Building your own CoreOS Army with Vagrant, because Orchestration Rocks

I'm all about saving time, money, and making things simple to use. Luckily, CoreOS and Vagrant have become a big part of making that happen for me lately with a lot of work I've been doing in and out of my day-to-day lab experiences. It was especially exciting for me today as I had an opportunity to meet with Alex Polvi, CEO of CoreOS at a Turbonomic event in San Francisco.

I'm very pleased with the work happening at CoreOS, around both the core product, as well as the work happening on the container side of the ecosystem with Rocket.

This is a quick primer to allow you to get some CoreOS goodness going in your lab using the simple and freely available Vagrant and VirtualBox tools that you may already be running.

We will need 3 things:

1. Git
2. Vagrant 1.6 or higher
3. VirtualBox 4.3.10 or higher

The goal of what this great little Vagrant build does is to enable you to spin up a lab running one or more CoreOS instances to kick the tires on what you can do with CoreOS. Not only that, but you can also build infrastructure which is clustered and distributed using the very easy configuration available right in the Vagrant configuration.

Step 1 – Pull down the Vagrant configuration to your system

It's as simple as running a git clone as shown here:

```
git clone https://github.com/coreos/coreos-vagrant/

cd coreos-vagrant
```

Now that you have the code on your system, we can edit the `config.rb` file to decide how many instances you want to run. By default, there will be a single instance launched.

In this case, let's spin up a 3-node configuration just to see what a multi-node environment looks like. There is a lot that we can do, but it is important that we take the first steps to just see how to get to the start line.

Step 2 – Edit your Vagrantfile to set your network

I'm assuming that you're aware of your available networks on your lab, so you will want to pick one to put in place for your CoreOS cluster. In my case, I'm editing the `Vagrantfile` and using the 10.200.1.0/24 network. You will find the line in the file towards the bottom as you can see here as `ip = "10.200.1.#{i+100}"`:

Now that we have our network defined, we have one more simple step to get closer to our CoreOS cluster deployment.

Step 3 – Edit your config.rb file

Almost there! Just open up your config.rb file and set the `$num_instances=3` to define the number of CoreOS instances you are going to launch. I've done this for a lot to confirm it works, and it's really almost too simple. That's a good thing

Step 4 – Vagrant up

You've made it! Now that we have our `config.rb` set for the number of instances, and the `Vagrantfile` configured for our network of choice, it's time to launch with the `vagrant up` command. You're going to see some messages scrolling by during the process, and then in the end we will have 3 brand new CoreOS instances in a matter of a couple of minutes (or less!).

We want to confirm what's been done as usual, and luckily all of the vagrant commands are applicable for our CoreOS instances too. We can confirm the status with our `vagrant status` command:

Next we will run our `vagrant ssh core-01` to confirm we have an active network. This gives us an SSH connection to the first of the three instances so that we are running commands from there. Once we are connected, you can see that the shell prompt is changed to `core@localhost ~ $` which is our first indicator. My preference is to always run an `ifconfig` to ensure the network interfaces are showing the right IP address:

Let's do one more quick test to confirm our IP connectivity to the gateway and external network by doing a `ping 8.8.8.8` to illustrate a working network connection.

As if by magic, or in this case Vagrant, we have a running set of CoreOS instances which we can now use for any purpose. This brings us to the start line as i like to say.

Why CoreOS?

CoreOS has a lot of very interesting use-cases. You may have one, two, or many reasons why you will want to use CoreOS as a part of your infrastructure offering. What we are going to explore in the next posts from here is a few distinct use-cases, how to deal with those using CoreOS as the solution, and from there you can get more comfortable with the abilities baked into CoreOS.

In what we've done here, these are a set of worker instances. The important next step will be to create the cluster management configuration and put our little CoreOS army to work.

We will also do a little extended work with both Docker on CoreOS, as well as with the new Rocket offering from CoreOS. In fact, there was a brand new update at the time of this writing.

I hope that this is a helpful start, and I look forward to bringing more CoreOS content. Feel free to drop me a line by reaching out on Twitter (@DiscoPosse) or leaving a comment here so that I can help to answer any questions about what we can do.