Maintaining Cassandra

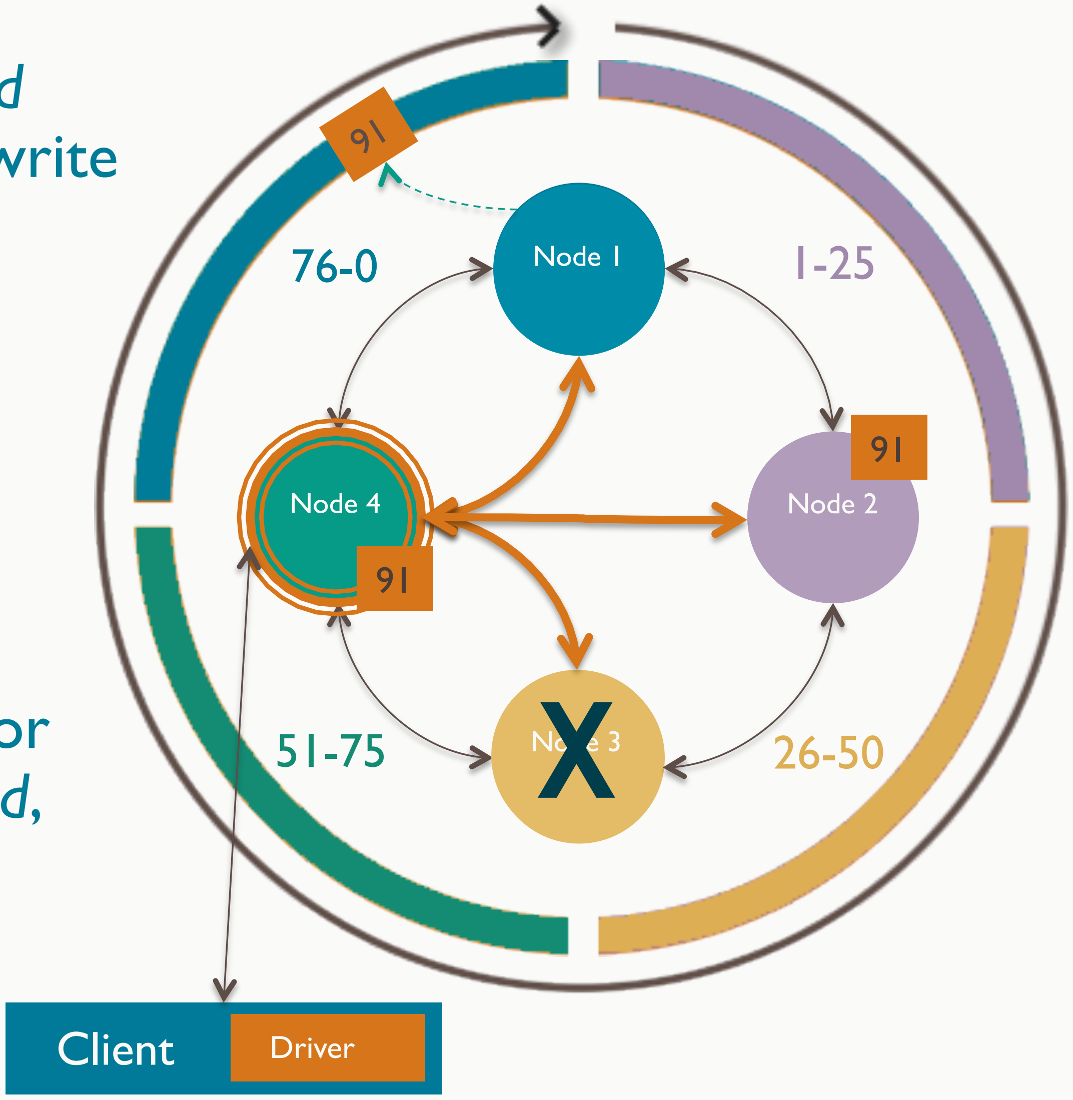
Apache Cassandra:
Operations and Performance Tuning

Learning Objectives

- **Hinted handoff**
- Change the replication factor
- Understand repair operations
- Perform backup and recovery
- Understand security
- Rebuild index

What is a hinted handoff?

- Recovery mechanism for writes targeting offline nodes.
- *Coordinator* can store a *hinted handoff* if target node for a write
  - is known to be down, or
  - fails to acknowledge.
- Coordinator stores the hint in its *system.hints* table.
- Write is replayed when the target node comes online.
- If a node is *decommissioned* or *removed*, or a table is *dropped*, the hints are automatically removed.



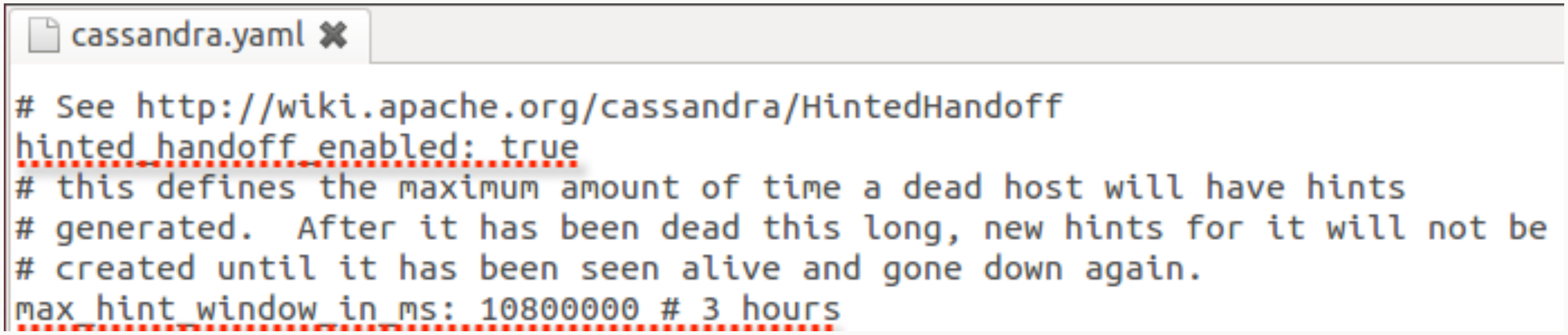
What gets stored in the system.hints table?

- Location of the downed node
- Partition which requires a replay
- Data to be written

```
CREATE TABLE hints (
  target_id uuid,
  hint_id timeuuid,
  message_version int,
  mutation blob,
  PRIMARY KEY (target_id, hint_id, message_version)
);
```

How do you configure hinted handoff?

- Configurations in the *cassandra.yaml* file include
  - hinted_handoff_enabled (default: true) – hinted handoff may be disabled
  - max_hint_window_in_ms (default: 3 hours) – after this consecutive outage period hints are no longer generated until target node comes back online
    - Nodes offline longer are made consistent using *repair* or other operations.



```
# See http://wiki.apache.org/cassandra/HintedHandoff
hinted_handoff_enabled: true
# this defines the maximum amount of time a dead host will have hints
# generated.  After it has been dead this long, new hints for it will not be
# created until it has been seen alive and gone down again.
max_hint_window_in_ms: 10800000 # 3 hours
```

Learning Objectives

- Hinted handoff
- **Change the replication factor**
- Understand repair operations
- Perform backup and recovery
- Understand security
- Rebuild index

How does hinted handoff affect replication?

- Within the hint window, if gossip reports a node is back up, the node with the hinted handoff information sends the data for each hint to the target.
  - Checks every 10 minutes for any hints for writes that timed out during a brief outage.
- Hinted write does not count toward consistency level requirements.
  - If there are not enough live replica targets to satisfy a requested consistency level, then an exception is thrown.
  - One way around this is to use consistency level ANY – guarantees that write is durable and will be readable after a replica becomes available and can receive the hint replay.

Relationship of consistency level and replication

Consistency Level	Read	Write
ALL	Returns record with most recent timestamp after all replicas respond; fails if replica does not respond.	Write to commit log and Memtable on all replicas.
ONE	Returns response from closest replica (determined by snitch). Read repair runs in background to make other replicas consistent.	Write to commit log and Memtable of at least one replica node.
ANY	Cannot be used for read.	Write to at least one node. Hinted handoff if no replica is up; not readable until replicas recover.
QUORUM	Returns response from most recent timestamp after quorum responds. Read repair runs in foreground to make other replicas consistent.	Write to commit log and Memtable on quorum of replicas.

What is the relationship of consistency level and replication? (continued)

- Meeting the consistency level set for a read or a write depends on
  - replication factor
  - replicas available at the time of the read/write operation
- Replication factor, or how many replicas are written, directly affects the calculation for QUORUM
  - QUORUM is calculated as one more than half the number of replicas
- LOCAL_ and EACH_ modify some consistency levels
  - LOCAL_ONE, LOCAL_QUORUM, and LOCAL_SERIAL restrict the validation to data center
  - EACH_QUORUM requires validation from each data center in the cluster

What are consequences of changing replication factor?

- Changing replication factor in a live production environment can have dire consequences.
- If replication factor is lowered to RF = 1, then QUORUM becomes ALL.
- If replication factor is raised, read and writes failures will increase until a repair operation completes.
- Consistency levels of two or higher are more likely to cause blocking read repairs.

How does the consistency level affect performance?

- If the consistency level is set too high
  - Many nodes must respond to complete an operation.
  - With latency for every additional node that must respond.
  - What happens if not enough nodes respond?.
    - Operation failure – either the read or the write does not succeed.
- Consistency level = ONE
  - Only one node must respond and the operation is done.
- Consistency level = QUORUM
  - For RF=3, two nodes must respond for operation to succeed.
  - Higher latency because more nodes must respond before request completes.

How does the replication factor affect performance?

- The slowest node will determine speed at which data is written. You may have to wait for one "slowpoke".
- With reads, there are more replicas to spread the work which can improve performance.
- More replicas, combined with a higher consistency level requirement for reads, can create an environment where failure can occur.
  - If you choose a higher consistency level, make sure you have the replicas to back it up.

Hinted Handoff performance

- Hinted handoff forces Cassandra to continue performing same number of writes even when cluster is operating at reduced capacity.
- If a replica node is overloaded or unavailable, but not marked as down:
  - writes will fail
  - *write_request_timeout_in_ms* (default 10 seconds) determines timeout limit
- If too many nodes reach timeout at once, there can be substantial memory pressure on coordinator node.
  - Coordinator tracks how many hints are currently writing.
  - If the number of hints being written gets too high, coordinator will temporarily refuse writes.
    - UnavailableException is thrown in this case.

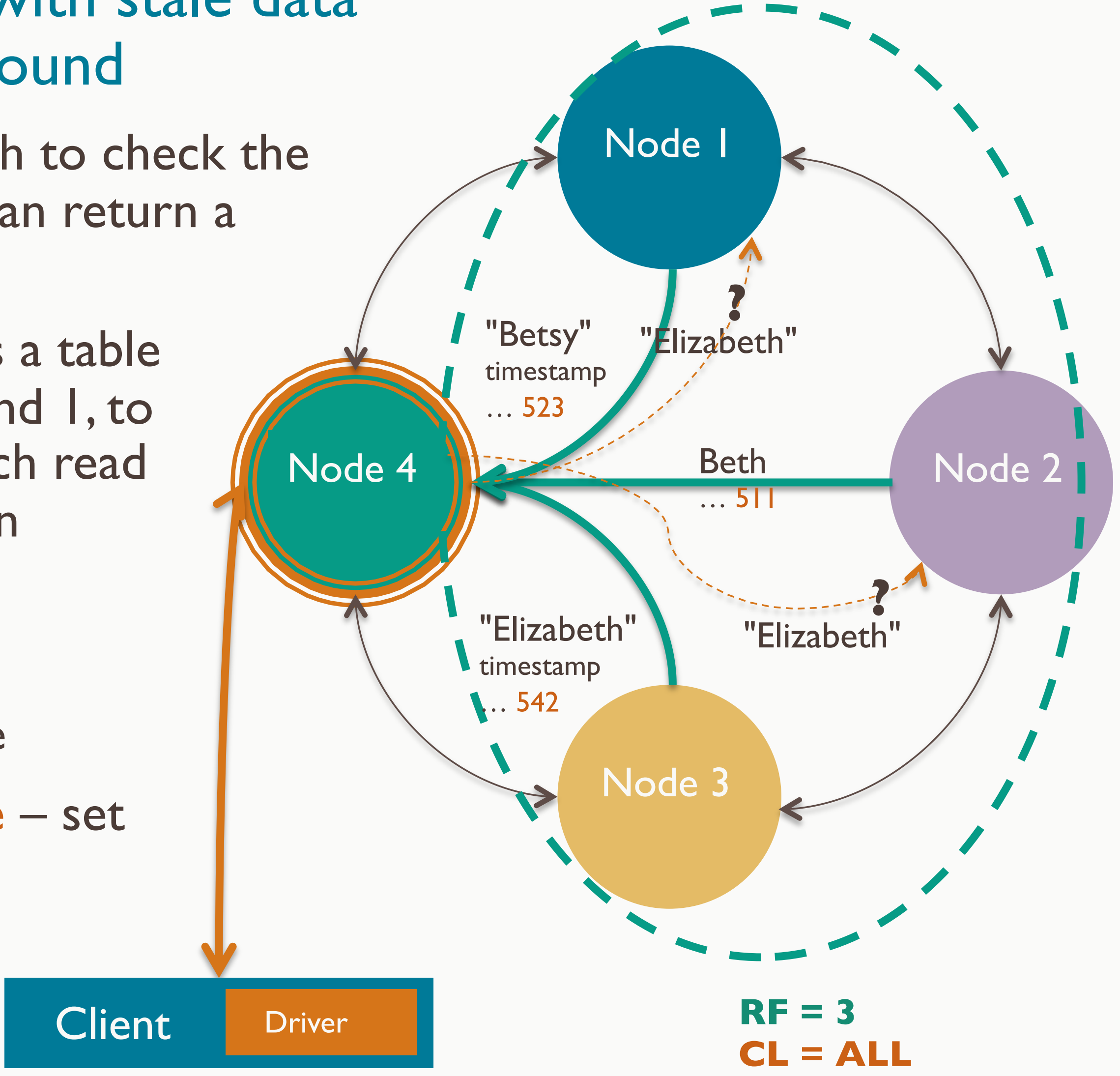
Exercise: Change replication factor

Learning Objectives

- Hinted handoff
- Change the replication factor
- **Understand repair operations**
- Perform backup and recovery
- Understand security
- Rebuild index

What is read repair?

- As part of a read, a *digest query* is sent to replica nodes, and nodes with stale data are updated in the background
  - digest query – returns a hash to check the current data state, rather than return a complete query result
  - read_repair_chance – set as a table property value between 0 and 1, to set the probability with which read repairs should be invoked on non-quorum reads
    - Defaults to 0
    - Can be configured per table
  - dclocal_read_repair_chance – set per data center
    - Defaults to 0.1



What is important about read_repair_chance?

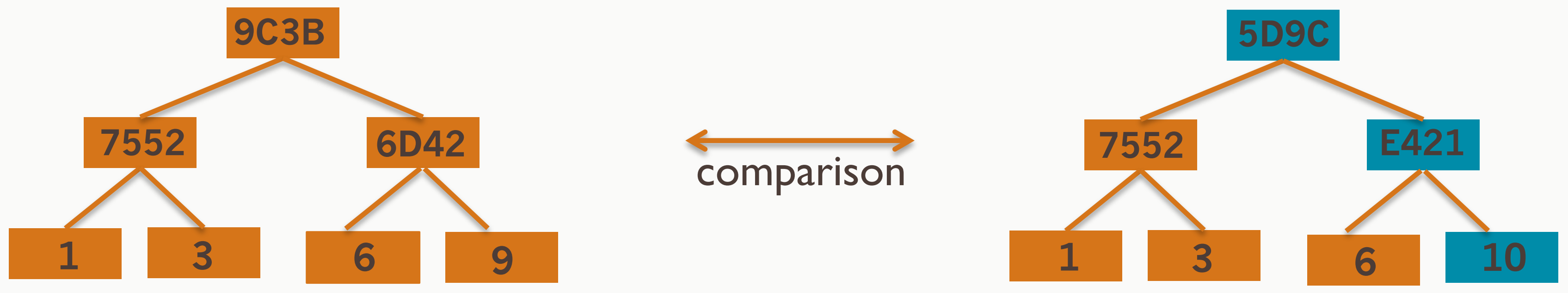
- read_repair_chance is a setting that ranges between 0 and 1
- configured per table for non-ALL consistency levels
- If read_repair_chance is set to 1, then a non-blocking read repair is always performed—but stale data may still be returned.
  - If the data is inconsistent, then all the replicas that need updating are rewritten to the newest data (last write wins).
- This means that every read requires more I/O than is necessary just to read the data.
- This is why read_repair_chance defaults to 0.
  - Default configuration minimizes I/O
- To avoid potentially expensive repairs across data centers, dclocal_read_repair_chance can also be set.

Demo: Read repair

What is a repair?

- Repair is a deliberate action to cope with cluster entropy.
  - Entropy can arise from nodes that were down longer than the hint window, dropped mutations, or other causes.
  - A repair operates on all of the nodes in a replica set by default.
- Ensures that all replicas have identical copies of a given partition
- Consists of two phases:
  - Build Merkle tree of the data per partition.
  - Replicas then compare the differences between their trees and stream the differences to each other as needed.

What does a Merkle tree exchange look like?



1. Start with the root of the tree (a list of one hash value).
2. The origin sends the list of hashes at the current level.
3. The destination diffs the list of hashes against its own, then requests subtrees that are different.
   - If there are no differences, the request can terminate.
4. Repeat steps 2 and 3 until leaf nodes are reached.
5. The origin sends the values of the keys in the resulting set.

Why is repair necessary?

- A node's data can get inconsistent over time.
  - Repair is just a maintenance action in this case.
- If a node goes down for some time, it misses writes and will need to catch up.
- Sometimes it is best to repair a node.
  - If the node has been down longer than the length specified in max_hint_window_in_ms, the node is out of sync. Depending on amount of data, might be faster to repair.
- Sometimes it is better to bring the node back as a new node.
  - If there is a significant amount of data, might be faster just to bring in a new node and stream data just to that node.

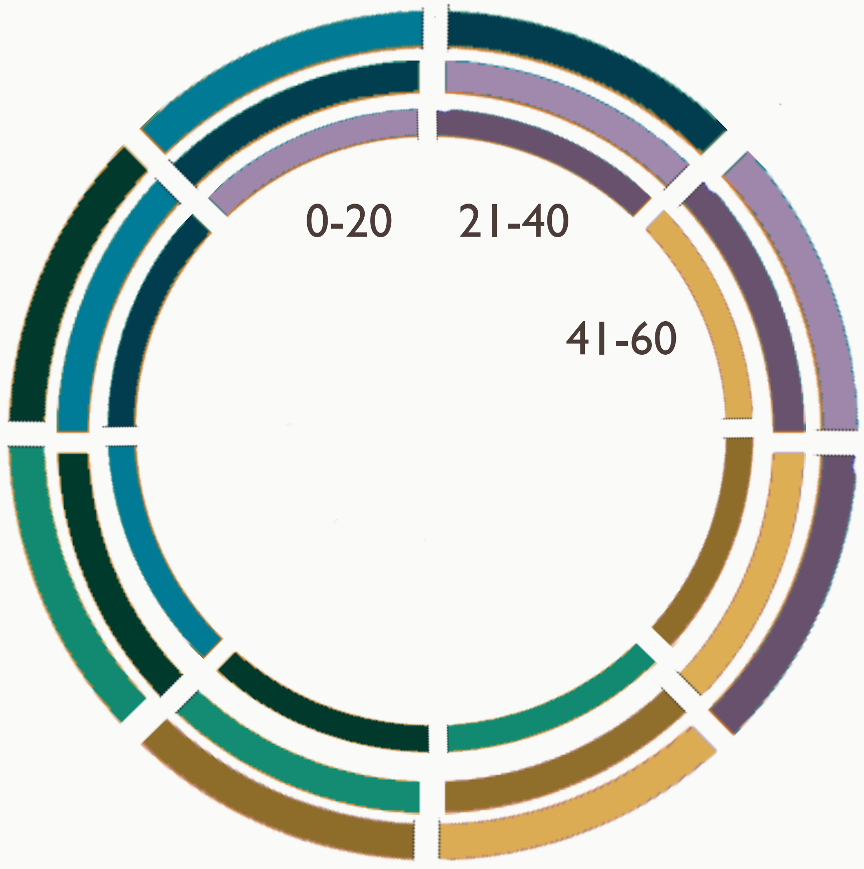
What are best practices for repair?

- Run repair weekly
  - Must run a repair within the value of *gc_grace_seconds* to ensure deleted data is not resurrected
  - If a system seldom deletes or overwrites data, raise the value of *gc_grace_seconds* to schedule wider intervals for repair.
- Run repair on a small number of nodes at a time
  - Preferably one node.
- Schedule for low-usage hours
  - Repair is disk I/O intensive – validation compaction of the Merkle trees.
  - Can be mitigated by using compaction throttling
- Run repair on a partition or subrange of a partition
  - Shortens the time needed to complete the operation

What are primary and secondary ranges?

- Primary Range (inside ring) – "first" node that data is stored on, based on the partition key
- Secondary Range (outside rings) – additional replicas of that same data

- What are the implications for repair?



How do these ranges affect best practices for repair?

- Repair a range of data (*nodetool repair –partitioner-range*)
  - Repairs only the primary range of the node
  - Otherwise, a repair job can take two or three times longer, depending on the number of replicas
- Repair subrange (*nodetool --start-token <token> --end-token <token>*)
  - Repairs only a portion of the data belonging to a node
  - Merkle tree precision is fixed, so many partitions per node may result overstreaming.
  - Important for repairing single-token nodes
  - Steps to use subrange repair:
    - Use the Java *describe_splits* call to ask for a split containing 32K partitions.
    - Iterate through the entire range incrementally or in parallel.
    - Pass the tokens received for the split to the *nodetool repair –st* and *–et* options.
    - Pass the –local options to repair only within the local data center, reducing cross data center transfer load.

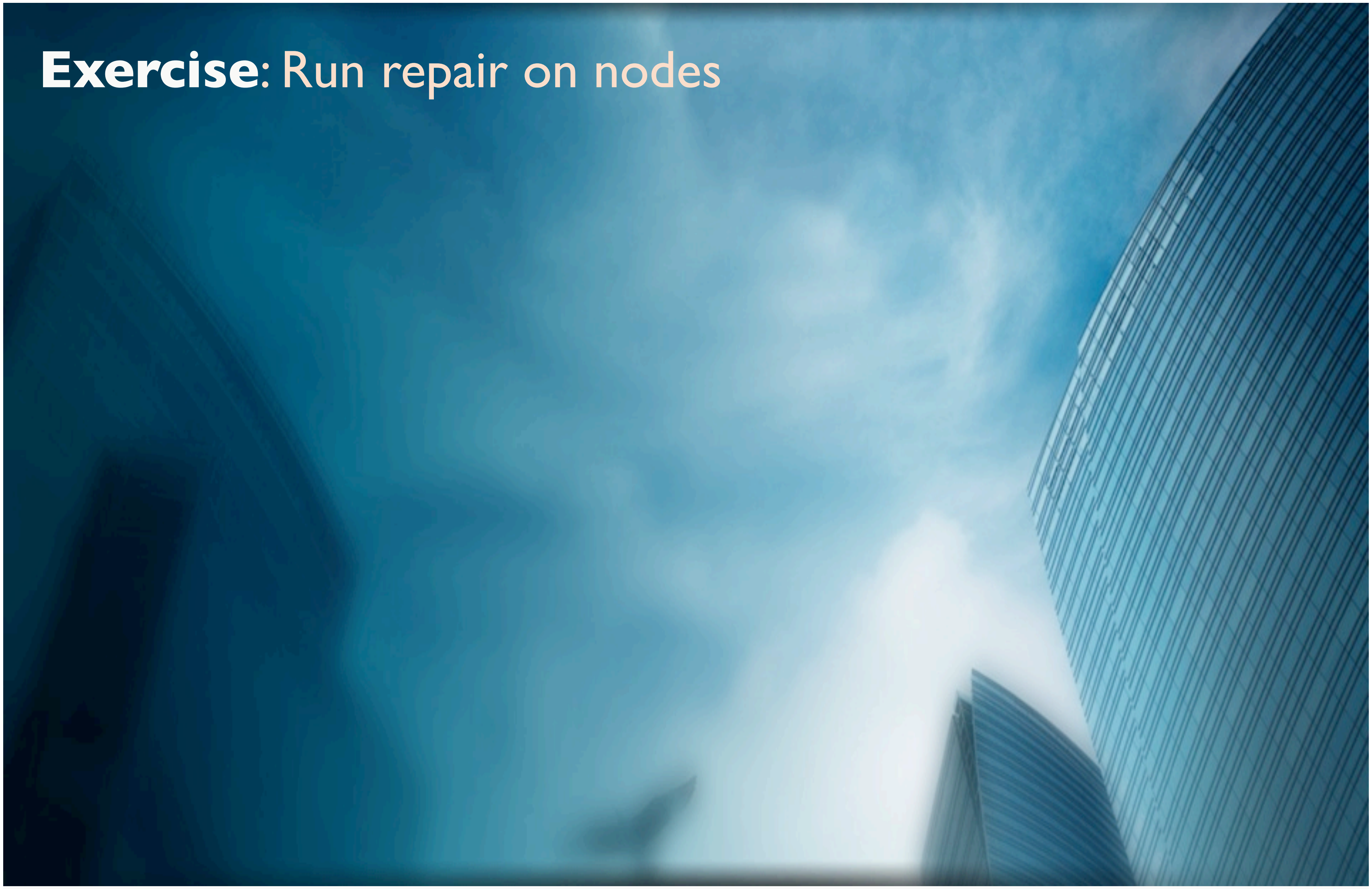
How do you perform repair using *nodetool*?

- *nodetool <options> repair*
- Options:
  - *-dc <dc_name>* identify data centers
  - *-et <end_token>* used when repairing a subrange
  - *-local* repairs only in the local data center
  - *-par* (parallel repair)
  - *-pr* (partitioner-range) repairs only primary range
  - *-st <start_token>* used when repairing a subrange
  - *-- <keyspace> <table>*
- If you don't specify an option, sequential is the default.

How do you perform repair using *OpsCenter*?

- Repair single nodes using the action.
- After choosing "repair", select the options:
  - Keyspace and tables—default is All
  - Only use nodes in the same data center for the repair.
  - Only repair the node's primary range (partitioner-range).
  - Fully concurrent repair—use only with SSDs or during time when intensive disk I/O is acceptable.
    - Runs repairs on all replicas with this node's range at the same time.

Exercise: Run repair on nodes



Learning Objectives

- Hinted handoff
- Change the replication factor
- Understand repair operations
- **Perform backup and recovery**
- Understand security
- Rebuild index

Why should backups be done?

- Programmatic accidental deletion or overwriting of data
  - Changes could be propagated to replicas before error is known, and wipe out all data.
  - Corrupted SSTables can be restored from a snapshot and incremental snapshots.
- For single node failure, recovery can be from a live replica.
  - If a replication factor > 1 is used, another node will have the data.
  - Add a new node and let the data stream to it.
- To recover from catastrophic data center failure
  - Multi-data center—new data center can be rebuilt from an existing one.
  - Not multi-data center—copied offline backup files may be needed.
- Two separate operations
  - Snapshots
  - Offline backups

What is a snapshot?

- Represents the state of the data files at a particular point in time.
- Consists of a single table, single keyspace, or multiple keyspaces.
- Flushes all in-memory writes to immutable SSTables on disk.
- Makes hard links to SSTables for each keyspace in a snapshot directory.
  - This is DIFFERENT than copying actual data files offline – takes less disk space.

How do incremental backups work?

- Incremental backups create a hard link to every SSTable upon flush.
  - User must manually delete them after creating a new snapshot.
- Incremental backups are disabled by default.
  - Configured in the *cassandra.yaml* file by setting *incremental_backups* to *true*.
- Need a snapshot before taking incremental backups
- Snapshot information is stored in a snapshots directory under each table directory.
  - Snapshot need only be stored once offsite.
- Incremental backups are all stored in a *backups* directory under the keyspace data directory.
  - Enables storing incremental backups offsite more easily
- Both snapshot and incrementals are needed to restore data.
- Incremental backup files are not automatically cleared.
  - Clear when a new snapshot is created.

Where are snapshots stored?

- Snapshots and incremental backups are stored on each Cassandra node.
  - Handy, if a simple restore is needed
  - Not so good if there is a hardware failure
- Commonly, files are copied to an off-node location.
  - Open source program, *tablesnap*, is useful for backing up to S3
  - Scripts can be used to automate backing up files to another machine – cron + bash script, rsync, etc

What is auto snapshot?

- A configuration setting in cassandra.yaml that indicates whether or not a snapshot is taken of data before tables are truncated and tables and keyspaces are dropped
- *STRONGLY* advise using the default setting of **true**

How do you create a snapshot using *nodetool*?

- The *nodetool snapshot* command takes a snapshot of:
  - One or more keyspaces
  - A table specified to backup data

```
bin/nodetool [options] snapshot (-cf <table> | -t <tag> -- keyspace)
```

- Same options as *nodetool* command
  - -h [host] | [IP address]
  - -p port
  - -pw password
  - -u username
- Can specify keyspace and/or table and tag to name the snapshot
  - All keyspaces will be snapshot if a keyspace is not specified.
- Can use a parallel *ssh* tool to snapshot entire cluster

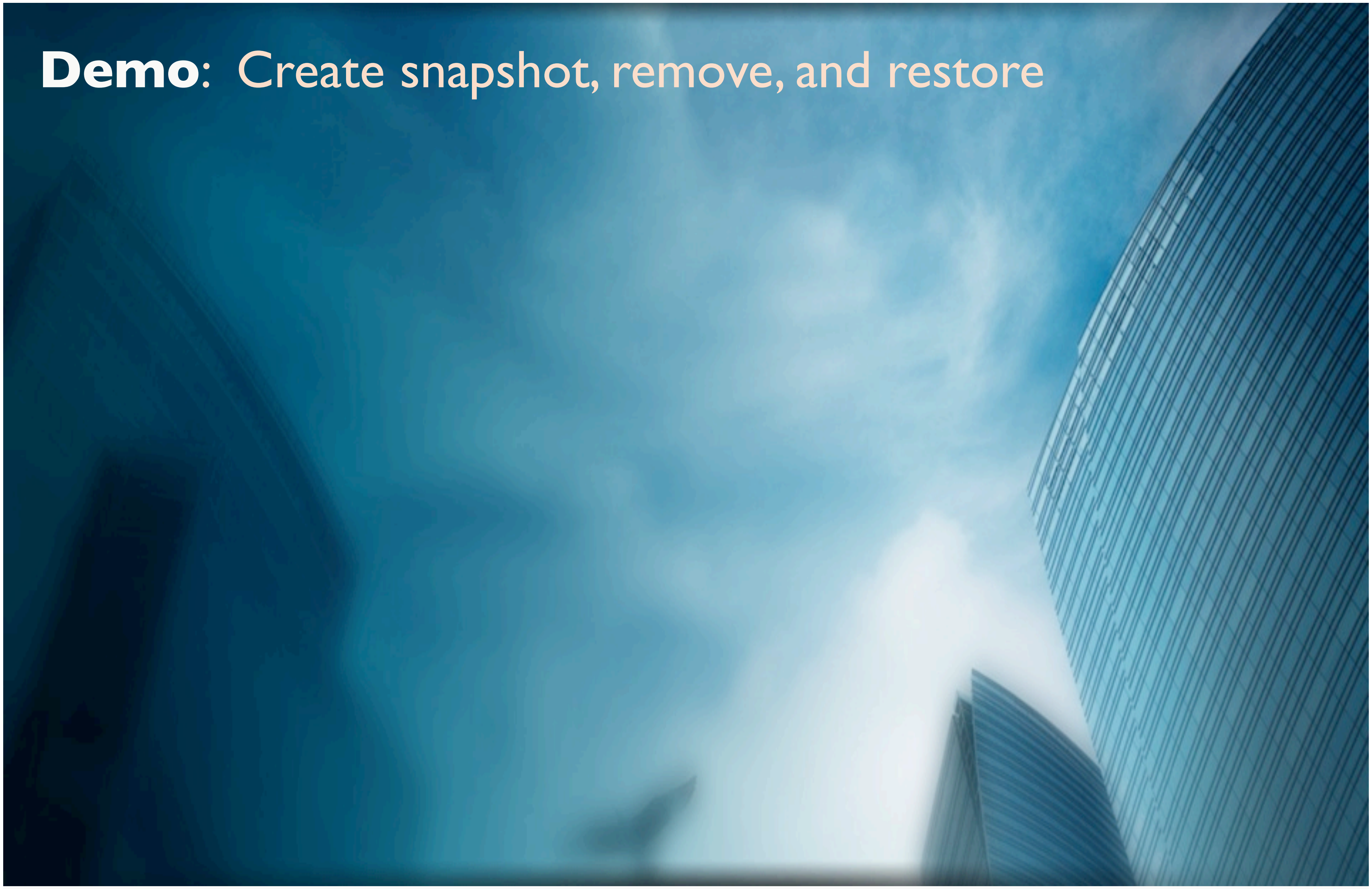
How do you remove a snapshot using *nodetool*?

- The *nodetool clearsnapshot* command removes snapshots.
- Same options as *nodetool* command
- Specify the snapshot file and keyspace
- Not specifying a snapshot name removes all snapshots.
- Remember to remove old snapshots before taking new ones—previous snapshots are not automatically deleted.
- To clear snapshots on all nodes at once, use a parallel ssh utility.

How is a snapshot restored using *nodetool*?

- Most common method is to delete the current data files and copy the snapshot and incremental files to the appropriate data directories.
  - If using incremental backups, copy the contents of the *backups* directory to each table directory.
  - Table schema must already be present in order to use this method.
  - Restart and repair the node after the file copying is done.
- Another method is to use the sstableloader
  - Must be careful about its use – can add significant load to cluster while loading.

Demo: Create snapshot, remove, and restore



How do you perform cluster-wide backup and restore?

- SSH programs
  - *pssh*
  - *clusterssh* is another tool that can be used to make changes on multiple servers at the same time
- Honorable mention—*tablesnap* and *tablerestore*
  - For Cassandra backup to AWS S3
- Recovery

Exercise: Perform cluster-wide backup and recovery

How is backup and recovery done with OpsCenter?

- Backup and recovery are only enabled for DSE
  - OpsCenter has a service to automatically schedule snapshot creation
  - Recovery is also automated, streamlining the process

Demo: DSE OpsCenter backup and recovery

Learning Objectives

- Change the replication factor
- Perform backup and recovery
- Understand repair operations
- **Understand security**
- Rebuild index

What security features are available in Cassandra?

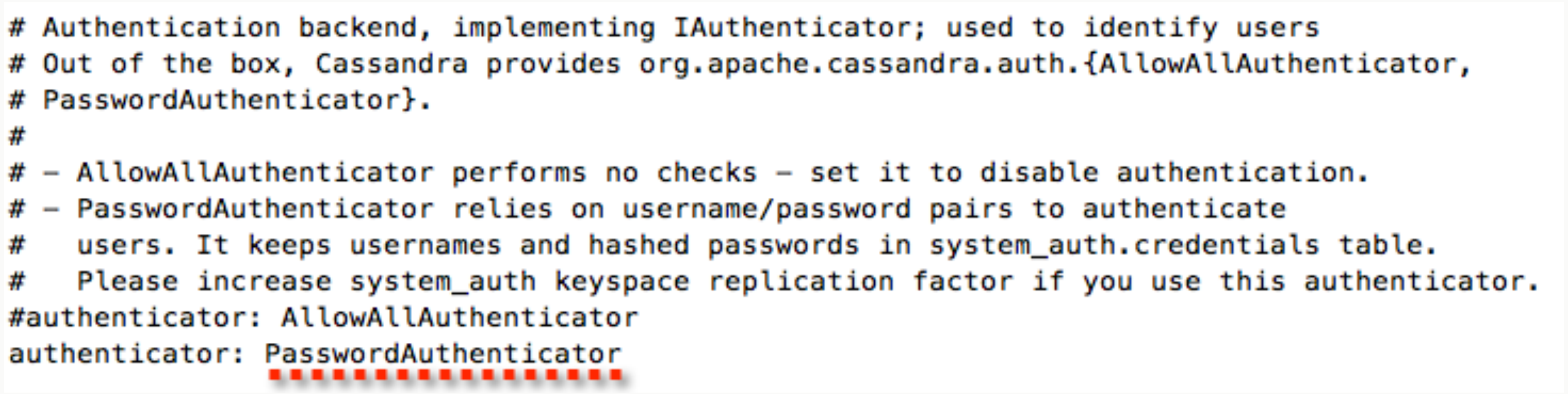
- Client-to-node Encryption
  - SSL is used to ensure data is not compromised while transferring between server and client.
- Node-to-node Encryption
  - Node-to-node encryption protects data transferred between nodes in a cluster using SSL.
- Authentication
  - Based on internally controlled accounts and passwords
- Object Permission Management
  - Authorization can be granted or revoked per user to access database objects

What are users and superusers?

- Superusers have super powers
  - Create users and assign passwords
  - Grant and revoke permissions to database tables
- Users can have access to tables
  - Tables that a user has access to can be specified by a superuser

How is the *cassandra.yaml* file configured for user accounts?

- Increase replication factor of system_auth keyspace
  - Change the authenticator to use PasswordAuthenticator
  - Change the authorizer to CassandraAuthorizer



```
# Authentication backend, implementing IAuthenticator; used to identify users
# Out of the box, Cassandra provides org.apache.cassandra.auth.{AllowAllAuthenticator,
# PasswordAuthenticator}.
#
# - AllowAllAuthenticator performs no checks - set it to disable authentication.
# - PasswordAuthenticator relies on username/password pairs to authenticate
#   users. It keeps usernames and hashed passwords in system_auth.credentials table.
#   Please increase system_auth keyspace replication factor if you use this authenticator.
#authenticator: AllowAllAuthenticator
authenticator: PasswordAuthenticator
```

- Start Cassandra using default superuser name/password (cassandra/cassandra)
- Start *cqlsh* with same credentials
- Setup user accounts
  - Create another superuser account--the default can be changed and not used

How are users created and modified?

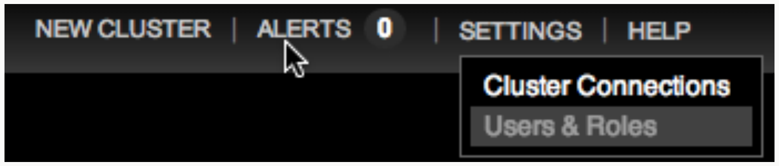
- CREATE USER
  - Specify user name and password
- ALTER USER
  - Any aspect of a user or superuser
  - Exception: superuser cannot change own superuser status to prevent accidental lockout
- LIST USERS
  - List all users and their User/Superuser status
- DROP USER
  - Delete a user and their password
- GRANT
  - Grant access to tables and/or keyspaces
- REVOKE
  - Revoke access to tables and/or keyspaces

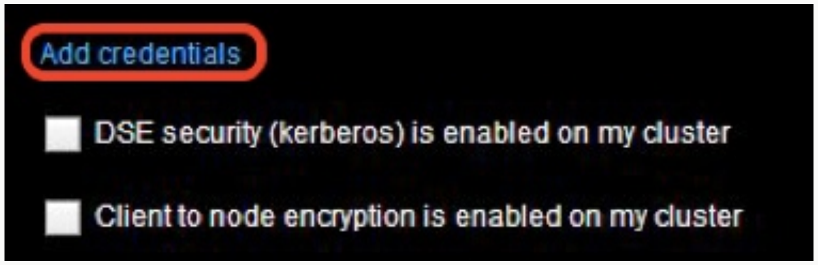
How are permissions granted and revoked?

- GRANT
  - Permissions can be to access all keyspaces, a named keyspace, or a named table.
  - CREATE keyspace or table, DROP keyspace or table
  - INSERT, DELETE, UPDATE, TRUNCATE
  - SELECT – Must have SELECT permission to perform SELECT queries.
- REVOKE
  - Everything that GRANT can do, except REVOKE undoes it

Does *OpsCenter* connect to clusters with authorization enabled?

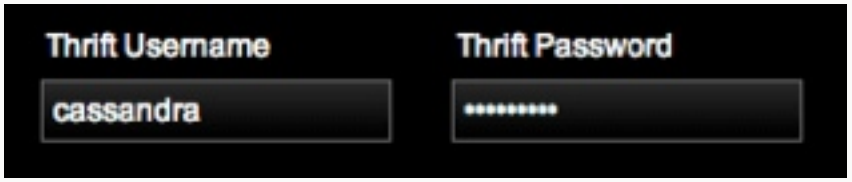
- The username and password can be configured when adding an existing cluster.
  - New Cluster > Manage Existing Cluster
- The username and password can be configured when editing a cluster already added to *OpsCenter*.
  - Settings > Users & Roles > Add Credentials
- The Thrift credentials are the ones that correspond to the user and password created in CQL





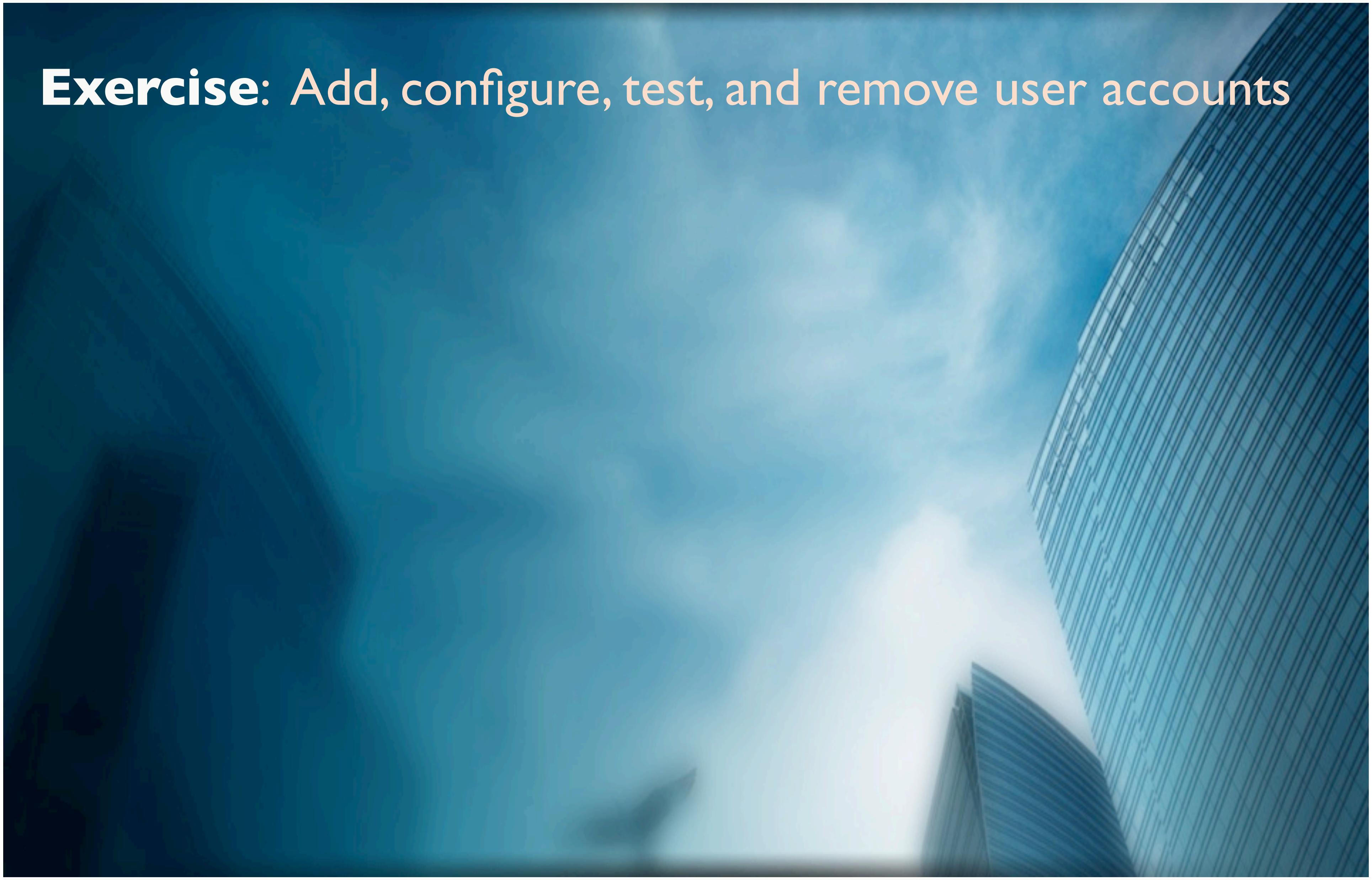
DSE security (kerberos) is enabled on my cluster

Client to node encryption is enabled on my cluster


Thrift Username: cassandra

Thrift Password: **********

Exercise: Add, configure, test, and remove user accounts



Learning Objectives

- Change the replication factor
- Perform backup and recovery
- Understand repair operations
- Understand security
- **Rebuild index**

Why would you rebuild indexes?

- Occasionally secondary indexes get out-of-sync.
- If you suspect that this has happened, it is easier to rebuild the indexes than to verify that you have a problem.
- Rebuilding the indexes:
  - Doesn't affect data consistency
  - Doesn't drop the index
  - Adds the data again to the index
- Consumes CPU and disk I/O while completing this operation
  - Probably not a good idea to do this in the middle of a busy day.

How do you use *nodetool rebuild index?*

- *nodetool <options> rebuild_index keyspace table.index*
  - *E.g. nodetool rebuild_index stock trades trades.trades.trades_ticker.idx*
- The usual options: host, port, username, password
- Run this on every node as each node maintains its own indexes

How do you rebuild indexes using OpsCenter?

- You can't – use *nodetool*

Demo: Rebuild indexes

Summary

- It's best to set the replication factor for a keyspace on creation, as changing it to either a higher or lower value later can cause issues.
- Read repairs take place when a read query is issued, and updates the data on non-query nodes.
- Repair is should be done at least once a week.
- Cassandra has security features that allow users/passwords to be used to grant/revoke access to the database.
- Backup and recovery should be used as part of a disaster recovery strategy.

Review Questions

- What is the relationship of hinted handoff and replication?
- How does the replication factor affect the consistency level?
- What is the difference between read repair and repair?
- How often should nodes be repaired?
- What are the impacts to the cluster of running *nodetool repair*?
- Describe the difference between the primary range and the secondary range of a node.
- Why would you want to repair a subrange on a node?
- What kind of security features does Cassandra have?
- How does backup work for Cassandra? What is a snapshot?
- What additional tasks should you do to ensure your data is safe?
- Why would you rebuild secondary indexes on a node?