Student Bogdan Shevchenko
Kharkiv National University of Radioelectronics

A Survey On OpenStack Deployment Automation and Scaling.
OpenStack-Ansible as an Automation Tool.

Introduction

OpenStack is a piece of software that enables creating the infrastructure and environment for building private and public clouds by controlling large pools of compute, networking, and storage resources [1]. It is a huge project, that overgrows even the Linux kernel in the lines of code and subprojects [2]. In spite of being initially targeted at both public and private clouds, its success in building private clouds significantly overgrows its success with public clouds. However, it does not matter what particular cloud you are going to build, the scaling of your cloud infrastructure is imminent. Here you have to ways to go: automatic and manual configuration. In the case of tens of compute nodes running, it may be possible to do it manually although being extremely hard and tiresome. So, automation of the deployment process is inevitable. OpenStack-Ansible is one of the first and finest tools that serves the purpose of deployment automation [3].

OpenStack installation has 2 important aspects when being installed: the ability to scale when the workload grows and irreversible changes to the underlying operating system upon which OpenStack is installed [3]. OpenStack has the following core components [4]:

1. Nova - compute;
2. Cinder - storage;
3. Neutron - networking;
4. Swift - object storage (reliable storage);
5. Keystone - authentication and discovery;
6. Glance - image storage;

When working with clouds the main purpose of it is to create VMs - a fundamental and basic thing for all cloud services (XaaS - Something-as-a-Service). A virtual machine mostly depends on two important things - CPU and RAM. Both these resources are given by the Nova component. So, scaling Nova is the task of the first priority.

The next thing that is needed by virtually any VM is network access to download and install all the application software required by the system to operate properly.
The third thing necessary for almost any VM is additional storage space. Depending on the conditions of the storage, a Cinder or Swift can be chosen. There is also a Manila - sharable file storage. The general thing that unites all three storage components is the ability to connect additional block devices to an existing computer or server to be used by OpenStack [5]. Logically, the only reason to scale similar storage nodes is to increase the number of connected devices. Thus, such scalability is essential but not a top priority task.

Another important thing about the OpenStack installation is that it drastically changes the environment of the operating system upon which it is being installed. The main issue about this is that this process is irreversible; you cannot back up to the initial state of the OS. There is a number of solutions to the process of environment alteration, that include [6]:

1. Starting particular service in virtualenv. This is the best approach for Python parts of OpenStack components, mostly because it is lightweight and convenient. Although, the official installation guides do not include information about component installation inside the virtualenv it is the recommended way to do whenever possible [1]. Not all components allow you to do so [1]. It has no significant drawbacks, but it does provide an additional isolation level only for Python whereas OpenStack relies on Linux features and OS real-time state a lot.

2. Starting OpenStack services inside a Virtual Machine (VM) itself. The main advantage is that this solution implies complete control over the OpenStack service environment like VM backups and snapshots. But such an approach has far more disadvantages [1]. To start with, the performance drops significantly. Moreover, nested virtualization does not play well all the time. Also, you have to supply additional disk space to store these backups and snapshots.

3. Running OpenStack services inside Linux containers. This is the preferred way of handling the isolation process. It has literally almost the same performance as virtualenv provides and gives the ability to isolate the environment without running a separate VM [1, 7]. Although kernel features are shared between multiple containers, other host OS aspects remain completely intact.

Obviously, the solution with containers is the most attractive among the alternatives. The official documentation of Linux containers says [8], “That is, containers which offer an environment as close as possible as the one you'd get from a VM but without the overhead that comes with running a separate kernel and
simulating all the hardware”. Current technologies that make up a Linux containerization framework include [8]:

1. LXC.
   “LXC a userspace interface for the Linux kernel containment features. Through a powerful API and simple tools, it lets Linux users easily create and manage system or application containers.”, the documentation says [8]. This project includes bindings for a range of languages (C, Python, Go, Lua, Ruby) and a set of CLI tools. Both CLI tools and programmatical access interact with LXC daemons via Unix socket connection.

2. LXD.
   It is an abstraction over LXC to provide a more unified experience by exposing additional daemon. This daemon includes a single unified CLI tool, which combines all LXC tools together and simplifies the API, and a REST API, used by the LXD to perform actions.

3. LXCFS.
   Linux containers include creating a virtualized container file system. Currently, not all necessary features are provided by the kernel. These features are implemented by LXCFS as a workaround until they will be a part of the kernel. It is worth to mention Docker - “an open platform for developing, shipping, and running applications” [9]. Basically, it adds some more high-level features to LXC, such as images, networks, volumes. It is built upon Go LXC bindings and achieved more popularity than the bare LXC API.

   As you see, the LXC project provides the core features for Linux containerization. Thus, if an application project needs the most performant and flexible way to operate containers, LXC must be used.

   So, an important prerequisite, Linux containerization, was discussed. Now an automatic tool for OpenStack components installation and configuration is needed. There is a wide range of tools for that: Ansible, Rudder, SaltStack, Puppet, Chef, CFEngine, Cake, GitLab. Among them, Ansible is the most attractive due to the project maturity and big community. “Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.”, as per documentation [10]. Ansible executes playbooks - YAML files that define the desired state of the system, not the algorithm of installation or configuration. It relies on modules - routines, included inside Ansible and accessible inside the playbooks. There also exist roles - complex playbooks that have a well-known structure and stored inside the Ansible installation directory.
Now all the prerequisites and components of OpenStack-Ansible are analyzed and now it is possible to discuss this project as a whole. OpenStack-Ansible requirements and dependencies include [11]:

1. Shell scripts for installing Ansible and openstack-ansible CLI tool to simplify the interaction.
2. Ansible playbooks to configure the OS system, Ansible itself, install LXC software, start LXC containers for OpenStack components, install OpenStack components, configure, bootstrap and start the OpenStack cloud.
3. LXC containers for running OpenStack components in an isolated environment.

OpenStack-Ansible (OSA) project started in 2016 by Rackspace so by 2018 it should be mature and capable. According to OpenStack-Ansible architecture reference [11], it is a “batteries-included project” that does not need a lot of end-user intervention. And Rackspace developers implemented the project as documented.

Another feature of the OpenStack-Ansible is All-in-One (OSA AIO). It means installing all the containers for OpenStack components on a single host.

Unfortunately, my experience shows that the project is not very usable due to maintenance issues. The project relies on countless Linux features and aims at supporting a wide range of Linux distributions. Combined diversity of technologies and constant changes in upstream projects the maintenance has become a huge task leading to project obtaining numerous bugs and problems.

The author of the article had tried installing OSA AIO onto a single host running Ubuntu Server 16.04. I had tried twice and both times I had faced issues with Swift and Gnocchi components. These issues were completely unexpected and involved changes into OpenStack-Ansible roles, additional changes to the filesystem, manipulating virtual interfaces and other manual activities that should not occur, according to the documentation [11]. As a whole, I had spent a month trying to install the OSA AIO in two different environments and both tries failed. Even when I tried to omit the errors, the basic Nova functionality did not work.

However, such a pessimism should make you abandon OpenStack for medium and large clouds. There is another project called Kolla-Ansible. It makes use of Docker containers instead of Ansible. According to the article from OpenStack Superuser magazine [12], the installation process is made easier by utilizing Docker containers and leaving low-level details of LXC configuration to the Docker itself. In conclusion, I want to express my belief in the capability of Kolla-Ansible project and the efficiency of all the future projects due to the demand for OpenStack installation automation. Possibly, even OpenStack-Ansible will be revived one day and rewritten or corrected like it was with OpenStack itself. The time will show.

References:


8. https://linuxcontainers.org/

9. https://docs.docker.com/engine/docker-overview/


Manual Deployment And Scaling of Apache Hadoop, OpenStack Clusters is time-consuming. Scaling the infrastructure is complicated due to configuration changes which are required at several places while adding or removing nodes. Maintenance of Hadoop, OpenStack cluster is also cumbersome. To address the challenges and explore the Infrastructure Automation tools Puppet and Ansible as an extensible platform to automate OpenStack and Apache Hadoop. Solution based on Puppet and Ansible for Configuration Management, Deployment, provisioning of Apache Hadoop and OpenStack Cluster for On-Premises and Amazon Web Services Cloud. Automate OpenStack Deployment using Puppet and Ansible.