RIFT Tomography Deployment Guide

Trident SemVerX Package Manager for OBINexus

"For what is yet to be, I became."

Table of Contents

Quick Start (5 Minutes)

Prerequisites

- GCC 7+ or Clang 10+ (for C11 support)
- Git
- Linux, macOS, or WSL/MSYS2 on Windows

Installation

```bash
```

```
# Clone main repository
git clone https://github.com/obinexus/rift.git
cd rift

# Build everything
chmod +x rift_build.sh
./rift_build.sh all

# Run demo
./build/bin/rift_demo
```

Expected Output:

```
✓ Obi (Heart/Soul) initialized
  Heartbeat: +

[Test 1] Diamond Conflict Detection
  Status: ✗ UNBOUND (diamond conflict detected)
  → Application remains safe, no inconsistent state

[Test 2] Hot-Swap Resolution
  Status: ✓ BOUND (hot-swap successful)
  → Runtime healed without restart
```

Repository Structure

Main Repository: `obinexus/rift`

```
rift/
├── include/
│   └── rift/
│       ├── rift.h              # Core API
│       ├── riftbridge.h        # Bridge layer
│       └── riftest.h           # Testing framework
├── src/
│   ├── core/
│   │   ├── eze_trident.c       # Trident topology
│   │   ├── uche_resolver.c     # Dependency resolver
│   │   └── obi_semver.c        # SemVerX parser
│   ├── bridge/
│   │   ├── c_bridge.c
│   │   ├── cpp_bridge.cpp
│   │   ├── cs_bridge.cs
```

```
│   │   └── iota_matrix.c        # Row/col matrix
│   ├── encoding/
│   │   └── rift_open.c          # Polyglot file I/O
│   └── demo/
│       └── rift_demo.c
├── build/                       # Generated artifacts
├── rift_build.sh                # Main build script
└── README.md
```

Bridge Repository: `obinexus/riftbridge`

```
riftbridge/
├── include/
│   └── riftbridge/
│       ├── package.h            # Package definitions
│       ├── registry.h           # Registry operations
│       └── polarity.h           # Polarity encoding
├── src/
│   ├── registry/
│   │   ├── eze_registry.c       # Local registry
│   │   ├── uche_fetch.c         # Remote fetch
│   │   └── obi_resolve.c        # Resolution engine
│   └── encoding/
│       └── sparse_encode.c      # 2→1 duplex encoding
└── riftbridge_build.sh
```

Step-by-Step Setup

Step 1: Clone Repositories

bash

```
# Create workspace
mkdir -p ~/obinexus
cd ~/obinexus

# Clone main engine
git clone https://github.com/obinexus/rift.git

# Clone bridge (optional, for package management)
git clone https://github.com/obinexus/riftbridge.git
```

Step 2: Build RIFT Core

```bash
cd rift

# Option A: Full build (C + C++ + C#)
./rift_build.sh all

# Option B: C only (fastest)
./rift_build.sh c

# Option C: Custom stages
./rift_build.sh c     # Build C library
./rift_build.sh cpp   # Add C++ bridge
./rift_build.sh demo # Build demo
```

Step 3: Verify Installation

```bash
# Check library exists
ls -lh build/lib/librift.a

# Run tests
./rift_build.sh test
```

Step 4: Build RIFTBridge (Optional)

```bash
cd ../riftbridge

# Build package manager
chmod +x riftbridge_build.sh
./riftbridge_build.sh

# Verify
ls -lh build/lib/libriftbridge.a
```

Usage Examples

Example 1: Basic Trident Diamond Resolution

File: `myapp.c`

c

```c
#include <rift/rift.h>
#include <stdio.h>

int main(void) {
    /* Create versions */
    RiftVersion stable = rift_semver_parse("4.stable.17.beta.2.stable");
    RiftVersion experimental = rift_semver_parse("4.experimental.17.beta.2.stable");

    /* Create trident nodes */
    RiftEzeNode *dep_b = rift_eze_create("lodash", stable);
    RiftEzeNode *dep_c = rift_eze_create("lodash", experimental);
    RiftEzeNode *app = rift_eze_create("myapp", stable);

    /* Connect inputs */
    app->incoming[0] = dep_b;
    app->incoming[1] = dep_c;

    /* Test for diamond conflict */
    if (rift_eze_bind(app)) {
        printf("✓ Dependencies resolved\n");
    } else {
        printf("✗ Diamond conflict: %s@", dep_b->name);
        rift_semver_print(stable);
        printf(" vs ");
        rift_semver_print(experimental);
        printf("\n");
    }

    /* Cleanup */
    free(dep_b);
    free(dep_c);
    free(app);

    return 0;
}
```

Compile:

bash

```bash
gcc -I rift/include myapp.c -L rift/build/lib -lrift -o myapp
./myapp
```

Example 2: Row/Column Semantic Intent

c

```c
#include <rift/riftbridge.h>
#include <stdio.h>

int main(void) {
    /* Create 4x4 matrix for parsing context */
    IotaMatrix *matrix = iota_matrix_create(4, 4);

    /* Row 0: Function declaration */
    iota_matrix_set(matrix, 0, 0, RIFT_POLARITY_POSITIVE, 0.95);  /* 'int' */
    iota_matrix_set(matrix, 0, 1, RIFT_POLARITY_POSITIVE, 0.92);  /* 'main' */
    iota_matrix_set(matrix, 0, 2, RIFT_POLARITY_POSITIVE, 0.88);  /* '(' */

    /* Row 1: Statement inside function */
    iota_matrix_set(matrix, 1, 1, RIFT_POLARITY_POSITIVE, 0.90);  /* nested */

    /* Check confidence threshold */
    IotaCell cell = iota_matrix_get(matrix, 0, 0);
    if (cell.confidence >= 0.85) {
        printf("✓ High confidence parse at (%u,%u)\n", cell.row, cell.col);
    }

    iota_matrix_destroy(matrix);
    return 0;
}
```

Example 3: Hot-Swap Resolution

c

```
#include <rift/rift.h>
#include <stdio.h>
#include <unistd.h>

int main(void) {
    RiftVersion v1 = rift_semver_parse("1.stable.0.stable.0.stable");
    RiftVersion v2 = rift_semver_parse("2.experimental.0.beta.0.beta");

    RiftEzeNode *source_a = rift_eze_create("lib", v1);
    RiftEzeNode *source_b = rift_eze_create("lib", v2);
    RiftEzeNode *app = rift_eze_create("myapp", v1);

    app->incoming[0] = source_a;
    app->incoming[1] = source_b;

    printf("Initial state (conflict expected):\n");
    rift_eze_bind(app);
    printf("  Bound: %s\n", app->is_bound ? "YES" : "NO");

    /* Simulate fix landing */
    printf("\nHot-swapping source_b to stable...\n");
    source_b->version = v1;

    rift_eze_bind(app);
    printf("  Bound: %s\n", app->is_bound ? "YES" : "NO");
    printf("  ✓ Runtime healed without restart\n");

    free(source_a);
    free(source_b);
    free(app);

    return 0;
}
```

Integration with Existing Projects

As a Static Library

bash

```
# Copy headers
cp -r rift/include/rift /usr/local/include/

# Copy library
cp rift/build/lib/librift.a /usr/local/lib/

# Link in your project
gcc myapp.c -lrift -o myapp
```

As a Submodule

```bash
# In your project root
git submodule add https://github.com/obinexus/rift.git deps/rift

# Build
cd deps/rift
./rift_build.sh c

# Link in your Makefile
CFLAGS += -I deps/rift/include
LDFLAGS += -L deps/rift/build/lib -lrift
```

CMake Integration

CMakeLists.txt:

```cmake
cmake_minimum_required(VERSION 3.10)
project(MyApp C)

# Add RIFT as subdirectory
add_subdirectory(deps/rift)

add_executable(myapp myapp.c)
target_link_libraries(myapp rift)
```

Troubleshooting

Build Errors

Problem: `rift.h: No such file or directory`

Solution:

```bash
# Ensure include path is correct
export C_INCLUDE_PATH=$PWD/include:$C_INCLUDE_PATH
```

Problem: `undefined reference to 'rift_eze_create'`

Solution:

```bash
# Link library explicitly
gcc myapp.c -L build/lib -lrift -o myapp
```

Runtime Issues

Problem: Segmentation fault in `rift_eze_bind`

Solution:

- Ensure both `incoming[0]` and `incoming[1]` are non-NULL
- Check node was created with `rift_eze_create`

Problem: Version parsing returns zeros

Solution:

- Verify format: `major.state.minor.state.patch.state`
- Example: `"4.stable.17.beta.2.stable"` not `"4.17.2"`

Constitutional Compliance

This implementation follows the **OBINexus NT License v1.0** and Constitutional Framework:

Core Principles

1. **Human Rights Respect (Anti-Harassment)**
   - No coordinated abuse of contributors
   - Ableist discourse is prohibited
   - Psychological safety for neurodivergent innovators
2. **Constructive Engagement (Anti-Entitlement)**

- Contribute solutions, not just complaints
- Patches > feature requests
- Financial support for enterprise use

3. **OpenSense Sustainability**
    - Core software remains free
    - Enterprise support via separate agreement
    - Funds feed back into open-source development

Naming Philosophy

Igbo Cultural Integration:

- **Eze** (𐌄𐌆): Power, leader, authority
    - Used for trident nodes (consensus leaders)
- **Uche** (𐌖𐌒𐌄): Knowledge, wisdom, intelligence
    - Used for resolver logic (learning from past)
- **Obi** (𐌀𐌁𐌉): Heart, soul, core essence
    - Used for central coordination (system heartbeat)
- **Iota** (ιωτα): Shared power, smallest unit
    - Used for matrix cells (distributed consensus)

What's Next?

Immediate Goals (Week 1-2)

- [ ] Port to Rust + WASM for browser execution
- [ ] Implement `uche_resolver.c` (learning resolver)
- [ ] Add cryptographic package signing

Short Term (Month 1-3)

- [ ] Registry simulator with 100k packages
- [ ] npm-compatible `semverx publish` command
- [ ] Python/Node.js drop-in resolver

Long Term (Month 3-12)

- [ ] OBINexus public registry (testnet)

- [ ] Constitutional compliance engine integration
- [ ] Zero-trust security layer

Community & Support

GitHub: https://github.com/obinexus/rift
Medium: https://obinexus.medium.com
Substack: https://obinexus.substack.com
Twitter: https://twitter.com/obinexus

License: OBINexus NT License v1.0 (MIT-compatible with constitutional terms)

Computing from the Heart. Building with Purpose. Running with Heart.

— Nnamdi Michael Okpala
Founder, OBINexus Computing
4 January 2025