Smooth Operator

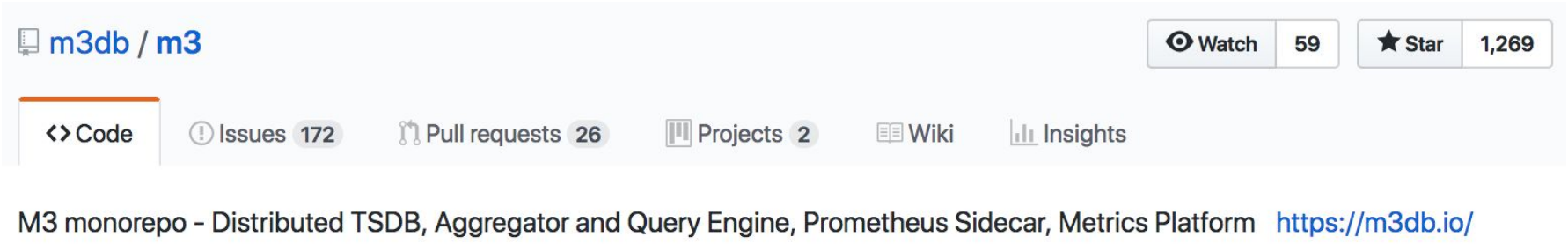
Large-Scale Automated Storage
with Kubernetes

Uber |

Celina Ward @shaleenaa

Matt Schallert @mattschallert

What is M3?


m3db / m3
Watch 59
Star 1,269
Code
Issues 172
Pull requests 26
Projects 2
Wiki
Insights
M3 monorepo - Distributed TSDB, Aggregator and Query Engine, Prometheus Sidecar, Metrics Platform https://m3db.io/

M3DB Scale

31M

Writes per second

50Gb

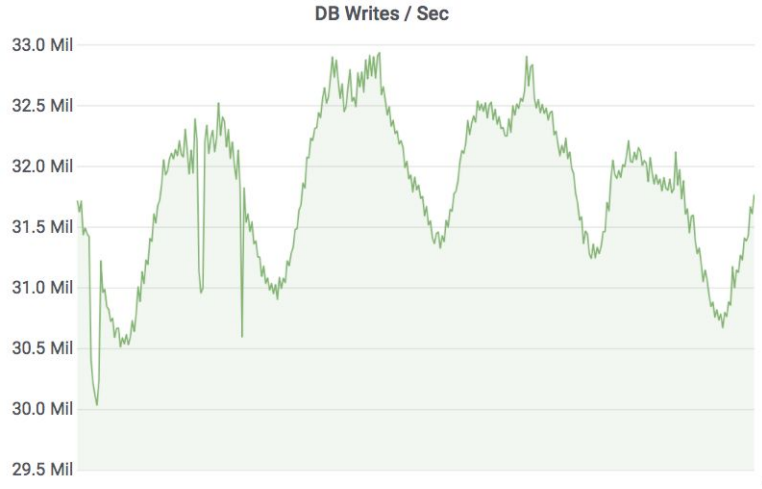
Gigabits per second

1000+

Instances running M3DB

9B

Unique Metric IDs



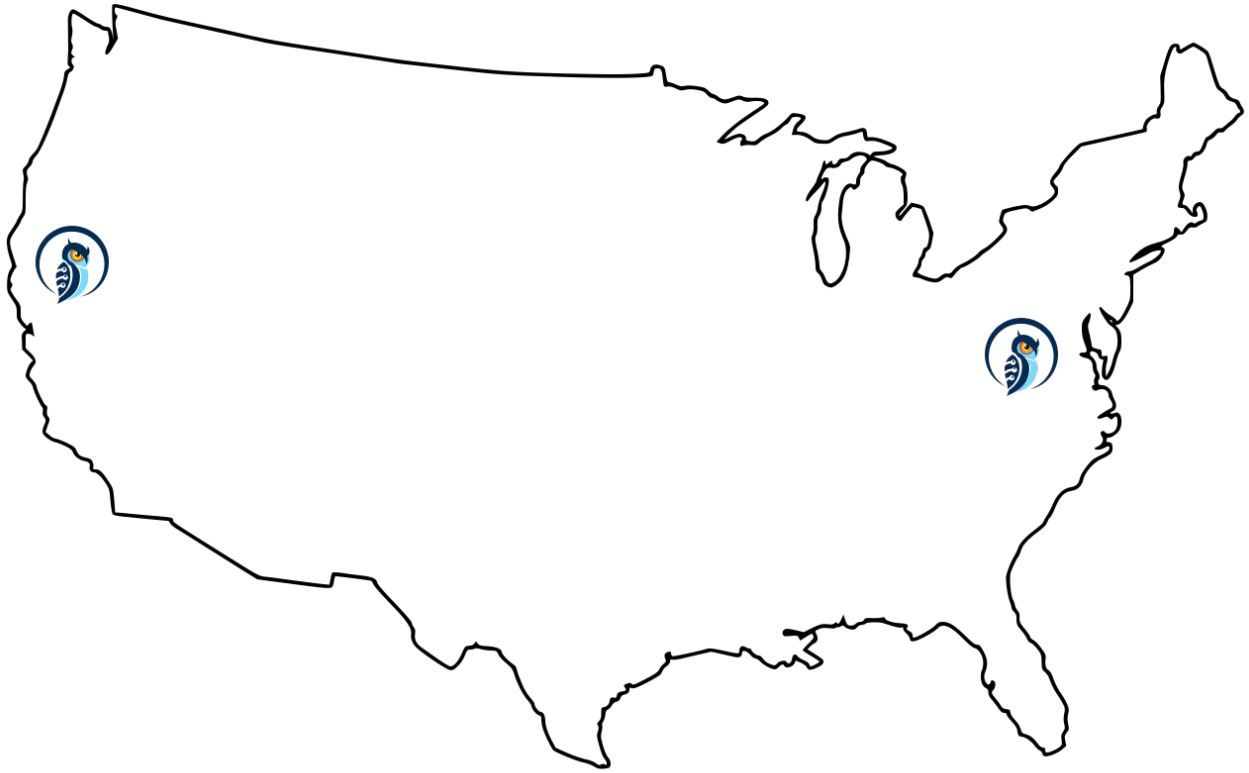
2016

2

Clusters

1

Configuration



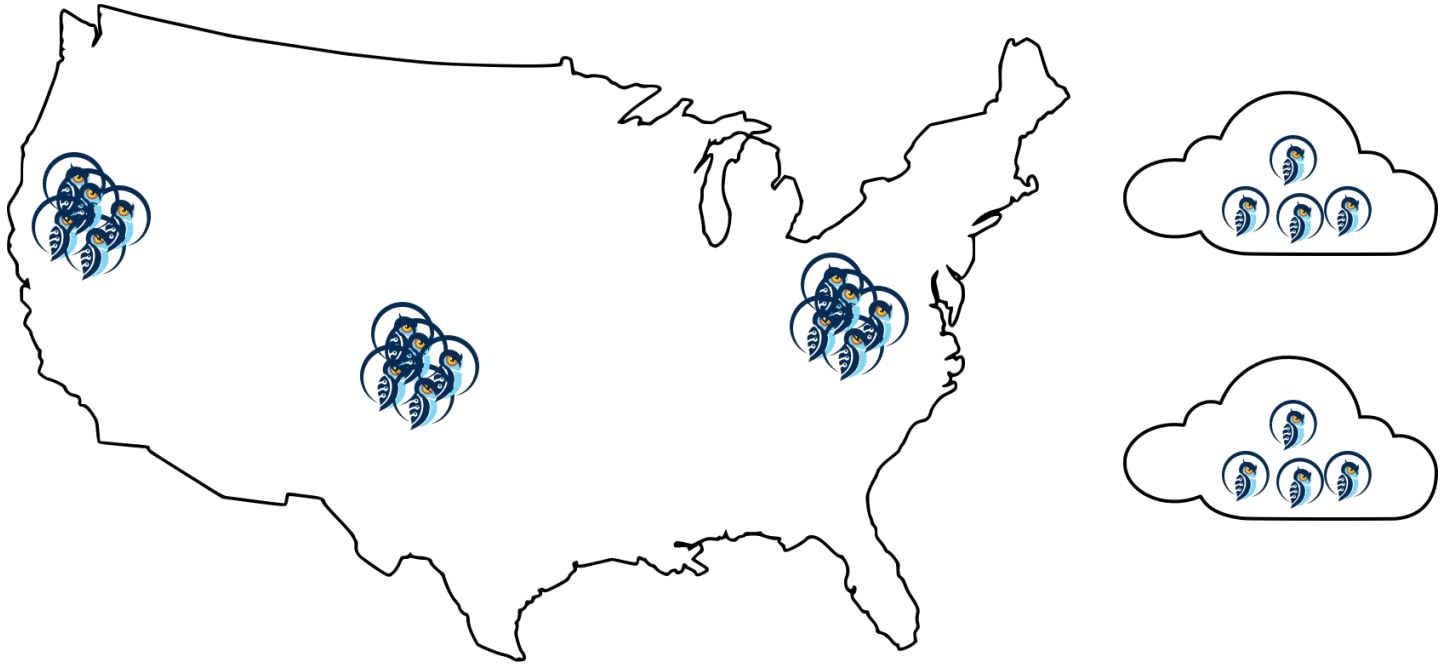
2018

40+

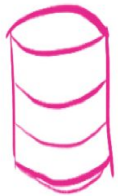
Clusters

10+

Configurations



M3DB Features



Sharding

Metrics are sharded at ingestion time

M3DB Features

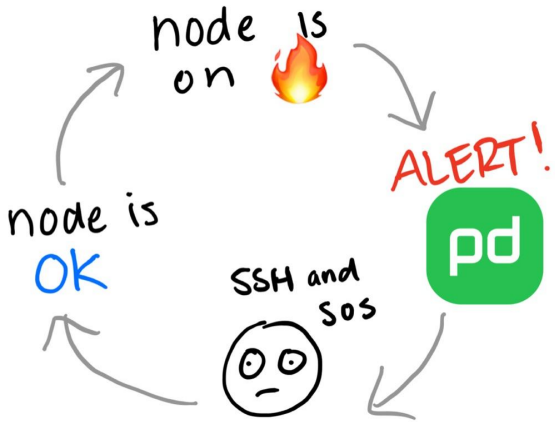
Sharding

Metrics are sharded at ingestion time

Replication

Replicates in 3 separate failure domains

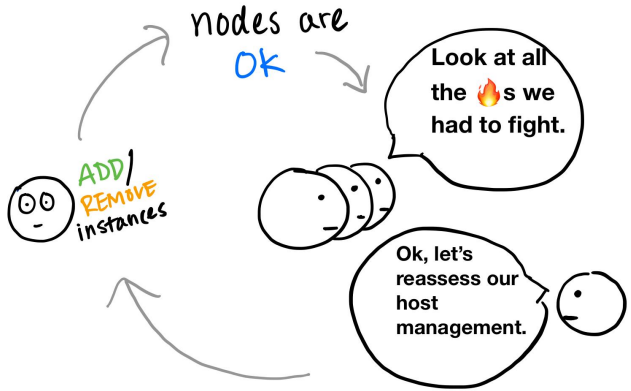
Managing M3DB Lifecycle



Reactive

1 hour per day,

5 hours per week



Proactive

2 hours per week

Managing Complexity

Cloud Native Landscape

2018-10-25T09:42:35Z 7ff5069

See the interactive landscape at l.cncf.io

Greyed logos are not open source

App Definition and Development

Database

CNCF Incubating: Vitess

Streaming & Messaging

CNCF Incubating: NATS

Application Definition & Image Build

CNCF Incubating: HELM

Continuous Integration & Delivery

Orchestration & Management

Scheduling & Orchestration

CNCF Graduated: kubernetes

Coordination & Service Discovery

CNCF Incubating: CoreDNS

Remote Procedure Call

CNCF Incubating: GRPC

Service Proxy

CNCF Incubating: envoy

API Gateway

Service Mesh

CNCF Incubating: LINKERD

Runtime

Cloud-Native Storage

CNCF Incubating: ROOK

Container Runtime

CNCF Incubating: rkt

Cloud-Native Network

CNCF Incubating: CNI

Provisioning

Automation & Configuration

Container Registries

Security & Compliance

CNCF Incubating: Notary

CNCF Incubating

Key Management

Cloud

Public

Platform

Certified Kubernetes - Distribution

Certified Kubernetes - Hosted

Certified Kubernetes - Installer

Non-Certified Kubernetes

PaaS/Container Service

Observability and Analysis

Monitoring

CNCF Graduated: Prometheus

Logging

CNCF Incubating: fluentd

Tracing

CNCF Incubating

CNCF Incubating: OPENTRACING

Chaos Engineering

Serverless

Special

Kubernetes Certified Service Provider

Kubernetes Training Partner

This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path

l.cncf.io

CLOUD NATIVE Landscape

CLOUD NATIVE COMPUTING FOUNDATION

Redpoint Amplify

Performant Stateful Primitives

Requirement #1:

Support a high-throughput, latency-sensitive workload

Ephemeral Instances?

- No durability
- Streaming terabytes of data on restart
- Dangerous reliability implications

Remote: Block Store?

- Increased latency
- We already replicate 3x
- Less portable (+ no on-prem)

Remote: Object Store?

- Deduplicate, store remotely
- Even worse latency
- Terabytes of data transfer

	Durability	Performance	Efficiency
No State	⛔	✅	⛔
Remote: Block	✅	⛔	⛔
Remote: Object	✅	⛔	⛔
???	✅	✅	✅

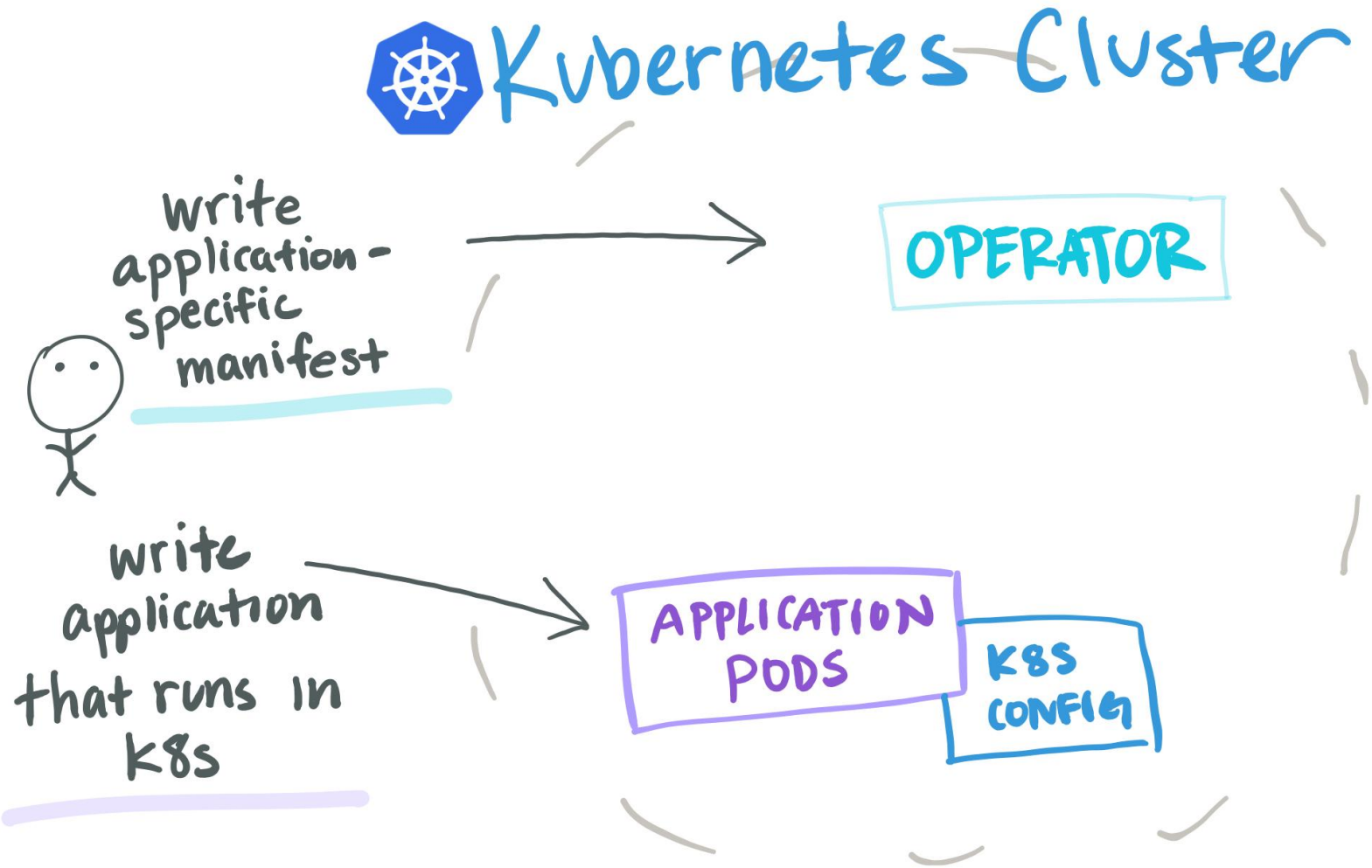
Data Centers & Cloud

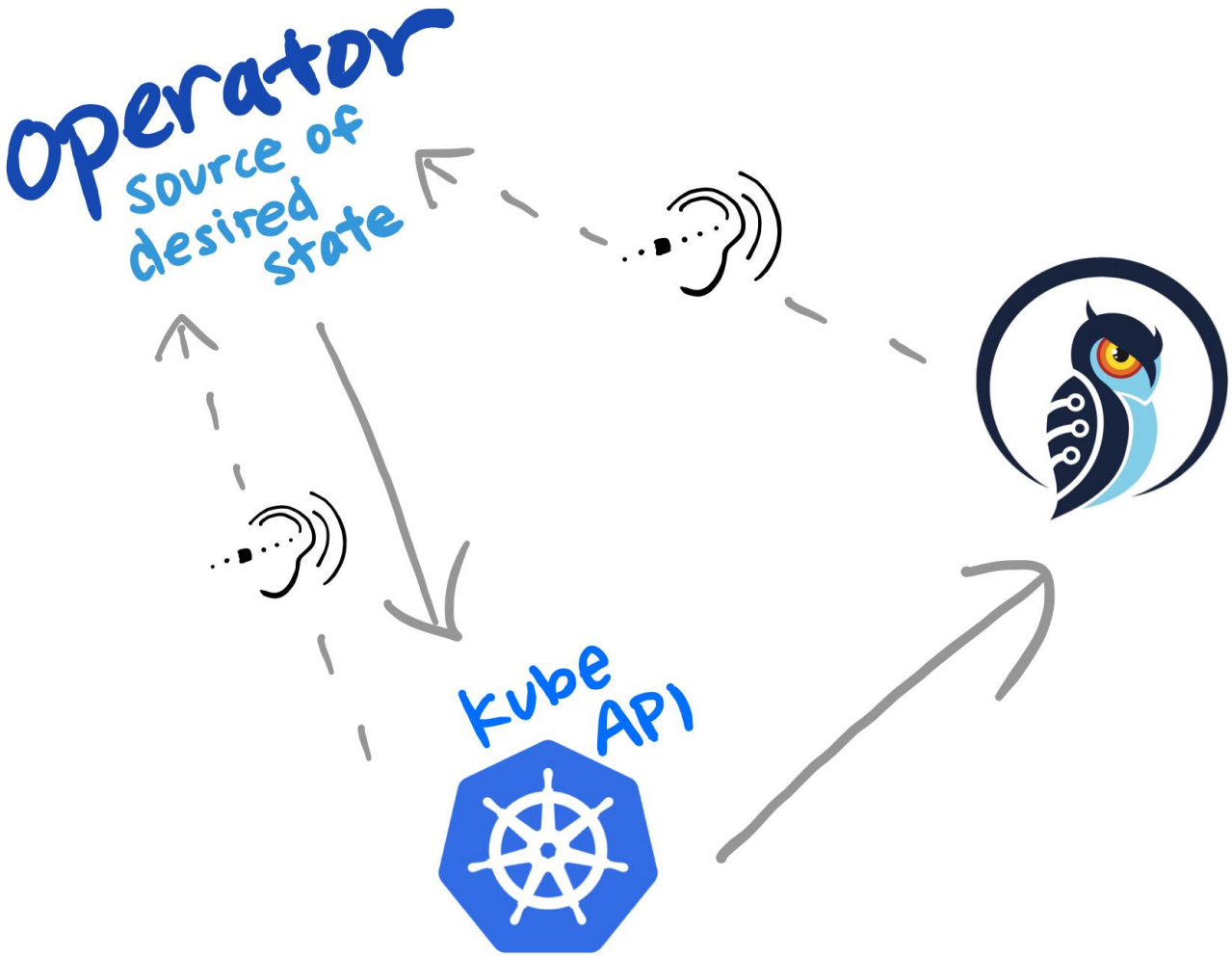
Requirement #2

Embrace the Community

Requirement #3

Uber Open Source


Kubernetes Cluster
write application-specific manifest
OPERATOR
write application that runs in k8s
APPLICATION PODS
K8S CONFIG


Operator
source of
desired
state
Kube
API

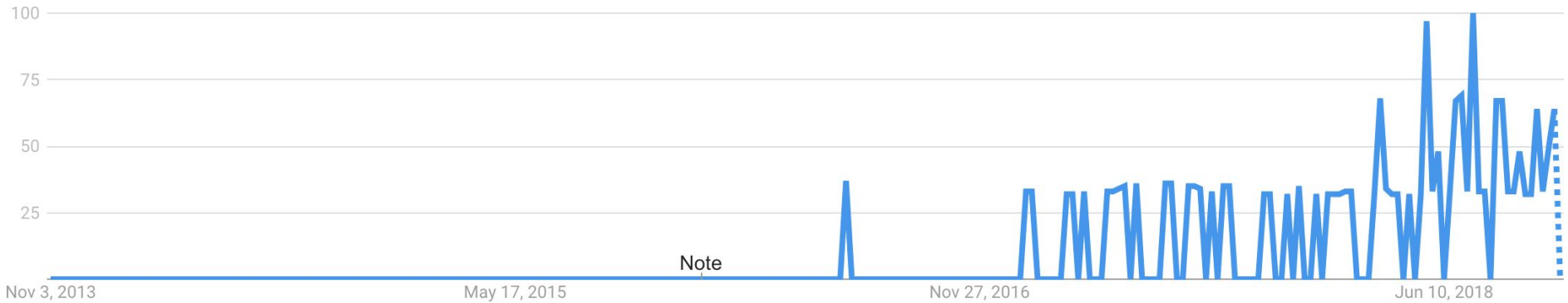
Google Trends | Explore

kubernetes operator
Search term

+ Compare

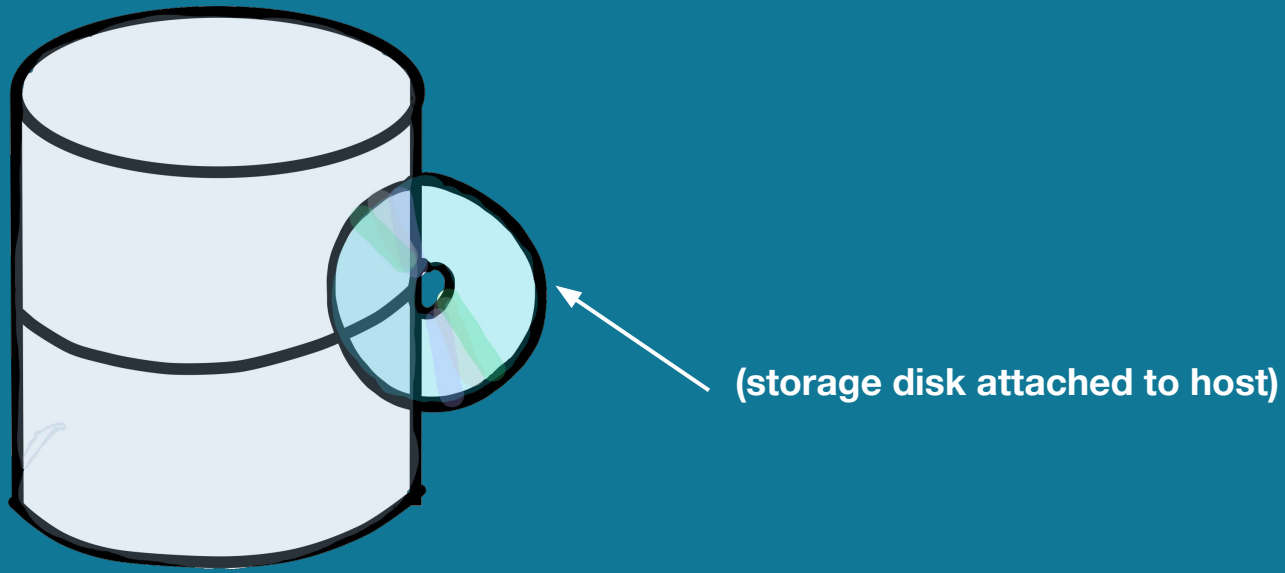
California | Past 5 years | All categories | Web Search

Interest over time



Local Volumes

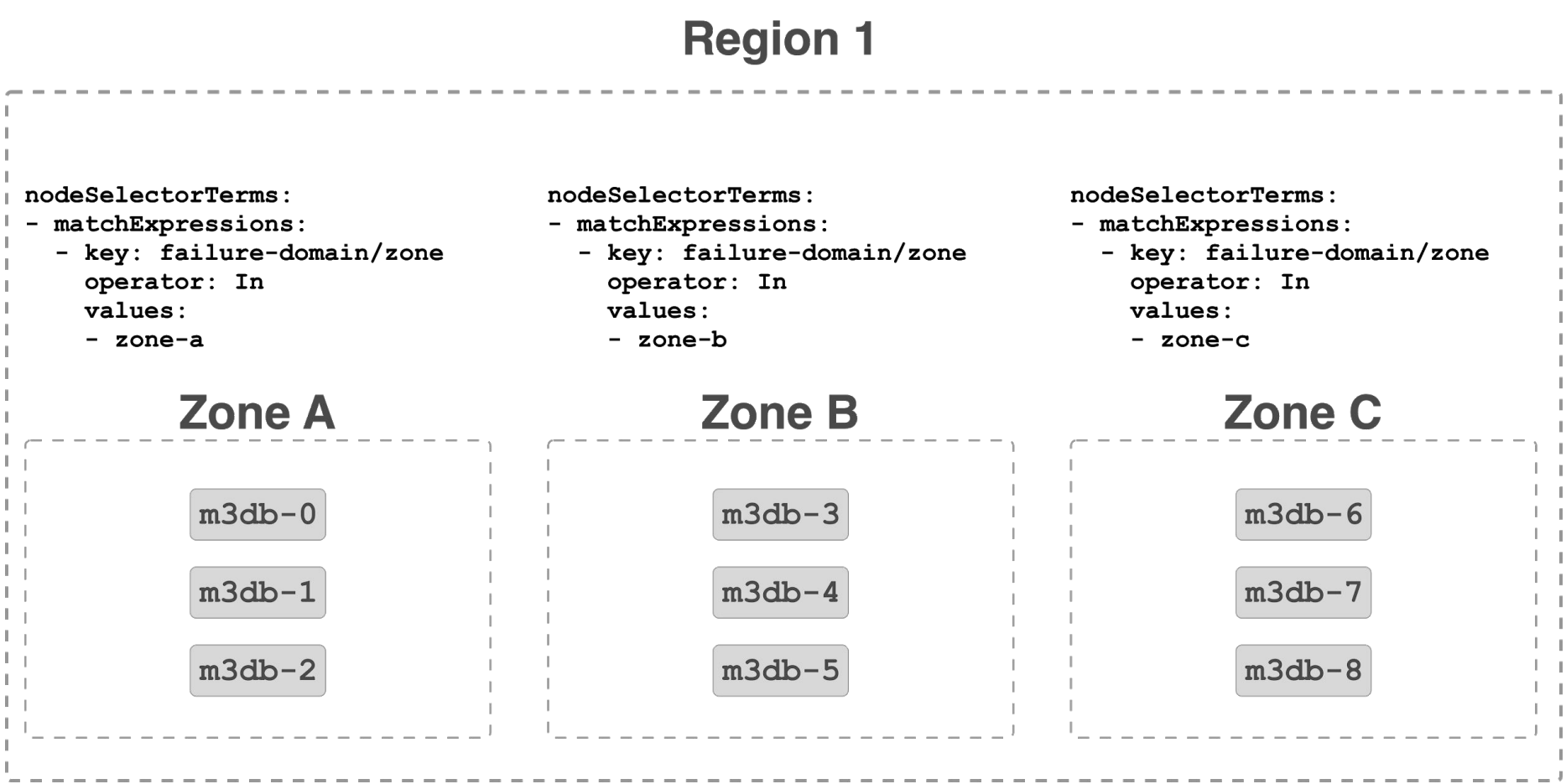
Performant Stateful Primitives



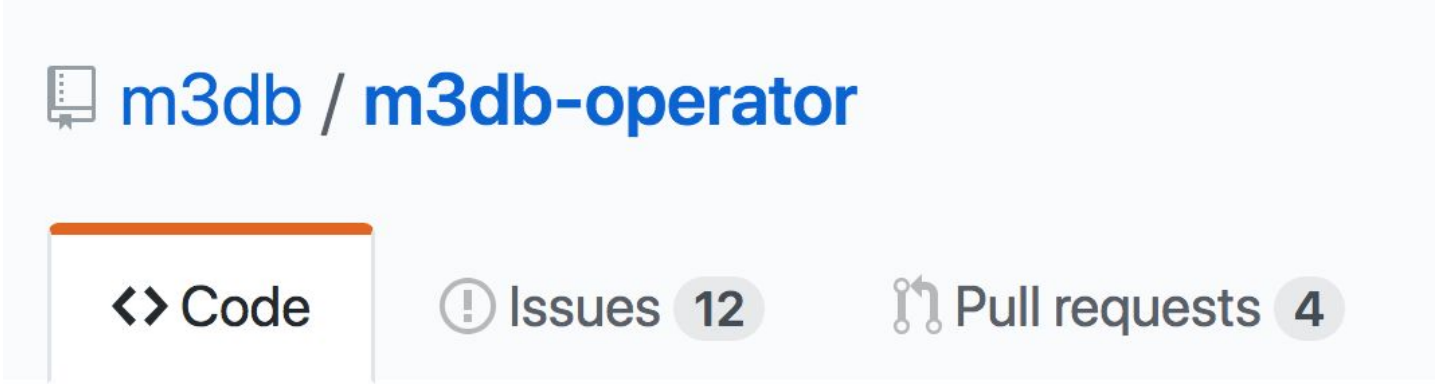
	Durability	Performance	Efficiency
No State	⛔	✅	⛔
Remote: Block	✅	⛔	⛔
Remote: Object	✅	⛔	⛔
Local Volumes: Kubernetes	✅	✅	✅

Node Affinity + StatefulSets

Data Centers and Cloud

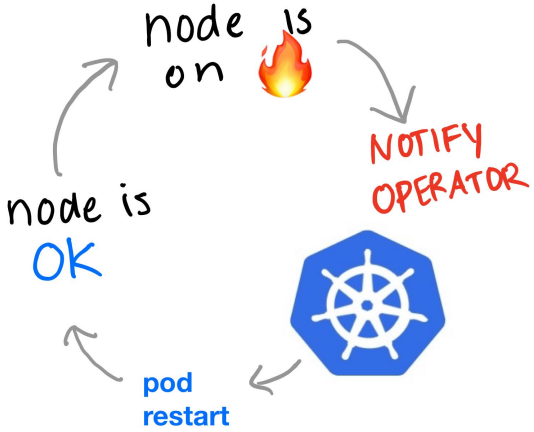

Region 1
nodeSelectorTerms:
- matchExpressions:
- key: failure-domain/zone
operator: In
values:
- zone-a
nodeSelectorTerms:
- matchExpressions:
- key: failure-domain/zone
operator: In
values:
- zone-b
nodeSelectorTerms:
- matchExpressions:
- key: failure-domain/zone
operator: In
values:
- zone-c
Zone A
m3db-0
m3db-1
m3db-2
Zone B
m3db-3
m3db-4
m3db-5
Zone C
m3db-6
m3db-7
m3db-8

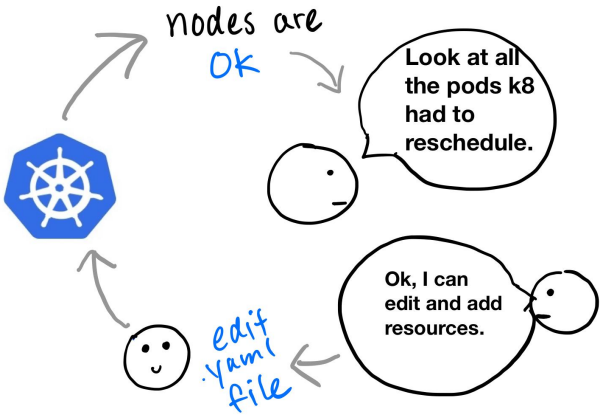
Results


m3db / m3db-operator
Code
Issues 12
Pull requests 4
Kubernetes operator for m3db

```
$ kubectl get pods
NAME                        ZONE
east1-prod-a-rep0-0         us-east1-b
east1-prod-a-rep0-1         us-east1-b
...
east1-prod-a-rep1-0         us-east1-c
east1-prod-a-rep1-1         us-east1-c
...
east1-prod-a-rep2-0         us-east1-d
east1-prod-a-rep2-1         us-east1-d
...
```

Where does our operator replace human effort?





Reactive

0 minutes / day

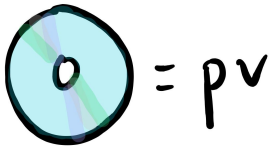
0 minutes / week

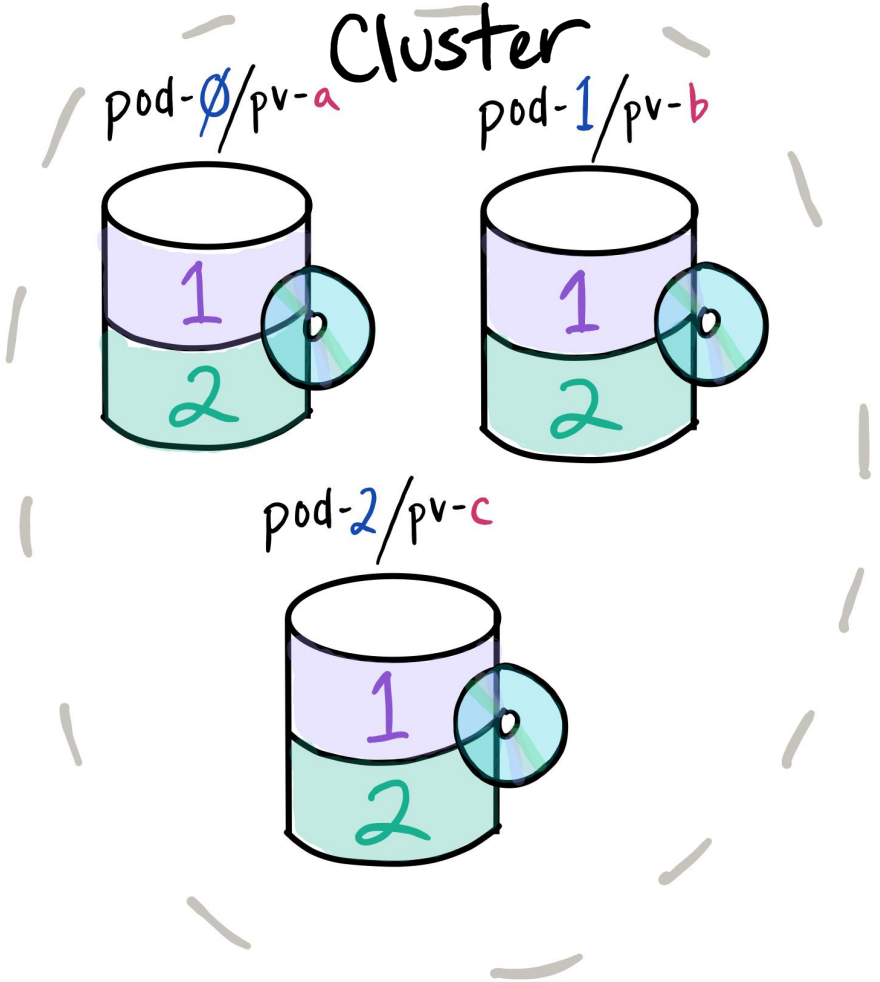
Proactive

20 minutes / week

etcd

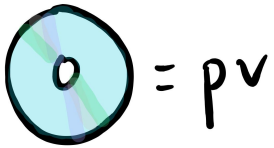
Hostname	Shards
pod-0/pv-a	1,2
pod-1/pv-b	1,2
pod-2/pv-c	1,2

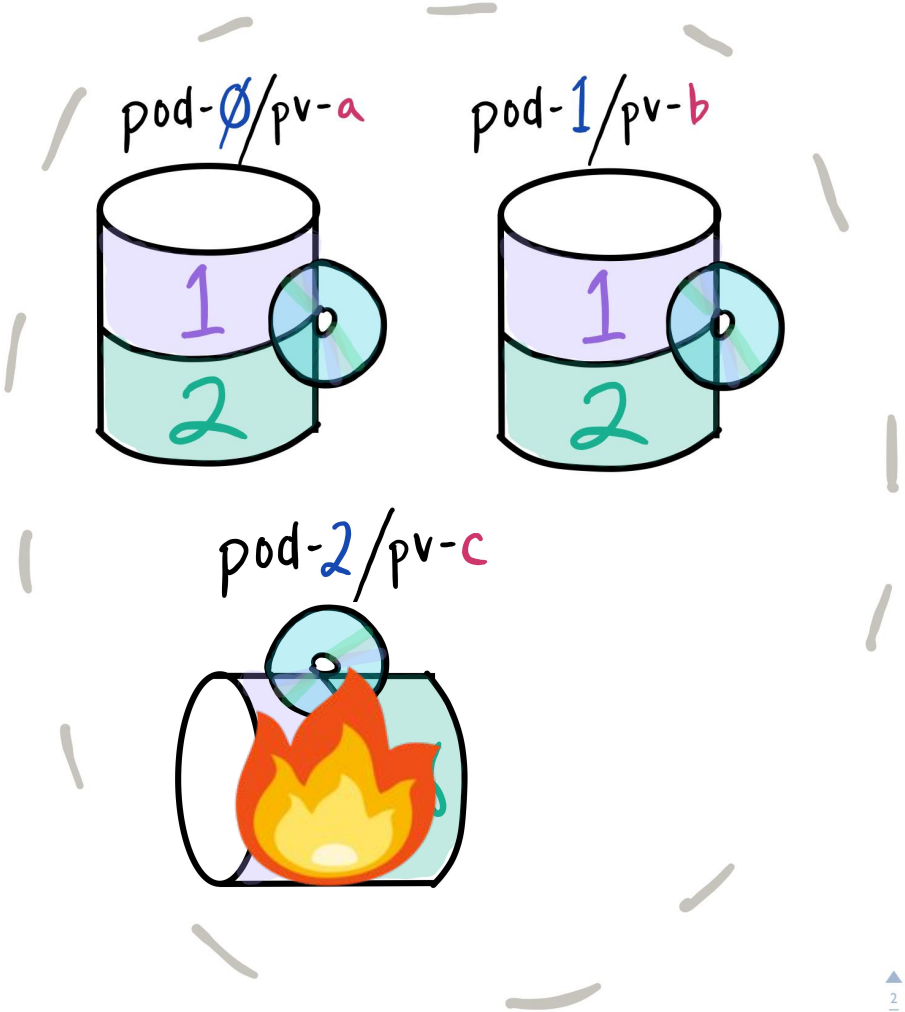




etcd

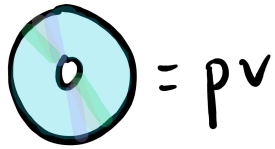
Hostname	Shards
pod-0/pv-a	1,2
pod-1/pv-b	1,2
pod-2/pv-c	1,2

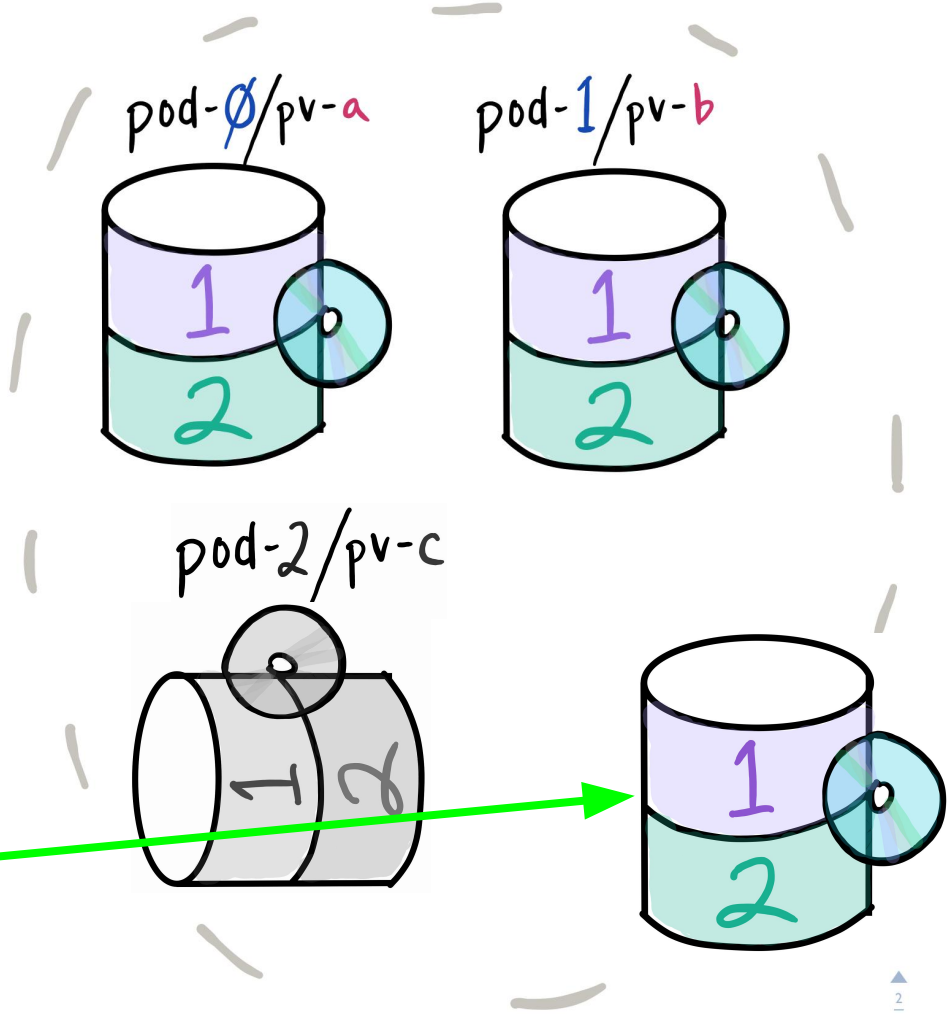




etcd

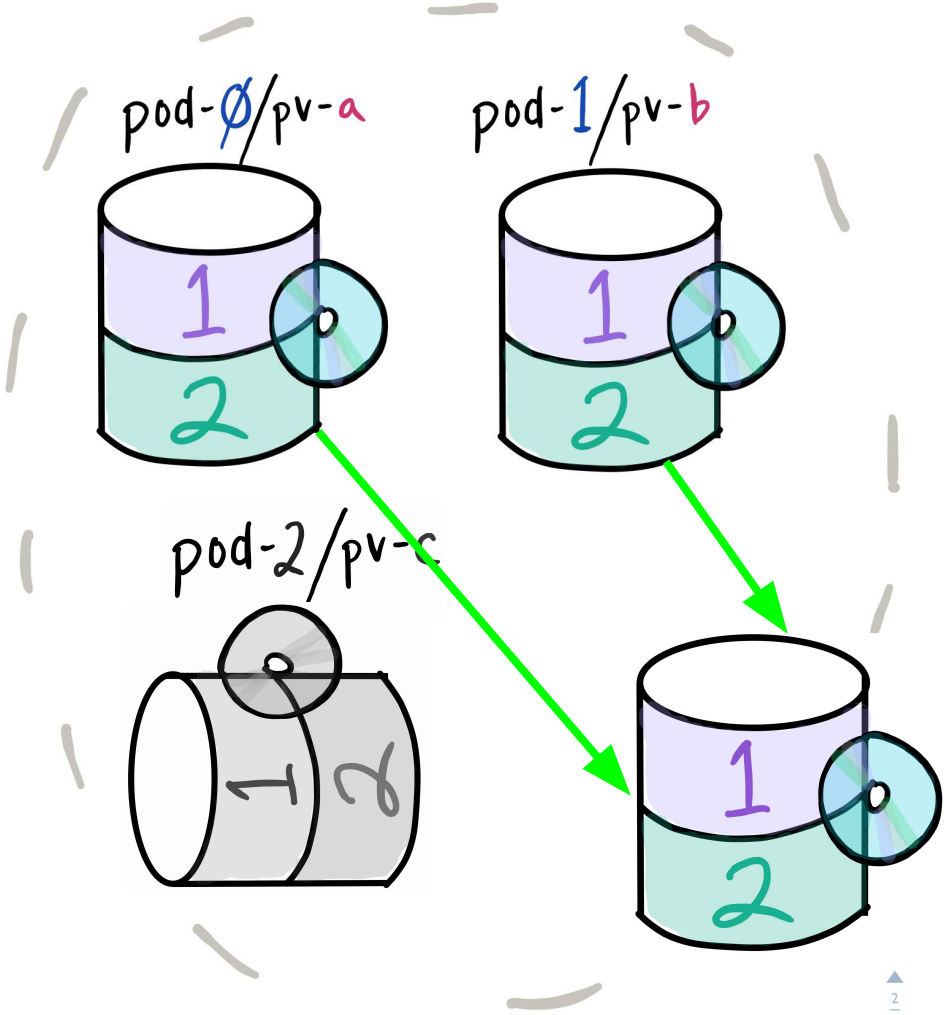
Hostname	Shards
pod-0/pv-a	1,2
pod-1/pv-b	1,2
pod-2/pv-c	1,2





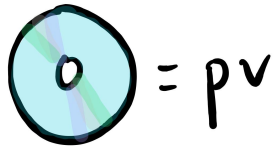
etcd

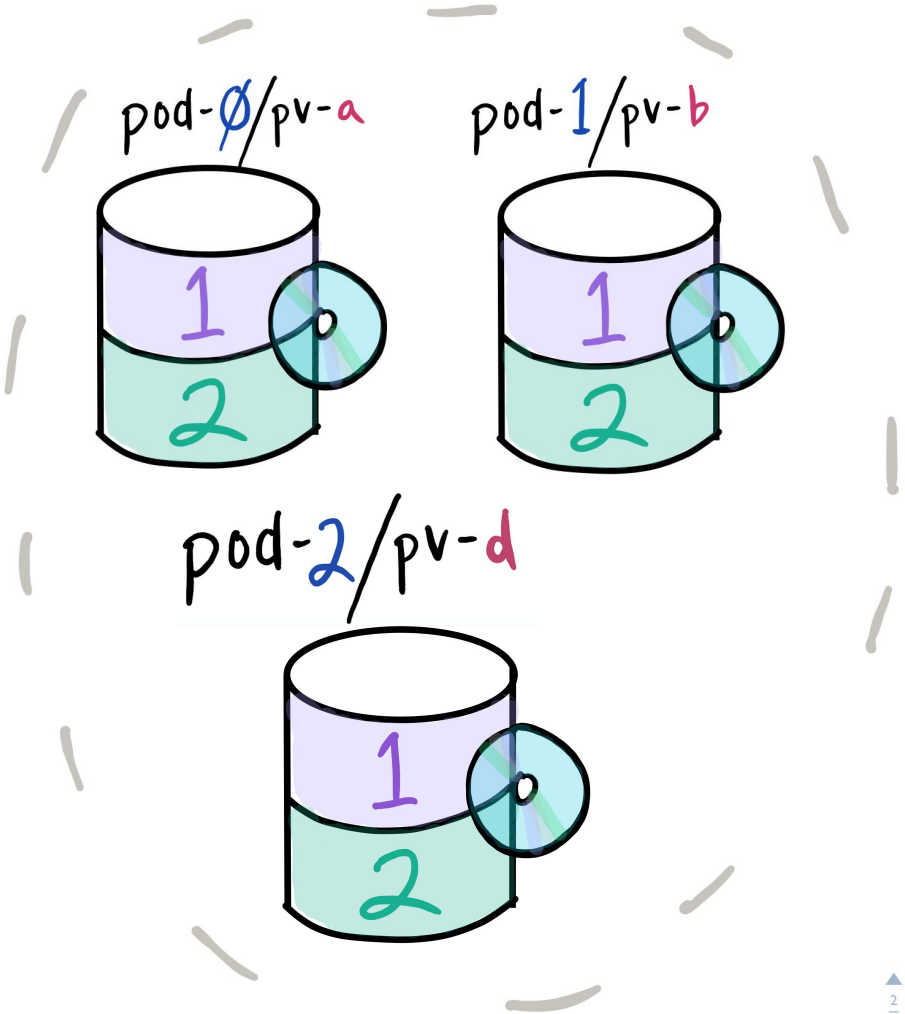
Hostname	Shards
pod-0/pv-a	1,2
pod-1/pv-b	1,2
pod-2/pv-c	1,2
pod-2/pv-d	1,2



etcd

Hostname	Shards
pod-0/pv-a	1,2
pod-1/pv-b	1,2
pod-2/pv-d	1,2





Lessons Learned

Broken Assumptions

- Kubernetes revealed assumptions we made
- Instance identity ≠ host
- Made M3DB more robust

Race in topology retrieval during bootstrap between CommitLog and DB #1011

Closed **prateek** opened this issue on Oct 3 · 1 comment

```
kubectl apply -f m3db_operator.yaml
```

M3 Documentation

Search

m3db/m3
1.3k Stars · 97 Forks

M3DB Cluster Deployment, Manually (The Hard Way)

Introduction

This document lists the manual steps involved in deploying a M3DB cluster. In practice, you'd be automating this using Terraform or using Kubernetes rather than doing this by hand; guides for doing so are available under the How-To section.

Table of contents

Primer Architecture

A quick primer on

A few different thi

Role Type

There are three 'ro

- Coordinator:
  It's a lightweig

- Storage Node: `m3dbnode` processes running on these hosts are the workhorses of the database; they store data; and serve reads and writes.
- Seed Node: First and foremost, these hosts are storage nodes themselves. In additi responsibility, they run an embedded ETCD server. This is to allow the various M3DB running across the cluster to reason about the topology/configuration of the cluste manner.

Note: In very large deployments, you'd use a dedicated ETCD cluster, and only use M3DB Coordinator Nodes

```
            "zone": "embedded",
            "weight": 100,
            "endpoint": "10.142.0.1:9000",
            "hostname": "m3db001",
            "port": 9000
        },
        {
            "id": "m3db002",
            "isolation_group": "us-east1-b",
            "zone": "embedded",
            "weight": 100,
            "endpoint": "10.142.0.2:9000",
            "hostname": "m3db002-us-east",
            "port": 9000
```

pass your host ID when launching the database process with an environment variable. You can set to the host ID and specify the environment variable name in config as `envVarName: M3DB_HOST_ID` if you are using an environment variable named `M3DB_HOST_ID`.

Relevant config snippet:

```
hostID:
  resolver: environment
  envVarName: M3DB_HOST_ID
```

Then start your process with:

```
M3DB_HOST_ID=m3db001 m3dbnode -f config.yml
```

Kernel

Ensure you review our recommended kernel configuration before running M3DB in production as M3DB may exceed the default limits for some default kernel values.

Config files

We wouldn't feel right to call this guide, "The Hard Way" and not require you to change some configs by hand.

Note: the steps that follow assume you have the following 3 seed nodes - make necessary adjustment if you have more or are using a dedicated ETCD cluster. Example seed nodes:

- m3db001 (Region=us-east1, Zone=us-east1-a, Static IP=10.142.0.1)
- m3db002 (Region=us-east1, Zone=us-east1-b, Static IP=10.142.0.2)
- m3db003 (Region=us-east1, Zone=us-east1-c, Static IP=10.142.0.3)

We're going to start with the M3DB config template and modify it to work for your cluster. Start by downloading the config. Update the config 'service' and 'seedNodes' sections to read as follows:

```
config:
```

```
    - hostID: m3db001
      endpoint: http://10.
    - hostID: m3db002
      endpoint: http://10.
    - hostID: m3db003
      endpoint: http://10.
```

create an instance click "Management, disks, networking, SSH keys" and under "Networ default interface and click the "Primary internal IP" drop down and select "Reserve a stat address" and give it a name, i.e. `m3db001`, a description that describes it's a seed node l use "Assign automatically".

Start the seed nodes

Transfer the config you just crafted to each host in the cluster. And th start up the m3dbnode process:

```
m3dbnode -f <config-name.yml>
```

Note, remember to daemon-ize this using your favourite utility: syste

Initialize Topology

M3DB calls its cluster topology 'placement'. Run the command below initialize your first placement.

Note: Isolation group specifies how the cluster places shards to avoi appearing in the same replica group. As such you must be using at le your replication factor. In this example we use the availibity zones `u east1-c` as our isolation groups which matches our replication fact

```
curl -X POST localhost:7201/api/v1/placement/init -d '{
```

```
    "writesToCommitLog": true,
    "cleanupEnabled": true,
    "snapshotEnabled": true,
    "repairEnabled": false,
    "retentionOptions": {
```

Read more about namespaces and the various knobs in the docs.

Test it out

Now you can experiment with writing tagged metrics:

```
curl -sSf -X POST localhost:9003/writetagged -d '{
  "namespace": "metrics",
  "id": "foo",
  "tags": [
    {
      "name": "city",
      "value": "new_york"
    },
    {
      "name": "endpoint",
      "value": "/request"
```

```
alue": 42.123456789
    }
}'
```

And reading the metrics you've written:

```
curl -sSf -X POST http://localhost:9003/query -
  "namespace": "metrics",
  "query": {
    "regexp": {
      "field": "city",
      "regexp": ".*"
    }
```

Advice for Large Stateful Workloads

Out-of-Cluster Reliability

- Years invested in M3DB reliability & tooling
- Considered Kubernetes once we faced operational scaling challenge
- Be mindful of adding complexity

Declarative > Imperative

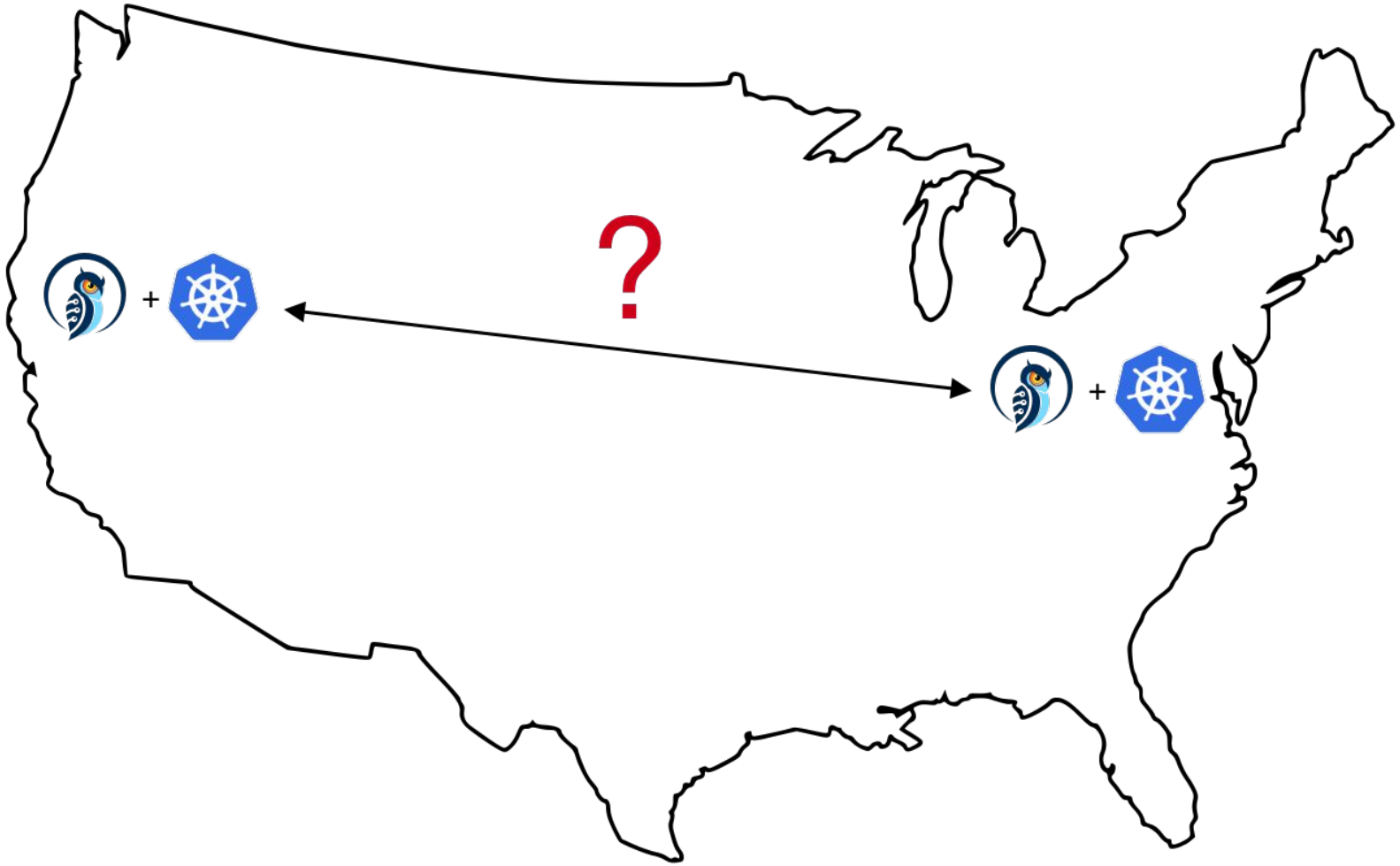
- Core to Kubernetes, great for stateful
- Operator exchanged desired states between Kubernetes and M3DB
- Storing topology externally → no hard dependency on Kubernetes API

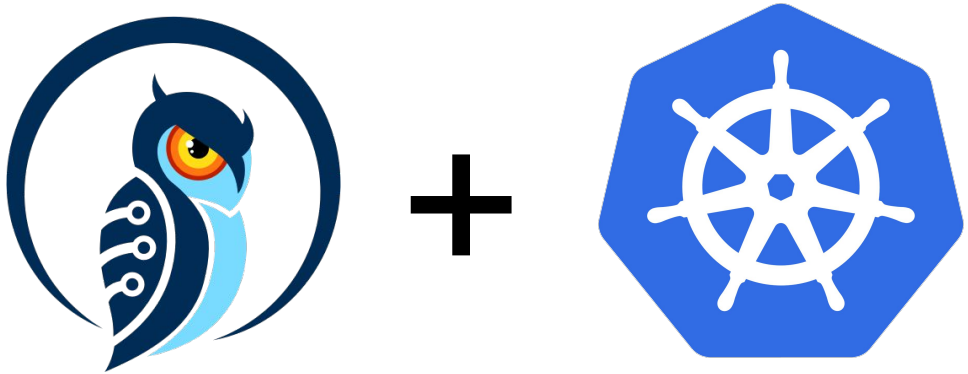
Iterate on Each Stateful Interaction

- Don’t try to do everything at once
- Edge case scenarios still need humans

Next Steps

- Data centers…
- Auto-scale M3DB clusters





Thank You to the Team

Special shout out to Paul Schooss

github.com/m3db/m3db-operator

m3db.io/talks

eng.uber.com/m3

@shaleenaa @mattschallert