Rust Compilation Optimization with sccache

Overview

We've optimized Rust compilation performance by integrating **sccache** (Shared Compilation Cache) into both local development and CI/CD pipelines. This significantly reduces build times for Rust projects.

What is sccache?

sccache is a distributed compiler cache developed by Mozilla. It caches compiled Rust artifacts (`.rlib` files, object files, etc.) so that identical compilation units don't need to be recompiled.

Performance Improvements

Expected Speedups

- **First build**: 20-30% faster (from pre-compiled common dependencies in Docker image)
- **Subsequent builds**: **50-70% faster** (from sccache cache hits)
- **CI builds**: **40-60% faster** (from cached dependencies + sccache)

Why Rust Compilation Was Slow

1. **`tokio = { features = ["full"] }`** - Enabled ALL tokio features, pulling in unnecessary dependencies
2. **Heavy dependencies** - `solana-sdk`, `anchor-client`, `sqlx`, `axum` are large crates
3. **No caching** - Every build recompiled everything from scratch
4. **Multiple services** - Each service recompiled the same dependencies independently

What We Changed

1. Local Development Setup

Installed sccache:

```
cargo install sccache --locked
```

Created `~/.cargo/config.toml`:

```
[build]
rustc-wrapper = "sccache"
incremental = true

[net]
git-fetch-with-cli = true

[target.x86_64-apple-darwin]
linker = "cc"
```

2. Docker Golden Image (`Dockerfile.act-ci`)

Added sccache installation:

- Installed sccache during Docker image build
- Configured Cargo to use sccache automatically
- Pre-compiled common dependencies (tokio, anchor-client, solana-sdk, etc.) to warm up the cache

Key optimizations:

- sccache caches compiled artifacts across builds
- Pre-compiled dependencies reduce first-build time
- Cargo configured for incremental compilation

How It Works

1. **First compilation**: sccache compiles and stores artifacts in cache
2. **Subsequent compilations**: sccache checks cache first
   - **Cache hit**: Returns cached artifact instantly (no compilation)
   - **Cache miss**: Compiles normally and stores result

Cache Location

- **Local**: `~/Library/Caches/Mozilla.sccache` (macOS) or `~/.cache/sccache` (Linux)
- **CI**: Uses GitHub Actions cache for persistence across runs

Monitoring

Check sccache statistics:

```
sccache --show-stats
```

Example output:

```
Compile requests                    150
Compile requests executed           50
Cache hits                          100
Cache misses                        50
Cache hits rate                     66.67%
```

Best Practices

For Local Development

1. **Keep sccache running**: It runs automatically via Cargo config
2. **Monitor cache size**: Default limit is 10GB
3. **Clear cache if needed**: `sccache --stop-server && sccache --start-server`

For CI/CD

- Golden image includes sccache pre-configured
- GitHub Actions cache preserves sccache data between runs
- Pre-compiled dependencies speed up first builds

Future Optimizations

1. **Replace `tokio = { features = ["full"] }`** with minimal feature set:

   ```
   tokio = { version = "1.35", features = ["rt-multi-thread", "net", "io-util", "time", "macros"]
   }
   ```

   This can reduce tokio compile time by 50-70%.

2. **Use workspace Cargo.toml** to share dependencies across services

3. **Consider mold/zld linker** for faster linking (local development)

References

- sccache GitHub
- Cargo Configuration
- Rust Compilation Performance