Public SSH key of the key specified at configs.yaml's pathToKey. (required)
securityGroupID	The ID of the Network Security Group to associate with the VMs's network interfaces (required)
subnetId	Reference to a subnet in which the NIC for the VM has been created. (required)
image.publisher	Specifies the publisher of the image used to create the virtual machines. (required)
image.offer	Specifies the offer of the image used to create the virtual machines. (required)
image.sku	Specifies the SKU of the image used to create the virtual machines. (required)
image.version	Specifies the version of the image used to create the virtual machines. (required)

Note: the security group and subnet -virtual network too- have to be created beforehand and their ID's used at configs.yaml. Also, if image's *publisher*, *offer*, *sku* and *version* are omitted, the following defaults will be used:

- publisher = OpenLogic
- offer = CentOS
- sku = 7.5
- version = latest

`AWS`

(Find the example files at *examples/aws*. It is also possible to use EKS to provision the cluster, for this refer to section "Using existing clusters".)

Variables for configs.yaml:

Name	Explanation / Values
providerName	It's value must be "aws". (required)
pathToKey	Path to the location of your private key (required)
flavor	Flavor to be used for the main cluster. (required)
openUser	User to be used for ssh connections. (required)
dockerCE	Version of docker-ce to be installed. Leave empty for latest.
dockerEngine	Version of docker-engine to be installed. Leave empty for latest.
kubernetes	Version of Kubernetes to be installed. Leave empty for latest.
region	The region in which to create the compute instances. (required)
sharedCredentialsFile	The authentication method supported is AWS shared credential file. Specify here the absolute path to such file. (required)
ami	AMI for the instances. (required)
keyName	Name of the key for the instances. (required)

`GCP`

(Example files at *examples/gcp*. It is also possible to use GKE to provision the cluster, for this refer to section "Using existing clusters". You will have to too.)

Variables for configs.yaml:

Name	Explanation / Values
providerName	It's value must be "google". (required)
pathToKey	Path to the location of your private key (required)
flavor	Flavor to be used for the main cluster. (required)
openUser	User to be used for ssh connections. (required)
dockerCE	Version of docker-ce to be installed. Leave empty for latest.
dockerEngine	Version of docker-engine to be installed. Leave empty for latest.
kubernetes	Version of Kubernetes to be installed. Leave empty for latest.
zone	The zone in which to create the compute instances. (required)
pathToCredentials	Path to the GCP JSON credentials file (note this file has to be downloaded in advance from the GCP console). (required)
image	Image for the instances. (required)
project	Google project under which the infrastructure has to be provisioned. (required)
gpuType	Type of GPU to be used. Needed if the Deep Learning test was selected at testsCatalog.yaml.

`OpenStack`

Regarding authentication, download the OpenStack RC File containing the credentials from the Horizon dashboard and source it.

Variables for configs.yaml:

Name	Explanation / Values
providerName	It's value must be "openstack". (required)
pathToKey	Path to the location of your private key (required)
flavor	Flavor to be used for the main cluster. (required)
openUser	User to be used for ssh connections. Root user will be used by default.
dockerCE	Version of docker-ce to be installed. Leave empty for latest.
dockerEngine	Version of docker-engine to be installed. Leave empty for latest.
kubernetes	Version of Kubernetes to be installed. Leave empty for latest.
imageName	OS Image to be used for the VMs. (required)
keyPair	Name of the key to be used. Has to be created or imported beforehand. (required)
securityGroups	Security groups array. Must be a String, example: "["default","allow_ping_ssh_rdp"]"
region	The region in which to create the compute instances. If omitted, the region specified in the credentials file is used.
availabilityZone	The availability zone in which to create the compute instances.

`CloudStack`

Variables for configs.yaml:

Name	Explanation / Values
providerName	It's value must be "cloudstack". (required)
pathToKey	Path to the location of your private key (required)
flavor	Flavor to be used for the main cluster. (required)
openUser	User to be used for ssh connections. Root user will be used by default.
dockerCE	Version of docker-ce to be installed. Leave empty for latest.
dockerEngine	Version of docker-engine to be installed. Leave empty for latest.
kubernetes	Version of Kubernetes to be installed. Leave empty for latest.
keyPair	Name of the key to be used. Has to be created or imported beforehand. (required)
securityGroups	Security groups array. Must be a String, example: "["default","allow_ping_ssh_rdp"]"
zone	The zone in which to create the compute instances. (required)
template	OS Image to be used for the VMs. (required)
diskSize	VM's disk size.
configPath	Path to the file containing the CloudStack credentials. See below the structure of such file. (required)

CloudStack credentials file's structure:

```
[cloudstack]
url = your_api_url
apikey = your_api_key
secretkey = your_secret_key
```

`Exoscale`

Variables for configs.yaml:

Name	Explanation / Values
providerName	It's value must be "exoscale". (required)
pathToKey	Path to the location of your private key (required)
flavor	Flavor to be used for the main cluster. (required)
dockerCE	Version of docker-ce to be installed. Leave empty for latest.
dockerEngine	Version of docker-engine to be installed. Leave empty for latest.
kubernetes	Version of Kubernetes to be installed. Leave empty for latest.
keyPair	Name of the key to be used. Has to be created or imported beforehand. (required)
security-Groups	Security groups array. Must be a String, example: "["default","allow_ping_ssh_rdp"]"
zone	The zone in which to create the compute instances. (required)
template	OS Image to be used for the VMs. (required)
diskSize	VM's disk size. (required)
configPath	Path to the file containing the Exoscale credentials. See below the structure of such file. (required)

Exoscale credentials file's structure:

```
[exoscale]
key = EXOe3ca3e7621b7cd7a20f7e0de
secret = 2_JvzFcZQL_Rg1nZSRNVheYQh9oYlL5aX3zX-eILiL4
```

1.6 1.6 Using Docker

A Docker image has been built and pushed to Docker hub. This image allows you to skip section "1.1 Dependencies" and jump to "1.2 SSH key".

Run the container (pulls the image first):

```
$ docker run --net=host -it cernefp/tslauncher
```

Note the option '–net=host'. Without it, the container wouldn't be able to connect to the nodes, as it would not be in the same network as them and it is likely the nodes will not have public IPs. With that option, the container will use the network used by its host, which will be sharing the network with the nodes.

You will get a session on the container, directly inside the cloned repository.

CHAPTER 2

2. Tests Catalog

In the root of the cloned repository, you will find a file named *testsCatalog.yaml*, in which you have to specify the tests you want to run. To run a certain test simply set its *run* variable to the True Boolean value. On the other hand, if you don't want it to be run set this value to False. Please find below, a description of each test that has already been integrated in the Test-Suite:

2.1 Deep Learning using GPUs

(This test is currently under development and will be available if following versions)

The 3DGAN application is a prototype developed to investigate the possibility to use a Deep Learning approach to speed-up the simulation of particle physics detectors. The benchmark measures the total time needed to train a 3D convolutional Generative Adversarial Network (GAN) using a data-parallel approach on distributed systems. It is based on MPI for communication. As such, it tests the performance of single nodes (GPUs cards) but also latency and bandwidth of nodes interconnects and data access. The training uses a Kubernetes cluster (GPU flavored) with Kubeflow and MPI.

If selected, the suite will provision a Kubernetes cluster -GPU flavored- specifically for this test. For this test, apart from the *run* variable, the following can be set in the *testsCatalog.yaml* file:

Name	Explanation / Values
nodes	Number of nodes to be used for the deployment. Default: max number of nodes available.
flavor	Terraform definition of the flavor to be used for this test's cluster. (required)

- Contributors/Owners: Sofia Vallecorsa (CERN) - sofia.vallecorsa AT cern.ch; Jean-Roch Vlimant (Caltech)
-

2.2 S3 endpoint tests

A simple S3 test script to test functionality of S3-like endpoints, checking the following: S3 authentication (access key + secret key), PUT, GET, GET with prefix matching, GET chunk, GET multiple chunks.

For this test, apart from the *run* variable, the following ones must be set on the *testsCatalog.yaml* file:

Name	Explanation / Values
endpoint	Endpoint under which your S3 bucket is reachable. This URL must not include the bucket name but only the host.
accessKey	Access key for S3 resource management.
secretKey	Secret key for S3 resource management.

- Contributors/Owners: Oliver Keeble (CERN) - oliver.keeble AT cern.ch
-

2.3 Data Export: From the commercial cloud provider to Zenodo

When using cloud credits, when the credit is exhausted, data can be repatriated or moved to a long-term data storage service. The example used in this test uses service maintained by CERN, verifying that the output data can be taken from the cloud provider to Zenodo.

- Contributors/Owners: Ignacio Peluaga (CERN) - ignacio.peluaga.lozada AT cern.ch
-

2.4 Containerised CPU Benchmarking

Suite containing several CPU benchmarks used for High Energy Physics (HEP). The following benchmarks are run on the cloud provider, using a containerised approach:

- DIRAC Benchmark
- ATLAS Kit Validation
- Whetstone: from the UnixBench benchmark suite.
- Hyper-benchmark: A pre-defined sequence of measurements and fast benchmarks.
- Contributors/Owners: Domenico Giordano (CERN) - domenico.giordano AT cern.ch
-

2.5 Networking performance measurements

perfSONAR is a network measurement toolkit designed to provide federated coverage of paths, and help to establish end-to-end usage expectations.

In this test, a perfSONAR testpoint is created using a containerised approach on the cloud provider infrastructure. The following tests are launched end to end:

- throughput: A test to measure the observed speed of a data transfer and associated statistics between two endpoints.
- rtt: Measure the round trip time and related statistics between hosts.
- trace: Trace the path between IP hosts.
- latencybg: Continuously measure one-way latency and associated statistics between hosts and report back results periodically.

The endpoint for these tests must be specified at testsCatalog.yaml's *perfsonarTest.endpoint* variable. Use endpoints from:

-
-
-
- Contributors/Owners: Shawn Mckee (University of Michigan) - smckee AT umich.edu; Marian Babik CERN) - marian.babik AT cern.ch
-

2.6 FDMNES: Simulation of X-ray spectroscopies

(This test is currently under development and will be available if following versions)

The FDMNES project provides the research the community a user friendly code to simulate x-ray spectroscopies, linked to the real absorption (XANES, XMCD) or resonant scattering (RXD in bulk or SRXRD for surfaces) of synchrotron radiation. It uses parallel calculations using OpenMPI. As an HPC test FDMNES is rather heavy on CPU and Memory and light on I/O. The objective of this test is to understand which configuration of FDMNES is the most efficient and which type of tasks and calculations can be done in a give cloud provider.

If selected, the suite will provision a Kubernetes cluster -HPC flavored- specifically for this test. For this test, apart from the *run* variable, the following can be set in the *testsCatalog.yaml* file:

Name	Explanation / Values
nodes	Number of nodes to be used for the deployment.
flavor	Terraform definition of the flavor to be used for this test's cluster. (required)

- Contributors/Owners: Rainer Wilcke (ESRF) - wilcke AT esrf.fr
-

2.7 DODAS: Dynamic On Demand Analysis Services test

DODAS is a system designed to provide a high level of automation in terms of provisioning, creating, managing and accessing a pool of heterogeneous computing and storage resources, by generating clusters on demand for the execution of HTCondor workload management system. DODAS allows to seamlessly join the HTCondor Global Pool of CMS to enable the dynamic extension of existing computing resources. A benefit of such an architecture is that it provides high scaling capabilities and self-healing support that results in a drastic reduction of time and cost, through setup and operational efficiency increases.

If one wants to deploy this test, the machines in the general cluster (to which such test is deployed), should have at least 50 gb disk as the image for this test is 16GB.

- Contributors/Owners: Daniele Spiga (INFN) - daniele.spiga@pg.infn.it ; Diego Ciangottini (INFN) - diego.ciangottini@cern.ch
-

CHAPTER 3

3. Run the test-suite

Once the configuration steps are completed, the Test-Suite is ready to be run:

```
$ ./test_suite <options>
```

3.1 Options

The following table describes all the available options:

Name	Explanation / Values
–only-test	Run without creating the infrastructure (VMs and cluster), only deploy tests. Not valid for the first run.
–retry	In case of errors on the first run, use this option for retrying. This will make the test-suite try and reuse already provisioned infrastructure. Not valid for the first run, use only when VMs were provisioned but kubernetes bootstrapping failed.

3.2 Other commands

Once the test suite is running, you can view the Terraform logs by doing:

```
$ tail -f logs
```

Once the provisioning has completed and tests are deployed, you can see the pods statuses by doing:

```
$ watch kubectl get pods
```

If GPU and HPC tests were deployed, see their pods by doing:

```
$ watch kubectl --kubeconfig src/tests/dlTest/config get pods # For GPU cluster
$ watch kubectl --kubeconfig src/tests/hpcTest/config get pods # For HPC cluster
```

CHAPTER 4

4. Using existing clusters

It's possible to use this tool for testing providers that support Kubernetes as a Service. This means the provider offers the user a way for simply creating a cluster. In case one wants to validate a provider that offers this and want to take advantage of it, simply skip steps 1.1 and 1.2 (install Terraform and manage ssh keys) and when running the test-suite, use option *–only-test*.

Note that you must have the file *~/.kube/config* for the previously provisioned cluster on your local machine so that it can be managed from there.

CHAPTER 5

5. Results

Once all the selected tests finish the run, the test-suite has completed its execution. The results and logs of the validation exercise can be seen at /results (JSON format). Prior to completing the runs, a message will be printed to the console showing the exact path to the results. There you will find a file *general.json* containing general information such as IPs of the provisioned VMs, estimated cost and brief test results information and also a directory *detailed* containing more detailed information for each test.

CHAPTER 6

6. Verification

In order to verify results, please run using *–via-backend* so that the proxy at CERN runs the Test-Suite (only deploys tests, no provisioning), harvests results and push them to CERN's S3 bucket. Before starting the run, a message will be shown asking for yes/no answer. It warns the user that backend runs publish results to the CERN bucket.

Note that this feature is still under development and testing and will be available on next releases.

CHAPTER 7

7. Cost of run calculation

An approximative cost of running the test-suite will be calculated in case the prices are specified at configs.yaml under the *costCalculation* section. In this configuration file, one must specify the price per hour for the different resources:

Name	Explanation / Values
generalInstancePrice	Price per hour of VM with the flavor chosen for the general cluster.
GPUInstancePrice	Price per hour of VM with the flavor chosen for the GPU cluster.
HPCInstancePrice	Price per hour of VM with the flavor chosen for the HPC cluster.
s3bucketPrice	S3 bucket price.

If a price value is required for the cost calculation but the *costCalculation* section is not properly filled (For example, S3 Endpoint test was set to True but *s3bucketPrice* was not set), no approximation will be given at all.

At the end of the run, the resulting approximated cost will be added to the file containing general test suit run results. In case this information isn't needed, simply leave the values on the section *costCalculation* empty. Note that this is a cost estimate and not an exact price.

The formula used is as follows:

(Number of VMs created) x (Price of a VM per hour) x (Time in hours the VMs were used for the test-suite run) + (Cost of other resources)

Where "Cost of other resources" are the cost of resources which are not simple compute, like storage. For example in the case of the S3 Endpoint test:

(Price of a S3 bucket per hour) x (Time in hours the bucket was used for the test)

Note that the price per request or data amount (GB) are not considered here as these are not significant since less than 10 requests are done for this test and for very small data sets. Note also that only the cost of the running time of the VM is considered, so if your provider charges for VM creation and not only for the time it is running, the cost obtained will vary to the real one.

The test-suite executes four main steps:

1) Infrastructure provisioning: VMs are created using Terraform and then Kubernetes and Docker are installed on them to create several k8s cluster according to the selected tests.

2) Deploy the tests: Kubernetes resource definition files (YAML) are used to deploy the tests, either as single pods or deployments.

3) Harvest results: at the end of each test run a result file -written in JSON- is created. This file is harvested from the cluster and stored locally.

4) Through a verification system, the Test-Suite can also be triggered from a service running at CERN. In this case, results are then pushed to a S3 Bucket at CERN. (Under development)

The test set described below is based on the tests used in Helix Nebula The Science Cloud PCP project funded by the European Commission.

The developers would like to thank all test owners and contributors to this project.

This test-suite has been tested on:

OS on launcher machine	Ubuntu, CentOS, CoreOS, Debian, RedHat, Fedora
OS running on provider's VMs	CentOS7
Providers / clouds	AWS Google Cloud Microsoft Azure Exoscale (CloudStack) CERN Private Cloud (OpenStack) CloudFerro (OpenStack) Cloudscale (OpenStack) CloudStack OpenStack

The test suite is being tested in several additional cloud providers. As tests are concluded the cloud providers names will be added in the table above.

CHAPTER 8

Release notes

(Note the versions are numbered with the date of the release: YEAR.MONTH)

`19.12`

-Project restructured.

-Improved support for running on Google, AWS, Azure, Exoscale, OpenStack and CloudStack.

`19.8`

-Parallel creation of clusters, with different flavors according to tests needs.

-New logging system to keep parallel running tests logs sorted.

-Restructured configuration: moved configuration files to */configurations* and created new files taking HCL code (terraform configuration code) to keep *configs.yaml* clean.

-Automated allowance of root ssh by copying open user's authorized_keys to root's ~/.ssh as well as *sshd_config* modification.

-Usage of Kubernetes API instead of Kubernetes CLI.

-For network test (perfSONAR), usage of API instead of pscheduler CLI.

-New test: Dynamic On Demand Analysis Service, provided by INFN.

-Added configurations validation with jsonschema.

-Created Docker image to run a Test-Suite launcher container: rapidly creates a ready to use Test-Suite launcher.

`19.4`

-New tests: perfsonar and cpu-benchmarking

`19.2`

-First release.

CHAPTER 9

Contact

For more information contact ignacio.peluaga.lozada AT cern.ch

CHAPTER 10

License