xOpera, an agile orchestrator

It seems that these days every tool does everything and just a bit more. This tool now solves a specific problem. Adding that tiny bit of more transforms it into a framework. And then, a developer feels that every couple of days, a new toolset is invented, which solves one specific problem splendidly, yet reinvents other parts of the typical software stack to be complete and useful. If the developer is particularly unlucky, the said toolset invents its own ways, APIs to achieve common tasks.

It appears that following the traditional UNIX philosophy of having a tool that does one thing, and does it right, seems to be largely forgotten. Except in rare cases.

One of these is Ansible. While Ansible is the tool of choice for configuration of practically anything with an API, Ansible itself, ironically, has trouble configuring itself. Namely, the playbooks can be reused, however, when it comes to managing elements from other playbooks, we hit a wall. We can, of course, manage and use the elements from different playbooks, however, the solution is rather inelegant. And yes, we are aware this is a design choice, with a significant number of benefits.

On the other hand, as a user, being aware of the design choices and the benefits they bring does not help when we configure complex multi-cloud, container, HPC scenarios. Ansible user needs some help, which should be provided in a similarly no-nonsense way, following the philosophy of small tools which do one thing and do it right.

Let's introduce xOpera - a standards compliant orchestrator which is following the paradigm of having a minimal set of features and is currently focusing on Ansible (and, by provided translators, Chef). With standards compliant, we mean compliance with TOSCA YAML Simple Profile v.1.2+. With a minimal set of features, we denote the above mentioned UNIX philosophy, thus xOpera will do just the orchestration, and do it well. On the licensing side, xOpera is available on GitHub (https://github.com/xlab-si/xopera-opera) under Apache License 2.0.

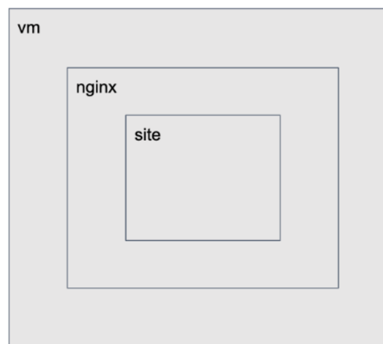
TOSCA stands for the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) standard. It's an industry-developed and supported standard, still lively and fast to adopt new technologies, approaches and paradigms. It's however mostly backwards compatible, so staying within the realm of TOSCA is currently a sound and, from the longevity perspective, a wise decision.

Using the TOSCA as the system-defining language for the xOpera means that we have an overarching declarative way that manages the actual deployment. The Ansible playbooks are now in the role of the actuators, tools that concretise the declared system, its topology and contextualisation of the components and networking.

This design takes the best of both worlds. TOSCA service template is a system definition, written in proverbial stone, while the qualities of the individual Ansible playbooks are now shining. Within the playbooks, we can now entirely focus on particular elements of the overall system, such as provisioning virtual machines at the cloud provider, installing and configuring a service on a target node, etc. xOpera, in its capacity, takes care of all the untidy inter-playbook coordination, state of the deployment and so on.

Let's show this on a simple example, where we deploy a nginx server on OpenStack infrastructure using xOpera with Ansible. The full code of the example is available at the official GitHub account of xOpera: https://github.com/xlab-si/xopera-opera/tree/master/examples.

First, the prerequisites are obviously to have a working OpenStack infrastructure, which we are targeting and to have a working version of xOpera orchestrator, running locally. The instructions for setting up xOpera are available on the front page of our xOpera GitHub account: https://github.com/xlab-si/xopera-opera. Once xOpera is set up, we want to deploy a virtual machine (vm) that contains nginx web server with a simple "Hello World" static web page, on the above-mentioned OpenStack infrastructure.



The process is quite simple. We must define the topology of the targeted infrastructure, with all the required attributes. The example below takes the TOSCA predefined OpenStack node type for building the outer layer (vm) and assigns the required attributes (like name, image, flavour and network). Then, we define nginx as our new node type, which requires previously declaratively built vm as host. Finally, we define a site, which requires nginx (and, transitively, vm). Here, however, we also define how we will create and delete this specific node - these are Ansible playbooks, which are executed by xOpera to actually create or tear down the server. These are stored under interfaces, where you can also see that we build the static web site through the site_config_dir attribute. This excerpt is part of the *service.yaml* file, which is used for deployment.

```
topology_template:
  node_templates:
    vm:
      type: my.nodes.VM.OpenStack
      properties:
        name: nginx_host
        image: 9ea4856a-32b2-4553-b408-cfa4cb1bb40b
        flavor: d3046a41-245a-4042-862e-59568e81f8fa
        network: 753940e0-c2a7-4c9d-992e-4d5bd71f85aa

    nginx:
      type: my.nodes.Nginx
      requirements:
        - host: vm

    site:
      type: my.nodes.Nginx.Site
      requirements:
        - host: nginx
```

```
      interfaces:
        Standard:
          inputs:
            site_config_dir: { get_attribute: [ nginx, site_config_dir ] }
          create: playbooks/site/create.yml
          delete: playbooks/site/delete.yml
```

Now, let's show how deployment looks like - it's rather simple, we merely say *opera deploy mysite service.yaml*. Below you can see the outputs from such an execution. First, vm is created, followed by nginx and then site. Finally, the site gets connected with the nginx through the add_source operation, and we have a working website.

```
$ opera deploy mysite service.yaml
Creating new deployment 'mysite' ...
Deploying vm-db399b47-f270-4f79-bc7f-662fdd1e64df
  Running create
Deploying nginx-4cb0c6a1-a200-4486-914a-7c61f8b0ab50
  Running create
Deploy site-9a05c261-4195-4722-b6f9-0bda762caf0e
  Running create
  Running add_source for nginx-4cb0c6a1-a200-4486-914a-7c61f8b0ab50
DONE
```

What's important to note is that state of the deployment is always stored. Below you can see the two example of what files and actual contents of the stored file.

```
$ tree .mysite.model
.mysite.model/
├── nginx-4cb0c6a1-a200-4486-914a-7c61f8b0ab50.json
├── site-9a05c261-4195-4722-b6f9-0bda762caf0e.json
└── vm-db399b47-f270-4f79-bc7f-662fdd1e64df.json

$ cat .mysite.model/vm-db399b47-f270-4f79-bc7f-662fdd1e64df.json
{
  "tosca_id": "vm-db399b47-f270-4f79-bc7f-662fdd1e64df",
  "tosca_name": "vm",
  "private_address": "",
  "public_address": "10.10.43.207",
  "id": "f45fcb9a-2739-491e-9781-fccd4fcb7a36"
  # More opera specific stuff here
}
```

The undeployment is similarly simple - we just undeploy mysite.

As you can see, the TOSCA definition is verbose but readable. It follows the logic of how the system is built (vm, nginx, the actual site) and the topology used. The Ansible parts are written once, targeting specific infrastructure (e.g. OpenStack, Kubernetes, any kind of HPC system) and from that point on, always reused. xOpera uses them as actuators when following the TOSCA specification.

Given the end user is exposed mostly to TOSCA declarative definitions, it is important to note that, once you get the hang of it, you can reuse many of the declarations or even, generate them from even a more straightforward, more human-readable form. Still, this is a not-too-complicated cornerstone of your deployment - one can always look it up and understand, how the system itself is composed. Ansible playbooks and their bindings are also quite readable and, more importantly, reusable. In the end, once you

have a library of tested and validated playbooks, this part can actually be more or less forgotten, as the focus of the deployment is in the TOSCA and thus orchestrator part. This means we can actually design service templates hand in hand with customers and deploy the so-crafted applications or services at customers from the service template portfolio. Stability of deployment and updates is guaranteed through Ansible's feature of idempotence. We can thus safely run, re-run updates (security updates, configuration changes), which will result in an always-known state, without any of the weird artefacts which are known to occur when deployment halts for whatever reason.

To sum up, with xOpera we can focus on the actual work and reuse of patterns and practices for automation. We will also get the same functionality across various IaaS providers, so our efforts will be multiplied and independent of the actual targets. Currently, xOpera targets OpenStack, with targets such as AWS, Kubernetes, etc, being in the works. Finally, we can get on with the provisioning of the infrastructure, without getting into the Ops' hair too much.