DarkAges MMO

Updated Remediation Analysis
Amended Recommendations

Update Date: January 30, 2026

Current Project Phase: 6 External Integration

Previous Analysis: Phase 0 Foundation

Codebase Size: ~30,000 lines (18K server + 3.5K client)

UPDATED ANALYSIS

1. Executive Summary - What's Changed

Major Project Evolution Since Initial Analysis

The DarkAges MMO project has **significantly evolved** from the Phase 0 Foundation state analyzed in the original remediation plan. The project is now in **Phase 6 External Integration** with:

- ✅ **Complete** Phases 0-5 implementation (~30,000 lines of code)
- ✅ **Complete** Phase 6 build system hardening
- ⏳ **Ready to Start** External library integration (GameNetworkingSockets, Redis, ScyllaDB, FlatBuffers)
- ⏳ **Planned** Phases 7-8 (Client Implementation, Production Hardening)

Key Changes Requiring Remediation Update

Aspect	Previous State	Current State	Impact on Recommendations
Project Phase	Phase 0 Foundation	Phase 6 External Integration	**MODIFIED**
Codebase Size	~5,000 lines (stubs)	~30,000 lines (implemented)	**MODIFIED**
CMake	Basic CMakeLists.txt	CMakeLists_enhanced.txt with options	**MODIFIED**
Documentation	Minimal docs	Comprehensive docs/API contracts	**NEW**
CI/CD	No GitHub Actions	Workflows folder exists	**NEW**
Database	Redis+ScyllaDB planned	Schemas defined, stubs ready	**MODIFIED**

Updated Priority Assessment

Area	Previous Priority	Updated Priority	Reasoning
External Library Integration	P1	P0	Blocking Phase 6 progress
Integration Testing	P1	P0	30K lines untested
Database Consolidation	P0	P1	Schemas already defined
Deployment Simplification	P0	P1	K8s not yet deployed
Godot Integration	P0	P0	Still blocking

2. Updated Research Area 1: Deployment Architecture

2.1 Revised Assessment

The project now has **comprehensive infrastructure documentation** including:

- API contracts between all modules (docs/API_CONTRACTS.md)
- Database schemas for Redis and ScyllaDB (docs/DATABASE_SCHEMA.md)
- Network protocol specification (docs/network-protocol/PROTOCOL_SPEC.md)
- Multiple docker-compose variants (dev, multi-zone, prod, phase6)

> **ADR-001-UPDATED: VPS Selection (Revised)**
>
> **Previous Decision:** Hetzner AX42 for all environments
>
> **Updated Decision:** Tiered approach based on project phase
>
> - **Development:** OVH VPS-3 ($15/month) - sufficient for integration testing
> - **Staging:** Hetzner EX44 (€44/month) - test production-like environment
> - **Production:** Hetzner AX42 (€49/month) - when ready for players
>
> **Rationale:** With external integration as current focus, start with lower-cost VPS for development. Scale up only after successful integration testing.

2.2 Docker Compose Strategy - Aligned with Project State

The project already has multiple docker-compose files. Recommended consolidation:

File	Purpose	Recommendation
docker-compose.yml	Development (Redis+ScyllaDB)	**Keep** - for Phase 6 integration
docker-compose.phase6.yml	Phase 6 specific	**Keep** - external deps testing
docker-compose.dev.yml	Multi-zone dev	**Simplify** - single zone for now
docker-compose.multi-zone.yml	Multi-zone production	**Defer** - not needed until 500+ players
docker-compose.prod.yml	Full production stack	**Defer** - overkill for Phase 6

> **ADR-002-NEW: Database Strategy for Phase 6**

> **Decision:** Proceed with Redis+ScyllaDB as designed for Phase 6 integration testing.
>
> **Rationale:** The schemas are already defined and stubs are ready. The complexity is already "baked in" - changing now would delay Phase 6.
>
> **Consolidation Path:** After Phase 6 integration testing, evaluate if PostgreSQL-only can handle the load before production deployment.
>
> **Timeline:** Re-evaluate at Phase 8 (Production Hardening) - approximately 8-12 weeks.

3. Updated Research Area 2: Godot 4 + C++ Integration

3.1 Current State Assessment

The client implementation has progressed significantly:

- 3,500+ lines of Godot 4.x client code
- Client-side prediction implemented
- Interpolation system in place
- Network protocol specification documented

> **ADR-003-UPDATED: Networking Integration (Revised)**
>
> **Previous Decision:** Evaluate GameNetworkingSockets vs ENet
>
> **Updated Decision:** Proceed with GameNetworkingSockets as planned
>
> **Rationale:** The CMakeLists_enhanced.txt already has ENABLE_GNS option. The stubs are in place. Switching now would waste integration effort.
>
> **Implementation Order:**
>
> 1. Complete GameNetworkingSockets integration (WP-6-1)
> 2. Implement FlatBuffers protocol (WP-6-4)
> 3. Wire up Godot client to use the protocol

3.2 Integration Testing Strategy

With 30,000 lines of untested code, integration testing is now **P0 CRITICAL**.

```
# Recommended integration test sequence
Phase 6.1: GameNetworkingSockets + FlatBuffers (no database)
  - Test: Server accepts connections, exchanges messages
  - Duration: 1 week

Phase 6.2: Add Redis hot-state
  - Test: Player sessions, positions cached
  - Duration: 1 week

Phase 6.3: Add ScyllaDB persistence
  - Test: Player profiles save/load
  - Duration: 1 week

Phase 6.4: Full integration
  - Test: End-to-end with Godot client
  - Duration: 2 weeks

Phase 6.5: Stress testing
  - Test: 10, 50, 100 concurrent connections
  - Duration: 1 week
```

4. Updated Research Area 3: ECS & Performance

4.1 Current ECS Implementation

The project has **18,000+ lines of server code** with:

- Complete ECS architecture with 10+ components
- Spatial hash collision detection
- Area of Interest (AOI) 3-tier system
- Delta compression for networking
- Lag-compensated combat system

> **ADR-004-NEW: ECS Performance Strategy**
>
> **Decision:** Keep current EnTT implementation. Do NOT refactor to groups yet.
>
> **Rationale:** With 18,000 lines already written, refactoring carries too much risk. Profile first, optimize second.
>
> **Performance Validation Plan:**
>
> 1. Build with ENABLE_PROFILING=ON
> 2. Run 100-player stress test
> 3. Identify actual bottlenecks via Perfetto traces
> 4. Optimize only proven hotspots

5. Updated Research Area 4: Anti-Cheat

5.1 Current Anti-Cheat Implementation

The project already has:

- Kinematic movement system with anti-cheat
- Server-authoritative validation
- Movement speed limits (6 m/s normal, 9 m/s sprint)

> **ADR-005-NEW: Anti-Cheat Enhancement**
>
> **Decision:** Enhance existing anti-cheat with server-side rewind
>
> **Current State:** Basic speed/teleport validation exists
>
> **Enhancement:** Add lag compensation for combat validation

```
// Pseudocode for lag-compensated combat
class LagCompensatedCombat {
    struct HistorySnapshot {
        uint32_t timestamp;
        Vector3 position;
        BoundingBox hitbox;
    };

    std::deque<HistorySnapshot> player_history;

    bool validate_hit(PlayerID attacker, PlayerID target, uint32_t client_time) {
        // Rewind target to when attacker fired
        int32_t rtt = estimate_rtt(attacker);
        uint32_t fire_time = get_server_time() - (rtt / 2);

        auto target_state = interpolate_history(target, fire_time);
        auto attacker_state = interpolate_history(attacker, fire_time);

        // Validate line of sight, range, etc.
        return check_hit_valid(attacker_state, target_state);
    }
};
```

6. Updated Research Area 5: CI/CD & Workflow

6.1 Build System Assessment

The project now has:

- CMakeLists_enhanced.txt with feature flags
- .github/workflows/ folder (workflows need implementation)
- tools/ folder with build, chaos, stress-test utilities

> **ADR-006-NEW: CI/CD Implementation**
>
> **Decision:** Implement GitHub Actions workflows for Phase 6 integration
>
> **Priority Workflows:**
>
> 1. **ci.yml** - Build server with all options enabled
> 2. **integration-test.yml** - Docker Compose + test suite
> 3. **godot-export.yml** - Build and export Godot client
>
> **CMake Integration:**
>
> ```
> # Use CMakeLists_enhanced.txt for CI
> cmake -DCMAKE_BUILD_TYPE=Release \
>       -DENABLE_GNS=ON \
>       -DENABLE_REDIS=ON \
>       -DENABLE_SCYLLA=ON \
>       -DENABLE_FLATBUFFERS=ON \
>       -DFETCH_DEPENDENCIES=ON \
>       -B build
> ```

7. Updated Research Area 6: MVP Scope

7.1 Revised Phase 6-8 Timeline

Phase	Duration	Exit Criteria	Risk Level
6.1 GNS Integration	1 week	Server accepts UDP connections	Medium
6.2 Redis Integration	1 week	Session data persists	Low
6.3 ScyllaDB Integration	1 week	Player profiles save/load	Medium
6.4 FlatBuffers Protocol	1 week	Messages serialize correctly	Low
6.5 Integration Testing	2 weeks	10 players, 1 hour, zero crashes	High
7 Client Implementation	4 weeks	Godot client connects, moves	High
8 Production Hardening	2 weeks	100 players, monitoring, backups	Medium

7.2 Success Criteria Revisited

Criterion	Original Target	Updated Target	Status
DevOps Reduction	docker-compose up	docker-compose -f docker-compose.phase6.yml up	**Achieved**
Build Simplicity	<2 minutes	<5 minutes (with deps)	**Pending**
Deterministic Proof	Fixed-point demo	Integration test passes	**Pending**
Cost Ceiling	$50/month	$50/month (production)	**On track**

8. Updated Risk Matrix

Risk	Severity	Likelihood	Mitigation
External library integration fails	High	Medium	Prototype each lib separately first
30K lines have hidden bugs	High	High	Incremental integration testing
Godot client integration issues	High	Medium	Early protocol testing
Performance at 100 players	Medium	Unknown	Profile early, optimize proven hotspots
Database complexity overhead	Medium	Low	Re-evaluate at Phase 8

9. Amended Implementation Roadmap

Immediate Actions (Next 2 Weeks)

1. P0 Implement GitHub Actions CI workflow
2. P0 Complete GameNetworkingSockets integration (WP-6-1)
3. P0 Create integration test harness
4. P1 Set up OVH VPS-3 for development testing

Short-Term Actions (Weeks 3-6)

1. P0 Complete Redis integration (WP-6-2)
2. P0 Complete ScyllaDB integration (WP-6-3)
3. P0 Complete FlatBuffers protocol (WP-6-4)
4. P1 Run 10-player integration test

Medium-Term Actions (Weeks 7-12)

1. P0 Godot client integration (Phase 7)
2. P1 50-player stress test
3. P1 Performance profiling and optimization
4. P2 Production deployment planning (Phase 8)

10. Summary of Changes from Original Analysis

Original Recommendation	Updated Recommendation	Reason
Replace Redis+ScyllaDB with PostgreSQL	Proceed with Redis+ScyllaDB for Phase 6	Schemas already defined, stubs ready
Single VPS for all environments	Tiered VPS approach	Cost optimization for development
Refactor EnTT to use groups	Profile first, optimize second	18K lines already written
Basic GitHub Actions	Comprehensive CI for Phase 6	Critical for integration testing
Anti-cheat from scratch	Enhance existing implementation	Anti-cheat stubs already exist

Bottom Line

The project has made **significant progress** since the initial Phase 0 analysis. The codebase is now substantial (30K lines) and well-documented. The primary risk has shifted from *"over-engineering"* to *"integration complexity with large untested codebase."*

Key Success Factor: Complete Phase 6 external integration with comprehensive testing before moving to client integration. The database consolidation and deployment simplification can be revisited at Phase 8 when production deployment approaches.