pcocc Documentation

Release 0.6.2

François Diakhaté

Sep 18, 2019

Contents

CHAPTER 1

Introduction

pcocc (pronounced like "peacock") stands for Private Cloud On a Compute Cluster. It allows users of an HPC cluster to host their own clusters of VMs on compute nodes, alongside regular jobs. Users are thus able to fully customize their software environments for development, testing, or facilitating application deployment. Compute nodes remain managed by the batch scheduler as usual since the clusters of VMs are seen as regular jobs. For each virtual cluster, pcocc allocates the necessary resources to host the VMs, including private Ethernet and/or Infiniband networks, creates temporary disk images from the selected templates and instantiates the requested VMs.

CHAPTER 2

Working principle

pcocc leverages SLURM to start, stop and supervise virtual clusters in the same way as regular parallel jobs. It allocates CPU and memory resources using sbatch/salloc and a SLURM plugin allows to setup virtual networks on the allocated nodes. Once the nodes are allocated and configured, VMs are launched by SLURM as any other task with the rights of the invoking user. VMs are configured to replicate, as much as possible, the resources and capabilities of the portion of the underlying host that is allocated for them (CPU model and core count, memory amount and NUMA topology, CPU and memory binding...) so as to maximize performance.

To launch a virtual cluster, the user selects a template from which to instantiate its VMs and the number of requested VMs (it is possible to combine several templates among a cluster). A template defines, among other things, the base image disk to use, the virtual networks to setup, and optional parameters such as host directories to export to the VMs via 9p. Administrators can define system-wide templates from which users can inherit to define their own templates. When a VM is instantiated from a template, an ephemeral disk image is built from the reference image using copy-on-write. By default, any changes made to the VMs' disks are therefore lost once the virtual cluster stops but it is possible to save these changes to create new revisions of the templates.

2.1 Installing pcocc

This guide describes the installation of pcocc on a CentOS / RHEL 7 distribution or derivative. Installation on other distributions is not supported at this time even though it should work with minor adaptations.

2.1.1 Requirements and dependencies

pcocc makes use of several external components or services among which:

- A Slurm cluster with the Lua SPANK plugin
- Open vSwitch
- An etcd database and the etcd python bindings
- Qemu and KVM
- The gRPC framework

For virtual Infiniband networks:

- Mellanox adapters and drivers supporting SRIOV
- Mellanox OFED is recommended (especially for adapters based on the mlx5 driver)
- Linux kernel with VFIO support (CentOS / RHEL 7 kernels support this feature)

pcocc makes a few assumptions about the configuration of the host clusters such as:

- Users have home directories shared between front-end and compute nodes
- Users may ssh to allocated compute nodes without a password (using GSSAPI or public key authentication for example)
- Slurm manages task affinity and memory allocation

On a CentOS / RHEL 7 distribution, most dependencies are provided with a combination of the standard repositories and EPEL.

This guide also assumes that you already have a working Slurm cluster. The following guidelines should help you install other dependencies which are not available from standard repositories:

Installing Open vSwitch

pcocc relies on Open vSwitch to provide the VMs with private virtual networks. Open vSwitch can be downloaded from official website. The official installation guide can be used as an additional source for this process.

Building the RPM

Get tarball and specfile from latest stable Open vSwitch release (2.5.3 at the time of this writing) and build the RPM:

```
curl -O http://openvswitch.org/releases/openvswitch-2.5.3.tar.gz
tar xzf openvswitch-2.5.3.tar.gz openvswitch-2.5.3/rhel/openvswitch.spec -O␣
↪openvswitch.spec
yum-builddep openvswitch.spec
rpmbuild -ba --define "_sourcedir $PWD" openvswitch.spec
```

Installation and configuration

Install the RPM on all compute nodes or wait for it to be pulled as a dependency by pcocc. You'll also want to enable the service and start it (or reboot your compute nodes):

```
# On all compute nodes as root
systemctl enable openvswitch
systemctl start openvswitch
```

Installing the Slurm SPANK Lua plugin

pcocc uses the Slurm SPANK plugin infrastructure, and in particular, its LUA interface to setup compute nodes for running VMs. This interface is provided by the slurm-spank-plugins-lua package. pcocc will install a LUA script in the `/etc/slurm/lua.d` directory, `vm-setup.lua`.

Installing SPANK

You may download the SPANK plugins from their Github. As of this writing, there is an unresolved bug in this plugin, which we will fix by applying a patch. Download the latest tarball from the releases page to build a RPM:

```
wget https://github.com/chaos/slurm-spank-plugins/archive/0.37.tar.gz
wget https://storage.googleapis.com/google-code-attachments/slurm-spank-plugins/issue-
↪3/comment-0/slurm-spank-plugins-0.23-get_item.patch
mkdir -p $HOME/rpmbuild/SOURCES/
cp 0.37.tar.gz $HOME/rpmbuild/SOURCES/slurm-spank-plugins-0.37.tgz
mv slurm-spank-plugins-0.23-get_item.patch $HOME/rpmbuild/SOURCES/
tar xvf 0.37.tar.gz
```

In the source directory, edit the RPM specfile as follows (adapt to the current version number):

Listing 1: ./slurm-spank-plugins.spec

```
Name:  slurm-spank-plugins
Version: 0.37
Release: 1
Patch0: slurm-spank-plugins-0.23-get_item.patch
```

And apply the patch in the *%package lua* section, beetwen *%prep* and *%setup*:

```
%patch0 -p1
```

Proceed to building the RPM after installing required dependencies:

```
yum-builddep ./slurm-spank-plugins.spec
yum install lua-devel
rpmbuild -ba ./slurm-spank-plugins.spec --with lua --with llnl_plugins
```

Put the resulting packages in your repositories and install slurm-spank-plugins-lua on your front-end and compute nodes or wait for pcocc to pull it as a dependency.

Configuring SPANK

For the pcocc plugin to be properly loaded, we have to instruct the Slurm SPANK infrastructure to load all LUA addons in the standard `/etc/slurm/lua.d` directory by setting the following content inside `/etc/slurm/plugstack.conf`:

Listing 2: /etc/slurm/plugstack.conf

```
required /usr/lib64/slurm/lua.so /etc/slurm/lua.d/*
```

Installing the python-etcd library

pcocc uses the python-etcd library, a Python interface to the etcd database.

Build the RPM

The RPM building process is quite straightforward:

```
git clone https://github.com/jplana/python-etcd.git
cd python-etcd
python setup.py bdist_rpm
```

Put the resulting packages in your local repositories and install `python-etcd` on your front-end and compute nodes or wait for pcocc to pull it as a dependency.

Installing the grpc python libraries

pcocc uses the python-grpcio library, a Python interface for the gRPC framework.

Build the RPMs

The required RPMs can be built using pyp2rpm.

You can install this tool on a build node using pip:

```
pip install --upgrade pyp2rpm
```

You may have to also upgrade setuptools via pip if it complains.

Generate RPMs for python-protobuf:

```
install pyp2rpm on a build node with pip
pyp2rpm -t epel7 -b 2 -p 2 protobuf -v 3.6.0 -s
```

The generated specfile `$HOME/rpmbuild/SPECS/python-protobuf.spec` may need to be edited. Under the *%files section* add if needed:

```
%{python2_sitelib}/%{pypi_name}-%{version}-py%{python2_version}-nspkg.pth
```

Build the RPMs:

```
rpmbuild -ba $HOME/rpmbuild/SPECS/python-protobuf.spec
```

Generate and build RPMs for grpcio and grpcio_tools:

```
pyp2rpm -t epel7 -b 2 -p 2 grpcio -v 1.13.0 -s
rpmbuild -ba $HOME/rpmbuild/SPECS/python-grpcio.spec
pyp2rpm -t epel7 -b 2 -p 2 grpcio_tools -v 1.13.0 -s
rpmbuild -ba $HOME/rpmbuild/SPECS/python-grpcio_tools.spec
```

Put the resulting packages in your local repositories and install them on your front-end and compute nodes or wait for pcocc to pull them as a dependency.

2.1.2 RPM based installation

Pre-generated RPMs can be downloaded directly from the pcocc website. To generate a RPM from the source distribution, go to the root directory of the sources and run the following command:

```
python setup.py bdist_rpm
```

You may need to install the `rpm-build` package first.

The pcocc RPM should be installed on all your compute and front-end nodes using to the package manager which will pull all the necessary dependencies from your configured repositories. If you are missing something, please have a look at the guidelines provided in the previous section.

2.1.3 Prepare compute nodes and required services

Hardware virtualization support

Check that your compute nodes processors have virtualization extensions enabled, and if not (and possible) enable them in the BIOS:

```
# This command should return a match
grep -E '(vmx|svm)' /proc/cpuinfo
```

The kvm module must be loaded on all compute nodes and accessible (rw permissions) by all users of pcocc. You can use a udev rule such as:

Listing 3: /etc/udev/rules.d/80-kvm.rules

```
KERNEL=="kvm", GROUP=="xxx", MODE="xxx"
```

Adjust the **GROUP** and **MODE** permissions to fit your needs. If virtualization extensions are not enabled or access to kvm is not provided, pcocc will run Qemu in emulation mode which will be slow.

Slurm setup

It is recommended that Slurm is configured to manage process tracking, CPU affinity and memory allocation with cgroups. Set the following parameters in your Slurm configuration files:

Listing 4: /etc/slurm/slurm.conf

```
TaskPlugin=task/cgroup
Proctracktype=proctrack/cgroup
SelectTypeParameters=CR_Core_Memory
```

Listing 5: /etc/slurm/cgroup.conf

```
ConstrainCores=yes
TaskAffinity=yes
```

Make sure that your node definitions have coherent memory size et CPU count parameters for example:

Listing 6: /etc/slurm/slurm.conf

```
DefMemPerCPU=2000
NodeName=Node1 CPUs=8 RealMemory=16000 State=UNKNOWN
...
```

Note how **DefMemPerCPU** times **CPUs** equals **RealMemory**. As described in the requirements section, you need to enable Lua SPANK plugins. Follow this guide if you haven't done it yet:

etcd setup

pcocc requires access to a working etcd cluster with authentication enabled. Since pcocc will dynamically create users and permissions, you will probably want to deploy a dedicated instance. In its most basic setup, etcd is very simple to deploy. You just need to start the daemon on a server without any specific configuration. Authentication can be enabled with the following commands (you'll have to define a root password which you'll reference later in the pcocc onciguration files):

```
$ etcdctl user add root
$ etcdctl auth enable
$ etcdctl -u root:<password> role remove guest
```

This configuration can be used for a quick evaluation of pcocc. For a more reliable and secure setup you may refer to this guide:

Deploy a secure etcd cluster

This guide explains how to setup a cluster of highly available etcd servers and to secure communications with TLS. This guide is adapted from the official etcd documentation in which you can find more detailed information.

Certificate Generation

To enable TLS you need to generate a self-signed certificate authority and server certificates. In this example, we will consider using the following nodes as a etcd servers.

Hostname	FQDN	IP
node1	node1.mydomain.com	10.19.213.101
node2	node2.mydomain.com	10.19.213.102
node3	node3.mydomain.com	10.19.213.103

Note: For high-availability it is best to use an odd number of servers. Adding more servers increases high-availability and can improve read performance but decrease write performance. It is recommendend to use 3, 5 or 7 servers.

To generate the CA and server certificates, we use Cloudflare's cfssl as suggested in the official documentation. It can be installed very easily as follows:

```
mkdir ~/bin
curl -s -L -o ~/bin/cfssl https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
curl -s -L -o ~/bin/cfssljson https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
chmod +x ~/bin/{cfssl,cfssljson}
export PATH=$PATH:~/bin
```

Create a directory to hold your certificates and private keys. You may need them in the future if you need to generate more certificates so please make sure to keep them in a secure location with restrictive access permissions:

```
mkdir ~/etcd-ca
cd ~/etcd-ca
```

Generate the CA certificate:

```
echo '{"CN":"CA","key":{"algo":"rsa","size":2048}}' | cfssl gencert -initca - |␣
↪cfssljson -bare ca -
echo '{"signing":{"default":{"expiry":"43800h","usages":["signing","key encipherment",
↪"server auth","client auth"]}}}' > ca-config.json
```

For each etcd server, generate a certificate as follows:

```
export NAME=node1
export ADDRESS=10.19.213.101,$NAME.mydomain.com,$NAME
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl gencert␣
↪-config=ca-config.json -ca=ca.pem -ca-key=ca-key.pem -hostname="$ADDRESS" - |␣
↪cfssljson -bare $NAME
```

Note: If your servers will be reached from other IPs or DNS aliases, make sure to reference them in the **ADDRESS** variable

You now have to deploy the generated keys and certificates in the `/etc/etcd/` directory of each server node. For example for node1:

```
scp ca.pem root@node1:/etc/etcd/etcd-ca.crt
scp node1.pem root@node1:/etc/etcd/server.crt
scp node1-key.pem root@node1:/etc/etcd/server.key
ssh root@node1 chmod 600 /etc/etcd/server.key
```

Note: The CA certificate ca.pem will later have to deployed on all nodes hosting pcocc (front-end and compute nodes). Make sure you keep a backup along with the whole etcd-ca directory.

etcd Configuration

etcd needs to be configured on each server node in the /etc/etcd/etcd.conf configuration file. Here is an example for node1:

```
ETCD_NAME=node1
ETCD_LISTEN_PEER_URLS="https://10.19.213.101:2380"
ETCD_LISTEN_CLIENT_URLS="https://10.19.213.101:2379"
ETCD_INITIAL_CLUSTER_TOKEN="pcocc-etcd-cluster"
ETCD_INITIAL_CLUSTER="node1=https://node1.mydomain.com:2380,node2=https://node2.
↪mydomain.com:2380,node3=https://node3.mydomain.com:2380"
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://node1.mydomain.com:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://node1.mydomain.com:2379"
ETCD_TRUSTED_CA_FILE=/etc/etcd/etcd-ca.crt
ETCD_CERT_FILE="/etc/etcd/server.crt"
ETCD_KEY_FILE="/etc/etcd/server.key"
ETCD_PEER_CLIENT_CERT_AUTH=true
ETCD_PEER_TRUSTED_CA_FILE=/etc/etcd/etcd-ca.crt
ETCD_PEER_KEY_FILE=/etc/etcd/server.key
ETCD_PEER_CERT_FILE=/etc/etcd/server.crt
```

Note: **ETCD_NAME**, **ETCD_ADVERTISE_CLIENT_URLS**, **ETCD_INITIAL_ADVERTISE_PEER_URLS**, **ETCD_LISTEN_PEER_URLS** and **ETCD_LISTEN_CLIENT_URLS** have to be adapted for each server node.

Finally, you may enable and start the service on all etcd nodes:

```
systemctl enable etcd
systemctl start etcd
```

Check etcd Status

To check if your etcd server is running correctly you may do:

```
$ etcdctl --endpoints=https://node1.mydomain.com:2379 --ca-file=~/etcd-ca/ca.pem␣
↪member list
6c86f26914e6ace, started, Node2, https://node3.mydomain.com:2380, https://node3.
↪mydomain.com:2379
1ca80865c0583c45, started, Node1, https://node2.mydomain.com:2380, https://node2.
↪mydomain.com:2379
99c7caa3f8dfeb70, started, Node0, https://node1.mydomain.com:2380, https://node1.
↪mydomain.com:2379
```

Configure etcd for pcocc

Before enabling authentication, configure a `root` user in etcd:

```
etcdctl --endpoints="https://node1.mydomain.com:2379" --ca-file=~/etcd-ca/ca.pem ␣
↪user add root
```

Warning: Choose a secure password. You'll have to reference it in the pcocc configuration files.

Enable authentication:

```
etcdctl --endpoints="https://node1.mydomain.com:2379" --ca-file=~/etcd-ca/ca.pem auth␣
↪enable
```

Remove the guest role:

```
$ etcdctl --endpoints="https://node1.mydomain.com:2379" --ca-file=~/etcd-ca/ca.pem -u␣
↪root:<password> role remove guest
Role guest removed
```

You should no longer be able to access the keystore without authentication:

```
$ etcdctl --endpoints "https://node1.mydomain.com:2379" --ca-file=~/etcd-ca/ca.pem ␣
↪get /
Error:  110: The request requires user authentication (Insufficient credentials) [0]
```

2.1.4 Edit pcocc configuration files

The configuration of pcocc itself consists in editing YAML files in `/etc/pcocc/`. These files must be present on all front-end and compute nodes.

First, create a root-owned file named `/etc/pcocc/etcd-password` with `0600` permissions containing the etcd root password in plain text.

The `/etc/pcocc/batch.yaml` configuration file contains configuration pertaining to Slurm and etcd. Define the hostnames and client port of your etcd servers:

Listing 7: /etc/pcocc/batch.yaml

```
type: slurm
settings:
  etcd-servers:
    - node1
    - node2
    - node3
  etcd-client-port: 2379
  etcd-protocol: http
  etcd-auth-type: password
```

If you enabled TLS, select the *https* **etcd-protocol** and define the **etcd-ca-cert** parameter to the path of the CA certificate created for etcd (see *Deploy a secure etcd cluster*).

The sample network configuration in `/etc/pcocc/networks.yaml` defines a single Ethernet network named *nat-rssh*. It allows VMs of each virtual cluster to communicate over a private Ethernet network and provides them with a network gateway to reach external hosts. Routing is performed by NAT (Network Address Translation) using the hypervisor IP as source. It also performs reverse NAT from ports allocated dynamically on the hypervisor to the SSH port of each VM. DHCP and DNS servers are spawned for each virtual cluster to provide network IP addresses for the VMs. For a more detailed description of network parameters, please see *pcocc-resources.yaml(5)*.

Most parameters can be kept as-is for the purpose of this tutorial, as long as the default network ranges do not conflict with your existing IP addressing plan. The **host-if-suffix** parameter can be used if compute nodes have specific hostnames to address each network interface. For example, if a compute node, known by Slurm as `computeXX`, can be reached more efficiently via IPoIB at the `computeXX-ib` address, the **host-if-suffix** parameter can be set to *-ib* so that the Ethernet tunnels between hypervisors transit over IPoIB. Raising the MTU may also help improve performance if your physical network allows it.

The `/etc/pcocc/resources.yaml` configuration file defines sets of resources, currently only networks, that templates may reference. The default configuration is also sufficient for this tutorial: a single resource set is defined, *default* which only provides the *nat-rssh* network. See *pcocc-resources.yaml(5)* for more information about this file.

The `/etc/pcocc/templates.yaml` configuration file contains globally defined templates which are available to all users. It does not need to be modified initially. See *pcocc-templates.yaml(5)* for more information about this file.

The `/etc/pcocc/repos.yaml` configuration file defines the repositories where pcocc images will be located. By default, a single repository, named *user*, located in each user home directory is defined. You may want to change this directory to another path / filesystem better suited to hosting VM image files depending on your site characteristics. You may also want to add another *global* repository located in a path only writable by administrators to provide common images to users.

2.1.5 Validate the installation

To validate this configuration, you may launch the following command on a compute node as root:

```
pcocc internal setup init
```

It must run without error, and a bridge interface named according to the configuration of the *nat-rssh* network must appear in the list of network interfaces on the node:

```
# ip a
[..]
5: nat_xbr: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
```

(continues on next page)

(continued from previous page)

```
   link/ether 52:54:00:ff:ff:ff brd ff:ff:ff:ff:ff:ff
   inet 10.250.255.254/16 scope global newnat_xbr
      valid_lft forever preferred_lft forever
   inet6 fe80::5054:ff:feff:ffff/64 scope link
      valid_lft forever preferred_lft forever
```

You may then launch as root, on the same node:

```
pcocc internal setup cleanup
```

It should also run without error and the bridge interface should have disappeared. You should now be able to run VMs with pcocc. Please follow the *pcocc-newvm-tutorial(7)* tutorial to learn how to define VM templates an run your first VMs.

2.2 Tutorials

The following tutorials are available to get started with pcocc:

2.2.1 How to import VM images and define VM templates

This tutorial details how VM templates can be added to pcocc. It shows how to import cloud VM images provided by various Linux distributions which can be customized using cloud-init. More detailed information on how to configure such images is provided in the tutorial dealing with the *configuration of VMs with cloud-init*.

VM templates

pcocc is built around the notion of templates which define the main attributes of the VMs that can be instantiated. In a *template*, you can define, among other things:

- The reference image for the VM boot disk
- The network resources provided to the VM
- A cloud-config file to configure a cloud image (see *pcocc-cloudconfig-tutorial(7)*)
- Host directories to expose in the VM

Two types of templates can be configured:

- System-wide templates in `/etc/pcocc/templates.yaml`
- Per-user templates in `~/.pcocc/templates.yaml` (by default)

A user has access to both his personal templates and the system-wide templates. Note that a per-user template can inherit from a system-wide template.

Importing VM images

pcocc runs standard VM images in the Qemu qcow2 format. Many Linux distributions provide handy cloud images in this format which can be configured at instantiation time thanks to cloud-init.

- For Ubuntu you may get images from https://cloud-images.ubuntu.com/
- For Debian from https://cdimage.debian.org/cdimage/openstack/

- For CentOS from https://cloud.centos.org/centos/
- For Fedora from https://alt.fedoraproject.org/cloud/

In this guide, we use the following images (x86_64):

- Ubuntu Server (Artful): https://cloud-images.ubuntu.com/artful/current/artful-server-cloudimg-amd64.img
- CentOS 7: https://cloud.centos.org/centos/7/images/CentOS-7-x86_64-GenericCloud.qcow2

You may now download these images or those that you want to install. Note that the import process below is the same whether you use cloud-init enabled VMs or regular qcow2 images that you have already configured.

Note: In this guide, we consider that the highest priority repository is a user specific repository writable by the user as in the default configuration.

We can now import these images to our default repository:

```
$ pcocc image import artful-server-cloudimg-amd64.img ubuntu-artful-cloud
$ pcocc image import CentOS-7-x86_64-GenericCloud.qcow2 centos7-cloud
```

Note: We used the "-cloud" suffix as a convention to identify cloud-init enabled images.

At this point you should have these two images available in your repository:

```
$ pcocc image list
NAME                TYPE    REVISION    REPO        OWNER       DATE
----                ----    --------    ----        -----       ----
[...]
centos7-cloud       vm      0           user        jdoe        2018-08-24 20:46:35
ubunt-artful-cloud  vm      0           user        jdoe        2018-08-24 20:45:20
```

Defining VM templates

Now that we have copied the images to our repository, we can define templates for them within the pcocc *templates.yaml* configuration file. A system administrator can define them as system-wide templates in `/etc/pcocc/templates.yaml` to make them avaialable to all users. Otherwise, define them in `~/.pcocc/templates.yaml`. We first define basic templates which only make the image available. We can then inherit from them to create custom VMs.

Here is the content of `templates.yaml` for these three VMs (don't forget to replace `$VMDIR` with the actual PATH):

```
centos7-cloud:
    image: "centos7-cloud"
    resource-set: "default"
    description: "Cloud enabled CentOS 7"

ubuntu-artful-cloud:
    image: "ubuntu-artful-cloud"
    resource-set: "default"
    description: "Cloud enabled Ubuntu 17.10"
```

We selected *default* as the **resource-set** for these VMs. It should reference one of the resource sets defined in the `/etc/resources.yaml` file. Please refer to the *resources.yaml* and *networks.yaml* configuration files for more informations on this option.

Following this step, you should be able to list your new templates:

```
$ pcocc template list
NAME                 DESCRIPTION                 RESOURCES    IMAGE
----                 -----------                 ---------    -----
ubuntu-artful-cloud  Cloud enabled Ubuntu 17.10  default      ubuntu-artful-cloud
centos7-cloud        Cloud enabled CentOS 7      default      centos7-cloud
```

Basic VM configuration

Cloud-init enabled VMs such as the ones we installed in the previous section must be configured with a cloud-config file. If you imported a regular image which was already configured to be accessible by SSH you can skip this step.

Note: The cloud-init enabled images used in this guide don't have default login credentials. This is by design to prevent anyone from accessing the VM before you would be able to change the password. The cloud-config file will allow creating a user with proper authentication credentials such as a SSH public key.

The most basic cloud-config file which you can use is as follows:

```
#cloud-config
users:
   - name: demo
     sudo: ['ALL=(ALL) NOPASSWD:ALL']
     ssh-authorized-keys:
      - <your ssh public key>
```

It creates a user named *demo* able to use sudo without password and which can login via SSH with the specified key.

Moreover, we will also install the Qemu guest agent in our VMs. The Qemu guest agent is a daemon running in VMs allowing to interact with the guest without depending on networking. pcocc makes use of this agent when it is available, most notably to freeze guest filesystems and obtain consistent snapshots when using the *pcocc-save(1)* command. Append the following content to your cloud-config file:

```
packages:
    - qemu-guest-agent

runcmd:
    # Make sure that the service is up on all distros
    - systemctl start qemu-guest-agent
```

To pass this cloud-config file to our VMs, we can specialize the generic templates. As a regular user you can then add the fllowing content to the `~/.pcocc/templates.yaml` configuration file:

```
mycentos:
    inherits: centos7-cloud
    user-data: ~/my-cloud-config
    description: "Custom CentOS 7"

myubuntu:
    inherits: ubuntu-artful-cloud
```

(continues on next page)

(continued from previous page)

```
    user-data: ~/my-cloud-config
    description: "Custom Ubuntu"
```

Note: This configuration file assumes that you saved the previous cloud-config file as `~/my-cloud-config` in your home directory. Please adapt the path to what you have used.

Launching a virtual cluster

We can now instantiate VMs:

```
pcocc alloc -c2 mycentos:3,myubuntu:1
```

If you encounter any issue, note that the verbosity of all pcocc commands can be increased with the *-v* option to help with troubleshooting, for example:

```
pcocc -vv alloc -c2 mycentos:3,myubuntu:1
```

Using this command, you will launch four VMs with two cores each:

- three *mycentos*
- one *myubuntu*

VMs are numbered in order so they will be as as follows:

ID	Type
vm0	CentOS (1)
vm1	CentOS (2)
vm2	CentOS (3)
vm3	Ubuntu (1)

The pcocc alloc command puts you in a subshell which controls your allocation. If you exit this shell, your virtual cluster will be terminated and the temporary disks of the VMs will be destroyed.

If you used the cloud-config file described in the previous steps, you now should be able to login as the demo user (this assumes your default SSH private key matches the public key you specified in the cloud-config file, otherwise, specify the correct private key with the *-i* option)

```
pcocc ssh vm0 -l demo
```

You should be logged into one of the CentOS VM:

```
[demo@vm0 ~]$ cat /etc/redhat-release
CentOS Linux release 7.3.1611 (Core)
```

Note that, since you are in the aforementioned subshell, pcocc commands such as *pcocc ssh* automatically target the current virtual cluster, but you can target a specific cluster by jobid/jobname from any shell using the -j/-J pcocc options.

To reach the Ubuntu VM:

```
pcocc ssh vm3 -l demo

$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=17.10
DISTRIB_CODENAME=artful
DISTRIB_DESCRIPTION="Ubuntu Artful Aardvark (development branch)"
```

You can connect to the serial consoles using the following command:

```
pcocc console vm1
```

Note: Hit CTRL+C three times to leave the serial console.

You can also look back at the serial console log with:

```
pcocc console -l
```

Note: The console is very helpful to follow the VM boot and cloud-init progress. Installing packages can take some time, and in this example, the Qemu guest agent will only be available once the configuration process is complete. If you run into any issue, check the serial console log for error messages and make sure your YAML syntax is correct.

Saving VM images

Instead of configuring your VMs with cloud-init each time you instantiate them, you may want to create templates from pre-configured images which already contain the necessary packages, configuration files, user defintions etc. pcocc allows you to create new images from a running VM with the *pcocc-save(1)* command.

2.2.2 How to configure cloud-init enabled VMs

This tutorial shows how to configure a cloud-init enabled VM image, that is, a VM image where the cloud-init service has been enabled to run a boot time. Cloud-init is a multi-distribution package that handles early initialization of a VM instance. It can perform various tasks such as configuring users and access credentials, installing packages or setting up mount points. These tasks are defined in a cloud-config file that can be passed to a pcocc VM using the **user-data** template parameter.

Many distributions provide cloud-init enabled VM images that you can easily import as pcocc templates. More information about downloading and importing these images can be found in *pcocc-newvm-tutorial(7)*.

Note: By default it is not possible to login to cloud-enabled VMs, you must first specify a cloud-config file to setup a SSH key or other authentication mechanism.

This tutorial provides a quick overview of some cloud-config directives which can be used to configure pcocc VMs. The complete documentation of cloud-init capabilities can be found at https://cloudinit.readthedocs.io/en/latest/.

Using cloud-config files with pcocc

A cloud-config file is a YAML formatted file beginning with the *#cloud-config* pragma and followed by various configuration directives, some of which we will cover in the next sections. It can be passed to pcocc VMs by adding the **user-data** template parameter, for example:

```
mycentos:
    inherits: centos7-ci
    user-data: ~/conf
```

Where `~/conf` is the cloud-config file which will be passed to cloud-init at VM boot.

Most cloud-config directives are *per-instance*, which means they are applied once per instanciated VM, when it first boots. This mechanism relies on the value of **instance-id** which defaults to a random uuid generated for each instanciated pcocc VM. Alternatively, the **instance-id** can be set to a fixed value in the VM template definition (see *pcocc-templates.yaml(5)*). Each time cloud-init runs, it records the current **instance-id** in the VM filesysterm and only applies *per-instance* directives if it differs from what was previously recorded.

Setting up user credentials

With cloud-init enabled VMs the first configuration task is often to define user credentials to login to the VM. This can be done with the following syntax:

```
users:
  - name : demo1
    ssh-authorized-keys:
      - <ssh pub key 1>
      - <ssh pub key 2>
  - name : demo2
    ssh-authorized-keys:
      - <ssh pub key 3>
```

This defines two demo users, with their respective public SSH keys which have to be copy/pasted in the appropriate fields. You can also provide sudo privileges to a user with the **sudo** parameter or define its numerical id with the **uid** parameter:

```
users:
    - name: demo1
      sudo: ['ALL=(ALL) NOPASSWD:ALL']
      uid: 1247
      ssh-authorized-keys:
        - <ssh pub key 1>
```

Hostname considerations

By default, cloud-init stores the VM hostname in /etc/hostname which makes it persistent across reboots. This may not be what you want if you plan to instantiate many VMs from the same disk image and need them to find out their hostname dynamically from DHCP. You can inhibit this behaviour with the preserve hostname option:

```
preserve_hostname: true
```

This option must also be set in the cloud-init configuration file in the VM to be persistent (see *Writing files*):

```
write_files:
  - path: /etc/cloud/cloud.cfg.d/99_hostname.cfg
    permissions: '0644'
    content: |
      preserve_hostname: true
```

Running early boot commands

Boot commands are executed first in the configuration process. They are run as root. In contrast to other directives, they are run on each boot instead of only once. The *cloud-init-per* wrapper command can be used to run these boot commands only once. For example, if you are relying on local mirrors of package repositories you may want to disable those configured by default in the cloud-init image. For a CentOS guest you may add:

```
bootcmd:
 - [ cloud-init-per, instance, yumcleanbase, yum-config-manager, --disable, base]
 - [ cloud-init-per, instance, yumcleanupdates, yum-config-manager, --disable,␣
↪updates]
 - [ cloud-init-per, instance, yumcleanextras, yum-config-manager, --disable, extras]
```

Installing packages

You can provide a list of packages to install, for example:

```
packages:
    - qemu-guest-agent
    - vim
    - gcc
    - gdb
```

You can also setup additional package repositories for yum:

```
yum_repos:
   epel_mirror:
    baseurl: http://local-mirror.mydomain/pub/epel/testing/7/$basearch
    enabled: true
```

Or for apt:

```
apt:
   primary:
     - arches: [default]
       search:
         - http://local-mirror.mydomain/pub/debian/
```

You can also ask for packages to be upgraded first:

```
package_update: false
```

Writing files

You can write arbitrary files in the VM filesystem. Files are written after packages have been installed which allows for customizing configuration files. For example to write a simple `/etc/hosts` file for VMs on a private network:

```
write_files:
  - path: /etc/hosts
    permissions: '0644'
    content: |
      #Host file
      127.0.0.1   localhost localhost.localdomain

      10.252.0.1 vm0-ib0
      10.252.0.2 vm1-ib0
      10.252.0.3 vm2-ib1
```

Mounting filesystems

You can add entries to the VM fstab to mount filesystems. For example, to mount a 9p filesystem:

```
mounts:
 - [ optmount, /opt, 9p, 'trans=virtio,version=9p2000.L,msize=262144,nofail', '0', '0
↪']
```

Running commands

You can run arbitrary commands as root once at the end of the configuration process. Commands will run once all packages have been installed and files written. It can be used to reload a service that you just reconfigured or amend a configuration file:

```
runcmd:
    - sed -i 's/a/b' /etc/config-file
    - sytemctl restart service
```

To go further

We only briefly covered part of the capabilities of cloud-init. Please refer to https://cloudinit.readthedocs.io/en/latest/index.html for an exhaustive documentation.

2.2.3 How to mount host directories in VMs

In this guide we detail how to mount host directories from VMs thank to 9p over virtio.

Warning: CentOS and RHEL guests lack of built-in support for the 9p filesystem in the kernel. You'll have to compile the 9p modules from the kernel sources before being able to use this guide.

Defining host directories to export

First, you have to define host directories to export as mount-points inside the VM template in *templates.yaml*.

A mount-point named *optmount* for `/opt` can be defined as follows:

```
mount-points:
    optmount:
        path: /opt/
        readonly: true
```

Note: The **readonly** parameter defaults to *false* and therefore can be omitted for RW mounts

In this definition *optmount* is the tag of the 9p export which will be exposed to the VM. This tag has to be unique and will be referred to when mounting the export (see next section). */opt/* is the host path that is associated to this tag.

Mounting exports in the guest

To mount a 9p export in the guest, you can use the following command:

```
mount -t 9p -o trans=virtio [mount tag] [mount point] -oversion=9p2000.L,msize=262144
```

With the previous example this gives:

```
mount -t 9p -o trans=virtio optmount /opt/ -oversion=9p2000.L,msize=262144
```

The `/opt` directory from the host should now be mounted on `/opt` inside the guest. Note that Qemu act as the 9p server and performs the actual I/O on the host filesystem with the permissions of the user launching the virtual cluster.

To mount the directory automatically at boot time you may put it in your fstab. This can be done with the following cloud-config snippet (see *pcocc-cloudconfig-tutorial(7)*):

```
mounts:
  - [ optmount, /opt, 9p, 'trans=virtio,version=9p2000.L,msize=262144,nofail', '0', '0
↪']
```

Mirroring UIDs

Since I/O is performed with permissions of the user launching the virtual cluster, the best way to avoid permission issues is to access 9p mounts in your VM with a user having the same uid as your user on the host.

For example let's assume your user on the host is user1, you may retrieve it's numeric id with:

```
id user1
```

Which would give, for example:

```
uid=1023(user1) gid=1023(user1) groups=1023(user1)
```

Therefore, you would need to create a 'user1' in your VM with uid 1023. This may be done with the following cloud-config snippet (see *pcocc-cloudconfig-tutorial(7)*):

```
users:
- name : user1
  uid: 1023
```

If applicable, another solution is to configure your VMs to access the same directory sever as your hosts.

Mounting your home directory

Mounting one's own home directory is a common use-case for this feature. It makes it easy to share files and facilitates SSH key deployment. To export your home directory, set the following parameter in the VM template:

```
mount-points:
    myhome:
        path: %{env:HOME}
```

Define the mount point in the VM fstab with a cloud-config file, for example:

```
mounts:
  - [ myhome, /home/user1, 9p, 'trans=virtio,version=9p2000.L,msize=262144', '0', '0']
```

With a shared home directory, one can simply generate a private SSH key on the host and add the corresponding public key to the host's `~/.ssh/authorized_keys` file to enable SSH connexion from host to VMs as well as between VMs.

2.3 Configuration Files

pcocc relies on various configuration files:

batch.yaml Batch environment configuration file

networks.yaml Networks configuration file

resources.yaml Resource sets configuration file

templates.yaml VM templates definition file

See the following related manual pages:

2.3.1 Batch environment configuration file

Description

`/etc/pcocc/batch.yaml` is a YAML formatted file describing how pcocc should interact with the cluster batch environment. At this time pcocc expects a SLURM batch environment along with an etcd key-value store.

Syntax

This configuration files contains two keys. The **type** key defines the target batch manager. Currently the only supported value is *slurm* for the aforementioned environment composed of SLURM and etcd. The **settings** key contains a key/value mapping defining parameters for the target batch manager. The following parameters can be defined:

SLURM settings

etcd-servers A list of hostnames of the etcd servers to use for pcocc.

etcd-ca-cert Path to the etcd CA certificate (required for the "https" etcd-protcol).

etcd-client-port Port to connect to etcd servers.

etcd-protocol Protocol used to connect to etcd servers among:

- *http*: plain http.
- *https*: http over secure transport.

etcd-auth-type Authentication method to access the etcd servers among:

- *password* use password authentication (recommended)
- *none* do not use authentication

batch-args A list of additionnal arguments passed to SLURM at allocation time.

Sample configuration file

This is the default configuration file for reference. Please note that indentation is significant YAML:

```
# Batch manager
type: slurm
settings:
     # List of etcd servers
     etcd-servers:
         - etcd1
         - etcd2
         - etcd3
     # CA certificate
     etcd-ca-cert: /etc/etcd/etcd-ca.crt
     # Client port
     etcd-client-port: 2379
     # Protocol
     etcd-protocol: http
     etcd-auth-type: password
```

2.3.2 Networks configuration file

Description

`/etc/pcocc/networks.yaml` is a YAML formatted file defining virtual networks available to pcocc VMs. Virtual networks are referenced through VM resource sets defined in the `/etc/pcocc/resources.yaml` configuration file. For each virtual cluster, private instances of the virtual networks referenced by its VMs are created, which means each virtual network instance is only shared by VMs within a single virtual cluster.

A network is defined by its name, type and settings, which are specific to each network type. Two types of networks are supported: Ethernet and Infiniband.

Warning: Before editing this configuration file on a compute node, you should first make sure that no VMs are running on the node and execute the following command, as root:

```
pcocc internal setup cleanup
```

Syntax

`/etc/pcocc/networks.yaml` contains a key/value mapping. Each key defines a network by its name and the associated value must contain two keys: **type** which defines the type of network to define, and **settings** which is a key/value mapping defining the parameters for this network. This is summed up in the example below:

```
# Define a network named 'network1'
network1:
    # Select the network type
    type: ethernet
    # Define settings for ethernet networks
    settings:
        setting1: 'foo'
        setting2: 'bar'
```

The following networks are supported:

Ethernet network

A virtual Ethernet network is defined by using the network type *ethernet*. A VM connected to a network of this type receives an Ethernet interface linked to an isolated virtual switch. All the VMs of a virtual cluster connected to a given network are linked to the same virtual switch. Connectivity is provided by encapsulating Ethernet packets from the VMs in IP tunnels between hypervisors. If the **network-layer** parameter is set to *L2* pcocc only provides Ethernet layer 2 connectivity between the VMs. The network is entirely isolated and no services (such as DHCP) are provided, which means the user is responsible for configuring the VM interfaces as he likes. If the **network-layer** is set to *L3* pcocc also manages IP addressing and optionally provides access to external networks through a gateway which performs NAT (Network Address Translation) using the hypervisor IP as source. Reverse NAT can also be setup to allow connecting to a VM port such as the SSH port from the outside. DHCP and DNS servers are automatically setup on the private network to provide IP addresses for the VMs. The available parameters are:

dev-prefix Prefix to use when assigning names to virtual devices such as bridges and TAPs created on the host.

network-layer Whether pcocc should provide layer 3 services or only a layer 2 Ethernet network (see above). Can be set to:

- *L3* (default): Manage IP layer and provide services such as DHCP
- *L2*: Only provide layer 2 connectivity

mtu MTU of the Ethernet network. (defaults to 1500)

> **Warning:** Please note that the MTU of the Ethernet interfaces in the VMs has to be set 50 bytes lower than this value to account for the encapsulation headers. The DHCP server on a L3 network automatically provides an appropriate value.

mac-prefix Prefix to use when assigning MAC addresses to virtual Ethernet interfaces. MAC addresses are assigned to each VM in order starting from the MAC address constructed by appending zeros to the prefix. (defaults to 52:54:00)

host-if-suffix Suffix to append to hostnames when establishing a remote tunnel if compute nodes have specific hostnames to address each network interface. For example, if a compute node known by SLURM as computeXX can reached more efficiently via IPoIB at the computeXX-ib address, the **host-if-suffix** parameter can be set to *-ib* so that the Ethernet tunnels between hypervisors transit over IPoIB.

The following parameters only apply for a *L3* network:

int-network IP network range in CIDR notation reserved for assigning IP addresses to VM network interfaces via DHCP. This network range should be unused on the host and not be routable. It is private to each virtual cluster and VMs get a fixed IP address depending on their rank in the virtual cluster. (defaults to 10.200.0.0/16)

ext-network IP network range in CIDR notation reserved for assigning unique VM IPs on the host network stack. This network range should be unused on the host and not be routable. (defaults to 10.201.0.0/16)

dns-server The IP of a domain name resolver to forward DNS requests. (defaults to reading resolv.conf on the host)

domain-name The domain name to provide to VMs via DHCP. (defaults to pcocc.<host domain name>)

dns-search: Comma separated DNS search list to provide to VMs via DHCP in addition to the domain name.

ntp-server The IP of a NTP server to provide to VMs via DHCP.

allow-outbound Set to *none* to prevent VMs from establishing outbound connections.

reverse-nat A key/value mapping which can be defined to allow inbound connections to a VM port via reverse NAT of a host port. It contains the following keys:

vm-port The VM port to make accessible.

min-host-port Minimum port to select on the host for reverse NATing.

max-host-port Maximum port to select on the host for reverse NATing.

The example below defines a managed network with reverse NAT for SSH access:

```
# Define an ethernet network NAT'ed to the host network
# with a reverse NAT for the SSH port
nat-rssh:
  type: ethernet
  settings:
    # Manage layer 3 properties such as VM IP adresses
    network-layer: "L3"

    # Name prefix used for devices created for this network
    dev-prefix: "nat"

    # MTU of the network
    mtu: 1500

    reverse-nat:
      # VM port to expose on the host
      vm-port: 22
      # Range of free ports on the host to use for reverse NAT
      min-host-port: 60222
      max-host-port: 60322
```

The example below defines a private layer 2 network

```
# Define a private ethernet network isolated from the host
pv:
  # Private ethernet network isolated from the host
  type: ethernet
  settings:
    # Only manage Ethernet layer
    network-layer: "L2"

    # Name prefix used for devices created for this network
    dev-prefix: "pv"

    # MTU of the network
    mtu: 1500
```

IB network

A virtual Infiniband network is defined by using the type *infiniband*. An Infiniband partition is allocated for each virtual Infiniband network instantiated by a virtual cluster. VMs connected to Infiniband networks receive direct access to an Infiniband SRIOV virtual function restricted to using the allocated partition as well as the default partition, as limited members, which is required for IPoIB.

> **Warning:** This means that, for proper isolation of the virtual clusters, physical nodes should be set as limited members of the default partition and/or use other partitions for their communications.

pcocc makes use of a daemon on the OpenSM node which dynamically updates the partition configuration (which means pcocc has to be installed on the OpenSM node). The daemon generates the configuration from a template holding the static configuration to which it appends the dynamic configuration. Usually, you will want to copy your current configuration to the template file (/etc/opensm/partitions.conf.tpl in the example below) and have pcocc append its dynamic configuration to form the actual partition file referenced in the OpenSM configuration. The following parameters can be defined:

host-device Device name of a physical function from which to map virtual functions in the VM.

min-pkey Minimum pkey value to assign to virtual clusters.

max-pkey Maximum pkey value to assign to virtual clusters.

opensm-daemon Name of the OpenSM process (to signal from the pkeyd daemon).

opensm-partition-cfg The OpenSM partition configuration file to generate dynamically.

opensm-partition-tpl The file containing the static partitions to include in the generatied partition configuration file.

The example below sums up the available parameters:

```
ib:
  # Infiniband network based on SRIOV virtual functions
  type: infiniband
  settings:
    # Host infiniband device
    host-device: "mlx5_0"
    # Range of PKeys to allocate for virtual clusters
    min-pkey: "0x2000"
    max-pkey: "0x3000"
    # Name of opensm process
    opensm-daemon: "opensm"
    # Configuration file for opensm partitions
    opensm-partition-cfg: /etc/opensm/partitions.conf
    # Template for generating the configuration file for opensm partitions
    opensm-partition-tpl: /etc/opensm/partitions.conf.tpl
```

As explained above, pcocc must be installed on the OpenSM node(s) and the *pkeyd* daemon must be running to manage the partition configuration file:

```
systemctl enable pkeyd
systemctl start pkeyd
```

Sample configuration file

This is the default configuration file for reference:

```
# Define an ethernet network NAT'ed to the host network
# with a reverse NAT for the SSH port
nat-rssh:
  type: ethernet
  settings:
    # Manage layer 3 properties such as VM IP adresses
    network-layer: "L3"

    # Private IP range for VM interfaces on this ethernet network.
    int-network: "10.251.0.0/16"

    # External IP range used to map private VM IPs to unique VM IPs on the
    # host network stack for NAT.
    ext-network: "10.250.0.0/16"

    # Name prefix used for devices created for this network
    dev-prefix: "nat"

    # MTU of the network
    mtu: 1500

    reverse-nat:
      # VM port to expose on the host
      vm-port: 22
      # Range of free ports on the host to use for reverse NAT
      min-host-port: 60222
      max-host-port: 60322

    # Suffix to append to remote hostnames when tunneling
    # Ethernet packets
    host-if-suffix: ""


# Define a private ethernet network isolated from the host
pv:
  # Private ethernet network isolated from the host
  type: ethernet
  settings:
    # Only manage Ethernet layer
    network-layer: "L2"

    # Name prefix used for devices created for this network
    dev-prefix: "pv"

    # MTU of the network
    mtu: 1500

    # Suffix to append to remote hostnames when tunneling
    # Ethernet packets
    host-if-suffix: ""


# Define a private Infiniband network
ib:
  # Infiniband network based on SRIOV virtual functions
  type: infiniband
  settings:
```

(continues on next page)

(continued from previous page)

```
    # Host infiniband device
    host-device: "mlx5_0"
    # Range of PKeys to allocate for virtual clusters
    min-pkey: "0x2000"
    max-pkey: "0x3000"
    # Resource manager token to request when allocating this network
    license: "pkey"
    # Name of opensm process
    opensm-daemon: "opensm"
    # Configuration file for opensm partitions
    opensm-partition-cfg: /etc/opensm/partitions.conf
    # Template for generating the configuration file for opensm partitions
    opensm-partition-tpl: /etc/opensm/partitions.conf.tpl
```

See also

pcocc-template(1), *pcocc-templates.yaml(5)*, *pcocc-resources.yaml(5)*, *pcocc-newvm-tutorial(7)*, *pcocc-configvm-tutorial(7)*

2.3.3 Resource sets configuration file

Description

`/etc/pcocc/resources.yaml` is a YAML formatted file describing sets of resources that pcocc templates may reference. Currently resource sets are only composed of networks defined in `/etc/pcocc/networks.yaml`.

Syntax

`/etc/pcocc/resources.yaml` contains a key/value mapping. Each key represents a set of resources and the associated value contains a key, **networks**, whose value is a list of networks to provide to VMs. Interfaces will be added to VMs in the same order as they appear in this list, which means that, for example, the first Ethernet network in the list should appear as eth0 in the guest operating system. In addition, a key **default** can be set to *True* on one of the resource sets. It will be used by default for VM templates which do not specify a resource-set.

Sample configuration file

This is the default configuration file for reference. Please note that indentation is significant in YAML:

```
# This configuration file holds system-wide definitions of set of resources
# which can be used by VM templates

default:
  networks:
    - nat-rssh
  default: True

ib-cluster:
  networks:
    - nat-rssh
    - ib
```

See also

pcocc-template(1), *pcocc-templates.yaml(5)*, *pcocc-networks.yaml(5)*, *pcocc-newvm-tutorial(7)*

2.3.4 Image repositories configuration file

Description

`/etc/pcocc/repos.yaml` is a YAML formatted file describing object repositories used to store VM images. This configuration can be read from several locations. System-wide definitions are read from `/etc/pcocc/repos.yaml` while user-specific repositories are read from `$HOME/.pcocc/repos.yaml`. A user has access to images located in both his personal repositories and in system-wide repositories.

Note: The location of user configuration files, by default `$HOME/.pcocc` can be changed to another directory by setting the **PCOCC_USER_CONF_DIR** environment variable.

To learn how to interact with image repositories, please refer to the *pcocc-image(1)* documentation.

Syntax

`/etc/pcocc/repos.yaml` contains a key/value mapping. At the top-level a key named *repos* is defined. The associated value is a list of repositories. Each repository is defined by a key/value mapping containing two keys: *path* the path to a directory holding the repository and *name* the name associated with the repository. The *path* must point to either an initialized repository or to a non-existing directory which will be automatically created and initialized to an empty repository on first use. The repositories must appear in the list by order of priority: the first repository in the list is the first considered when looking up an image. Repositories defined in the user configuration file are considered before those defined in the system configuration file.

Sample configuration file

This is the default configuration file for reference. Please note that indentation is significant in YAML:

```
# This file defines a list of pcocc image repositories sorted by
# priority (highest priority first). To define a new repository, add a
# path to an non-existing directory

repos:
  - name: user
    path: "%{env:HOME}/.pcocc/repo"
# - name: global
#   path: "/var/lib/pcocc/images"
```

See also

pcocc-image(1), *pcocc-template(1)*, *pcocc-templates.yaml(5)*, *pcocc-newvm-tutorial(7)*

2.3.5 Template configuration file

Description

`templates.yaml` is a YAML formatted file defining VM templates that can be instantiated with pcocc. This configuration can be read from several locations. System-wide definitions are read from `/etc/pcocc/templates.yaml` while user-specific templates are read from `$HOME/.pcocc/templates.yaml` and from any file matching `$HOME/.pcocc/templates.d/*.yaml`. A user has access to both his personal templates and the system-wide templates.

Note: The location of user configuration files, by default `$HOME/.pcocc` can be changed to another directory by setting the **PCOCC_USER_CONF_DIR** environment variable.

Syntax

The `templates.yaml` file contains a key/value mapping. Each key represents a template whose parameters are defined in the associated value. If the system configuration doesn't define a default **resource-set** it is a mandatory parameter. It can however be inherited from a parent template. All other parameters are optional.

Template parameters

image URI of a boot disk image in a pcocc repository. VMs instantiated from this template will boot from an ephemeral private copy of this image. See *pcocc-image(1)* and *pcocc-newvm-tutorial(7)* for importing existing images and *pcocc-save(1)* for creating new images or revisions from running VMs. In previous pcocc releases, images were stored in standalone directories. While still operational, this image format is considered as deprecated and support for these images will removed in a future version.

resource-set Resources to provide to VMs instantiated from this template. This must reference a resource set defined in *resources.yaml*.

inherits Name of a "parent" template from which to inherit parameters. Parameters defined in the template will override parameters inherited from the parent. User-defined templates can inherit from other user-defined templates or system-wide templates. System-wide templates can only inherit from other system-wide templates.

description A string describing the VM template. This parameter is not inheritable.

user-data A cloud-config file to configure a VM image with cloud-init (see *pcocc-configvm-tutorial(7)*)

instance-id Instance ID to provide to cloud-init (defaults to a randomly generated uuid).

mount-points A key/value mapping defining directories to export as 9p mount points (see *pcocc-9pmount-tutorial(7)*). Each key defines a 9p mount tag and the associated value defines the directory to export. The following parameters are supported:

> **path** The host directory to export.
>
> **readonly** If set to *true* the export will be read-only.

persistent-drives A list of persistent drives to provide to the VMs. Each element of the list is a single key/value mapping where the key is the path to the VM disk file (in raw format), and the value defines parameters for the drive. VMs have direct access to the source data which means changes are persistent and the template should usually only be instantiated once at a time. When a virtual cluster contains VMs instianciated from templates with persistent drives, pcocc will try to properly shutdown the guest operating when the user relinquishes the resource allocation. For each drive, the following parameters can be configured:

> **cache** Qemu cache policy to apply to the drive (defaults to *writeback*)

mmp Type of Multi-mount protection to apply to the drive (note that these guarantees do not hold if multiple users try to access the same drive file). The following parameters are available:

- *yes* (default): Only allow the drive to be attached once.
- *cluster*: Allow the drive to be attached to multiple VMs of a single cluster.
- *no*: Disable this feature.

remote-display A protocol for exporting the graphical console of the VMs. The only supported value is *spice*.

custom-args A list of arguments to append to the Qemu command line.

qemu-bin Path to the Qemu binary to use to run the VMs (defaults to searching for qemu-system-x86 in the user's PATH)

nic-model Model of Qemu virtual Ethernet network card to provide to VMs (defaults to "virtio-net").

machine-type Type of Qemu machine to emulate (defaults to "pc").

disk-model Model of Qemu virtual drive to provide to VMs. Valid parameters are *virtio* (default), *virtio-scsi* or *ide*.

emulator-cores Number of cores to reserve for Qemu threads. These cores are deducted from the cores allocated for each VM (defaults to 0).

bind-vcpus Controls whether pcocc attempts to bind vCPUS and memory to underlying cores and NUMA nodes (defaults to True).

Sample configuration file

This is a sample template definition. Please note that indentation is significant in YAML:

```
# Define a template named 'example'
example:
      # Inherit parameters from a parent template (default: no inheritance)
      # inherits: 'parent-example'

      # Resources to allocate (required)
      resource-set: 'cluster'

      # Directory holding the image template for the CoW boot drive (default: no␣
↪image)
      image: '/path/to/images/myexample'

      # Model of Qemu virtual drive for the image (default: virtio)
      disk-model: 'ide'

      # List of additional persistent (non CoW) drives. For templates lacking
      # an image, the first drive will be used as the default boot drive
      persistent-drives:
      # Simple syntax
      - '/path/to/first/drive'
      # Extended syntax with parameters
      - '/path/to/second/drive':
        # Multi-mount protection
        # Valid values:
        #  - yes (default): drive can only be attached once
        #  - cluster: drive can be attached to multiple VMs of a single cluster
        #  - no: disable this feature
        # These guarantees do not apply if multiple users try to attach the
```

(continues on next page)

(continued from previous page)

```
      # same drive
      mmp: 'no'
      # Qemu caching mode (default: 'writeback')
      cache: 'unsafe'

  # Description of this template (default: none)
  description: 'Example of a template'

  # Mount points to expose via virtio-9p (default: none)
  mount-points:
   # 9p mount tag
   homedir:
     # Host path to export
     path: '/home'
     # Set to true for readonly export
     readonly: false

  # Custom arguments to pass to Qemu (default: none)
  custom-args:
    - '-cdrom'
    - '/path/to/my-iso'

  # Qemu executable to use (default: look for qemu-system-x86_64 in user PATH)
  qemu-bin: '/path/to/qemu/bin/qemu-system-x86_64'

  #  Model of Ethernet cards (default: virtio-net)
  nic-model: 'e1000'

  # Reserved cores for Qemu emulation (default: 0)
  emulator-cores: 2
```

See also

pcocc-template(1), *pcocc-image(1)*, *pcocc-batch(1)*, *pcocc-alloc(1)*, *pcocc-save(1)*, *pcocc-resources.yaml(5)*, *pcocc-networks.yaml(5)*, *pcocc-newvm-tutorial(7)*

2.4 Command Line Reference

pcocc provides the following commands:

2.4.1 Instantiate or restore a virtual cluster (interactive mode)

Synopsis

pcocc alloc [OPTIONS] [BATCH_OPTIONS]... CLUSTER_DEFINITION

Description

Instantiate or restore a virtual cluster in interactive mode. A cluster definition is expressed as a list of templates and counts. For example, pcocc alloc *tpl1:6,tpl2:2* instantiates a cluster with 6 VMs from template *tpl1* and 2 VMs from template *tpl2*.

By default, an interactive shell is launched which allows to easily interact with the virtual cluster as all pcocc commands launched from the shell implicitly target the related virtual cluster. Resources are relinquished when either the VMs are powered off or the interactive shell exits (see below). Any data stored on ephemeral disks is lost after the allocation completes.

Batch options are passed on to the underlying batch manager (see salloc(1)). By default allocations are created with the name *pcocc* unless specified otherwise in the batch options. From outside the interactive shell, pcocc commands look for a job named *pcocc* and will target it if there is only one match. Otherwise, the id or name of the allocation hosting the cluster must be specified.

Instead of launching an interactive shell, it is possible to execute a script on the front-end node with the *-E* option. The cluster will be terminated once the script exits. As in the in interactive shell, pcocc commands launched within the script implicitely target the current cluster.

Options

-r, –restart-ckpt [DIR] Restart cluster from the specified checkpoint

-E, –alloc-script [SCRIPT] Execute a script on the allocation node

-h, –help Show this message and exit.

Examples

Instanciate a new virtual cluster

To allocate eight VMs with four cores each with the *test* SLURM qos, six from template *tpl1* and two from *tpl2*:

```
pcocc alloc -c 4 --qos=test -J ubuntu tpl1:6,tpl2:2
```

Restore a checkpointed cluster

The *pcocc-ckpt(1)* command allows to save the state of a whole cluster (disks and memory) in a checkpoint. Assuming a cluster was submitted and checkpointed as follows:

```
pcocc alloc -c 8 myubuntu:3
pcocc ckpt ./savedalloc
```

To restore it from the checkpoint in interactive mode:

```
pcocc alloc -c 8 -r $PWD/savedalloc myubuntu:3
```

Warning:

- Make sure that the parameters in the restore command (core count, template types, ...) are the same that were used when the cluster was first allocated. The cluster also has to be restored on the same model of physical nodes as when it was first allocated.
- The restore path must be an absolute path

See also

pcocc-batch(1), *pcocc-ckpt(1)*, *pcocc-template(1)*, *pcocc-templates.yaml(5)*

2.4.2 Instantiate or restore a virtual cluster (batch mode)

Synopsis

pcocc batch [OPTIONS] [BATCH_OPTIONS]... CLUSTER_DEFINITION

Description

Instantiate or restore a virtual cluster in batch mode. A cluster definition is expressed as a list of templates and counts. For example, pcocc batch *tpl1:6,tpl2:2* instantiates a cluster with 6 VMs from template *tpl1* and 2 VMs from template *tpl2*. Resources are relinquished when the VMs are powered off or when the batch script exits. Any data stored on ephemeral disks is lost after the job completes.

Batch options are passed on to the underlying batch manager (see sbatch(1)). By default batch jobs are submitted with the name *pcocc* unless specified otherwise in the batch options. pcocc commands which target a virtual cluster look for a job named *pcocc* and will select it if there is only one match. Otherwise, the id or name of the batch job hosting the cluster must be specified.

It is possible to execute a script on the first physical compute node with the *-E* option or on the first VM with the *-b* option. The latter requires that the pcocc guest agent is running in the VM. The cluster will be terminated once the script exits. As in a *pcocc-alloc(1)<alloc>* interactive shell, pcocc commands launched within a host script implicitely target the current pcocc cluster.

Options

-r, –restart-ckpt [DIR] Restart cluster from the specified checkpoint

-b, –batch-script [FILENAME] Launch a batch script in the first VM

-E, –host-script [FILENAME] Launch a batch script on the first host

-h, –help Show this message and exit.

Examples

Instanciate a new virtual cluster

For example to allocate eight VMs with four cores each with the *ubuntu* job name, six from template *tpl1* and two from *tpl2*:

```
pcocc batch -J ubuntu -c 4 tpl1:6,tpl2:2
```

Restore a checkpointed cluster

The *pcocc-ckpt(1)* command allows to save the state of a whole cluster (disks and memory) in a checkpoint. Assuming a cluster was submitted and checkpointed as follows:

```
$ pcocc batch -c 8 myubuntu:3
Submitted batch job 311244
$ pcocc ckpt -j 311244 ./savedbatch
```

To restore it from the checkpoint in batch mode:

```
pcocc batch -c 8 -r $PWD/savedbatch myubuntu:3
```

Warning:

- Make sure that the parameters in the restore command (core count, template types, ...) are the same that were used when the cluster was first submitted. The cluster also has to be restored on the same model of physical nodes as when it was first submitted.
- The restore path must be an absolute path

See also

pcocc-alloc(1), *pcocc-ckpt(1)*, *pcocc-template(1)*, *pcocc-templates.yaml(5)*

2.4.3 Checkpoint a virtual cluster

Synopsis

pcocc ckpt [OPTIONS] CKPT_DIR

Description

Checkpoint the current state of a cluster. Both the disk image and memory of all VMs of the cluster are saved and the cluster is terminated. It is then possible to restart from this state using the *–restart-ckpt* option of the alloc and batch commands.

CKPT_DIR should not already exist unless *-F* is specified. In that case, make sure you're not overwriting the checkpoint from which the cluster was restarted.

Warning: Qemu does not support checkpointing all types of virtual devices. In particular, it is not possible to checkpoint a VM with 9p exports mounted or attached to host devices such as an Infiniband virtual function.

Options

-j, –jobid [INTEGER] Jobid of the selected cluster

-J, –jobname [TEXT] Job name of the selected cluster

-F, –force Overwrite directory if exists

-h, –help Show this message and exit.

Example

To checkpoint the cluster with jobid 256841 in the `$HOME/ckpt1/` directory:

```
pcocc ckpt -j 256841 $HOME/ckpt1/
```

This produces a disk and a memory image for each VM:

```
ls ./ckpt1/
disk-vm0  disk-vm1  memory-vm0  memory-vm1
```

To restore a virtual cluster, see *pcocc-alloc(1)* or *pcocc-batch(1)*.

See also

pcocc-alloc(1), *pcocc-batch(1)*, *pcocc-save(1)*, *pcocc-dump(1)*

2.4.4 Connect to a VM console

Synopsis

pcocc console [OPTIONS] [VM]

Description

Connect to a VM console.

Note: In order to leave the interactive console, hit CTRL+C three times.

Options

-j, –jobid [INTEGER] Jobid of the selected cluster

-J, –jobname [TEXT] Job name of the selected cluster

-l, –log Show console log

-h, –help Show this message and exit.

Examples

Connect to a VM console

To connect to vm3 console in the default job:

```
pcocc console vm3
```

Note:

- If you connect while the VM is booting, you should see the startup messages appear in the interactive console. In this case, wait until the login prompt appears.
- If the VM has already booted, you may need to push enter a few times for the login prompt to appear, as only new console output is displayed.

See the console log

The *-l* flag allows looking at past output:

```
pcocc console -l vm0
```

This produces a paged output of vm0 logs.

Note: When using cloud-init debug information can be found in the console, which allows to check the configuration process.

See also

pcocc-ssh(1), *pcocc-scp(1)*, *pcocc-exec(1)*, *pcocc-nc(1)*, *pcocc-reset(1)*

2.4.5 Dump the memory of a VM to a file

Synopsis

pcocc dump [OPTIONS] VM DUMPFILE

Decription

Dump the memory of a VM to a file. The file is saved as ELF and includes the guest's memory mappings. It can be processed with crash or gdb.

Options

-j, –jobid [INTEGER] Jobid of the selected cluster

-J, –jobname [TEXT] Job name of the selected cluster

-h, –help Show this message and exit.

Examples

To dump the memory of the first VM in the `ouptut.bin` file:

```
pcocc dump vm1 output.bin
```

See also

pcocc-ckpt(1), *pcocc-reset(1)*

2.4.6 Display the graphical output of a VM

Synopsis

pcocc display [OPTIONS] [VM]

Description

Display the graphical output of a VM. By default, the *remote-viewer* tool is invoked to display the graphical console of a VM. The *-p* switch can be used to display the content of a *remote-viewer* connection file instead. This allows to launch *remote-viewer* manually.

Note: This requires the VM to have a **remote-display** method defined in it's template (see *pcocc-templates.yaml(7)*)

Options

-j, –jobid [INTEGER] Jobid of the selected cluster

-J, –jobname [TEXT] Job name of the selected cluster

–user [TEXT] Select cluster among jobs of the specified user

-p, –print_opts Print remote-viewer options

-h, –help Show this message and exit.

Examples

To display to the graphical console of the first VM:

```
pcocc display vm0
```

See also

pcocc-ssh(1), *pcocc-console(1)*, *pcocc-template(1)*, *pcocc-templates.yaml(7)*

2.4.7 Execute commands through the pcocc guest agent

Synopsis

pcocc exec [OPTIONS] [CMD]...

Decription

Execute commands through the guest agent

For this to work, the pcocc guest agent must be started in the guest. This is mostly available for internal use where we do not want to rely on a network connexion/ssh server.

Note: It is possible to detach from the output by typing *Escape* + *Enter*. In this case you may end the execution with *pcocc command release*.

Options

-i, –index [INTEGER] Index of the VM on which the command should be executed

-j, –jobid [INTEGER] Jobid of the selected cluster

-J, –jobname [TEXT] Job name of the selected cluster

-w, –rng [TEXT] Rangeset or vmid on which to run the command

-c, –cores [TEXT] Number of cores on which to run the command

-u, –user [TEXT] User id to use to execute the command

-g, –gid Group id to use to execute the command

-h, –help Show this message and exit.

Examples

Execute a command

To run a command in the first VM of the default job as the current user:

```
pcocc exec "hostname"
```

To run the same command on all VMs (the "-" rangeset means all VM):

```
pcocc exec -w - "hostname"
```

To run a command as root in the third VM of the job named *centos*:

```
pcocc exec -J centos -u root -i 2 cat /etc/shadow
```

See also

pcocc-ssh(1), *pcocc-scp(1)*, *pcocc-exec(1)*, *pcocc-nc(1)*

2.4.8 List and manage VM images

Synopsis

pcocc image [COMMAND] [ARG]

Description

List and manage virtual machine images.

All the subcommands of *pcocc image* operate on images stored in pcocc repositories. The list of pcocc repositories is defined in *repos.yaml* (see *pcocc-repos.yaml(5)*).

Images in repositories are uniquely identified by a name and revision number. In all pcocc commands and configuration files, images in repositories are specified with the following URI syntax: [REPO:]IMAGE[@REVISION]. If REPO is omitted the command will look in all defined repositories by order of priority until it finds a matching image. If REVISION is omitted, the highest revision of the image is selected.

Images are made of a stack of layers with each layer containing the differences from the previous layers. Layers can be shared between images in a repository which allows to reduce the storage footprint and speeds up operations by avoiding unnecessary data movement.

Sub-Commands

list [-R repo] [REGEX] List image in repositories. The result can be filtered by repository and/or by image name with a regular expression.

show [IMAGE] Show a detailed description of the specified imagge

import [-t fmt] [KIND] [SOURCE] [DEST] Import the source image file to an image in the destination repository. The destination image name must not already be used in the destination repository and the revision is ignored since the import operation creates the first revision of a new image. See below for supported image kinds and file formats.

export [-t fmt] [SOURCE] [DEST] Export the source image file from a repository to the destination file.

copy [SOURCE] [DEST] Copy an image from a repository to another image in a repository. The destination image name must not already be used in the destination repository and the destination revision is ignored since a copy operation creates the first revision of a new image.

delete [IMAGE] Delete an image from a repository. If a revision is specified, only the specified revision is deleted, otherwise all revisions of the image are deleted.

resize [IMAGE] [NEW_SZ] A new image revision is created with the new image size.

repo list List configured repositories

repo gc [REPO] Cleanup unnecessary data from a repository. This command should be run to free space used by data no longer used by any image.

Import and export file formats

pcocc repositories currently only manage VM images so the only valid value for KIND is *vm*. The following VM image file formats are supported: *raw*, *qcow2*, *qed*, *vdi*, *vpc*, *vmdk*. By default, pcocc will try to guess the file format from the image file itself or from its extension. The file format of the imported / exported file can be forced with the -t option.

Examples

To list available images:

```
pcocc image list
```

To import an image into a repository named *global*:

```
pcocc image import vm $HOME/CentOS-7-x86_64-GenericCloud.qcow2 global:centos7-cloud
```

To copy an image between repositories:

```
pcocc image copy global:centos7-cloud user:mycentos7
```

To get detailed information relative to an image:

```
pcocc image show user:mycentos7
```

To delete a specific revision of an image:

```
pcocc image delete user:mycentos7@5
```

To completely delete all revisions of an image:

```
pcocc image delete myrepo:centos7-cloud
```

See also

pcocc-save(1), *pcocc-repos.yaml(5)*, *pcocc-templates.yaml(5)*

2.4.9 Send a command to the monitor

Synopsis

pcocc monitor-cmd [OPTIONS] [VM] [CMD]...

Description

Send a command to the Qemu monitor of a VM. The commands are not interpreted by pcocc and are directly passed to the Qemu monitor. It allows to use specific Qemu features not exposed by pcocc. For detailed documentation on the available commands, refer to the Qemu documentation: https://www.qemu.org/documentation/.

Options

-j, –jobid [INTEGER] Jobid of the selected cluster

-J, –jobname [TEXT] Job name of the selected cluster

-h, –help Show this message and exit.

Examples

Obtain help on available Qemu monitor commands:

```
$ pcocc monitor-cmd help
acl_add aclname match allow|deny [index] -- add a match rule to the access control␣
↪list
acl_policy aclname allow|deny -- set default access control list policy
..
xp /fmt addr -- physical memory dump starting at 'addr'
```

Get help on a specific Qemu monito command (here the info command):

```
$ pcocc monitor-cmd help info
info balloon  -- show balloon information
info block [-v] [device] -- show info of one block device or all block devices
..
info version  -- show the version of QEMU
info vnc  -- show the vnc server status
```

Run a command:

```
$ pcocc monitor-cmd vm0 info version
```

See also

pcocc-dump(1)

2.4.10 Connect to a VM via nc

Synopsis

pcocc nc [OPTIONS] [NC_OPTS]...

Description

Connect to a VM via nc

> **Warning:** This requires the VM to have the selected port reverse NAT'ed to the host in its NAT network configuration.

Options

-j, –jobid INTEGER Jobid of the selected cluster

-J, –jobname TEXT Job name of the selected cluster

–user TEXT Select cluster among jobs of the specified user

-h, –help Show this message and exit.

Example

To open a connection to the SSH server running in the first VM of the xjob called *ubuntu*:

```
pcocc nc -J ubuntu vm0 22
```

This is can be useful to simplify connections to pcocc VMs using SSH ProxyCommands. For example by adding the following content to the *~.ssh/config* file:

```
Host ubuntu-vm0
ProxyCommand pcocc nc -J ubuntu vm0 22
```

It is possible to connect to the first VM of the job named *ubuntu* without relying on pcocc ssh:

```
ssh ubuntu-vm0
```

See also

pcocc-ssh(1), *pcocc-scp(1)*, *pcocc-exec(1)*, *pcocc-networks.yaml(5)*

2.4.11 List current pcocc jobs

Synopsis

pcocc ps [OPTIONS]

Description

List currently running pcocc jobs. By default, only the jobs of the current user are listed. This behaviour can be changed with the *–all* and *–user* options.

Options

-u –user [TEXT] List jobs of the specified user

-a, –all List all pcocc jobs

-h, –help Show this message and exit.

Examples

To list the pcocc jobs of the current user

pcocc ps

2.4.12 Reset a VM

Synopsis

pcocc reset [OPTIONS] [VM]

Description

Reset a VM. The effect is similar to the reset button on a physical machine.

Note: When you reset a VM, it is not returned to it's initial state and modifications to its ephemeral disks are kept. Cloud-init enabled VMs will not replay instantiation-time configuration directives.

Options

-j, –jobid [INTEGER] Jobid of the selected cluster

-J, –jobname [TEXT] Job name of the selected cluster

-h, –help Show this message and exit.

Example

To power cycle vm1:

```
pcocc reset vm1
```

See also

pcocc-console(1), *pcocc-dump(1)*

2.4.13 Save the disk of a VM

Synopsis

pcocc save [OPTIONS] [VM]

Description

Save the disk of a VM to a new disk image.

By default, only the differences between the current state of the VM disk and the image from which it was instantiated are saved in an incremental file to form a new revision of the image. When a VM is instantiated it uses the latest revision of the image defined in its template. The *-d* option allows to create a new image instead of a new revision of the current image. The *–full* flag allows to make the new image or revision from a standalone layer containing the whole image instead of a succession of incremental layers. Making a full image can be useful for performance reasons once the number of layers gets too large.

Warning: It is recommended to have the *qemu-guest-agent* package installed in the guest (see next section).

Note: In previous releases, pcocc images were saved in standalone directories. While this style of images is still properly handled by pcocc save, it is now considered deprecated and support will be removed in a future version.

Recommendations

Saving a running VM may lead to corruption if the filesystem is being accessed. To ensure a consistent filesystem image, pcocc tries to contact the Qemu guest agent in the VM to freeze the filesystems before creating a new image from this disk. Therefore, it is recommended to make sure that the qemu guest agent is running in the guest (see : *pcocc-newvm-tutorial(7)*).

If pcocc cannot contact the agent, it will emit a warning message but it will try to save the VM anyway. If installing the agent is not possible, you should freeze the filesystems by hand or simply shutdown your VM before calling pcocc save. In a Linux guest, you can use, as root *shutdown -H now* to shutdown a VM without powering it off (as you want to keep your resource allocation).

Options

-j, –jobid INTEGER Jobid of the selected cluster

-J, –jobname TEXT Job name of the selected cluster

-d, –dest URI Make a full copy in a new directory

-s, –safe Wait indefinitely for the Qemu agent to freeze filesystems

–full Save a full image even if not necessary

-h, –help Show this message and exit.

Examples

In these examples, we consider that the *qemu-guest-agent* is installed.

Create a new image revision

If you have write permissions on the image directory used by your VMs, you can create new image revisions. For example to create a new revision of the image used by first VM of your virtual cluster use:

```
$ pcocc save vm0
Saving image...
vm0 frozen
vm0 thawed
vm0 disk successfully saved to centos7-cloud revision 1
```

A new image revision is created

```
$ pcocc image show centos7-cloud
[..]

REVISION    SIZE       CREATION DATE
--------    ----       -------- ----
0           958  MB    2018-08-03 16:04:12
1           44.0 MB    2018-08-03 16:09:54
```

The next VM instantiated with this image will use the new revision. You can undo saves by removing the latest revisions (see *pcocc-image(1)*) or specify a specific revision in your template image URI.

Create a new independent images

If you want to create a new image or do not have write permissions on the image repository used by your VM you can use the *-d* flag to save to a new VM image:

```
$ pcocc save vm0 -d user:mycentos7
Saving image...
vm0 frozen
vm0 thawed
Merging snapshot with backing file to make it standalone...
vm0 disk successfully saved to user:mycentos revision 1
```

You can now create a template inheriting from the original one, but using the new image, by editing your `templates.yaml` file:

```
mycentos:
    inherits: centos7
    image: user:mycentos
```

See also

o:ref:*pcocc-image(1)<image>*, *pcocc-templates.yaml(5)*, *pcocc-newvm-tutorial(7)*, *pcocc-ckpt(1)*, *pcocc-dump(1)*

2.4.14 Transfer files to a VM via scp

Synopsis

pcocc scp [OPTIONS] [SCP_OPTIONS]...

Description

Transfer files to a VM via scp. See the scp(1) manpage for documentation on scp options.

Warning: This requires the VM to have its ssh port reverse NAT'ed to the host in its NAT network configuration.

Options

-j, –jobid [INTEGER] Jobid of the selected cluster

-J, –jobname [TEXT] Job name of the selected cluster

–user [TEXT] Select cluster among jobs of the specified user

-h, –help Show this message and exit.

Examples

To copy a directory to vm0 of the job named *centos*:

```
pcocc scp -J centos -r dir vm0:
```

Note: By default, scp(1) uses the host username to log in. Depending on the VM configuration, it may be necessary to specify another username.

To copy a file to vm1 of the default job as the demo user:

```
pcocc scp ./foo demo@vm1:~/foo
```

To copy a file from vm2 of the default job as root:

```
pcocc scp root@vm2:~/data ./data
```

See also

scp(1), *pcocc-ssh(1)*, *pcocc-nc(1)*, *pcocc-exec(1)*, *pcocc-networks.yaml(5)*, *pcocc-9pmount-tutorial.yaml(7)*

2.4.15 Connect to a VM via ssh

Synopsis

pcocc ssh [OPTIONS] [SSH_OPTIONS]...

Description

Connect to a VM via ssh. See the ssh(1) manpage for documentation on ssh options.

Warning: This requires the VM to have its ssh port reverse NAT'ed to the host in its NAT network configuration.

Options

-j, –jobid [INTEGER] Jobid of the selected cluster

-J, –jobname [TEXT] Job name of the selected cluster

–user [TEXT] Select cluster among jobs of the specified user

-h, –help Show this message and exit.

Examples

To log in to vm0 of the job named *centos*:

```
pcocc ssh -J centos vm0
```

Note: By default, ssh(1) uses the host username to login. Depending on the VM configuration, it may be necessary to specify another username.

To log in to vm4 of the default job as root:

```
pcocc ssh root@vm4
```

See also

ssh(1), *pcocc-scp(1)*, *pcocc-console(1)*, *pcocc-nc(1)*, *pcocc-display(1)*, *pcocc-exec(1)*, *pcocc-networks.yaml(5)*, *pcocc-9pmount-tutorial.yaml(7)*

2.4.16 List and manage VM templates

Synopsis

pcocc template [COMMAND] [ARG]

Description

List and manage virtual machine templates.

Sub-Commands

list Display a list of all templates (system-wide and user-defined)

show [tpl] Show a detailed description of the template named *tpl*

Examples

To list available templates:

```
pcocc template list
```

This produces an output similar to:

```
NAME            DESCRIPTION                  RESOURCES    IMAGE
----            -----------                  ---------    -----
mydebian        My custom Debian VM          cluster      /vms/debian9-cloud
centos7-cloud   Cloud-init enabled CentOS 7  cluster      /vms/centos7-cloud
debian9-cloud   Cloud-init enabled Debian 9  cluster      /vms/debian9-ci
```

To get detailed information relative to a template:

```
pcocc template show mydebian
```

It produces an output such as:

```
ATTRIBUTE         INHERITED    VALUE
---------         ---------    -----
inherits          No           debian9-ci
user-data         No           ~/conf
image             Yes          /vms/debian9-ci
resource-set      Yes          cluster
image-revision    No           0 (Sun Jul  9 22:58:41 2017)
```

See also

pcocc-image(1), *pcocc-templates.yaml(5)*, *pcocc-resources.yaml(5)*

2.4.17 Private Cloud on a Compute Cluster

Introduction

pcocc (pronounced like "peacock") stands for Private Cloud On a Compute Cluster. It allows users of an HPC cluster to host their own clusters of VMs on compute nodes, alongside regular jobs. Users are thus able to fully customize their software environments for development, testing, or facilitating application deployment. Compute nodes remain managed by the batch scheduler as usual since the clusters of VMs are seen as regular jobs. For each virtual cluster, pcocc allocates the necessary resources to host the VMs, including private Ethernet and/or Infiniband networks, creates temporary disk images from the selected templates and instantiates the requested VMs.

Working principle

pcocc leverages SLURM to start, stop and supervise virtual clusters in the same way as regular parallel jobs. It allocates CPU and memory resources using sbatch/salloc and a SLURM plugin allows to setup virtual networks on the allocated nodes. Once the nodes are allocated and configured, VMs are launched by SLURM as any other task with the rights of the invoking user. VMs are configured to replicate, as much as possible, the resources and capabilities of the portion of the underlying host that is allocated for them (CPU model and core count, memory amount and NUMA topology, CPU and memory binding...) so as to maximize performance.

To launch a virtual cluster, the user selects a template from which to instantiate its VMs and the number of requested VMs (it is possible to combine several templates among a cluster). A template defines, among other things, the base image disk to use, the virtual networks to setup, and optional parameters such as host directories to export to the VMs via 9p. Administrators can define system-wide templates from which users can inherit to define their own templates. When a VM is instantiated from a template, an ephemeral disk image is built from the reference image using copy-on-write. By default, any changes made to the VMs' disks are therefore lost once the virtual cluster stops but it is possible to save these changes to create new revisions of the templates.

List of help topics

This documentation is organized into help topics which are listed in the following sections. These topics include tutorials to help you get started, individual pcocc sub-commands to manage and interact with virtual clusters and configuration files.

You may get further information on each of these topics listed below by doing:

pcocc help [*TOPIC*]

For example to open the newvm tutorial:

pcocc help newvm-tutorial

To example read help about the ssh sub-command:

pcocc help ssh

For installing pcocc on a cluster, have a look at the *installation guide*.[1]

[1] Local installation guide: /usr/share/doc/pcocc-0.6.2/install.html

Sub-Commands

pcocc supports the following sub-commands:

- Define and Allocate VMs:

  alloc Instantiate or restore a virtual cluster (interactive mode)

  batch Instantiate or restore a virtual cluster (batch mode)

  template List and manage VM templates

  image List and manage VM images

- Connect to VMs:

  console Connect to a VM console

  nc Connect to a VM via nc

  scp Transfer files to a VM via scp

  ssh Connect to a VM via ssh

  exec Execute commands through the pcocc guest agent

  display Display the graphical output of a VM

- Manage running VMs:

  reset Reset a VM

  ckpt Checkpoint a virtual cluster

  dump Dump the memory of a VM to a file

  monitor-cmd Send a command to the monitor

  save Save the disk of a VM

  ps List current pcocc jobs

Tutorials

newvm-tutorial How to import VM images and define VM templates

cloudconfig-tutorial How to configure cloud-init enabled VMs

9pmount-tutorial How to mount host directories in VMs

Configuration Files

batch.yaml Batch environment configuration file

networks.yaml Networks configuration file

resources.yaml Resource sets configuration file

repos.yaml Image repositories configuration file

templates.yaml VM templates definition file

See also

pcocc-alloc(1), *pcocc-batch(1)*, *pcocc-ckpt(1)*, *pcocc-console(1)*, *pcocc-display(1)*, *pcocc-dump(1)*, *pcocc-exec(1)*, *pcocc-monitor-cmd(1)*, *pcocc-image(1)*, *pcocc-nc(1)*, *pcocc-reset(1)*, *pcocc-save(1)*, *pcocc-scp(1)*, *pcocc-ssh(1)*, *pcocc-template(1)*, *pcocc-batch.yaml(5)*, *pcocc-networks.yaml(5)*, *pcocc-resources.yaml(5)*, *pcocc-repos.yaml(5)*, *pcocc-templates.yaml(5)*, *pcocc-9pmount-tutorial(7)*, *pcocc-cloudconfig-tutorial(7)*, *pcocc-newvm-tutorial(7)*

CHAPTER 3

Navigation

- search