DevOps S25-3 Team

- Jonathan Bezeau
- Anzhelika Kostyuk
- Arsh Chauhan
- Jindo Kim

Unity WebGL Build Time & Size Reduction via Project Settings and PID-Level Profiling

- What Prompted the Use of PID Profiling and Unity Build Configuration Tuning
  - Current Pipeline Execution Pattern
  - Challenge – Clean Builds Without Sacrificing Delivery Speed
- PID-Level Profiling for WebGL Build
- Project Settings That Impact WebGL Build Time and Size
- Build Time Reduction with Optimized Unity Settings
- Toward Cloud-Based(Serverless) Unity Build Pipelines

What Prompted the Use of PID Profiling and Unity Build Configuration Tuning

Current Pipeline Execution Pattern

🔹 Problem: Resource Bottleneck

- Unity WebGL builds consume **high CPU & memory**
- 3+ pipelines on 8-core / 32GB VM → 💥 **system crash**

🔹 Current Approach: Artifact Reuse

- ❌ Build results could be inconsistent
- ❌ Old or broken files were reused without warning
- ❌ Couldn't ensure a fully clean build

3 🔹 Pipeline Step Breakdown & Timing

- Each pipeline runs **5 sequential Unity Editor steps**:
  → ✅ Linting → EditMode → PlayMode → Code Coverage → WebGL Build
- With **artifact reuse**, total duration:
  → ⏱️ ~6 minutes per pipeline (RT project baseline)
- With **clean build**, estimated duration:
  → ⏱️ **25+ minutes** (RT project baseline)

Challenge: Clean Builds Without Sacrificing Delivery Speed

1

🔹 Parallel Execution via Unity Containers in the Cloud

- Each Unity step (Linting, EditMode, PlayMode, Code Coverage, WebGL) → can run in **parallel containers**, not sequentially

2

🔹 WebGL as the Bottleneck

- Among all steps, WebGL clean build is **most resource-heavy**
- On CPU 8 / RAM 32 container → ⏱️ **13–14 minutes**

3

🔹 Pipeline Time Driven by Slowest Step

- When all steps run in parallel, **WebGL defines total pipeline duration**

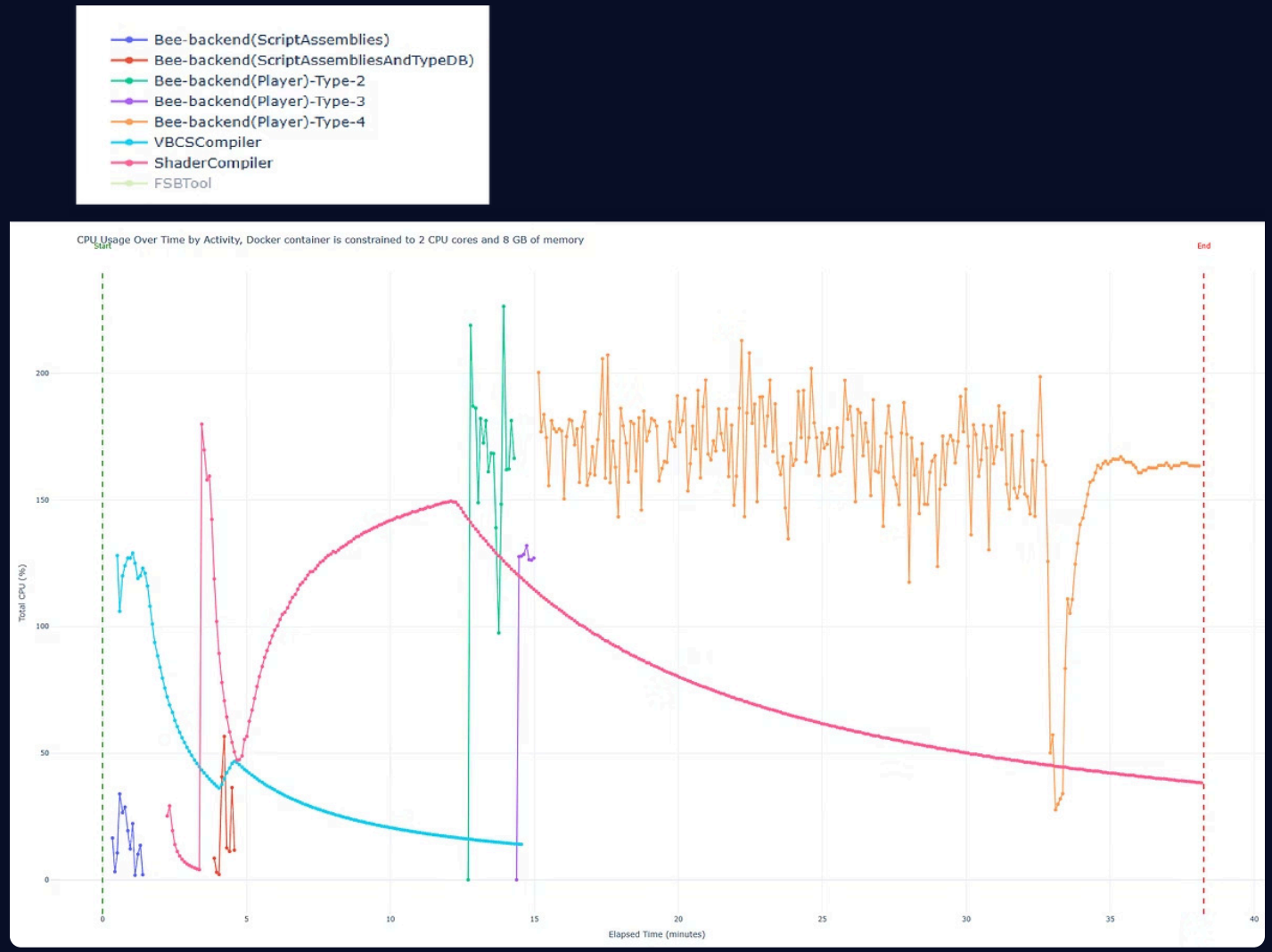
4

🎯 Why Profiling and Configuration Tuning Were Needed

- To hit both targets:
  → ✅ **Reproducibility** (clean builds)
  → ✅ **Speed** (match current delivery time)
- Solution: **Project setting tuning** + **PID-level process profiling**
  → 🎯 One stone, two birds

PID-Level Profiling for WebGL Build

- Every 5 seconds / at 5-second intervals
- All active processes (PIDs)
- Top 8 resource-consuming parent-level tasks
- CPU usage over time



Project Settings That Impact WebGL Build Time and Size

managedStrippingLevel

- Defines how aggressively Unity strips unused managed (C#) code.

WebGL.emscriptenArgs

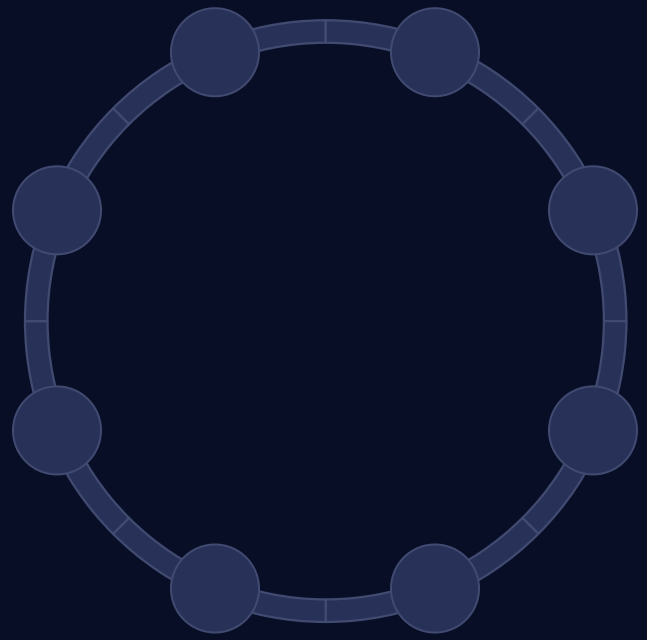
- Controls WebAssembly optimization level

Il2CppCodeGeneration

- Controls how IL code is converted into C++ in IL2CPP builds

WebGL.compressionFormat

- Determines how Unity compresses files during the build step.



webGLBuildSubtarget

- Compressed texture format for target build platform.

stripEngineCode

- Remove unused Engine code from your build

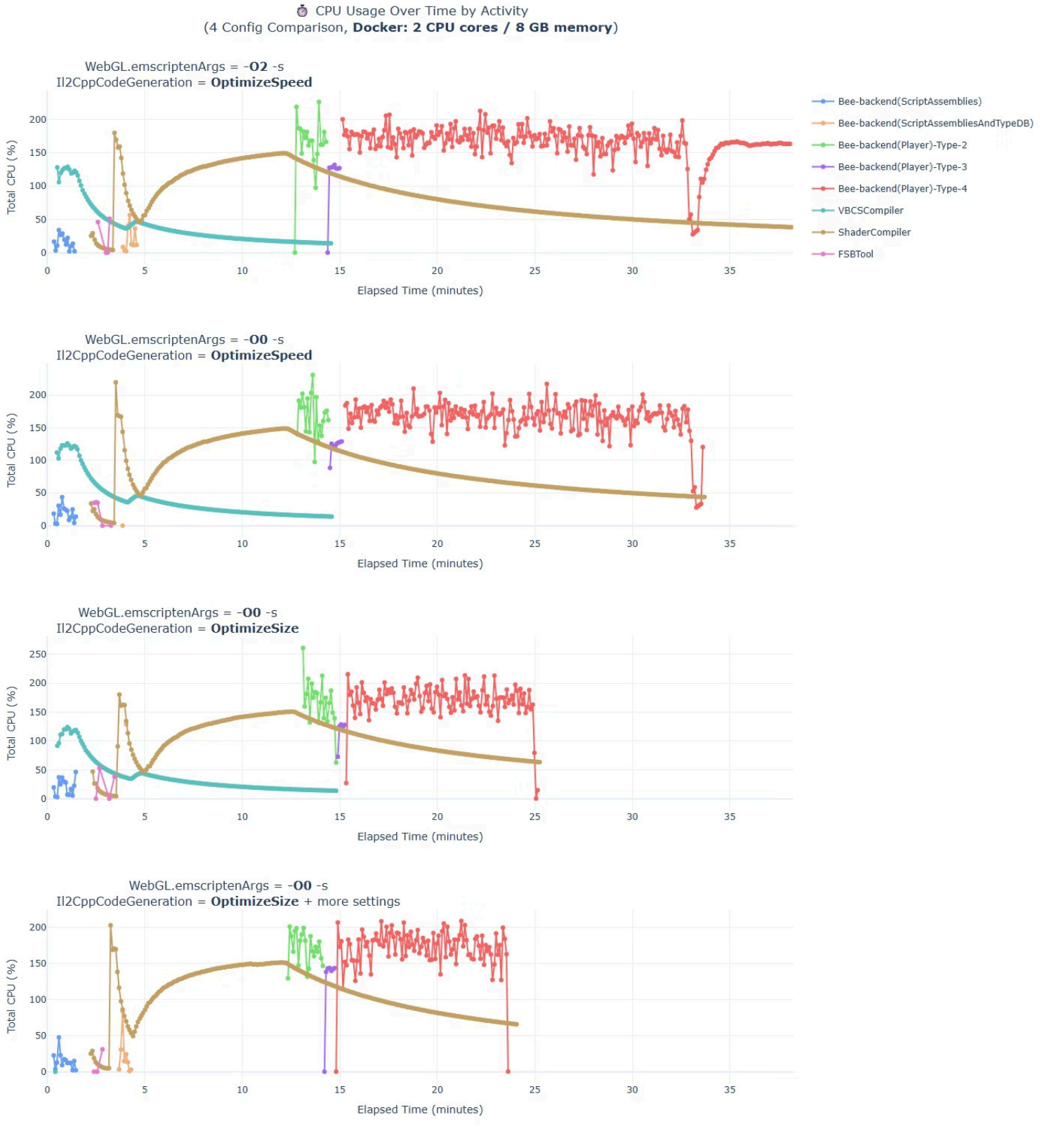
gcIncremental

- Allows you to enable or disable incremental mode for garbage collection.

codeOptimization

- Options for the optimization mode to use for compiling Web code.

Build Time Reduction with Optimized Unity Settings



Config A

- emscriptenArgs: -O2
- Il2CppCodeGeneration: OptimizeSpeed
- ⏱️ Build Time: 38.4 min (baseline)
- 📦 Build Size: 146 MB

Config B

- emscriptenArgs: -O0
- Il2CppCodeGeneration: OptimizeSpeed
- ⏱️ **Build Time**: **33.8 min** **(↓ 11.9%)**
- 📦 **Build Size**: 193 MB

Config C

- emscriptenArgs: -O0
- Il2CppCodeGeneration: OptimizeSize
- ⏱️ **Build Time**: **25.5 min** **(↓ 33.6%)**
- 📦 **Build Size**: 155 MB

Config D

- emscriptenArgs: -O0 + more settings
- Il2CppCodeGeneration: OptimizeSize
- ⏱️ **Build Time**: **22.4 min** **(↓ 41.7%)**
- 📦 **Build Size**: *(not measured)*

Build Time Reduction with Optimized Unity Settings

Build Time (on 8-core CPU)

- Original config took 13–14 minutes
- ⏱️ **Build Time:** 13–14 min → **7.4 min**
  (47% Reduced)
- Tuned with `emscriptenArgs`, `Il2CppCodeGeneration`, etc.

Build Size Reduced to 95.2 MB

- Adjusted:
  - managedStrippingLevel
  - compressionFormat

🎯 From 146 MB (Original Setting)
→ down to 95.2 MB

Toward Cloud-Based(Serverless) Unity Build Pipelines

🔹 We’re Ready for the Cloud

- ✅ Clean builds now possible under **8 minutes**
  (RT project scale, 8-core/32GB Cloud container)
- ✅ Performance parity with current VM
- ➕ Artifact reuse no longer required to maintain speed
- ☁️ Unity Editor can safely migrate to **containerized cloud infrastructure**

🔹 What Comes Next

- WebGL builds are **no longer opaque**
- PID-level profiling allows **precise CPU/memory analysis**
- Unity project settings are now **measurable, tunable, and reproducible**
- 🔄 Enables automated A/B testing across build configs
- 🧪 Opens the door to **automated integration testing** + **runtime telemetry** in batchmode
  → Performance-aware, self-optimizing pipelines