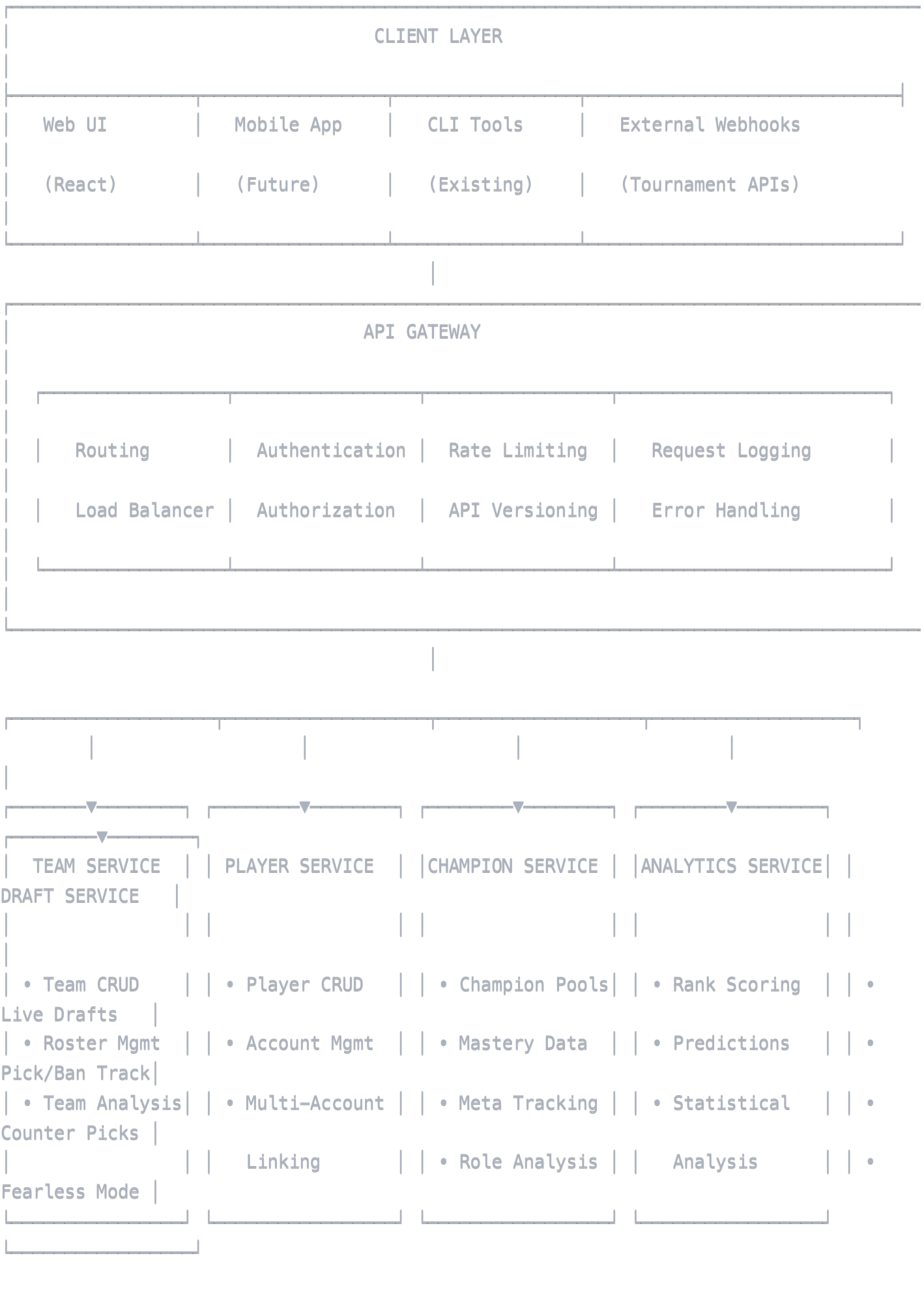
Zephyr Microservices Architecture Breakdown

Service Architecture Diagram


CLIENT LAYER
Web UI
(React)
Mobile App
(Future)
CLI Tools
(Existing)
External Webhooks
(Tournament APIs)
API GATEWAY
Routing
Load Balancer
Authentication
Authorization
Rate Limiting
API Versioning
Request Logging
Error Handling
TEAM SERVICE
• Team CRUD
• Roster Mgmt
• Team Analysis
PLAYER SERVICE
• Player CRUD
• Account Mgmt
• Multi-Account Linking
CHAMPION SERVICE
• Champion Pools
• Mastery Data
• Meta Tracking
• Role Analysis
ANALYTICS SERVICE
• Rank Scoring
• Predictions
• Statistical Analysis
DRAFT SERVICE
• Live Drafts
• Pick/Ban Track
• Counter Picks
• Fearless Mode


ORCHESTRATION SERVICES
REPORT SERVICE
• Scout Reports
• Multi-Service Orchestration
• Report Caching
TOURNAMENT SVC
• LEPL Logic
• GCS Logic
• Tournament Scoring
NOTIFICATION SERVICE
• Event Bus
• WebSocket Hub
• Real-time Updates
CACHE SERVICE
• Redis Cache
• Data Freshness
• Cache Invalidation
• Background Refresh
RIOT API SVC
• All Riot APIs
• Rate Limiting
• API Queue
• Retry Logic
• Data Mapping
SCRAPER SERVICE
• OP.GG Scraper
• LOG Scraper
• Rewind.lol
• GCS Scraper
• Selenium Mgmt
SHEETS SERVICE
• Sheet Creation
• Rich Formatting
• Data Insertion
• Hyperlinks
• Template Mgmt
QUEUE SERVICE
• Celery Tasks
• Job Queues
• Task Routing
• Failure Retry
• Priority Queue
DATABASE
• PostgreSQL
• Primary Data
• Relationships
• Transactions
• ACID Guarantees

Detailed Service Breakdown

1. API Gateway Service

Role: Single entry point for all client requests **Scope**: Cross-cutting concerns and request routing

Responsibilities:

- Route requests to appropriate microservices
- Handle authentication and authorization
- Rate limiting per client/endpoint
- Request/response logging and monitoring
- API versioning and backwards compatibility
- Load balancing across service instances
- Circuit breaker patterns for service failures

Technology:

- Kong, Zuul, or custom FastAPI gateway
- JWT token validation
- Redis for rate limiting counters

Key APIs Exposed:

```python
# Client-facing APIs
GET    /api/v1/teams/{team_id}
POST   /api/v1/teams/{team_id}/scout/{target_team_id}
GET    /api/v1/players/{player_id}/champion-pool
POST   /api/v1/draft/{draft_id}/pick
WebSocket: /api/v1/draft/{draft_id}/live
```

---

2. Team Management Service

Role: Manages teams, rosters, and team-level operations **Scope**: Everything related to team entities and team-level analysis

Responsibilities:

- Team CRUD operations (create, read, update, delete teams)
- Roster management (add/remove players from teams)
- Team-level champion pool aggregation
- Team vs team historical analysis
- Tournament team registration and management
- Team configuration and settings

Data Owned:

- Team metadata (name, region, tournament participation)
- Roster compositions and role assignments
- Team-level statistics and performance metrics
- Team preferences and configurations

Key Internal APIs:

```python
# Team Management APIs
POST   /internal/teams                    # Create team
GET    /internal/teams/{team_id}          # Get team details
PUT    /internal/teams/{team_id}/roster   # Update roster
GET    /internal/teams/{team_id}/players  # Get all team players
POST   /internal/teams/{team_id}/analyze  # Trigger team analysis
```

Service Dependencies:

- **Player Service**: Get player details for roster operations
- **Champion Service**: Aggregate team champion pools
- **Analytics Service**: Team-level statistical analysis
- **Database**: Team and roster persistence

---

3. Player Data Service

Role: Manages individual players and their associated accounts **Scope**: Player lifecycle, account linking, and basic player operations

Responsibilities:

- Player CRUD operations
- Multi-account management per player (main + smurf accounts)
- Player account linking/unlinking
- Player profile management
- Account verification and validation
- Player search and lookup operations

Data Owned:

- Player profiles (Discord username, declared positions)
- Account associations (Riot IDs, PUUIDs, summoner IDs)
- Account metadata (region, primary/secondary status)
- Player preferences and settings

Key Internal APIs:

```python
# Player Data APIs
POST   /internal/players                      # Create player
GET    /internal/players/{player_id}          # Get player details
POST   /internal/players/{player_id}/accounts # Link new account
GET    /internal/players/by-team/{team_id}    # Get team players
PUT    /internal/players/{player_id}/verify   # Verify player accounts
```

Service Dependencies:

- **Riot API Service**: Validate and resolve Riot accounts
- **Database**: Player and account persistence
- **Cache Service**: Frequently accessed player data

---

4. Champion Pool Service

Role: Manages champion mastery data and champion pool analysis **Scope**: Champion-related data processing and meta analysis

Responsibilities:

- Champion mastery data aggregation across multiple accounts
- Role-specific champion pool analysis

- Champion meta tracking and trends
- Champion performance statistics
- Multi-account champion data combination with weighted algorithms
- Champion recommendation systems

Data Owned:

- Champion mastery data (games played, winrate, KDA, mastery points)
- Role-specific champion statistics
- Meta trends and champion priority rankings
- Champion synergy and counter-pick data

Key Internal APIs:

```python
# Champion Pool APIs
GET    /internal/champion-pools/{player_id}        # Get player champion pool
POST   /internal/champion-pools/aggregate          # Aggregate multi-account data
GET    /internal/champion-pools/meta-trends        # Current meta analysis
POST   /internal/champion-pools/bulk-update        # Bulk champion data update
GET    /internal/champion-pools/role/{role}        # Role-specific pools
```

Service Dependencies:

- **Scraper Service**: Champion mastery data from Rewind.lol
- **Riot API Service**: Champion mastery from Riot API
- **Analytics Service**: Statistical processing and aggregation
- **Cache Service**: Champion pool caching for performance

---

5. Analytics Engine Service

Role: Data processing, statistical analysis, and predictive modeling **Scope**: All computational analysis and scoring algorithms

Responsibilities:

- Rank scoring and point value calculations
- Statistical analysis and trend detection
- Predictive modeling for draft recommendations

- Player performance analysis and benchmarking
- Team composition analysis
- Counter-pick suggestion algorithms
- Tournament scoring systems (LEPL, GCS)

Data Owned:

- Calculated scores and rankings
- Statistical models and algorithms
- Performance benchmarks and comparisons
- Prediction confidence intervals

Key Internal APIs:

```python
# Analytics APIs
POST   /internal/analytics/rank-scoring          # Calculate rank points
POST   /internal/analytics/predict-picks         # Draft predictions
POST   /internal/analytics/counter-suggestions   # Counter-pick analysis
POST   /internal/analytics/team-composition      # Team comp analysis
GET    /internal/analytics/meta-insights         # Meta trend insights
POST   /internal/analytics/tournament-scoring    # Tournament point calculation
```

Service Dependencies:

- **Champion Service**: Champion pool data for analysis
- **Player Service**: Player rank and performance data
- **Draft Service**: Historical draft data for predictions
- **Cache Service**: Cache computed results for performance

---

6. Live Draft Service

Role: Handles real-time draft functionality and pick/ban tracking **Scope**: Live draft sessions, real-time updates, and draft analysis

Responsibilities:

- Live draft session management
- Pick and ban phase tracking

- Real-time draft state synchronization
- Fearless draft mode support
- Draft timer management
- Pick/ban suggestions during live drafts
- Draft history and replay functionality

Data Owned:

- Active draft sessions and state
- Draft sequence and timing data
- Pick/ban history per session
- Draft configuration and rules

Key Internal APIs:

```python
# Draft Service APIs
POST   /internal/drafts                      # Create draft session
PUT    /internal/drafts/{draft_id}/pick      # Record pick/ban
GET    /internal/drafts/{draft_id}/state     # Get current draft state
POST   /internal/drafts/{draft_id}/suggest   # Get pick suggestions
WebSocket: /internal/drafts/{draft_id}/live # Real-time updates
```

Service Dependencies:

- **Analytics Service**: Pick prediction and counter-pick suggestions
- **Champion Service**: Available champion pools for suggestions
- **Team Service**: Team information for draft participants
- **Notification Service**: Real-time draft updates

---

7. Report Orchestration Service

Role: Orchestrates complex multi-service operations for report generation **Scope**: Scouting reports, tournament reports, and cross-service workflows

Responsibilities:

- Scouting report generation workflow orchestration
- Multi-service data aggregation

- Report template management
- Report caching and expiration
- Async report generation job management
- Report delivery and notification

Data Owned:

- Report metadata and generation status
- Report templates and configurations
- Generated report cache and links
- Report generation job queue

Key Internal APIs:

```python
# Report Service APIs
POST   /internal/reports/scouting/{target_team_id}  # Generate scouting report
GET    /internal/reports/{report_id}/status         # Check report status
GET    /internal/reports/{report_id}                # Get generated report
POST   /internal/reports/tournament                 # Tournament report
DELETE /internal/reports/{report_id}                # Delete/expire report
```

Service Dependencies:

- **Team Service**: Target team information
- **Player Service**: Player details for reports
- **Champion Service**: Champion pool data
- **Analytics Service**: Scoring and analysis
- **Scraper Service**: Fresh rank data
- **Sheets Service**: Report formatting and output

---

8. Tournament Management Service

Role: Handles tournament-specific logic and scoring systems **Scope**: Tournament operations, specialized scoring, and tournament data

Responsibilities:

- LEPL tournament processing and scoring

- GCS League tournament data management
- Tournament registration and validation
- Tournament-specific scoring algorithms
- Tournament bracket and match management
- Tournament reporting and analytics

Data Owned:

- Tournament configurations and rules
- Tournament participant data
- Tournament-specific scoring formulas
- Tournament match results and brackets

Key Internal APIs:

```python
# Tournament Service APIs
POST   /internal/tournaments/{type}/register        # Register for tournament
POST   /internal/tournaments/lepl/process-forms     # Process LEPL forms
GET    /internal/tournaments/{tournament_id}/standings # Tournament standings
POST   /internal/tournaments/{tournament_id}/score     # Calculate tournament scores
```

Service Dependencies:

- **Player Service**: Tournament participant management
- **Analytics Service**: Tournament scoring calculations
- **Scraper Service**: Verification of tournament data
- **Report Service**: Tournament report generation

---

9. External Data Integration Services

9a. Riot API Service

Role: Manages all Riot Games API interactions with proper rate limiting **Scope**: Riot API wrapper with intelligent caching and retry logic

Responsibilities:

- All Riot API endpoint interactions (Account, Summoner, Match, League)

- API rate limiting and queue management
- Automatic retry with exponential backoff
- API response validation and error handling
- Data transformation from Riot format to internal DTOs
- API key management and rotation

Key Internal APIs:

```python
# Riot API Service APIs
GET    /internal/riot/account/{riot_id}          # Get account by Riot ID
GET    /internal/riot/summoner/{puuid}           # Get summoner by PUUID
GET    /internal/riot/matches/{puuid}            # Get match history
GET    /internal/riot/ranked-stats/{summoner_id} # Get ranked statistics
POST   /internal/riot/batch-lookup               # Batch account lookups
```

9b. Web Scraper Service

Role: Handles all web scraping operations with proper resource management **Scope**: Multi-site scraping with Selenium orchestration

Responsibilities:

- OP.GG rank and statistics scraping
- League of Graphs peak rank scraping
- Rewind.lol champion mastery scraping
- GCS League tournament data scraping
- Selenium browser instance management
- Anti-detection and rate limiting for scraping
- Data validation and quality checks

Key Internal APIs:

```python
# Scraper Service APIs
POST   /internal/scrapers/opgg/rank              # Scrape OP.GG rank data
POST   /internal/scrapers/log/peak-ranks         # Scrape League of Graphs
POST   /internal/scrapers/rewind/champion-pool   # Scrape Rewind.lol
POST   /internal/scrapers/gcs/tournament-data    # Scrape GCS data
GET    /internal/scrapers/health                 # Scraper health status
```

9c. Google Sheets Service

Role: Manages Google Sheets integration with advanced formatting **Scope**: Sheet creation, data insertion, and rich formatting

Responsibilities:

- Google Sheets API interactions
- Sheet creation and template management
- Rich text formatting with hyperlinks
- Bulk data insertion and updates
- Advanced styling (colors, fonts, alignment)
- Permission management for sheets
- Sheet sharing and access control

Key Internal APIs:

```python
# Sheets Service APIs
POST   /internal/sheets/create                   # Create new sheet
POST   /internal/sheets/{sheet_id}/data          # Insert data
POST   /internal/sheets/{sheet_id}/format        # Apply formatting
GET    /internal/sheets/{sheet_id}/url           # Get sheet URL
PUT    /internal/sheets/{sheet_id}/permissions   # Manage permissions
```

---

10. Infrastructure Services

10a. Cache Service

Role: Manages data caching, freshness, and background refresh **Scope**: Redis caching with intelligent invalidation strategies

Responsibilities:

- Redis cache management and operations
- Data freshness tracking and validation
- Cache invalidation strategies
- Background cache refresh scheduling
- Cache performance monitoring
- Cache key management and namespacing

Key Internal APIs:

```python
# Cache Service APIs
GET    /internal/cache/{key}                    # Get cached data
PUT    /internal/cache/{key}                    # Store data in cache
DELETE /internal/cache/{key}                    # Invalidate cache entry
POST   /internal/cache/refresh/{pattern}        # Refresh cache pattern
GET    /internal/cache/stats                    # Cache performance stats
```

10b. Queue & Task Service

Role: Manages background job processing and task orchestration **Scope**: Celery task management with priority queues

Responsibilities:

- Celery task queue management
- Job scheduling and prioritization
- Task retry logic and failure handling
- Long-running task orchestration
- Task progress tracking and reporting
- Worker scaling and load balancing

Key Internal APIs:

```python
# Queue Service APIs
POST   /internal/queue/task                    # Queue new task
GET    /internal/queue/task/{task_id}/status   # Check task status
DELETE /internal/queue/task/{task_id}          # Cancel task
GET    /internal/queue/stats                   # Queue statistics
POST   /internal/queue/priority/{task_id}      # Change task priority
```

10c. Notification Service

Role: Handles events, real-time updates, and cross-service communication **Scope**: Event bus and real-time notification delivery

Responsibilities:

- Event publishing and subscription management
- WebSocket connection management for real-time updates
- Cross-service event routing
- Notification delivery (webhook, WebSocket, etc.)
- Event logging and replay functionality
- Real-time data synchronization

Key Internal APIs:

```python
# Notification Service APIs
POST   /internal/events/publish                # Publish event
POST   /internal/notifications/subscribe       # Subscribe to events
WebSocket: /internal/notifications/live        # Real-time updates
GET    /internal/events/history                # Event history
POST   /internal/webhooks/register             # Register webhook
```

Service Communication Patterns

Synchronous Communication (REST APIs)

- **Request/Response**: Direct service-to-service calls for immediate data needs
- **Used For**: Data queries, CRUD operations, validation
- **Example**: Team Service → Player Service to get roster information

Asynchronous Communication (Message Queues)

- **Event-Driven**: Services publish events when data changes
- **Used For**: Background processing, long-running tasks, data synchronization
- **Example**: Scraper Service publishes "rank_updated" event → Cache Service invalidates related cache

Real-Time Communication (WebSockets)

- **Bidirectional**: Live updates during draft sessions
- **Used For**: Live draft updates, real-time notifications
- **Example**: Draft Service broadcasts pick updates to all connected clients

Data Flow Example: Scouting Report Generation

```
1. Client Request → API Gateway → Report Service
2. Report Service → Team Service (get target team players)
3. Report Service → Scraper Service (queue fresh rank data)
4. Report Service → Champion Service (get champion pools)
5. Report Service → Analytics Service (calculate scores)
6. Report Service → Sheets Service (generate formatted report)
7. Report Service → Cache Service (cache report for reuse)
8. Report Service → Notification Service (notify completion)
9. Response → API Gateway → Client (report URL + status)
```

This microservice architecture maintains clear boundaries, enables independent scaling, and preserves all your existing League of Legends domain logic while making it maintainable and performant.