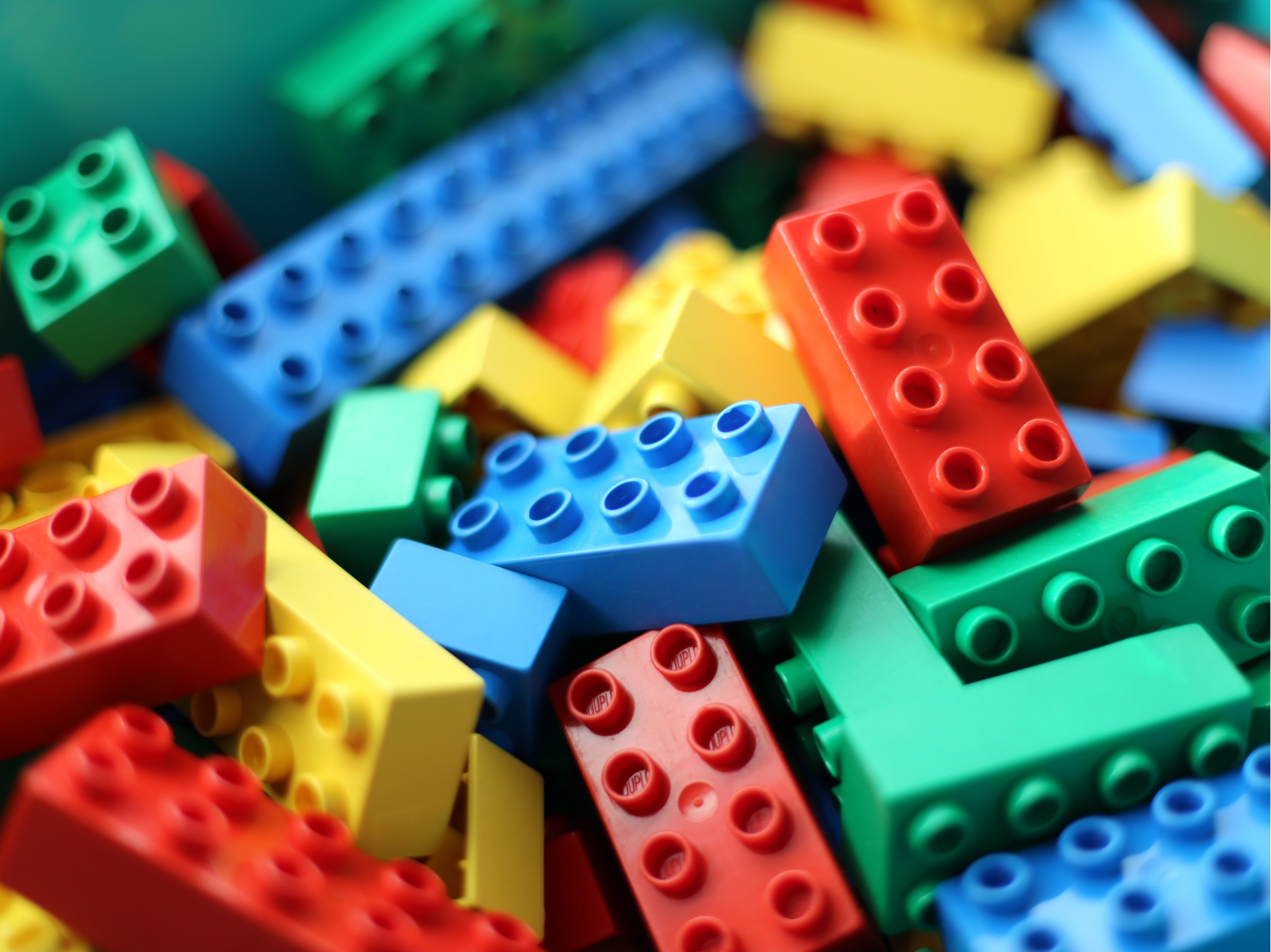
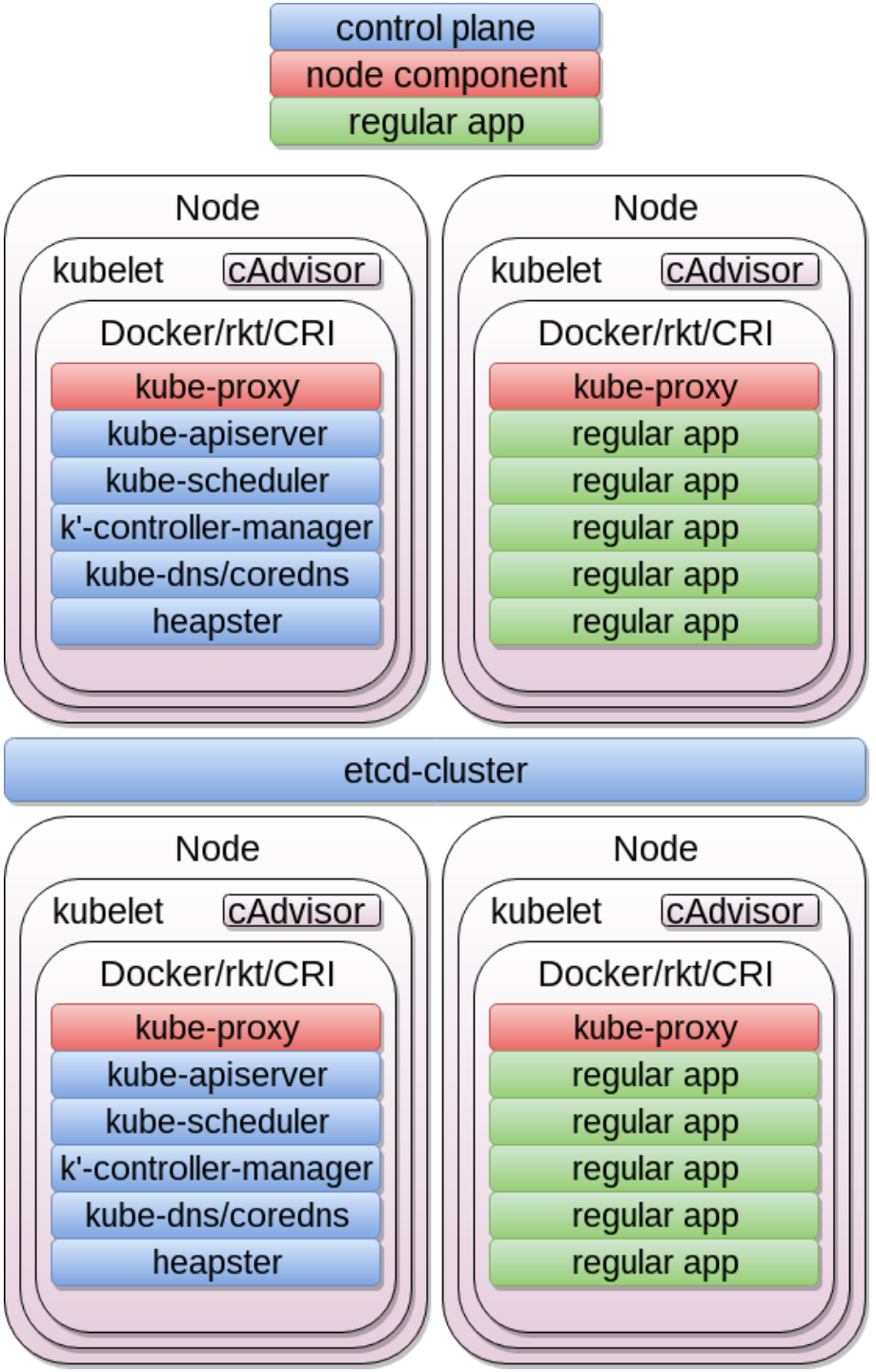
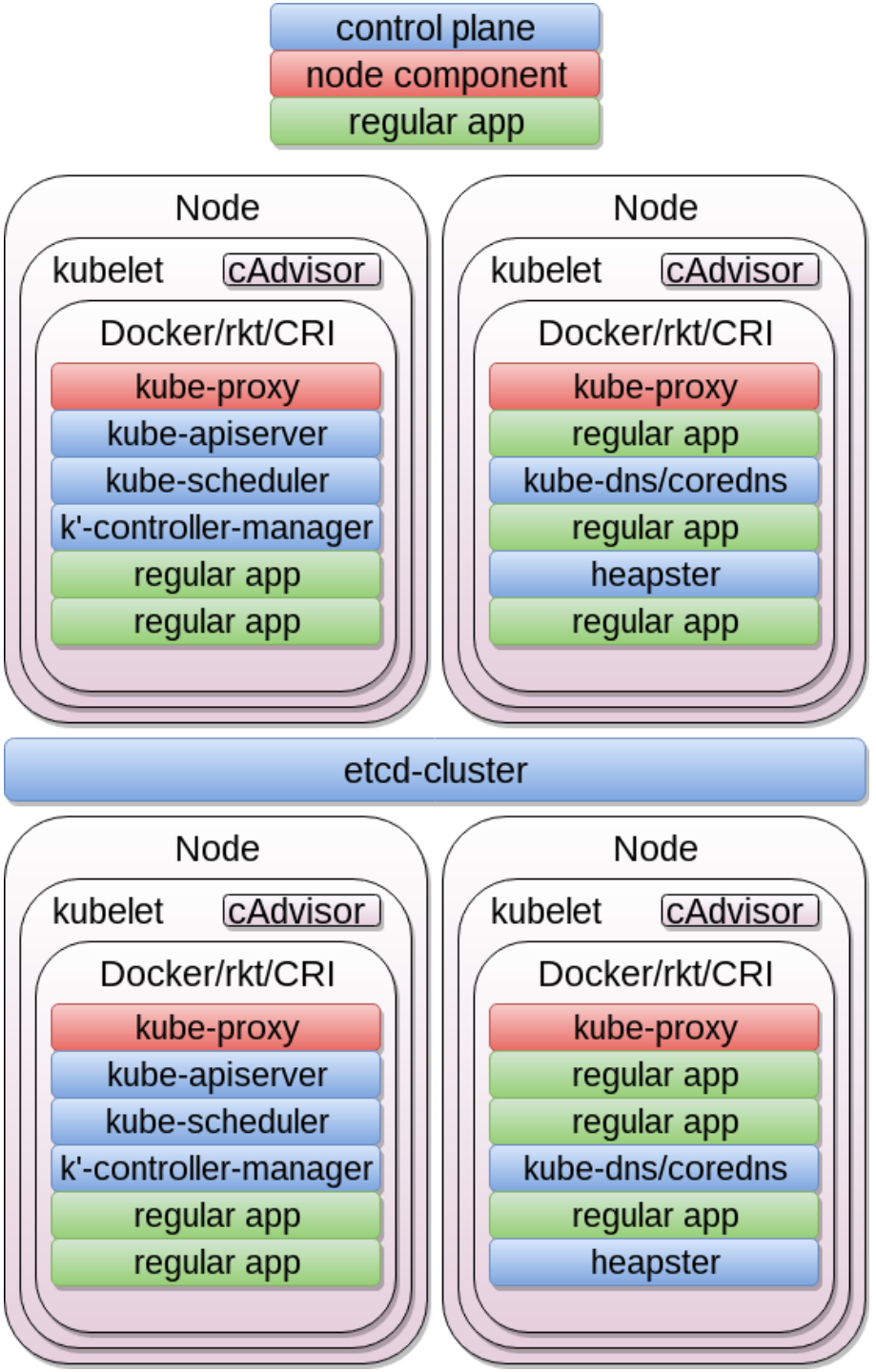
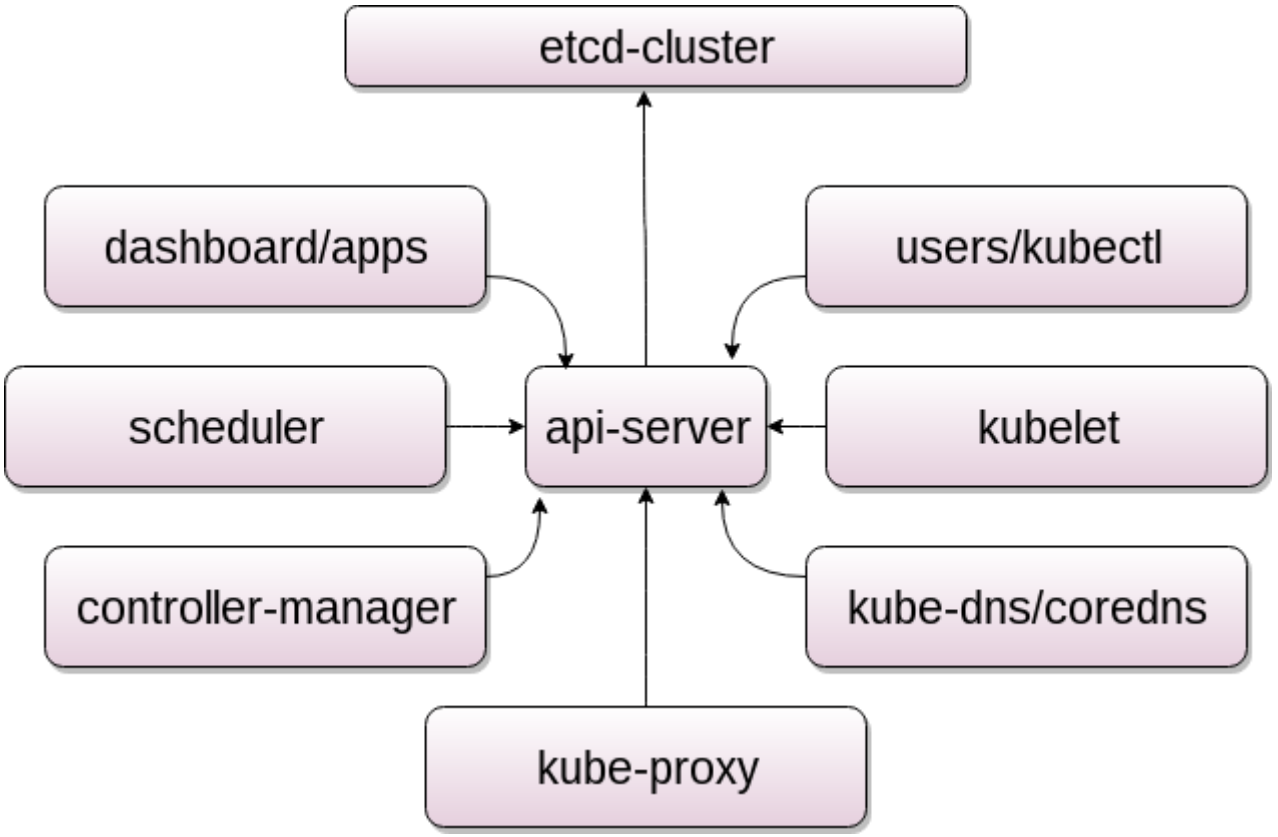
Core Kubernetes components

Stockholm, September 2018
By Kristian Lyngstøl




control plane
node component
regular app
Node
kubelet
cAdvisor
Docker/rkt/CRI
kube-proxy
kube-apiserver
kube-scheduler
k'-controller-manager
kube-dns/coredns
heapster
Node
kubelet
cAdvisor
Docker/rkt/CRI
kube-proxy
regular app
regular app
regular app
regular app
regular app
etcd-cluster
Node
kubelet
cAdvisor
Docker/rkt/CRI
kube-proxy
kube-apiserver
kube-scheduler
k'-controller-manager
kube-dns/coredns
heapster
Node
kubelet
cAdvisor
Docker/rkt/CRI
kube-proxy
regular app
regular app
regular app
regular app
regular app


control plane
node component
regular app
Node
kubelet
cAdvisor
Docker/rkt/CRI
kube-proxy
kube-apiserver
kube-scheduler
k'-controller-manager
regular app
regular app
Node
kubelet
cAdvisor
Docker/rkt/CRI
kube-proxy
regular app
kube-dns/coredns
regular app
heapster
regular app
etcd-cluster
Node
kubelet
cAdvisor
Docker/rkt/CRI
kube-proxy
kube-apiserver
kube-scheduler
k'-controller-manager
regular app
regular app
Node
kubelet
cAdvisor
Docker/rkt/CRI
kube-proxy
regular app
regular app
kube-dns/coredns
regular app
heapster


etcd-cluster
dashboard/apps
users/kubectl
scheduler
api-server
kubelet
controller-manager
kube-dns/coredns
kube-proxy

A quick note on PODs

- A „Pod“ is the smallest unit you work with in Kubernetes
- One or more container
- Share network namespace
- Example:
  - kube-dns Pod is actually kube-dns+ipmasq
- Pods are „ephemeral“: users deal with e.g. Deployment objects, a deployment manager then creates the appropriate pods
- Generally: Pods can be deleted on a whim

etcd

- clustered key/value store
- kube-apiserver talks directly to etcd
  - but *everything* interacts with it indirectly
- Used for quorum decisions for all other services
- Minimum 3 nodes
- The heart of every cluster

Control plane services

- No „primary“ node, only control plane services
- Can run inside Kubernetes
- Can run on regular nodes, mixed with ordinary apps
- Should run multiple instances
  - Only one (of each) is active at a time
- All control plane services are stateless
  - State is kept in etcd.

kube-apiserver:

- Gateway to etcd
- All other control plane services connect to the apiserver
- All interaction with Kubernetes goes through apiserver
- Provides authentication, enforces RBAC, enforces a variety of other limitations.
- Should be reachable from outside the cluster

More on the API server

- Also does schema-“verification“ or linting
- Checks resource quotas, applies default values to e.g. StorageClass, attaches a service account, etc
- Will almost certainly require your attention at some point

kube-scheduler

- Schedules pods on nodes
- .... that‘s it.

kube-controller-manager

- Makes „all the decisions“
  - attachdetach, bootstrapsigner, clusterrole-aggregation, cronjob, csrapproving, csrcleaner, csrsigning, daemonset, deployment, disruption, endpoint, garbagecollector, horizontalpodautoscaling, job, namespace, nodeipam, nodelifecycle, persistentvolume-binder, persistentvolume-expander, podgc, pv-protection, pvc-protection, replicaset, replicationcontroller, resourcequota, route, service, serviceaccount, serviceaccount-token, statefulset, tokencleaner, ttl

Controller manager, part 2

- Actually modularized
- You can create/delete a Deployment without it, but no pod will be stopped or started
- Feature gates: You can enable/disable a range of features
- E.g.: Automated PKI, automated node bootstrapper, etc

Control plane in general

- Commonly assigned to fixed nodes
- Should run a „reasonable amount“ of them
  - To handle outages, not balance load

„Soft“ control plane

- Certain services are expected, but not required
  - Cluster DNS, metrics, log-shipper, dashboard, storage, ingress controllers, etc
- More common to treat them like regular services, instead of affixing them to nodes
- Scaling characteristics vary
  - More DNS pods will spread load
  - More heapster/stats pods will _increase_ load

coredns/kube-dns

- Provides DNS resolution between service names and either service IPs or POD IPs
- Critical to practical usage, but not used by control plane services internally
- coredns is the „new thing“, kube-dns is „old“ - both work and can do the same thing.

Metrics: heapster

- Kubelet provides cAdvisor metrics per node
- heapster collects metrics from all nodes
- ships it to one or more sink:
  - influx, prometheus, elasticsearch, ???
- Some development in place to change metric APIs - can create headaches with mix of old-style and new-style services.

Logging, events

- Pod logs and cluster events are frequently garbage collected in-cluster
- Various means of shipping it off to ELK exist
  - fluentd for app logs
  - „eventer“ for cluster events („Pod created“, „Deployment scaled down“, etc)

„Worker“ nodes

- Every Kubernetes-node needs three services:
- kubelet - talks to apiserver to start/stop pods
- kube-proxy - materializes the services network
- container runtime, typically docker

kubelet - start/stop pods

- The kubelet service starts and stops pods.
- It does this through TWO mechanisms:
  - Talking to the API server to get tasks
  - Reading a manifest from disk
- The second method is used to ensure control plane services start even when the apiserver is down (e.g.: if the entire cluster is down)
- Uses either docker or other container runtimes to start pods.

kube-proxy

- kube-proxy implements the service network
- Every node can answer every service address, regardless of where the actual POD servicing it is located.
- Originally done in user-space. Now done in iptables. In the future, done in LVS.
- Control plane should still work without this - most other things will not.

Router

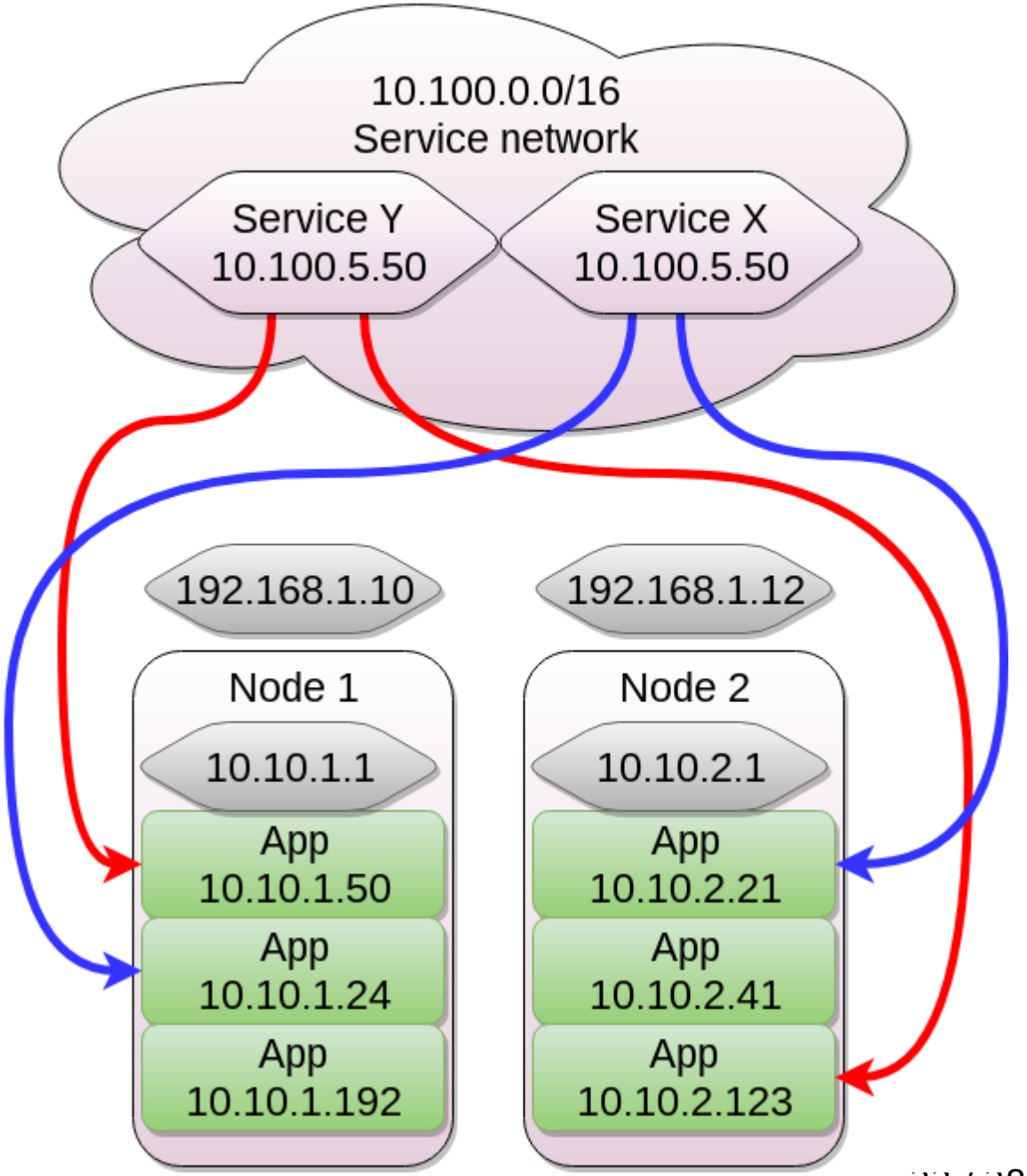
10.10.1.0/24 via 192.168.1.10

10.10.2.0/24 via 192.168.1.12

10.100.0.0/16 via 192.168.1.5

192.168.1.5
Anycast? Keepalived?

Any node can terminate service net.



Networking - pods

- Every POD has a unique IP
- Pod IP‘s are per-node
- Every node must be able to reach every pod IP
- Typically assign a /24 network (255 IPs) per node
- IPs are re-used when pods are deleted.

Networking - service addresses

- A single service network is established across the cluster
- Every node can terminate the service network: map a service address to one or more Pod
- Today: Iptables. Previously: user-space. Future: lvs.
- IPs are permanent - until service is deleted.
- Should have at least /20 - recommended: /16

DNS

- Kubernetes DNS is semi-required in-cluster, and very convenient if exposed
- Two types of services: Service addresses and headless:
  - Service addresses has a permanent fixed IP
  - Headless only exist in DNS, and expose Pod IPs as round robin DNS
- Without routing, this is almost useless

Ingress services

- Ingress controllers can be used to „ingress“ HTTP and HTTPS traffic into the cluster.
- Still uses a service, but multiple ingress objects can use the same ingress service.
- Layer 7
- Can provide neat things such as automatic let‘s encrypt integration
- Various implementations
  - nginx, træææfikk (or whatever), F5 BigIP, etc

Networking - routing

- Overlay networks are NOT required
- Pod and Service networks must be routable internally in the cluster
- Routing Pod network „globally“ is very convenient, and achievable
- Routing Service network is even more convenient, but slightly trickier (due to HA)
- Using ingress services or NodePorts or LoadBalancer objects is a bit of a crutch

I ran out of slides

- Questions?