Appendix A. Building a Raspberry Pi Kubernetes Cluster

While Kubernetes is often experienced through the virtual world of public cloud computing, where the closest you get to your cluster is a web browser or a terminal, it can be a very rewarding experience to physically build a Kubernetes cluster on bare metal. Likewise, nothing compares to physically pulling the power or network on a node and watching how Kubernetes reacts to heal your application to convince you of its utility.

Building your own cluster might seem like both a challenging and an expensive effort, but fortunately it is neither. The ability to purchase low-cost, system-on-chip computer boards as well as a great deal of work by the community to make Kubernetes easier to install mean that it is possible to build a small Kubernetes cluster in a few hours.

In the following instructions, we focus on building a cluster of Raspberry Pi machines, but with slight adaptations the same instructions could be made to work with a variety of different single-board machines.

Parts List

The first thing you need to do is assemble the pieces for your cluster. In all of the examples here, we'll assume a four-node cluster. You could build a cluster of three nodes, or even a cluster of a hundred nodes if you wanted to, but four is a pretty good number.

To start, you'll need to purchase (or scrounge) the various pieces needed to build the cluster. Here is the shopping list, with some approximate prices as of the time of writing:

1. Four Raspberry Pi 3 boards (Raspberry Pi 2 will also work) — $160
2. Four SDHC memory cards, at least 8 GB (buy high-quality ones!) — $30–50
3. Four 12-inch Cat. 6 Ethernet cables — $10
4. Four 12-inch USB A–Micro USB cables — $10
5. One 5-port 10/100 Fast Ethernet switch — $10
6. One 5-port USB charger — $25
7. One Raspberry Pi stackable case capable of holding four Pis — $40 (or build your own)
8. One USB-to-barrel plug for powering the Ethernet switch (optional) — $5

The total for the cluster comes out to be about $300, which you can drop down to $200 by building a three-node cluster and skipping the case and the USB power cable for the switch (though the case and the cable really clean up the whole cluster).

One other note on memory cards: do not scrimp here. Low-end memory cards behave unpredictably and make your cluster really unstable. If you want to save some money, buy a smaller, high-quality card. High-quality 8 GB cards can be had for around $7 each online.

Anyway, once you have your parts, you’re ready to move on to building the cluster.

These instructions also assume that you have a device capable of flashing an SDHC card. If you do not, you will need to purchase a USB → memory card reader/writer.

Flashing Images

The default Raspbian image now supports Docker through the standard install methods, but to make things even easier, the Hypriot project provides images with Docker preinstalled.

Visit the Hypriot downloads page and download the latest stable image. Unzip the image, and you should now have an *.img* file. The Hypriot project also provides really excellent documentation for writing this image to your memory card:

- macOS
- Windows
- Linux

Write the same image onto each of your memory cards.

First Boot: Master

The first thing to do is to boot just your master node. Assemble your cluster, and decide which is going to be the master node. Insert the memory card, plug the board into an HDMI output, and plug a keyboard into the USB port.

Next, attach the power to boot the board.

Log in at the prompt using the username `pirate` and the password `hypriot`.

> **WARNING**
>
> The very first thing you should do with your Raspberry Pi (or any new device) is to change the default password. The default password for every type of install everywhere is well known by people who will misbehave given a default login to a system. This makes the internet less safe for everyone. Please change your default passwords!

Setting Up Networking

The next step is to set up networking on the master.

First, set up WiFi. This is going to be the link between your cluster and the outside world. Edit the */boot/device-init.yaml* file. Update the WiFi SSID and password to match your environment. If you ever want to switch networks, this is the file you need to edit. Once you have edited this, reboot with `sudo reboot` and validate that your networking is working.

The next step in networking is to set up a static IP address for your cluster's internal network. To do this, edit */etc/network/interfaces.d/eth0* to read:

```
allow-hotplug eth0
iface eth0 inet static
    address 10.0.0.1
    netmask 255.255.255.0
    broadcast 10.0.0.255
    gateway 10.0.0.1
```

This sets the main Ethernet interface to have the statically allocated address 10.0.0.1.

Reboot the machine to claim the 10.0.0.1 address.

Next, we're going to install DHCP on this master so it will allocate addresses to the worker nodes. Run:

```
$ apt-get install isc-dhcp-server
```

Then configure the DHCP server as follows:

```
# Set a domain name, can basically be anything
option domain-name "cluster.home";

# Use Google DNS by default, you can substitute ISP-supplied values here
option domain-name-servers 8.8.8.8, 8.8.4.4;

# We'll use 10.0.0.X for our subnet
subnet 10.0.0.0 netmask 255.255.255.0 {
    range 10.0.0.1 10.0.0.10;
    option subnet-mask 255.255.255.0;
    option broadcast-address 10.0.0.255;
    option routers 10.0.0.1;
}
default-lease-time 600;
max-lease-time 7200;
authoritative;
```

Restart the DHCP server with `sudo systemctl restart dhcpd`.

Now your machine should be handing out IP addresses. You can test this by hooking up a second machine to the switch via the Ethernet. This second machine should get the address 10.0.0.2 from the DHCP server.

Remember to edit the */boot/device-init.yaml* file to rename this machine to `node-1`.

The final step in setting up networking is setting up network address translation (NAT) so that your nodes can reach the public internet (if you want them to be able to do so).

Edit */etc/sysctl.conf* and set `net.ipv4.ip_forward=1` to turn on IP forwarding.

Then edit */etc/rc.local* (or the equivalent) and add `iptables` rules for forwarding from `eth0` to `wlan0` (and back):

```
$ iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
$ iptables -A FORWARD -i wlan0 -o eth0 -m state \
  --state RELATED,ESTABLISHED -j ACCEPT
$ iptables -A FORWARD -i eth0 -o wlan0 -j ACCEPT
```

At this point, basic networking setup should be complete. Plug in and power up the remaining two boards (you should see them assigned the addresses 10.0.0.3 and 10.0.0.4). Edit the */boot/device-init.yaml* file on each machine to name them `node-2` and `node-3`, respectively.

Validate this by first looking at */var/lib/dhcp/dhcpd.leases* and then SSH to the nodes (remember again to change the default password first thing). Validate that the nodes can connect to the external internet.

Extra credit

There are a couple of extra things in networking that make it easier to manage your cluster.

The first is to edit */etc/hosts* on each machine to map the names to the right addresses. On each machine, add:

```
...
10.0.0.1 kubernetes
10.0.0.2 node-1
10.0.0.3 node-2
```

```
10.0.0.4 node-3
...
```

Now you can use those names when connecting to those machines.

The second is to set up passwordless SSH access. To do this, run `ssh-keygen` and then copy the *$HOME/.ssh/id_rsa.pub* file into `/home/pirate/.ssh/authorized_keys` on `node-1`, `node-2`, and `node-3`.

Installing Kubernetes

At this point you should have all nodes up, with IP addresses and capable of accessing the internet. Now it's time to install Kubernetes on all of the nodes.

Using SSH, run the following commands on all nodes to the `kubelet` and `kubeadm` tools. You will need to be root for the following commands. Use `sudo su` to elevate to the root user.

First, add the encryption key for the packages:

```
# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
```

Then add the repository to your list of repositories:

```
# echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" \
  >> /etc/apt/sources.list.d/kubernetes.list
```

Finally, update and install the Kubernetes tools. This will also update all packages on your system for good measure:

```
# apt-get update
$ apt-get upgrade
$ apt-get install -y kubelet kubeadm kubectl kubernetes-cni
```

Setting Up the Cluster

On the master node (the one running DHCP and connected to the internet), run:

```
$ kubeadm init --pod-network-cidr 10.244.0.0/16 \
--api-advertise-addresses 10.0.0.1
```

Note that you are advertising your internal-facing IP address, not your external address.

Eventually, this will print out a command for joining nodes to your cluster. It will look something like:

```
$ kubeadm join --token=<token> 10.0.0.1
```

SSH onto each of the worker nodes in your cluster and run that command.

When all of that is done, you should be able to run and see your working cluster:

```
$ kubectl get nodes
```

Setting up cluster networking

You have your node-level networking setup, but you need to set up the pod-to-pod networking. Since all of the nodes in your cluster are running on the same physical Ethernet network, you can simply set up the correct routing rules in the host kernels.

The easiest way to manage this is to use the Flannel tool created by CoreOS. Flannel supports a number of different routing modes; we will use the `host-gw` mode. You can download an example configuration from the Flannel project page:

```
$ curl https://rawgit.com/coreos/flannel/master/Documentation/kube-flannel.yml \
  > kube-flannel.yaml
```

The default configuration that CoreOS supplies uses `vxlan` mode instead, and also uses the AMD64 architecture instead of ARM. To fix this, open up that configuration file in your favorite editor; replace `vxlan` with `host-gw` and replace all instances of `amd64` with `arm`.

You can also do this with the `sed` tool in place:

```
$ curl https://rawgit.com/coreos/flannel/master/Documentation/kube-flannel.yml \
|  sed "s/amd64/arm/g" | sed "s/vxlan/host-gw/g" \
  > kube-flannel.yaml
```

Once you have your updated *kube-flannel.yaml* file, you can create the Flannel networking setup with:

```
$ kubectl apply -f kube-flannel.yaml
```

This will create two objects, a `ConfigMap` used to configure Flannel and a `DaemonSet` that runs the actual Flannel daemon. You can inspect these with:

```
$ kubectl describe --namespace=kube-system configmaps/kube-flannel-cfg
$ kubectl describe --namespace=kube-system daemonsets/kube-flannel-ds
```

Setting up the GUI

Kubernetes ships with a rich GUI. You can install it by running:

```
$ DASHSRC=https://raw.githubusercontent.com/kubernetes/dashboard/master/
$ curl -sSL \
  $DASHSRC/src/deploy/kubernetes-dashboard.yaml \
  | sed "s/amd64/arm/g" \
  | kubectl apply -f -
```

To access this UI, you can run `kubectl proxy` and then point your browser to *http://localhost:8001/ui*, where *localhost* is local to the master node in your cluster. To view this from your laptop/desktop, you may need to set up an SSH tunnel to the root node using `ssh -L8001:localhost:8001 <master-ip-address>`.

Summary

At this point you should have a working Kubernetes cluster operating on your Raspberry Pis. This can be great for exploring Kubernetes. Schedule some jobs, open up the UI, and try breaking your cluster by rebooting machines or disconnecting the network.