
KubeCon
CloudNativeCon
North America 2023

The Kubernetes Storage Layer: Peeling The Onion Minus The Tears

Madhav Jivrajani, VMware

$ whoami

- Work @ VMware
- Do work in API Machinery, Scalability, Architecture and ContribEx
- TL for SIG ContribEx and GitHub Admin of the project

Before We Start...

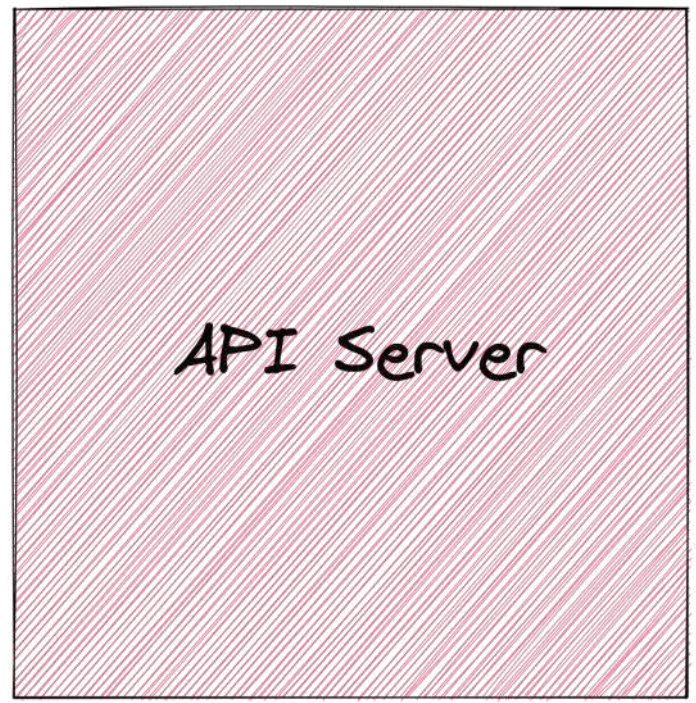
🚨 Help migrate Prow jobs to community clusters!

See https://github.com/kubernetes/test-infra/issues/29722 for details.

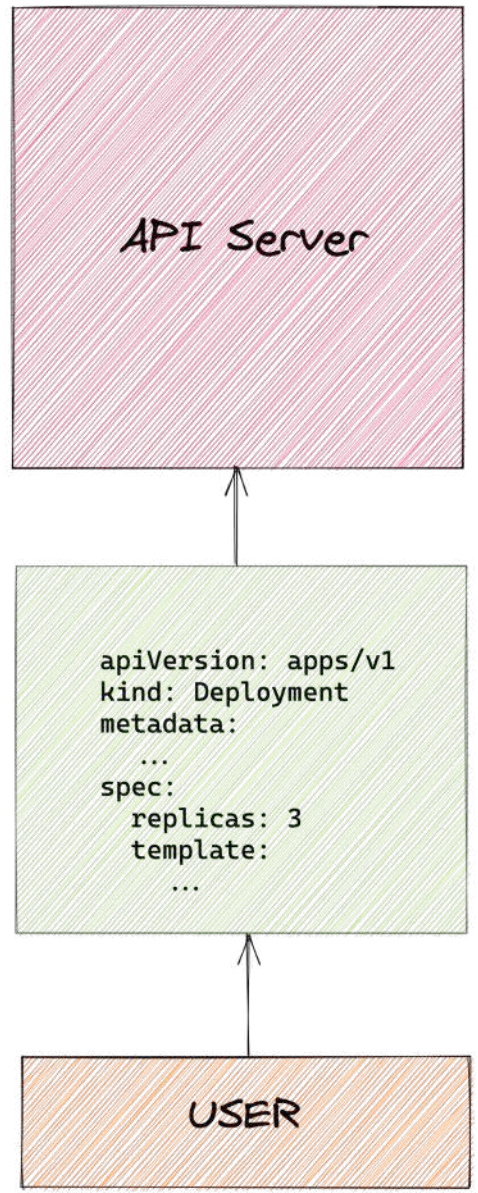
Prelude

A 50,000 ft. view of how the Kubernetes “machine” works.

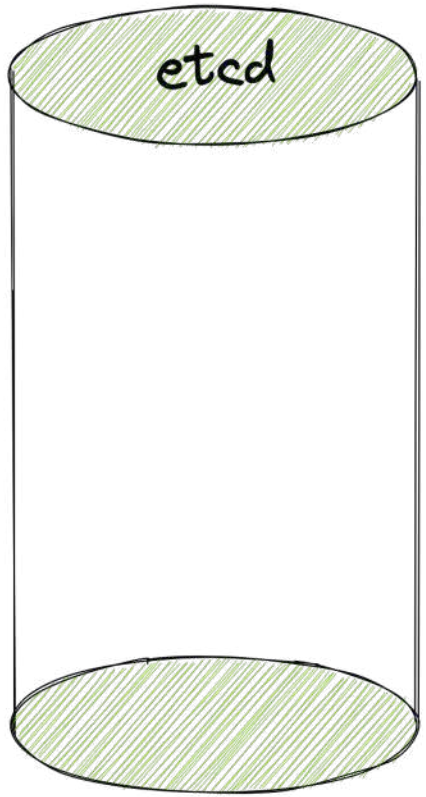
The K8s Machine



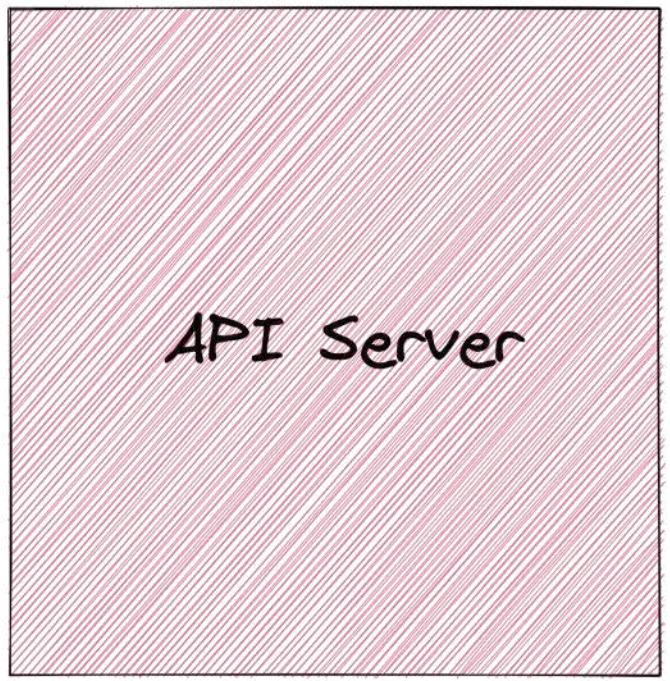
The K8s Machine

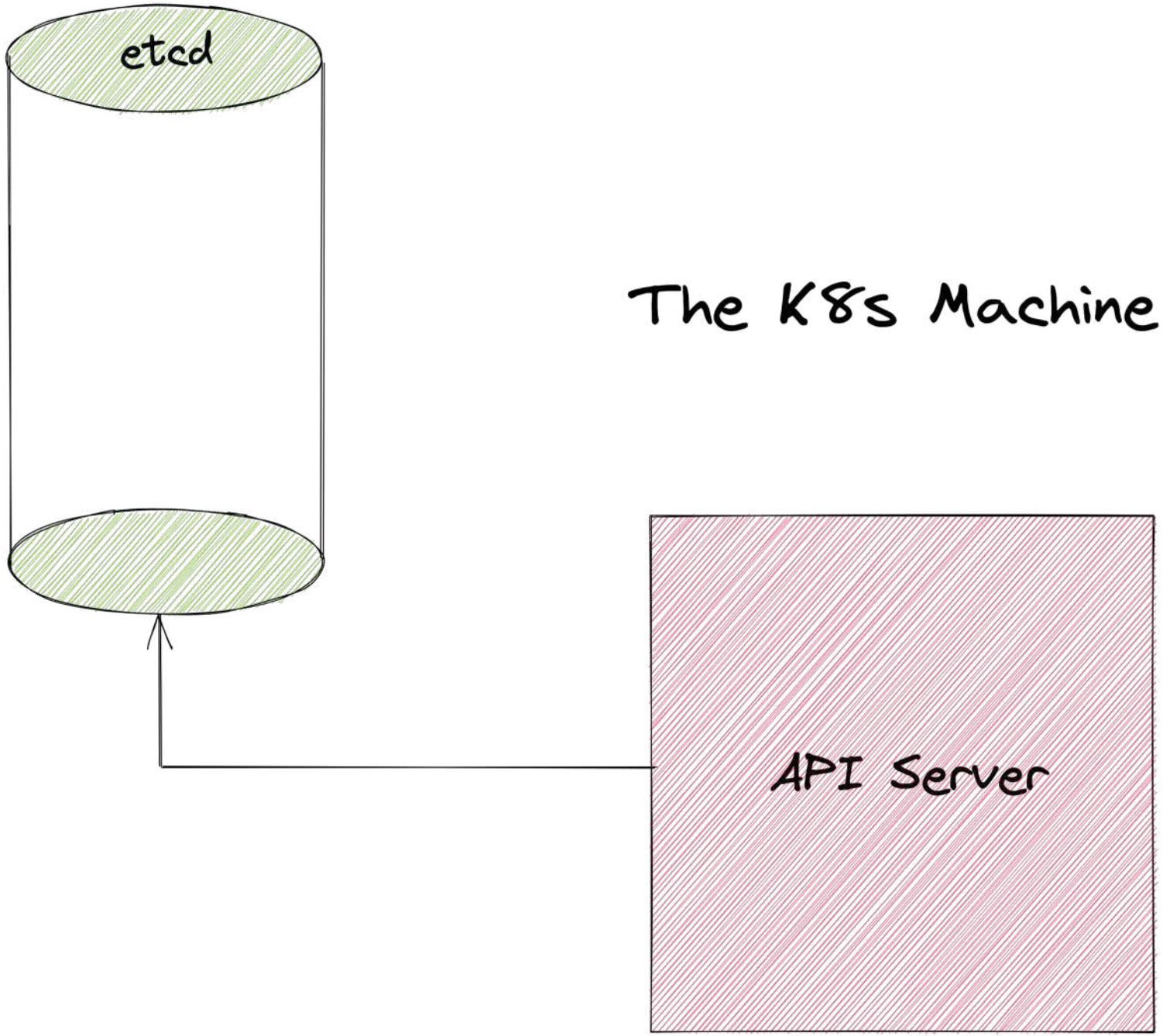


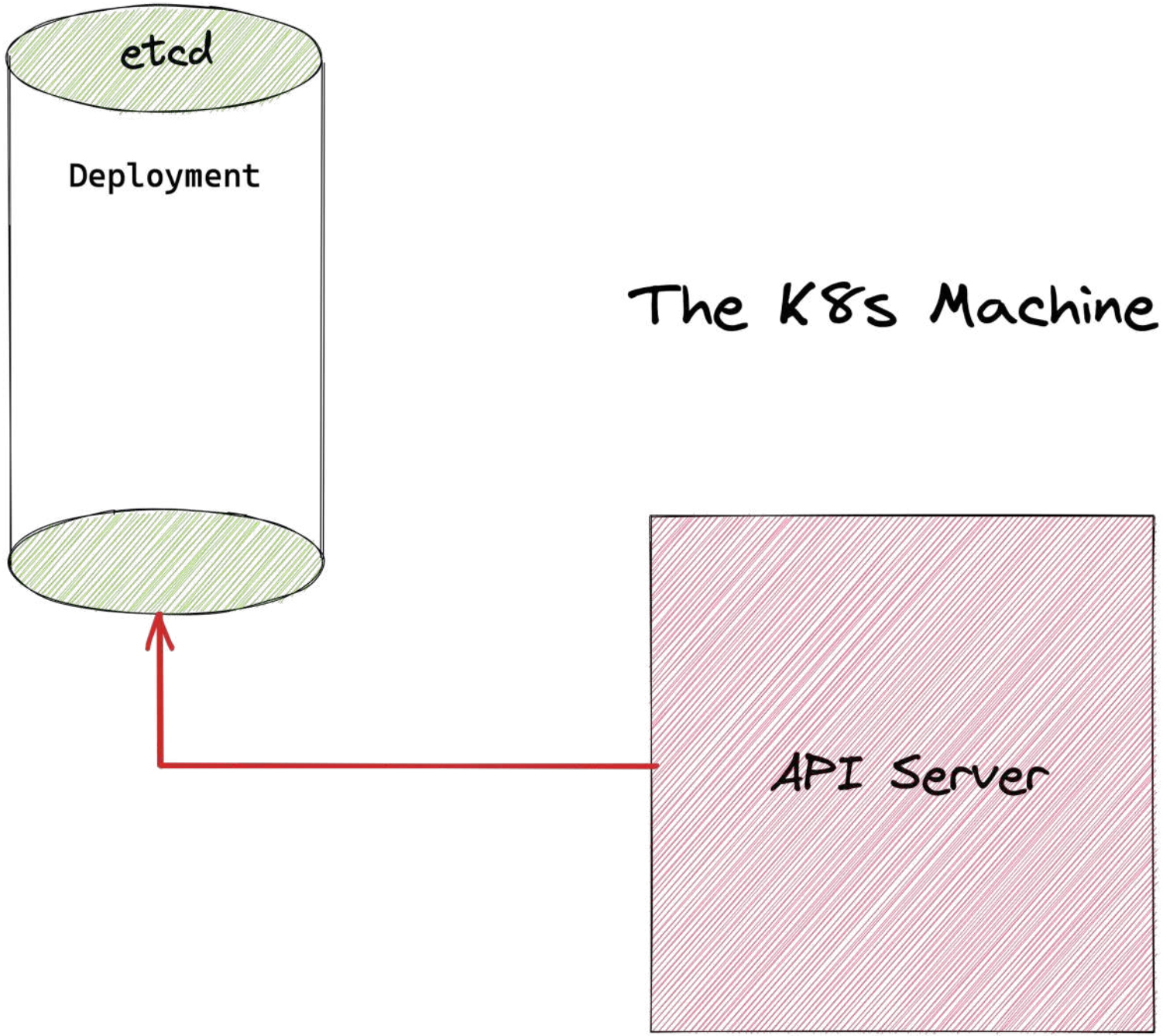

```
kubectl apply -f deployment.yaml
```



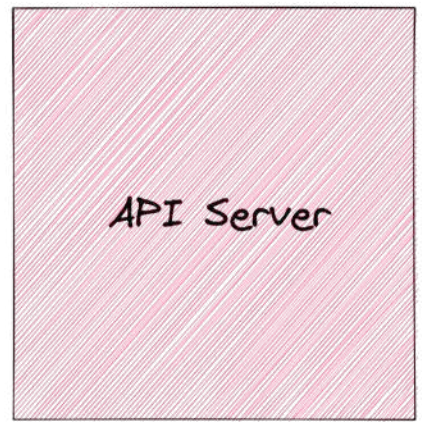
The K8s Machine

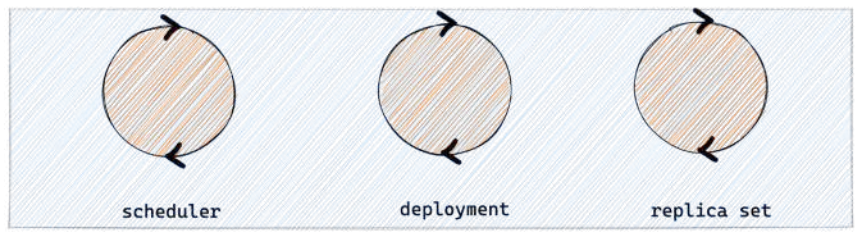



etcd
The K8s Machine
API Server


etcd
Deployment
The K8s Machine
API Server

The K8s Machine

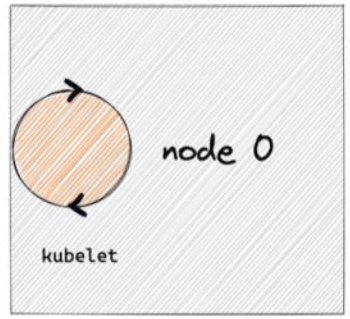


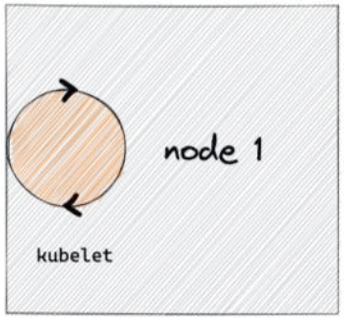


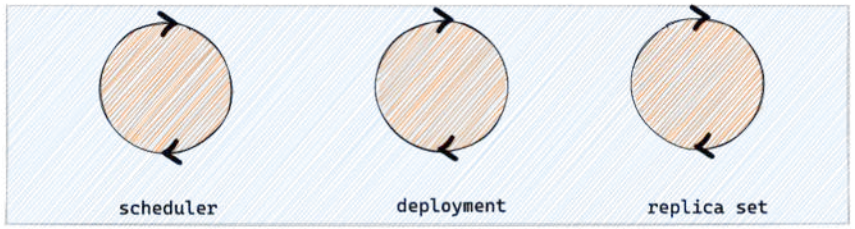
Controllers

The K8s Machine

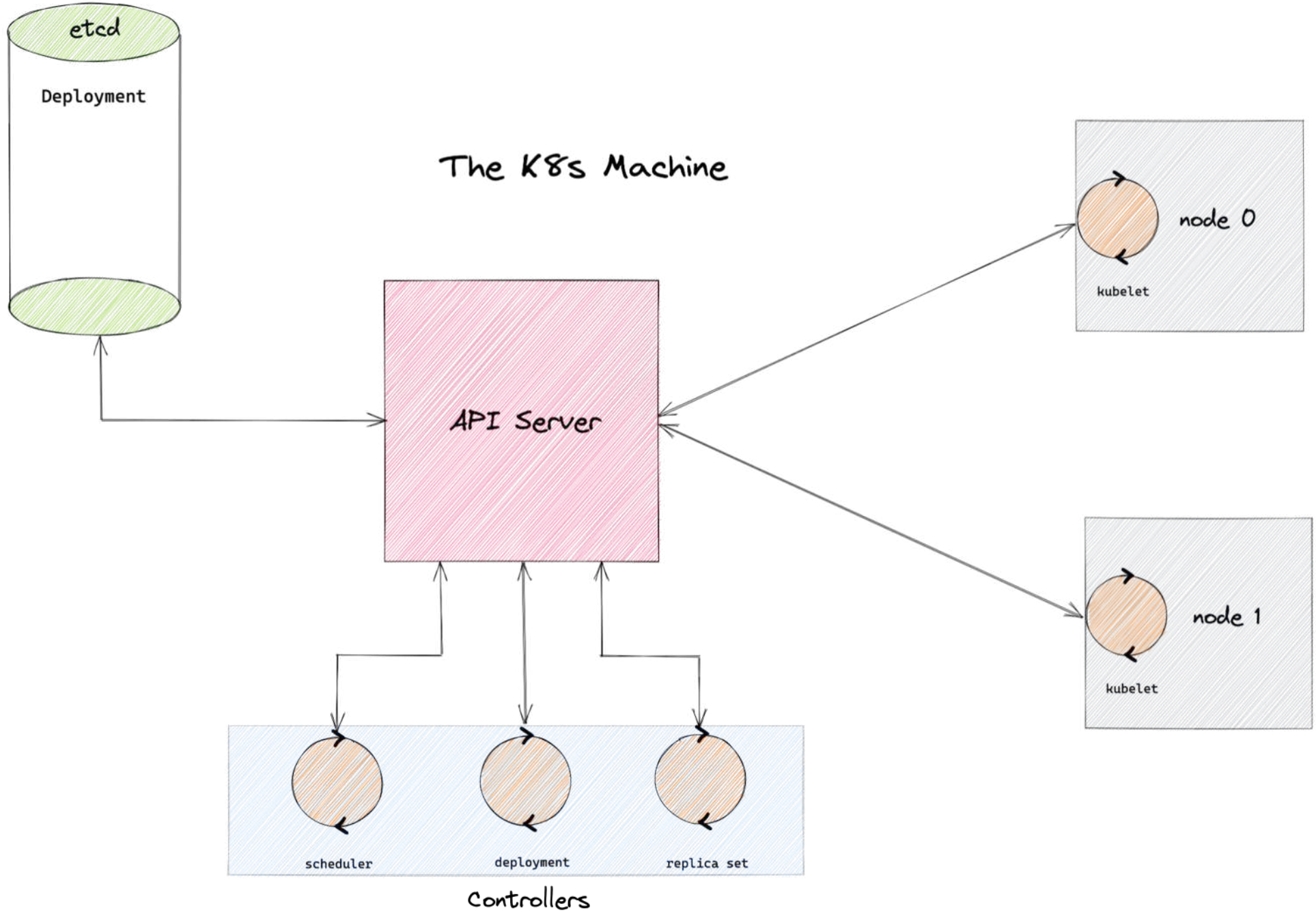
API Server

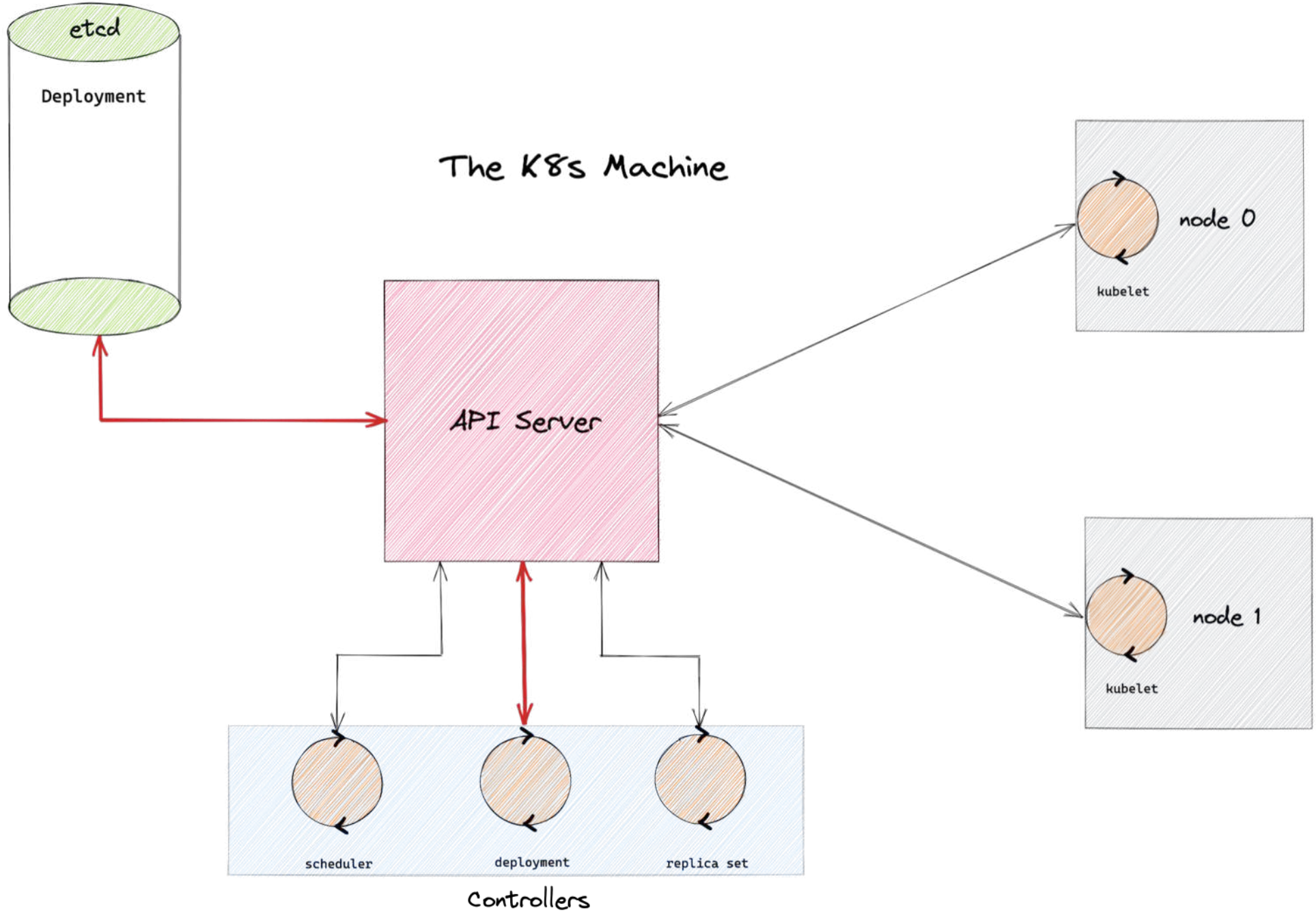


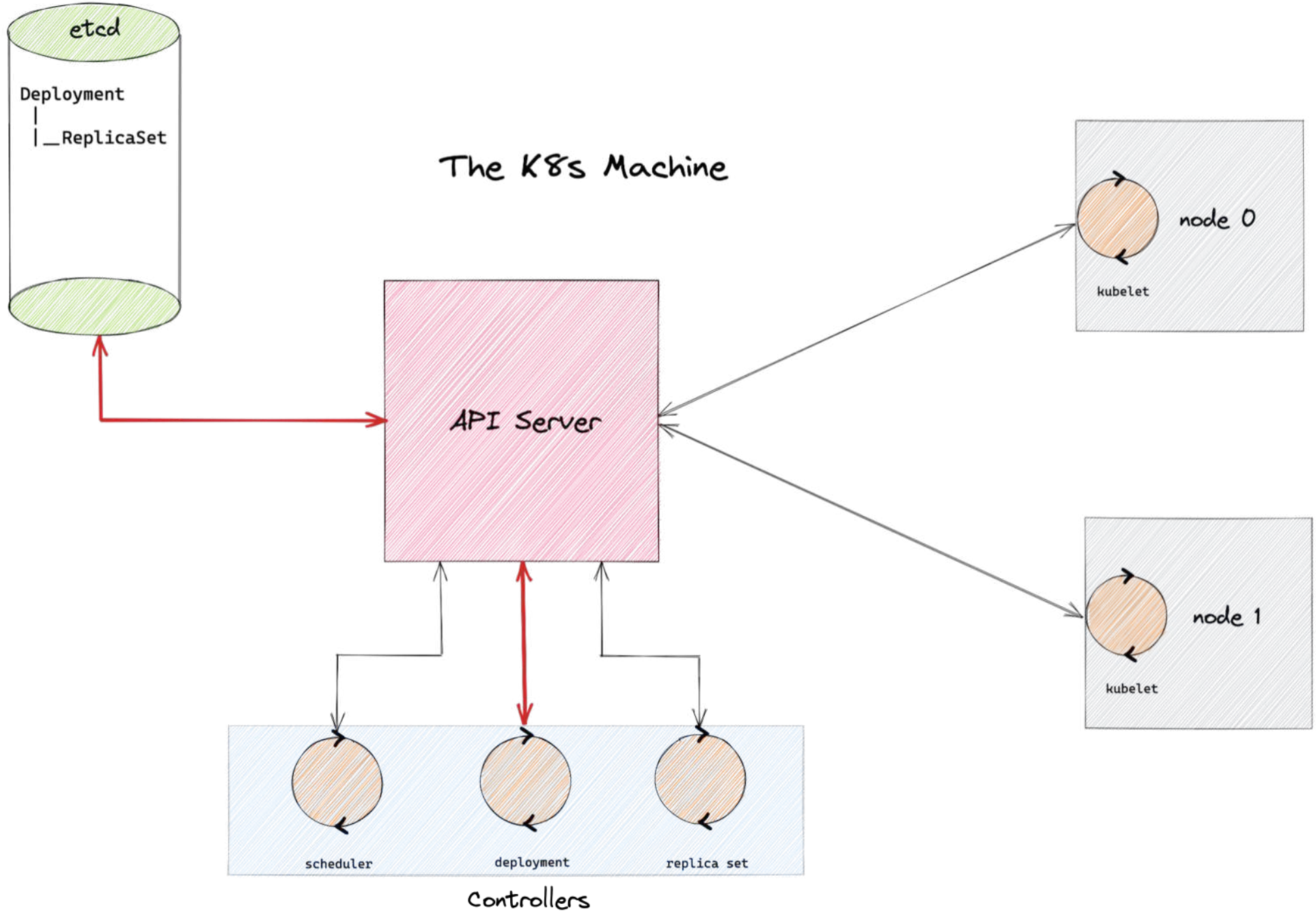


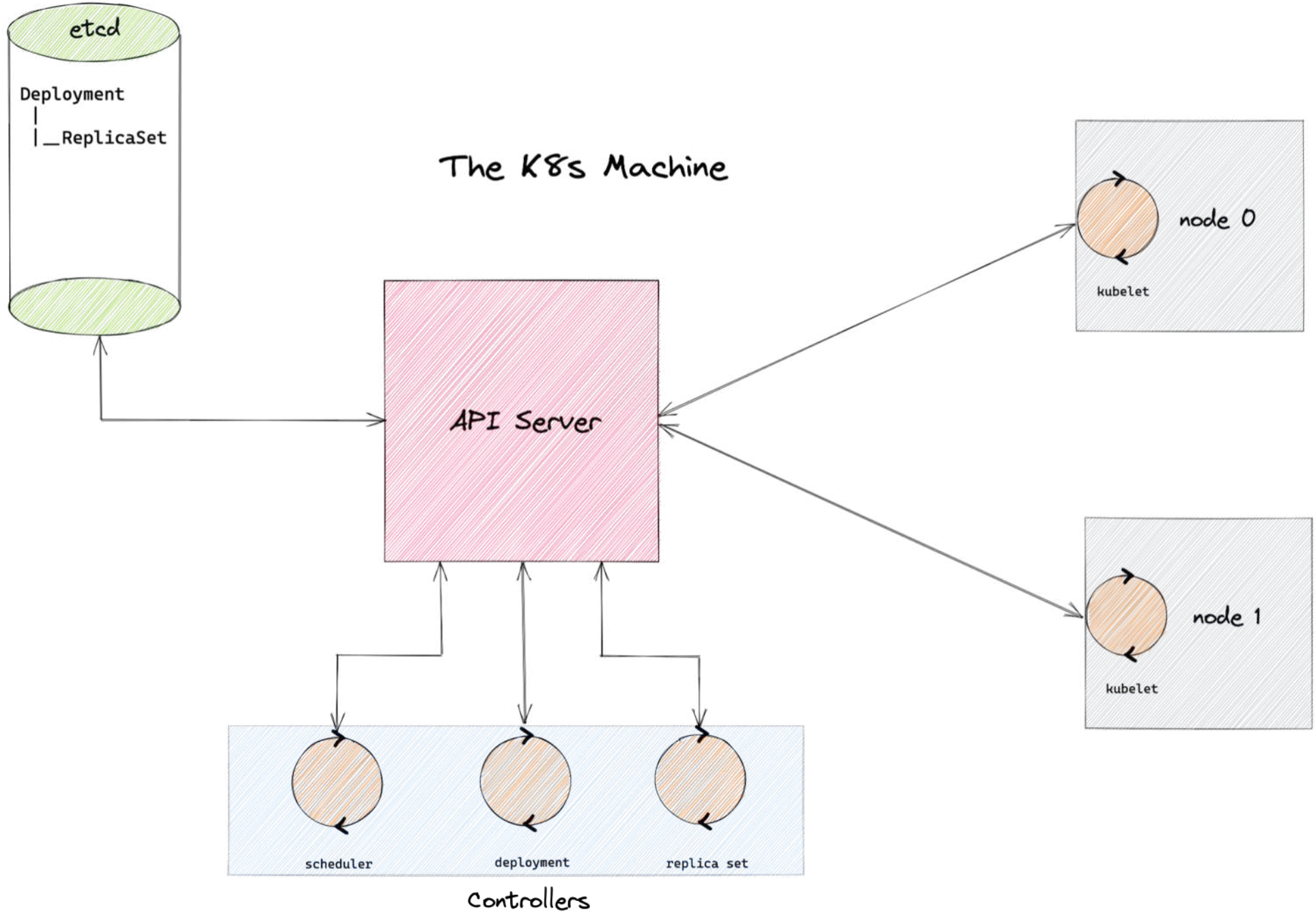


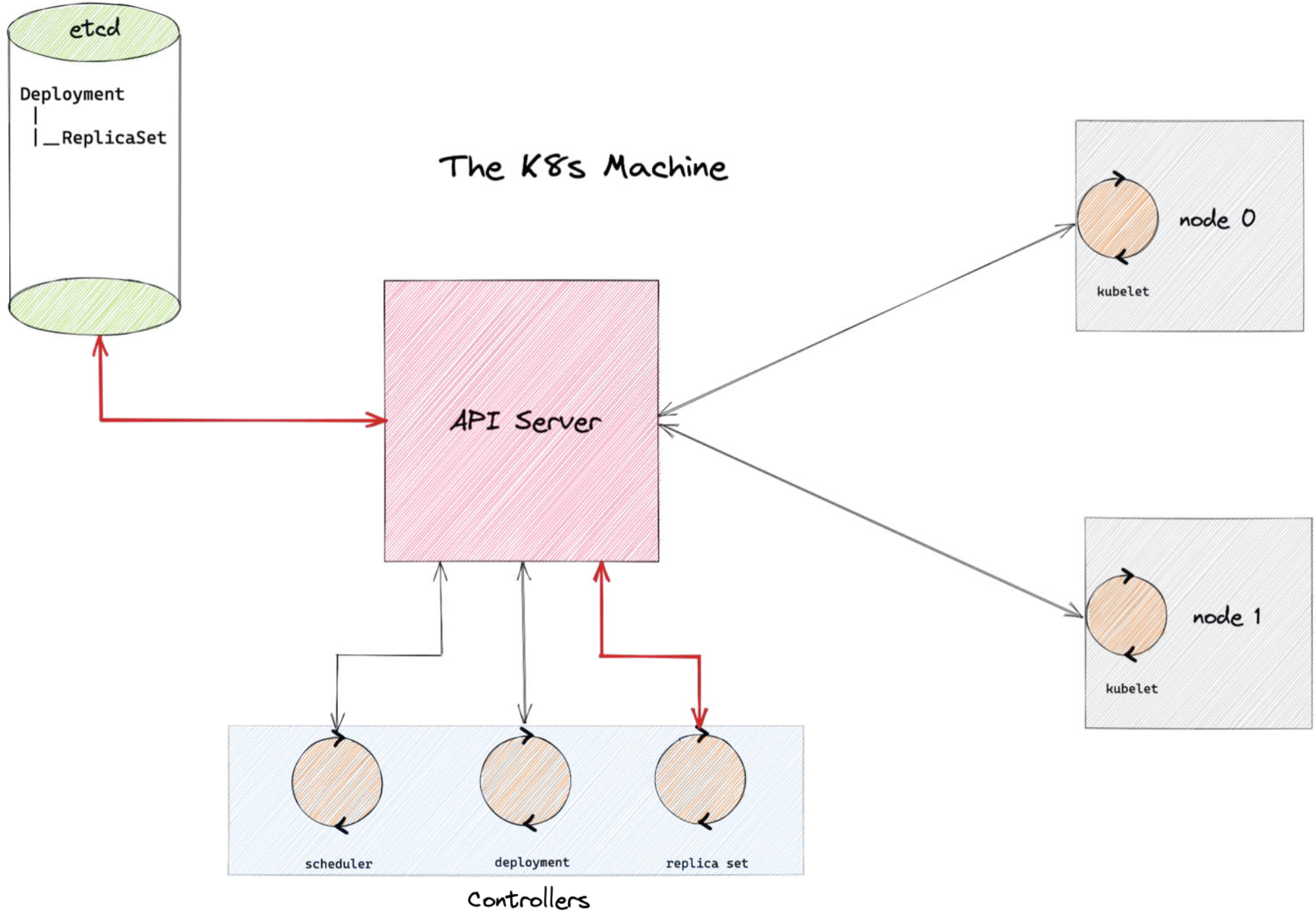
Controllers

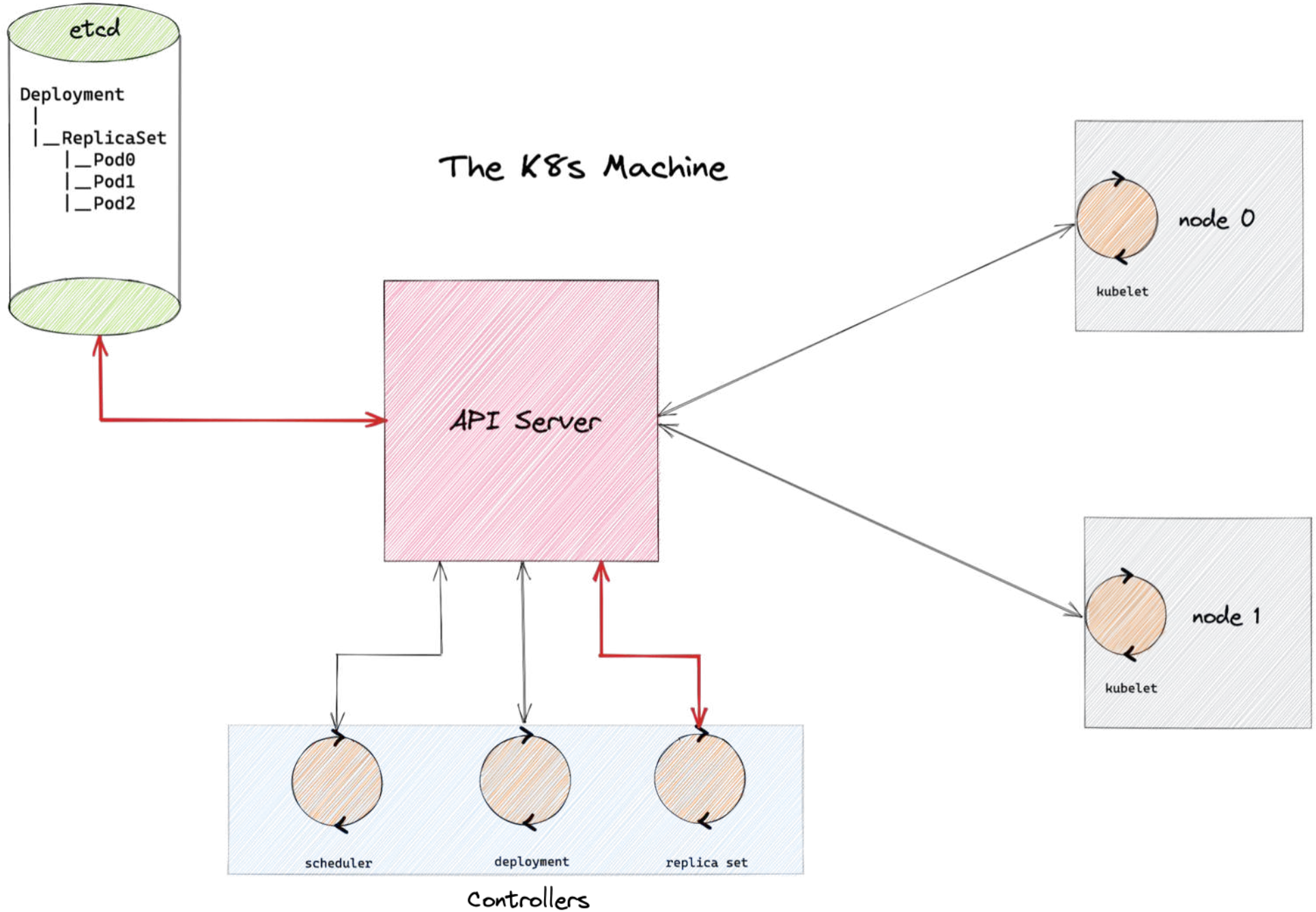

etcd
Deployment
The K8s Machine
API Server
node 0
kubelet
node 1
kubelet
scheduler
deployment
replica set
Controllers

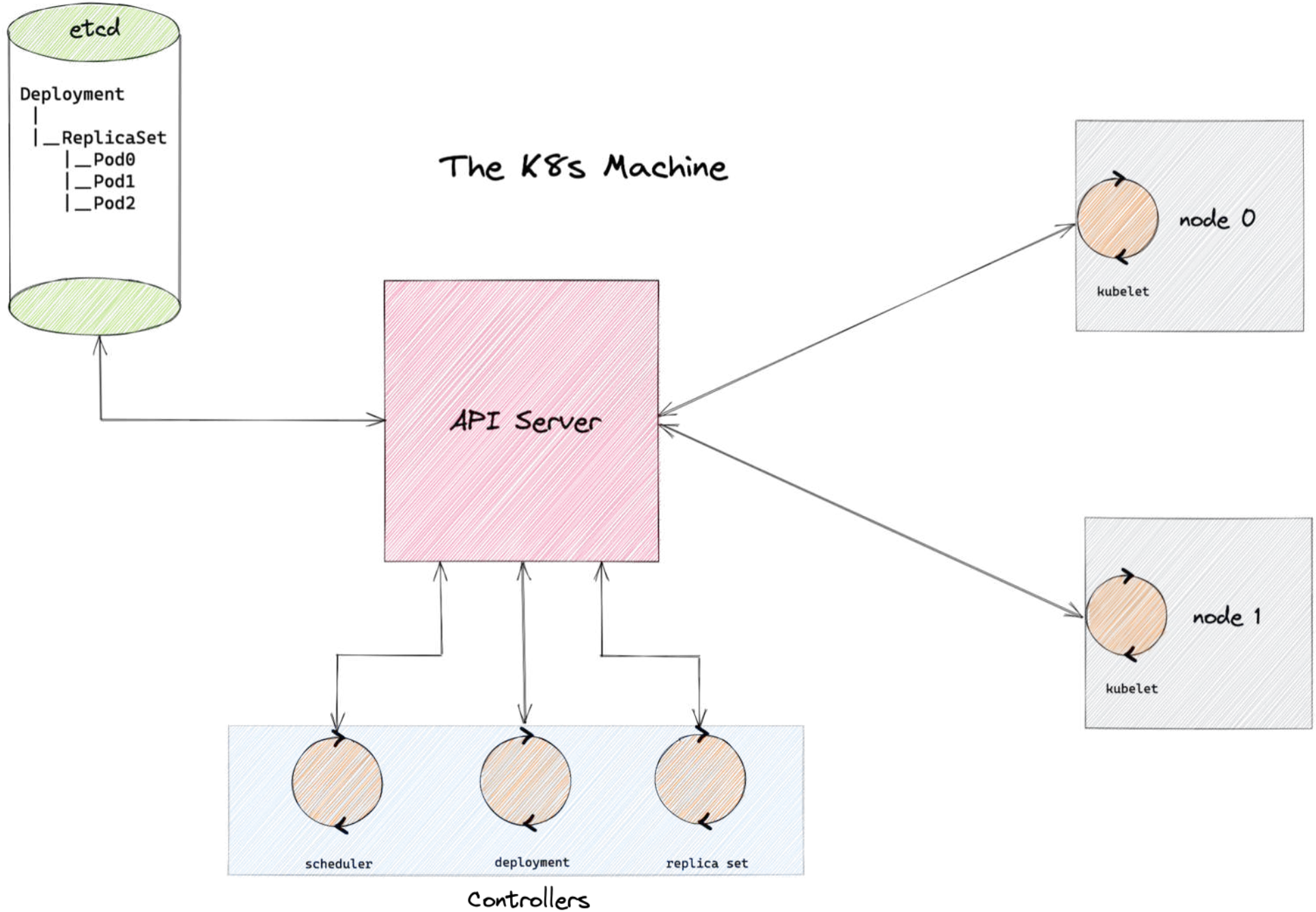

etcd
Deployment
The K8s Machine
API Server
node 0
kubelet
node 1
kubelet
scheduler
deployment
replica set
Controllers

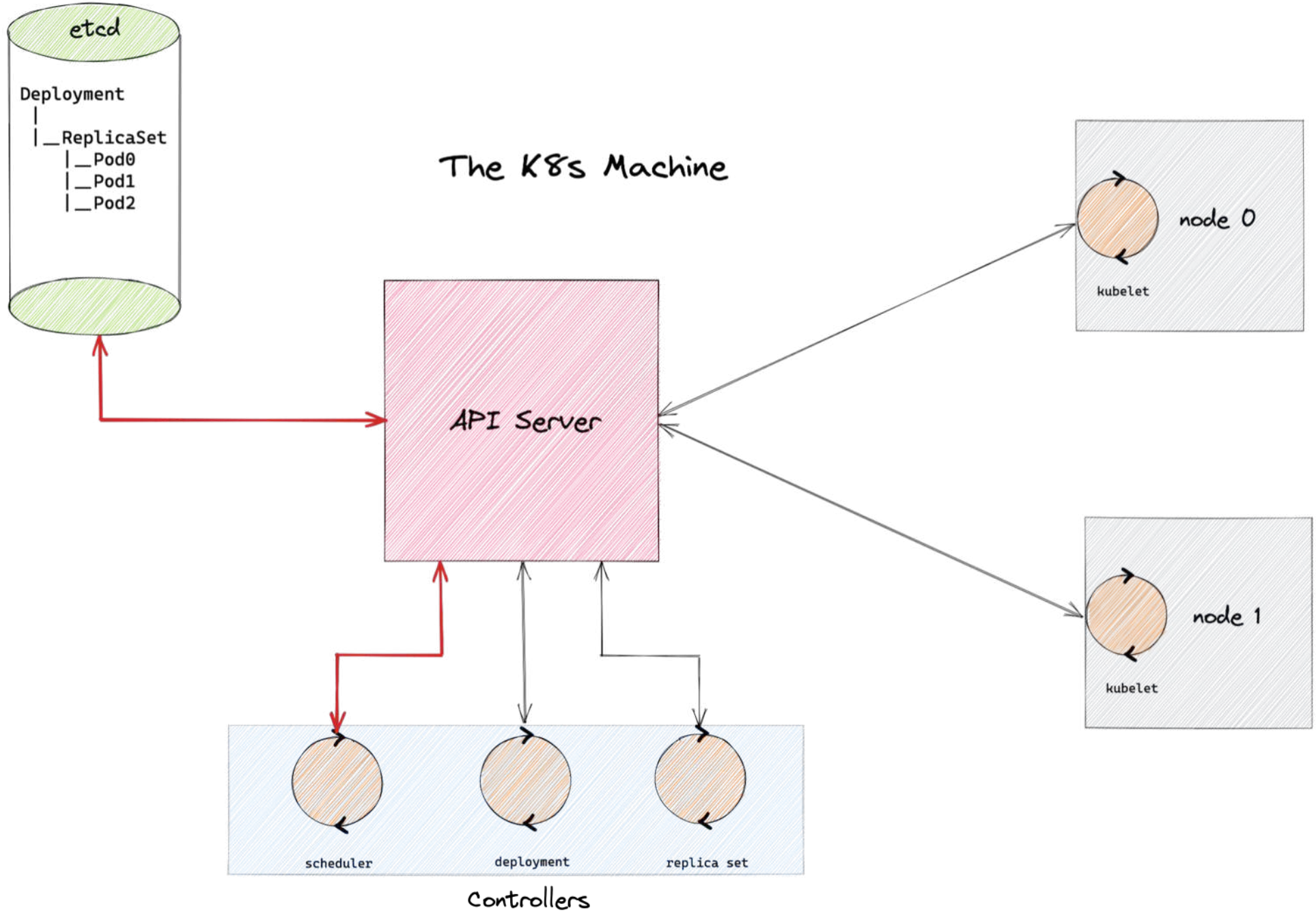

etcd
Deployment
|
|__ReplicaSet
The K8s Machine
API Server
node 0
kubelet
node 1
kubelet
scheduler
deployment
replica set
Controllers

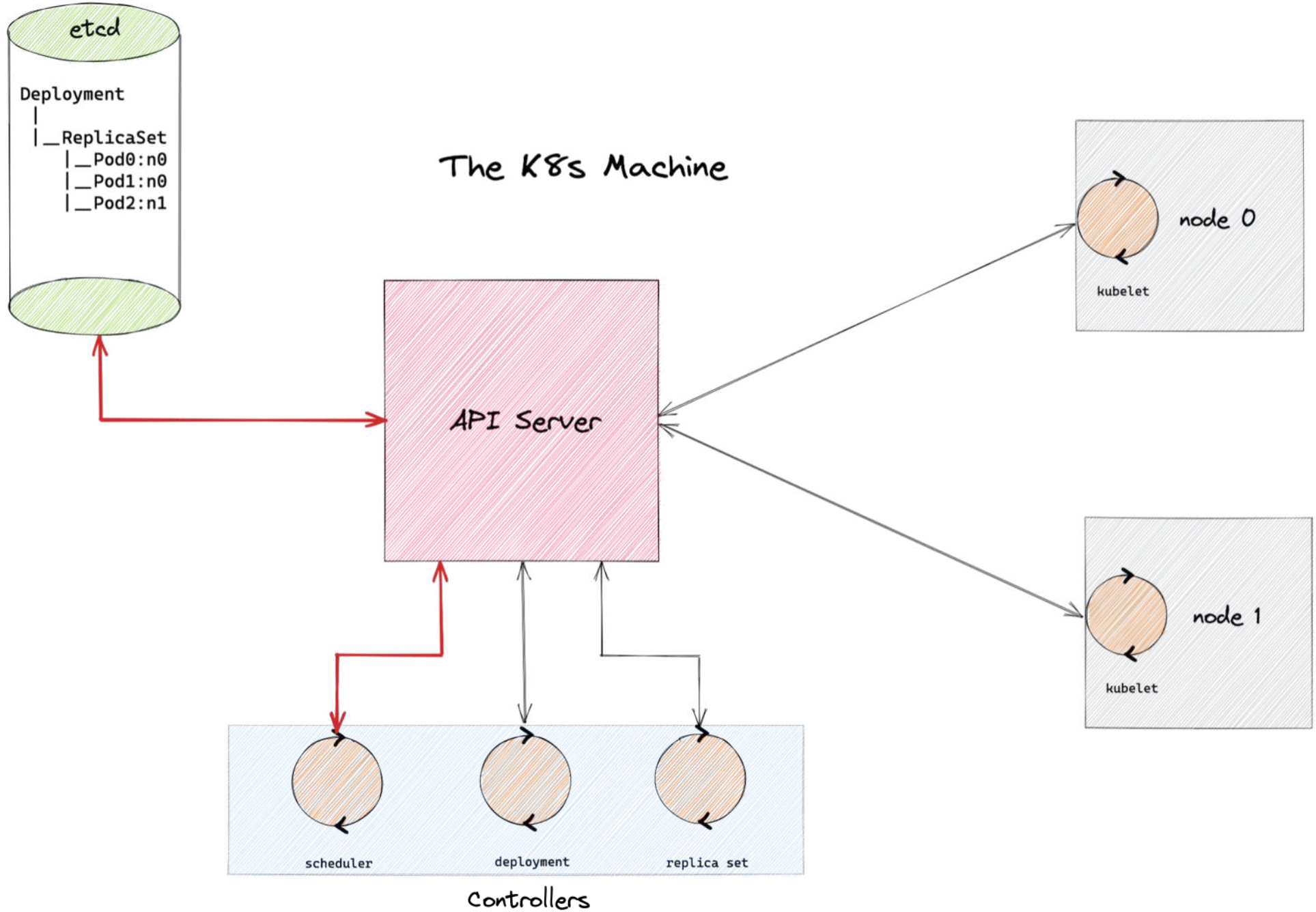

etcd
Deployment
|
|__ReplicaSet
The K8s Machine
API Server
node 0
kubelet
node 1
kubelet
scheduler
deployment
replica set
Controllers

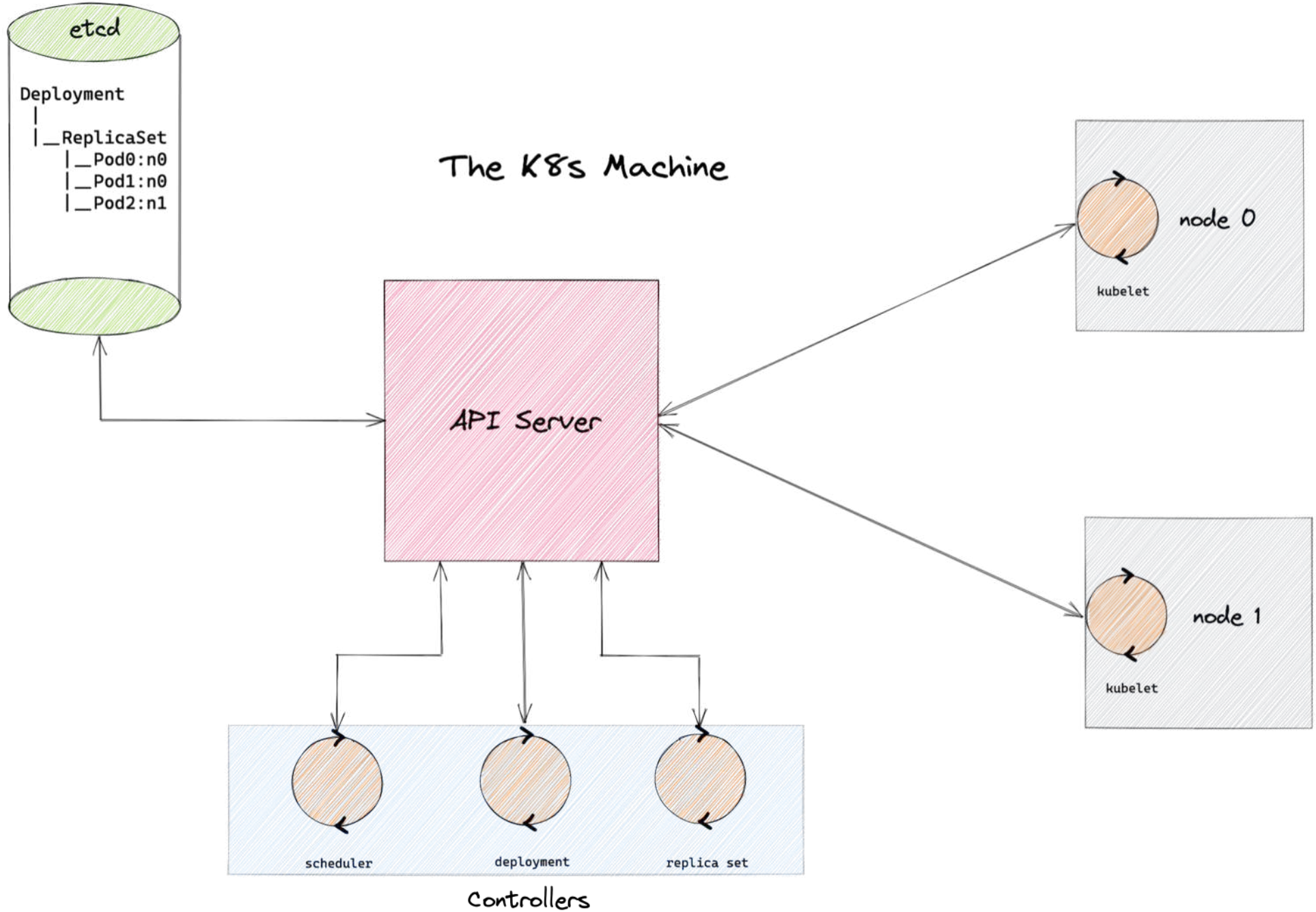

etcd
Deployment
|
|__ReplicaSet
The K8s Machine
API Server
node 0
kubelet
node 1
kubelet
scheduler
deployment
replica set
Controllers

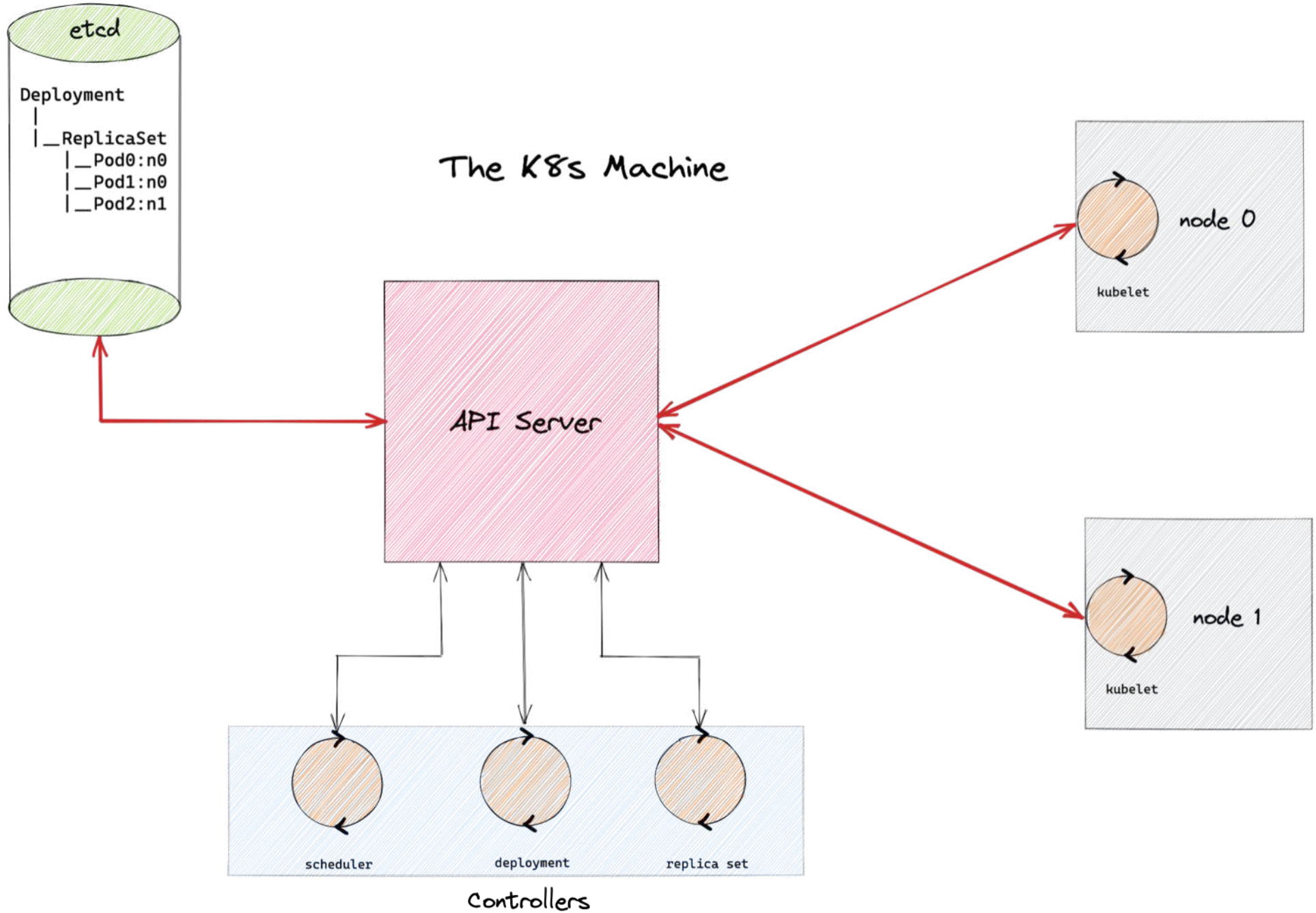

etcd
Deployment
|
|__ReplicaSet
|__Pod0
|__Pod1
|__Pod2
The K8s Machine
API Server
node 0
kubelet
node 1
kubelet
scheduler
deployment
replica set
Controllers

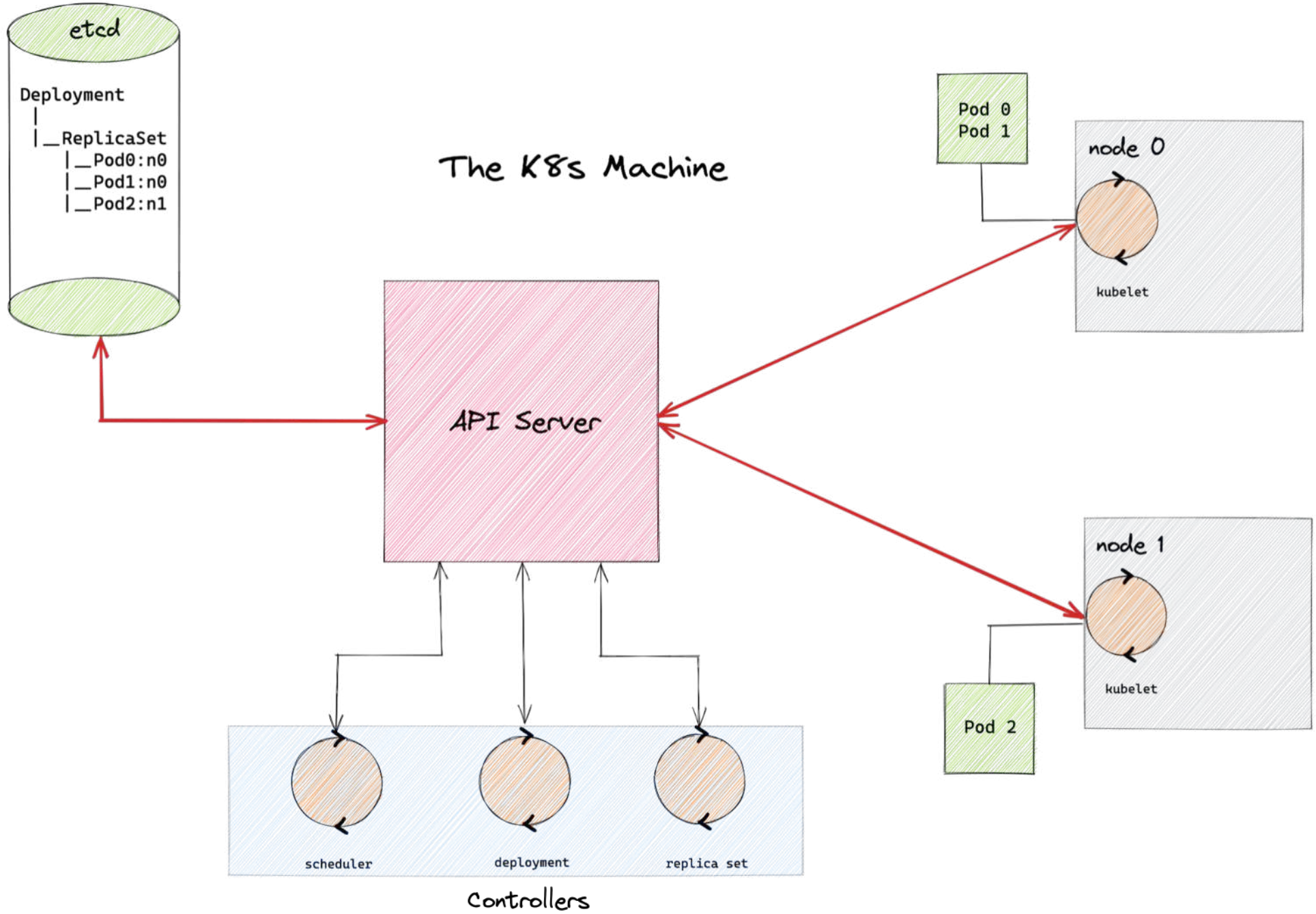

etcd
Deployment
|
|__ReplicaSet
|__Pod0
|__Pod1
|__Pod2
The K8s Machine
API Server
node 0
kubelet
node 1
kubelet
scheduler
deployment
replica set
Controllers

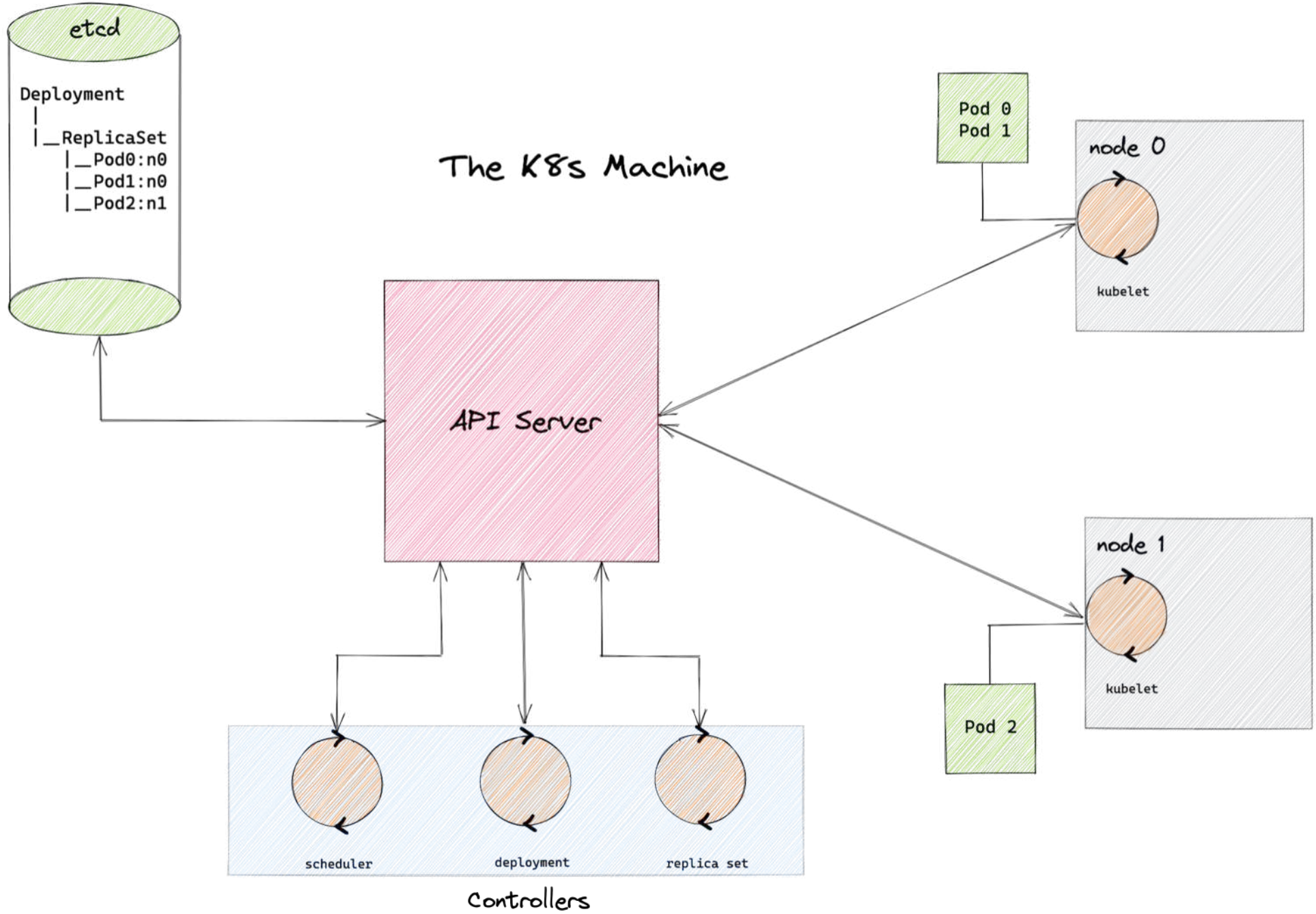

etcd
Deployment
|
|_ReplicaSet
|_Pod0
|_Pod1
|_Pod2
The K8s Machine
API Server
node 0
kubelet
node 1
kubelet
scheduler
deployment
replica set
Controllers

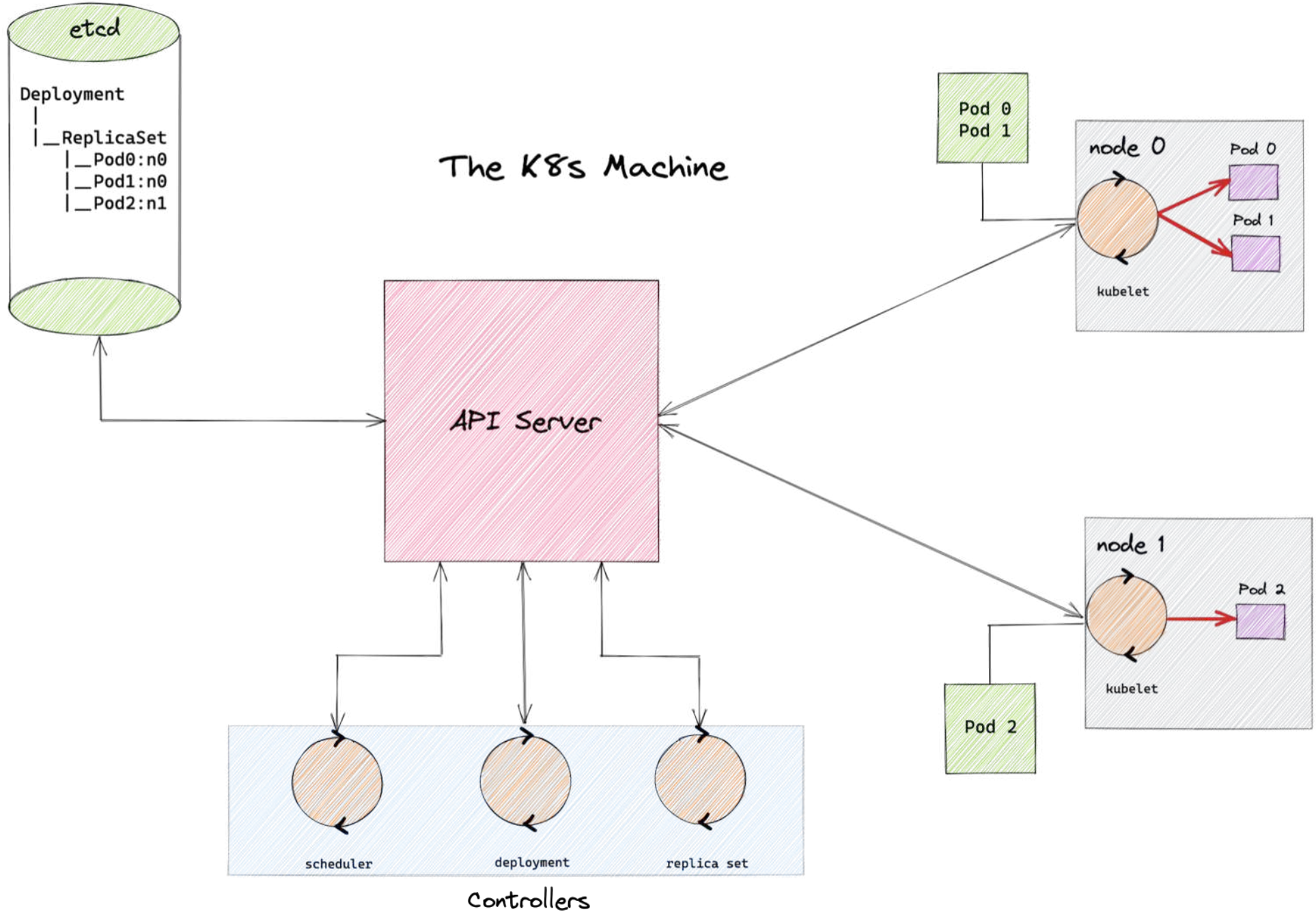

etcd
Deployment
|
|_ReplicaSet
|_Pod0:n0
|_Pod1:n0
|_Pod2:n1
The K8s Machine
API Server
node 0
kubelet
node 1
kubelet
scheduler
deployment
replica set
Controllers

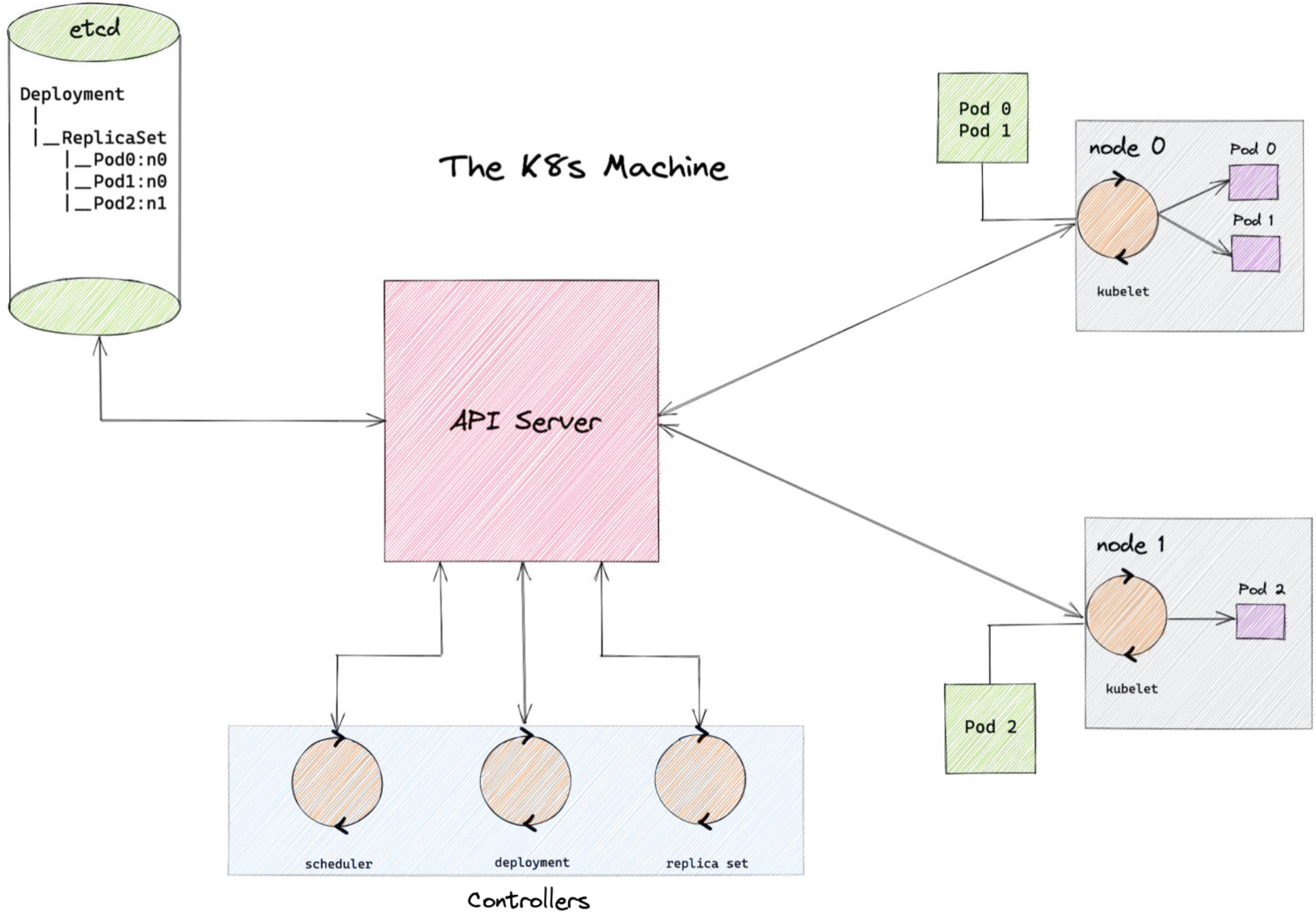

etcd
Deployment
|
|_ReplicaSet
|_Pod0:n0
|_Pod1:n0
|_Pod2:n1
The K8s Machine
API Server
node 0
kubelet
node 1
kubelet
scheduler
deployment
replica set
Controllers


etcd
Deployment
|
|_ReplicaSet
|_Pod0:n0
|_Pod1:n0
|_Pod2:n1
The K8s Machine
API Server
node 0
kubelet
node 1
kubelet
scheduler
deployment
replica set
Controllers


etcd
Deployment
|
|_ReplicaSet
|_Pod0:n0
|_Pod1:n0
|_Pod2:n1
The K8s Machine
API Server
Pod 0
Pod 1
node 0
kubelet
node 1
kubelet
Pod 2
scheduler
deployment
replica set
Controllers


etcd
Deployment
|
|_ReplicaSet
|_Pod0:n0
|_Pod1:n0
|_Pod2:n1
The K8s Machine
API Server
Pod 0
Pod 1
node 0
kubelet
node 1
kubelet
Pod 2
scheduler
deployment
replica set
Controllers


etcd
Deployment
|
|_ReplicaSet
|_Pod0:n0
|_Pod1:n0
|_Pod2:n1
The K8s Machine
Pod 0
Pod 1
node 0
Pod 0
Pod 1
kubelet
API Server
node 1
Pod 2
kubelet
Pod 2
scheduler
deployment
replica set
Controllers


etcd
Deployment
|
|_ReplicaSet
|_Pod0:n0
|_Pod1:n0
|_Pod2:n1
The K8s Machine
Pod 0
Pod 1
node 0
Pod 0
Pod 1
kubelet
API Server
node 1
Pod 2
kubelet
Pod 2
scheduler
deployment
replica set
Controllers

List + Watch

List + Watch

“Kubernetes is a declarative, event-driven system.”

List + Watch

“Kubernetes is a declarative, event-driven system.”

List

“Kubernetes is a declarative, event-driven system.”

List

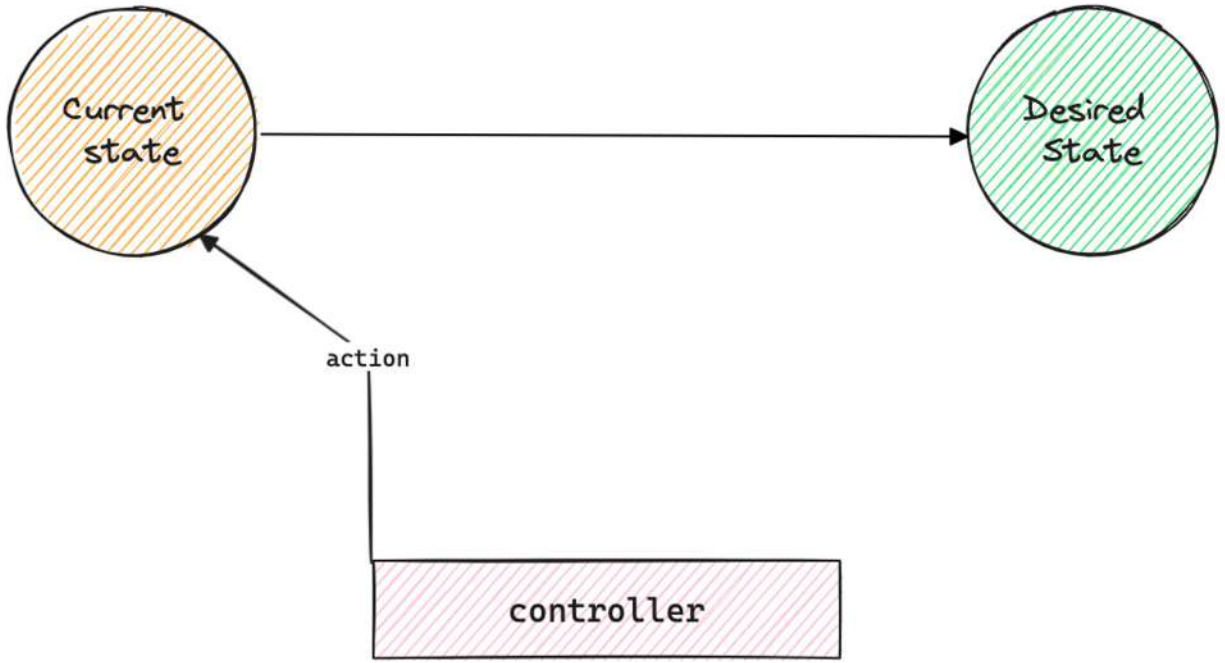
“Kubernetes is a declarative, event-driven system.”

We specify intent.

```
❯ kubectl apply -f 3-replica-deployment.yaml
```

List

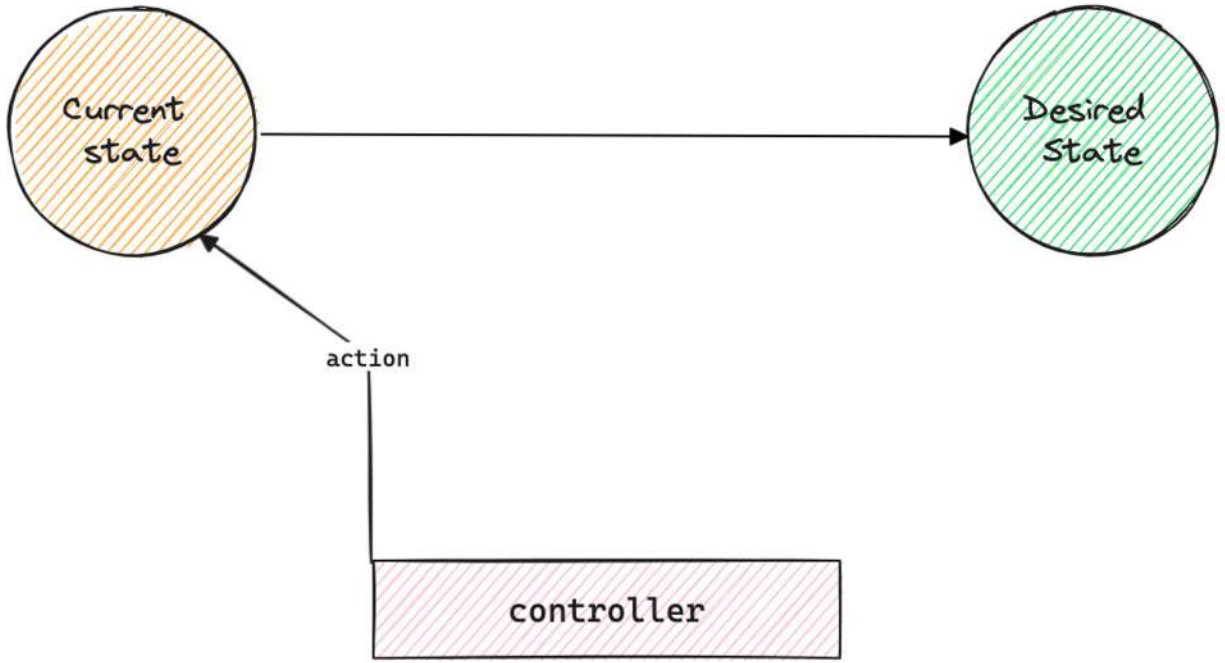
"Kubernetes is a declarative, event-driven system."



List

"Kubernetes is a declarative, event-driven system."

- We need to start somewhere, in order to take actions, we need to know what the "current state" looks like.



List

"Kubernetes is a declarative, event-driven system."

- We need to start somewhere, in order to take actions, we need to know what the "current state" looks like.
- To do this, we perform a **`LIST`** operation.

```
❯ kubectl get --raw
'/api/v1/namespaces/default/pods'
```

```
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion":"1452",
    ...
  },
  "items": [...] // all pods
}
```

List

"Kubernetes is a declarative, event-driven system."

- In order to get the "current state", we perform a **`LIST`** operation.
- Responses can get huge, sometimes we paginate.

```
› kubectl get --raw
'/api/v1/namespaces/default/pods?limit=100'

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion":"1452",
    "continue": "ENCODED_CONTINUE_TOKEN",
    ...
  },
  "items": [...] // pod0-pod99
}
```

List

"Kubernetes is a declarative, event-driven system."

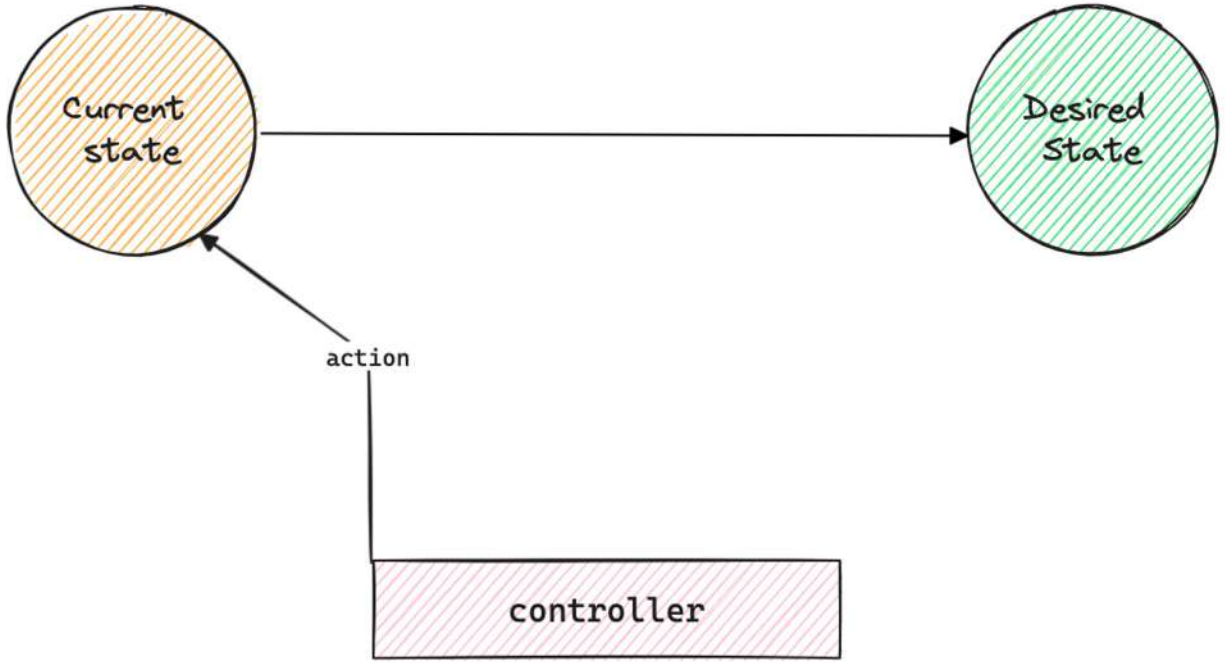
- In order to get the "current state", we perform a **`LIST`** operation.
- Responses can get huge, sometimes we paginate.
- We can continue doing this till we get the entire "current state" (full list).

```
› kubectl get --raw
'/api/v1/namespaces/default/pods?limit=100&cont
inue=ENCODED_CONTINUE_TOKEN'
```

```
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion":"1452",
    "continue": "ENCODED_CONTINUE_TOKEN_2",
    ...
  },
  "items": [...] // pod100-pod199
}
```

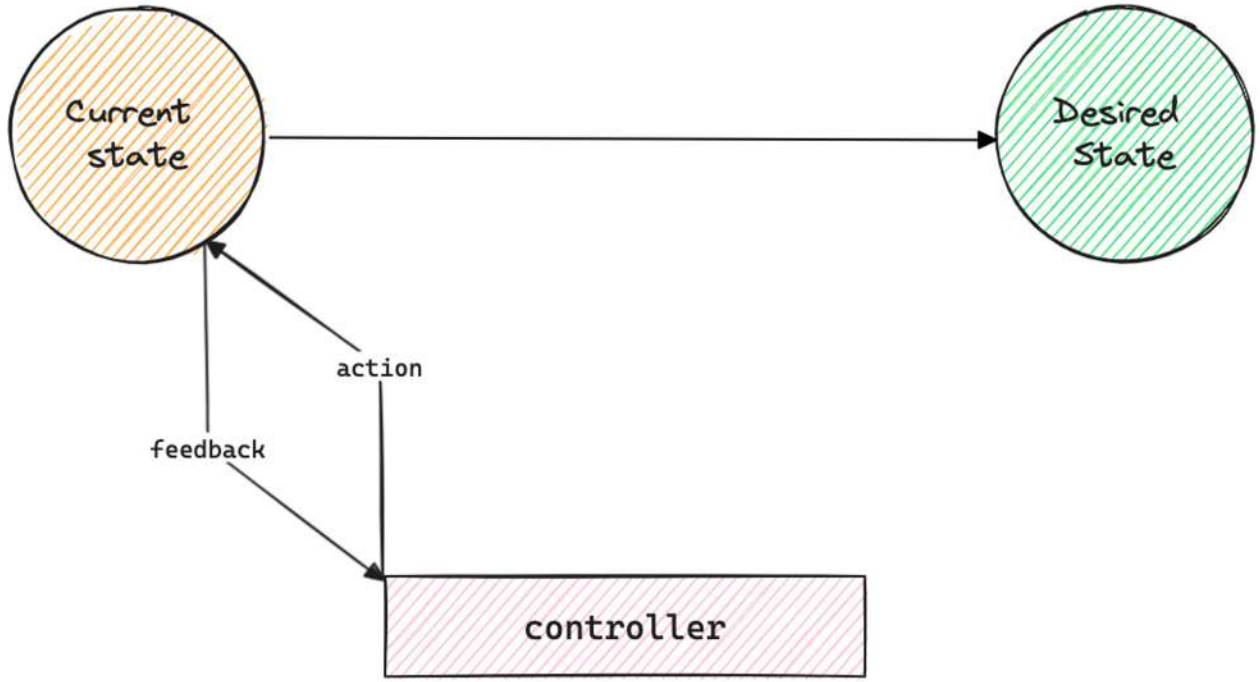
Watch

"Kubernetes is a declarative, event-driven system."



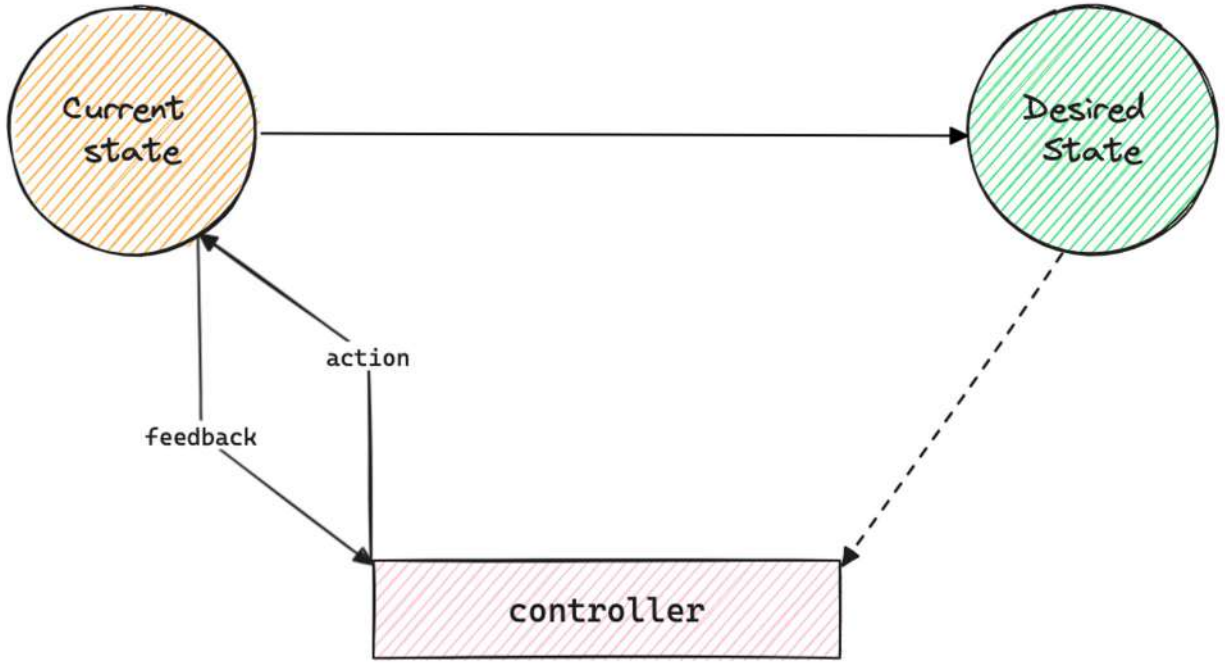
Watch

"Kubernetes is a declarative, event-driven system."



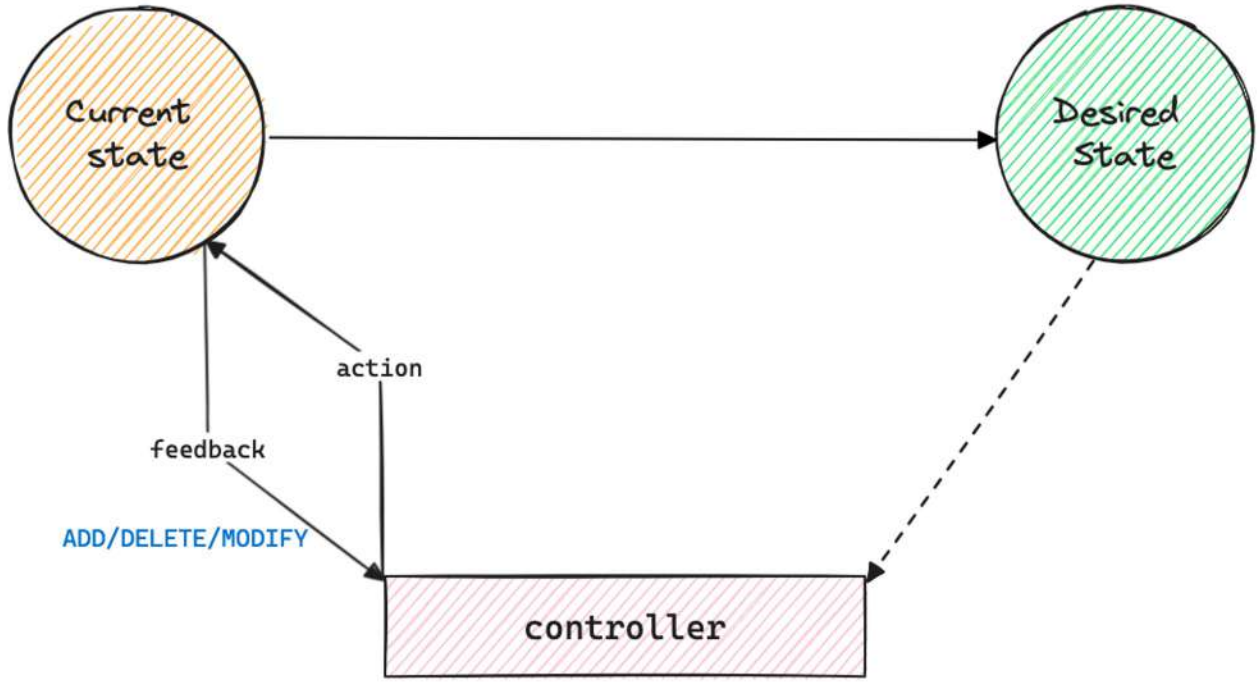
Watch

"Kubernetes is a declarative, event-driven system."



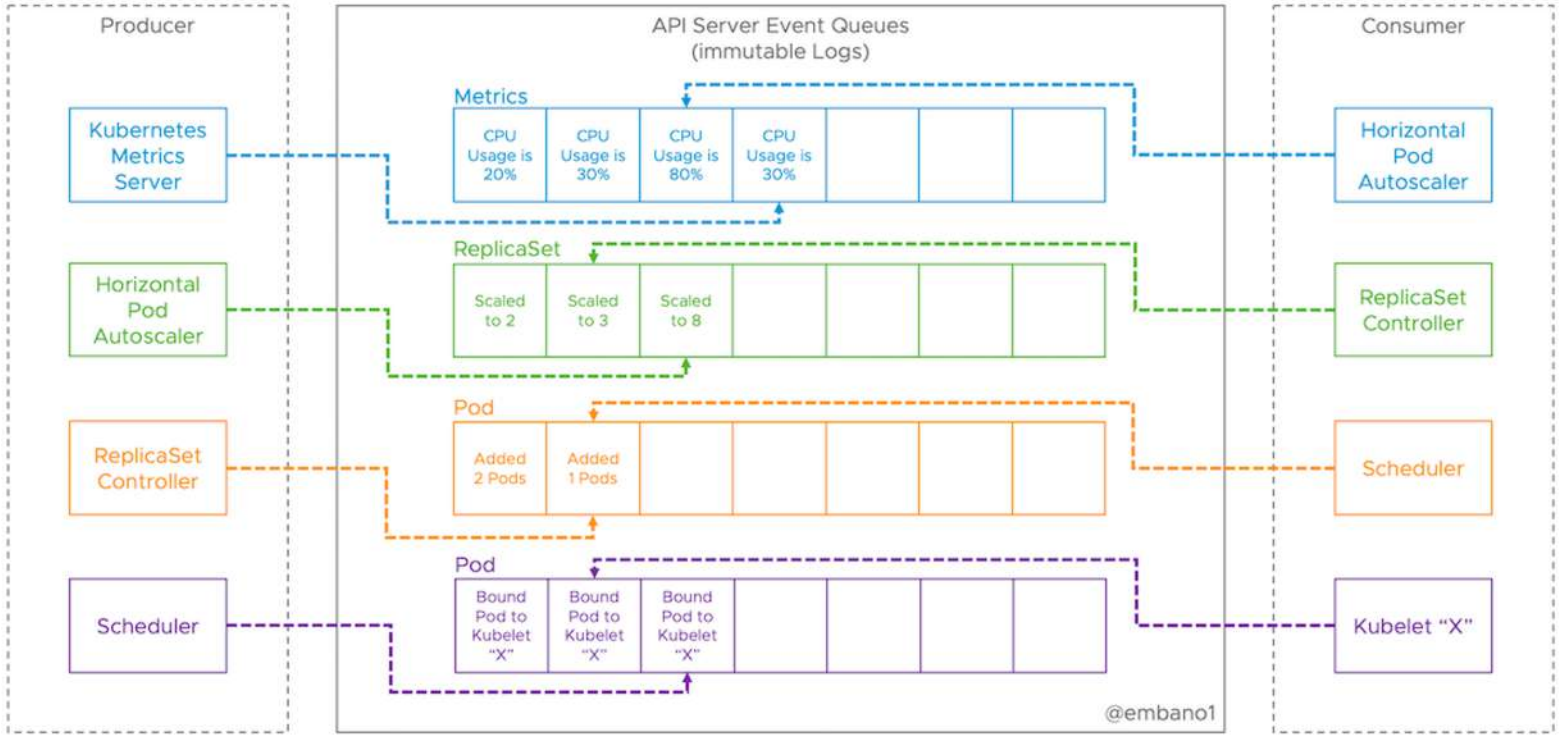
Watch

"Kubernetes is a declarative, event-driven system."



Watch

"Kubernetes is a declarative, event-driven system."



https://www.mgasch.com/2018/08/k8sevents/

Watch

"Kubernetes is a declarative, event-driven system."

- I have my state of the world from LIST. Now I need to know as and when events happen that modify this state so that I can take corrective action.

```
> kubectl get --raw
'/api/v1/namespaces/default/pods?limit=100&cont
inue=ENCODED_CONTINUE_TOKEN_2'
```

```
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion":"1452",
    "continue": "ENCODED_CONTINUE_TOKEN_2",
    ...
  },
  "items": [...] // pod100-pod199
}
```

Watch

"Kubernetes is a declarative, event-driven system."

- I have my state of the world from LIST. Now I need to know as and when events happen that modify this state so that I can take corrective action.

```
> kubectl get --raw
'/api/v1/namespaces/default/pods?limit=100&cont
inue=ENCODED_CONTINUE_TOKEN_2'
```

```
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion":"1452",
    "continue": "ENCODED_CONTINUE_TOKEN_2",
    ...
  },
  "items": [...] // pod100-pod199
}
```

Watch

"Kubernetes is a declarative, event-driven system."

- I have my state of the world from LIST. Now I need to know as and when events happen that modify this state so that I can take corrective action.
- **WATCH** for changes. The API Server gives us a stream of notifications on a single connection that we can "react" to.

```
> kubectl get --raw
'/api/v1/namespaces/default/pods?
watch=1&resourceVersion=1452'
```

```
{
  "type": "MODIFIED",
  "object": {
      "kind": "Pod", "apiVersion": "v1",
      "metadata": {"resourceVersion":"1650", ...}, ...}
}
...
{
  "type": "DELETED",
  "object": {
      "kind": "Pod", "apiVersion": "v1",
      "metadata": {"resourceVersion":"1734", ...}, ...}
}
```

resourceVersion

resourceVersion

- Opaque string representing “internal version” of an object.
- One big, global, logical clock.

`resourceVersion`

- Opaque string representing “internal version” of an object.
- One big, global, logical clock.
- `resourceVersion` is backed by etcd’s `store revisions`* – which provide a global ordering.
- Increases monotonically whenever *any* change to the state of the world happens.

`resourceVersion`

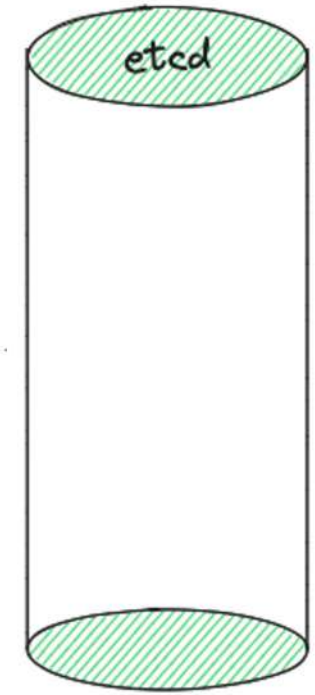
- Opaque string representing “internal version” of an object.
- One big, global, logical clock.
- `resourceVersion` is backed by etcd’s `store revisions`* – which provide a global ordering.
- Increases monotonically whenever *any* change to the state of the world happens.
- Gives you a global order of events that happen in the system.
- Most importantly - they enable *optimistic concurrency control*.

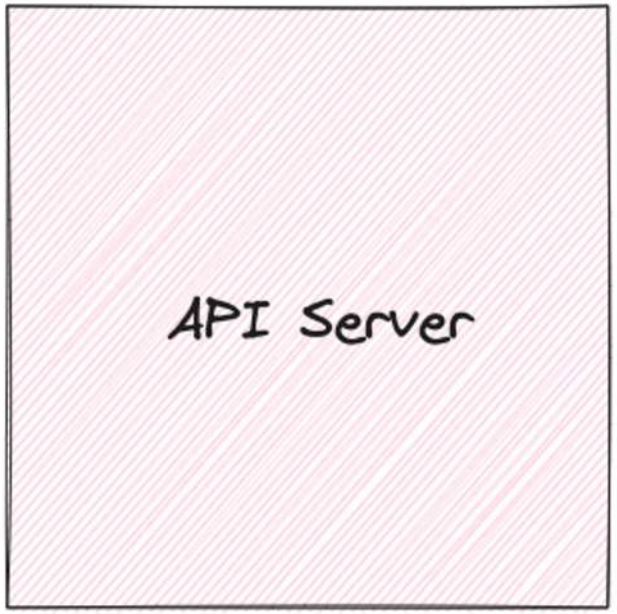
resourceVersion

https://sched.co/1R2m8

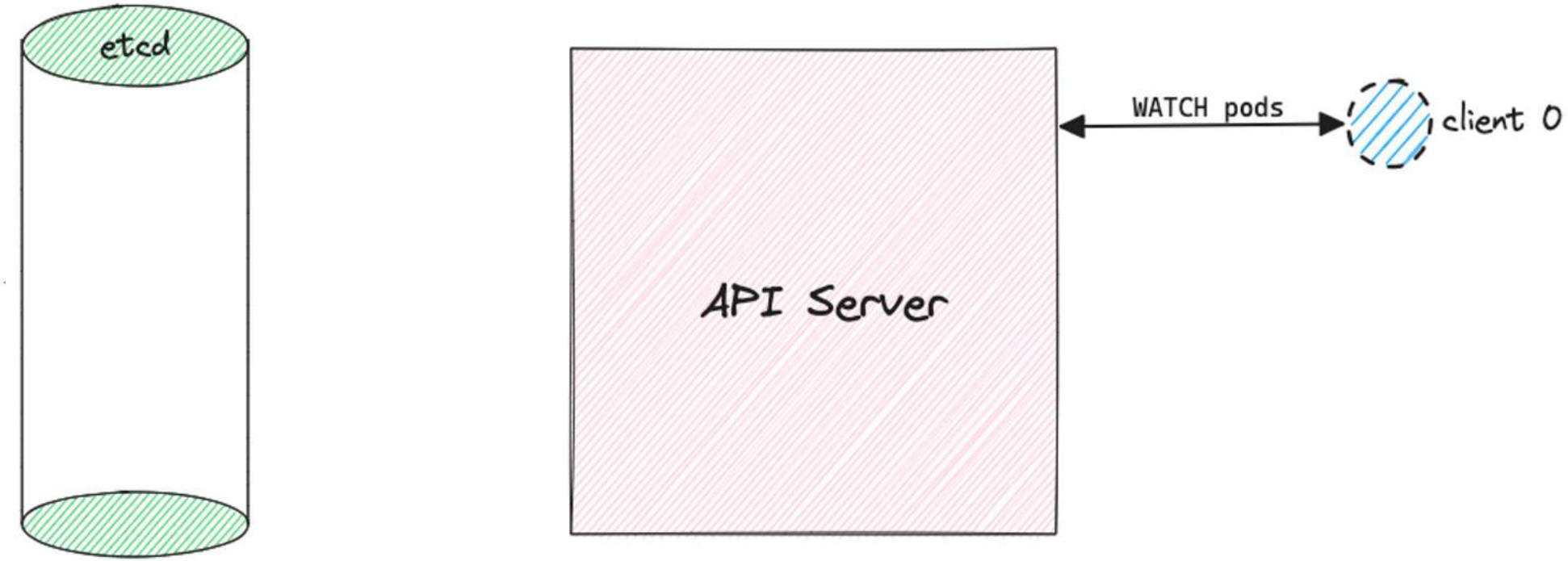
The Kubernetes Storage Layer - Past

The Kubernetes Storage Layer - Past

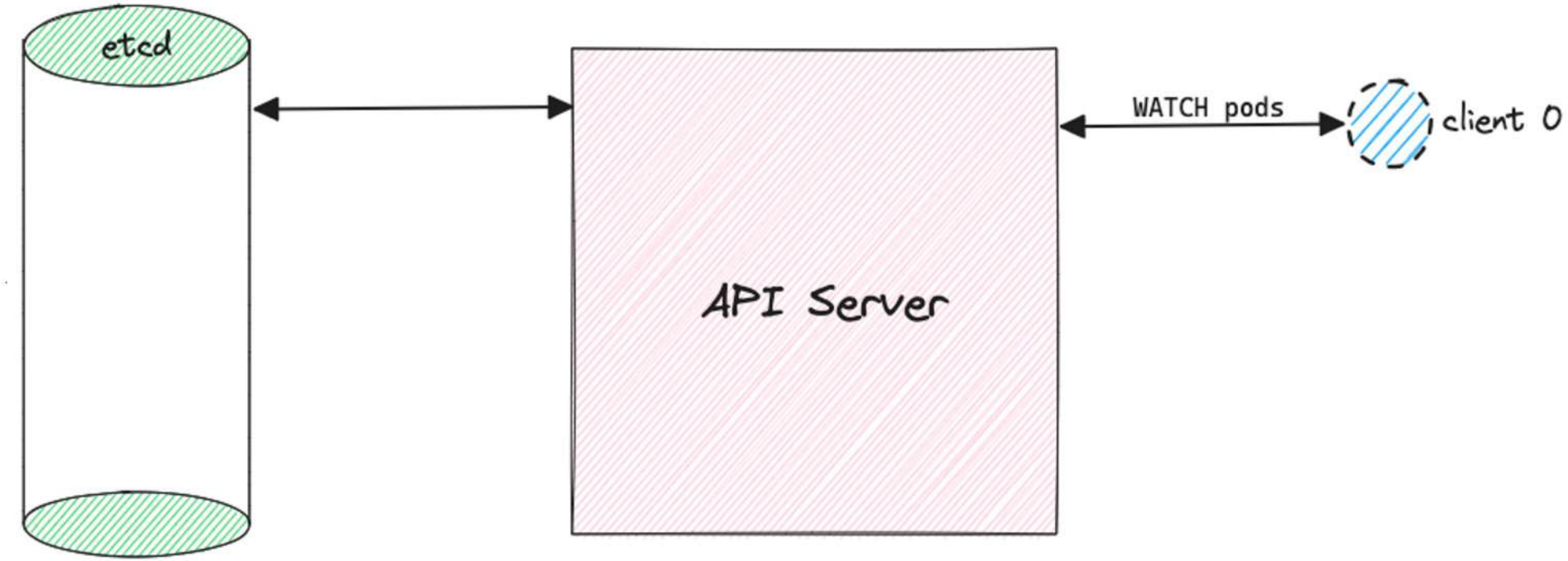




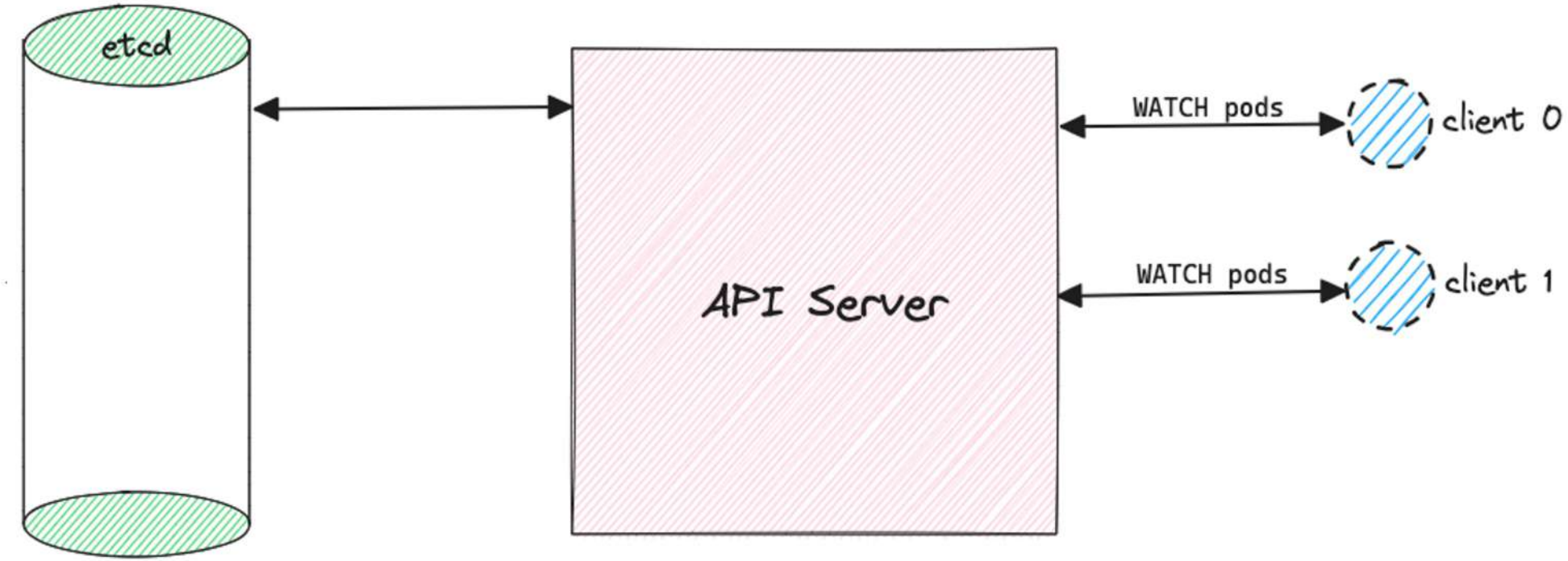
The Kubernetes Storage Layer - Past



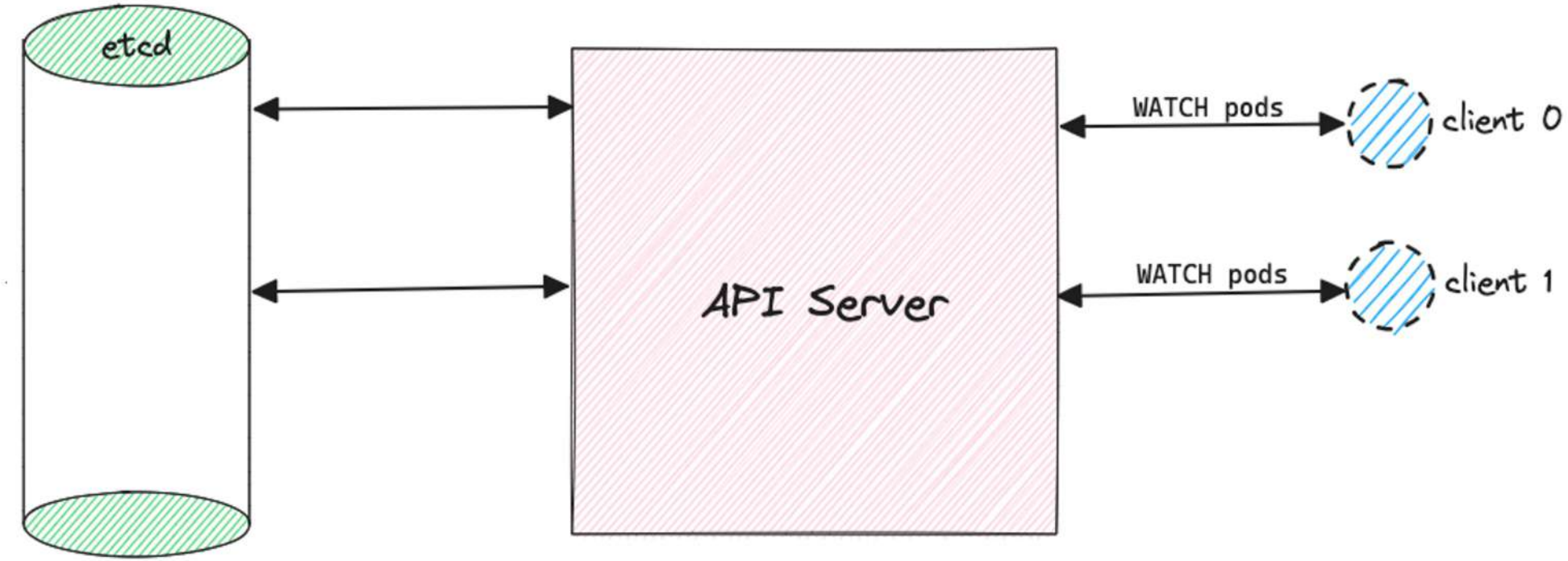
The Kubernetes Storage Layer - Past



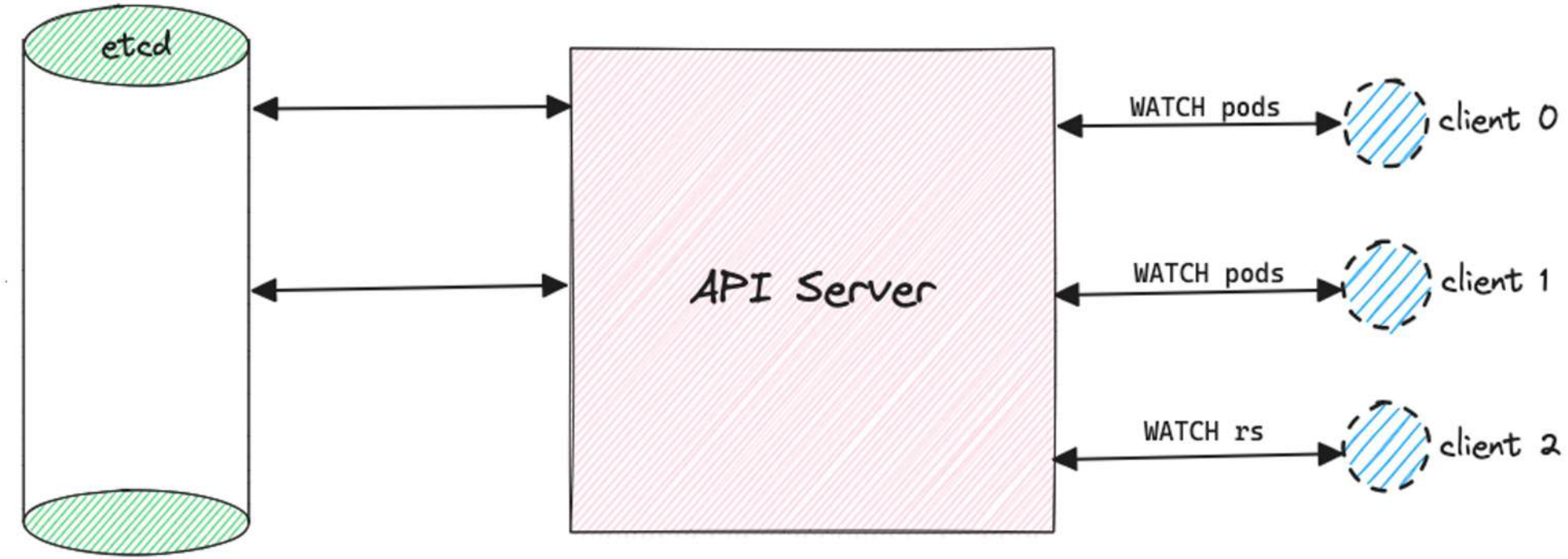
The Kubernetes Storage Layer - Past



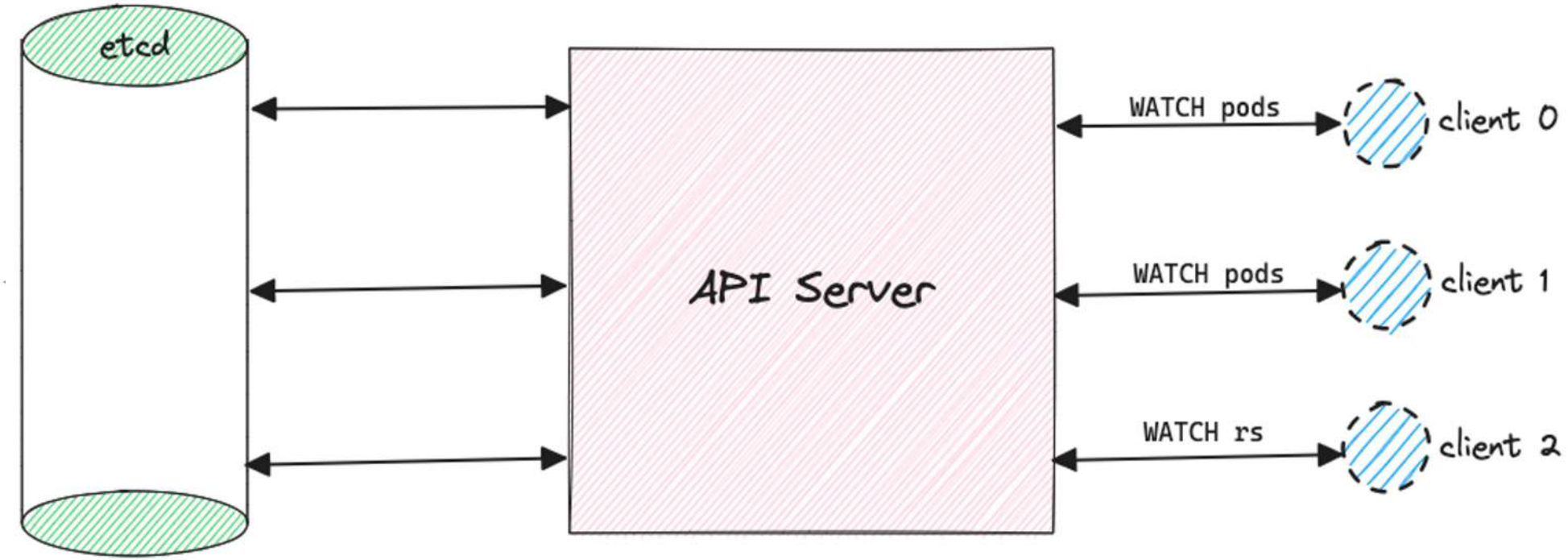
The Kubernetes Storage Layer - Past



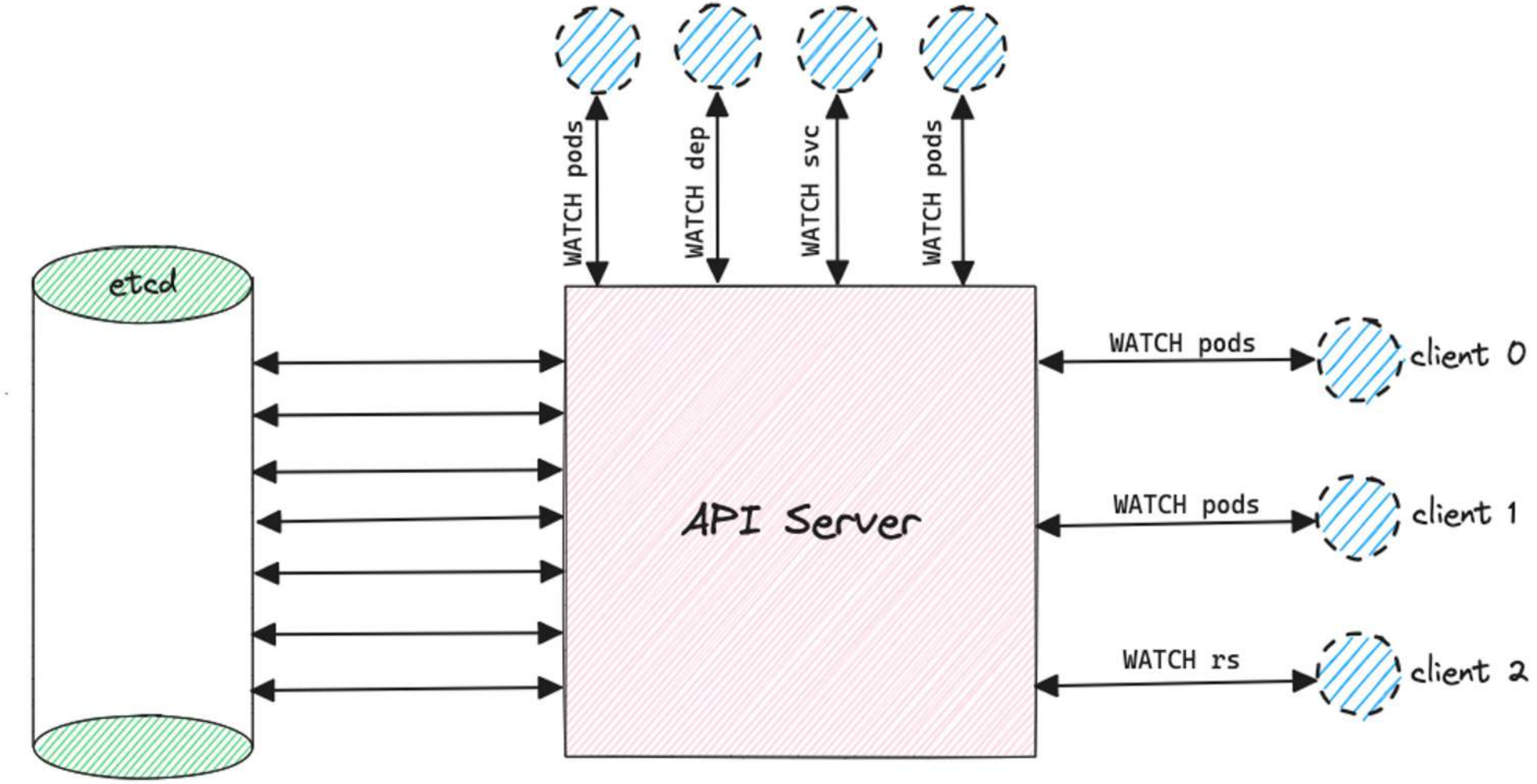
The Kubernetes Storage Layer - Past



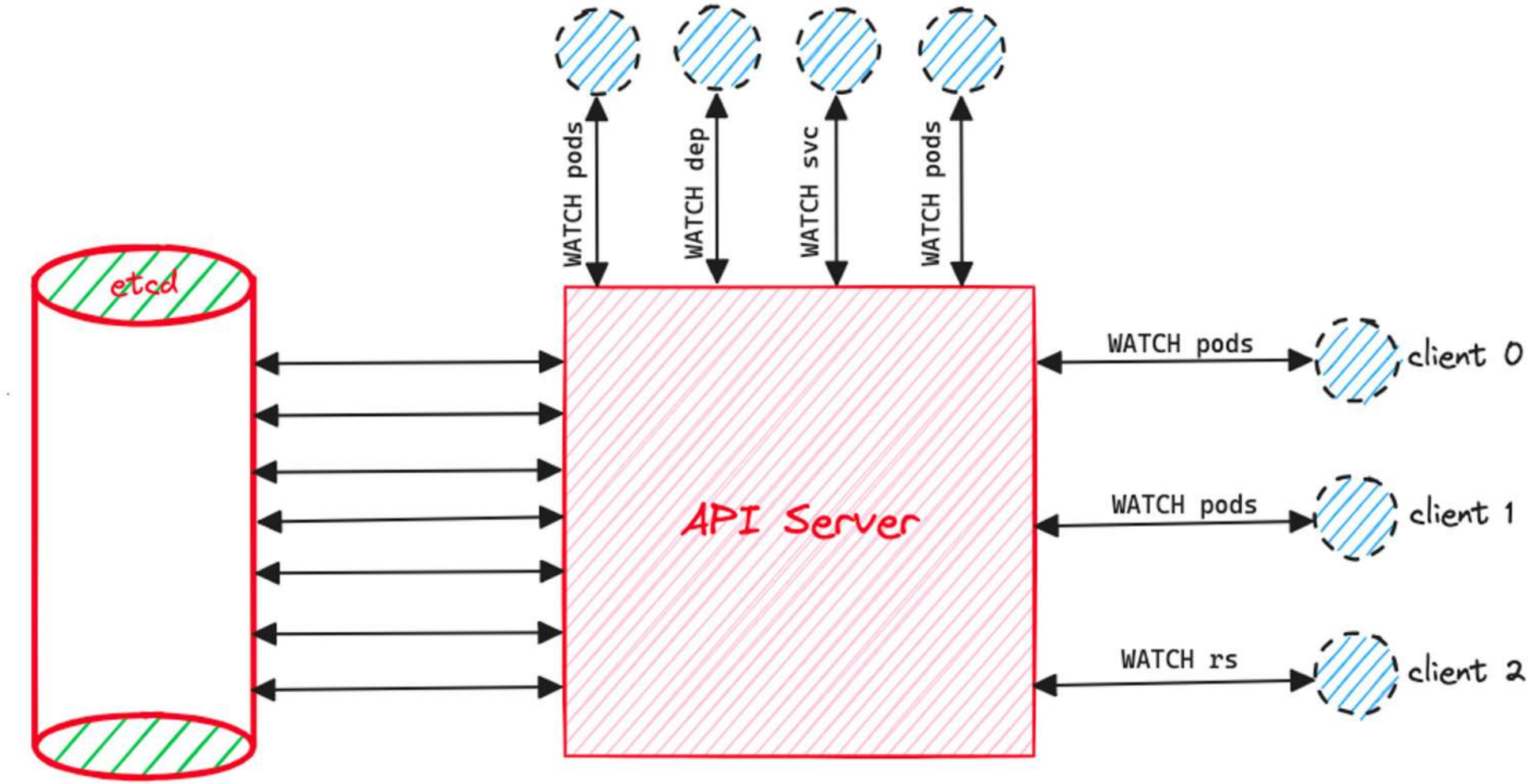
The Kubernetes Storage Layer - Past



The Kubernetes Storage Layer - Past



The Kubernetes Storage Layer - Past



The Kubernetes Storage Layer - Past

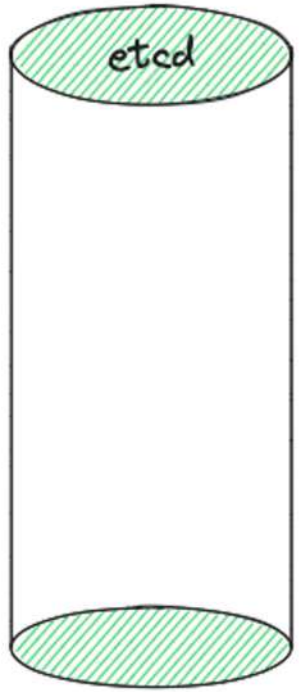
If you had a controller, more the replicas, lesser the scalability of **`etcd`**.

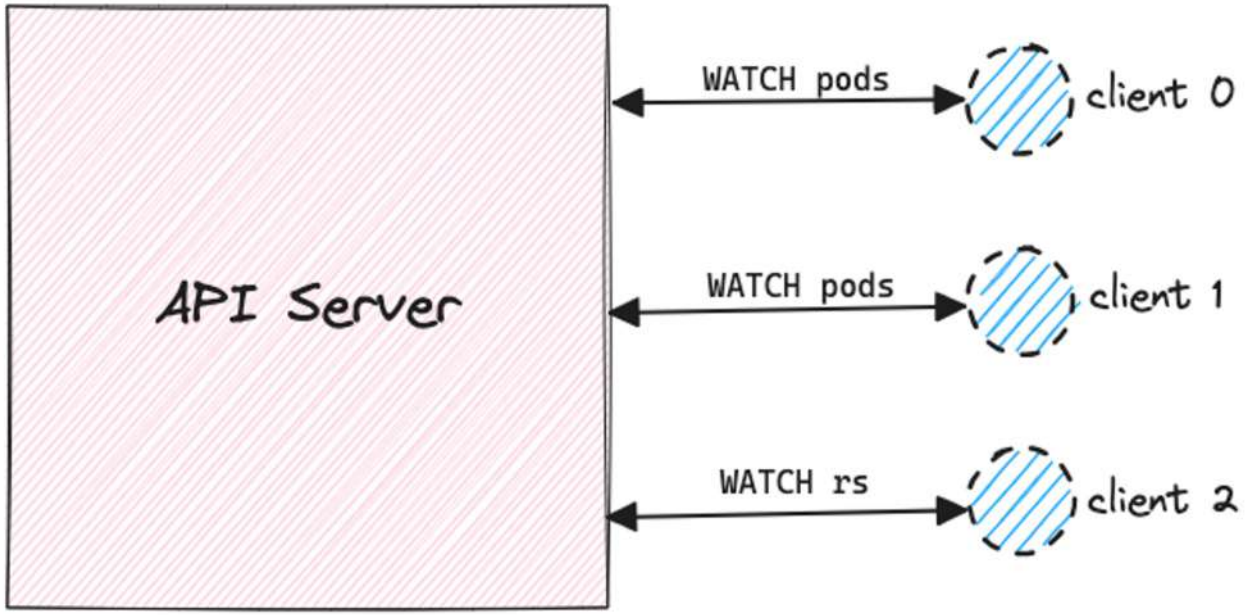
The Kubernetes Storage Layer - Present

The Kubernetes Storage Layer - Present

As with any problem in Computer Science, we solve this also with a layer(s) of indirection.

The Kubernetes Storage Layer - Present



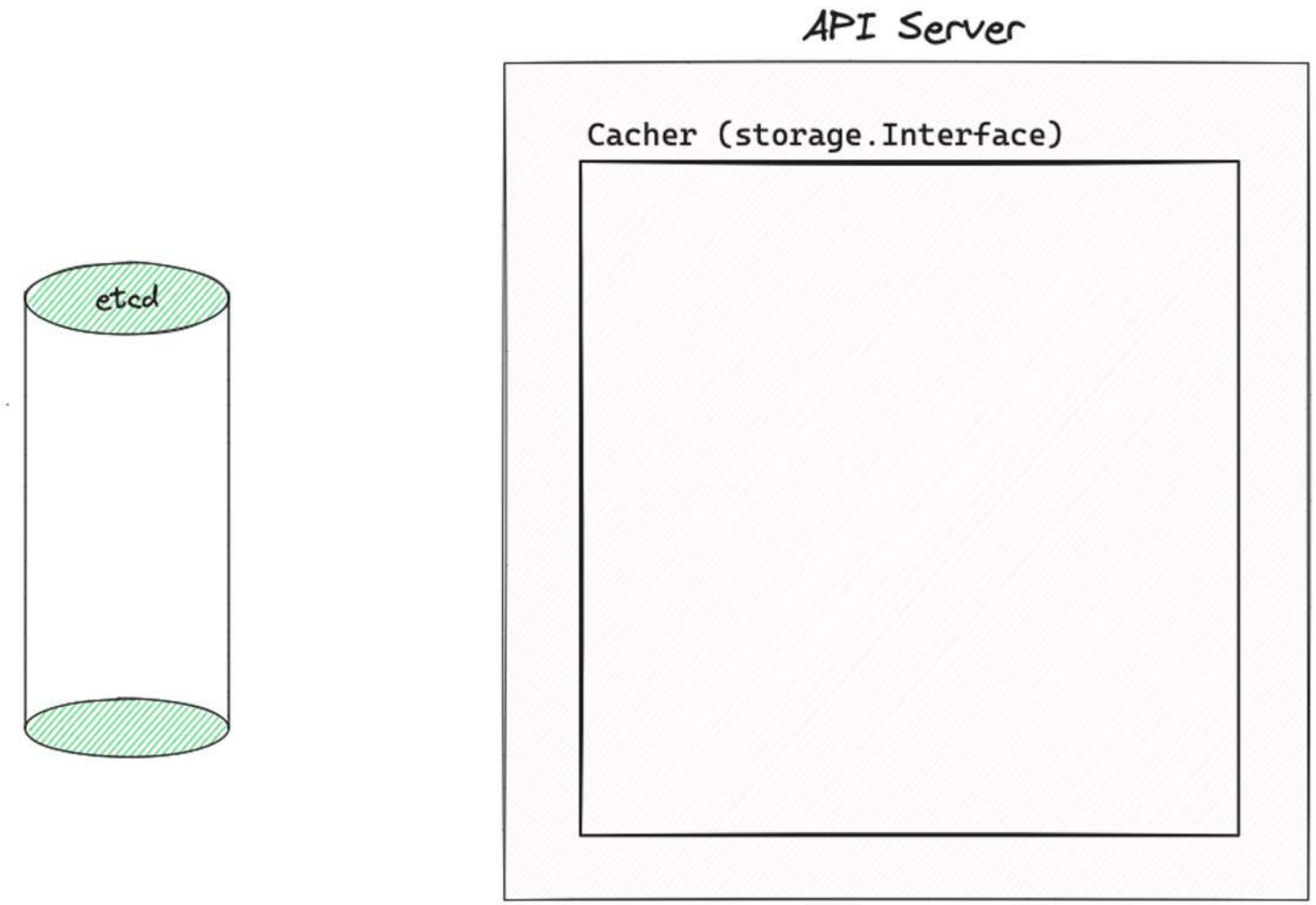


The Kubernetes Storage Layer - Present

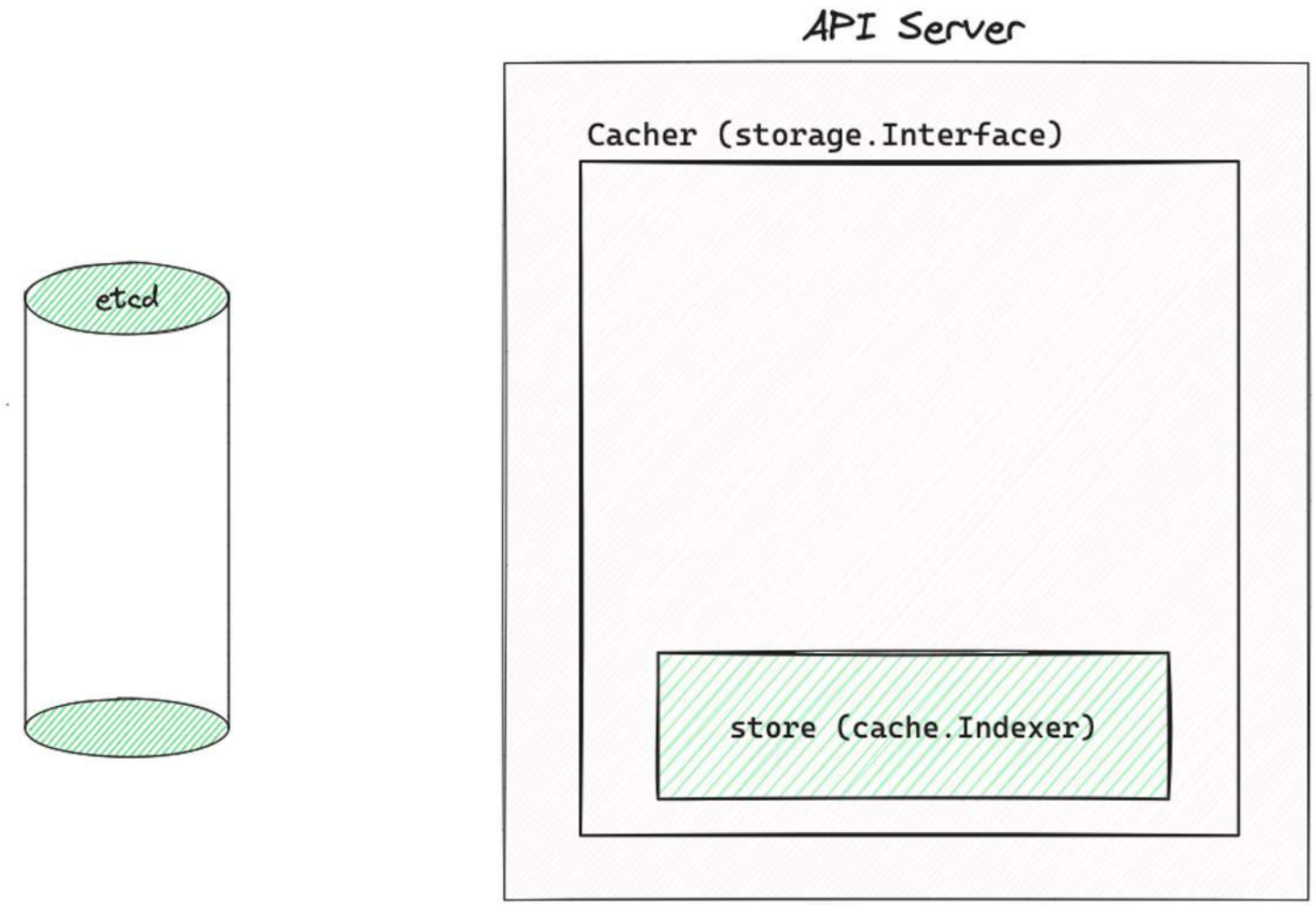
Zooming in…

The Kubernetes Storage Layer - Present

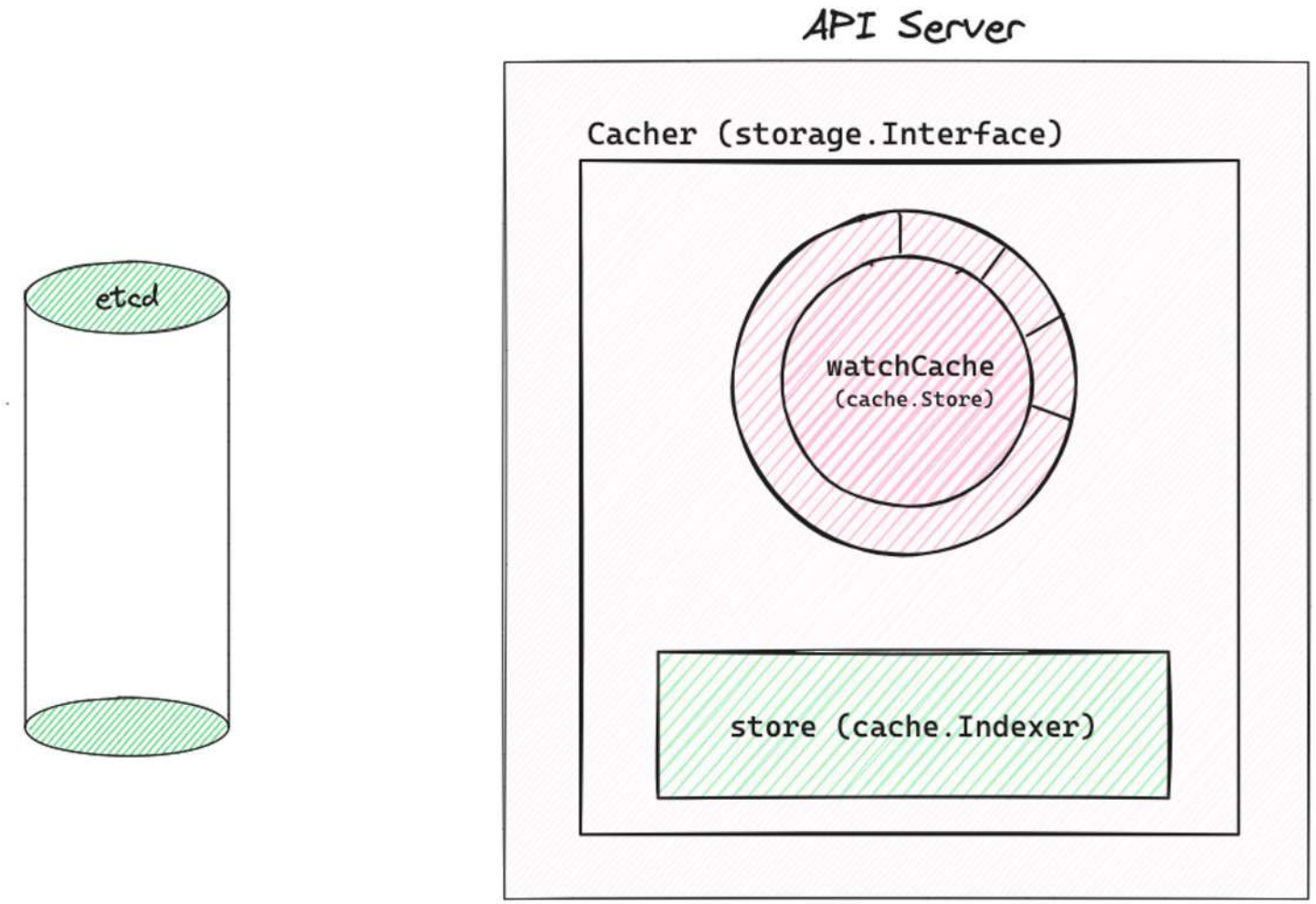
The Kubernetes Storage Layer - Present



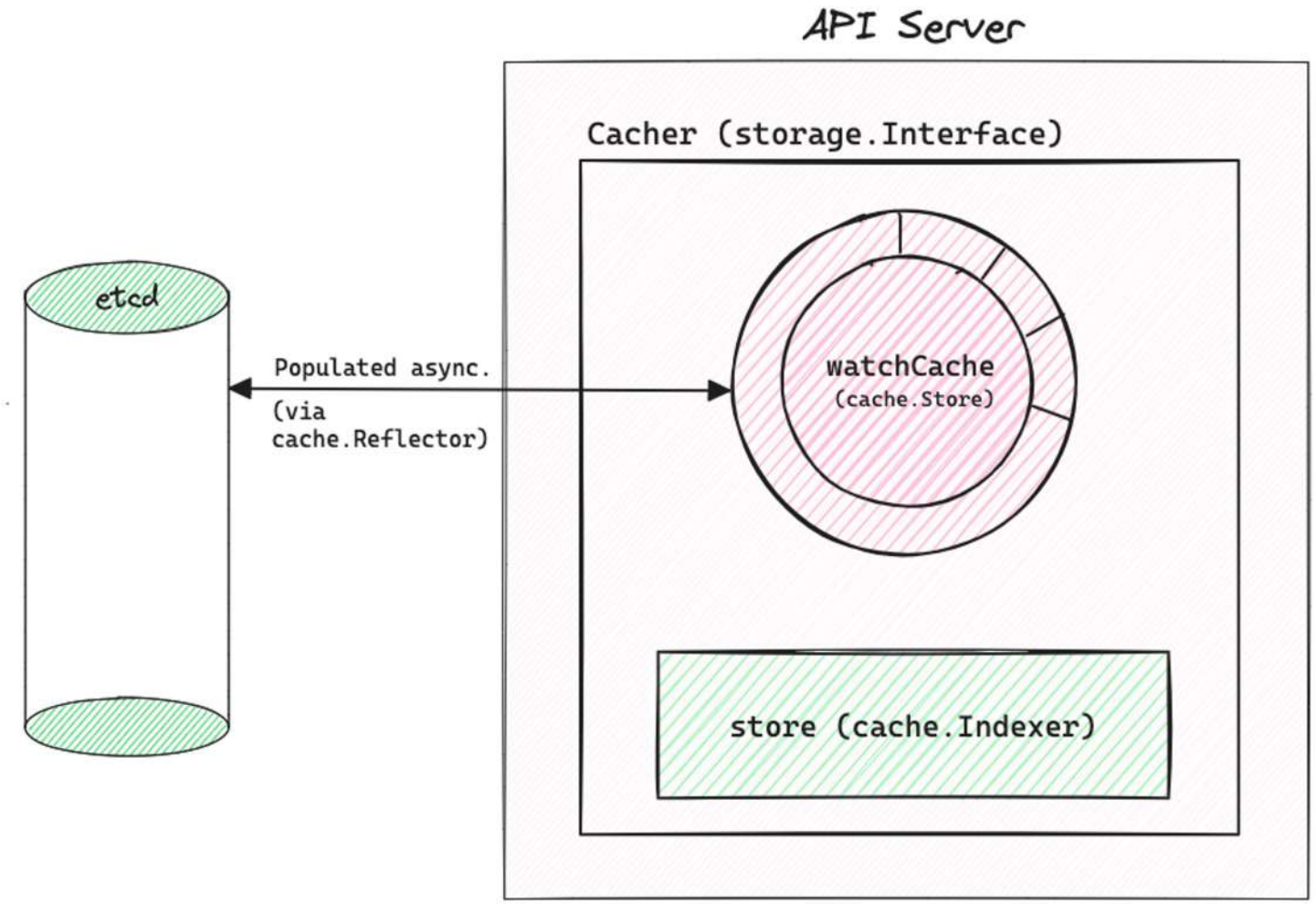
The Kubernetes Storage Layer - Present



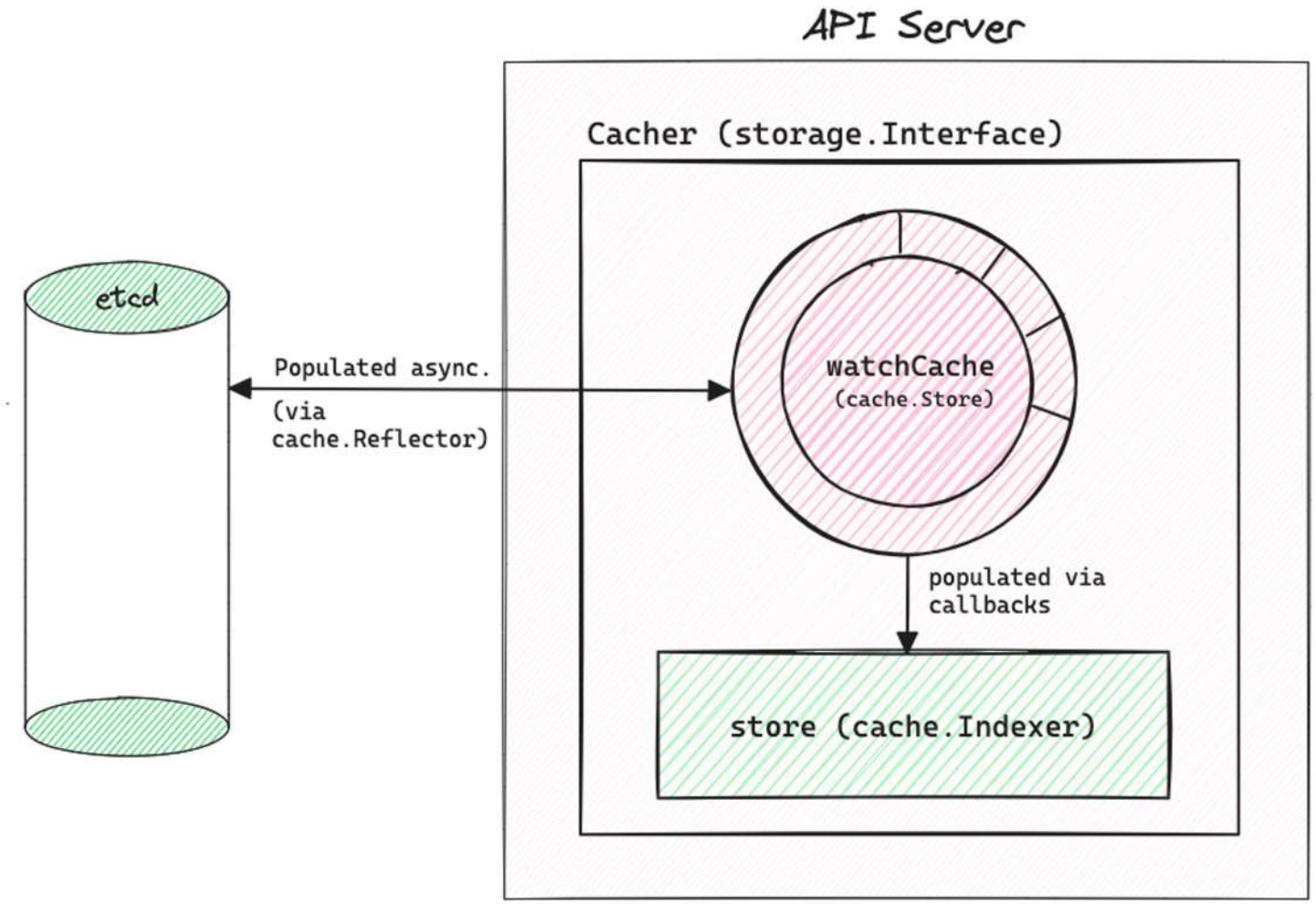
The Kubernetes Storage Layer - Present



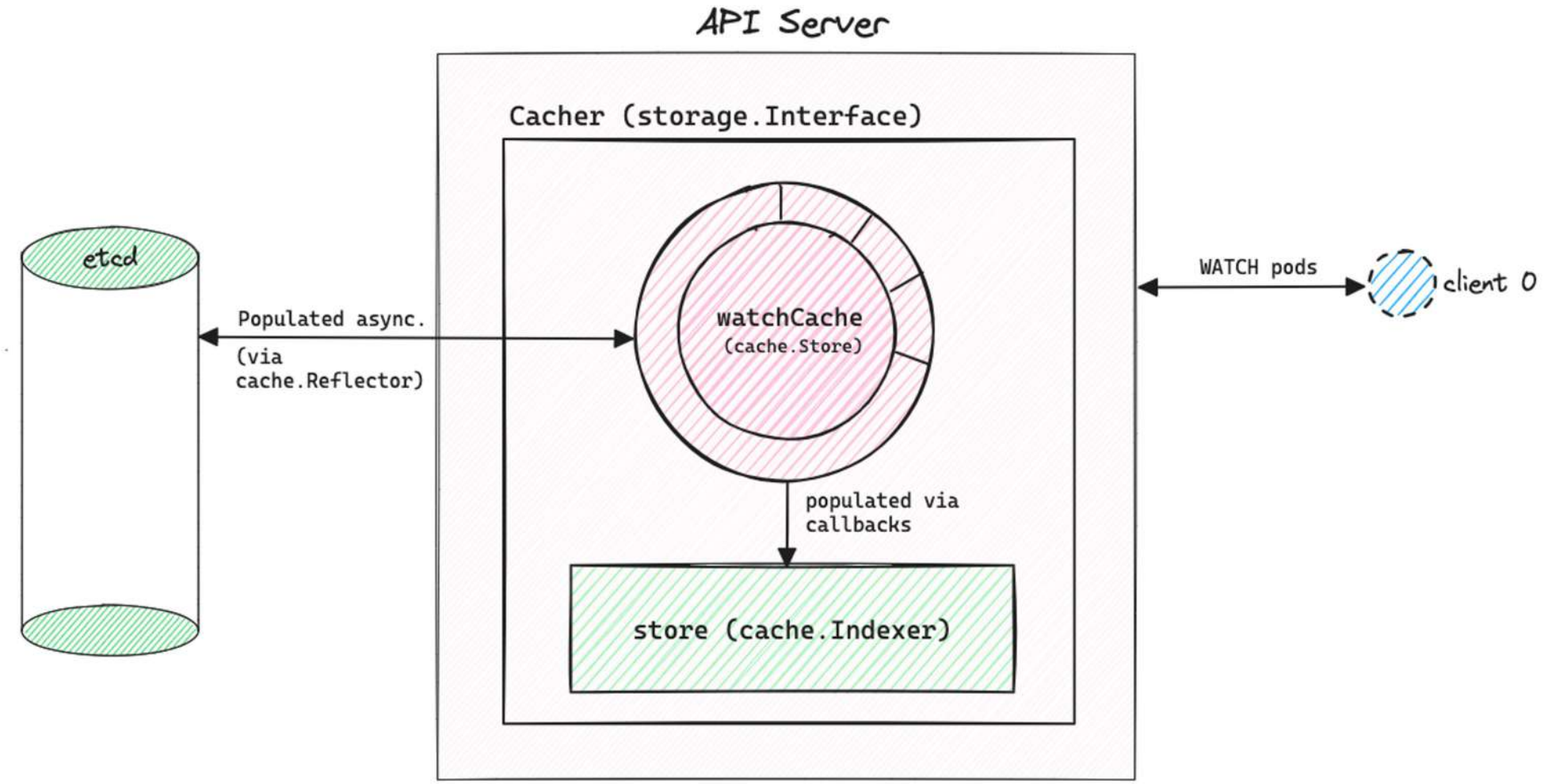
The Kubernetes Storage Layer - Present



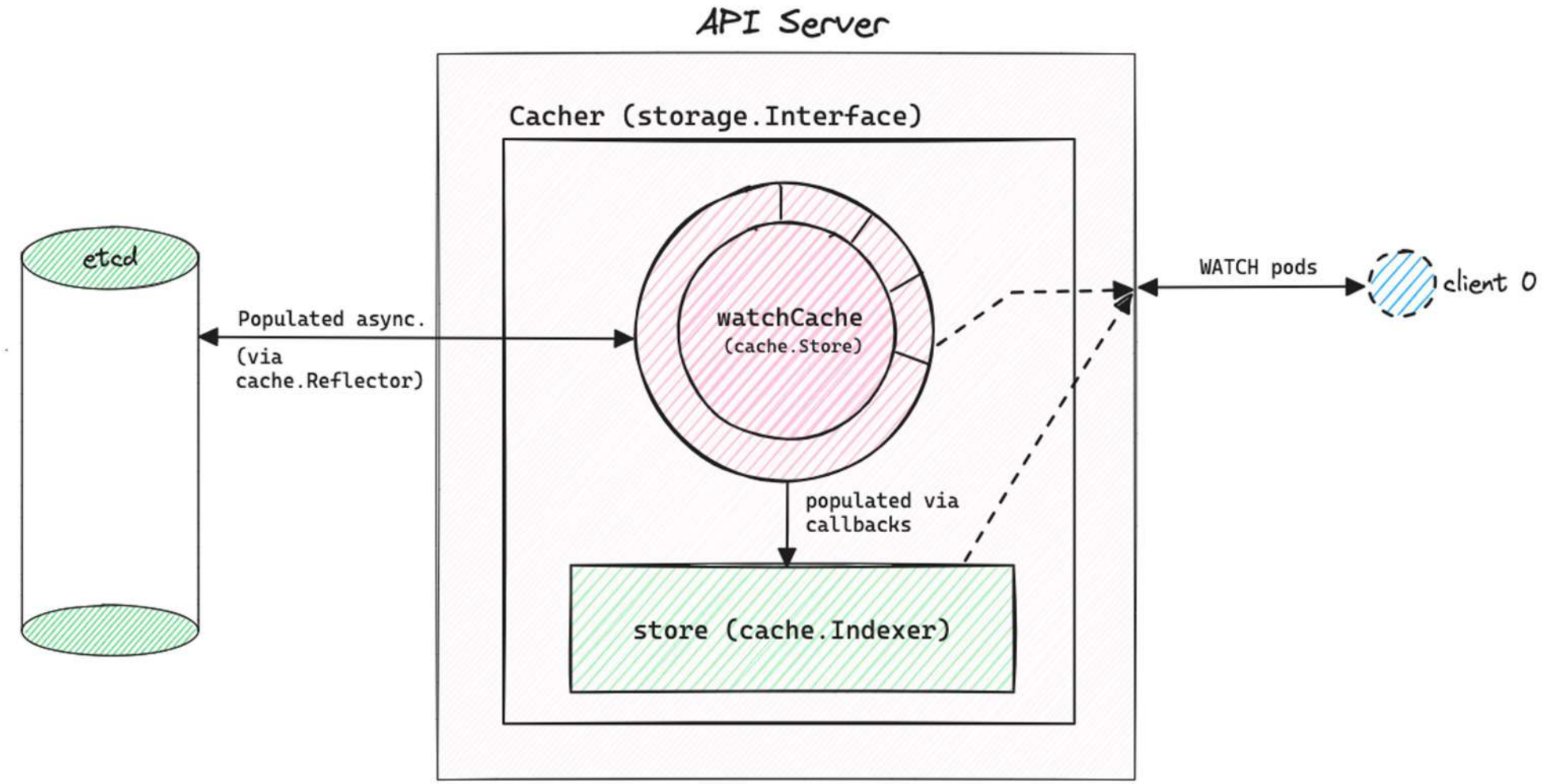
The Kubernetes Storage Layer - Present



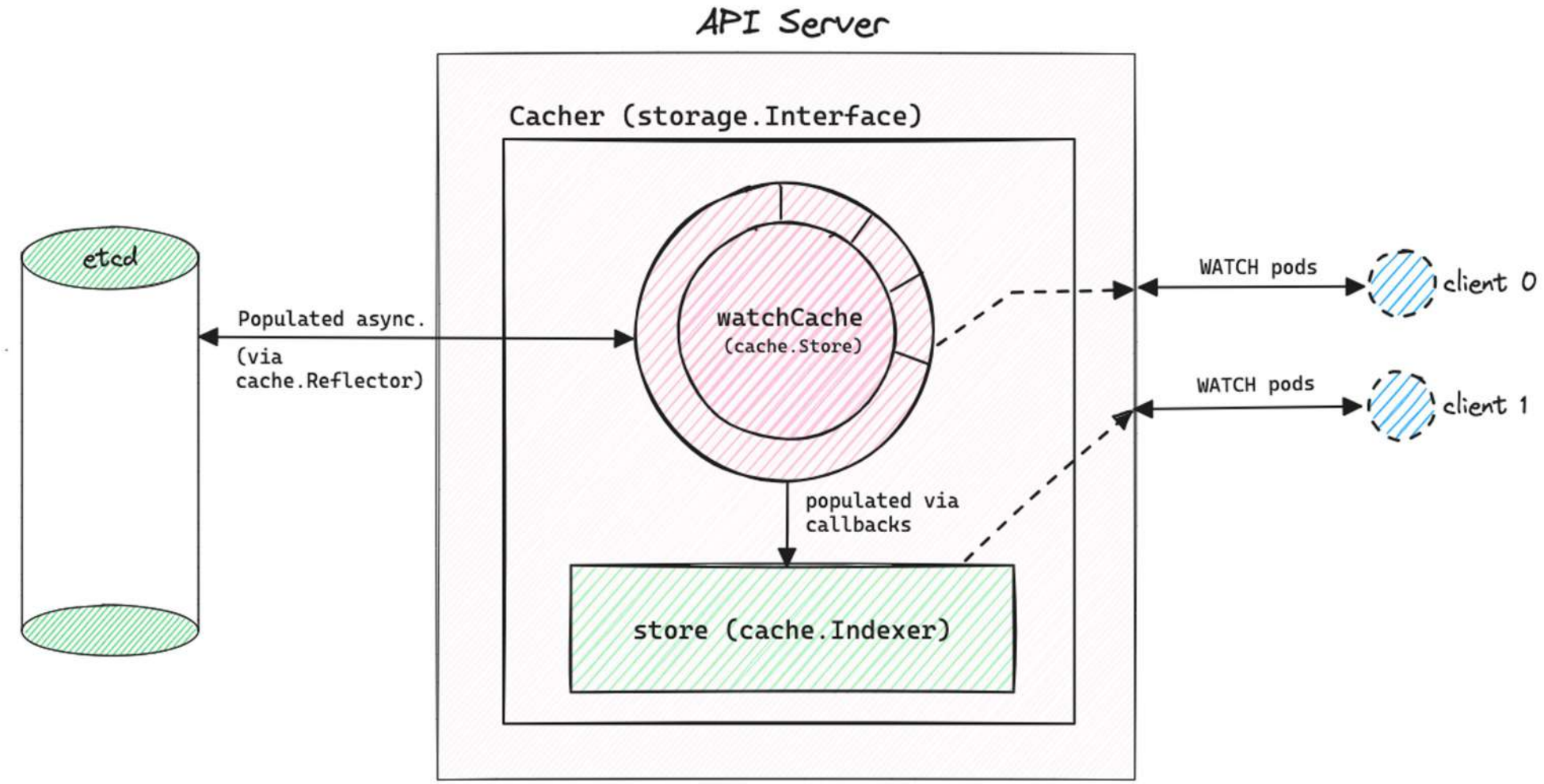
The Kubernetes Storage Layer - Present



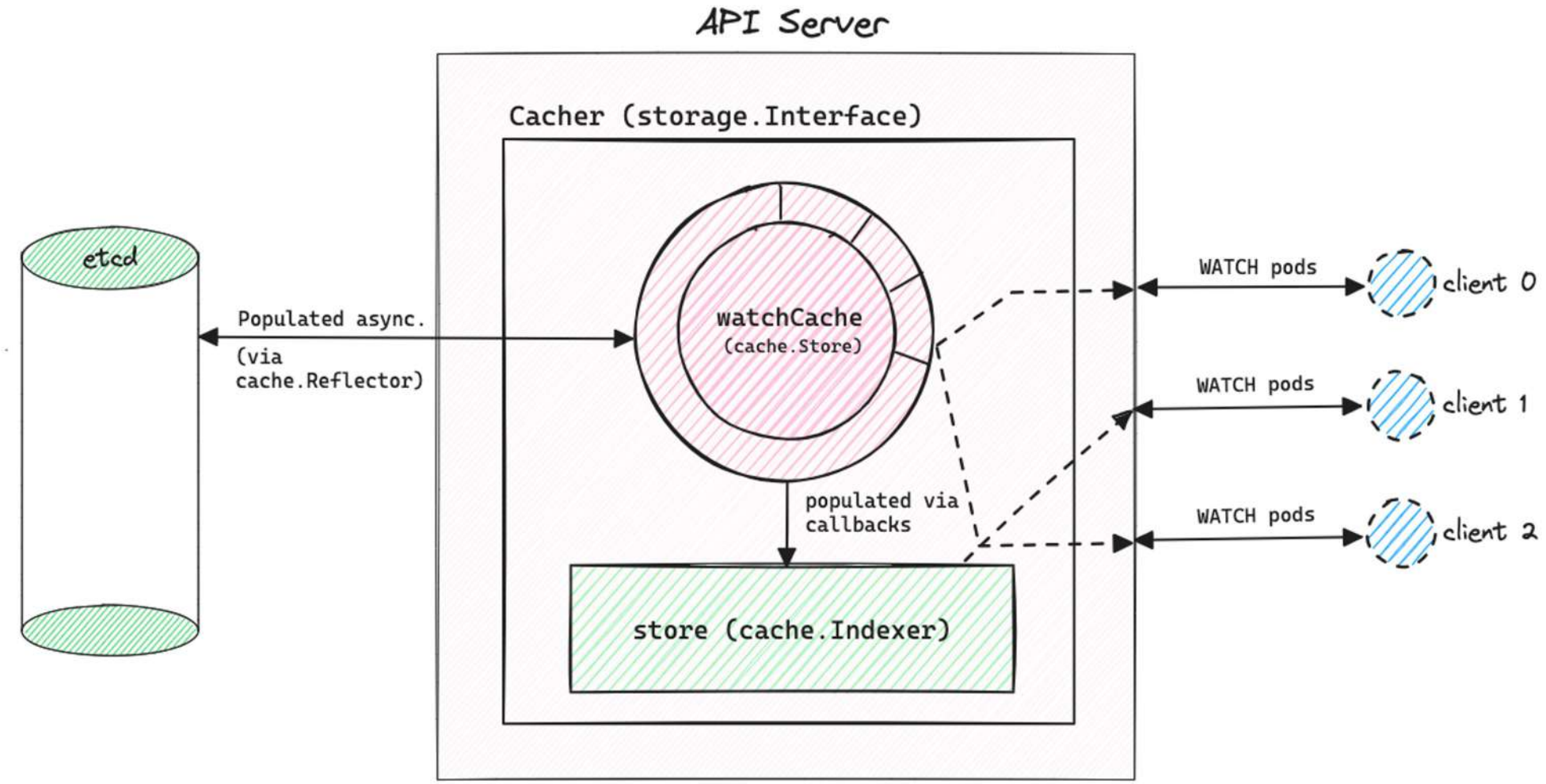
The Kubernetes Storage Layer - Present



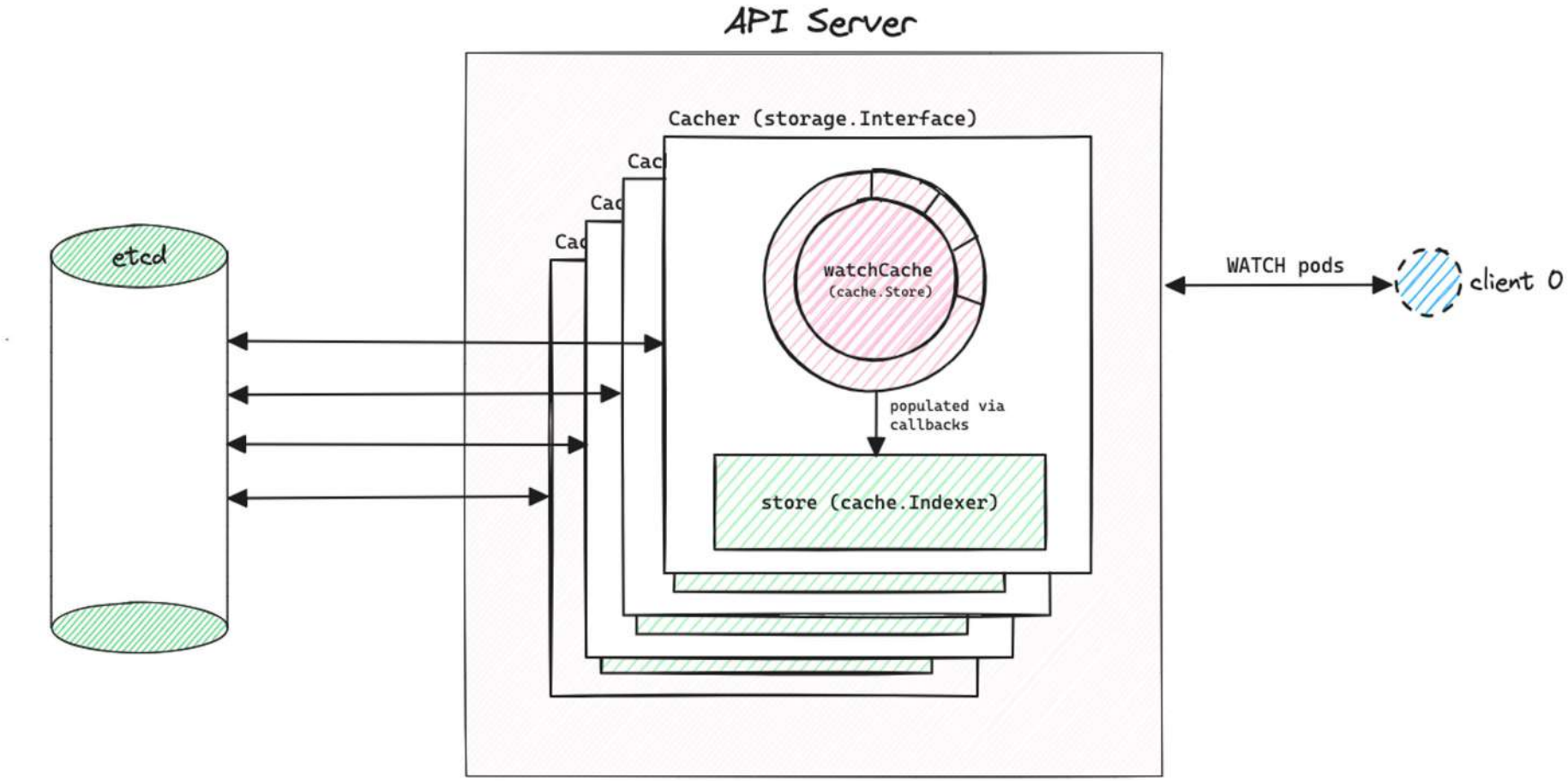
The Kubernetes Storage Layer - Present



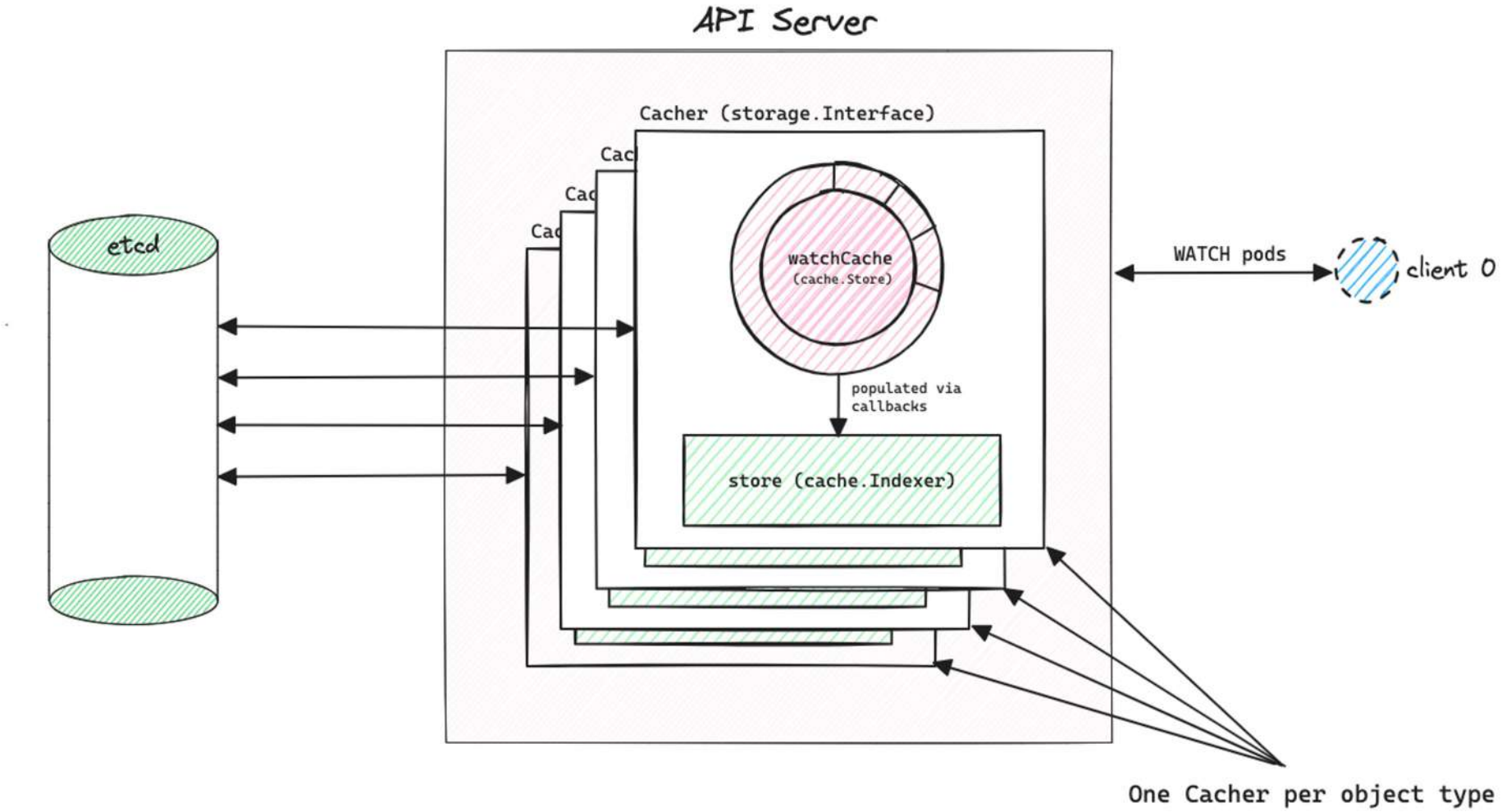
The Kubernetes Storage Layer - Present



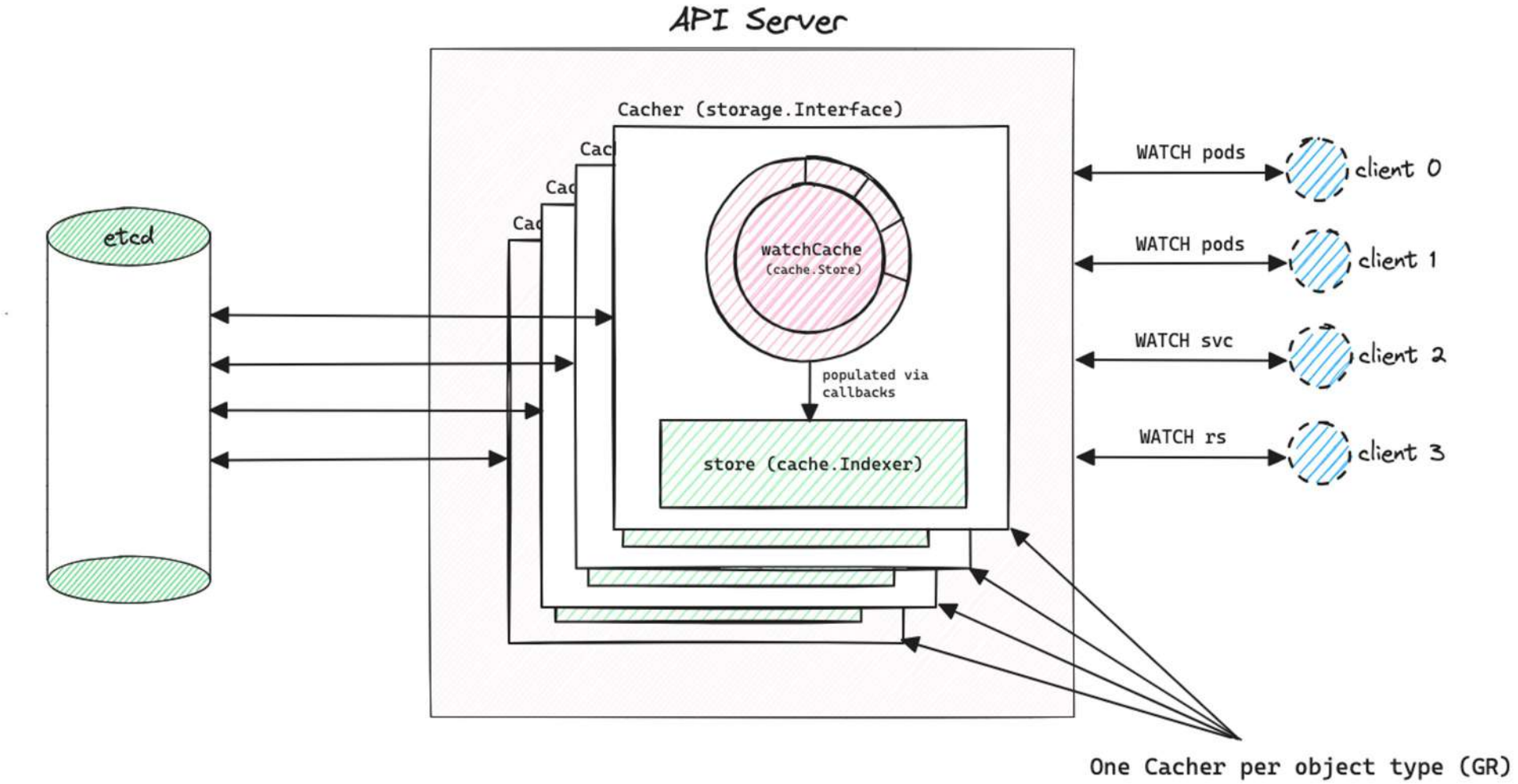
The Kubernetes Storage Layer - Present



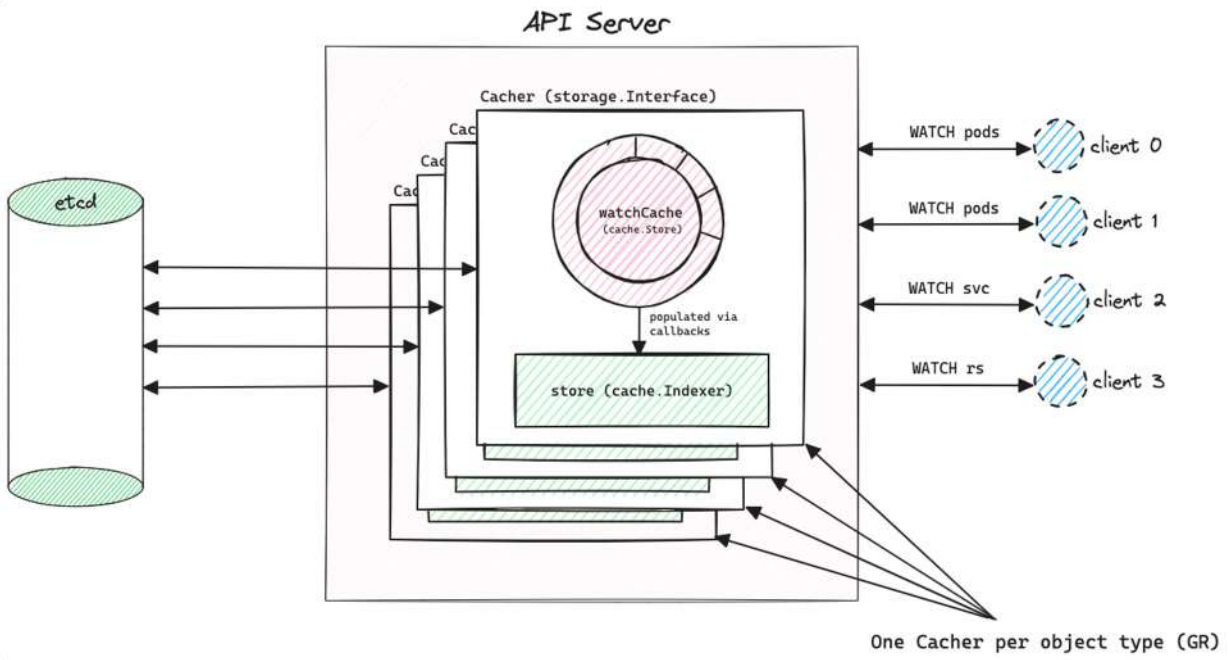
The Kubernetes Storage Layer - Present



The Kubernetes Storage Layer - Present

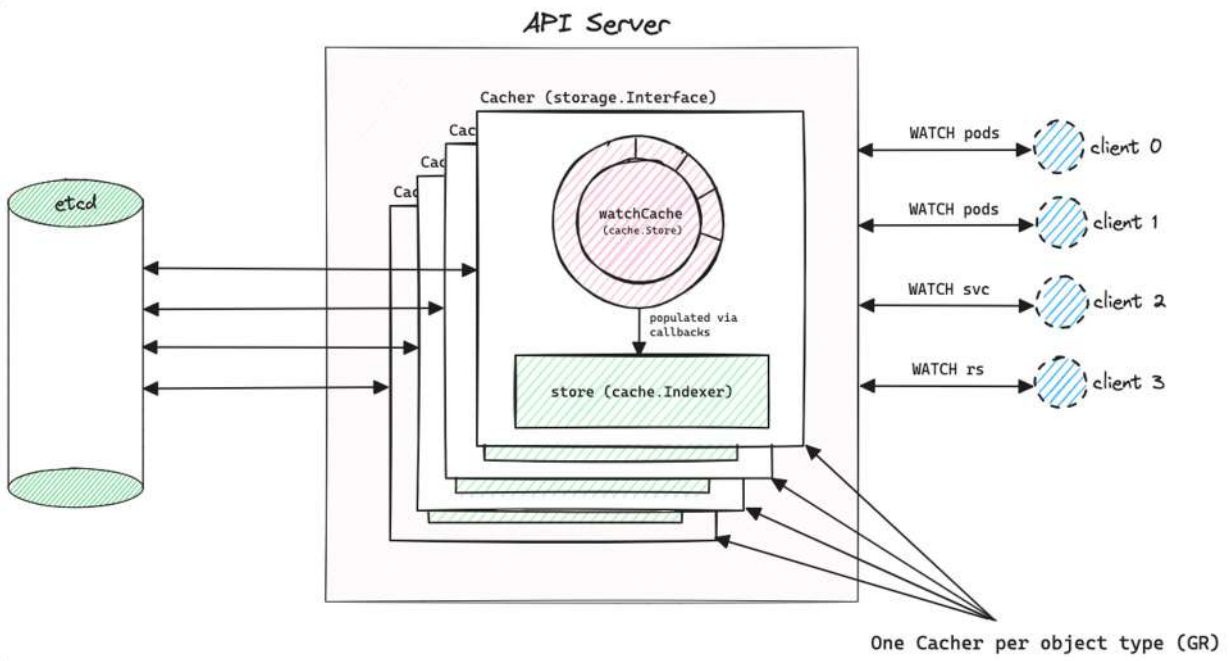


The Kubernetes Storage Layer - Present



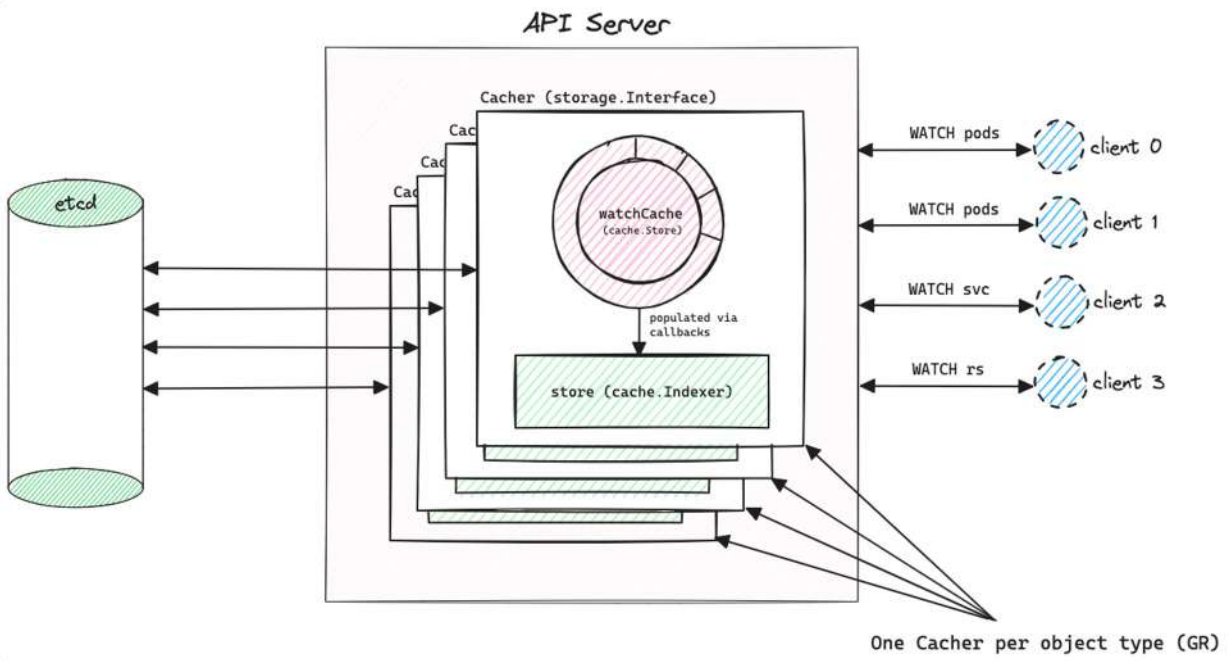
The Kubernetes Storage Layer - Present

- The **`store`** component is meant to reflect the state of **`etcd`**.



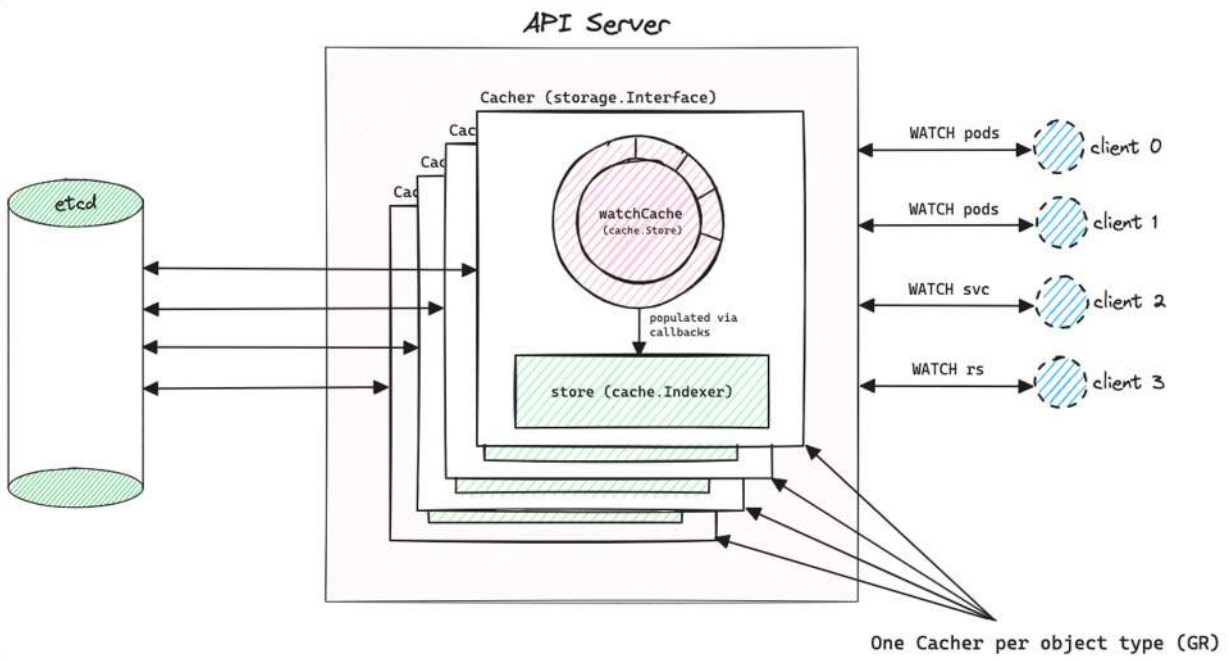
The Kubernetes Storage Layer - Present

- The **`store`** component is meant to reflect the state of **`etcd`**.
- **`Cacher`** per object type is created at API Server start-up time.



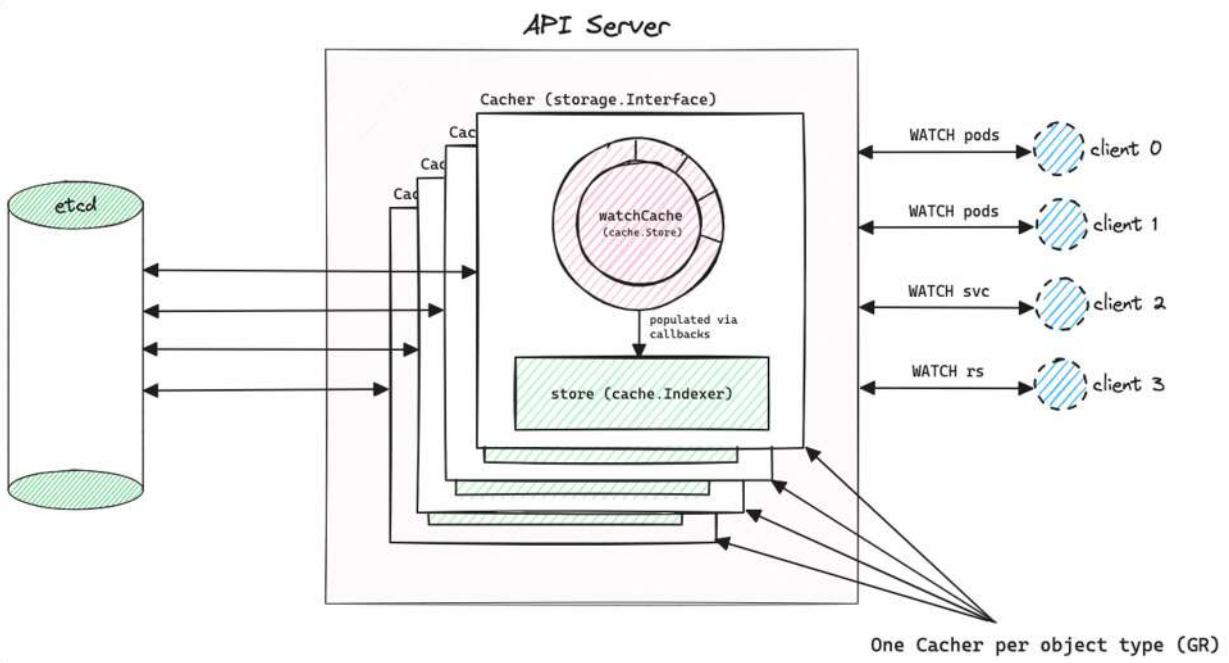
The Kubernetes Storage Layer - Present

- The **`store`** component is meant to reflect the state of **`etcd`**.
- **`Cacher`** per object type is created at API Server start-up time.
- The caching layer can be disabled altogether (**`--watch-cache=false`**).



The Kubernetes Storage Layer - Present

- The **`store`** component is meant to reflect the state of **`etcd`**.
- **`Cacher`** per object type is created at API Server start-up time.
- The caching layer can be disabled altogether (**`--watch-cache=false`**).
- The caching layer can be disabled on a per object-type (**`GroupResource`**) basis (**`--watch-cache-sizes`**) by setting the size to 0, all non-zero values are equivalent.



The Kubernetes Storage Layer - Present

How do different requests interact with our present storage layer?

The Kubernetes Storage Layer - Present

Interlude – `resourceVersion` semantics

`resourceVersion` semantics

- In each type of CRUD request, you can pass a `resourceVersion` parameter.

`resourceVersion` semantics

- In each type of CRUD request, you can pass a `resourceVersion` parameter.
- The interpretation of this parameter translates into data consistency guarantees.

`resourceVersion` semantics

- In each type of CRUD request, you can pass a `resourceVersion` parameter.
- The interpretation of this parameter translates into data consistency guarantees.
- Knowing how behaviour changes with `resourceVersion` interpretation can be crucial to scalability in some cases.

`resourceVersion` semantics

- In each type of CRUD request, you can pass a **`resourceVersion`** parameter.
- The interpretation of this parameter translates into data consistency guarantees.
- Knowing how behaviour changes with **`resourceVersion`** interpretation can be crucial to scalability in some cases.

For any **`GET`** request (**`Get()`**, **`GetList()`**, **`Watch()`**)

`resourceVersion` semantics

- In each type of CRUD request, you can pass a **`resourceVersion`** parameter.
- The interpretation of this parameter translates into data consistency guarantees.
- Knowing how behaviour changes with **`resourceVersion`** interpretation can be crucial to scalability in some cases.

For any **`GET`** request (**`Get()`, `GetList()`, `Watch()`**)

`resourceVersion = “”`	Most recent data

`resourceVersion` semantics

- In each type of CRUD request, you can pass a **`resourceVersion`** parameter.
- The interpretation of this parameter translates into data consistency guarantees.
- Knowing how behaviour changes with **`resourceVersion`** interpretation can be crucial to scalability in some cases.

For any **`GET`** request (**`Get()`, `GetList()`, `Watch()`**)

`resourceVersion = ""`	Most recent data
`resourceVersion = "0"`	Any data (arbitrarily stale)

`resourceVersion` semantics

- In each type of CRUD request, you can pass a **`resourceVersion`** parameter.
- The interpretation of this parameter translates into data consistency guarantees.
- Knowing how behaviour changes with **`resourceVersion`** interpretation can be crucial to scalability in some cases.

For any **`GET`** request (**`Get()`, `GetList()`, `Watch()`**)

`resourceVersion = ""`	Most recent data
`resourceVersion = "0"`	Any data (arbitrarily stale)
`resourceVersion = "n"`	Data at **`n`**

`resourceVersion` semantics

- In each type of CRUD request, you can pass a **`resourceVersion`** parameter.
- The interpretation of this parameter translates into data consistency guarantees.
- Knowing how behaviour changes with **`resourceVersion`** interpretation can be crucial to scalability in some cases.

For any **`GET`** request (**`Get()`**, **`GetList()`**, **`Watch()`**)

`resourceVersion = ""`	Most recent data
`resourceVersion = "0"`	Any data (arbitrarily stale)
`resourceVersion = "n"`	Data at **`n`**

"Most recent data" is ensured by doing a quorum read in **`etcd`** (a round of raft happens, and you get a linearizable read).

`resourceVersion` semantics

- In each type of CRUD request, you can pass a **`resourceVersion`** parameter.
- The interpretation of this parameter translates into data consistency guarantees.
- Knowing how behaviour changes with **`resourceVersion`** interpretation can be crucial to scalability in some cases.

For any **`GET`** request (**`Get()`**, **`GetList()`**, **`Watch()`**)

`resourceVersion = ""`	Most recent data
`resourceVersion = "0"`	Any data (arbitrarily stale)
`resourceVersion = "n"`	Data at **`n`**

There is also **`resourceVersionMatch`** which compliments **`resourceVersion`** in how they are interpreted. You always need to provide this if you specify a **`resourceVersion`** in a **`LIST`** request.

`resourceVersion` semantics

- In each type of CRUD request, you can pass a **`resourceVersion`** parameter.
- The interpretation of this parameter translates into data consistency guarantees.
- Knowing how behaviour changes with **`resourceVersion`** interpretation can be crucial to scalability in some cases.

For any **`GET`** request (**`Get()`**, **`GetList()`**, **`Watch()`**)

`resourceVersion = ""`	Most recent data
`resourceVersion = "0"`	Any data (arbitrarily stale)
`resourceVersion = "n"`	Data at **`n`**

There is also **`resourceVersionMatch`** which compliments **`resourceVersion`** in how they are interpreted. You always need to provide this if you specify a **`resourceVersion`** in a **`LIST`** request.

- **`resourceVersionMatch=NotOlderThan`**
- **`resourceVersionMatch=Exact`**

`resourceVersion` semantics

- In each type of CRUD request, you can pass a `resourceVersion` parameter.
- The interpretation of this parameter translates into data consistency guarantees.
- Knowing how behaviour changes with `resourceVersion` interpretation can be crucial to scalability in some cases.

For any `GET` request (`Get()`, `GetList()`, `Watch()`)

`resourceVersion = ""`	Most recent data
`resourceVersion = "0"`	Any data (arbitrarily stale)
`resourceVersion = "n"`	Data at `n`

This still isn't the full picture! Please see:

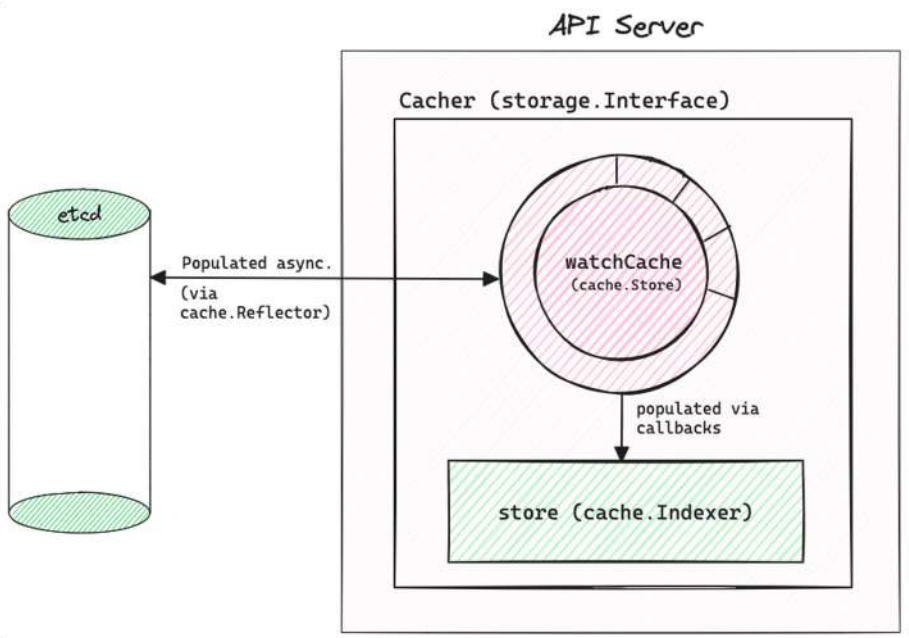
https://kubernetes.io/docs/reference/using-api/api-concepts/#resource-versions

Request Behaviour

The best way to look at how the different layers of the Kubernetes Storage Layer come into play and their scalability aspects, is to look at how different type of requests are served.

Request Behaviour

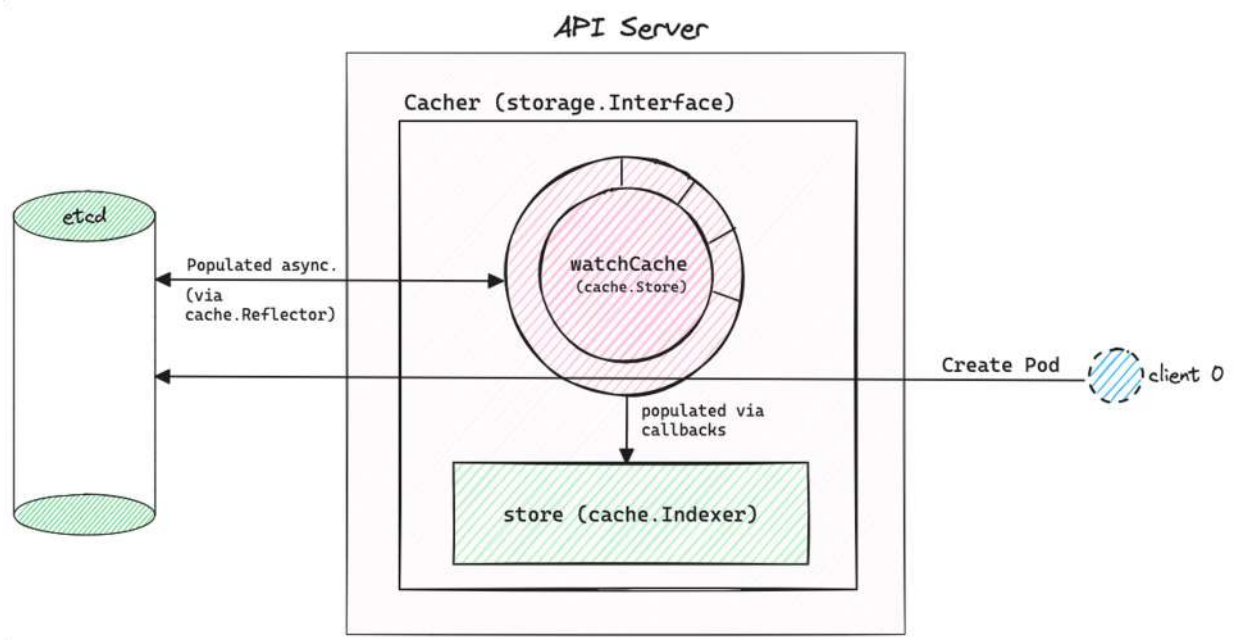
`Create()`



Request Behaviour

`Create()`

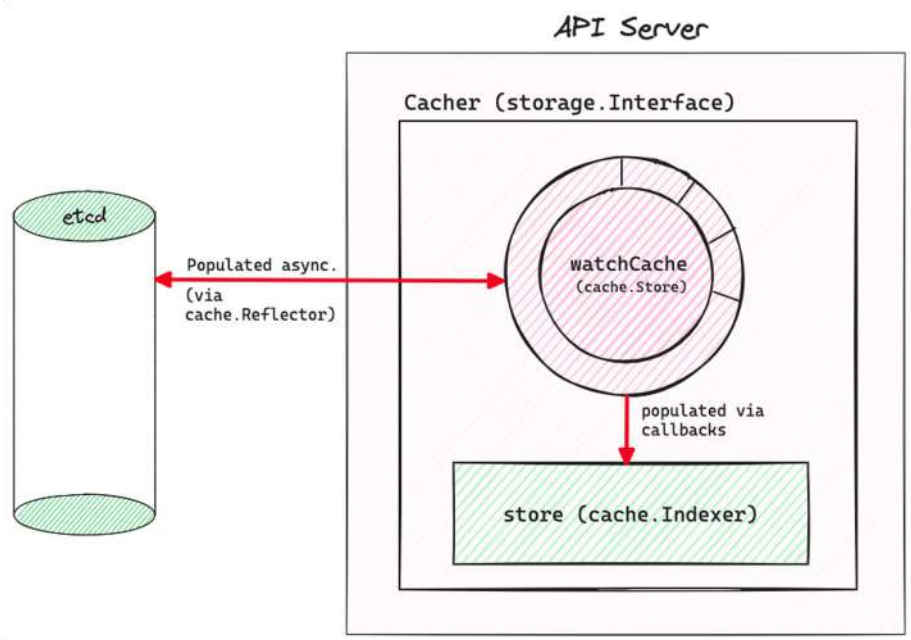
- A **`Create()`** request goes straight to **`etcd`**.



Request Behaviour

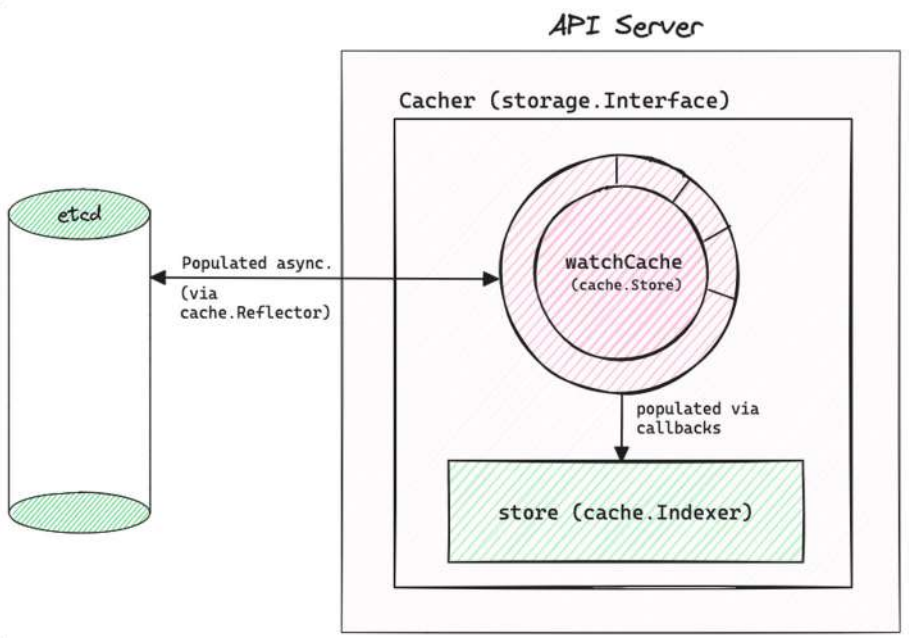
`Create()`

- A **`Create()`** request goes straight to **`etcd`**.
- The created object gets populated in the **`watchCache`** async. because the **`Cacher`** also has a **`WATCH`** open on **`etcd`**.



Request Behaviour

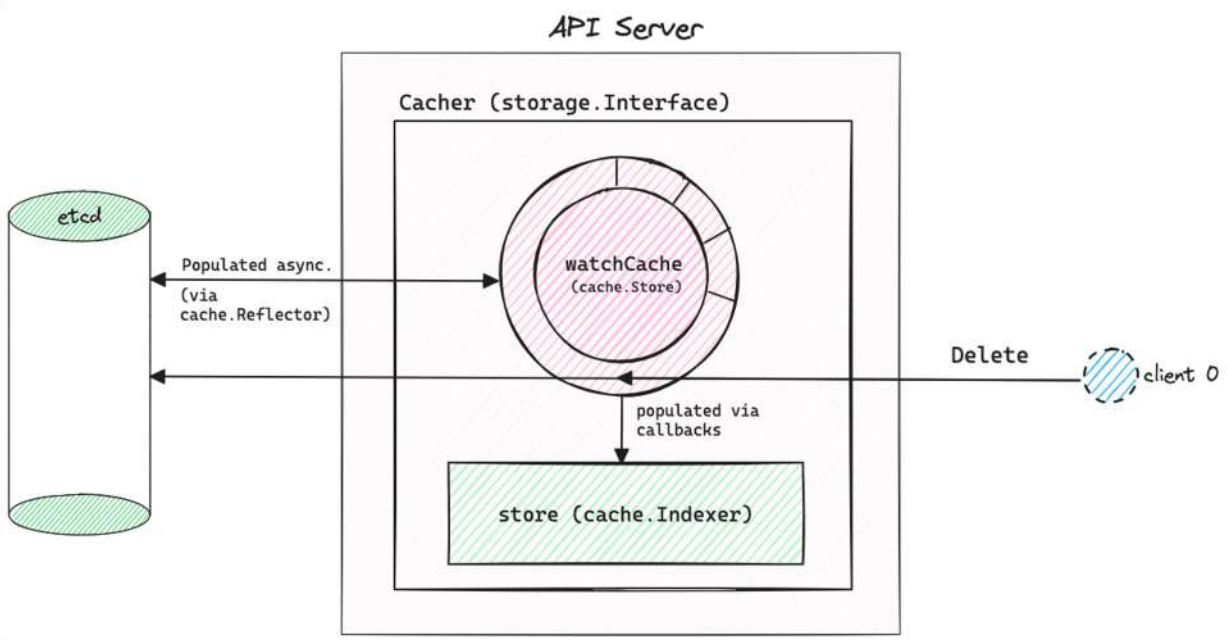
`Delete()`



Request Behaviour

`Delete()`

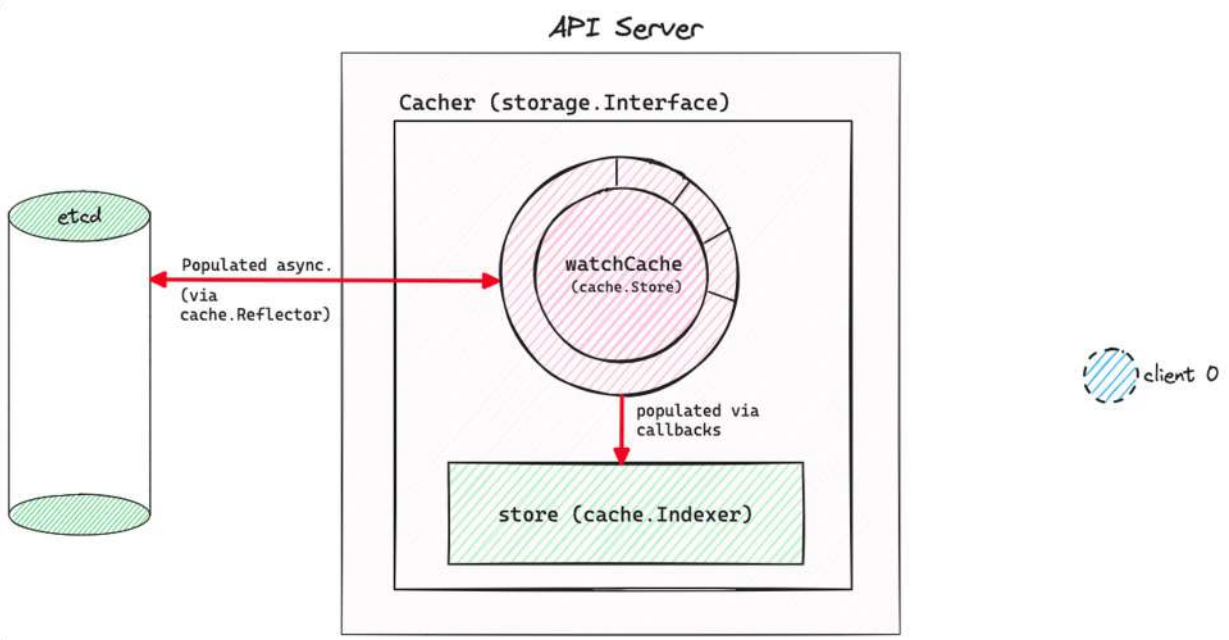
- A **`Delete()`** request tries to delete the version of the object that exists in the **`watchCache`** (performs a read op. (**`GetByKey`**) on the **`watchCache`** before going to **`etcd`**.



Request Behaviour

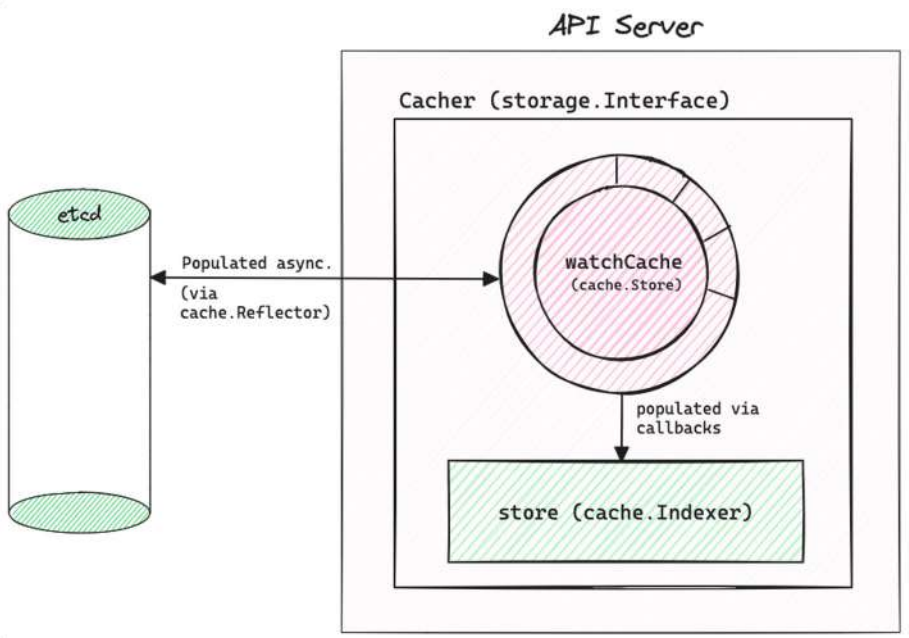
`Delete()`

- A **`Delete()`** request tries to delete the version of the object that exists in the **`watchCache`** (performs a read op. (**`GetByKey`**) on the **`watchCache`** before going to **`etcd`**.
- As usual, the changes are propagated back via the **`WATCH`** on **`etcd`**.



Request Behaviour

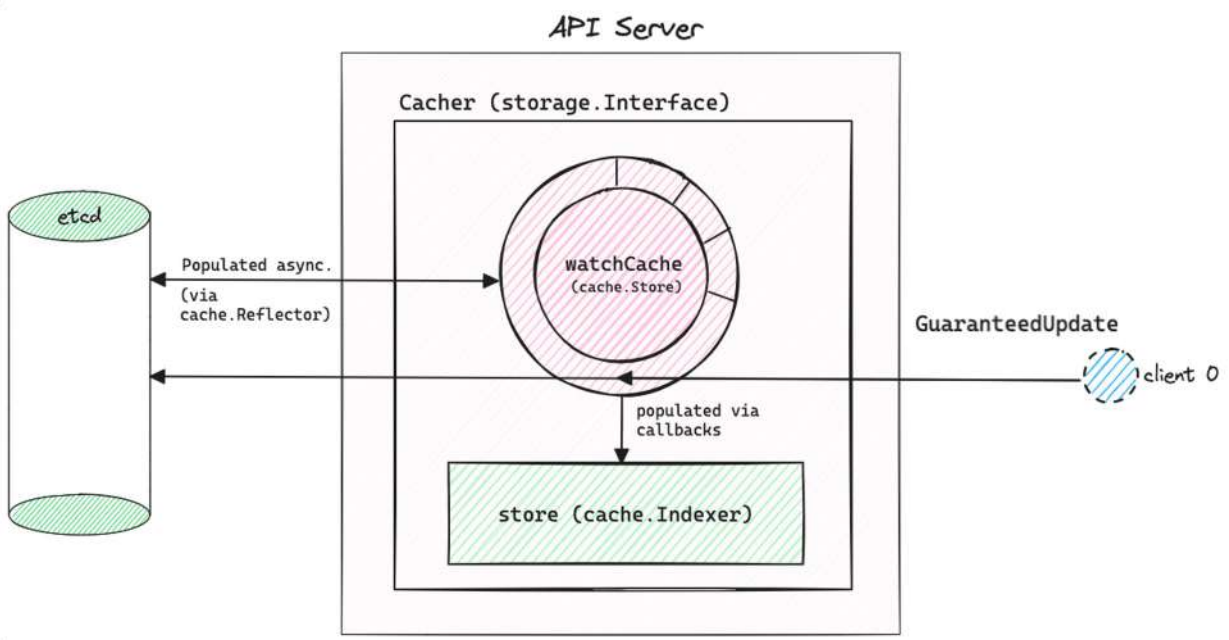
`GuaranteedUpdate()`



Request Behaviour

`GuaranteedUpdate()`

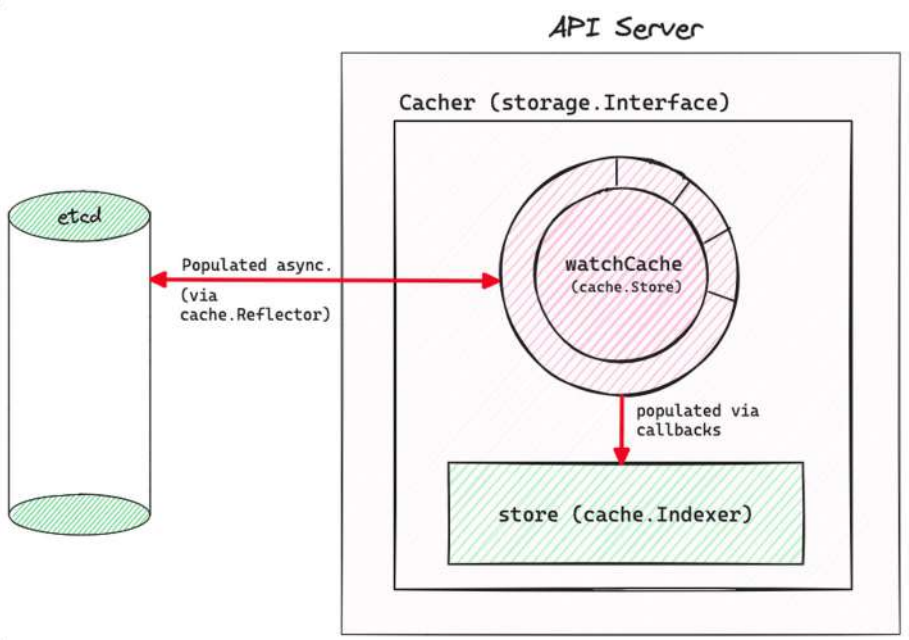
- Similar to **`Delete()`**, we try and update the version of the object that exists in the **`watchCache`**.



Request Behaviour

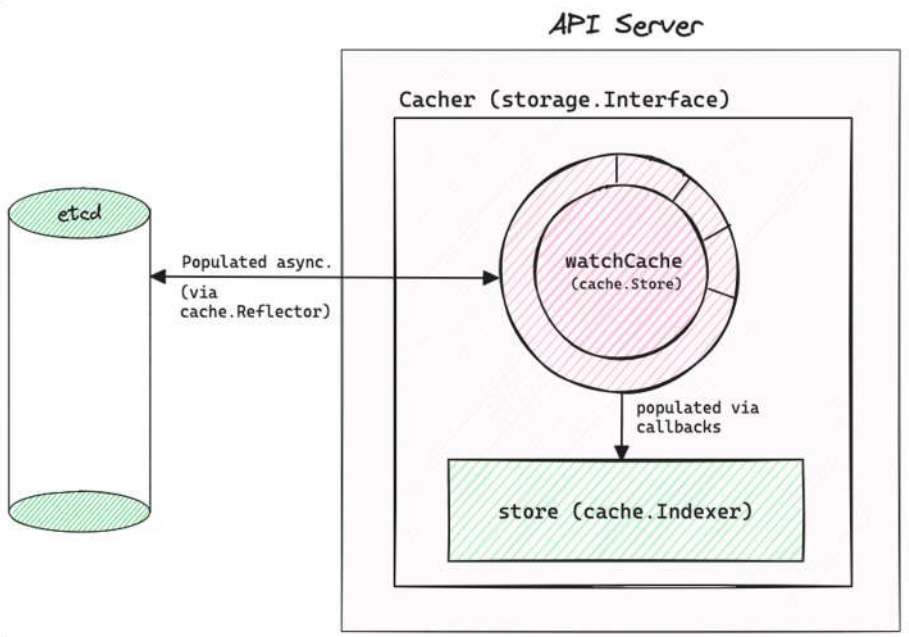
`GuaranteedUpdate()`

- Similar to **`Delete()`**, we try and update the version of the object that exists in the **`watchCache`**.
- As usual, the changes are propagated back via the **`WATCH`** on **`etcd`**.



Request Behaviour

`Get()`

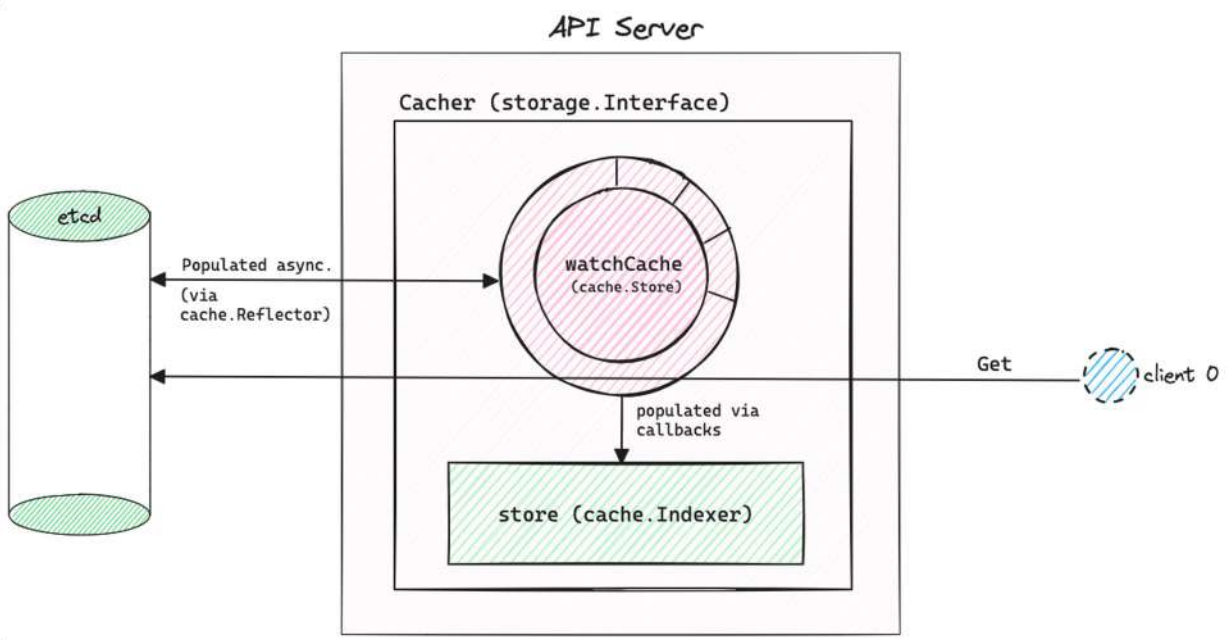


Request Behaviour

`Get()`

If `resourceVersion = ""`

Request goes straight to **`etcd`**, served after a quorum read (linearizable).

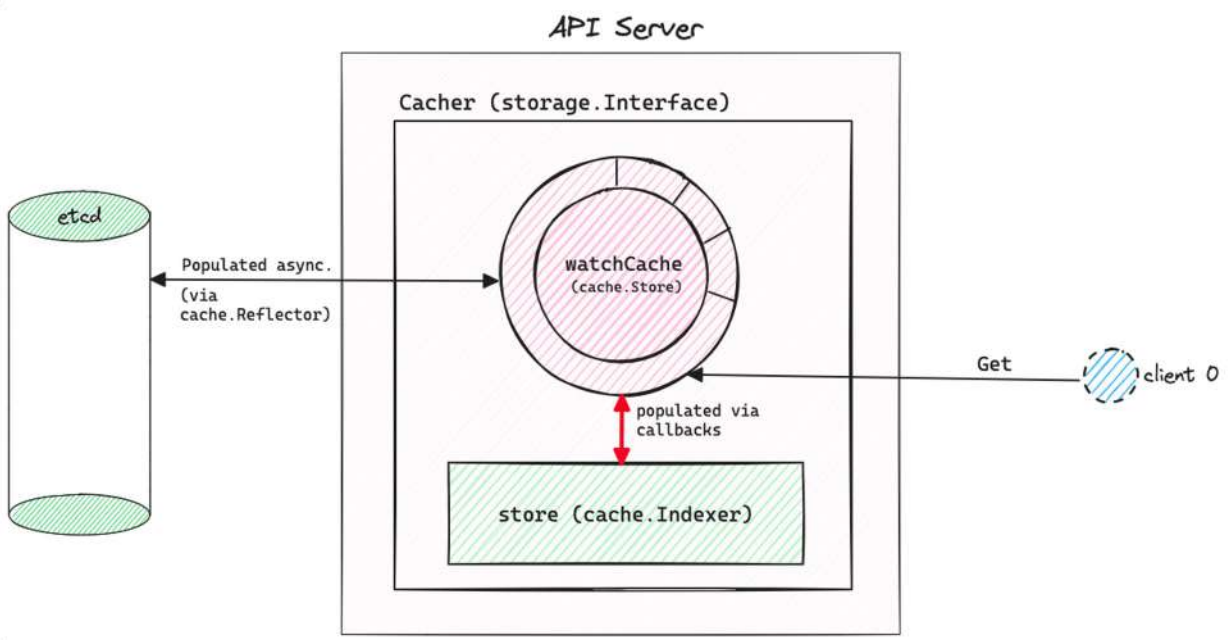


Request Behaviour

`Get()`

If `resourceVersion = "0"`

Request returns after performing a read on the **`watchCache`** (which in turn queries the **`store`**), no concern for freshness of data. Request doesn't reach **`etcd`**.

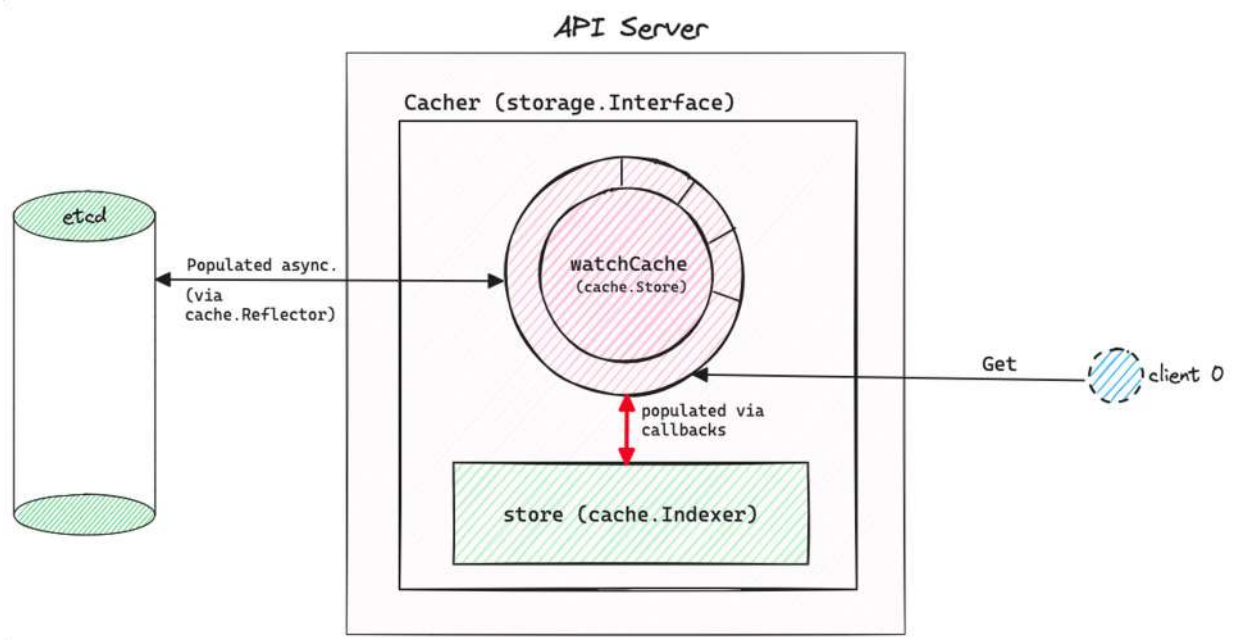


Request Behaviour

`Get()`

If `resourceVersion = "n"; n != "0"`

- We first wait for the cache to become as fresh as `n`.
  - Waiting has a timeout of ~3 seconds.

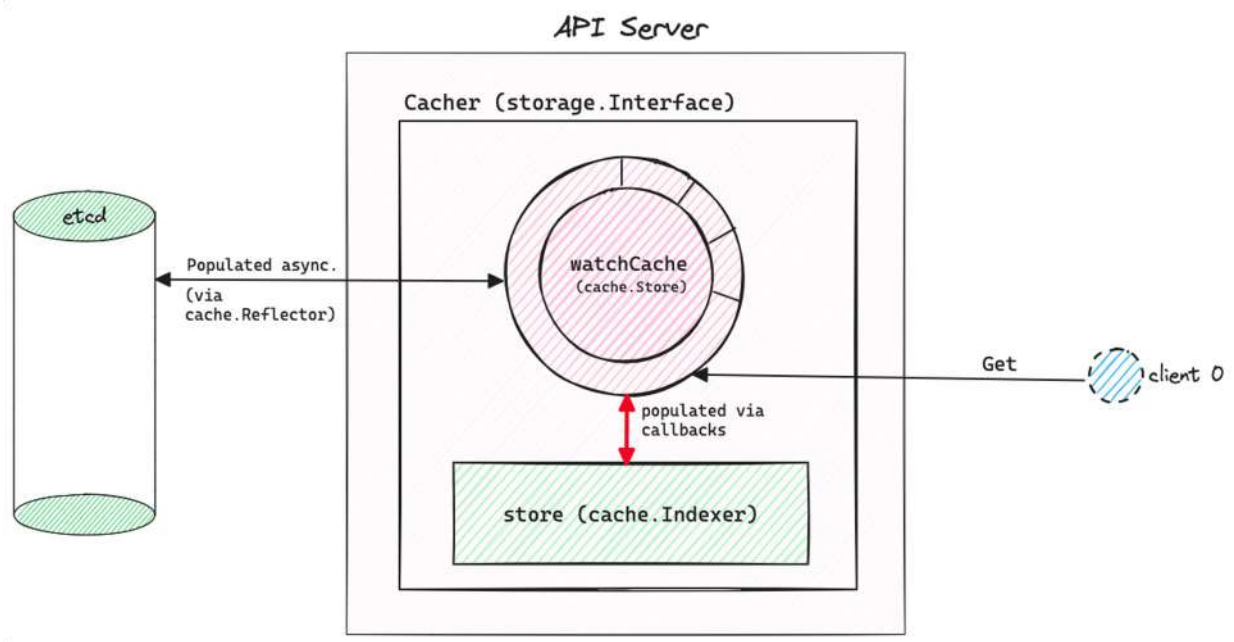


Request Behaviour

`Get()`

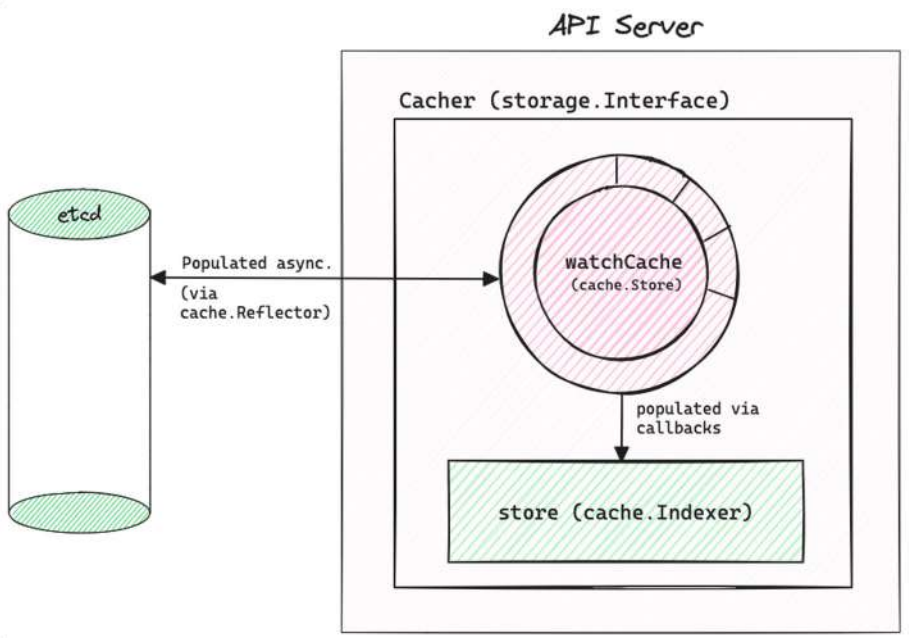
If `resourceVersion = "n"; n != "0"`

- We first wait for the cache to become as fresh as `n`.
  - Waiting has a timeout of ~3 seconds.
- Once that happens, the read happens on the `watchCache` (which queries the underlying `store`) to return the result.



Request Behaviour

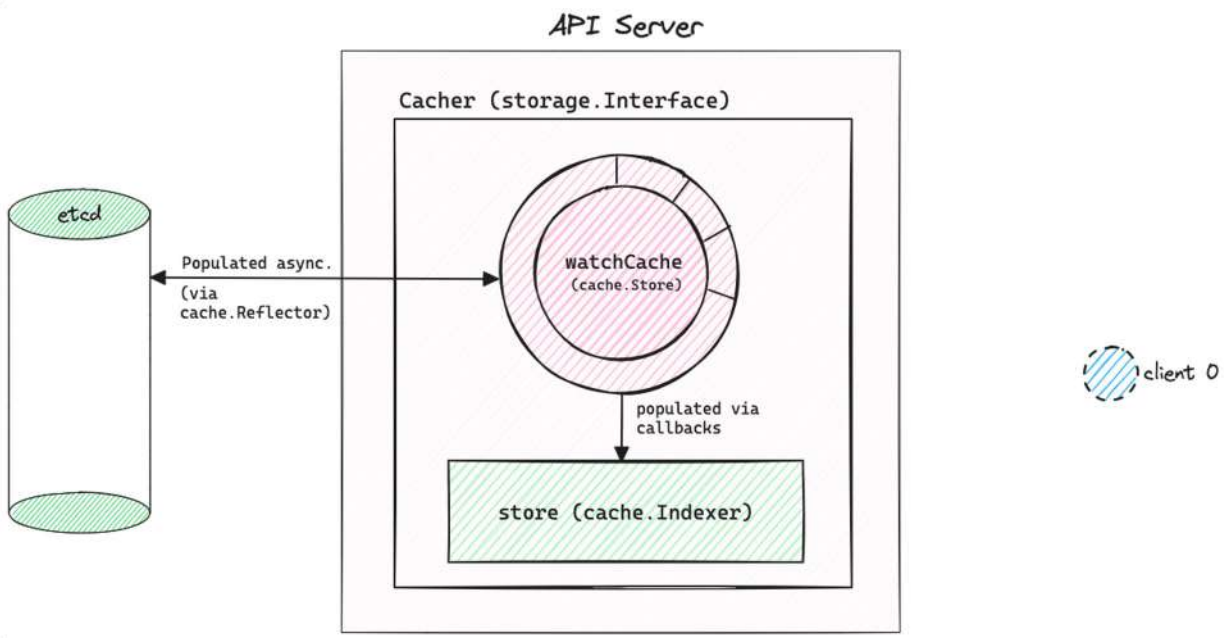
`GetList()`



Request Behaviour

GetList()

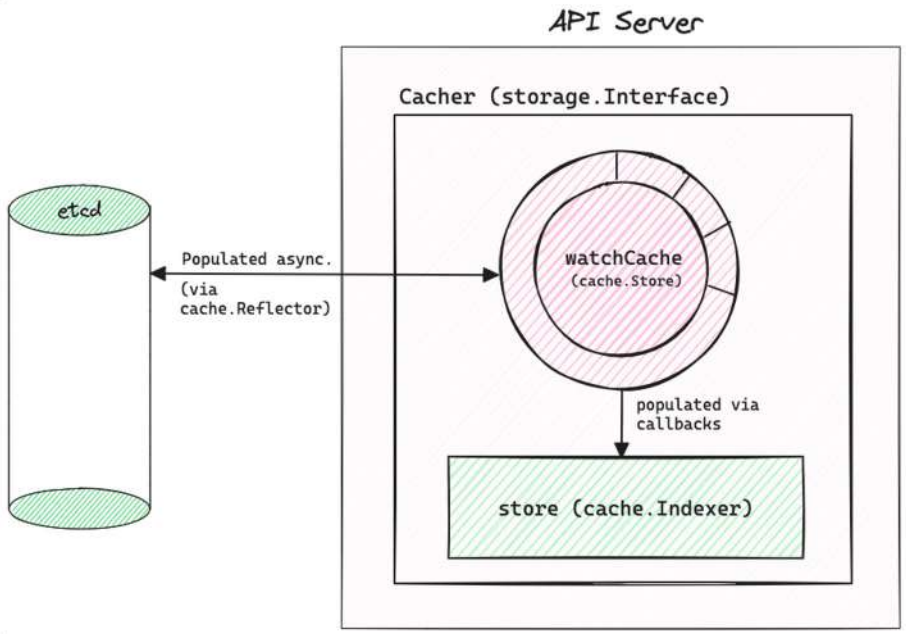
```
func shouldDelegateList(...) bool {

      consistentReadFromStorage := resourceVersion == ""

      hasContinuation := len(pred.Continue) > 0

      hasLimit := pred.Limit > 0 && resourceVersion != "0"

      unsupportedMatch := match != "" && match !=
      metav1.ResourceVersionMatchNotOlderThan


      return consistentReadFromStorage || hasContinuation ||
      hasLimit || unsupportedMatch

}
```



Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {

    consistentReadFromStorage := resourceVersion == ""

    ...

    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch

}
```



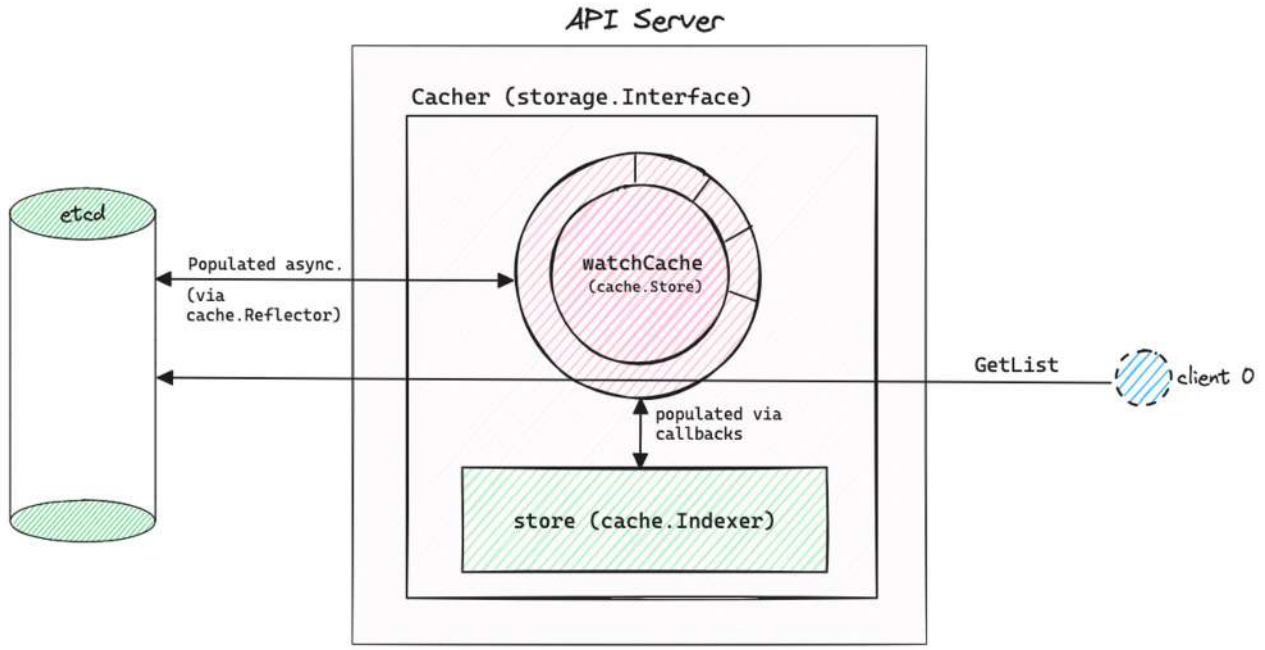
client 0

Request Behaviour

`GetList()`

```
func shouldDelegateList(...) bool {
    consistentReadFromStorage := resourceVersion == ""
    ...
    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch
}
```

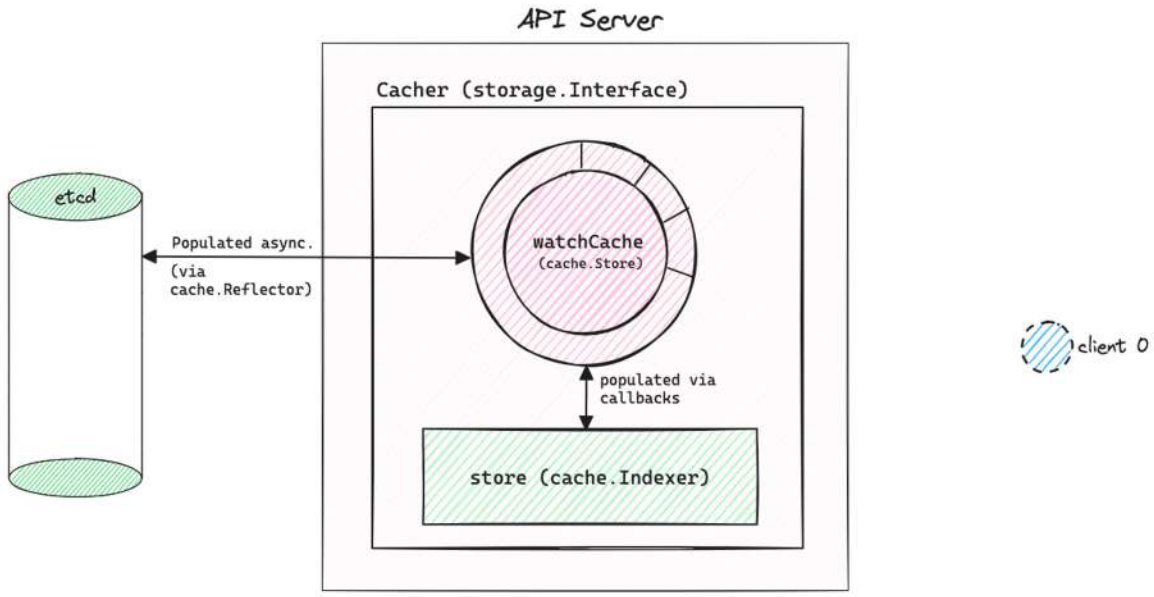
Request goes straight to **`etcd`** and is served as a linearizable read.



Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {

    ...

    hasContinuation := len(pred.Continue) > 0

    ...


    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch

}
```

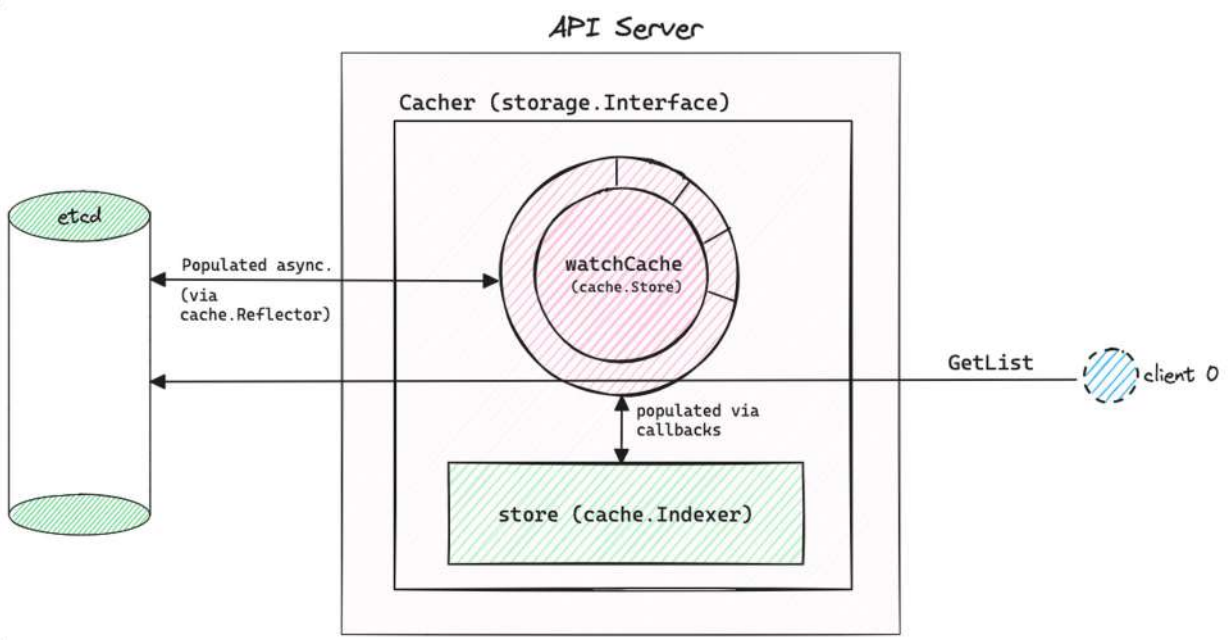


Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {
    ...
    hasContinuation := len(pred.Continue) > 0
    ...

    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch
}
```

- If the **LIST** is a paginated one, no matter what **resourceVersion** you give, the request is going to be served from **etcd**.
- **watchCache** does not support pagination yet.



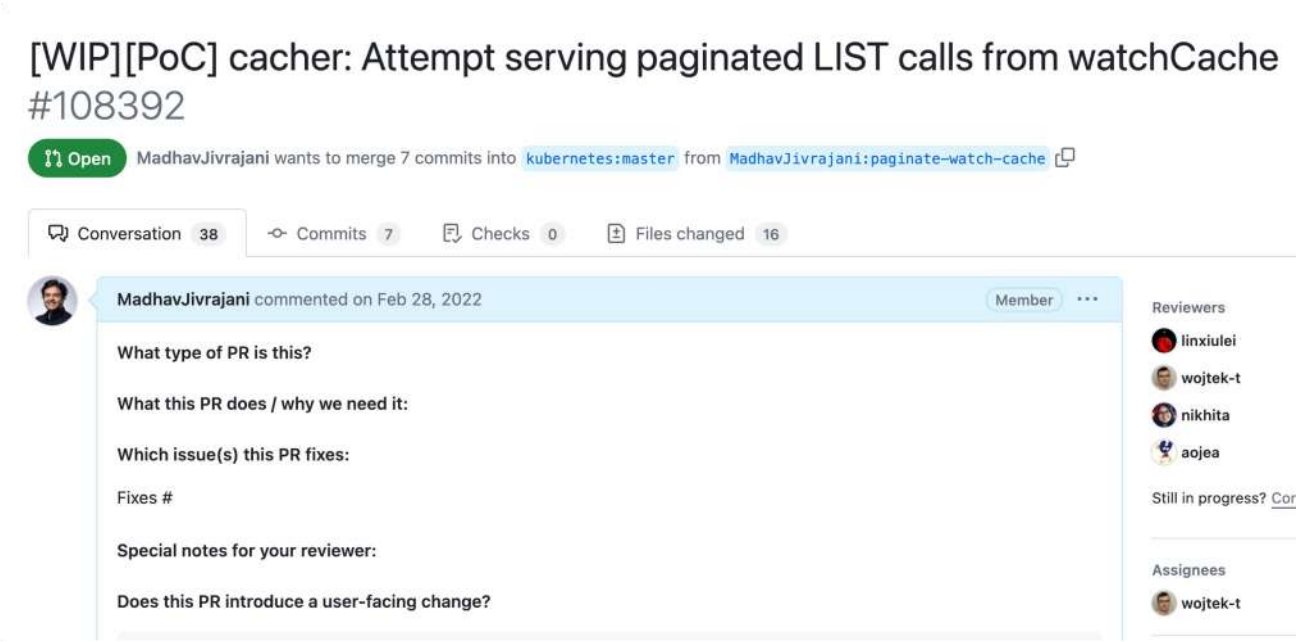
Request Behaviour

KubeCon CloudNativeCon North America 2023

GetList()

```
func shouldDelegateList(...) bool {
    ...
    hasContinuation := len(pred.Continue) > 0
    ...

    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch
}
```

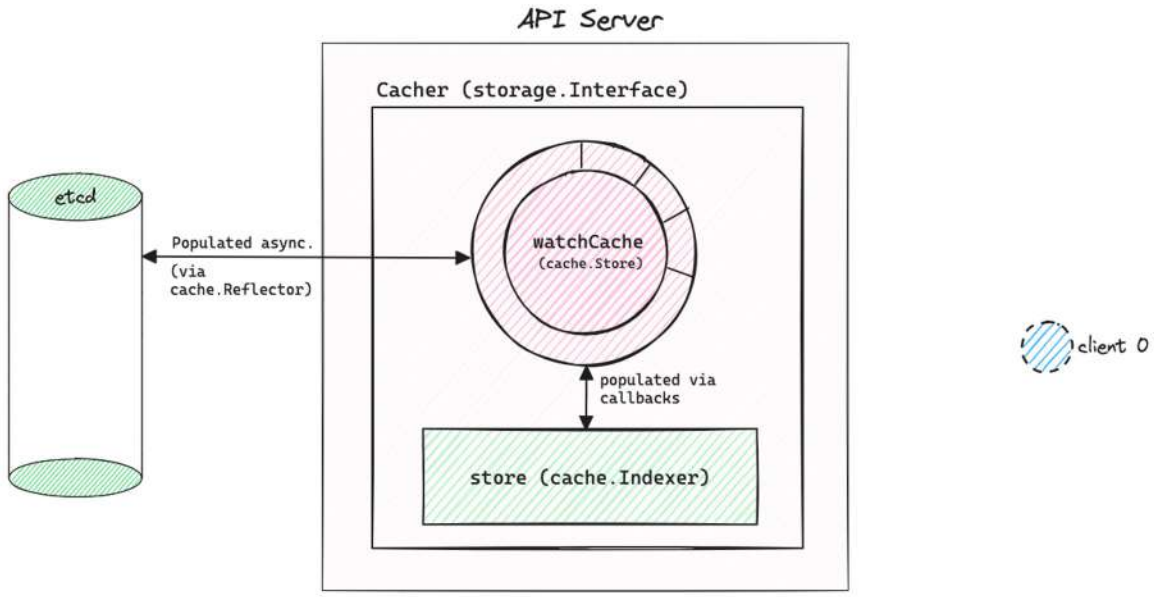
- If the **LIST** is a paginated one, no matter what **resourceVersion** you give, the request is going to be served from **etcd**.
- **watchCache** does not support pagination yet.



Request Behaviour

GetList()

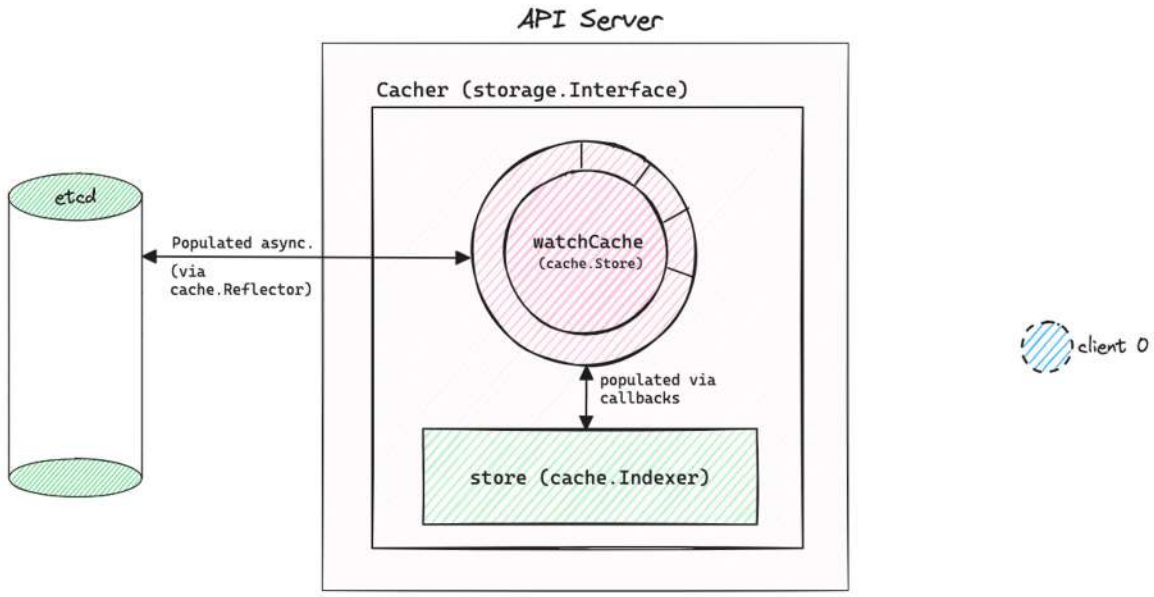
```
func shouldDelegateList(...) bool {

    ...

    hasLimit := pred.Limit > 0 && resourceVersion != "0"

    ...


    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch
}
```



Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {

    ...

    hasLimit := pred.Limit > 0 && resourceVersion != "0"

    ...


    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch
}
```

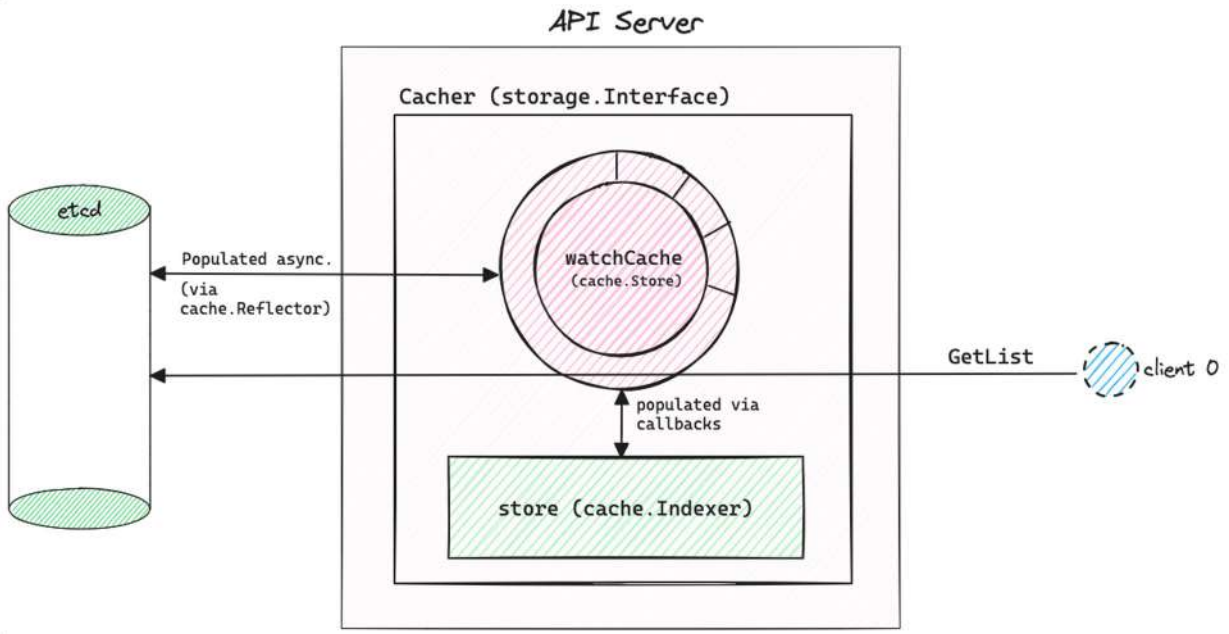


Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {

    ...

    hasLimit := pred.Limit > 0 && resourceVersion != "0"

    ...


    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch
}
```

- If we have a limit set on our **LIST** with a non-zero resourceVersion, we send it to **etcd**.
- Doesn't matter if we have consistent data in the cache or not, we cannot support a **continue** from this limit later anyway.

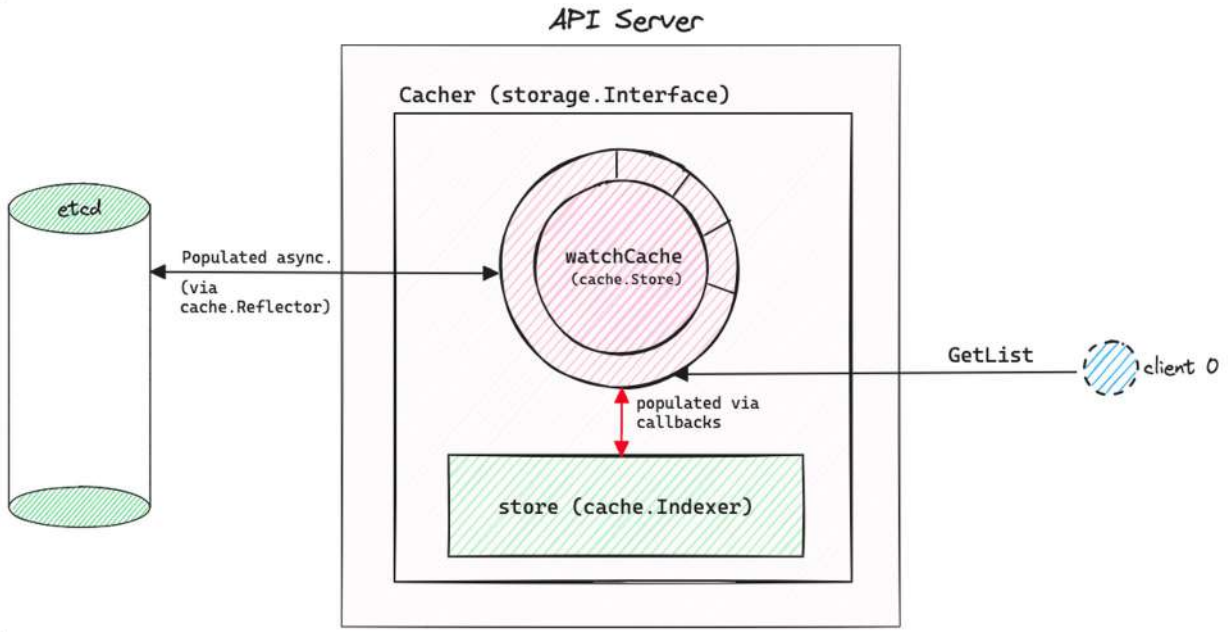


Request Behaviour

`GetList()`

```
func shouldDelegateList(...) bool {

    ...

    hasLimit := pred.Limit > 0 && resourceVersion != "0"

    ...


    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch

}
```

- If no limit is set, we can serve the **`LIST`** from the **`watchCache`** itself.

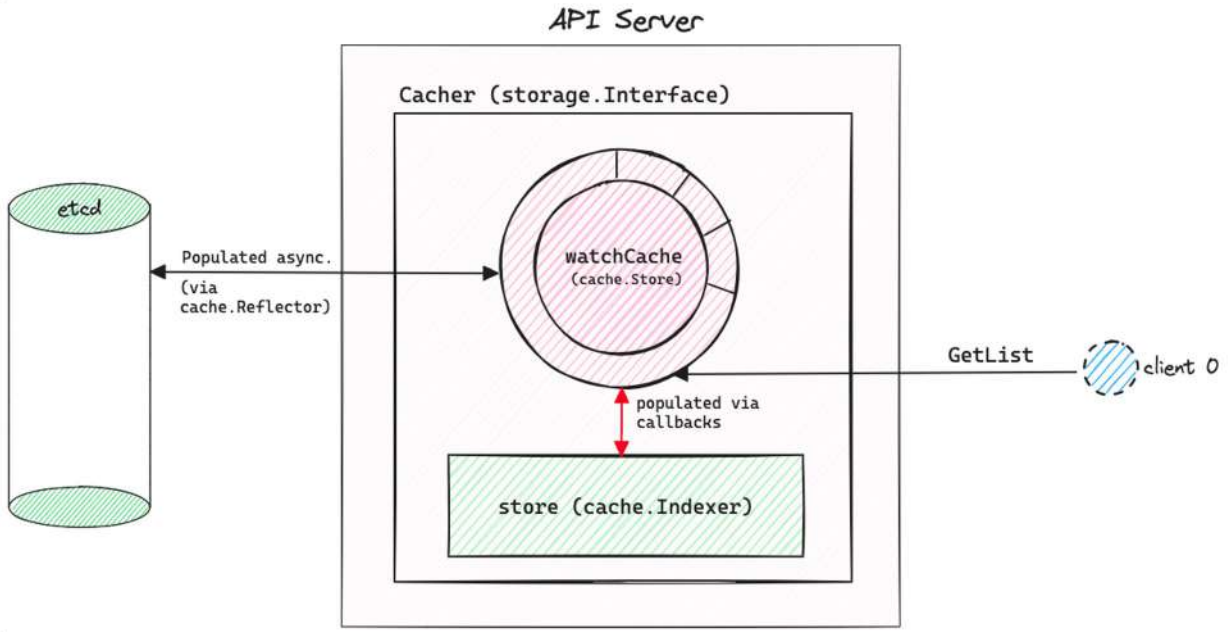


Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {
    ...
    hasLimit := pred.Limit > 0 && resourceVersion != "0"
    ...

    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch
}
```

- But... if we set a limit and put **resourceVersion** as **0**, we essentially ignore the limit and list from the cache anyway? Why?

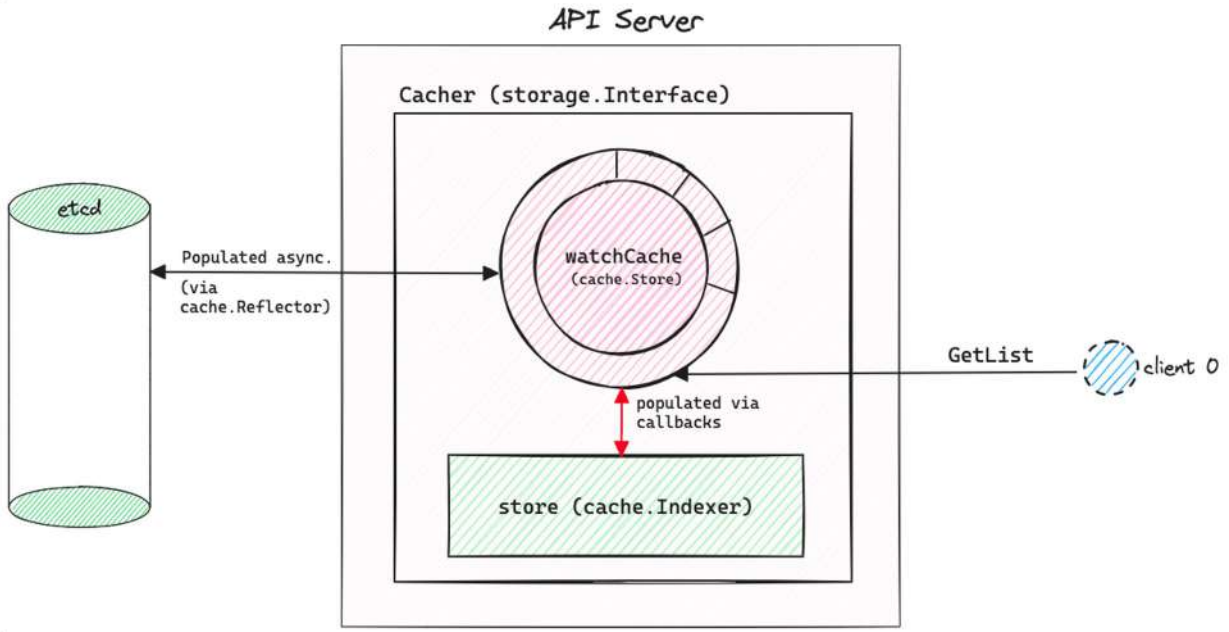


Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {

    ...

    hasLimit := pred.Limit > 0 && resourceVersion != "0"

    ...


    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch

}
```

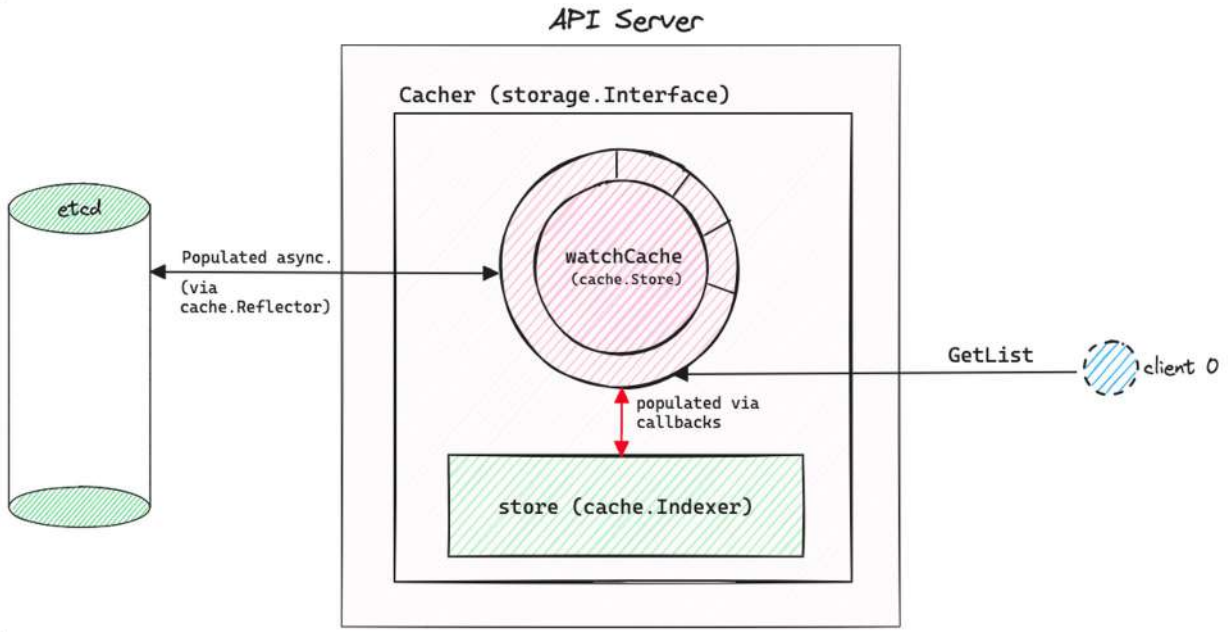
- Well… **`resourceVersion="0"`** is “Any data” semantics, so cache makes sense.



Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {

    ...

    hasLimit := pred.Limit > 0 && resourceVersion != "0"

    ...


    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch

}
```

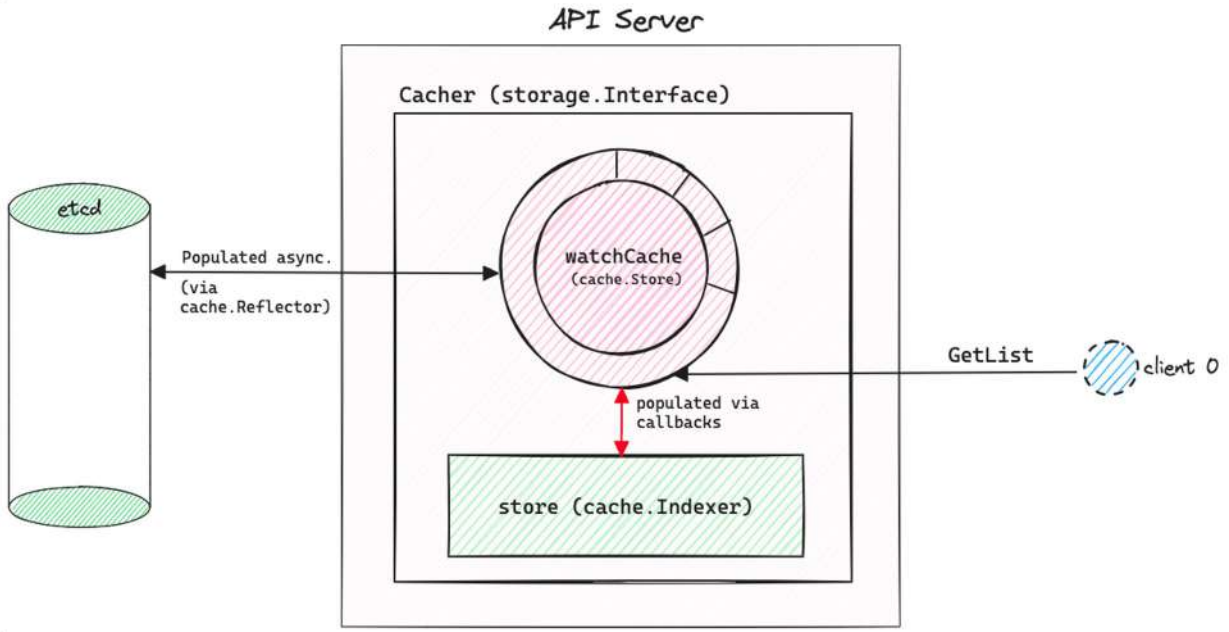


Well… **`resourceVersion="0"`** is “Any data” semantics, so cache makes sense

Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {

      ...

      hasLimit := pred.Limit > 0 && resourceVersion != "0"

      ...


      return consistentReadFromStorage || hasContinuation ||
      hasLimit || unsupportedMatch

}
```

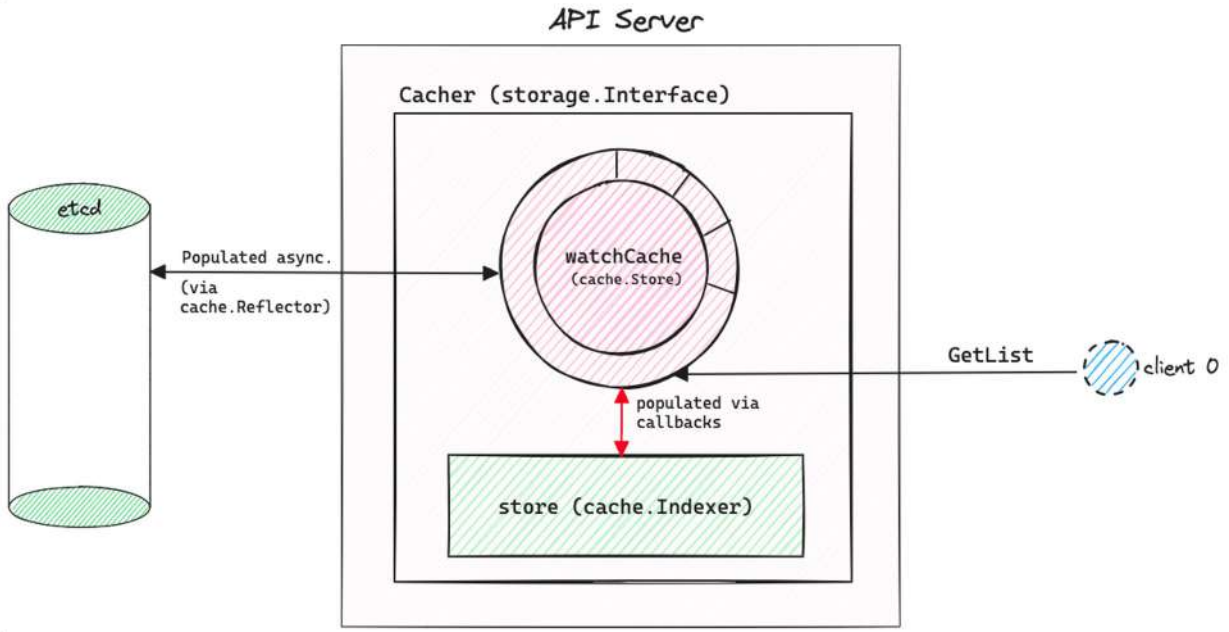


More importantly, it allows us to support listing whose responses we know have a good chance of being massive thus reducing the load on **etcd**, i.e. initial lists.

Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {

      ...

      hasLimit := pred.Limit > 0 && resourceVersion != "0"

      ...


      return consistentReadFromStorage || hasContinuation ||
      hasLimit || unsupportedMatch

}
```

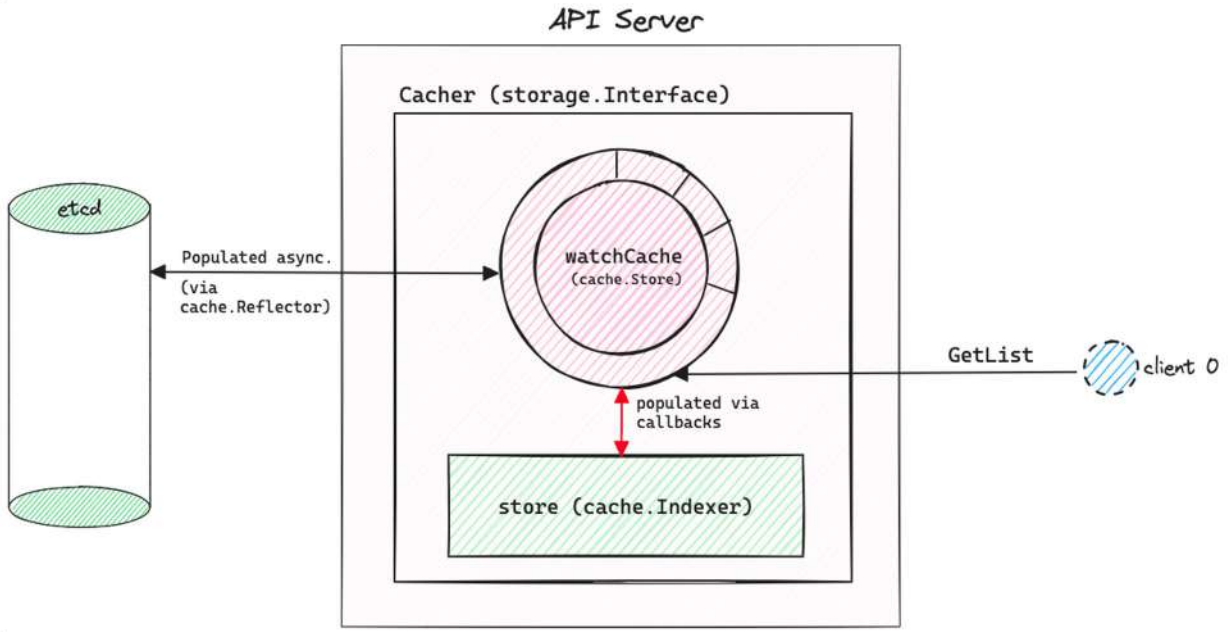


Ex: - a ~large cluster can have O(1000) nodes, each node having O(100) pods, so if a **`kubelet`** or a **`StatefulSet`** controller were to perform a list on the pods…

Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {

      ...

      hasLimit := pred.Limit > 0 && resourceVersion != "0"

      ...


      return consistentReadFromStorage || hasContinuation ||
      hasLimit || unsupportedMatch

}
```

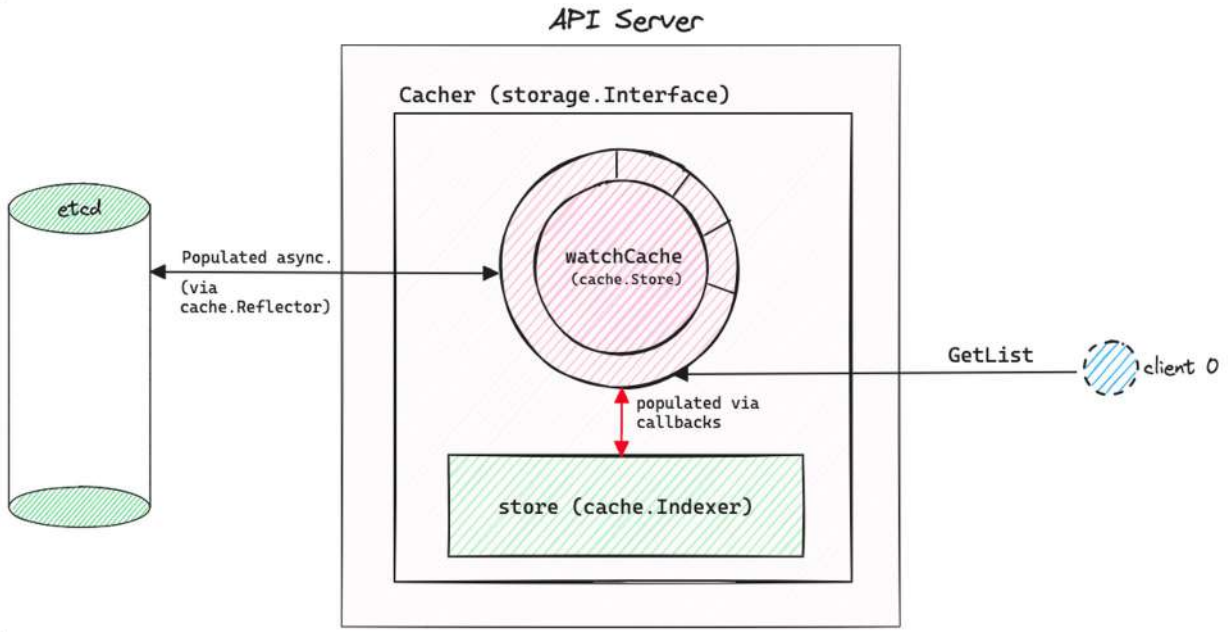


Clients that support **List/Watch** functionality (**client-go reflectors**) ensure to put **resourceVersion** as **0** when performing the first list.

Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {

    ...

    unsupportedMatch := match != "" && match !=
    metav1.ResourceVersionMatchNotOlderThan


    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch

}
```

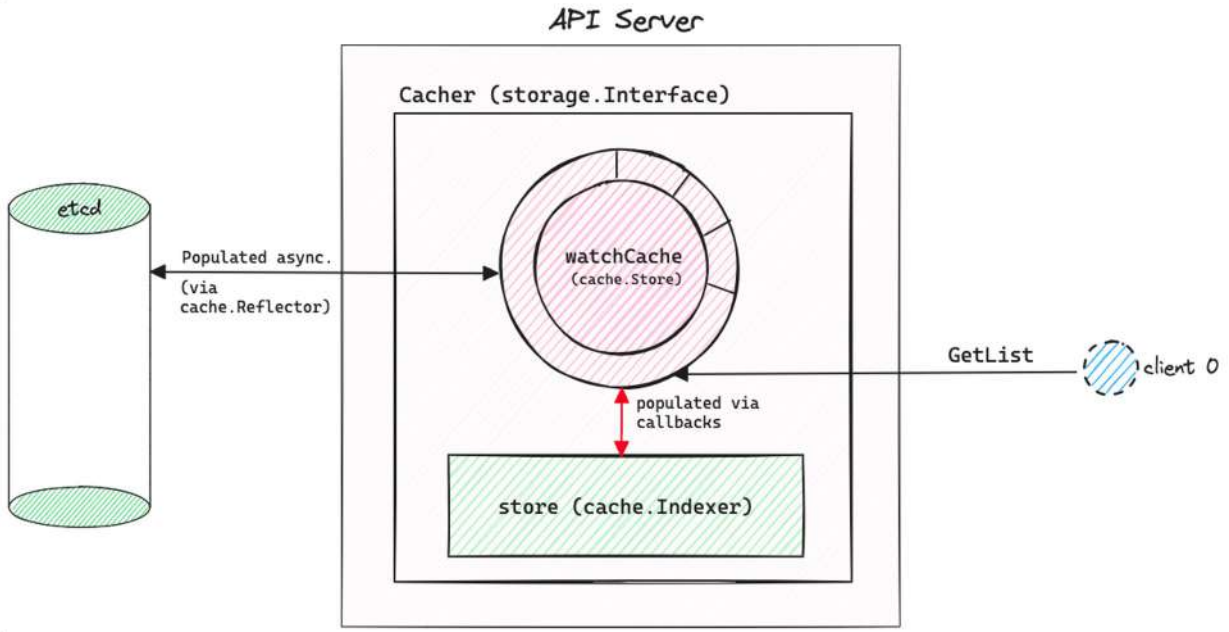


Request Behaviour

GetList()

```
func shouldDelegateList(...) bool {

    ...

    unsupportedMatch := match != "" && match !=
    metav1.ResourceVersionMatchNotOlderThan


    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch

}
```

- The **watchCache** only supports **NotOlderThan**, so if that is set, we serve the list from the **watchCache.**

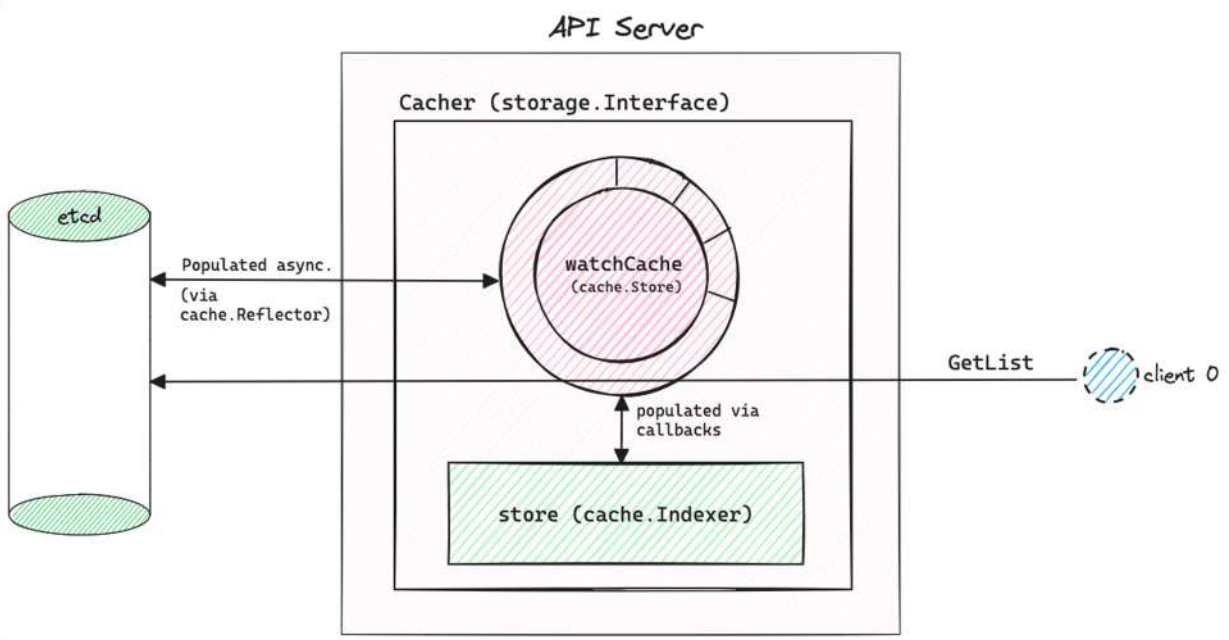


Request Behaviour

`GetList()`

```
func shouldDelegateList(...) bool {

    ...

    unsupportedMatch := match != "" && match !=
    metav1.ResourceVersionMatchNotOlderThan


    return consistentReadFromStorage || hasContinuation ||
    hasLimit || unsupportedMatch

}
```

- If not, we serve the list from **`etcd`**, honouring exact semantics.

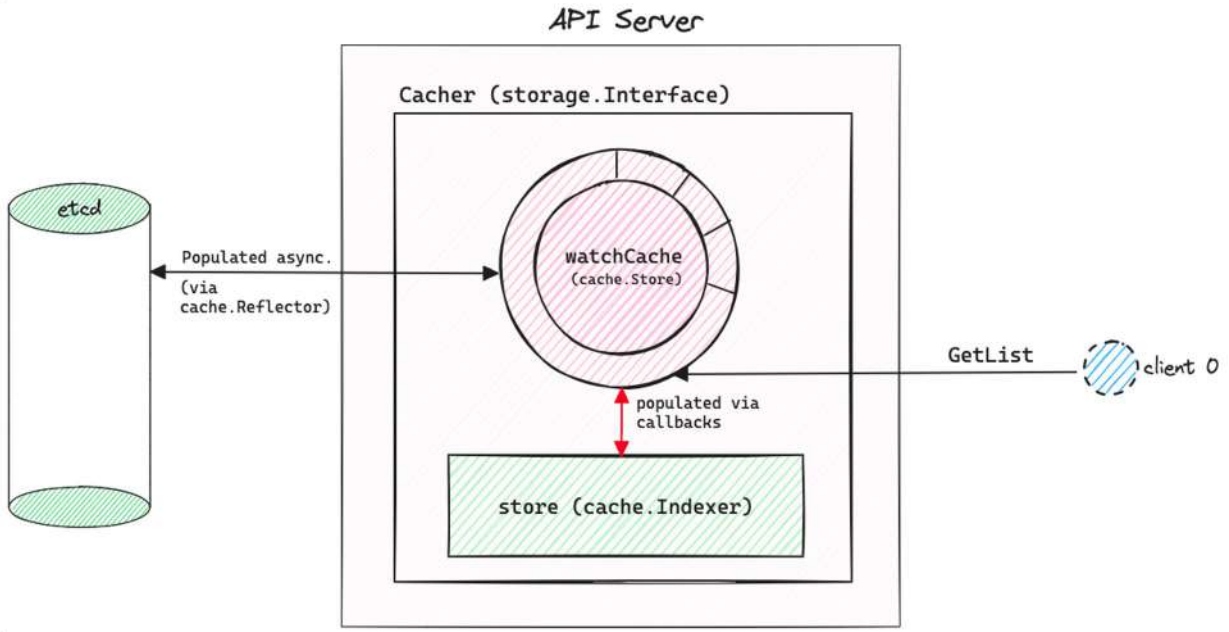


Request Behaviour

`GetList()`

```
func shouldDelegateList(...) bool {

      ...

      return consistentReadFromStorage || hasContinuation ||
      hasLimit || unsupportedMatch

}
```

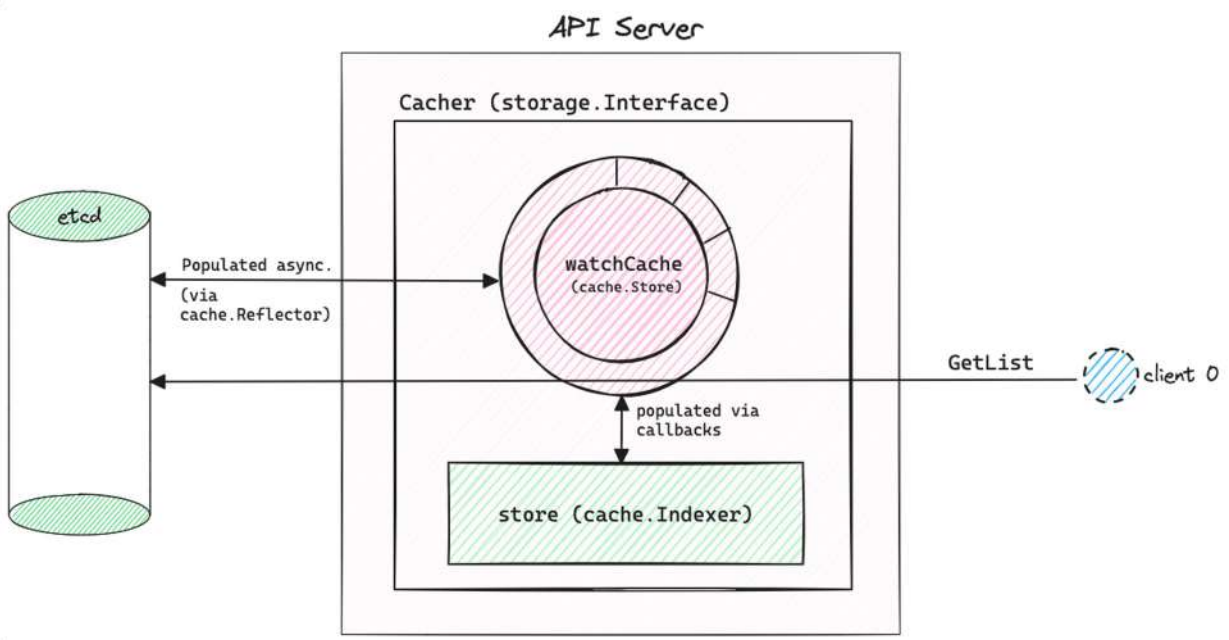
- The only time we serve a list from the **`watchCache`** if we specify a non-empty **`resourceVersion`**
- AND it is not a paginated list (no limit or continue).
- AND we specify **`NotOlderThan`** semantics.



Request Behaviour

`GetList()`

There's a few gotchas to keep in mind here!

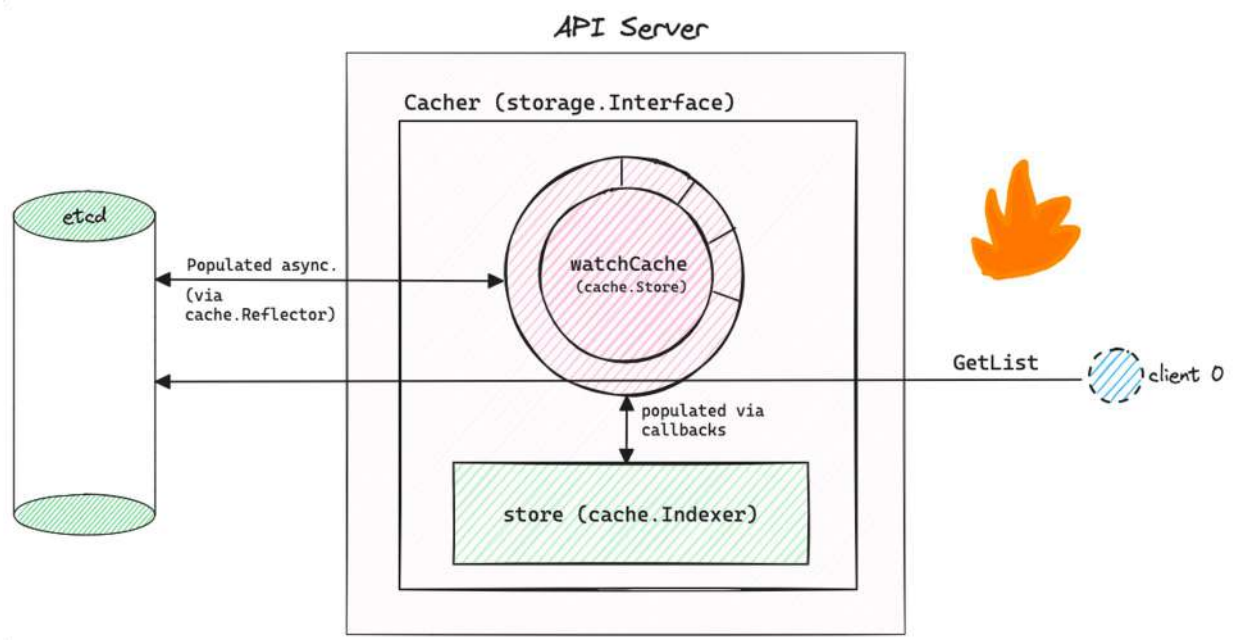


Request Behaviour

`GetList()`

There's a few gotchas to keep in mind here!

- When you need consistent **`LISTs`**, and the request goes to **`etcd`**, the API Server can see spikes of unbounded memory growth depending on response sizes.

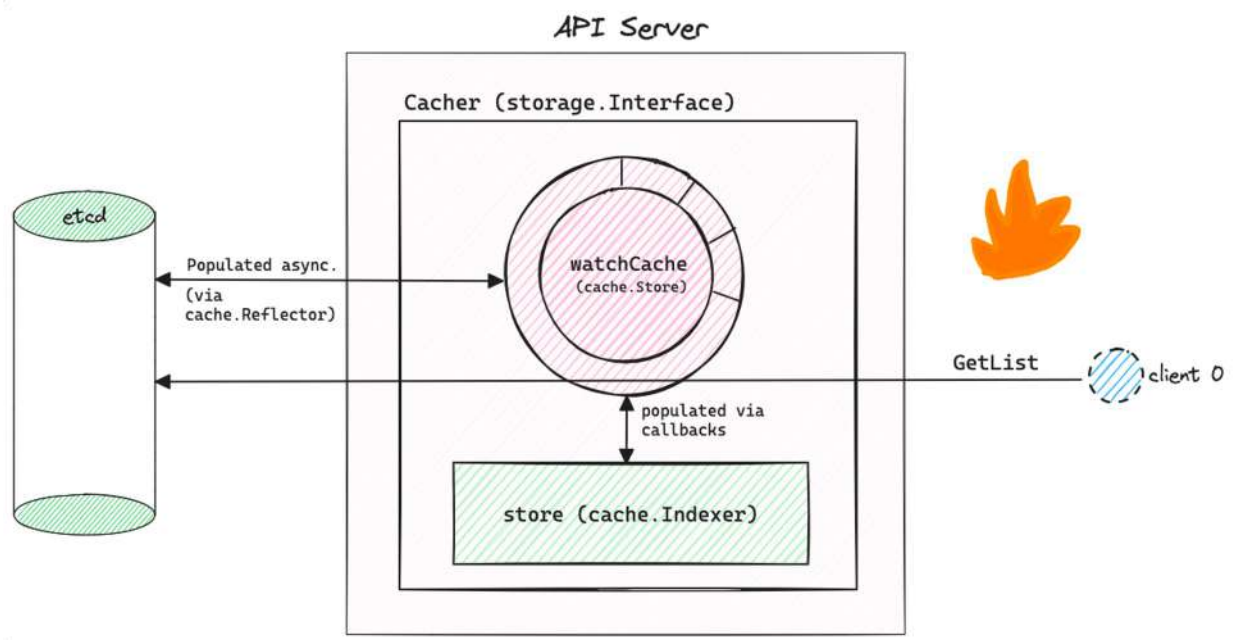


Request Behaviour

`GetList()`

There's a few gotchas to keep in mind here!

- When you need consistent **`LISTs`**, and the request goes to **`etcd`**, the API Server can see spikes of unbounded memory growth depending on response sizes.
- Data needs to be fetched from **`etcd`**, unmarshalled, conversions take place, response is prepared.

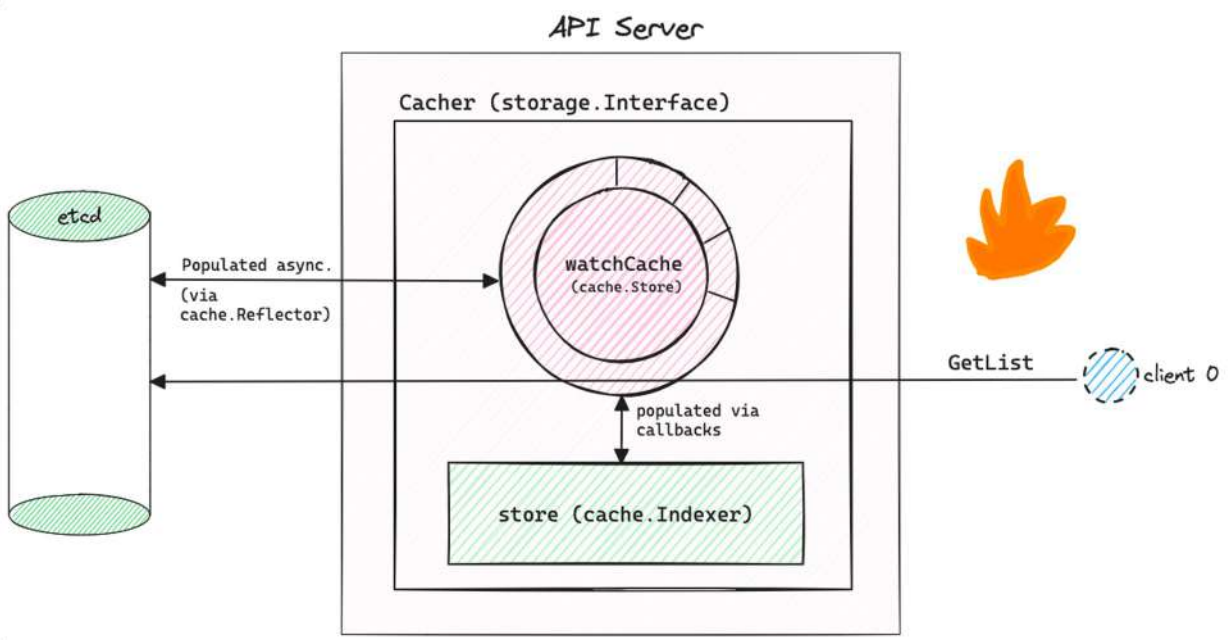


Request Behaviour

`GetList()`

There's a few gotchas to keep in mind here!

- When you need consistent **`LISTs`**, and the request goes to **`etcd`**, the API Server can see spikes of unbounded memory growth depending on response sizes.
- Data needs to be fetched from **`etcd`**, unmarshalled, conversions take place, response is prepared.
- Sometimes, paginating responses also will not help, if each chunk itself is large.



Request Behaviour

`GetList()`

- KEP-3157 proposes, for informers, streaming data from **`watchCache`** rather than paging in **`etcd`**.

KEP-3157: allow informers for getting a stream of data instead of chunking.

https://github.com/kubernetes/enhancements/tree/master/keps/sig-api-machinery/3157-watch-list

Request Behaviour

`GetList()`

- KEP-3157 proposes, for informers, streaming data from **`watchCache`** rather than paging in **`etcd`**.
- Predictable memory footprint irrespective of **`LIST`** response sizes and consistency requirements.

KEP-3157: allow informers for getting a stream of data instead of chunking.

https://github.com/kubernetes/enhancements/tree/master/keps/sig-api-machinery/3157-watch-list

Request Behaviour

`GetList()`

- KEP-3157 proposes, for informers, streaming data from **`watchCache`** rather than paging in **`etcd`**.
- Predictable memory footprint irrespective of **`LIST`** response sizes and consistency requirements.
- Handles the lack of pagination in **`watchCache`**.

KEP-3157: allow informers for getting a stream of data instead of chunking.

- Release Signoff Checklist
- Summary
- Motivation
  - Goals
  - Non-Goals
- Proposal
  - Risks and Mitigations
- Design Details
  - Required changes for a WATCH request with the SendInitialEvents=true
    - API changes
    - Important optimisations
    - Manual testing without the changes in place
    - Results with WATCH-LIST
  - Required changes for a WATCH request with the RV set to the last observed value (RV > 0)
  - Provide a fix for the long-standing issue kubernetes/kubernetes#59848
  - Test Plan
    - Prerequisite testing updates

https://github.com/kubernetes/enhancements/tree/master/keps/sig-api-machinery/3157-watch-list

Request Behaviour

`GetList()`

- KEP-3157 proposes, for informers, streaming data from **`watchCache`** rather than paging in **`etcd`**.
- Predictable memory footprint irrespective of **`LIST`** response sizes and consistency requirements.
- Handles the lack of pagination in **`watchCache.`**

This is set to be in Alpha as of Kubernetes v1.28, please try it out and provide feedback!

(Feature Gate: **`WatchList`**)

KEP-3157: allow informers for getting a stream of data instead of chunking.

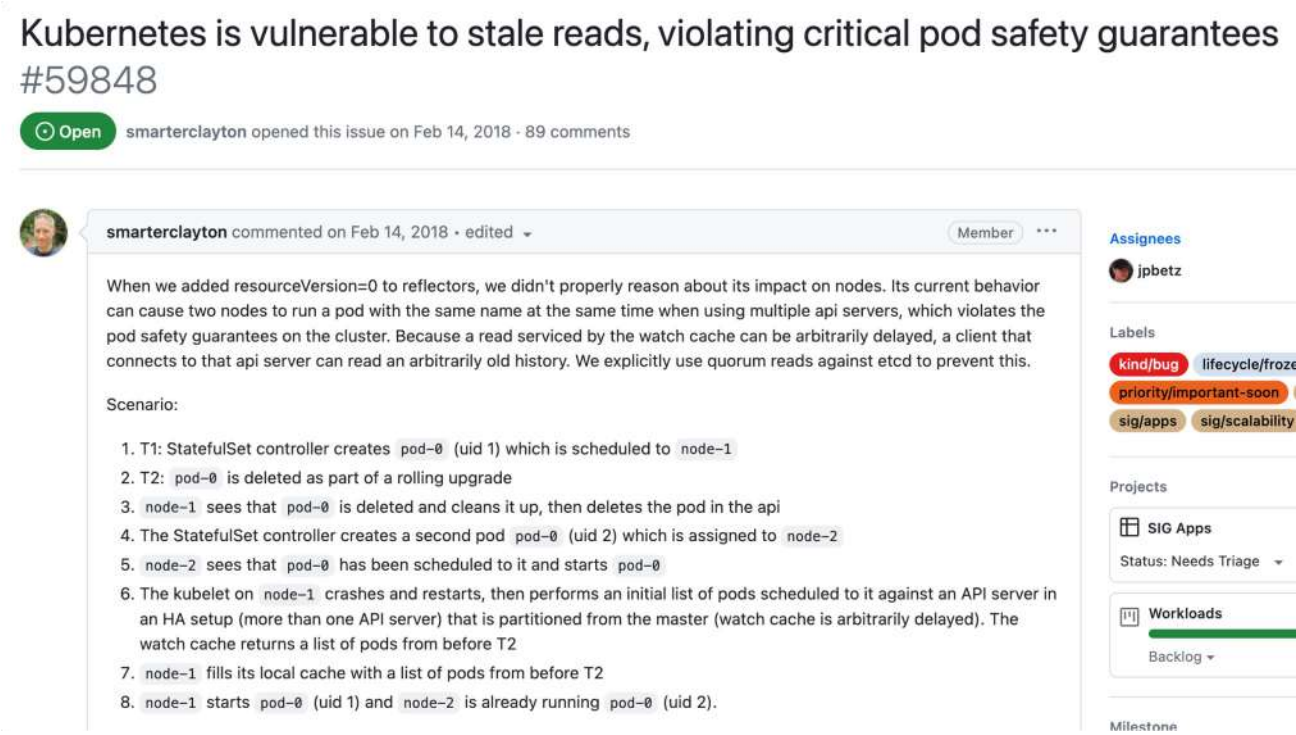
- Release Signoff Checklist
- Summary
- Motivation
  - Goals
  - Non-Goals
- Proposal
  - Risks and Mitigations
- Design Details
  - Required changes for a WATCH request with the SendInitialEvents=true
    - API changes
    - Important optimisations
    - Manual testing without the changes in place
    - Results with WATCH-LIST
  - Required changes for a WATCH request with the RV set to the last observed value (RV > 0)
  - Provide a fix for the long-standing issue kubernetes/kubernetes#59848
  - Test Plan
    - Prerequisite testing updates

https://github.com/kubernetes/enhancements/tree/master/keps/sig-api-machinery/3157-watch-list

Request Behaviour

`GetList()`

Another gotcha - time travelling, stale reads from `watchCache`!



Kubernetes is vulnerable to stale reads, violating critical pod safety guarantees #59848

Open smarterclayton opened this issue on Feb 14, 2018 · 89 comments

smarterclayton commented on Feb 14, 2018 • edited — Member

When we added resourceVersion=0 to reflectors, we didn't properly reason about its impact on nodes. Its current behavior can cause two nodes to run a pod with the same name at the same time when using multiple api servers, which violates the pod safety guarantees on the cluster. Because a read serviced by the watch cache can be arbitrarily delayed, a client that connects to that api server can read an arbitrarily old history. We explicitly use quorum reads against etcd to prevent this.

Scenario:

1. T1: StatefulSet controller creates `pod-0` (uid 1) which is scheduled to `node-1`
2. T2: `pod-0` is deleted as part of a rolling upgrade
3. `node-1` sees that `pod-0` is deleted and cleans it up, then deletes the pod in the api
4. The StatefulSet controller creates a second pod `pod-0` (uid 2) which is assigned to `node-2`
5. `node-2` sees that `pod-0` has been scheduled to it and starts `pod-0`
6. The kubelet on `node-1` crashes and restarts, then performs an initial list of pods scheduled to it against an API server in an HA setup (more than one API server) that is partitioned from the master (watch cache is arbitrarily delayed). The watch cache returns a list of pods from before T2
7. `node-1` fills its local cache with a list of pods from before T2
8. `node-1` starts `pod-0` (uid 1) and `node-2` is already running `pod-0` (uid 2).

Assignees: jpbetz

Labels: kind/bug, lifecycle/froze, priority/important-soon, sig/apps, sig/scalability

Projects: SIG Apps — Status: Needs Triage; Workloads — Backlog

Milestone

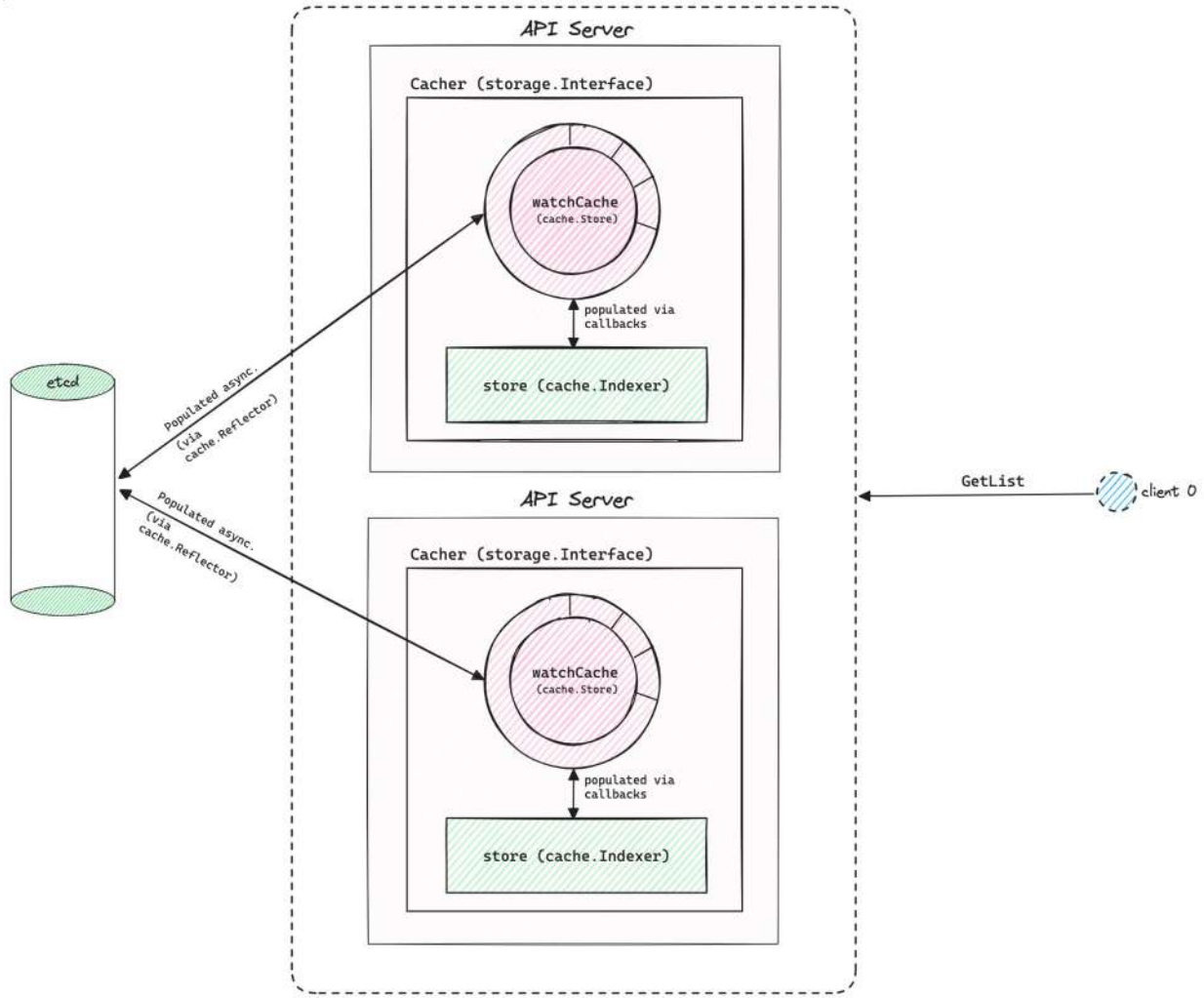
https://github.com/kubernetes/kubernetes/issues/59848

Request Behaviour

`GetList()`

Another gotcha - time travelling, stale reads from **`watchCache`**!

- If you have an HA setup, with **`watchCache`** enabled, one of them can be far behind the other.

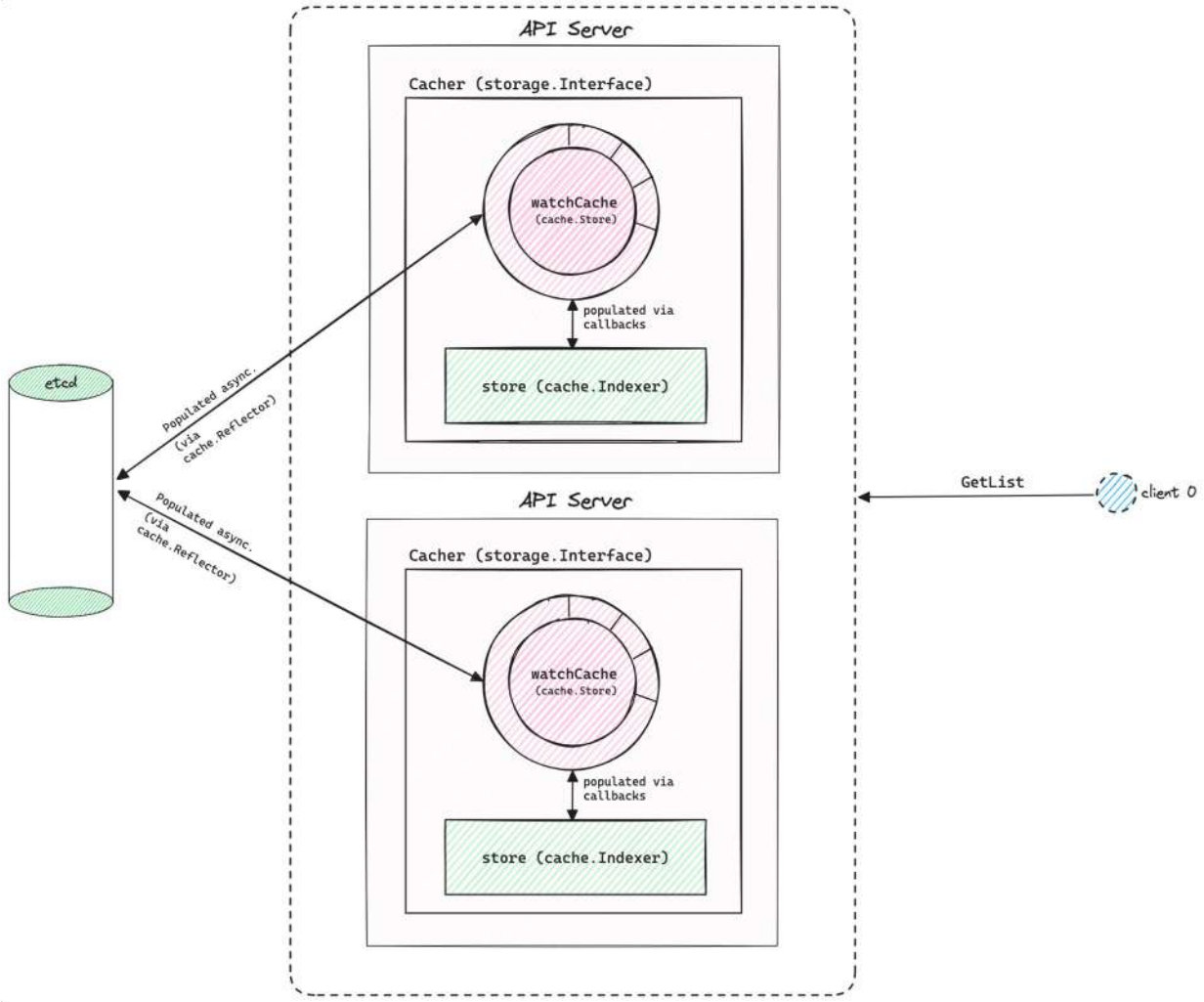


Request Behaviour

`GetList()`

Another gotcha - time travelling, stale reads from **`watchCache`**!

- If you have an HA setup, with **`watchCache`** enabled, one of them can be far behind the other.
- Since informers/reflectors default to **`resourceVersion="0"`** for their first **`LIST`** due scalability reasons, and these **`LISTs`** are served from the **`watchCache`**, we can get “data from the past”.



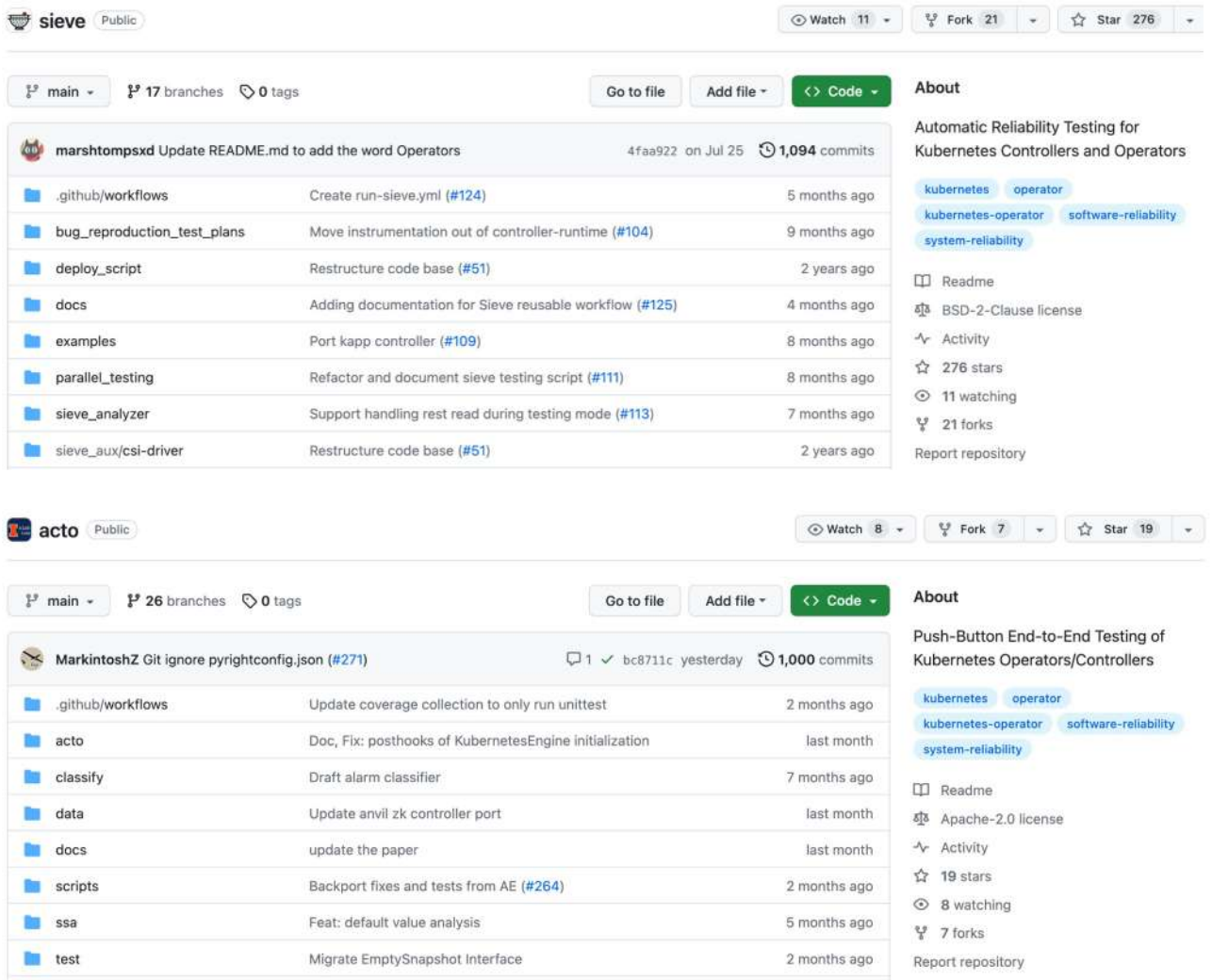
Request Behaviour

`GetList()`

Another gotcha - time travelling, stale reads from `watchCache`!

Externally to Kubernetes - there are a few tools that have come from collaboration between industry and academia that can help automatically detect such issues (and more) if your controllers are susceptible to them:

- `sieve`: https://github.com/sieve-project/sieve
- `acto`: https://github.com/xlab-uiuc/acto



Request Behaviour

`GetList()`

Another gotcha - time travelling, stale reads from **`watchCache`**!

Within Kubernetes –

- There are a couple of KEPs that are attempting to solve this in a scoped manner:
  - KEP-3157: Watch List

KEP-3157: allow informers for getting a stream of data instead of chunking.

- Release Signoff Checklist
- Summary
- Motivation
  - Goals
  - Non-Goals
- Proposal
  - Risks and Mitigations
- Design Details
  - Required changes for a WATCH request with the SendInitialEvents=true
    - API changes
    - Important optimisations
    - Manual testing without the changes in place
    - Results with WATCH-LIST
  - Required changes for a WATCH request with the RV set to the last observed value (RV > 0)
  - Provide a fix for the long-standing issue kubernetes/kubernetes#59848
  - Test Plan
    - Prerequisite testing updates

https://github.com/kubernetes/enhancements/tree/master/keps/sig-api-machinery/3157-watch-list

Request Behaviour

`GetList()`

Another gotcha - time travelling, stale reads from **`watchCache`**!

Within Kubernetes –

- There are a couple of KEPs that are attempting to solve this in a scoped manner:
  - KEP-3157: Watch List
  - KEP-2340: Consistent Reads From Cache

Consistent Reads from Cache

Kubernetes Get and List requests are guaranteed to be "consistent reads" if the `resourceVersion` parameter is not provided. Consistent reads are served from etcd using a "quorum read".

But often the watch cache contains sufficiently up-to-date data to serve the read request, and could serve it far more efficiently.

This KEP proposes a mechanism to serve most reads from the watch cache while still providing the same consistency guarantees as serving the read from etcd.

Table of Contents

https://github.com/kubernetes/enhancements/tree/master/keps/sig-api-machinery/2340-Consistent-reads-from-cache

Request Behaviour

`GetList()`

Another gotcha - time travelling, stale reads from **`watchCache`**!

Within Kubernetes –

- There are a couple of KEPs that are attempting to solve this in a scoped manner:
  - KEP-3157: Watch List
  - KEP-2340: Consistent Reads From Cache

This is in Alpha since Kubernetes v1.28, please try it out and provide feedback!

(Feature Gate: **`ConsistentListFromCache`**)

Consistent Reads from Cache

Kubernetes Get and List requests are guaranteed to be "consistent reads" if the `resourceVersion` parameter is not provided. Consistent reads are served from etcd using a "quorum read".

But often the watch cache contains sufficiently up-to-date data to serve the read request, and could serve it far more efficiently.

This KEP proposes a mechanism to serve most reads from the watch cache while still providing the same consistency guarantees as serving the read from etcd.

Table of Contents

https://github.com/kubernetes/enhancements/tree/master/keps/sig-api-machinery/2340-Consistent-reads-from-cache

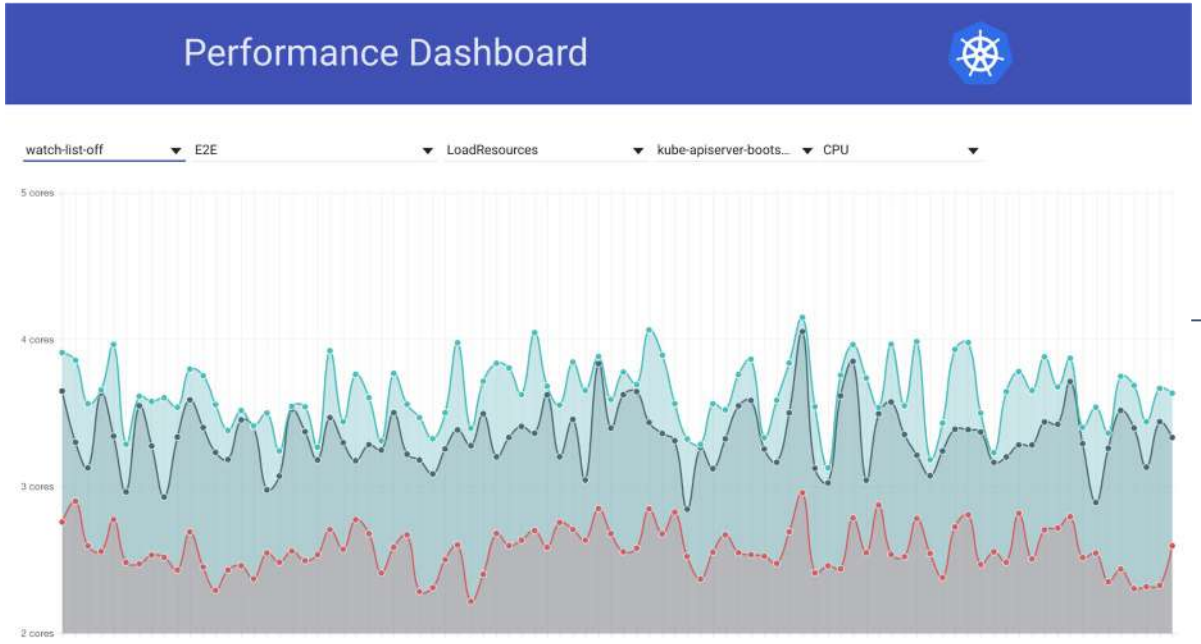
Request Behaviour

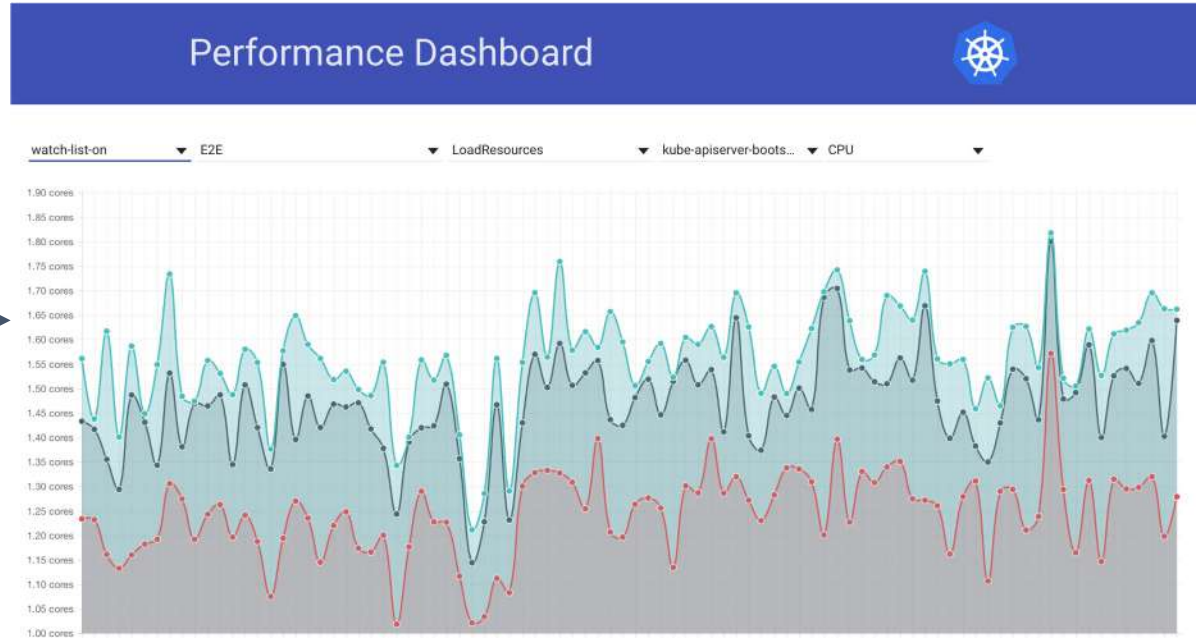
`GetList()`

You get some nice performance benefits from both these KEPs!

For KEP-3157: Watch List

(http://perf-dash.k8s.io)





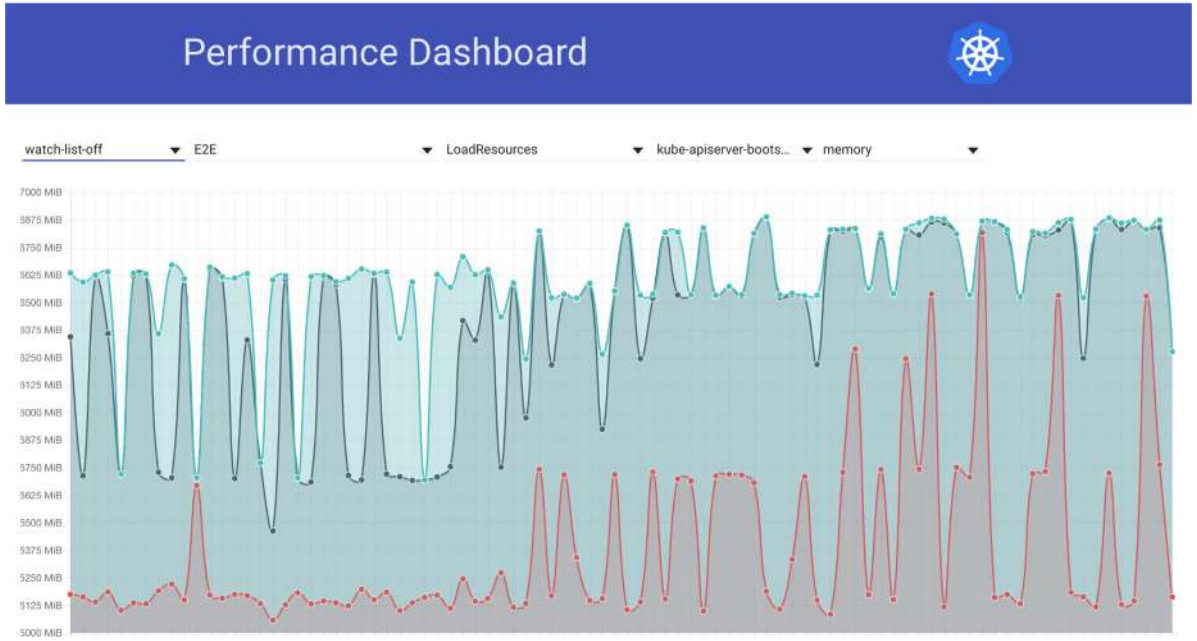
Request Behaviour

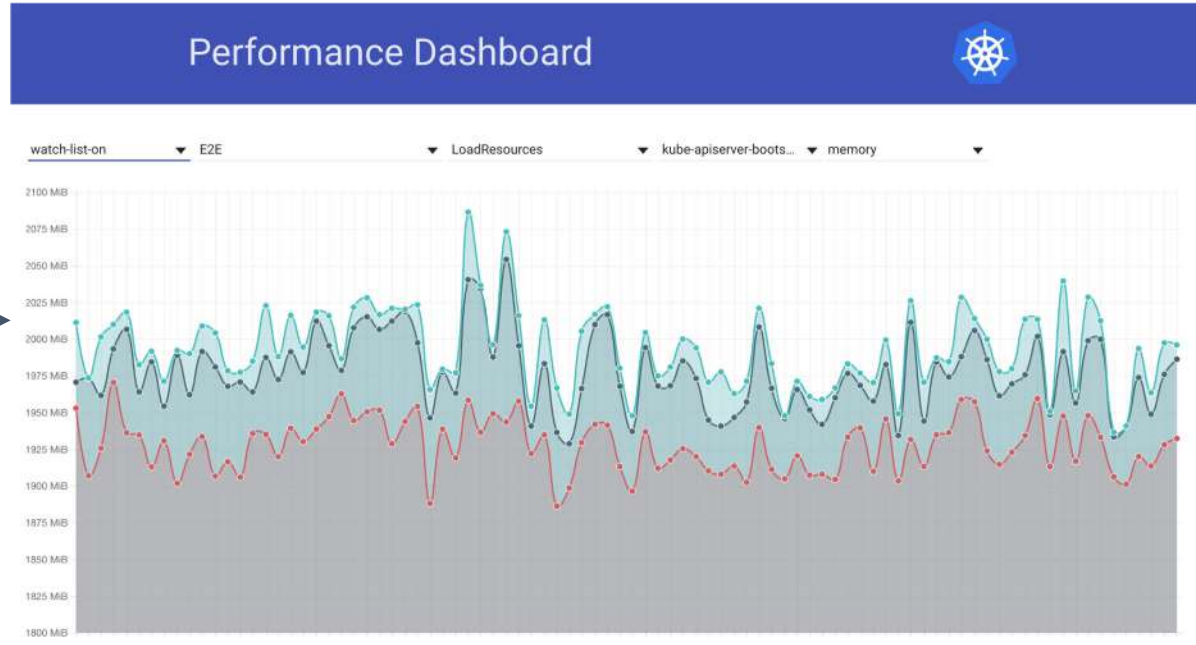
`GetList()`

You get some nice performance benefits from both these KEPs!

For KEP-3157: Watch List

(http://perf-dash.k8s.io)





Request Behaviour

`GetList()`

You get some nice performance benefits from both these KEPs!

For KEP-2340: Consistent Reads From Cache

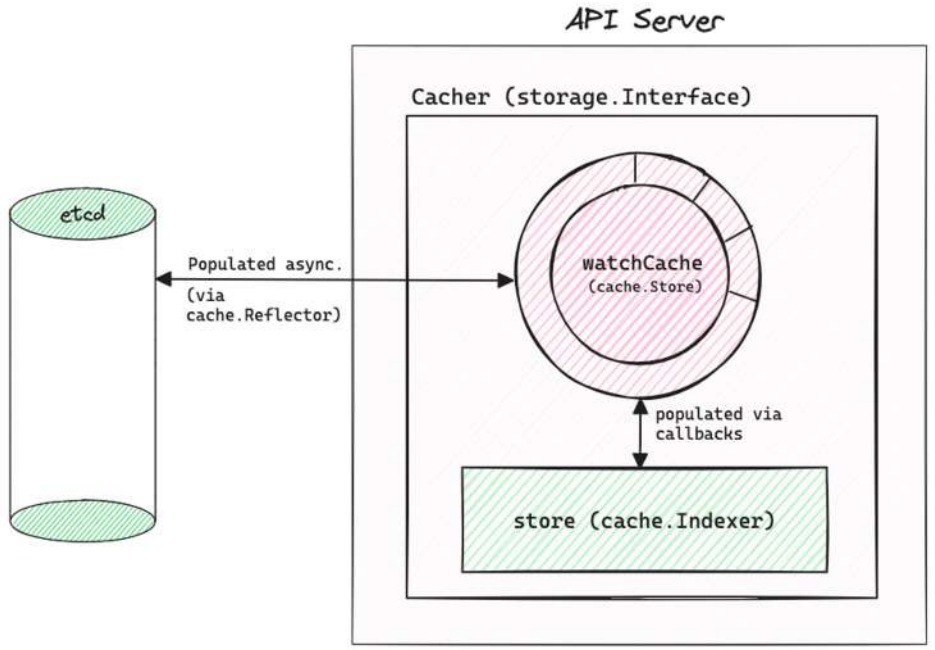
(https://github.com/kubernetes/test-infra/pull/30094)

Preliminary results

```
No QPS
kube-system          etcd-kind-control-plane                  324m         3364Mi
kube-system          kube-apiserver-kind-control-plane        538m         2080Mi

ConsistentListFromCache=false 100QPS
kube-system          etcd-kind-control-plane                  1584m        9861Mi
kube-system          kube-apiserver-kind-control-plane        2000m        9759Mi

ConsistentListFromCache=true 100QPS
kube-system          etcd-kind-control-plane                  434m         1293Mi
kube-system          kube-apiserver-kind-control-plane        662m         2139Mi
```

Request Behaviour

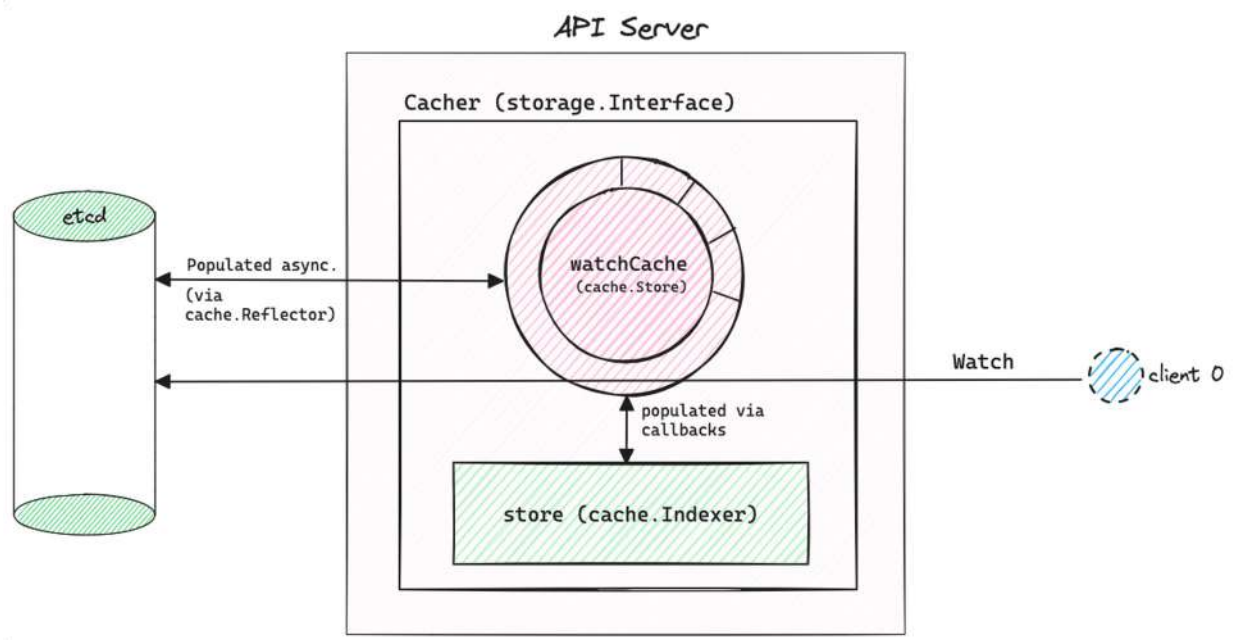
`Watch()`



Request Behaviour

`Watch()`

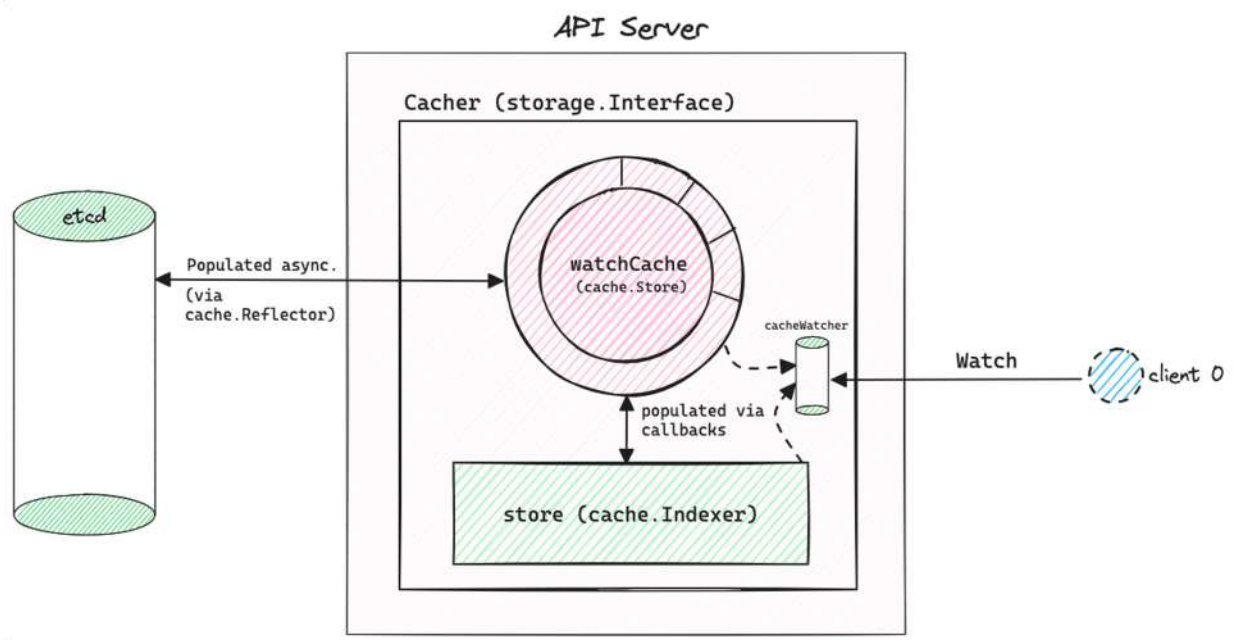
If `resourceVersion = ""`, we delegate the request to **etcd** as always.



Request Behaviour

`Watch()`

Otherwise, we serve it from the **`watchCache`**.

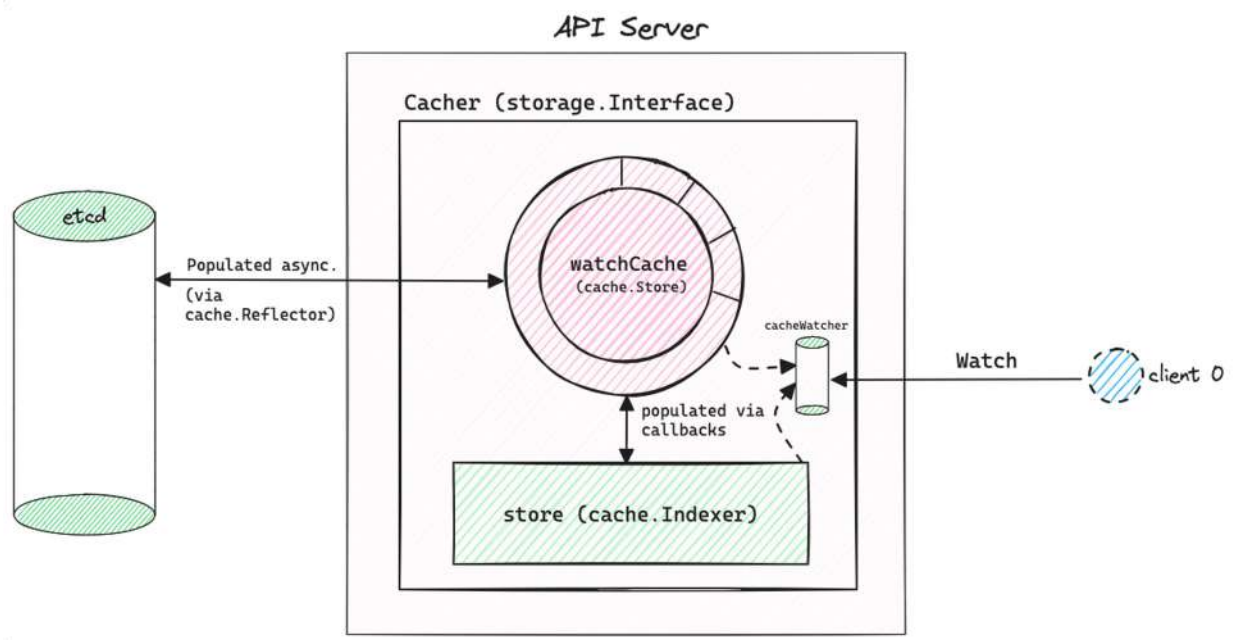


Request Behaviour

`Watch()`

Otherwise, we serve it from the **`watchCache`**.

- To do so - we first setup a **`cacheWatcher`** which is responsible for service a **`Watch`** request.

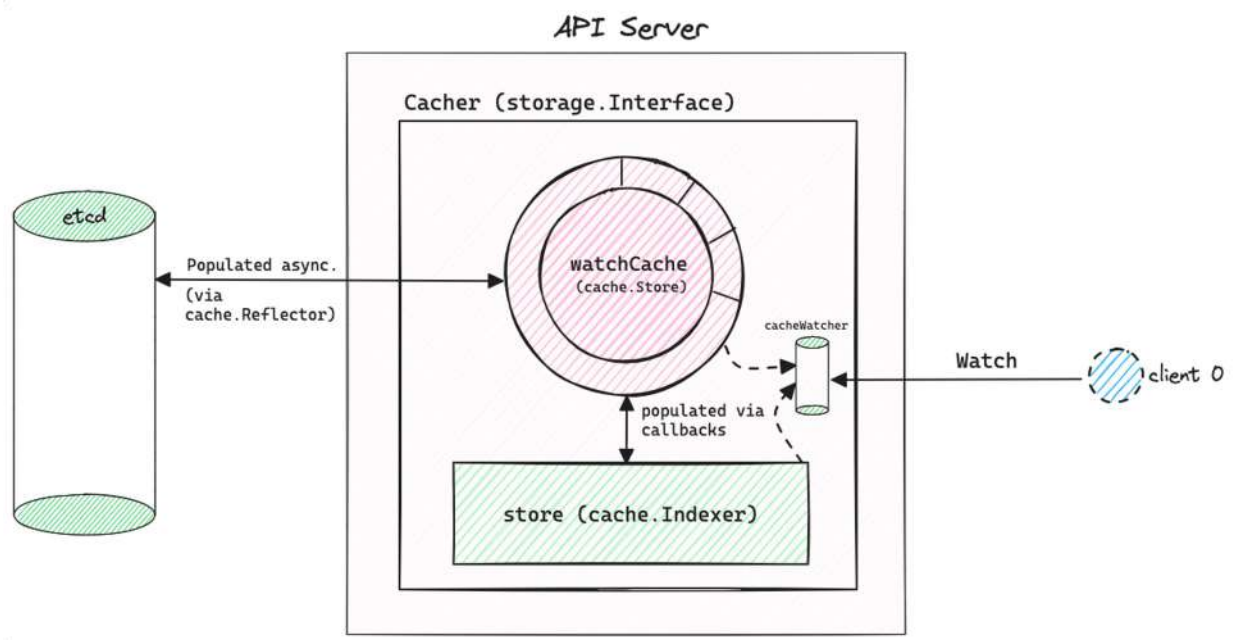


Request Behaviour

`Watch()`

Otherwise, we serve it from the **`watchCache`**.

- To do so - we first setup a **`cacheWatcher`** which is responsible for service a **`Watch`** request.
- Each **`cacheWatcher`** allocates an input buffer statically, size of which is determined by some heuristics we've seen in our scale testing.

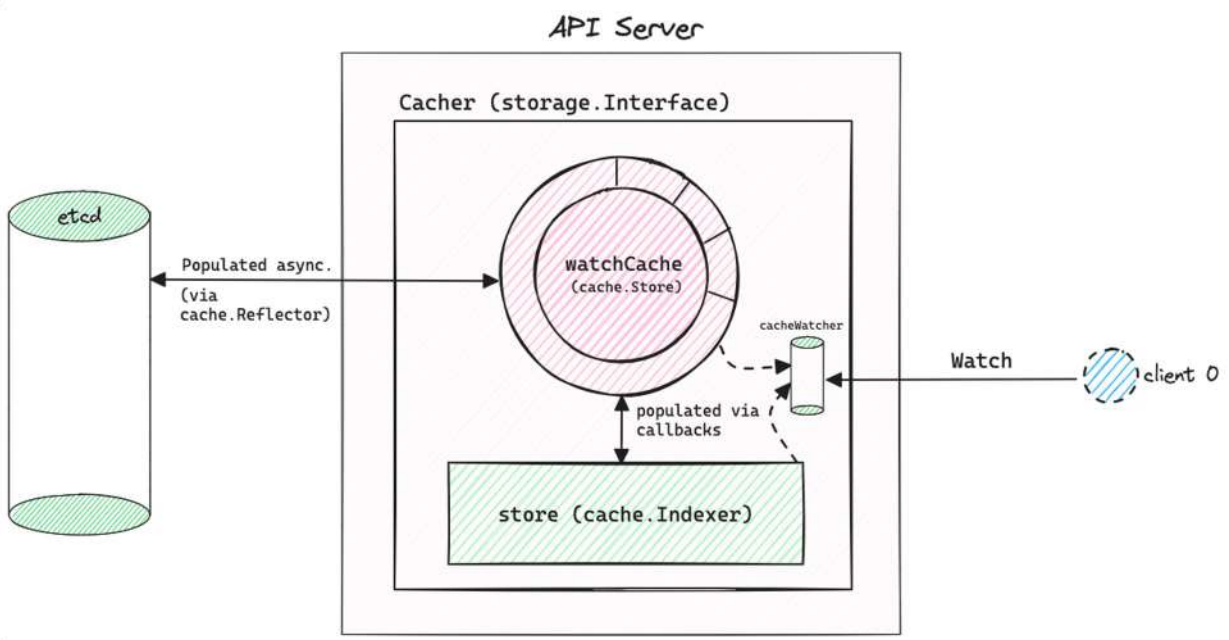


Request Behaviour

`Watch()`

Otherwise, we serve it from the **`watchCache`**.

- To do so - we first setup a **`cacheWatcher`** which is responsible for service a **`Watch`** request.
- Each **`cacheWatcher`** allocates an input buffer statically, size of which is determined by some heuristics we've seen in our scale testing.
- As soon as buffer becomes full, we terminate the **`Watch`** and clients re-establish one again against the last observed **`resourceVersion.`**

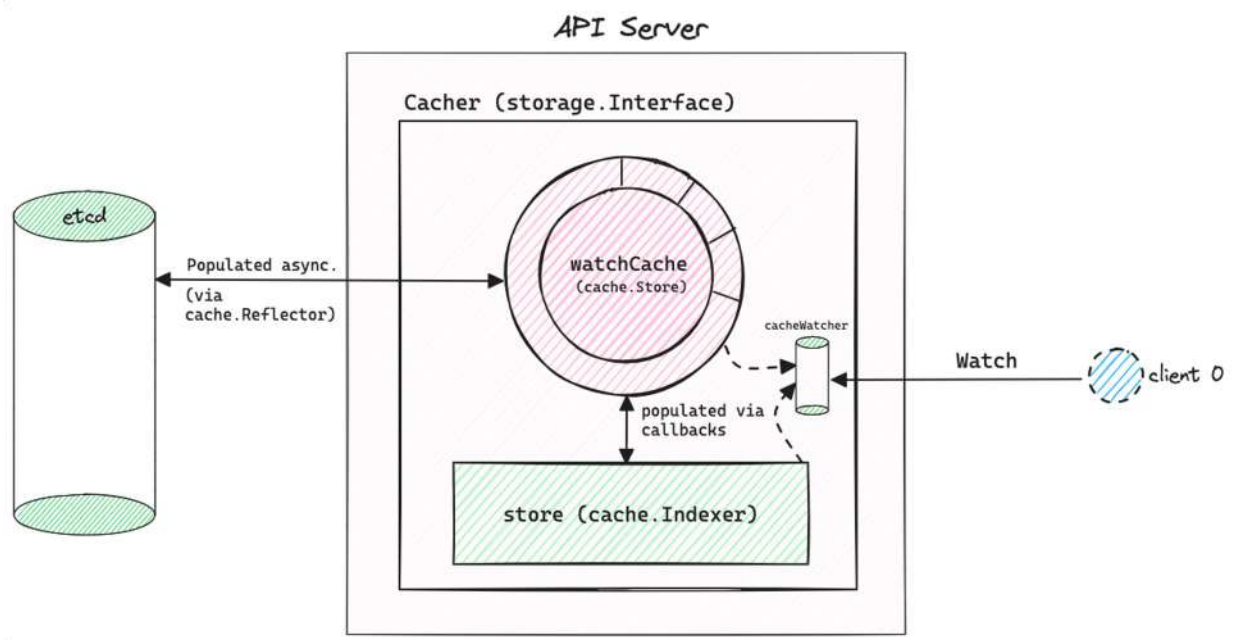


Request Behaviour

`Watch()`

Otherwise, we serve it from the **`watchCache`**.

- Essentially, the cost of keeping-up with **`Watch`** events, is establishing a **`Watch`** connection.

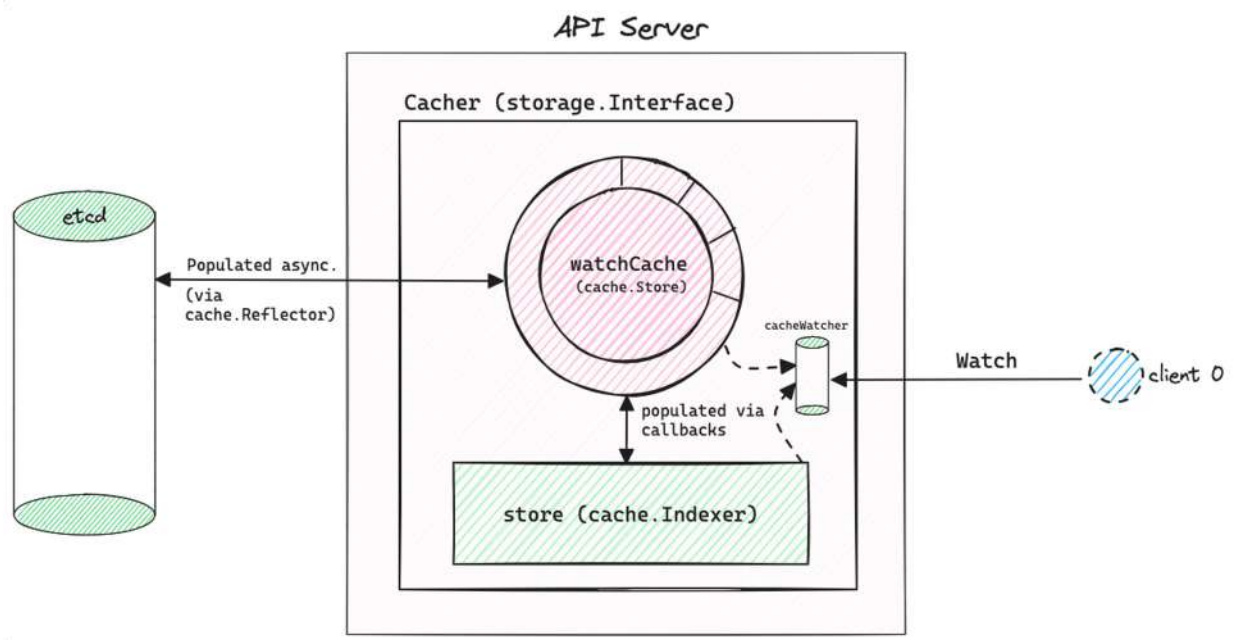


Request Behaviour

`Watch()`

Otherwise, we serve it from the **`watchCache`**.

- Essentially, the cost of keeping-up with **`Watch`** events, is establishing a **`Watch`** connection.
- However, a slow client, slow server, or just a storm of rapid updates can cause the buffer to become full, and necessitating a new connection.

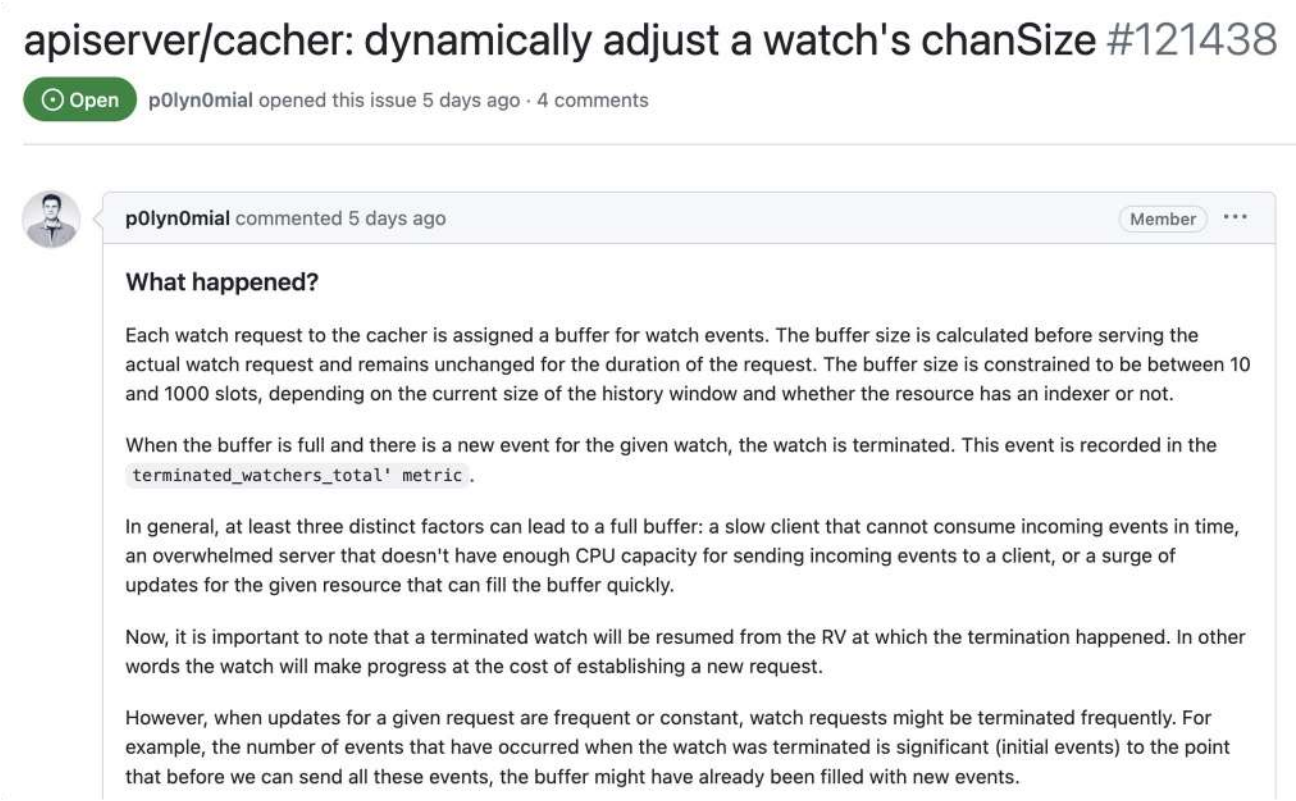


Request Behaviour

`Watch()`

Otherwise, we serve it from the **`watchCache`**.

- Essentially, the cost of keeping-up with **`Watch`** events, is establishing a **`Watch`** connection.
- However, a slow client, slow server, or just a storm of rapid updates can cause the buffer to become full, and necessitating a new connection.



apiserver/cacher: dynamically adjust a watch's chanSize #121438

Open p0lyn0mial opened this issue 5 days ago · 4 comments

p0lyn0mial commented 5 days ago Member

What happened?

Each watch request to the cacher is assigned a buffer for watch events. The buffer size is calculated before serving the actual watch request and remains unchanged for the duration of the request. The buffer size is constrained to be between 10 and 1000 slots, depending on the current size of the history window and whether the resource has an indexer or not.

When the buffer is full and there is a new event for the given watch, the watch is terminated. This event is recorded in the `terminated_watchers_total' metric` .

In general, at least three distinct factors can lead to a full buffer: a slow client that cannot consume incoming events in time, an overwhelmed server that doesn't have enough CPU capacity for sending incoming events to a client, or a surge of updates for the given resource that can fill the buffer quickly.

Now, it is important to note that a terminated watch will be resumed from the RV at which the termination happened. In other words the watch will make progress at the cost of establishing a new request.

However, when updates for a given request are frequent or constant, watch requests might be terminated frequently. For example, the number of events that have occurred when the watch was terminated is significant (initial events) to the point that before we can send all these events, the buffer might have already been filled with new events.

https://github.com/kubernetes/kubernetes/issues/121438

Conclusion

- The **`List + Watch`** pattern is a central theme to how the Kubernetes machine works, and helps enable the controller pattern.
- Different requests interact differently with each of the layers depending on the type of request and the value of the **`resourceVersion`** (and **`resourceVersionMatch`**) specified.
- Specification of **`resourceVersion`** and **`resourceVersionMatch`** can help you make the tradeoff between data consistency and latency, majorly impacting the scalability of your cluster.
- Unless you have strict consistency requitements, trust the **`watchCache`**, but beware of time travel queries!

References

- [Design Proposal] New storage layer design
- Cacher Source Code
- etcd3 storage layer source code
- `shouldDelegateList`
- [Kubernetes Enhancement Proposal] Consistent Reads From Cache
- [Kubernetes Enhancement Proposal] Watch List
- Sieve: Automatic Reliability Testing for Kubernetes Controllers and Operators
- Acto: Push-Button End-to-End Testing of Kubernetes Operators/Controllers

Thank you!

Twitter (X?): @MadhavJivrajani

Kubernetes/CNCF Slack: @madhav

Please scan the QR Code above
to leave feedback on this session