Weaviate

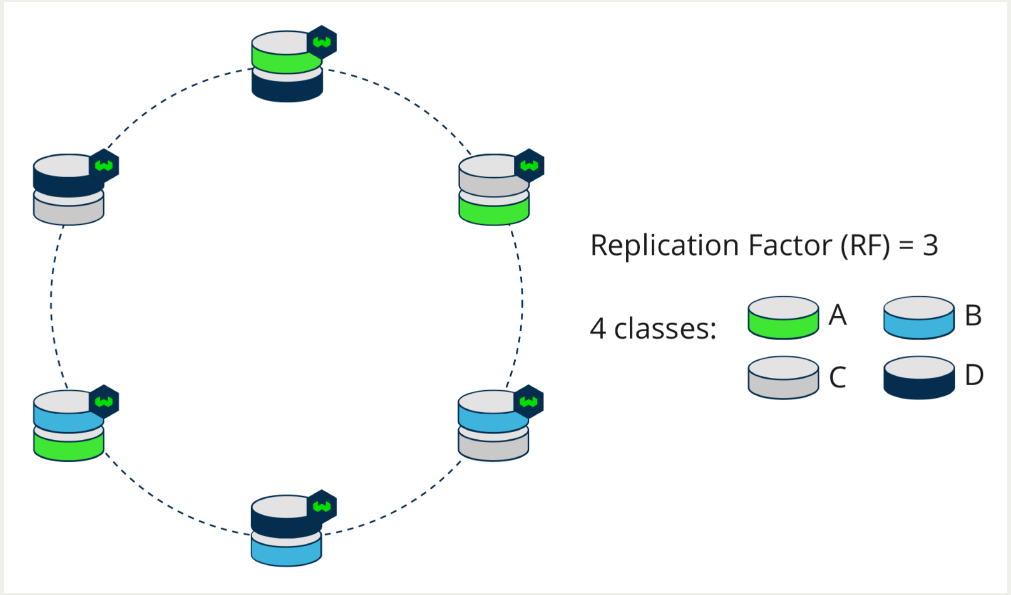
Rafting into the future: Weaviate Evolution Unleashed - Why You Should Care

Agenda

- Introduction
- Current implementation limitations[Demo]
- RAFT-Based Solution[Demo]
- Local deployment [Demo]
- Kubernetes deployment
- Kubernetes in action [Demo]
- Software Design
- Workflow for Schema Updates
- Trade-offs
- Future plans
- Q&A

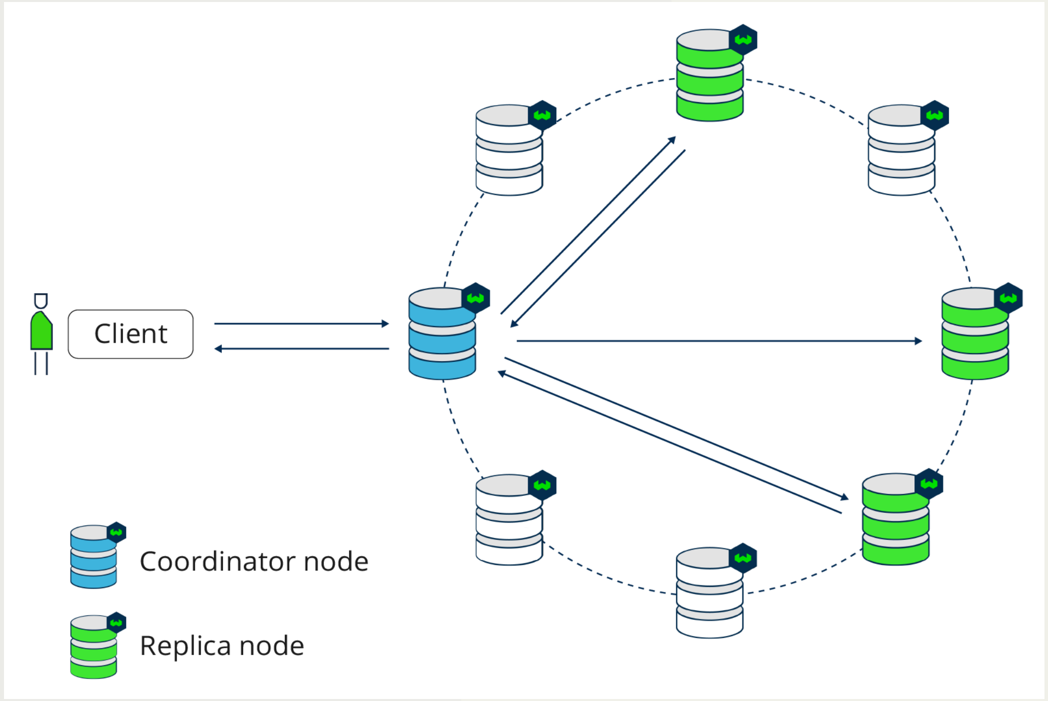
Introduction

Weaviate Leaderless Architecture: Overview



- Cassandra Architecture, P2P, high scalability

Data Replication



Schema Synchronization

- 2-Phase commit to process changes
- Higher latency but strong consistency
- No concurrent schema update
- Present and implemented when it was needed, for the challenges at point in time.

- 2PC Process:
  1. Are all node able to receive and process the change ?
  2. Coordinator commits the change to all nodes

Schema Synchronization Limitations

- Every node must have an up to date to act as coordinator
  - What if a node ends up with an inconsistent schema ?
- 2PC requires all node to be healthy
  - What if a node is in a crashloop ?
- No concurrent schema update
  - What if a user needs to create 1000 tenants in parallel ?
  - What if a user has a very high frequency of schema update ?

Current Limitations

Demo

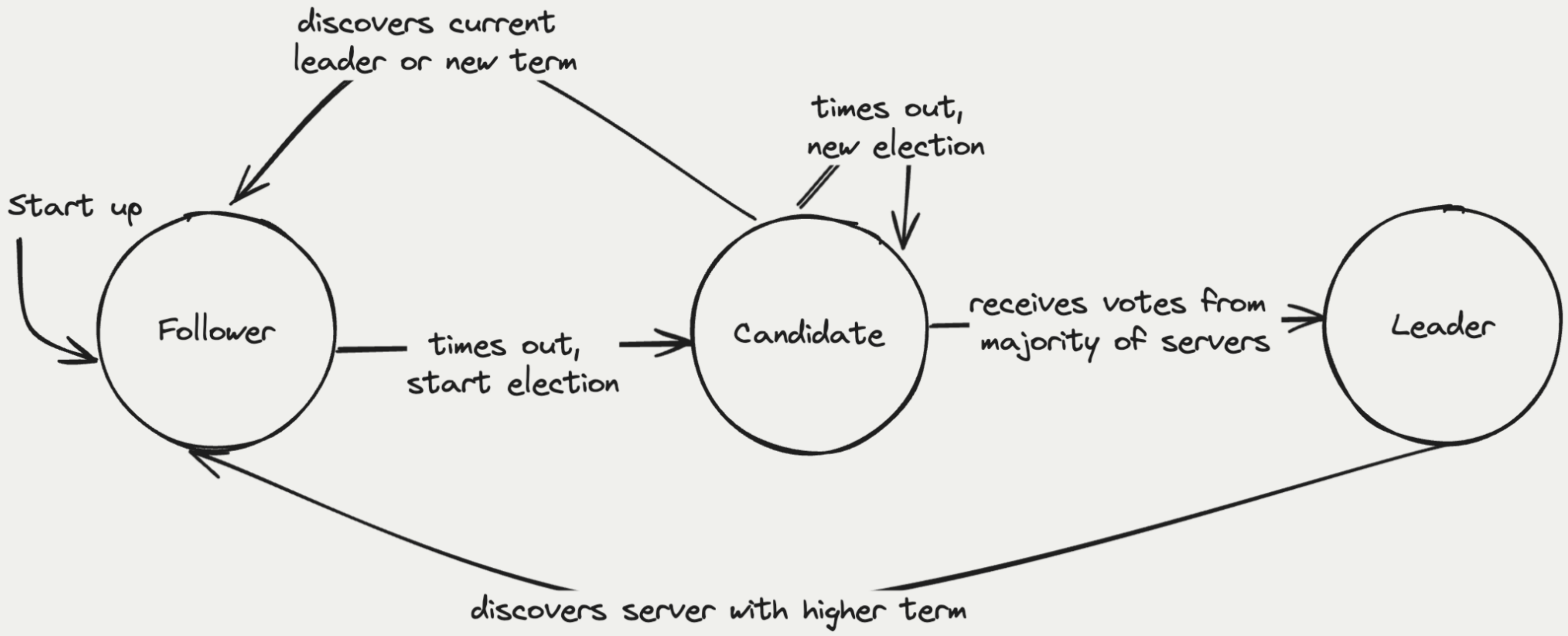
Consensus-Based Solution

RAFT
THE FINAL CHAPTER
1

RAFT Algorithm

- **R**eliable, **R**eplicated, **R**edundant **A**nd **F**ault **T**olerant

RAFT in Nutshell


discovers current leader or new term
Start up
times out, new election
Follower
times out, start election
Candidate
receives votes from majority of servers
Leader
discovers server with higher term

Raft Guarantees

- Election safety
- Leader append only
- Log matching
- State machine safety

Raft implementation

- Fault tolerant
- Reliable
- Available
- Concurrent updates
- Eventually consistent is challenging

Non-Raft implementation

- Strongly consistent most of the time
- Simple implementation
- Requires all node to be online
- Concurrent transactions
- Schema could diverge
- Requires manual intervention which could pose risk

RAFT in Action

Demo

Kubernetes deployment

Demo

Local deployment

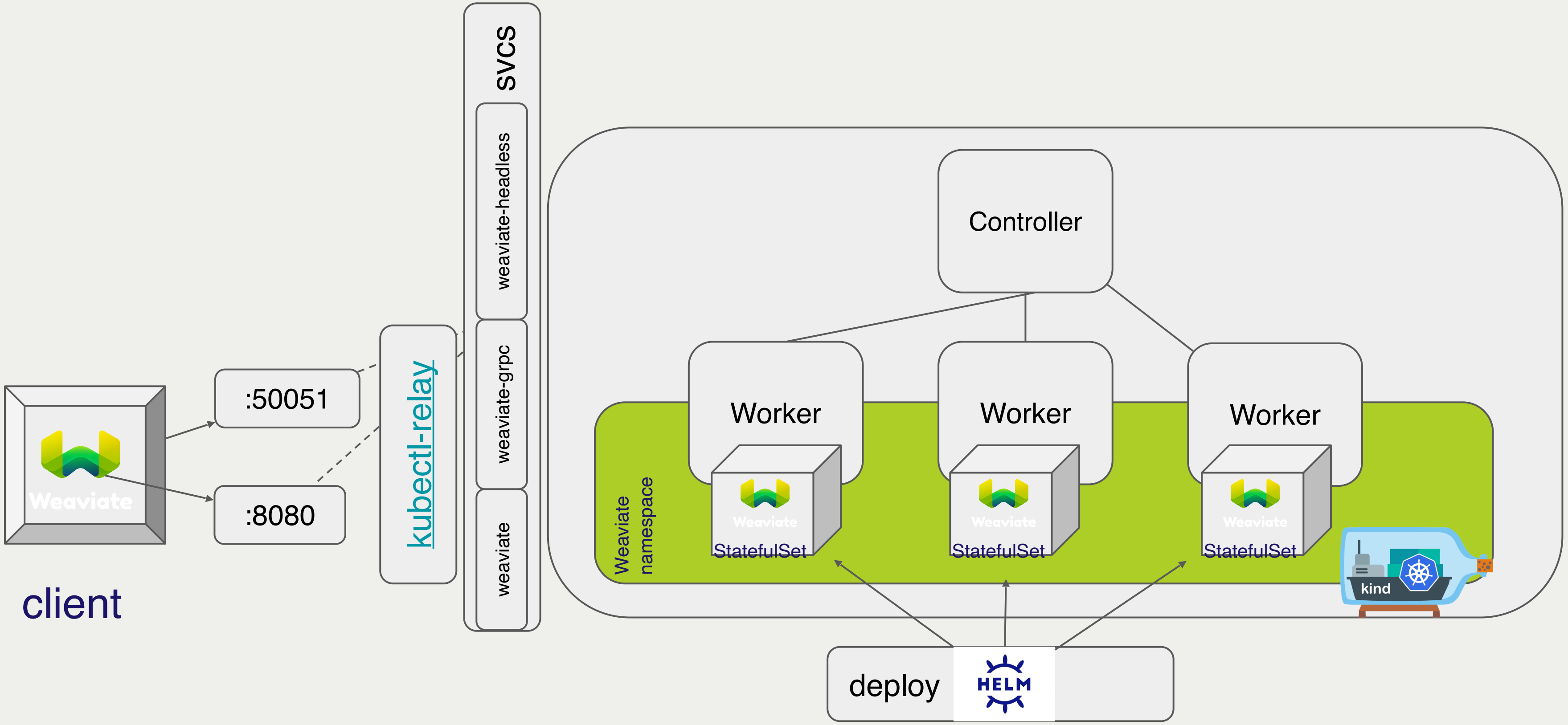
Demo

Local deployment options

- **Run_dev_server.sh**:
  - Running three nodes using the Golang binary
- **Docker-compose-raft**:
  - Creates a multi-node docker setup with a variable set of nodes
  - `./docker-compose-raft/raft_cluster.sh <NUM_VOTERS>`
  - Creates `docker-compose-raft.yml` file, run with:

    ```
    ./docker-compose-raft/raft_cluster.sh 3
    docker-compose -f docker-compose-raft.yml up -d --scale weaviate=2
    ```
- **Local Kubernetes cluster**:
  - Creates a full kubernetes cluster with several nodes (or only one if desired), deploying Weaviate with weaviate-helm.

RAFT Local k8s cluster



RAFT Local k8s cluster

- Parameters:
  - Global:
    - **WEAVIATE_VERSION**: Weaviate docker image to use.
    - **REPLICAS**: Number of nodes in the cluster. It will create as many workers as Weaviate instances
    - **HELM_BRANCH**: Branch from weaviate-helm to use during deployment.
  - Local:
    - **WEAVIATE_PORT**: Local port in which weaviate's load balancer is forwarded. Default: 8080
    - **WEAVIATE_GRPC_PORT**: Local port in which weaviate-grpc service is forwarded. Default: 50051

Kubernetes in action

Demo

Design goals

Solve one and only one Problem

Better structure
Separation of concerns, extensibility, better code structure

Maintain System Behavior

Seamless Deployment

Smooth Migration

Availability, Fault tolerance, Reliability

Workflow for Schema updates

- **H**TTP Handler for Validation
- Raft Store for Schema Updates
- *Executor for Local Execution*

Trade-offs

Eventual consistency

Future Plans

Near goals: finalizing implementation, improve handling of EC, enhance observability

Mid-term goals: boast performance,shard movement

Resources

- https://raft.github.io/
- http://thesecretlivesofdata.com/raft/

Q&A

Thank you