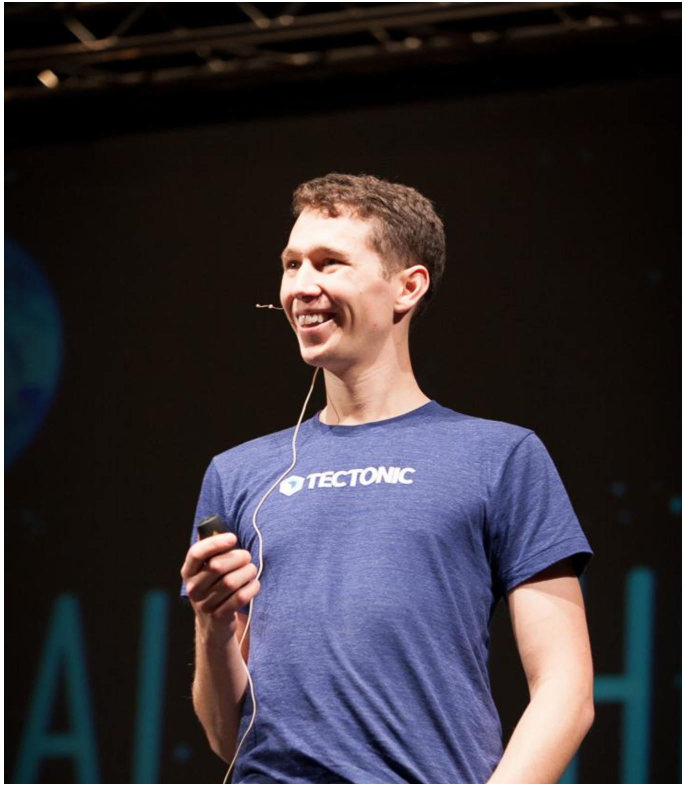
5 years of etcd

Brandon Philips
Xiang Li

Brandon Philips

CTO, CoreOS
Technical Staff, Red Hat

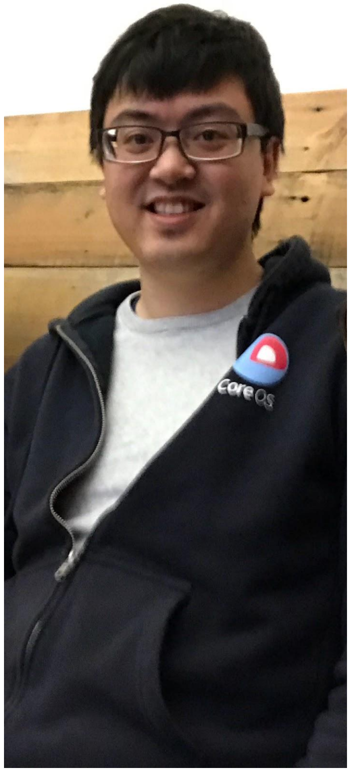
keybase.io/philips
github.com/philips
twitter.com/brandonphilips



Xiang Li

Engineering Mgr., CoreOS
(founding eng. of etcd)
Senior Staff Software Eng. Alibaba

github.com/xiang90
twitter.com/xiangli0227



Hello, etcd

database

Flat Key Value

```
$ ./etcd &


$ etcdctl put hello etcd
OK
$ etcdctl get hello
hello
etcd
```

replicated

Consistent and Partition Tolerant

In Search of an Understandable Consensus Algorithm (Extended Version)

Diego Ongaro and John Ousterhout
Stanford University

Abstract

Raft is a consensus algorithm for managing a replicated log. It produces a result equivalent to (multi-)Paxos, and it is as efficient as Paxos, but its structure is different from Paxos; this makes Raft more understandable than Paxos and also provides a better foundation for building practical systems. In order to enhance understandability, Raft separates the key elements of consensus, such as leader election, log replication, and safety, and it enforces a stronger degree of coherency to reduce the number of states that must be considered. Results from a user study demonstrate that Raft is easier for students to learn than Paxos. Raft also includes a new mechanism for changing the cluster membership, which uses overlapping majorities to guarantee safety.

1 Introduction

Consensus algorithms allow a collection of machines to work as a coherent group that can survive the failures of some of its members. Because of this, they play a key role in building reliable large-scale software systems. Paxos [15, 16] has dominated the discussion of consensus algorithms over the last decade: most implementations of consensus are based on Paxos or influenced by it, and Paxos has become the primary vehicle used to teach students about consensus.

Unfortunately, Paxos is quite difficult to understand, in spite of numerous attempts to make it more approachable. Furthermore, its architecture requires complex changes to support practical systems. As a result, both system builders and students struggle with Paxos.

After struggling with Paxos ourselves, we set out to find a new consensus algorithm that could provide a better foundation for system building and education. Our approach was unusual in that our primary goal was *understandability*: could we define a consensus algorithm for practical systems and describe it in a way that is significantly easier to learn than Paxos? Furthermore, we wanted the algorithm to facilitate the development of intuitions that are essential for system builders. It was important not just for the algorithm to work, but for it to be obvious why

state space reduction (relative to Paxos, Raft reduces the degree of nondeterminism and the ways servers can be inconsistent with each other). A user study with 43 students at two universities shows that Raft is significantly easier to understand than Paxos: after learning both algorithms, 33 of these students were able to answer questions about Raft better than questions about Paxos.

Raft is similar in many ways to existing consensus algorithms (most notably, Oki and Liskov's Viewstamped Replication [29, 22]), but it has several novel features:

- **Strong leader:** Raft uses a stronger form of leadership than other consensus algorithms. For example, log entries only flow from the leader to other servers. This simplifies the management of the replicated log and makes Raft easier to understand.
- **Leader election:** Raft uses randomized timers to elect leaders. This adds only a small amount of mechanism to the heartbeats already required for any consensus algorithm, while resolving conflicts simply and rapidly.
- **Membership changes:** Raft's mechanism for changing the set of servers in the cluster uses a new *joint consensus* approach where the majorities of two different configurations overlap during transitions. This allows the cluster to continue operating normally during configuration changes.

We believe that Raft is superior to Paxos and other consensus algorithms, both for educational purposes and as a foundation for implementation. It is simpler and more understandable than other algorithms; it is described completely enough to meet the needs of a practical system; it has several open-source implementations and is used by several companies; its safety properties have been formally specified and proven; and its efficiency is comparable to other algorithms.

The remainder of the paper introduces the replicated state machine problem (Section 2), discusses the strengths and weaknesses of Paxos (Section 3), describes our general approach to understandability (Section 4), presents the Raft consensus algorithm (Sections 5–8), evaluates

etcd cluster properties

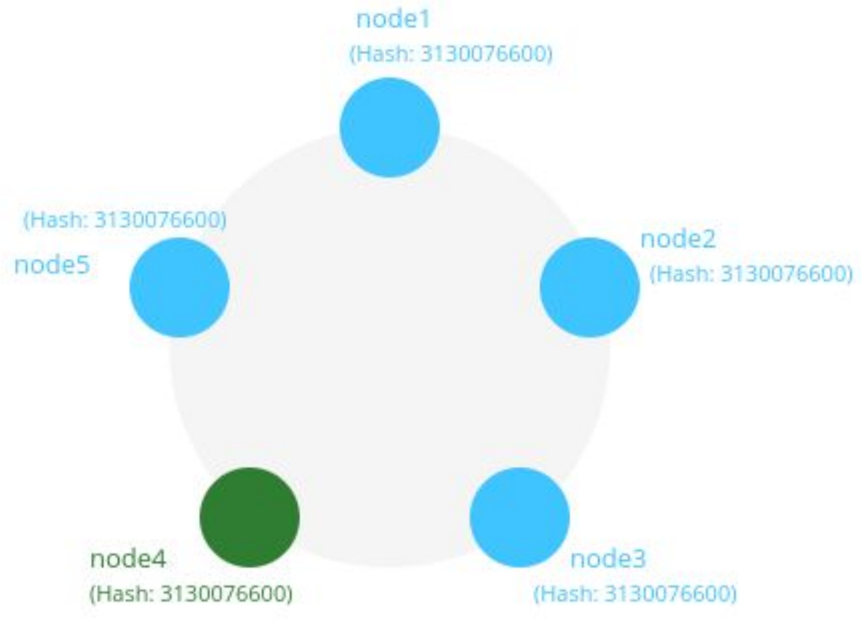
1, 3, or 5 members

Homogeneous CPU/RAM/disk

Automatic leader election

Resilient to long partitions

Replace nodes at runtime



database

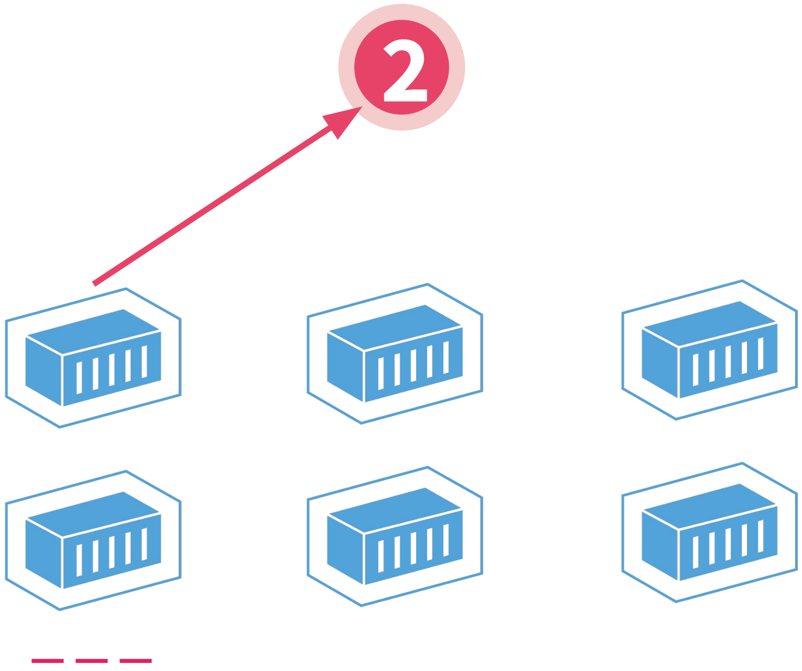
Used by Kube

```
$  etcdctl get '' --prefix

/apiregistration.k8s.io/apiservices/v1.
...
/ranges/serviceips
...
/masterleases/100.115.92.206
...
/namespaces/default
/namespaces/kube-public
/namespaces/kube-system
```

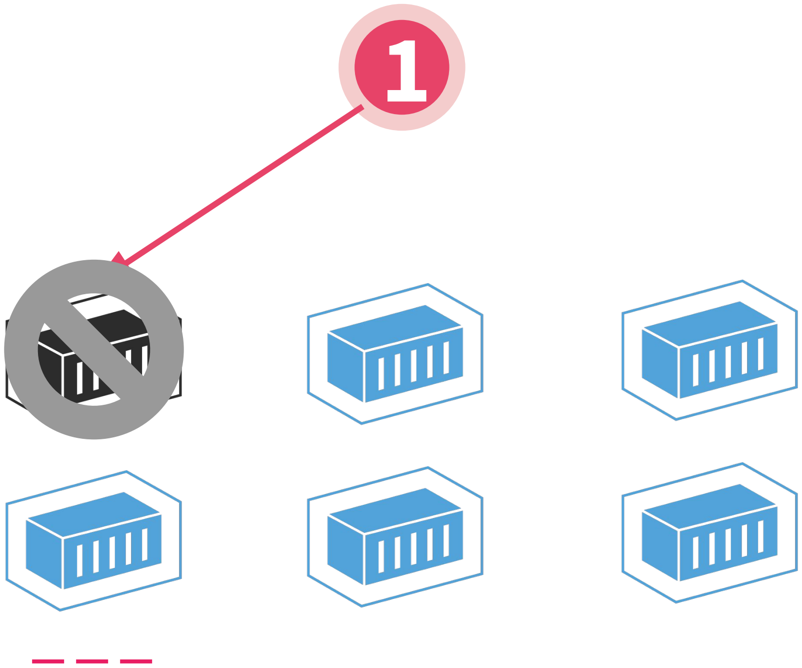
CoreOS's Problem

avoid app downtime w/
automatic OS updates



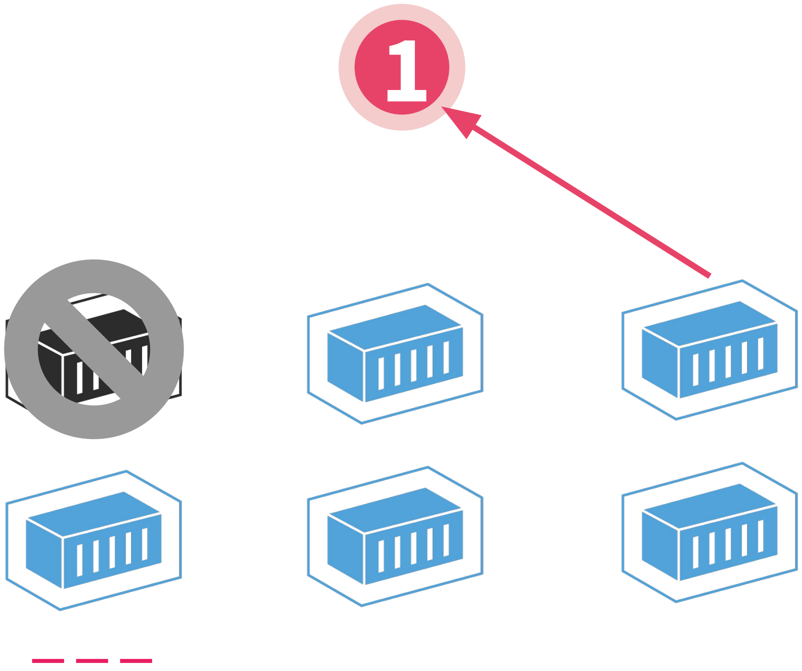
CoreOS's Problem

avoid app downtime w/
automatic OS updates



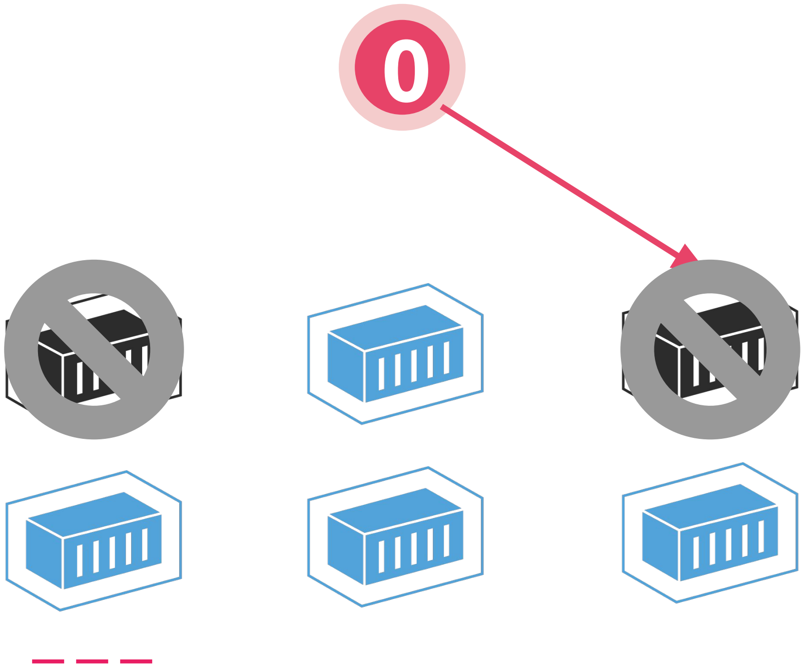
CoreOS's Problem

avoid app downtime w/
automatic OS updates



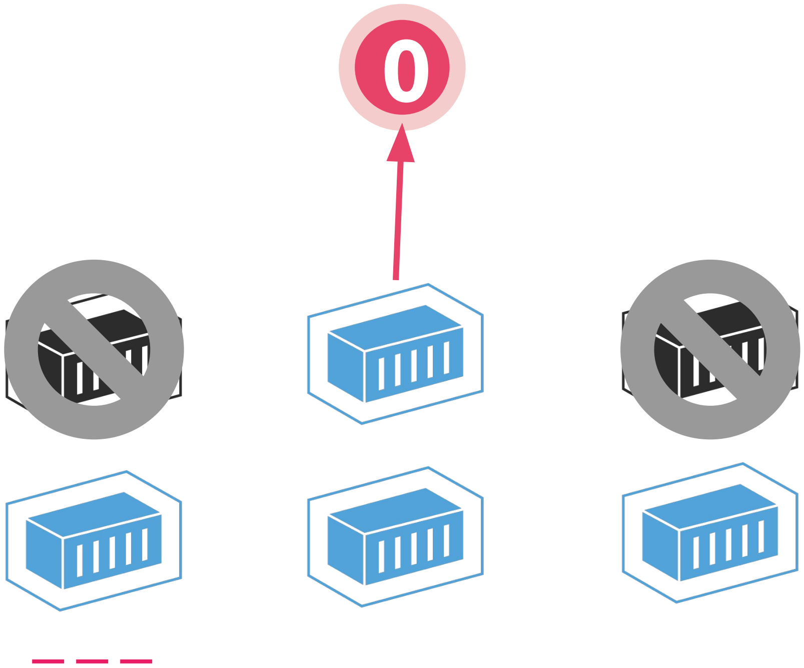
CoreOS's Problem

avoid app downtime w/
automatic OS updates



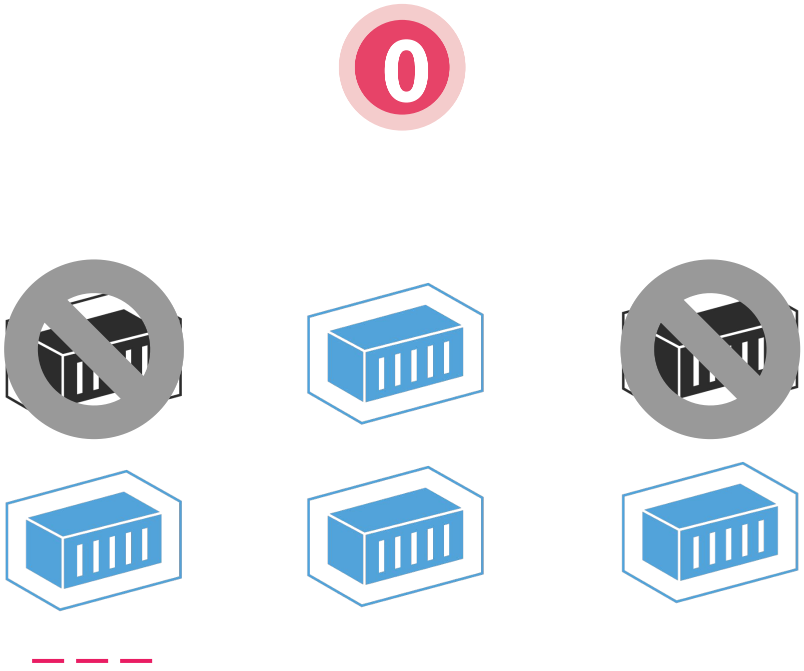
CoreOS's Problem

avoid app downtime w/
automatic OS updates



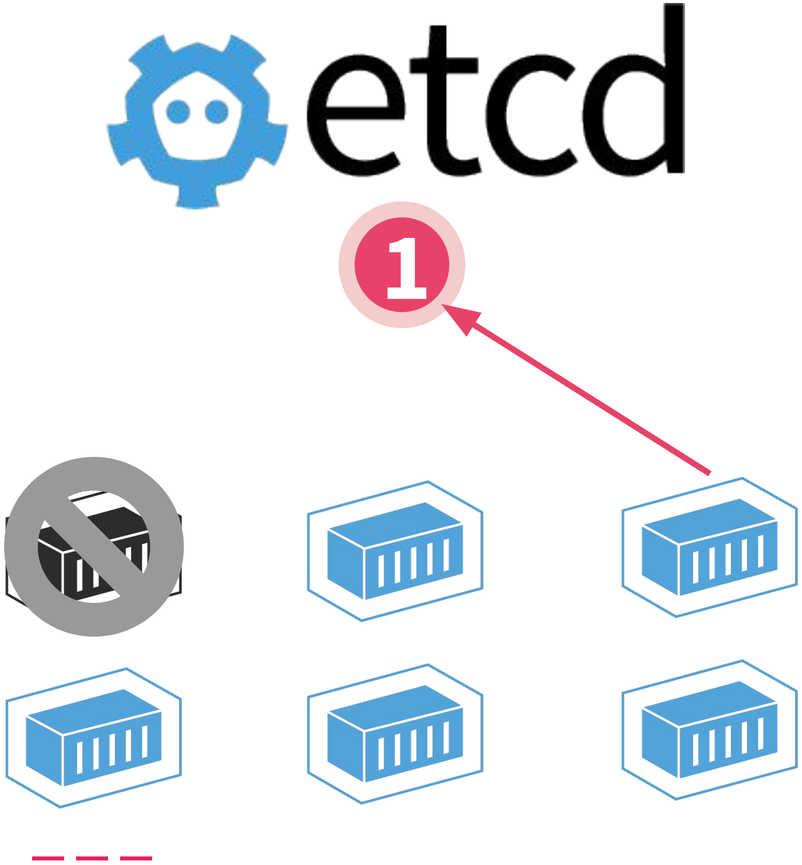
CoreOS's Problem

avoid app downtime w/
automatic OS updates



CoreOS's Problem

avoid app downtime w/
automatic OS updates



README Driven Development

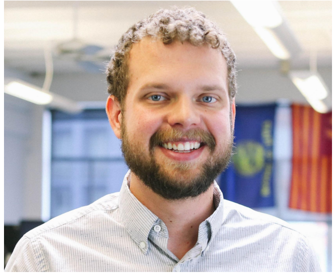
Simple: curl'able user facing API (HTTP+JSON)

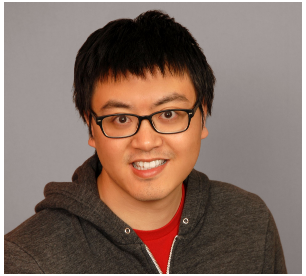
Secure: optional SSL client cert authentication

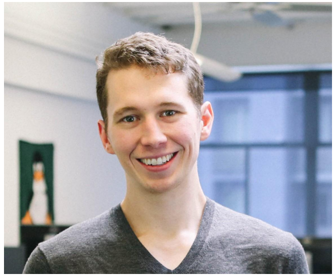
Fast: benchmarked 1000s of writes/s per instance

Reliable: properly distributed using Raft

Persistent: cluster failure is recoverable from disk







database

Flat Key Value

```
$ ./etcd &

$ etcdctl put hello etcd
OK
$ etcdctl get hello
hello
etcd
```

replicated

Consistent and Partition Tolerant

In Search of an Understandable Consensus Algorithm (Extended Version)

Diego Ongaro and John Ousterhout
Stanford University

Abstract

Raft is a consensus algorithm for managing a replicated log. It produces a result equivalent to (multi-)Paxos, and it is as efficient as Paxos, but its structure is different from Paxos; this makes Raft more understandable than Paxos and also provides a better foundation for building practical systems. In order to enhance understandability, Raft separates the key elements of consensus, such as leader election, log replication, and safety, and it enforces a stronger degree of coherency to reduce the number of states that must be considered. Results from a user study demonstrate that Raft is easier for students to learn than Paxos. Raft also includes a new mechanism for changing the cluster membership, which uses overlapping majorities to guarantee safety.

1 Introduction

Consensus algorithms allow a collection of machines to work as a coherent group that can survive the failures of some of its members. Because of this, they play a key role in building reliable large-scale software systems. Paxos [15, 16] has dominated the discussion of consensus algorithms over the last decade: most implementations of consensus are based on Paxos or influenced by it, and Paxos has become the primary vehicle used to teach students about consensus.

Unfortunately, Paxos is quite difficult to understand, in spite of numerous attempts to make it more approachable. Furthermore, its architecture requires complex changes to support practical systems. As a result, both system builders and students struggle with Paxos.

After struggling with Paxos ourselves, we set out to find a new consensus algorithm that could provide a better foundation for system building and education. Our approach was unusual in that our primary goal was *understandability*: could we define a consensus algorithm for practical systems and describe it in a way that is significantly easier to learn than Paxos? Furthermore, we wanted the algorithm to facilitate the development of intuitions that are essential for system builders. It was important not just for the algorithm to work, but for it to be obvious why

state space reduction (relative to Paxos, Raft reduces the degree of nondeterminism and the ways servers can be inconsistent with each other). A user study with 43 students at two universities shows that Raft is significantly easier to understand than Paxos: after learning both algorithms, 33 of these students were able to answer questions about Raft better than questions about Paxos.

Raft is similar in many ways to existing consensus algorithms (most notably, Oki and Liskov's Viewstamped Replication [29, 22]), but it has several novel features:

- **Strong leader:** Raft uses a stronger form of leadership than other consensus algorithms. For example, log entries only flow from the leader to other servers. This simplifies the management of the replicated log and makes Raft easier to understand.
- **Leader election:** Raft uses randomized timers to elect leaders. This adds only a small amount of mechanism to the heartbeats already required for any consensus algorithm, while resolving conflicts simply and rapidly.
- **Membership changes:** Raft's mechanism for changing the set of servers in the cluster uses a new *joint consensus* approach where the majorities of two different configurations overlap during transitions. This allows the cluster to continue operating normally during configuration changes.

We believe that Raft is superior to Paxos and other consensus algorithms, both for educational purposes and as a foundation for implementation. It is simpler and more understandable than other algorithms; it is described completely enough to meet the needs of a practical system; it has several open-source implementations and is used by several companies; its safety properties have been formally specified and proven; and its efficiency is comparable to other algorithms.

The remainder of the paper introduces the replicated state machine problem (Section 2), discusses the strengths and weaknesses of Paxos (Section 3), describes our general approach to understandability (Section 4), presents the Raft consensus algorithm (Sections 5–8), evaluates

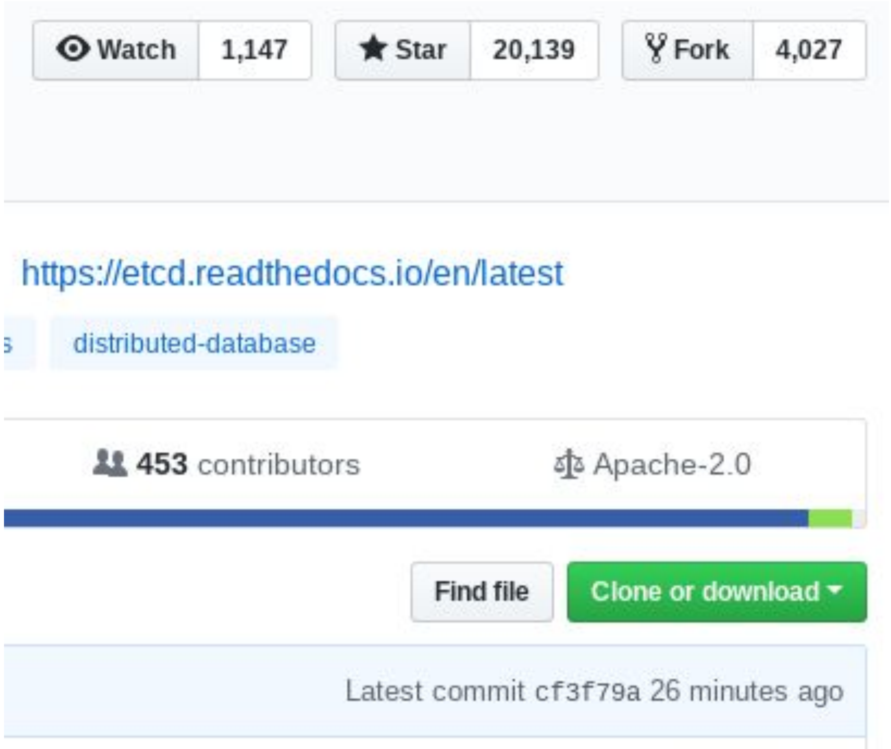
open source

Written in Go
Apache 2.0

```
$ go get go.etcd.io/etcd
```

☆ on GitHub

#11 Github Go Project



etcd Top-level Maintainers

Anthony Romano

Brandon Philips

Fanmin Shi

Gyuho Lee

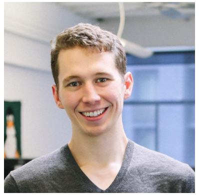
Hitoshi Mitake

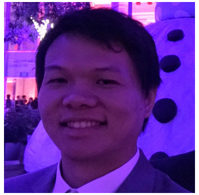
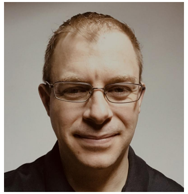
Joe Betz

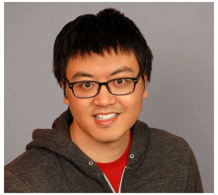
Sam Batschelet

Xiang Li







Red Hat Contributes etcd to the CNCF

etcd trademark & logos

etcd.io domain

discovery service

dev/test infra

Hand off to Xiang

5 years

400+ contributors with 14,000+ commits

150+ releases with 3 major ones:

- etcd alpha
- etcd 2
- etcd 3

Major releases

etcd alpha

Cloud native distributed consensus system

CLOUDFOUNDRY

flannel

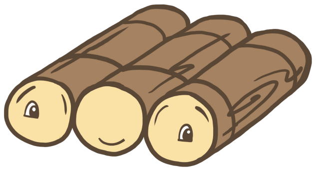
Major releases

etcd 2

Simple API with solid core

A solid core

etcd/raft



TiKV

Dgraph

CockroachDB

HYPERLEDGER

Major releases

etcd 3

Efficiency, Reliability, Useability

Efficiency, Reliability, Usability

- New storage backend
- New APIs
  - MVCC, Transaction
- Remote snapshot
- Learner, Pre-vote, Proxy

Promote cloud native technologies

GRPC

2014 2015 2016

Kubernetes + etcd

- List and Watch pattern influences the design of Kubernetes

- Kubernetes pushes etcd forward
  - MVCC
  - Transaction
  - Scalability
  - Observability

A widely adopted technology

A widely adopted technology

amazon web services

Azure

Alibaba Cloud

IBM Cloud

DigitalOcean

Nearly all major cloud providers run etcd

Hand off to Brandon

CNCF Support

Supporting existing services

Test/dev cloud services

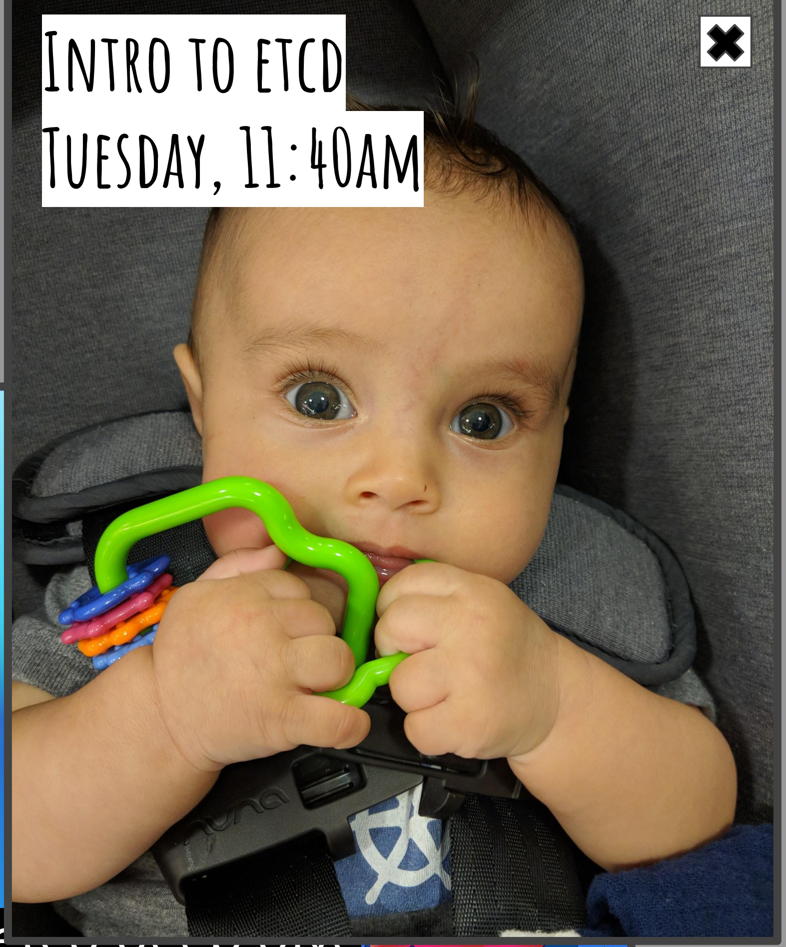
discovery.etcd.io operation

CNCF Support

New community service investments

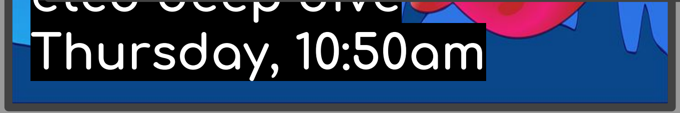
3rd party security audit

3rd party correctness audit


Intro to etcd
Tuesday, 11:40am


Debugging etcd
Tuesday, 2:35pm


What's next with etcd
Tuesday 1:45pm


Thursday, 10:50am

Thank You!

@brandonphilips
@xiangli0227

Intro December 11,
11:40am–12:15pm

Deep Dive December 13,
10:50am–11:25am

What’s next for etcd
Tuesday 1:45pm - 2:20pm

Debugging etcd Tuesday
2:35pm - 3:10pm