

---

# **Yorc Documentation**

***Release 3.1.2***

**Atos BDS R&D**

**Feb 13, 2019**



<b>1</b>	<b>Install Yorc and requirements</b>	<b>1</b>
1.1	Typical Yorc deployment for OpenStack . . . . .	1
1.2	Host requirements . . . . .	2
1.3	Packages installation . . . . .	2
1.4	Final setup . . . . .	4
<b>2</b>	<b>Bootstrap a full stack installation</b>	<b>5</b>
2.1	Prerequisites . . . . .	5
2.2	Bootstrap process overview . . . . .	6
2.3	Bootstrapping the setup in interactive mode . . . . .	7
2.4	Bootstrapping the setup using command line options . . . . .	7
2.5	Bootstrapping the setup using environment variables . . . . .	9
2.6	Bootstrapping the setup using a configuration file . . . . .	9
<b>3</b>	<b>Troubleshooting</b>	<b>13</b>
<b>4</b>	<b>Yorc Server Configuration</b>	<b>15</b>
4.1	Globals Command-line options . . . . .	15
4.2	Configuration files . . . . .	17
4.3	Environment variables . . . . .	23
4.4	Infrastructures configuration . . . . .	24
4.5	Builtin infrastructures configuration . . . . .	24
4.6	Vault configuration . . . . .	27
4.7	Builtin Vaults configuration . . . . .	28
<b>5</b>	<b>Yorc Client CLI Configuration</b>	<b>29</b>
5.1	Command-line options . . . . .	29
5.2	Configuration files . . . . .	30
5.3	Environment variables . . . . .	30
<b>6</b>	<b>Starting Yorc</b>	<b>31</b>
6.1	Starting Consul . . . . .	31
6.2	Starting Yorc . . . . .	31
<b>7</b>	<b>Yorc Command Line Interface</b>	<b>33</b>
7.1	General Options . . . . .	33
7.2	CLI Commands related to deployments . . . . .	33

7.3	CLI Commands related to hosts pool . . . . .	37
<b>8</b>	<b>Yorc Supported infrastructures</b>	<b>43</b>
8.1	Hosts Pool . . . . .	43
8.2	Slurm . . . . .	45
8.3	Google Cloud Platform . . . . .	45
8.4	AWS . . . . .	46
8.5	OpenStack . . . . .	46
8.6	Kubernetes . . . . .	47
<b>9</b>	<b>TOSCA support in Yorc</b>	<b>49</b>
9.1	TOSCA Operations . . . . .	49
<b>10</b>	<b>Run Yorc in Secured mode</b>	<b>55</b>
10.1	Generate SSL certificates with SAN . . . . .	55
10.2	Secured Consul cluster Setup . . . . .	56
10.3	Secured Yorc Setup . . . . .	57
10.4	Secured Yorc CLI Setup . . . . .	58
10.5	Setup Alien4Cloud security . . . . .	58
<b>11</b>	<b>Integrate Yorc with a Vault</b>	<b>59</b>
11.1	HashiCorp's Vault integration . . . . .	59
<b>12</b>	<b>Run Yorc in High Availability (HA) mode</b>	<b>61</b>
12.1	High level view of a typical HA installation . . . . .	61
12.2	Yorc HA setup . . . . .	62
<b>13</b>	<b>Run Yorc in a docker container</b>	<b>63</b>
13.1	Image components . . . . .	63
<b>14</b>	<b>Yorc Telemetry</b>	<b>67</b>
14.1	Key metrics . . . . .	68
<b>15</b>	<b>Performance</b>	<b>71</b>
15.1	TOSCA Operations . . . . .	71
<b>16</b>	<b>Upgrades</b>	<b>73</b>
16.1	Upgrading to Yorc 3.1.0 . . . . .	73
<b>17</b>	<b>Known issues</b>	<b>75</b>
17.1	BER for SSH private key is not supported . . . . .	75

---

## Install Yorc and requirements

---

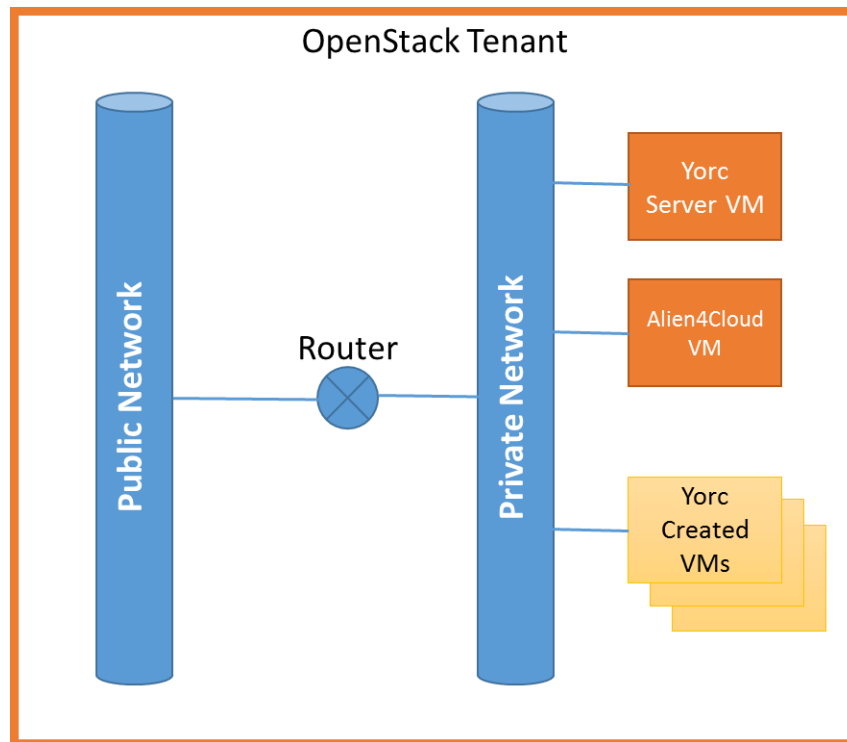
This section describes how to install Yorc manually. Different ways of installing Yorc are described in other sections:

- *Bootstrap a full stack installation* using `yorc bootstrap` CLI. This will allow you deploy Yorc, its dependencies, and the UI Alien4Cloud on a given infrastructure (Google Cloud, AWS, OpenStack or Hosts Pool)
- *Run Yorc in a docker container*.

### 1.1 Typical Yorc deployment for OpenStack

In order to provision softwares on virtual machines that do not necessary have a floating IP we recommend to install Yorc itself on a virtual machine in your OpenStack tenant. Alien4Cloud and the Alien4Cloud Yorc Plugin (see their dedicated documentation to know how to install them) may be collocated on the same VM or resides in a different VM.

Virtual Machines created by Yorc should be connected to the same private network as the Yorc VM (the `-infrastructure_openstack_private_network_name` configuration flag allows to do it automatically). In order to provision Floating IPs, this private network should be connected to the public network of the tenant through a router.



## 1.2 Host requirements

Yorc requires a Linux x86\_64 system to operate with at least 2 CPU and 2 Go of RAM.

## 1.3 Packages installation

Following packages are required to perform the installation:

- python
- python-pip
- zip/unzip
- openssh-client
- wget

Now you can proceed with the installation of softwares used by Yorc.

```
sudo pip install ansible==2.7.2
wget https://releases.hashicorp.com/consul/1.2.3/consul_1.2.3_linux_amd64.zip
sudo unzip consul_1.2.3_linux_amd64.zip -d /usr/local/bin
wget https://releases.hashicorp.com/terraform/0.11.8/terraform_0.11.8_linux_
→amd64.zip
sudo unzip terraform_0.11.8_linux_amd64.zip -d /usr/local/bin
```

As Terraform uses plugins for required providers, you can pre-installed them in a directory you specify with the Yorc Server configuration option (`-terraform_plugins_dir`). Here are the command lines for installing all

providers in `/var/terraform/plugins`. See <https://www.terraform.io/guides/running-terraform-in-automation.html#pre-installed-plugins> for more information.

```
sudo mkdir -p /var/terraform/plugins
```

```
wget https://releases.hashicorp.com/terraform-provider-consul/2.1.0/terraform-
  ↪provider-consul_2.1.0_linux_amd64.zip
sudo unzip terraform-provider-consul_2.1.0_linux_amd64.zip -d /var/terraform/
  ↪plugins
```

```
wget https://releases.hashicorp.com/terraform-provider-null/1.0.0/terraform-
  ↪provider-null_1.0.0_linux_amd64.zip
sudo unzip terraform-provider-null_1.0.0_linux_amd64.zip -d /var/terraform/
  ↪plugins
```

```
wget https://releases.hashicorp.com/terraform-provider-aws/1.36.0/terraform-
  ↪provider-aws_1.36.0_linux_amd64.zip
sudo unzip terraform-provider-aws_1.36.0_linux_amd64.zip -d /var/terraform/
  ↪plugins
```

```
wget https://releases.hashicorp.com/terraform-provider-google/1.18.0/
  ↪terraform-provider-google_1.18.0_linux_amd64.zip
sudo unzip terraform-provider-google_1.18.0_linux_amd64.zip -d /var/terraform/
  ↪plugins
```

```
wget https://releases.hashicorp.com/terraform-provider-openstack/1.9.0/
  ↪terraform-provider-openstack_1.9.0_linux_amd64.zip
sudo unzip terraform-provider-openstack_1.9.0_linux_amd64.zip -d /var/
  ↪terraform/plugins
```

```
sudo chmod 775 /var/terraform/plugins/*
```

Finally you can install the Yorc binary into `/usr/local/bin`.

To support *Orchestrator-hosted operations* sandboxed into Docker containers the following softwares should also be installed.

```
# for apt based distributions
sudo apt install Docker
# for yum based distributions
sudo yum install Docker
# Docker should be running and configured to works with http proxies if any
sudo systemctl enable docker
sudo systemctl start docker

sudo pip install docker-py
```

For a complete Ansible experience please install the following python libs:

```
# To support json_query filter for jinja2
sudo pip install jmespath
# To works easily with CIDRs
sudo pip install netaddr
```

To support Ansible SSH password authentication instead of common ssh keys, the sshpass helper program needs to be installed too.

```
# for apt based distributions
sudo apt install sshpass
# for yum based distributions
sudo yum install sshpass
```

## 1.4 Final setup

In order to provision softwares through ssh, you need to store the ssh private key that will be used to connect to the nodes under `$HOME/.ssh/yorc.pem` where `$HOME` is the home directory of the user running Yorc. This key should part of the authorized keys on remote hosts. Generally, for OpenStack, it corresponds to the private key of the keypair used to create the instance.

---

**Note:** A common issue is to create a key file that does not comply the ssh requirements for private keys (should be readable by the user but not accessible by group/others read/write/execute).

---



---

## Bootstrap a full stack installation

---

The command `yorc bootstrap` can be used to bootstrap the full stack, from Alien4Cloud to Yorc and its dependencies, over different types of infrastructures, on a single node or distributed on several nodes.

### 2.1 Prerequisites

#### 2.1.1 Hosts

The local host from where the command `yorc bootstrap` will be run, as well as remote hosts where the full stack will be deployed, should be Linux x86\_64 systems operating with at least 2 CPUs and 4 Go of RAM. The bootstrap was validated on CentOS 7 and Red Hat Enterprise Linux 7.5.

#### 2.1.2 Packages

The bootstrap operation will install a basic Yorc setup on the local host.

It requires the following packages to be installed on the local host:

- python
- python-pip
- zip/unzip
- openssh-client
- wget

This basic installation on the local host will attempt to install without sudo privileges python ansible module 2.7.2 if needed as well as python packages MarkupSafe, jinja2, PyYAML, six, cryptography, setuptools. So if this ansible module or python packages are not yet installed on the local host, you could either add them yourself, with sudo privileges, running for example:

```
sudo pip install ansible==2.7.2
```

Or you could create a python virtual environment, and let the Yorc bootstrap command install the ansible module within this virtual environment (operation which doesn't require sudo privileges).

You can run these commands to create a virtual environment, here a virtual environment called `yorcenv` :

```
sudo pip install virtualenv
virtualenv yorcenv
source yorcenv/bin/activate
```

You are now ready to download Yorc binary, running:

```
wget https://github.com/ystia/yorc/releases/download/v3.1.2/yorc-3.1.2.tgz
tar xzf yorc-3.1.2.tgz
./yorc bootstrap --help
```

## 2.1.3 Define firewall rules on your Cloud Provider infrastructure

To access Alien4Cloud UI from the local host, you may need to define firewall rules before attempting to bootstrap the full stack on a cloud provider infrastructure.

For example on Google Cloud, you could define this firewall rule for the port 8088 used by the UI, and associate it to a tag (here `a4c`) :

```
$ gcloud compute firewall-rules create a4c-rule \
--allow tcp:8088 --target-tags a4c
```

You could then specify this tag `a4c` in the compute instance to create by the bootstrap deployment, as it is done in [Google Configuration file example](#). This way the created compute instance where Alien4Cloud will be deployed will have its port 8088 open.

## 2.2 Bootstrap process overview

The command `yorc bootstrap` will configure on the local host a basic setup with Yorc and a Consul data store.

This basic setup will then be used to bootstrap the full stack Alien4Cloud/Yorc and its dependencies over a selected infrastructure.

You can deploy the full stack either on a single node (by default), or distributed on several nodes as described in [Run Yorc in HA mode](#), using `yorc bootstrap` command line option `--deployment_type HA` described below.

Configuration values can be provided by the user:

- in interactive mode,
- through a configuration file,
- using `yorc bootstrap` command line options
- using environment variables.

You can combine these modes, `yorc bootstrap` will check in any case if required configuration values are missing, and will ask for missing values.

These configuration values will allow you to specify:

- optional Alien4Cloud configuration values
- Yorc configuration values, all optional, except from the path to a ssh private key that will be used by the local orchestrator to connect to the bootstrapped setup

- Infrastructure configuration with required configuration values depending on the infrastructure, as described at [Infrastructures Configuration](#)
- Configuration of compute Nodes to create on demand,
- User used to connect to these compute nodes,
- Configuration of the connection to public network created on demand.

Details of these on-demand resources configuration values are provided in the Alien4Cloud Yorc plugin Documentation at <https://yorc-a4c-plugin.readthedocs.io/en/latest/location.html>. For example, in the *Google Configuration file example*, you can see on-demand compute and address configuration values.

Once configuration settings are provided, `yorc bootstrap` will proceed to the full stack deployment, showing deployment steps progress (by default, but you can see deployment logs instead through the option `--follow logs` described below).

Once the deployment is finished, the orchestrator on the local host is still running, so you can perform commands like `./yorc deployments list`, `./yorc deployments logs -b`, etc... Or perform any deployment troubleshooting if needed.

To undeploy a bootstrapped setup, you can also use the CLI, running `./yorc deployments undeploy <deployment id>`.

To clean the local host setup, run:

```
./yorc bootstrap cleanup
```

This will only clean the local host environment, it won't undeploy the bootstrapped setup installed on remote hosts.

## 2.3 Bootstrapping the setup in interactive mode

You can bootstrap the setup in interactive mode running:

```
./yorc bootstrap [--review]
```

You will have then to select the type of infrastructure (Google Cloud, AWS, OpenStack, Hosts Pool) on which you want to deploy the full stack, then you will be asked to provide configuration values depending on the selected infrastructure.

The command line option `--review` allows to review and update all configuration values before proceeding to the deployment, opening the editor specified in the environment variable `EDITOR` if defined or using `vi` or `vim` if available.

## 2.4 Bootstrapping the setup using command line options

The following `yorc bootstrap` options are available:

- `--alien4cloud_download_url` Alien4Cloud download URL (defaults to the Alien4Cloud version compatible with this Yorc, under <https://fastconnect.org/maven/content/repositories/opensource/alien4cloud/alien4cloud-dist/>)
- `--alien4cloud_password` Alien4Cloud password (default, admin)
- `--alien4cloud_port` Alien4Cloud port (default 8088)
- `--alien4cloud_user` Alien4Cloud user (default, admin)
- `--ansible_extra_package_repository_url` URL of package indexes where to find the ansible package, instead of the default Python Package repository

- `--ansible_version` Ansible version (default 2.7.2)
- `--consul_download_url` Consul download URL (default, Consul version compatible with this Yorc, under <https://releases.hashicorp.com/consul/>)
- `--consul_port` Consul port (default 8500)
- `--credentials_user` User Yorc uses to connect to Compute Nodes
- `--deployment_type` Define deployment type: `single_node` or `HA` (default, `single_node`)
- `--follow` Follow bootstrap deployment steps, logs, or none (default, steps)
- `--infrastructure` Define the type of infrastructure where to deploy Yorc: `google`, `openstack`, `aws`, `host-spool`
- `--jdk_download_url` Java Development Kit download URL (default, JDK downloaded from <https://edelivery.oracle.com/otn-pub/java/jdk/>)
- `--jdk_version` Java Development Kit version (default 1.8.0-131-b11)
- `--resources_zip` Path to bootstrap resources zip file (default, zip bundled within Yorc)
- `--review` Review and update input values before starting the bootstrap
- `--terraform_download_url` Terraform download URL (default, Terraform version compatible with this Yorc, under <https://releases.hashicorp.com/terraform/>)
- `--terraform_plugins_download_urls` Terraform plugins download URLs (default, Terraform plugins compatible with this Yorc, under <https://releases.hashicorp.com/terraform-provider-xxx/>)
- `--values` Path to file containing input values
- `--working_directory` Working directory where to place deployment files (default, `work`)
- `--yorc_data_dir` Bootstrapped Yorc Home directory (default, `/var/yorc`)
- `--yorc_download_url` Yorc download URL (default, current Yorc release under <https://github.com/ystia/yorc/releases/>)
- `--yorc_plugin_download_url` Yorc plugin download URL (default, current Yorc plugin release under <https://github.com/ystia/yorc-a4c-plugin/releases>)
- `--yorc_port` Yorc HTTP REST API port (default 8800)
- `--yorc_private_key_file` Path to ssh private key accessible locally
- `--yorc_workers_number` Number of Yorc workers handling bootstrap deployment tasks (default 30)

In addition, similarly to the configuration of infrastructures in `yorc server` command described at [Infrastructures Configuration](#), you can use options to define infrastructure and on-demand resources configuration values, for example :

- `--infrastructure_openstack_auth_url` allows to define the authentication URL of an OpenStack infrastructure.

The option `--resources_zip` is an advanced usage option allowing you to change the bootstrap deployment description. You need to clone first the Yorc source code repository at <https://github.com/ystia/yorc>, go into to directory `commands`, change deployment description files under `bootstrap/resources/topology`, then zip the content of `bootstrap/resources/` so that this zip will be used to perform the bootstrap deployment.

## 2.5 Bootstrapping the setup using environment variables

Similarly to the configuration of `yorc` server through environment variables described at *Yorc Server Configuration*, the bootstrap configuration can be provided through environment variables following the same naming rules, for example:

- `YORC_ALIEN4CLOUD_PORT` allows to define the Alien4Cloud port
- `YORC_INFRA_OPENSTACK_AUTH_URL` allows to define the authentication URL of an OpenStack infrastructure.

Once these environment variables are defined, you can bootstrap the setup running : .. parsed-literal:

```
./yorc bootstrap [--review]
```

## 2.6 Bootstrapping the setup using a configuration file

You can bootstrap the setup using a configuration file running:

```
./yorc bootstrap --values <path to configuration file> [--review]
```

Similarly to the configuration of `yorc` server through a configuration file, described at *Yorc Server Configuration*, the bootstrap configuration can be provided in a configuration file following the same naming rules for configuration variables, for example :

```
alien4cloud:
  user: admin
  port: 8088
infrastructures:
  openstack:
    auth_url: http://10.1.2.3:5000/v2.0
```

Sections below provide examples of configuration files for each type of infrastructure.

### 2.6.1 Example of a Google Cloud deployment configuration file

```
yorc:
  # Path to private key file on local host
  # used to connect to hosts on the bootstrapped setup
  private_key_file: /home/myuser/.ssh/yorc.pem
infrastructures:
  google:
    # Path on local host to file containing Google service account private keys
    application_credentials: /home/myuser/gcp/myproject-a90a&bf599ef.json
    project: myproject
address:
  region: europe-west1
compute:
  image_project: centos-cloud
  image_family: centos-7
  machine_type: n1-standard-2
  zone: europe-west1-b
  # User and public key to define on created compute instance
```

(continues on next page)

(continued from previous page)

```
metadata: "ssh-keys=user1:ssh-ed25519 AAAABCD/gV/C+b3h3r5K011evEELMD72S4..."
tags: a4c
credentials:
  # User on compute instance created on demand
  user: user1
```

## 2.6.2 Example of an AWS deployment configuration file

```
yorc:
  # Path to private key file on local host
  # used to connect to hosts on the bootstrapped setup
  private_key_file: /home/myuser/.ssh/yorc.pem
infrastructures:
  aws:
    region: us-east-2
    access_key: ABCDEFABCEDEFABCD12DA
    secret_key: aabcdxYxABC/albcdef
address:
  ip_version: 4
compute:
  image_id: ami-18f8df7d
  instance_type: t2.large
  key_name: key-yorc
  security_groups: janus-securityGroup
  delete_volume_on_termination: true
credentials:
  # User on compute instance created on demand
  user: user1
```

## 2.6.3 Example of an OpenStack deployment configuration file

```
yorc:
  # Path to private key file on local host
  # used to connect to hosts on the bootstrapped setup
  private_key_file: /home/myuser/.ssh/yorc.pem
infrastructures:
  openstack:
    auth_url: http://10.1.2.3:5000/v2.0
    default_security_groups:
      - secgroup1
      - secgroup2
    password: mypasswd
    private_network_name: private-test
    region: RegionOne
    tenant_name: mytenant
    user_name: myuser
address:
  floating_network_name: mypublic-net
compute:
  image: "7d9bd308-d9c1-4952-123-95b761672499"
  flavor: 3
  key_pair: yorc
```

(continues on next page)

(continued from previous page)

```
credentials:
  # User on compute instance created on demand
  user: user1
```

## 2.6.4 Example of a Hosts Pool deployment configuration file

```
yorc:
  # Path to private key file on local host
  # used to connect to hosts on the bootstrapped setup
  private_key_file: /home/myuser/.ssh/yorc.pem
compute:
  shareable: "false"
hosts:
- name: host1
  connection:
    user: user1
    host: 10.129.1.10
    port: 22
  labels:
    host.cpu_frequency: 3 GHz
    host.disk_size: 40 GB
    host.mem_size: 4GB
    host.num_cpus: "2"
    os.architecture: x86_64
    os.distribution: centos
    os.type: linux
    os.version: "7.3.1611"
    private_address: "10.0.0.10"
    public_address: "10.129.1.10"
- name: host2
  connection:
    user: user1
    host: 10.129.1.11
    port: 22
  labels:
    environment: dev
    host.cpu_frequency: 3 GHz
    host.disk_size: 40 GB
    host.mem_size: 4GB
    host.num_cpus: "2"
    os.architecture: x86_64
    os.distribution: centos
    os.type: linux
    os.version: "7.3.1611"
    private_address: "10.0.0.11"
    public_address: "10.129.1.11"
```





## CHAPTER 3

---

### Troubleshooting

---

By default, debug logs are disabled. To enable them, you can export the environment variable `YORC_LOG` and set it to 1 or `DEBUG` before starting the bootstrap:

```
export YORC_LOG=1
```

Once the bootstrap deployment has started, the local yorc server logs are available under `<working dir>/yorc.log`, (`<working dir>` default value being the directory `./work`).

To get the bootstrap deployment ID and current status, run :

```
./yorc deployments list
```

To follow deployment logs and see these logs from the beginning, run :

```
./yorc deployments logs <deployment ID> --from-beginning
```

When a deployment has failed, in addition to logs failure in the logs, you can also get of summary of the deployment steps statuses to identify quickly which step failed, running :

```
./yorc deployments info <deployment ID>
```

If a step failed on a transient error that is now addressed, it is possible to run again manually the failed step, and resume the deployment running the following commands.

First from the previous command `./yorc deployments info <deployment ID>` output, you can find the task ID that failed.

You can now run this command to get the exact name of the step that failed :

```
./yorc deployments tasks info --steps <deployment ID> <task ID>
```

Identify the name of the step that failed.

Let's say for the example that it is the step `TerraformRuntime_create` which failed on timeout downloading the Terraform distribution.

You can then go to the directory where you will find the ansible playbook corresponding to this step :

```
cd <working directory>/deployments/<deployment ID>/ansible/<task ID>/TerraformRuntime/  
↪standard.create/
```

And from this directory, run again this step through this command:

```
ansible-playbook -i hosts run.ansible.yml -v
```

If this manual execution was successful, you can mark the corresponding step as fixed in the deployment, running :

```
./yorc deployments tasks fix <deployment ID> <task ID> TerraformRuntime
```

You can now resume the bootstrap deployment running :

```
./yorc deployments tasks resume <deployment ID>
```

---

## Yorc Server Configuration

---

Yorc has various configuration options that could be specified either by command-line flags, configuration file or environment variables.

If an option is specified several times using flags, environment and config file, command-line flag will have the precedence then the environment variable and finally the value defined in the configuration file.

### 4.1 Globals Command-line options

- `--ansible_use_openssh`: Prefer OpenSSH over Paramiko a Python implementation of SSH (the default) to provision remote hosts. OpenSSH have several optimization like reusing connections that should improve performance but may lead to issues on older systems.
- `--ansible_debug`: Prints massive debug information from Ansible especially about connections
- `--ansible_connection_retries`: Number of retries in case of Ansible SSH connection failure.
- `--ansible_cache_facts`: If set to true, caches Ansible facts (values fetched on remote hosts about network/hardware/OS/virtualization configuration) so that these facts are not recomputed each time a new operation is a run for a given deployment (false by default: no caching).
- `--ansible_archive_artifacts`: If set to true, archives operation bash/python scripts locally, copies this archive and unarchives it on remote hosts (requires tar to be installed on remote hosts), to avoid multiple time consuming remote copy operations of individual scripts (false by default: no archive).
- `--ansible_job_monitoring_time_interval`: Default duration for monitoring time interval for jobs handled by Ansible (defaults to 15s).
- `--ansible_keep_generated_recipes`: If set to true, generated Ansible recipes on Yorc server are not deleted. (false by default: generated recipes are deleted).
- `--operation_remote_base_dir`: Specify an alternative working directory for Ansible on provisioned Compute.
- `--config` or `-c`: Specify an alternative configuration file. By default Yorc will look for a file named `config.yorc.json` in `/etc/yorc` directory then if not found in the current directory.

- `--consul_address`: Specify the address (using the format host:port) of Consul. Consul default is used if not provided.
- `--consul_token`: Specify the security token to use with Consul. No security token used by default.
- `--consul_datacenter`: Specify the Consul's datacenter to use. Consul default (dc1) is used by default.
- `--consul_key_file`: Specify the Consul client's key to use when communicating over TLS.
- `--consul_cert_file`: Specify the Consul client's certificate to use when communicating over TLS.
- `--consul_ca_cert`: Specify the CA used to sign Consul certificates.
- `--consul_ca_path`: Specify the path to the CA used to sign Consul certificates
- `--consul_ssl`: If set to true, enable SSL (false by default).
- `--consul_ssl_verify`: If set to false, disable Consul certificate checking (true by default is ssl enabled).
- `--consul_tls_handshake_timeout`: Maximum duration to wait for a TLS handshake with Consul, the default is 50s.
- `--terraform_plugins_dir`: Specify the directory where to find Terraform pre-installed providers plugins. If not specified, required plugins will be downloaded during deployment. See <https://www.terraform.io/guides/running-terraform-in-automation.html#pre-installed-plugins> for more information.
- `--terraform_aws_plugin_version_constraint`: Specify the Terraform AWS plugin version constraint. Default one compatible with our source code is "`~> 1.36`". If you choose another, it's at your own risk. See <https://www.terraform.io/docs/configuration/providers.html#provider-versions> for more information.
- `--terraform_consul_plugin_version_constraint`: Specify the Terraform Consul plugin version constraint. Default one compatible with our source code is "`~> 2.1`". If you choose another, it's at your own risk. See <https://www.terraform.io/docs/configuration/providers.html#provider-versions> for more information.
- `--terraform_google_plugin_version_constraint`: Specify the Terraform Google plugin version constraint. Default one compatible with our source code is "`~> 1.18`". If you choose another, it's at your own risk. See <https://www.terraform.io/docs/configuration/providers.html#provider-versions> for more information.
- `--terraform_openstack_plugin_version_constraint`: Specify the Terraform OpenStack plugin version constraint. Default one compatible with our source code is "`~> 1.9`". If you choose another, it's at your own risk. See <https://www.terraform.io/docs/configuration/providers.html#provider-versions> for more information.
- `--terraform_keep_generated_files`: If set to true, generated Terraform infrastructures files on Yorc server are not deleted. (false by default: generated files are deleted).
- `--consul_publisher_max_routines`: Maximum number of parallelism used to store key/values in Consul. If you increase the default value you may need to tweak the ulimit max open files. If set to 0 or less the default value (500) will be used.
- `--graceful_shutdown_timeout`: Timeout to wait for a graceful shutdown of the Yorc server. After this delay the server immediately exits. The default is 5m.
- `--wf_step_graceful_termination_timeout`: Timeout to wait for a graceful termination of a workflow step during concurrent workflow step failure. After this delay the step is set on error. The default is 2m.
- `--http_address`: Restrict the listening interface for the Yorc HTTP REST API. By default Yorc listens on all available interfaces
- `--http_port`: Port number for the Yorc HTTP REST API. If omitted or set to '0' then the default port number is used, any positive integer will be used as it, and finally any negative value will let use a random port.

- `--keep_operation_remote_path`: If set to true, do not delete temporary artifacts on provisioned Compute at the end of deployment (false by default for deployment temporary artifacts cleanup).
- `--key_file`: File path to a PEM-encoded private key. The key is used to enable SSL for the Yorc HTTP REST API. This must be provided along with `cert_file`. If one of `key_file` or `cert_file` is not provided then SSL is disabled.
- `--cert_file`: File path to a PEM-encoded certificate. The certificate is used to enable SSL for the Yorc HTTP REST API. This must be provided along with `key_file`. If one of `key_file` or `cert_file` is not provided then SSL is disabled.
- `--ca_file`: If set to true, enable TLS certificate checking. Must be provided with `cert_file` ; `key_file` and `ca_file`. Disabled by default.
- `--ssl_verify`: If set to true, enable TLS certificate checking for clients of the Yorc's API. Must be provided with `cert_file` ; `key_file` and `ca_file`. Disabled by default.
- `--plugins_directory`: The name of the plugins directory of the Yorc server. The default is to use a directory named *plugins* in the current directory.
- `--resources_prefix`: Specify a prefix that will be used for names when creating resources such as Compute instances or volumes. Defaults to `yorc-`.
- `--workers_number`: Yorc instances use a pool of workers to handle deployment tasks. This option defines the size of this pool. If not set the default value of 30 will be used.
- `--working_directory` or `-w`: Specify an alternative working directory for Yorc. The default is to use a directory named *work* in the current directory.
- `--server_id`: Specify the server ID used to identify the server node in a cluster. The default is the hostname.
- `--disable_ssh_agent`: Allow disabling ssh-agent use for SSH authentication on provisioned computes. Default is false. If true, compute credentials must provide a path to a private key file instead of key content.

## 4.2 Configuration files

Configuration files are either JSON or YAML formatted as a single object containing the following configuration options. By default Yorc will look for a file named `config.yorc.json` in `/etc/yorc` directory then if not found in the current directory. The `-config` command line flag allows to specify an alternative configuration file.

Below is an example of configuration file.

```
{
  "resources_prefix": "yorc1-",
  "infrastructures": {
    "openstack": {
      "auth_url": "http://your-openstack:5000/v2.0",
      "tenant_name": "your-tenant",
      "user_name": "os-user",
      "password": "os-password",
      "private_network_name": "default-private-network",
      "default_security_groups": ["default"]
    }
  }
}
```

Below is an example of configuration file with TLS enabled.

```
{
  "resources_prefix": "yorc1-",
  "key_file": "/etc/pki/tls/private/yorc.key",
  "cert_file": "/etc/pki/tls/certs/yorc.crt",
  "infrastructures": {
    "openstack": {
      "auth_url": "http://your-openstack:5000/v2.0",
      "tenant_name": "your-tenant",
      "user_name": "os-user",
      "password": "os-password",
      "private_network_name": "default-private-network",
      "default_security_groups": ["default"]
    }
  }
}
```

- `server_graceful_shutdown_timeout`: Equivalent to `-graceful_shutdown_timeout` command-line flag.
- `wf_step_graceful_termination_timeout`: Equivalent to `-wf_step_graceful_termination_timeout` command-line flag.
- `http_address`: Equivalent to `-http_address` command-line flag.
- `http_port`: Equivalent to `-http_port` command-line flag.
- `key_file`: Equivalent to `-key_file` command-line flag.
- `cert_file`: Equivalent to `-cert_file` command-line flag.
- `ssl_verify`: Equivalent to `-ssl_verify` command-line flag.
- `ca_file`: Equivalent to `-ca_file` command-line flag.
- `plugins_directory`: Equivalent to `-plugins_directory` command-line flag.
- `resources_prefix`: Equivalent to `-resources_prefix` command-line flag.
- `workers_number`: Equivalent to `-workers_number` command-line flag.
- `working_directory`: Equivalent to `-working_directory` command-line flag.
- `server_id`: Equivalent to `-server_id` command-line flag.
- `disable_ssh_agent`: Equivalent to `-disable_ssh_agent` command-line flag.

### 4.2.1 Ansible configuration

Below is an example of configuration file with Ansible configuration options.

```
{
  "resources_prefix": "yorc1-",
  "infrastructures": {
    "openstack": {
      "auth_url": "http://your-openstack:5000/v2.0",
      "tenant_name": "your-tenant",
      "user_name": "os-user",
      "password": "os-password",
      "private_network_name": "default-private-network",
      "default_security_groups": ["default"]
    }
  }
}
```

(continues on next page)

(continued from previous page)

```

    }
  },
  "ansible": {
    "use_openssh": true,
    "connection_retries": 3,
    "hosted_operations": {
      "unsandboxed_operations_allowed": false,
      "default_sandbox": {
        "image": "jffloff/alpine-python:2.7-slim",
        "entrypoint": ["python", "-c"],
        "command": ["import time;time.sleep(31536000);"]
      }
    }
  }
}

```

All available configuration options for Ansible are:

- `use_openssh`: Equivalent to `-ansible_use_openssh` command-line flag.
- `debug`: Equivalent to `-ansible_debug` command-line flag.
- `connection_retries`: Equivalent to `-ansible_connection_retries` command-line flag.
- `cache_facts`: Equivalent to `-ansible_cache_facts` command-line flag.
- `archive_artifacts`: Equivalent to `-ansible_archive_artifacts` command-line flag.
- `job_monitoring_time_interval`: Equivalent to `-ansible_job_monitoring_time_interval` command-line flag.
- `operation_remote_base_dir`: Equivalent to `-operation_remote_base_dir` command-line flag.
- `keep_operation_remote_path`: Equivalent to `-keep_operation_remote_path` command-line flag.
- `keep_generated_recipes`: Equivalent to `-ansible_keep_generated_recipes` command-line flag.
- `hosted_operations`: This is a complex structure that allow to define the behavior of a Yorc server when it executes an hosted operation. For more information about hosted operation please see [The hosted operations paragraph in the TOSCA support section](#). This structure contains the following configuration options:
  - `unsandboxed_operations_allowed`: This option control if operations can be executed directly on the system that hosts Yorc if no default sandbox is defined. **This is not permitted by default.**
  - `default_sandbox`: This complex structure allows to define the default docker container to use to sandbox orchestrator-hosted operations. Bellow configuration options `entrypoint` and `command` should be carefully set to run the container and make it sleep until operations are executed on it. Defaults options will run a python inline script that sleeps for 1 year.
    - \* `image`: This is the docker image identifier (in the docker format `[repository/]name[:tag]`) is option is **required**.
    - \* `entrypoint`: This allows to override the default image entrypoint. If both `entrypoint` and `command` are empty the default value for `entrypoint` is `["python", "-c"]`.
    - \* `command`: This allows to run a command within the container. If both `entrypoint` and `command` are empty the default value for `command` is `["import time;time.sleep(31536000);"]`.
    - \* `env`: An optional list environment variables to set when creating the container. The format of each variable is `var_name=value`.

## Ansible performance considerations

As described in TOSCA *Supported Operations implementations*, Yorc supports these builtin implementations for operations to execute on remote hosts :

- Bash scripts
- Python scripts
- Ansible Playbooks

It is recommended to implement operations as Ansible Playbooks to get the best execution performance.

When operations are not implemented using Ansible playbooks, see the Performance section on *TOSCA Operations* to improve the performance of scripts execution on remote hosts.

### 4.2.2 Consul configuration

Below is an example of configuration file with Consul configuration options.

```
{
  "resources_prefix": "yorc1-",
  "infrastructures": {
    "openstack": {
      "auth_url": "http://your-openstack:5000/v2.0",
      "tenant_name": "your-tenant",
      "user_name": "os-user",
      "password": "os-password",
      "private_network_name": "default-private-network",
      "default_security_groups": ["default"]
    }
  },
  "consul": {
    "address": "http://consul-host:8500",
    "datacenter": "dc1",
    "publisher_max_routines": 500
  }
}
```

All available configuration options for Consul are:

- address: Equivalent to `-consul_address` command-line flag.
- token: Equivalent to `-consul_token` command-line flag.
- datacenter: Equivalent to `-consul_datacenter` command-line flag.
- key\_file: Equivalent to `-consul_key_file` command-line flag.
- cert\_file: Equivalent to `-consul_cert_file` command-line flag.
- ca\_cert: Equivalent to `-consul_ca_cert` command-line flag.
- ca\_path: Equivalent to `-consul_ca_path` command-line flag.
- ssl: Equivalent to `-consul_ssl` command-line flag.
- ssl\_verify: Equivalent to `-consul_ssl_verify` command-line flag.
- tls\_handshake\_timeout: Equivalent to `-consul_tls_handshake_timeout` command-line flag.
- publisher\_max\_routines: Equivalent to `-consul_publisher_max_routines` command-line flag.



### 4.2.3 Terraform configuration

Below is an example of configuration file with Terraform configuration options.

```
{
  "resources_prefix": "yorc1-",
  "infrastructures": {
    "openstack": {
      "auth_url": "http://your-openstack:5000/v2.0",
      "tenant_name": "your-tenant",
      "user_name": "os-user",
      "password": "os-password",
      "private_network_name": "default-private-network",
      "default_security_groups": ["default"]
    }
  },
  "terraform": {
    "plugins_dir": "home/yourc/terraform_plugins_directory",
  }
}
```

All available configuration options for Terraform are:

- `plugins_dir`: Equivalent to `-terraform_plugins_dir` command-line flag.
- `aws_plugin_version_constraint`: Equivalent to `-terraform_aws_plugin_version_constraint` command-line flag.
- `consul_plugin_version_constraint`: Equivalent to `-terraform_consul_plugin_version_constraint` command-line flag.
- `google_plugin_version_constraint`: Equivalent to `-terraform_google_plugin_version_constraint` command-line flag.
- `openstack_plugin_version_constraint`: Equivalent to `-terraform_openstack_plugin_version_constraint` command-line flag.
- `keep_generated_files`: Equivalent to `-terraform_keep_generated_files` command-line flag.

### 4.2.4 Telemetry configuration

Telemetry configuration can only be done via the configuration file. By default telemetry data are only stored in memory. See [Yorc Telemetry](#) for more information about telemetry.

Below is an example of configuration file with telemetry metrics forwarded to a Statsd instance and with a Prometheus HTTP endpoint exposed.

```
{
  "resources_prefix": "yorc1-",
  "infrastructures": {
    "openstack": {
      "auth_url": "http://your-openstack:5000/v2.0",
      "tenant_name": "your-tenant",
      "user_name": "os-user",
      "password": "os-password",
      "private_network_name": "default-private-network",
      "default_security_groups": ["default"]
    }
  }
}
```

(continues on next page)

(continued from previous page)

```
}
"telemetry": {
  "statsd_address": "127.0.0.1:8125",
  "expose_prometheus_endpoint": true
}
}
```

All available configuration options for telemetry are:

- `service_name`: Metrics keys prefix, defaults to `yorc`.
- `disable_hostname`: Specifies if gauge values should not be prefixed with the local hostname. Defaults to `false`.
- `disable_go_runtime_metrics`: Specifies Go runtime metrics (goroutines, memory, ...) should not be published. Defaults to `false`.
- `statsd_address`: Specify the address (in form `<address>:<port>`) of a statsd server to forward metrics data to.
- `statsite_address`: Specify the address (in form `<address>:<port>`) of a statsite server to forward metrics data to.
- `expose_prometheus_endpoint`: Specify if an HTTP Prometheus endpoint should be exposed allowing Prometheus to scrape metrics.

## 4.2.5 Deprecated configuration options

Deprecated since version 3.0.0.

- `ansible_use_openssh`: Equivalent to `-ansible_use_openssh` command-line flag.
- `ansible_debug`: Equivalent to `-ansible_debug` command-line flag.
- `ansible_connection_retries`: Equivalent to `-ansible_connection_retries` command-line flag.
- `operation_remote_base_dir`: Equivalent to `-operation_remote_base_dir` command-line flag.
- `keep_operation_remote_path`: Equivalent to `-keep_operation_remote_path` command-line flag.
- `consul_address`: Equivalent to `-consul_address` command-line flag.
- `consul_token`: Equivalent to `-consul_token` command-line flag.
- `consul_datacenter`: Equivalent to `-consul_datacenter` command-line flag.
- `consul_key_file`: Equivalent to `-consul_key_file` command-line flag.
- `consul_cert_file`: Equivalent to `-consul_cert_file` command-line flag.
- `consul_ca_cert`: Equivalent to `-consul_ca_cert` command-line flag.
- `consul_ca_path`: Equivalent to `-consul_ca_path` command-line flag.
- `consul_ssl`: Equivalent to `-consul_ssl` command-line flag.
- `consul_ssl_verify`: Equivalent to `-consul_ssl_verify` command-line flag.
- `consul_publisher_max_routines`: Equivalent to `-consul_publisher_max_routines` command-line flag.

## 4.3 Environment variables

- `YORC_ANSIBLE_USE_OPENSSH`: Equivalent to `-ansible_use_openssh` command-line flag.
- `YORC_ANSIBLE_DEBUG`: Equivalent to `-ansible_debug` command-line flag.
- `YORC_ANSIBLE_CONNECTION_RETRIES`: Equivalent to `-ansible_connection_retries` command-line flag.
- `YORC_ANSIBLE_CACHE_FACTS`: Equivalent to `-ansible_cache_facts` command-line flag.
- `YORC_ANSIBLE_JOB_MONITORING_TIME_INTERVAL`: Equivalent to `-ansible_job_monitoring_time_interval` command-line flag.
- `YORC_ANSIBLE_KEEP_GENERATED_RECIPES`: Equivalent to `-ansible_keep_generated_recipes` command-line flag.
- `YORC_OPERATION_REMOTE_BASE_DIR`: Equivalent to `-operation_remote_base_dir` command-line flag.
- `YORC_CONSUL_ADDRESS`: Equivalent to `-consul_address` command-line flag.
- `YORC_CONSUL_TOKEN`: Equivalent to `-consul_token` command-line flag.
- `YORC_CONSUL_DATACENTER`: Equivalent to `-consul_datacenter` command-line flag.
- `YORC_CONSUL_KEY_FILE`: Equivalent to `-consul_key_file` command-line flag.
- `YORC_CONSUL_CERT_FILE`: Equivalent to `-consul_cert_file` command-line flag.
- `YORC_CONSUL_CA_CERT`: Equivalent to `-consul_ca_cert` command-line flag.
- `YORC_CONSUL_CA_PATH`: Equivalent to `-consul_ca_path` command-line flag.
- `YORC_CONSUL_SSL`: Equivalent to `-consul_ssl` command-line flag.
- `YORC_CONSUL_SSL_VERIFY`: Equivalent to `-consul_ssl_verify` command-line flag.
- `YORC_CONSUL_TLS_HANDSHAKE_TIMEOUT`: Equivalent to `-consul_tls_handshake_timeout` command-line flag.
- `YORC_CONSUL_PUBLISHER_MAX_ROUTINES`: Equivalent to `-consul_publisher_max_routines` command-line flag.
- `YORC_SERVER_GRACEFUL_SHUTDOWN_TIMEOUT`: Equivalent to `-graceful_shutdown_timeout` command-line flag.
- `YORC_WF_STEP_GRACEFUL_TERMINATION_TIMEOUT`: Equivalent to `-wf_step_graceful_termination_timeout` command-line flag.
- `YORC_HTTP_ADDRESS`: Equivalent to `-http_address` command-line flag.
- `YORC_HTTP_PORT`: Equivalent to `-http_port` command-line flag.
- `YORC_KEEP_OPERATION_REMOTE_PATH`: Equivalent to `-keep_operation_remote_path` command-line flag.
- `YORC_KEY_FILE`: Equivalent to `-key_file` command-line flag.
- `YORC_CERT_FILE`: Equivalent to `-cert_file` command-line flag.
- `YORC_SSL_VERIFY`: Equivalent to `-ssl_verify` command-line flag.
- `YORC_CA_FILE`: Equivalent to `-ca_file` command-line flag.
- `YORC_PLUGINS_DIRECTORY`: Equivalent to `-plugins_directory` command-line flag.
- `YORC_RESOURCES_PREFIX`: Equivalent to `-resources_prefix` command-line flag.
- `YORC_WORKERS_NUMBER`: Equivalent to `-workers_number` command-line flag.

- `YORC_WORKING_DIRECTORY`: Equivalent to `-working_directory` command-line flag.
- `YORC_SERVER_ID`: Equivalent to `-server_id` command-line flag.
- `YORC_DISABLE_SSH_AGENT`: Equivalent to `-disable_ssh_agent` command-line flag.
- `YORC_LOG`: If set to 1 or `DEBUG`, enables debug logging for Yorc.
- `YORC_TERRAFORM_PLUGINS_DIR`: Equivalent to `-terraform_plugins_dir` command-line flag.
- `YORC_TERRAFORM_AWS_PLUGIN_VERSION_CONSTRAINT`:                      Equivalent                      to                      `-terraform_aws_plugin_version_constraint` command-line flag.
- `YORC_TERRAFORM_CONSUL_PLUGIN_VERSION_CONSTRAINT`:                      Equivalent                      to                      `-terraform_consul_plugin_version_constraint` command-line flag.
- `YORC_TERRAFORM_GOOGLE_PLUGIN_VERSION_CONSTRAINT`:                      Equivalent                      to                      `-terraform_google_plugin_version_constraint` command-line flag.
- `YORC_TERRAFORM_OPENSTACK_PLUGIN_VERSION_CONSTRAINT`:                      Equivalent                      to                      `-terraform_openstack_plugin_version_constraint` command-line flag.
- `YORC_TERRAFORM_KEEP_GENERATED_FILES`:                      Equivalent                      to                      `-terraform_keep_generated_files` command-line flag.

## 4.4 Infrastructures configuration

Due to the pluggable nature of infrastructures support in Yorc their configuration differ from other configurable options. An infrastructure configuration option could be specified by either a its configuration placeholder in the configuration file, a command line flag or an environment variable.

The general principle is for a configurable option `option_1` for infrastructure `infra1` it should be specified in the configuration file as following:

```
{
  "infrastructures": {
    "infra1": {
      "option_1": "value"
    }
  }
}
```

Similarly a command line flag with the name `--infrastructure_infra1_option_1` and an environment variable with the name `YORC_INFRA_INFRA1_OPTION_1` will be automatically supported and recognized. The default order of precedence apply here.

## 4.5 Builtin infrastructures configuration

### 4.5.1 OpenStack

OpenStack infrastructure key name is `openstack` in lower case.

Option Name	Description	Data Type	Required	Default
auth_url	Specify the authentication url for OpenStack (should be the Keystone endpoint ie: <a href="http://your-openstack:5000/v2.0">http://your-openstack:5000/v2.0</a> ).	string	yes	
tenant_id	Specify the OpenStack tenant id to use.	string	Either this or tenant_name should be provided.	
tenant_name	Specify the OpenStack tenant name to use.	string	Either this or tenant_id should be provided.	
user_name	Specify the OpenStack user name to use.	string	yes	
password	Specify the OpenStack password to use.	string	yes	
region	Specify the OpenStack region to use	string	no	RegionOne
private_network_name	Specify the name of private network to use as primary administration network between Yorc and Compute instances. It should be a private network accessible by this instance of Yorc.	string	Required to use the <code>PRIVATE</code> keyword for TOSCA admin networks	
provisioning_over_fip_allowed	This allows to perform the provisioning of a Compute over the associated floating IP if it exists. This is useful when Yorc is not deployed on the same private network than the provisioned Compute.	boolean	no	false
default_security_groups	Default security groups to be used when creating a Compute instance. It should be a comma-separated list of security group names	list of strings	no	
insecure	Trust self-signed SSL certificates	boolean	no	false
cacert_file	Specify a custom CA certificate when communicating over SSL. You can specify either a path to the file or the contents of the certificate	string	no	
cert	Specify client certificate file for SSL client authentication. You can specify either a path to the file or the contents of the certificate	string	no	
key	Specify client private key file for SSL client authentication. You can specify either a path to the file or the contents of the key	string	no	

## 4.5.2 Kubernetes

Kubernetes infrastructure key name is `kubernetes` in lower case.

Option Name	Description	Data Type	Re-quired	De-fault
<code>kubeconfig</code>	Path or content of Kubernetes cluster configuration file*	string	no	
<code>application_credentials</code>	Path or content of file containing credentials**	string	no	
<code>master_url</code>	URL of the HTTP API of Kubernetes is exposed. Format: <code>https://&lt;host&gt;:&lt;port&gt;</code>	string	no	
<code>ca_file</code>	Path to a trusted root certificates for server	string	no	
<code>cert_file</code>	Path to the TLS client certificate used for authentication	string	no	
<code>key_file</code>	Path to the TLS client key used for authentication	string	no	
<code>insecure</code>	Server should be accessed without verifying the TLS certificate (testing only)	boolean	no	
<code>job_monitoring_time_interval</code>	Default duration for job monitoring time interval	string	no	5s

- `kubeconfig` is the path (accessible to Yorc server) or the content of a Kubernetes cluster configuration file. When `kubeconfig` is defined, other infrastructure configuration properties (`master_url`, keys or certificates) don't have to be defined here.

If neither `kubeconfig` nor `master_url` is specified, the Orchestrator will consider it is running within a Kubernetes Cluster and will attempt to authenticate inside this cluster.

- `application_credentials` is the path (accessible to Yorc server) or the content of a file containing Google service account private keys in JSON format. This file can be downloaded from the Google Cloud Console at [Google Cloud service account file](#). It is needed to authenticate against Google Cloud when the `kubeconfig` property above refers to a Kubernetes Cluster created on Google Kubernetes Engine, and the orchestrator is running on a host where `gcloud` is not installed.

## 4.5.3 Google Cloud Platform

Google Cloud Platform infrastructure key name is `google` in lower case.

Option Name	Description	Data Type	Re-quired	Default
<code>project</code>	ID of the project to apply any resources to	string	yes	
<code>application_credentials</code>	Path of file containing credentials*	string	no	Google Application Default Credentials
<code>credentials</code>	Content of file containing credentials	string	no	Google Application Default Credentials
<code>region</code>	The region to operate under	string	no	

`application_credentials` is the path (accessible to Yorc server) of a file containing service account private keys in JSON format. This file can be downloaded from the Google Cloud Console at [Google Cloud service account file](#).

If no file path is specified in `application_credentials` and no file content is specified in `credentials`, the orchestrator will fall back to using the [Google Application Default Credentials](#) if any.

#### 4.5.4 AWS

AWS infrastructure key name is `aws` in lower case.

Option Name	Description	Data Type	Required	Default
<code>access_key</code>	Specify the AWS access key credential.	string	yes	
<code>secret_key</code>	Specify the AWS secret key credential.	string	yes	
<code>region</code>	Specify the AWS region to use.	string	yes	

#### 4.5.5 Slurm

Slurm infrastructure key name is `slurm` in lower case.

Option Name	Description	Data Type	Required	Default
<code>user_name</code>	SSH Username to be used to connect to the Slurm Client's node	string	yes	
<code>password</code>	SSH Password to be used to connect to the Slurm Client's node	string	Either this or <code>private_key</code> should be provided	
<code>private_key</code>	SSH Private key to be used to connect to the Slurm Client's node	string	Either this or <code>password</code> should be provided	
<code>url</code>	IP address of the Slurm Client's node	string	yes	
<code>port</code>	SSH Port to be used to connect to the Slurm Client's node	string	yes	
<code>default_job_name</code>	Default name for the job allocation.	string	no	
<code>job_monitoring_time</code>	Default duration for job monitoring time interval	string	no	5s

## 4.6 Vault configuration

Due to the pluggable nature of vaults support in Yorc their configuration differ from other configurable options. A vault configuration option could be specified by either its configuration placeholder in the configuration file, a command line flag or an environment variable.

The general principle is for a configurable option `option_1` it should be specified in the configuration file as following:

```
{
  "vault": {
    "type": "vault_implementation",
    "option_1": "value"
  }
}
```

Similarly a command line flag with the name `--vault_option_1` and an environment variable with the name `YORC_VAULT_OPTION_1` will be automatically supported and recognized. The default order of precedence apply here.

`type` is the only mandatory option for all vaults configurations, it allows to select the vault implementation by specifying it's ID. If the `type` option is not present either in the config file, as a command line flag or as an environment variable, Vault configuration will be ignored.

The integration with a Vault is totally optional and this configuration part may be leave empty.

## 4.7 Builtin Vaults configuration

### 4.7.1 HashiCorp's Vault

This is the only builtin supported Vault implementation. Implementation ID to use with the vault type configuration parameter is `hashicorp`.

Bellow are recognized configuration options for Vault:

Option Name	Description	Data Type	Required	Default
<code>address</code>	Address is the address of the Vault server. This should be a complete URL such as <code>"https://vault.example.com"</code> .	string	yes	
<code>max_retries</code>	MaxRetries controls the maximum number of times to retry when a 5xx error occurs. Set to 0 or less to disable retrying.	integer	no	0
<code>timeout</code>	Timeout is for setting custom timeout parameter in the HttpClient.	string	no	
<code>ca_cert</code>	CACert is the path to a PEM-encoded CA cert file to use to verify the Vault server SSL certificate.	string	no	
<code>ca_path</code>	CAPath is the path to a directory of PEM-encoded CA cert files to verify the Vault server SSL certificate.	string	no	
<code>client_cert</code>	ClientCert is the path to the certificate for Vault communication.	string	no	
<code>client_key</code>	ClientKey is the path to the private key for Vault communication	string	no	
<code>tls_server_name</code>	TLSServerName, if set, is used to set the SNI host when connecting via TLS.	string	no	
<code>tls_skip_verify</code>	Disables SSL verification	boolean	no	false
<code>token</code>	Specifies the access token to use to connect to vault. This is highly discouraged to this option in the configuration file as the token is a sensitive data and should not be written on disk. Prefer the associated environment variable	string	no	



---

## Yorc Client CLI Configuration

---

This section is dedicated to the CLI part of yorc that covers everything except the server configuration detailed above. It focus on configuration options commons to all the commands. Sub commands may have additional options please use the cli *help* command to see them.

Just like for its server part Yorc Client CLI has various configuration options that could be specified either by command-line flags, configuration file or environment variables.

If an option is specified several times using flags, environment and config file, command-line flag will have the precedence then the environment variable and finally the value defined in the configuration file.

### 5.1 Command-line options

- `--ca_file`: This provides a file path to a PEM-encoded certificate authority. This implies the use of HTTPS to connect to the Yorc REST API.
- `--ca_path`: Path to a directory of PEM-encoded certificates authorities. This implies the use of HTTPS to connect to the Yorc REST API.
- `--cert_file`: File path to a PEM-encoded client certificate used to authenticate to the Yorc API. This must be provided along with key-file. If one of key-file or cert-file is not provided then SSL authentication is disabled. If both cert-file and key-file are provided this implies the use of HTTPS to connect to the Yorc REST API.
- `-c` or `--config`: config file (default is `/etc/yorc/yorc-client.[json|yaml]`)
- `--key_file`: File path to a PEM-encoded client private key used to authenticate to the Yorc API. This must be provided along with cert-file. If one of key-file or cert-file is not provided then SSL authentication is disabled. If both cert-file and key-file are provided this implies the use of HTTPS to connect to the Yorc REST API.
- `--skip_tls_verify`: Controls whether a client verifies the server's certificate chain and host name. If set to true, TLS accepts any certificate presented by the server and any host name in that certificate. In this mode, TLS is susceptible to man-in-the-middle attacks. This should be used only for testing. This implies the use of HTTPS to connect to the Yorc REST API.
- `-s` or `--ssl_enabled`: Use HTTPS to connect to the Yorc REST API. This is automatically implied if one of `--ca_file`, `--ca_path`, `--cert_file`, `--key_file` or `--skip_tls_verify` is provided.

- `--yorc_api`: specify the host and port used to join the Yorc' REST API (default "localhost:8800")

## 5.2 Configuration files

Configuration files are either JSON or YAML formatted as a single object containing the following configuration options. By default Yorc will look for a file named `yorc-client.json` or `yorc-client.yaml` in `/etc/yorc` directory then if not found in the current directory. The `-config` command line flag allows to specify an alternative configuration file.

- `ca_file`: Equivalent to `-ca_file` command-line flag.
- `ca_path`: Equivalent to `-ca_path` command-line flag.
- `cert_file`: Equivalent to `-cert_file` command-line flag.
- `key_file`: Equivalent to `-key_file` command-line flag.
- `skip_tls_verify`: Equivalent to `-skip_tls_verify` command-line flag.
- `ssl_enabled`: Equivalent to `-ssl_enabled` command-line flag.
- `yorc_api`: Equivalent to `-yorc_api` command-line flag.

## 5.3 Environment variables

- `YORC_CA_FILE`: Equivalent to `-ca_file` command-line flag.
- `YORC_CA_PATH`: Equivalent to `-ca_path` command-line flag.
- `YORC_CERT_FILE`: Equivalent to `-cert_file` command-line flag.
- `YORC_KEY_FILE`: Equivalent to `-key_file` command-line flag.
- `YORC_SKIP_TLS_VERIFY`: Equivalent to `-skip_tls_verify` command-line flag.
- `YORC_SSL_ENABLED`: Equivalent to `-ssl_enabled` command-line flag.
- `YORC_API`: Equivalent to `-yorc_api` command-line flag.

---

## Starting Yorc

---

### 6.1 Starting Consul

Yorc requires a running Consul instance prior to be started.

Here is how to start a standalone Consul server instance on the same host than Yorc:

```
consul agent -server -bootstrap-expect 1 -data-dir ./consul-data
```

---

**Note:** Wait for the agent : `Synced service 'consul'` log message to appear before continuing

---

### 6.2 Starting Yorc

Please report to the [Yorc Server Configuration](#) for an exhaustive list of Yorc' configuration options. At least OpenStack access configuration files should be provided either by command-line flags, environment variables or configuration elements. They are omitted bellow for brevity and considered as provided by a configuration file in one of the default location.

Note that if you are using a passphrase on your ssh key, you have to start an ssh-agent before launching yorc. It is strongly recommended to start one by giving him a socket name.

```
eval `ssh-agent -a /tmp/ssh-sock`
```

So in case of your ssh-agent process die, just restart it with the command above.

If your ssh key does not have a passphrase, **do not start any ssh-agent** before starting yorc and make sure that environment variable SSH\_AUTH\_SOCKET is not set.

```
killall ssh-agent  
unset SSH_AUTH_SOCKET
```

Then start yorc

```
yorc server
```

---

## Yorc Command Line Interface

---

You can interact with a Yorc server using a command line interface (CLI). The same binary as for running a Yorc server is used for the CLI.

### 7.1 General Options

- `--yorc-api`: Specifies the host and port used to join the Yorc' REST API. Defaults to `localhost:8800`. Configuration entry `yorc_api` and env var `YORC_API` may also be used.
- `--no-color`: Disable coloring output (By default coloring is enable).
- `-s` or `--secured`: Use HTTPS to connect to the Yorc REST API
- `--ca-file`: This provides a file path to a PEM-encoded certificate authority. This implies the use of HTTPS to connect to the Yorc REST API.
- `--skip-tls-verify`: `skip-tls-verify` controls whether a client verifies the server's certificate chain and host name. If set to true, TLS accepts any certificate presented by the server and any host name in that certificate. In this mode, TLS is susceptible to man-in-the-middle attacks. This should be used only for testing. This implies the use of HTTPS to connect to the Yorc REST API.

### 7.2 CLI Commands related to deployments

All deployments related commands are sub-commands of a command named `deployments`. In practice that means that the commands starts with

```
yorc deployments
```

For brevity `deployments` supports the following aliases: `depls`, `depl`, `deps`, `dep` and `d`.

### 7.2.1 Deploy a CSAR

Deploys a file or directory pointed by <csar\_path> If <csar\_path> point to a valid zip archive it is submitted to Yorc as it. If <csar\_path> point to a file or directory it is zipped before being submitted to Yorc. If <csar\_path> point to a single file it should be TOSCA YAML description.

```
yorc deployments deploy <csar_path> [flags]
```

**Flags:**

- --id: Specify a id for this deployment. This id should not already exist, should respect the following format: `^[ _0-9a-zA-Z ]+$` and should be less than 36 characters long (Optional otherwise a unique ID is generated by Yorc)
- -e, --stream-events: Stream events after deploying the CSAR.
- -l, --stream-logs: Stream logs after deploying the CSAR. In this mode logs can't be filtered, to use this feature see the "log" command.

### 7.2.2 Undeploy a deployment

Undeploy an application specifying the deployment ID.

```
yorc deployments undeploy <DeploymentId> [flags]
```

**Flags:**

- -p, --purge: To use if you want to purge instead of undeploy.
- -e, --stream-events: Stream events after deploying the CSAR.
- -l, --stream-logs: Stream logs after deploying the CSAR. In this mode logs can't be filtered, to use this feature see the "log" command.

### 7.2.3 List deployments

List active deployments. Giving there ids and statuses.

```
yorc deployments list
```

### 7.2.4 Get information on a specific deployment

Display information about a given deployment. It prints the deployment status and the status of all the nodes contained in this deployment.

```
yorc deployments info <DeploymentId> [flags]
```

**Flags:**

- -d, --detailed: Add details to the info command making it less concise and readable.
- -f, --follow: Follow deployment info updates (without details) until the deployment is finished.

## 7.2.5 Get deployment events

Streams events for all or a given deployment id

```
yorc deployments events [<DeploymentId>] [flags]
```

### Flags:

- `-b, --from-beginning`: Show events from the beginning of a deployment
- `-n, --no-stream`: Show events then exit. Do not stream events. It implies `--from-beginning`

## 7.2.6 Get deployment logs

Streams logs for all or a given deployment id. The log format is: [Timestamp][Level][DeploymentID][WorkflowID][ExecutionID][NodeID][InstanceID][InterfaceName][OperationName][TypeID]Content

```
yorc deployments logs [<DeploymentId>] [flags]
```

### Flags:

- `-b, --from-beginning`: Show logs from the beginning of a deployment
- `-n, --no-stream`: Show logs then exit. Do not stream logs. It implies `--from-beginning`

## 7.2.7 Get deployment tasks

Display info about the tasks related to a given deployment. It prints the tasks ID, type and status.

```
yorc deployments tasks <DeploymentId> [flags]
```

## 7.2.8 Get deployment task info

Display information about a given task specifying the deployment id and the task id.

```
yorc deployments task info <DeploymentId> <TaskId> [flags]
```

### Flags:

- `-w, --steps`: Show steps of the related workflow associated to the task

## 7.2.9 Cancel a deployment task

Cancel a task specifying the deployment id and the task id. The task should be in status “INITIAL” or “RUNNING” to be canceled.

```
yorc deployments tasks cancel <DeploymentId> <TaskId> [flags]
```

### 7.2.10 Resume a deployment task

Resume a task specifying the deployment id and the task id. The task should be in status “FAILED” to be resumed.

```
yorc deployments tasks resume <DeploymentId> <TaskId> [flags]
```

### 7.2.11 Fix a deployment task step

Fix a task step specifying the deployment id, the task id and the step name. The task step must be on error to be fixed.

```
yorc deployments tasks fix <DeploymentId> <TaskId> <StepName> [flags]
```

### 7.2.12 Scale a specific node

Scale a given node of a deployment <DeploymentId> by adding or removing the specified number of instances.

```
yorc deployments scale <DeploymentId> [flags]
```

**Flags:**

- `-d, --delta`: The non-zero number of instance to add (if > 0) or remove (if < 0).
- `-n, --node`: The name of the node that should be scaled.
- `-e, --stream-events`: Stream events after issuing the scaling request.
- `-l, --stream-logs`: Stream logs after issuing the scaling request. In this mode logs can't be filtered, to use this feature see the “log” command.

### 7.2.13 Execute a custom command

Executes a custom command for a given node of a deployment <DeploymentId>.

```
yorc deployments custom <DeploymentId> [flags]
```

**Flags:**

- `--custom`: Provide the custom command name (mandatory)
- `--interface`: Provide the interface name (mandatory)
- `-d, --data`: Provide the JSON format of the custom command with node, interface, custom and inputs data
- `-i, --input`: Provide the input for the custom command
- `-n, --node`: Provide the node name (mandatory)

Example using `--input` flags:

```
yorc deployments custom deployID --custom cmdName --interface interfaceName --node_↵
↵nodeName --input 'key1=["value1","value2"]' --input 'key2="value3"'
```

Example using `--data` flag:

```
yorc deployments custom deployID --data '{"name":"cmdName","interface":"interfaceName_↵
↵","node":"nodeName","inputs":{"key1":["value1","value2"],"key2":"value3"}}'
```



### 7.2.14 List workflows of a given deployment

Lists workflows defined in a deployment <DeploymentId>.

```
yorc deployments workflows list <DeploymentId> [flags]
```

### 7.2.15 Execute a workflow on a given deployment

Trigger a workflow on deployment <DeploymentId>.

```
yorc deployments workflows execute <DeploymentId> [flags]
```

#### Flags:

- `--continue-on-error`: By default if an error occurs in a step of a workflow then other running steps are cancelled and the workflow is stopped. This flag allows to continue to the next steps even if an error occurs.
- `-e, --stream-events`: Stream events after riggering a workflow.
- `-l, --stream-logs`: Stream logs after triggering a workflow. In this mode logs can't be filtered, to use this feature see the "log" command.
- `-w, --workflow-name`: The workflows name (**mandatory**)

### 7.2.16 Show a workflow on a given deployment

Show a human readable textual representation of a given TOSCA workflow defined in deployment <DeploymentId>.

```
yorc deployments workflows show <DeploymentId> [flags]
```

#### Flags:

- `-w, --workflow-name`: The workflows name (**mandatory**)

### 7.2.17 Generate a graphical representation of a workflow on a given deployment

Generate a GraphViz Dot format representation of a given workflow. The output can be easily converted to an image by making use of the dot command provided by GraphViz:

```
yorc deployments workflows graph <DeploymentId> [flags] | dot -Tpng > graph.png
```

#### Flags:

- `-w, --workflow-name`: The workflows name (**mandatory**)
- `--horizontal`: Draw graph with an horizontal layout. (layout is vertical by default)

## 7.3 CLI Commands related to hosts pool

All hosts pool related commands are sub-commands of a command named `hostspool`. In practice that means that the commands starts with

```
yorc hostspool
```

For brevity `hostspool` supports the following aliases: `hostpool`, `hostsp`, `hpool` and `hp`.

### 7.3.1 Add a host pool

Adds a host to the hosts pool managed by this Yorc cluster. The `<hostname>` should not already exist. The connection object of the JSON request is mandatory while the labels list is optional. This labels list should be composed with elements with the “op” parameter set to “add” but it could be omitted.

```
yorc hostspool add <hostname> [flags]
```

#### Flags:

- `--data` or `-d` : Specify a JSON format for the host pool to add. The JSON format for the host pool is described below.
- `--key` or `-k` : Specify a private key to access host if no host connection is defined in JSON format. **(mandatory if no password is defined)**
- `--password` or `-p` : Specify a password to access host if no host connection is defined in JSON format. **(mandatory if no private key is defined)**
- `--host`: Hostname or ip address used to connect to the host. (defaults to the hostname in the hosts pool)
- `--label`: Label in form `key=value` to add to the host. May be specified several time.
- `--port`: Port used to connect to the host. (default 22)
- `--user`: User used to connect to the host (default “root”)

Host pool (JSON):

```
{
  "connection": {
    "host": "defaults_to_<hostname>",
    "user": "defaults_to_root",
    "port": "defaults_to_22",
    "private_key": "one_of_password_or_private_key_required",
    "password": "one_of_password_or_private_key_required"
  },
  "labels": [
    {"name": "os.type", "value": "linux"},
    {"op": "add", "name": "host.mem_size", "value": "4G"}
  ]
}
```

### 7.3.2 Update a host pool

Update labels list or connection of a host of the hosts pool managed by this Yorc cluster. The `<hostname>` should exists. Both connection and labels list object of the JSON request are optional. This labels list should be composed with elements with the “op” parameter set to “add” or “remove” but defaults to “add” if omitted. *Adding* a tag that already exists replace its value.

```
yorc hostspool update <hostname> [flags]
```

#### Flags:

- `--data` or `-d` : Specify a JSON format for the host pool to update. The JSON format for the host pool is described below.
- `--add-label`: Add a label in form ‘key=value’ to the host. May be specified several time.
- `--host`: Hostname or ip address used to connect to the host. (defaults to the hostname in the hosts pool)
- `--key` or `-k`: At any time a host of the pool should have at least one of private key or password. To delete a registered private key use the “-” character.
- `--password` or `-p`: At any time a host of the pool should have at least one of private key or password. To delete a registered password use the “-” character.
- `--port`: Port used to connect to the host. (defaults to the hostname in the hosts pool) (default 22)
- `--remove-label`: Remove a label from the host. May be specified several time.
- `--user`: User used to connect to the host (default “root”)

Host pool (JSON):

```
{
  "connection": {
    "host": "defaults_to_<hostname>",
    "user": "defaults_to_root",
    "port": "defaults_to_22",
    "private_key": "one_of_password_or_private_key_required",
    "password": "one_of_password_or_private_key_required"
  },
  "labels": [
    {"name": "os.type", "value": "linux"},
    {"op": "add", "name": "host.mem_size", "value": "4G"},
    {"op": "remove", "name": "host.disk_size"}
  ]
}
```

### 7.3.3 Delete a host pool

Deletes a host from the hosts pool managed by this Yorc cluster. The <hostname> should exists.

```
yorc hostspool delete <hostname> [<hostname>...]
```

### 7.3.4 List hosts in the pool

Lists hosts of the hosts pool managed by this Yorc cluster.

```
yorc hostspool list [flags]
```

#### Flags:

- `--filter` or `-f`: Filter hosts based on their labels. May be specified several time, filters are joined by a logical ‘and’. Please refer to [Filters Grammar](#) for more details. Note: If the filter expression contains a comma as in “mylabel in (v1,v2)”, wrap it with single quotes as in the example below:

```
yorc hp list -f '"mylabel in (v1, v2)'"
```

### 7.3.5 Get information on a specific host in the pool

Gets the description of a host of the hosts pool managed by this Yorc cluster.

```
yorc hostspool info <hostname>
```

### 7.3.6 Apply a Hosts Pool configuration

Applies a Hosts Pool configuration provided in a YAML or JSON file. This command will compare and display the differences between the current Hosts Pool configuration and the configuration specified in the file. A user confirmation will be asked before proceeding. The command will fail if the new configuration would result in the removal of a host currently allocated for a deployment.

```
yorc hostspool apply <filename>
```

#### Flags:

- `--auto-approve`: Skip interactive approval before applying the new Hosts Pool configuration.

YAML and JSON formats are accepted. The following properties are supported :

- **hosts**: List of hosts configuration. A host configuration supports the following properties,
  - **name**: mandatory string identifying the host, no other host entry can have the same name value in the file
  - **connection**: Connection configuration,
    - \* **host**: Hostname or ip address used to connect to the host (defaults to the name described above)
    - \* **user**: name of the user used to connect to the host (default “root”)
    - \* **password**: either a password or a private key should be provided
    - \* **private\_key**: Path to a private key file (or private key file content), either a password or a private key should be provided
    - \* **port**: Port used to connect to the host (default 22)
  - **labels**: key/value pairs (see *Filters Grammar* for more details on labels)

Example of a YAML Hosts Pool configuration file :

```
hosts:
- name: host1
  connection:
    host: host1.example.com
    user: test
    private_key: /path/to/secrets/id_rsa
    port: 22
  labels:
    environment: dev
    testlabel: hello
    host.cpu_frequency: 3 GHz
    host.disk_size: 50 GB
    host.mem_size: 4GB
    host.num_cpus: "4"
    os.architecture: x86_64
    os.distribution: ubuntu
```

(continues on next page)

(continued from previous page)

```
    os.type: linux
    os.version: "17.1"
- name: host2
  connection:
    host: host2.example.com
    user: test
    password: test
```

### 7.3.7 Export a Hosts Pool configuration

Exports a Hosts Pool configuration as a YAML or JSON representation, to the standard output or a file.

```
yorc hostspool export
```

**Flags:**

- `--output` or `-o`: Output format, `yaml` or `json` (default `yaml`)
- `--file` or `-f`: Path to a file where to store the output (default standard output)



---

## Yorc Supported infrastructures

---

This section describes the state of our integration with supported infrastructures and their specificities

### 8.1 Hosts Pool

The Hosts Pool is a very special kind of infrastructure. It consists in registering existing Compute nodes into a pool managed by Yorc. Those compute nodes could be physical or virtual machines, containers or whatever as long as Yorc can SSH into it. Yorc will be responsible to allocate and release hosts for deployments. This is safe to use it concurrently in a Yorc cluster, Yorc instances will synchronize amongst themselves to ensure consistency of the pool.

To sum up this infrastructure type is really great when you want to use an infrastructure that is not yet supported by Yorc. Just take care you're responsible for handling the compatibility or conflicts of what is already installed and what will be by Yorc on your hosts pool. The best practice is using container isolation. This is especially true if a host can be shared by several apps by specifying in Tosca with the Compute **shareable** property.

#### 8.1.1 Hosts management

Yorc comes with a REST API that allows to manage hosts in the pool and to easily integrate it with other systems. The Yorc CLI leverage this REST API to make it user friendly, please refer to *CLI Commands related to hosts pool* for more information

#### 8.1.2 Hosts Pool labels & filters

It is strongly recommended to associate labels to your hosts. Labels allow to filter hosts based on criteria. Labels are just a couple of key/value pair

##### Filters Grammar

There are four kinds of filters supported by Yorc:

- Filters based on the presence of a label `label_identifier` will match if a label with the given name is associated with a host whatever its value is.
- Filters based on equality to a value `label_identifier (|=|!=) value` will match if the value associated with the given label is equals (= and ==) or different (!=) to the given value
- Filters based on sets `label_identifier (in | not in) (value [, other_value])` will match if the value associated with the given label is one (in) or is not one (not in) of the given values
- Filters based on comparisons `label_identifier (< | <= | > | >=) number[unit]` will match if the value associated with the given label is a number and matches the comparison sign. A unit could be associated with the number, currently supported units are golang durations (“ns”, “us”, “ms”, “s”, “m” or “h”), bytes units (“B”, “KiB”, “KB”, “MiB”, “MB”, “GiB”, “GB”, “TiB”, “TB”, “PiB”, “PB”, “EiB”, “EB”) and [International System of Units \(SI\)](#). The case of the unit does not matter.

Here are some example:

- `gpu`
- `os.distribution != windows`
- `os.architecture == x86_64`
- `environment = "Q&A"`
- `environment in ( "Q&A", dev, edge)`
- `gpu.type not in (k20, m60)`
- `gpu_nb > 1`
- `os.mem_size >= 4 GB`
- `os.disk_size < 1tb`
- `max_allocation_time <= 120h`

## Implicit filters & labels

TOSCA allows to specify [requirements on Compute hardware](#) and [Compute operating system](#) . These are capabilities named `host` and `os` in the [TOSCA node Compute](#) . If those are specified in the topology, Yorc will automatically add a filter `host.<property_name> >= <property_value> <property_unit>` or `os.<property_name> = <property_value>` This will allow to select hosts matching the required criteria.

**This means that it is strongly recommended to add the following labels to your hosts:**

- `host.num_cpus` (ie. `host.num_cpus=4`)
- `host.cpu_frequency` (ie. `host.cpu_frequency=3 GHz`)
- `host.disk_size` (ie. `host.disk_size=50 GB`)
- `host.mem_size` (ie. `host.mem_size=4GB`)
- `os.architecture` (ie. `os.architecture=x86_64`)
- `os.type` (ie. `os.type=linux`)
- `os.distribution` (ie. `os.distribution=ubuntu`)
- `os.version` (ie. `os.version=17.10`)

Some labels are also automatically exposed as TOSCA Compute instance attributes:



- if present a label named `private_address` will be used as attribute `private_address` and `ip_address` of the Compute. If not set the connection host will be used instead this allows to specify a network different for the applicative communication and for the orchestrator communication
- if present a label named `public_address` will be used as attribute `public_address` of the Compute.
- if present, following labels will fill the `networks` attribute of the Compute node:
  - `networks.<idx>.network_name` (ie. `networks.0.network_name`)
  - `networks.<idx>.network_id` (ie. `networks.0.network_id`)
  - `networks.<idx>.addresses` as a coma separated list of addresses (ie. `networks.0.addresses`)

The resources host pool labels (`host.num_cpus`, `host.disk_size`, `host.mem_size`) are automatically decreased and increased respectively when a host pool is allocated and released only if you specify any of these Tosca host resources capabilities Compute in its Alien4Cloud applications. If you apply a new configuration on allocated hosts with new host resources labels, they will be recalculated depending on existing allocations resources.

## 8.2 Slurm

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. It is widely used in High Performance Computing and it is the default scheduler of the [Bull Super Computer Suite](#).

Yorc interacts with Slurm to allocate nodes on its cluster but also to run Slurm jobs.

Jobs have been modeled in Tosca and this allows Yorc to execute them, either as simple jobs or as [Singularity](#) jobs.

[Singularity](#) is a container system similar to Docker but designed to integrate well HPC environments. and let users execute a command inside a Singularity or Docker container as a job submission. See [Working with jobs](#) for more information.

Yorc supports the following resources on Slurm:

- Node Allocations as Computes
- Jobs
- Singularity Jobs.

### 8.2.1 Resources based scheduling

TOSCA allows to specify [requirements on Compute nodes](#) if specified `num_cpus` and `mem_size` requirements are used to allocate only the required resources on computes. This allows to share a Slurm managed compute across several deployments. If not specified a whole compute node will be allocated.

Yorc also support [Slurm GRES](#) based scheduling. This is generally used to request a host with a specific type of resource (consumable or not) such as GPUs.

## 8.3 Google Cloud Platform

The Google Cloud Platform integration within Yorc is ready for production and we support the following resources:

- Compute Instances

- Persistent Disks
- Virtual Private Clouds (VPC)
- Static IP Addresses.

### 8.3.1 Future work

It is planned to support soon the following feature:

- Cloud VPN

## 8.4 AWS

The AWS integration within Yorc allows to provision:

- EC2 Compute Instances.
- Elastic IPs.

This part is ready for production but we plan to support soon the following features to make it production-ready:

- Elastic Block Store provisioning
- Networks provisioning with Virtual Private Cloud

### 8.4.1 Future work

- We plan to work on modeling [AWS Batch Jobs](#) in TOSCA and execute them thanks to Yorc.
- We plan to work on [AWS ECS](#) to deploy containers

## 8.5 OpenStack

The [OpenStack](#) integration within Yorc is production-ready. Yorc is currently supporting:

- Compute Instances
- Block Storages
- Virtual Networks
- Floating IPs provisioning.

### 8.5.1 Future work

- We plan to work on modeling [OpenStack Mistral workflows](#) in TOSCA and execute them thanks to Yorc.
- We plan to work on [OpenStack Zun](#) to deploy containers directly on top of OpenStack

## 8.6 Kubernetes

The [Kubernetes](#) integration within Yorc is now production-ready. Yorc is currently supporting the following K8s resources:

- Deployments.
- Jobs.
- Services.

The [Google Kubernetes Engine](#) is also supported as a Kubernetes cluster.

### 8.6.1 Future work

It is planned to support soon the following features:

- Persistent Volume Claims.
- StatefulSets.



---

## TOSCA support in Yorc

---

TOSCA stands for Topology and Orchestration Specification for Cloud Applications. It is an [OASIS](#) standard specification. Currently Yorc implements the version [TOSCA Simple Profile in YAML Version 1.2](#) of this specification.

Yorc is a TOSCA based orchestrator, meaning that it consumes TOSCA definitions and services templates to perform applications deployments and lifecycle management.

Yorc is also workflow driven meaning that it will execute workflows defined in a TOSCA service template to perform a deployment. The easiest way to generate valid TOSCA definitions for Yorc is to use a TOSCA web composer called [Alien4Cloud](#).

Yorc provides an Alien4Cloud (A4C) plugin that allows A4C to interact with Yorc.

Alien4Cloud provides a [great documentation on writing TOSCA components](#).

Bellow are the specificities of Yorc

## 9.1 TOSCA Operations

### 9.1.1 Supported TOSCA functions

Yorc supports the following functions that could be used in value assignments (generally for attributes and operation inputs).

- `get_input: [input_name]`: Will retrieve the value of a Topology's input
- `get_property: [<entity_name>, <optional_cap_name>, <property_name>, <nested_property_name_or_index_1>, ..., <nested_property_name_or_index_n>]`: Will retrieve the value of a property in a given entity. `<entity_name>` could be the name of a given node or relationship template, `SELF` for the entity holding this function, `HOST` for one of the hosts (in the hosted-on hierarchy) of the entity holding this function, `SOURCE`, `TARGET` respectively for the source or the target entity in case of a relationship. `<optional_cap_name>` is optional and allows to specify that we target a property in a capability rather directly on the node.

- `get_attribute`: [`<entity_name>`, `<optional_cap_name>`, `<property_name>`, `<nested_property_name_or_index_1>`, ..., `<nested_property_name_or_index_n>`]: see `get_property` above
- `concat`: [`<string_value_expressions_*>`]: concatenates the result of each nested expression. Ex: `concat`: [ "http://", `get_attribute`: [ `SELF`, `public_address` ], ":", `get_attribute`: [ `SELF`, `port` ] ]
- `get_operation_output`: [`<modelable_entity_name>`, `<interface_name>`, `<operation_name>`, `<output_variable_name>`]: Retrieves the output of an operation
- `get_secret`: [`<secret_path>`, `<optional_implementation_specific_options>`]: instructs to look for the value within a connected vault instead of within the Topology. Resulting value is considered as a secret by Yorc.

### 9.1.2 Supported Operations implementations

Currently Yorc supports as builtin implementations for operations:

- Bash scripts
- Python scripts
- Ansible Playbooks

New implementations can be plugged into Yorc using its plugin mechanism.

### 9.1.3 Execution Context

Python and Bash scripts are executed by a wrapper script used to retrieve operations outputs. This script itself is executed using a `bash -l` command meaning that the login profile of the user used to connect to the host will be loaded.

**Warning:** Defining operations inputs with the same name as Bash reserved variables like `USER`, `HOME`, `HOSTNAME` and so on... may lead to unexpected results... Avoid to use them.

### 9.1.4 Injected Environment Variables

When operation scripts are called, some environment variables are injected by Yorc.

- For Python and Bash scripts those variables are injected as environment variables.
- For Python scripts they are also injected as global variables of the script and can be used directly.
- For Ansible playbooks they are injected as [Playbook variables](#).

### 9.1.5 Operation outputs

TOSCA defines a function called `get_operation_output`, this function instructs Yorc to retrieve a value at the end of a operation. In order to allow Yorc to retrieve those values you should depending on your operation implementation:

- in Bash scripts you should export a variable named as the output variable (case sensitively)
- in Python scripts you should define a variable (globally to your script root not locally to a class or function) named as the output variable (case sensitively)

- in Ansible playbooks you should set a fact named as the output variable (case sensitively)

## Node operation

For node operation script, the following variables are available:

- **NODE**: the node name.
- **INSTANCE**: the unique instance ID.
- **INSTANCES**: A comma separated list of all available instance IDs.
- **HOST**: the node name of the node that host the current one.
- **DEPLOYMENT\_ID**: the unique deployment identifier.

In addition, any inputs parameters defined on the operation definition are also injected.

## Relationship operation

For relationship operation script, the following variables are available:

- **TARGET\_NODE**: The node name that is targeted by the relationship.
- **TARGET\_INSTANCE**: The instance ID that is targeted by the relationship.
- **TARGET\_INSTANCES**: Comma separated list of all available instance IDs for the target node.
- **TARGET\_HOST**: The node name of the node that host the node that is targeted by the relationship.
- **SOURCE\_NODE**: The node name that is the source of the relationship.
- **SOURCE\_INSTANCE**: The instance ID of the source of the relationship.
- **SOURCE\_INSTANCES**: Comma separated list of all available source instance IDs.
- **SOURCE\_HOST**: The node name of the node that host the node that is the source of the relationship.
- **DEPLOYMENT\_ID**: the unique deployment identifier.

In addition, properties and attributes of the target capability of the relationship are injected automatically. We do this as they can only be retrieved by knowing in advance the actual name of the capability, which is not very practical when designing a generic operation. As a target component may have several capabilities that match the relationship capability type we inject the following variables:

- **TARGET\_CAPABILITY\_NAMES**: comma-separated list of matching capabilities names. It could be used to loop over the injected variables
- **TARGET\_CAPABILITY\_<capabilityName>\_TYPE**: actual type of the capability
- **TARGET\_CAPABILITY\_TYPE**: actual type of the capability of the first matching capability
- **TARGET\_CAPABILITY\_<capabilityName>\_PROPERTY\_<propertyName>**: value of a property
- **TARGET\_CAPABILITY\_PROPERTY\_<propertyName>**: value of a property for the first matching capability
- **TARGET\_CAPABILITY\_<capabilityName>\_<instanceName>\_ATTRIBUTE\_<attributeName>**: value of an attribute of a given instance
- **TARGET\_CAPABILITY\_<instanceName>\_ATTRIBUTE\_<attributeName>**: value of an attribute of a given instance for the first matching capability

Finally, any inputs parameters defined on the operation definition are also injected.

### Attribute and multiple instances

When an operation defines an input, the value is available by fetching an environment variable. If you have multiple instances, you'll be able to fetch the input value for all instances by prefixing the input name by the instance ID.

Let's imagine you have an relationship's configure interface operation defined like this:

```
add_target:
  inputs:
    TARGET_IP: { get_attribute: [TARGET, ip_address] }
  implementation: scripts/add_target.sh
```

Let's imagine we have a node named MyNodeS with 2 instances: MyNodeS\_1, MyNodeS\_2. The node MyNodeS is connected to the target node MyNodeT which has also 2 instances MyNodeT\_1 and MyNodeT\_2.

When the add\_target.sh script is executed for the relationship instance that connects MyNodeS\_1 to MyNodeT\_1, the following variables will be available:

```
TARGET_NODE=MyNodeT
TARGET_INSTANCE=MyNodeT_1
TARGET_INSTANCES=MyNodeT_1,MyNodeT_2
SOURCE_NODE=MyNodeS
SOURCE_INSTANCE=MyNodeS_1
SOURCE_INSTANCES=MyNodeS_1,MyNodeS_2
TARGET_IP=192.168.0.11
MyNodeT_1_TARGET_IP=192.168.0.11
MyNodeT_2_TARGET_IP=192.168.0.12
```

## 9.1.6 Orchestrator-hosted Operations

In the general case an operation is an implementation of a step within a node's lifecycle (install a software package for instance). Those operations should be executed on the Compute that hosts the node. Yorc handles this case seamlessly and execute your implementation artifacts on the required host.

But sometimes you may want to model in TOSCA an interaction with something (generally a service) that is not hosted on a compute of your application. For those usecases the TOSCA specification support a tag called *operation\_host* this tag could be set either on an [operation implementation](#) or on a [workflow step](#). If set to the keyword ORCHESTRATOR this tag indicates that the operation should be executed on the host of the orchestrator.

For executing those kind of operations Yorc supports two different behaviors. The first one is to execute implementation artifacts directly on the orchestrator's host. But we think that running user-defined bash or python scripts directly on the orchestrator's host may be dangerous. So, Yorc offers an alternative that allows to run those scripts in a sandboxed environment implemented by a Docker container. This is the recommended solution.

Choosing one or the other solution is done by configuration see [ansible hosted operations options in the configuration section](#). If a *default\_sandbox* option is provided, it will be used to start a docker sandbox. Otherwise if *unsandboxed\_operations\_allowed* is set to `true` (defaults to `false`) then operations are executed on orchestrator's host. Otherwise Yorc will rise an error if an orchestrator hosted operation should be executed.

In order to let Yorc interact with Docker to manage sandboxes some requirements should be met on the Yorc's host:

- Docker service should be installed and running
- Docker CLI should be installed
- the *pip package* `docker_py` should be installed

Yorc uses standard Docker's APIs so `DOCKER_HOST` and `DOCKER_CERT_PATH` environment variables could be used to configure the way Yorc interacts with Docker.



In order to execute operations on container some minimal requirements should also be met on Docker images used as sandboxes:

- the `/usr/bin/env` command should be present
- a python 2 interpreter compatible with ansible 2.7.2 should be available as the `python` command

apart those above requirements you can install whatever you want in your Docker image as prerequisites of your operations artifacts.

Yorc will automatically pull the required Docker image and start a separated Docker sandbox before each orchestrator-hosted operation and automatically destroy it after the operation execution.

**Caution:** Currently setting `operation_host` on operation implementation is supported in Yorc but not in Alien4Cloud. That said, when using Alien4Cloud workflows will automatically be generated with `operation_host=ORCHESTRATOR` for nodes that are not hosted on a Compute.



---

## Run Yorc in Secured mode

---

To run Yorc in secured mode, the following issues have to be addressed:

- Setup a secured Consul cluster
- Setup a secured Yorc server and configure it to use a secured Consul client
- Setup Alien4Cloud security and configure it to use a secured Yorc server

In the case of Yorc HA setup (see *High level view of a typical HA installation*), all the Yorc servers composing the cluster need to be secured.

To secure the components listed above, and enable TLS, Multi-Domain (SAN) certificates need to be generated. A short list of commands based on openSSL is provided below.

### 10.1 Generate SSL certificates with SAN

The SSL certificates you will generate need to be signed by a Certificate Authority. You might already have one, otherwise, create it using OpenSSL commands below:

```
openssl genrsa -aes256 -out ca.key 4096
openssl req -new -x509 -days 365 -key ca.key -sha256 -out ca.pem
```

#### 10.1.1 Generate certificates signed by your CA

You need to generate certificates for all the software component to be secured (Consul agents, Yorc servers, Alien4Cloud).

Use the commands below for each component instance (where <IP> represents IP address used to connect to the component). Replace `comp` by a string of your choice corresponding to the components to be secured.

```
openssl genrsa -out comp.key 4096
openssl req -new -sha256 -key comp.key -subj "/C=FR/O=Atos/CN=127.0.0.1" -reqexts_
↪SAN -config <(cat /etc/pki/tls/openssl.cnf <(printf "[SAN]\nsubjectAltName=IP:127.0.
↪0.1,IP:<IP>,DNS:localhost")) -out comp.csr
openssl x509 -req -in comp.csr -CA ca.pem -CAkey ca.key -CAcreateserial -out comp.pem_
↪-days 2048 -extensions SAN -extfile <(cat /etc/pki/tls/openssl.cnf <(printf
↪"[SAN]\nsubjectAltName=IP:127.0.0.1,IP:<IP>,DNS:localhost"))
```

In the sections below, the `comp.key` and `comp.pem` files path are used in the components' configuration file.

## 10.2 Secured Consul cluster Setup

---

**Note:** You need to generate certificates for all the Consul agents within the Consul cluster you setup.

---

In a High Availability cluster, you need to setup at least 3 consul servers, and one consul client on each host where a Yorc server is running.

Check Consul documentation for details about [agent's configuration](#).

You may find below a typical configuration file for a consul server ; to be updated after having generated the `consul_server.key` and `consul_server.pem` files.

```
{
  "domain": "yorc",
  "data_dir": "/tmp/work",
  "client_addr": "0.0.0.0",
  "advertise_addr": "{SERVER_IP}",
  "server": true,
  "bootstrap": true,
  "ui": true,
  "encrypt": "{ENCRYPT_KEY}",
  "ports": {
    "https": 8543
  },
  "key_file": "{PATH_TO_CONSUL_SERVER_KEY}",
  "cert_file": "{PATH_TO_CONSUL_SERVER_PEM}",
  "ca_file": "{PATH_TO_CA_PEM}",
  "verify_incoming": true,
  "verify_outgoing": true
}
```

And below, a typical configuration file for a consul client.

```
{
  "domain": "yorc",
  "data_dir": "/tmp/work",
  "client_addr": "0.0.0.0",
  "advertise_addr": "{IP}",
  "retry_join": [ "{SERVER_IP}" ],
  "encrypt": "{ENCRYPT_KEY}",
  "ports": {
    "https": 8543
  },
  "key_file": "{PATH_TO_CONSUL_CLIENT_KEY}",
```

(continues on next page)

(continued from previous page)

```

"cert_file": "{PATH_TO_CONSUL_CLIENT_PEM}",
"ca_file": "{PATH_TO_CA_PEM}",
"verify_incoming_rpc": true,
"verify_outgoing": true
}

```

In the above example, the encryption is enabled for the gossip traffic inside the Consul cluster. Check Consul documentation for details [network traffic encryption](#).

You can also consult this [Blog](#). You may find useful information about how to install CA certificate in the OS, in case you get errors about trusting the signing authority.

## 10.3 Secured Yorc Setup

Generate a `yorc_server.key` and `yorc_server.pem` using the above commands and replace `<IP>` by the host's IP address.

Bellow is an example of configuration file with TLS enabled and using the collocated and secured Consul client.

```

{
  "consul": {
    "ssl": "true",
    "ca_cert": "{PATH_TO_CA_PEM}",
    "key_file": "{PATH_TO_CONSUL_CLIENT_KEY}",
    "cert_file": "{PATH_TO_CONSUL_CLIENT_PEM}",
    "address": "127.0.0.1:8543"
  },
  "resources_prefix": "yorc1-",
  "key_file": "{PATH_TO_YORC_SERVER_KEY}",
  "cert_file": "{PATH_TO_YORC_SERVER_PEM}",
  "ssl_verify": true,
  "infrastructures": {
    "openstack": {
      "auth_url": "https://your-openstack:{OPENSTACK_PORT}/v2.0",
      "tenant_name": "your-tenant",
      "user_name": "os-user",
      "password": "os-password",
      "private_network_name": "default-private-network",
      "default_security_groups": ["default"]
    }
  }
}

```

In the above example SSL verification is enabled for Yorc (`ssl_verify` set to `true`). In this case, the Consul Agent must be enabled to use TLS configuration files for HTTP health checks. Otherwise, the TLS handshake may fail. You can find below the Consul agent's configuration:

```

{
  "domain": "yorc",
  "data_dir": "/tmp/work",
  "client_addr": "0.0.0.0",
  "advertise_addr": "{IP}",
  "ui": true,
  "retry_join": [ "{SERVER_IP}" ],
  "encrypt": "{ENCRYPT_KEY}",

```

(continues on next page)

(continued from previous page)

```
"ports": {
  "https": 8543
},
"key_file": "{PATH_TO_CONSUL_CLIENT_KEY}",
"cert_file": "{PATH_TO_CONSUL_CLIENT_PEM}",
"ca_file": "{PATH_TO_CA_PEM}",
"enable_agent_tls_for_checks": true,
"verify_incoming_rpc": true,
"verify_outgoing": true
}
```

As for Consul, you may need to install CA certificate in the OS, in case you get errors about trusting the signing authority.

## 10.4 Secured Yorc CLI Setup

If `ssl_verify` is enabled for Yorc server, the Yorc CLI have to provide a client certificate signed by the Yorc's Certificate Authority.

So, create a `yorc_client.key` and `yorc_client.pem` using the above commands and replace `<IP>` by the host's IP address.

Bellow is an example of configuration file with TLS enabled. Refer to *Yorc Client CLI Configuration* for more information.

```
{
  "key_file": "{PATH_TO_YORC_CLIENT_KEY}",
  "cert_file": "{PATH_TO_YORC_CLIENT_PEM}",
  "ca_file": "{PATH_TO_CA_PEM}",
  "yorc_api": "<YORC_SERVER_IP>:8800"
}
```

## 10.5 Setup Alien4Cloud security

See the corresponding Chapter in Alien4Cloud plugin documentation

---

## Integrate Yorc with a Vault

---

A Vault is used to store secrets in a secured way.

Yorc allows to interact with a Vault to retrieve sensitive data linked to infrastructures such as passwords.

Currently Yorc supports only [Vault from HashiCorp](#) we plan to support others implementations in Yorc either builtin or by plugins.

The vault integration allows to specify infrastructures parameters as [Go Template](#) format and to use a specific function called `secret` this function takes one argument that refers to the secret identifier and an optional list of string arguments whose signification is dependent to the Vault implementation. This function returns an object implementing the `vault.Secret` interface which has two functions `String()` that returns the string representation of a secret and `Raw()` that returns a Vault implementation-dependent object. The second way most powerful but you should look at the Vault implementation documentation to know how to use it.

### 11.1 HashiCorp's Vault integration

HashiCorp's Vault integration is builtin Yorc. Please refer to [the HashiCorps Vault configuration](#) section to know how to setup a connection to a running Vault. For more information about Vault itself please refer to its [online documentation](#).

Here is how the `secret` function is handled by this implementation, the usage is:

```
secret "/secret/path/in/vault" ["options" ...]
```

Recognized options are:

- `data=targetdata`: Vault allows to store multiple keys/values within a map called *Data*, this option allows to render only the key named `targetdata`. Only one data option is allowed.

The `String()` function on the returned secret will render the whole map if there is no data options specified.

The `Raw()` function on the returned secret will return a [github.com/hashicorp/vault/api.Secret](https://github.com/hashicorp/vault/api.Secret).

Bellow are some of the most common ways to get a specific secret using the templating language:

- `{{ with (secret "/secret/yorc/mysecret").Raw }}{{ .Data.myKey }}{{end}}`

- `{{ secret "/secret/yorc/mysecret" "data=myKey" | print }}`
- `{{ (secret "/secret/yorc/mysecret" "data=myKey").String }}`



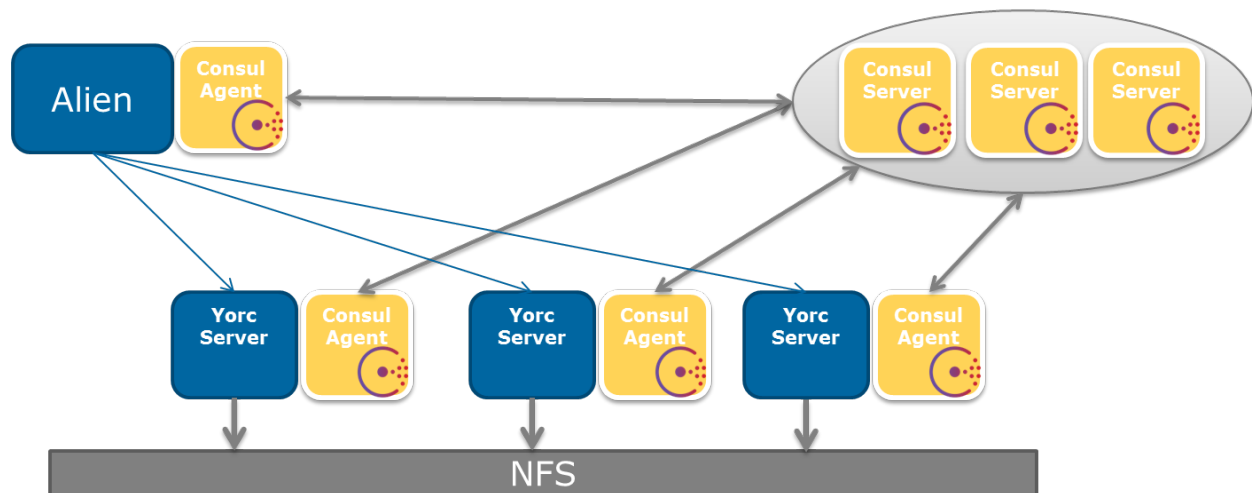
---

## Run Yorc in High Availability (HA) mode

---

### 12.1 High level view of a typical HA installation

The bellow figure illustrates how a typical Yorc setup for enabling High Availability looks like.



This setup is composed by the following main components:

- A POSIX distributed file system (NFS as an example in the figure above) to store deployments recipes (Shell scripts, Ansible recipes, binaries...)
- A cluster of Consul servers
- A cluster of Yorc servers each one collocated with a Consul agent and connected to the distributed filesystem
- A Alien4Cloud with the Yorc plugin collocated with a Consul agent

The next sections describes how to setup those components.

## 12.2 Yorc HA setup

### 12.2.1 Distributed File System

Describing how to setup a Distributed File System (DSF) is out of the scope of this document. When choosing your DSF please take care to verify that it is POSIX compatible and can be mounted as linux partition.

### 12.2.2 Consul servers

To setup a cluster of Consul servers please refer to the [Consul online documentation](#). One important thing to note is that you will need 3 or 5 Consul servers to ensure HA.

### 12.2.3 Yorc servers

Each Yorc server should be installed on its own host with a local Consul agent and a partition mounted on the Distributed File System. The Consul agent should run in client mode (by opposition to the server mode). Here is how to run a Consul agent in client mode and connect it to a running Consul server cluster.

```
consul agent -config-dir ./consul-conf -data-dir ./consul-data -retry-join  
↪<ConsulServer1IP> -retry-join <ConsulServer2IP> -retry-join <ConsulServer3IP>
```

When starting the Yorc server instance, a Consul service is automatically created with a defined TCP check on Yorc port.

When running Yorc you should use the `-server_id` command line flag (or equivalent configuration options or environment variable) to specify the server ID used to identify the server node in a cluster.

When running Yorc you should use the `-working_directory` command line flag (or equivalent configuration options or environment variable) to specify a working directory on the Distributed File System.

### 12.2.4 Alien4Cloud

Please refer to the dedicated Yorc plugin for Alien4Cloud documentation for its typical installation and configuration.

Install and run Consul agent in client mode.

```
consul agent -config-dir ./consul-conf -data-dir ./consul-data -retry-join  
↪<ConsulServer1IP> -retry-join <ConsulServer2IP> -retry-join <ConsulServer3IP> -  
↪recursor <ConsulServer1IP> -recursor <ConsulServer2IP> -recursor <ConsulServer3IP>
```

Configure Consul DNS forwarding in order to be able to resolve `yorc.service.consul` DNS domain name.

In the Yorc plugin for Alien4Cloud configuration use `http://yorc.service.consul:8800` as Yorc URL instead of using a IP address. This DNS name will be resolved by Consul (using a round-robin algorithm) to available Yorc servers.

If a Yorc server becomes unavailable, then Consul will detect it by using the service check and will stop to resolve the DNS requests to this Yorc instance, allowing seamless failover.

---

## Run Yorc in a docker container

---

Along with a Yorc release we also provide a docker image that ships Yorc and its dependencies.

This docker image is published in several ways:

- A `tgz` version of the image is published on the [github release page](#) for each release.
- Pre-release (milestone) versions and development branches are published in [Artifactory](#). You can use it to pull the image directly from the docker command. For instance: `docker pull ystia-docker.jfrog.io/ystia/yorc:3.0.0-M5`
- Starting with Yorc 3.0 GA version are also published on [the docker hub](#). You can use it to pull the image directly from the docker command. For instance: `docker pull ystia/yorc:3.0.0`

### 13.1 Image components

This Docker image is made of the following components:

#### 13.1.1 S6 init system

We use the [S6 overlay for containers](#) in order to have a minimal init system that supervise our services and [reap zombies processes](#).

#### 13.1.2 Consul

[Consul](#) is the distributed data store for Yorc. By default it will run in an isolated server mode to provide a fully functional container out of the box. But it could be configured to connect to a server and run in agent mode.

The Consul binary is installed in this image as described in the [Yorc install section](#). It is run as a user and group named `consul`. Consul data directory is set by default to `/var/consul`.

## Configure Consul

To configure Consul in the container you can either mount configuration files into `/etc/consul` or use environment variables. Following special variables are recognized:

- `CONSUL_MODE`: if set to `server` or not defined then Consul will run in server mode. Any other configuration will lead to Consul running in agent mode.
- `NB_CONSUL_SERVER`: allows to set the `bootstrap-expect` command line flag for consul. If `CONSUL_MODE` is `server` and `NB_CONSUL_SERVER` is not defined then it defaults to 1.
- `SERVERS_TO_JOIN`: allows to provide a coma-separated list of server to connects to. This works either in server or agent mode.

In addition any environment variable that starts with `CONSUL_ENV_` will be added to a dedicated consul configuration file. The format is `CONSUL_ENV_<option_name>=<config_snippet>`. Here are some examples to make it clear:

```
docker run -e 'CONSUL_ENV_ui=true' -e 'CONSUL_ENV_watches=[{"type":"checks",
"handler":"/usr/bin/health-check-handler.sh"}]' -e 'CONSUL_ENV_datacenter="east-aws" '
ystia/yorc
```

Will result in the following configuration file:

```
{
  "ui": true,
  "watches": [{"type":"checks","handler":"/usr/bin/health-check-handler.sh"}],
  "datacenter": "east-aws"
}
```

### 13.1.3 go-dnsmasq

go-dnsmasq is a lightweight DNS caching server/forwarder with minimal filesystem and runtime overhead. It is used in this image to forward any `*.consul` dns request directly to Consul and forward others dns requests to your standard upstream dns server. This allows to support dns resolving of Consul services out of the box.

### 13.1.4 Ansible & Terraform

Ansible and Terraform are installed in this image as described in the *Yorc install section*.

There is no specific configuration needed for those components.

### 13.1.5 Docker

The Docker client binary and the `docker-py` python library are installed in this image as described in the *Yorc install section*.

They are necessary to support *Orchestrator-hosted operations* isolated in a Docker sandbox.

In order to let Yorc run Docker containers you should either expose the Docker service of your host in TCP and configure Yorc to use this endpoint or mount the Docker socket into the container (recommended).

Here is the command line that allows to mount the Docker socket into the Yorc container:

```
# Using the --mount flag (recommended way on Docker 17.06+)
docker run --mount "type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock"
↳ ystia/yorc
# Using the -v flag (for Docker < 17.06)
docker run -v /var/run/docker.sock:/var/run/docker.sock ystia/yorc
```

### 13.1.6 Yorc

The Yorc binary is installed in this image as described in the *Yorc install section*.

Yorc is run as a `yorc` (group `yorc`) user. This user's home directory is `/var/yorc` and the `yorc` process is run within that directory. Yorc's plugins can be added using a mount within the `/var/yorc/plugins` directory.

#### Configuring Yorc

To configure Yorc you can either mount a `config.yorc.[json|yaml]` into the `/etc/yorc` directory or use Yorc standard environment variables (for both cases see *Yorc Server Configuration* section)



# CHAPTER 14

## Yorc Telemetry

Yorc collects various runtime metrics. These metrics are aggregated on a ten second interval and are retained for one minute.

To view this data, you must send a signal to the Yorc process: on Unix, this is USR1 while on Windows it is BREAK. Once Yorc receives the signal, it will dump the current telemetry information to stderr.

Telemetry information can be streamed to both statsite as well as statsd or pull from Prometheus based on providing the appropriate configuration options. See [Telemetry configuration](#) for more information.

Below is sample output (lot of metrics omitted for brevity) of a telemetry dump:

```
[2017-07-19 16:31:00 +0200 CEST] [G] 'yorc.yorc-server-0.runtime.alloc_bytes':  
→73723728.000  
[2017-07-19 16:31:00 +0200 CEST] [G] 'yorc.yorc-server-0.workers.free': 2.000  
[2017-07-19 16:31:00 +0200 CEST] [C] 'yorc.http.200.GET.metrics': Count: 2 Sum: 2.000  
→LastUpdated: 2017-07-19 16:31:06.253380804 +0200 CEST  
[2017-07-19 16:31:00 +0200 CEST] [S] 'yorc.tasks.maxBlockTimeMs': Count: 10 Sum: 0.000  
→LastUpdated: 2017-07-19 16:31:09.805073861 +0200 CEST  
[2017-07-19 16:31:00 +0200 CEST] [S] 'yorc.http.GET.metrics': Count: 2 Min: 27.765  
→Mean: 29.474 Max: 31.183 Stddev: 2.417 Sum: 58.948 LastUpdated: 2017-07-19 16:31:06.  
→253392224 +0200 CEST  
[2017-07-19 16:31:10 +0200 CEST] [S] 'yorc.tasks.maxBlockTimeMs': Count: 10 Sum: 0.000  
→LastUpdated: 2017-07-19 16:31:19.986227315 +0200 CEST  
[2017-07-19 16:31:20 +0200 CEST] [C] 'yorc.http.200.GET.metrics': Count: 2 Sum: 2.000  
→LastUpdated: 2017-07-19 16:31:26.257243322 +0200 CEST  
[2017-07-19 16:31:20 +0200 CEST] [S] 'yorc.tasks.maxBlockTimeMs': Count: 9 Sum: 0.000  
→LastUpdated: 2017-07-19 16:31:29.138694946 +0200 CEST  
[2017-07-19 16:31:20 +0200 CEST] [S] 'yorc.http.GET.metrics': Count: 2 Min: 32.371  
→Mean: 41.727 Max: 51.083 Stddev: 13.232 Sum: 83.454 LastUpdated: 2017-07-19  
→16:31:26.257253638 +0200 CEST
```

## 14.1 Key metrics

### 14.1.1 Metric Types

Type	Description	Quantiles
Gauge	Gauge types report an absolute number at the end of the aggregation interval.	false
Counter	Counts are incremented and flushed at the end of the aggregation interval and then are reset to zero.	true
Timer	Timers measure the time to complete a task and will include quantiles, means, standard deviation, etc per interval.	true

### 14.1.2 Go Runtime metrics

Metric Name	Description	Unit	Metric Type
<code>yorc.runtime.num_goroutines</code>	This tracks the number of running goroutines and is a general load pressure indicator. This may burst from time to time but should return to a steady state value.	number of goroutines	gauge
<code>yorc.runtime.alloc_bytes</code>	This measures the number of bytes allocated by the Yorc process. This may burst from time to time but should return to a steady state value.	bytes	gauge
<code>yorc.runtime.heap_objects</code>	This measures the number of objects allocated on the heap and is a general memory pressure indicator. This may burst from time to time but should return to a steady state value.	number of objects	gauge
<code>yorc.runtime.sys</code>	Sys is the total bytes of memory obtained from the OS. Sys measures the virtual address space reserved by the Go runtime for the heap, stacks, and other internal data structures. It's likely that not all of the virtual address space is backed by physical memory at any given moment, though in general it all was at some point.	bytes	gauge
<code>yorc.runtime.malloc_count</code>	Mallocs is the cumulative count of heap objects allocated. The number of live objects is Mallocs - Frees.	number of Mallocs	gauge
<code>yorc.runtime.free_count</code>	Frees is the cumulative count of heap objects freed.	number of frees	gauge
<code>yorc.runtime.total_gc_pause_ns</code>	PauseTotalNs is the cumulative nanoseconds in GC stop-the-world pauses since the program started. During a stop-the-world pause, all goroutines are paused and only the garbage collector can run.	nanoseconds	gauge
<code>yorc.runtime.total_gc_runs</code>	Gc runs is the number of completed GC cycles.	number of cycles	gauge
<code>yorc.runtime.gc_pause_ns</code>	Latest GC run stop-the-world pause duration.	nanoseconds	timer



### 14.1.3 Yorc REST API metrics

Metric Name	Description	Unit	Metric Type
yorc.http.<Method>.<Path>	This measures the duration of an API call. <Method> is the HTTP verb and <Path> the Path part of the URL where slashes are replaced by dashes.	milliseconds	timer
yorc.http.<Status>.<Method>.<Path>	This counts the number of API calls by HTTP status codes (ie: 200, 404, 500, ...) , HTTP verb and URL path as described above.	number of requests	counter

### 14.1.4 Yorc Workers & Tasks metrics

Metric Name	Description	Unit	Metric Type
yorc.workers.free	This tracks the number of free Yorc workers.	number of free workers	gauge
tasks.maxBlockTimeMs	This measures the highest duration since creation for all waiting tasks.	milliseconds	timer
tasks.nbWaiting	This tracks the number of tasks waiting for being processed.	number of tasks	gauge
tasks.wait	This measures the finally waited time for a task being processed.	milliseconds	timer
task.<DepID>.<Type>.<FinalStatus>	This counts by deployment and task type the final status of a task.	number of tasks	counter
task.<DepID>.<Type>	This measures the task processing duration.	milliseconds	timer

### 14.1.5 Yorc Executors metrics

There are two types of executors in Yorc “delegates executors” and “operations executors”. Delegates executors handle the deployment of Yorc natively supported TOSCA nodes (like an Openstack compute for instance) while Operations executors handle implementations of an lifecycle operations provided as part of the TOSCA node definition (like a shell script or an ansible playbook).

In the below table <ExecType> is the executor type, <DepID> the deployment ID, <NodeType> the fully qualified TOSCA node type where dots where replaced by dashes and <OpName> the TOSCA operation name where dots where replaced by dashes.

Metric Name	Description	Unit	Metric Type
yorc.executor.<ExecType>.<DepID>.<NodeType>.<OpName>	This measures the duration of an execution.	milliseconds	timer
yorc.executor.<ExecType>.<DepID>.<NodeType>.<OpName>.failures	This counts the number of failed executions.	number of failures	counter
yorc.executor.<ExecType>.<DepID>.<NodeType>.<OpName>.successes	This counts the number of successful executions.	number of successes	counter

### 14.1.6 Yorc SSH connection pool

Metric Name	Description	Unit	Metric Type
<code>yorc.ssh-connections-pool.&lt;connection_id&gt;.sessions.open-failed</code>	This tracks the number of failures when opening an SSH session (multiplexed on top of an existing connection).	number of failures	counter
<code>yorc.ssh-connections-pool.&lt;connection_id&gt;.sessions creations</code>	This measures the number of sessions created for a given connection.	number of sessions	counter
<code>yorc.ssh-connections-pool.&lt;connection_id&gt;.sessions.closes</code>	This measures the number of sessions closed for a given connection.	number of sessions	counter
<code>yorc.ssh-connections-pool.&lt;connection_id&gt;.sessions.open</code>	This tracks the number of currently open sessions per connection	number of sessions	gauge
<code>yorc.ssh-connections-pool.creations.&lt;connection_id&gt;</code>	This measures the number of created connections.	number of connection	counter
<code>yorc.ssh-connections-pool.closes.&lt;connection_id&gt;</code>	This measures the number of closed connections.	number of connection	counter

### 15.1 TOSCA Operations

As described in TOSCA *Supported Operations implementations*, Yorc supports these builtin implementations for operations to execute on remote hosts :

- Bash scripts
- Python scripts
- Ansible Playbooks

It is recommended to implement operations as Ansible Playbooks to get the best execution performance.

When operations are not implemented using Ansible playbooks, the following Yorc Server *Ansible configuration* settings allow to improve the performance of scripts execution on remote hosts :

- `use_openssh`: Prefer OpenSSH over Paramiko, a Python implementation of SSH (used by default) to provision remote hosts. OpenSSH have several optimization like reusing connections that should improve performance but may lead to issues on older systems See Ansible documentation on [Remote connection information](#).
- `cache_facts`: Caches [Ansible facts](#) (values fetched on remote hosts about network/hardware/OS/virtualization configuration) so that these facts are not recomputed each time a new operation is a run for a given deployment.
- `archive_artifacts`: Archives operation bash/python scripts locally, copies this archive and unarchives it on remote hosts (requires tar to be installed on remote hosts), to avoid multiple time consuming remote copy operations of individual scripts.



# CHAPTER 16

---

## Upgrades

---

Upgrades of Yorc that preserve already deployed applications are available starting with Yorc 3.1.0. It is safe to upgrade from Yorc 3.0.0 but upgrading from previous releases can lead to unpredictable results.

An upgrade leads to a service interruption. Currently the standard process is to stop all running instances of Yorc. Upgrade dependencies like Consul, Terraform, Ansible for instance, then upgrade Yorc itself and restart it. Yorc will automatically take care of upgrading its database schema by its own from version 3.0.0 up to its current version.

---

**Note:** A rolling upgrade without interruption feature is planned for future versions.

---

## 16.1 Upgrading to Yorc 3.1.0

### 16.1.1 Ansible

Ansible needs to be upgraded to version 2.7.2, run the following command to do it:

```
sudo pip install ansible==2.7.2
```

### 16.1.2 Terraform

Terraform needs to be upgraded to version 0.11.8. Moreover this version comes with a new packaging where providers are not shipped anymore in the main binary. So you also need to download them separately.

```
# Install the new Terraform version
wget https://releases.hashicorp.com/terraform/0.11.8/terraform_0.11.8_linux_amd64.zip
sudo unzip terraform_0.11.8_linux_amd64.zip -d /usr/local/bin

# Now install Terraform plugins
sudo mkdir -p /var/terraform/plugins
```

(continues on next page)

(continued from previous page)

```
wget https://releases.hashicorp.com/terraform-provider-consul/2.1.0/terraform-  
↪provider-consul_2.1.0_linux_amd64.zip  
sudo unzip terraform-provider-consul_2.1.0_linux_amd64.zip -d /var/terraform/plugins  
  
wget https://releases.hashicorp.com/terraform-provider-null/1.0.0/terraform-provider-  
↪null_1.0.0_linux_amd64.zip  
sudo unzip terraform-provider-null_1.0.0_linux_amd64.zip -d /var/terraform/plugins  
  
wget https://releases.hashicorp.com/terraform-provider-aws/1.36.0/terraform-provider-  
↪aws_1.36.0_linux_amd64.zip  
sudo unzip terraform-provider-aws_1.36.0_linux_amd64.zip -d /var/terraform/plugins  
  
wget https://releases.hashicorp.com/terraform-provider-google/1.18.0/terraform-  
↪provider-google_1.18.0_linux_amd64.zip  
sudo unzip terraform-provider-google_1.18.0_linux_amd64.zip -d /var/terraform/plugins  
  
wget https://releases.hashicorp.com/terraform-provider-openstack/1.9.0/terraform-  
↪provider-openstack_1.9.0_linux_amd64.zip  
sudo unzip terraform-provider-openstack_1.9.0_linux_amd64.zip -d /var/terraform/  
↪plugins  
  
sudo chmod 775 /var/terraform/plugins/*
```

### 16.1.3 Consul

Consul needs to be upgraded to version 1.2.3, run the following command to do it:

```
wget https://releases.hashicorp.com/consul/1.2.3/consul_1.2.3_linux_amd64.zip  
sudo unzip consul_1.2.3_linux_amd64.zip -d /usr/local/bin
```

Then restart Consul.

The recommended way to upgrade Consul is to perform a rolling upgrade. See [Consul documentation](#) for details.

### 17.1 BER for SSH private key is not supported

Yorc uses SSH to connect to provisioned Computes or to hosts from hosts pool.

Default behavior is to add related private keys to SSH-agent in order to handle authentication.

But in some cases, SSH-agent can't be used and authentication must be done with private key file with the *-disable\_ssh\_agent* command-line flag

As we use Golang ssh package (<https://godoc.org/golang.org/x/crypto/ssh>) to parse the private key, we don't support BER encoding format (<https://github.com/golang/go/issues/14145>). This kind of format is especially used by Open-Stack Liberty SSH keypair generator.