Life on the Edge: A First Look at Rancher's Lightweight Kubernetes Distro K3s

At ParkBee, we are taking our first steps into performing containerized deployments in our edge locations. Since we are a mobility hub provider, the need to deploy applications which contain our business logic is crucial to making our locations work as expected.

ParkBee uses Kubernetes for our deployments in Amazon Web Services (AWS) using Kops at the moment. While this kind of setup works well for our cloud-based services, for our edge deployments, this is not as simple. **Our ideal goal would be:** have a Kubernetes cluster comprised of edge Kubernetes nodes at each of our locations, with the Kubernetes master nodes present in AWS.

This past week, Rancher Labs made an announcement that they had released the first version of their new Kubernetes distribution, K3s. Essentially the idea is to make Kubernetes easier to install and maintain for low-resource computing platforms like the Raspberry Pi.

You can read the full annoucement here: Rancher Labs Introduces Lightweight Distribution of Kubernetes to Simplify Operations in Low-Resource Computing Environments.

Essentially, K3s promises to be a lightweight, easy-to-use Kubernetes provisioner using only a single binary. Here's a blurb about the key features from the annoucement:

> Key features include:
>
> - **Production-grade Kubernetes:** K3s is a standards-compliant, Kubernetes distribution engineered for mission-critical, production use cases.
> - **One binary with zero host dependencies:** Everything necessary to install Kubernetes on any device is included in a single, 40mb binary. There is no requirement for an external installer like KubeSpray, KubeADM or RKE. With a single command, a single-node k3s cluster can be provisioned or upgraded.
> - **Simple to add nodes to a cluster:** To add additional nodes to a cluster, admins run a single command on the new node, pointing it to the original server and passing through a secure token.

> - **Automatic certificate generation:** All of the certificates needed to establish TLS between the Kubernetes masters and nodes are automatically created when a cluster is launched. Encryption keys for service accounts are also automatically created.

Prerequisites

For this article, I'll be using the first official version of the K3s binary (taken from tag v0.1.0) that was referred to in the press announcement. If you want to replicate this, you'll need the following:

- Raspberry Pi 3 Model B+ with Raspbian Stretch Lite flashed on a microSD card.
- A local network; for ease, I'll just be using my LAN at home.
- Vagrant installed locally on your laptop; this could also be Docker for Mac, but essentially, the k3s binary was only built for Linux, arm64, and armhf architectures.

Vagrant

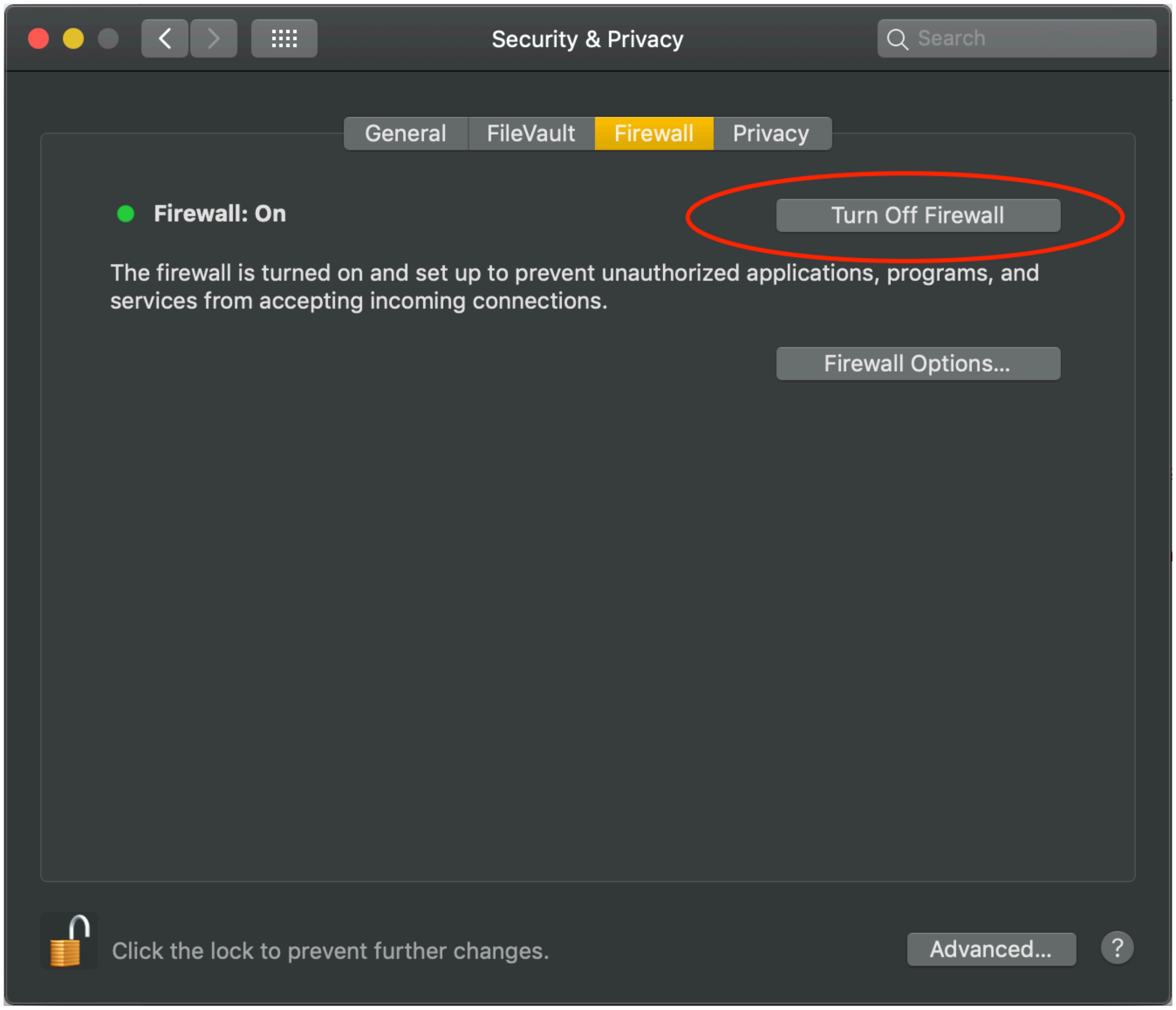
If you're using Vagrant, you can create a `Vagrantfile` in your test directory with the following contents:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANT_API = 2

Vagrant.configure(VAGRANT_API) do |config|
  config.vm.box = "bento/ubuntu-18.04"
  config.vm.box_check_update = false
  config.vm.network "forwarded_port", guest: 6443, host: 6443, host_ip: "0.0.0.0"

  config.vm.provider "virtualbox" do |vb|
    vb.cpus = 1
    vb.gui = true
    vb.memory = "2048"
    vb.name = "k3s-master"
  end

  config.vm.provision :docker

  config.vm.provision "shell", inline: <<-SHELL
    sudo modprobe vxlan
    curl -sfL https://get.k3s.io | sh -
    hostnamectl set-hostname k3s-master
  SHELL
end
```

This will install Docker, as well as the K3s binary. The install script will also conveniently symlink the `kubectl` binary to `k3s` as well since it's built-in.

Firewalling

A quick note: the Raspberry Pi will need to connect to your local machine in your LAN. In macOS, make sure to go to **Settings => Security and Privacy => Firewall**, and then click the button that says **Turn Off**.



Firewall: On

The firewall is turned on and set up to prevent unauthorized applications, programs, and services from accepting incoming connections.

Click the lock to prevent further changes.

For Windows, you can reference [Lifewire's article] for turning off the firewall in Windows.

Running the K3s master

We'll be using the Vagrant machine as the K3s master node; once this is working, we will then attempt to connect the Raspberry Pi to the K3s master node on the local LAN.

As with any Vagrant machine, simply run `vagrant up` to get things started. Vagrant will run the K3s automatic installation script, open the port **6443** on your local machine for K3s nodes to join, and create the join token needed for later.

First, verify that the master installation was successful:

```
root@k3s-master:~# kubectl get nodes
NAME         STATUS   ROLES    AGE     VERSION
k3s-master   Ready    <none>   4m51s   v1.13.3-k3s.6
```

By default, the K3s installation script **does not** label `k3s-master` as a master node; since the `kubectl` binary is already pre-installed, we can take care of that now:

```
root@k3s-master:~# kubectl label node k3s-master kubernetes.io/role=master
node/k3s-master labeled
root@k3s-master:~# kubectl label node k3s-master node-
role.kubernetes.io/master=""
node/k3s-master labeled
```

The K3s installation also **does not** taint the master nodes for **NoSchedule**. For this test, we want to ensure that the Raspberry Pi receives the test deployment. Taint the master node using the following:

```
root@k3s-master:~# kubectl taint nodes k3s-master node-
role.kubernetes.io/master=effect:NoSchedule
node/k3s-master tainted
```

Next, we'll need the token that's used for joining K3s nodes to the new master. The `k3s server` command should have already created this for you at `/var/lib/rancher/k3s/server/node-token`. Run the following command:

```
root@k3s-master:~# cat /var/lib/rancher/k3s/server/node-token
<some-long-node-token>
```

Running the K3s node on Raspberry Pi

First, we need to prepare the Pi with some initial steps. First, disable swap using the following commands:

```
dphys-swapfile swapoff && \
dphys-swapfile uninstall && \
update-rc.d dphys-swapfile remove
```

Then, append the following text to the first line in `/boot/cmdline.txt`:

```
cgroup_enable=cpuset cgroup_memory=1 cgroup_enable=memory
```

Afterwards, issue a reboot on the Raspberry Pi. When it returns, log back in, and run the following command to download the `k3s` binary.

```
curl -fSL "https://github.com/rancher/k3s/releases/download/v0.1.0/k3s-armhf" \
  -o /usr/local/bin/k3s && \
chmod +x /usr/local/bin/k3s
```

It's not required to install Docker since K3s uses `containerd`, but it's useful to be able to verify that the pods are actually running. Docker can be installed quickly by running the following command:

```
curl -fsSL https://get.docker.com | sh - && \
usermod -aG docker pi
```

Grab the token created from the master server, and export it as an environment variable:

```
export NODE_TOKEN="<some-long-node-token>"
```

And finally, run the `k3s agent` command to start the agent, and join the master node. In my case, **192.168.0.10** is the address of the Vagrant machine running on my local laptop in my network. Make sure to replace that value with the appropriate address.

```
k3s agent \
--docker \
--server https://192.168.0.10:6443 \
--token ${NODE_TOKEN} \
> /root/logs.txt 2>&1 &
```

Similar to the K3s master node, the installation **does not** label the Raspberry Pi with the proper node labels. On the **k3s-master**, run the following commands after the Raspberry Pi has joined the cluster.

```
root@k3s-master:~# kubectl label node raspberrypi kubernetes.io/role=node
node/raspberrypi labeled
root@k3s-master:~# kubectl label node raspberrypi node-role.kubernetes.io/node=""
node/raspberrypi labeled
```

If the Raspberry Pi successfully joined, you should see something like the following when running the command on the master server:

```
root@k3s-master:~# kubectl get nodes
NAME                                       STATUS   ROLES    AGE   VERSION
raspberrypi                                Ready    node     2m    v1.13.3-k3s.6
k3s-master                                 Ready    master   20m   v1.13.3-k3s.6
```

Deploying a test NGINX container

In order to make sure that the K3s cluster is actually working, we can deploy a test NGINX pod and `NodePort` service to make sure that the Raspberry Pi creates the pods, and successfully opens the port.

On the K3s master Vagrant machine, create a file at `/root/nginx-test.yaml` with the following contents:

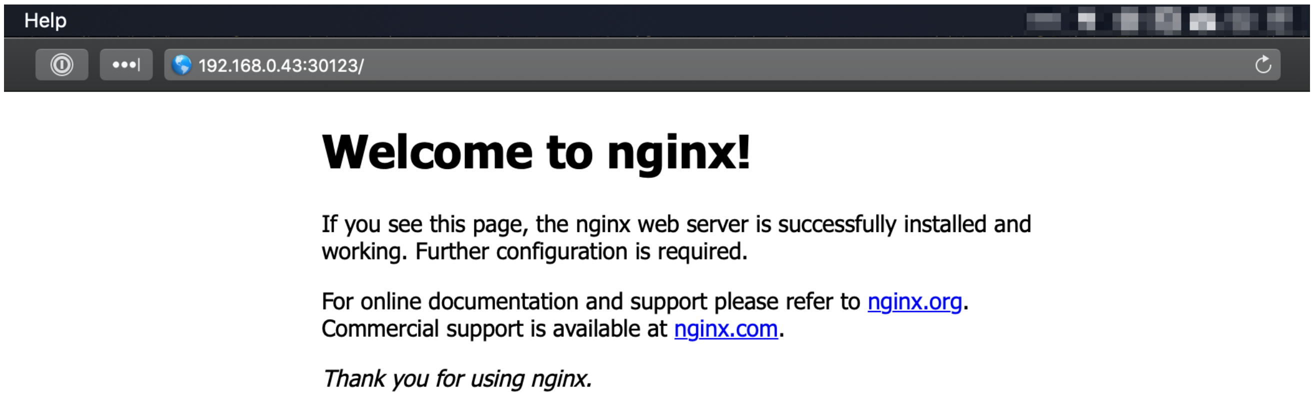
```
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-unprivileged-test
  namespace: default
spec:
  type: NodePort
  selector:
    app: nginx-unprivileged-test
  ports:
  - protocol: TCP
    nodePort: 30123
    port: 8080
    name: http
    targetPort: 8080
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-unprivileged-test
  namespace: default
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx-unprivileged-test
    spec:
      containers:
      - image: nginxinc/nginx-unprivileged
```

```
        name: nginx-unprivileged-test
        ports:
        - containerPort: 8080
          name: http
        livenessProbe:
          httpGet:
            path: /
            port: http
          initialDelaySeconds: 3
          periodSeconds: 3
```

And then finally deploy it to the cluster:

```
root@k3s-master:~# kubectl apply -f /root/nginx-test.yaml
service/nginx-unprivileged-test created
deployment.extensions/nginx-unprivileged-test created
```

Since this is a `NodePort` service, K3s will open a port on the Raspberry Pi at `30123`. On my local network, the Raspberry Pi is located on **192.168.0.43**.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Conclusion

K3s looks to be a very promising project for using Kubernetes in edge locations. A few notes that I made while progressing through this tutorial that might catch you along the way:

- While deploying the NGINX test container, initially I used the regular `nginx:latest` image from Docker Hub. However, it seems that K3s does not yet support ports lower than 1024. The `nginx` image by default tries to open port 80 inside of the container, and that causes issues.
- As mentioned, K3s doesn't fully implement all of the labels and taints that you would *normally* find in a Kubernetes distribution. While this isn't super crucial for testing, in production, it's important that usually pods are not deployed on master nodes. In the case of IoT edge

deployments, you want pods to always be scheduled on the worker nodes.

About ParkBee

I work for Parkbee. We develop smart tech. Our Mobility Management Solution optimizes the use of underutilized parking space to get cars off the street and KEEP YOUR CITY MOVING.

Check out Parkbee’s Careers Site (we’re hiring!), or follow our @lifeatparkbee Instagram account.