OCI Torrent Proxy System

Architectural Design Report

Project Name:	[OCI Torrent Proxy System]
Project Number:	[Your Project Number]
Course Name:	[Your Course Name]
Team Name:	[Your Team Name]
Team Members:	[Member 1, Member 2, ...]
Date:	[Current Date, e.g., April 4, 2025]

Table of Contents

1 Introduction

This report details the architectural design for the OCI Torrent Proxy System. The system aims to provide college students and staff a method to bypass campus network restrictions on torrent traffic by utilizing an Oracle Cloud Infrastructure (OCI) instance as a remote download proxy. This document covers the initial architectural planning stages, including requirements elicitation, subsystem identification, stakeholder analysis, key design decisions (ADRs), and the selection of architectural tactics and patterns. Architectural diagrams illustrating the proposed structure are included. This report currently covers Tasks 1, 2, and 3 of the project deliverables. Task 4 (Prototype Implementation and Analysis) will be added subsequently.

1.1 Task 1: Requirements and Subsystems

1.1.1 Functional Requirements (FR)

1. **FR1: Remote Torrent Download & Transfer:** Utilize an Oracle Cloud compute instance with qBittorrent for downloading torrents. Provide authenticated users a mechanism (via TUI) to initiate downloads and securely transfer completed files (e.g., via user-initiated SFTP/SCP) to local devices.
2. **FR2: User and Access Management:** Implement user authentication (login/password) and authorization. Define distinct roles: 'User' (manages personal downloads, feeds, watchlist) and 'Admin' (manages users, system monitoring access).
3. **FR3: RSS Feed Management & Automation:** Allow users to add, view, remove, and define filter criteria (keywords, regex) for RSS feeds. The system backend must periodically check feeds, match against filters, and auto-initiate downloads for new items via qBittorrent.
4. **FR4: Watchlist Integration & Automation:** Allow users (via TUI) to manage a watchlist (add/remove shows). The backend must integrate with external services (e.g., Anilist API) to fetch watchlist details and identify new episodes/content based on user entries, triggering downloads (potentially correlated with RSS feeds).
5. **FR5: Torrent Lifecycle Management:** Enable users (via TUI) to view download status (downloading, seeding, completed, error), pause, resume, and delete their torrents managed by the system.
6. **FR6: Admin Monitoring View:** Provide administrators access to a web-based dashboard (Grafana) displaying real-time server resource usage (CPU, RAM, disk, network), qBittorrent statistics, database health, and potentially aggregated user activity/stats.
7. **FR7: User Interface (TUI):** Provide a cross-platform Terminal User Interface (Textual) for user interaction: displaying status, adding torrents/feeds, managing the watchlist, and getting information for file transfers. Runs on the user's machine.

1.1.2 Non-Functional Requirements (NFR)

1. **NF1: Performance (RSS Processing):** Check each RSS feed at least every 10 minutes. Processing time per feed check (filtering, download initiation) < 30 seconds under normal load.
2. **NF2: Performance (Concurrency):** Support at least 50 concurrent torrent downloads/seeds without significant API/TUI responsiveness degradation. (Network throughput depends on OCI/peers).
3. **NF3: Reliability (Backup & Recovery):** Perform automated daily database backups (PostgreSQL). RTO: 1 hour. RPO: 24 hours.
4. **NF4: Security (Authentication):** Secure user authentication (hashed+salted passwords). Protect API endpoints with token-based authentication (e.g., JWT).
5. **NF5: Security (Authorization):** Enforce authorization rules: users manage only their own resources; admins have specific elevated privileges (user management, monitoring access).
6. **NF6: Usability (TUI):** The TUI should be intuitive for terminal users, providing clear feedback and navigation.
7. **NF7: Resource Constraints (OCI):** Operate within OCI Always Free tier limits (Ampere A1 CPU/RAM, storage, network egress). Monitor usage.

8. **NF8: Maintainability:** Ensure modular, documented backend code with tests for easier updates.
9. **NF9: Monitoring Availability & Freshness:** The Admin monitoring view (Grafana) should be highly available (best effort on free tier) and display metrics with a latency of <= 1 minute from collection time.

1.1.2.1 Architecturally Significant Requirements

The following requirements have the most significant impact on the architectural design:

- **NF1 & NF2 (Performance):** Drive choices for async backend, database tuning, OCI instance sizing.
- **NF3 (Reliability):** Dictates backup strategy and operational procedures.
- **NF4 & NF5 (Security):** Fundamental; require secure design of API, auth flows, and data storage.
- **NF6 & FR7 (Usability/TUI):** Drive the choice of Textual and interaction design.
- **NF7 (Resource Constraints):** A primary constraint influencing technology choices and scale.
- **FR6 & NF9 (Monitoring):** Dictates the monitoring stack (Prometheus/Grafana) and its configuration.
- **FR4 (Watchlist Integration):** Requires careful design of backend interaction with external APIs.

1.1.3 Subsystem Overview

The system is composed of the following main subsystems:

- **User Interface (TUI - Textual):** User-facing terminal application running on the user's machine. Displays info, takes input, communicates **only** with the API Backend via HTTPS.
- **API Backend (FastAPI):** Central API service running on OCI. Handles authentication, authorization, business logic (RSS, watchlist), orchestrates qBittorrent, interacts with the Database, and integrates with external APIs (Anilist).
- **Torrent Client Service (qBittorrent):** The qBittorrent server process running on OCI, managed via its Web API by the API Backend. Handles P2P communication.
- **Database (PostgreSQL):** Stores persistent state: users, torrent metadata, RSS feeds/filters, watchlists. Runs on OCI.
- **Monitoring Service (Prometheus + Grafana + Exporters):** Runs on OCI. Collects metrics from OS, PostgreSQL, and API Backend. Grafana provides dashboards accessed by the Admin.
- **Watchlist API (External - e.g., Anilist):** External service providing show information and user watchlist data. Accessed **by the API Backend**.
- **(Implicit) File Storage:** The filesystem location on the OCI instance where qBittorrent saves downloaded files. Users access this via secure protocols like SFTP/SCP.

1.2 Task 2: Architecture Framework

1.2.1 Stakeholder Identification (IEEE 42010)

The following table identifies key stakeholders, their primary concerns, and the architectural viewpoints relevant to addressing those concerns.

Stakeholder	Concerns	Relevant Viewpoint(s)
End User	Ease of use (TUI), reliable downloads, bypassing restrictions, status visibility, file transfer ease, account security, availability	Functional, Usability (Interaction), Operational, Security
System Admin	System uptime, resource monitoring (OCI limits/cost), performance, backup/	Deployment, Operational, Performance, Security, Development

	recovery, user mgmt, security posture, monitoring dashboard usability & accuracy	
Developer	Modularity, maintainability, testability, ease of adding features (APIs, filters), clear architecture, tech stack suitability	Development, Logical (Container), Deployment, Information
(Implicit) OCI	Resource consumption within limits, ToS compliance (network usage)	Deployment, Operational

1.2.2 Major Design Decisions (ADRs)

The following Architecture Decision Records document key design choices made for the system.

1.2.2.1 ADR 001: Cloud Provider Selection

- **Status:** Accepted
- **Context:** Need for a remote server to bypass network restrictions, with cost as a major constraint for students.
- **Decision:** Utilize Oracle Cloud Infrastructure (OCI) Always Free Tier compute instances (Ampere A1).
- **Rationale:** Cost-effectiveness, generous CPU/RAM in free tier, full OS control.
- **Consequences:** Potential I/O/network bottlenecks, resource availability/reclamation risks, requires usage monitoring within free limits.

1.2.2.2 ADR 002: Torrent Client and Interaction Method

- **Status:** Accepted
- **Context:** Requirement for a controllable torrent client on the remote server, manageable by the backend.
- **Decision:** Use qBittorrent server controlled via its Web API (v2) using a Python client library within the FastAPI backend.
- **Rationale:** Feature-rich client, mature Web API, headless operation support, available Python libraries.
- **Consequences:** Dependency on qBittorrent process stability, API interaction failure points, requires qBittorrent configuration management.

1.2.2.3 ADR 003: Backend Technology Choice

- **Status:** Accepted
- **Context:** Need for an efficient backend API service for orchestration and handling I/O-bound tasks.
- **Decision:** Use FastAPI (Python framework).
- **Rationale:** Asynchronous performance, Python ecosystem alignment (TUI, libraries), good developer experience (docs, validation), concurrency handling via `asyncio`.
- **Consequences:** Requires `asyncio` proficiency, ASGI server deployment (Uvicorn).

1.2.2.4 ADR 004: User Interface Technology

- **Status:** Accepted
- **Context:** Need for a cross-platform user interface suitable for terminal users, avoiding web frontend complexity.

- **Decision:** Use Textual (Python TUI framework) running on the user's machine.
- **Rationale:** Cross-platform, Python-based, avoids web dev complexities, rich TUI components, suitable for target users.
- **Consequences:** TUI limitations vs GUI, requires terminal comfort, file transfer initiation is less direct (e.g., showing SCP command).

1.2.2.5 ADR 005: Monitoring Stack Implementation

- **Status:** Accepted
- **Context:** Requirement FR6 necessitates an admin monitoring view for system health and performance (NF9).
- **Decision:** Implement monitoring using Prometheus, Grafana, `node_exporter` (OS metrics), and `pgexporter` (PostgreSQL metrics). FastAPI backend exposes a `/metrics` endpoint.
- **Rationale:** Industry standard, powerful visualization (Grafana), extensible (exporters), good integration capabilities.
- **Consequences:** Consumes resources on OCI instance, adds configuration complexity, requires familiarity with the tools.

1.2.2.6 ADR 006: Watchlist API Integration Point

- **Status:** Accepted
- **Context:** Need to integrate with external watchlist APIs (e.g., Anilist, FR4). Decision needed on where this interaction occurs.
- **Decision:** All interactions with external Watchlist APIs will be handled exclusively by the FastAPI Backend. The TUI communicates only with the Backend for watchlist actions.
- **Rationale:** Enhances security (API keys server-side), centralizes logic, enables backend automation, maintains consistency (Backend as mediator), improves maintainability.
- **Consequences:** Increases Backend responsibilities, TUI fully dependent on Backend for watchlist data.

1.3 Task 3: Architectural Tactics and Patterns

1.3.1 Architectural Tactics

The following tactics are employed to address specific non-functional requirements:

1. **Performance - Concurrent Processing:** Utilize `async`/`await` in FastAPI for non-blocking I/O (API calls, DB, qBittorrent, external APIs, RSS checks). (Addresses NF1, NF2)
2. **Availability - Health Monitoring:** Implement `/health` endpoints. Use Prometheus to monitor these and metrics from FastAPI, PostgreSQL (pgexporter), qBittorrent (API checks), and host OS (node_exporter). (Addresses NF9, supports NF3)
3. **Availability - Data Redundancy:** Automated daily logical backups (`pg_dump`) of PostgreSQL, stored securely (e.g., OCI Object Storage). (Addresses NF3 RPO/RTO)
4. **Modifiability - Encapsulation / Interface:** API Backend acts as a facade. Clear API contracts (OpenAPI). Abstract external API (Anilist) and data access (PostgreSQL) behind service/repository classes. (Addresses NF8)
5. **Security - Authentication & Authorization:** Use JWT for API tokens. Enforce auth via FastAPI dependencies. Use `passlib` for password hashing. Implement role checks (User vs. Admin). (Addresses NF4, NF5)

1.3.2 Implementation Patterns

Key design patterns planned for the implementation include:

- **Repository Pattern:** Abstract data access logic for PostgreSQL (Users, Torrents, Feeds, Watchlists). Backend services use repository interfaces, improving testability and decoupling logic from storage.

- **Observer Pattern:** Potentially used for RSS feed checks: A scheduler triggers checks; upon finding new matching items, it notifies/triggers the download logic within the backend.
- **Factory Pattern:** Useful if supporting multiple watchlist providers (Anilist, MAL etc.) or potentially different torrent clients. A factory creates the correct client instance based on configuration.
- **Command Pattern:** Encapsulate qBittorrent actions (Add, Pause, Delete) initiated via the API. Simplifies API endpoint logic and allows for potential queuing/logging.
- **Singleton Pattern:** Applied judiciously for shared resources like DB connection pools or the qBittorrent client instance within the backend's lifetime. FastAPI's dependency injection often manages this implicitly.

1.3.3 Architectural Diagrams (C4 Model - PlantUML)

The following C4 diagrams illustrate the system architecture at different levels of abstraction. The diagrams are represented using PlantUML code.

1.3.3.1 System Context Diagram

This diagram shows the Torrent Proxy system as a black box interacting with users and external systems.