Technical Disclosure Commons

Defensive Publications Series

March 19, 2018

CloudSlurm: a multi-provider approach for HPC in the cloud

Glaucimar Da Silva Aguiar
Hewlett Packard Enterprise

Andrey Elision Monteiro Brito
Hewlett Packard Enterprise

Flávio Fonseca
Hewlett Packard Enterprise

Luís Eduardo Tenório Silva
Hewlett Packard Enterprise

Lucas Lima Vieira
Hewlett Packard Enterprise

CloudSlurm: a multi-provider approach for HPC in the cloud

This disclosure relates HPC clusters elasticity using cloud nodes. A framework is disclosed that implements an integration of HPC clusters based on the Slurm workload manager and a cloud infrastructure. The main goal is to be transparent for the Slurm user, who may not be familiar with cloud technologies or with a provider-specific API.

This solution uses a similar architecture to the one described by [1], with a gateway and OpenVPN for connecting the headnode with the cloud. However, that approach considers wrappers for submitting jobs to the workload manager while the approach disclosed here keeps it transparent to the user and requires no additional commands. In addition, this solution installs OpenVPN server in the on-premise headnode instead of the gateway as that above-mentioned approach. This facilitates integrating with multiple cloud providers at the same time by making it easier to manage, enabling even the creation of different partitions that are each assigned to a specific cloud provider (e.g., due to the capabilities or cost trade-offs that different providers may offer).

Another approach to cloud bursting in HPC is described by [2]. Besides being targeted at OpenStack clouds only, it requires manual intervention from the specialist team and, thus, does not scale.

The approach described here layers the foundations for the development of a tool for automated setup of multi-cloud, multi-provider solutions that are simple – but still flexible – to the operator, transparent for the final user, and extendable to implement more efficient solutions in different cloud providers.

CloudSlurm overall solution can be found in Fig. #1. In this approach, besides the typical roles of the head node in the local cluster, it is also responsible for the following activities: (i) it is the VPN endpoint of the communication with the remote nodes; (ii) it hosts the definitions of the cloud-enabled partitions, including the flavours and base images; (iii) it hosts and triggers the scripts that will start or terminate the cloud resources.

On the other end, there is a special node in the cloud provider side, the gateway (identified by GT in Fig. #1). The gateway is the other endpoint of the VPN connection. By using a gateway node, the Slurm cloud slaves are isolated from the Internet and make them part of the local compute node group (see Fig. #2 for details). The configuration and usage of CloudSlurm is detailed below and serves also to detail the components involved and the hooks for extension and customization. The installation is done through a sequence of steps executed using the CLI clients provided by the developed framework, detailed below.

- CloudSlurm installation: requires the name of the headnode, the location of Slurm configurations inside that node and the address of the network used for the local computing hosts. It will install the base scripts (Python executables and Ansible recipes).
- Cloud provider configuration: creates the configurations for the cloud provider, in the case of the OpenStack plugin, it requires the authentication URL and

the names for the region, domain, project, and user to associate with Slurm. Different cloud providers would require different plugins to be created in CloudSlurm application, with potentially different parameters (e.g., no domain name would be necessary).

- OpenVPN configuration: install OpenVPN locally and creates configuration files for a private network address to be used by the cloud hosted workers. By using the OpenVPN server on the headnode, it enables easier extensions to have different partitions on different cloud providers.
- Cloud gateway configuration: receives a host name for the gateway, a network name for the private network in the cloud provider, plus the configurations for the network (netmask, DNS, IP) and cloud instance (flavour, base image, configuration script). This step also uses the specific plugin of the cloud provider, as the creation of networks and the usage of configuration scripts can be specific to the provider.
- Cloud-specific Slurm configuration: configures the cloud integration scripts, especially the timer for terminating idle cloud nodes and the timer for issuing errors on the creation of cloud nodes. Fig. #3 contains details on slurm.conf file changes and Fig #4 and Fig #5 the scripts used to create and terminating VMs on an OpenStack cloud.
- Addition of a new (cloud-based) partition: creates a new partition, configuring the underlying Slurm. The parameters are the name of the partition, a regular expression for the names of the nodes (e.g., cloudnode[11-20]), a regular expression for the IP range (e.g., 192.168.1.[11-20]), and a name for the machine flavour (specifying the configuration of the virtual machines) and the name of the base operating system image.
- Creation of a base image in the cloud provider: creates a base machine that will be automatically configured to generate a base operating system image in the cloud provider. When the machine is configured, a snapshot is taken to host this base image in the cloud provider. This step uses the cloud provider plugin to instantiate a machine and then installs the requirements using Ansible. Currently, only the bare minimum is installed, but customizations can be easily done between the base installation and the creation of the snapshot.

Once the steps enumerated above have been executed, the usage of cloud resources and Slurm commands are transparent. For example: `srun -n10 -p mycloudpartition hostname`, would execute the Linux hostname command in ten nodes in the partition named mycloudpartition. The command would be the same regardless of the partition being based on cloud or on-premise nodes.

[1] Bright Computing, “Bright Computing Cluster Manager for HPC”, http://www.brightcomputing.com/product-offerings/bright-cluster-manager-for-hpc, as of September 2017.
[2] Intel Corporation, “Enabling High Performance Computing in the Cloud Solution Brief”, http://www.intel.com/content/www/us/en/high-performance-computing/enabling-high-performance-computing-in-the-cloud.html, as of September 2017.

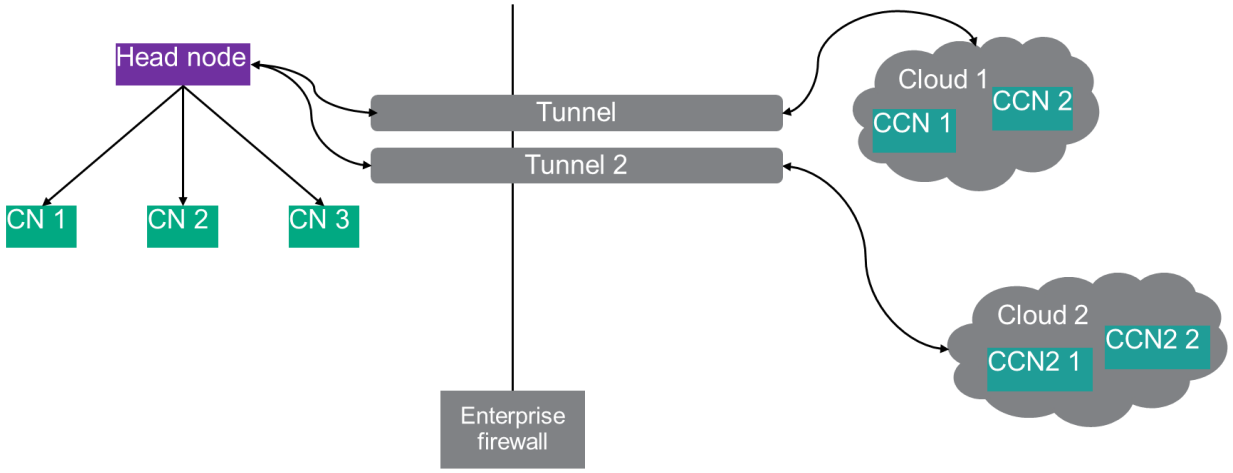

Figure 1 - overall solution

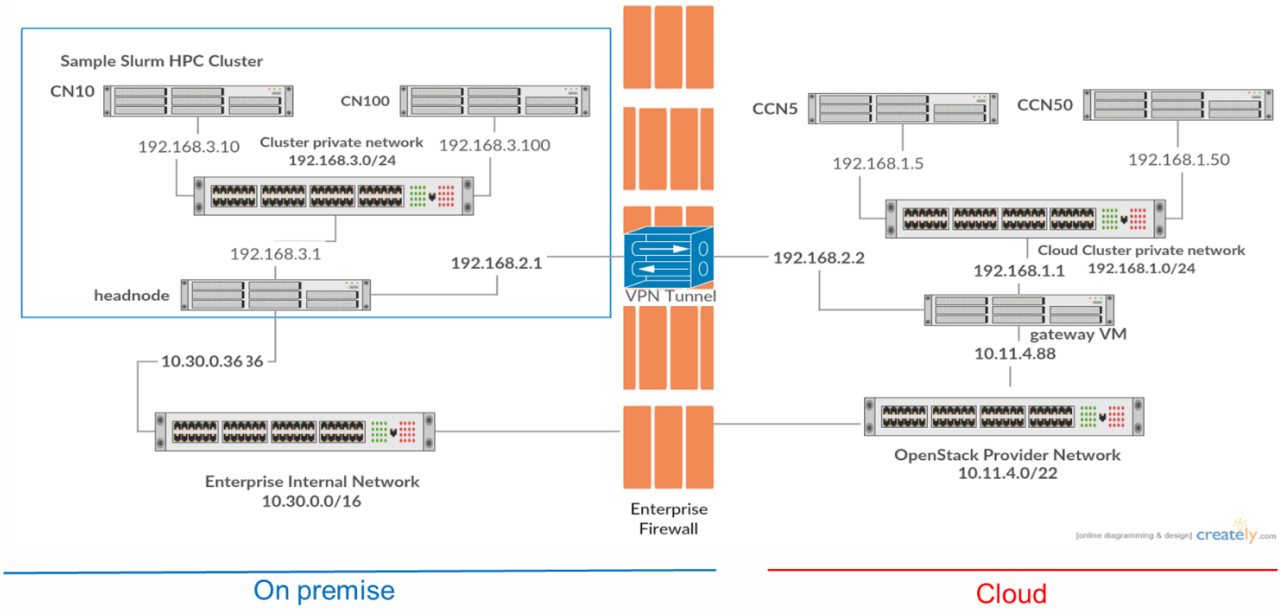

Figure 2 - network configurations

```
SelectType=select/linear # this will allocate a full node for a job instead of CPU's of a node
SuspendProgram=/usr/sbin/slurm_suspend # delete a cloud instance node
ResumeProgram=/usr/sbin/slurm_suspend # launch a cloud instance node
SuspendTime=600 # idle time of the instance before it´s deleted
SuspendExcNodes=tux[0-127] # node that should not be affected by suspend and resume programs (usually on premises)
TreeWidth=128 #(It is expected that cloud instances will not be reached by other compute nodes (like on premises) as they will not have their hostname/IP Address. So The tree width should be large enough to get all nodes from your cluster so all communication with the nodes will be done by the controller and will be no communication between nodes
NodeName=ec[0-127]  Weight=8 Feature=cloud State=CLOUD # declaration of cloud nodes. State=CLOUD tells the controller that those nodes are available but cant be reached until resumed
PartitionName=batch MaxTime=8:00:00 Nodes=tux[0-127],ec[0-127] Default=no # delcaration of a partition mixing on premises and cloud nodes
PrivateData=cloud # the controller will list cloud nodes shutdown when a regular user uses sinfo
```

Figure 3 – sample changes required to slurm.conf

```
def LaunchVM(OSC, VM, NodeList, Config):
  AppDir = os.path.dirname(os.path.abspath(sys.argv[0]))
  args = Args()
  args.NetID = Config["Gateway"]["InternalNetID"]
  NetName = OSC.GetNetworkName(args)

  args = GetNodesInfo(NodeList, VM)
  if args is None:
    syslog.syslog("VM "+VM+" not found in NodeList.conf. Aborting...")
    exit(1)
  args.ImageID = Config["Snapshot"]["ID"]
  args.Key = Config["HeadNode"]["RootKeyName"]
  args.CloudInitFile = AppDir+"/"+Config["Gateway"]["CloudInitFile"]
  args.Net = "uuid:"+Config["Gateway"]["InternalNetID"]+",port:"+args.NodePortUUID
  args.ReturnID = False
  syslog.syslog("Starting VM: "+VM)
  RetVM = OSC.CreateVM(args)
  subprocess.call(['scontrol', 'update', 'nodename=' + VM, 'nodeaddr=' + RetVM.addresses[NetName][0]['addr'], 'nodehostname=' + VM])

```

Figure 4 - launchVm in resume.py script

```
#!/usr/bin/python3

import sys
import os
import subprocess
import configparser
import OpenStackClass
import Slurm
import syslog

class Args():
  pass

AppDir = os.path.dirname(os.path.abspath(sys.argv[0]))
OSC = OpenStackClass.OpenStack(AppDir+'/slurm.ini')

args = Args()
args.Nodes = sys.argv[1]
VMList = Slurm.GetVMList(args)
for VM in VMList:
  args.VMNameOrID = VM
  syslog.syslog("Deleting VM: "+VM)
  try:
    OSC.DeleteVM(args)
  except Exception as e:
    print(e)
```

Figure 5 - suspend.py script

Disclosed by:

Glaucimar Da Silva Aguiar - Hewlett Packard Enterprise

Andrey Elision Monteiro Brito - Hewlett Packard Enterprise

Flávio Fonseca - Hewlett Packard Enterprise

Luís Eduardo Tenório Silva - Hewlett Packard Enterprise

Lucas Lima Vieira - Hewlett Packard Enterprise