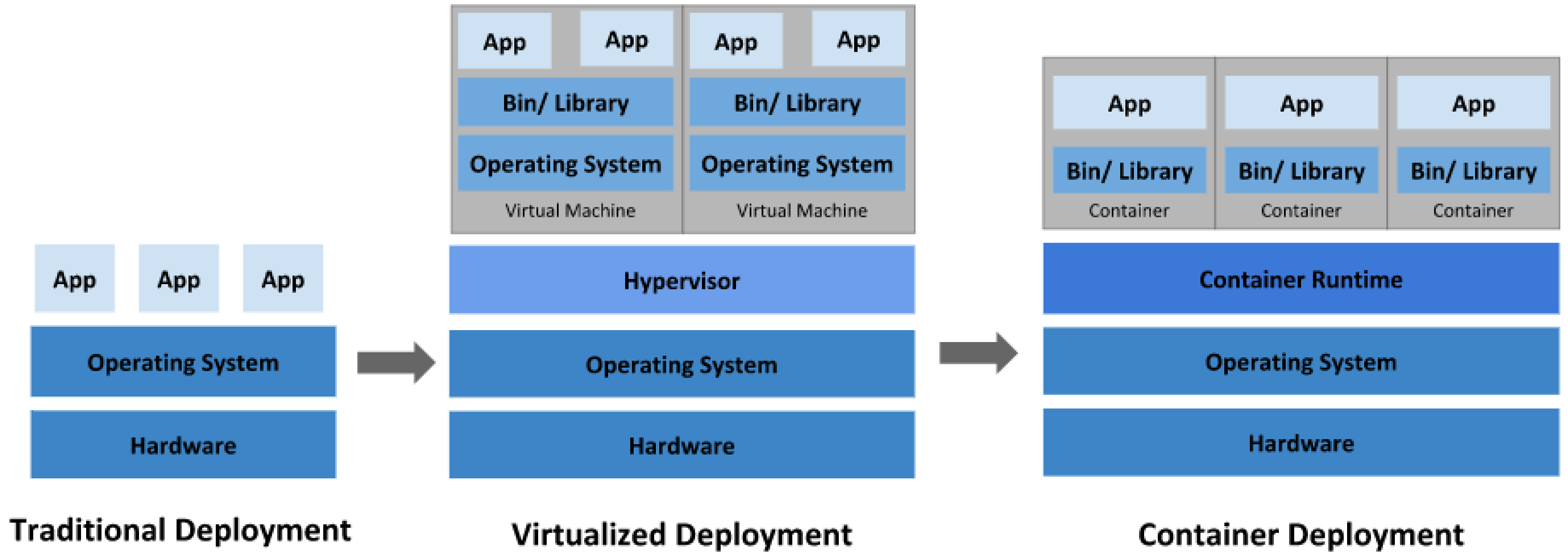
IERG 4330

Tutorial 3

What is Kubernetes?

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.



Why you need Kubernetes

In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behavior was handled by a system?

Before you begin

You need:

- One or more machines running a deb/rpm-compatible Linux OS; for example: Ubuntu or CentOS.
- 2 GiB or more of RAM per machine--any less leaves little room for your apps.
- At least 2 CPUs on the machine that you use as a control-plane node.
- Full network connectivity among all machines in the cluster. You can use either a public or a private network.

Kubernetes Cluster Installation

Install docker

- Follow installation guide in the first tutorial

Installing kubeadm, kubelet and kubectl

- kubeadm: the command to bootstrap the cluster.
- kubelet: the component that runs on all of the machines in your cluster and does things like starting pods and containers.
- kubectl: the command line tool to talk to your cluster.

Kubernetes Cluster Installation

Installing kubeadm, kubelet and kubectl

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

Creating a cluster with kubeadm

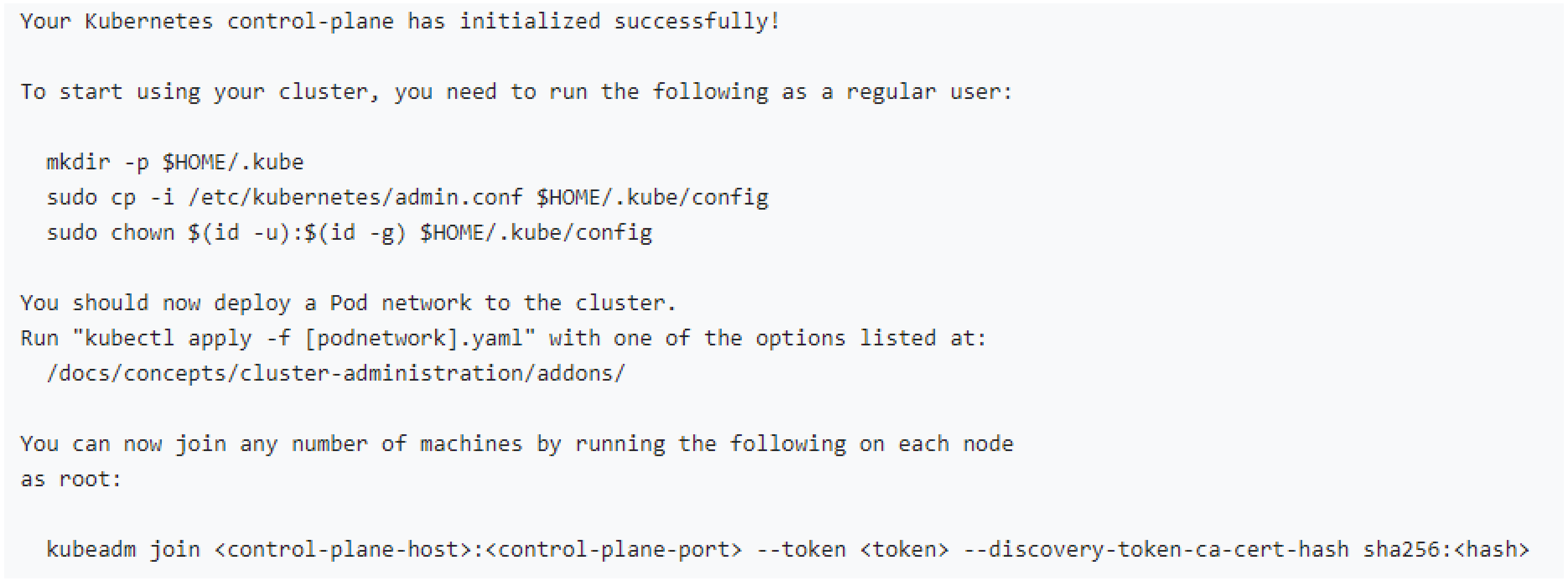
```
kubeadm init <args>
```

kubeadm init first runs a series of prechecks to ensure that the machine is ready to run Kubernetes. These prechecks expose warnings and exit on errors.

Then it will download and install the cluster control plane components. This may take several minutes.

Creating a cluster with kubeadm

After it finishes you should see:



```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

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

You should now deploy a Pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  /docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each node
as root:

  kubeadm join <control-plane-host>:<control-plane-port> --token <token> --discovery-token-ca-cert-hash sha256:<hash>
```

Installing a Pod network add-on

- See the list of add-ons that implement the Kubernetes networking model.
- You can install a Pod network add-on with the following command on the control-plane node

```
kubectl apply -f <add-on.yaml>
```

Joining your nodes

The nodes are where your workloads (containers and Pods, etc) run. To add new nodes to your cluster, do the following for each machine:

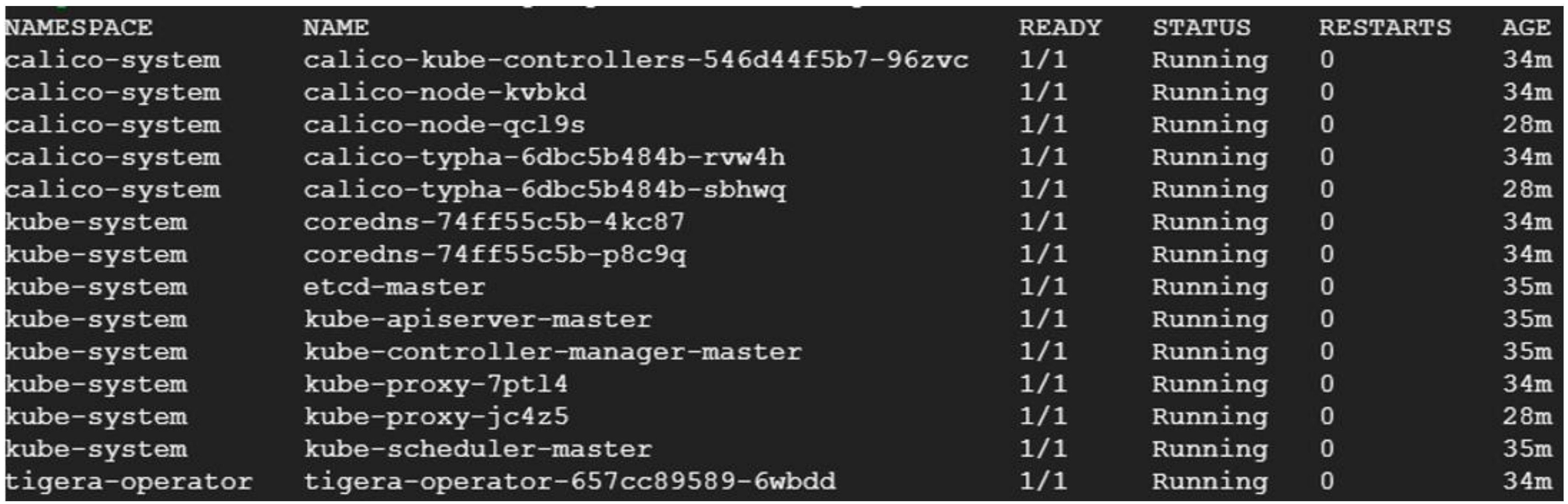
- SSH to the machine
- Become root (e.g. sudo su -)
- Run the command the following commands:

```
kubeadm join --token <token> <control-plane-host>:<control-plane-port> --discovery-token-ca-cert-hash sha256:<hash>
```

Check the status of your cluster

```
kubectl get nodes
kubectl get pods --all-namespaces
```

An example output(calico is one of the network pod):



```
NAMESPACE         NAME                                       READY   STATUS    RESTARTS   AGE
calico-system     calico-kube-controllers-546d44f5b7-96zvc   1/1     Running   0          34m
calico-system     calico-node-kvbkd                          1/1     Running   0          34m
calico-system     calico-node-qcl9s                          1/1     Running   0          28m
calico-system     calico-typha-6dbc5b484b-rvw4h              1/1     Running   0          34m
calico-system     calico-typha-6dbc5b484b-sbhwq              1/1     Running   0          28m
kube-system       coredns-74ff55c5b-4kc87                    1/1     Running   0          34m
kube-system       coredns-74ff55c5b-p8c9q                    1/1     Running   0          34m
kube-system       etcd-master                                1/1     Running   0          35m
kube-system       kube-apiserver-master                      1/1     Running   0          35m
kube-system       kube-controller-manager-master             1/1     Running   0          35m
kube-system       kube-proxy-7ptl4                           1/1     Running   0          34m
kube-system       kube-proxy-jc4z5                           1/1     Running   0          28m
kube-system       kube-scheduler-master                      1/1     Running   0          35m
tigera-operator   tigera-operator-657cc89589-6wbdd           1/1     Running   0          34m
```

Run an example on your k8s cluster

Check the API version in your cluster:

```
kubectl api-resources
```

Create hello.yaml with the following code:

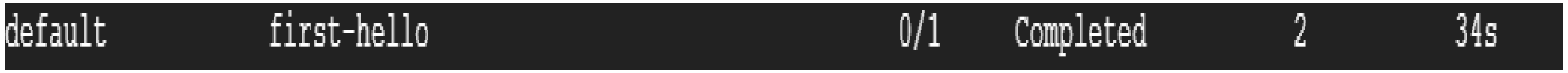
```
apiVersion: v1
King: Pod
Metadata:
  name: first-hello
Spec:
    containers:
        - name: hello
           image: hello-world
```

Run an example on your k8s cluster

Then run the following command to start:

```
kubectl create –f  Tesing_for_Image_pull.yaml
```

Use kubectl get pods --all-namespaces to check the status

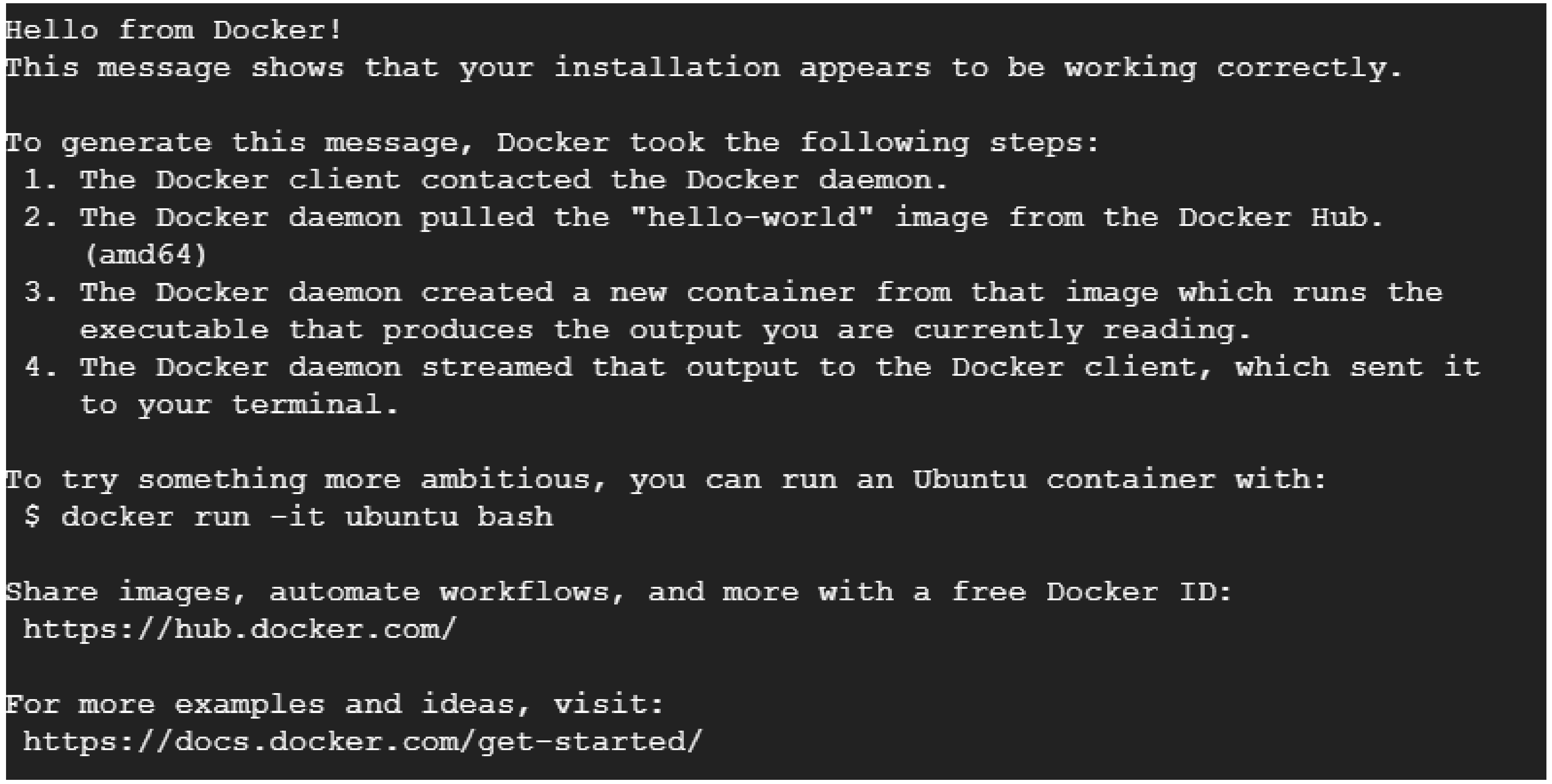


```
default          first-hello                          0/1    Completed       2         34s
```

Run an example on your k8s cluster

Get output by the following command:

```
kubectl logs first-hello
```



```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Q&A