DarkAges MMO

Zero-Budget Architecture Remediation & Phase 0 Optimization Strategy

Research Areas: 6 Critical Domains

Target: 100-1000 Concurrent Players

Budget Ceiling: $50/month

Date: January 30, 2026

1. Executive Summary

The DarkAges MMO repository reveals a sophisticated but potentially over-engineered architecture for zero-budget, solo/small-team development. Current stack includes C++20/EnTT server, Godot 4 client, GameNetworkingSockets, FlatBuffers, Redis, ScyllaDB, and Kubernetes manifests targeting 100-1000 concurrent players.

Key Findings

Area	Current State	Recommendation	Priority
Deployment	K8s + ScyllaDB	Single VPS + Docker Compose	P0
Database	Redis + ScyllaDB	SQLite/PostgreSQL single node	P0
ECS	EnTT (complex)	Keep EnTT, simplify usage	P1
Networking	GameNetworkingSockets stub	Complete integration	P0
Anti-Cheat	Basic stubs	Server rewind + heuristics	P1
CI/CD	Build scripts only	GitHub Actions workflow	P1

2. Zero-Budget Deployment Architectures

2.1 VPS Comparison & Recommendation

Provider	Model	CPU	RAM	Price	PassMark
Hetzner	AX42	AMD Ryzen 7 PRO 8700GE	64GB	€49/mo	27,882
Hetzner	EX44	Intel i5-13500	64GB	€44/mo	31,857
OVH	VPS-3	8 vCores	24GB	$15/mo	N/A

> **ADR-001: VPS Selection**
>
> **Decision:** Hetzner AX42 for production, OVH VPS-3 for development.
>
> **Rationale:** AX42 offers best price/performance at €49/month with 64GB RAM and high single-thread performance (critical for game servers).

2.2 Database Consolidation Strategy

Feature	SQLite (WAL)	PostgreSQL	Redis+ScyllaDB
Setup Complexity	Zero	Low	High
Write Throughput	~50K TPS	~100K TPS	~1M TPS
Memory Footprint	Minimal	Moderate	High
Operational Cost	None	Low	High

> **ADR-002: Database Consolidation**
>
> **Decision:** Replace Redis+ScyllaDB with PostgreSQL for Phase 0.
>
> **Rationale:** For under 200 concurrent players, PostgreSQL single instance can handle all hot-state and persistence needs.

2.3 Docker Compose Manifest

```
version: '3.8'
services:
  postgres:
    image: postgres:16-alpine
    environment:
      POSTGRES_USER: darkages
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: darkages
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    command: >
      postgres
      -c shared_buffers=1GB
      -c effective_cache_size=3GB
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 2G
  server:
    build:
      context: .
      dockerfile: infra/dockerfile.server
    ports:
      - "7777:7777/udp"
    depends_on:
      - postgres
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 4G
volumes:
  postgres_data:
```

3. Godot 4 + C++ Server Integration

3.1 Serialization Strategy

Library	Serialization	Deserialization	Zero-Copy
FlatBuffers	Fast	Instant	Yes
Protocol Buffers	Fast	Fast	No
StreamPeerBuffer	Manual	Manual	No

> **ADR-003: Serialization Strategy**
>
> **Decision:** Keep FlatBuffers for server-side, use Godot's ByteBuffer for client.
>
> **Rationale:** FlatBuffers' zero-copy deserialization is critical for server performance at 60Hz tick rate.

3.2 Deterministic Simulation

> **ADR-004: Deterministic Math**
>
> **Decision:** Use custom int64_t fixed-point (32.32) implementation in C++ server.
>
> **Rationale:** Cross-platform determinism requires controlling the entire math pipeline.

3.3 Headless Testing

```
# Run Godot in headless mode for CI
godot --headless --path . --import --quit
export GODOT_DISABLE_LEAK_CHECKS=1
godot --headless -d     --display-driver headless     --audio-driver Dummy     --
disable-render-loop     --path .     -s res://addons/gut/gut_cmdln.gd     -
gdir=res://tests     -gexit
```

4. Simplified ECS & Performance

4.1 ECS Library Comparison

Framework	1 Entity	1K Entities	16K Entities	1M Entities
EnTT (group)	89ns	18us	403us	35ms
EnTT (stable)	131ns	21us	365us	26ms
Flecs	1741ns	10us	120us	19ms

> **ADR-005: ECS Framework**
>
> **Decision:** Continue using EnTT, but adopt group/stable iteration patterns.
>
> **Rationale:** EnTT is already integrated. Groups provide 4-5x performance over runtime views.

4.2 Memory Management with std::pmr

```
#include <memory_resource>
class GameWorld {
    alignas(64) char frame_buffer[1024 * 1024];
    std::pmr::monotonic_buffer_resource frame_arena;
public:
    GameWorld() : frame_arena(frame_buffer, sizeof(frame_buffer)) {}
    void tick(float delta) {
        frame_arena.release();  // Reset at start of frame
        std::pmr::vector<EntityUpdate> updates(&frame_arena);
        // Process systems...
    }
};
```

5. Zero-Budget Anti-Cheat

5.1 Movement Validation with Lag Compensation

```
class MovementValidator {
    static constexpr float MAX_SPEED = 6.0f;
    static constexpr float MAX_SPRINT = 9.0f;
    static constexpr float TELEPORT_THRESHOLD = 10.0f;

    bool validate_movement(uint32_t client_time, Vector3 claimed_pos) {
        uint32_t server_time = get_server_time();
        int32_t rtt = estimate_rtt(player_id);
        uint32_t action_time = server_time - (rtt / 2);

        Vector3 historical_pos = interpolate_position(action_time);
        float max_distance = MAX_SPEED * (rtt / 1000.0f + TICK_RATE);
        float actual_distance = (claimed_pos - historical_pos).length();

        return actual_distance <= max_distance * 1.2f;  // 20% tolerance
    }
};
```

5.2 Statistical Detection

```
class HeuristicAntiCheat {
    void record_input(PlayerID id, uint32_t timestamp, InputState input) {
        // Detect inhuman reaction times (< 100ms consistently)
        float reaction = calculate_reaction_time(id, timestamp);
        if (reaction < 0.1f) stats[id].impossible_actions++;

        // Detect perfect input patterns (bots)
        float variance = calculate_input_variance(id, input);
        if (variance < 0.01f) flag_for_review(id, "suspicious consistency");

        if (stats[id].impossible_actions > 10) {
            kick_player(id, "suspicious gameplay patterns");
        }
    }
};
```

6. CI/CD & Workflow

6.1 CMake Presets

```
{
  "version": 3,
  "configurePresets": [
    {
      "name": "ci",
      "generator": "Ninja",
      "cacheVariables": {
        "CMAKE_BUILD_TYPE": "Release",
        "CMAKE_CXX_COMPILER_LAUNCHER": "ccache"
      }
    }
  ]
}
```

6.2 GitHub Actions Workflow

```
name: CI
on: [push, pull_request]
jobs:
  server:
    runs-on: ubuntu-22.04
    steps:
      - uses: actions/checkout@v4
      - uses: hendrikmuhs/ccache-action@v1.2
      - run: cmake --preset ci
      - run: cmake --build --preset ci --parallel
      - run: ctest --preset default
```

6.3 Monitoring

> **ADR-007: Monitoring Strategy**
>
> **Decision:** Start with StatsD + simple console dashboard, migrate to Prometheus later.
>
> **Rationale:** StatsD has minimal performance impact (UDP fire-and-forget).

7. MVP Scope Definition

7.1 Phase 0 Exit Criteria

Week	Goal	Acceptance Criteria
1	Server foundation	Single player connects, moves, disconnects cleanly
2	Client integration	Godot client displays player movement
3	Multiplayer basics	10 concurrent players, no desyncs
4	Validation	48-hour playtest with 10 players, zero crashes

7.2 Network Budget

```
// Position update: ~37 bytes + headers
// 20 players * 60Hz * 37 bytes = 44.4 KB/s per client
// Server total: ~7.1 Mbps outbound
// With delta compression + AOI: ~2-3 Mbps
```

8. Risk Matrix & ADRs

Technical Debt Assessment

Risk	Severity	Acceptable?	Mitigation
Single point of failure	High	Yes (Phase 0)	Document manual recovery
No automated backups	High	Yes (Phase 0)	Daily pg_dump cron
Memory leaks	High	No	ASan in CI
Desynchronization	High	No	Periodic state sync

ADR Summary

ADR	Decision	Status
ADR-001	Hetzner AX42 for production	Recommended
ADR-002	PostgreSQL replaces Redis+ScyllaDB	P0
ADR-003	FlatBuffers server, ByteBuffer client	Keep current
ADR-004	Custom int64_t fixed-point math	P1
ADR-005	EnTT with group iteration	Refactor
ADR-006	Keep custom spatial hash	Keep current
ADR-007	StatsD for monitoring	P2

9. Implementation Roadmap

Week	Focus	Deliverables
1	Infrastructure simplification	Docker Compose, PostgreSQL migration
2	Networking integration	Godot ↔ C++ handshake working
3	Determinism + anti-cheat	Fixed-point movement, validation
4	CI/CD + testing	GitHub Actions, 10-player test

Recommended Libraries

Purpose	Library	License
Fixed-point math	libfixmath	MIT
Game networking	GameNetworkingSockets	BSD
Serialization	FlatBuffers	Apache-2.0
ECS	EnTT	MIT
Testing	Catch2	BSL-1.0

10. References

1. VPSBenchmarks. (2024). Hetzner AX42 Performance Analysis.
2. abeimler/ecs_benchmark. (2024). ECS Framework Benchmarks. GitHub.
3. Godot Engine Documentation. (2024). High-level Multiplayer.
4. Google FlatBuffers. (2024). FlatBuffers Benchmarks.
5. Unity Documentation. (2024). Dealing with Latency - Server Side Rewind.
6. Red Hat Developers. (2021). Memory Error Checking in C and C++.
7. Hafiz, A. (2021). Hot Code Reloading with libc.