kumina

MANAGED OPERATIONS

Container Management With Kubernetes

Waarom in het Engels?

- Gemakzucht (Nederlanders spreken prima Engels, niet-Nederlanders spreken zelden Nederlands)
- Vertaal deze woorden (met behoud van context) zonder te lachen:

Container	Federation
Image	Load
Performance	Engine
Orchestration	Namespace
Resource allocation	...

$(whoami)

- Proud father
- Working professionally as Linux sysop since 1997
- Started what would be my current company in 2004, as a student
- Cynical by nature
- Vim

Regarding Kumina

- Creation and implementation of custom made IT infrastructures
- Remote maintenance with 24x7 monitoring and alerting
- Remote support with unlimited support hours
- Consultancy regarding infrastructure requirements

In the beginning...

- Docker created containers
- Containers will solve all your problems
- Containers are secure
- Containers run anywhere without hassle

In the beginning...

- ~~Docker created containers~~

chroot became available in 1982 in BSD

- Containers will solve all your problems
- Containers are secure
- Containers run anywhere without hassle

In the beginning...

- ~~Docker created containers~~

chroot became available in 1982 in BSD

- Containers will solve ~~all~~ your problems

some (if your app is stateless)

- Containers are secure

- Containers run anywhere without hassle

In the beginning...

- ~~Docker created containers~~

chroot became available in 1982 in BSD

- Containers will solve ~~all~~ your problems

some (if your app is stateless)

- Containers are ~~secure~~

- Containers run anywhere without hassle

In the beginning...

- ~~Docker created containers~~

chroot became available in 1982 in BSD

- Containers will solve ~~all~~ your problems

some (if your app is stateless)

- Containers are ~~secure~~
- Containers run anywhere without hassle

In the beginning...

- ~~Docker created containers~~

chroot became available in 1982 in BSD

- Containers will solve ~~all~~ your problems

some (if your app is stateless)

- Containers are [redacted]

- Containers run anywhere without hassle

In the beginning...

- ~~Docker created containers~~

chroot became available in 1982 in BSD

- Containers will solve ~~all~~ your problems

some (if your app is stateless)

- Containers are [redacted] security is process, not a product

- Containers run anywhere without hassle

In the beginning...

- ~~Docker created containers~~

chroot became available in 1982 in BSD

- Containers will solve ~~all~~ your problems

some (if your app is stateless)

- Containers are [redacted] security is process, not a product

- Containers run anywhere ~~without hassle~~

good luck with that

~~In the beginning...~~

- ~~Docker created containers~~

  chroot became available in 1982 in BSD

- Containers will solve ~~all~~ your problems

  some (if your app is stateless)

- Containers are [redacted] security is process, not a product

- Containers run anywhere ~~without hassle~~

  good luck with that

In the (real) beginning...

- Operating-systems were designed in another era, with different requirements
- We still live with that legacy
- Process isolation was not important in those early years, as all software was developed locally and was trusted

Fake it!

- “chroot” was created by Bill Joy in 1982
- added to test BSD's installation and build system
- “CHange ROOT”, meaning the “/” as seen by the process(es) running inside a chroot jail
  - “Jail” came around in 1991 when Bill Cheswick created a honeypot for a cracker
- Still used a lot by for example Postfix and OpenSSH daemon
- After chroot, it took a while before new changes would surface

Jailbreak!

- FreeBSD Jails got introduced in 2000 by Poul-Henning Kamp
- Born out of a need to compartmentalise a system for multiple tenants
- Operating-system level virtualisation
- Generally used as VM hosting, but in fact eerily similar to LXC and Docker
- Origin of the word “jailbreak” (Wikipedia told me)

Advances

- Linux VServer (2001)
- Mount namespace (2002)
- Solaris Zones (2004)
- OpenVZ (2005)

Towards the Modern Stack

- Process containers/Cgroups (2006)
- Additional namespaces being introduced (2006)
  - UTS (2.6.19)
  - IPC (2.6.19)
  - user (2.6.23)
  - PID (2.6.24)
  - net (2.6.24)
  - more info: https://lwn.net/Articles/531114/

Towards the Modern Stack

- LXC (2008)
- LMCTFY (2013)
- Docker (2013)
- Rkt (2014)

Good Containers

- Only contain the least amount of dependencies necessary to start the application
- Run a single process or at least a single service
- Are stateless, or better yet, immutable
- Get config out of environment variables
- Export health metrics

Container Management

- Allows scaling of containers
- Manages connections to containers
- Manages container lifecycles
- Does not get in the way

- Containerization transforms the data center from being machine oriented to being application oriented

Google

- Disclaimer: Hard to find exact information (with Google, oh the irony)
- Distant past: Babysitter and Global Work Queue (up till ~2006)
- Present day: Borg (production from ~2005 onwards)
- Omega Project improved Borg (scheduling, PAXOS)

Mesos

- Mesos started as student project at UC Berkeley RAD Lab
- First called Nexus (and demostrated under that name in 2009)
- Twitter got (very) involved in 2010 after presentation from Benjamin Hindman
- Modelled after Google's Borg
  - It is said that Google's Omega takes lessons learned from Mesos back into Google

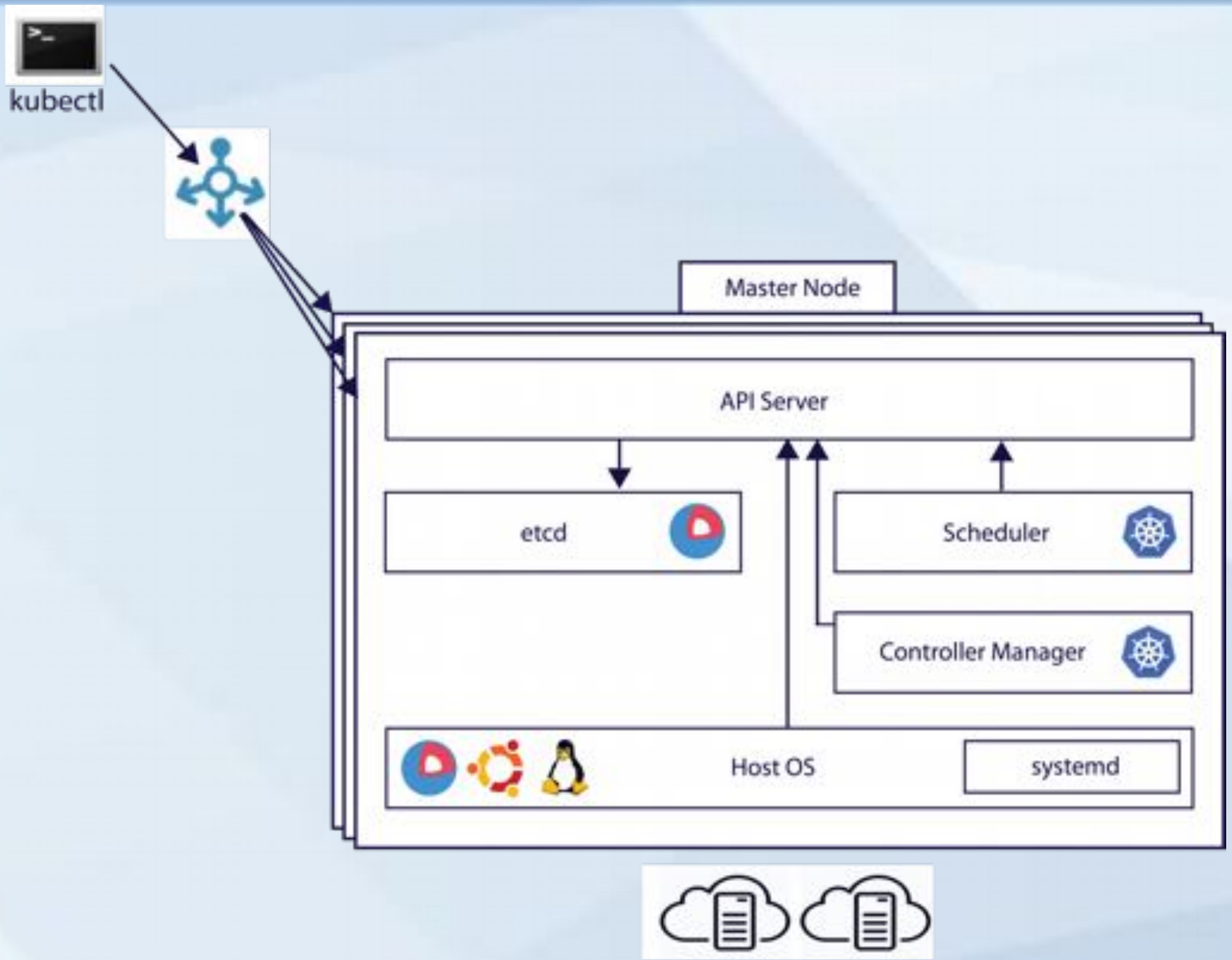
Kubernetes

- Started by Joe Beda, Brendan Burns and Craig McLuckie
  - Quickly joined by other Google Engineers
- Announced by Google mid-2014
- v1.0 released July 21, 2015
  - At which time it was donated to the Cloud Native Computing Foundation

Kubernetes Managing Core

- etcd
- API server
- Controller-manager
- Scheduler

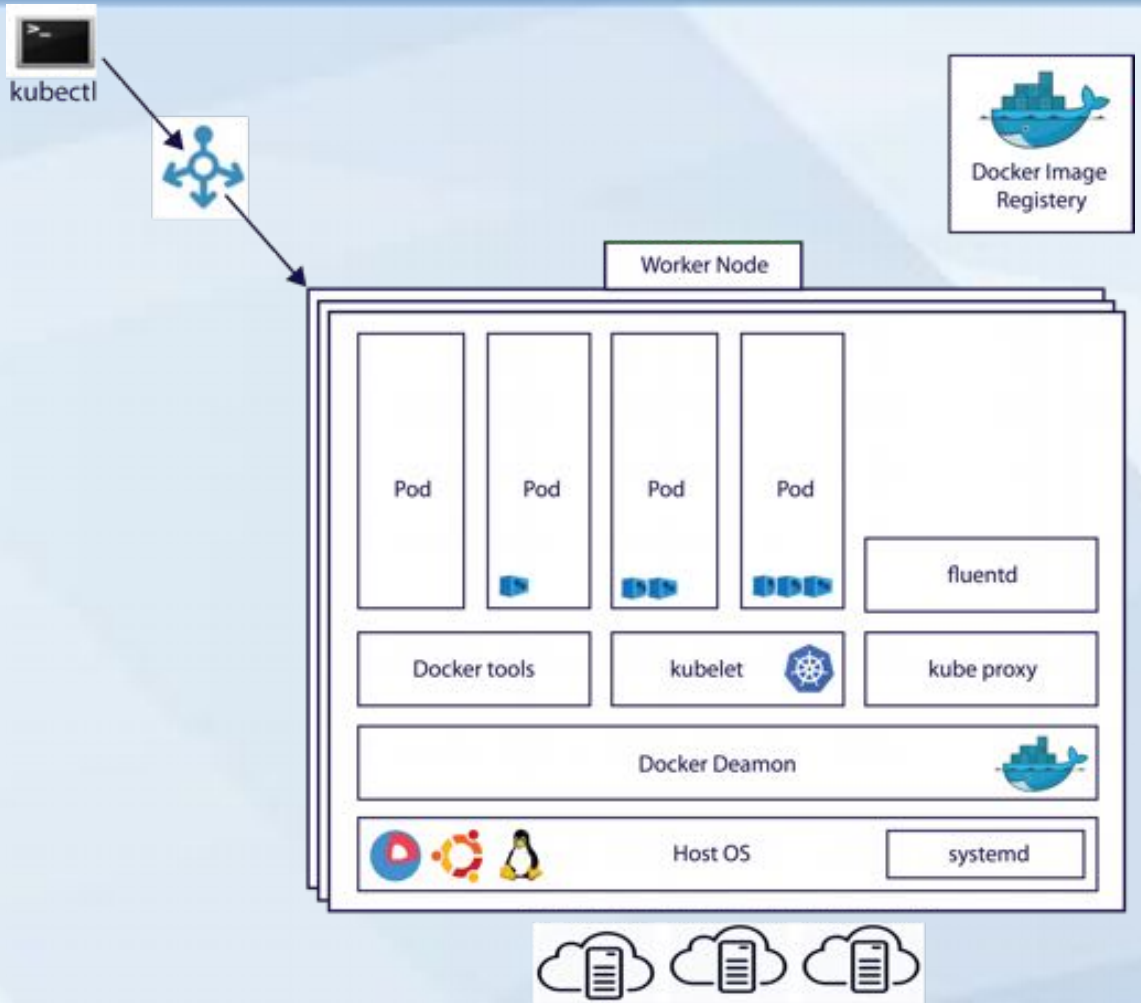
Kubernetes Managing Core



Kubernetes Worker Core

- Kubelet
- Kube-proxy
- cAdvisor
- Nodes get Labels
  - Examples:
    - `disk=ssd`
    - `in_dmz=true`
    - `have_crypto_accelerator=true`

Kubernetes Worker Core



The Components You Work With

- Pods
  - One or more containers running a service
  - Exposes ports
  - Has a unique IP address
  - Has Labels assigned
    - Examples:
      - `tier=frontend`
      - `release=prod`
  - Placement based on Selectors
    - Example: `disk=ssd`

The Components You Work With

- Controllers
  - Manages collections of Pods
  - Based on Selectors
    - Example: `app=bg-worker`
  - Different kinds of Controllers
    - Examples:
      - Replication Controller
      - DaemonSet Controller
      - StatefulSet Controller
      - Job Controller
      - Ingress Controller

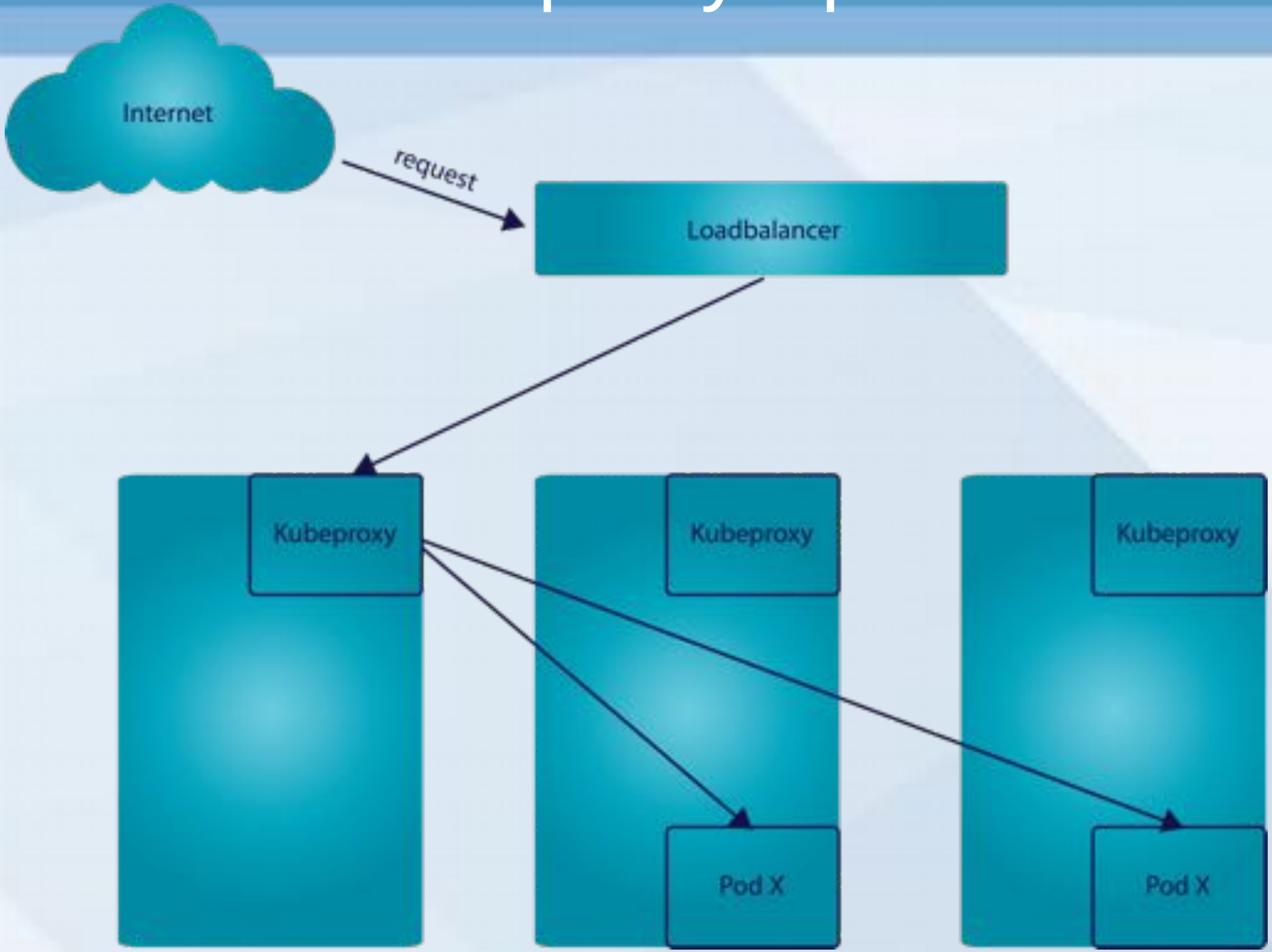
The Components You Work With

- Services
  - Provide access to applications running in Pods (if required)
  - Group the Pods associated with an application for easy access
  - Round-robin through all the Endpoints
  - Pods (not Controllers!) get added based on... Labels!
  - Can be used for internal services (backend) as well as exposed external services (frontend)
  - External services require external support (Ingress, ELB, traefik, etc.)

Kube-proxy Specifics

- Exposes a random port to the network on all nodes
- Routing to endpoints happens within kube-proxy

Kube-proxy Specifics



Lots of Options

- Deployments
- Namespaces
- Network Policies
- Persistent Volumes
- Secrets
- Service Accounts
- Federation

Failure Handling

- Pod eviction
  - First BestEffort Pods
  - Then Burstable Pods
  - Guaranteed Pods are not evicted
- Eviction based on resources usage
  - Example with OOM-killer:
    - BestEffort: oom_score_adj = 1000
    - Guaranteed: oom_score_adj = -998
    - Burstable: oom_score_adj = min(max(2, 1000 - (1000 * memoryRequestBytes) / machineMemoryCapacityBytes), 999)

Supporting Services

- Log Aggregation
- Trending, monitoring and alerting
- Dashboards

Federation

- Can be used to run applications over several Zones (eg. eu-west1 and eu-central1 for AWS)
- Can be used to create hybrid clouds
- Can be used to run an application over multiple clouds (!!)
  - Scale up where it is cheapest
  - Move entire workloads in case of datacenter failure
- One unified interface for all clouds
  - Splits dev from ops (again)

Questions?

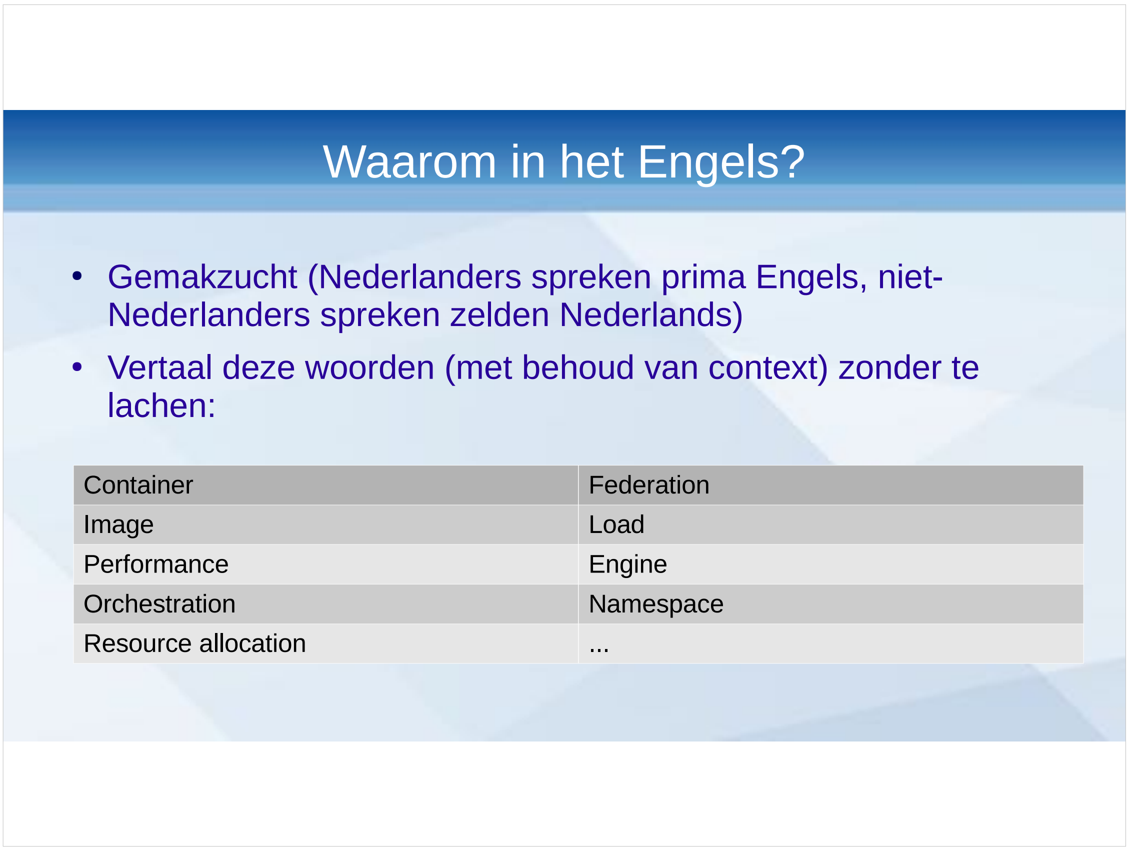
- Kubernetes community is very open, start at https://kubernetes.io

kumina
MANAGED OPERATIONS

Thanks for listening

www.kumina.nl

Container Management With Kubernetes

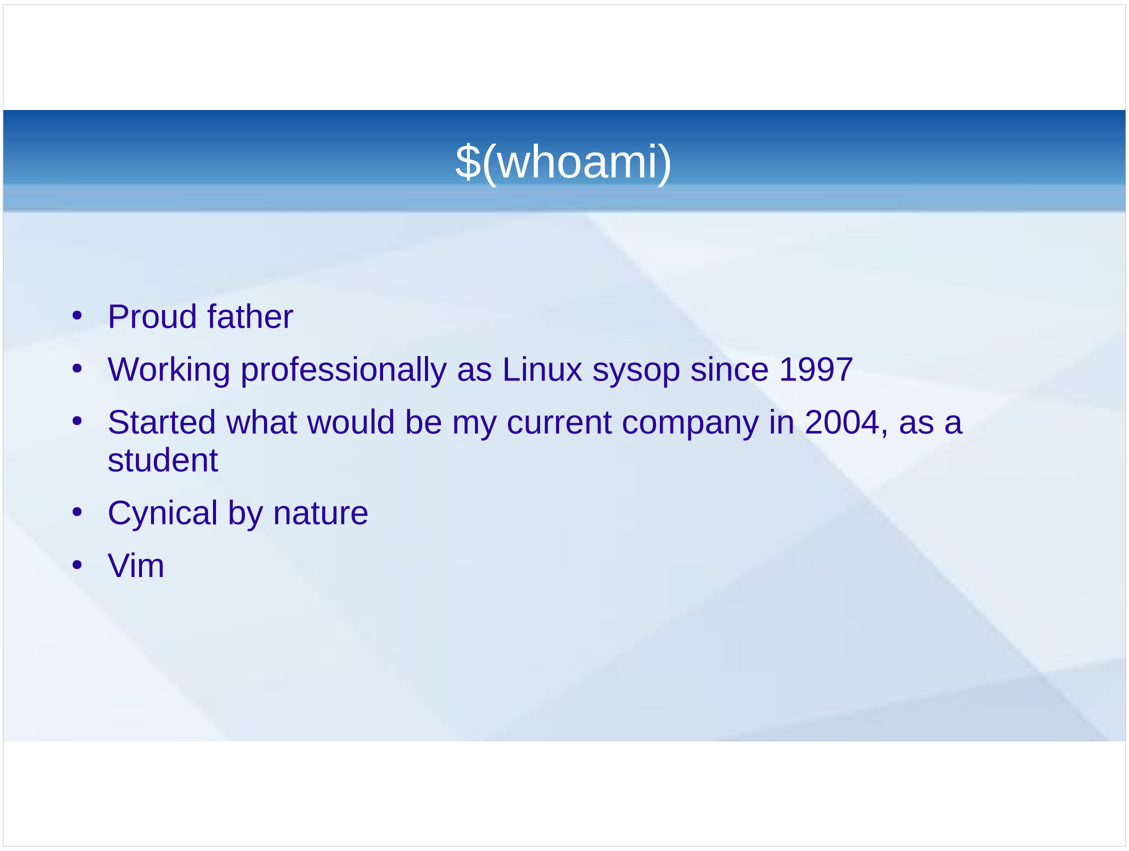


Waarom in het Engels?

- Gemakzucht (Nederlanders spreken prima Engels, niet-Nederlanders spreken zelden Nederlands)
- Vertaal deze woorden (met behoud van context) zonder te lachen:

Container	Federation
Image	Load
Performance	Engine
Orchestration	Namespace
Resource allocation	...

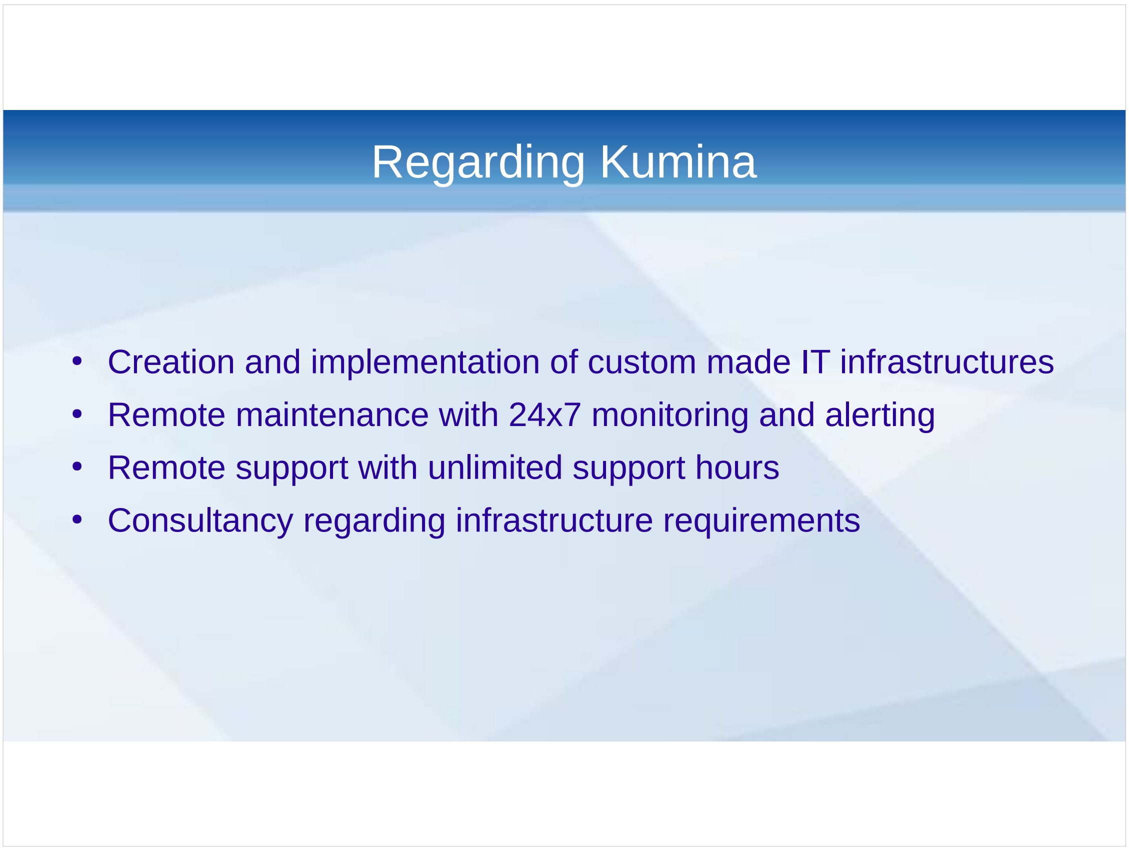
Makkelijk dat ik hem ook voor andere presentaties kan gebruiken.



$(whoami)

- Proud father
- Working professionally as Linux sysop since 1997
- Started what would be my current company in 2004, as a student
- Cynical by nature
- Vim

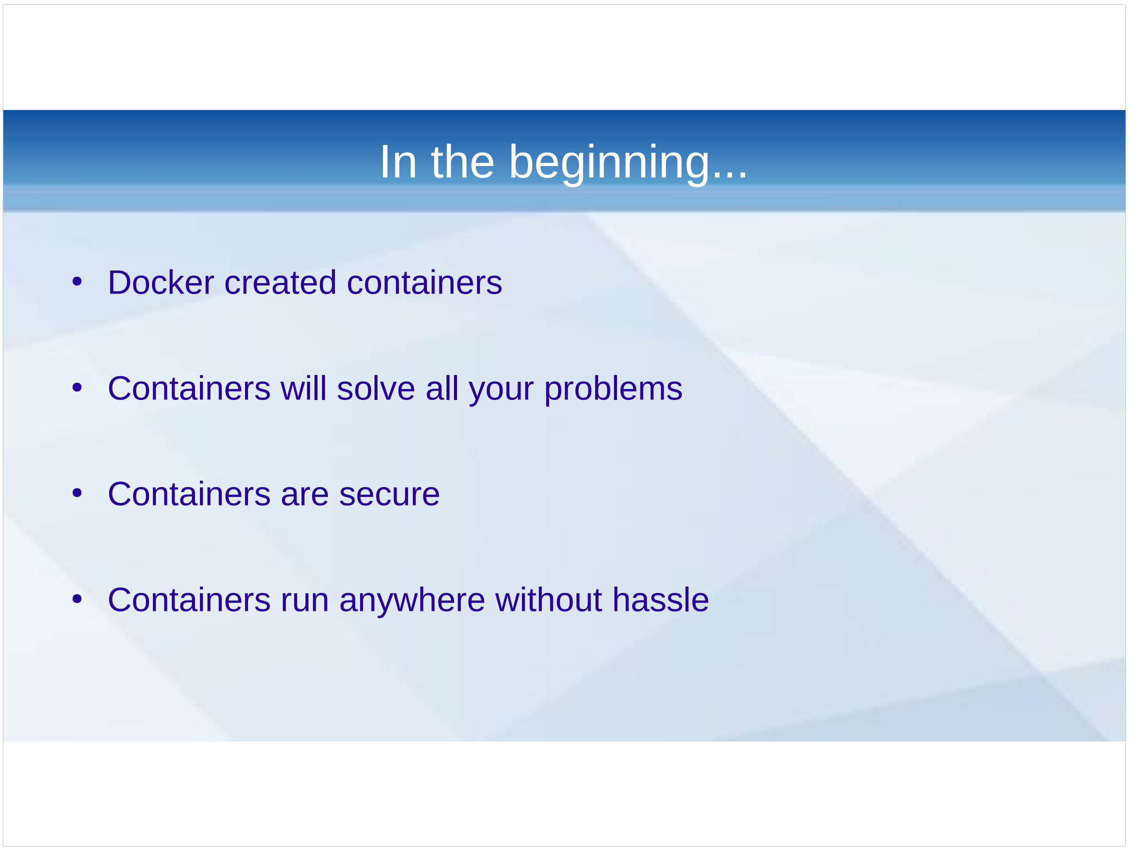
Why “proud father” first? Look up the definition of “proud” :P



Regarding Kumina

- Creation and implementation of custom made IT infrastructures
- Remote maintenance with 24x7 monitoring and alerting
- Remote support with unlimited support hours
- Consultancy regarding infrastructure requirements

We're specialising on Kubernetes and are a registered Kubernetes Service Partner.

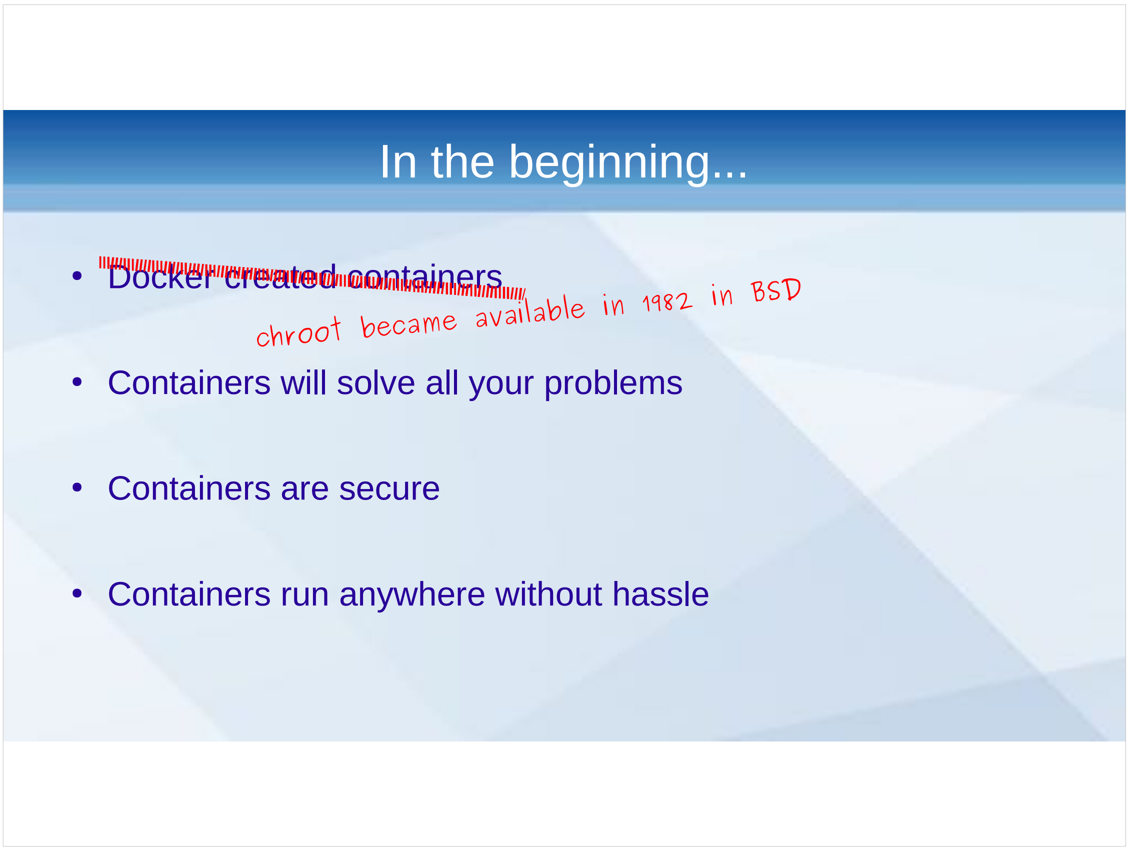


In the beginning...

- Docker created containers
- Containers will solve all your problems
- Containers are secure
- Containers run anywhere without hassle

You know all this?

It's all wrong.



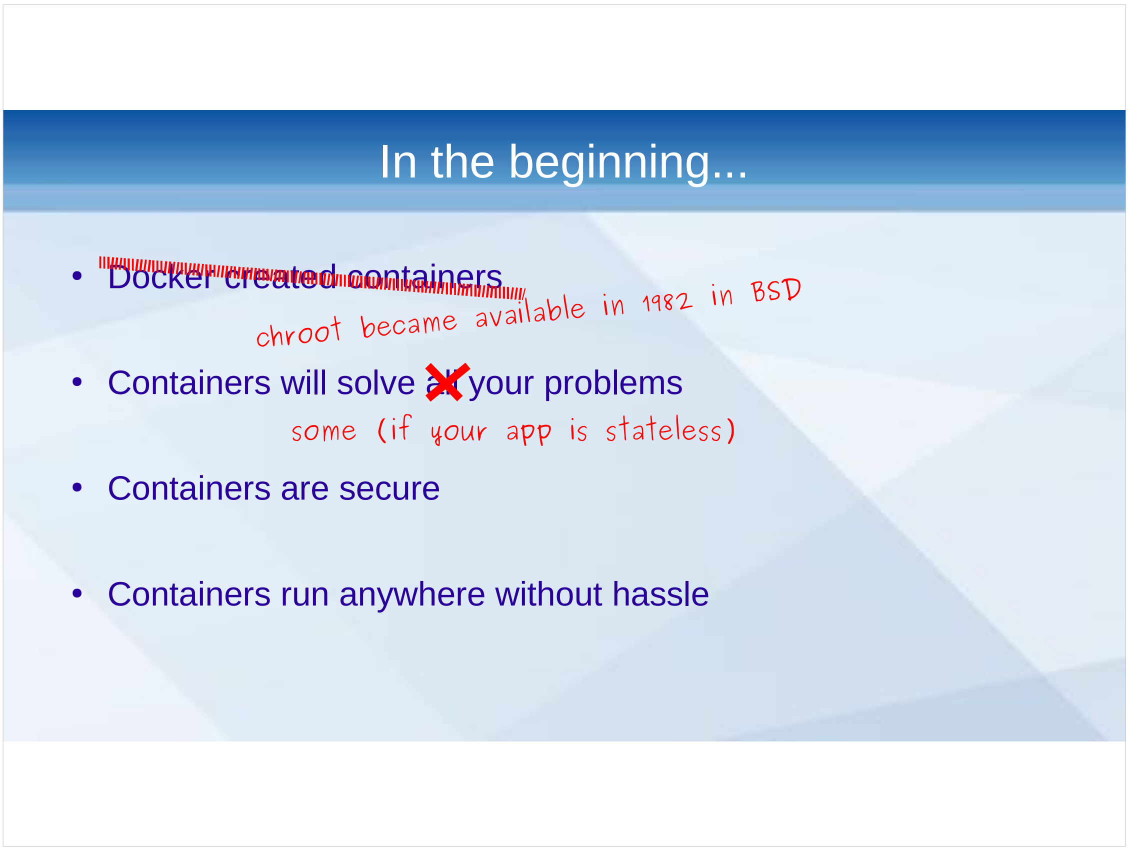
In the beginning...

- ~~Docker created containers~~

  chroot became available in 1982 in BSD

- Containers will solve all your problems
- Containers are secure
- Containers run anywhere without hassle

It depends on how you define a container, but if it's simply “not seeing other processes”, chroot already did a fairly good job.



In the beginning...

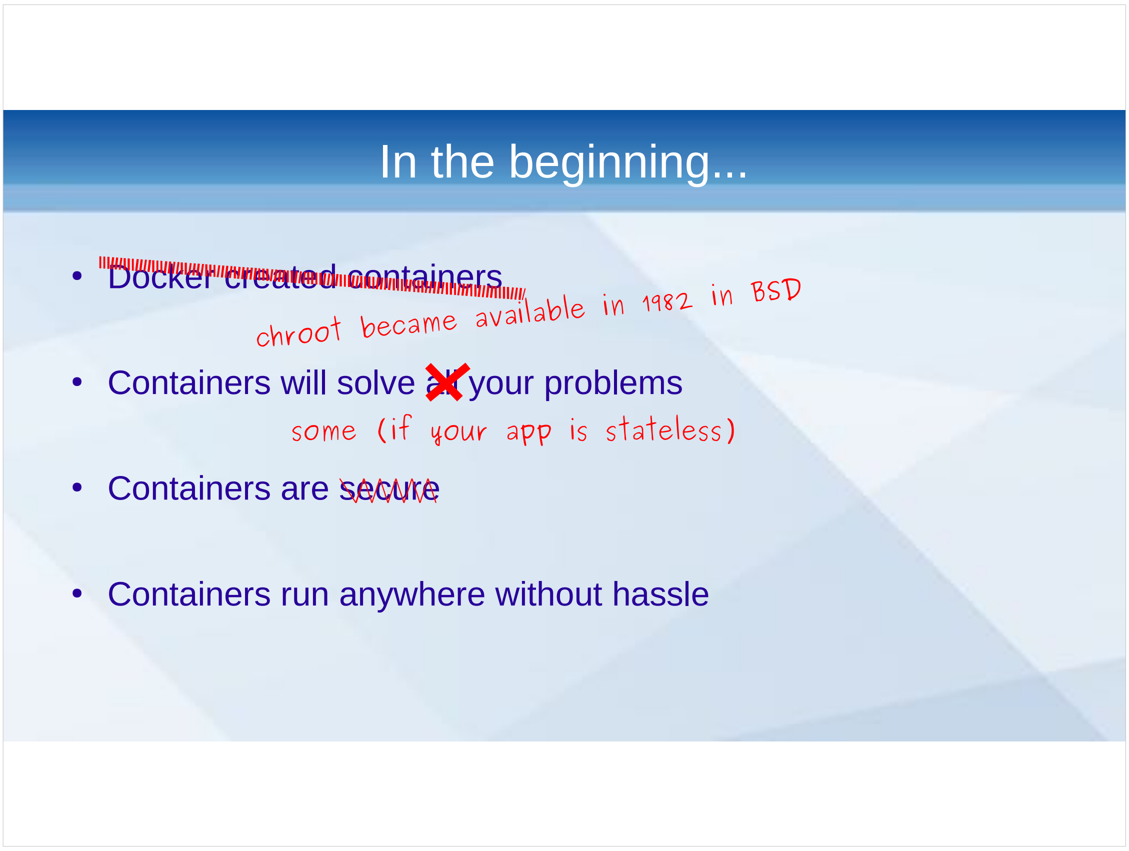
- ~~Docker created containers~~

  chroot became available in 1982 in BSD
- Containers will solve ~~all~~ your problems

  some (if your app is stateless)
- Containers are secure
- Containers run anywhere without hassle

Containers are not for everyone. They do add a certain level of complexity (if you do it correctly) and sometimes it just does not make sense to spend time on that.

Generally speaking, anything PHP should not run in a container.



In the beginning...

- ~~Docker created containers~~

  chroot became available in 1982 in BSD

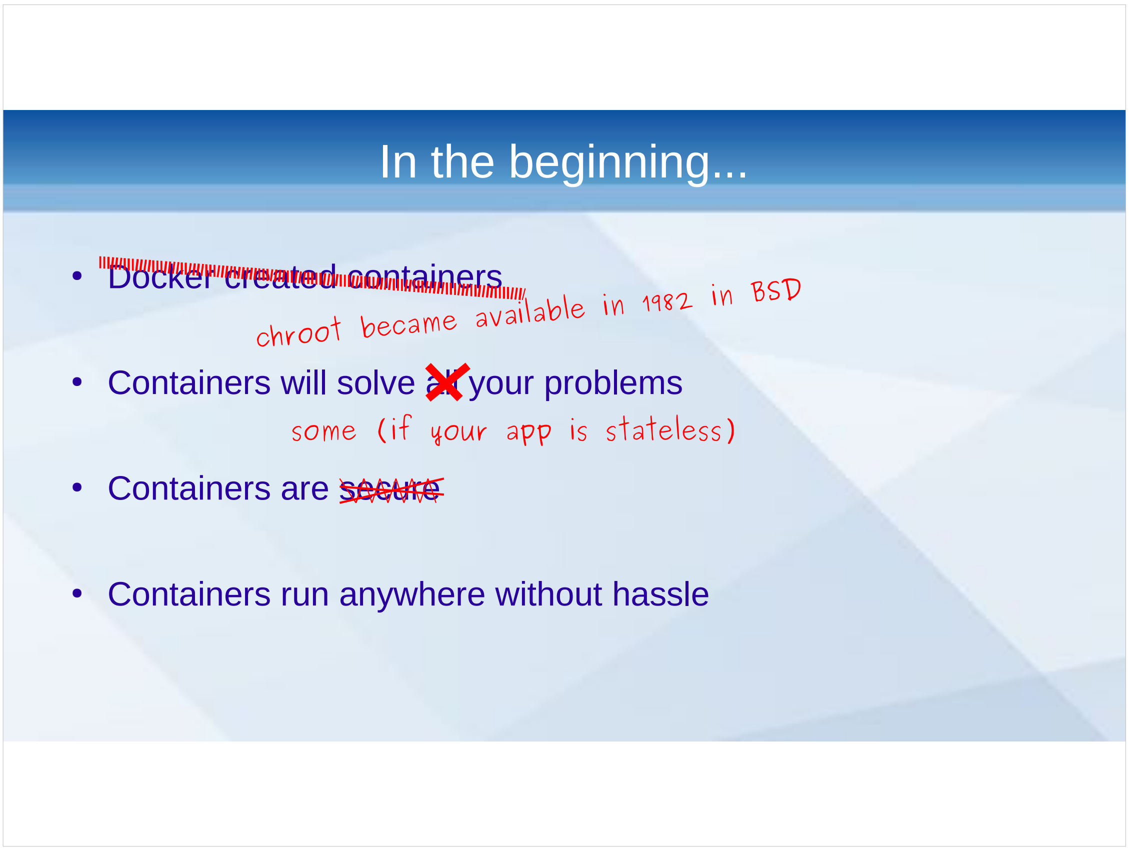
- Containers will solve ~~all~~ your problems

  some (if your app is stateless)

- Containers are ~~secure~~

- Containers run anywhere without hassle

No.



In the beginning...

- ~~Docker created containers~~

  chroot became available in 1982 in BSD

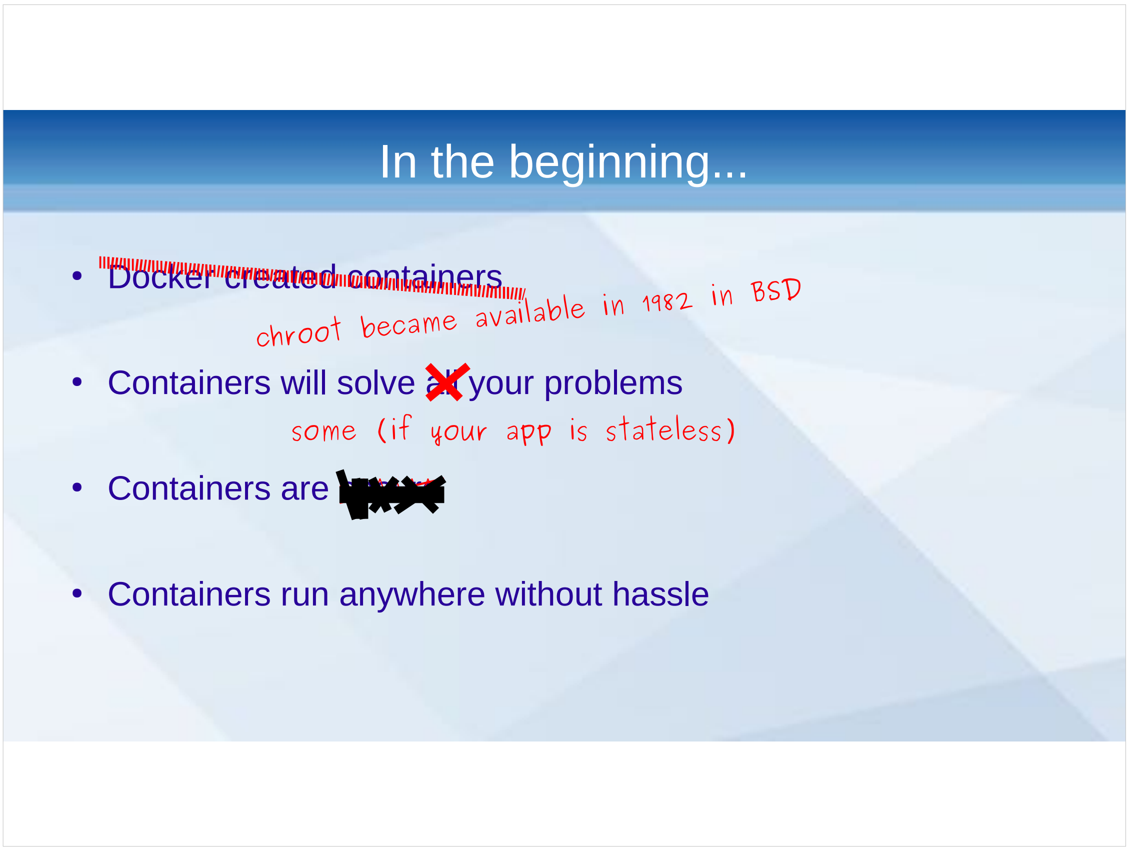
- Containers will solve ~~all~~ your problems

  some (if your app is stateless)

- Containers are ~~secure~~

- Containers run anywhere without hassle

No!



In the beginning...

- ~~Docker created containers~~

  chroot became available in 1982 in BSD

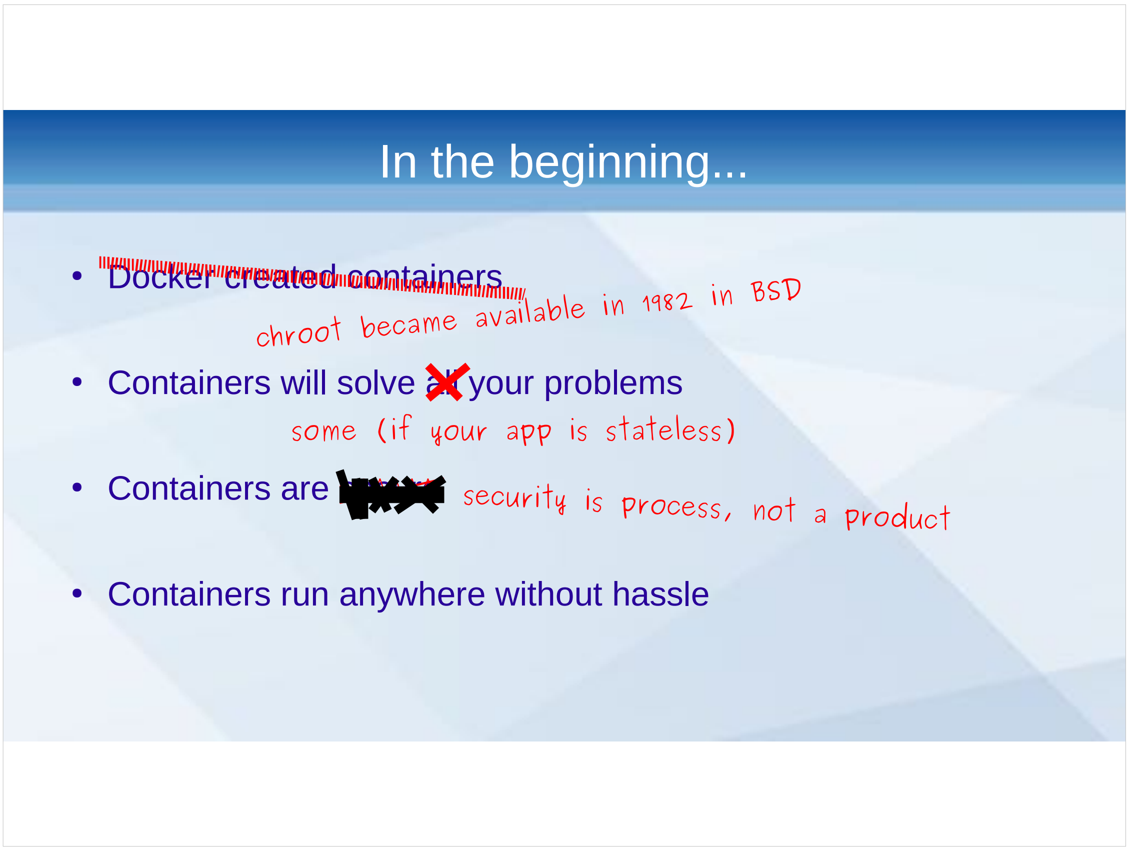
- Containers will solve ~~all~~ your problems

  some (if your app is stateless)

- Containers are [redacted]

- Containers run anywhere without hassle

NO!



In the beginning...

- ~~Docker created containers~~

  chroot became available in 1982 in BSD

- Containers will solve ~~all~~ your problems

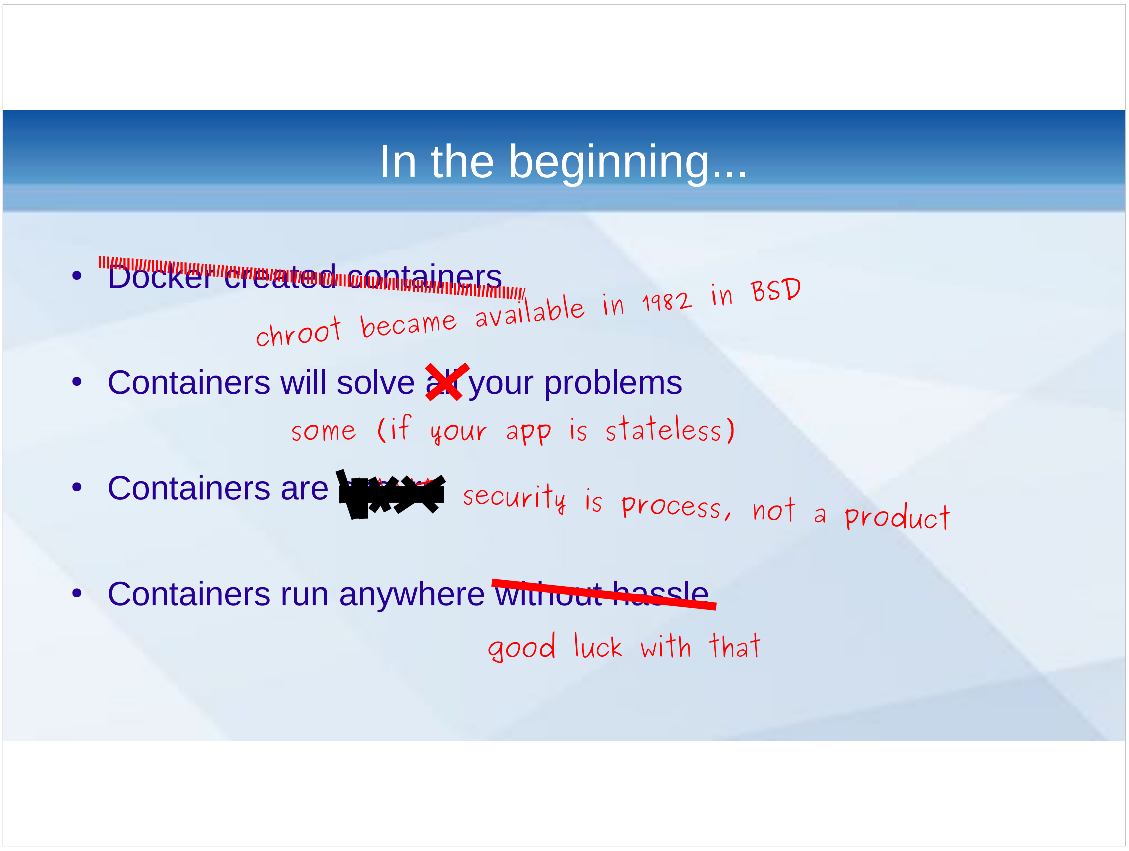
  some (if your app is stateless)

- Containers are [redacted] security is process, not a product

- Containers run anywhere without hassle

You get some added features that can help improve generic security, but if your code is vulnerable, it's vulnerable. Broken code can still wipe the database. Broken code can still send spam. Broken code can still crawl the web in search for other broken code.

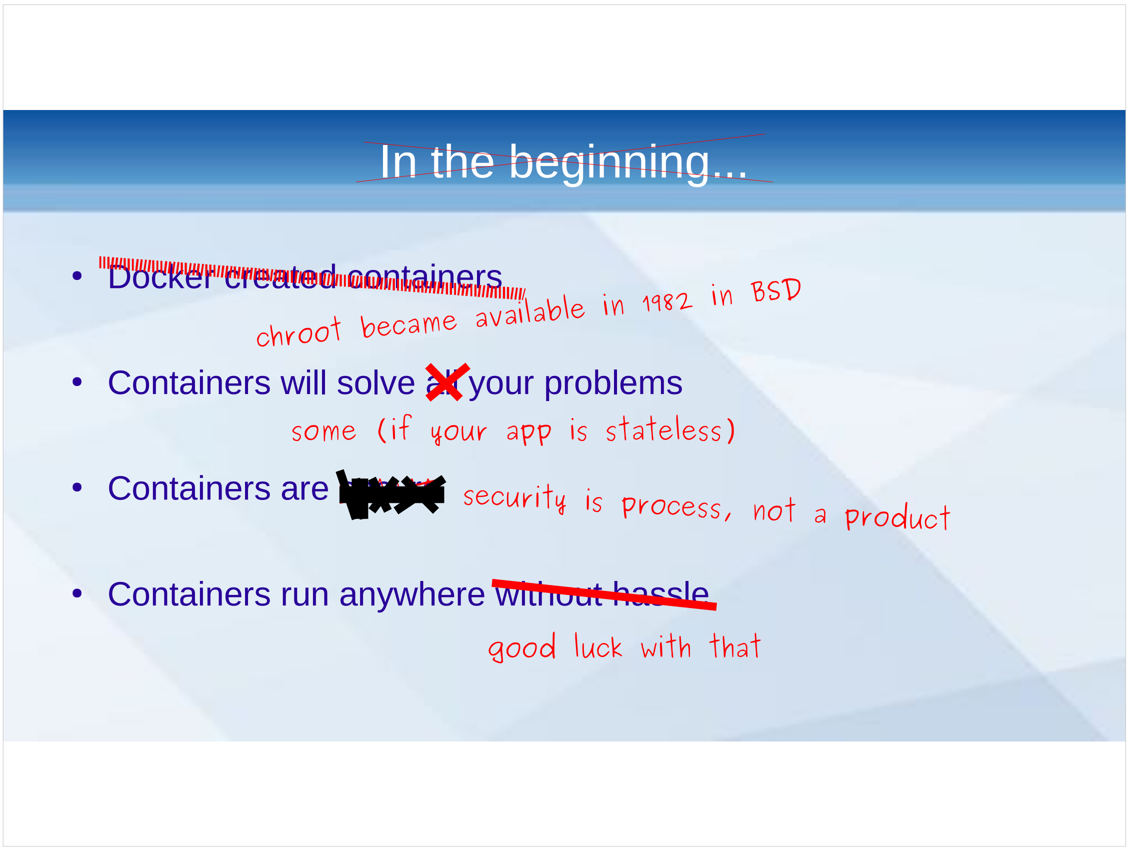
Low level entry points get the most press, but that's because they are the most interesting to look at. Most security problems are just bad code.



In the beginning...

- ~~Docker created containers~~
  chroot became available in 1982 in BSD
- Containers will solve ~~all~~ your problems
  some (if your app is stateless)
- Containers are [redacted] security is process, not a product
- Containers run anywhere ~~without hassle~~
  good luck with that

You still need a compatible system. Think CPU (Raspberry PI anyone?) and the like.



~~In the beginning...~~

- ~~Docker created containers~~

  chroot became available in 1982 in BSD

- Containers will solve ~~all~~ your problems

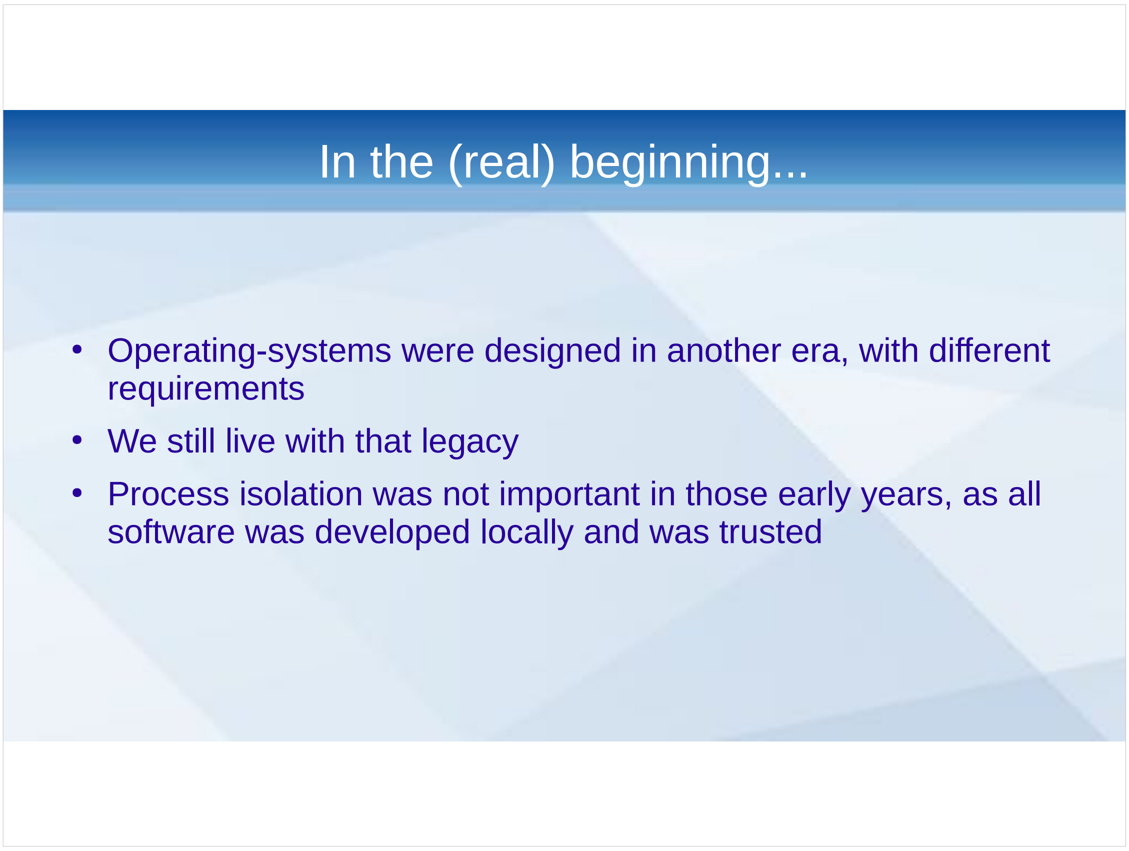
  some (if your app is stateless)

- Containers are [redacted] security is process, not a product

- Containers run anywhere ~~without hassle~~

  good luck with that

Not the beginning.

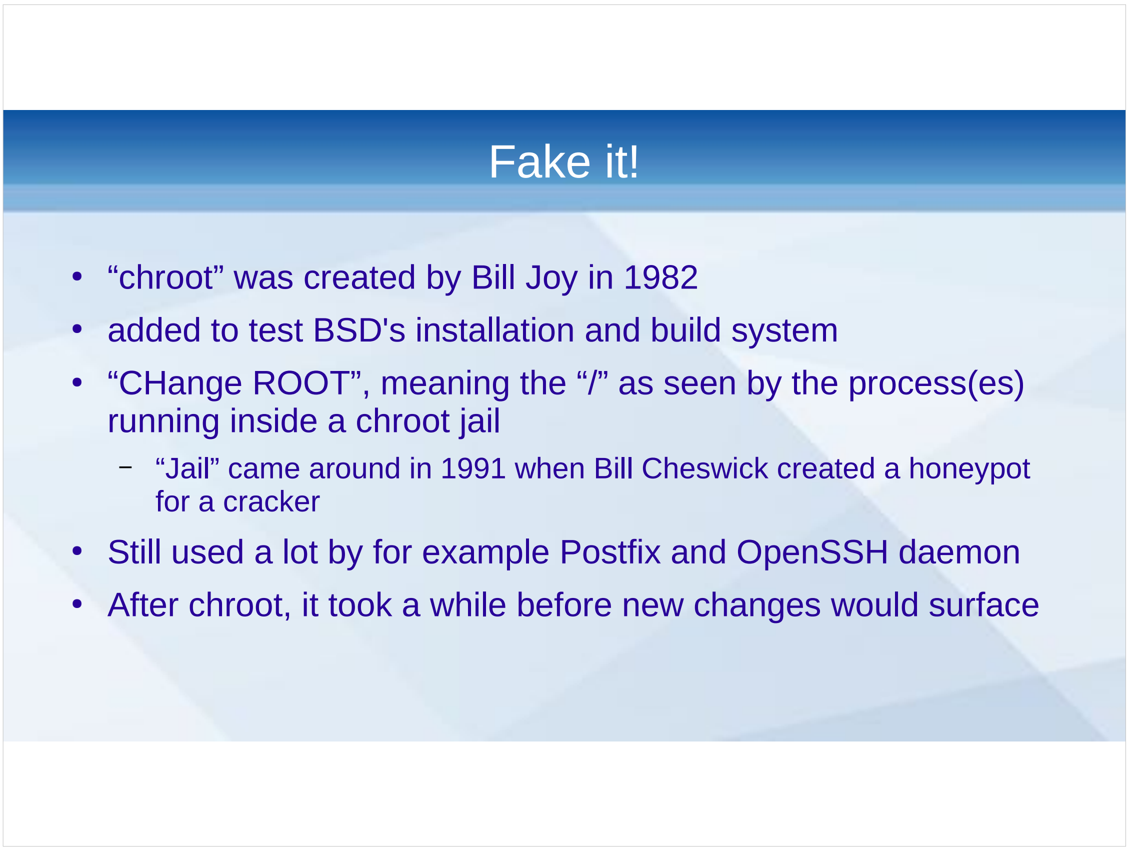


In the (real) beginning...

- Operating-systems were designed in another era, with different requirements
- We still live with that legacy
- Process isolation was not important in those early years, as all software was developed locally and was trusted

These days, you bought a computer and you got an entire manually regarding how to program it (in assembly!). Languages like C and Pascal were just emerging, standard operating systems were just emerging. Computers were still special-purpose devices that required an electrical engineer to program and troubleshoot.

Process isolation is a broad term, think namespacing and the like.



Fake it!

- “chroot” was created by Bill Joy in 1982
- added to test BSD's installation and build system
- “CHange ROOT”, meaning the “/” as seen by the process(es) running inside a chroot jail
  - “Jail” came around in 1991 when Bill Cheswick created a honeypot for a cracker
- Still used a lot by for example Postfix and OpenSSH daemon
- After chroot, it took a while before new changes would surface

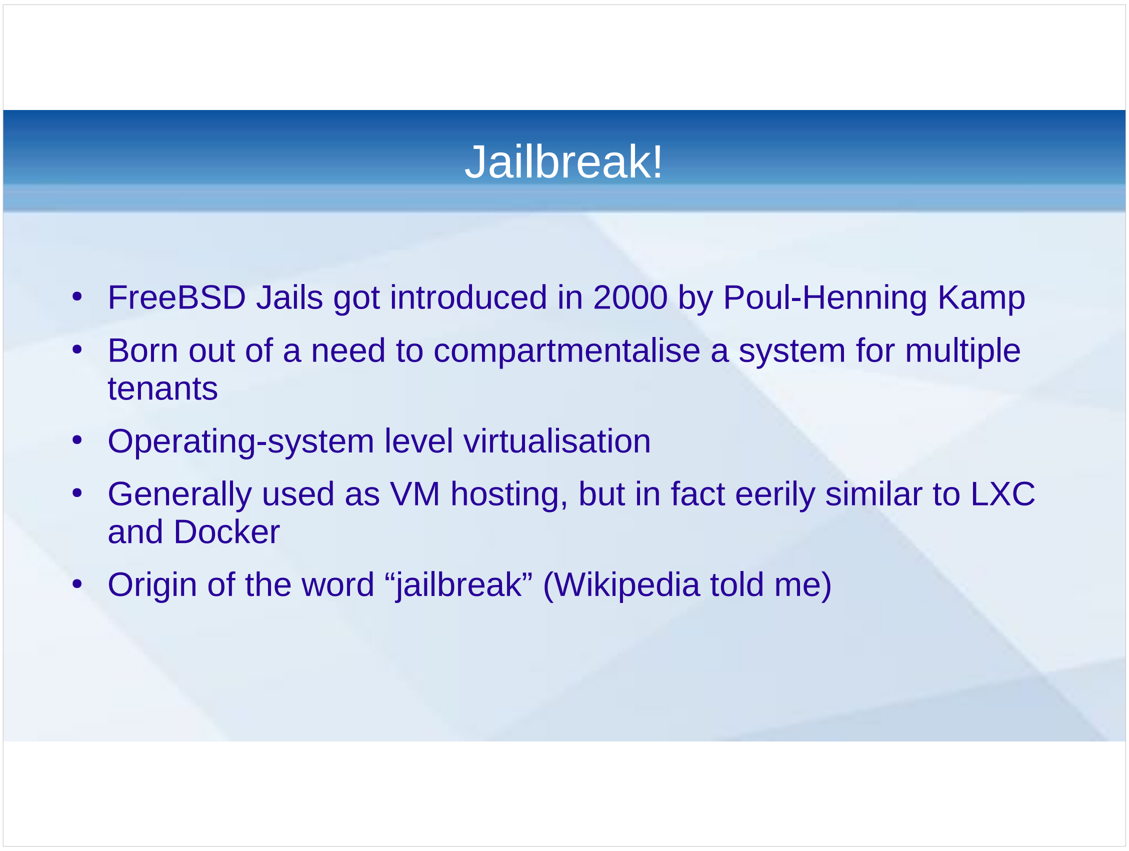
“chroot” got added to BSD on 18 March 1982 (Wikipedia tells me). Not security feature, just path limitation.

Bill Joy was a student of Berkeley, got contracted into Sun Microsystem as a co-founder (6 months after company creation). He worked on it during his time as a graduate student (a student working at University after graduation), in a time when he also wrote the ex and vi editors.

Keep in mind that this (1982) was the time that DARPA began to expand Arpanet and AT&T Unix would be transformed at Berkeley to the Berkeley Software Distribution (BSD). If you were into software, you were hacking on kernels and file systems. Your interface was a blinking whitish cursor on a black background. Editing was done with line editors (until Joy created vi!).

Bill Cheswick created firewalls (together with Steven Bellovin) in 1987 at Bell Labs.

Developed to protext applications from noisy, nosey and messy neighbours.



Jailbreak!

- FreeBSD Jails got introduced in 2000 by Poul-Henning Kamp
- Born out of a need to compartmentalise a system for multiple tenants
- Operating-system level virtualisation
- Generally used as VM hosting, but in fact eerily similar to LXC and Docker
- Origin of the word “jailbreak” (Wikipedia told me)

Poul-Henning Kamp, of Varnish notoriety. Software-craftman, did and does a lot for the FreeBSD OS. This was paid for by Derrick T. Woolworth, as he wanted to use it for his hosting company.

Stable and secure environment for process isolation. In general used as a VM hosting solution (so it has an entire OS and even runs an init, both of which can be done on Docker as well, but is discouraged).

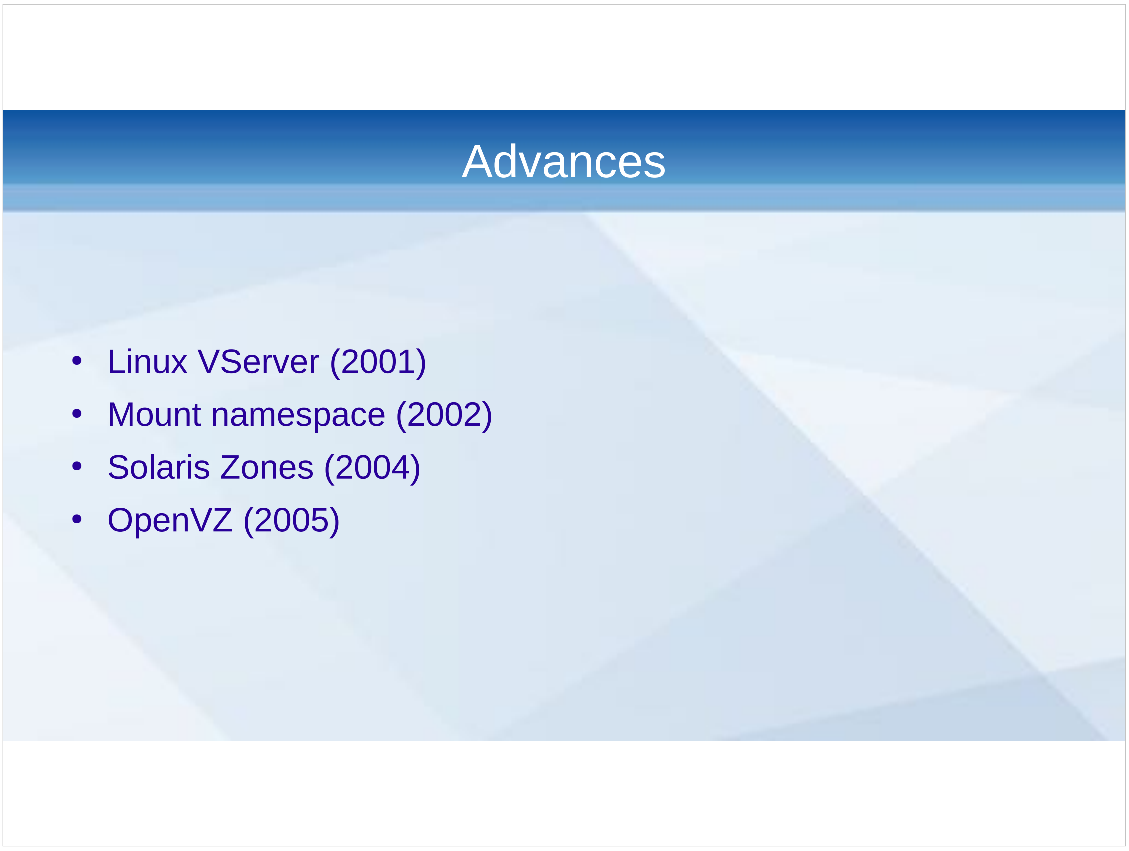
Used a lot in days of yore by shared hosting providers (keep in mind that this was way before the advent of cloud in 2006).

Very mature system, very flexible when using ZFS as the backing store. Features are still being added.

Does not support live migration (but neither does Docker).

We're done here. Feature-wise they compare. Jails just don't run on Linux. Linux needed some more iterations before it could do this.

(Also, Docker is about portable images, but FreeBSD has ZFS...)



Advances

- Linux VServer (2001)
- Mount namespace (2002)
- Solaris Zones (2004)
- OpenVZ (2005)

VServer is not related to Linux Virtual Server (which is loadbalancer). Shares host kernel and can share filesystem as well. Processes within VServer are processes on host (slightly better performance, cache handling)

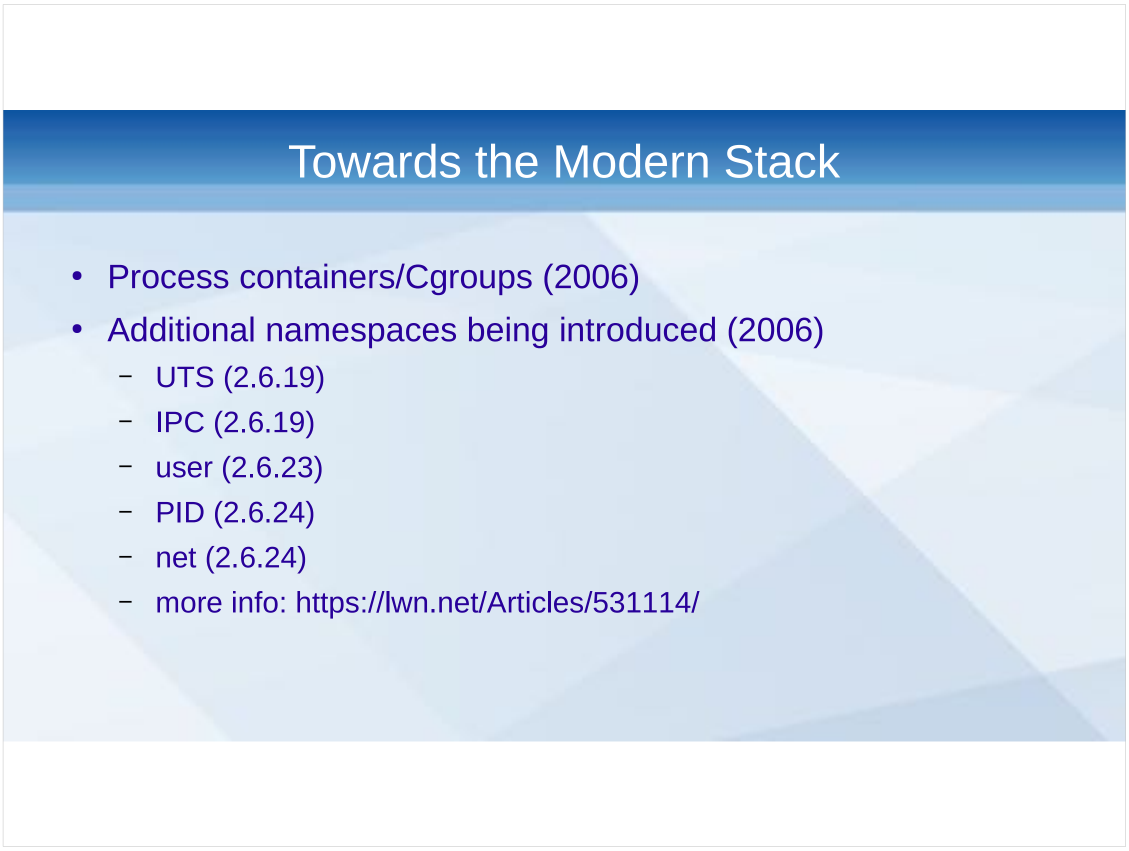
“Light” container (network isolation, not virtualisation, so cannot have routing and the like), has less control over disk I/O resource allocation and misses parts of /proc and /sys.

VServer has last stable release in 2006 (but is still being developed, it seems)

Mount namespace added, precursor of things to come

Solaris is dead, but pionereed a lot of innovations.

OpenVZ by Parallels is a nice solution, but shares kernel.



Towards the Modern Stack

- Process containers/Cgroups (2006)
- Additional namespaces being introduced (2006)
  - UTS (2.6.19)
  - IPC (2.6.19)
  - user (2.6.23)
  - PID (2.6.24)
  - net (2.6.24)
  - more info: https://lwn.net/Articles/531114/

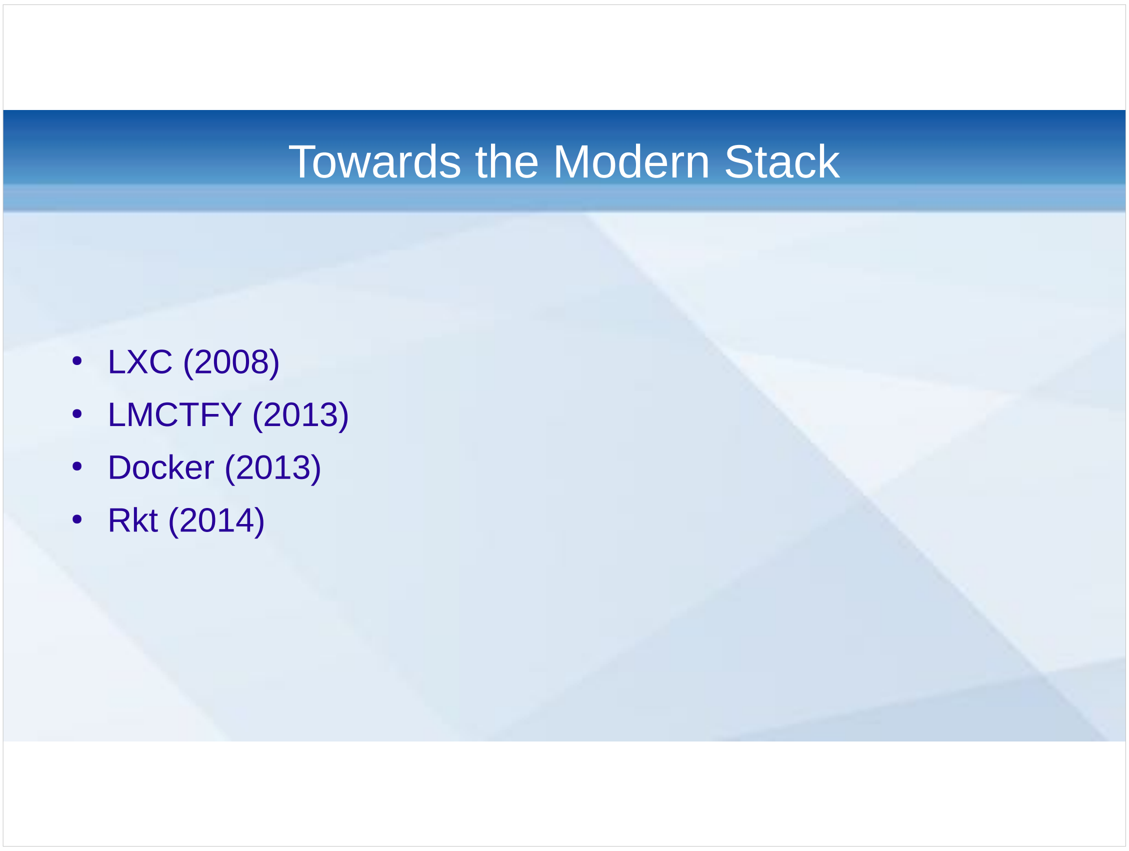
Process containers developed by Google (Paul Menage and Rohit Seth) to allow grouping of processes within the kernel for resource management

Merged as cgroups in 2007/2008 (kernel 2.6.24), features were added over the years. Eventually used for

- Resource limiting (memory)
- Prioritization (cpu and disk i/o)
- Accounting (resource usage)
- Control (freezing processes)

Namespaces were added for better isolation, uts = unix timesharing system

Finished in 3.8, which is required for Docker et al.



Towards the Modern Stack

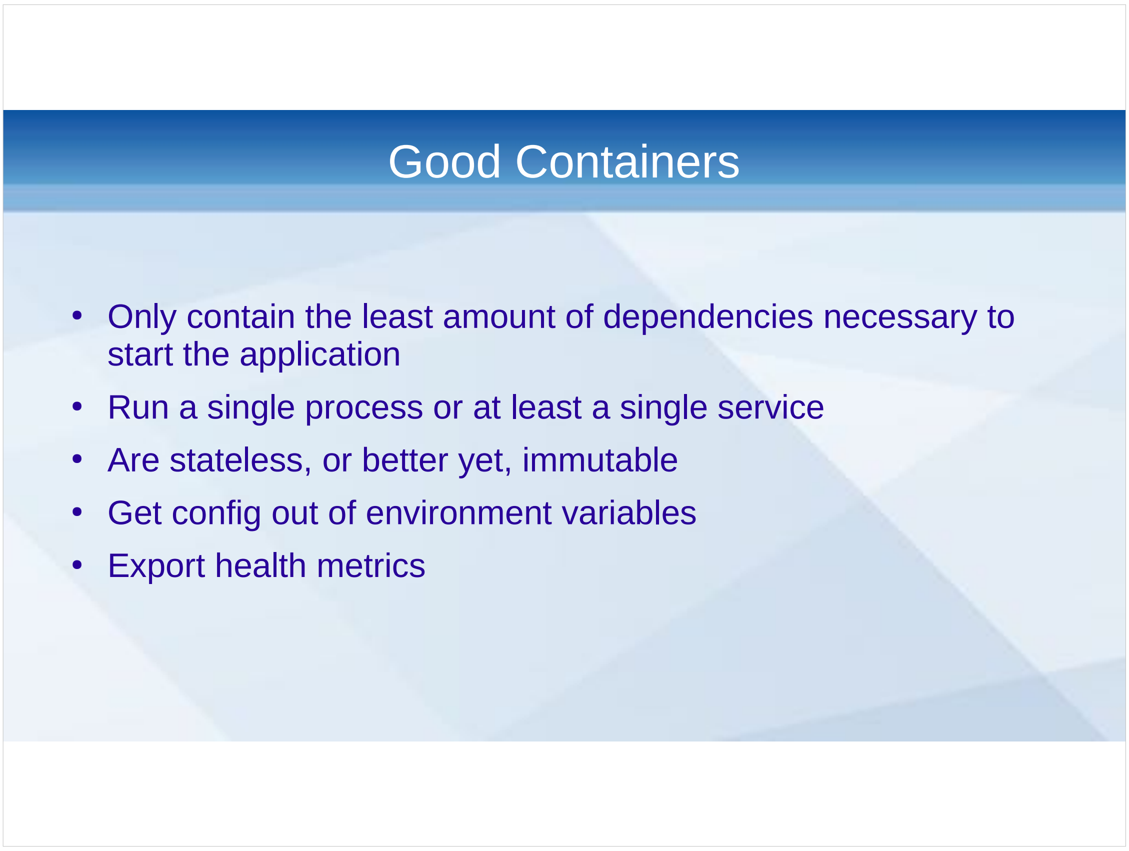
- LXC (2008)
- LMCTFY (2013)
- Docker (2013)
- Rkt (2014)

LinuX Containers were developed as a replacement for chroot, allowing jails-like functionality using namespaces and cgroups. Security was not too good at the beginning (root was really root), got better with unprivileged mode.

Let Me Contain That For You is an API based on user intent. Low level (you set specific cgroups yourself). Has some kernel patches. Merging into libcontainer

Docker was started by Solomon Hykes at dotCloud (FR). Main advantage is (imho) container format.

Rkt by CoreOS wants to standardize. Appc to describe a container's needs, different image format (Application Container Image ACI), both under control of Open Container Initiative



Good Containers

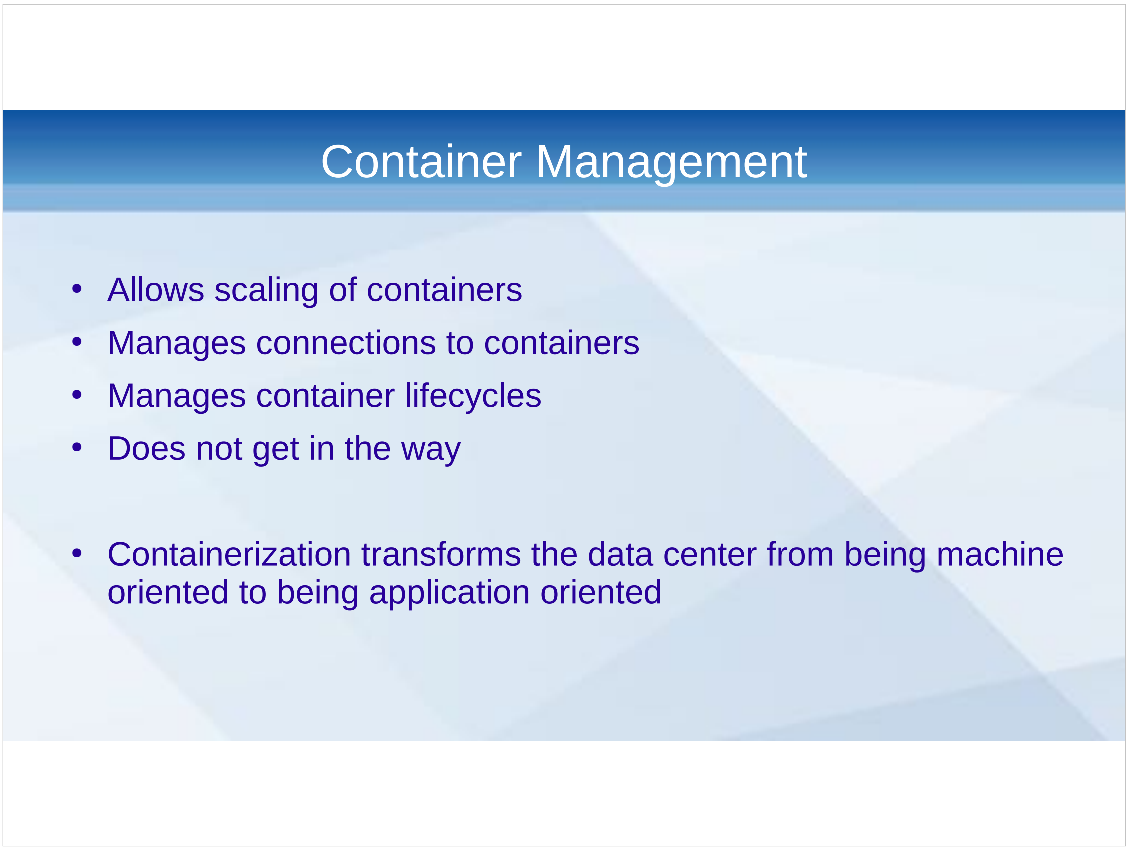
- Only contain the least amount of dependencies necessary to start the application
- Run a single process or at least a single service
- Are stateless, or better yet, immutable
- Get config out of environment variables
- Export health metrics

Not an entire OS, no Ubuntu installation, preferably just a single binary (Go is awesome for that)

No systemd, no supervisord, no init. Just the application.

Containers are cattle, they come and go (as they please?). Local state makes the container a pet, losing it loses data. Immutable forces you to be stateless

Environment variables are generally the easiest way to pass local config. Service discovery works as well, of course (etcd, Consul, etc.)



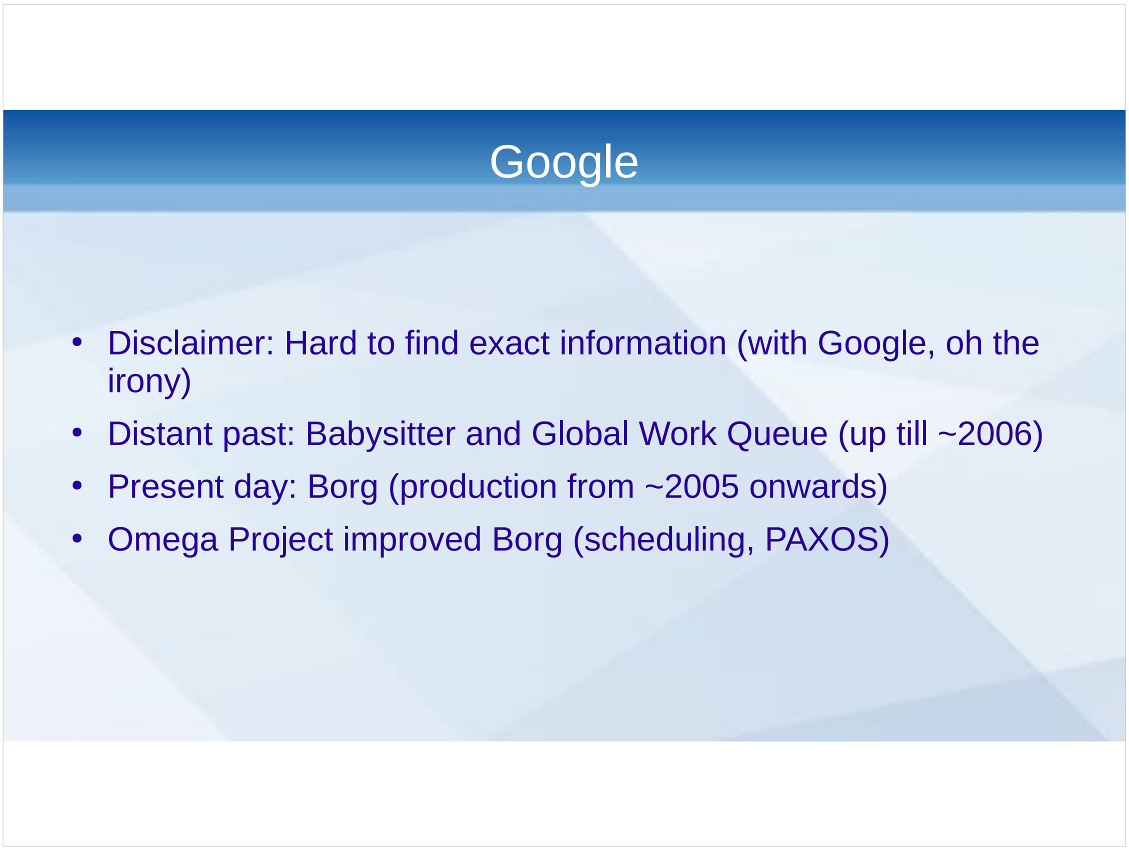
Container Management

- Allows scaling of containers
- Manages connections to containers
- Manages container lifecycles
- Does not get in the way

- Containerization transforms the data center from being machine oriented to being application oriented

Management API around containers rather than machines shifts the “primary key” of the data center from machine to application. Benefits: no longer worry about machine specifics, allow rollout of new hardware easily, telemetry is tied to application

Self-healing is key building block



Google

- Disclaimer: Hard to find exact information (with Google, oh the irony)
- Distant past: Babysitter and Global Work Queue (up till ~2006)
- Present day: Borg (production from ~2005 onwards)
- Omega Project improved Borg (scheduling, PAXOS)

Google infra is pretty proprietary and considered trade secrets, so a lot is “heard from” or guessed.

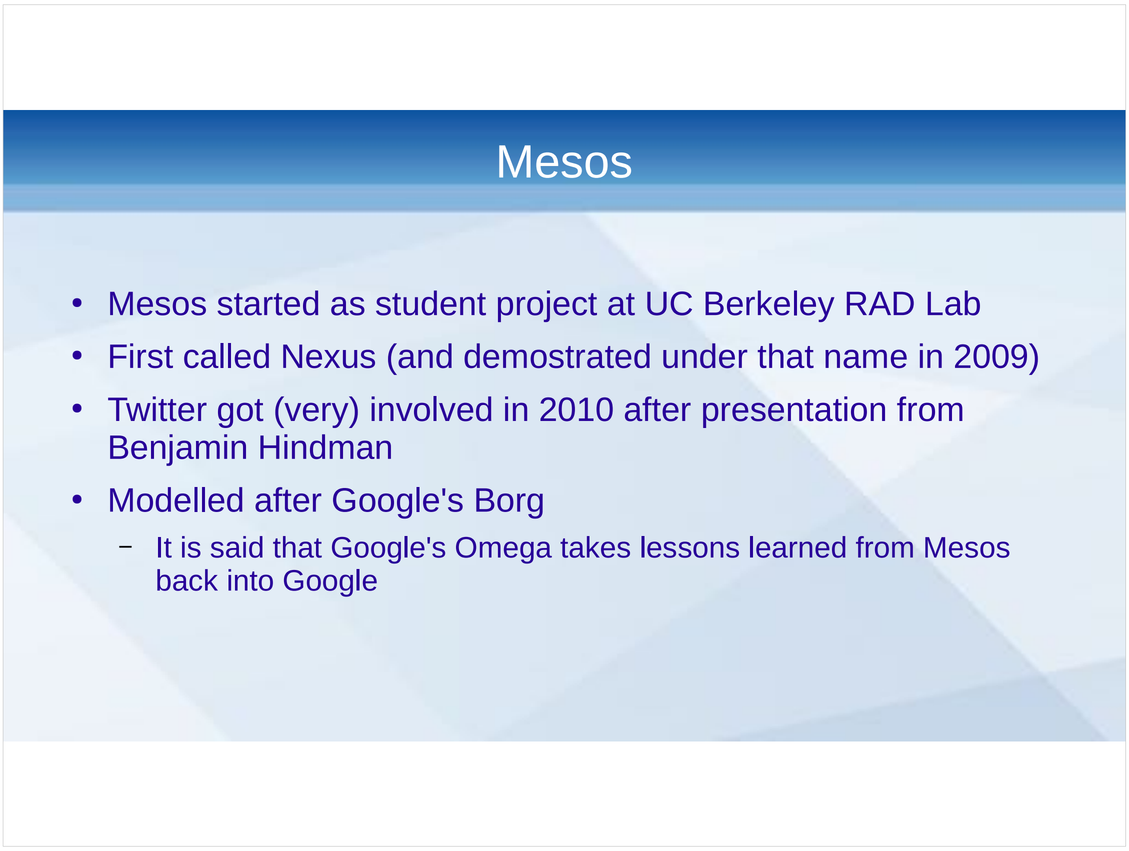
GWQ used all machines in the Google network to assign work to, mostly non-time-sensitive large computations (batch jobs). Babysitter kept jobs running.

Google's cale is insane and one-of-a-kind, still a lot we can learn from it. Google has high expectations, always on.

Borg was probably developed around 2003 and brought into production around 2005, to replace a lot of the custom management functions that were around then.

Projects switched to Borg one by one (“resistance is futile, you will be assimilated”)

Developed by Dougie Howser MD (j/k)



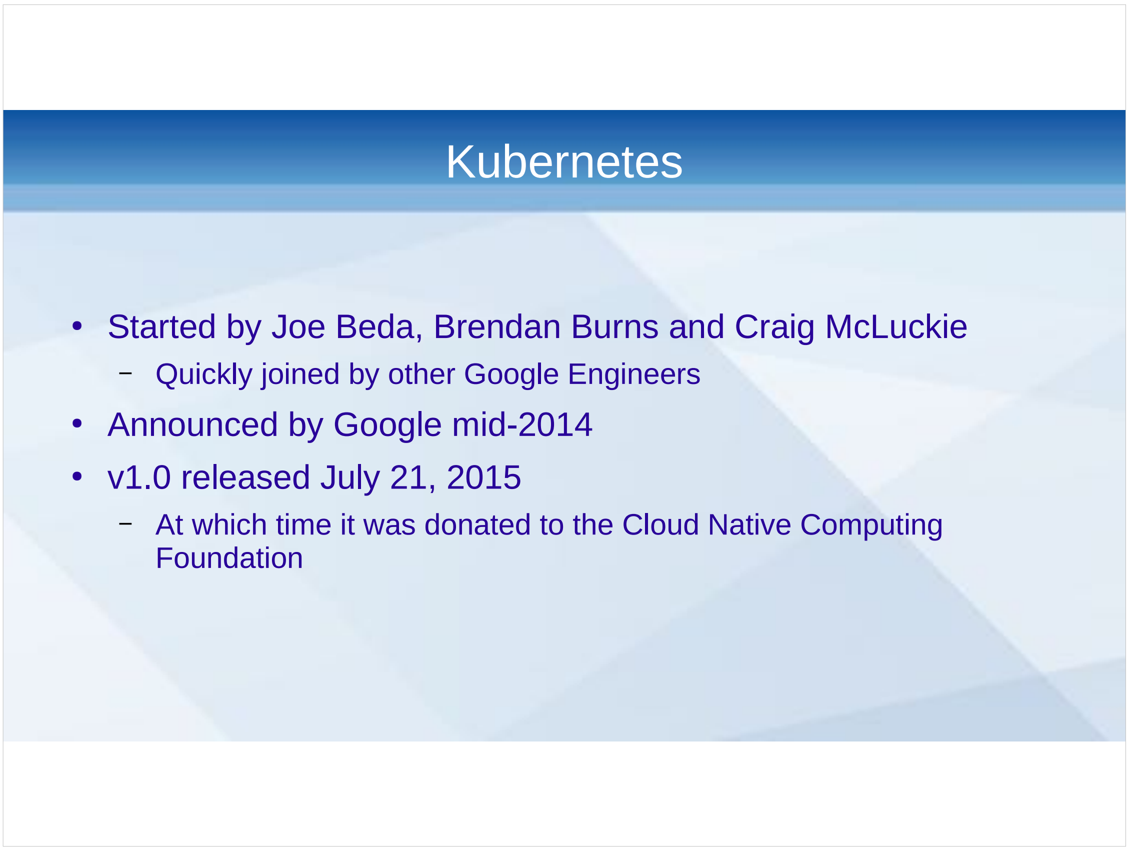
Mesos

- Mesos started as student project at UC Berkeley RAD Lab
- First called Nexus (and demostrated under that name in 2009)
- Twitter got (very) involved in 2010 after presentation from Benjamin Hindman
- Modelled after Google's Borg
  - It is said that Google's Omega takes lessons learned from Mesos back into Google

Source: https://www.wired.com/2013/03/google-borg-twitter-mesos/all/

Project started by Benjamin Hindman (who went to Twitter), Andy Konwinski and Matei Zaharia (Databricks) and professor Ion Stoica (Databricks)

As Mesos is modeled on Borg, it's very well suited for large computational jobs. Job support in K8s is fairly under-highlighted. Hadoop on Mesos for specific questions is probably a lot better.



Kubernetes

- Started by Joe Beda, Brendan Burns and Craig McLuckie
  - Quickly joined by other Google Engineers
- Announced by Google mid-2014
- v1.0 released July 21, 2015
  - At which time it was donated to the Cloud Native Computing Foundation

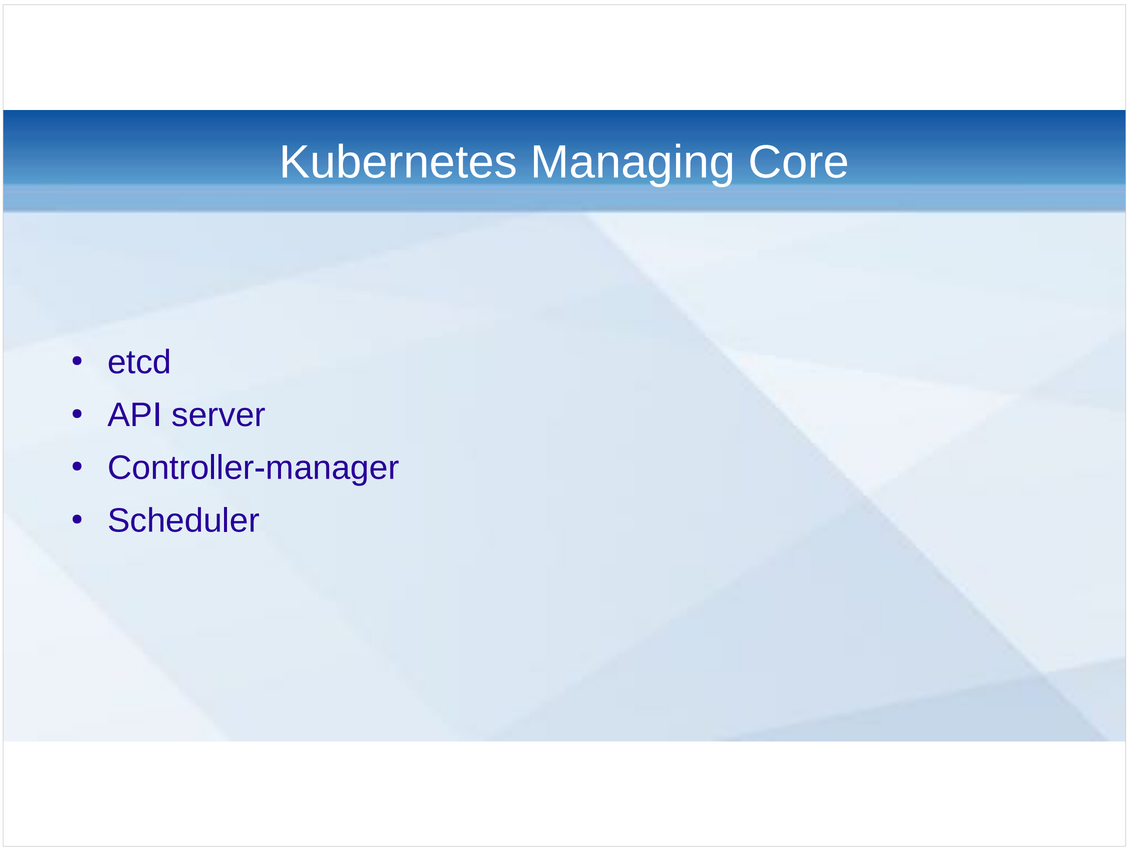
First called seven-of-nine (“friendlier Borg”) but lawyers intervened.

CNCF was formed by Google and the Linux Foundation

RedHat uses it for OpenShift

Developed in Go, to make sure no Borg/Omega code would leak to the project

Interesting: http://queue.acm.org/detail.cfm?id=2898444



Kubernetes Managing Core

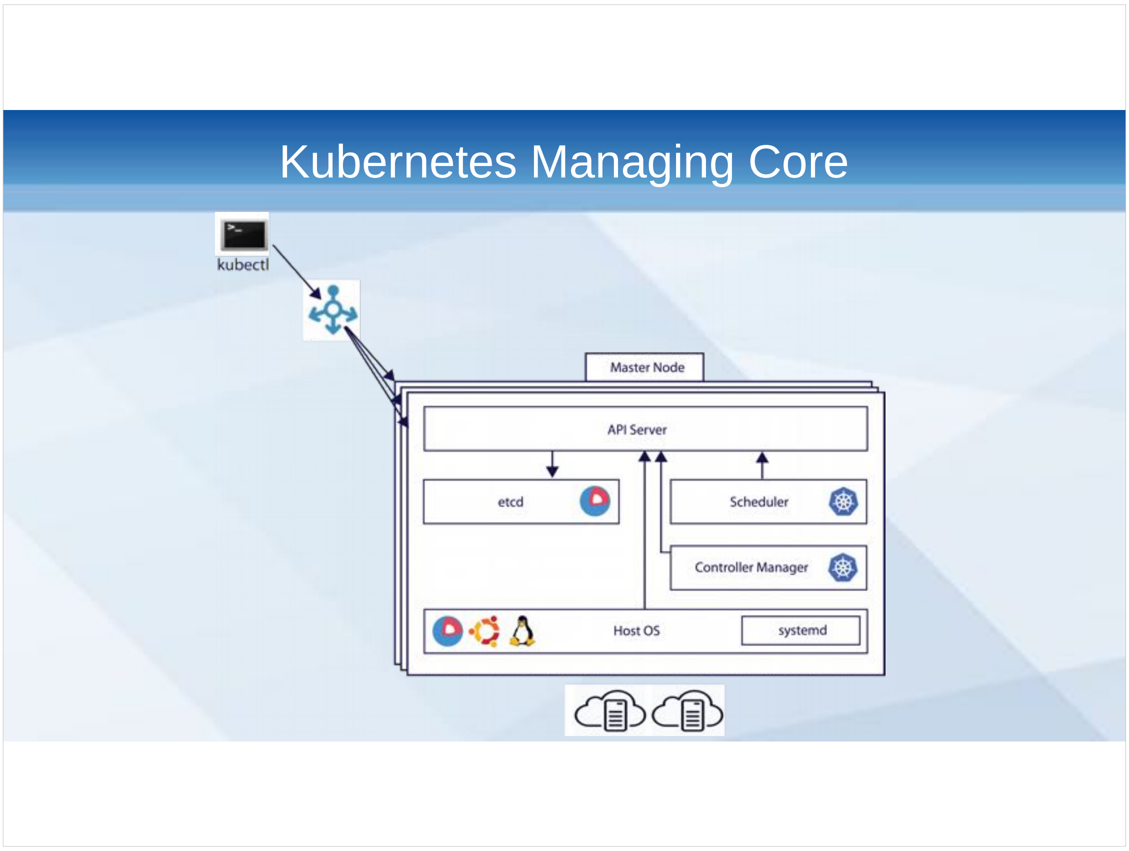
- etcd
- API server
- Controller-manager
- Scheduler

Etcd used as the PAXOS layer on which to share data

API server is entrypoint, for kubelets and kube-proxy as well as kubectl. Update objects in etcd. Designed to scale horizontally.

Controller-manager manages all Controllers (DaemonSets, Replication Controller, Node Controller, Endpoints Controller, Token Controllers), talks with API server directly. Lots of separate Controllers, compiled into one binary (Go Go!)

Scheduler places pods and tracks resource utilisation. Talks to API server.



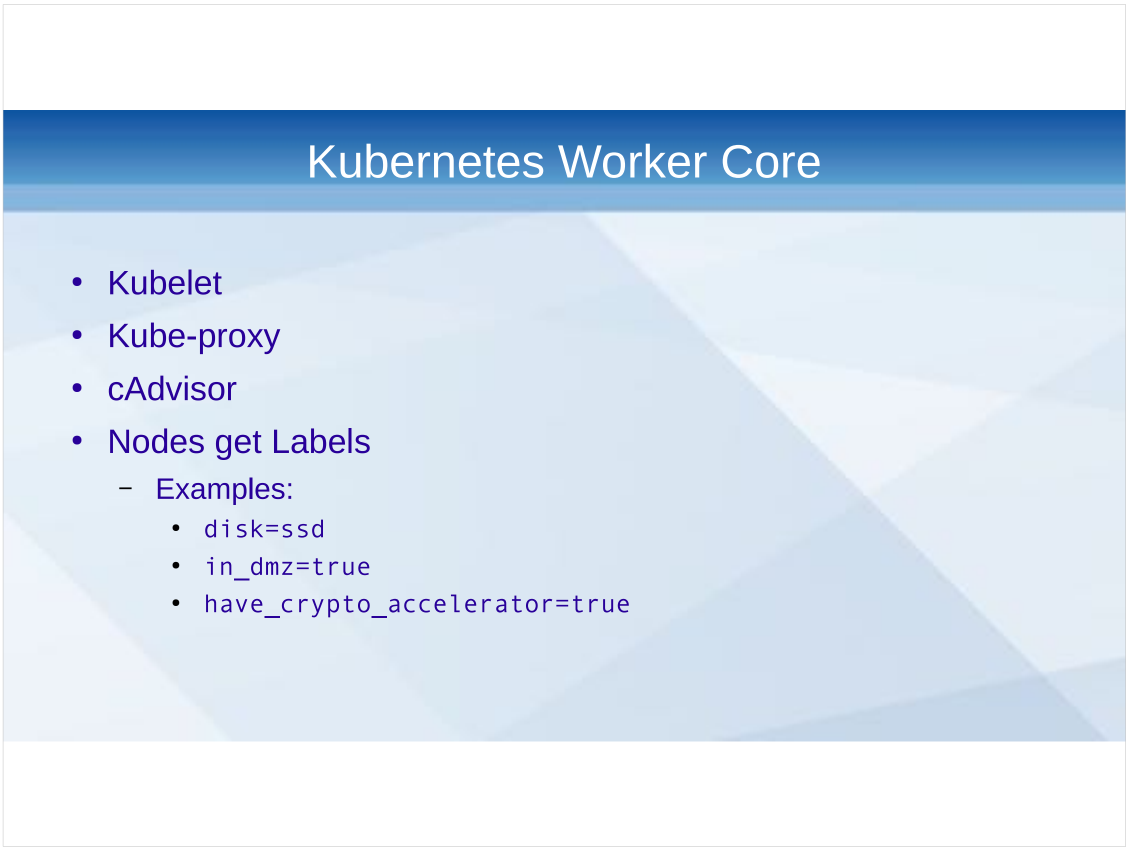
Kubernetes Managing Core

Etcd used as the PAXOS layer on which to share data

API server is entrypoint, for kubelets and kube-proxy as well as kubectl. Update objects in etcd. Designed to scale horizontally.

Controller-manager manages all Controllers (DaemonSets, Replication Controller, Node Controller, Endpoints Controller, Token Controllers), talks with API server directly. Lots of separate Controllers, compiled into one binary (Go Go!)

Scheduler places pods and tracks resource utilisation. Talks to API server.



Kubernetes Worker Core

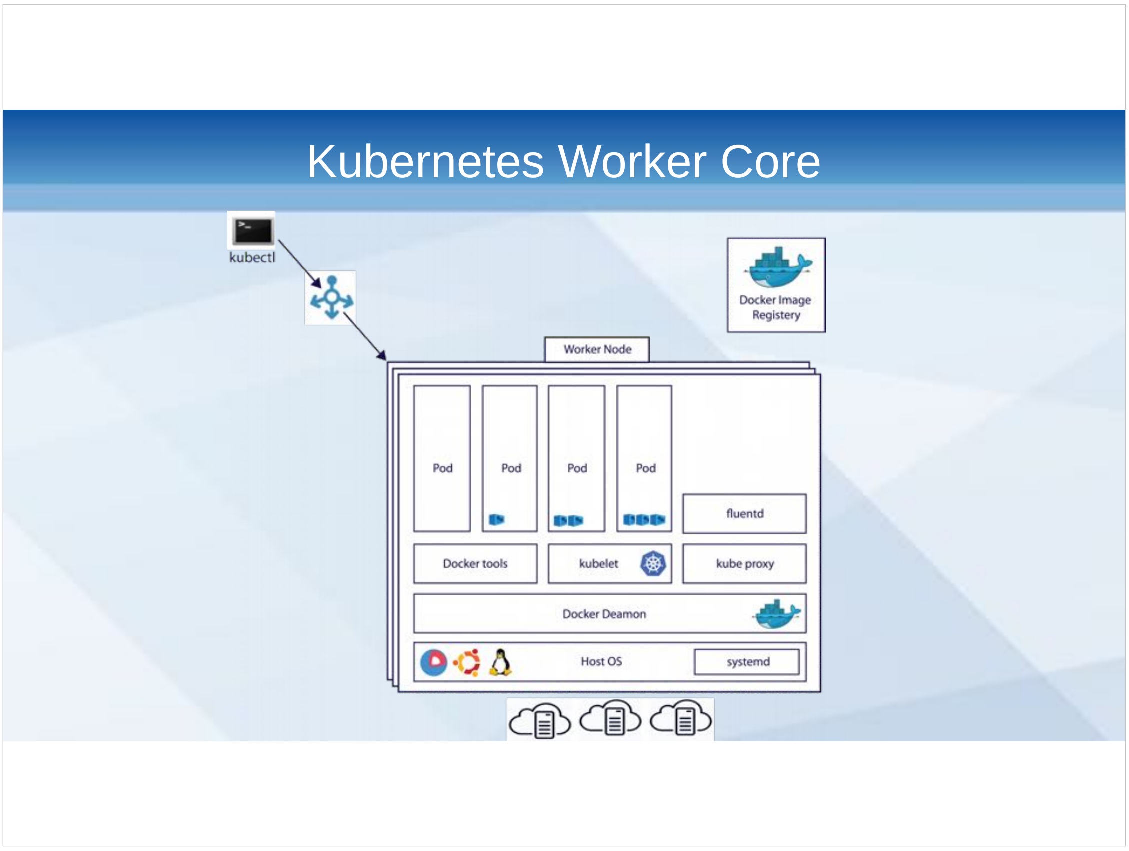
- Kubelet
- Kube-proxy
- cAdvisor
- Nodes get Labels
  - Examples:
    - `disk=ssd`
    - `in_dmz=true`
    - `have_crypto_accelerator=true`

Kubelet is primary node agent, downloads and starts pods scheduled for this node, mounts required volumes (including secrets), executes liveness probes, reports back to api server regarding node and pods

Kube-proxy provides advanced routing (wouldn't call it SDN), by mapping endpoints to specific ports on **all** workers

cAdvisor is used by Kubelet to detect resource shortages (probably replaced in the future with more real-time methods)

Addons are available, kube-dns is very much needed.



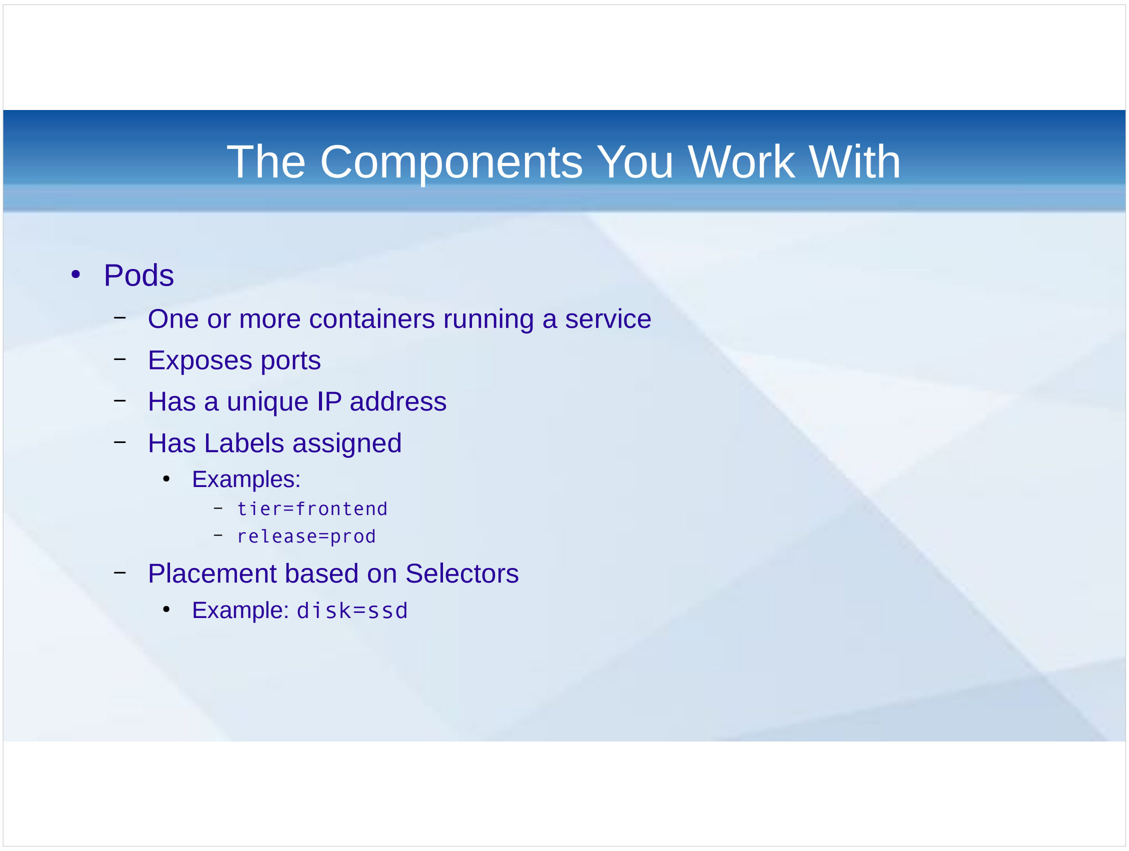
Kubernetes Worker Core

Kubelet is primary node agent, downloads and starts pods scheduled for this node, mounts required volumes (including secrets), executes liveness probes, reports back to api server regarding node and pods

Kube-proxy provides advanced routing (wouldn't call it SDN), by mapping endpoints to specific ports on **all** workers

cAdvisor is used by Kubelet to detect resource shortages (probably replaced in the future with more real-time methods)

Addons are available, kube-dns is very much needed.



The Components You Work With

- Pods
  - One or more containers running a service
  - Exposes ports
  - Has a unique IP address
  - Has Labels assigned
    - Examples:
      - `tier=frontend`
      - `release=prod`
  - Placement based on Selectors
    - Example: `disk=ssd`

Remember how containers run a single process? Multi-container Pods consist of a container for nginx, a container for fpm (php execution), a container for a nginx metrics exporter and a container for fpm metrics exporter

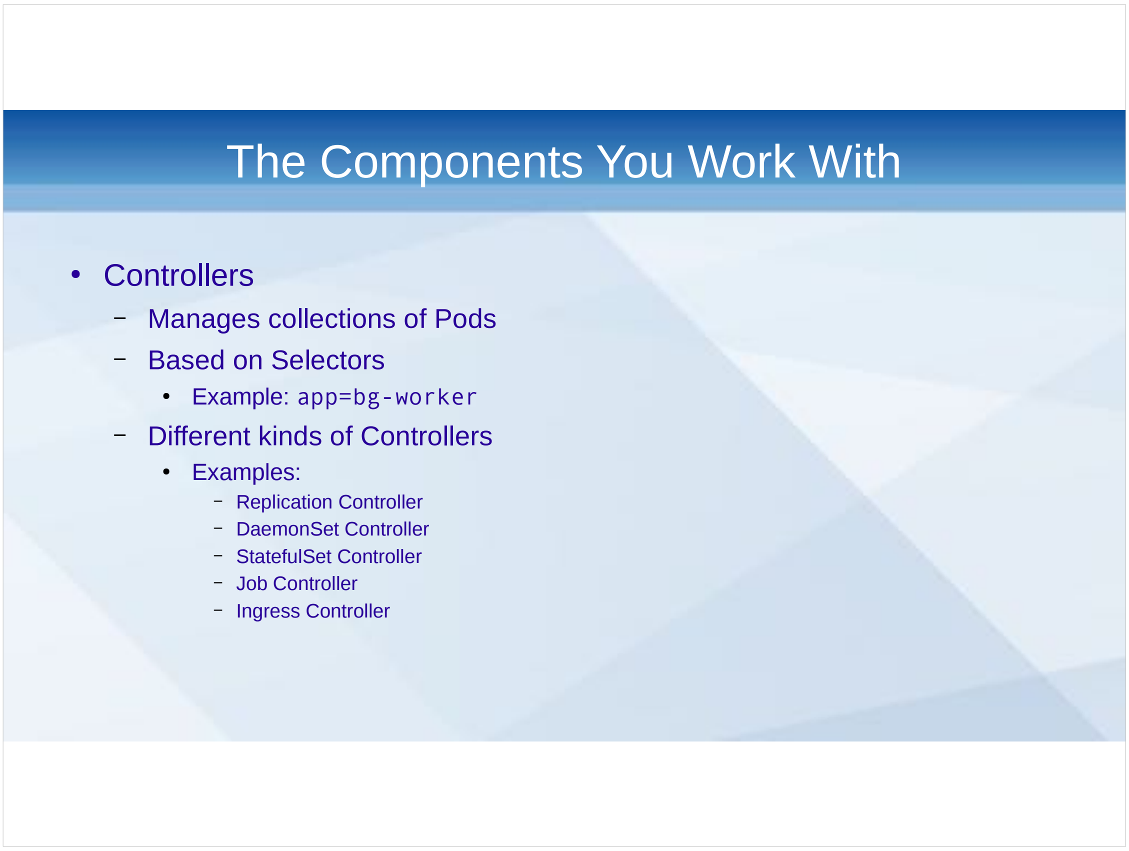
Only expose ports that are required for external communication. Internal communication is “free”

Labels are very important, they group and indentify pods.

Resources can be shared within a Pod (but can be set per container as well)

Resources limits determine Pod priority, require=limit is highest priority (Guaranteed), require<limit Burstable, no limits or only require=Best Effort

IPs are chosen from pool (either local or global), SDN can be used if needed (Calico, Flannel, etc.). IP are shared between containers within a pod (same net namespace)



The Components You Work With

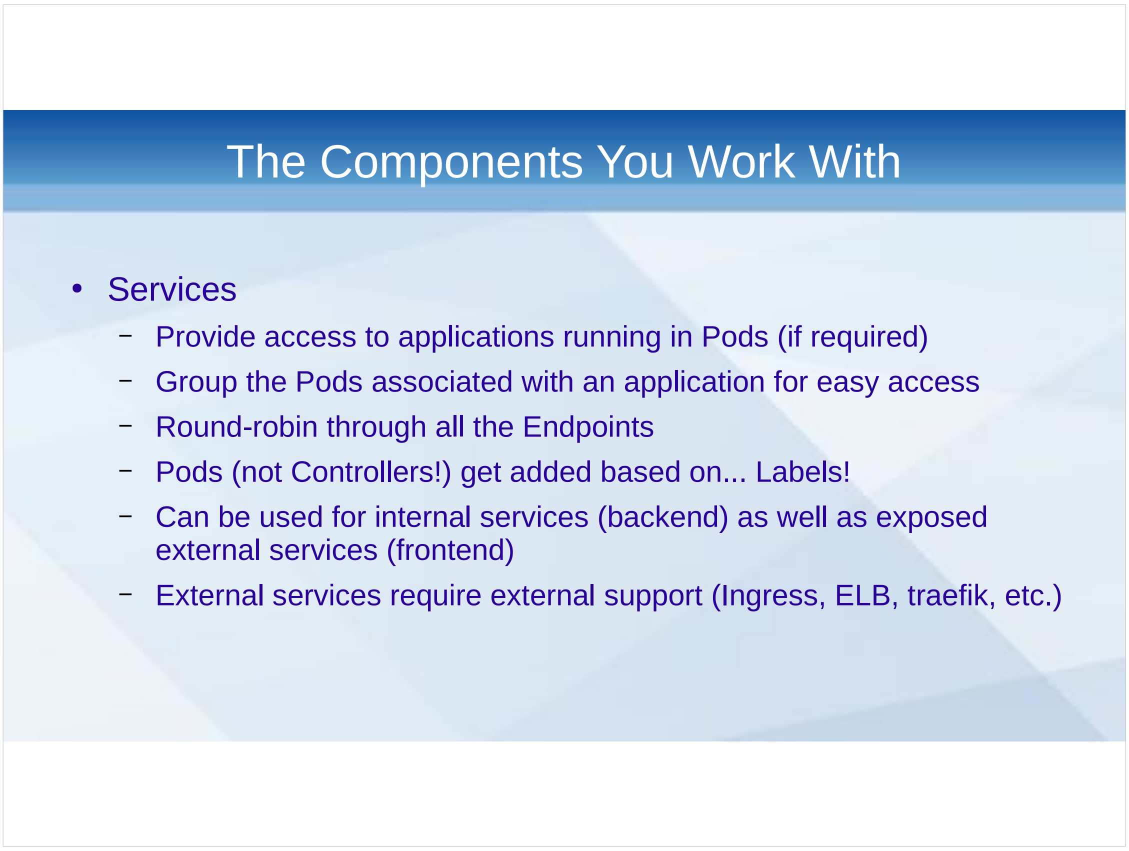
- Controllers
  - Manages collections of Pods
  - Based on Selectors
    - Example: `app=bg-worker`
  - Different kinds of Controllers
    - Examples:
      - Replication Controller
      - DaemonSet Controller
      - StatefulSet Controller
      - Job Controller
      - Ingress Controller

Controllers create Pods in API based on cluster statistics. Replication Controllers eg. make sure there are always X Pods running of a certain type.

Example daemonset: fluentd (log aggregation)

Example statefulset: elasticsearch, databases

Some Controllers you no longer use directly (Replication is managed via Deployment).

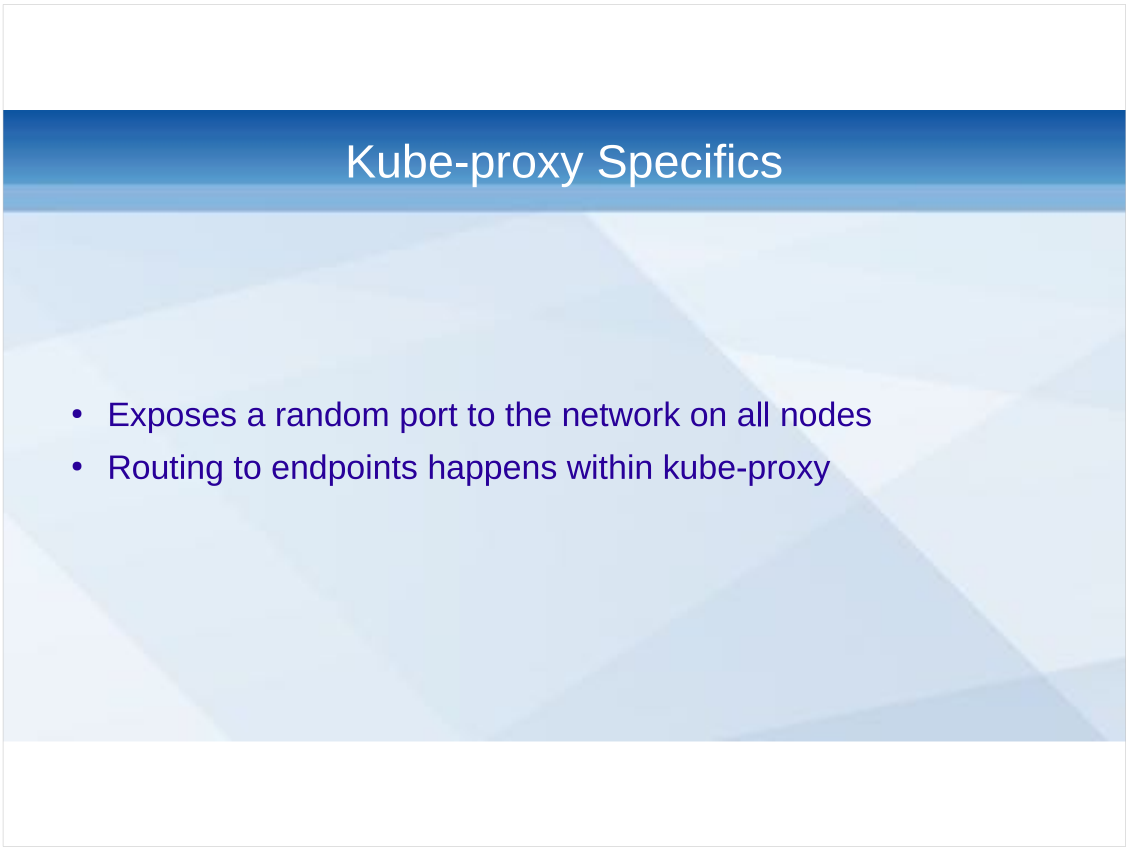


The Components You Work With

- Services
  - Provide access to applications running in Pods (if required)
  - Group the Pods associated with an application for easy access
  - Round-robin through all the Endpoints
  - Pods (not Controllers!) get added based on... Labels!
  - Can be used for internal services (backend) as well as exposed external services (frontend)
  - External services require external support (Ingress, ELB, traefik, etc.)

Services allow for 'fixed' connection strings, eg. you can always connect to 'elasticsearch:9200' within the cluster (this does require kube-dns).

Can interact with external services (like ELB) using annotations, “comments” added to a Service description that can be read via the API. Kubernetes-aware services will then be able to use these as parameters (eg. to provide an SSL certificate to the ELB via an arn resource).

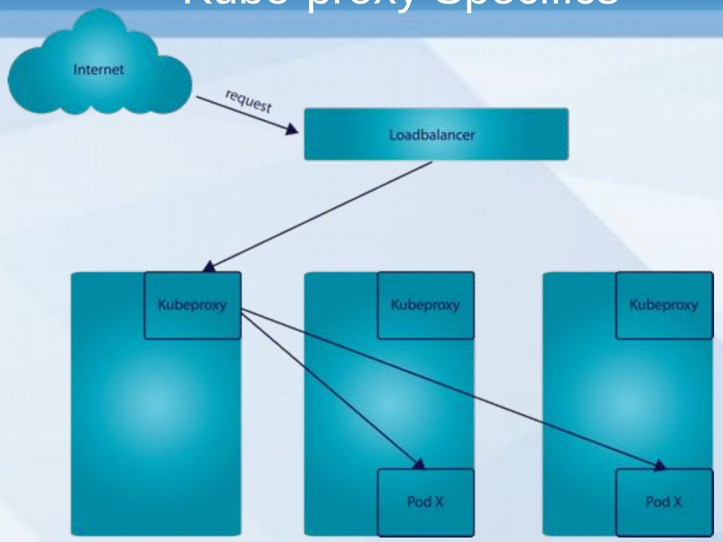


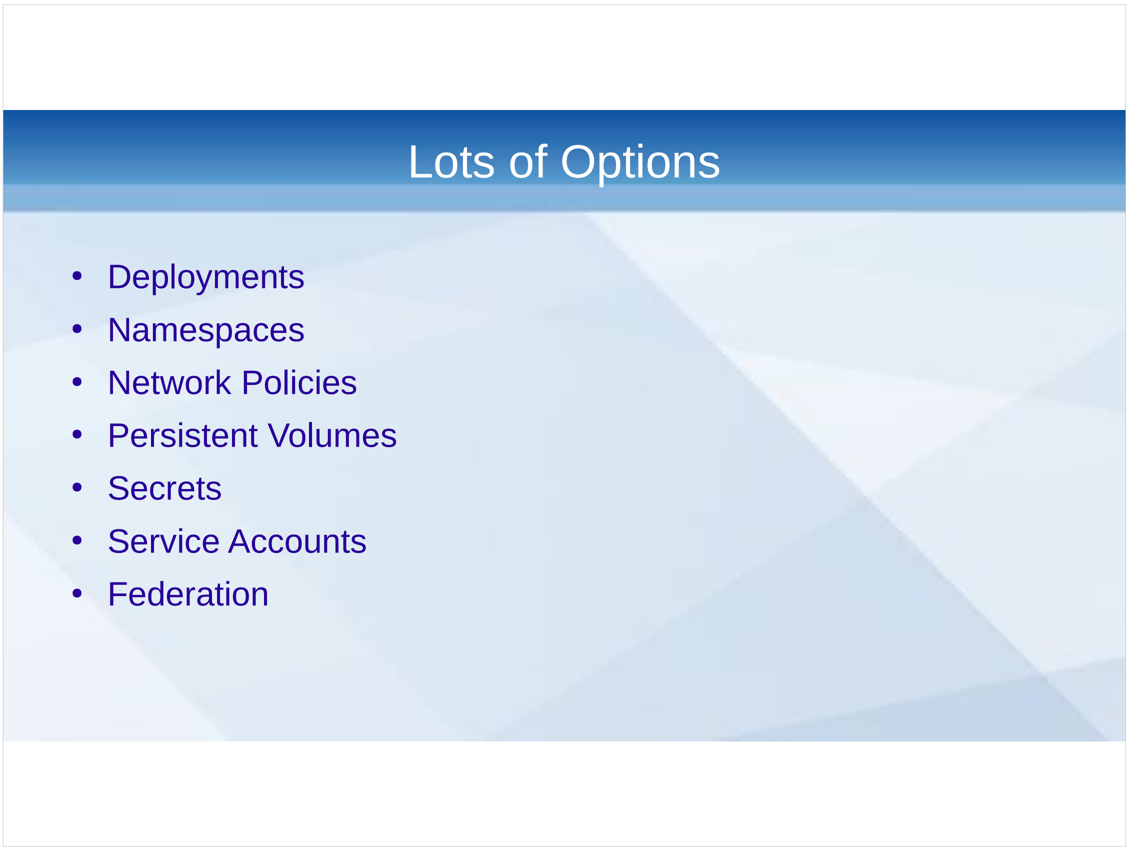
Kube-proxy Specifics

- Exposes a random port to the network on all nodes
- Routing to endpoints happens within kube-proxy

Routing happens via iptables, using a refresh interval.

Kube-proxy Specifics





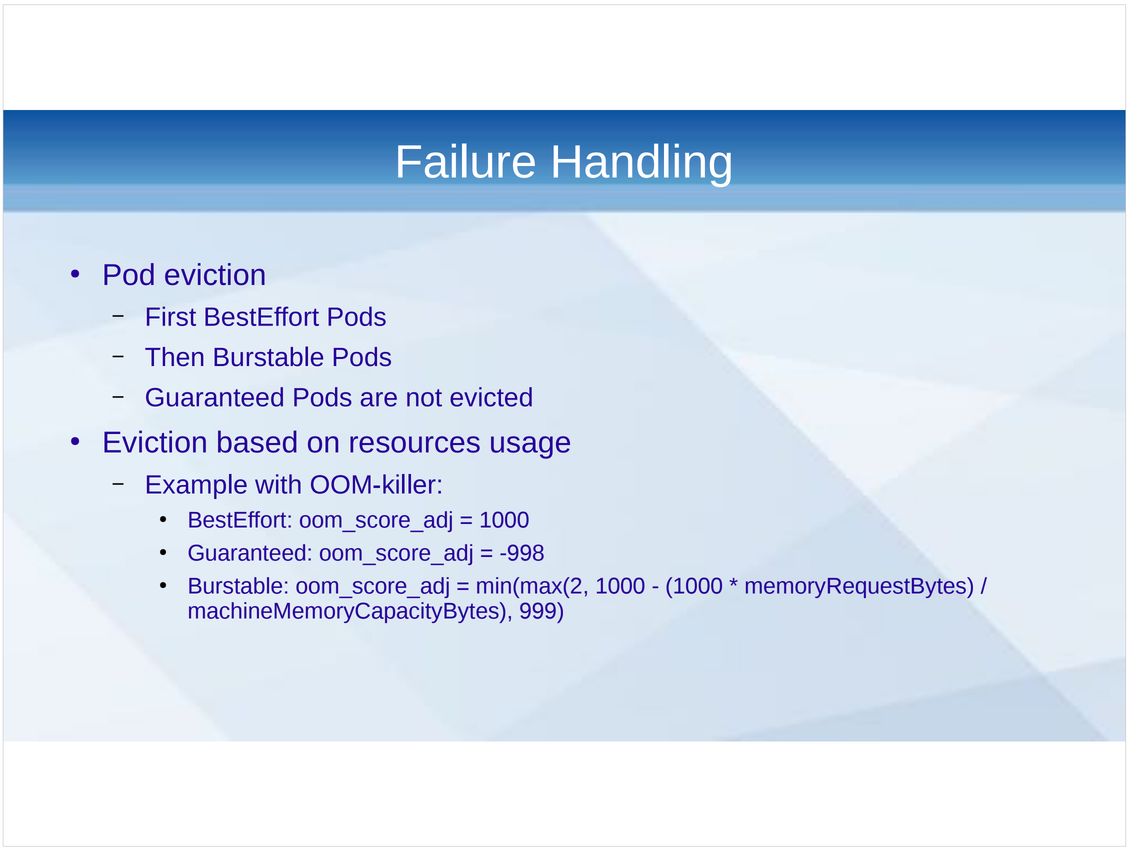
Lots of Options

- Deployments
- Namespaces
- Network Policies
- Persistent Volumes
- Secrets
- Service Accounts
- Federation

Barely discussed the basics, lots of additional options are being added. Kubernetes is under heavy development. Every 4 weeks (more or less) a new release.

As of March 6, 2017, 45k commits (since June 2014!), 1100 contributors (~44 commits per day on average)

Want to learn more, check out Kelsey Hightower on Youtube (he's super dope).

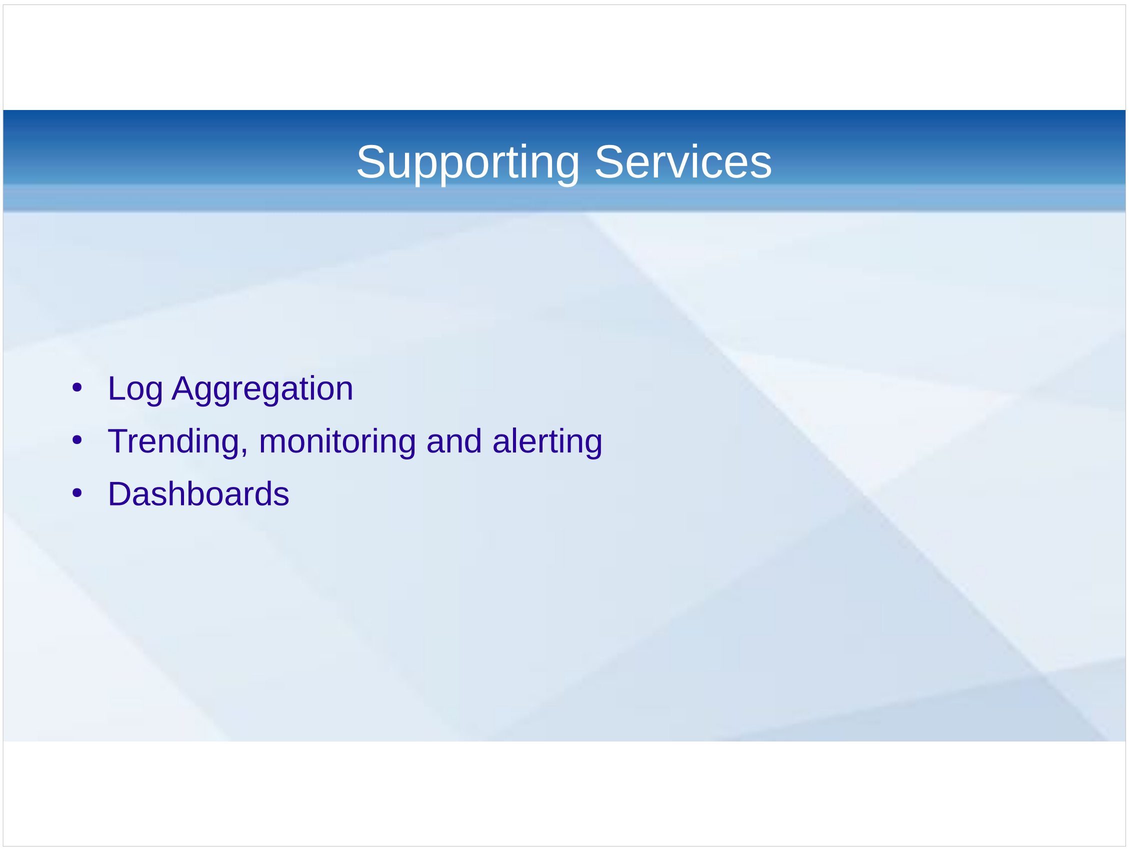


Failure Handling

- Pod eviction
  - First BestEffort Pods
  - Then Burstable Pods
  - Guaranteed Pods are not evicted
- Eviction based on resources usage
  - Example with OOM-killer:
    - BestEffort: oom_score_adj = 1000
    - Guaranteed: oom_score_adj = -998
    - Burstable: oom_score_adj = min(max(2, 1000 - (1000 * memoryRequestBytes) / machineMemoryCapacityBytes), 999)

Failure handling is important aspect of container management. Kubernetes has a lot of heuristics in place to determine the best way of dealing with resource shortage.

But failure handling is also in place when nodes just disappear, pods get automatically rescheduled (it's bliss to be able to just reboot a machine without customer noticing!)



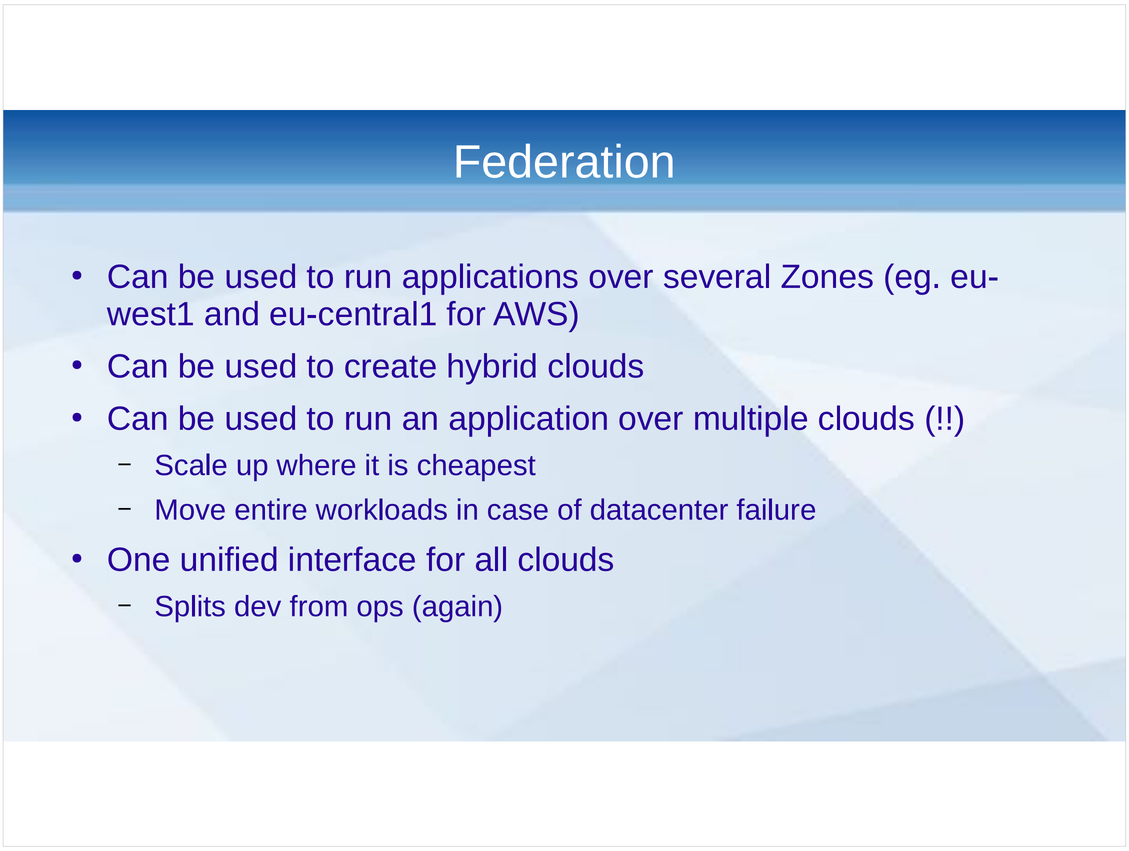
Supporting Services

- Log Aggregation
- Trending, monitoring and alerting
- Dashboards

Logs are lost if not collected. Checking multiple logs is tedious. Log aggregation makes it easier (and searchable). We use ElasticSearch as storage (statefulset!), Kibana as the frontend and fluentd (daemonset) for shipping.

Prometheus works great with Kubernetes. Has native support for scraping. Also written in Go. Also derived from a Google project (Borgmon). Very powerful. We use Grafana for graphing and dashboards, alertmanager for ... alerting!

Several dashboard are available to give easier insight into Kubernetes. We generally don't use them (either too powerful or not powerful enough).

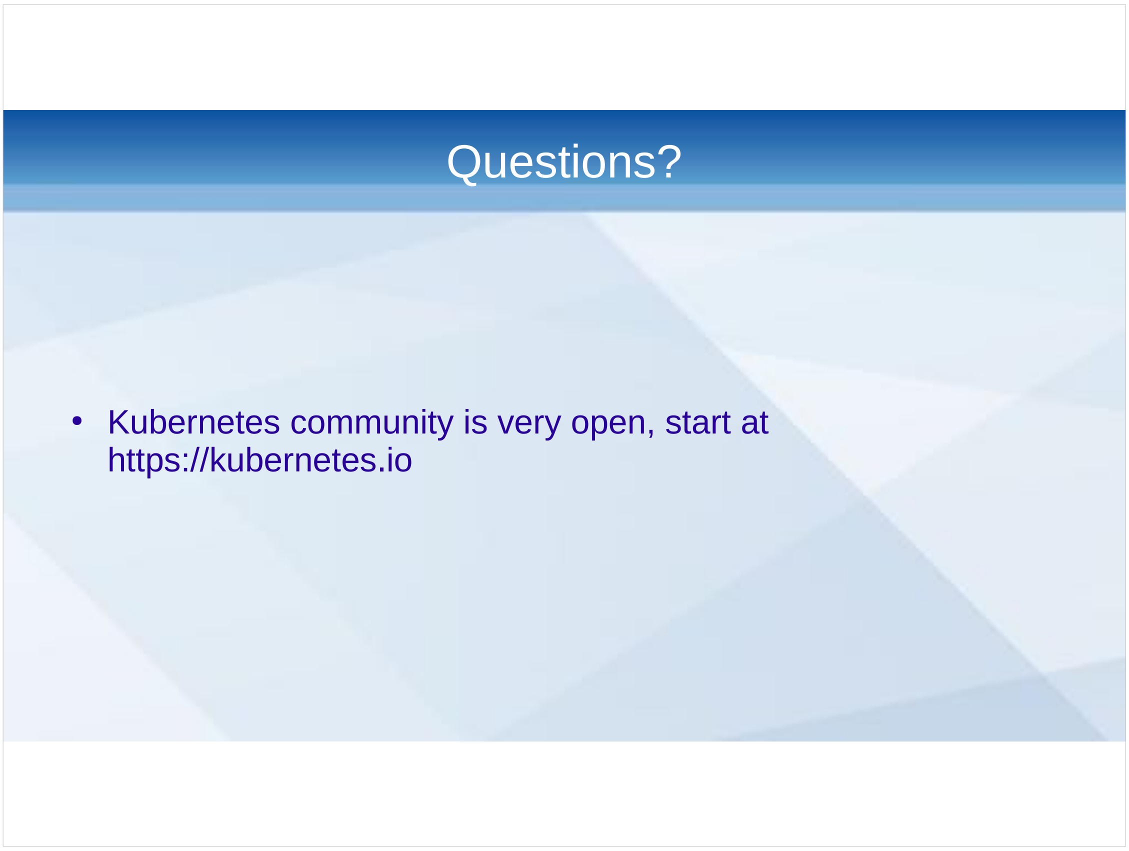


Federation

- Can be used to run applications over several Zones (eg. eu-west1 and eu-central1 for AWS)
- Can be used to create hybrid clouds
- Can be used to run an application over multiple clouds (!!)
  - Scale up where it is cheapest
  - Move entire workloads in case of datacenter failure
- One unified interface for all clouds
  - Splits dev from ops (again)

Hybrid clouds = Private Cloud + Public Cloud
Multiple clouds = AWS and GCE... Or Digital Ocean and Scaleway... etc.
Moving workloads across clouds is fun and all, but your DNS will still be pointing towards a failing cloud, if you're unlucky.

Interesting future!



Questions?

- Kubernetes community is very open, start at https://kubernetes.io

Was I any good?
What did you miss?

Thanks for listening

www.kumina.nl