

# Master Thesis Defence: High Availability in Lifecycle Management of Cloud-Native Network Functions

A Near-Zero Downtime Database Version Change Prototype

Ziheng Zhang <sup>1,2</sup> Examiner: Mihhail Matskin <sup>1</sup>

Supervisor: Aleksandar Igic <sup>2</sup> Natalino Romio <sup>2</sup> Amirhossein Layegh Kheirabadi <sup>1</sup>

<sup>1</sup>KTH Royal Institue of Technology, <sup>2</sup>Ericsson AB

### **Overview**





- 1. Backgound
- 2. Deployment Strategy
- 3. Detailed Implementation
- 4. Demonstration
- 5. Conclusion
- 6. Future Work





#### **Our Goal:**

- Perform database version change
- Minimise the downtime
- Automate the procedure to reduce manual intervention

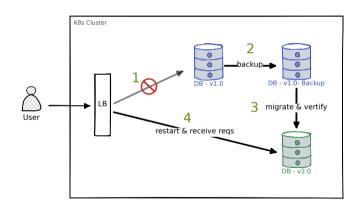








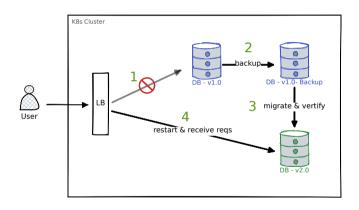
- Suspend the service
- Suspend all user requests
- Backup and migrate
- Verify data consistency
- Restart the service
- Receive requests







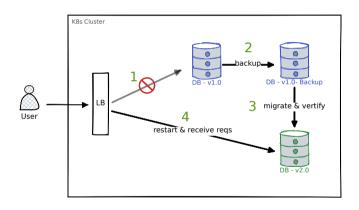
- Suspend the service
- Suspend all user requests
- Backup and migrate
- Verify data consistency
- Restart the service
- Receive requests







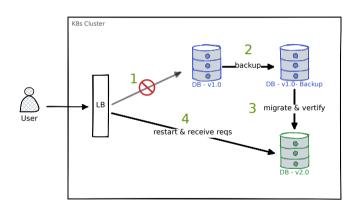
- Suspend the service
- Suspend all user requests
- Backup and migrate
- Verify data consistency
- Restart the service
- Receive requests







- Suspend the service
- Suspend all user requests
- Backup and migrate
- Verify data consistency
- Restart the service
- Receive requests







#### **Limitations of Conventional Strategy:**

- Require downtime
- Significant downtime affects high availability
- **Five-nine goal** (99.999% high availability)
- Too many human interventions







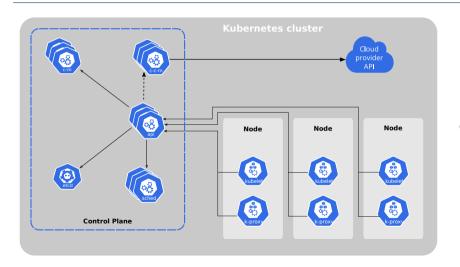
#### Kubernetes (K8s) & K8s Operator:

- Extends Kubernetes API for application-specific management.
- Combines custom resources (desired state) and custom controllers (logic).
- Automates deployment, scaling, and management of stateful applications.
- Encapsulates operational knowledge and best practices.
- Reduces manual intervention and enhances reliability.

More details to the following two slides.

































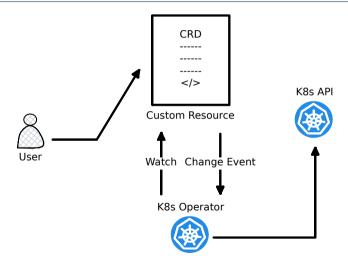










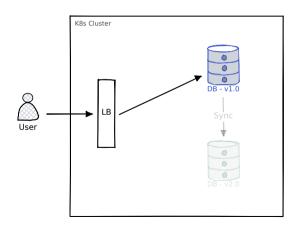


### **Blue-Green Deployment (1/4)**





- The initial state of the cluster. Only a *Master* node.

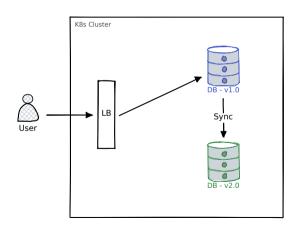


### **Blue-Green Deployment (2/4)**





- Create the DB with the new version (the Follower) and start the sync.

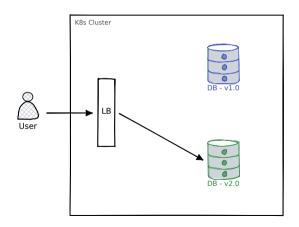


### **Blue-Green Deployment (3/4)**





- Change the data flow from the *Master* to the *Follower*.

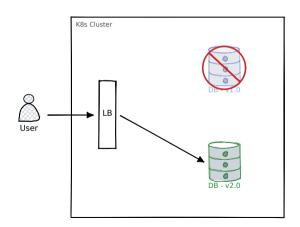


### **Blue-Green Deployment (4/4)**





- Delete the master DB (Optional).

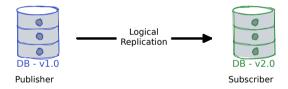


## **How To Achieve Synchronization? (1/3)**





- Logical Replication.



### **How To Achieve Synchronization? (2/3)**





#### In the *Master* node,

- Configure the PostgreSQL settings
  - \$ set wal\_level=logical in postgresql.conf
- Create a publication
  - \$ create publication [pub\_name] for all tables
- Export the schema.
  - \$ pg\_dump -U [usr\_name] -t [table\_name] [db\_name] > [file.sql]

### **How To Achieve Synchronization? (2/3)**





#### In the Master node,

- Configure the PostgreSQL settings
  - \$ set wal\_level=logical in postgresql.conf
- Create a publication
  - \$ create publication [pub\_name] for all tables
- Export the schema
  - \$ pg\_dump -U [usr\_name] -t [table\_name] [db\_name] > [file.sql]

### **How To Achieve Synchronization? (2/3)**





#### In the *Master* node.

- Configure the PostgreSQL settings
  - \$ set wal\_level=logical in postgresql.conf
- Create a publication
  - \$ create publication [pub\_name] for all tables
- Export the schema.
  - \$ pg\_dump -U [usr\_name] -t [table\_name] [db\_name] > [file.sql]

### **How To Achieve Synchronization? (3/3)**





#### In the Follower node.

- Sync the schema in the *Master* node.

```
$ psql -U [usr_name] -h [Master_IP] -p [port] \
  -d [db_name] -f [file.sql]
```

- Create the subscription
  - \$ create subscription [sub\_name] connection \
     'dbname=[db\_name] host=[Master\_IP] user=[usr\_name] \
     password=[pwd] port=[port]' publication [pub\_name]

## **How To Achieve Synchronization? (3/3)**





#### In the Follower node.

- Sync the schema in the *Master* node.
  - \$ psql -U [usr\_name] -h [Master\_IP] -p [port] \
     -d [db\_name] -f [file.sql]
- Create the subscription.
  - \$ create subscription [sub\_name] connection \
     'dbname=[db\_name] host=[Master\_IP] user=[usr\_name] \
     password=[pwd] port=[port] ' publication [pub\_name]

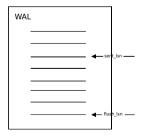
### When will Synchronization end? (1/2)





#### In PostgreSQL, pg\_stat\_replication view provides the replication info.

- sent lsn
  - represents the latest WAL position sent by the Master to the Follower
- pg\_current\_wal\_flush\_lsn
  - returns the latest WAL position that has been flushed to disk on the Master node.
- pg\_wal\_lsn\_diff(lsn1, lsn2
  - returns the byte difference between two WAL positions.



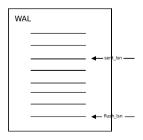
### When will Synchronization end? (1/2)





In PostgreSQL, pg\_stat\_replication view provides the replication info.

- sent\_lsn
  - represents the latest WAL position sent by the Master to the Follower.
- pg\_current\_wal\_flush\_lsn
  - returns the latest WAL position that has been flushed to disk on the Master node
- pg\_wal\_lsn\_diff(lsn1, lsn2
  - returns the byte difference between two WAL positions



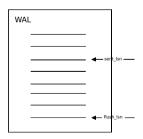
### When will Synchronization end? (1/2)





In PostgreSQL, pg\_stat\_replication view provides the replication info.

- sent\_lsn
   represents the latest WAL position sent by the Master to the Follower.
- pg\_current\_wal\_flush\_lsn
   returns the latest WAL position that has been flushed to disk on the Master node.
- pg\_wal\_lsn\_diff(lsn1, lsn2)
   returns the byte difference between two WAL positions.

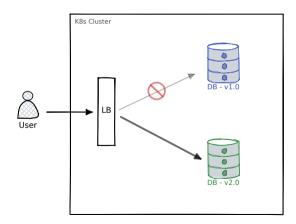


### When will Synchronization end? (2/2)





When Synchronization is finished, change the dataflow from the *Master* to the *Follower*.







#### What is Custom Resource Definition (CRD)?

- Extends the Kubernetes API with custom resources.
- Allows for application-specific customization.
- Simplifies the integration of custom logic.
- Enables creation, modification, and management of custom objects.
- Eliminates the need for direct changes to the core Kubernetes codebase.





### Service in K8s: apiVersion: v1 kind: Service metadata: name: postgres-master spec: selector: app: postgres-master ports: - name: postgres-master port: 5432 targetPort: 5432 type: LoadBalancer

#### CRD in K8s:

```
apiVersion: pgupgrade.zzh.domain/v1
kind: PgUpgrade
metadata
   . . .
spec:
   # mv desired state in cluster
   image: postgres:15
   dbname: mydatabase
   subname: zzhsub1
   pubname: zzhpub1
   olddbhost: "10.244.1.76"
   . . .
```



- image: postgres:15 # version of db



```
- image: postgres:15 # version of db
- dbname: mydatabase # db name that we need to sync with
- subname: zzhsub1 # subscription name
- pubname: zzhpub1 # publication name
- olddbhost: "10.244.1.76" # IP address of the Master node
- olddbport: "5432" # port of the Master node
- finishsync: false # whether the sync is finished
```





```
- image: postgres:15 # version of db
- dbname: mydatabase # db name that we need to sync with
- subname: zzhsub1 # subscription name
- pubname: zzhpub1 # publication name
- olddbhost: "10.244.1.76" # TP address of the Master node
- olddbport: "5432" # port of the Master node
- finishsync: false # whether the sync is finished
```





```
- image: postgres:15 # version of db
- dbname: mydatabase # db name that we need to sync with
- subname: zzhsub1 # subscription name
- pubname: zzhpub1 # publication name
- olddbhost: "10.244.1.76" # IP address of the Master node
- olddbport: "5432" # port of the Master node
```





```
- image: postgres:15 # version of db
- dbname: mydatabase # db name that we need to sync with
- subname: zzhsub1 # subscription name
- pubname: zzhpub1 # publication name
- olddbhost: "10.244.1.76" # IP address of the Master node
- olddbport: "5432" # port of the Master node
- finishsync: false # whether the sync is finished
```

- killdeployments: pg-delete-test # delete the master node

### **Implementation**





Detailed implementation can be found here:

- C zzheng2020/Master-Thesis
- 🗘 zzheng2020/pgoperator

### **Demonstration**





Demo Video

### Conclusion





#### In this project, we

- Successfully implemented a prototype for near-zero downtime database version upgrades.
- Requires only one database restart (if the configuration is pre-set, no restart is needed).
- Automates the process, reducing manual intervention.
- Achieves high availability for the database system in the Kubernetes cluster.

### **Future Work**





#### Future work could include:

- Integrating it into PostgreSQL Operator.
- Enhancing automation and error handling.
- Developing a user-friendly interface.
- Evaluating security and compliance.

# The End