Rearchitecting Kubernetes for the Edge

EdgeSys '21

Andrew Jeffery
andrew.jeffery@cl.cam.ac.uk

Heidi Howard
heidi.howard@cl.cam.ac.uk

Richard Mortier
richard.mortier@cl.cam.ac.uk

University of Cambridge

Monday 26th April, 2021

One thing to take away

Orchestration should not require strong consistency!

Actual deployment case studies

Scale of communication is always increasing. Technology enables faster communication with more devices and more dynamic content:

- Next generation(s) of connectivity: 5G
- More devices: IoT
- Dynamic content: Elastic CDNs

What is Kubernetes?

Production-Grade Container Orchestration[1]

- Scaling
- Healing
- Routing
- Extensibility

59 percent of large organizations use Kubernetes in production[2]

[1] https://kubernetes.io/

[2] https://tanzu.vmware.com/content/blog/why-large-organizations-trust-kubernetes

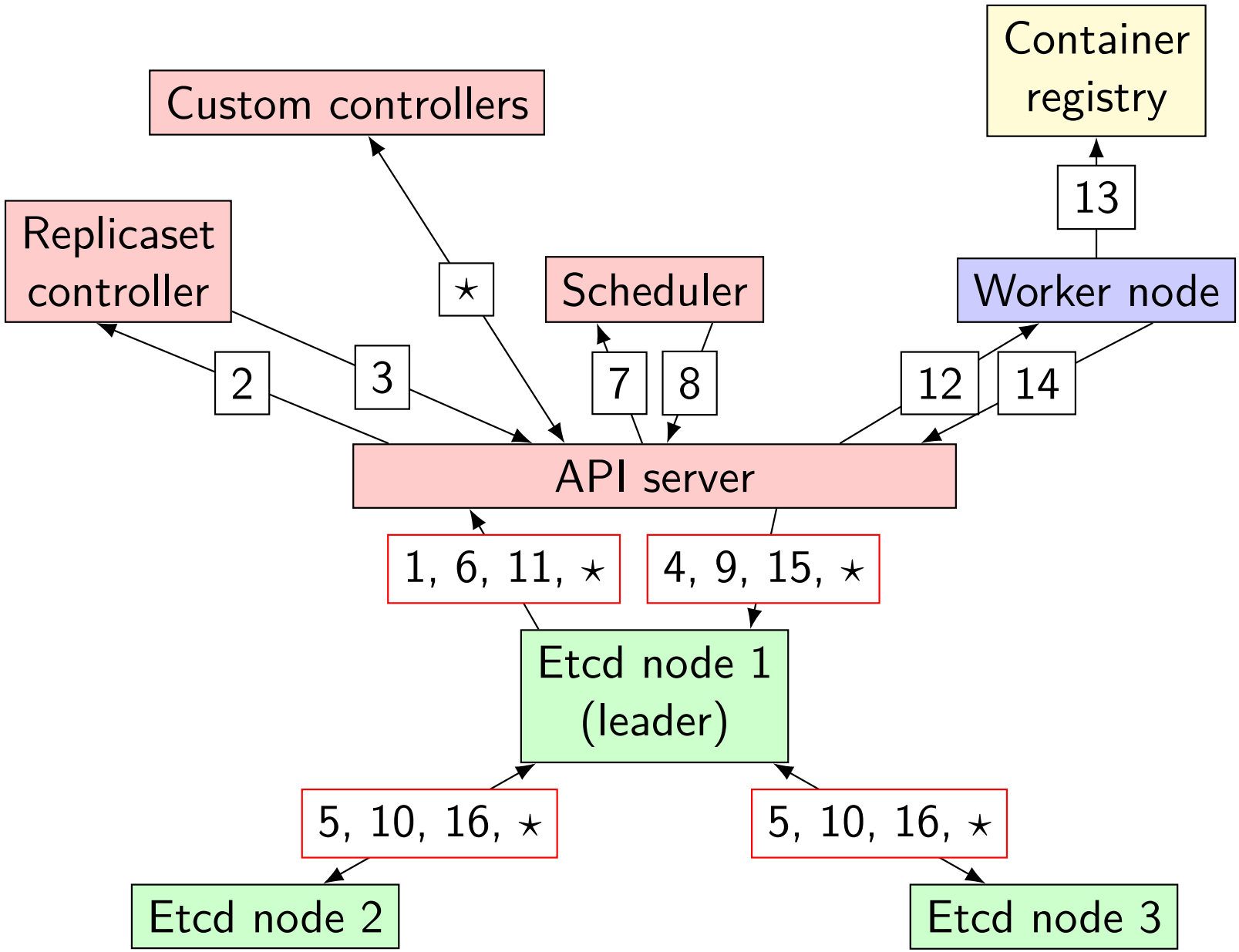
What is Etcd?

A distributed, reliable key-value store for the most critical data of a distributed system[3]

- Critical data here is Kubernetes state
- Supports transactions on data
- Also has concept of watches: notifications of changes

[3] https://etcd.io/

Scheduling with centralised state



Key limitation - Etcd

Centralised, strongly consistent state
This is all etcd

Key Takeaway

Orchestration should not require strong consistency

Key limitations revisited

Decentralised, **eventually** consistent state

Key Takeaway

Orchestration should not require strong consistency

What gets stored?

JSON-like Data!

We can use CRDTs for this

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: nginx
        image: nginx
```

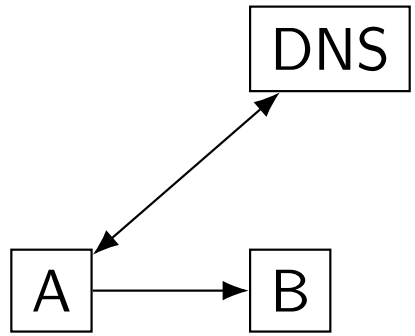
Dealing with stale data

Assuming we don't want our state to be stale...

But maybe we can use staleness to our advantage?

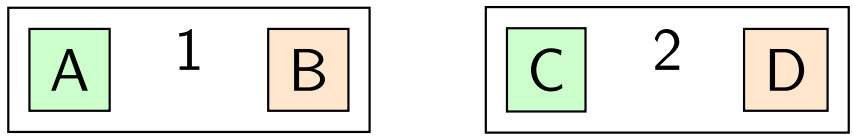
It is a dynamic environment and at best we can only model it so Kubernetes likely already acts on stale information

Stale state: Services



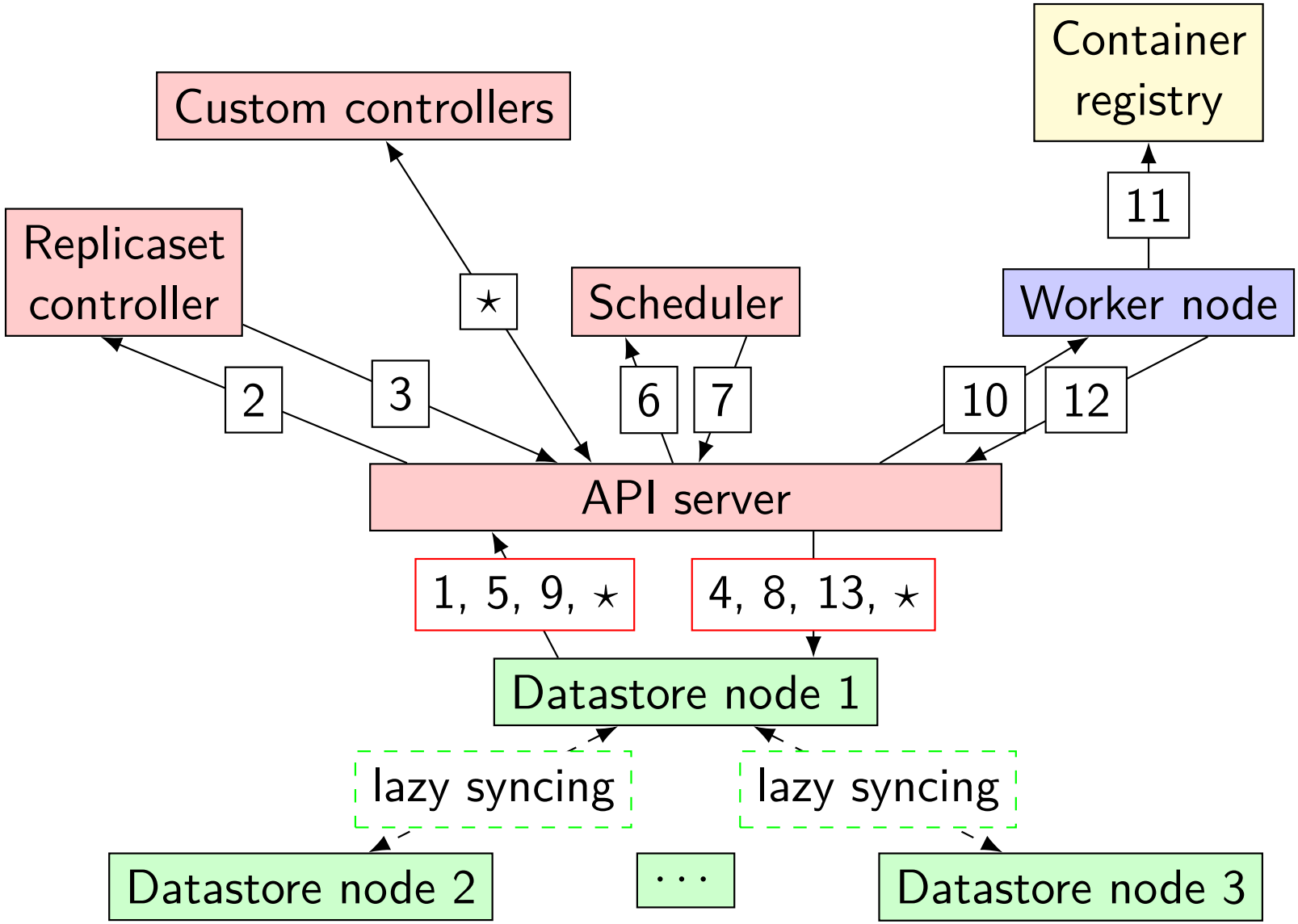
- A wants to send to B but must first resolve the DNS name
- DNS could resolve to IP address which matches a dead container
- So, this could already be an issue

Stale state: ReplicaSets



- Initially A, B and C are running, then B fails causing D to be created
- Under failures we can have more scheduled than we'd expect
- So, again, this could already be an issue in our distributed system
- With CRDTs we can control over-replication rather than under-replication on merge

Scheduling with the decentralised state



Saved at least 3 critical path network hops!

Rearchitect with eventual consistency?

Our new datastore gives us more flexibility

- We can now scale across multiple edge sites
- We can run on slower links and still be efficient
- We can use fewer resources and maintain the same fault tolerance

Conclusion

Orchestration should not require strong consistency

By adhering to this we can

- Improve performance and reactivity of our clusters
- Make them more tolerant to failure
- Scale them across more sites without fear

Implementation is in progress and we welcome comments, suggestions and collaborations.

Andrew Jeffery
andrew.jeffery@cl.cam.ac.uk