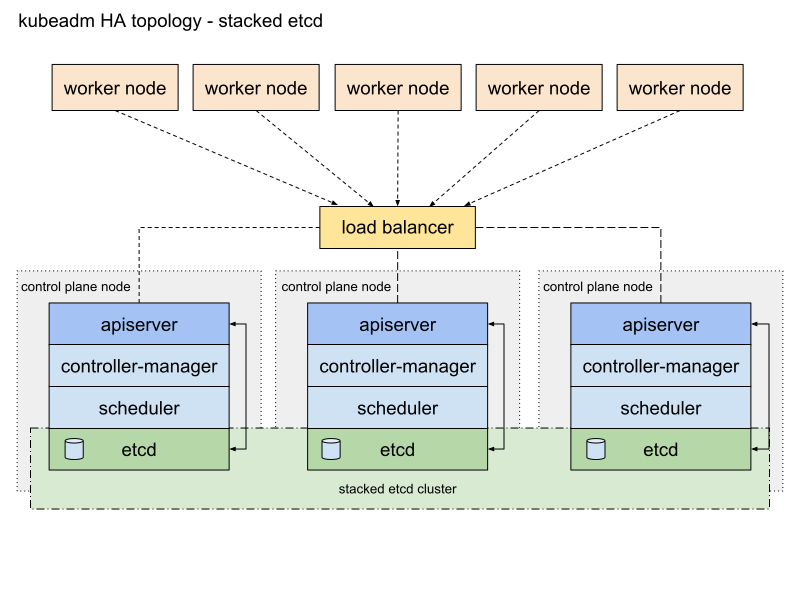
 A high availability cluster consists of multiple control plane nodes (master nodes), multiple worker nodes and a load balancer. This setup makes the system more robust since any node can fail without the application going offline or data being lost. It also makes it easy to add more compute power or replace old nodes without any downtime. An illustration of this setup is shown below. The ETCD cluster makes sure that all data is synced across the master nodes and the load balancer regulates the traffic distribution. The cluster can therefore be accessed through one single entry point (the load balancer and the request will be passed to an arbitrary node.



Reference: <https://kubernetes.io/docs/setup/independent/ha-topology/#stacked-etcd-topology>

**Requirements**

* At least 2 master nodes (min 2GB ram, 2CPUs, see: [Kubeadm, before you begin](https://kubernetes.io/docs/setup/independent/install-kubeadm/" \l "before-you-begin" \t "_blank))
* At least 1 worker node (2 recommended, )
* One load balancer machine
* **All** nodes must have swap disabled

**ALL NODES: DEPENDENCIES**

* Docker:

sudo apt-get update*# Install packages to allow apt to use a repository over HTTPS:*  
sudo apt-get install -y \  
 apt-transport-https \  
 ca-certificates \  
 curl \  
 gnupg-agent \  
 software-properties-common*# Add docker key*   
curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo apt-key add -*# Add docker repo*  
sudo add-apt-repository \  
 "deb [arch=amd64] <https://download.docker.com/linux/ubuntu> \  
 **$(**lsb\_release -cs**)** \  
 stable"sudo apt-get update*# Install DockerCE*  
sudo apt-get install -y docker-ce

* Kubernetes (kubelet, kubeadm & 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> | apt-key add -  
cat <<EOF >/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

In order for the nodes to register again after failure or reboot turn on the systemctl services on boot:

systemctl enable kubelet  
systemctl enable docker

**LOAD BALANCER**

SSH to the node which will function as the load balancer and execute the following commands to install HAProxy:

sudo apt-get update  
sudo apt-get upgrade  
sudo apt-get install haproxy

Edit haproxy.cfg to connect it to the master nodes, set the correct values for <kube-balancer-ip> and <kube-masterx-ip> and add an extra entry for each additional master:

$ sudo nano /etc/haproxy/haproxy.cfgglobal  
 ....  
defaults  
 ....  
frontend kubernetes  
 bind <kube-balancer-ip>:6443  
 option tcplog  
 mode tcp  
 default\_backend kubernetes-master-nodesfrontend http\_front  
 mode http  
 bind <kube-balancer-ip>:80  
 default\_backend http\_backfrontend https\_front  
 mode http  
 bind <kube-balancer-ip>:443  
 default\_backend https\_backbackend kubernetes-master-nodes  
 mode tcp  
 balance roundrobin  
 option tcp-check  
 server k8s-master-0 <kube-master0-ip>:6443 check fall 3 rise 2  
 server k8s-master-1 <kube-master1-ip>:6443 check fall 3 rise 2backend http\_back  
 mode http  
 server k8s-master-0 <kube-master0-ip>:32059 check fall 3 rise 2  
 server k8s-master-0 <kube-master0-ip>:32059 check fall 3 rise 2backend https\_back  
 mode http  
 server k8s-master-0 <kube-master0-ip>:32423 check fall 3 rise 2  
 server k8s-master-0 <kube-master0-ip>:32423 check fall 3 rise 2

Restart HAproxy:

sudo systemctl restart haproxy

**FIRST MASTER NODE**

Create a config file called kubeadmcf.yaml and add the following in this file:

apiVersion: kubeadm.k8s.io/v1beta1  
kind: ClusterConfiguration  
kubernetesVersion: stable  
apiServer:  
 certSANs:  
 - "LOAD\_BALANCER\_DNS"  
controlPlaneEndpoint: "LOAD\_BALANCER\_DNS:LOAD\_BALANCER\_PORT"

Initialize the node:

sudo kubeadm init --config**=**kubeadmcf.yaml

**Save the join command for later usage**

In order to make kubectl work for non-root users execute the following:

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

Activate the Weave CNI plugin:

kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=**$(**kubectl version | base64 | tr -d '\n'**)**"

Use the following command to check the new pods:

kubectl get pod -n kube-system -w

**ADDITIONAL MASTER NODES**

Copy all config files to the other master nodes:

USER**=**kg *# customizable*  
*# Set the control\_plane\_ips to all other master node ips or hostnames*  
CONTROL\_PLANE\_IPS**=**"10.0.0.7 10.0.0.8"  
**for** host in ${CONTROL\_PLANE\_IPS}; **do**  
 scp /etc/kubernetes/pki/ca.crt "${USER}"@$host:  
 scp /etc/kubernetes/pki/ca.key "${USER}"@$host:  
 scp /etc/kubernetes/pki/sa.key "${USER}"@$host:  
 scp /etc/kubernetes/pki/sa.pub "${USER}"@$host:  
 scp /etc/kubernetes/pki/front-proxy-ca.crt "${USER}"@$host:  
 scp /etc/kubernetes/pki/front-proxy-ca.key "${USER}"@$host:  
 scp /etc/kubernetes/pki/etcd/ca.crt "${USER}"@$host:etcd-ca.crt  
 scp /etc/kubernetes/pki/etcd/ca.key "${USER}"@$host:etcd-ca.key  
 scp /etc/kubernetes/admin.conf "${USER}"@$host:  
**done**

Log into each master node and move the copied files to the correct location:

USER**=**kg *# customizable*  
mkdir -p /etc/kubernetes/pki/etcd  
mv /home/${USER}/ca.crt /etc/kubernetes/pki/  
mv /home/${USER}/ca.key /etc/kubernetes/pki/  
mv /home/${USER}/sa.pub /etc/kubernetes/pki/  
mv /home/${USER}/sa.key /etc/kubernetes/pki/  
mv /home/${USER}/front-proxy-ca.crt /etc/kubernetes/pki/  
mv /home/${USER}/front-proxy-ca.key /etc/kubernetes/pki/  
mv /home/${USER}/etcd-ca.crt /etc/kubernetes/pki/etcd/ca.crt  
mv /home/${USER}/etcd-ca.key /etc/kubernetes/pki/etcd/ca.key  
mv /home/${USER}/admin.conf /etc/kubernetes/admin.conf

After that execute the join command you obtained in the previous step prefixed with--experimental-control-plane, so it looks like this:

kubeadm join <kube-balancer-ip>:6443 --token <your\_token> --discovery-token-ca-cert-hash <your\_discovery\_token> --experimental-control-plane

**WORKER NODES**

Join each worker to the cluster with the join command, like:

kubeadm join <kube-balancer-ip>:6443 --token <your\_token> --discovery-token-ca-cert-hash <your\_discovery\_token>

**Connecting to the cluster**

In order to be able to connect to the cluster you need to download the kubeconfig file to your local machine (located in /etc/kubernetes/ or $HOME/.kube, the default file is called admin.conf).

It is recommended to put this file in $HOME/.kube/ and name it config, this will let Kubernetes use this file by default. Otherwise it needs to be specified with each command (with --kubeconfig).

Open a terminal on your local machine and now you can execute every kubectl command like this, which will run the command on the remote cluster:

kubectl get nodes

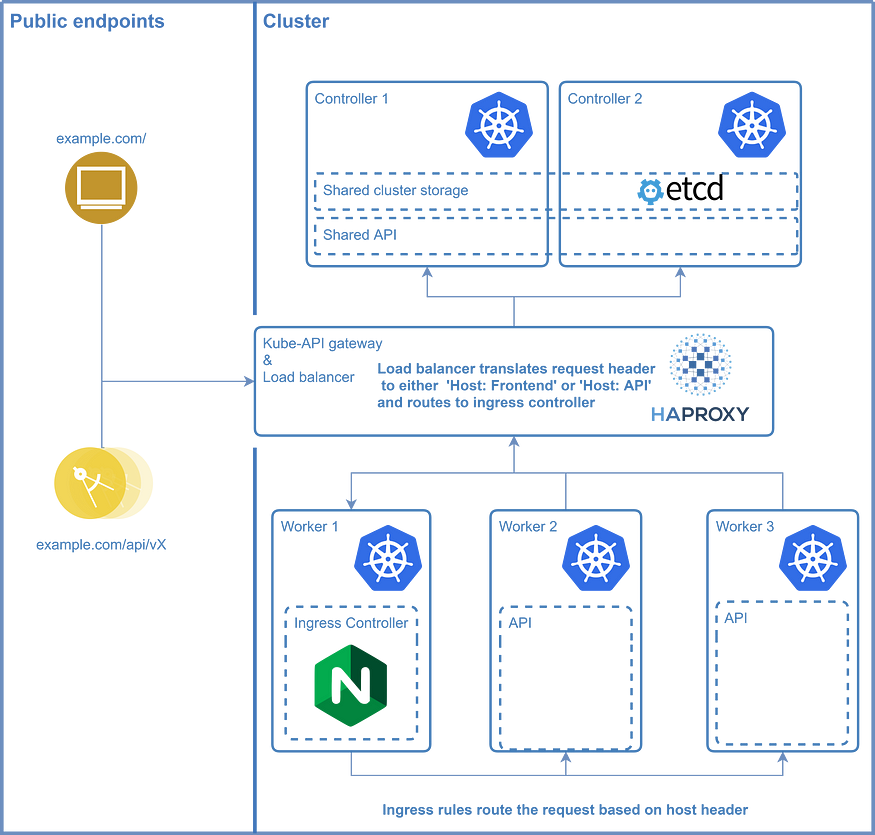
To test the connection, run:

kubectl cluster-info

This command will show if the connection can be made to the cluster.

**Ingress Controller**

An ingress controller provides a few different options for serving your services on regular ports without conflicts. The ingress controller makes sure that you can connect to the right pod on your worker nodes while letting HAProxy route the request to one of your controllers on a regular port. The nginx-ingress uses a specific port on the cluster which allows access to all other services which have an ingress rule. The reason why an ingress controller is recommended is because Kubernetes services use ports in a very high port range which have to be bound in HAProxy. By using an ingress on a predefined port only this port has to be defined in HAProxy, allowing easy routing and maintainability of the services.



An overview of the cluster topology

**DEPLOYING**NGINX-INGRESS

The first step in this process is to download the deployment file from [github](https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/mandatory.yaml" \t "_blank) to deploy nginx-ingress to the cluster, or at once with the following command:

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.2.0/deploy/static/provider/baremetal/deploy.yaml*# If you downloaded the file use:*  
kubectl apply -f mandatory.yaml

The next step is to expose nginx-ingress with a service, defined in a file called nginx-service.yaml:

apiVersion: v1  
kind: Service  
metadata:  
 name: ingress-nginx  
 namespace: ingress-nginx  
 labels:  
 app.kubernetes.io/name: ingress-nginx  
 app.kubernetes.io/part-of: ingress-nginx  
spec:  
 type: NodePort  
 ports:  
 - name: http  
 port: 80  
 targetPort: 80  
 protocol: TCP  
 nodePort: 32059  
 - name: https  
 port: 443  
 targetPort: 443  
 protocol: TCP  
 nodePort: 32423  
 selector:  
 app.kubernetes.io/name: ingress-nginx  
 app.kubernetes.io/part-of: ingress-nginx

Then run kubectl apply -f nginx-service.yaml to deploy this custom service.

Please note that you can change the nodePort to any value you like, however make sure to also change these values in the HAProxy since it should route to this port on the cluster.

**NOTE:** it is recommended to upgrade the deployment to run more than 1 replica of the nginx-ingress pod, otherwise a failure of the node it is one could break the access to your application

**SETTING UP AN INGRESS RULE**

Ingress rules define a set of rules on how NGINX should route the traffic based on the Host header in the request. An ingress rule looks like this:

apiVersion: extensions/v1beta1  
kind: Ingress  
metadata:  
 name: <your-service-name>  
spec:  
 rules:  
 - host: <hostname-which-triggers-rule>  
 http:  
 paths:  
 - path:  
 backend:  
 serviceName: <your-service-name>  
 servicePort: 80

The name defines how your rule can be found, serviceName specifies the service to which the requests that match this should be routed and servicePort specifies the port of that service which is accessible. The host variable takes a hostname (myapp.example.com for example) and determines the trigger for this rule. If the Host header matches this value the rule will be used. The Host header can either be used by using different (sub)domains which all point to your cluster or by specifying rewrite rules in HAProxy, an example of such a rewrite is shown below:

frontend http\_front  
 **mode** http  
 bind **<**kube**-**balancer**-**ip**>**:80  
 acl api url\_beg **/**api  
 default\_backend http\_back  
 use\_backend http\_back\_api\_v1 **if** apibackend http\_back\_api\_v1  
 **mode** http  
 http**-**request del**-**header **Host**  
 http**-**request **add-**header **Host** api  
 server k8s**-**master**-**0 **<**kube**-**master0**-**ip**>**:32059 **check** fall 3 rise 2  
 server k8s**-**master**-**0 **<**kube**-**master0**-**ip**>**:32059 **check** fall 3 rise 2

In this file the acl rule sets a variable api (boolean) based on the path (if it starts with /api api=true), the default backend for this service is http\_back, but if api is true it will use the http\_back\_api\_v1 backend. This backend in turn deletes the current Host header and replaces it with api. If an ingress rule is defined with host: api this request will route to that service.

**Deploying Kubernetes dashboard**

Deploy the Kubernetes dashboard:

kubectl create -f <https://raw.githubusercontent.com/kubernetes/dashboard/master/aio/deploy/recommended/kubernetes-dashboard.yaml>

**AUTHORIZATION**

In order to be able to access the Kubernetes dashboard a token is required. In order to use this an admin user has to be created, first create these two files:

*# admin-user.conf*  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: admin-user  
 namespace: kube-system*# admin-user-binding.conf*  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
 name: admin-user  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: ClusterRole  
 name: cluster-admin  
subjects:  
- kind: ServiceAccount  
 name: admin-user  
 namespace: kube-system

Next, create the user and the role binding:

kubectl create -f admin-user.conf  
kubectl create -f admin-user-binding.conf

**RETRIEVING ACCESS TOKEN**

The access token can be retrieved using the following command:

kubectl -n kube-system describe secret **$(**kubectl -n kube-system get secret | grep admin-user-token | awk '{print $1}'**)**

**ACCESSING THE DASHBOARD**

The dashboard can only be accessed through a proxy to the cluster, open a terminal and execute the following command:

kubectl proxy

As long as this command is active a proxy connection is available on your local machine (port 8001 by default) to the cluster. You can then access the Kubernetes dashboard on:

<http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/>

You will be prompted for authentication, select the token and paste the contents of [Retrieving access token](https://bitbucket.org/svenhakvoort/workspace/snippets/Leang9/kubernetes-cluster-setup-multiple-master#retrieving-access-token).

**ENABLING METRICS**

To enable dashboard metrics first deploy Grafana and InfluxDB:

kubectl apply -f <https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/kube-config/influxdb/influxdb.yaml>kubectl create -f <https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/kube-config/influxdb/grafana.yaml>

Create config files for Heapster:

*# heapster-rbac.yaml*  
kind: ClusterRoleBinding  
apiVersion: rbac.authorization.k8s.io/v1beta1  
metadata:  
 name: heapster  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: ClusterRole  
 name: heapster  
subjects:  
- kind: ServiceAccount  
 name: heapster  
 namespace: kube-system*# heapster.yaml*  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: heapster  
 namespace: kube-system  
---  
apiVersion: extensions/v1beta1  
kind: Deployment  
metadata:  
 name: heapster  
 namespace: kube-system  
spec:  
 replicas: 1  
 template:  
 metadata:  
 labels:  
 task: monitoring  
 k8s-app: heapster  
 spec:  
 serviceAccountName: heapster  
 containers:  
 - name: heapster  
 image: k8s.gcr.io/heapster-amd64:v1.5.4  
 imagePullPolicy: IfNotPresent  
 command:  
 - /heapster  
 - --source=kubernetes.summary\_api:''?useServiceAccount=true&kubeletHttps=true&kubeletPort=10250&insecure=true  
 - --sink=influxdb:http://monitoring-influxdb.kube-system.svc:8086  
---  
apiVersion: v1  
kind: Service  
metadata:  
 labels:  
 task: monitoring  
 kubernetes.io/cluster-service: 'true'  
 kubernetes.io/name: Heapster  
 name: heapster  
 namespace: kube-system  
spec:  
 ports:  
 - port: 80  
 targetPort: 8082  
 selector:  
 k8s-app: heapster

Create the heapster user config:

*# heapster-user.yaml*  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRole  
metadata:  
 name: heapster  
rules:  
- apiGroups:  
 - ""  
 resources:  
 - pods  
 - nodes  
 - namespaces  
 verbs:  
 - get  
 - list  
 - watch  
- apiGroups:  
 - extensions  
 resources:  
 - deployments  
 verbs:  
 - get  
 - list  
 - update  
 - watch  
- apiGroups:  
 - ""  
 resources:  
 - nodes/stats  
 verbs:  
 - get

Create the user and deploy Heapster:

kubectl create -f heapster-role.yamlkubectl create -f heapster.yaml  
kubectl create -f heapster-rbac.yaml

Result:

