Maximizing Developer Productivity with Gradle Enterprise

Training content

- What is Gradle Enterprise?
- Leveraging the build cache
- Working with build scans
- Performing build analytics

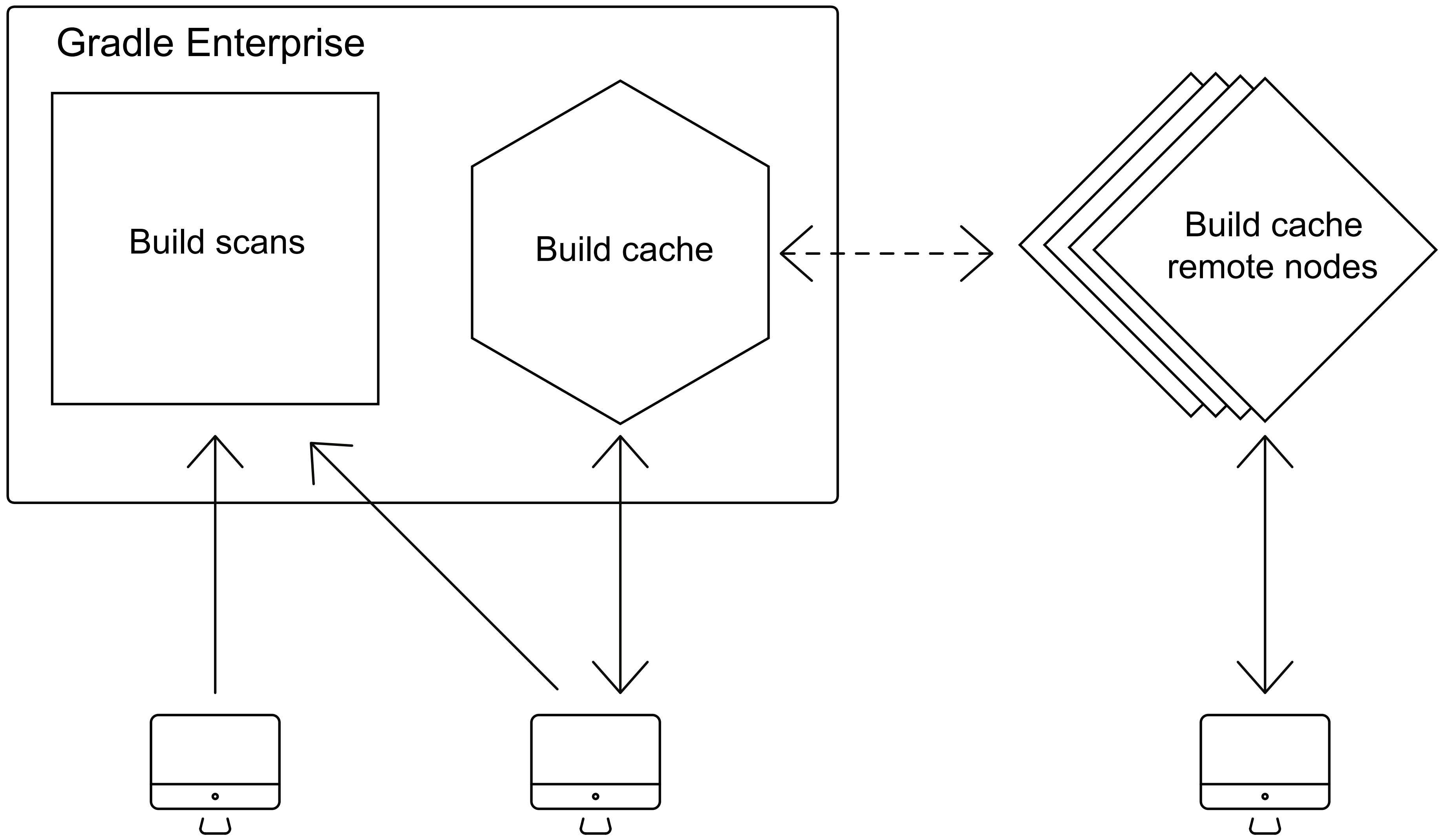
Training material

- Gradle Enterprise training instance
  @ https://enterprise-training.gradle.com
- Zip with hands-on labs and slides
  @ https://enterprise-training.gradle.com/developer-productivity-with-gradle-enterprise

What is Gradle Enterprise?

Gradle Enterprise is a platform on top of the Gradle build tool that allows to maximize productivity of developers and build teams, hosted on-premises.

The feature sets



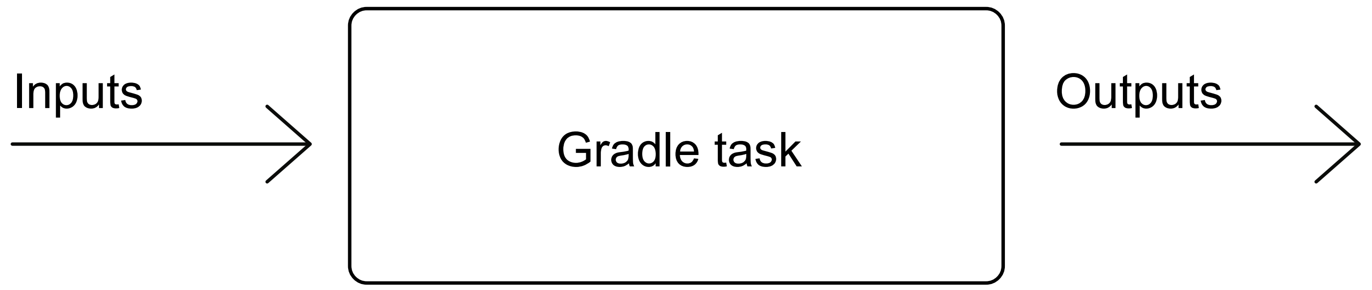
Operations

- Easy installation
- Automatic license handling
- One-click version upgrades
- Systems health monitoring
- Automatic backups
- Support bundles

Leveraging the build cache

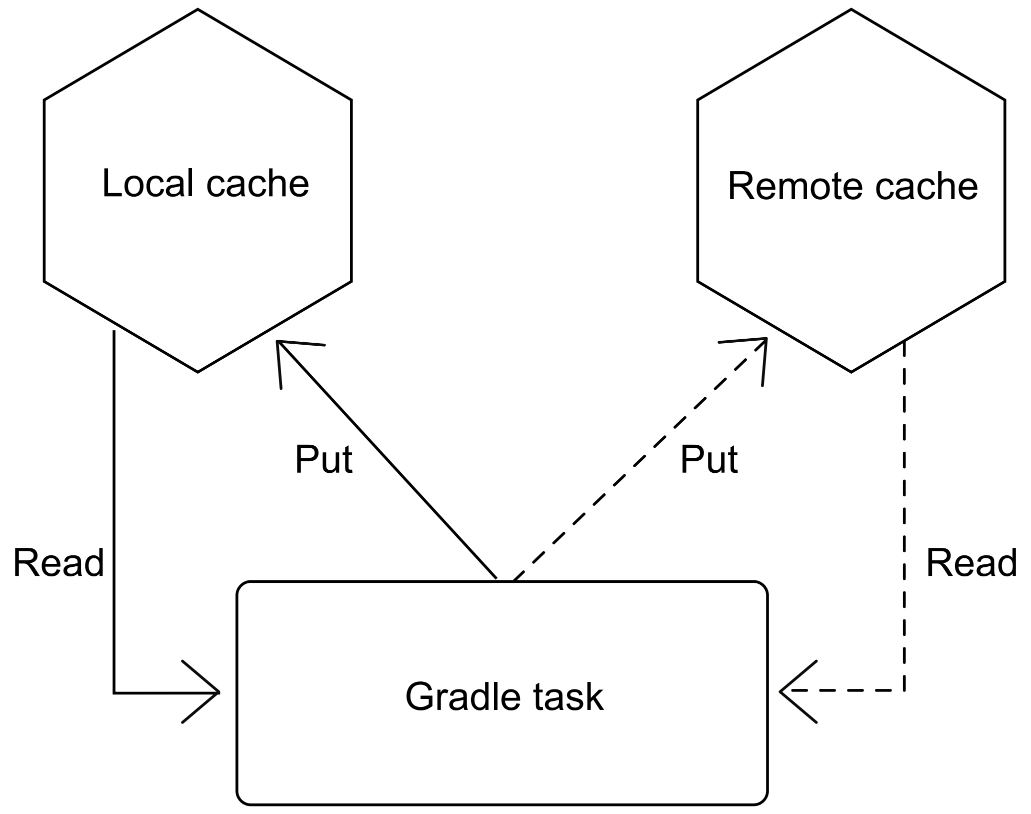
What is build caching in Gradle?

- Cache mechanism that aims to save time by reusing task outputs produced by other builds
- Works by storing task outputs and allowing builds to fetch these task outputs when the task inputs have not changed



What is build caching in Gradle?

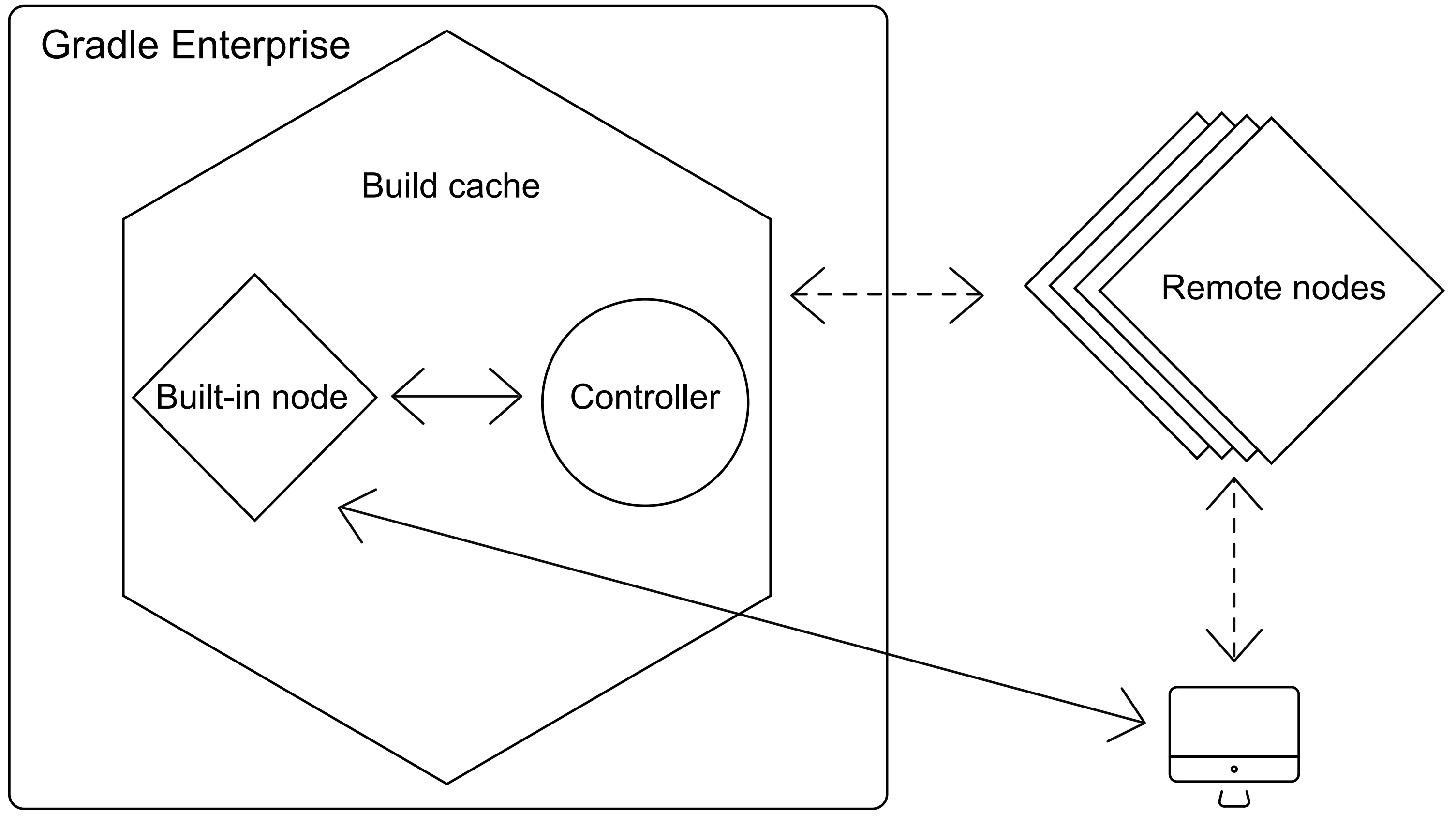
- Enabled via `--build-cache` flag or system property
- Local and remote cache can be enabled and configured individually
- Gradle Enterprise provides a high-performance build cache back-end



Gradle Enterprise build cache architecture

- Cache controller
- Cache nodes
  - Built-in cache node
  - Remote cache nodes

Gradle Enterprise build cache architecture



Gradle Enterprise build cache

Demo

Lab 01

Use the Gradle Enterprise build cache

Optimize for cache artifact reuse

- Make tasks cacheable
- Populate cache early for downstream consumers
  - CI pipeline with downstream builds consuming the outputs of upstream builds
  - CI builds with artifacts for local developers

Working with build scans

What are build scans?

- Persistent record of what happened during a build
- Permanent and shareable URL
- For developers and build engineers

Short tour of build scans

- Publishing a build scan
- Browsing the build scan UI
- Seeing all build scans

Build scan plugin configuration

- Pointing to Gradle Enterprise instance
- Publishing scans for all builds
- Injecting custom values
- Using life-cycle hooks

Lab 02

Inspect a build scan

Fixing build failures and code issues faster

Share console output you don’t understand

scan

Pull in help for an unexpectedly failing test

scan

See all locally failing tests across all projects

scan

Check if your code relies on a specific dependency and if so on what version

scan

Understand why a given third-party library ends up on your classpath

scan

Find out what dependencies of your project use dynamic versions

scan

Find out what concrete version was used for a dependency with a dynamic version

scan

Determine if changed dynamic dependencies broke the build

scan list

See all external Gradle plugins applied to your build

scan

Understand why a given Gradle plugin was applied to your build

scan

Investigate why your project does not compile on your colleague’s machine

failing

successful

scan list

Lab 03

Find out if the developers in your company run the `clean` task

Adding your own data to build scans

Distinguish CI build scans from developer build scans

Understand the difference in build duration for a given project built locally vs. on CI

scan list

Add source control information to your build scans

Surface static code analysis issues in build scans

Reach out for help when local build fails to succeed

scan

Categorize build failures

Lab 04

See all builds that ran tests

Enhancing build performance proactively

Make any build faster

scan

Investigate what has the biggest impact on your configuration time

scan

Investigate why your configuration time is slower than it should be

scan

Determine if your build needs more memory

scan

Determine how much time was spent resolving dependencies

scan

Make the build faster by optimizing task parallelization

scan

scan 2

Verify local optimization experiments

scan

Lab 05

Find potential performance killers

Optimizing incremental build and use of build cache

Find out why a task was not up-to-date but got executed

scan

scan 2

Analyze build cache hit rate

scan

Investigate why a certain task is slow even though its output is taken from the cache

scan

Determine what tasks to make cacheable next

scan

Investigate why you are getting an unexpected build cache miss

scan

Investigate why you are getting an unexpected build cache hit and want to find the producing build

scan

scan 2

Lab 06

Decide how you could increase the cacheability of the given build

Jump straight to the build scan of a build run on CI

- TeamCity integration
- Jenkins integration

Performing build analytics

Scan list

- What builds were run?
- By whom were the builds run?
- How long did the builds take to run?
- What is the build failure rate?

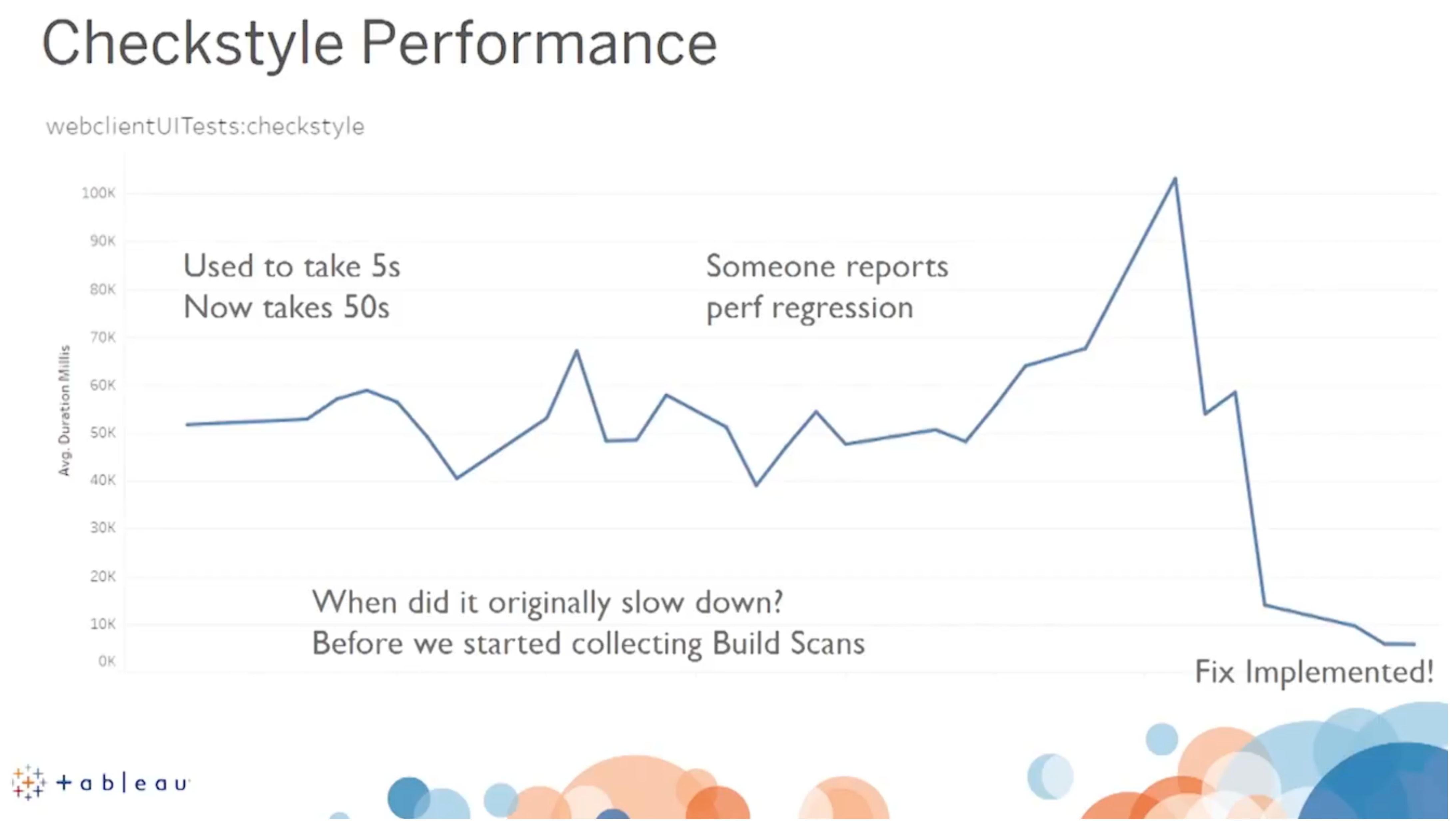
Export API

- Whilst running a build, build data is mapped to a series of events
- Build events can be exported from your Gradle Enterprise instance via an HTTP endpoint
- Any data available in a build scan is available for export
- Build events can be exported since a point in time or given build
- Build event streams can be filtered to include only the events you are interested in
- Real-time streaming is supported

Lab 07

You want a live dashboard of build activity

You want to see whether a performance fix worked as expected



You want to prioritize build problems to tackle first

You want to push your build data into your own BI tool

- Exporting events from Gradle Enterprise via Export API
- Pushing the captured events into the BI tool of your choice for further analysis

Resources

https://gradle.com/enterprise/resources

https://guides.gradle.org/using-build-cache

Thank you