IIITorrent

1 Use Case

College students and staff often face torrenting bans on campus networks, limiting access to essential content. This project circumvents such restrictions by using Oracle Cloud Infrastructure as a proxy for torrent downloads. qBittorrent runs on a cloud server to fetch torrents remotely, which are then transferred to users' local devices. This approach ensures secure, efficient access while bypassing network limitations and optimizing resource management. The project falls within the domain of cloud-based content distribution and network circumvention.

2 Requirements

2.1 Functional Requirements

- **FR1**: Utilize Oracle Cloud compute instances to download torrents and transfer the files to users' local devices. Include user authentication and authorization to manage access.
- **FR2**: Enable users to add and manage RSS feeds that periodically update based on fixed intervals. The system will filter these feeds against user-defined criteria and automatically trigger downloads for new content.
- **FR3**: Allow users to create and maintain a watchlist of shows. Integrate with external APIs to update watchlists and automatically download new episodes or related content.
- **FR4**: Provide an admin web view to monitor server resource usage, download statistics, and system health in real time.
- **FR5**: Develop a cross-platform user interface that displays download statuses, watchlist updates, and RSS feed information.

2.2 Non-Functional Requirements

- **NF1**: The system must update and process each RSS feed every 10 minutes, ensuring that the processing time for each feed does not exceed 30 seconds.
- **NF2**: The system should support up to 50 concurrent torrent downloads without noticeable degradation in performance.
- **NF3**: Perform automated daily backups of the server database with a Recovery Time Objective (RTO) of 1 hour and a Recovery Point Objective (RPO) of 15 minutes.

3 Technical Overview

3.1 Tech Stack Overview

- **Backend (FastAPI)**: Manages user authentication, interfaces with qBittorrent Web API, and provides API endpoints for watchlists, RSS feeds, and downloads.
- **Database (PostgreSQL)**: Stores user watchlists, torrent statuses, and RSS subscriptions.
- **Monitoring (Prometheus + Grafana + pgexporter)**: Tracks server health, storage, and database info.
- **Torrent Management (qBittorrent Web API)**: Handles downloads and torrent status.
- **Watchlist Sync (Anilist API)**: Fetches user watchlists, auto-downloads new episodes.
- **User Interface (Textual - Python TUI)**: Terminal UI for users.

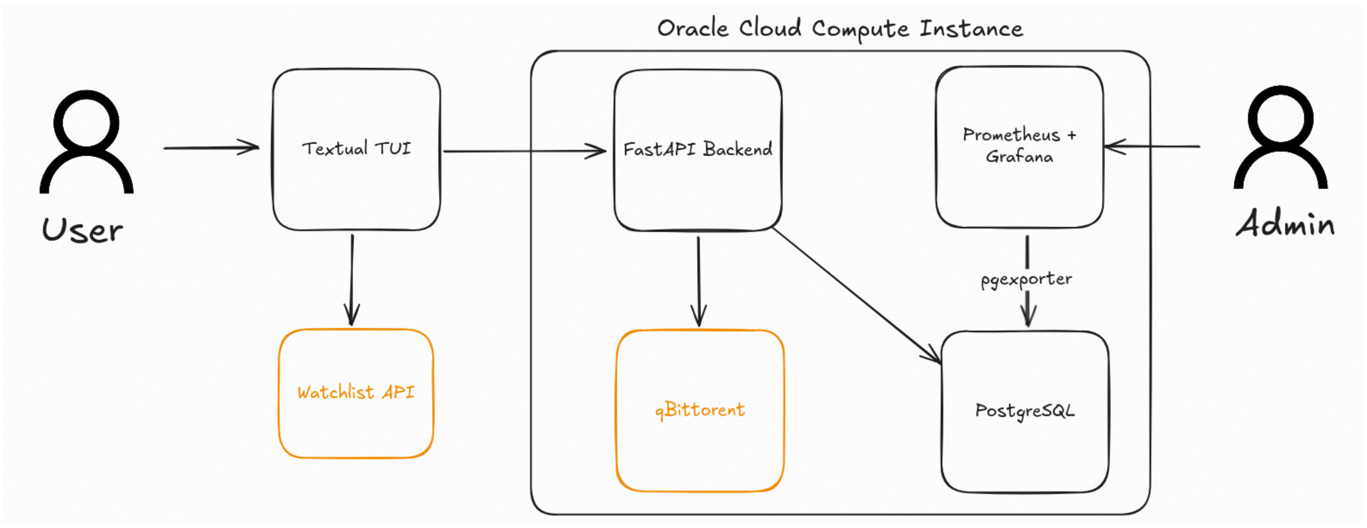
3.2 Workflow

- User adds a show via Anilist API.
- System updates RSS feeds and downloads new episodes.
- qBittorrent downloads torrents to OCI.
- User uses Textual TUI and transfers files.
- Grafana/Prometheus monitor server performance.

4 Design Patterns

- **Observer Pattern**: Used for RSS feed updates; when new torrents appear, the system notifies relevant modules.
- **Factory Pattern**: Simplifies API integration by creating different clients (e.g., Anilist, IMDB) dynamically.
- **Command Pattern**: Encapsulates torrent download requests, allowing easy queuing and retries.
- **Singleton Pattern**: Ensures only one instance of qBittorrent API client and monitoring services run at a time.

5 Architecture



6 Timeline

Phase	Tasks	Duration	Team Members
Research & Planning	Define features and scope. Research OCI, qbittorent API, Anilist API	3 days	All 5
Design	DB Schema, API design, user flow	3 days	2 FastAPI, 2 TUI, 1 Grafana
Development	Implement backend, develop TUI, integrate Anilist API, Setup qbittorent automation, configure admin view	10 days	2 FastAPI, 2 TUI, 1 Grafana
Testing & Refinement	Unit & integration testing, fix bugs	10 days	All 5
Deployment	Dockerize & deploy	1 day	All 5