DOORDASH

Who says you can't use Gradle for a Monorepo?

Ashwin Kachhara
Yunji Zhong
09/21/2023
DPE Summit 2023

Ashwin Kachhara

Yunji Zhong

Yes, we looked into Bazel, but...

Contents

WHO SAYS YOU CAN'T USE GRADLE FOR A MONOREPO?

Building a monorepo would help, but it comes with challenges to the build system

Backend Development @ DoorDash

Tens of millions of lines of Kotlin code

~700 backend developers working on ~200 backend poly-repos

Problems we were facing

- Build Pipelines & IDE Experience for large repositories can be slow! 🐌
- Separate codebases lead to utilities, build standards and pipelines performing a similar function
- Upgrading dependencies becomes a complex endeavour involving multiple organizations
- Propagating best practices & defragmenting tooling is a never-ending task

Exploring Bazel

Bazel

PROS

- Designed for large monorepos
- Distributed builds
- Remote execution
- Unify build systems across Web, Mobile, Backend

CONS

- All of Eng would need to migrate to Bazel
- Learning curve would slow developers down
- Kotlin development experience is subpar
- Lack of specific plugins that are available in Gradle

Custom Plugins

Plugins we’ve developed improve developer experience

- Reduce build boilerplate
- Optimized Maven Repository settings
- Apply internal code quality rules
- Standard application settings

Replicating this in Bazel would take work

Developer Empathy

Kotlin ❤️ Gradle

Developers can’t pause all other development to migrate build systems.

🚀

Is Gradle still viable?

Intro: Gradle Project Structures

SINGLE PROJECT

```
.
├── build.gradle
├── settings.gradle
└── src
```

MULTI-PROJECT

```
.
├── project1
│   ├── build.gradle
│   └── src
├── project2
│   ├── build.gradle
│   └── src
├── project3
│   ├── build.gradle
│   └── src
├── build.gradle
└── settings.gradle
```

Degraded build latency and IDE experience past a certain number of projects

COMPOSITE

```
├── project1
│   ├── build.gradle
│   ├── settings.gradle
│   └── src
├── project2
│   ├── build.gradle
│   ├── settings.gradle
│   └── src
├── project3
│   ├── build.gradle
│   ├── settings.gradle
│   └── src
└── settings.gradle
```

Composite Builds

PROS

- Familiarity and in-house expertise
- Robust 3P and internal plugin ecosystem
- Significantly better DevEx w/ Kotlin/Java[1]
- Better IDE Experience

CONS

- Distributed execution
  - limited to tests
  - requires Gradle Enterprise[2]
- Remote execution is not supported[2]
- Opaque project dependency graph[2]
- Doesn't solve our problems for Web/Mobile repos

1. bazelbuild/rules_kotlin lags behind equivalent in Gradle
2. monorepo.tools

Gradle Composite Builds

WITH A FEW KEY AUGMENTATIONS

How to do tasks correctly, fast and efficiently?

	In School	**In Industry**
Correctly	Copy answers from cheatsheet	Single source of truth
Fast	Do as little work as possible	Minimal set of changed targets
Efficiently	Ask classmates to do your homework	Distributed execution
	Detention	Promotion

Design Principles

- **Correct:** Define dependency versions centrally for easier upgrades
- **Fast:** Build only the projects that are affected by a changeset (directly or transitively)
- **Efficient:** Distribute builds across multiple machines for maximum parallelism
- Empathy for developers

Version Catalogs

Monorepo structure

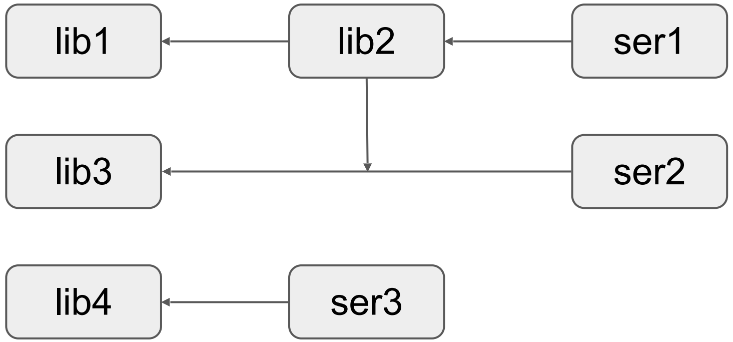
```
.
├── libraries
│   ├── lib1
│   ├── lib2
│   ├── lib3
│   └── lib4
├── services
│   ├── ser1
│   ├── ser2
│   └── ser3
└── libs.versions.toml
```

```
[versions]
kotlin = "1.8.22"

[libraries]
kotlin-stdlib = { module = "org.jetbrains.kotlin:kotlin-stdlib",
version.ref = "kotlin" }
```

Affected Targets Computation

Reversed dependencies graph

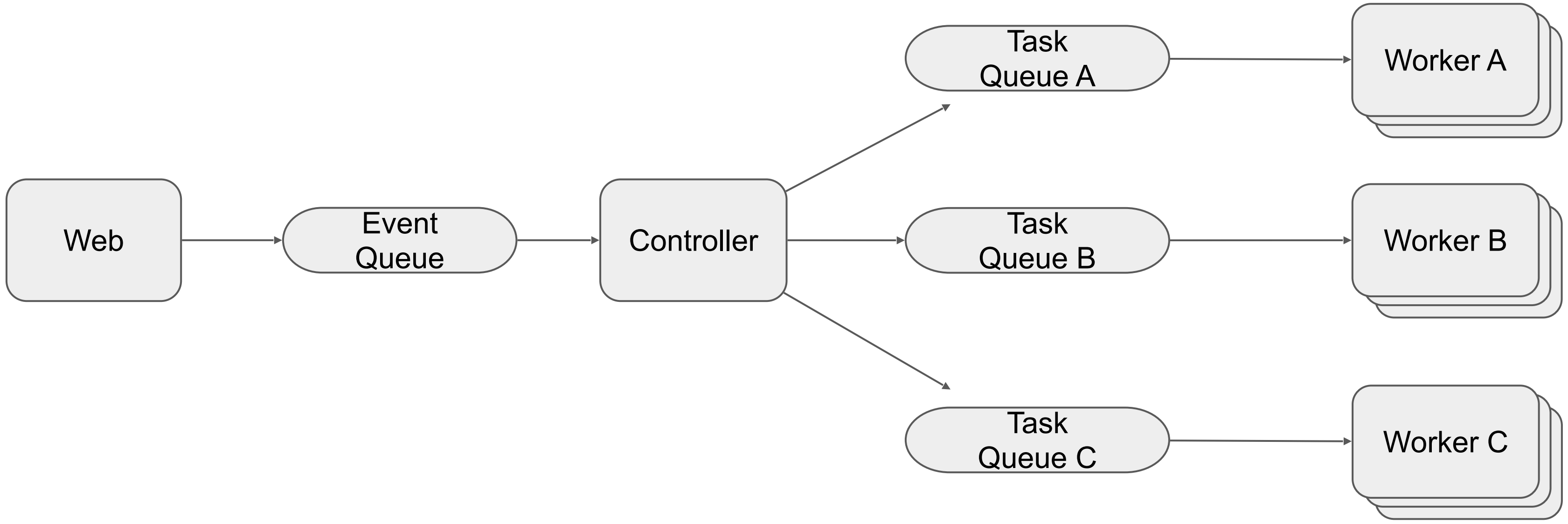


What if we make a change on lib3?

```
[[lib3],[lib2, ser2],[ser1]]
```

Distributed Builds

Each project can be built and tested independently



Choice of worker containers

- Ephemeral
  - Provision at each incoming build request
  - Single build per container
- Long-running
  - No boot-up time
  - Isolated gradle daemons per pod for parallel building
  - Local build cache

Takeaway(s)

Results

- 🏅 Composite Builds provide a great IDE experience
- 🐇Cross-cutting changes can run a full CI pipeline quickly
- Significant reduction of service boilerplate
- Boosted code-sharing and reuse

What’s next?

- Explore the remote parallel execution model for other parts of the stack
- Continue onboarding projects into our monorepo

Acknowledgements

We would like to specifically thank Jamil Seaidoun, Yazan Mimi, Nico Semko, Claire Han, Francis Tian, Mingmei Zhang, Matthew Anger, Michael Sitter, Aaron Livingstone, Nick Firmani, Haitham Gabr, Arjun Shantharam, Mathieu Courtemanche and Adam Rogal

Thank you

Ashwin Kachhara

Yunji Zhong