GDPR Compliance in Storage Systems

Irvin Lim (ilim5)
Brown University

Back to GDPR...

- Article 5: Principles relating to processing of personal data
- Article 15: Right of access by the data subject
- Article 17: Right to erasure (“right to be forgotten”)
- Article 20: Right to data portability
- Article 21: Right to object
- Article 25: Protection by design and by default
- Article 30: Records of processing activities
- Article 32: Security of processing
- Articles 33/34: Notification of personal data breach to supervisory authority and data subject
- Article 46: Transfers subject to appropriate safeguards

Back to GDPR...

- Article 5: Principles relating to processing of personal data
- Article 15: Right of access by the data subject
- Article 17: Right to erasure (“right to be forgotten”)
- Article 20: Right to data portability
- Article 21: Right to object
- Article 25: Protection by design and by default
- Article 30: Records of processing activities
- Article 32: Security of processing
- Articles 33/34: Notification of personal data breach to supervisory authority and data subject
- Article 46: Transfers subject to appropriate safeguards

Main Goals

- Right to access:
  Retrieve all personal data associated with user in a timely fashion
- Right to erasure:
  Delete all personal data associated with user in a timely fashion
- Right to data portability:
  Transfer data between data controllers on request
- Right to object:
  Data subjects have the right to object to use their personal data for specific purposes
- Monitoring/Logging:
  Maintain a record of all processing activities, including purpose, who has access

Degrees of Compliance

- Response time: Real-time vs Eventual Compliance
  - Real-time compliance requires each GDPR task to be performed synchronously
  - Writes will incur significant overhead
- Capability: Full vs Partial Compliance
  - Full compliance: Natively supporting all GDPR features
  - Partial compliance: Requires external infrastructure or policy components

Realistically, what extent of compliance is expected of companies?

GDPR-Compliant Redis

What is Redis?

- Key-value store
- Single-threaded, in-memory store
  - Ensures atomicity (linearizability) with single replica
  - Very fast since memory-backed
- Disk-backed using AOF and RDB persistence
  - Append-only file (AOF) stores each and every Redis operation
  - RDB persistence stores point-in-time snapshots of datasets periodically
  - Redo journaling model:
    i. Operations are written and persisted in AOF logfile
    ii. Operations are immediately applied to in-memory data
    iii. Occassionally, AOF log is compacted in the background into RDB snapshot

Let's Use Redis!

```
redis> SET key value [expiration EX seconds|PX milliseconds] [NX|XX]
redis> SET 96903602 '{"user_id": 587, "name": "Irvin Lim", "credit_score": 850}'
OK

redis> GET key
redis> GET 96903602
"{\"user_id\": 587, \"name\": \"Irvin Lim\", \"credit_score\": 850}"
```

Redis uses text-based protocol

All values in Redis are strings!

Indexing by User ID

```
redis> SADD key member [member ...]
redis> SADD user_associations:587 96903602
(integer) 1

redis> SMEMBERS key
redis> SMEMBERS user_associations:587
1) "96903602"

redis> SISMEMBER key member
redis> SISMEMBER user_associations:587 123
(integer) 0
```

Redis Sets:

- Insert: O(1)
- Membership check: O(1)
- List: O(N), where N is set cardinality

Attaching Auxiliary Metadata

```
redis> HSET key field value
redis> HSET metadata:96903602 data_purpose cc_marketing_emails
(integer) 1

redis> HGET key field
redis> HGET metadata:96903602 data_purpose
"cc_marketing_emails"

redis> HGETALL key
redis> HGETALL metadata:96903602
1) "data_purpose"
2) "cc_marketing_emails"
```

Redis Hashes: Namespaced key-value lookups

Should also add reverse index to lookup all keys associated with purpose

Timely Expiry

```
redis> EXPIRE key seconds
redis> EXPIRE 96903602 86400
(integer) 1
```

- Eventual compliance:
  - Does not guarantee removal of expired key from storage (without `GET`)
  - Lazy probabilistic algorithm: https://redis.io/commands/expire#how-redis-expires-keys
  - Every 0.1 seconds, test 20 random keys with an associated expire, and delete all keys found expired. If more than 25% of keys were deleted (i.e. more than 5 keys), then repeat.
- Real-time compliance:
  - Not supported natively
  - Suggests to use timeseries databases (TSDBs) or appropriate data structures
  - Maybe use Redis sorted sets indexed on timestamp?

Timely Deletion

```
redis> DEL key [key ...]
redis> DEL 96903602
(integer) 1
```

- Eventual compliance:
  - Deleted data persists on disk until AOF compaction occurs
- Real-time compliance:
  - Use “`appendfsync always`” option - will call `fsync(2)` on each update
  - Throughput drops by 20x!
- Redis Persistence Demystified: http://oldblog.antirez.com/post/redis-persistence-demystified.html

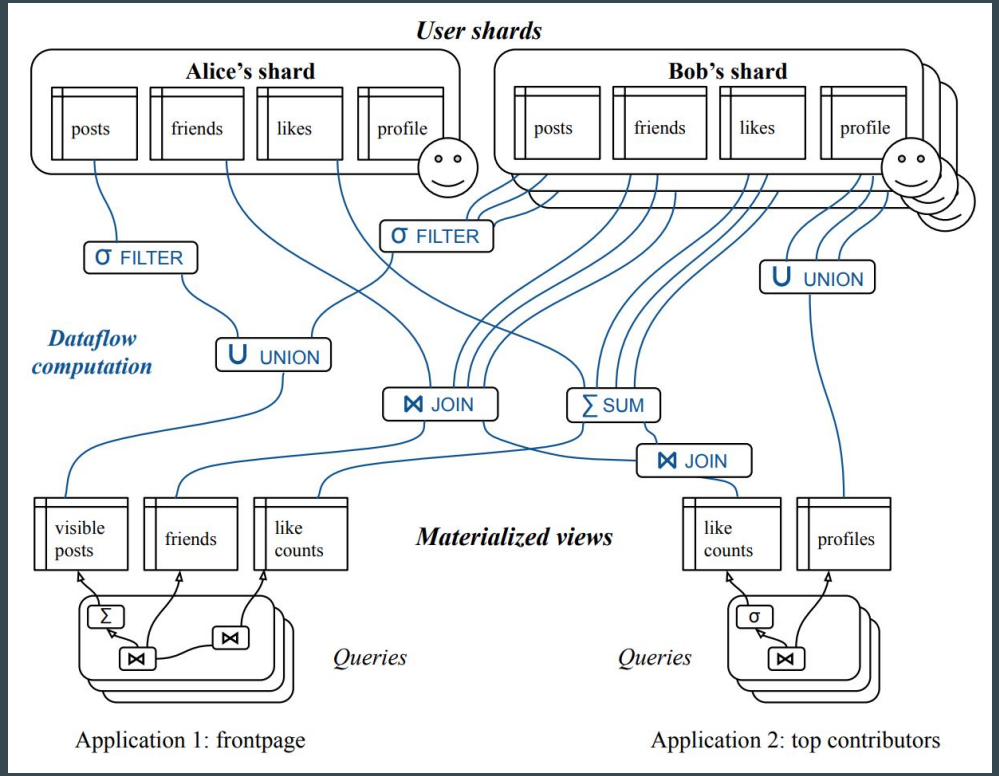
Discussion: Is this level of compliance sufficient?

- Eventual compliance to avoid poor performance
  - Lose at most 1 second of logs
  - AOF compaction happens at least once an hour
  - Expiry is not guaranteed to ever occur!
- Easy to make mistakes
  - Redis does not provide any guarantees about compliance on its own
  - Must be used strictly in the way as described

What are the advantages and drawbacks of this approach?
Can we design better abstractions?

GDPR Compliance by Construction (CbyC)

Looks Familiar?



System Design

- User shards
  - Never contains (derived) information associated with other users
- Materialized views
  - Derived from user shard data, possibly from multiple user shards
  - Dynamic creation and destruction of views
  - Excellent read and incremental update performance
- Partially-stateful dataflow
  - Writes will start from user shard and flow through all dependency paths
  - Support additions, updates, removals as incremental computations
  - Selective materialization of downstream views

Data Deletion

- Removing dependent downstream data requires operators to understand insertions and revocations (`ON DELETE CASCADE`)
- Another challenge: Data stored in user shard might be co-owned by other users
  - Can make individual exceptions using policy, e.g. data might want to be persisted through user deletions
  - Move derived data records into an "anonymous" user shard
  - Complications?

Data Deletion in the Real World

- NPM `left-pad` incident (2016)
  - Simple 17-line package was a dependency of many popular packages (e.g. Babel)
  - Developer rage quits and deletes all modules from Node.js package repository
  - Broke the builds for everyone until it was manually restored
- GitHub “ghost” user
- GitLab user deletion policy

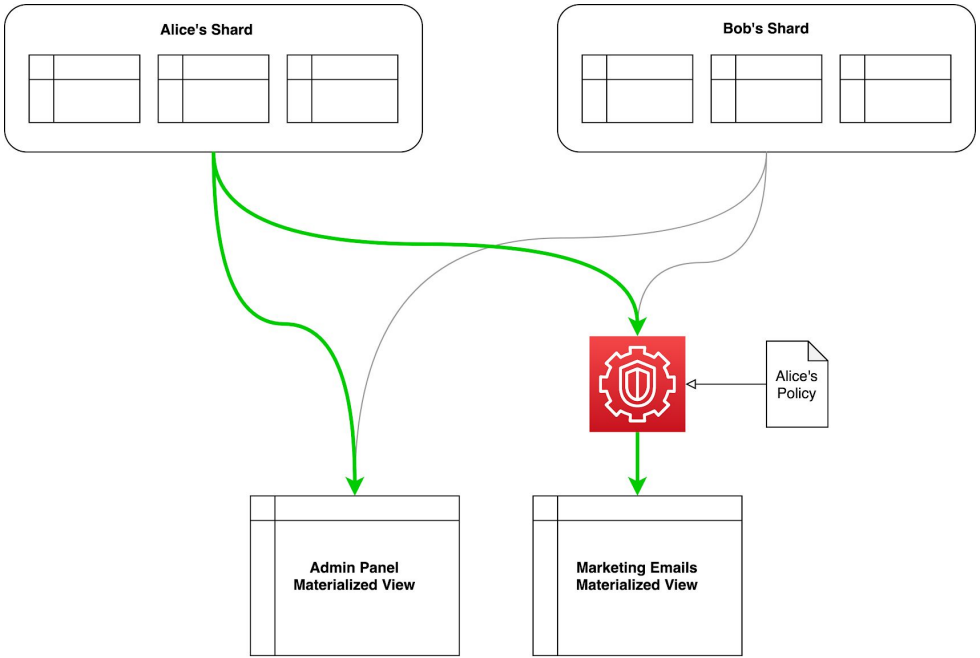
When a user account is deleted, not all associated records are deleted with it. Here's a list of things that will **not** be deleted:

- Issues that the user created.
- Merge requests that the user created.
- Notes that the user created.
- Abuse reports that the user reported.
- Award emoji that the user created.

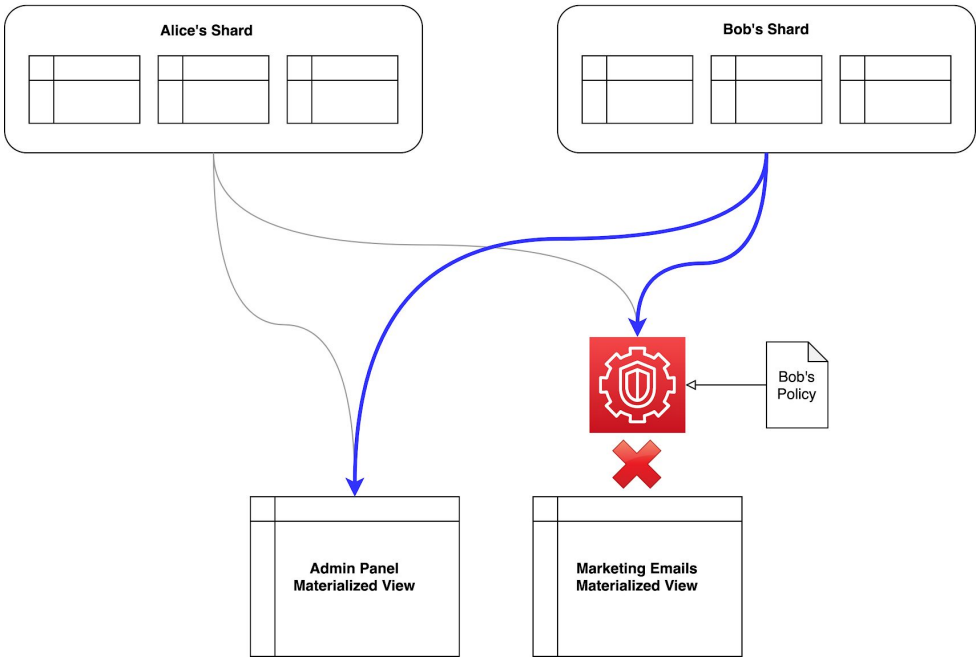
Guarded Data Processing

- Augment materialized views with data processing purpose
- Prevent flow of data into subgraphs rooted at “guard” operator if the user objected to processing for a specific purpose

Guarded Data Processing



Guarded Data Processing



How well does each solution provide GDPR compliance?

Right of Access

Article 15: Right of access by users

Retrieve all personal data associated with user in a timely fashion

- Redis: Metadata indexing on user ID
  - Application must make sure to add correct indices on write
  - Use `SMEMBERS` on `user_associations:uid` set
- CbyC: Retrieve entire user shard

Right to Erasure

Article 17: Right to erasure (“right to be forgotten”)
Delete all personal data associated with user in a timely fashion

- Redis: Delete all associated keys in the `user_associations` set
  - Eventual compliance: Deletion and expiry is not guaranteed
- CbyC: Withdraw entire user shard
  - Dataflow system ensures revocation update is sent to downstream views
  - Does not provide auto-expire capabilities

Right to Data Portability

Article 20: Right to data portability

Transfer data between data controllers on request

- Redis: Export all associated keys in the `user_associations` set, import into new system
  - Destination database needs to re-add associations
- CbyC: Export entire user shard, import into new system
  - Dataflow system ensures incremental updates of all downstream views

Right to Object

Article 21: Right to object

Data subjects have the right to object to use their personal data for specific purposes

- Redis: Blacklist per-user data processing purposes
  - General idea: Lookup associated purpose metadata to decide whether to return data
  - Tedious to manage in application logic
- CbyC: “Guard” operators that provide guarantee

Article 30: Records of processing activities

Monitoring/Logging:
Maintain a record of all processing activities, including purpose, who has access

- Redis: Make use of existing AOF log
  - Tradeoff between compliance guarantee and performance
- CbyC: Native RDBMS audit log capabilities?

Comparing Both Solutions

How do the two stack up?

GDPR-Compliant Redis	Compliant by Construction DBMS
NoSQL	Relational
Primitive key-value lookup	Supports rich relational queries
Tunable compliance levels	Only full compliance
Linearizable consistency (on single replica) Eventual consistency (in cluster)	Eventual consistency
Can retrofit existing Redis deployments	Might need to redesign database schema
Flexible in terms of deciding where to store user data	Restrictive, all user-owned data must live in user shard
Tedious labeling of associations, easy to make mistakes	Safer because compliance abstractions are provided out of the box

Discussion

Retrofit or Design new paradigms?

- Retrofitting existing systems can lead to substantially degraded performance
  - Redis with synchronous logging: 20x slower
- Backward compatibility of existing systems
  - Redis: No migration needed
  - CbyC: Design materialized view schemas to match existing schemas
- Compliance abstractions vs Flexibility
  - Don’t allow the developer to make mistakes!
- Tunable compliance levels
  - Is strict compliance completely idealistic or actually necessary?

Which approach is better?

Thank you!