Kubernetes

Under the hood journey

Marcos Vallim • Developer Master • mvallim@ciandt.com

Agenda

- Objective
- Networking
- Virtualization
- Components Setup
  - Network Principal (Gateway, DNS, DHCP, NAT, etc.)
  - Cluster HAProxy
  - Kubernetes: Master
  - Kubernetes: Worker
  - Kubernetes Dashboard
  - MetalLB: Extension for Kubernetes Load Balancer
  - Storage: GlusterFS
  - Heketi: Extension for Kubernetes Storage

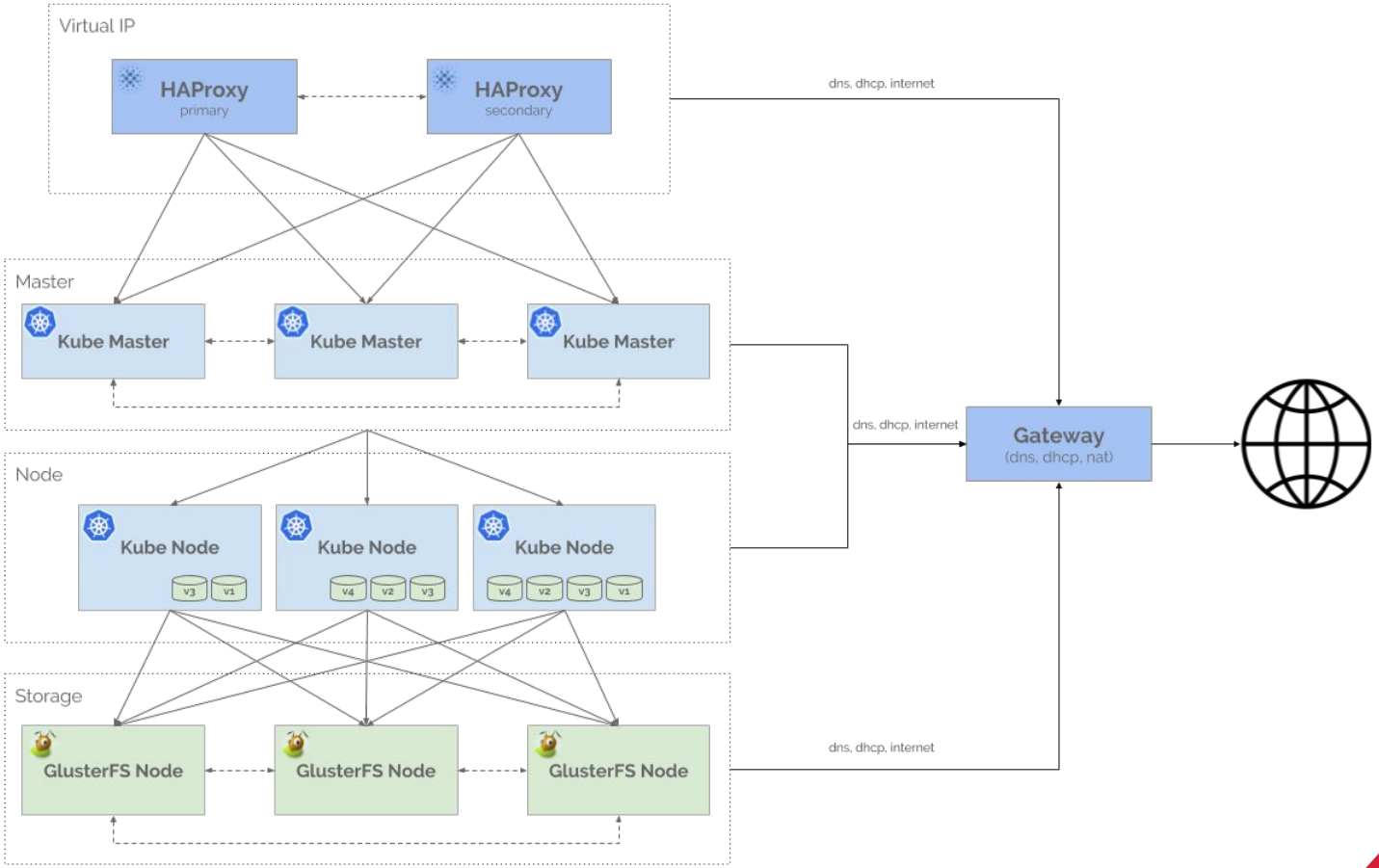
Objective

Objective

The main purpose here are the steps for installing and configuring a **Kubernetes** cluster and all its components, care and planning.

Technologies

Common Cluster

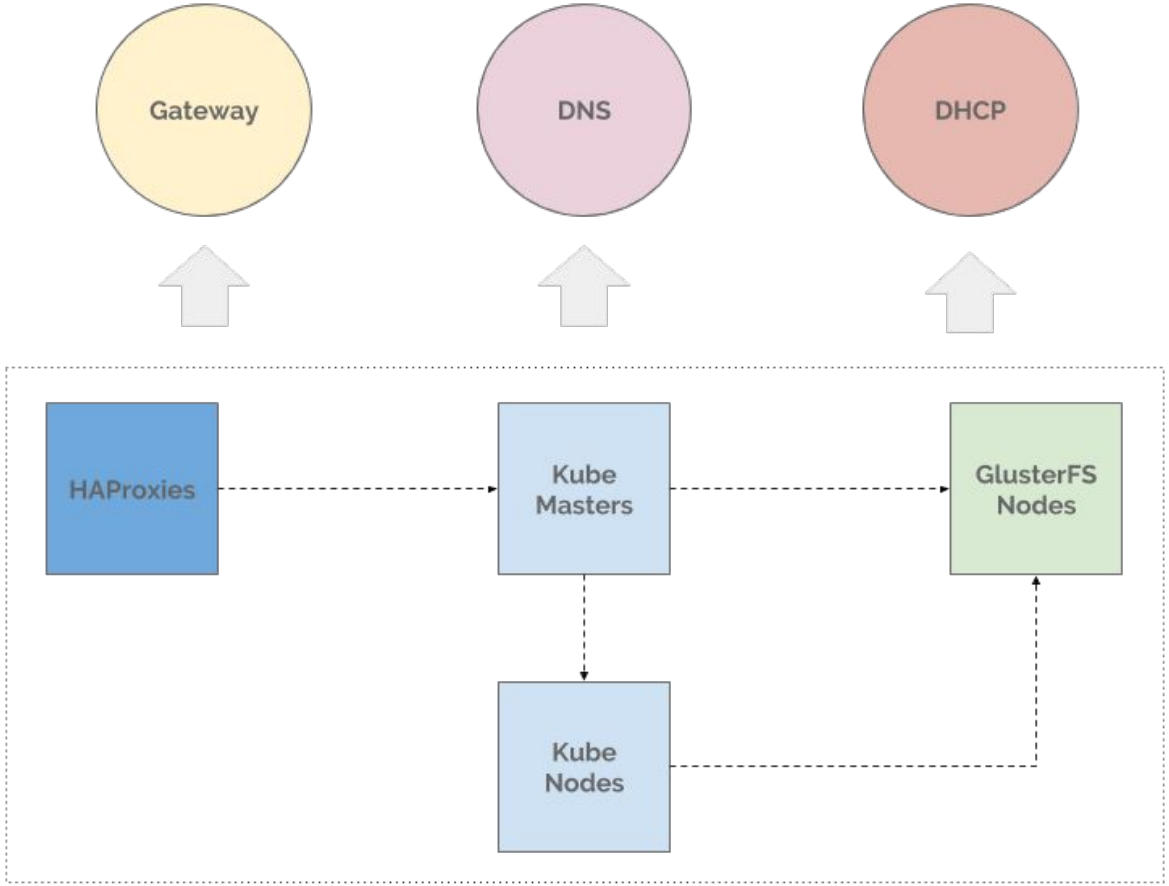


Networking

Network

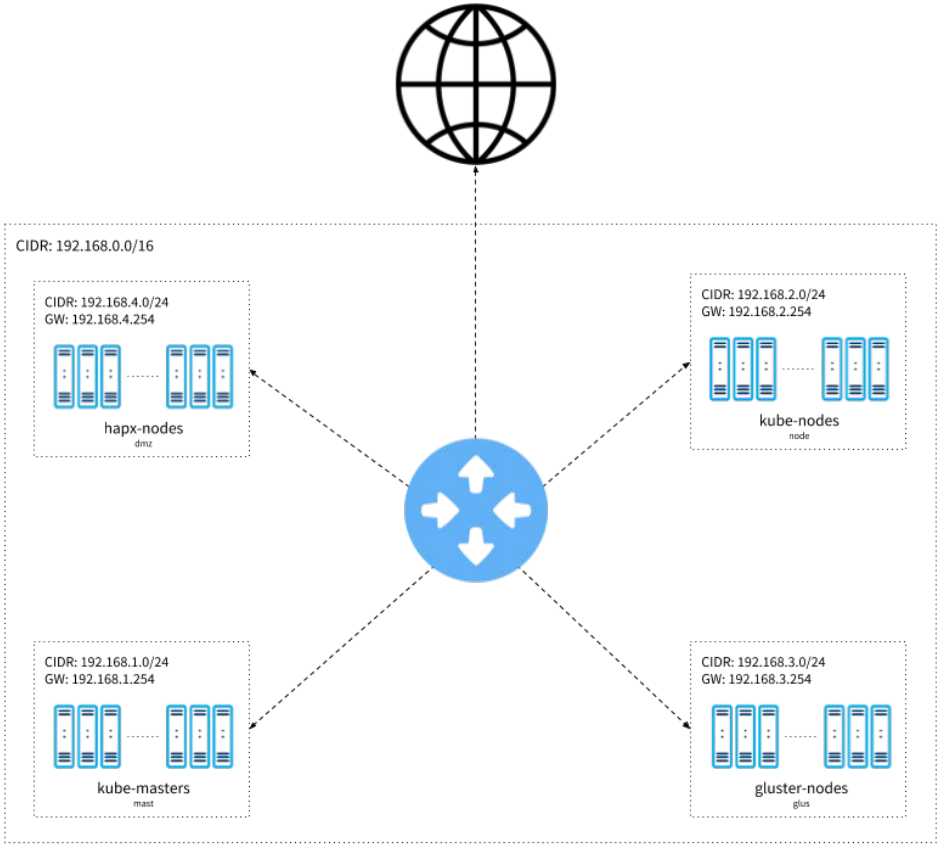
For everything to work as expected, proper network planning is required, as are network services - **DNS**, **DHCP**, **Router**, and **NAT**.

Network Services



Networking Diagram

CIDR	192.168.0.0/16
POD CIDR	10.244.0.0/16
Subnet - mast	192.168.1.0/24
Subnet - node	192.168.2.0/24
Subnet - glus	192.168.3.0/24
Subnet - dmz	192.168.4.0/24
DHCP - mast	192.168.1.50 - 192.168.1.200
DHCP - node	192.168.2.50 - 192.168.2.200
DHCP - glus	192.168.2.50 - 192.168.2.200
DHCP - dmz	192.168.4.50 - 192.168.4.200
DNS/Gateway - mast	192.168.1.254
DNS/Gateway - node	192.168.2.254
DNS/Gateway - glus	192.168.3.254
DNS/Gateway - dmz	192.168.4.254
Control Plane	192.168.4.20
MetalLB	192.168.2.10 - 192.168.2.49



Virtualization

Virtualization

We will use **VirtualBox** as a virtualization tool. Let's take advantage of the ability to create an OS base image with some software already pre-installed and configured, and configure the others using **cloud-init**.

Base Image

SO	**Debian 9 (stretch)**
Packages	build-essential
	module-assistant
	resolvconf
	ntp
	sudo
	cloud-init
	VirtualBox Guest OS

Resources	
RAM	512Mb
CPU	1
NIC	1 (Host-Only)
HDD	50Gb

Gateway Image

SO	**Debian 9 (stretch)**
Packages	dnsmasq
	iptables
	ntopng

Resources	
RAM	512Mb
CPU	1
NIC	1 (NAT)
	1 (Internal Network - **dmz**)
	1 (Internal Network - **mast**)
	1 (Internal Network - **node**)
	1 (Internal Network - **glus**)
HDD	50Gb

HAProxy Image

SO	**Debian 9 (stretch)**
Packages	pacemaker
	corosync
	crmsh
	haproxy

Resources	
RAM	512Mb
CPU	1
NIC	1 (Internal Network - **dmz**)
HDD	50Gb

Kube Master Image

SO	**Debian 9 (stretch)**
Packages	docker-ce
	containerd.io
	kubectl
	kubelet
	kubeadm
	glusterfs-client
	iptables

Resources	
RAM	2048Mb
CPU	2
NIC	1 (Internal Network - **mast**)
HDD	50Gb

Kube Node Image

SO	**Debian 9 (stretch)**
Packages	docker-ce
	containerd.io
	kubectl
	kubelet
	kubeadm
	glusterfs-client
	iptables

Resources	
RAM	2048Mb
CPU	2
NIC	1 (Internal Network - **node**)
HDD	50Gb

Gluster Image

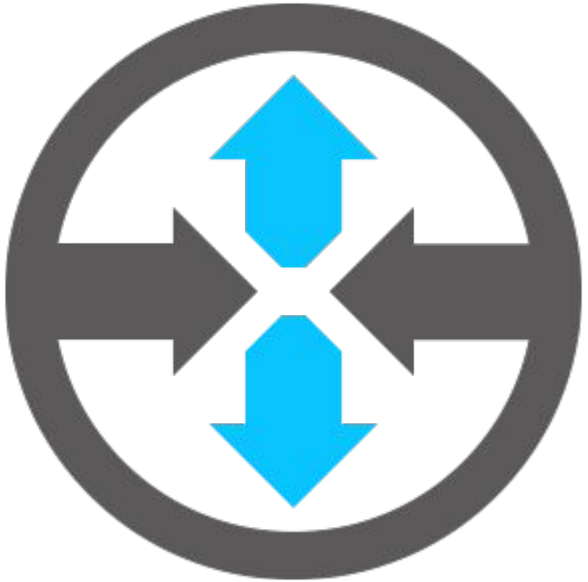
SO	**Debian 9 (stretch)**
Packages	gluster-server
	gluster-common
	gluster-client
	thin-provisioning-tools
	xfsprogs

Resources	
RAM	2048Mb
CPU	2
NIC	1 (Internal Network - **glus**)
HDD	50Gb - SO
	500Gb - store

Network Principal

Components Setup: Network Principal

Basic resource that will provide routing between networks and vital services - **DNS**, **DHCP** and **NAT** / **Routing**.



Network Principal: NAT

/etc/sysctl.d/10-gateway.conf

```
net.ipv4.ip_forward=1
```

```
#iptables -A FORWARD -i enp0s8 -j ACCEPT
#iptables -A FORWARD -o enp0s8 -j ACCEPT

#iptables -A FORWARD -i enp0s9 -j ACCEPT
#iptables -A FORWARD -o enp0s9 -j ACCEPT

#iptables -A FORWARD -i enp0s10 -j ACCEPT
#iptables -A FORWARD -o enp0s10 -j ACCEPT

#iptables -A FORWARD -i enp0s16 -j ACCEPT
#iptables -A FORWARD -o enp0s16 -j ACCEPT

#iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE
```

Network Principal: DNS

/etc/dnsmasq.d/dns

```
domain-needed
bogus-priv
bind-interfaces
domain=kube.local
```

/etc/dnsmasq.d/resolv-file

```
server=208.67.222.222
server=208.67.220.220
```

Network Principal: DHCP / Routing

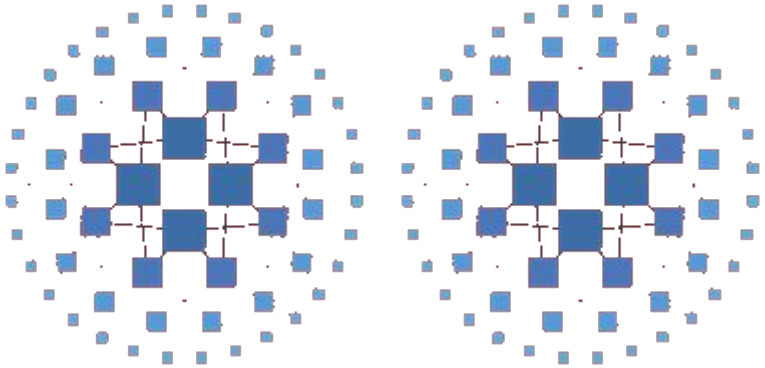
/etc/dnsmasq.d/dhcp

```
dhcp-range=enp0s8,192.168.1.50,192.168.1.100,12h
dhcp-range=enp0s9,192.168.2.50,192.168.2.200,12h
dhcp-range=enp0s10,192.168.3.50,192.168.3.200,12h
dhcp-range=enp0s16,192.168.4.50,192.168.4.200,12h

dhcp-option=enp0s8,option:dns-server,192.168.1.254
dhcp-option=enp0s9,option:dns-server,192.168.2.254
dhcp-option=enp0s10,option:dns-server,192.168.3.254
dhcp-option=enp0s16,option:dns-server,192.168.4.254

dhcp-option=enp0s8,option:router,192.168.1.254
dhcp-option=enp0s9,option:router,192.168.2.254
dhcp-option=enp0s10,option:router,192.168.3.254
dhcp-option=enp0s16,option:router,192.168.4.254

dhcp-option=enp0s8,option:classless-static-route,0.0.0.0/0,192.168.1.254
dhcp-option=enp0s9,option:classless-static-route,0.0.0.0/0,192.168.2.254
dhcp-option=enp0s10,option:classless-static-route,0.0.0.0/0,192.168.3.254
dhcp-option=enp0s16,option:classless-static-route,0.0.0.0/0,192.168.4.254
```

Load Balancer: HAProxy

Components Setup: Cluster HAProxy

Resource that will be responsible for the high availability of the **HAProxy** service which in turn is responsible for load balancing for the **Master Nodes**.



Components Setup: Cluster

/etc/corosync/corosync.conf

```
totem {
  version: 2
  cluster_name: debian
  token: 3000
  token_retransmits_before_loss_const: 10
  clear_node_high_bit: yes
  crypto_cipher: aes256
  crypto_hash: sha256
  interface {
    ringnumber: 0
    bindnetaddr: 192.168.4.0
    mcastaddr: 239.255.1.1
    mcastport: 5405
    ttl: 1
  }
}
```

Components Setup: Cluster

/etc/corosync/corosync.conf

```
logging {
  fileline: off
  to_stderr: no
  to_logfile: yes
  logfile: /var/log/corosync/corosync.log
  to_syslog: yes
  syslog_facility: daemon
  debug: off
  timestamp: on
  logger_subsys {
    subsys: QUORUM
    debug: off
  }
}

quorum {
  provider: corosync_votequorum
  two_node: 1
  expected_votes: 2
}
```

Components Setup: Load Balancer HAProxy

/etc/haproxy/haproxy.cfg

```
defaults
  timeout client 20s
  timeout server 20s
  timeout connect 4s
  default-server init-addr last,libc,none

resolvers dns
  nameserver dns-01 192.168.4.254:53
  resolve_retries 3
  timeout retry 1s
  hold other 30s
  hold refused 30s
  hold nx 30s
  hold timeout 30s
  hold valid 10s
```

Components Setup: Load Balancer HAProxy

/etc/haproxy/haproxy.cfg

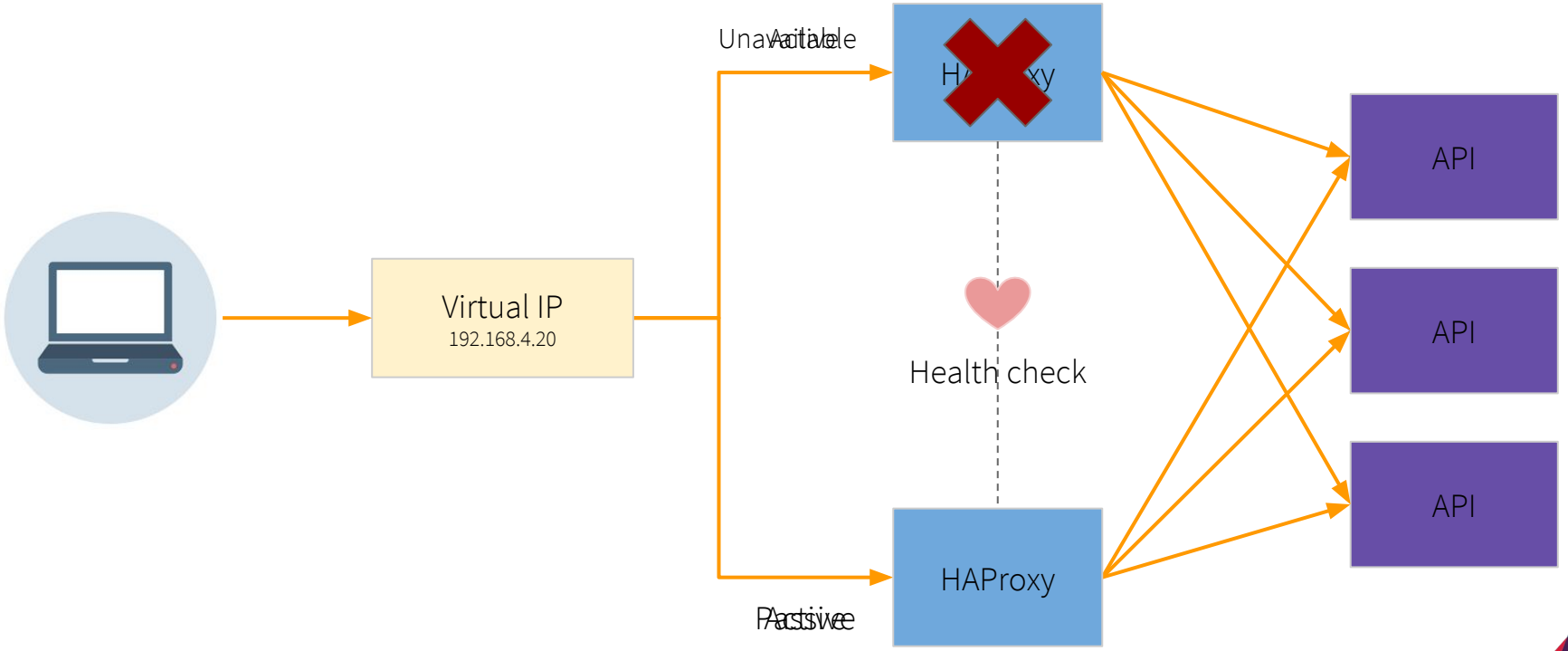
```
frontend kubernetes-apiserver-https
  bind *:6443
  mode tcp
  default_backend kubernetes-master-nodes

backend kubernetes-master-nodes
  mode tcp
  option tcp-check
  balance roundrobin
  server kube-mast01 kube-mast01:6443 check resolvers dns fall 3 rise 2
  server kube-mast02 kube-mast02:6443 check resolvers dns fall 3 rise 2
  server kube-mast03 kube-mast03:6443 check resolvers dns fall 3 rise 2 backup

listen stats
  bind *:32700
  stats enable
  stats uri /
  stats hide-version
  stats auth admin:admin
```

Components Setup: Pacemaker/Corosync

#crm configure

```
property stonith-enabled=no
property no-quorum-policy=ignore
property default-resource-stickiness=100
primitive virtual-ip-resource ocf:heartbeat:IPaddr2 params ip="192.168.4.20" nic="enp0s3"
cidr_netmask="32" meta migration-threshold=2 op monitor interval=20 timeout=60
on-fail=restart
primitive haproxy-resource ocf:heartbeat:haproxy op monitor interval=20 timeout=60
on-fail=restart
colocation loc inf: virtual-ip-resource haproxy-resource
order ord inf: virtual-ip-resource haproxy-resource
commit
bye
```

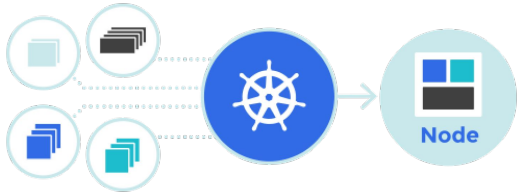
Cluster HAProxy

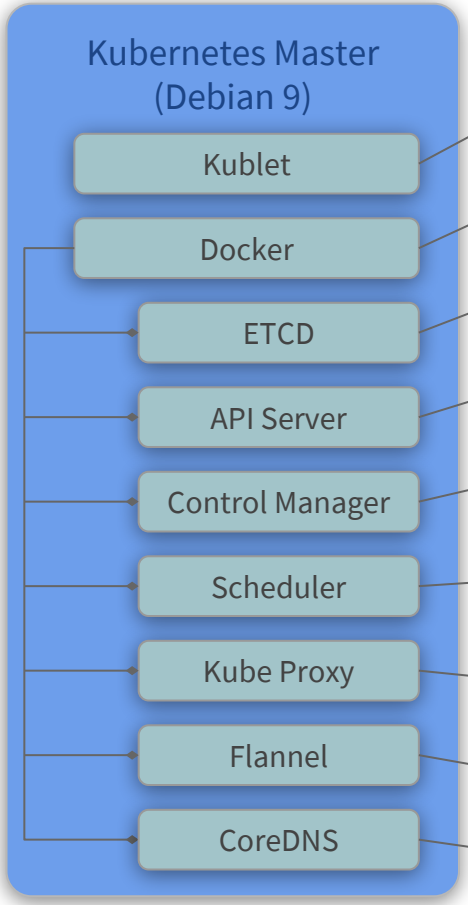


DEMO
High Availability

Kubernetes

Components Setup: Kubernetes Master





Agent que faz a gestão dos containers (cria, executa, remove)

Excução de containers.

Armazena configurações e estado do cluster.

Comunicação com o cluster.

Monitora os controladores de replicação e cria pods para manter o estado desejado.

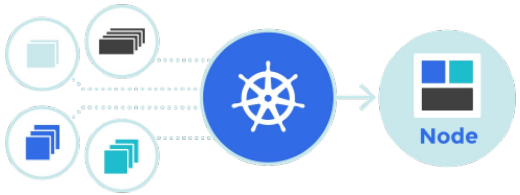
Executa tarefas de agendamento. Ex. criação de containers baseado na disponibilidade de recursos.

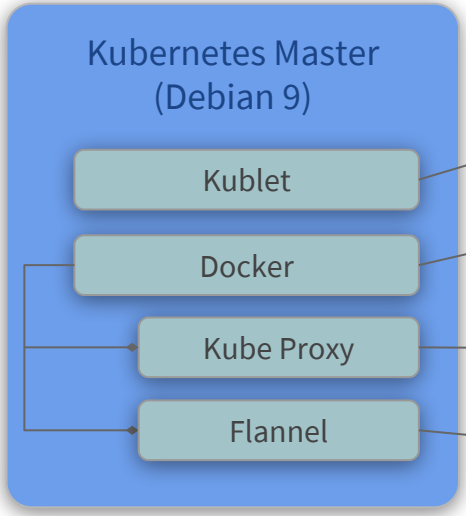
Encaminha o tráfego de rede para os containers.

Backend de rede, comunicação entre os nós.

Resolução de nomes interna do cluster

Components Setup: Kubernetes Worker





Agent que faz a gestão dos containers (cria, executa, remove)

Excução de containers.

Encaminha o tráfego de rede para os containers.

Backend de rede

Components Setup: Forward and Bridge

/etc/modules-load.d/bridge.conf

```
br_netfilter
```

/etc/sysctl.d/10-kubernetes.conf

```
net.ipv4.ip_forward=1
net.bridge.bridge-nf-call-iptables=1
net.bridge.bridge-nf-call-arptables=1
```

Components Setup: Tools

~/bin/copy-certificates.sh

```
USER=debian
CONTROL_PLANE_IPS="kube-mast02 kube-mast03"
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
```

~/bin/move-certificates.sh

```
USER=debian
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
```

Components Setup: Kubernetes Master Init

vi kubeadm-config.yaml

```
apiVersion: kubeadm.k8s.io/v1beta1
kind: ClusterConfiguration
kubernetesVersion: stable
apiServer:
  certSANs:
  - "192.168.4.20"
controlPlaneEndpoint: "192.168.4.20:6443"
networking:
  podSubnet: 10.244.0.0/16
```

Only on first master node

```
#kubeadm init --config=kubeadm-config.yaml

#ssh-keygen -t rsa -b 4096

#ssh-copy-id debian@kube-mast02
#ssh-copy-id debian@kube-mast03

#~/bin/copy-certificates.sh
```

Components Setup: Kubernetes Network Model

Assumptions:

- All containers can communicate with all other containers without NAT
- All nodes can communicate with all containers (and vice versa) without NAT
- The IP with which a container sees itself is the same IP as the others see it

Components Setup: Kubernetes Backend Network

“Flannel is a simple and easy way to configure a layer 3 network fabric designed for Kubernetes.
Flannel runs a small, single binary agent called flanneld on each host, and is responsible for allocating a subnet lease to each host out of a larger, preconfigured address space. Flannel uses either the Kubernetes API or etcd directly to store the network configuration, the allocated subnets, and any auxiliary data (such as the host's public IP). Packets are forwarded using one of several backend mechanisms including VXLAN and various cloud integrations.”

Only on first master node

```
#kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/a70459be0084506e4ec919aa1c114638878db11b/Documentation/kube-flannel.yml
```

Components Setup: Kubernetes Master Replica Join

Only on first master node

```
#kubeadm token create --print-join-command
```

Only on master replicas

```
#~/bin/move-certificates.sh

#kubeadm join 192.168.4.20:6443 \
      --token ??? \
      --discovery-token-ca-cert-hash sha256:??? \
      --experimental-control-plane
```

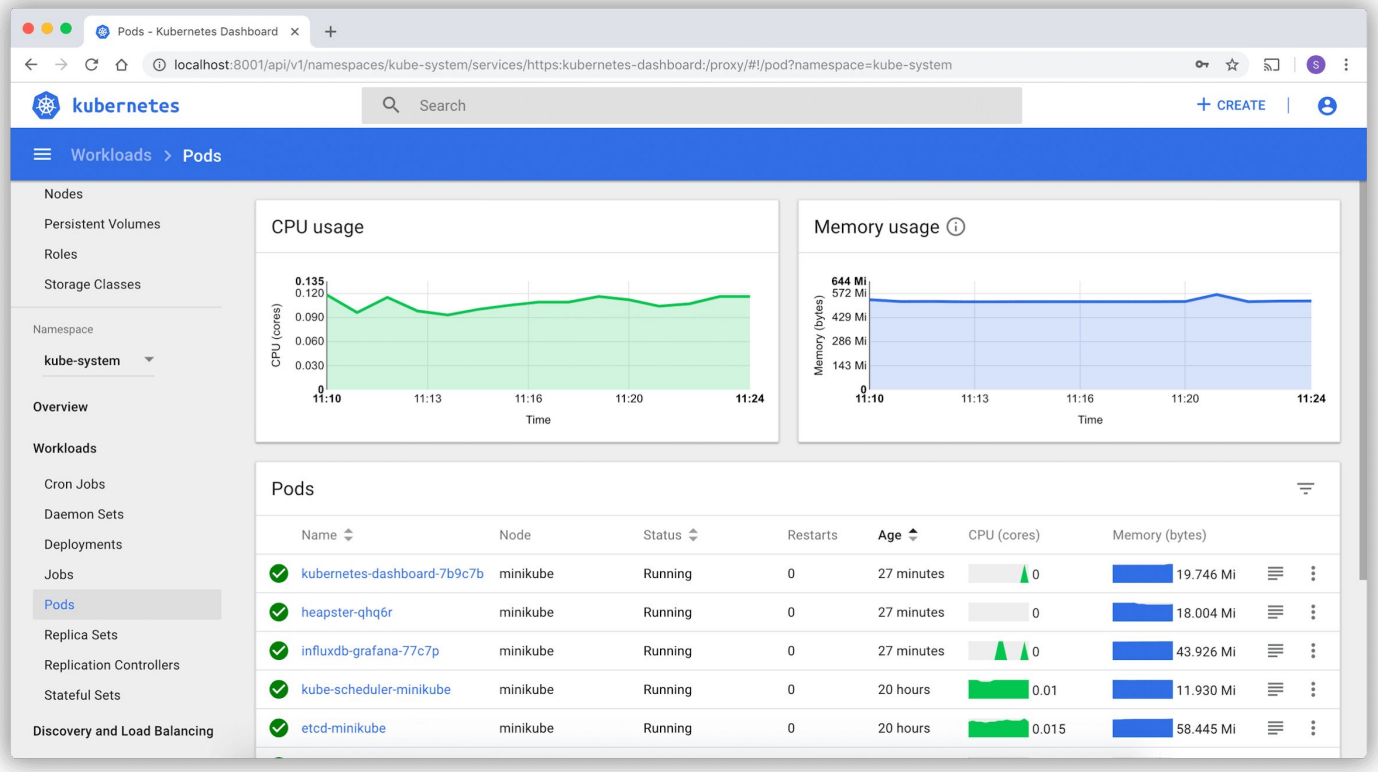
Components Setup: Kubernetes DNS

“CoreDNS is a fast and flexible DNS server. The keyword here is flexible: with CoreDNS you are able to do what you want with your DNS data by utilizing plugins. If some functionality is not provided out of the box you can add it by writing a plugin.”

Components Setup: Fixing CoreDNS

```
kubectl get nodes

kubectl get deploy --all-namespaces -o wide

kubectl get deploy coredns -n kube-system -o yaml > coredns.yaml

==========
vi coredns.yaml

replicas: 3

nodeSelector:
  node-role.kubernetes.io/master: ""

==========

kubectl apply -f coredns.yaml -n kube-system

watch -n1 kubectl get pods -n kube-system
```

Components Setup: Kubernetes Dashboard



Components Setup: Kubernetes Dashboard

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

kubectl get deploy --all-namespaces -o wide

kubectl get deploy kubernetes-dashboard -n kube-system -o wide

kubectl get deploy kubernetes-dashboard -n kube-system -o yaml > kubernetes-dashboard.yaml

==========

vi kubernetes-dashboard.yaml

replicas: 3

nodeSelector:
  node-role.kubernetes.io/master: ""

==========

kubectl apply -f kubernetes-dashboard.yaml -n kube-system

watch -n1 kubectl get pods -n kube-system
```

Components Setup: Kubernetes Dashboard

```
kubectl create serviceaccount cluster-admin-dashboard -n kube-system

kubectl create clusterrolebinding cluster-admin-dashboard \
      --clusterrole=cluster-admin \
      --serviceaccount=default:cluster-admin-dashboard \
      --dry-run -o yaml -n kube-system > cluster-admin-role-binding.yaml

=================

vi cluster-admin-role-binding.yaml

namespace: kube-system

=================

kubectl apply -f cluster-admin-role-binding.yaml -n kube-system

kubectl get secret -n kube-system

kubectl describe secret cluster-admin-dashboard-token-????? -n kube-system

kubectl proxy
```

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

DEMO
Dashboard

Components Setup: Kubernetes MetalLB

“MetalLB is a load-balancer implementation for bare metal Kubernetes clusters, using standard routing protocols.”

Why?

Kubernetes does not offer an implementation of network load-balancers (Services of type LoadBalancer) for bare metal clusters. The implementations of Network LB that Kubernetes does ship with are all glue code that calls out to various IaaS platforms (GCP, AWS, Azure…). If you’re not running on a supported IaaS platform (GCP, AWS, Azure…), Load Balancers will remain in the “pending” state indefinitely when created.

Components Setup: Kubernetes MetalLB

```
kubectl apply -f
https://raw.githubusercontent.com/google/metallb/v0.7.3/manifests/metallb.yaml

kubectl get deploy --all-namespaces -o wide

kubectl get deploy --all-namespaces -o wide

kubectl get deploy controller -n metallb-system -o wide

kubectl get deploy controller -n metallb-system -o yaml > controller.yaml

================
vi ~/controller.yaml

replicas: 3

tolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
nodeSelector:
  node-role.kubernetes.io/master: ""
================

kubectl apply -f ~/controller.yaml -n metallb-system
```

Components Setup: Kubernetes MetalLB

```
vi metallb-configmap.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.2.10-192.168.2.49
================

kubectl apply -f metallb-configmap.yaml -n metallb-system

kubectl get configmap -n metallb-system -o wide

watch -n1 kubectl get pods -n metallb-system
```

Example application: https://kubernetes.io/docs/tutorials/stateless-application/guestbook/

DEMO
Load balancer

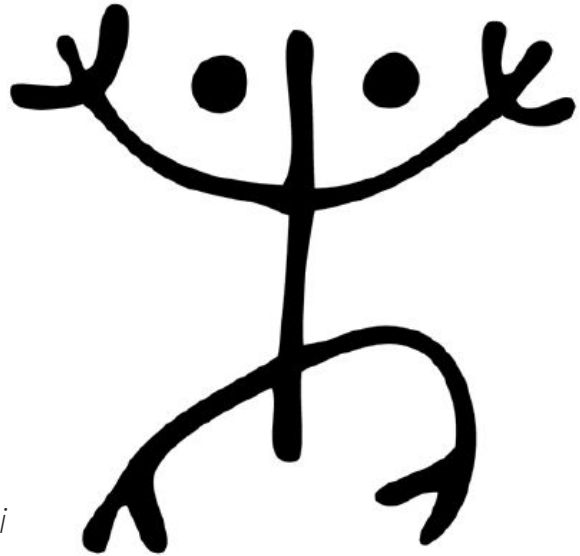
Storage: GlusterFS

Components Setup: GlusterFS

Resource that will be responsible for the availability of the service of storage of dynamic volumes requested by **Kubernetes**.

Heketi

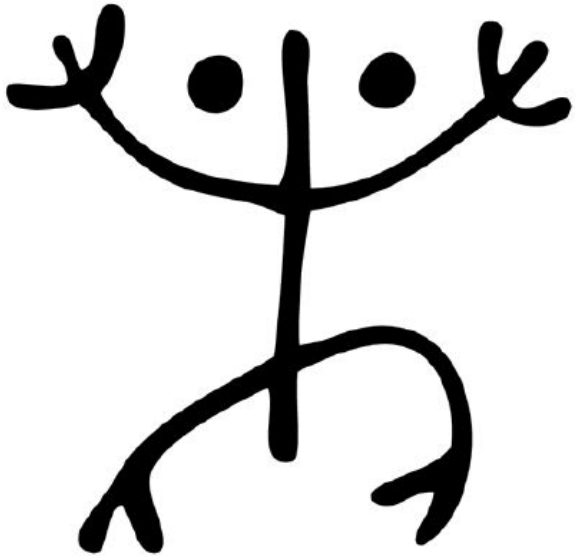
Components Setup: Heketi



“Heketi provides a RESTful management interface which can be used to manage the life cycle of GlusterFS volumes. With Heketi, cloud services like OpenStack Manila, Kubernetes, and OpenShift can dynamically provision GlusterFS volumes with any of the supported durability types. Heketi will automatically determine the location for bricks across the cluster, making sure to place bricks and its replicas across different failure domains. Heketi also supports any number of GlusterFS clusters, allowing cloud services to provide network file storage without being limited to a single GlusterFS cluster.”

Components Setup: Heketi

```
git clone git@github.com:gluster/gluster-kubernetes.git

cd deploy

vi topology

kubectl create namespace glusterfs

./gk-deploy --ssh-keyfile ~/.ssh/id_rsa --ssh-user root --cli kubectl \
       --templates_dir ./kube-templates --namespace glusterfs \
       --user-key none --admin-key none topology.json

==================
vi glusterfs-storageclass.yaml

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: glusterfs-storage
  namespace: glusterfs
provisioner: kubernetes.io/glusterfs
allowVolumeExpansion: true
reclaimPolicy: Retain
parameters:
  resturl: "http://10.244.xxx.xxx:8080"
  restuser: "admin"
  restuserkey: "none"
  volumetype: "replicate:3"
==================

kubectl create -f glusterfs-storageclass.yaml
```



DEMO
Volumes

Questions?

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

Martin Fowler

References

- **NAT** - https://en.wikipedia.org/wiki/Network_address_translation
- **DNS** - https://en.wikipedia.org/wiki/Domain_Name_System
- **DHCP** - https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol
- **cloud-init** - https://cloudinit.readthedocs.io/en/latest/
- **dnsmasq** - http://www.thekelleys.org.uk/dnsmasq/doc.html
- **corosync** - http://corosync.github.io/corosync/
- **pacemaker** - https://clusterlabs.org/
- **HAProxy** - http://www.haproxy.org/
- **Docker** - https://docs.docker.com/
- **GlusterFS** - https://docs.gluster.org/en/latest/
- **XFS** - http://xfs.org/index.php/Main_Page
- **LVM** - http://www.sourceware.org/lvm2/
- **VirtualBox** - https://www.virtualbox.org/
- **Debian** - https://www.debian.org/
- **Kubernetes** - https://kubernetes.io/
- **Flannel** - https://github.com/coreos/flannel
- **CoreDNS** - https://github.com/coredns/coredns
- **MetalLB** - https://metallb.universe.tf/
- **Heketi** - https://github.com/heketi/heketi

Thank you!