Backing up Wikipedia Databases

Jaime Crespo & Manuel Aróstegui

WIKIMEDIA
FOUNDATION

Data Persistence

Subteam,
Site Reliability Engineering

Contents

1) Existing Environment
2) Design
3) Implementation Details
4) Results
5) Planned Work & Lessons Learned

*What we are going to mention in this talk is **our experience** and our learnings - this is what worked for our environment at the time. Your needs and requirements may be different.*

WIKIMEDIA
FOUNDATION

Existing Environment

WIKIMEDIA
FOUNDATION

Why backups?

- We use RAID 10, read replicas, multiple DCs for High Availability
- Public XMLDumps
- But what about...
  - Checking a concrete record back in time?
  - Application bug changing data on all servers?
  - Operator mistake?
  - Abuse of external user?

WIKIMEDIA
FOUNDATION

Database context (mid-2019)

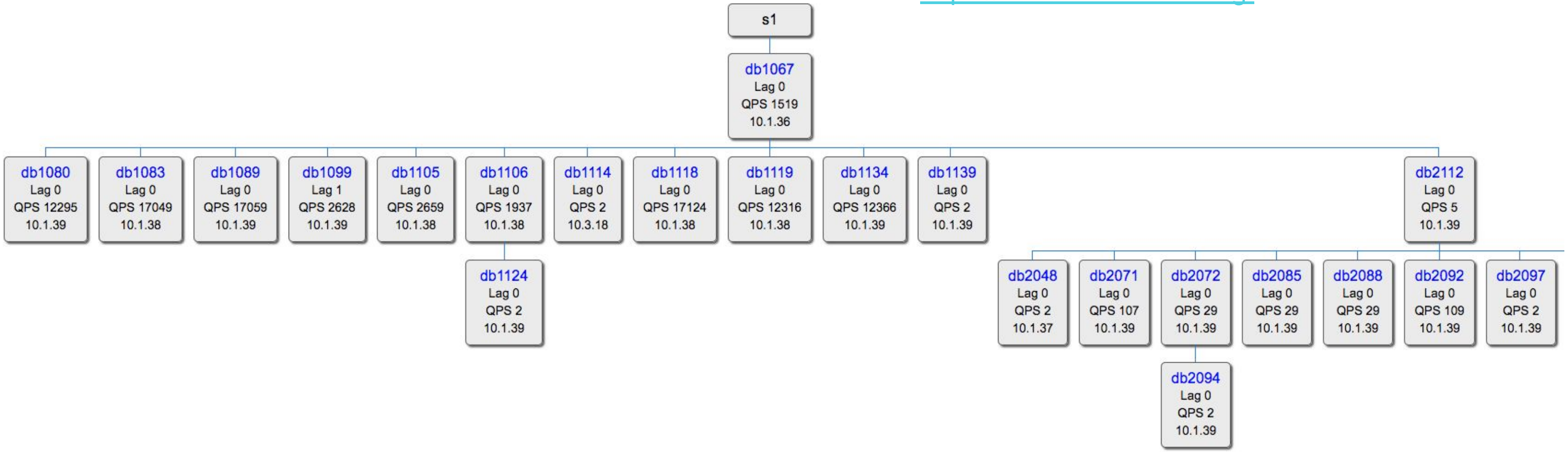
- Aside from the English Wikipedia, 800 other wikis in 300 languages
- ~550 TB of data of **relational data** over 24+ replica groups
- ~60 TB of those is **unique** data, of those:
  - ~24TB of compressed mediawiki insert-only **content**
  - The rest is **metadata**, local content, misc services, disk cache, analytics, backups, ...

WIKIMEDIA FOUNDATION

Brief description of our environment

- Self hosted on bare metal
- Only open source software
- 2 DCs holding data - at the moment, one active and one passive
- Normal replication topology with several intermediate masters

https://dbtree.wikimedia.org/


s1
db1067 Lag 0 QPS 1519 10.1.36
db1080 Lag 0 QPS 12295 10.1.39
db1083 Lag 0 QPS 17049 10.1.38
db1089 Lag 0 QPS 17059 10.1.39
db1099 Lag 1 QPS 2628 10.1.39
db1105 Lag 0 QPS 2659 10.1.38
db1106 Lag 0 QPS 1937 10.1.38
db1114 Lag 0 QPS 2 10.3.18
db1118 Lag 0 QPS 17124 10.1.38
db1119 Lag 0 QPS 12316 10.1.38
db1134 Lag 0 QPS 12366 10.1.39
db1139 Lag 0 QPS 2 10.1.39
db2112 Lag 0 QPS 5 10.1.39
db1124 Lag 0 QPS 2 10.1.39
db2048 Lag 0 QPS 2 10.1.37
db2071 Lag 0 QPS 107 10.1.39
db2072 Lag 0 QPS 29 10.1.39
db2085 Lag 0 QPS 29 10.1.39
db2088 Lag 0 QPS 29 10.1.39
db2092 Lag 0 QPS 109 10.1.39
db2097 Lag 0 QPS 2 10.1.39
db2094 Lag 0 QPS 2 10.1.39

We were using only mysqldump

- Coordinates were not being saved
- No good monitoring in place, failures could be missed
- Single file with the whole database (100GB+ compressed file)
- Slow to backup and recover

Backup hosts were *different* from production

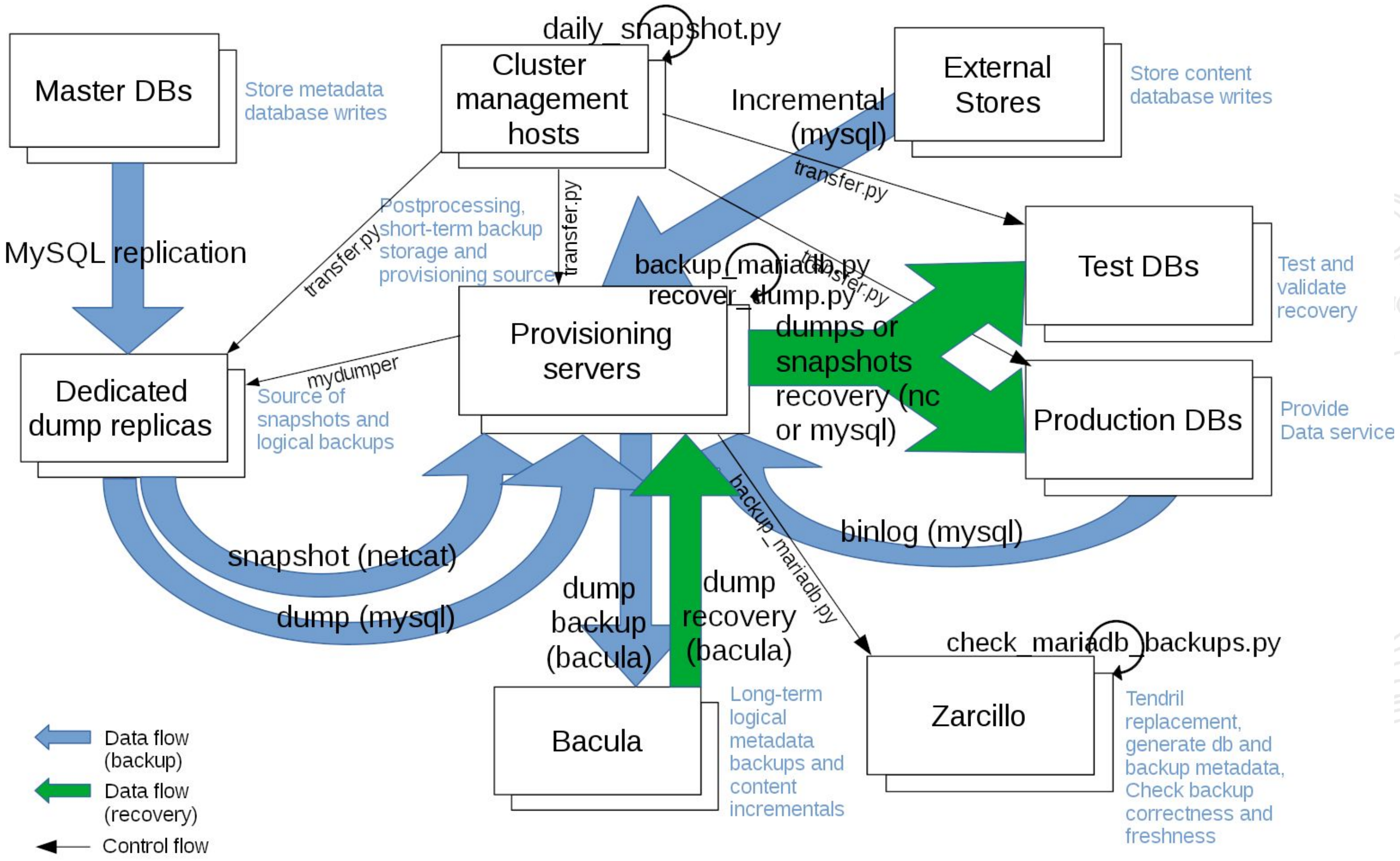
- Used TokuDB for compression and to maximize disk space resources whilst production runs InnoDB
- Running multisource replication
    - It could not be used for an automatic provisioning system

Hardware needed to be refreshed

- Hardware was old, and prone to suffer issues
- More disk and IOPS needed
- Lack of proper DC redundancy

Design

WIKIMEDIA
FOUNDATION



Master DBs

Store metadata database writes

MySQL replication

Dedicated dump replicas

Source of snapshots and logical backups

daily_snapshot.py

Cluster management hosts

Postprocessing, short-term backup storage and provisioning source

transfer.py

transfer.py

mydumper

Provisioning servers

backup_mariadb.py

recover_dump.py

External Stores

Store content database writes

Incremental (mysql)

transfer.py

transfer.py

Test DBs

Test and validate recovery

dumps or snapshots recovery (nc or mysql)

Production DBs

Provide Data service

binlog (mysql)

snapshot (netcat)

dump (mysql)

dump backup (bacula)

dump recovery (bacula)

backup_mariadb.py

Bacula

Long-term logical metadata backups and content incrementals

check_mariadb_backups.py

Zarcillo

Tendril replacement, generate db and backup metadata, Check backup correctness and freshness

Data flow (backup)

Data flow (recovery)

Control flow

New backup system requirements

- For simplicity, we started with **full backups only**
- Cross-dc redundancy
- Scale over several instances for flexibility and performance
- Aiming for 30 minute TTR
- Row granularity
- 90 day retention
- Fully automated creation and recovery

WIKIMEDIA FOUNDATION

Storage

- **Bacula** is used as cold, long term storage, primarily because it's the tool shared with the rest of the infrastructure backups
- Data deduplication was considered but no good solution that fit our needs
    - Space saving at application side, **InnoDB compression and parallel gzip** were considered good enough

WIKIMEDIA FOUNDATION

Logical Backups vs Snapshots

- Logical backups provide great flexibility, small size, good compatibility, and less prone to data-corruption
- Logical backups are fast to generate but slow to recover
- Snapshots are faster to recover, but take more space and are less flexible

- We decided to do ***both***!
  - Snapshots will be used for full disaster recovery, and provisioning
  - Dumps to be used for long term archival and small-scale recoveries

* Image from Old El Paso commercial own by General Mills, Inc
Use under fair use

mysqlpump vs mysqldump vs mydumper

- **mysqlpump** discarded early due to incompatibilities (mariadb GTID)
- **mysqldump** is the standard tool, but required hacks to make it parallel, too slow to recover
- **mydumper** has good MariaDB support, integrated compression, a flexible dump format and is fast and multithreaded

Our choice

WIKIMEDIA
FOUNDATION

LVM vs Xtrabackup vs Cold Backup vs Delayed slave (I)

- LVM
  - Disk-efficient (especially for multiple copies)
  - Fast to recover if kept locally
  - Requires dedicated partition
  - Needs to be done locally and then moved remotely to be stored

LVM vs Xtrabackup vs Cold Backup vs Delayed slave (II)

- xtrabackup*
  - --prepare
  - Can be piped through network
  - More resources on generation
  - xtrabackup works at innodb level and lvm at filesystem level

Our choice

* We use mariabackup as xtrabackup isn’t supported for MariaDB

WIKIMEDIA
FOUNDATION

LVM vs Xtrabackup vs Cold Backup vs Delayed slave (III)

- Cold backups
  - Requires stopping MySQL
  - Consistent on a file level wise
  - Combined with LVM can give good results

LVM vs Xtrabackup vs Cold Backup vs Delayed slave (IV)

- Delayed slave
  - Faster recovery: for a given time period
  - We used to have it and had bad experiences
  - Not great for provisioning **new** hosts

Provisioning & testing

- Backups will not be just tested on a lab
    - New hosts will be provisioned from the existing backups

- Dedicated backup testing hosts:
    - Replication will automatically validate most “live data”
    - We already have production row-by-row data comparison

Implementation Details

WIKIMEDIA
FOUNDATION

Hardware

Per Datacenter

- 5 dedicated replicas with 2 mysql instances each (consolidation)
- 2 provisioning hosts (SSDs + HDs)
- 1 new bacula host
    - 1 disk array dedicated for databases
- 1 test host (same spec as regular replicas)

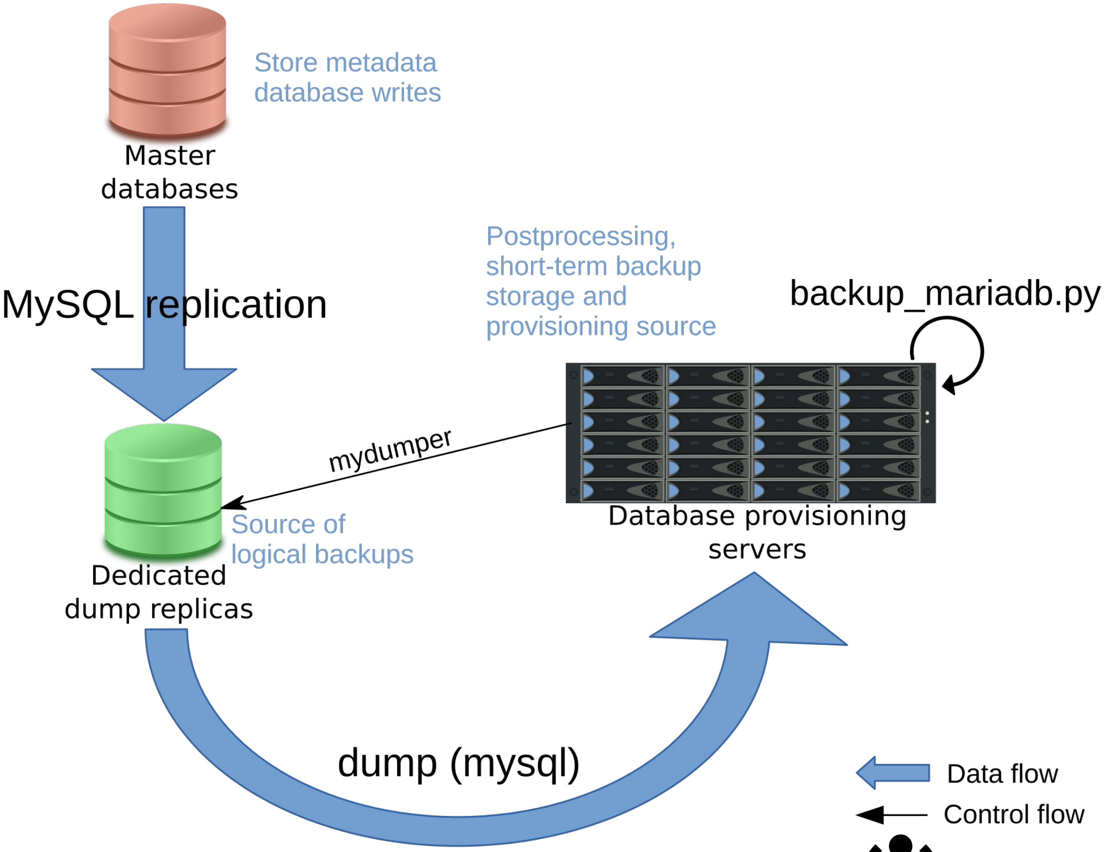
Development

- Python 3 for gluing underlying applications
- WMF-specific development and deployment is done though puppet so not a portable “product”
  - WMFMariaDBpy: https://phabricator.wikimedia.org/diffusion/OSMD/
  - Our Puppet: https://phabricator.wikimedia.org/source/operations-puppet/
- Very easy to add new backup methods

```
class NullBackup:

    config = dict()

    def __init__(self, config, backup):
    """
    Initialize commands
    """
    self.config = config
    self.backup = backup
    self.logger = backup.logger

    def get_backup_cmd(self, backup_dir):
    """
    Return list with binary and options to execute to generate a new backup at backup_dir
    """
    return '/bin/true'

    def get_prepare_cmd(self, backup_dir):
    """
    Return list with binary and options to execute to prepare an existing backup. Return
    none if prepare is not necessary (nothing will be executed in that case).
    """
    return ''
```

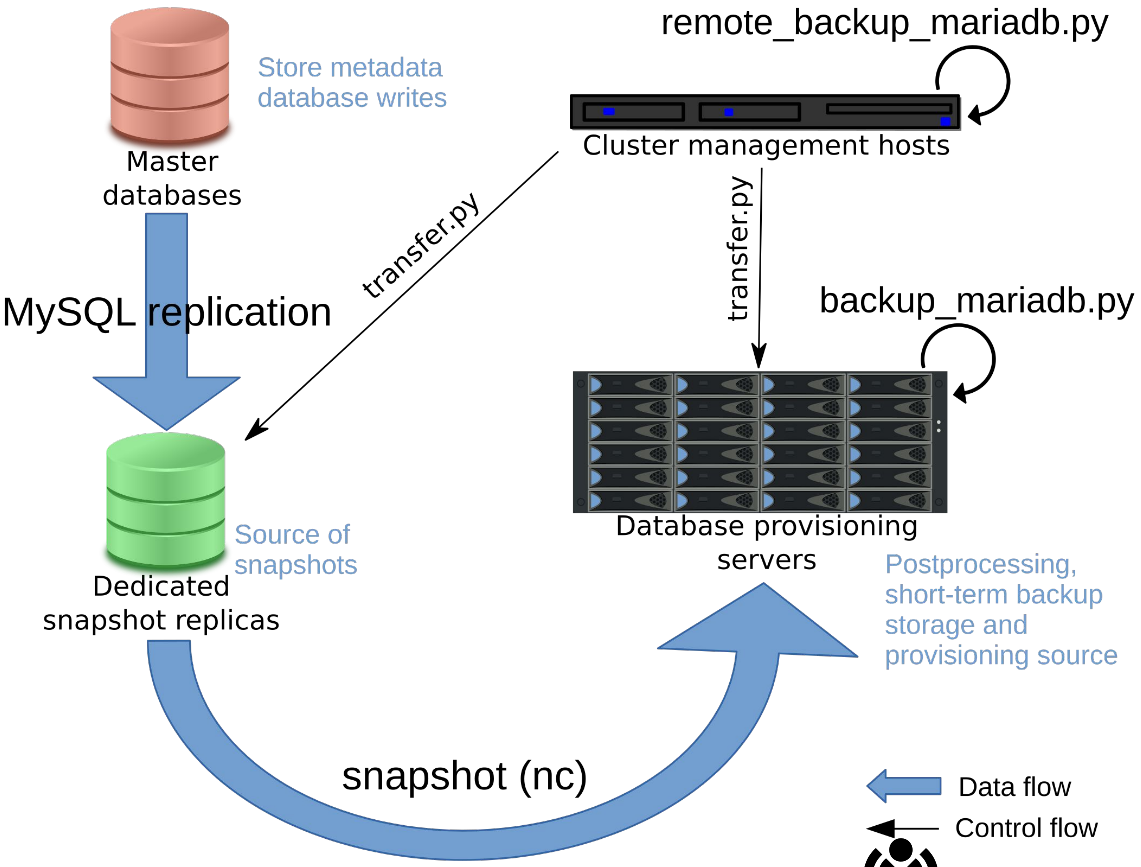
Configuration

```
root@cumin1001:~$ cat /etc/mysql/backups.cnf
type: snapshot
rotate: True
retention: 4
compress: True
archive: False
statistics:
  host: db1115.eqiad.wmnet
  database: zarcillo
sections:
  s1:
    host: db1139.eqiad.wmnet
    port: 3311
    destination: dbprov1002.eqiad.wmnet
    stop_slave: True
    order: 2
  s2:
    host: db1095.eqiad.wmnet
    port: 3312
    destination: dbprov1002.eqiad.wmnet
    order: 4
```


Store metadata database writes
Master databases
MySQL replication
Postprocessing, short-term backup storage and provisioning source
backup_mariadb.py
mydumper
Source of logical backups
Dedicated dump replicas
Database provisioning servers
dump (mysql)
Data flow
Control flow

- Backups are taken from *dedicated replicas* for convenience
- A cron job starts the backup on the provisioning servers, running mydumper
- Several threads used to dump in *parallel*, result is automatically *compressed* per table

WIKIMEDIA FOUNDATION

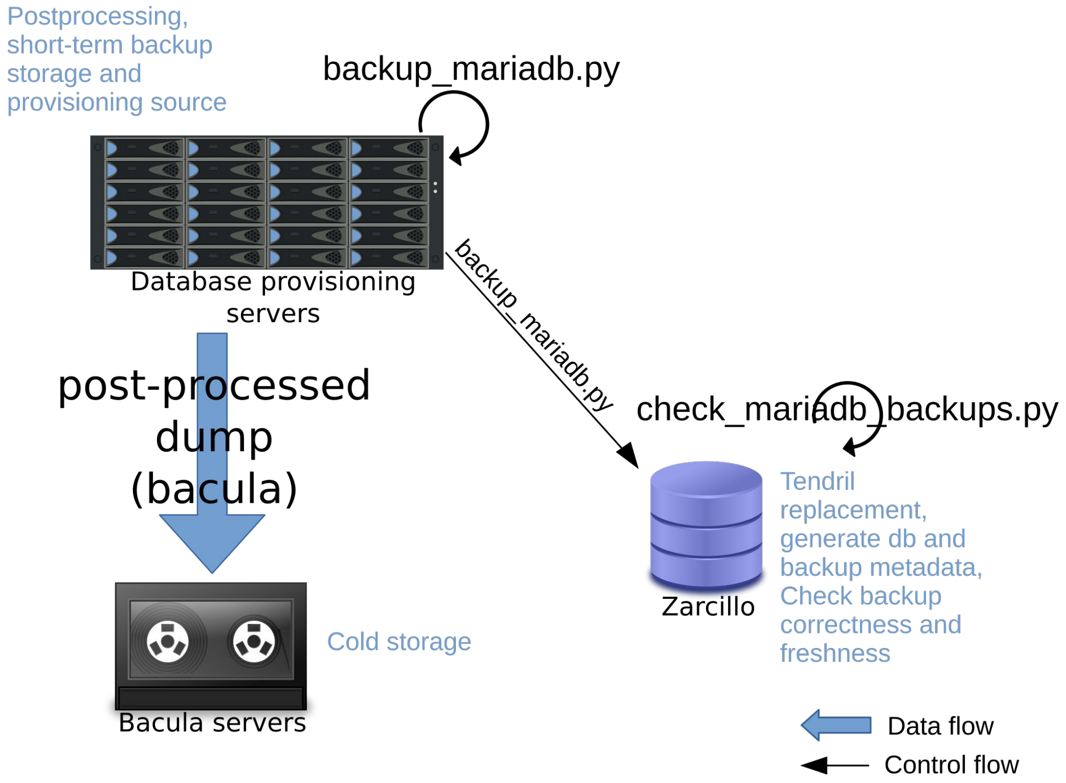


WIKIMEDIA FOUNDATION

- Snapshots have to be coordinated remotely as it requires file transfer
- Xtrabackup installed on the source db is used to prevent incompatibilities
- Content is piped directly through network to avoid local disk write step

```
root@cumin1001:~$ transfer.py  --help
usage: transfer.py [-h] [--port PORT] [--type {file,xtrabackup,decompress}]
                   [--compress | --no-compress] [--encrypt | --no-encrypt]
                   [--checksum | --no-checksum] [--stop-slave]
                   source target [target ...]
positional arguments:
  source [...]
  target [...]
optional arguments:
  -h, --help            show this help message and exit
  --port PORT           Port used for netcat listening on the source. By default, 4444,
but it must be changed if more
                        than 1 transfer to the same host happen at the same time, or the
second copy will fail top open
                        the socket again. This port has its firewall disabled during
transfer automatically with an extra
                        iptables rule.
  --type {file,xtrabackup,decompress}
                        File: regular file or directory recursive copy
                        xtrabackup: runs mariabackup on source
  --compress            Use pigz to compress stream using gzip format (ignored on
decompress mode)
  --no-compress         Do not use compression on streaming
  --encrypt             Enable compression using openssl and algorithm chacha20 (default)
  --no-encrypt          Disable compression- send data using an unencrypted stream
  --checksum            Generate a checksum of files before transmission which will be
used for checking integrity after
                        transfer finishes. It only works for file transfers, as there is
no good way to checksum a running
                        mysql instance or a tar.gz
  --no-checksum         Disable checksums
  --stop-slave          Only relevant if on xtrabackup mode: attempt to stop slave on the
mysql instance before running
                        xtrabackup, and start slave after it completes to try to speed up
backup by preventing many changes
                        queued on the xtrabackup_log. By default, it doesn't try to stop
replication.
```

- A wrapper utility to transfer files, precompressed tarballs and piping xtrabackup output


Postprocessing, short-term backup storage and provisioning source
backup_mariadb.py
Database provisioning servers
backup_mariadb.py
post-processed dump (bacula)
Bacula servers
Cold storage
check_mariadb_backups.py
Zarcillo
Tendril replacement, generate db and backup metadata, Check backup correctness and freshness
Data flow
Control flow

- Postprocessing both types of backups involves:
  - `--prepare`
  - consolidation of files
  - metadata gathering
  - compression
  - validation
- Main monitoring is done from the backup metadata database

WIKIMEDIA FOUNDATION

```
root@dbprov2001:/srv$ tree
├── backups
│   ├── dumps
│   │   ├── archive
...
│   │   ├── latest
│   │   │   ├── dump.m2.2019-09-10--00-00-01
│   │   │   │   ├── debmonitor.auth_group_permissions-schema.sql.gz
│   │   │   │   ├── debmonitor.auth_group-schema.sql.gz
...
│   │   │   │   ├── wikidatawiki.wbt_item_terms.00000.sql.gz
│   │   │   │   ├── wikidatawiki.wbt_item_terms.00001.sql.gz
│   │   │   │   ├── wikidatawiki.wbt_item_terms.00002.sql.gz
│   │   │   ├── dump.x1.2019-09-10--00-00-01
│   │   │   │   ├── 10wikipedia.gz.tar
│   │   │   │   ├── aawikibooks.gz.tar
│   │   │   │   ├── aawiki.gz.tar
│   │   │   │   ├── aawiktionary.gz.tar
│   │   │   │   ├── abwiki.gz.tar
│   │   └── ongoing
│   └── snapshots
│       ├── archive
│       │   ├── snapshot.m5.2019-05-07--20-00-02.tar.gz
│       │   ├── snapshot.s4.2019-09-24--21-45-51.tar.gz
│       │   ├── snapshot.s5.2019-09-25--01-08-39.tar.gz
│       │   ├── snapshot.s6.2019-09-25--02-55-21.tar.gz
│       │   ├── snapshot.s8.2019-09-24--19-00-01.tar.gz
│       │   └── snapshot.x1.2019-09-25--06-52-57.tar.gz
│       ├── latest
│       └── ongoing
```

Large tables are split into several files

Small databases are consolidated into one file

At least 2 (normally 3) copies are kept of each backup from different timestamps

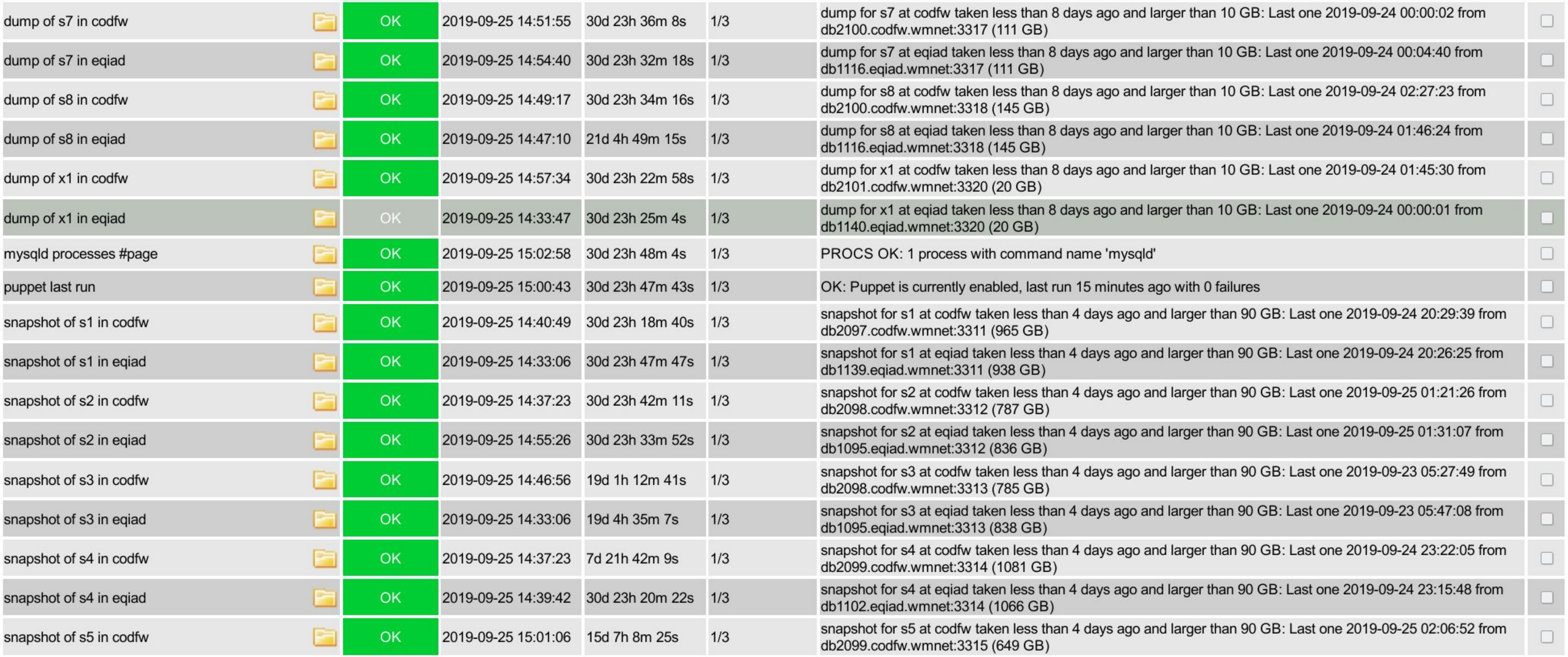
Backup validation & monitoring

- Backup failure cannot be 100% avoided
- Once backups are done, a few checks are performed:
  - Did the process exit with an error?
  - Any errors logged?
  - Are expected final files present?
- Alerting is based on metadata heuristics:
  - A correct backup for the section, type and datacenter exists?
  - With a size larger than *X* bytes?
  - Newer than *X* days?

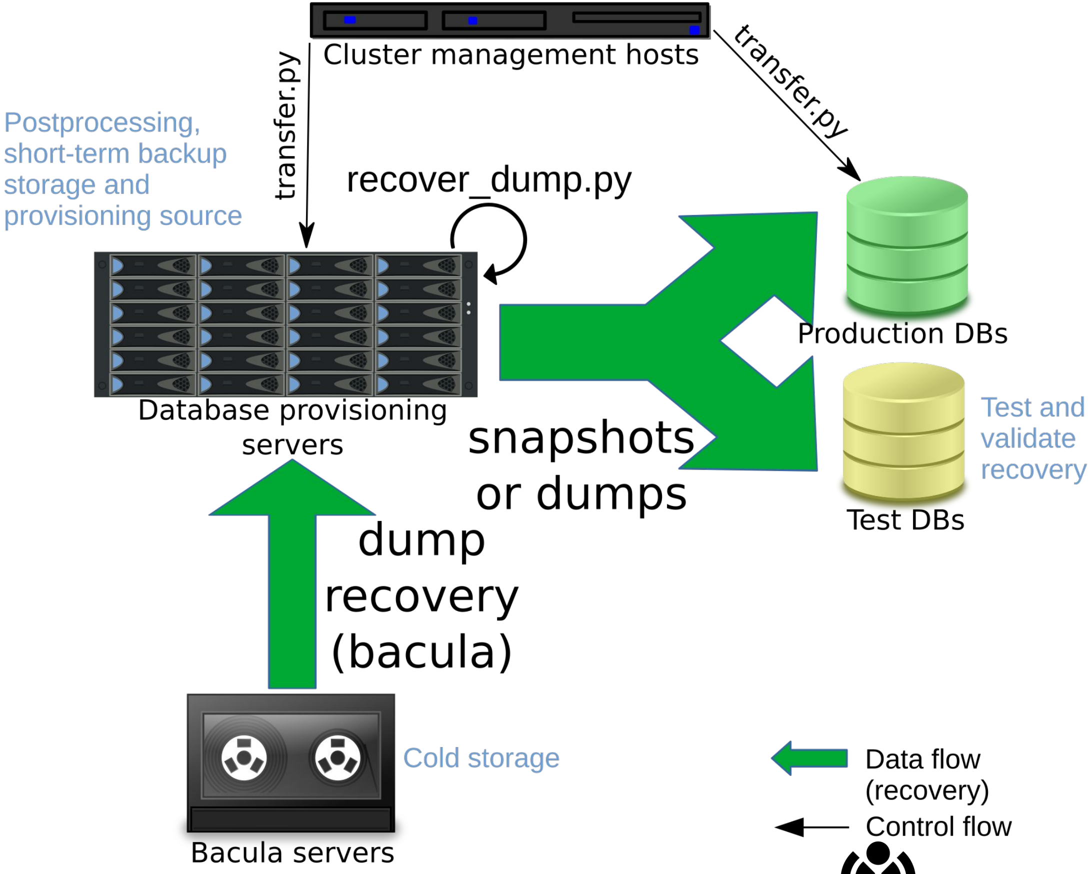
WIKIMEDIA
FOUNDATION

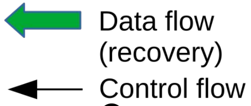
```
db1115[zarcillo]> SELECT * FROM backups WHERE [..]\G
******************** 1. row ********************
        id: 2921
      name: dump.s1.2019-09-24--03-27-38
    status: finished
    source: db1139.eqiad.wmnet:3311
      host: dbprov1002.eqiad.wmnet
      type: dump
   section: s1
start_date: 2019-09-24 03:27:38
  end_date: 2019-09-24 05:00:01
total_size: 159537777604
******************** 2. row ********************
        id: 1310
      name: snapshot.s1.2019-05-09--20-38-02
    status: failed
    source: db2097.codfw.wmnet:3311
      host: dbprov2002.codfw.wmnet
      type: snapshot
   section: s1
start_date: 2019-05-09 22:10:53
  end_date: NULL
total_size: NULL

2 rows in set (0.00 sec)
```

```
db1115[zarcillo]> SELECT * FROM backup_files WHERE [..]
*********************** 1. row ***********************
       backup_id: 2930
       file_path: enwiki
       file_name: recentchanges.frm
            size: 8412
       file_date: 2019-09-24 20:26:18
backup_object_id: NULL
*********************** 2. row ***********************
       backup_id: 2930
       file_path: enwiki
       file_name: recentchanges.ibd
            size: 3573547008
       file_date: 2019-09-24 20:35:25
backup_object_id: NULL
*********************** 3. row ***********************
       backup_id: 2930
       file_path: enwiki
       file_name: revision.frm
            size: 4926
       file_date: 2019-09-24 20:26:21
backup_object_id: NULL
*********************** 4. row ***********************
       backup_id: 2930
       file_path: enwiki
       file_name: revision.ibd
            size: 186025771008
       file_date: 2019-09-24 20:35:25
backup_object_id: NULL
```



dump of s7 in codfw	📁	OK	2019-09-25 14:51:55	30d 23h 36m 8s	1/3	dump for s7 at codfw taken less than 8 days ago and larger than 10 GB: Last one 2019-09-24 00:00:02 from db2100.codfw.wmnet:3317 (111 GB)	□
dump of s7 in eqiad	📁	OK	2019-09-25 14:54:40	30d 23h 32m 18s	1/3	dump for s7 at eqiad taken less than 8 days ago and larger than 10 GB: Last one 2019-09-24 00:04:40 from db1116.eqiad.wmnet:3317 (111 GB)	□
dump of s8 in codfw	📁	OK	2019-09-25 14:49:17	30d 23h 34m 16s	1/3	dump for s8 at codfw taken less than 8 days ago and larger than 10 GB: Last one 2019-09-24 02:27:23 from db2100.codfw.wmnet:3318 (145 GB)	□
dump of s8 in eqiad	📁	OK	2019-09-25 14:47:10	21d 4h 49m 15s	1/3	dump for s8 at eqiad taken less than 8 days ago and larger than 10 GB: Last one 2019-09-24 01:46:24 from db1116.eqiad.wmnet:3318 (145 GB)	□
dump of x1 in codfw	📁	OK	2019-09-25 14:57:34	30d 23h 22m 58s	1/3	dump for x1 at codfw taken less than 8 days ago and larger than 10 GB: Last one 2019-09-24 01:45:30 from db2101.codfw.wmnet:3320 (20 GB)	□
dump of x1 in eqiad	📁	OK	2019-09-25 14:33:47	30d 23h 25m 4s	1/3	dump for x1 at eqiad taken less than 8 days ago and larger than 10 GB: Last one 2019-09-24 00:00:01 from db1140.eqiad.wmnet:3320 (20 GB)	□
mysqld processes #page	📁	OK	2019-09-25 15:02:58	30d 23h 48m 4s	1/3	PROCS OK: 1 process with command name 'mysqld'	□
puppet last run	📁	OK	2019-09-25 15:00:43	30d 23h 47m 43s	1/3	OK: Puppet is currently enabled, last run 15 minutes ago with 0 failures	□
snapshot of s1 in codfw	📁	OK	2019-09-25 14:40:49	30d 23h 18m 40s	1/3	snapshot for s1 at codfw taken less than 4 days ago and larger than 90 GB: Last one 2019-09-24 20:29:39 from db2097.codfw.wmnet:3311 (965 GB)	□
snapshot of s1 in eqiad	📁	OK	2019-09-25 14:33:06	30d 23h 47m 47s	1/3	snapshot for s1 at eqiad taken less than 4 days ago and larger than 90 GB: Last one 2019-09-24 20:26:25 from db1139.eqiad.wmnet:3311 (938 GB)	□
snapshot of s2 in codfw	📁	OK	2019-09-25 14:37:23	30d 23h 42m 11s	1/3	snapshot for s2 at codfw taken less than 4 days ago and larger than 90 GB: Last one 2019-09-25 01:21:26 from db2098.codfw.wmnet:3312 (787 GB)	□
snapshot of s2 in eqiad	📁	OK	2019-09-25 14:55:26	30d 23h 33m 52s	1/3	snapshot for s2 at eqiad taken less than 4 days ago and larger than 90 GB: Last one 2019-09-25 01:31:07 from db1095.eqiad.wmnet:3312 (836 GB)	□
snapshot of s3 in codfw	📁	OK	2019-09-25 14:46:56	19d 1h 12m 41s	1/3	snapshot for s3 at codfw taken less than 4 days ago and larger than 90 GB: Last one 2019-09-23 05:27:49 from db2098.codfw.wmnet:3313 (785 GB)	□
snapshot of s3 in eqiad	📁	OK	2019-09-25 14:33:06	19d 4h 35m 7s	1/3	snapshot for s3 at eqiad taken less than 4 days ago and larger than 90 GB: Last one 2019-09-23 05:47:08 from db1095.eqiad.wmnet:3313 (838 GB)	□
snapshot of s4 in codfw	📁	OK	2019-09-25 14:37:23	7d 21h 42m 9s	1/3	snapshot for s4 at codfw taken less than 4 days ago and larger than 90 GB: Last one 2019-09-24 23:22:05 from db2099.codfw.wmnet:3314 (1081 GB)	□
snapshot of s4 in eqiad	📁	OK	2019-09-25 14:39:42	30d 23h 20m 22s	1/3	snapshot for s4 at eqiad taken less than 4 days ago and larger than 90 GB: Last one 2019-09-24 23:15:48 from db1102.eqiad.wmnet:3314 (1066 GB)	□
snapshot of s5 in codfw	📁	OK	2019-09-25 15:01:06	15d 7h 8m 25s	1/3	snapshot for s5 at codfw taken less than 4 days ago and larger than 90 GB: Last one 2019-09-25 02:06:52 from db2099.codfw.wmnet:3315 (649 GB)	□



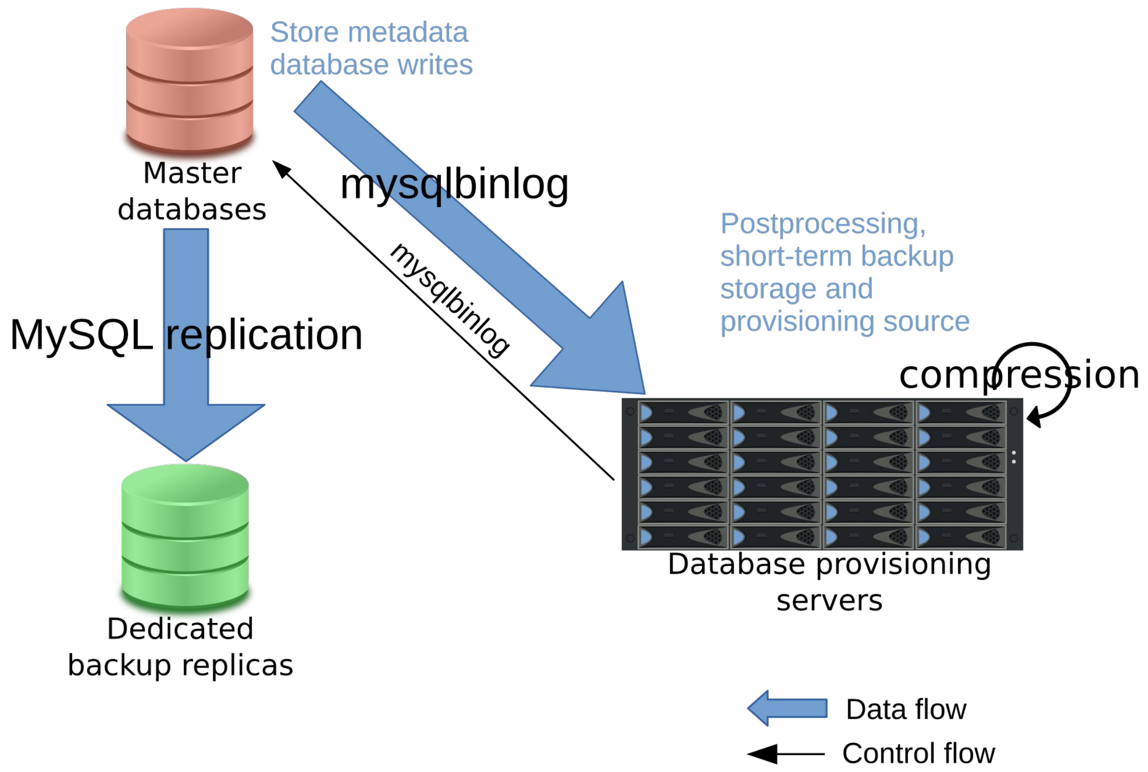


WIKIMEDIA FOUNDATION

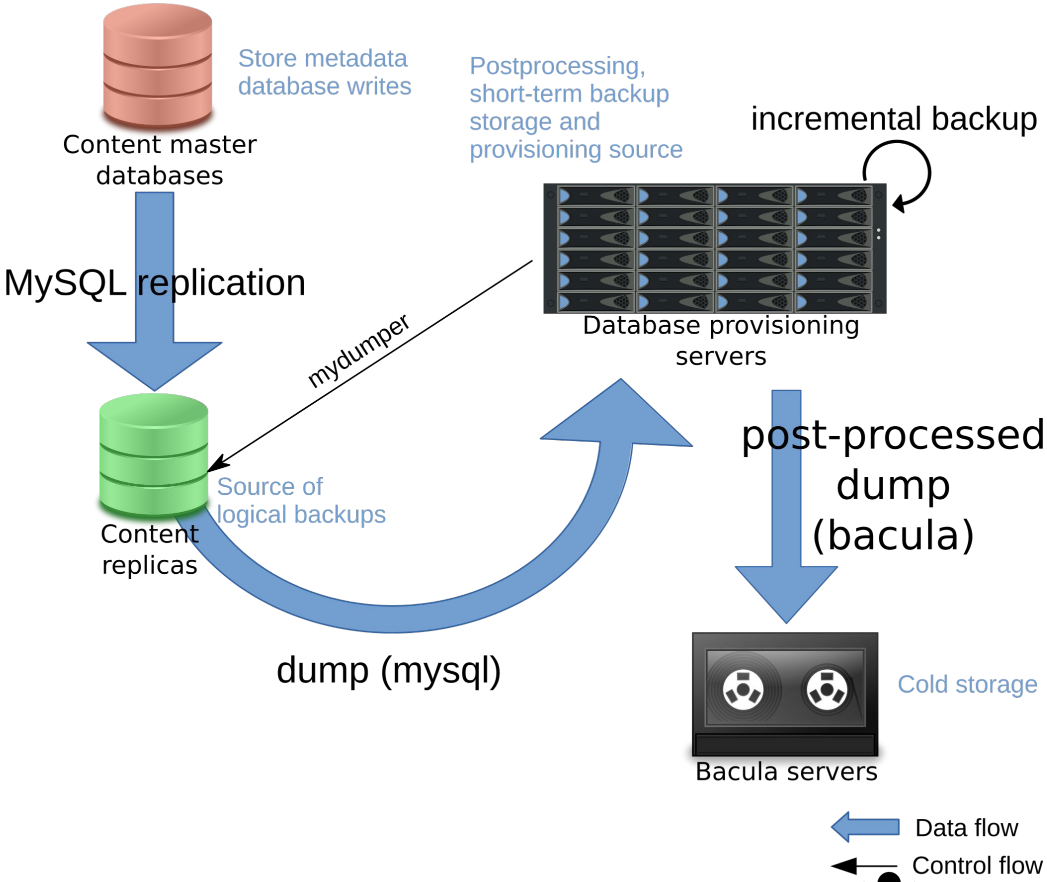
- Regular day-to-day provisioning is done with the exact same workflow
- Recovery can be done from logical backups or snapshots, in both hot and cold storage

```
root@dbprov2002:~$ recover_dump.py --help
usage: recover_dump.py [-h] [--host HOST] [--port PORT]
[--threads THREADS]
                        [--user USER] [--password PASSWORD]
[--socket SOCKET]
                        [--database DATABASE] [--replicate]
                        section

Recover a logical backup

positional arguments:
  section               Section name or absolute path of the
directory to
                        recover("s3",
"/srv/backups/archive/dump.s3.2022-11-12
                        --19-05-35")

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           Host to recover to
  --port PORT           Port to recover to
  --threads THREADS     Maximum number of threads to use for
recovery
  --user USER           User to connect for recovery
  --password PASSWORD  Password to recover
  --socket SOCKET       Socket to recover to
  --database DATABASE  Only recover this database
  --replicate           Enable binlog on import, for imports
to a master that
                        have to be replicated (but makes
load slower).By
                        default, binlog writes are disabled.
```

- A myloader wrapper simplifies the recovery
- .sql.gz files per table are easy to process and recover individually


Store metadata database writes
Master databases
mysqlbinlog
mysqlbinlog
MySQL replication
Postprocessing, short-term backup storage and provisioning source
compression
Dedicated backup replicas
Database provisioning servers
Data flow
Control flow

- Binlogs obtained directly from the master with mysqlbinlog and archived on provisioning servers for point in time recovery
- Not implemented yet


Content master databases
Store metadata database writes
MySQL replication
Content replicas
Source of logical backups
mydumper
dump (mysql)
Postprocessing, short-term backup storage and provisioning source
incremental backup
Database provisioning servers
post-processed dump (bacula)
Bacula servers
Cold storage
Data flow
Control flow

- Content databases are special because they are *append-only*
- Incremental logical backups are sent to cold storage
- Not yet implemented

Results

WIKIMEDIA
FOUNDATION

Total dataset backed up & retention policy

Per Datacenter

- Per run, **18 TB** of metadata and misc source hosts + **15 TB** of read write content
- Weekly **1.4 TB** of dumps after compression
  - Also **12 TB** of content dumps
- **3 latest dumps** are stored on hot storage
  - Latest **3 months** (~12 copies) on cold
- **2.7 TB** of snapshots every other day
  - Retention of 1 week (3 copies)

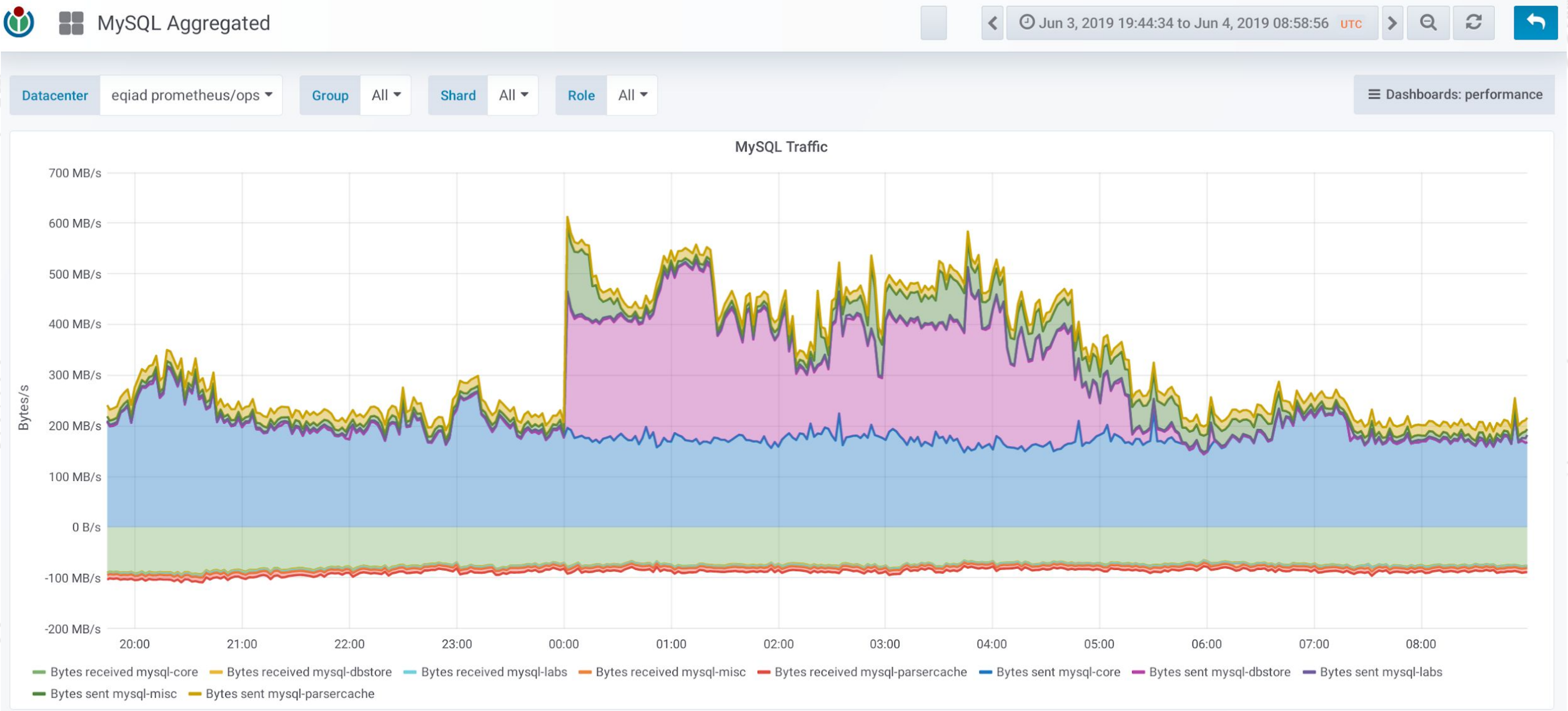
Available disk & Example Size

Per Datacenter

- Total database backup storage available at the moment (hot + cold): **75 TB**
- Example: English Wikipedia metadata (enwiki)- Sept 2019
  - Production host: **2.0 TB**
  - Backup source: **1.3TB** (no binlogs, InnoDB compressed)
  - Mydumper, compressed: **149 GB**
  - Snapshot, compressed: **371GB**

Time to backup

- 4 dumps + 2 snapshot jobs are processed **in parallel** on each datacenter
- Total backup time:
  - All dumps: **~7 hours**
  - All snapshots: **~12 hours**
- enwiki (2TB) takes:
  - **1h25m** for mydumper + **10m** for post-processing
  - **1h20m** for xtrabackup transfer + **1h20m** for post-processing
- Replication is stopped on replicas with high write throughput



MySQL Aggregated

Jun 3, 2019 19:44:34 to Jun 4, 2019 08:58:56 UTC

Datacenter: eqiad prometheus/ops

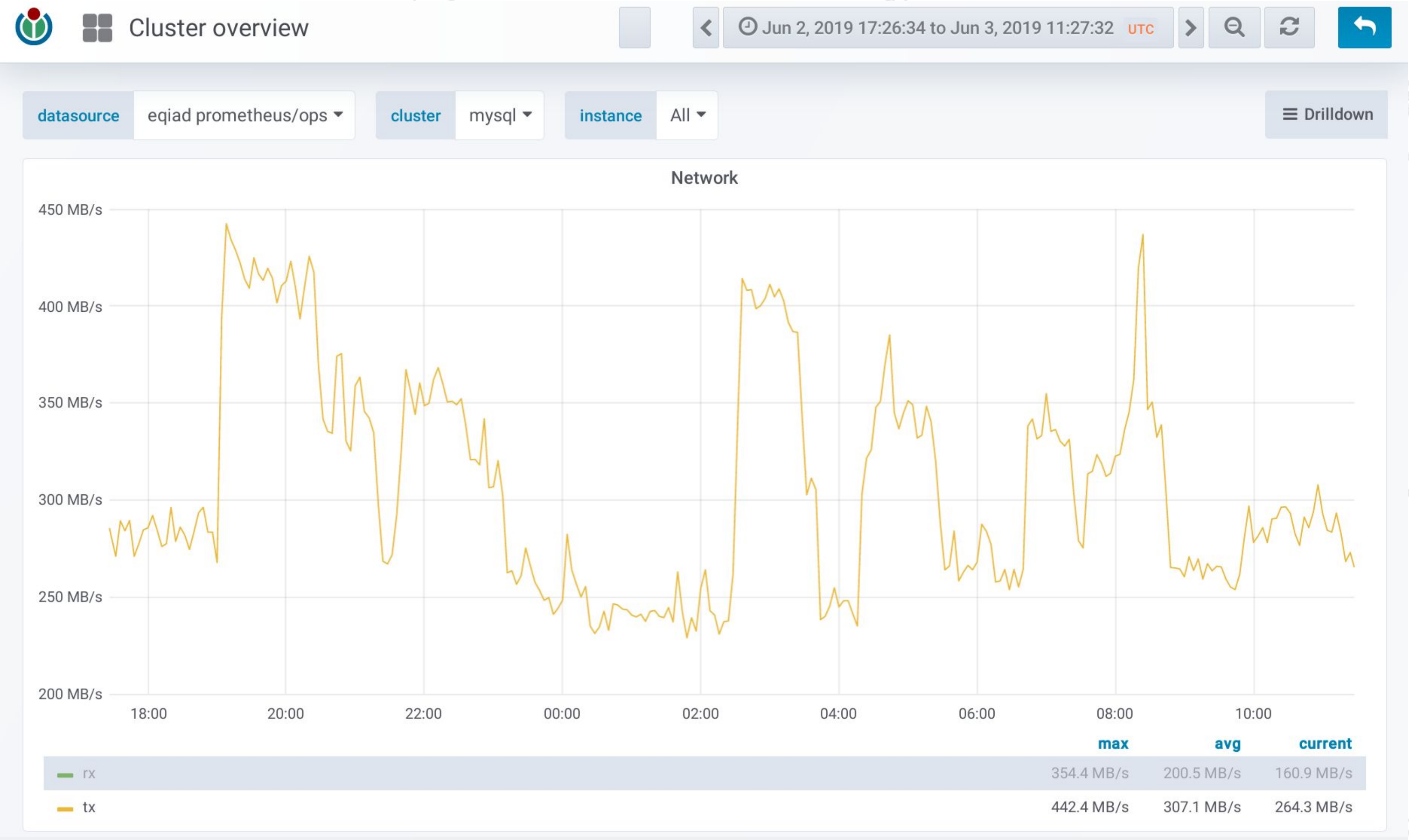
Group: All

Shard: All

Role: All

Dashboards: performance

MySQL Traffic



Cluster overview

Jun 2, 2019 17:26:34 to Jun 3, 2019 11:27:32 UTC

datasource: eqiad prometheus/ops

cluster: mysql

instance: All

	max	avg	current
🟩 rx	354.4 MB/s	200.5 MB/s	160.9 MB/s
🟨 tx	442.4 MB/s	307.1 MB/s	264.3 MB/s

Time to Recovery

- The fastest time our enwiki database (2TB) can be recovered from the provisioning host is 12m30s:
  - Not all steps have been automated yet (not real TTR)
  - Requires 10Gbit
  - Requires resources (network, cpu) not always available
  - Large number of small files has extra overhead
- Realistically: 30m-60m for a full cluster

WIKIMEDIA FOUNDATION

Planned Work & Lessons Learned

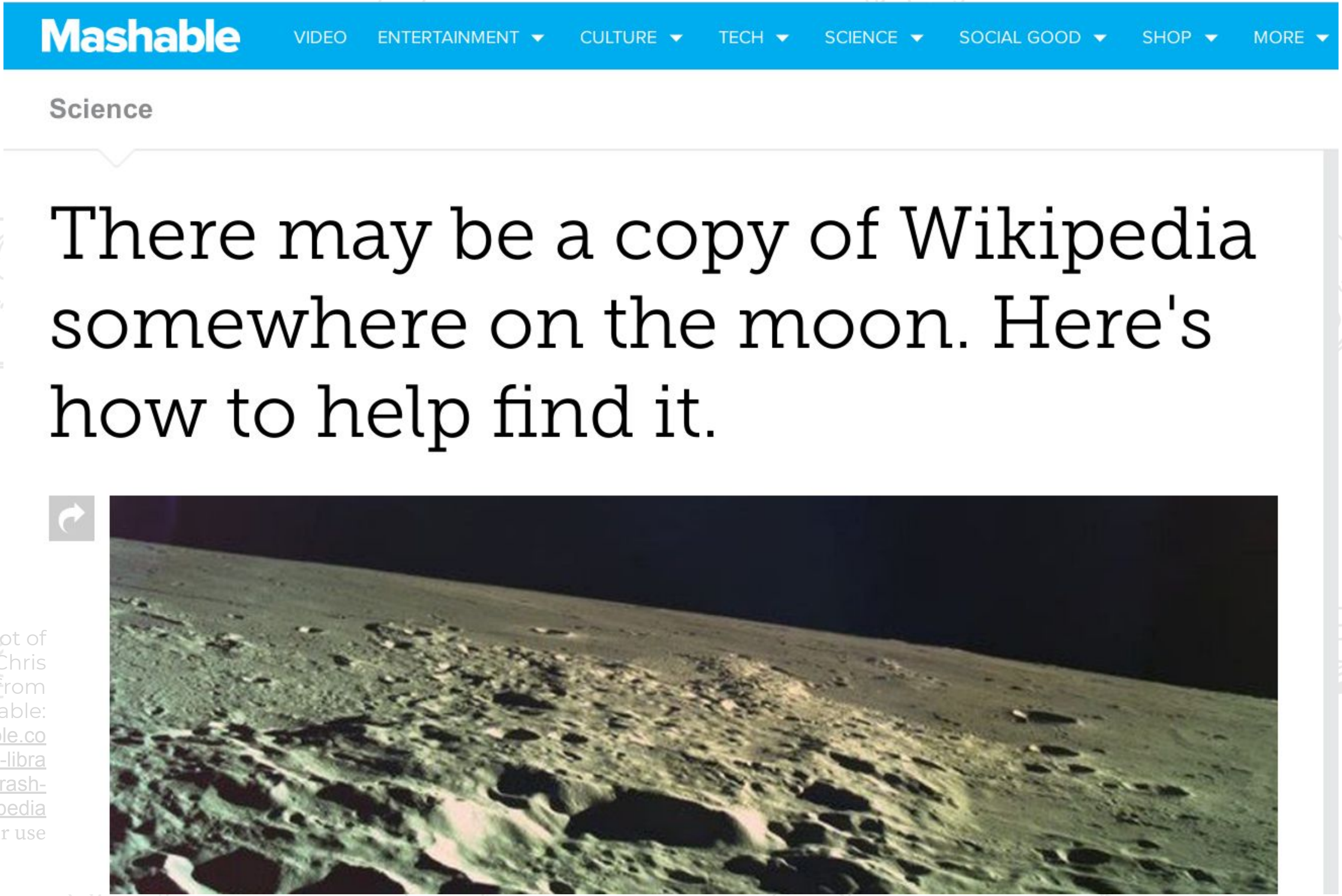
WIKIMEDIA
FOUNDATION

Coming next...

- Fully automated provisioning & testing cycle
- Improve monitoring
- Fully automated content backups
- Automated point in time recovery
- Research incrementals methods
- Offline backups

Lessons Learned

- Parallelize (and redundancy)
- Get Data about your Backups
- Plan, but be open to changes
- Think about recovery first; design your backups for it
- Have a plan B, plan C, …
  and even a plan D…



Science

There may be a copy of Wikipedia somewhere on the moon. Here's how to help find it.

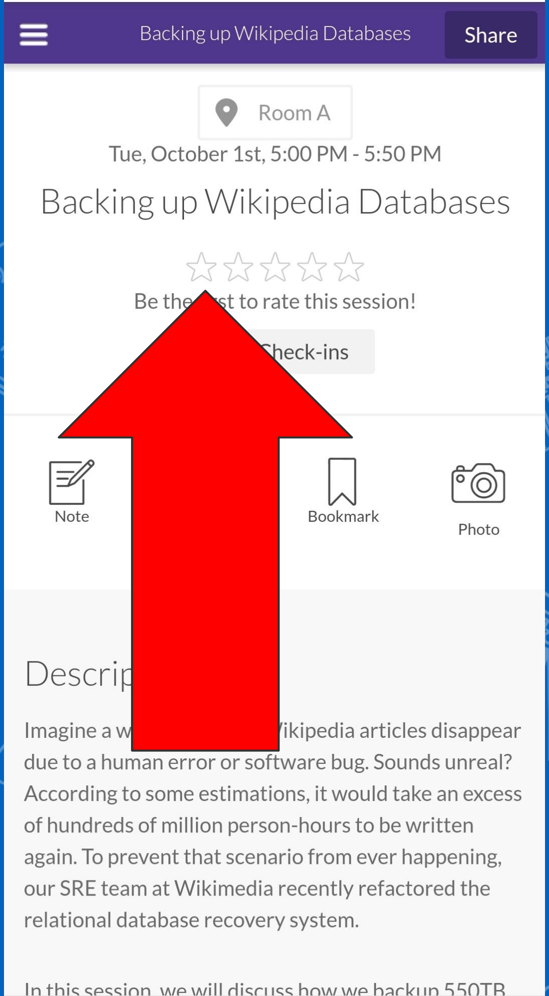
* Screenshot of article by Chris Taylor from Mashable: https://mashable.com/article/moon-library-beresheet-crash-wikipedia
Used under fair use

Thank you!

Special thanks: Alex, Ariel, Effie, Mark, Rubén, WMF SRE Team and Percona Live Committee

WIKIMEDIA FOUNDATION

Author: Jaime Crespo & Manuel Arostegui, Wikimedia Foundation



Backing up Wikipedia Databases

Room A

Tue, October 1st, 5:00 PM - 5:50 PM

Backing up Wikipedia Databases

☆☆☆☆☆

Be the [illegible]st to rate this session!

Descrip[illegible]

Imagine a w[illegible]ikipedia articles disappear due to a human error or software bug. Sounds unreal? According to some estimations, it would take an excess of hundreds of million person-hours to be written again. To prevent that scenario from ever happening, our SRE team at Wikimedia recently refactored the relational database recovery system.

In this session, we will discuss how we backup 550TB

Please rate us!

We are hiring:

https://wikimediafoundation.org/about/jobs/

WIKIMEDIA
FOUNDATION

Author: Jaime Crespo & Manuel Arostegui, Wikimedia Foundation