BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

http://fit.vut.cz

Project practice
2021/2022

Slurm on OpenStack Computing Platform for Development

Jan Swiatkowski

Abstract
The article is about creating a temporary development environment for developing extensions for Slurm batch scheduler. It's rather impossible to easily modify the configuration of the production deployment without affecting other users of HPC systems. Therefore, I created a script to automate the HPC development environment on OpenStack. The main advantage of using the OpenStack is, that it can effortlessly stop or update our cluster stack. I used the Heat Orchestration Templates (HOT) to describe the OpenStack architecture and the Ansible automation tool for installation and configuration. The installation and configuration supports the CentOS 8 Stream, but can be quite simply adapted for other operating systems supported by the Slurm (batch scheduler).

Keywords: Batch Scheduler — OpenStack — Development — HPC — Deployment — Orchestration

Supplementary Material: Source Code *(github.com/jsw0011/slurm-openstack-devel-plugin)*

*xswiat00@fit.vut.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

[Motivation] Preparing a development environment for an application development could be time-consuming for the complex application for use case of HPC. In my case it is the development environment for the Slurm - workload manager. This paper is about using the automation tools such as Ansible and OpenStack Heat Orchestration Template. In the end is possible to create a development environment with few compute nodes just in a few steps.

[Problem definition] I have access to the HPC cluster and I need to develop an extension for a batch scheduler or test various configuration. In my case the Slurm. The main problem is, that is impossible to easily modify the configuration of the production deployment without affecting other users of HPC systems. Therefore, we need to have a separate instance of the batch scheduler, just for development and testing purposes. A good practice in software engineering is to have a consistent development environment, but installation of the batch scheduler is a time-consuming and complicated task. Thanks to modern virtualization technologies, it is possible to automate such task without difficulty.

[Existing solutions]

The Containers *(e.g. Docker)* The docker is fine for local working environment. It can run on a single machine *(bare metal or virtual)*. But on the other hand, it can't simulate true HPC environment. Furthermore, the limitations of the host machine are a disadvantage.

Puppet *[7, Puppet]* The Puppet is one of *the DevOps* utilities, which use the master - slave design to configure and maintain multiple servers at one time. The configuration is written in a file using declarative language. The slaves can use agents to check and set new updates, whenever the master sends the new ones.

The CSC - Slurm Ansible Role *[2, Documentation]* The CSC is a Finnish HPC center operating several HPC and Cloud systems. They created a Slurm deployment script with Ansible role to install and set up the Slurm on *production* HPC clusters. Their role configuration contains a lot of modifiable variables and is almost generic. They provide variables to configure

e.g. backup servers, type of authentication, billing and more.

[Our solution] I created a HOT template for OpenStack to smoothly set up the infrastructure. After setting up the infrastructure a script which runs Ansible roles is executed. The script installs the Slurm with simplified configuration for development purposes on the machines. While creating a stack, user is prompted with a short form with properties. After filling out the form, all the things are set up automatically. The script is ease to modify for custom aims.

[Contributions] This work can help the Slurm plugins developer to make the development easier.

2. Concept

I would like to start with description of used tools for creating the environment. There are 3 options how to set up the architecture on OpenStack platform, what I chose. The first option is the graphical user interface *(GUI)* on the web page where it is possible to set up each component one by one. The second is the terminal utility communicating with OpenStack application programming interface *(API)* and the last is the Heat Orchestration Template (HOT) to set up all components at one. I chose the HOT in combination with the GUI since it is the most suitable option for the task. User does not need to set up the terminal util, and he can just generate the HOT by the shell or python script, or modify it manually, then upload the HOT by the GUI to OpenStack and fill out few required properties.

For the Slurm installation I used Ansible - the tool for configuration of servers. Not only it has a huge advantage of composing task in playbooks and roles, it also uses a smart templating engine Jinja2. This makes it more readable and better maintainable in comparison e.g. the basic bash script. More about the Jinja2 can be found in [5, Chapter 8. Complex Playbooks].

The Slurm installation is a complex task with many steps. Slurm can also be installed directly from the package repositories for Debian, Ubuntu, Fedora, NetBSD, FreeBSD and CentOS 7. In infrastructure, I use, the prefered choice is the CentOS, but I decided to use the CentOS 8 Stream instead of the old version 7, which is almost and of the lifetime. For this version the official repositories do not contain the installation package, so I was forced to do the compilation and installation from the source code. I discovered a little difference between the installation on the CentOS 7 and CentOS 8 mainly in the Slurm dependencies. In the end I was able to manage whole stack effortlessly in GUI of OpenStack platform.

```
compute1:
 type: OS::Nova::Server
   properties:
    flavor: 'normal'
    image: { get_param: def_image }
    key_name: { get_param : key_1 }
    networks:
    - network: { get_param: net_1 }
```

Figure 1. HOT - Nova server

2.1 OpenStack

The OpenStack is a platform for managing sets of interrelated components like compute pools, networking and storages. It provides a web GUI and terminal utility to manage the sources and services. User can set up limits and quotas for sources like IP addresses, CPUs, or number of compute instances. The platform also offers various operations with storage, like snapshoting. The main functionality for my work is the stack creation. User can specify the targeted setup of architecture in the file by HOT and then OpenStack creates it as one group of components, which can be updated or deleted with one click. More about OpenStack architecture can be found in chapter 4 in [6, Containers in OpenStack]

2.2 Heat Orchestration Template

The Heat is an orchestration engine of OpenStack. In short, the Heat Orchestration Template is a structured *yaml* file describing the resources. The resources could be of various types e.g. an IP address, a virtual machine *(a part of the Nova component)*, or a storage volume. The Heat creates a dependency tree and deploys the infrastructure. The HOT has useful variables and methods to pass the properties of the services into another one. I also used the boot script functionality which allows to run a script after the first boot in OS of the Nova service. An example of the Nova server properties in HOT 1.

The HOT properties are:

- `heat_template_version`
- `description`
- `parameters`
- `resources`
- `outputs`

Heat template version - depending on the selected version, the Heat engine supports various functionalities e.g. the loop generated parameters. It's important to check which version is supported by OpenStack installation.

Parameters - are declared variables. The variables can be filled by the users while creating a stack.
Resources - specifies OpenStack Services like Nova compute service.
Outputs - store and print defined values after stack creation.

2.3 Ansible

Ansible is a useful automation tool. It's written in python, so it can be easily installed by *pip* in your system. It can manage tasks such as the continuous deployment or zero downtime rolling updates with ease. The setup of commands can be executed by command in terminal `'ansible-playbook playbook.yml'`. I would like to clarify some terms regarding Ansible.

Task is a set of commands.
Inventory is a group of defined variables for use in playbook, roles and tasks.
Role is a set of tasks to e.g. setup proxy server.
Playbook is a file that specifies to Ansible what set of roles are to run on which machine.

```
playbooks
|___ playbook.yml
|___ inventory.yml
|___ roles
 |___ proxy
  |---- README.md
  |---- defaults
  |   |___ main.yml
  |---- files
  |---- handlers
  |   |___ main.yml
  |---- meta
  |   |___ main.yml
  |---- tasks
  |   |___ main.yml
  |---- templates
  |---- tests
  |   |___ inventory
  |   |___ test.yml
  |___ vars
   |___ main.yml
```

Figure 2. Basic Ansible folder structure

The basic folder structure is described in the figure No. 2. The structure contains a playbook file `playbook.yml`, an inventory file `inventory.yml` and role *proxy*. There are plenty of possibilities in playbook such as execute a pre-task, then the roles and a post-task *(restart a system-V service)*.

```
- name: setup firewalld ports
  become: True
  ansible.posix.firewalld:
    port: 7321/tcp
    permanent: True
    zone: public
    state: enabled
```

Figure 3. Opening the firewall port

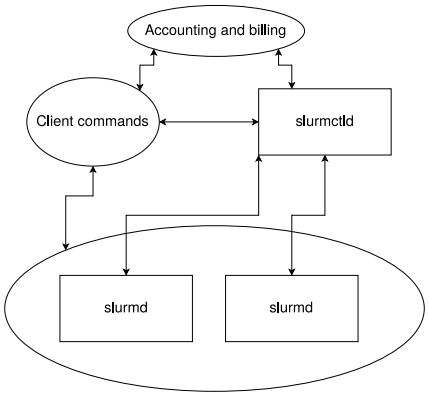

Figure 4. Slurm - basic architecture overview

Tasks have a special syntax in Ansible. They are written in `yaml` and for the most of the basic commands, the Ansible has a handler implemented directly in the core of the tool. But there are also many community addons. As an example I chose opening the firewall port in the figure No. 3.

2.4 Slurm - Workload Manager

Slurm is an open source highly scalable cluster management and job scheduling system for Linux clusters. As a workload manager it provides resources allocation or sharing, starting, executing and monitoring of jobs on the nodes. It also manages and solves resources requests. Optional functionalities are accounting and billing, advanced reservation or multifactor job prioritization. The Slurm has the master-slave architecture, which is described in diagram 4. It is composed of two main daemon types, `slurmctld` i.e. controller daemon, and `slurmd` i.e. compute node daemon. The controller deamon can be in the setup twice, as the backup. There are other optional daemons e.g. for accounting and billing, which I mentioned above, or REAST API daemon.

The configuration file of the Slurm is located in `/etc/slurm.conf`. The compute nodes should have the same configuration file as the controller node, or they can get the configuration directly from the master through a network. The configuration file should describe the compute nodes and their resources and IP addresses, and the controller, and optionaly the backup

controller IP addresses.

3. Solution

I targeted to have a no time-consuming automated solution for setting up a development environment for Slurm on OpenStack platform. But also the key attribute was to have it on the CentOS 8 - Stream, as the future proof solution. This makes it a little bit more complicated, then on CentOS 7 regarding the installation part. I prepared a modifiable Slurm configuration file, where the properties can be changed, excluding the compute nodes and controller node variables. They are automatically generated. When creating a development environment I just need to set up how many machines I would like to use in HOT.

3.1 Heat Orchestration Template

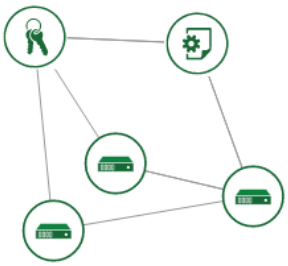

Figure 5. OpenStack - Slurm stack *(screenshot of the GUI)*

For stack creation in OpenStack I wrote the HOT named `slurm-company.yml`. In the figure 5 is an example stack, which I created by the template. There was a resource that was required, where the nodes will comunicate, SSH key for installation and the boot script.

```
metadata:
 ips:
  compute1: {
   get_attr: [
     compute1,
     first_address
    ]
  }
  compute2: {
    get_attr: [
     compute2,
     first_address
    ]
  }
```

Figure 6. HOT - Passing IP adresses to the metadata

The HOT template allows to pass the attributes and parameters as the metadata to the OS Nova, so there was no complication while passing the IP adresses of the compute nodes to the controller node as can be seen in the figure 6. The complication came with the SSH key. The Heat doesn't allow such long strings as the private key. My solution was to pass the private key into the boot script of the controller node and the public key to the boot script of the compute nodes. For ability to pass the metadata, the Heat creates a dependency tree for itself and deploys the sources with respect to the tree.

The boot script installs Python and Git. It retreives and parses the metadata from OpenStack API. It downloads the ansible playbook and files. The hostnames and IP adresses of the node are appended to the `/etc/hosts` file for communication. It configures the SSH key for instalation. It also sets up the nodes adresses into the inventory of Ansible. And runs Ansible playbook.

3.2 Installation and configuration (Ansible)

The installation of Slurm requires the following steps:

- Dependency installation from the package manager.
- Generation of the Munge key.
- Ensures the right permissions to the files and directories.
- Starts the Munge system-V service.
- Gets the Slurm source code.
- Compiles the Slurm.
- Installs the Slurm.
- Creates a *slurmctld* or a *slurmd* system-V service.
- Generates the noded and controller configuration to the `slurm.conf`.
- Opens the firewall ports for Slurm communication.
- Starts and enables the Slurm system-V service

The Munge authentication is a service for creating and validating credentials used by the Slurm. It allows the processes to authenticate the GID and UID within the hosts having common users and groups. These hosts form a security realm that is defined by a shared cryptographic key. Therefore, the Munge key should be spreaded to the compute nodes, after generation in the controller node.

The Slurm compilation and installation is done by the *make* tool from the source code cloned from the git repository. The installation is done in a temporary folder.

Slurm configuration file is concatenated with generated compute nodes configuration, which contains IP address, node name, and CPU count. The CPU count is dynamically retrieved from the compute node by the SSH. It is, because I added possibility to select from the avaliable OS Nova flavors in OpenStack, while creating the stack.

Slurm services are prepared in files and retrieved with Ansible playbook. They are just copied to the proper directory. After all, the *slurmd* service is enabled and started on the compute nodes, and the *slurmctld* is enabled and started on the controller node.

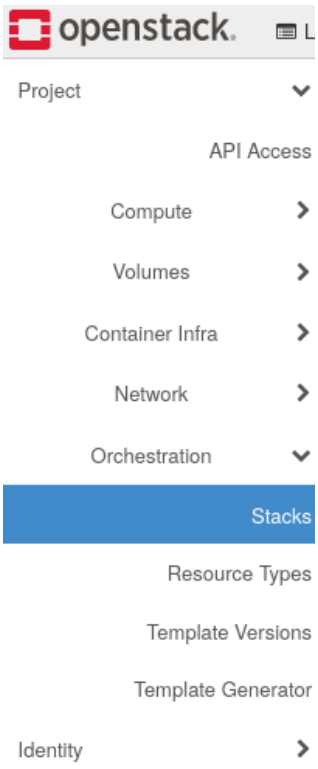

Figure 7. OpenStack - Menu

3.3 Guide to run the development environment

To successfully create the environment, the following steps should be followed:

1. Get the `slurm-company.yml` from the project repository and add as many compute nodes as needed.
2. Go to OpenStack GUI. Sign in. Select the Orchestration in the left menu bar, as shown in the figure 7.
3. Click on the butteon *Launch Stack*. Select the `slurm-company.yml` from your computer, or put an url address. Click on the *Next* button.

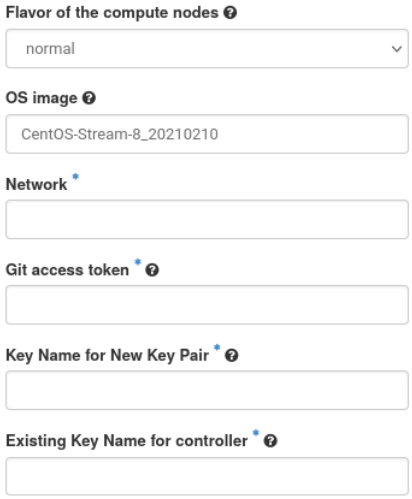

Figure 8. OpenStack - stack form fields

4. Fill the stack name, user password and additional fields 8.
5. Wait until the installation is done

After installation the controller node is accessible by the SSH key specified in the form field called *Existing Key Name for controller* 8.

4. Use cases

The script is specialised for the Slurm plugin development and testing the Slurm configuration, but can be also used for other conceptually similar use cases like testing an application, which requires multiple machines to run. It can be used also for deployment.

4.1 Slurm plugin development

Slurm has two types of plug-ins. The first type is the SPANK plug-in type. The advantage is, it can be compiled just against the Slurm's *spank.h* header file without access to the source code of the Slurm. The SPANK can be loaded at runtime during the next job launch. It just needs to be defined in the configuration file to be loaded *(plugstack.conf)*. The SPANK plugins provied an interface for adding new options for slurm commands *srun*, *salloc* and *sbatch*. That means, the SPANK plugins can for example add dynamically new options for setting up the job priority.

The second type of plug-ins requires the compilation with the Slurm's source code. They should be located in folder *src/plugins*. The Slurm installation then needs to be autoreconfigured and compiled by automake tools.

The plugins use the shared internal data structures of the Slurm and the Slurm API. Each plugin should be identified by a *plugin_name*, *plugin_type (major*

or minor), and a *plugin_version*, and should contain required functions *int init(void)* and *void fini(void)*.

4.2 Development of Third-Party Adapter

As a use case, for which I used the development environment for, was to develop a third-party library as an adapter for a more complex application. I decided to use the Slurm extension for it, the **REST API** accessible through Internet. REST API have endpoints for all usual use cases similar to terminal approach. Thanks to the REST API, the usability of Slurm improves to a new level.

In our case, in our work group, we can use the REST API to call custom Slurm's plugin, which is implemented in separate VMs and it will manage data oriented services. The idea was to be able to prepare data to BurstBuffers, allocate required nodes for computations and various other useful tasks before the workflow starts on HPC. Therefore, I got the task to create a package in Go (programming language), that can be used later directly for Yorc, Ystia TOSCA orchestrator [8], and it may be used as a concept for later extensions of Heappe [4], middleware for access to the HPC infrastructure.

4.2.1 Expose the REST API

The REST API for HTTP communication between Slurm and some HTTP client is an extension and Slurm does not require it for its basic functionality. Therefore, it is implemented by Slurm as a plugin. The service itself is called `slurmrestd`. The way to enable the Slurm's REST API is not very straightforward and it requires some dependencies and libraries. It also requires to have an account database. The account database could be MySQL database or MariaDB database. I chose MariaDB for it. Slurm uses `slurmdbd` service to access the database where the accounts information is stored. A new user must be created for the `slurmrestd` service and that user needs to be granted with access privileges for the `slurm_acct_db` database. The user should be usually unprivileged and the username should be passed as an argument at the start of `slurmrestd`.

The service `slurmdbd` has its own configuration file `slurmdbd.conf` located in directory `/usr/local/etc/`. Information about the host machine of the database (IP address), login information (username, password) and name of database on server will be set in the configuration file. It is also recommended to set a password for the user *root* to restrict access to the database.

Before compilation of Slurm source code, the following libraries should be installed on the host system with header files for development.

- json-c-devel *(JSON format parser)*
- libjwt-devel *(JWT authorization [1])*
- http-parser *(HTTP parser http-parser.git)*
- libyaml *(YAML format parser libyaml.git)*

When requesting Slurm's REST API, whe JWT token and username should be passed in HTTP header. The JWT token can be obtained with following command: `scontrol token username=$USER`. As described in [3].

The whole proccess that is mentioned above, is implemented in an automation script to deploy Slurm development environment to OpenStack, which can be found in GIT repository.

4.2.2 The adapter for more complex application

I implemented an example code that shows how to call Slurm's REST API from Go language. I chose the Go language, because the target application Ystia is written in it. The sample is there to present a preparatory call for staging of dataset slices and creation of session, which will be used later in the workflow.

5. Conclusions

[Paper Summary]

- Automation tools are useful for the development.
- OpenStack's functionality for orchestration of the stack can be used for setting up the development environment
- The environment is now ready for development.

[Paper Contributions] This paper aims to prepare the development environment for Slurm plugin development. The task was done by using the automation tools Ansible and OpenStack orchestration. The product of this work offers a possibility to create a stack of Nova servers instances dedicated for development. The creation can be done in few steps mentioned in the guide 3.3.

[Future Work] The work can be improved for example by snapshoting storages of the virtual machines for speeding up the deployments. The automation script can be also made more generic and easier to use. Generation of basic structure of plugins can be added. The automated compilation of the code and so on.

Acknowledgements

I would like to thank my supervisor Ing. Martin Golasowski, Ph.D. and doc. Ing. Jiří Jaroš, Ph.D. from IT4Innovations for their help.

References

[1] *RFC 7519. JSON Web Token (JWT)*. Internet Engineering Task Force (IETF), 2015. ISSN 2070-1721.

[2] CSC - IT CENTER FOR SCIENCE. *The CSC - Slurm Ansible Role*. January 2022. Documentation and Source Code, Home page: https://www.csc.fi/en/home. Available at: https://github.com/CSCfi/ansible-role-slurm.

[3] GNU. *Slurm - JWT configuration*. 2022. Available at: https://slurm.schedmd.com/jwt.html.

[4] HEAPPE (IT4INNOVATIONS). *Middleware for access to the HPC infrastructure*. 2022. Available at: https://heappe.eu/web/.

[5] HOCHSTEIN, L. and MOSER, R. Advanced templating. In: *Ansible: Up and Running*. 2nd ed. O'Reilly Inc., 2017, chap. 8th - Complex Playbooks. ISBN 978-1491979808.

[6] MADHURI KUMARI, P. K. S. OpenStack Architecture. In: *Containers in OpenStack*. 1st ed. Packt Publishing, 2017, chap. 4th - OpenStack Architecture. ISBN 9781788394383.

[7] PUPPET. *About Puppet*. January 2022. Available at: https://puppet.com/.

[8] YSTIA. *Yorc is a hybrid cloud/HPC TOSCA Orchestrator*. 2022. Available at: https://github.com/ystia/yorc.