Action Plan: Installing Kubernetes

→ Some considerations

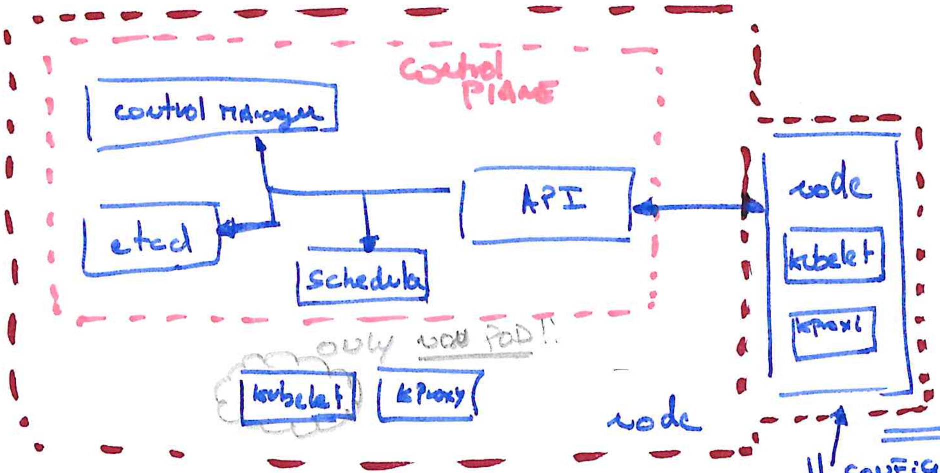
- Access by many user → we need some control management
- → Consistently available →
- adaptable / Scalable on Demand.
- HA ⇒ Distributed {in every Part}

→ ~~[illegible]~~

More?? on

kube Proxy

the parts of a kubernetes cluster



|| configured to run kubernetes Pods ||

etc pods but not only pods

GOAL!!

Simpler installation: everything on the same node

→ resources in kubernetes are again organized in namespaces. ≠ kernel namespaces

kube-System namespace } → base namespace where it is ~~[illegible]~~ possible to find all the managin ~~[illegible]~~ resources } →

obj Spects on: Persistent entities ~~that~~ represent the state of your cluster

control-manager : Loop that watches the status of the klurter (by looking into the Api Server) And moves ~~the~~ it towards the Desired state

APID server : Validate ~~[illegible]~~ Data (obsect) in input, Provides a frontend for the cluster showed state

kube-proxy : kubernetes network proxy

Scheduler : Assign PoD to nodes. The scheduler determines wich nodes are valid placements accordingly to available resources.

kubelet: Responsible for registering the node to the API server

→ One of the few **non-container** applications part of kubernetes

note: All these pods (Special pods) share the same NetNS or pid ls

⇒ you can check it by observing that

- kube-proxy
- kube-API
- kube-schedder
- etcd

are opening ~~ports~~ [**network**] **ports** **on the node**!!

[ss -tunalp]

Bird
CALICO ?
CEPH

starting from our VM we are going to **Install kubernetes**
↳ 2core + 2GB ram

This first installation will not be HA, Later on we will see How to move from a basic installation to HA.

Prefered Method ⇒ kube ADM

- Prepae the field for cri-o & kubelet

kernel modules

→ overlay → enable overlay fs kernel module

→ br_net_filter → Allow Net FILTER @ Bridge LEVEL.

But what is kubernetes

K8S → by GOOGLE (....)

K3S → by RANCHER (SUSE)
(Light kubernetes
less components, eg
Etcd → Sql Lite)

K0S → even smaller than K3S
(young project)

kernel Parameters

kernel Parameters.

→ bridge_nf_call_* ⇒ enable nf tables calls (for Firewall @ bridge level)

→ ip4.if_forward ⇒ Allows pkgs forwarding
↳ GET PKGS in from an interface and throw them out of another interface.

Cgroups: they perform resource control and are managed by SYSTEMD
↳ {How much ram / " " CPU a Process can use?} SAME as SLURM

LAter

Mishellaneous

Disable: Se Linux → 2 security modules on top of everything we have seen
- Se Linux
- App Armor

} → 2 diffrnt philosophies, kub only support Apparmor for now

Disable z-ram → a kind of swap in ram.

CRI-O configuration

/etc/crio/ → pure crio conf. {• CNI is conf. here partially

/etc/containers/~~registries.conf~~ for containers (registries, storage conf, ...)

/etc/cni/net.d/* ← real CNI configuration!!

```
type : bridge
bridge : cni0
   ⋮
masq : ...
```

} → Simplest Plugin.

obs on worker nodes?
obs on user Management?
Limits on worker nodes?

Limitation of this plugin! no inter node communication between pods

Very Good Startingpoint anyway!

FINAL INSTALLATION

KUBEADM

-- pod network -cidr → the ips of our pods! :-) Must be = to what we declared to ~~[illegible]~~. CNI!

-- services - cidr → we need some more knowledge to set this up... Later on...

Classes Inter Domain Routing

Just refer to the standard notation

-- control-plane -endpoint → we need to tell kub. where is the control plane!

this should point to a DNS name or better to an External load balancer over All the control planes we are hosting.

control plane HA

Hurray!! our "cluster" is up & running!!

~~Show that~~ crio- is playing a central role in ~~[illegible]~~ operations.

To the point that we can bypass kubelet to ask the status of the SINGLE node kubctl is the tool to explore the cluster we just created:

kube ctl version : obs: ±3 from kubernetes API minor version! $1.\overset{\text{minor}}{1}.0$

states of the cluster

kube ctl get --raw
- =/healtz
- =/livez
- =/readyz

?verbose optional.

if

~~it is better to define what they are used for~~

~~if FAILS the container is killed And restorted by kubelet~~

~~SAME AS HEALT but Act Before~~

~~when is ok (200) your application starts to receive traffic~~

Where is this information used?

well written application should expose All these 3 endpoints in their API in order to show

Probed by KUBECTl as

→ Livez ⇒ that their booting has ended correctly

Startup Probe

Longer failure threshold Between differt checks Are usually Done [Act As Liveness]

→ ready z ⇒ that they are ready to accept traffic

readiness Prob

once good, ~~your~~ the APP will start to receive traffic from outside

→ Healt z → ~~Startup~~ that the app is working as expected

Liveness Probe

when it fails the container is killed Abruptly. and restorted.

RANDOM NAMES can be API endpoints or files crated at runtime, kubernetes Doesn't care as long as ~~you~~ it is instructed to PROBE them.

Defined in the object as part of the SPEC

CONTAINER