Weathering the Cloud Storms with Multi-Region Workflows

Amit Chauhan

Agenda

- Airflow as a distributed System
- Central role of Database
- High Availability, Resilience and Horizontal Scalability
- Why we should consider using Distributed Database instead of default choices.
- Introduce YugabyteDB for resilience and scalability
- Demo: Integrating Airflow with YugabyteDB
- High level overview of distributed database internals

Architecture of Airflow

Main Components

Scheduler

Responsible for scheduling DAGs (Directed Acyclic Graphs) and placing tasks in the queue.

Executor

Manages task execution. Options include CeleryExecutor (for distributed systems) and LocalExecutor (for single-node setups).

Workers

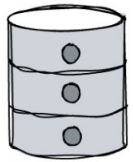
Execute the tasks in parallel.

Queue

When using CeleryExecutor, a message queue like Redis or RabbitMQ manages task queues for workers.

Webserver

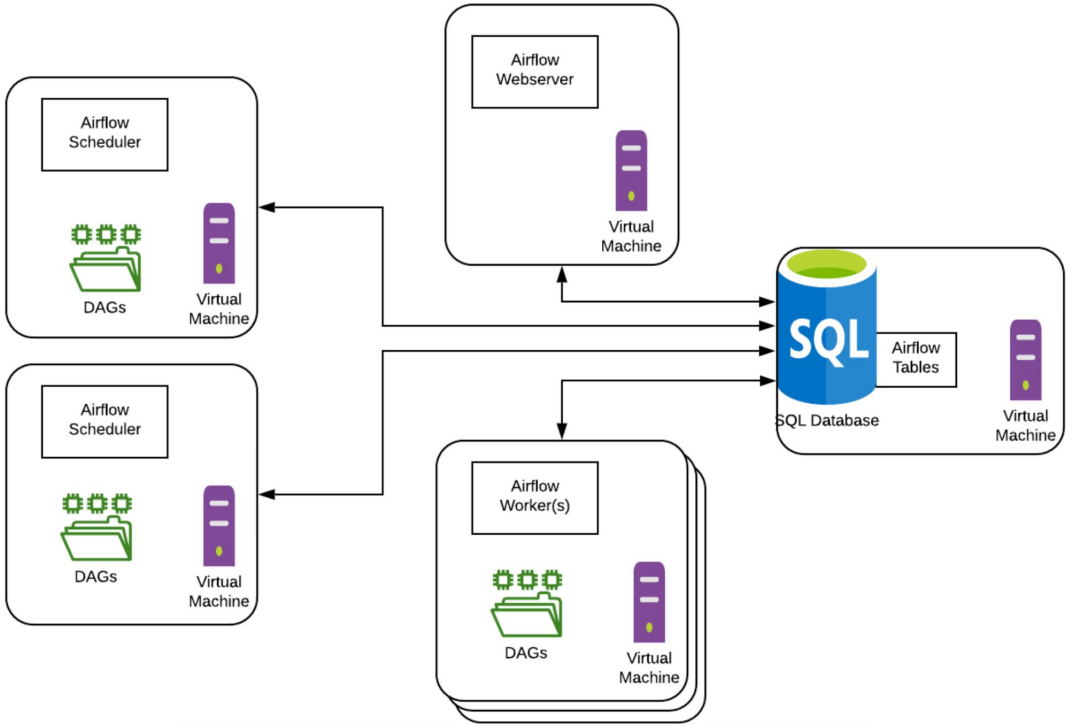
Provides the user interface for monitoring and managing DAGs.



Stores metadata about DAG runs, task instances, logs, etc. Typically PostgreSQL or MySQL.

Typical High Availability Airflow setup

- Diagram of a typical production Airflow setup
- Single or Multi-node Airflow deployment (Scheduler, Web Server, Workers)
- Metadata database (PostgreSQL)
- Object storage (for DAGs or logs)
- Multi-AZ or single region



Airflow as a Distributed System

Key Points

- ✓ **Airflow's architecture is built to support distributed execution, allowing it to scale horizontally.**

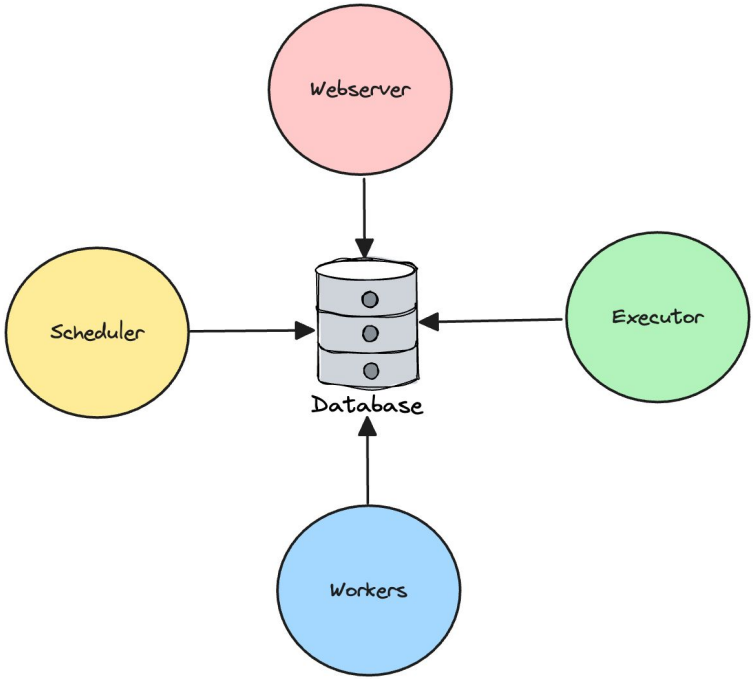
Component	Multi AZ/Region Setup
Scheduler	✓ Multiple schedulers can be configured in a high-availability setup, ensuring tasks are scheduled even in case of failure.
Workers	✓ Workers are processes that actually execute the tasks (jobs). They can be distributed across multiple machines or regions, allowing tasks to run in parallel.
Executors	✓ There are several types of executors (e.g., LocalExecutor, CeleryExecutor), with distributed executors like CeleryExecutor enabling task distribution across multiple nodes.
Web Servers	✓ You can add new nodes horizontally
Database	??? 

Metadata is stored in DB of your choice

Airflow DB choices

Central Role of the DB

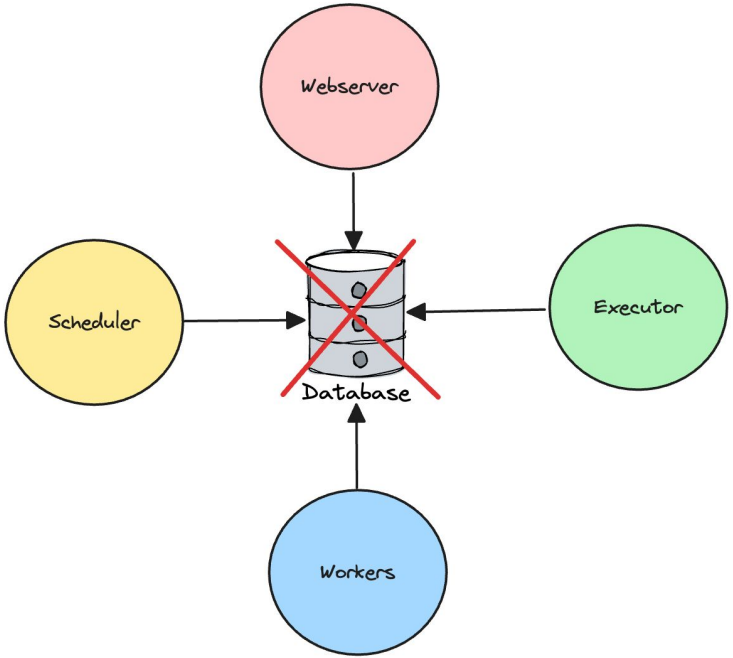
- **Single Source of Truth**: The metadata database stores critical information like:
  - DAG execution statuses (success, failure, etc.)
  - Task dependencies and their states
  - Task logs and historical execution data
- **Communication Hub**: Every core Airflow component (Scheduler, Workers, Web Server) interacts with the metadata database. This interaction is essential for:
  - Task scheduling and dependency resolution (Scheduler)
  - Access to logs and status updates (Workers, Web Server)
  - User-facing metrics and reports (Web Server)



Single Point of Failure

Vulnerability: Since the metadata database is centralized, its failure can disrupt the entire Airflow environment.

- **Scheduler Failure**
- **Worker Failure**
- **Web Server Failure**



Horizontal Scalability

These default choices (postgres, mysql) are designed to scale **vertically**—by adding more CPU, RAM, or storage to a single instance.

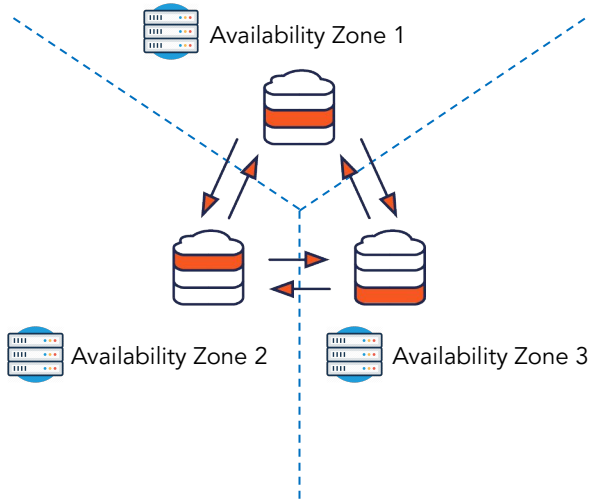
- **Not Designed for Horizontal Scaling**: Scaling PostgreSQL across multiple nodes (horizontal scaling) is challenging and often requires complex replication setups, which don’t fully mitigate the risk of bottlenecks.
- **Challenging Replication**: Replicating a traditional RDBMS across regions or zones is difficult and often leads to inconsistency, high-latency reads, and write conflicts.

High Availability and Resiliency

- Achieving high availability especially across multi region layout is operationally challenging.
- Can often result in downtime of few to several minutes even during planned maintenance.
- Replicating a traditional RDBMS across regions or zones is difficult.

What can we do about this?

- Airflow's reliance on a centralized metadata database introduces limitations in scalability, resilience, and performance.
- As workloads grow, the database becomes a bottleneck, and its single-point-of-failure nature adds significant operational risk.
- **Transitioning to a distributed database solution can mitigate these challenges, ensuring high availability and horizontal scalability.**



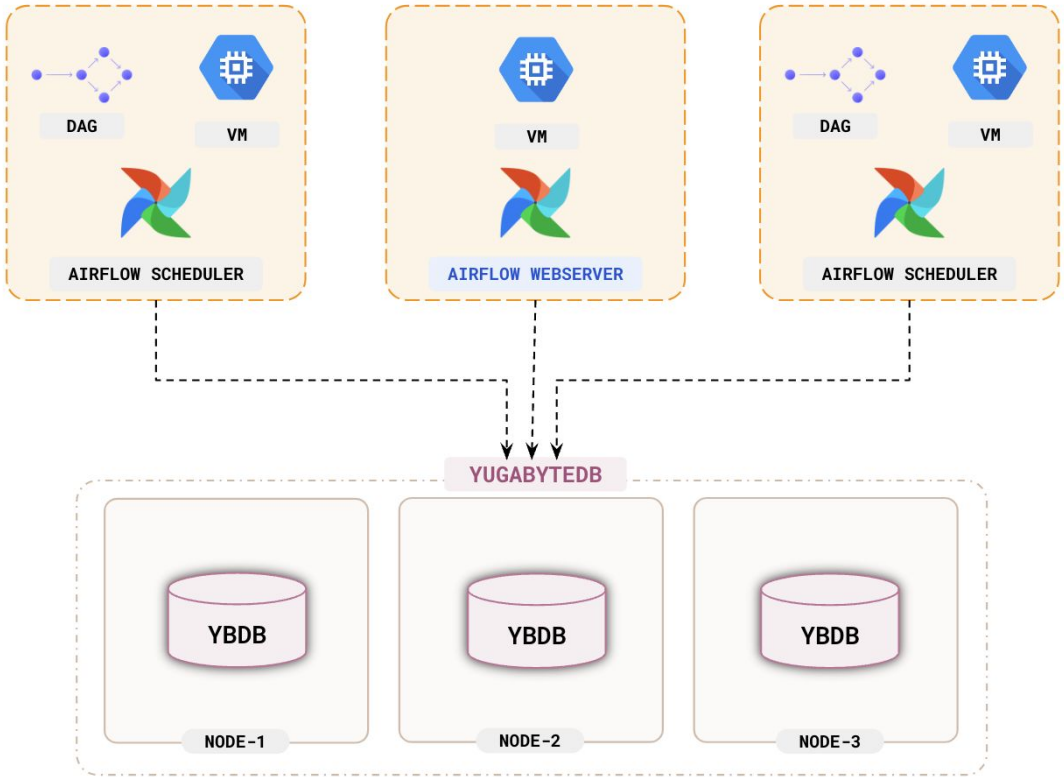
What are my choices for distributed SQL database?

- There are many choices - YugabyteDB, TiDB, Spanner, etc.
- If you are using OSS Airflow and want something cloud agnostic then **YugabyteDB (Postgres driver) is a great choice**.

Proposed Architecture with YugabyteDB

New diagram for Airflow setup using YugabyteDB:

- Multi-node, distributed database
- Multi-region/multi-AZ database nodes for high availability
- High resilience at the database layer to match Airflow distributed architecture
- Automatic failover, scalability, geographic redundancy



Cloud native relational database for cloud native applications

yugabyteDB

Distributed SQL database for transactional applications.
100% open source. Runs on any cloud.

Scale, Resilience & Developer Friendly APIs

PostgreSQL Compatibility

Resilience and High Availability

Horizontal Scalability

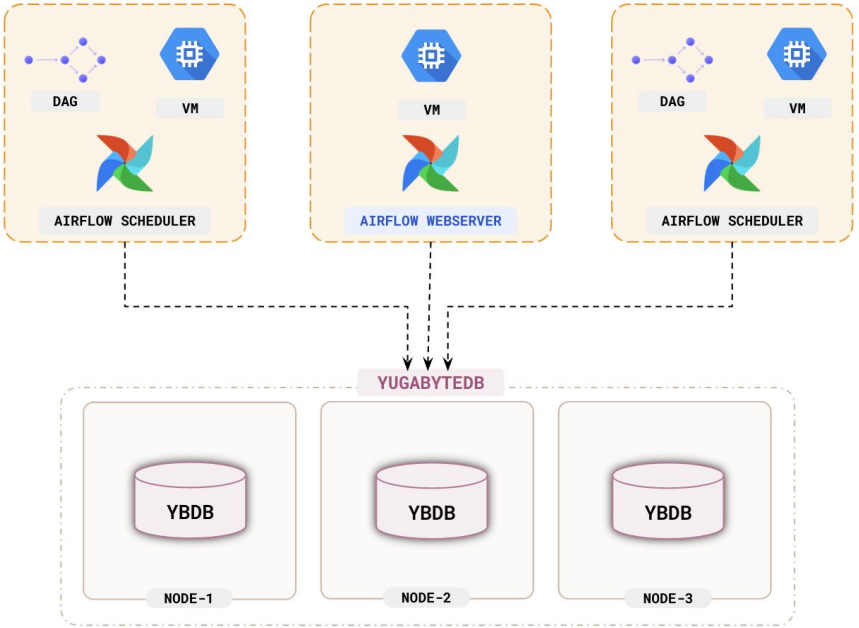
Geographic Distribution

ACID Transactions

Security

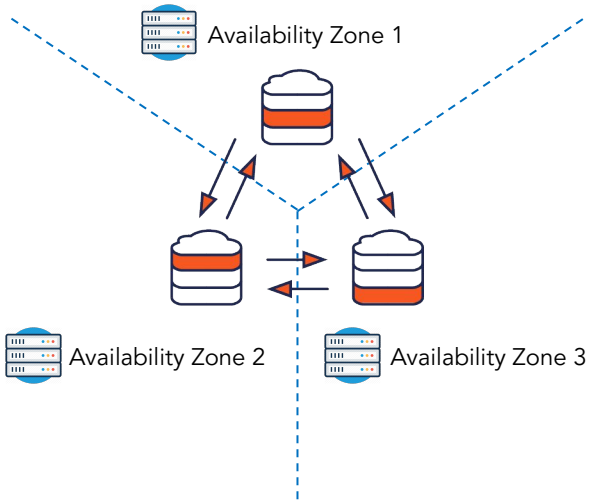
Benefits of Using a Distributed Database (YugabyteDB)

- 100% Open source (Apache 2.0)
- Plug and Play replacement for Postgres
- No single point of failure for the metadata database
- Horizontal scalability
- Fault tolerance across regions/AZs
- Performance improvements with automatic sharding and rebalancing of data



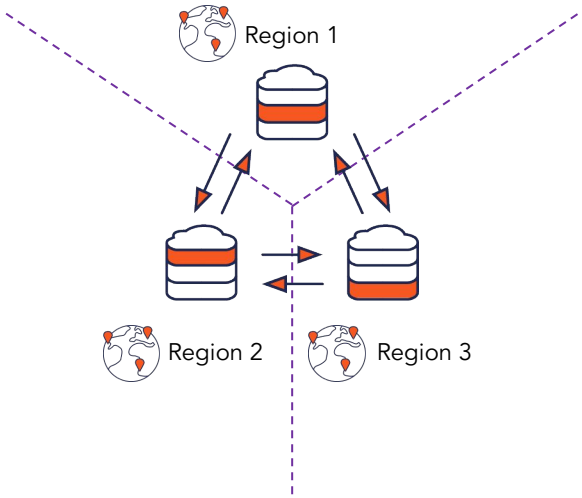
Flexible deployment options

1. Single Region, Multi-Zone



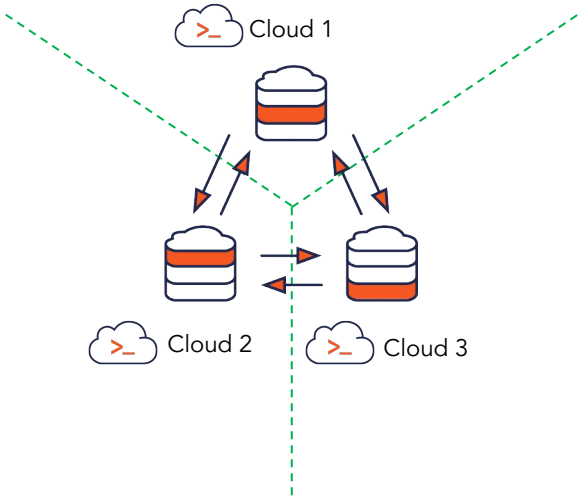
Consistent Across Zones
No WAN Latency But No
Region-Level Failover/Repair

2. Single Cloud, Multi-Region



Consistent Across Regions
with Auto Region-Level
Failover/Repair

3. Multi-Cloud, Multi-Region

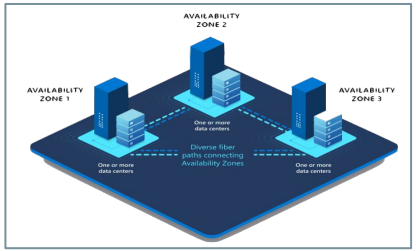


Consistent Across Clouds
with Auto Cloud-Level
Failover/Repair

Choose Your Required Level Of Resilience

Fault Zone:
Node or Rack

Latency
Read < 1ms
Write < 2ms



Fault Zone:
Availability Zone or Data Centre

Latency
Read < 1ms
Write < 3ms

Fault Zone:
Continental Regions

Latency
Read 3-10ms
Write 10-30ms

Fault Zone:
Global Regions

Latency
Read ~100ms
Write ~200ms

yugabyteDB also has advanced features for performance tuning geo-distribution including
Geo-placement of data by partition, Read Replicas, Follower Reads, Prefered Zones, xCluster Async Replication

Setting Up Airflow with YugabyteDB

All you need to do is provide YB Connectivity details in airflow.cfg. That's it!

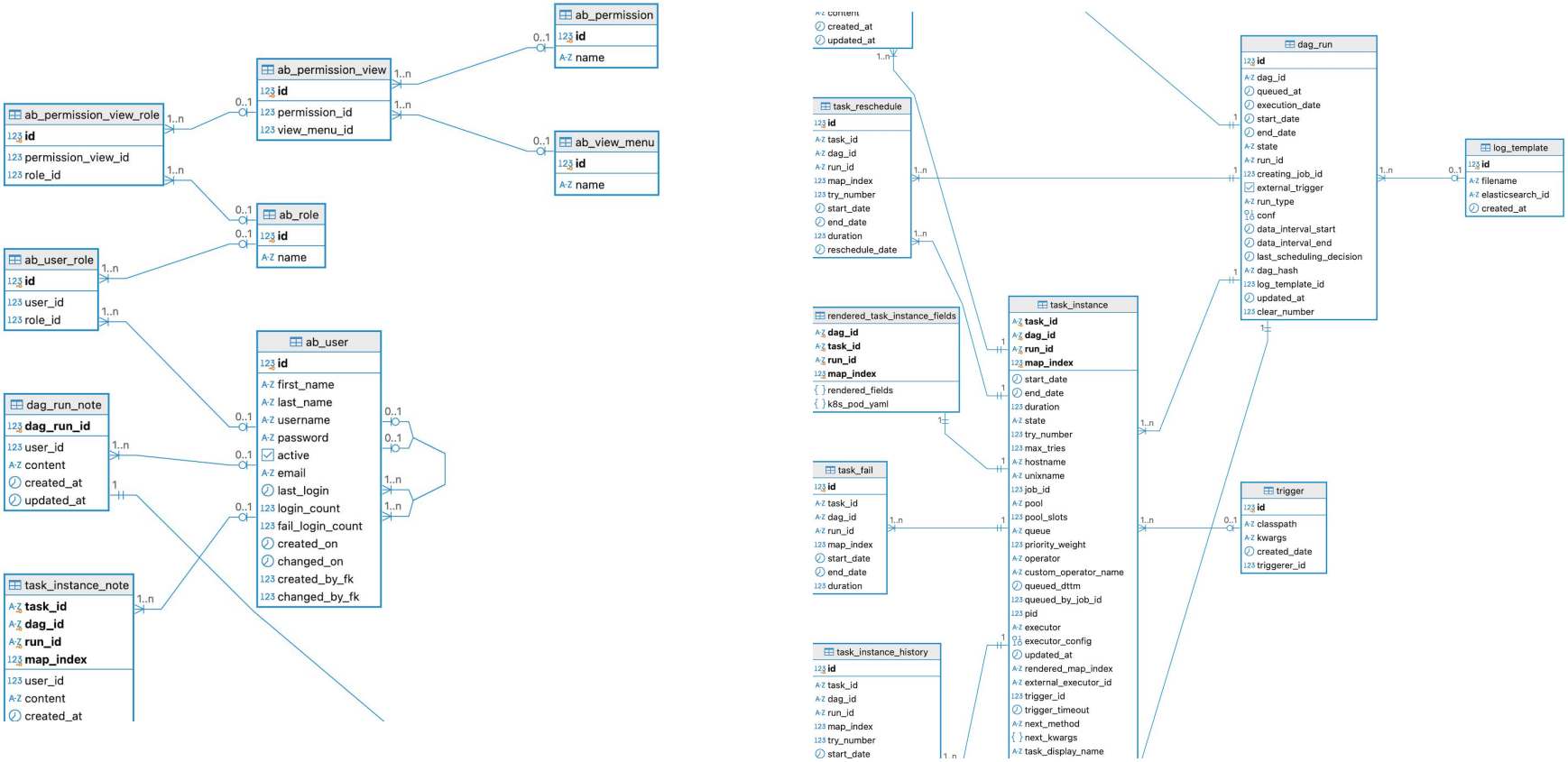
```
AIRFLOW__DATABASE__SQL_ALCHEMY_CONN: postgresql+psycopg2://yugabyte:yugabyte@<YB-DB-HOST>:5433/yugabyte

AIRFLOW__CELERY__RESULT_BACKEND: db+postgresql://yugabyte:yugabyte@<YB-DB-HOST>:5433/yugabyte
```

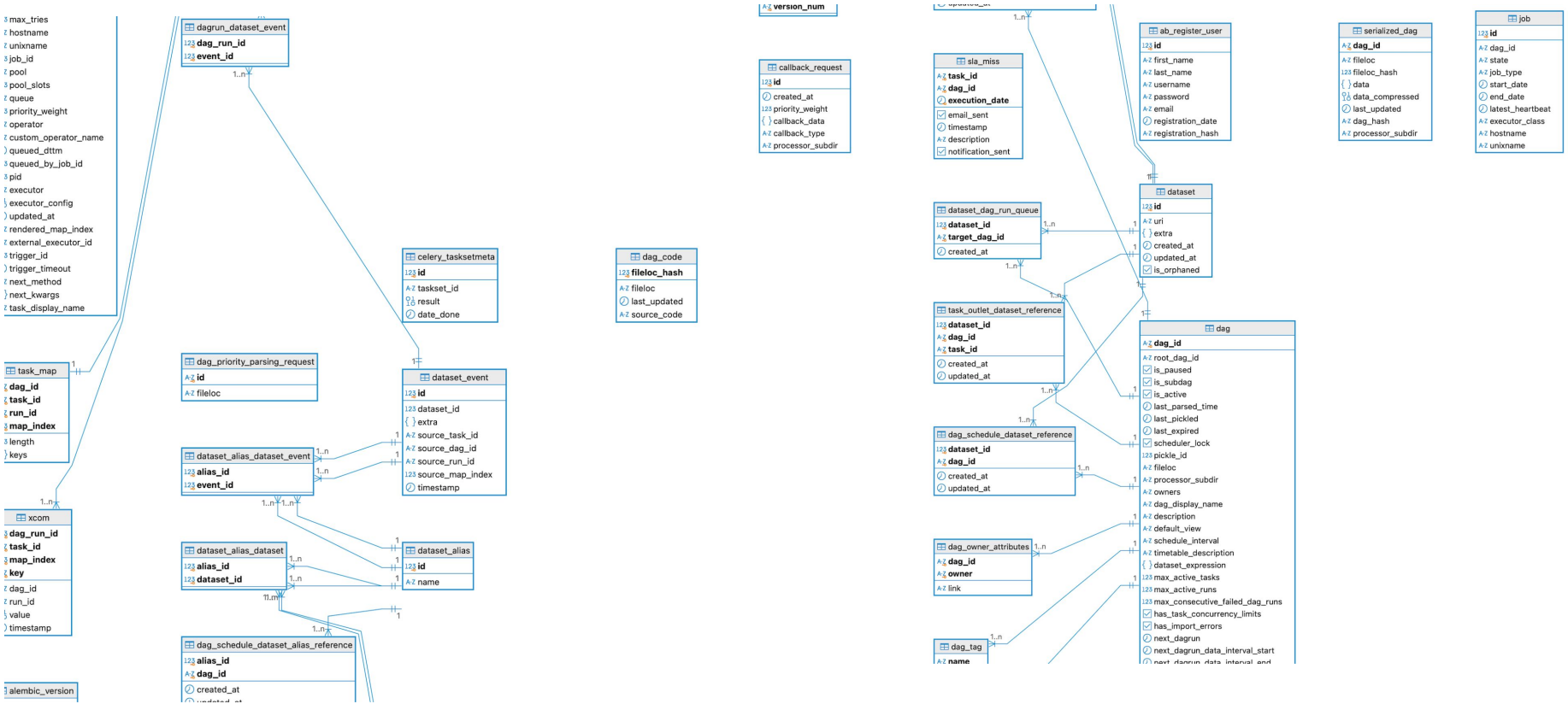
In database, You might need to do one more thing:
ALTER ROLE <DB_UserName> SET yb_silence_advisory_locks_not_supported_error=on;

This won't be needed once following Github issue is fixed: https://github.com/yugabyte/yugabyte-db/issues/3642

ER Diagram


ab_permission
id
name
ab_permission_view
id
permission_id
view_menu_id
ab_view_menu
id
name
ab_permission_view_role
id
permission_view_id
role_id
ab_role
id
name
ab_user_role
id
user_id
role_id
ab_user
id
first_name
last_name
username
password
active
email
last_login
login_count
fail_login_count
created_on
changed_on
created_by_fk
changed_by_fk
dag_run_note
dag_run_id
user_id
content
created_at
updated_at
task_instance_note
task_id
dag_id
run_id
map_index
user_id
content
created_at
created_at
updated_at
task_reschedule
id
task_id
dag_id
run_id
map_index
try_number
start_date
end_date
duration
reschedule_date
dag_run
id
dag_id
queued_at
execution_date
start_date
end_date
state
run_id
creating_job_id
external_trigger
run_type
conf
data_interval_start
data_interval_end
last_scheduling_decision
dag_hash
log_template_id
updated_at
clear_number
log_template
id
filename
elasticsearch_id
created_at
rendered_task_instance_fields
dag_id
task_id
run_id
map_index
rendered_fields
k8s_pod_yaml
task_instance
task_id
dag_id
run_id
map_index
start_date
end_date
duration
state
try_number
max_tries
hostname
unixname
job_id
pool
pool_slots
queue
priority_weight
operator
custom_operator_name
queued_dttm
queued_by_job_id
pid
executor
executor_config
updated_at
rendered_map_index
external_executor_id
trigger_id
trigger_timeout
next_method
next_kwargs
task_display_name
task_fail
id
task_id
dag_id
run_id
map_index
start_date
end_date
duration
trigger
id
classpath
kwargs
created_date
triggerer_id
task_instance_history
id
task_id
dag_id
run_id
map_index
try_number
start_date

ER Diagram


max_tries
hostname
unixname
job_id
pool
pool_slots
queue
priority_weight
operator
custom_operator_name
queued_dttm
queued_by_job_id
pid
executor
executor_config
updated_at
rendered_map_index
external_executor_id
trigger_id
trigger_timeout
next_method
next_kwargs
task_display_name
dagrun_dataset_event
dag_run_id
event_id
celery_tasksetmeta
id
taskset_id
result
date_done
dag_code
fileloc_hash
fileloc
last_updated
source_code
version_num
callback_request
id
created_at
priority_weight
callback_data
callback_type
processor_subdir
sla_miss
task_id
dag_id
execution_date
email_sent
timestamp
description
notification_sent
ab_register_user
id
first_name
last_name
username
password
email
registration_date
registration_hash
serialized_dag
dag_id
fileloc
fileloc_hash
data
data_compressed
last_updated
dag_hash
processor_subdir
job
id
dag_id
state
job_type
start_date
end_date
latest_heartbeat
executor_class
hostname
unixname
task_map
dag_id
task_id
run_id
map_index
length
keys
dag_priority_parsing_request
id
fileloc
dataset_event
id
dataset_id
extra
source_task_id
source_dag_id
source_run_id
source_map_index
timestamp
dataset_alias_dataset_event
alias_id
event_id
dataset_alias_dataset
alias_id
dataset_id
dataset_alias
id
name
xcom
dag_run_id
task_id
map_index
key
dag_id
run_id
value
timestamp
dag_schedule_dataset_alias_reference
alias_id
dag_id
created_at
alembic_version
dataset_dag_run_queue
dataset_id
target_dag_id
created_at
dataset
id
uri
extra
created_at
updated_at
is_orphaned
task_outlet_dataset_reference
dataset_id
dag_id
task_id
created_at
updated_at
dag_schedule_dataset_reference
dataset_id
dag_id
created_at
updated_at
dag_owner_attributes
dag_id
owner
link
dag_tag
name
dag
dag_id
root_dag_id
is_paused
is_subdag
is_active
last_parsed_time
last_pickled
last_expired
scheduler_lock
pickle_id
fileloc
processor_subdir
owners
dag_display_name
description
default_view
schedule_interval
timetable_description
dataset_expression
max_active_tasks
max_active_runs
max_consecutive_failed_dag_runs
has_task_concurrency_limits
has_import_errors
next_dagrun
next_dagrun_data_interval_start




Demo

Demo Setup: Airflow with YugabyteDB in Multi-AZ/Multi Region setup

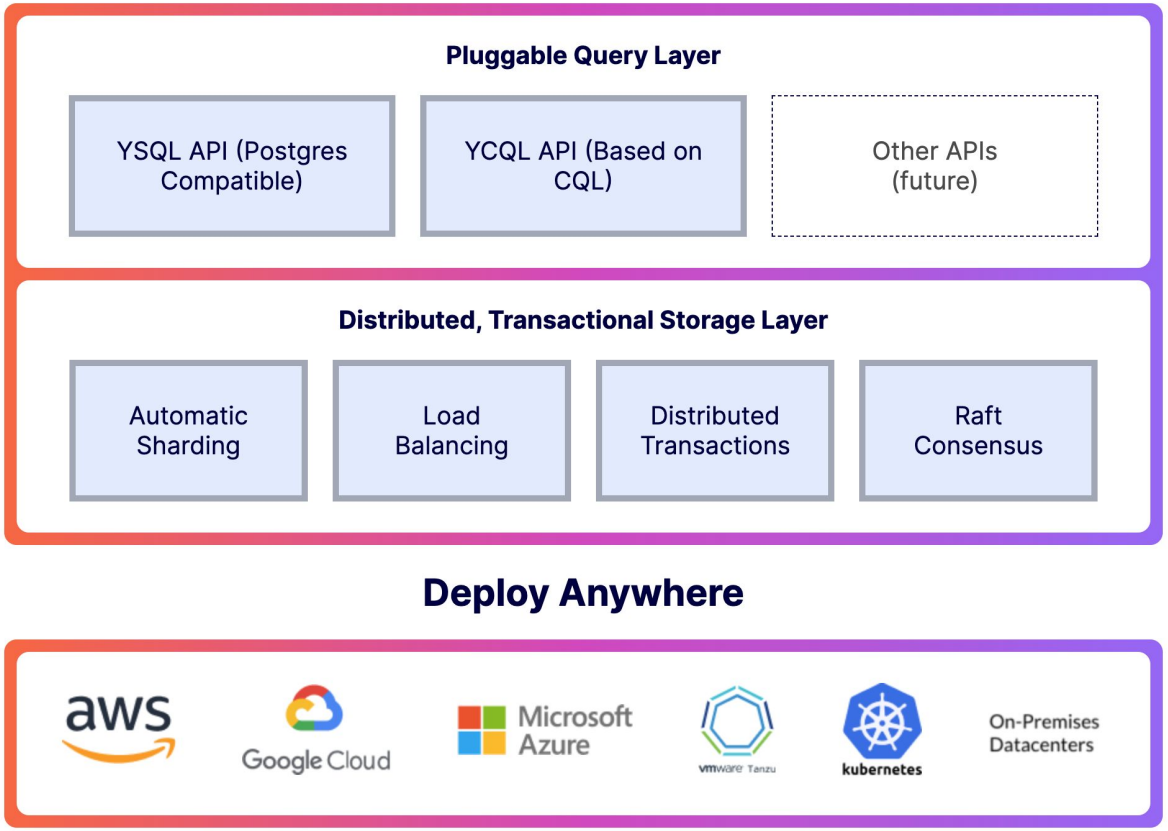
- We will setup YugabyteDB cluster (multi-node, multi-AZ or multi-region)
- Show how easy it is to Setup Airflow to use YugabyteDB cluster.
- We will bring down entire datacenter and showcase how Airflow can still remain functional.
- Discuss the High Availability during planned and unplanned outages.



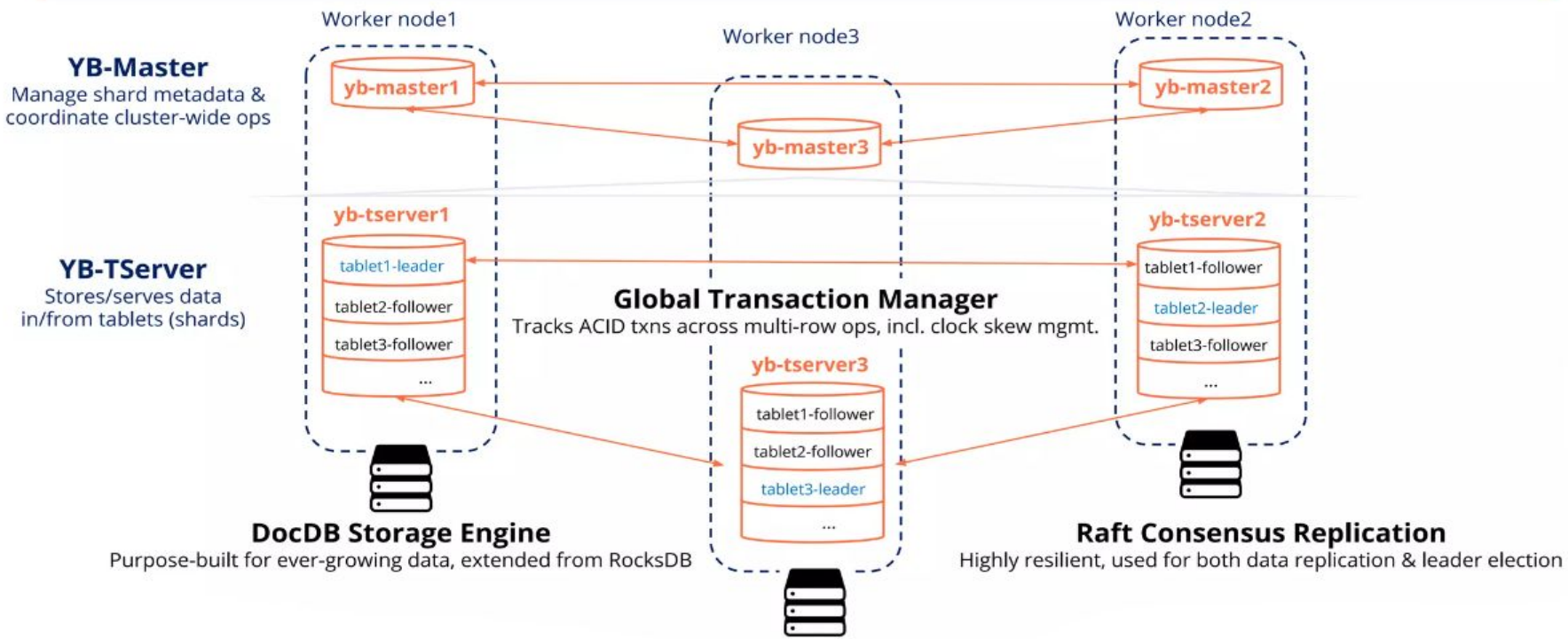

Distributed DB Internals

How they provide resilience and Horizontal Scalability

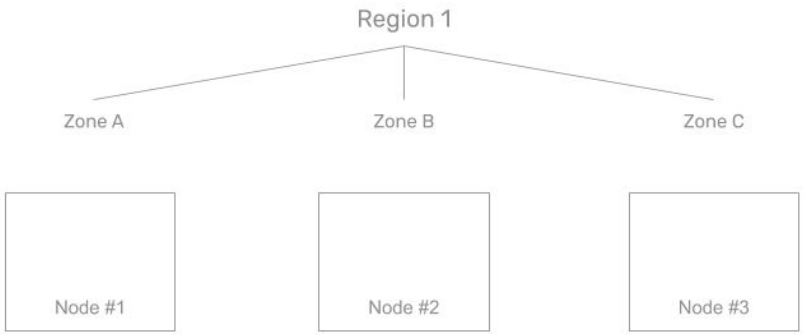
Transactional Distributed SQL Database with a Pluggable API Layer



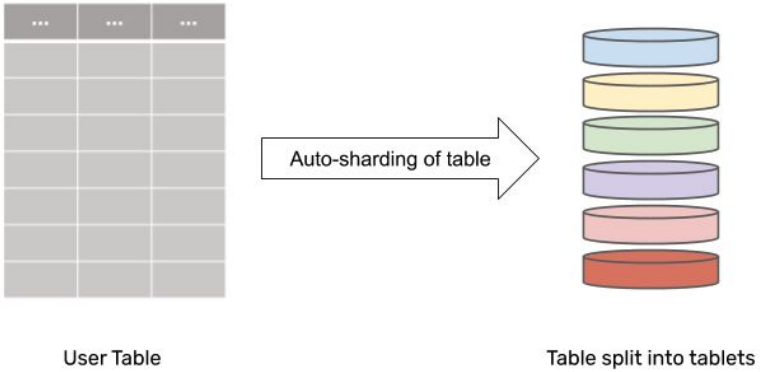
High-Level Architecture: Under the Hood of a 3-Node Cluster



Distributing Data

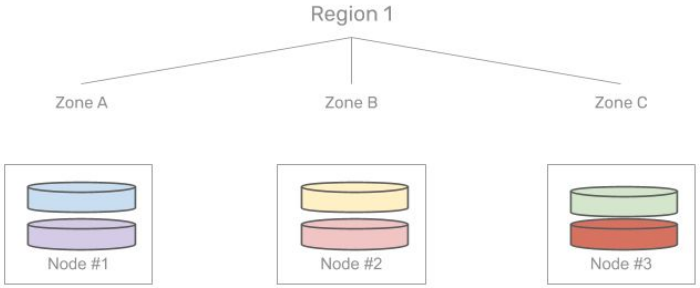


- Assume 3-nodes across zones
- How to distribute data across nodes?



- User tables sharded into tablets
- Tablet = group of rows
- Sharding is transparent to user
  - HASH, RANGE supported

Distributing Data Across Nodes, Zones, Regions

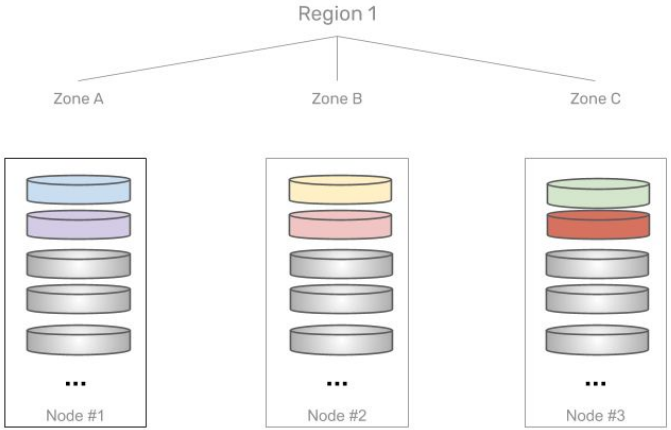


Tablets (per-table, across tables)

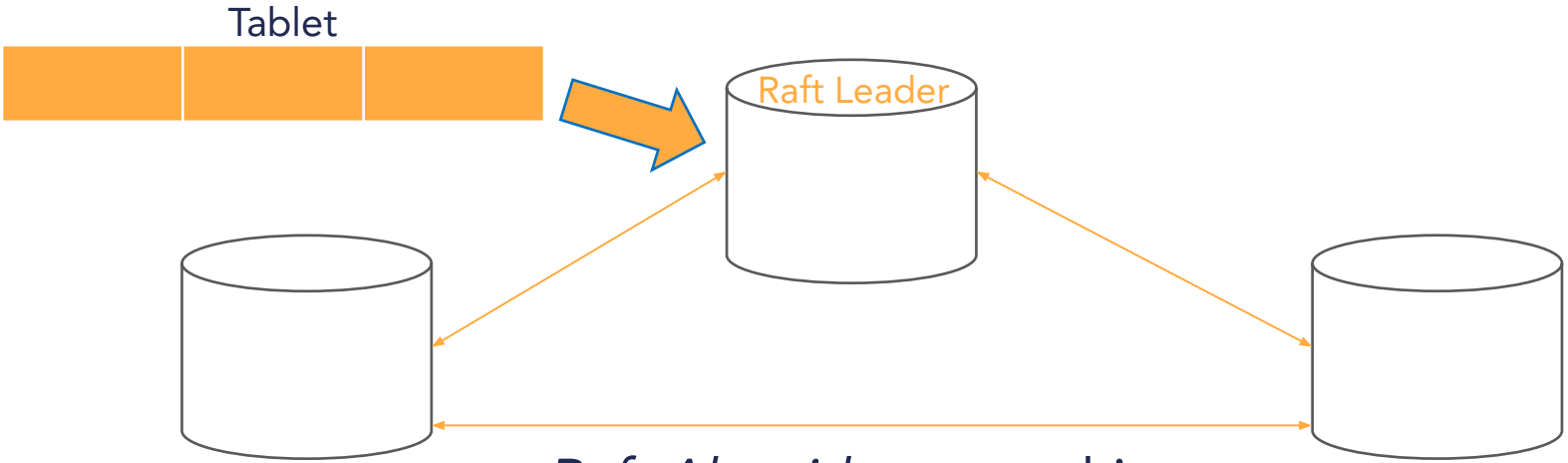
evenly distributed across nodes

In real deployments,

many tablets per node



High Availability - Replication uses Raft Consensus algorithm



Raft Algorithm can achieve
per-row consistency across nodes

On failure, HA achieved because
new leader elected quickly

Replication in a 3 node cluster

- Assume rf = 3
- Survives 1 node or zone failure
- Tablets replicated across 3 nodes
- Follower (replica) tablets balanced across nodes in cluster

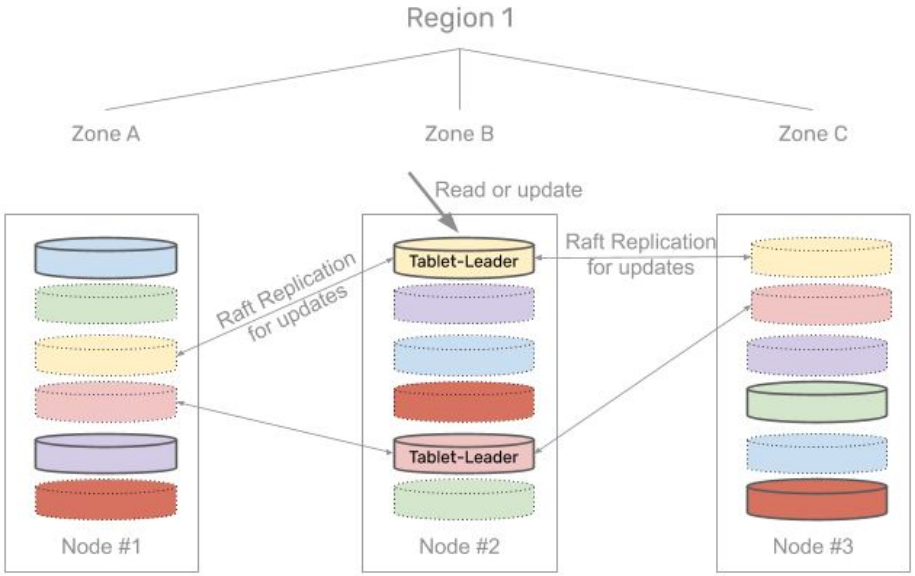
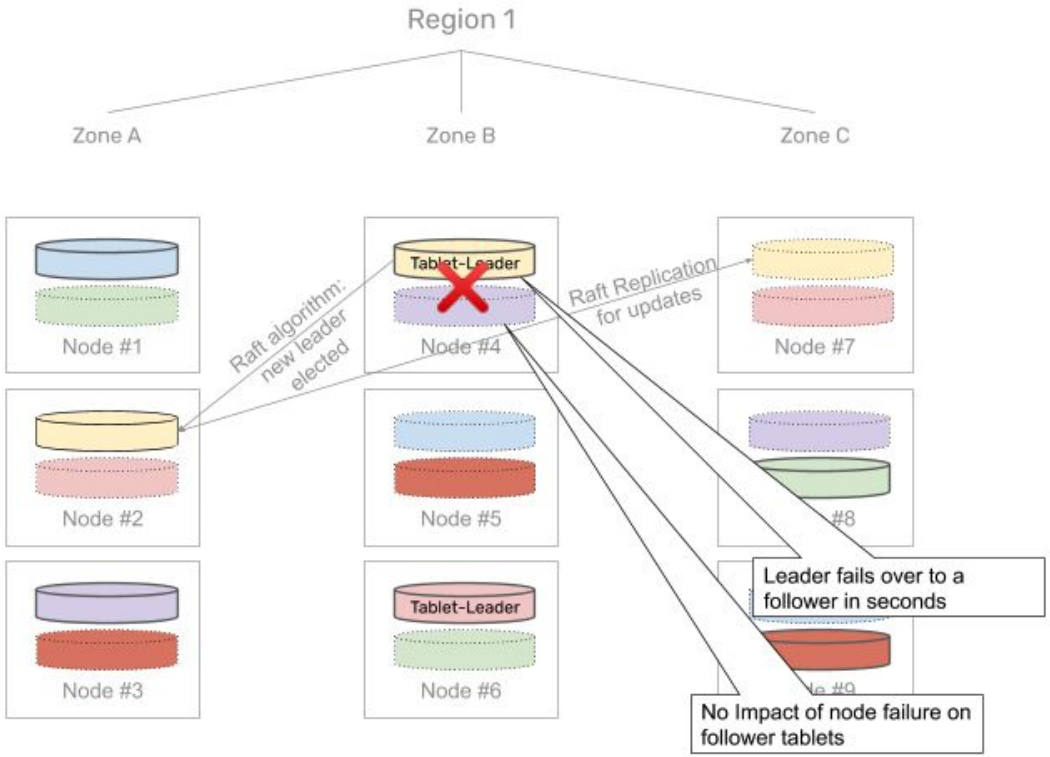


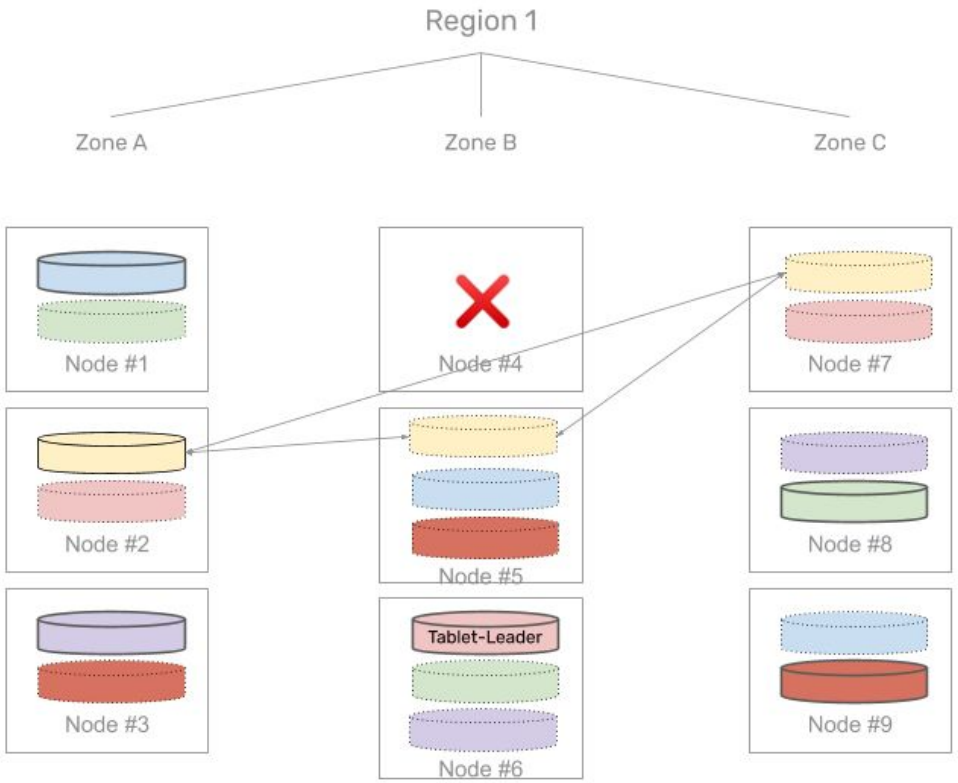
Diagram with replication factor = 3

Tolerating Node Outage



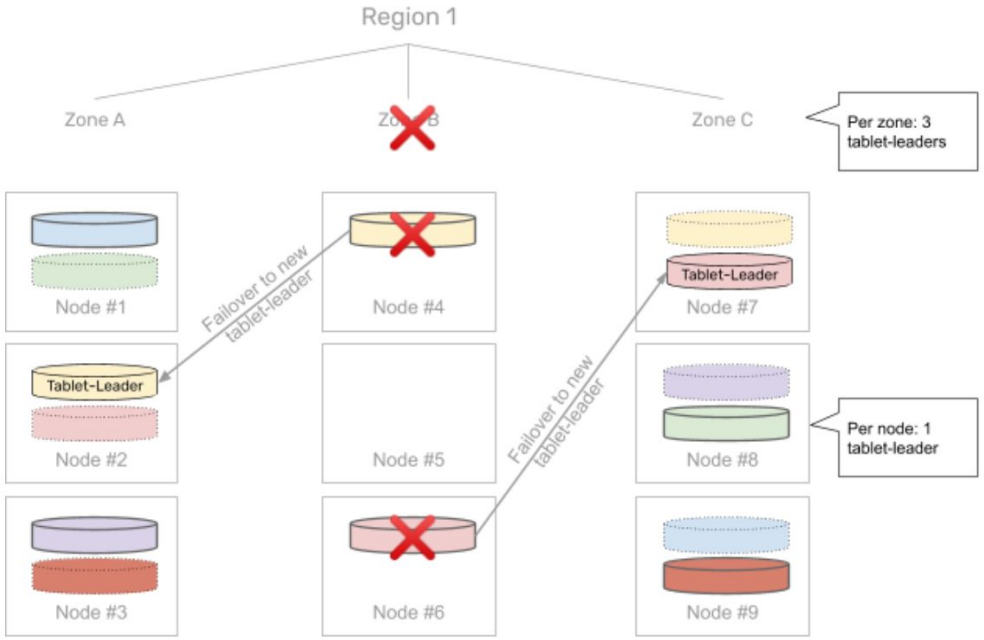
- New tablet leaders re-elected (~3 sec)
- No impact on tablet follower outage
- Follower reads ok

Automatic Resilience



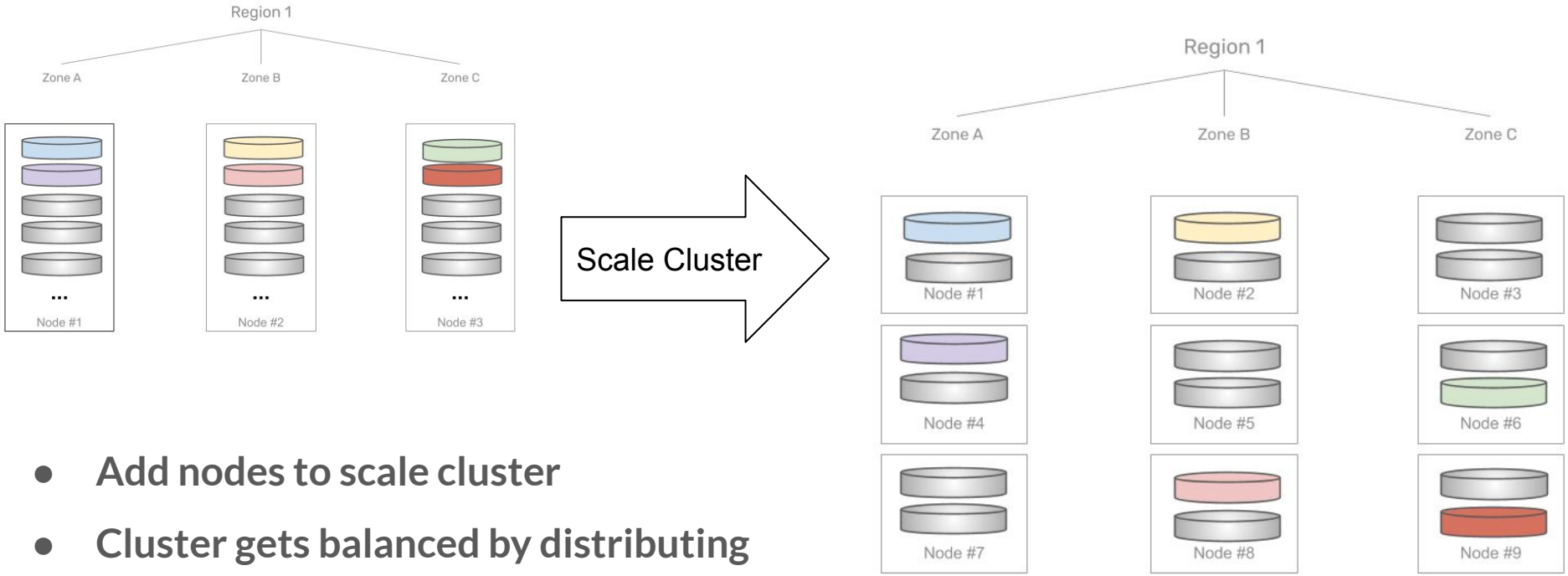
- After 15 mins, data is re-replicated (if possible)
- On failed node recovery, automatically catch up
- Tablet leaders auto-rebalanced

Automatic rebalancing



- New leaders evenly rebalanced
- On failed node recovery, automatically catch up

How Horizontal Scalability Works



- **Add nodes to scale cluster**
- **Cluster gets balanced by distributing existing tablets to new nodes**

Conclusion

- Architecture of Airflow allows you to easily consume distributed database.
- With minimal change you can make your Airflow Cluster truly resilient and horizontally scalable even in Multi-Zone or Multi Region setup.

Airflow + Distributed SQL DB (YugabyteDB) = Awesomeness!




Questions?

LinkedIn: www.linkedin.com/in/thechauhan

E-mail: amchauhan@yugabyte.com

Github: https://github.com/akscjo/yb-utils