SQLite DB Backup — Pillar #9 LITE

Genesis: Pillar #9 LITE (CEO approved 2026-05-05, DR-only scope). Spec: ~/system/specs/agentic-os-pillar9-LITE-2026-05-05.md. MC: #99248.

Daemon file: /Users/makinja/system/daemons/azure-db-backup.sh (524 lines)

Schedule: Every 4 hours via LaunchAgent com.alai.azure-db-backup (StartInterval=14400)

Overview

This runbook covers the SQLite backup phase of azure-db-backup.sh. It documents what gets backed up, how backups are produced, how to restore from Azure Blob Storage, and how to add new databases to the backup set.

The SQLite backup extension was introduced under Pillar #9 LITE to provide DR coverage for the four critical local SQLite databases that drive ALAI operational systems. Backups land in the same Azure Blob Storage container as Docker volume, Postgres, and Qdrant backups, under the sqlite/ prefix.

Blob lifecycle policy: 30 days Cool to Archive, deleted at 365 days (existing container policy, no additional configuration required).

Slack channel #ops receives an alert if 3 or more consecutive backup runs fail (MAX_FAILURES=3 in the script).

What Gets Backed Up

Four databases are covered, defined at lines 466-471 of the daemon file (SQLITE_DBS array):

Label	Path	Purpose
mission-control	$HOME/system/databases/mission-control.db	All MC tasks, owners, priorities, status history
hivemind	$HOME/system/databases/hivemind.db	Institutional knowledge, facts, session summaries
costs	$HOME/system/databases/costs.db	Token cost tracking, budget records

Label	Path	Purpose
knowledge	$HOME/system/databases/knowledge.db	Extracted knowledge index (187 MB)

The authoritative list lives in the SQLITE_DBS array at lines 466-471. Add new databases there; no other code change is required.

How It Works

The backup_sqlite() function (lines 420-462) executes four steps per database:

1. **Snapshot via online-backup API.** sqlite3 .backup command is WAL-safe: it acquires a shared lock just long enough to copy pages, then releases it. The running application continues reading and writing throughout. Output: /tmp/alai-azure-backup-ts/sqlite-label-DATE.db
2. **Gzip compression.** gzip -f replaces the snapshot file with .gz in-place.
3. **SHA-256 sidecar.** The upload_blob() function computes a .sha256 file alongside the .gz before upload, enabling integrity verification at restore time.
4. **Azure Blob upload.** az storage blob upload sends both the .gz and .sha256 sidecar to the container. Blob path pattern: sqlite/YYYY-MM-DD/label.db.gz

If the source database file is absent, the step is skipped with a WARN log entry (non-fatal). If sqlite3 .backup or gzip fails, the run is counted as a failure and the consecutive-failure counter increments.

Dry-run mode: Pass --dry-run to the script. No snapshot is taken, no upload occurs; the planned blob path is logged.

Restore Procedure

Restore target: vm-alai-support (4.223.110.181). SSH: ssh -i ~/.ssh/azure_alai alai-admin@4.223.110.181

Step 1 - Identify the backup to restore

List blobs: az storage blob list --account-name $AZURE_STORAGE_ACCOUNT --container-name $AZURE_CONTAINER_DB --prefix "sqlite/2026-05-05/" --auth-mode login --output table

Pick the label.db.gz blob you need.

Step 2 - Download and verify

Download the .db.gz blob and its .db.gz.sha256 sidecar to /tmp/restore/. Verify: sha256sum -c mission-control.db.gz.sha256

Step 3 - Decompress

gunzip /tmp/restore/mission-control.db.gz

Step 4 - Verify schema and data

Schema: sqlite3 /tmp/restore/mission-control.db ".schema"

Data: sqlite3 /tmp/restore/mission-control.db "SELECT id, title, status FROM tasks LIMIT 10;"

mc.js list should return task rows if the DB is valid.

Step 5 - Place into production path

Check for open handles: lsof | grep mission-control.db. Then: cp /tmp/restore/mission-control.db ~/system/databases/mission-control.db

Never overwrite a live database without passing the schema and data checks above.

Monitoring

Log file: ~/system/logs/azure-db-backup.log

Phase markers: grep "SQLite backup phase" ~/system/logs/azure-db-backup.log | tail -20

Per-DB success: grep "SQLite snapshot done" ~/system/logs/azure-db-backup.log | tail -20

Errors: grep -i "error.*sqlite\|warn.*sqlite" ~/system/logs/azure-db-backup.log | tail -20

Verify blobs for today: az storage blob list --account-name $AZURE_STORAGE_ACCOUNT --container-name $AZURE_CONTAINER_DB --prefix "sqlite/$(date +%Y-%m-%d)/" --output table

Expected: 8 blobs (4 x .db.gz + 4 x .db.gz.sha256) per successful run.

LaunchAgent health: launchctl list | grep azure-db-backup (PID non-zero = running; last column = exit code, 0 = success)

Adding a New DB

SQLITE_DBS array at lines 466-471 of /Users/makinja/system/daemons/azure-db-backup.sh is the single point of configuration.

1. Open azure-db-backup.sh and locate the SQLITE_DBS array (lines 466-471).
2. Append a new entry: "label:$HOME/system/databases/name.db". Use lowercase, hyphen-separated labels with no spaces.
3. Test: bash ~/system/daemons/azure-db-backup.sh --dry-run 2>&1 | grep sqlite
4. Confirm the planned blob path appears in log output for the new label.
5. Update the "What Gets Backed Up" table in this runbook.
6. Update MC #99248 or open a follow-up MC to track the addition.

No other code changes required. backup_sqlite() handles all databases uniformly.

Troubleshooting

Symptom	Likely cause	Resolution
WARN: SQLite DB not found, skipping	Database file does not exist at registered path	ls -lh ~/system/databases/name.db. If path moved, update SQLITE_DBS array.
ERROR: sqlite3 .backup failed	sqlite3 CLI missing, DB corrupted, or disk full	which sqlite3; df -h /tmp; sqlite3 name.db "PRAGMA integrity_check;"
ERROR: gzip failed	Disk full on /tmp	df -h /tmp; rm -rf /tmp/alai-azure-backup-*
Blob missing after upload reports success	SP credential expired or wrong container	Verify AZURE_CONTAINER_DB in ~/system/config/azure-backup.env.
Slack alert fires repeatedly	3+ consecutive run failures	tail -100 ~/system/logs/azure-db-backup.log. Fix phase. echo 0 > /tmp/azure-db-backup-failcount
LaunchAgent not running (PID=0)	LaunchAgent unloaded or crashed	launchctl load ~/Library/LaunchAgents/com.alai.azure-db-backup.plist

Related Runbooks

Telegram Bot Intent Classifier — comms-responder (#99290) — Intent classification fix for the ALAI Telegram bot. Same Operations Runbooks shelf.

Revision #4
Created 2026-05-05 20:19:21 UTC by John
Updated 2026-06-07 20:01:18 UTC by John