Cluster in the cloud Documentation

Matt Williams

Apr 01, 2019

Contents:

CHAPTER 1

Set up the cloud infrastructure

1.1 Getting ready

The first step is to get a bunch of servers powered on in your cloud. We do this using a tool called Terraform.

Make sure that Terraform is installed by running the following in the command-line:

```
$ terraform version
```

you should get output like:

```
Terraform v0.11.8
```

We're now ready to start configuring our infrastructure.

1.2 Setting the config

Start by making a new directory which will hold all our configuration. We will refer to this directory as the *base config directory*. Change to that directory in your terminal.

Grab the Terraform config from Git using:

```
$ git clone https://github.com/ACRC/oci-cluster-terraform.git
```

Now move into that directory and initialise the Terraform repo:

```
$ terraform init
```

Now, when you check the Terraform version, you should see the OCI provider showing up:

```
$ terraform version
Terraform v0.11.8
+ provider.null v1.0.0
+ provider.oci v3.2.0
+ provider.tls v1.2.0
```

Rename the example config file `terraform.tfvars.example` to `terraform.tfvars` and open it in a text editor:

```
$ mv terraform.tfvars.example terraform.tfvars
$ vim terraform.tfvars
```

Following the instructions at the Oracle Terraform plugin docs, set the values of `tenancy_ocid`, `user_ocid`, `private_key_path`, `fingerprint` and `region`. Make sure that the user account you use for `user_ocid` has admin access in your tenancy to create infrastructure.

You will also need to set the compartment OCID of the compartment that you are using. If you are using the default root compartment, this will be the same as your tenancy OCID.

The next thing to set is an SSH key that you will use to connect to the server once it is built. See GitHub's documentation on information on how to do this and then paste the contents of the public key into the `ssh_public_key` config variable between the two `EOF`s.

Finally, you need to decide what type of machines will make up your cluster. This is dependent on what shapes you have access to so check your service limits in the OCI web console. You will want a simple, lightweight VM for the management node and a set of more powerful VMs (or better, bare metal) machines for the compute nodes. For this tutorial, we will use `VM.Standard2.16` for the management node and 4 `VM.Standard2.24` for the compute nodes but it will depend on what you have access to.

Set the `ManagementShape` config variable to the shape you want for the management node:

```
ManagementShape = "VM.Standard2.16"
```

To set the compute nodes, there are two config variables we need to set. The variable `ComputeShapes` contains a list of all the shapes for each node and `InstanceADIndex` contains a list of numbers referring to the availability domain each node should be in:

```
InstanceADIndex = ["1", "1", "1", "1"]
ComputeShapes = ["VM.Standard2.24", "VM.Standard2.24", "VM.Standard2.24", "VM.
↪Standard2.24"]
```

You see that there are two lists, each with four elements. The n^{th} element in each list are related to each other. Once the nodes are created, they will be named `compute001`, `compute002` etc. in the order they are listed here.

If we instead wanted a `BM.GPU2.2` in AD 1, three `BM.Standard1.36` in AD 2 and one `BM.DenseIO1.36` in AD3 we would instead write:

```
InstanceADIndex = ["1", "2", "2", "2", "3"]
ComputeShapes = ["BM.GPU2.2", "BM.Standard1.36", "BM.Standard1.36", "BM.Standard1.36",
↪ "BM.DenseIO1.36"]
```

Finally, we need to tell Terraform about all of the ADs that we are putting this in to make sure that the networking is working correctly. Set `ADS` to a list of all the availability domains that we have put infrastructure in:

```
ADS = ["1"]
```

That has defined the types and location of all the nodes we are installing. We need to tell OCI what OS to install onto each machine which we do by setting `ComputeImageOCID` and `ManagementImageOCID`. To decide what

values to put in these, look at OCI's list of images. We will install the latest version of Oracle Linux onto each:

```
ComputeImageOCID = {
  VM.Standard2.24 = {
    eu-frankfurt-1 = "ocid1.image.oc1.eu-frankfurt-1.
↪aaaaaaaa7qdjjqlvryzxx4i2zs5si53edgmwr2ldn22whv5wv34fc3sdsova"
  }
}
ManagementImageOCID = {
  eu-frankfurt-1 = "ocid1.image.oc1.eu-frankfurt-1.
↪aaaaaaaa7qdjjqlvryzxx4i2zs5si53edgmwr2ldn22whv5wv34fc3sdsova"
}
```

At this point, we are ready to provision our infrastructure. Check that there's no immediate errors with

```
$ terraform validate
```

It should return with no errors. If there are any problems, fix them before continuing.

Next, check that Terraform is ready to run with

```
$ terraform plan
```

which should have, near the end, something like `Plan: 13 to add, 0 to change, 0 to destroy.`.

We're now ready to go. Run

```
$ terraform apply
```

and, when prompted, tell it that "yes", you do want to apply.

It will take some time but should return without any errors with something green that looks like:

```
Apply complete! Resources: 13 added, 0 changed, 0 destroyed.

Outputs:

ComputeHostnames = [
    compute001,
    compute002,
    compute003,
    compute004
]
ManagementPublicIPs = [
    130.61.43.69
]
```

You are now ready to move on to *installing the software on the cluster*.

CHAPTER 2

Finalising the setup

Terraform will have automatically started the cluster software configuration step. It will run in the background and will take some time to complete. In the meantime, you can connect to the custer and follow its progress.

2.1 Final configuration step

You can log into the management node at `yourusername@mgmtipaddress`, using the IP address that terraform printed at the end of its run. You can forward your SSH key from your workstation to avoid copying the private key in to the cloud. First test that you can talk to the SSH agent by asking it to list the keys:

```
$ ssh-add -L
```

Then connect to the managment node. For example:

```
$ ssh -A opc@130.61.43.69
```

Once logged in, you can run the `finish` script:

```
[opc@mgmt ~]$ ./finish
```

It will most likely tell you that the nodes have not finished configuring. If the `finish` script is not there, wait a minute or two and it should appear.

To follow the progress, you can look at the file `ansible-pull.log` in `opc`'s home directory.

You can keep on running trying to run `finish` until all nodes have finished configuring. Once they have, you need to tell the system about what user accounts you want to create.

Copy the `users.yml.example` file to `users.yml`:

```
[opc@mgmt ~]$ cp users.yml.example users.yml
```

and edit it to contain the users you want. For the `key` attribute you can specify a URL of a file which contains a list of public keys (such as provided by GitHub) or explicitly provide a public key inline. For example, it might look like:

```
---
users:
  - name: matt
    key: https://github.com/milliams.keys
  - name: anotheruser
    key: ssh-rsa␣
↪UmFuZG9tIGtleSBjb250ZW50cy4gUHV0IHlvdXIgb3duIGtleSBpbiBoZXJlIG9idmlvdXNseS4=␣
↪user@computer
```

Run `finish` again and it should create those users across the system:

```
[opc@mgmt ~]$ ./finish
```

Once it has succeeded, log out and try logging as one of those users.

2.2 Check Slurm is running

```
$ ssh -A matt@130.61.43.69
```

Once logged in, try running the `sinfo` command to check that Slurm is running:

```
[matt@mgmt ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
compute*     up   infinite      4   idle compute[001-004]
```

Brilliant! Start submitting jobs.

Check out the information on *running the cluster*.

CHAPTER 3

Running the cluster

Now that you have a cluster which is up and running, it's worth knowing what you can do with it.

3.1 Slurm jobs

A full Slurm tutorial is outside of the scope of this document but it's configured in a fairly standard way. By default there's one single partition called `compute` which contains all the compute nodes.

A simple first Slurm script, `test.slm`, could look like:

```
#! /bin/bash

#SBATCH --job-name=test
#SBATCH --partition=compute
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=1
#SBATCH --time=10:00
#SBATCH --exclusive

echo start
srun -l hostname
echo end
```

which you could run with:

```
[matt@mgmt ~]$ sbatch test.slm
```

3.2 Slurm elastic scaling

Slurm is configured to use its elastic computing mode. This allows Slurm to automatically turn off any nodes which are not currently being used for running jobs and turn on any nodes which are needed for running jobs. This is particularly

useful in the cloud as a node which has been shut down will not be charged for.

Slurm does this by calling a script `/usr/local/bin/startnode` as the `slurm` user. If necessary, you can call this yourself from the `opc` user like:

```
[opc@mgmt ~]$ sudo -u slurm /usr/local/bin/startnode compute001
```

to turn on the node `compute001`.

You should never have to do anything to explicitly shut down the cluster, it will automatically turn off all nodes which are not in use after a timeout. The management node will always stay running which is why it's worth only using a relatively cheap VM for it.

Warning: Currently, due to a quirk in OCI, it seems that while all VMs and most bare-metal nodes are not charged for while *stopped*, the DenseIO nodes *are*. This means that the auto-shutdown will not work as well for those shapes and **you will be charged**. Development is ongoing to avoid this.

The rate at which Slurm shuts down is managed in `/mnt/shared/apps/slurm/slurm.conf` by the `SuspendTime` parameter. See the slurm.conf documentation for more details.

3.3 Cluster shell

A common task is to want to run commands across all nodes in a cluster. By default you have access to clustershell. Read the documentation there to get details of how to use the tool.

The gist is that you give it a hostname or a group and a command to run. You can see a list of the available groups with `cluset`:

```
[opc@mgmt ~]$ cluset --list-all
@compute
@state:idle
@role:mgmt
```

You can then run a command with `clush`:

```
[opc@mgmt ~]$ clush -w @compute uname -r
compute001: 3.10.0-862.2.3.el7.x86_64
compute002: 3.10.0-862.2.3.el7.x86_64
compute003: 3.10.0-862.2.3.el7.x86_64
compute004: 3.10.0-862.2.3.el7.x86_64
```

You can combine the output from different nodes using the `-b` flag:

```
[opc@mgmt ~]$ clush -w @compute -b uname -r
---------------
compute[001-004] (4)
---------------
3.10.0-862.2.3.el7.x86_64
```

3.4 Installing software on your cluster

In order to do any actual work you will likely need to install some software. There are many ways to get this to work but I would recommend either using `clush` to install the software or, preferably, create a local Ansible playbook

which installs it for you across the cluster.

In the latter case, you can use `/home/opc/hosts` as an inventory file and point your playbook to use it.

3.5 Performance metrics

The cluster automatically collects data from all the nodes and makes them available in a web dashboard.

It is available at the IP address of you management node on port 3000. Point your browser at http://your.mgmt.ip.address:3000 and log in with the username *admin* and the password *admin*. You will be prompted to create a new password before you continue.

3.6 Destroying the whole cluster

> **Warning:** Please bear in mind that this will also destroy your file system which contains your user's home area and any data stored on the cluster.

When you've completely finished with the cluster, you can destroy it using Terraform.

```
$ terraform destroy
```

Welcome to the documentation for cluster in the cloud. By the end of this you will have a fully-operational, elastically-scaling Slurm cluster running on cloud resources.

In the future, the intention is that this tutorial will cover installing on all major cloud providers but for now only Oracle Public Cloud is covered.

This tutorial was created by Matt Williams at the ACRC in Bristol. Contributions to this document are welcome at GitHub.

CHAPTER 4

Prerequisites

To complete this tutorial you will need:

- access to a command-line (i.e. Linux, MacOS Terminal or WSL)
- an SSH key pair
- an account with credit on Oracle cloud
  - the account must have admin permissions to create infrastructure
- local software installed
  - Terraform 0.11
  - SSH

Start by *creating the infrastructure*.