Neo Matrix Bot — Frequently Asked Questions (FAQ)

Generated: 2025-12-20

This document collects common technical and interview-style questions and concise answers about the Neo Matrix bot project. Use it to prepare for developer interviews or to onboard contributors.

---

1. What is Neo Matrix bot? ■

Neo Matrix is a Telegram bot written in Python using the python-telegram-bot library. It offers features such as media saving and tracking, automatic deletion, scheduled backups of JSON databases, favorites management, and admin controls.

---

2. Which technologies and libraries are used? ■

- Language: Python - Telegram library: python-telegram-bot (telegram, telegram.ext) - Data storage: JSON files (e.g., `users_db.json`, `media_db.json`, `favorites_db.json`) - Backup: `backup_manager.py` to archive database snapshots - Deployment targets: Railway, Heroku style (contains `Procfile`, `runtime.txt`, `railway.toml`)

---

3. How does the bot store and manage data? ■

The bot uses structured JSON files inside the repository (or `data/`) to persist state: user info, media metadata, favorites, and configuration. Changes are written to JSON with safety checks; backups are managed by `backup_manager.py` which creates timestamped snapshots.

---

4. How do backups work? ■

A scheduled process in `bot.py` tracks time since last backup and triggers `DatabaseBackupManager` to copy JSON files into the `backups/` folder with timestamps. The interval is configurable (default 24 hours).

---

5. How does autodelete work? ■■

Files and messages flagged for auto-deletion are tracked (e.g., `auto_delete_tracking.json`), and a periodic cleanup routine inspects and removes entries when the configured retention has passed. The behavior is governed by `autodelete_config.json`.

---

6. How are admin users and permissions handled? ■

Admin IDs are configured via environment variables (`ADMIN_ID`) or stored in `admin_list.json`. Command handlers check the sender ID against the admin list before allowing privileged operations like backups, restores, or config changes.

---

7. How to handle bot tokens and secrets securely? ■

- Use environment variables (e.g., `BOT_TOKEN` in `.env`) and avoid committing tokens to Git. - Add `.env` to `.gitignore` and rotate the token if leaked. - Use platform secret management (Railway/Heroku secrets, GitHub Actions secrets) for CI/CD.

---

8. How to deploy the bot? ■

- Install dependencies in `requirements.txt`. - Configure environment variables in platform settings. - Use `Procfile` for process startup; platforms like Railway can use `runtime.txt` and `railway.toml`.

---

9. How does the bot handle rate limits and network errors? ■■

The bot catches `telegram.error` subclasses such as `TimedOut`, `NetworkError`, and `RetryAfter`, and implements retry/backoff strategies in long-running processes. Command handlers should avoid blocking operations and use asynchronous/queued processing when needed.

---

10. How is logging and monitoring implemented? ■

The bot emits startup and periodic status logs (see console prints in `bot.py`). You can forward logs to external systems (Papertrail, LogDNA) or augment with structured logging for easier analysis.

---

11. How do you test this bot? ■

- Unit test core functions (file IO, backup manager, config reads). - Integration test with a bot token in a test chat and check message flows. - Mock Telegram API calls when running offline unit tests.

---

12. How to add a new command or feature? ■

- Add a new handler in `bot.py` using `CommandHandler` or `MessageHandler`. - Keep handler logic small and move complex logic to helper modules for testability. - Update README and add unit tests for the new behavior.

---

13. How are media files handled and deduplicated? ■

Media metadata (file IDs, sizes, captions) is stored in `media_db.json`. Deduplication can be done by comparing Telegram file IDs or computing local hashes when needed.

---

14. How would you scale this bot? ■

- Move from file-based JSON storage to a proper DB (Postgres, Redis) for concurrency and performance. - Split processing into worker processes (task queue: Celery/RQ) for heavy tasks. - Use horizontal scaling with sticky sessions or external storage for shared state.

---

15. What security/privacy considerations exist? ■■

- Avoid storing sensitive PII unless necessary; minimize who can access data files. - Secure backups and rotate credentials. - Validate and sanitize user inputs, and handle untrusted media cautiously.

---

16. How does the project handle configuration changes? ■■

Configs are stored in JSON files like `autodelete_config.json` and `caption_config.json`. Changes are applied after saves and optionally require a restart for global reloading depending on implementation.

---

17. How to recover from a corrupted JSON database? ■

- Use the most recent backup from `backups/`. - Restore the JSON files and restart the bot. - Consider adding validation checks and atomic writes to reduce corruption risk.

---

18. What are common bugs or edge cases to watch for? ■

- Concurrent writes to JSON causing race conditions - Network flakiness during backups or long operations - Rate-limiting errors when sending many messages - Missing environment variables on deploy

---

19. How would you add unit/integration tests here? ■

- Extract pure functions for file I/O, parsing, and scheduling to make them testable. - Use pytest with temporary directories for JSON fixtures. - Mock Telegram API responses for integration tests.

---

20. Example interview question: "Explain a design decision you made in this bot." ■■

Focus on trade-offs: e.g., using JSON file storage was chosen for simplicity and quick iteration; for production scale we would move to a robust DB to handle concurrent writes and large datasets.

---

21. How to contribute? ■

- Read `README.md`, follow contribution guidelines, run tests, and submit PRs. - Keep changes small and document new behaviors.

---

22. Where to find critical files? ■

- Main bot logic: `bot.py` - Backup utilities: `backup_manager.py` - Config and data: `*.json` files at repo root and `data/` - Deployment configuration: `Procfile`, `runtime.txt`, `railway.toml`

---

23. Additional tips for interviews ■

- Be ready to discuss trade-offs (simplicity vs scalability), error handling, and security. - Show familiarity with Telegram API concepts (file IDs, update handling, webhook vs polling).

If you'd like, I can: - Convert this document to a PDF and add it to the repository as `FAQ.pdf` ■ - Expand or tailor the questions for a specific interview (backend, SRE, or product) ■
