facebook

XDB

Shared MySQL hosting at Facebook scale

Evan Elias
Percona Live MySQL Conference, April 2015

What is XDB?

- In-house system for self-service database creation
  - Web UI
  - API for automated creation and management

- Intended for production databases
  - replication, backups, monitoring, etc
  - supported by all standard Facebook MySQL automation

- Shared resources
  - several MySQL instances per host
  - many database schemas per MySQL instance

Motivations

- Facebook employs thousands of engineers across hundreds of teams

- Constant need for new special-purpose MySQL databases
  - A few hundred new specialized databases created *every month*
  - Most are unsharded, but a few with tens to thousands of shards

- MySQL Infrastructure team currently has a dozen engineers
  - One on-call at a time
  - Production issues take precedence

Facebook MySQL tiers

- Over a dozen sharded “tiers”, each with own hostname scheme
    - Separate tiers for user data, timeline data, message data, etc
    - Vast majority of our database hosts belong to these tiers
    - Within one of these tiers, every shard is uniform
    - i.e. same tables, query pattern, sharding scheme

- A tiny fraction of database hosts are devoted to hosting “special snowflake” databases that do not fall into these tiers

Special-purpose databases

Why are "special snowflakes" hard to manage?

- Thousands of distinct databases in this category
  - Each has a different workload and table schema
  - Mix of internal applications, backend data for services, experimental features, offline/OLAP workloads, etc
  - Owned by wide range of teams
- Vast majority of these data sets are unsharded
  - For the few that *are* sharded, the sharding schemes vary

Why shared hosting?

- Different motivations than a public cloud
  - All internal
  - Teams don’t have to “pay” for their hardware :(
  - Easy to communicate with the “customers” :)

- Pack many databases per physical host
- Avoid overhead of virtualization
- Avoid complexity from too many MySQL instances on host
- Maximize utilization of resources

Before XDB

CDB (“Central Database”) tier

- Completely manual setup process
    1. Obtain spare MySQL instances
    2. Set up replication
    3. Create the database schema
    4. Enter into service discovery system
- Time-consuming
- Risk of human error

XDB v1

Live in early 2013

- Web UI, written in PHP, for creating MySQL databases
  - User submits form with db name, description, master region, estimated max size, etc
  - Request goes into a queue
- Every 30 minutes, a Python cron processes the queued requests
  - Create database on an existing master that is at < 90% capacity
  - Inserts into service discovery system
- Massively better than CDB process — huge win for DBA time

XDB v1 shortcomings

- Asynchronous creation is brittle, slow, and not automation-friendly
- Only shared hosting
- Allocation logic too naive
- No API or centralized control
  - Each system (some PHP, some Python) directly manipulated XDB's metadata tables
  - No sane way to create a large sharded deployment
- No self-service way to drop unneeded databases
- No easy way for DBAs to add capacity

XDB v2

- Major refactor / iterative rewrite, started in summer 2014

- Goals:
  - More self-service offerings = engineers can move faster
  - Reduce on-call burden for MySQL Infrastructure team
  - Better user experience, stability, resiliency
  - Handle sharded deployments without new hostname schemes

Design and Implementation

XDB software components

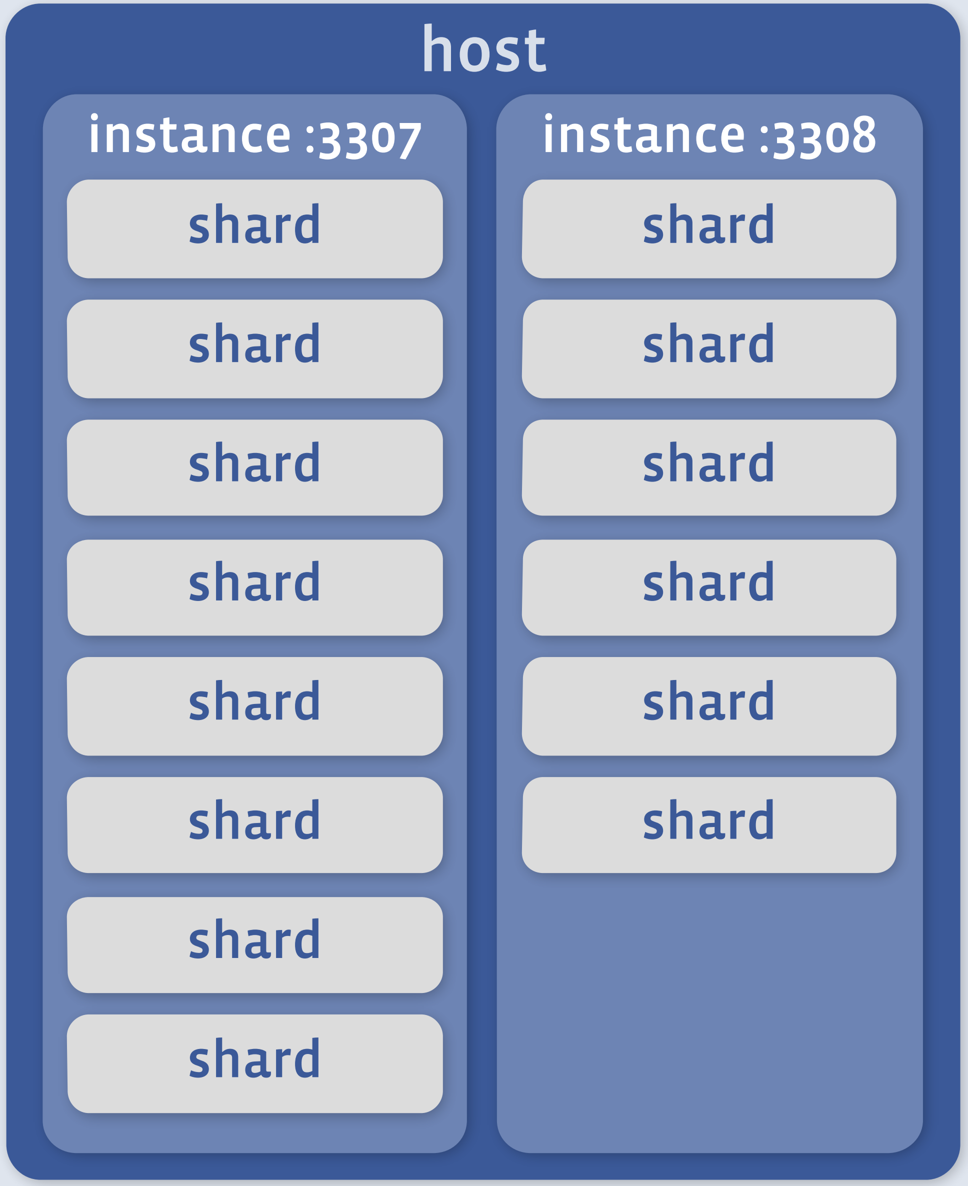
- Centralized service, written in Python
  - Real-time Thrift API
  - Background threads for periodic tasks
  - Multiple copies running, for HA
- Web UI
  - Written in PHP / Hack
  - Interacts *only* with the API — doesn't touch underlying data directly
- Agent on each XDB database host
  - Track size/growth metadata

XDB data components

- Metadata store
  - MySQL DB with sizes, ownership, etc tracking all resources managed by XDB
- Service discovery (SMC)
  - All Facebook systems use this to map service names to host:port
- Timeseries data (ODS)
  - Obtain historical sizes/growth per table, shard, replica set

XDB host layout

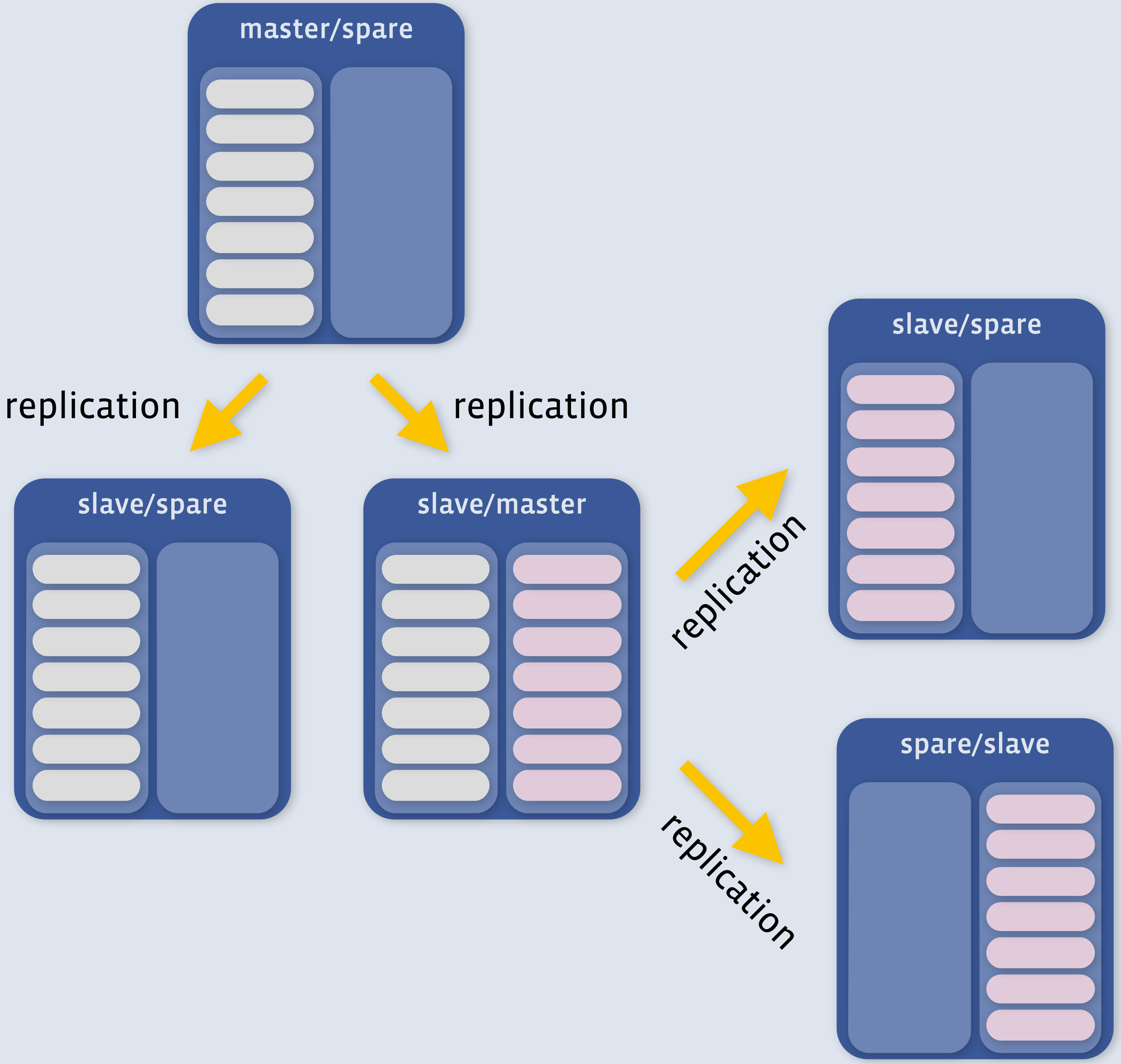
- Each host has N MySQL instances
  - Typically N=2, but we now support other values
- Each MySQL instance can have many database schemas
- We call each database schema a *shard*, even if it is a totally independent data set (functional partition)



Replication topology

- A *replica set* consists of a master instance, plus some number of replicas

- Each XDB replica set is either *shared* or *dedicated*, with respect to hosting shards from multiple owners



XDB API: shard endpoints

- Create shard
- Update shard metadata
- List summary information on many shards
- Get extended information about one shard
- List tables in a shard
- Queue a shard for deletion
- Revert a prior deletion request
- Generate numbered shard names for a sharded data set

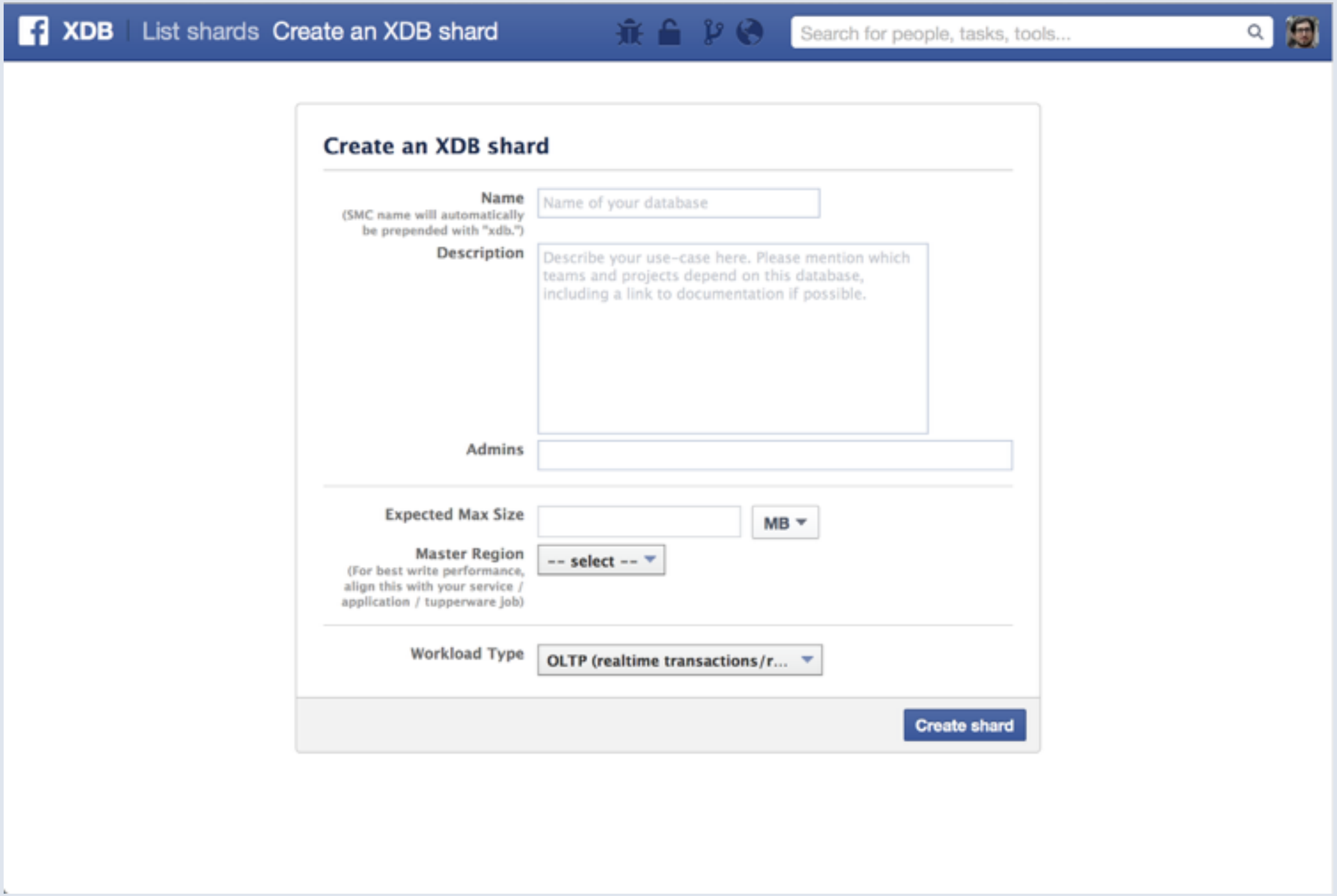
XDB API: replica set endpoints

- Create replica set
- Delete empty replica set
- List replica sets
- Find a shared replica set capable of holding a given shard or size

Web UI

- List shards (yours / someone else’s / all)
  - name, creator, table count, total size, links to graph dashboards
- Create shard
- View and edit metadata for a shard
- List tables (along with sizes) in a shard
- Delete shard
- Revert prior deletion request

Web UI: create shard



Create an XDB shard

Name
(SMC name will automatically be prepended with "xdb.")
Name of your database

Description
Describe your use-case here. Please mention which teams and projects depend on this database, including a link to documentation if possible.

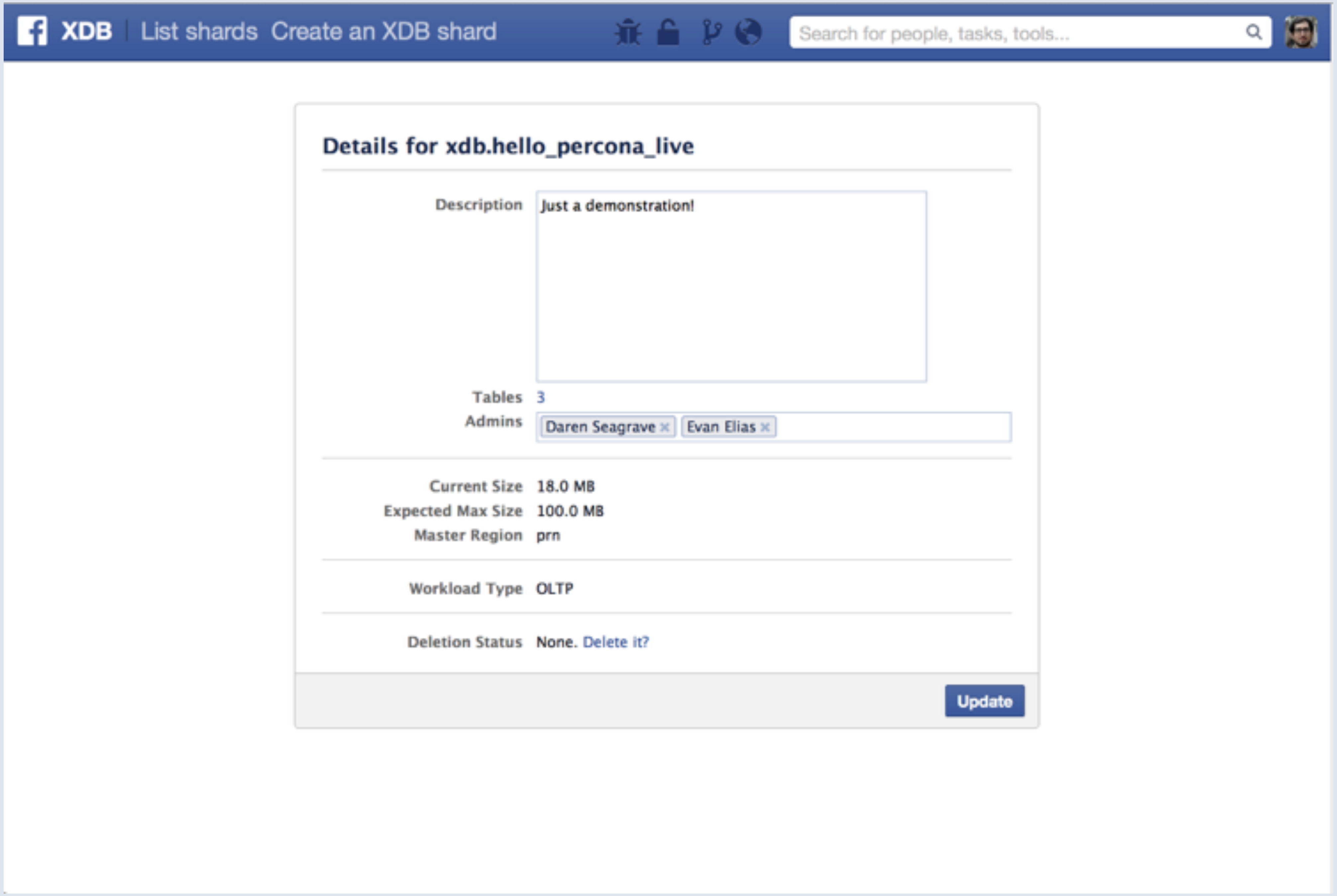
Admins

Expected Max Size: MB

Master Region
(For best write performance, align this with your service / application / tupperware job)
-- select --

Workload Type: OLTP (realtime transactions/r...

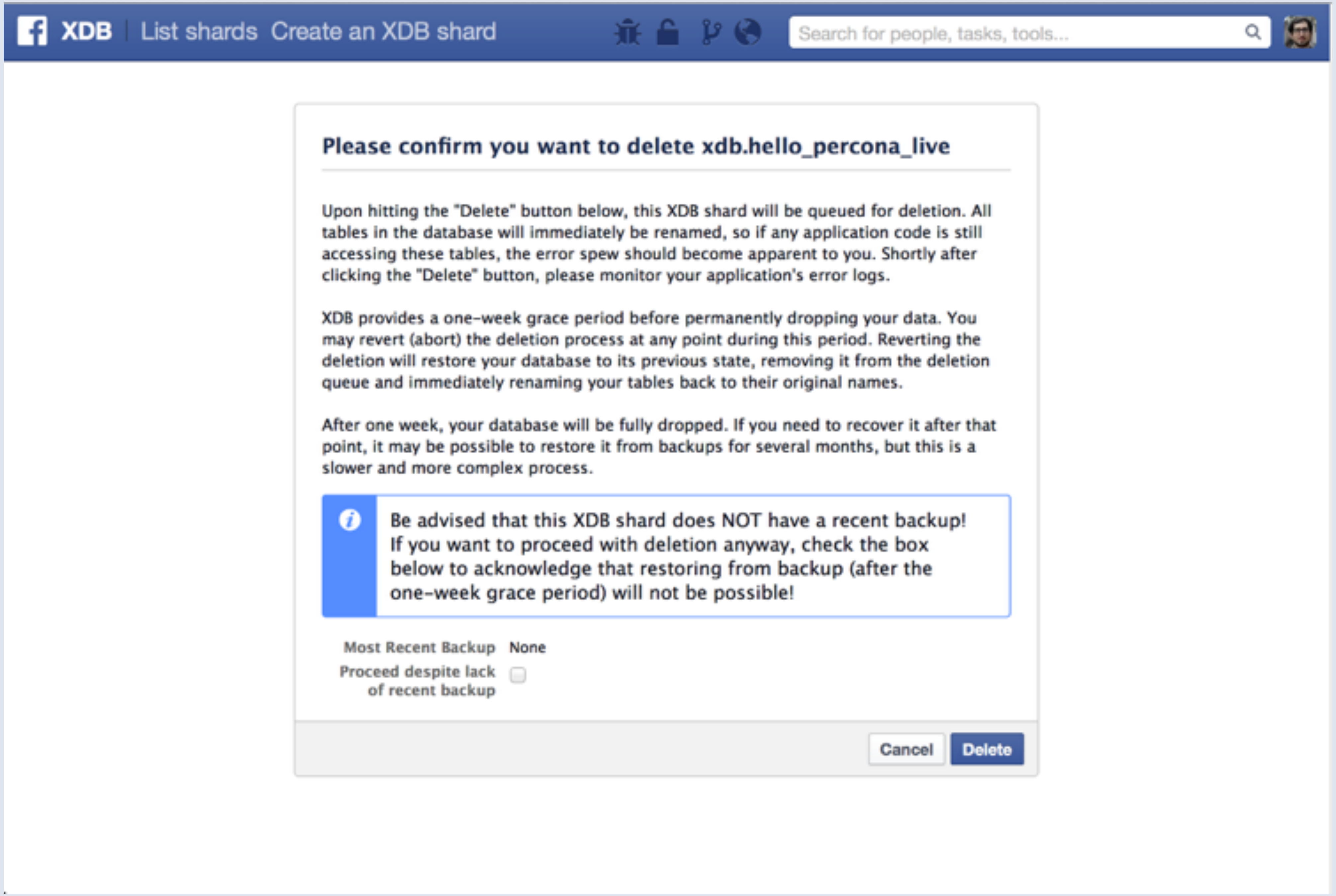
Web UI: view and edit shard metadata



Details for xdb.hello_percona_live

Description	Just a demonstration!
Tables	3
Admins	Daren Seagrave × Evan Elias ×
Current Size	18.0 MB
Expected Max Size	100.0 MB
Master Region	prn
Workload Type	OLTP
Deletion Status	None. Delete it?

Web UI: delete shard



Please confirm you want to delete xdb.hello_percona_live

Upon hitting the "Delete" button below, this XDB shard will be queued for deletion. All tables in the database will immediately be renamed, so if any application code is still accessing these tables, the error spew should become apparent to you. Shortly after clicking the "Delete" button, please monitor your application's error logs.

XDB provides a one-week grace period before permanently dropping your data. You may revert (abort) the deletion process at any point during this period. Reverting the deletion will restore your database to its previous state, removing it from the deletion queue and immediately renaming your tables back to their original names.

After one week, your database will be fully dropped. If you need to recover it after that point, it may be possible to restore it from backups for several months, but this is a slower and more complex process.

Be advised that this XDB shard does NOT have a recent backup! If you want to proceed with deletion anyway, check the box below to acknowledge that restoring from backup (after the one-week grace period) will not be possible!

Most Recent Backup None

Proceed despite lack of recent backup □

Allocation logic

- How to assign new shards to shared replica sets?
  - Too many shards: risk of filling disk, excessive I/O, replication lag
  - Too few shards: waste of hardware
  - Can move shards later, but not light-weight

- Current logic
  - Skip replica sets where actual size is over 50% of disk space
  - Skip replica sets with too many shards
  - Down-weight user-supplied size info over time

Capacity management

- Periodic server thread checks available shared capacity per region

- Create new shared replica set if insufficient capacity
  - Escalate failures to a human
  - Intentionally rate-limited

- Spare instance pool is maintained by non-XDB-specific automation
  - Balance spares between tiers / hostname schemes

Bad neighbors

- Instance-level problems
  - Replication lag
  - Purge lag
  - Too many connections
  - Spiky workloads

- Host-level problems
  - Full disk
  - Resource saturation (network, i/o, cpu)

Dedicated replica sets

- Allow whitelisted teams to “own” replica sets
- Shards may only be placed here deliberately by owner teams
- Supports different levels of instance density per host
  - 1 instance per host, for workloads requiring full isolation
  - 8 instances per host, for smaller data sets
  - 2 instances per host, for everyone else

Lessons Learned

Managing support burden

- XDB creation volume is skyrocketing
  - More bad neighbors
  - More support questions
  - More dedicated resource requests

- Good docs and FAQ are essential
- Answer generic questions in public
- Encourage use of the official MySQL manual
- Teach MySQL best practices at new engineer onboarding

Conflicting sources of truth

- Discrepancies between key data stores, re: which databases exist and where
  - XDB metadata
  - Service discovery (SMC)
  - Each replica set's master
- Pesky engineers may be creating/dropping things out-of-band
  - Catch this via automated monitoring
- Creation and deletion processes must handle failures gracefully

Database deletion flow

- Must be low-friction and self-service
  - … but also needs effective safeguards!

- Confirm that recent backup exists before proceeding

- Don’t drop the database right away
  - Tables are immediately renamed, but not dropped
  - One-week grace period before actual drop occurs
  - Self-service revert restores table names

- User can override the backup check or grace period, but not by accident

Resource management and quotas

- Don’t take creator-supplied size expectations at face value
- Everyone wants dedicated resources
  - Most don’t actually need it
  - Create new databases on shared replica sets by default
  - Move one-off DBs to dedicated replica sets only if/when justified
  - Sharded data sets should go to dedicated resources from the start
- Tools to track usage per team
- Automation to identify abandoned databases

Sharding support

- Generic foundation for sharding at the allocation/provisioning level

- Automation to *move* shards is simpler than automation to *split* them
  - Prefer many small shards to fewer huge shards

- Offering generic sharding support at the *application* level is a separate, much more complex can of worms

Future Directions

Shard migration automation

Automatic migration opens many possibilities

- Self-service master region change requests
- Quota-triggered shard moves
- Bad neighbor isolation
- Offload shards from oversubscribed replica sets

User / grant management

- Create one user per database, and have application use it automatically

- Powerful in combination with information_schema.user_statistics (FB patch in WebScaleSQL or Percona Server)

- Lock down default set of grants

Good neighbor enforcement

- Integrate with company systems to auto-task an appropriate on-call rotation
  - Replication lag offenders
  - Excessive workloads via user stats

- Enforce size quotas
  - Self-service interface for requesting increases, trigger shard move if needed
  - Task owners at soft limit, revoke write privileges after hard limit

- Experiment with cgroups

Dedicated replica sets

Run 32x instance density and make dedicated default?

Dedicated replica sets

Or just move shards when they hit a certain size?

DEDICATED REPLICA SETS FOR SOME
MINIATURE AMERICAN FLAGS FOR OTHERS

Open source efforts

- Many dependencies will need to be abstracted away
  - Hardware asset tracker
  - Hardware provisioning system
  - Service discovery
  - Employee / team directory
  - Timeseries data
  - Alerting / monitoring
  - Python service framework, packaging, containerization
  - MySQL automation systems and libraries
- Many permutations of MySQL branch, MySQL version, Linux distributions, Python versions

Questions?

facebook