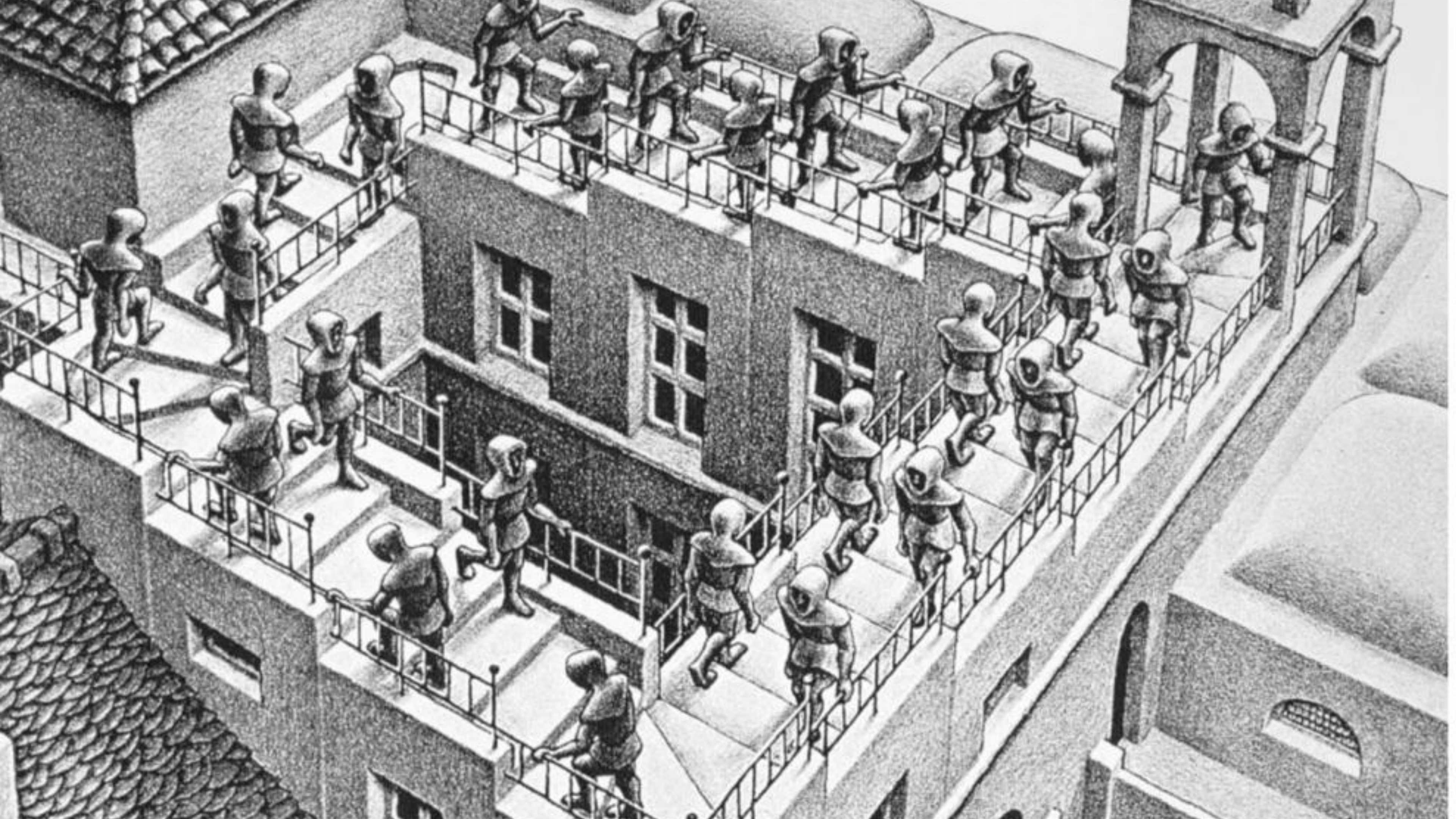
KubeCon | CloudNativeCon

Europe 2019

Operating kube-apiserver

Without Hiccups

Stefan Schimanski & David Eads



Agenda

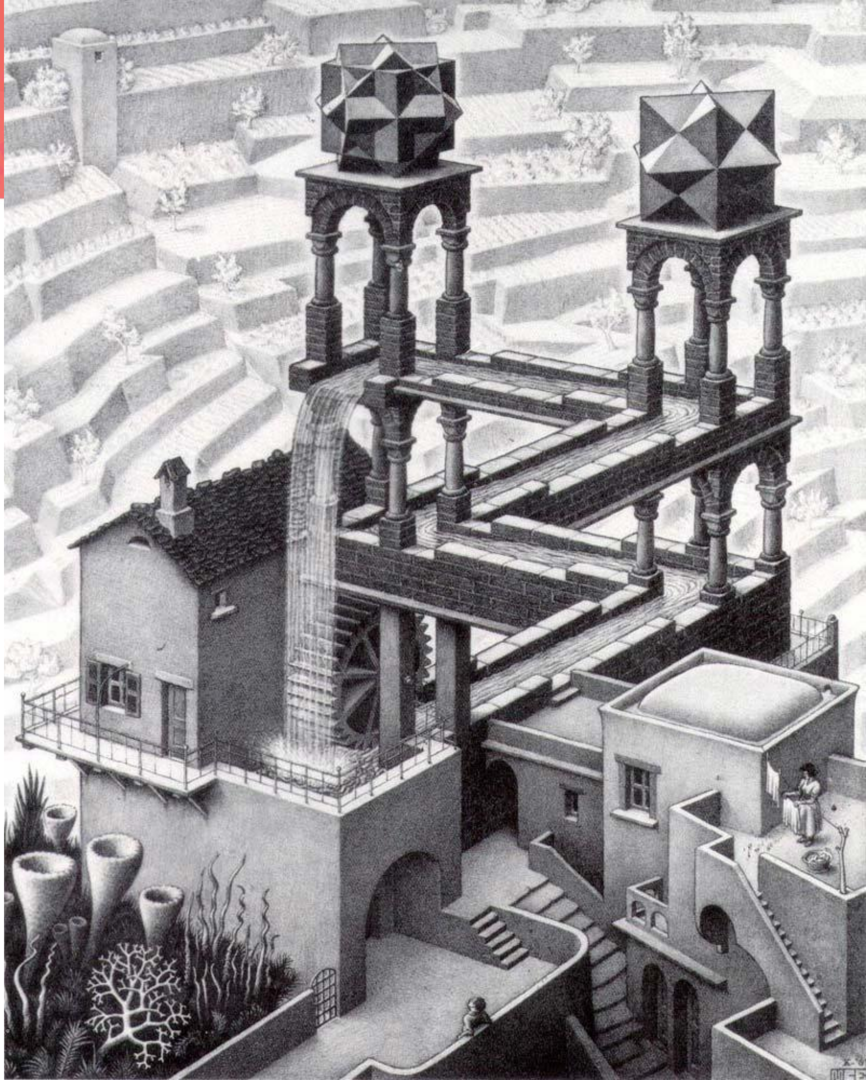
- control plane self-hosting reloaded
- idea meets reality
- graceful termination
- health and readiness checks
- cert rotation
- disaster recovery

bootkube, the vision

- **initialize** a cluster **with static pods**
- use them to **create daemonsets**
- **kill static pods**

- **all future updates** using **DaemonSet**

- but what if we **crash**...



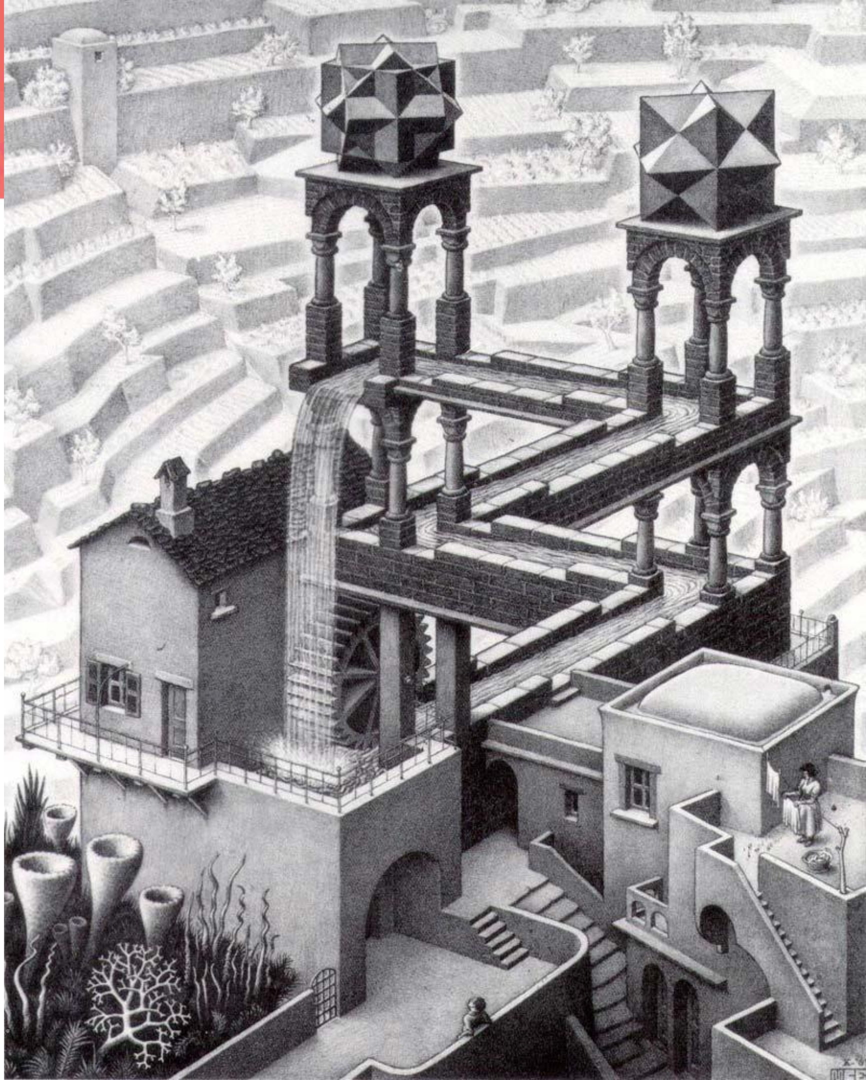
bootkube, the vision

- **initialize** a cluster **with static pods**
- use them to **create daemonsets**
- **kill static pods**

- **all future updates** using **DaemonSet**

- but what if we **crash**…

We have the **static pod checkpointer**!



bootkube checkpointer / recover

- What does the **checkpointer** actually do?

  tl/dr:
  copying pod manifests, secrets, configmaps to the filesystem
  and **rewriting** the **pod yaml** … to hopefully **run as static pod**.

- **Does it work?** Most of the time, but happens if...

bootkube: update of doom

1. **Update kube-controller-manager** version, **one node at a time**

2. Recall that the kube-controller-manager is lease based
3. Leases are acquired by “old” nodes.
4. **`/healthz` is ok** immediately **=> checkpointing**
5. **New node** finally **gets lease** and **panics** 💥
6. **Rollback the daemonset** to fix it,
   but there is **no kube-controller-manager running** to recover it.
7. Checkpointer on every node has pods that can’t run.
8. Cry.

the alternative: static pods

1. Avoid pivot
2. Avoid checkpointer
3. Always run the same pod
4. Always tolerate kubelet connection failures
5. Always have a local backup
6. No cyclical dependencies

Self-hosting is awesome, but in a down-to-earth way:

control-plane as static pods – operator as DaemonSet

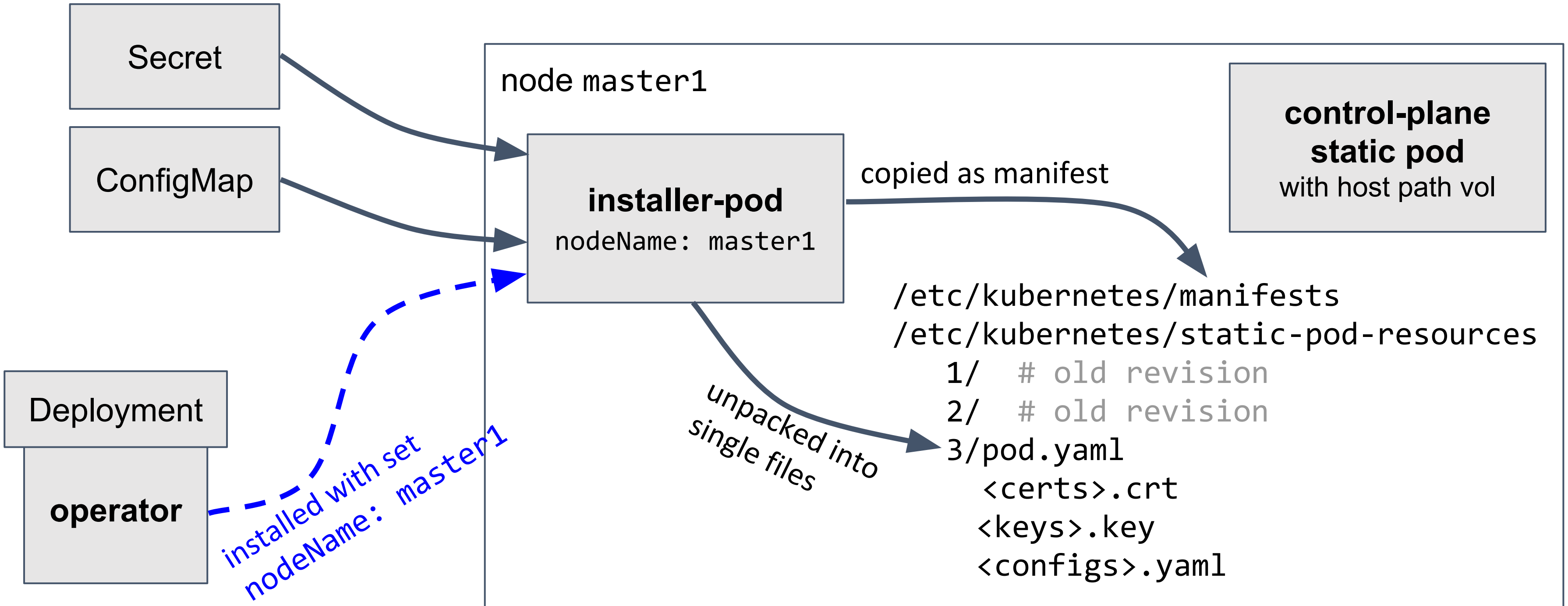
static pod management

1. Create a **set of immutable configmaps & secrets**: a **revision**.
2. Create a **installer-pods**, forced to a particular master without scheduler

   ```
   spec.NodeName: master<n>
   ```

   with **hosts mounts**: **static pod manifest** & **static resources directories**
3. The **installer-pod copies**: **configmaps** & **secrets** → **static resources** dir
   static pod manifest → **static pod** dir
4. **Wait for new static pod** to become ready
5. Then **move to next node**
6. If you hit an “update of doom”, you can retry by creating new revisions and static pods because no workload resource is required.

static pod management



idea meets reality

- make sure to **keep nodes up to date** (daemonset)
- make sure to **remain available** (deployments, PDB)
- make sure to **clean up after yourself** (pruner, reminds of cronjobs)
- make sure **clients don’t get immediately dropped**
- make sure you’re **not-ready**, **but** you are **healthy**
- make sure that your **load-balancer stops sending traffic**
- make sure that the **service network stops sending traffic**

idea meets reality

- make sure to **keep nodes up to date** (daemonset)
- make sure to **remain available** (deployments, PDB)
- make sure to **clean up after yourself** (pruner, reminds of cronjobs)
- make sure **clients don’t get immediately dropped**
- make sure you’re **not-ready**, **but** you are **healthy**
- make sure that your **load-balancer <u>REALLY</u> stops sending traffic**
- make sure that the **service network stops sending traffic**

Errors everywhere

Failed to list *core.Service: Get https://172.30.0.1/api/v1/services?limit=500&resourceVersion=0:dial tcp 172.30.0.1:6443: **connect: connection refused**

✗ no HTTP server listening

I0326 20:03:52.589926 3853 streamwatcher.go:107] Unable to decode an event from the watch stream: http2: **server sent GOAWAY and closed the connection**; LastStreamID=53, ErrCode=NO_ERROR, debug=""

✓ long-running request HTTP/2

F1030 18:27:51.842709 4254 server.go:262] cannot create certificate signing request: Post https://1.2.3.4:6443/apis/certificates.k8s.io/v1beta1/certificatesigningrequests: **EOF**

✗ cut-off request non-long-running

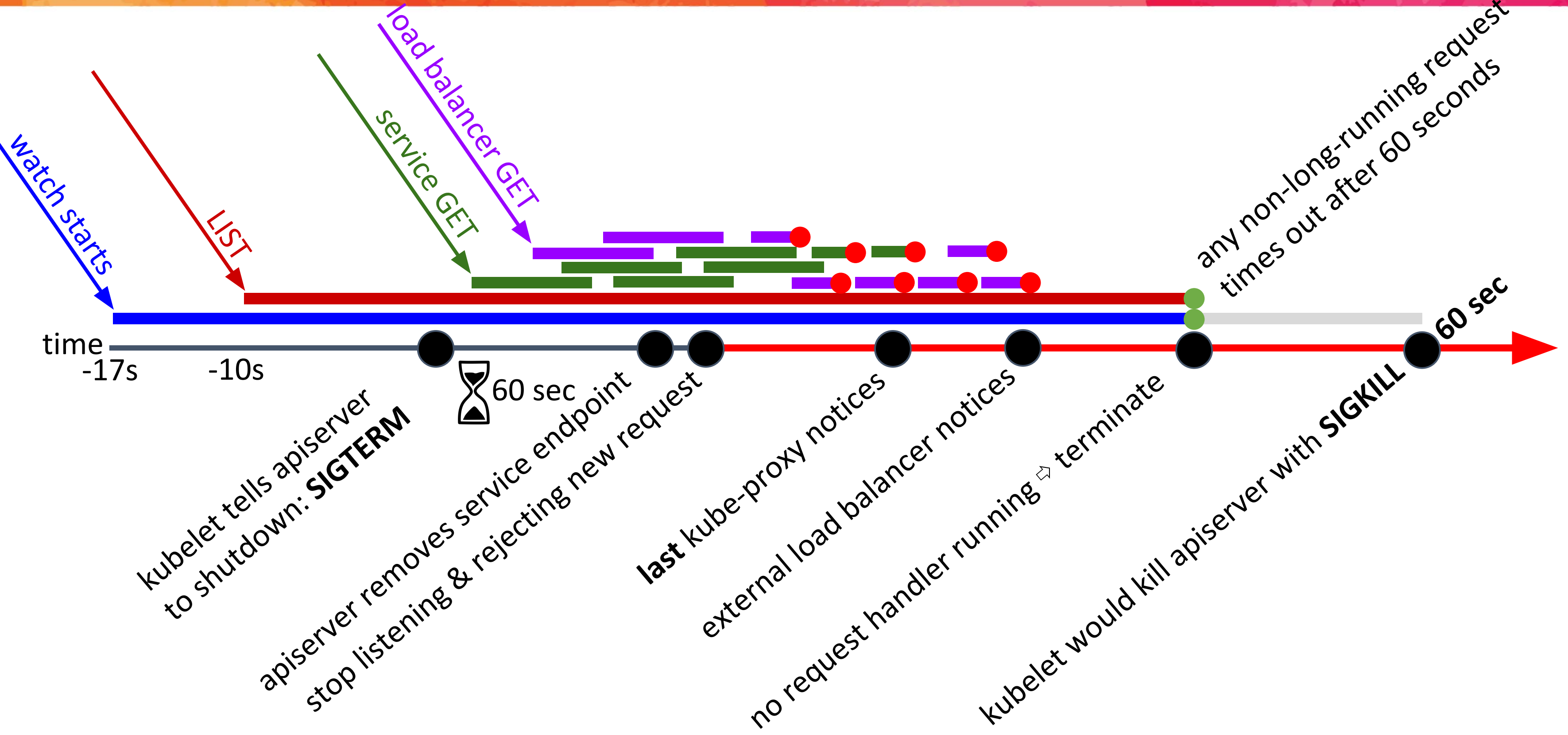
I0417 12:18:54.309074 1 streamwatcher.go:103] Unexpected EOF during watch stream event decoding: **unexpected EOF**

✓ long-running request

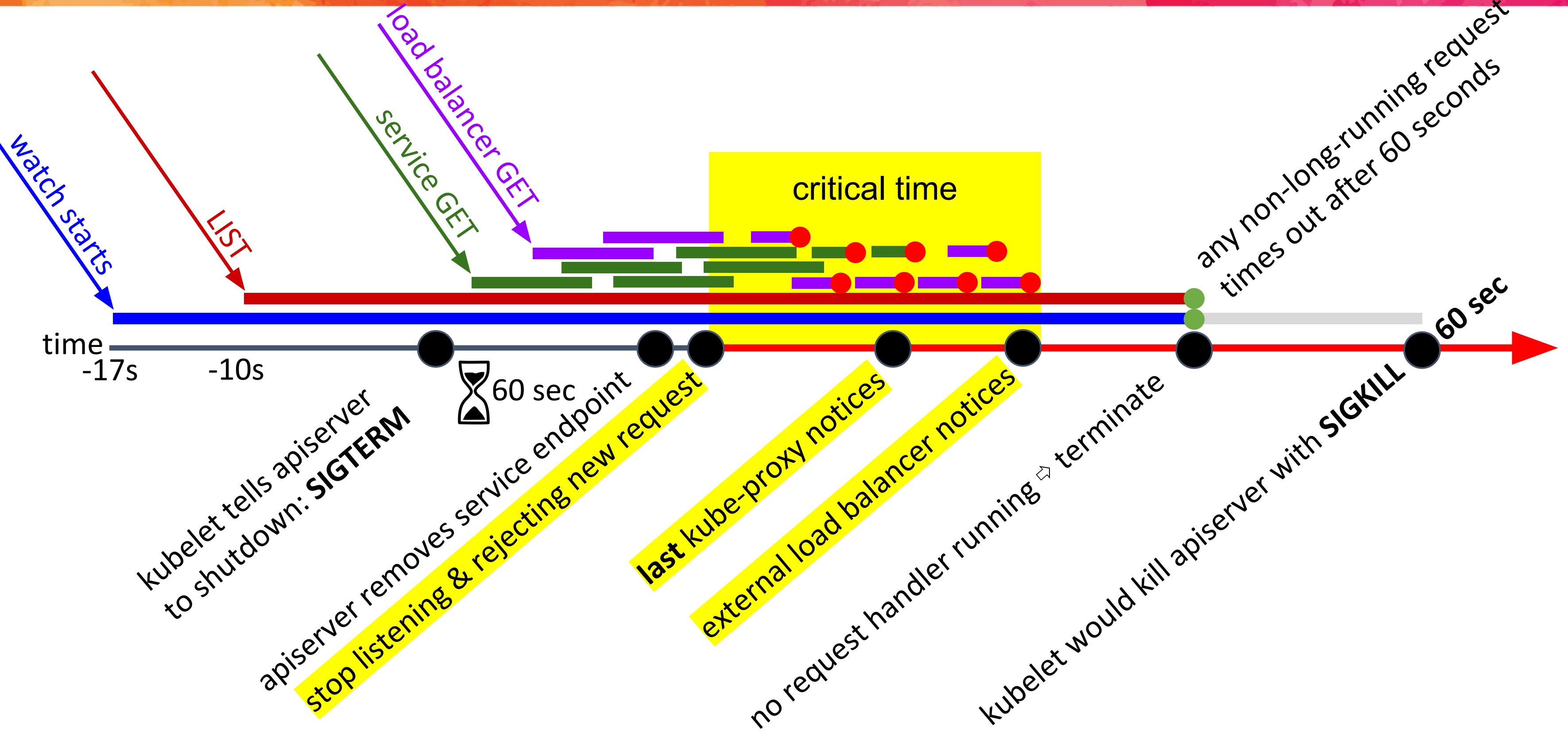
apiservice/v1.apps.openshift.io: not available: no response from https://172.30.83.61:443: Get https://172.30.83.61:443: dial tcp 172.30.83.61:443: **connect: no route to host**.

~ probably network issue or rebooting node

Graceful Termination



Graceful Termination

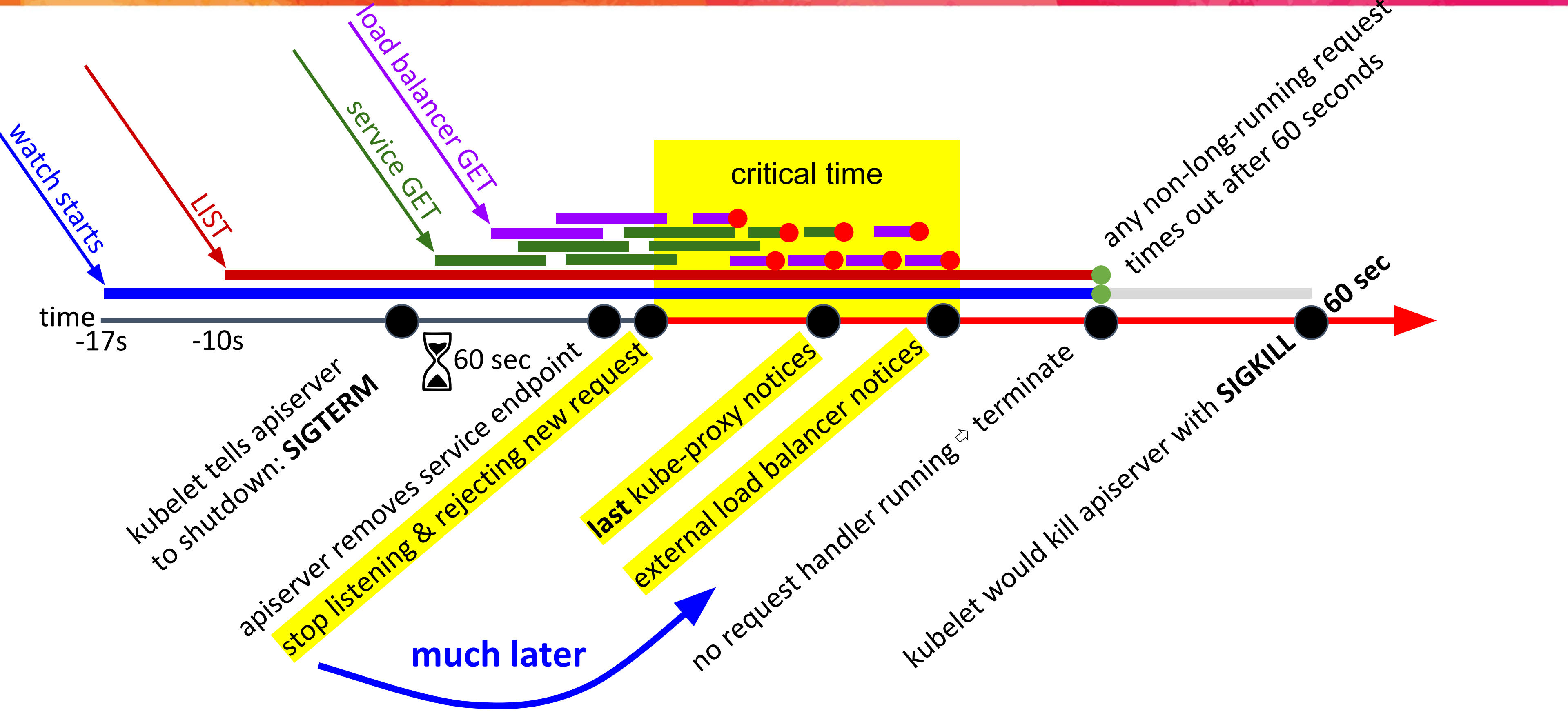


Minimum Shutdown Duration

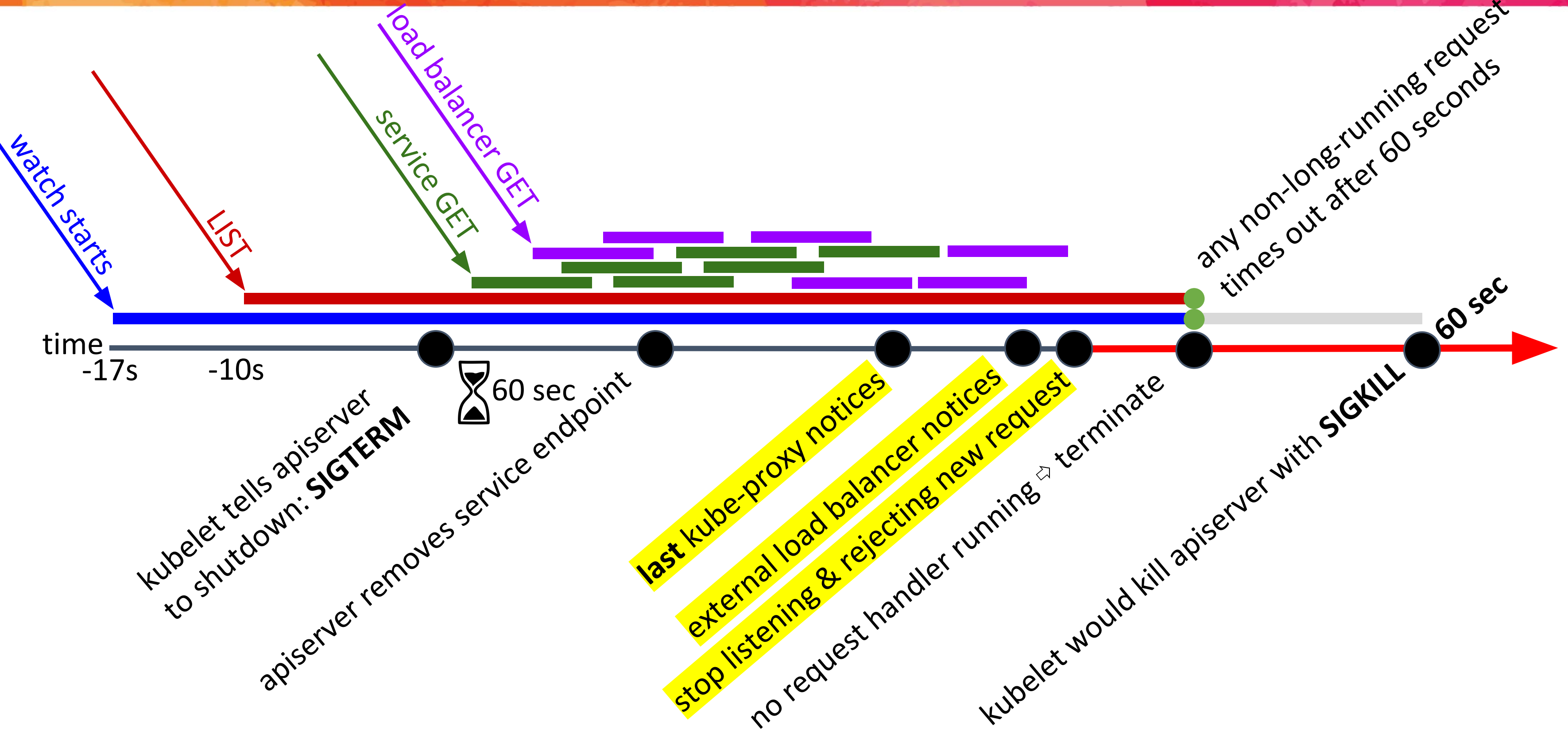
- **keep listening & responding** normally for **n seconds**
- **immediately** report **readiness=False**
  to internal & external load balancers
- give them **time to adapt**
- **then stop listening** and doing graceful termination.

does not exist today
PR 74416

Graceful Termination



Graceful Termination



Liveness vs. Readiness – healthz or readyz

- **/healthz** error => kubelet kills the pod
- **/readyz** error => endpoint controller removes endpoint **internal**
  load balancers remove target **external**

We need (don't have today):

1. **SIGTERM** to kube-apiserver
   a. **/readyz error**
   b. **remove endpoint** from `default/kubernetes`
2. wait **minimum-shutdown-period**
3. **graceful shutdown** up to 60 sec

Why we do all this

Config changes

Upgrades

Cert Rotation

Cert Rotation – when it gets tricky

parties have to trust new cert (via CA)

parties might be slow loading new CAs

parties might be slow using new certs

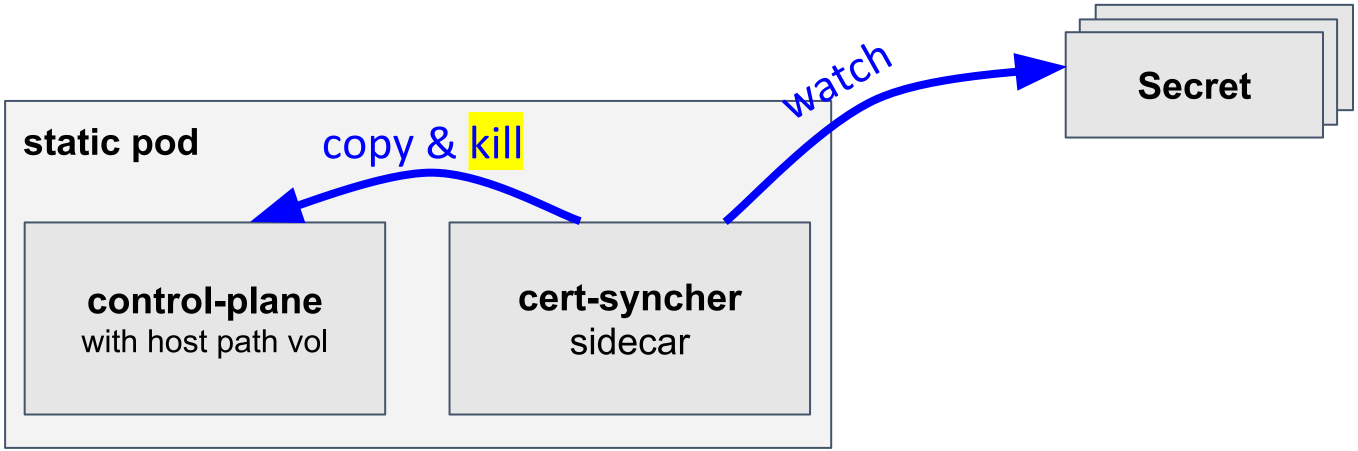
possible lag (minutes!) **in both directions** => **keep CAs around**

lack of synchronization => **be slow**

unbounded lag: nodes are down or **golden images**

Live Reload of Certs

- **possible in Golang** without losing requests
- **kubelet remounts** secrets & **cert-syncher** for static pods
- but we don't support live reload. We should.



certificate settings today

certificate settings today

--client-ca-file	CA bundle used to verify client certificate connections from clients and identify users. (I am Bob). Must be able to verify `kube-controller-manager --cluster-signing-cert-file` or `kubelet --rotate-certificates` will fail.
--requestheader-client-ca-file	CA bundle used to verify client certificate connections from front proxies that are asserting the identity of user. (This request is from Bob). Must be able to verify `kube-apiserver --proxy-client-cert-file` or aggregation in the cluster will fail by default.
--kubelet-certificate-authority	CA bundle used to verify kubelets for connections from KAS to kubelet. (Think logs,exec,etc). Must be able to verify `kubelet --tls-cert-file`. Must be able to verify `kube-controller-manager --cluster-signing-cert-file` or `kubelet --rotate-server-certificates` will fail.
--kubelet-client-certificate	Client cert used to identify KAS to the kubelets. Must be verifiable by `kubelet --client-ca-file`.
--kubelet-client-key	Client key used to identify KAS to the kubelets
--proxy-client-cert-file	Client cert used to identify KAS to aggregated API servers as a front proxy. Must be verifiable by `kube-apiserver --requestheader-client-ca-file` or aggregation in the cluster will fail by default
--proxy-client-key-file	Client key used to identify KAS to aggregated API servers as a front proxy
--service-account-key-file	RSA keys used to verify ServiceAccount tokens. Must be able to verify `kube-controller-manager --service-account-private-key-file` for all keys you want to continue working.
kube-controller-manager	**What's it for**
--client-ca-file	CA bundle used to verify client certificate connections from clients and identify users. (I am Bob)
--tls-cert-file	Serving cert used to serve requests
--tls-private-key-file	Serving key used to serve requests

certificate settings today

--tls-private-key-file	Serving key used to serve requests not matching SNI
--tls-sni-cert-key	Special flag format to specify hostname-pattern,cert,key tuples to serve matching SNI requests. If used for kubernetes.default.service, must be verifiable with `kube-controller-manager --root-ca-file`.
--client-ca-file	CA bundle used to verify client certificate connections from clients and identify users. (I am Bob). Must be able to verify `kube-controller-manager --cluster-signing-cert-file` or `kubelet --rotate-certificates` will fail.
--requestheader-client-ca-file	CA bundle used to verify client certificate connections from front proxies that are asserting the identity of user. (This request is from Bob). Must be able to verify `kube-apiserver --proxy-client-cert-file` or aggregation in the cluster will fail by default.
--kubelet-certificate-authority	CA bundle used to verify kubelets for connections from KAS to kubelet. (Think logs,exec,etc). Must be able to verify `kubelet --tls-cert-file`. Must be able to verify `kube-controller-manager --cluster-signing-cert-file` or `kubelet --rotate-server-certificates` will fail.
--kubelet-client-certificate	Client cert used to identify KAS to the kubelets. Must be verifiable by `kubelet --client-ca-file`.
--kubelet-client-key	Client key used to identify KAS to the kubelets
--proxy-client-cert-file	Client cert used to identify KAS to aggregated API servers as a front proxy. Must be verifiable by `kube-apiserver --requestheader-client-ca-file` or aggregation in the cluster will fail by default
--proxy-client-key-file	Client key used to identify KAS to aggregated API servers as a front proxy
--service-account-key-file	RSA keys used to verify ServiceAccount tokens. Must be able to verify `kube-controller-manager --service-account-private-key-file` for all keys you want to continue working.

kube-controller-manager	**What's it for**
--client-ca-file	CA bundle used to verify client certificate connections from clients and identify users. (I am Bob)
--tls-cert-file	Serving cert used to serve requests
--tls-private-key-file	Serving key used to serve requests
--cluster-signing-cert-file	Signing cert used to issue approved CSR requests. Must be verifiable with `kube-apiserver --kubelet-client-certificate` and `kube-apiserver --client-ca-file` or `kubelet --rotate-certificates` will fail.
--cluster-signing-key-file	Signing key used to issue approved CSR requests
--requestheader-client-ca-file	CA bundle used to verify client certificate connections from front proxies that are asserting the identity of user. (This request is from Bob)
--root-ca-file	CA bundle injected into ServiceAccount token secrets. It is **only** intended to be used to verify a connection to the kube-apiserver on the service network. All other uses are either wrong or coincidence. Must be able to verify `kube-apiserver --tls-cert-file`
--service-account-private-key-file	RSA key used to sign ServiceAccount tokens. Must be verifiable by `kube-apiserver --service-account-key-file` or ServiceAccounts will not be able to

certificate settings today

kube-apiserver	What's it for
--etcd-cafile	CA bundle used to verify the etcd server really is the etcd server
--etcd-certfile	Client cert used to identify KAS to the etcd server
--etcd-keyfile	Client key used to identify KAS to the etcd server
--tls-cert-file	Serving cert used to serve requests not matching SNI. Must be verifiable with `kube-controller-manager --root-ca-file` or service must be handled with SNI.
--tls-private-key-file	Serving key used to serve requests not matching SNI
--tls-sni-cert-key	Special flag format to specify hostname-pattern,cert,key tuples to serve matching SNI requests. If used for kubernetes.default.service, must be verifiable with `kube-controller-manager --root-ca-file`..
--client-ca-file	CA bundle used to verify client certificate connections from clients and identify users. (I am Bob). Must be able to verify `kube-controller-manager --cluster-signing-cert-file` or `kubelet --rotate-certificates` will fail.
--requestheader-client-ca-file	CA bundle used to verify client certificate connections from front proxies that are asserting the identity of user. (This request is from Bob). Must be able to verify `kube-apiserver --proxy-client-cert-file` or aggregation in the cluster will fail by default.
--kubelet-certificate-authority	CA bundle used to verify kubelets for connections from KAS to kubelet. (Think logs,exec,etc). Must be able to verify `kubelet --tls-cert-file`. Must be able to verify `kube-controller-manager --cluster-signing-cert-file` or `kubelet --rotate-server-certificates` will fail.
--kubelet-client-certificate	Client cert used to identify KAS to the kubelets. Must be verifiable by `kubelet --client-ca-file`.
--kubelet-client-key	Client key used to identify KAS to the kubelets
--proxy-client-cert-file	Client cert used to identify KAS to aggregated API servers as a front proxy. Must be verifiable by `kube-apiserver --requestheader-client-ca-file` or aggregation in the cluster will fail by default
--proxy-client-key-file	Client key used to identify KAS to aggregated API servers as a front proxy
--service-account-key-file	RSA keys used to verify ServiceAccount tokens. Must be able to verify `kube-controller-manager --service-account-private-key-file` for all keys you want to continue working.

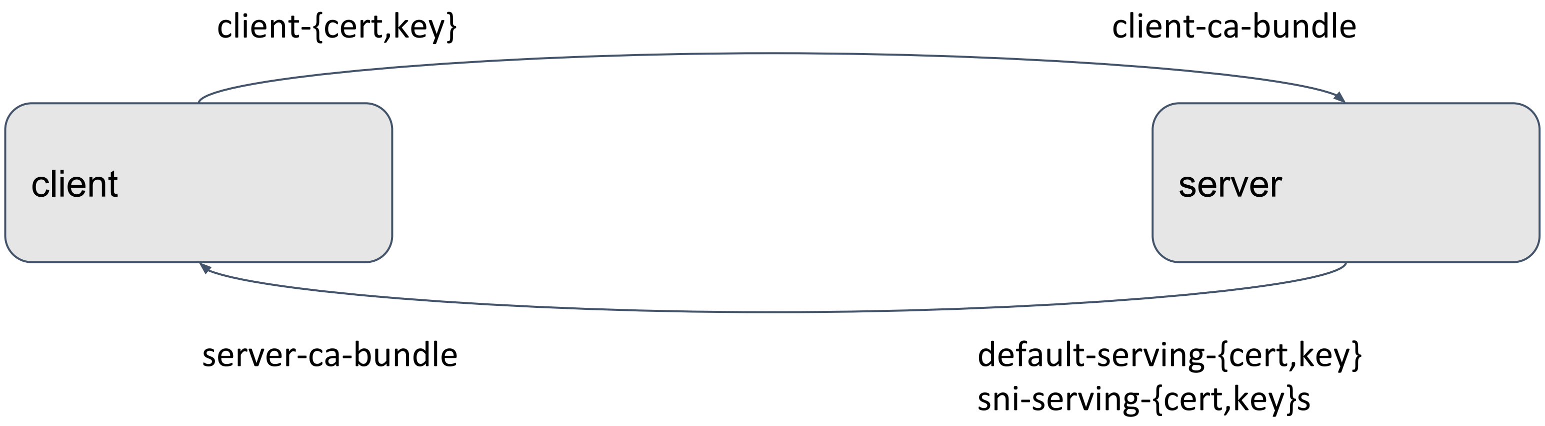
kube-controller-manager	What's it for
--client-ca-file	CA bundle used to verify client certificate connections from clients and identify users. (I am Bob)
--tls-cert-file	Serving cert used to serve requests
--tls-private-key-file	Serving key used to serve requests
--cluster-signing-cert-file	Signing cert used to issue approved CSR requests. Must be verifiable with `kube-apiserver --kubelet-client-certificate` and `kube-apiserver --client-ca-file` or `kubelet --rotate-certificates` will fail.
--cluster-signing-key-file	Signing key used to issue approved CSR requests
--requestheader-client-ca-file	CA bundle used to verify client certificate connections from front proxies that are asserting the identity of user. (This request is from Bob)
--root-ca-file	CA bundle injected into ServiceAccount token secrets. It is **only** intended to be used to verify a connection to the kube-apiserver on the service network. All other uses are either wrong or coincidence. Must be able to verify `kube-apiserver --tls-cert-file`
--service-account-private-key-file	RSA key used to sign ServiceAccount tokens. Must be verifiable by `kube-apiserver --service-account-key-file` or ServiceAccounts will not be able to authenticate

kubelet	What's it for
--client-ca-file	CA bundle used to verify client certificate connections from clients and identify users. (I am Bob). Must be able to verify `kube-apiserver --kubelet-client-certificate` or some endpoints will fail.
--tls-cert-file	Serving cert used to serve requests. Must be verifiable by `kube-apiserver--kubelet-certificate-authority` or some endpoints will fail.
--tls-private-key-file	Serving key used to serve requests
--rotate-certificates	If true, get client cert/key pairs for authentication to kube-apiserver from the CSR API. This interacts with `kube-apiserver --client-ca-file` and `kube-controller-manager --cluster-signing-cert-file`.
--rotate-server-certificates	If true, get serving cert/key pairs (--tls-cert-file/--tls-private-key-file) from the CSR API. This interacts with `kube-apiserver--kubelet-certificate-authority` and `kube-controller-manager --cluster-signing-cert-file`.

certs: broad strokes

- **etcd** - verify the etcd-server, identify the kube-apiserver
- **kube-apiserver serving** - SNI and default, verify from a pod
- **client mTLS** - derive identity from a request
    - End users
    - kubelets
- **front proxy** - proxy aggregated API requests and recognize them
- **kubelet serving** - both serving and verification from kube-apiserver
- **kubelet client mTLS** - identify the kube-apiserver and recognize it
- **SA tokens** - sign and verify JWT
- **CSR** - drives kubelet related certs

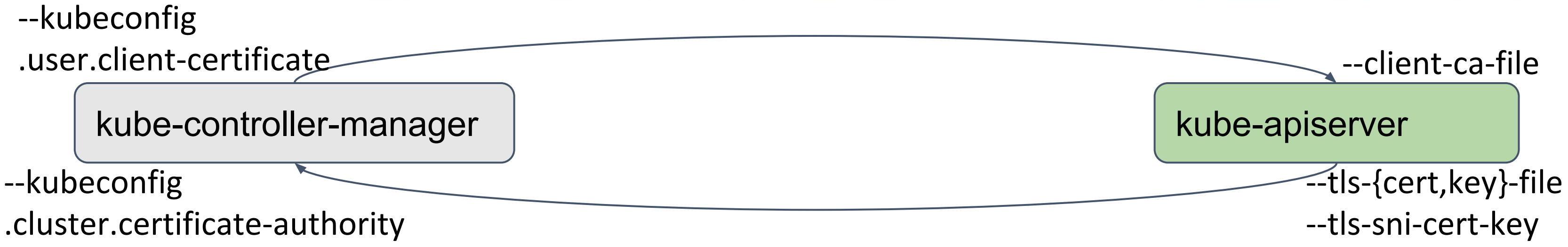
certs: super basic mTLS



Reminders

- Certificates are signed by issuers
- CA bundles contain every valid issuer and possibly its chain
- Rotate by expanding trust first

certs: simple rotation



To rotate...

1. create a new signing certificate
2. --client-ca-file expands trust for new signer
    a. restart all kube-apiservers
3. sign a new client cert and place in --kubeconfig .user.client-certificate
    a. restart all kube-controller-managers
4. (optional), tighten trust in --client-ca-file

certs: manage them automatically

- Relationships are hard
  - kubelet --rotate-certificates --rotate-server-certificates
  - kube-controller-manager --cluster-signing-cert-file --cluster-signing-key-file
  - kube-apiserver --kubelet-certificate-authority --client-ca-file

- Cluster-admins care about 2 of 25 flags

disaster recovery

- All the machines stopped.
  - Just start them back up and watch it come back without any action.
  - Static pods require no special bootstrapping
- I lost the machines, but I have my backups
  - Take your time making your DNS entries match so certificates and configuration is good
  - Start the kubelet back up and static pods will re-bootstrap