Dqlite: High-availability SQLite

Fast, embedded, persistent SQL database with Raft consensus

Free Ekanayaka
Dqlite author & LXD core developer

free.ekanayaka@canonical.com

CANONICAL ubuntu

In the beginning

When would you use SQLite?

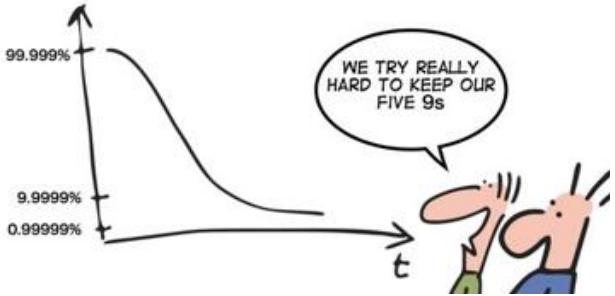
01 Embedded devices and Internet of Things
Cellphones, medical devices, airplanes, machine tools.

02 Agents
Container managers, continuous integration workers, monitoring daemons.

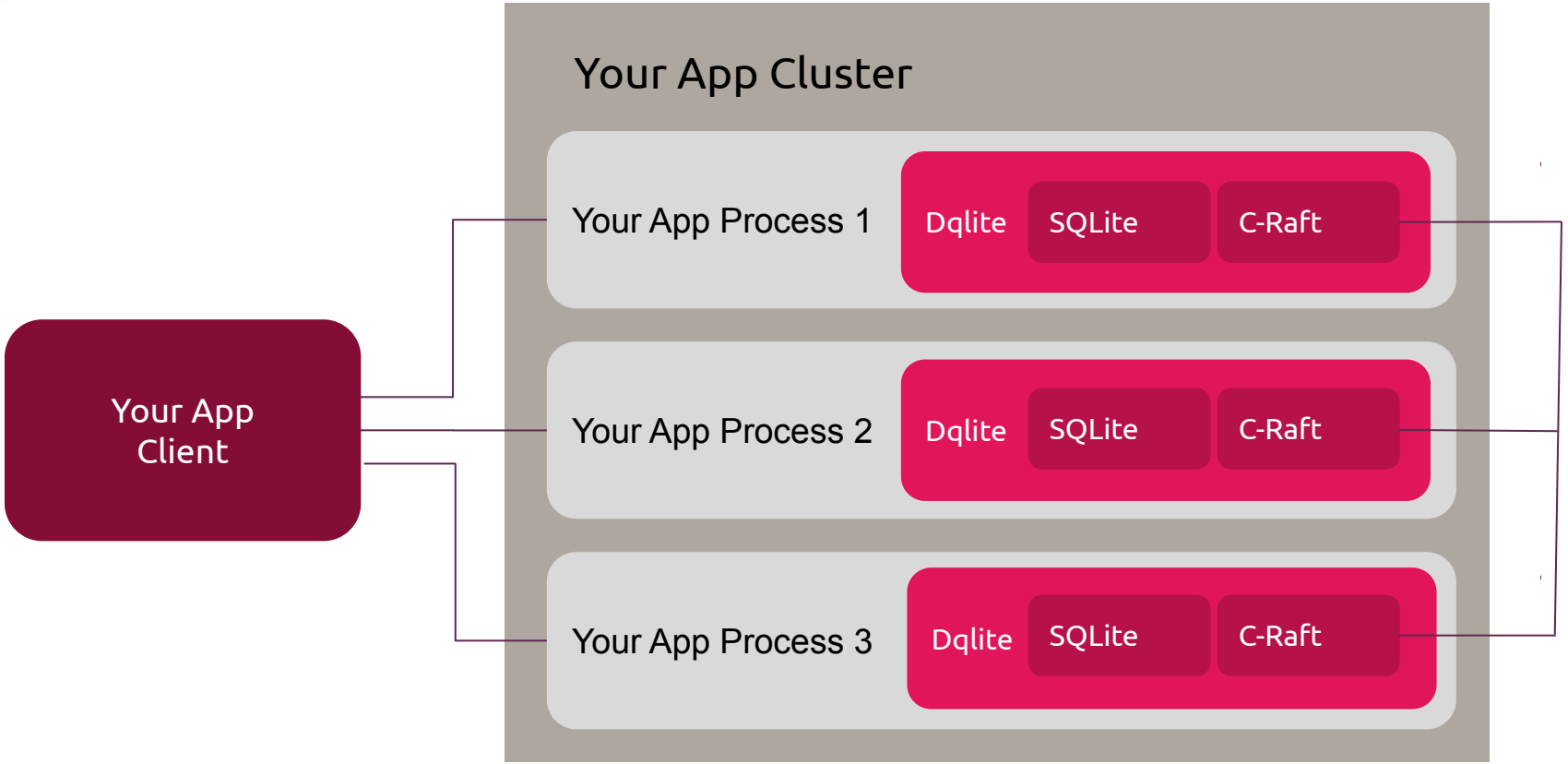
03 Desktop applications
Browsers, media cataloging, record-keeping programs.

When would you use Dqlite?

HA!



How does it work



Talk is cheap. Show me the code.

Case study:

A pulse oximeter

“Medical device used to monitor the amount of oxygen carried in the body.”

Features

01 Measure oxygen saturation at regular intervals.

02 Persist measurement history in an embedded database.

03 Expose an API to retrieve average saturation over a given time interval.

Measure oxygen saturation

```
func measureSaturation() float64 {
    return 95.0 + 5*rand.Float64()
}
```

Database initialization

```
func getDatabase() *sql.DB {
    db, err := sql.Open("sqlite3", "oximeter.db")
    if err != nil {
        log.Fatal(err)
    }
    _, err = db.Exec(
        "CREATE TABLE IF NOT EXISTS saturation " +
            "(value FLOAT, timestamp DATETIME DEFAULT CURRENT_TIMESTAMP)")
    if err != nil {
        log.Fatal(err)
    }
    return db
}
```

Persist saturation measurement

```
func persistSaturation(db *sql.DB, value float64) {
    err := db.Exec("INSERT INTO saturation (value) VALUES(?)", value)
    if err != nil {
        log.Fatal(err)
    }
}
```

Retrieve average saturation

```
func retrieveAverageSaturation(db *sql.DB, tail time.Duration) float64 {
    row := db.QueryRow(
        "SELECT avg(value) FROM saturation WHERE time >= ?",
        time.Now().UTC().Add(-tail))
    var average float64
    err := row.Scan(&average)
    if err != nil {
        log.Fatal(err)
    }
    return average
}
```

Putting it all together

```
func main() {
    db := getDatabase()
    go func() {
        for {
            persistSaturation(db, measureSaturation())
            time.Sleep(15*time.Second)
        }
    }()
    handler := http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        tail, err := time.ParseDuration(r.URL.Query()["tail"][0])
        if err != nil {
            log.Fatal(err)
        }
        io.WriteString(w, fmt.Sprintf("%f\n", retrieveSaturationAverage(db, tail)))
    })
    http.ListenAndServe(":8080", handler)
}
```

What if

01 The underlying storage media dies and data is lost.

02 The device needs to be replaced without interrupting the monitoring.

03 The measurement could be inaccurate and more than one device is needed.

Use Dqlite!

01 Run a cluster of three pulse oximeters.

02 Measurement data is replicated on all nodes.

03 If one node dies, it’s still all good.

Assign an identity to each node

```
func main() {
    id, _ := strconv.Atoi(os.Args[1])
    db := getDatabase()
    db := getDatabase(id)
    // ...
    http.ListenAndServe(":8080", handler)
    http.ListenAndServe(fmt.Sprint(":808%d", id), handler)
}
```

Cluster database

```
func getDatabase(id uint64) *sql.DB {
    address := fmt.Sprintf("127.0.0.1:900%d", id)
    startEngine(id, address)
    joinCluster(id, address)
    registerDriver()
    db, err := sql.Open("sqlite3", "oximeter.db")
    db, err := sql.Open("dqlite", "oximeter.db")
    // ...
}
```

Start the replication engine

```
func startEngine(id uint64, address string) {
    dir := fmt.Sprintf("./oximeter-data-%d", id)
    os.Mkdir(dir, 0755)
    node, err = dqlite.New(
        id, dir, address,
        dqlite.WithBindAddress(address),
        dqlite.WithNetworkLatency(10*time.Millisecond))
    if err != nil {
        log.Fatal(err)
    }
    if err := node.Start(); err != nil {
        log.Fatal(err)
    }
}
```

Join the cluster

```
func joinCluster(id uint64, address string) {
    if id == 1 {
        return
    }
    cli, err := client.New(context.Background(), "127.0.0.1:9001")
    if err == nil {
        cli.Add(context.Background(), client.NodeInfo{
            ID: id,
            Address: address,
        })
        cli.Close()
    }
}
```

Register the driver

```
func registerDriver() {
    store := client.NewInmemNodeStore()
    store.Set(context.Background(), []client.NodeInfo{
            {Address: "127.0.0.1:9001"},
            {Address: "127.0.0.1:9002"},
            {Address: "127.0.0.1:9003"},
    })
    driver, err := driver.New(store)
    if err != nil {
        log.Fatal(err)
    }
}
```

Profit!

```
~$ wc -l < oximeter-ha.go
133
```

Questions so far?

Extended play.

CAP theorem: pick two

01 Consistent. YES.

02 Every request receives a (non-error) response. NO.

03 Tolerant to network partitions. YES.

Taint SQLite

01 Upstream modified to support replication hooks during transaction lifecycle.

02 Patch is small and regularly maintained (mostly conflict-free so far).

03 Will need to grow to support Btree-based and statement-based replication .

Upstreaming

01 ”SQLite is not open-contribution”.

02 ”The project does not accept patches”.

03 ”Only 27 individuals have ever contributed any code to SQLite, and of those only 16 still have traces in the latest release. Only 3 developers have contributed non-comment changes within the previous five years and 96.4% of the latest release code was written by just two people.”

Wire protocol

01 Clients always connect to the leader (read-only on followers is planned).

02 Binary request/response API

03 Designed to be CPU-efficient (Cap'n Proto zero-serialization style).

Fully asynchronous I/O

01 Single thread engine.

02 Event loop powered by libuv (epoll for network).

03 Kernel AIO for disk (io_uring planned).

Replication

01 Full Write-Ahead Log pages (current).

02 Btree cells (planned).

03 SQL statements (planned).

Client and bindings

01 Go: complete.

02 C: for internal use in unit tests.

03 Wire protocol is documented, write a client for your favorite language!

Questions ?

Website: https://dqlite.io
GitHub: https://github.com/canonical/dqlite

Toy oximeter: https://bit.ly/2U9RoL7

Free ekanayaka
Dqlite author and LXD core developer

free.ekanayaka@canonical.com