High Availability

Stardog clustering for High Availability

Taught by:

Joseph Hayes
Senior Software Engineer

Learning Objectives

- Become familiar with Stardog cluster usage
- Understand node purposes
- Build cluster with ZooKeeper and a load balancer
- Set up a cache node to cache a VG

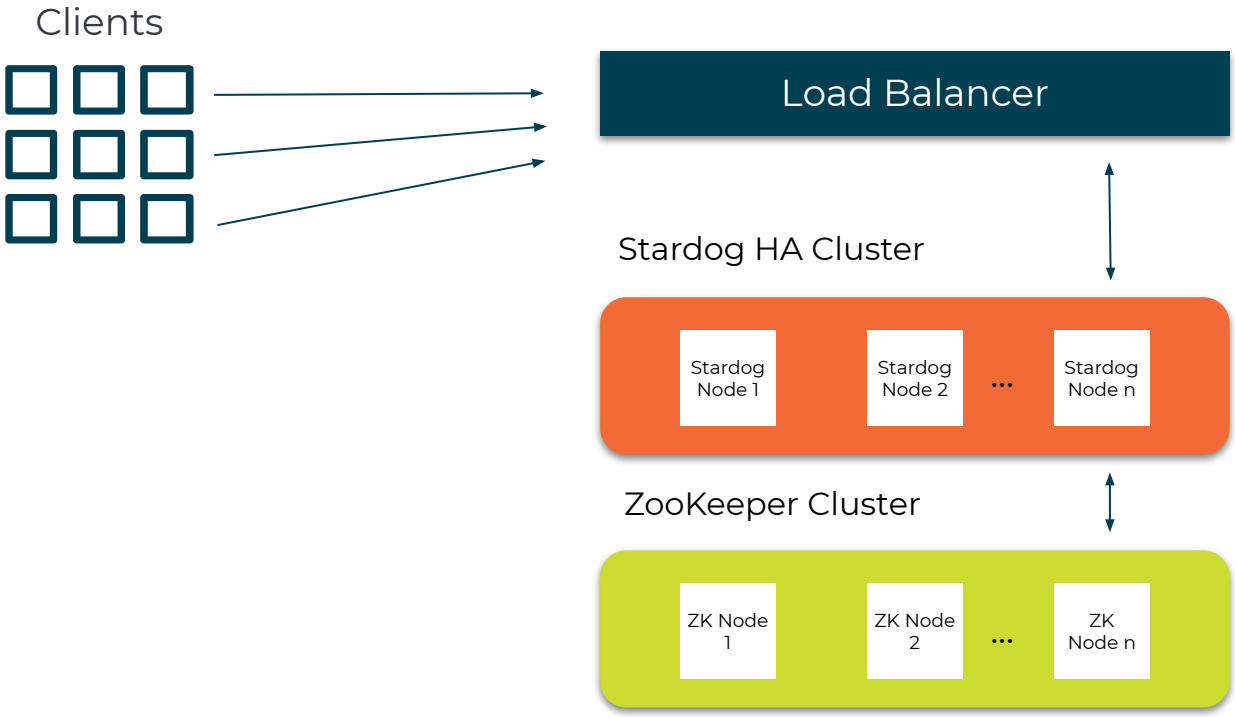
Stardog Cluster

Stardog Cluster

- Multiple Stardog servers behaving like one
  - Indistinguishable from a single Stardog instance for the client
- Provides High Availability (HA)
  - Data replication across all nodes in the cluster

Architecture

Components


Clients
Load Balancer
Stardog HA Cluster
Stardog Node 1
Stardog Node 2
...
Stardog Node n
ZooKeeper Cluster
ZK Node 1
ZK Node 2
...
ZK Node n

Components

ZooKeeper

Nodes

Load Balancer

- At least 3 ZooKeeper (ZK) nodes working together as an ensemble
- Provides centralized configuration information and distributed synchronization
- Manages cluster locks and keeps track of transaction IDs and node participation

Components

ZooKeeper

Nodes

Load Balancer

Standard nodes

- Coordinator - orchestrates transactions and maintains consistency by expelling any nodes that fail an operation
- Participants

Components

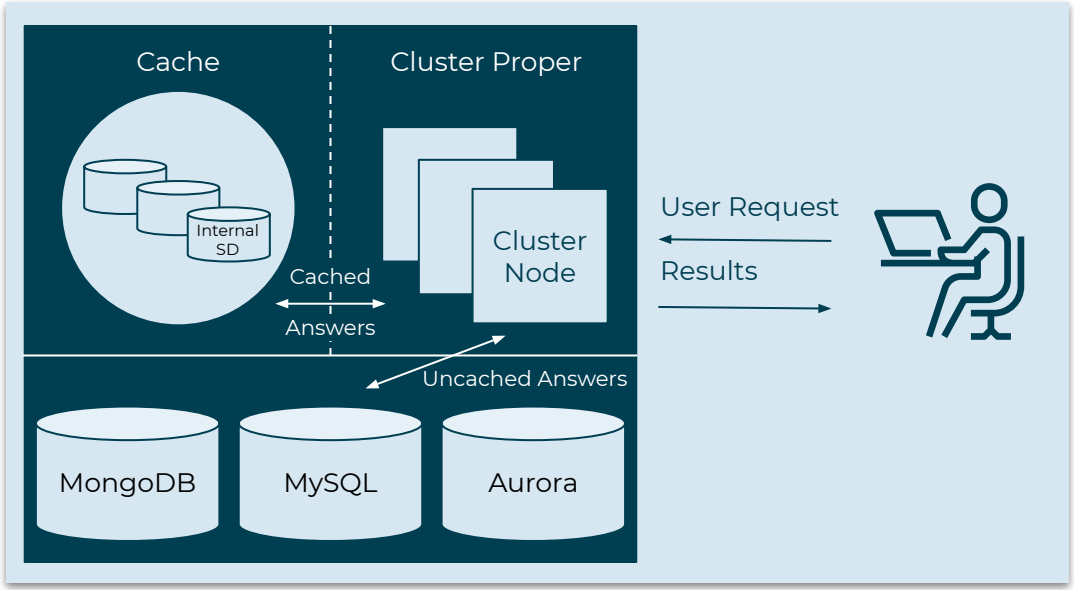
ZooKeeper

Nodes

Load Balancer

Cache nodes

- Reduce Load on Upstream Database Servers
- Read Scale-out for Data Source Access
- Partial Materialization of Slowly Changing Data



Components

ZooKeeper

Nodes

Load Balancer

Standby nodes

- Safely run database and server backups without taking CPU cycles from servicing user requests
- Closely synced to cluster
- Can upgrade to a full node in the event that a cluster node needs to be replaced or the cluster expanded

Components

ZooKeeper

Nodes

Load Balancer

- Any Load Balancer (LB) can be used
- Liveness checks
  - Node is working (joining cluster, etc.)
- Health checks
  - Node is full participating member and ready for traffic
- (Optional) provide route for coordinator
  - Transactions get passed to the coordinator directly where appropriate

Cluster Details

- A cluster guarantees that all nodes are consistent
- Nodes with failed operations are expelled from the cluster
- An expelled node must synchronize with the cluster before it can rejoin
- The Coordinator node is responsible for maintaining consistency
- Any node can handle a client request
    - If a request needs to go through the coordinator (e.g., admin operations) it is forwarded to the current coordinator

Cluster Details

Coordination

Locks

Joining

Synchronization

Transactions

- One node serves as the Coordinator
  - Orchestrates transactions
  - Maintains consistency
  - If it fails, a new Coordinator is elected
- Other nodes serve as Participants

Cluster Details

Coordination

Locks

Joining

Synchronization

Transactions

- Admin
  - Acquired before every admin operation
- Transaction (Read)
  - Read lock acquired before every transaction
- Cluster Join
  - Transaction write lock and an Admin lock

Cluster Details

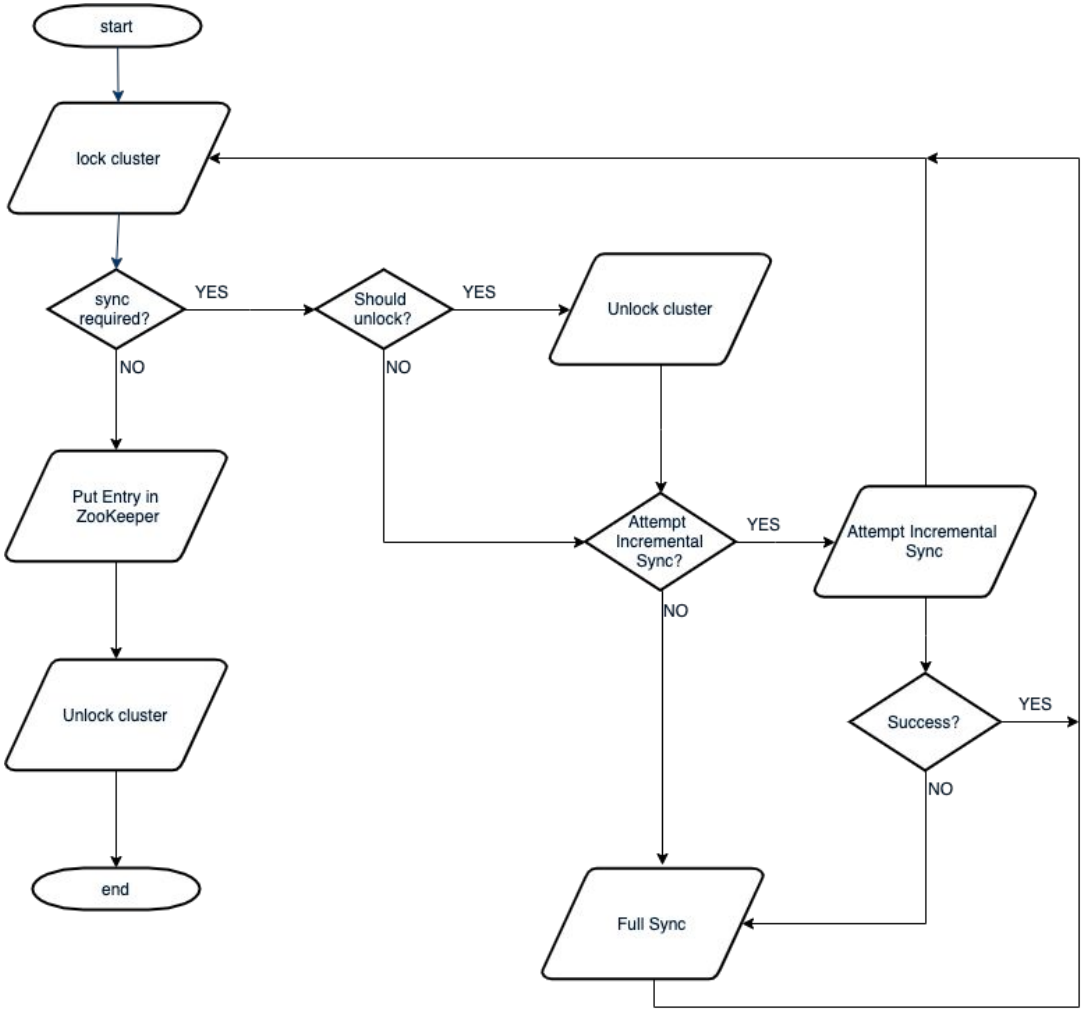
Coordination

Locks

Joining

Synchronization

Transactions



Cluster Details

Coordination

Locks

Joining

Synchronization

Transactions

- If there are constant writes a joining node must either
  - Wait until the updates subside
  - Obtain a lock that temporarily blocks the writes until the node is synchronized and can join
- Nodes trying to join a cluster will attempt to sync their data before they obtain a lock
  - If writes occur too often the joining node may never catch up
  - If a node fails to join after several attempts, it will forcibly obtain the lock and sync.
    - This blocks writes until the operation is complete

Cluster Details

Coordination

Locks

Joining

Synchronization

Transactions

Action	Example Triggers
DROP	Triggers if local DB on a node does not exist in ZK
SYNC_FULL	Triggers if DB seen in ZK does not exist locally (sends backup of complete DB)
SYNC	Triggers if local transaction does not match ZK transaction (sends transaction log contents after specified transaction)
NOOP	No operation is performed if the previous criteria does not trigger

Note: This is not a comprehensive list as actions can be performed in several different situations

Cluster Details

Coordination

Locks

Joining

Synchronization

Transactions

Action	Description
Begin*	• Acquire transaction read lock • Begin replication
Data Add/Remove	• Replicate to all nodes • Record failures
Prepare	• Expel any node that fails replication • Update RDF index and check ICV • Send prepare request to every node*
Commit*	• Send commit request to every node • Update transaction information in ZK

**Only on Coordinator*

Implementation

Sizing, Performance, & Best Practices

- ZK fault tolerance works best with an odd numbered ensemble of at least 3 (i.e. 3, 5, 7...etc.)
- Larger Stardog clusters
  - More performant with Reads
  - Less performant with Writes
- Tuning: https://www.stardog.com/blog/tuning-cluster-for-cloud/

Sizing, Performance, & Best Practices

Deployment Optimization Example:

- Load once, read many
  - Each Stardog node in the cluster can mount a volume created from the snapshot, bulk load the data at startup, and since any node can independently respond to a read request the load balancer can distribute requests round-robin
  - Joining nodes aren't blocked by read requests

Sizing, Performance, & Best Practices

Deployment Optimization Example:

- Frequent writes, followed by periods of quiescence
  - *Data is written to Stardog throughout the day in frequent transactions but not at night*
  - If it's important to your use case that a joining node not block writes
  - Configure Stardog to never forcibly obtain the join lock
  - If you deploy a three-node cluster but it's too risky to operate your production cluster with only two nodes for HA, then it may make sense to deploy a larger cluster so you can afford to lose more nodes during write-heavy times and wait for nodes to rejoin once writes subside

Sizing, Performance, & Best Practices

Deployment Optimization Example:

- Continuous small writes
  - Cluster rarely experiences quiet time with respect to writes and you want nodes to rejoin as quickly as possible
  - Can configure a joining node to obtain the lock on the second attempt
  - In this case the joining node will block the writes; but, since the node will sync without the lock on the first attempt, it will be able to mostly catch up to the other nodes in the cluster
  - On the second attempt it will forcibly obtain the lock and sync any transactions it missed in that short window and join, only blocking writes for a short time

Demo

Demo Setup

Pre-requisites:

- Docker Desktop (with hardware virtualization turned on)
- Existing DB to create a Virtual Graph (VG)

What we will create:

- ZK quorum
- Stardog cluster
- LB for the cluster
- VG of a DB
- Cache of the VG

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

- There will be 3 ZK containers
- Using the default Docker Hub image of ZK* and add in ENV variables
  - ZOO_MY_ID
    - The server's ID
  - ZOO_SERVERS
    - A list of all the servers
- ZK documentation
  - https://zookeeper.apache.org/doc/current/zookeeperStarted.html

* https://hub.docker.com/_/zookeeper
Note: Currently (Feb 2021) ZK 3.4 is supported and ZK 3.5 is in preview mode

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

Docker Compose Example:

```
version: '3.8'
services:
   zoo1:
       image: zookeeper:${ZK_VERSION_TAG}
       hostname: zoo1
       container_name: zkn1
       environment:
           ZOO_MY_ID: 1
           ZOO_SERVERS: server.1=0.0.0.0:2888:3888;2181
server.2=zoo2:2888:3888;2181 server.3=zoo3:2888:3888;2181
       ports:
           - "${HOST_MAP_ZKN1}:8080" # http:/localhost:8080/commands
       networks:
           - backend
   zoo2:
       ...
```

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

- There will be 3 Stardog containers
- Using the default Docker Hub image of Stardog* (you can use a custom image instead)
- A license and properties file must be passed in on build

* https://hub.docker.com/r/stardog/stardog

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

Docker Compose Example:

```
version: '3.8'
services:
   stardog1:
       container_name: sdn1
       hostname: stardog1
       depends_on:
           - zoo1
           - zoo2
           - zoo3
       environment:
           PATH: $PATH:/opt/stardog/bin
           STARDOG_EXT: /var/opt/stardog/ext
       build:
           context: ./stardog
           args:
               - TAG=${STARDOG_VERSION_TAG}
               - LICENSE=${STARDOG_LIC}
               - NODE_TYPE=node
       ports:
           - "${HOST_MAP_SDN1}:5820"
       networks:
           - backend
   stardog2:
       ...
```

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

Dockerfile Example:

```
ARG TAG

FROM stardog:$TAG

ARG LICENSE
ARG NODE_TYPE

COPY license/$LICENSE /var/opt/stardog/stardog-license-key.bin
COPY stardog.$NODE_TYPE.properties /var/opt/stardog/stardog.properties
COPY log4j2.xml /var/opt/stardog/log4j2.xml
RUN mkdir -p /var/opt/drivers/
COPY ./postgresql-42.2.5.jar /var/opt/drivers/

RUN mkdir /var/opt/stardog/ext
COPY ext /var/opt/stardog/ext
```

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

Stardog Properties Example:

```
# Flag to enable the cluster, without this flag set, the rest of the
properties have no effect
pack.enabled=true
# this node's IP address (or hostname) where other Stardog nodes are going
to connect
# this value is optional but if provided it should be unique for each
Stardog node
#pack.node.address=196.69.68.4
# the connection string for ZooKeeper where cluster state is stored
pack.zookeeper.address=zoo1:2181,zoo2:2181,zoo3:2181

# would need to change for production
pack.zookeeper.auth=admin:admin
```

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

- There will be a LB container
- Using the default Docker Hub image of HAProxy* (you can use a custom container instead)
- A configuration file must be passed in on build

* https://hub.docker.com/_/haproxy

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

Docker Compose Example:

```
version: '3.8'
services:
   stardog:
       image: haproxy:${HAPROXY_VERSION_TAG}
       container_name: sdlb
       depends_on:
           - stardog1
           - stardog2
           - stardog3
       build:
           context: ./haproxy
       ports:
           - "${HOST_MAP_SDLB}:5820"
       networks:
           - frontend # This will service requests for the Stardog cluster
           - backend
    ...
```

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

Dockerfile Example:

```
FROM haproxy:2.3
COPY haproxy.cfg /usr/local/etc/haproxy/
```

Note: You could also simply pass in the file via a volume mount instead of using a Dockerfile

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

HAProxy Configuration Example:

- https://docs.stardog.com/cluster/installation-and-setup/#3-start-haproxy-or-equivalent
- This demo build will have the following differences:

```
...
backend all_stardogs
   ...
   balance source # Maintain client connections
   …
   # Backend and All Stardog server sections
   server stardog1 stardog1:5820 maxconn 64 check
   server stardog2 stardog2:5820 maxconn 64 check
   server stardog3 stardog3:5820 maxconn 64 check
```

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

- Create a VG with your DB
  - https://www.stardog.com/tutorials/using-virtual-graphs/

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

- There will be 1 Stardog container
- This can be same as the Stardog Setup, but properties file can be blank

* https://hub.docker.com/r/stardog/stardog

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

Docker Compose Example:

```
version: '3.8'
services:
   cache1:
       container_name: sdc1
       hostname: cache1
       depends_on:
           - stardog
       environment:
           PATH: $PATH:/opt/stardog/bin
           STARDOG_EXT: /var/opt/stardog/ext
       build:
           context: ./stardog
           args:
               - TAG=${STARDOG_VERSION_TAG}
               - LICENSE=${STARDOG_LIC}
               - NODE_TYPE=cache
       ports:
           - "${HOST_MAP_SDC1}:5820"
       networks:
           - backend
    ...
```

Demo Build

ZK Setup

Stardog Setup

LB Setup

VG Creation

Cache Creation

Cluster Commands:

```
# Create Cache Target
stardog-admin cache target add mycache cache1:5820 admin admin

# Create Cache
stardog-admin cache create cache://mycache --graph virtual://myvg --target
cachea
```

DEMO

SWITCH TO LIVE DEMO - time stamp 3:28

SWITCH BACK TO MONITORING SLIDE (41) - time stamp 14:25

Monitoring

- Cluster Info
  - `stardog-admin cluster info`
- Cluster Status
  - `stardog-admin cluster status`
- Cluster Metrics
  - `stardog-admin cluster metrics`

DEMO

SWITCH TO LIVE DEMO - time stamp 14:49

SWITCH BACK TO BACKUP AND RESTORE SLIDE (43) - time stamp 17:10 - STAYS ON SLIDE DECK AFTER THIS

Backup & Restore

- Backup the cluster
  - `stardog-admin [server/db] backup`
    - Will run on each host
- Restore the cluster
  - `stardog-admin db restore`
    - Will replicate to each host
  - `stardog-admin server restore`
    - It is recommended that a fresh ZK ensemble deployment

Steps

1. Shutdown Stardog on all nodes in the cluster
2. Shutdown the ZooKeeper ensemble, if possible. If that's not possible we recommend backing up ZooKeeper's state and wiping the contents stored by Stardog.
3. Create an empty $STARDOG_HOME directory on all of the Stardog Cluster nodes.
4. Export $STARDOG_HOME to the empty home and run server restore (the same as you would for a single node) on a single node.
5. Start a fresh ZooKeeper ensemble with an empty data directory.
6. Start ONLY the Stardog node where you performed server restore. Verify the node starts and is in the cluster with the cluster info command before continuing to step 7.
7. Start a second node in the cluster with its empty home directory, wait for it to sync and join the cluster, as reported by cluster info. Wait until the node joins before moving to step 8.
8. Repeat step 7, one node at a time, for the remaining cluster nodes.

Upgrading

1. Confirm coordinator
   - `stardog-admin cluster info`
2. Ensure no transactions are running
   - `stardog-admin db status <db name>`
3. Shutdown the cluster
   - `stardog-admin cluster stop`
4. Backup `STARDOG_HOME`
5. Switch to the new version and bring up the nodes

Learning Objectives

Learning Objectives

- Become familiar with Stardog cluster usage
- Understand node purposes
- Build cluster with ZooKeeper and a load balancer
- Set up a cache node to cache a VG

Thank you