



Build Cache Deep Dive

Boosting productivity for the whole team!

About the trainer

Prerequisites

- Skills
 - Good understanding of Java language
 - Basic understanding of Gradle concepts
- Tools
 - Java 8
 - Latest Gradle version

Training content

- Understand the benefits of using the Gradle build cache
- Use and configure the build cache
- Tune build logic for maximum cacheability
- Maximize the benefits with Gradle Enterprise

Training material

- Gradle Enterprise training instance
@ <https://enterprise-training.gradle.com>
- Zip with hands-on labs and slides
@ <https://enterprise-training.gradle.com/build-cache-deep-dive>

Build scans

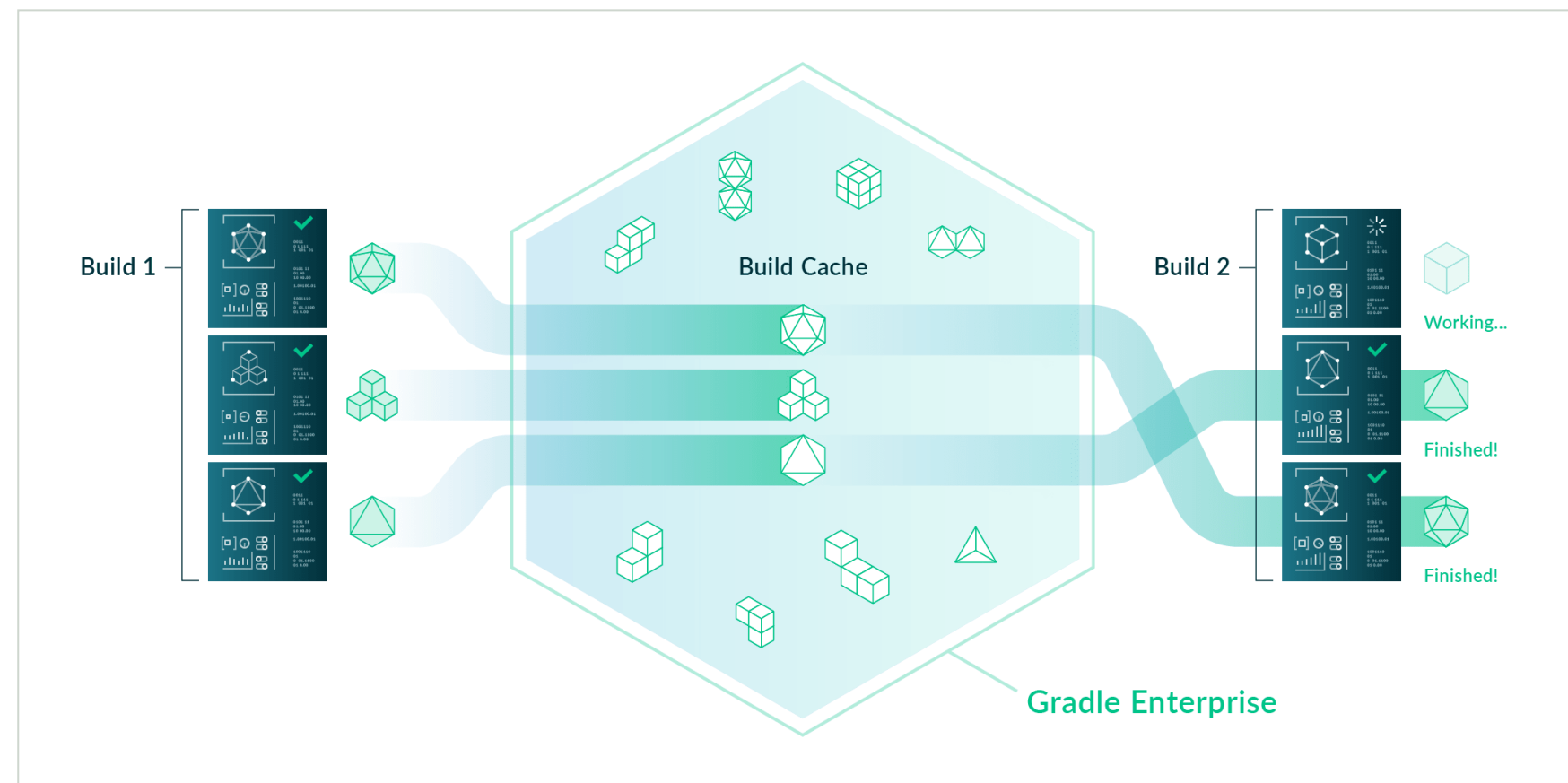
- Gathers details about build
- Generated and published with `--scan`
- Captures IP address and host name
- Published information will be publicly-available
- Can be deleted manually with minus icon in toolbar at the top

Performance is key

- Faster build times lead to faster feedback
- Faster feedback leads to better developer productivity
- Better developer productivity ships features quicker
- See blog post "[Quantifying the costs of builds](#)"

Approaches for build avoidance

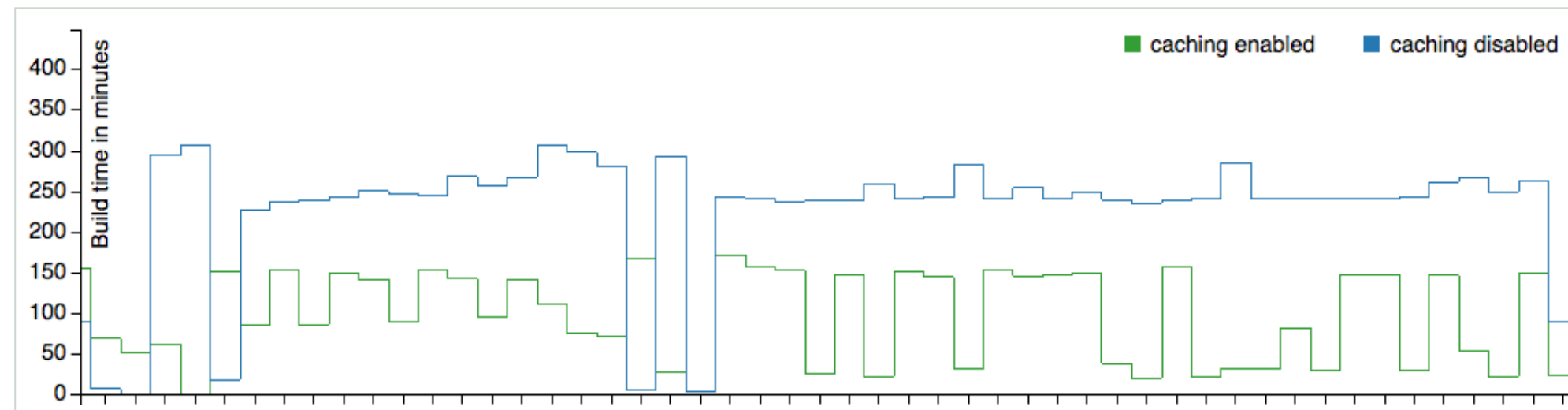
- **Incremental build:** build avoidance in same workspace
- **Build cache:** share build results across multiple workspaces



Common use cases

- Speed up developers' builds when switching branches
- Share results between CI builds
- Accelerate developer builds by reusing CI results

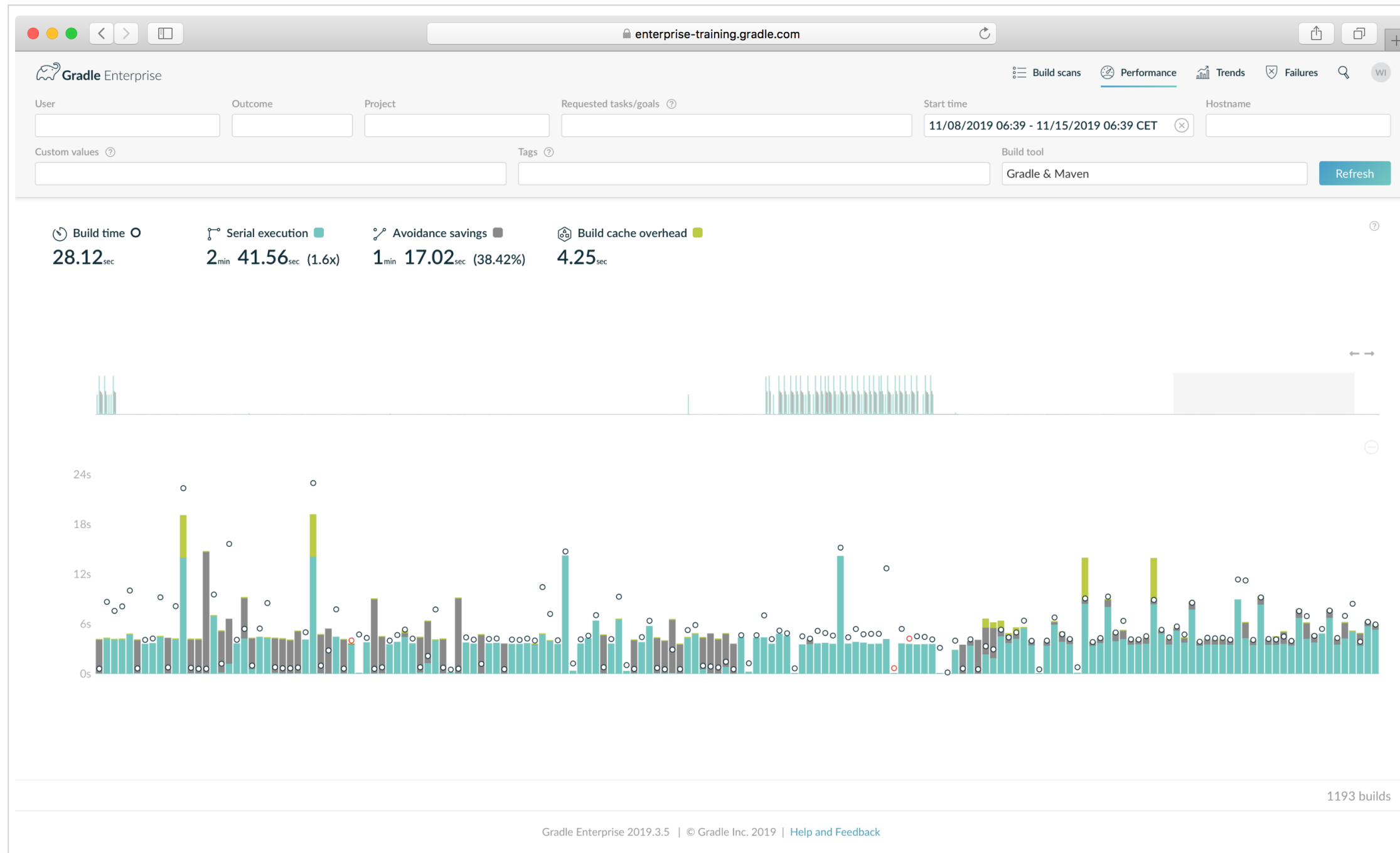
Faster Gradle builds on CI



Build times are > **80%** faster for Gradle core.

We see similar improvements for projects of our customers.

Visualized savings in Gradle Enterprise

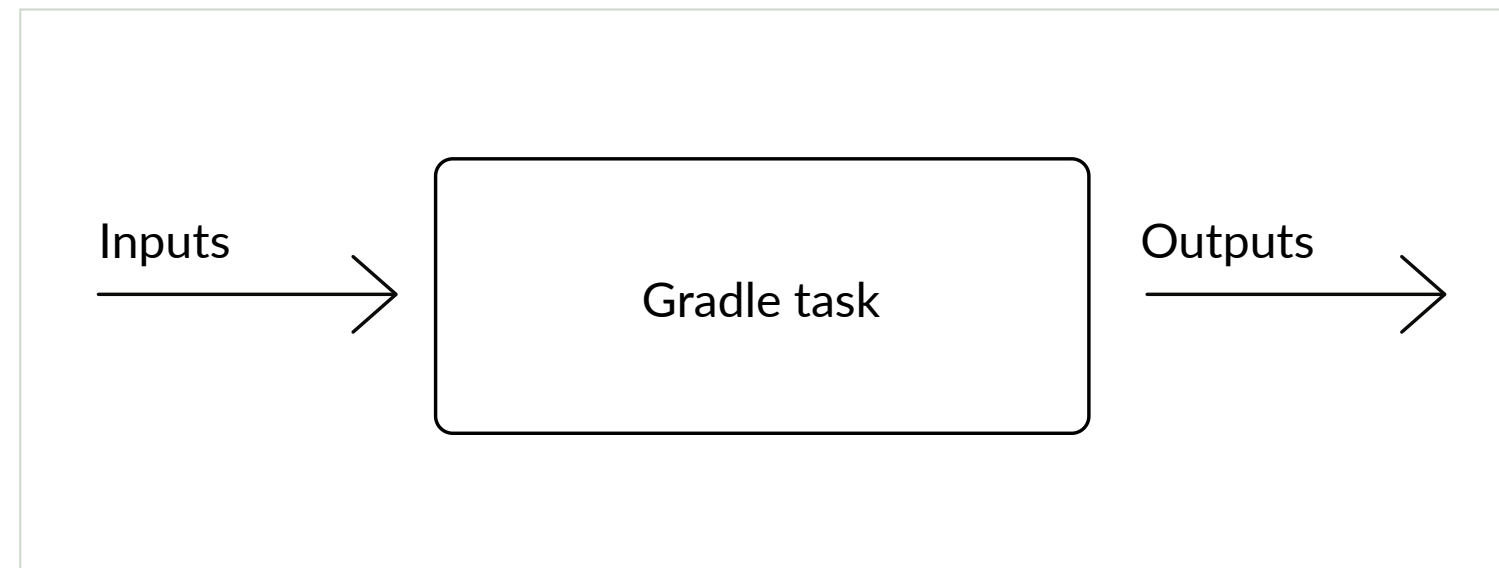


Recap: incremental builds

- Important feature from the beginning
- Optimized for single developer running the build
- Underlying mechanism for the build cache

How does it work?

- Task needs to define inputs and outputs
- Hashes of inputs and outputs are stored on disk
- Actions are only executed if inputs and/or outputs have changed



Execution marker in console

- Gradle marks task **UP-TO-DATE**
- Build summary indicates high-level statistics

```
$ gradle compileJava --console=verbose  
:compileJava UP-TO-DATE
```

```
BUILD SUCCESSFUL in 0s  
1 actionable task: 1 up-to-date
```

Declaring inputs and outputs with annotations

Generate.groovy

```
class Generate extends DefaultTask {  
    @Input  
    int fileCount = 10  
  
    @OutputDirectory  
    File generatedFileDir = project.file("${project.buildDir}/generate")  
  
    @TaskAction  
    void perform() {  
        for (int i=0; i<fileCount; i++) {  
            new File(generatedFileDir, "${i}.txt").text = i  
        }  
    }  
}
```

Assign annotations to task properties or getter methods for all of your custom task implementations.

Declaring inputs and outputs with runtime API

build.gradle

```
generate {  
    inputs.property 'fileCount', 10  
    outputs.dir project.file("${project.buildDir}/generated")  
}
```

Use runtime task API (see **TaskInputs** and **TaskOutputs**) if task source code cannot be changed easily.

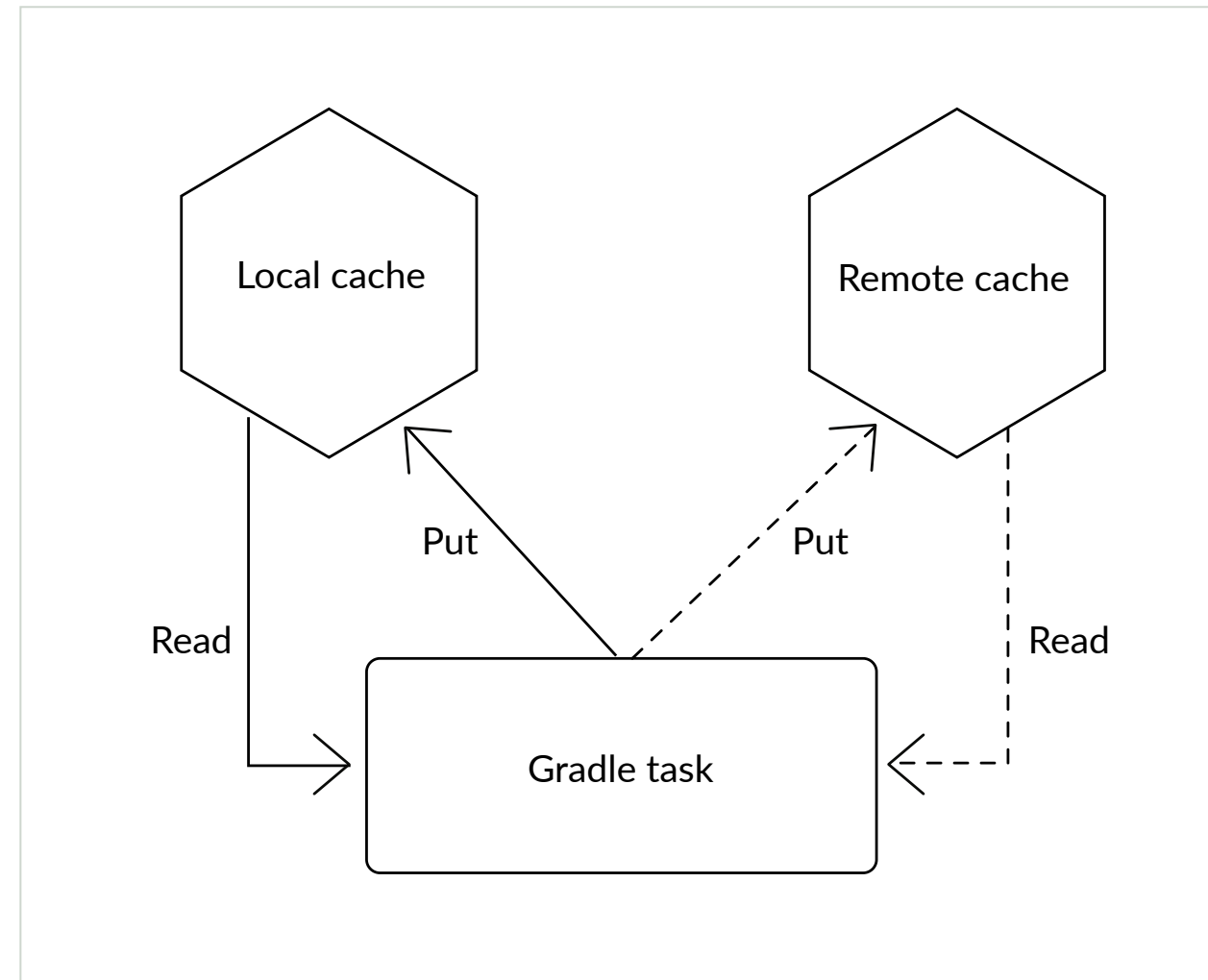
What are the limitations?

- Only uses the result of the previous execution
- Restricted to execution on single machine
- Cache is not shared among team members

What is the build cache?

- Reuse build outputs of *any* previous execution
- Reuse build outputs even if run with `clean` task
- Uniquely identifies outputs of tasks by inputs
- First stable version with Gradle 4.0

Different types of build caches



Local build cache

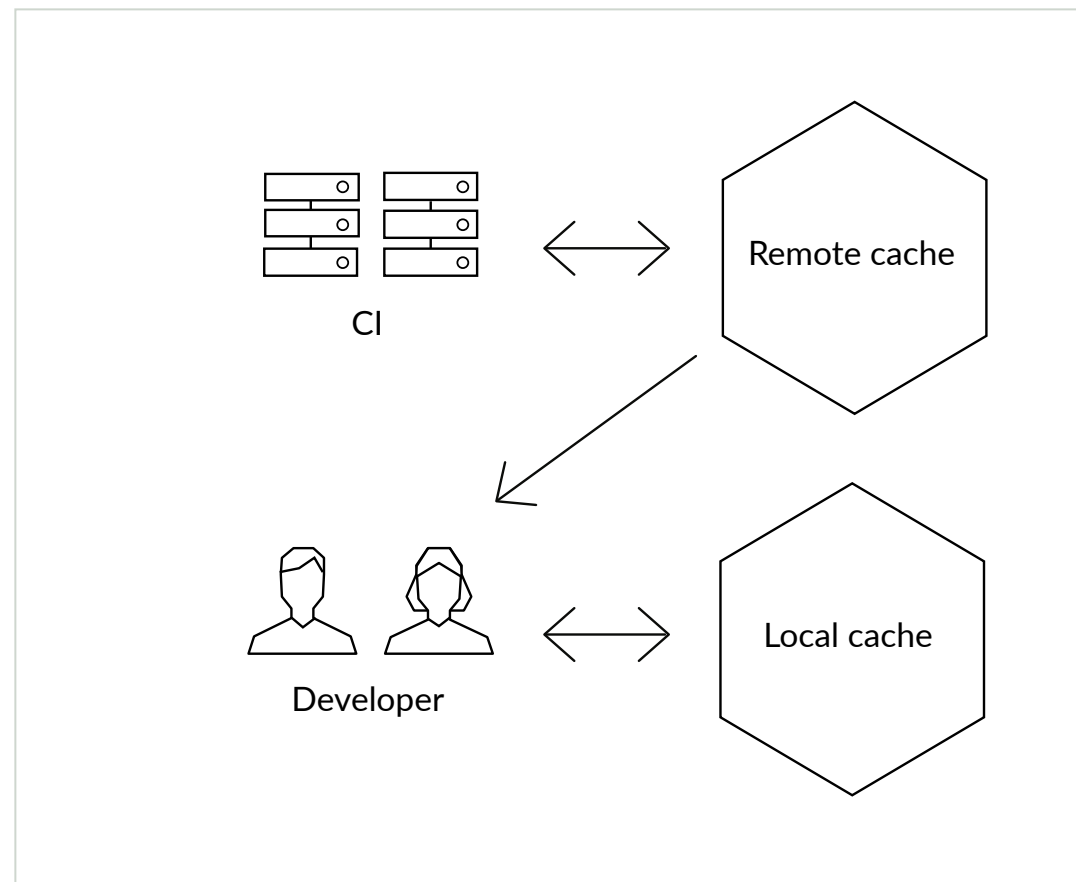
- Uses cache in directory on local machine
- Speeds up development for single developer or build agent
- Reuses build results when switching branches locally
- Particularly useful for Android variants

Remote build cache

- Shared among different machines
- Speeds up development for the whole team
- Reuses build results among CI agents/jobs and individual developers

Recommended sharing strategy

- only push to the shared cache from CI
- avoid sharing from developer machines



Influencing factors

- Architecture of code
- Nature of change
- Are tasks cacheable?
- Do outputs change with every build?

Using the build cache

Enabling the cache

- This build invocation only: `--build-cache` command line option
- All build invocations: `org.gradle.caching=true` in `gradle.properties`
- The `buildsrc` project needs to be explicitly enabled for cacheability

```
gradle --build-cache clean assemble
```

Lab 01

Using the local build cache

Configuring the local build cache

settings.gradle

```
buildCache {  
    local(DirectoryBuildCache) {  
        directory = new File(rootDir, 'build-cache')  
        removeUnusedEntriesAfterDays = 30  
    }  
}
```

Domain class for configuring local cache: **DirectoryBuildCache**

Configuring the remote build cache

settings.gradle

```
buildCache {  
    remote(HttpBuildCache) {  
        url = 'http://example.com:8123/cache/'  
        credentials {  
            username = 'build-cache-user'  
            password = 'some-complicated-password'  
        }  
    }  
}
```

Domain class for configuring remote cache: [HttpBuildCache](#)

Conditional cache configuration

settings.gradle

```
def ciServer = System.getenv().containsKey('CI')

buildCache {
    local {
        enabled = !ciServer
    }
    remote(HttpBuildCache) {
        url = 'https://example.com:8123/cache/'
        push = ciServer
    }
}
```

Standardizing build cache configuration

init.gradle

```
def ciServer = System.getenv().containsKey('CI')

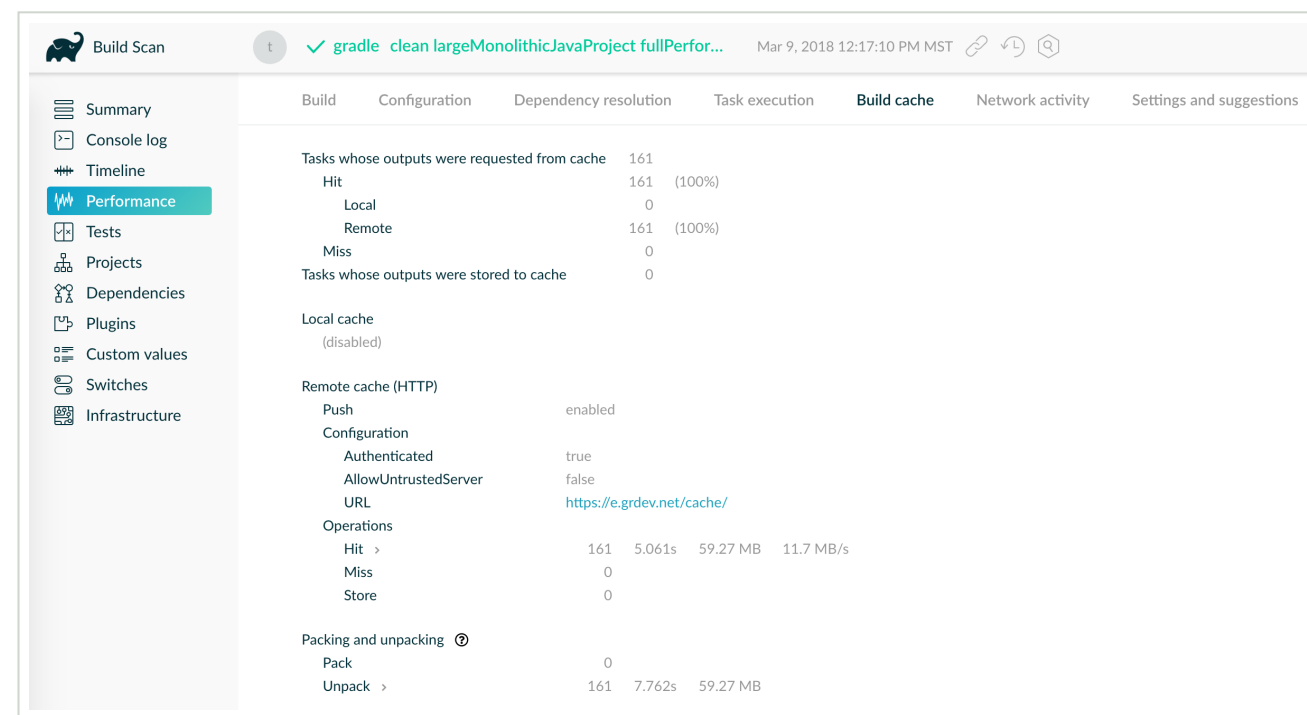
gradle.settingsEvaluated { settings ->
    settings.buildCache {
        local {
            enabled = !ciServer
        }
        remote(HttpBuildCache) {
            url = 'https://example.com:8123/cache/'
            push = ciServer
        }
    }
}
```

Computing the build cache key

- The task implementation
- The task action implementations
- The names of the output properties
- The names and values of task inputs

Build cache operations

- Hit
- Miss
- Store
- Packing
- Unpacking



Lab 02

Using the remote build cache

What makes a task "cacheable"?

- Task needs to define inputs and outputs
- Task type implementation needs to declare `@CacheableTask` annotation
- `@CacheableTask` is not inherited by subclasses
- Custom task types have to opt into cacheability

Cacheability influencing factors

- Declared inputs and outputs
- Repeatable output
- Relocatability vs. absolute paths

Built-in cacheable tasks

- Some but not all built-in Gradle tasks are cacheable
- Tasks involving copy operations are usually not cacheable

Enabling cacheability by annotation

Generate.groovy

```
@CacheableTask
class Generate extends DefaultTask {
    @Input
    int fileCount = 10

    @OutputDirectory
    File generatedFileDir = project.file("${project.buildDir}/generated")

    @TaskAction
    void perform() {
        for (int i=0; i<fileCount; i++) {
            new File(generatedFileDir, "${i}.txt").text = i
        }
    }
}
```

Only applicable to custom task implementations!

Enabling cacheability by runtime API

build.gradle

```
generateCode {  
    outputs.cacheIf {  
        // return boolean expression  
    }  
}
```

Ad-hoc tasks or tasks from plugins can determine cacheability via

`TaskOutputs.cacheIf (Spec).`

Disabling cacheability by runtime API

build.gradle

```
generateCode {  
    outputs.doNotCacheIf('Actions produce volatile results') {  
        true  
    }  
}
```

Disabling the cache for a task with `TaskOutputs.doNotCacheIf(String, Spec)` requires providing a reason.

Lab 03

Equipping tasks with caching capabilities

Troubleshooting the build cache

Possible approaches

- Low-level troubleshooting
 - Identify task outcome with `--console=verbose`
 - Retrieve cache key information by changing the log level
 - Compare cache keys and root causes
- Visual and convenient troubleshooting
 - Create a build scan
 - Use GE deep insight features

Info log level console information

```
$ gradle helloWorld --build-cache -i  
  
> Task :helloWorld UP-TO-DATE  
Build cache key for task ':helloWorld' is 16f4fbc007345a854d49302279d1
```

Info log level displays cache key generated for each task.

Debug log level console information

```
$ gradle helloWorld --build-cache -Dorg.gradle.caching.debug=true

> Task :helloWorld UP-TO-DATE
Appending taskClass to build cache key: HelloWorld_Decorated
Appending classLoaderHash to build cache key: 575dae0f1414d5dfd4ef14b6
Appending actionType to build cache key: HelloWorld_Decorated
Appending actionClassLoaderHash to build cache key: 575dae0f1414d5dfd4
Appending inputPropertyHash for 'message' to build cache key: f81fd65e
Appending outputPropertyName to build cache key: outputFile
Build cache key for task ':helloWorld' is 16f4fbc007345a854d49302279d1
```

Debug log level displays more detailed information.

Using build scans

- Task input comparison
- Task details (cache key, cacheability reason)
- Determining origin build of cache output
- Performance breakdown

Requirements for cacheable tasks

Repeatable task outputs

- Same inputs should produce the same outputs
- Byte-for-byte equivalent or semantically equivalent (with normalization)

Stable task inputs

- Inputs need to be stable over time
- Potential source of volatility
 - Timestamps
 - Absolute file paths
 - Non-deterministic ordering

Path sensitivity

- File paths for input properties are absolute by default
- Shared build results between machine requires exact same path
- Controllable via annotation `@PathSensitive`

```
@PathSensitive(PathSensitivity.RELATIVE)
@InputFiles
public FileTree getSources() {
    // ...
}
```

Input normalization

- Task inputs between two executions are compared to determine cacheability
- Controllable via annotations `@Classpath` and `@CompileClasspath`
- Example: For compile classpath Gradle extracts ABI signature from the classes on the classpath
- Configurable to ignore volatile files via `Project.normalization(Action)`

build.gradle

```
normalization {  
    runtimeClasspath {  
        ignore 'build-info.properties'  
    }  
}
```

Handling cases affecting cache correctness

Overlapping outputs

- Two or more tasks write to the same directory
- Difficult for Gradle to determine which output belongs to which task
- Build scan renders reason for this case

External inputs like system properties

- System properties often use absolute path
- Use relative path to fix

File encoding

- Java tools use the system file encoding when no specific encoding specified
- Can cause incorrect builds
- Always set the file system encoding to avoid issues

Line endings

- Important when build cache is shared across different OSes
- Set `autocrlf=false` if Git is used

Symlinks

- Symlinks are not stored in build cache
- Uses actual file contents of the destination of the link
- Some OSes (e.g. Windows) do not support symlinks
- Tasks will not be cacheable across different OSes

Java versions

- Gradle tracks only the major version of Java as input
- Usually applicable to compilation and test tasks
- Vendor and the minor version may influence the bytecode
- Suggested to add vendor as an input to the corresponding tasks

Lab 04

Handling cache misses

Getting started with the build cache

Recommended approach

- Equip tasks with inputs and outputs
- Use local build cache
- Set up remote build cache
- Roll out usage to team
- Use Gradle Enterprise for cache monitoring and optimization
- Report from the field: [Tableau using Gradle Enterprise](#)

Installing the remote build cache

- Build cache node available as Docker image or JAR file
- Docker image freely-available from Docker Hub
- Requires Docker installation on host machine

PUBLIC REPOSITORY

gradle/build-cache-node ☆

Last pushed: 25 days ago

Repo Info

Tags

Short Description

A remote Gradle build cache, capable of connecting to Gradle Enterprise.

Full Description

A [Gradle](#) build cache node operates as a remote Gradle build cache, and can connect with [Gradle Enterprise](#) for centralized management. The cache node can also be used without a Gradle Enterprise installation with restricted functionality.

For more information on installing and operating a build cache node, please see the [Gradle Enterprise Admin Manual](#).


For more information on Gradle build caching, please see the [Gradle User Guide](#).

For help and assistance, please use [discuss.gradle.org](#).

Docker Pull Command

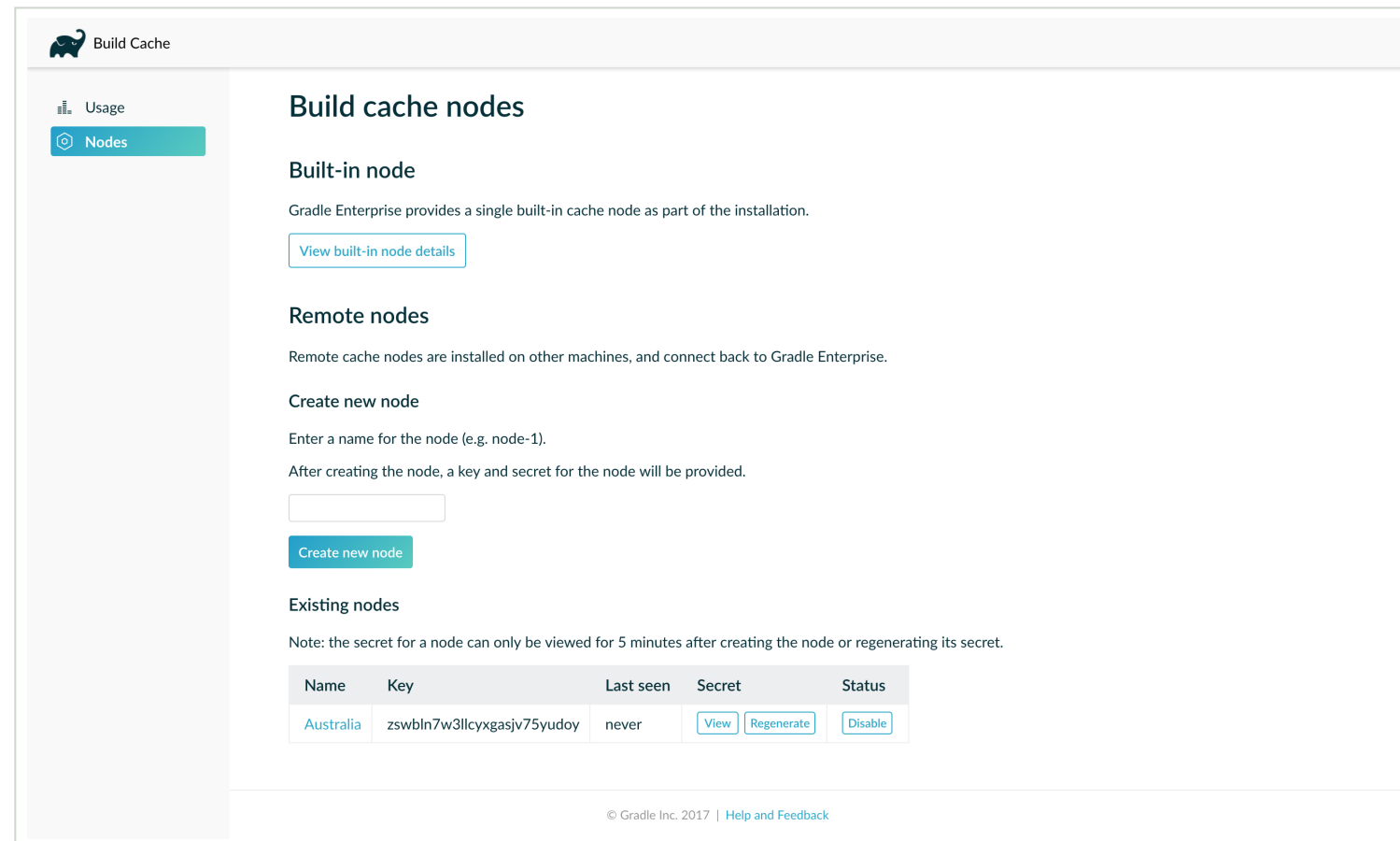
`docker pull gradle/build-cache-node`

Owner

 gradle

Connecting to Gradle Enterprise

- Optionally **register** with Gradle Enterprise for centralized management
- **Replication capabilities** for geographically distributed teams



The screenshot displays the 'Build Cache' interface in Gradle Enterprise. On the left, a sidebar contains 'Usage' and 'Nodes' (selected). The main content area is titled 'Build cache nodes' and includes sections for 'Built-in node', 'Remote nodes', and 'Create new node'. The 'Create new node' section has a text input and a 'Create new node' button. Below, the 'Existing nodes' section shows a table with one node named 'Australia'.

Build Cache

Build cache nodes

Built-in node

Gradle Enterprise provides a single built-in cache node as part of the installation.

[View built-in node details](#)

Remote nodes

Remote cache nodes are installed on other machines, and connect back to Gradle Enterprise.

Create new node

Enter a name for the node (e.g. node-1).

After creating the node, a key and secret for the node will be provided.

[Create new node](#)

Existing nodes

Note: the secret for a node can only be viewed for 5 minutes after creating the node or regenerating its secret.

Name	Key	Last seen	Secret	Status
Australia	zswbln7w3llcyxgasjv75yudoy	never	View Regenerate	Disable

© Gradle Inc. 2017 | [Help and Feedback](#)

Wrap up

Documentation and resources

- gradle.com/enterprise/resources
- docs.gradle.org/current/userguide/build_cache.html
- guides.gradle.org/using-build-cache

Try Gradle Enterprise for free

- Free 30-day trial
 - hosted by Gradle: zero installation, ready-to-go
 - on your infrastructure: quick setup, maximum control
- Technical support included

<https://gradle.com/enterprise/trial>



Thank you