1. How would I go about it?

I would use Google Cloud Platform to create clusters, join the two master nodes using a virtual IP address to allow for load balancing and also failsafe, join worker nodes to the master nodes and configure centralized logging from docker.

Implementation of the Approach

Kubeadm is a cloud provider agnostic tool that automates many of the tasks required to get a cluster up and running. Users of kubeadm can run a few simple commands on individual servers to turn them into a Kubernetes cluster consisting of a master node and worker nodes. This guide will walk you through installing kubeadm and using it to deploy a Kubernetes cluster.This solution will be covered as way to dive deeper into the various components that make up a Kubernetes cluster and the ways in which they interact with each other to provide a scalable and reliable container orchestration mechanism.

Install the Container Runtime: Docker

Docker is the software responsible for running the pod containers on each node.

1. Make sure you have the necessary packages to allow the use of Docker's repository:

sudo apt install apt-transport-https ca-certificates curl software-properties-common

2. Add Docker's GPG key:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

3. Verify the fingerprint of the GPG key:

sudo apt-key fingerprint 0EBFCD88

Output:

```
etes_test$ sudo apt-key fingerprint 0EBFCD88
pub   rsa4096 2017-02-22 [SCEA]
      9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid           [ unknown] Docker Release (CE deb) <docker@docker.com>
sub   rsa4096 2017-02-22 [S]
```

4. Add the stable Docker repository:

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $ (lsb_release -cs) stable"

5. Update your package index and install Docker CE:

```
sudo apt update
sudo apt install docker-ce
```

6. Check that the installation was successful by running the built-in “Hello World” program:

```
sudo docker run hello-world
```

Output:

```
etes_test$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:9572f7cdcee8591948c2963463447a53466950b3fc15a247fcad1917ca215a2f
Status: Downloaded newer image for hello-world:latest

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

7. Setup the Docker daemon to use systemd as the cgroup driver. Systemd provides a logging daemon and other tools and utilities to help with common system administration tasks. This is a recommended step so that Kubelet and Docker are both using the same cgroup manager. This will make it easier for Kubernetes to know which resources are available on your cluster’s nodes.

```
sudo bash -c 'cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF'
```

8. Create a systemd directory for Docker:

```
sudo mkdir -p /etc/systemd/system/docker.service.d
```

9. Restart Docker:

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

Install kubeadm, kubelet, and kubectl

1. Update the system and install the required dependencies for installation:

```
sudo apt-get update && sudo apt-get install -y apt-transport-https curl
```

2. Add the required GPG key to your apt-sources keyring to authenticate the Kubernetes related packages you will install:

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

3. Add Kubernetes to the package manager's list of sources:

```
sudo bash -c "cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF"
```

4. Update apt, install Kubeadm, Kubelet, and Kubectl, and hold the installed packages at their installed versions:

```
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

5. Verify that kubeadm, kubelet, and kubectl have installed by retrieving their version information. Each command should return version information about each package.

```
kubeadm version
kubelet --version
kubectl version
```

Build a Kubernetes Cluster

A Kubernetes cluster consists of a master node and worker nodes. The master node hosts the *control plane*, which is the combination of all the components that provide it the ability to maintain the desired cluster state.

1. Login to your google cloud console:

```
etes_test$ gcloud auth login
Your browser has been opened to visit:

    https://accounts.google.com/o/oauth2/auth?code_challenge=iIvaDZLTfQPg5rM-SKE
JJO5XGCDmexgjh_bsvr-DhkQ&prompt=select_account&code_challenge_method=S256&access
_type=offline&redirect_uri=http%3A%2F%2Flocalhost%3A8085%2F&response_type=code&c
lient_id=32555940559.apps.googleusercontent.com&scope=https%3A%2F%2Fwww.googleap
is.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-p
latform+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fappengine.admin+https%3A%2F%2F
www.googleapis.com%2Fauth%2Fcompute+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fac
counts.reauth



You are now logged in as [ssouljabee@gmail.com].
```

2. To create a Kubernetes cluster using Kubernetes Engine, run the following:

gcloud container --project "savvy-cinema-259506" clusters create "networkloadb" --zone "us-central1-a" --num-nodes "2"

Output:

```
Creating cluster networkloadb in us-central1-a... Cluster is being configured..
.⁝.
Creating cluster networkloadb in us-central1-a... Cluster is being health-check
ed (master is healthy)...done.
Created [https://container.googleapis.com/v1/projects/savvy-cinema-259506/zones/
us-central1-a/clusters/networkloadb].
To inspect the contents of your cluster, go to: https://console.cloud.google.com
/kubernetes/workload_/gcloud/us-central1-a/networkloadb?project=savvy-cinema-259
506
kubeconfig entry generated for networkloadb.
NAME          LOCATION       MASTER_VERSION  MASTER_IP      MACHINE_TYPE   NODE_
VERSION     NUM_NODES  STATUS
networkloadb  us-central1-a  1.13.11-gke.14  35.188.30.186  n1-standard-1  1.13.
11-gke.14   2          RUNNING
```

3. Let us build another one so as to have two masters:

gcloud container --project "savvy-cinema-259506" clusters create "networkloadb2" --zone "us-central1-a" --num-nodes "2"

```
Creating cluster networkloadb2 in us-central1-a... Cluster is being health-checked (mas
ter i
s healthy)...done.

Created [https://container.googleapis.com/v1/projects/savvy-cinema-259506/zones/us-cent
ral1-a/clusters/networkloadb2].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubern
etes/workload_/gcloud/us-central1-a/networkloadb2?project=savvy-cinema-259506
kubeconfig entry generated for networkloadb2.
NAME           LOCATION       MASTER_VERSION  MASTER_IP      MACHINE_TYPE   NODE_VERSIO
N    NUM_NODES  STATUS
networkloadb2  us-central1-a  1.13.11-gke.14  34.66.254.215  n1-standard-1  1.13.11-gke
.14  2          RUNNING
```

Setup load balancer

A **virtual IP address** is an address the nodes used to communicate with each other. It is internally known only to the nodes themselves.

There are multiple cloud provider solutions for load balancing like AWS elastic load balancer, GCE load balancing etc. There might not be a physical load balancer available, we can setup a virtual IP load balancer to healthy node master. We are using keepalived for load balancing.

1. First, we install some requirements:

$ apt install libssl-dev build-essential

2. Then download and extract keepalived sources:

$ wget https://www.keepalived.org/software/keepalived-2.0.15.tar.gz
$ tar xvzf keepalived-2.0.15.tar.gz

3. Then enter to extracted folder and build keepalived:

$ cd keepalived-2.0.15
$ ./configure
$ make && make install

Configuring Keepalived

Now that keepalived is installed, we need to configure it. Of course the configuration is different (almost specular) between lnetworkloadb and networkloadb2.

On both networkloadb and networkloadb2 instances, the configuration is stored on /etc/keepalived/keepalived.conf.

networkloadb configuration is:

```
global_defs {
   enable_script_security
   script_user node
}vrrp_script chk_haproxy {
   script "/usr/bin/pkill -0 haproxy"
   interval 2
   weight 2
}vrrp_instance VI_1 {
   interface vlan.99
   state MASTER
   priority 101    virtual_router_id 42
   unicast_src_ip {{ 35.188.30.186 }}    unicast_peer {
      {{ lb2_ip_address }}
   }    authentication {
      auth_type PASS
      auth_pass AaP51Mdi
   }    track_script {
      chk_haproxy
```

networkloadb2 configuration is:

```
global_defs {
   enable_script_security
```

```
    script_user node
}vrrp_script chk_haproxy {
    script "/usr/bin/pkill -0 haproxy"
    interval 2
    weight 2
}vrrp_instance VI_1 {
    interface vlan.99
    state BACKUP
    priority 100    virtual_router_id 42
    unicast_src_ip {{ 34.66.254.215 }}    unicast_peer {
        {{ lb1_ip_address }}
    }    authentication {
        auth_type PASS
        auth_pass AaP51Mdi
    }    track_script {
        chk_haproxy
    }    notify_master /etc/keepalived/master.sh
}
```

On global_defs, enable_script_security is active and the user which will run the script is node. Remember to avoid running scripts as root.

The track_script chk_haproxy executes /usr/bin/pkill -0 haproxy to check if HAProxy instance is still alive on server. There are a lot of different ways to check it, e.g. killall -0 haproxy and so on. Behaviour might be slightly different between systems, I found the check with pkill to be reliable enough.

After configuration, on both instances you need to restart keepalived issuing:

```
service keepalived restart
```