Table of Contents

Sean Griffin *Raytheon BBN Technologies*

Jupiter HOWTO

This HOWTO describes the steps taken to get the USC Jupiter Orchestrator running on a set of BBN rack mounted Linux servers. The information captures the installation, configuration, and execution instructions for the required components and provides a convenient "cookbook recipe" that can be used to get the Jupiter Orchestrator running on a set of new machines. Throughout the discussion, there are links provided to pages that contain additional details and configuration options for the various components.

1. Execution Environment

The following list provides some of the relevant details about the environment in which the Jupiter Orchestrator is run. This is for illustrative purposes only and the instructions can be easily modified to include additional (or fewer) nodes, as desired.

- 6 Rack Mounted Linux servers for kubernetes cluster (1 master node, 5 worker nodes)
  - Operating System: Ubuntu 16.04 LTS
  - Memory: 4GB RAM
- Kubernetes: v1.10.4
- Docker: v 1.13.1

2. Clone Jupiter Repository

The following commands can be used to clone the Jupiter Repository:

```
git clone --recurse-submodules https://github.com/ANRGUSC/Jupiter.git
cd Jupiter/
git submodule update --remote
```

3. Installing Kubernetes and Required Components

The Jupiter documentation identifies a set of requirements for running the Jupiter system. The following set of instructions elaborates on the information provided in the Jupiter documentation. The scripts in the following sections must be run as root or with sudo privileges.

3.1. Install Kubernetes

The following script installs the kubernetes components:

```
#!/bin/bash
```

```
apt-get update && apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF

apt-get update
apt-get install -y kubectl
```

3.2. Install pip3

The following script was used to install pip3:

```
#!/bin/bash

apt-get install python3-pip
pip3 install --upgrade pip
```

3.3. Install Required Python Packages

The following script installs the Python packages that Jupiter requires:

```
#/bin/bash

pip3 install -r Jupiter/k8_requirements.txt
```

The Jupiter/k8_requirements.txt file is provided as part of the Jupiter repo. The script, as written, requires that the Jupiter repo be located in the Jupiter directory in the script directory's location.

3.4. Install Kubernetes Tools

The following script installs the required kubernetes tools, namely kubelet and kubeadm:

```
#!/bin/bash

apt-get update && apt-get install -y apt-transport-https curl
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF

apt-get update
apt-get install -y kubelet kubeadm
```

3.5. Install Minikube

The following script install minikube:

```
#/bin/bash

curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.27.0/minikube-
linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

3.6. Docker

3.6.1. Install Docker

The following script installs Docker:

```
#/bin/bash

apt-get update
apt-get install -y docker.io
```

3.6.2. Configure Docker to Execute Commands Without sudo

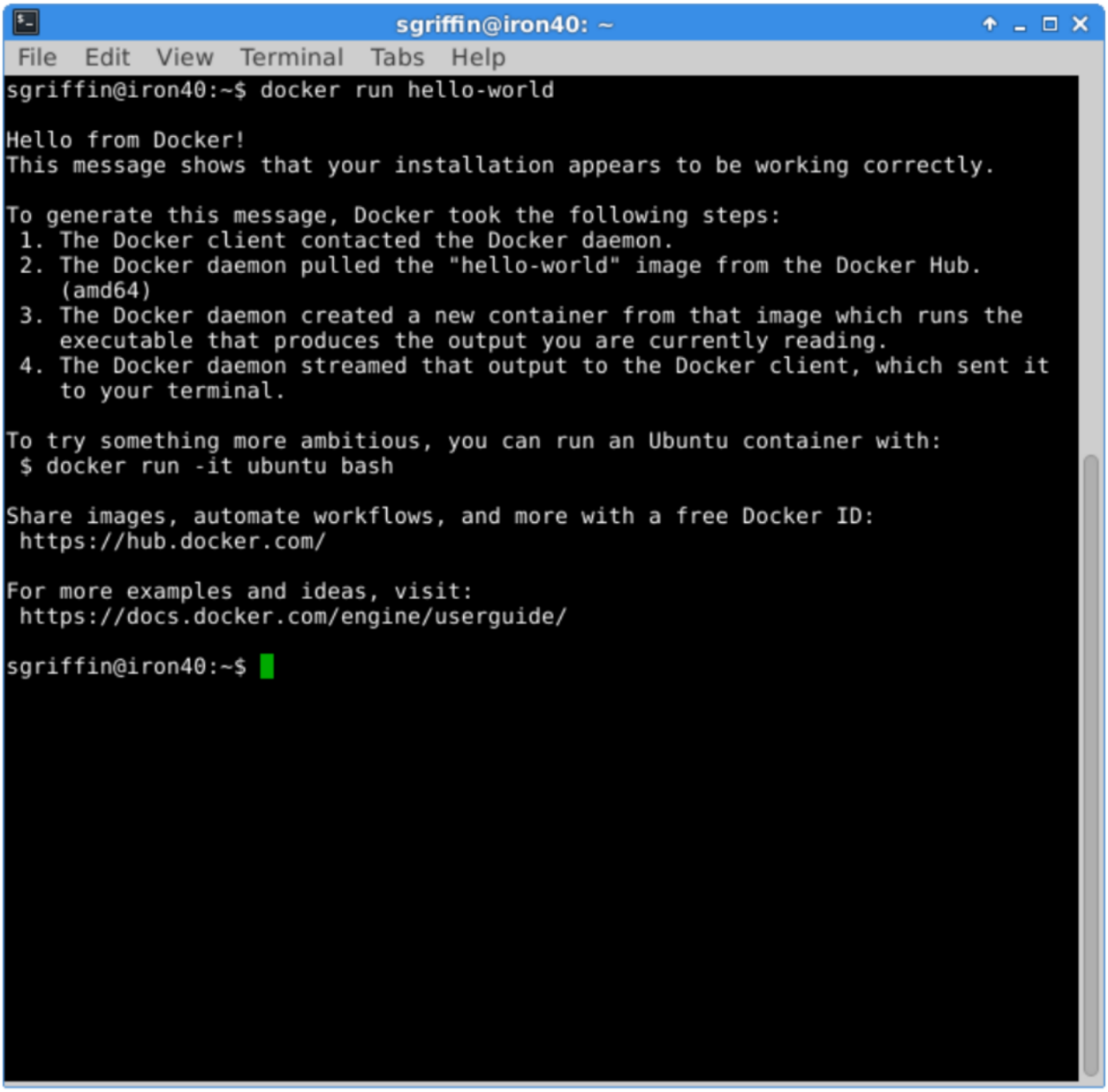
To run the Jupiter Orchestrator, it is necessary to run docker commands without sudo privileges. This can be accomplished by executing the following command:

```
sudo usermod -aG docker <username>
```

for each <username> to be added to the docker group. This can be tested by executing the following command for <username> after logging into the modified node:

```
docker run hello-world
```

The result of executing the above command is shown below:



4. Setting Up Kubernetes Cluster

The kubernetes cluster was created using **kubeadm**, as described here. The following sections go into a little more detail and try to capture the information that we accumulated by visiting many different web pages.

Now that all of the required components have been installed and configured, we can set up our kubernetes cluster, composed of a single master node and a set of worker nodes. For illustrative purposes, the master node will be iron40.bbn.com and the worker nodes will be iron43.bbn.com, iron46.bbn.com, iron53.bbn.com, iron56.bbn.com, and iron59.bbn.com. The cluster nodes will

be set up on bare metal, no virtual machines were used. All commands in this section should be run as root.

To run without error, kubernetes requires that swap be disabled. This is accomplished by executing the following command on all nodes:

```
swapoff -a
```

The nodes that will be part of the kubernetes cluster have 2 physical interfaces, a public network control plane interface and a private network data plane interface. We will configure the cluster so that all cluster and pod communications occur over the private network interface. It just happens that this is not the default route interface and will require some additional command-line options when starting the cluster components, which is described below.

4.1. Kubernetes Master Node

4.1.1. Initializing Master Node

The first step to create the kubernetes cluster is to initialize the master node. The kubeadm executable is utilized to accomplish this. Two command-line options, *--pod-network-cidr and --apiserver-advertise-address*, will be provided to kubeadm during the initialization process. The *--pod-network-cidr* option is required because flannel will be used as the pod network add-on, chosen because it is the pod network the Jupiter team uses. The *--apiserver-advertise-address* option is provided to ensure that the cluster communications occur over the private network interface and not the default public network interface.

See here for a more complete description of the **kubeadm init** command.

The following command is used to initialize the kubernetes master node:

```
export PATH=$PATH:$HOME/go/bin
kubeadm --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=172.24.201.2
init
```

The 10.244.0.0/16 address is the address that **must** be provided when flannel is used as the pod network (see flannel documentation) and the 172.24.201.2 address is the interface address of the private network on the master node.

The results of running the above command to initialize the kubernetes master node is depicted below:


root@iron40: ~
File Edit View Terminal Tabs Help
root@iron40:~# kubeadm --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=172.24.201.2 init
[init] Using Kubernetes version: v1.10.5
[init] Using Authorization modes: [Node RBAC]
[preflight] Running pre-flight checks.
[certificates] Generated ca certificate and key.
[certificates] Generated apiserver certificate and key.
[certificates] apiserver serving cert is signed for DNS names [iron40.bbn.com kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.24.201.2]
[certificates] Generated apiserver-kubelet-client certificate and key.
[certificates] Generated sa key and public key.
[certificates] Generated front-proxy-ca certificate and key.
[certificates] Generated front-proxy-client certificate and key.
[certificates] Generated etcd/ca certificate and key.
[certificates] Generated etcd/server certificate and key.
[certificates] etcd/server serving cert is signed for DNS names [localhost] and IPs [127.0.0.1]
[certificates] Generated etcd/peer certificate and key.
[certificates] etcd/peer serving cert is signed for DNS names [iron40.bbn.com] and IPs [172.24.201.2]
[certificates] Generated etcd/healthcheck-client certificate and key.
[certificates] Generated apiserver-etcd-client certificate and key.
[certificates] Valid certificates and keys now exist in "/etc/kubernetes/pki"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/admin.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/kubelet.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/controller-manager.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/scheduler.conf"
[controlplane] Wrote Static Pod manifest for component kube-apiserver to "/etc/kubernetes/manifests/kube-apiserver.yaml"
[controlplane] Wrote Static Pod manifest for component kube-controller-manager to "/etc/kubernetes/manifests/kube-controller-manager.yaml"
[controlplane] Wrote Static Pod manifest for component kube-scheduler to "/etc/kubernetes/manifests/kube-scheduler.yaml"
[etcd] Wrote Static Pod manifest for a local etcd instance to "/etc/kubernetes/manifests/etcd.yaml"
[init] Waiting for the kubelet to boot up the control plane as Static Pods from directory "/etc/kubernetes/manifests".
[init] This might take a minute or longer if the control plane images have to be pulled.
[apiclient] All control plane components are healthy after 23.002530 seconds
[uploadconfig] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[markmaster] Will mark node iron40.bbn.com as master by adding a label and a taint
[markmaster] Master iron40.bbn.com tainted and labelled with key/value: node-role.kubernetes.io/master=""
[bootstraptoken] Using token: m5l3da.nbeu81vmqemxzhp5
[bootstraptoken] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstraptoken] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstraptoken] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstraptoken] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: kube-dns
[addons] Applied essential addon: kube-proxy
Your Kubernetes master has initialized successfully!
To start using your cluster, you need to run the following as a regular user:
mkdir -p $HOME/.kube

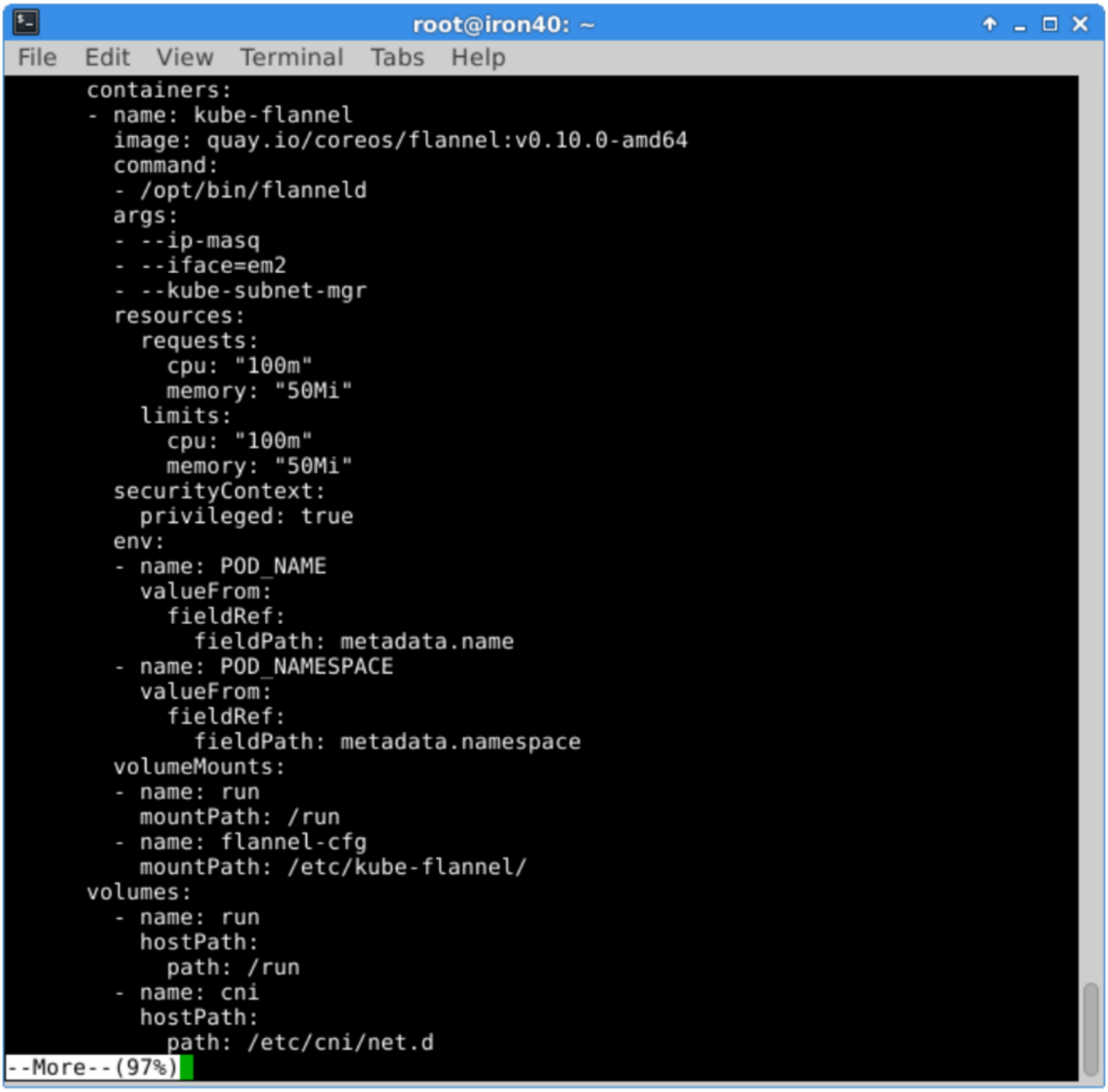
The last line of output from the *kubeadm init* command will be used when the worker nodes join the cluster.

4.1.2. Pod Network

As mentioned earlier, a pod network add-on must be installed so the pods can communicate. The default behavior for flannel is to use the default route interface for pod-to-pod communications. Since we want this to occur over the private network interface, we must modify the flannel configuration. First, get the flannel configuration file using the following command:

```
wget raw.githubusercontent.com/coreos/flannel/v0.10.0/Documentation/kube-flannel.yml
```

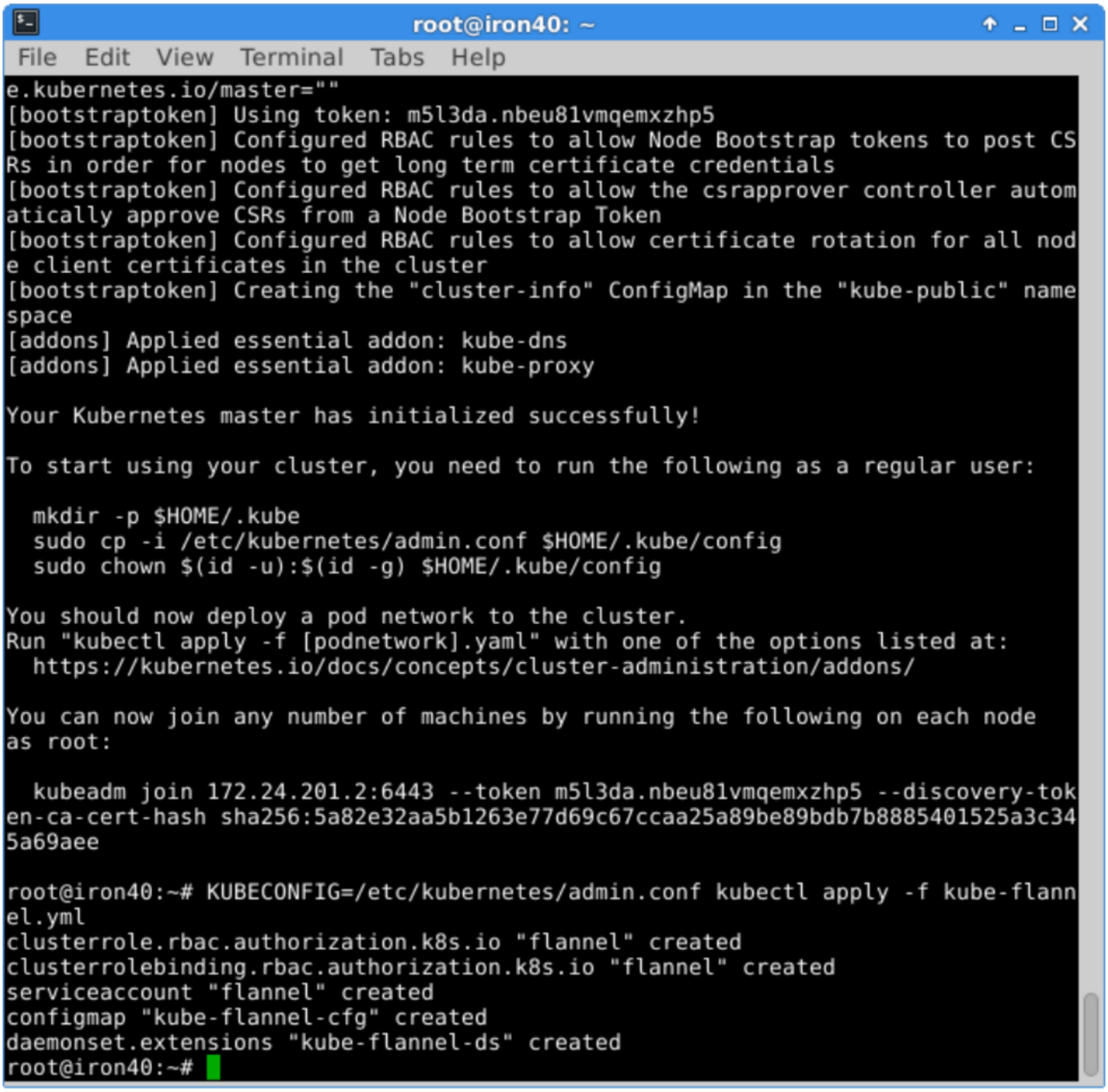
Next, add the required "−iface=em2" argument to the "args" section of the "kube-flannel" container of the kube-flannel.yml configuration file, as illustrated below:



We can now install the pod network with the following command:

```
KUBECONFIG=/etc/kubernetes/admin.conf kubectl apply -f kube-flannel.yml
```

The result from running the above command is shown below:



4.1.3. Controlling Kubernetes Cluster as Non-Root User

Once the master node has been initialized and the pod network has been loaded, set up the local user's environment so the cluster can be controlled without the *sudo* command. The following commands are used to establish this:

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
sudo cp -i /etc/kubernetes/admin.conf $HOME
sudo chown $(id -u):$(id -g) $HOME/admin.conf
```

Note: lines 3 and 4 are required for the Jupiter deployment scripts to execute properly.

4.1.4. Start Kubernetes Proxy

The Jupiter Orchestrator requires a working kubernetes cluster with proxy capability, so we need to start the proxy next. This is typically done in a separate terminal as the local user and is accomplished with the following **kubectl** command:

```
kubectl proxy -p 8080
```

4.1.5. Master Isolation

Be default, kubernetes will not schedule pods for the master node. The Jupiter Orchestrator, however, schedules pods on the master node. This is enabled by executing the following command on the master node:

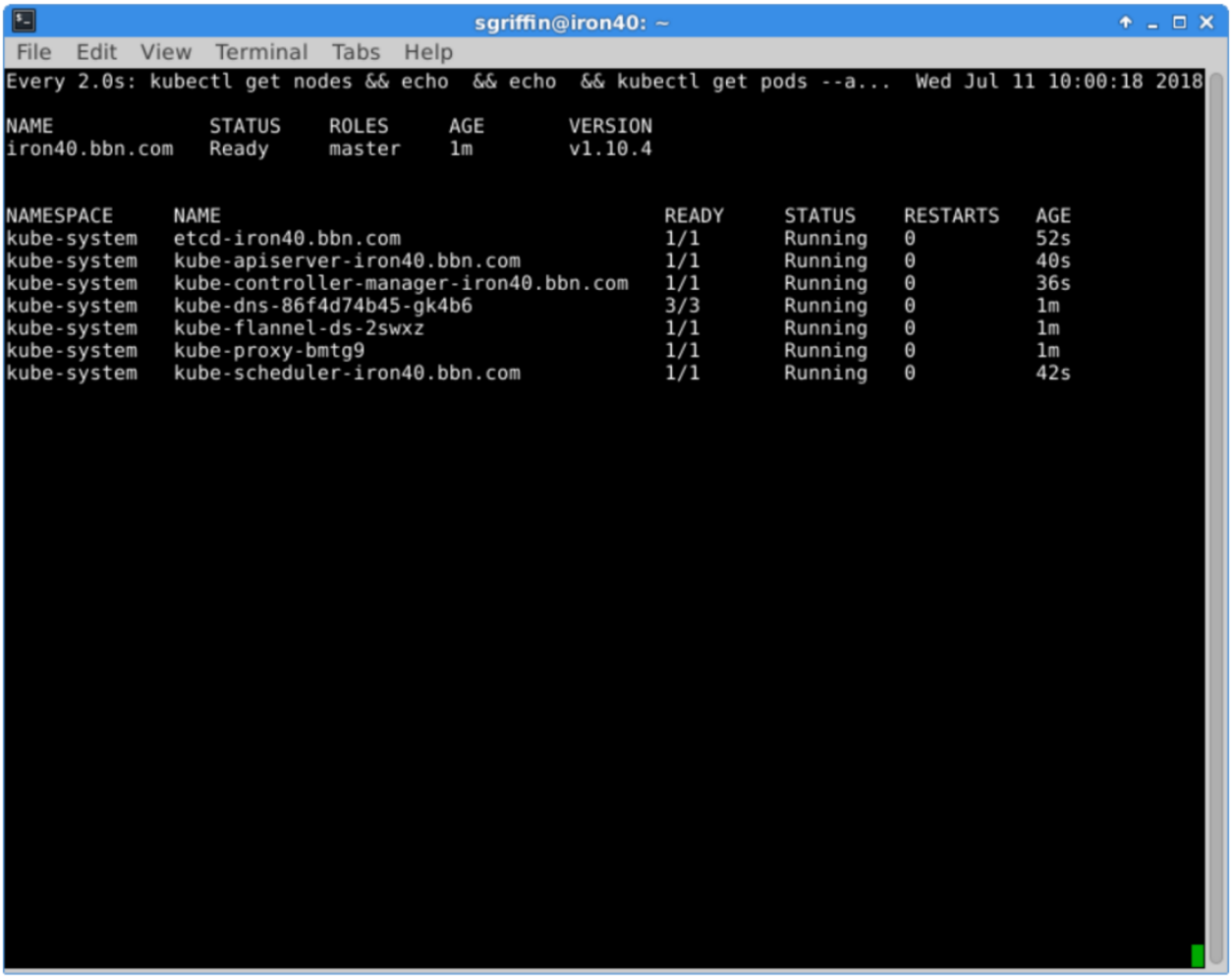
```
kubectl taint nodes --all node-role.kubernetes.io/master-
```

4.1.6. Verifying Master Initialization

Now that we have a master node initialized and a pod network installed, we can view the current state of our cluster using the **kubectl get** command. The following command can be used to continuously monitor, using the Linux **watch** command, the nodes and pods in our cluster:

```
watch "kubectl get nodes && echo "" && echo "" && kubectl get pods --all-namespaces"
```

The result of executing the above command is shown below:



We can see that we currently have 1 node in our cluster, the master node, iron40.bbn.com and that there are a number of pods running in the **kube-system** namespace. Now that we have a working master node, we can add the worker nodes to our cluster.

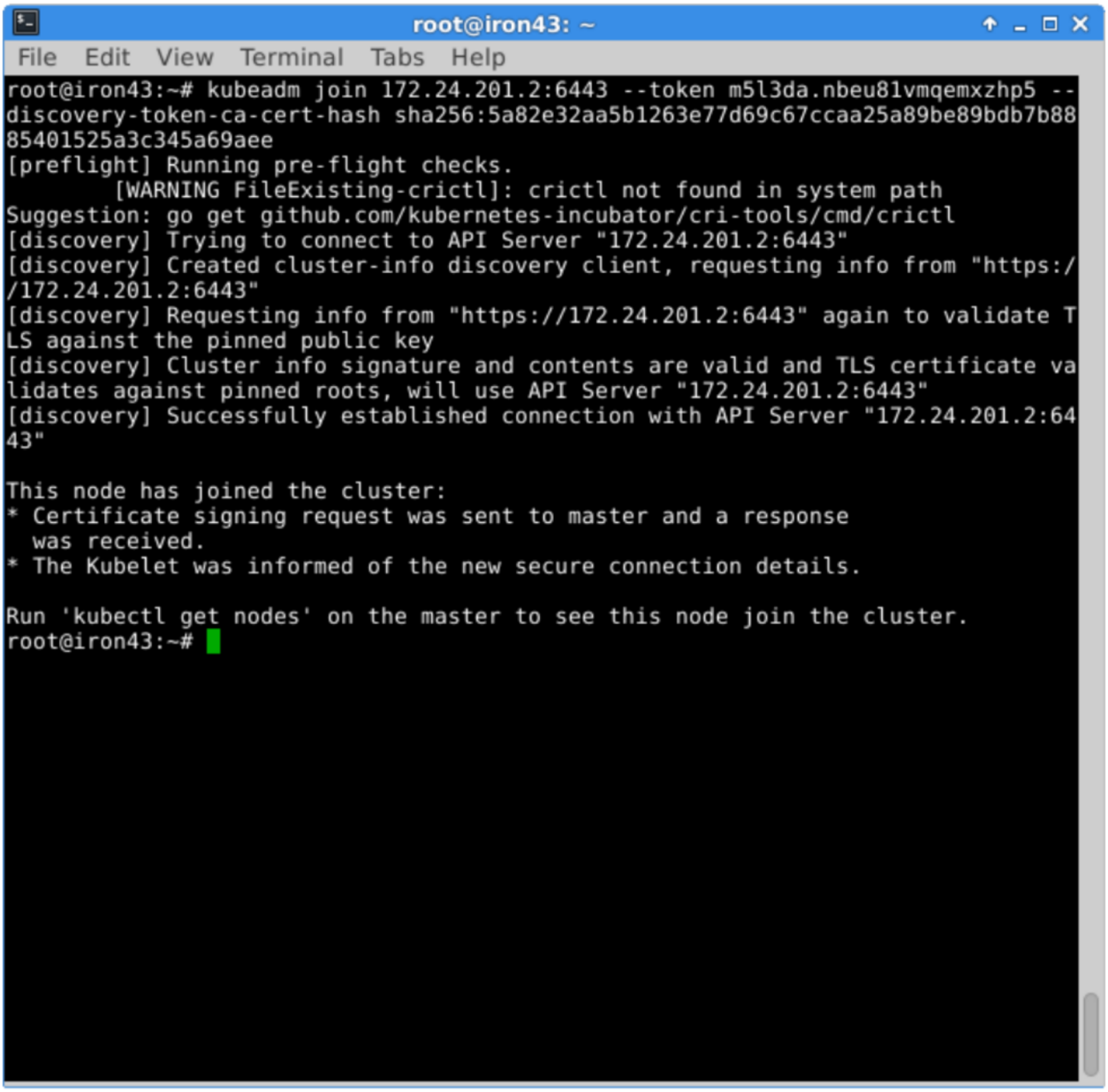
4.2. Kubernetes Worker Nodes

4.2.1. Join Kubernetes Worker Nodes

To join the worker nodes to the kubernetes cluster, execute the following command, as **root**, in a terminal on each of the worker nodes (namely, iron43, iron46, iron53, iron56, and iron59):

```
kubeadm join 172.24.201.2:6443 --token m5l3da.nbeu81vmqemxzhp5 --discovery-token-ca-
cert-hash sha256:5a82e32aa5b1263e77d69c67ccaa25a89be89bdb7b8885401525a3c345a69aee
```

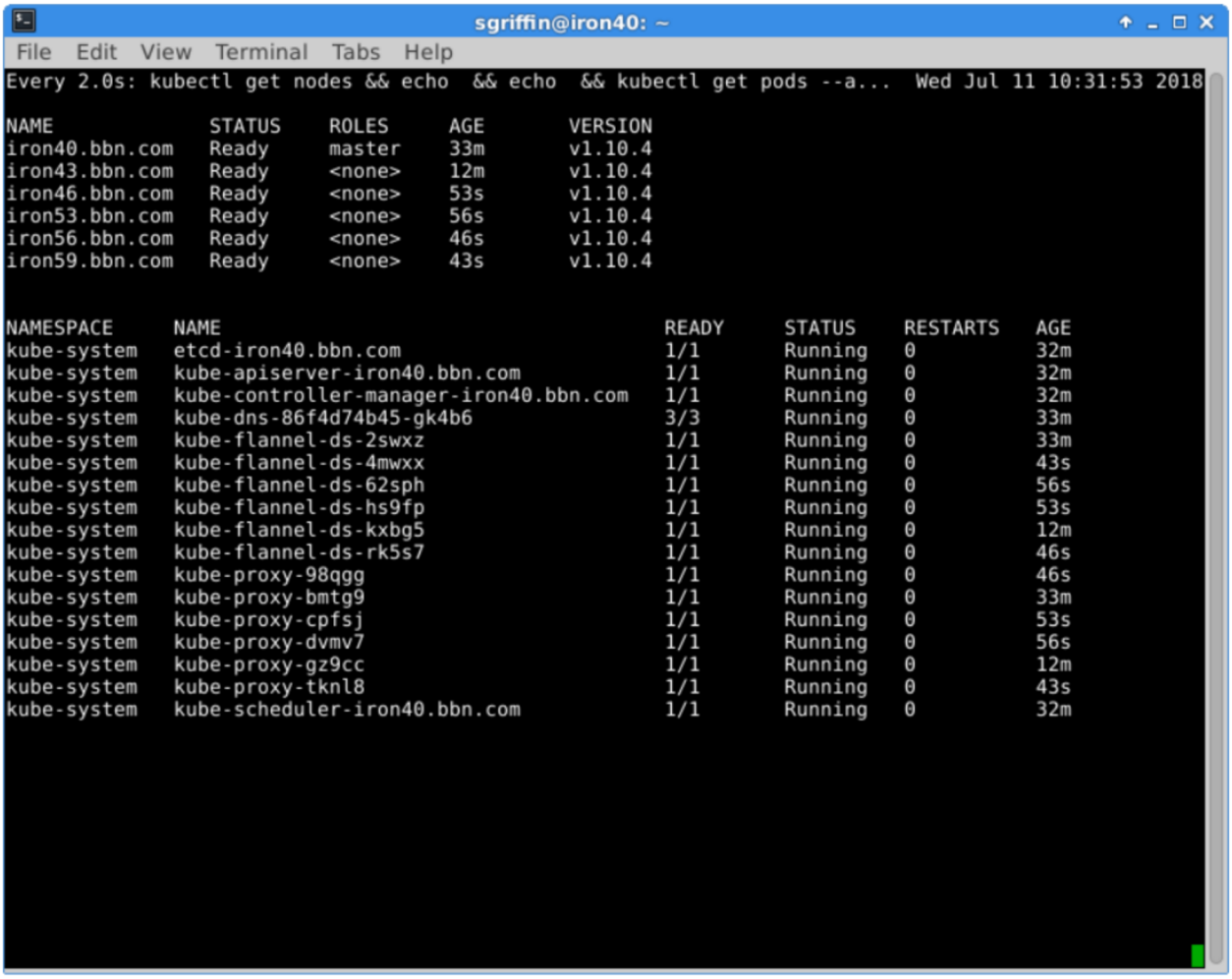
Note that the above command is provided at the end of the master node **kube init** command that was executed to initialize the master node. The result of executing the above command is shown below for node iron43:



Similar output can be observed on each of the other worker nodes.

4.2.2. Verifying Worker Nodes Cluster Joins

Now that we have joined all of the worker nodes, we can once again look at the output from the continuous monitoring of the cluster state, first described in verifying master initialization. The output from this monitoring is shown below:



We can now see that we have 1 master node, iron40.bbn.com, and 5 worker nodes, iron43.bbn.com, iron46.bbn.com, iron53.bbn.com, iron56.bbn.com, and iron59.bbn.com, as desired. We also have additional pods running, which are associated with the added worker nodes. Now that we have a working kubernetes cluster, we can configure and deploy the Jupiter Orchestrator.

5. Setup Local Private Docker Registry

5.1. Registry Configuration and Execution

We set up a local private Docker registry for the Jupiter pods so that we didn't have to push them to the publicly available Docker Hub. The instructions for deploying a Docker registry server can be found here. To run our experiments, we set up an insecure registry. Note that Docker must be installed on the machine that will host the registry. A summary of the steps taken are described below.

1. Configure the registry to be insecure, as described in detail here. Add the following information to the **/etc/docker/daemon.json** file on all nodes that will host the Jupiter pods:

```
{
  "insecure-registries" : ["iron40.bbn.com:5000"]
}
```

   If there is no **/etc/docker/daemon.json** file create it. If the file exists and has entries in it, terminate the last line of the existing configuration with a ',' before adding the "insecure-registries" line from above. Once the **daemon.json** file has been updated, restart docker, as **root**, with the following commands:

```
systemctl daemon-reload
systemctl restart docker
```

5. Start the Docker registry with the following command, as **root**, on the machine to host the registry:

```
docker run -d -p 5000:5000 --name registry registry:2
```

   We are now able to push and pull docker images from our insecure private docker registry.

5.2. Examining Registry Contents

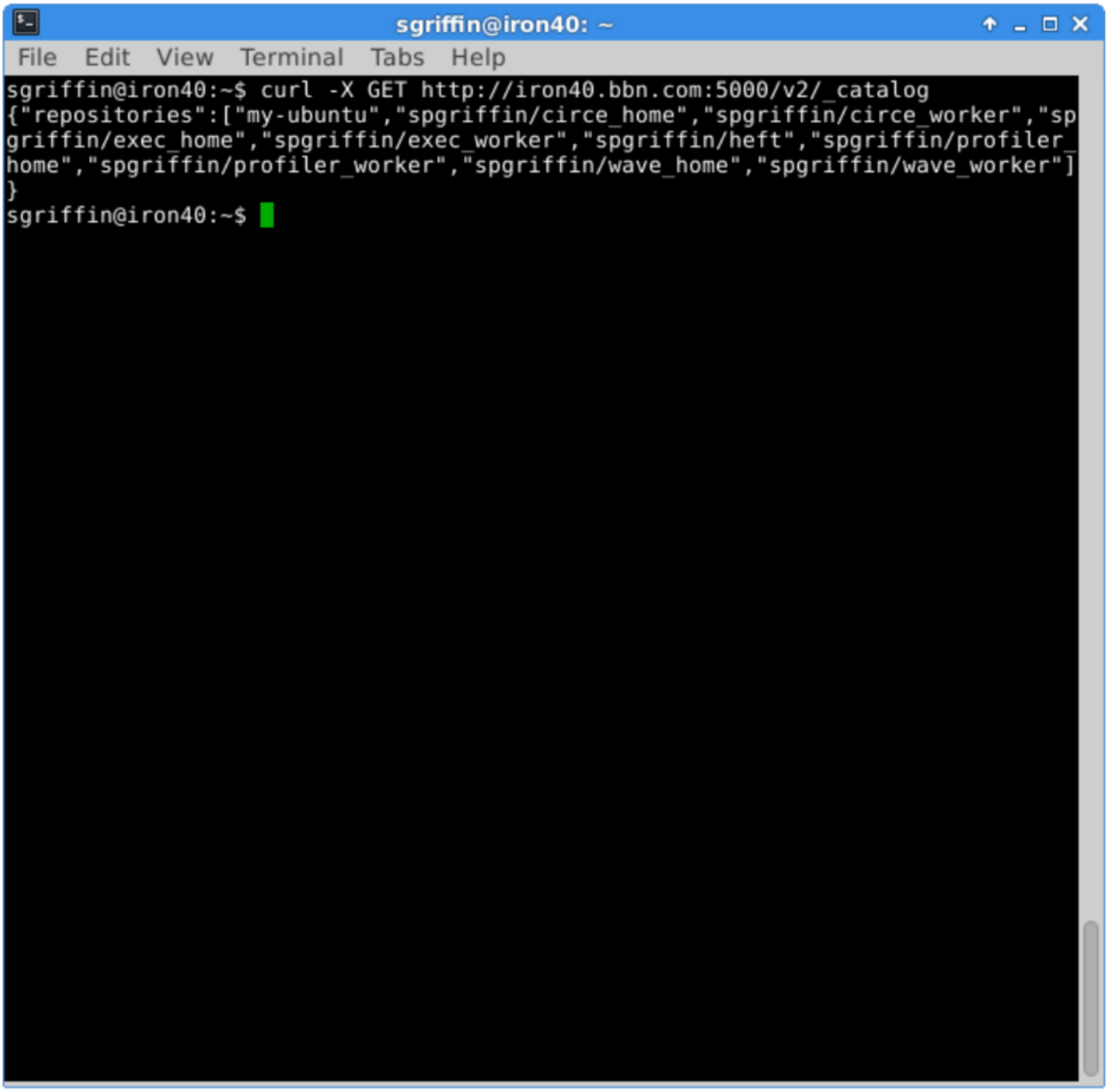
Once we have pushed some images to our newly created registry, we can examine the contents of it.

5.2.1. Listing Repositories

The following command is used to list all repositories in our Docker registry:

```
curl -X GET http://<registry_host>:<registry_port>/v2/_catalog
```

Example output from executing this command is shown below.

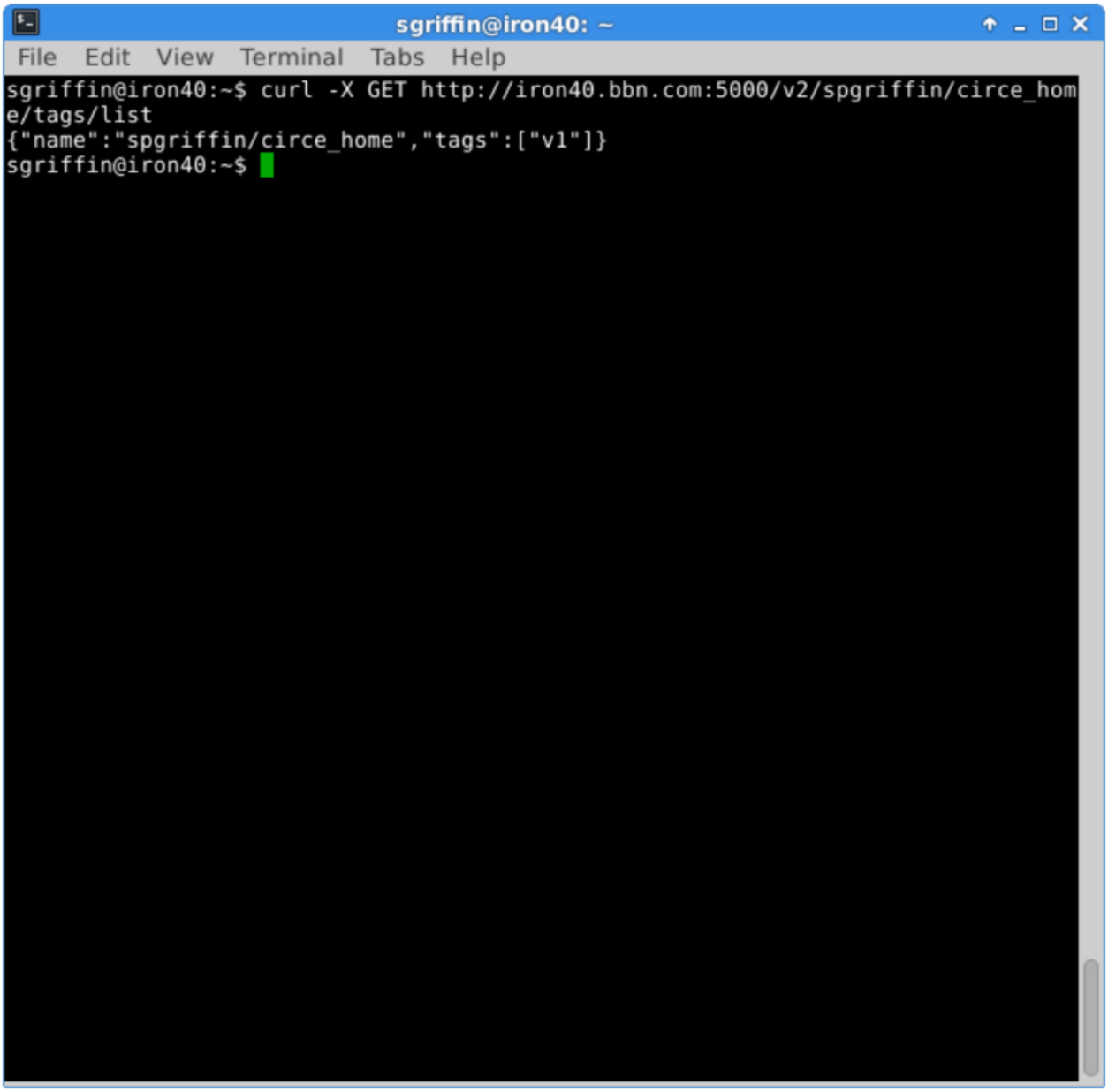


5.2.2. Listing Repository Tags

The following command, executed on the node hosting our docker registry, is used to list the tags associated with a docker repository:

```
curl -X GET http://<registry_host>:<registry_port>/v2/<repository_name>/tags/list
```

Example output from executing this command is shown below.



5.3. Registry Control

5.3.1. Stopping Registry

The registry can be stopped with the following command:

```
docker container stop registry
```

5.3.2. Starting Registry

The registry can be started with the following command:

```
docker container start registry
```

5.3.3. Removing Registry Data

Data can be removed from the registry (once stopped) with the following command:

```
docker container rm -v registry
```

6. Configuring and Deploying Jupiter

The Jupiter documentation provides a comprehensive set of steps to deploy Jupiter. The following set of changes were made to the Jupiter configuration files per the instructions:

- **Step 2: Update Node List** – The *nodes.txt* file was modified as instructed. The contents of the modified file are as follows:

```
home iron40.bbn.com root PASSWORD
node2 iron43.bbn.com root PASSWORD
node3 iron46.bbn.com root PASSWORD
node4 iron53.bbn.com root PASSWORD
node5 iron56.bbn.com root PASSWORD
node6 iron59.bbn.com root PASSWORD
```

- **Step 3: Setup Home Node** – The following line was changed in the *jupiter_config.py* input file:

```
HOME_CHILD = 'localpro'
```

- **Step 4: Setup APP Folder** – We did not do this step. Instead, we simply used the sample application that is distributed with the Jupiter Orchestrator.
- **Step 7: Push the Dockers** – Since we set up a local private Docker registry, modify the *jupiter_config.py* file to point to our registry. Change all instances of **docker.io** with **iron40.bbn.com:5000**.
- **Step 8: Setup the Proxy** – We did not do this here. This was done when we started a kubernetes proxy.
- **Step 9: Create the Namespaces** – The following script was used to create the namespaces:

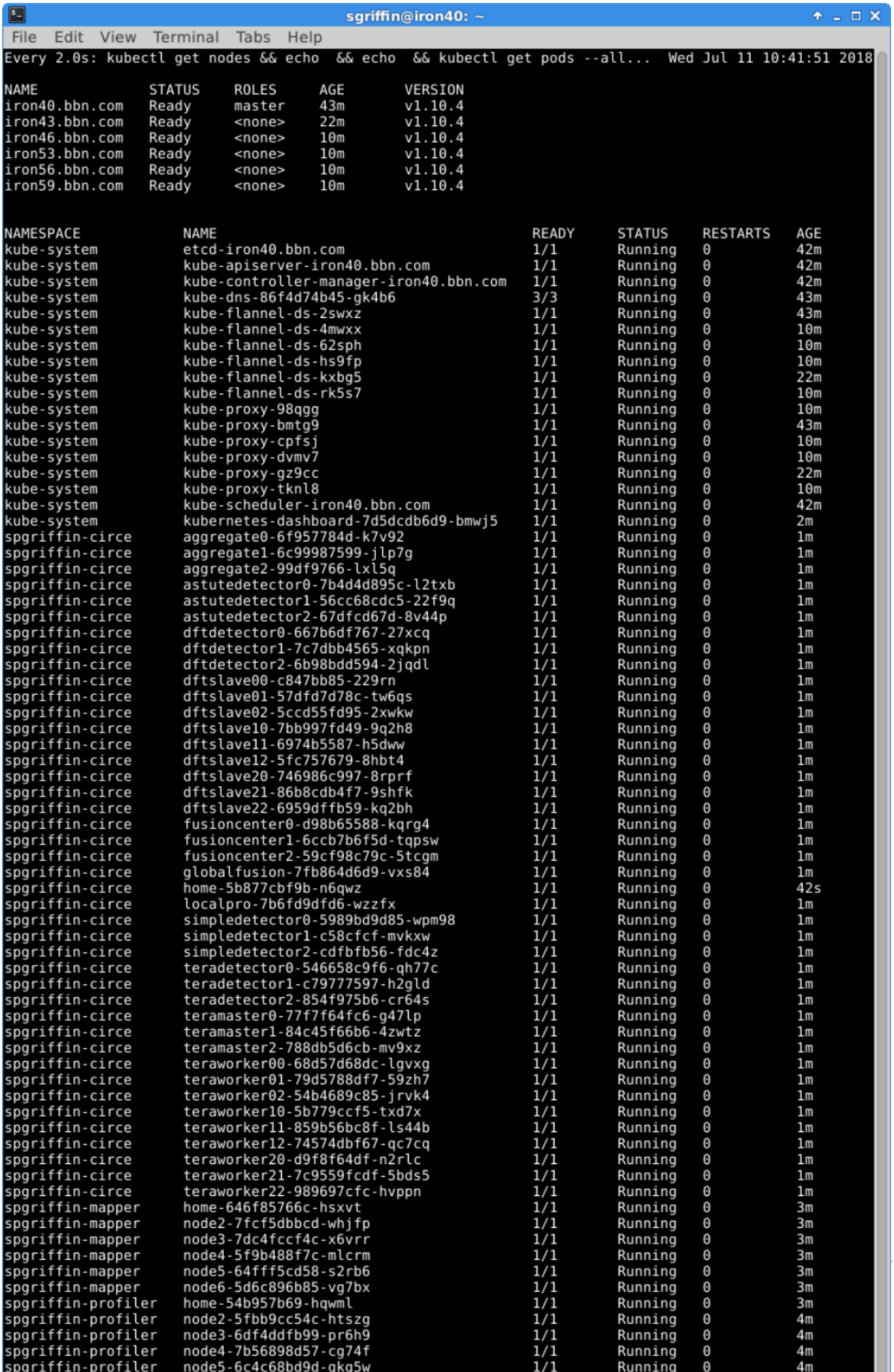
```
#!/bin/sh

kubectl create namespace spgriffin-profiler
kubectl create namespace spgriffin-exec
kubectl create namespace spgriffin-mapper
kubectl create namespace spgriffin-circe
```

Also, replace all instances of 'johndoe' with 'spgriffin' in *jupiter_config.py*. These names must match the names of the namespaces created by the above script. Here one would substitute their login username in place of 'spgriffin'.

- **Step 10: Run the Jupiter Orchestrator** – Before running the documented commands, we had to set the following environment variables in the terminal:
- `export KUBECONFIG=$HOME/admin.conf`
- `export PATH=$PATH:/usr/local/go/bin`
  `export PYTHONPATH=/home/sgriffin/Jupiter/Jupiter`

6.1. Verifying Jupiter Deployment

Following the execution of the command to execute the Jupiter Orchestrator, we can one more time take a look at the continuous cluster monitoring that we started in verifying master initialization.


sgriffin@iron40: ~
File Edit View Terminal Tabs Help
Every 2.0s: kubectl get nodes && echo && echo && kubectl get pods --all... Wed Jul 11 10:41:51 2018
NAME STATUS ROLES AGE VERSION
iron40.bbn.com Ready master 43m v1.10.4
iron43.bbn.com Ready <none> 22m v1.10.4
iron46.bbn.com Ready <none> 10m v1.10.4
iron53.bbn.com Ready <none> 10m v1.10.4
iron56.bbn.com Ready <none> 10m v1.10.4
iron59.bbn.com Ready <none> 10m v1.10.4
NAMESPACE NAME READY STATUS RESTARTS AGE
kube-system etcd-iron40.bbn.com 1/1 Running 0 42m
kube-system kube-apiserver-iron40.bbn.com 1/1 Running 0 42m
kube-system kube-controller-manager-iron40.bbn.com 1/1 Running 0 42m
kube-system kube-dns-86f4d74b45-gk4b6 3/3 Running 0 43m
kube-system kube-flannel-ds-2swxz 1/1 Running 0 43m
kube-system kube-flannel-ds-4mwxx 1/1 Running 0 10m
kube-system kube-flannel-ds-62sph 1/1 Running 0 10m
kube-system kube-flannel-ds-hs9fp 1/1 Running 0 10m
kube-system kube-flannel-ds-kxbg5 1/1 Running 0 22m
kube-system kube-flannel-ds-rk5s7 1/1 Running 0 10m
kube-system kube-proxy-98qgg 1/1 Running 0 10m
kube-system kube-proxy-bmtg9 1/1 Running 0 43m
kube-system kube-proxy-cpfsj 1/1 Running 0 10m
kube-system kube-proxy-dvmv7 1/1 Running 0 10m
kube-system kube-proxy-gz9cc 1/1 Running 0 22m
kube-system kube-proxy-tknl8 1/1 Running 0 10m
kube-system kube-scheduler-iron40.bbn.com 1/1 Running 0 42m
kube-system kubernetes-dashboard-7d5dcdb6d9-bmwj5 1/1 Running 0 2m
spgriffin-circe aggregate0-6f957784d-k7v92 1/1 Running 0 1m
spgriffin-circe aggregate1-6c99987599-jlp7g 1/1 Running 0 1m
spgriffin-circe aggregate2-99df9766-lxl5q 1/1 Running 0 1m
spgriffin-circe astutedetector0-7b4d4d895c-l2txb 1/1 Running 0 1m
spgriffin-circe astutedetector1-56cc68cdc5-22f9q 1/1 Running 0 1m
spgriffin-circe astutedetector2-67dfcd67d-8v44p 1/1 Running 0 1m
spgriffin-circe dftdetector0-667b6df767-27xcq 1/1 Running 0 1m
spgriffin-circe dftdetector1-7c7dbb4565-xqkpn 1/1 Running 0 1m
spgriffin-circe dftdetector2-6b98bdd594-2jqdl 1/1 Running 0 1m
spgriffin-circe dftslave00-c847bb85-229rn 1/1 Running 0 1m
spgriffin-circe dftslave01-57dfd7d78c-tw6qs 1/1 Running 0 1m
spgriffin-circe dftslave02-5ccd55fd95-2xwkw 1/1 Running 0 1m
spgriffin-circe dftslave10-7bb997fd49-9q2h8 1/1 Running 0 1m
spgriffin-circe dftslave11-6974b5587-h5dww 1/1 Running 0 1m
spgriffin-circe dftslave12-5fc757679-8hbt4 1/1 Running 0 1m
spgriffin-circe dftslave20-746986c997-8rprf 1/1 Running 0 1m
spgriffin-circe dftslave21-86b8cdb4f7-9shfk 1/1 Running 0 1m
spgriffin-circe dftslave22-6959dffb59-kq2bh 1/1 Running 0 1m
spgriffin-circe fusioncenter0-d98b65588-kqrg4 1/1 Running 0 1m
spgriffin-circe fusioncenter1-6ccb7b6f5d-tqpsw 1/1 Running 0 1m
spgriffin-circe fusioncenter2-59cf98c79c-5tcgm 1/1 Running 0 1m
spgriffin-circe globalfusion-7fb864d6d9-vxs84 1/1 Running 0 1m
spgriffin-circe home-5b877cbf9b-n6qwz 1/1 Running 0 42s
spgriffin-circe localpro-7b6fd9dfd6-wzzfx 1/1 Running 0 1m
spgriffin-circe simpledetector0-5989bd9d85-wpm98 1/1 Running 0 1m
spgriffin-circe simpledetector1-c58cfcf-mvkxw 1/1 Running 0 1m
spgriffin-circe simpledetector2-cdfbfb56-fdc4z 1/1 Running 0 1m
spgriffin-circe teradetector0-546658c9f6-qh77c 1/1 Running 0 1m
spgriffin-circe teradetector1-c79777597-h2gld 1/1 Running 0 1m
spgriffin-circe teradetector2-854f975b6-cr64s 1/1 Running 0 1m
spgriffin-circe teramaster0-77f7f64fc6-g47lp 1/1 Running 0 1m
spgriffin-circe teramaster1-84c45f66b6-4zwtz 1/1 Running 0 1m
spgriffin-circe teramaster2-788db5d6cb-mv9xz 1/1 Running 0 1m
spgriffin-circe teraworker00-68d57d68dc-lgvxg 1/1 Running 0 1m
spgriffin-circe teraworker01-79d5788df7-59zh7 1/1 Running 0 1m
spgriffin-circe teraworker02-54b4689c85-jrvk4 1/1 Running 0 1m
spgriffin-circe teraworker10-5b779ccf5-txd7x 1/1 Running 0 1m
spgriffin-circe teraworker11-859b56bc8f-ls44b 1/1 Running 0 1m
spgriffin-circe teraworker12-74574dbf67-qc7cq 1/1 Running 0 1m
spgriffin-circe teraworker20-d9f8f64df-n2rlc 1/1 Running 0 1m
spgriffin-circe teraworker21-7c9559fcdf-5bds5 1/1 Running 0 1m
spgriffin-circe teraworker22-989697cfc-hvppn 1/1 Running 0 1m
spgriffin-mapper home-646f85766c-hsxvt 1/1 Running 0 3m
spgriffin-mapper node2-7fcf5dbbcd-whjfp 1/1 Running 0 3m
spgriffin-mapper node3-7dc4fccf4c-x6vrr 1/1 Running 0 3m
spgriffin-mapper node4-5f9b488f7c-mlcrm 1/1 Running 0 3m
spgriffin-mapper node5-64fff5cd58-s2rb6 1/1 Running 0 3m
spgriffin-mapper node6-5d6c896b85-vg7bx 1/1 Running 0 3m
spgriffin-profiler home-54b957b69-hqwml 1/1 Running 0 3m
spgriffin-profiler node2-5fbb9cc54c-htszg 1/1 Running 0 4m
spgriffin-profiler node3-6df4ddfb99-pr6h9 1/1 Running 0 4m
spgriffin-profiler node4-7b56898d57-cg74f 1/1 Running 0 4m
spgriffin-profiler node5-6c4c68bd9d-gkg5w 1/1 Running 0 4m

We see from the above output that the Jupiter pods, in the spgriffin-circe, spgriffin-mapper, and spgriffin-profiler namespaces, are in the Running state.This means that the Jupiter Orchestrator has been successfully deployed on our new kubernetes cluster.

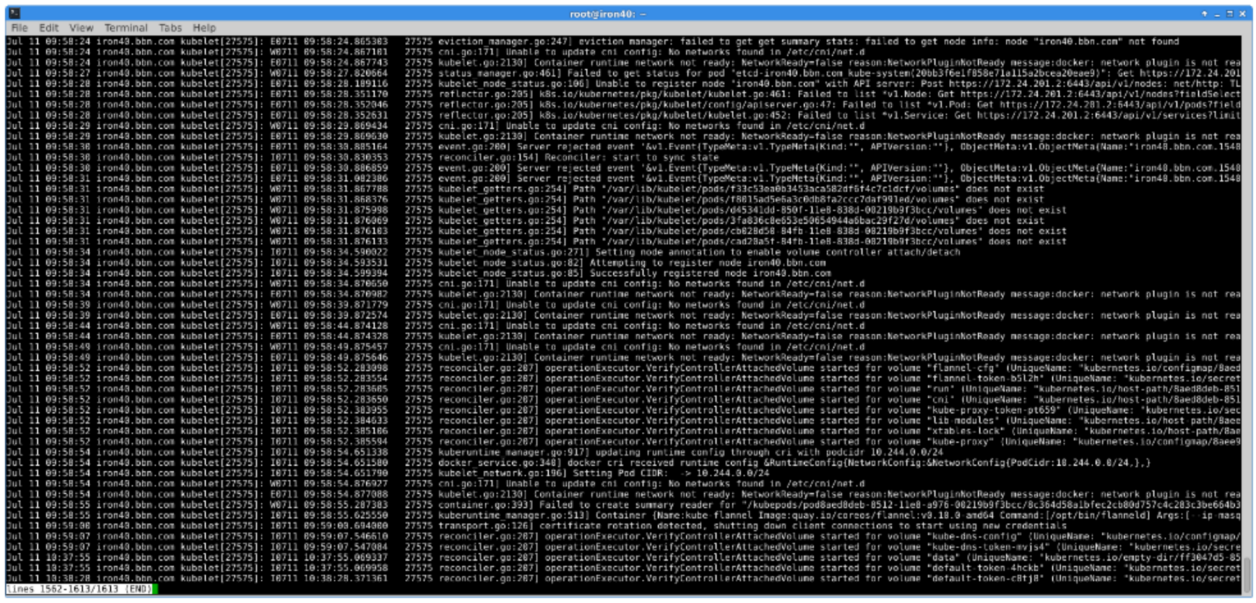
7. Some Useful Debugging Techniques

7.1. kubelet logs

The state of the kubernetes kubelets can be viewed by executing the following command:

```
journalctl -xeu kubelet
```

During the creation of our cluster, this command identified some configuration errors that we were able to address. The result of executing the command is shown below:



7.2. Kubernetes Dashboard

Dashboard is a web-based kubernetes user interface.

7.2.1. Dashboard Deployment

The following command was used to deploy the Dashboard:

```
kubectl create -f
https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/k
ubernetes-dashboard.yaml
```

After deploying the Dashboard, configure it so the login step can be skipped (as described in full detail here). We found this to be necessary due to some recent security additions to kubernetes. In summary, this is accomplished by creating a file of the name, **dashboard-admin.yaml**, with the following contents:

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: kubernetes-dashboard
  labels:
    k8s-app: kubernetes-dashboard
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kubernetes-dashboard
  namespace: kube-system
```

Then, this was loaded with the following command:

```
kubectl create -f ~/dashboard-admin.yaml
```

The commands and their resulting outputs are shown below:

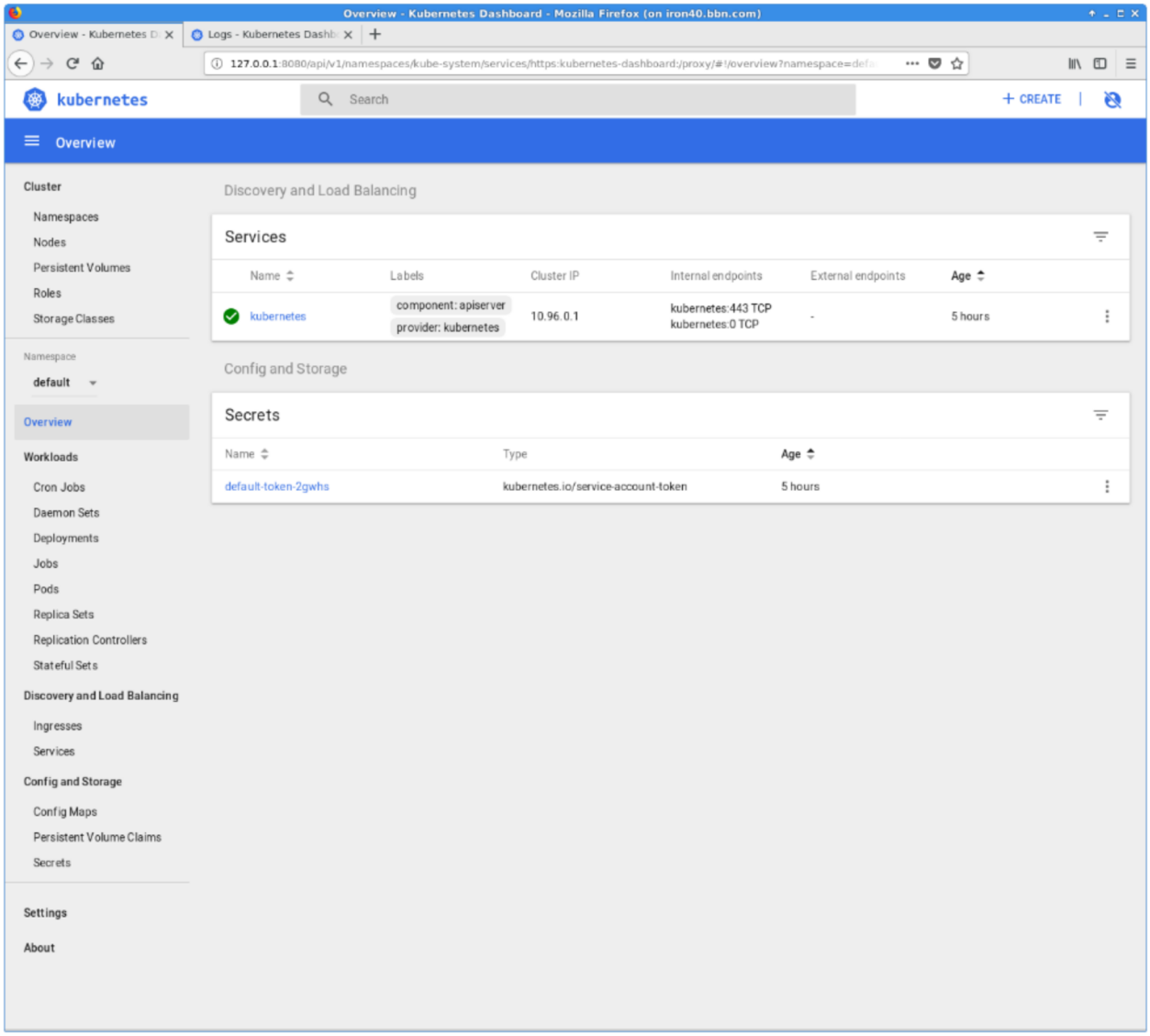
```
sgriffin@iron40:~$
sgriffin@iron40:~$
sgriffin@iron40:~$ kubectl create -f https://raw.githubusercontent.com/kubernete
s/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml
secret "kubernetes-dashboard-certs" created
serviceaccount "kubernetes-dashboard" created
role.rbac.authorization.k8s.io "kubernetes-dashboard-minimal" created
rolebinding.rbac.authorization.k8s.io "kubernetes-dashboard-minimal" created
deployment.apps "kubernetes-dashboard" created
service "kubernetes-dashboard" created
sgriffin@iron40:~$ kubectl create -f ~/dashboard-admin.yaml
clusterrolebinding.rbac.authorization.k8s.io "kubernetes-dashboard" created
```

7.2.2. Interacting with the Cluster via the Dashboard

Now that the Dashboard has been deployed, we can interact with our cluster. Launch a browser and visit the following URL:

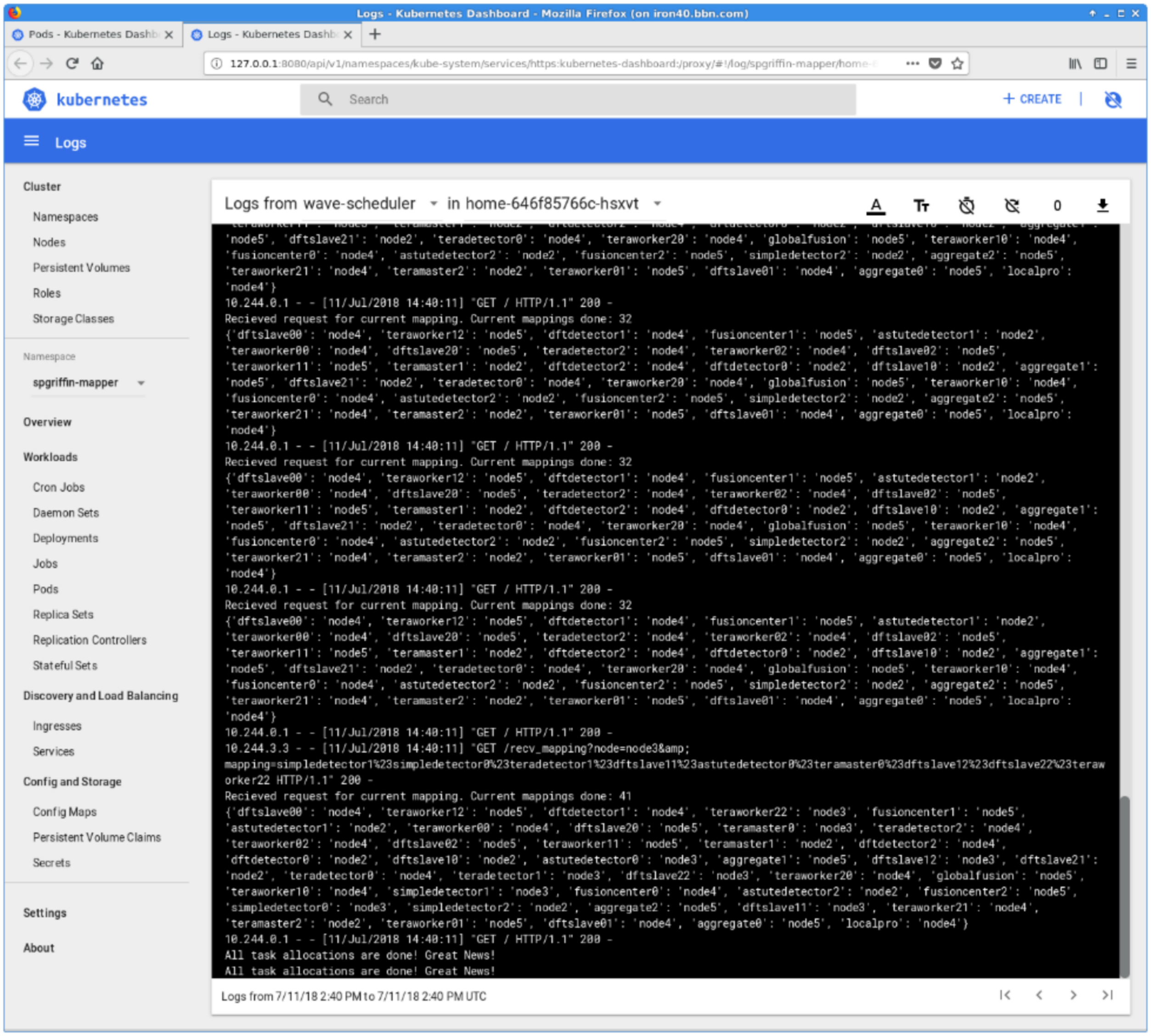
```
http://127.0.0.1:8080/api/v1/namespaces/kube-system/services/https:kubernetes-
dashboard:/proxy/
```

In the pop-up dialog, select 'Skip' to get to the main Dashboard page, illustrated below:



We can now view the Pods logs on the home node by performing the following:

1. Choose the desired Namespace in the Namespace drop down box on the left hand side of the Dashboard. To look at the mapper logs, shown below, the sgriffin-mapper item was chosen.
2. Choose the Pods button in the Workloads section
3. Choose the logs icon (denoted by 3 1/2 vertical lines) on the right hand side of the Pod whose name starts with home-...
4. View the contents of the home mapper Pod log file, shown below.

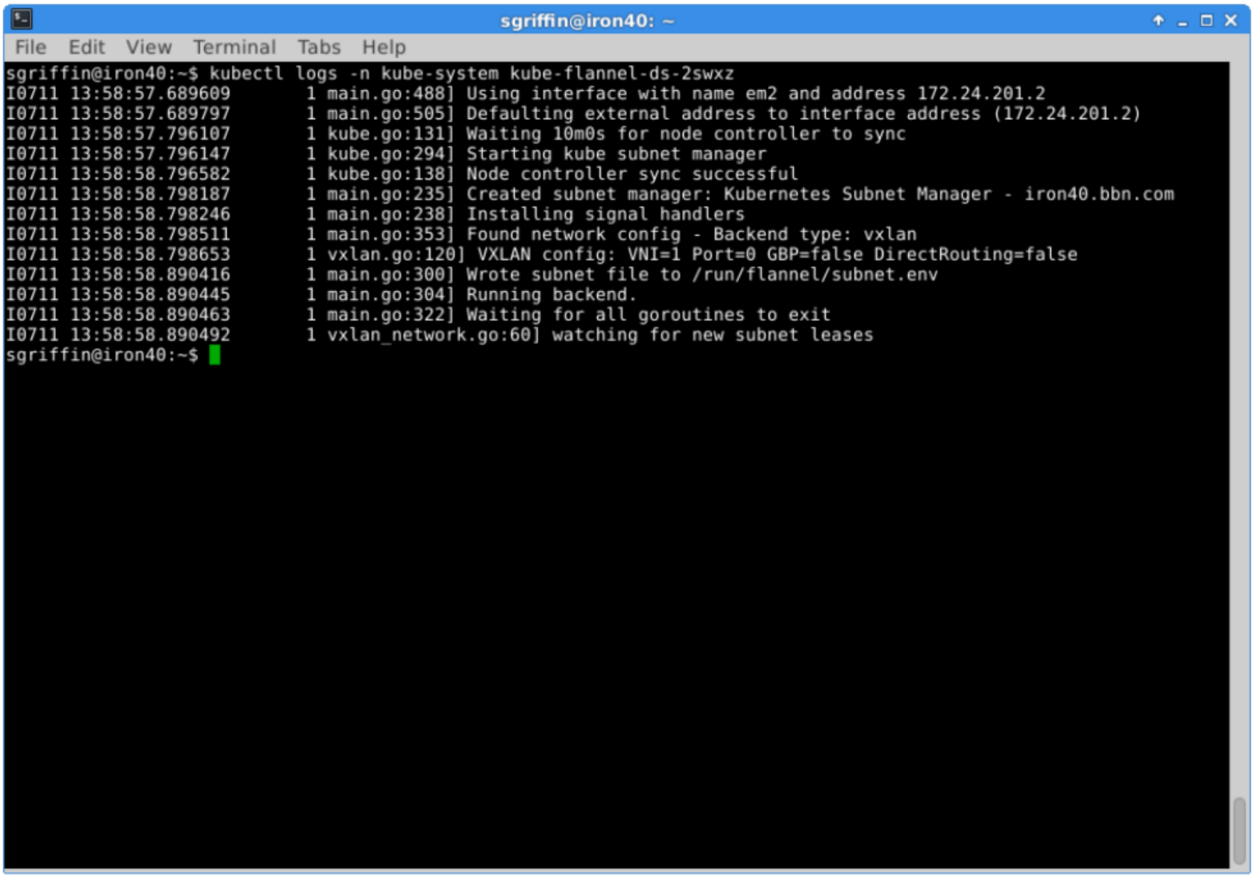


7.3. Viewing Pod Logs

In addition to the Dashboard, described above, the kubernetes **kubectl** command can be used to view the Pod log files. To do this, view all pods as described in verifying cluster operation and find the name of the Pod of interest and its corresponding Namespace. The following command was used to view the contents of one of the flannel Pods in our cluster:

```
kubectl logs -n kube-system kube-flannel-ds-2swxz
```

The result of executing this command is shown below:



8. Teardown

8.1. Jupiter Teardown

Follow the Jupiter instructions for stopping the Jupiter Orchestrator.

8.2. Kubernetes Cluster Teardown

8.2.1. Worker Nodes Cleanup

The first step in tearing down the cluster is to drain and delete the worker nodes. This is accomplished by executing the following commands on the master node:

```
#!/bin/sh

kubectl drain iron43.bbn.com --delete-local-data --force --ignore-daemonsets
kubectl delete node iron43.bbn.com
kubectl drain iron46.bbn.com --delete-local-data --force --ignore-daemonsets
kubectl delete node iron46.bbn.com
kubectl drain iron53.bbn.com --delete-local-data --force --ignore-daemonsets
kubectl delete node iron53.bbn.com
kubectl drain iron56.bbn.com --delete-local-data --force --ignore-daemonsets
```

```
kubectl delete node iron56.bbn.com
kubectl drain iron59.bbn.com --delete-local-data --force --ignore-daemonsets
kubectl delete node iron59.bbn.com
```

Once the above commands have completed, run the following commands on each of the worker nodes, as **root**, to terminate any kubernetes processes:

```
#!/bin/sh

kubeadm reset
systemctl stop kubelet
systemctl stop docker
rm -rf /var/lib/cni/
rm -rf /var/lib/kubelet/*
rm -rf /etc/cni/
ifconfig docker0 down
ifconfig flannel.1 down
ip link del flannel.1
ifconfig cni0 down
ip link del cni0
systemctl start kubelet
systemctl start docker
```

8.2.2. Master Node Cleanup

Once the kubernetes components have been stopped on the worker nodes, run the following commands, as **root**, on the master node to terminate any kubernetes processes:

```
#!/bin/sh

kubeadm reset
systemctl stop kubelet
systemctl stop docker
rm -rf /var/lib/cni/
rm -rf /var/lib/kubelet/*
rm -rf /etc/cni/
ifconfig docker0 down
ifconfig flannel.1 down
ip link del flannel.1
ifconfig cni0 down
ip link del cni0
systemctl start kubelet
systemctl start docker
```

Once this step is complete, our continuous cluster monitoring terminal displays errors connecting to the cluster, as expected. We have now successfully destroyed our kubernetes cluster.