How we Built a Distributed Testing Platform

Marc Philipp
Senior Principal Software Engineer

Roberto Perez Alcolea
Senior Software Engineer

How we Built a Distributed Testing Platform

Marc Philipp

Sr. Principal Software Engineer

@marcphilipp

Roberto Perez Alcolea

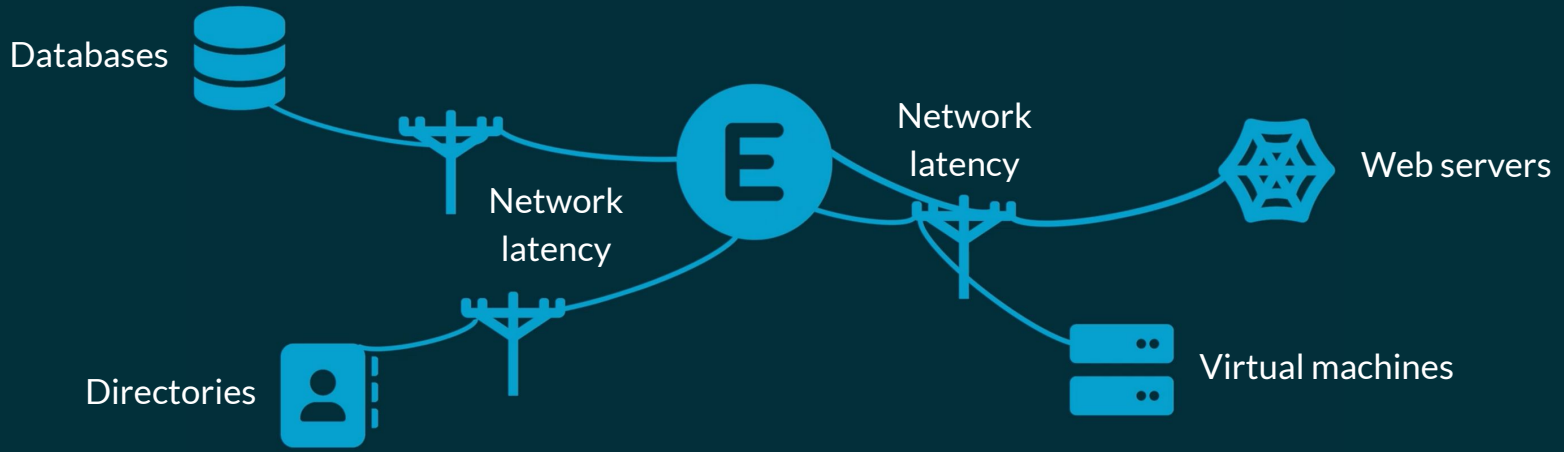
Senior Software Engineer

@rpalcolea

Why Distribute Tests?

Testing usually dominates build times

Why is testing so slow?



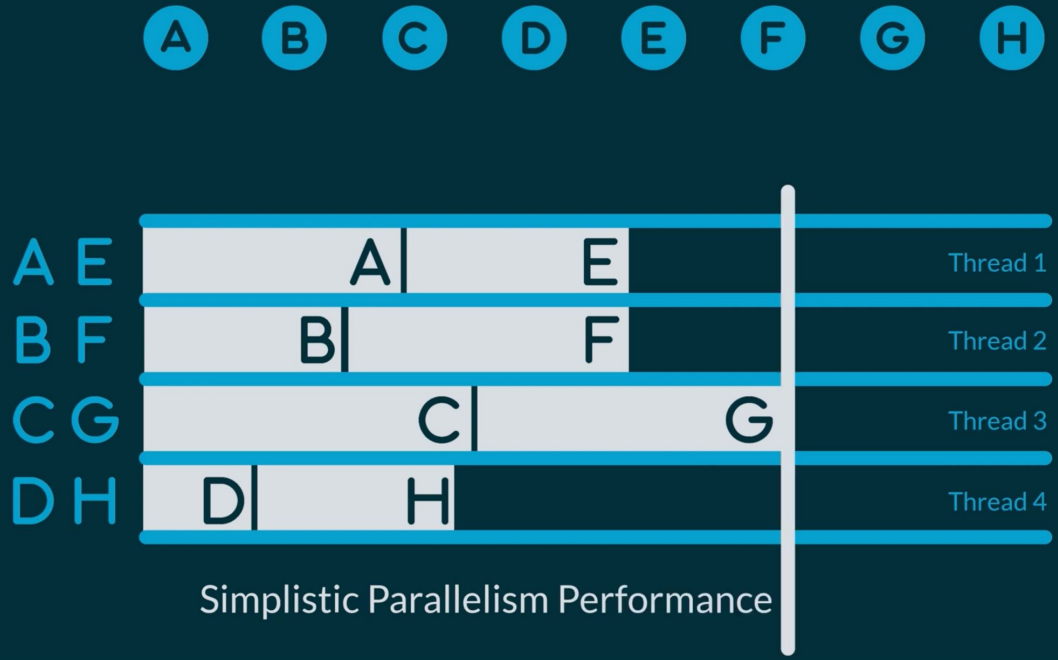
Reduced test time yields increase in productivity

More build and test executions
Shift-left testing from CI to local builds
Less context-switching

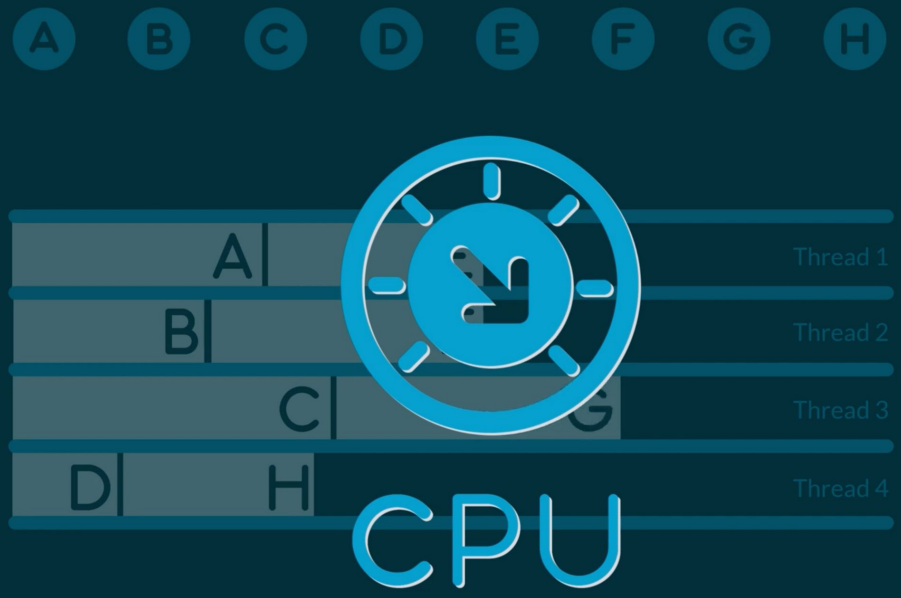
Existing Solutions

for running tests in parallel

Local parallelization lacks historical information



Single-machine parallelism does not scale



CI Fanout

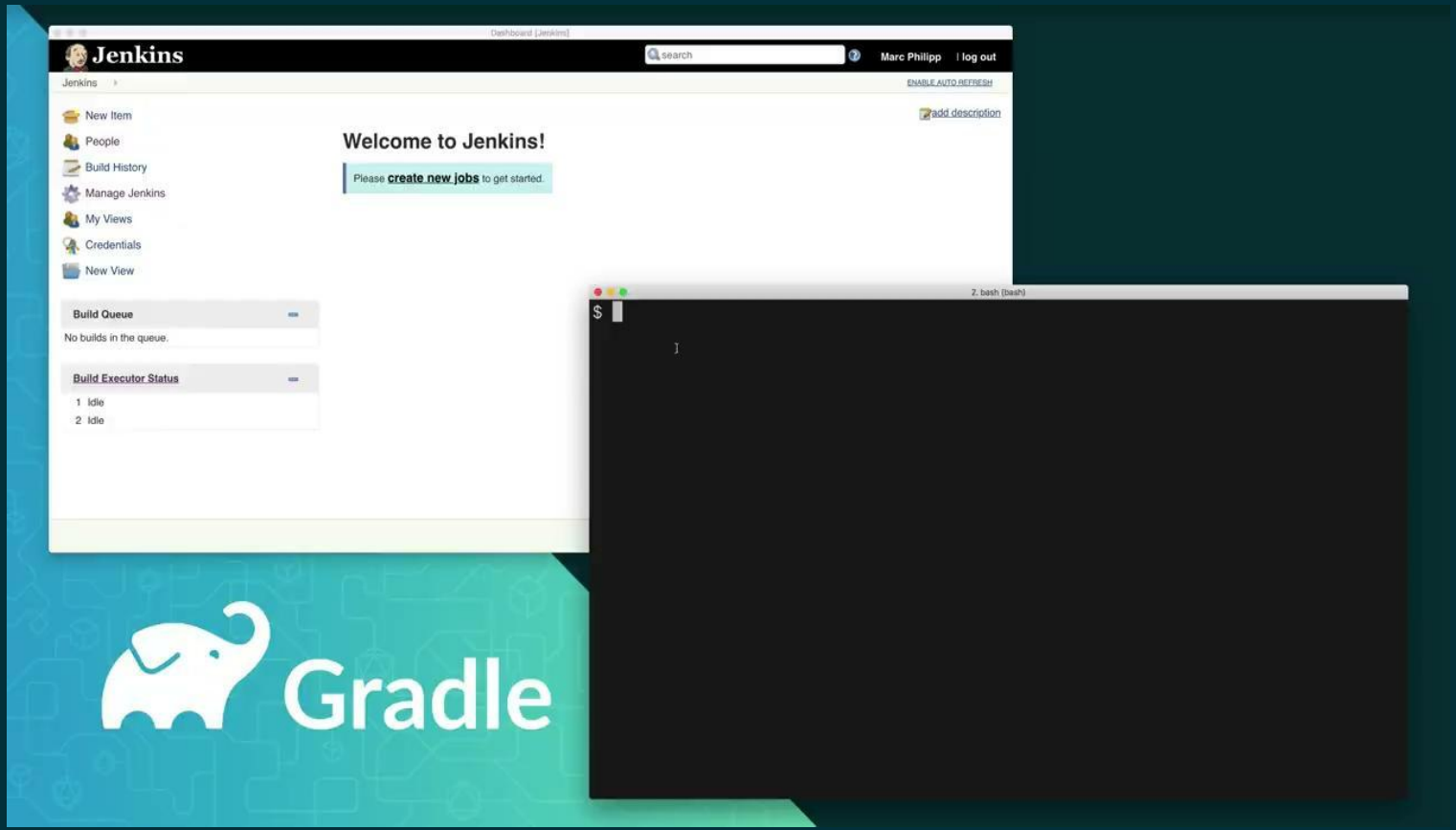
Idea: run subset of tests on CI agents in parallel

- Grouping of tests often manual
- Large overhead because each CI agent has to run build up to test task
- Test results are scattered over multiple CI jobs
- Does not support local builds

Building Test Distribution

Our Journey

Initial prototype (2019)

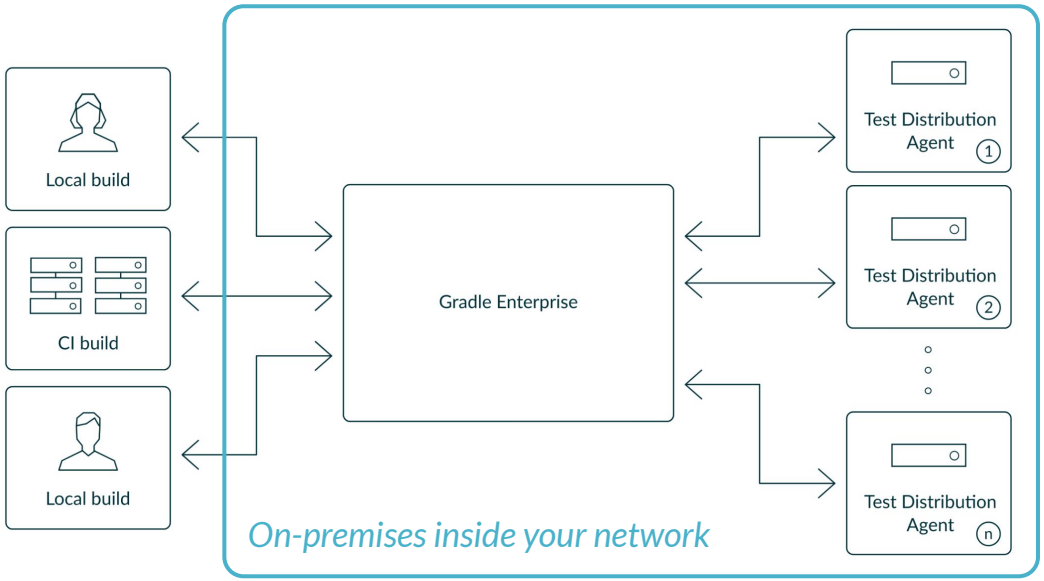


Initial prototype (2019)

- Implemented as Jenkins plugin
- Idea: reuse Jenkins nodes and infrastructure to run tests remotely
- Pros:
  - Jenkins was widely used by our largest customers at the time
- Cons:
  - Not all customers use Jenkins
  - Installing/updating Jenkins plugins in a corporate environment is not as simple as it seems (i.e. we couldn't move as fast as we wanted to)

Initial release

Gradle Enterprise **2020.2**



- Broker component included in Gradle Enterprise server
- Agents connect to the broker
- Builds connect to the broker and request agents
- Works for local and CI builds

Running Test Distribution agents

Gradle Enterprise **2020.2**

```
java -jar gradle-enterprise-test-distribution-agent.jar \
    --server https://ge.example.com \
    --api-key «api-key» \
    --capabilities docker,postgres=14
```

Runs **on-premises** inside your own network infrastructure

```
docker run \
    --env TEST_DISTRIBUTION_AGENT_SERVER=https://ge.example.com \
    --env TEST_DISTRIBUTION_AGENT_API_KEY=«api-key» \
    --env TEST_DISTRIBUTION_AGENT_CAPABILITIES=postgres=14 \
    gradle/gradle-enterprise-test-distribution-agent
```

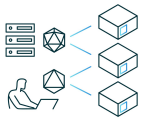
- The agent comes in two flavors: Jar and Docker image
- Runs on Java 11+, requires about 128 MB of memory
- Runs on Windows, macOS, and Linux
- Detects its environment (JDKs, OS) during startup
- Administrators can pass additional capabilities as command line parameters

Integrates with default Test task

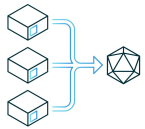
Gradle Enterprise **2020.2**

```
tasks.test {
    useJUnitPlatform()
    distribution {
        enabled.set(true)
    }
}
```

Gradle



Input files (e.g. classpath) are **automatically transferred** to remote agents



Code coverage and other output files are **transferred back and merged** automatically

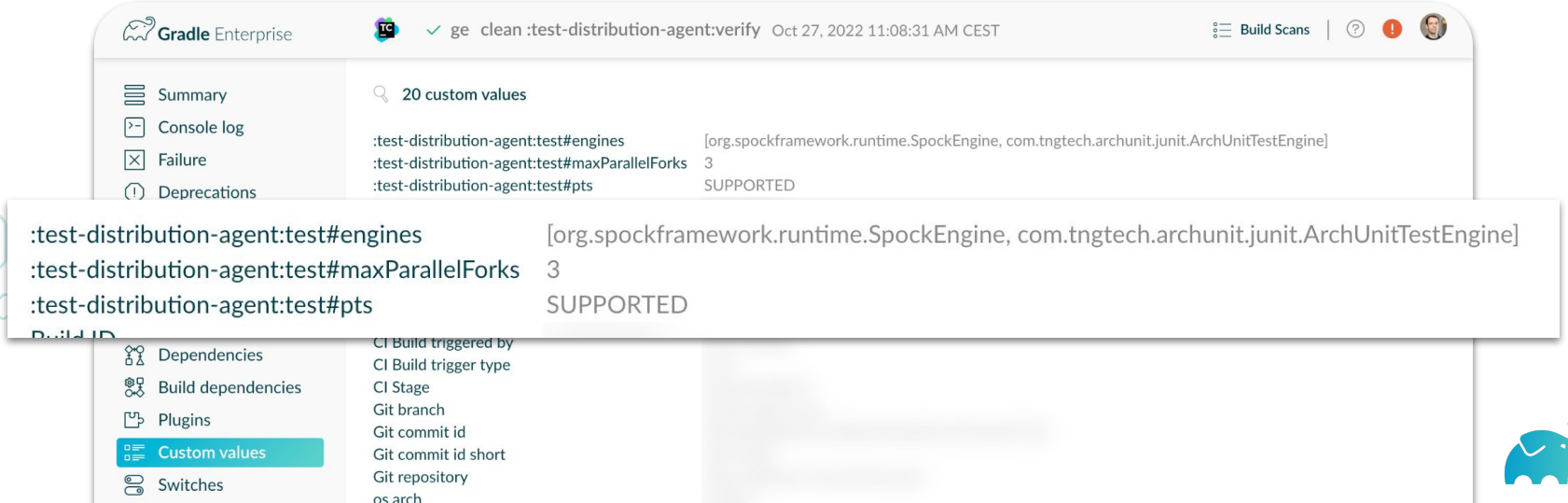
Test Distribution requires JUnit Platform

- Most test frameworks with JUnit Platform test engines are supported:
  - JUnit 5 (Jupiter) ✔
  - JUnit 3/4 and Spock 1.x (via junit-vintage-engine included in JUnit 5) ✔
  - Spock 2.x ✔
  - TestNG (via testng-engine) ✔
  - ScalaTest (via scalatest-junit-runner) ✔
  - ArchUnit ✔
  - jqwik ✔
- Currently unsupported (we have plans to add support where possible):
  - Kotest
  - Spek
  - Cucumber

Checking compatibility

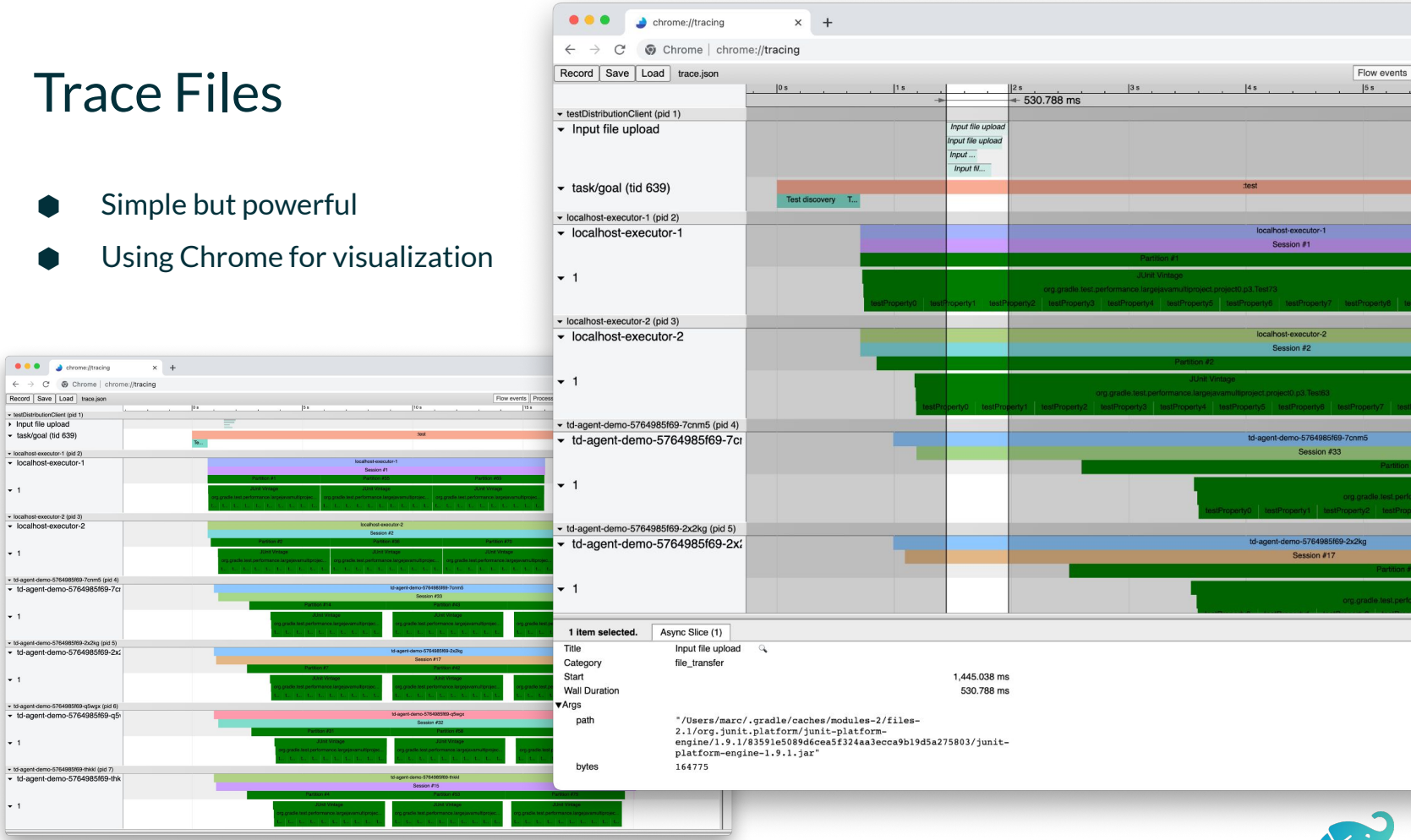
- Compatibility can be checked before adopting Test Distribution by applying a custom build script that adds custom values to the Build Scan

https://github.com/gradle/gradle-enterprise-build-config-samples/pull/469

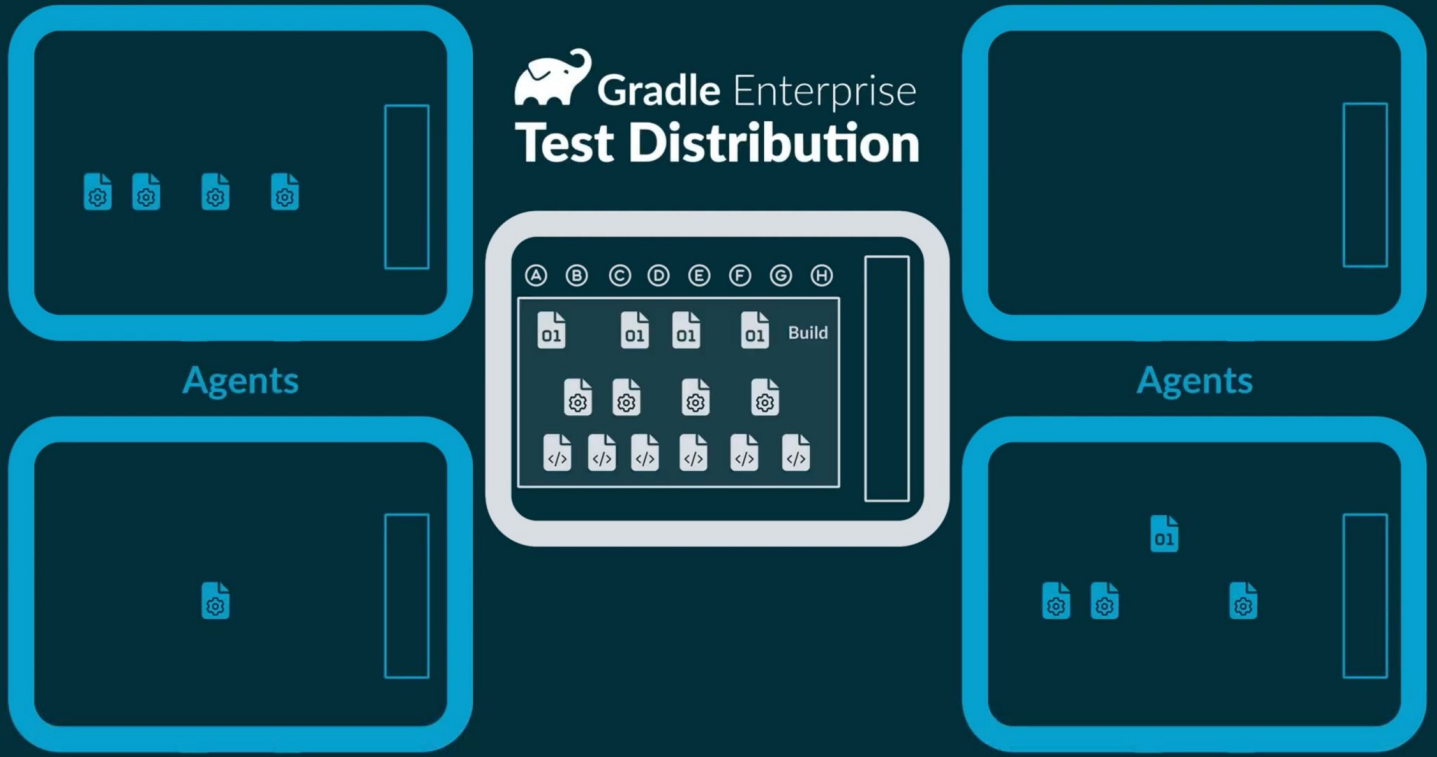


Trace Files

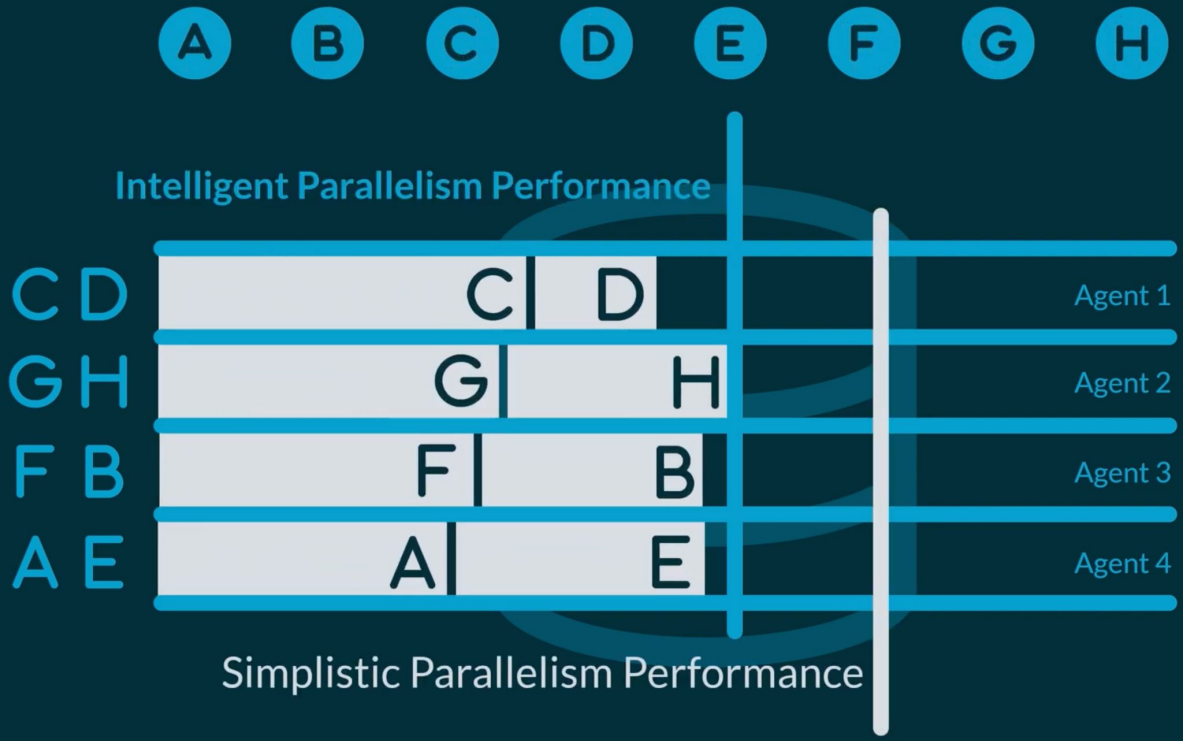
- Simple but powerful
- Using Chrome for visualization



Run build only once, distribute test execution

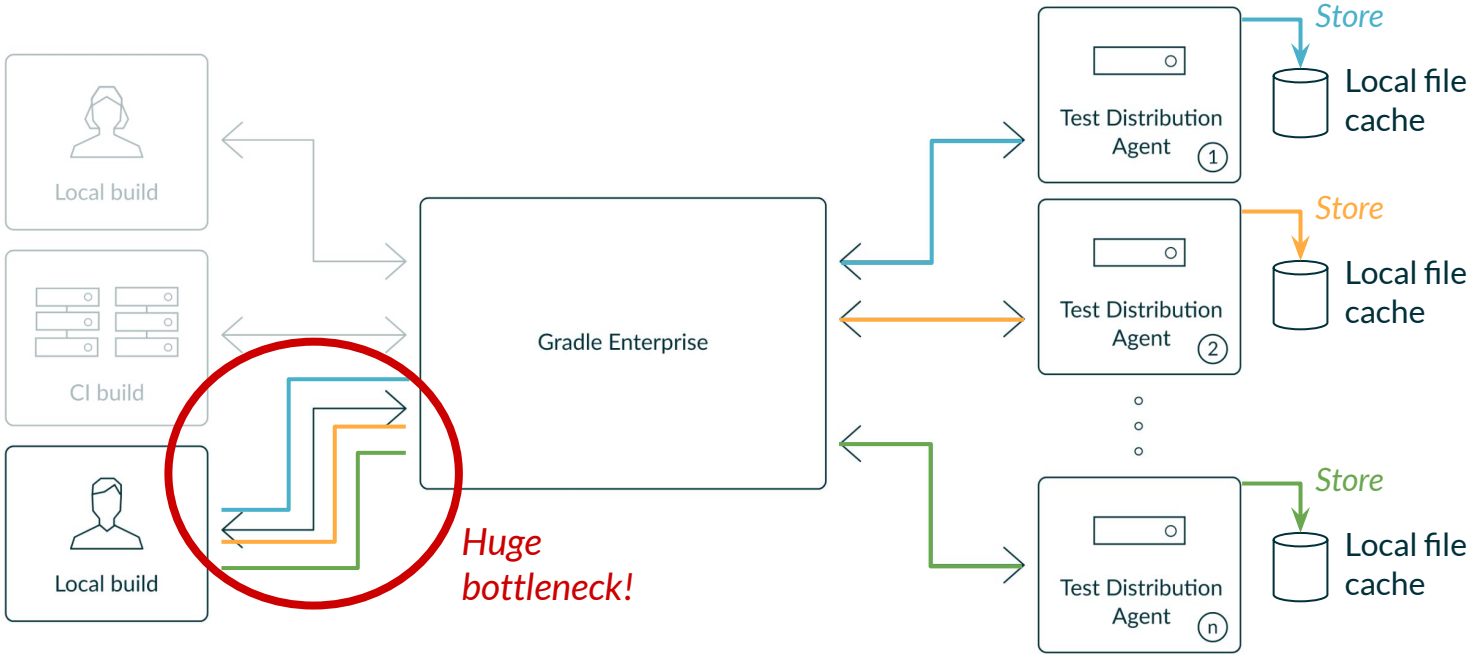


Automatic distribution based on previous execution times



Biggest shortcoming of initial release: file transfer

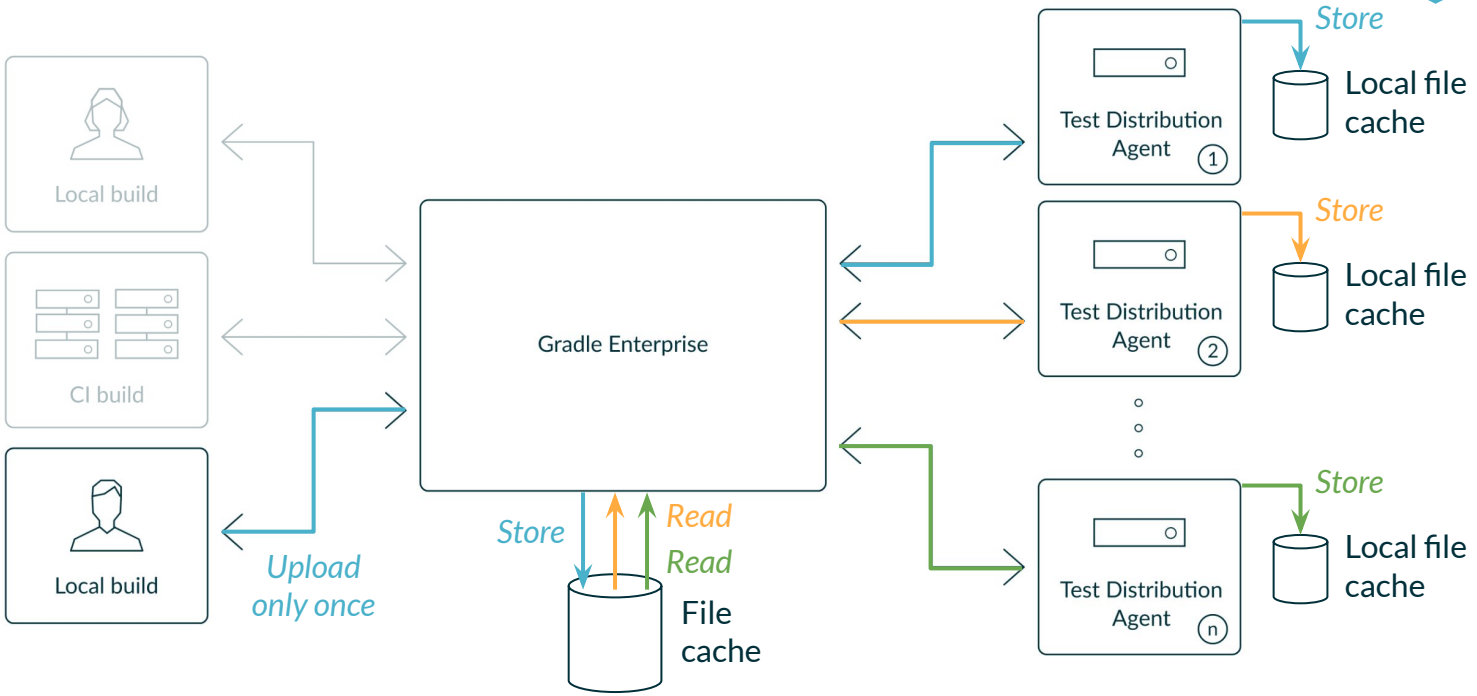
Worst case: m input files with n agents results in n*m files sent over WebSocket connection



File transfer multiplexing/deduplication

Gradle Enterprise **2020.3**

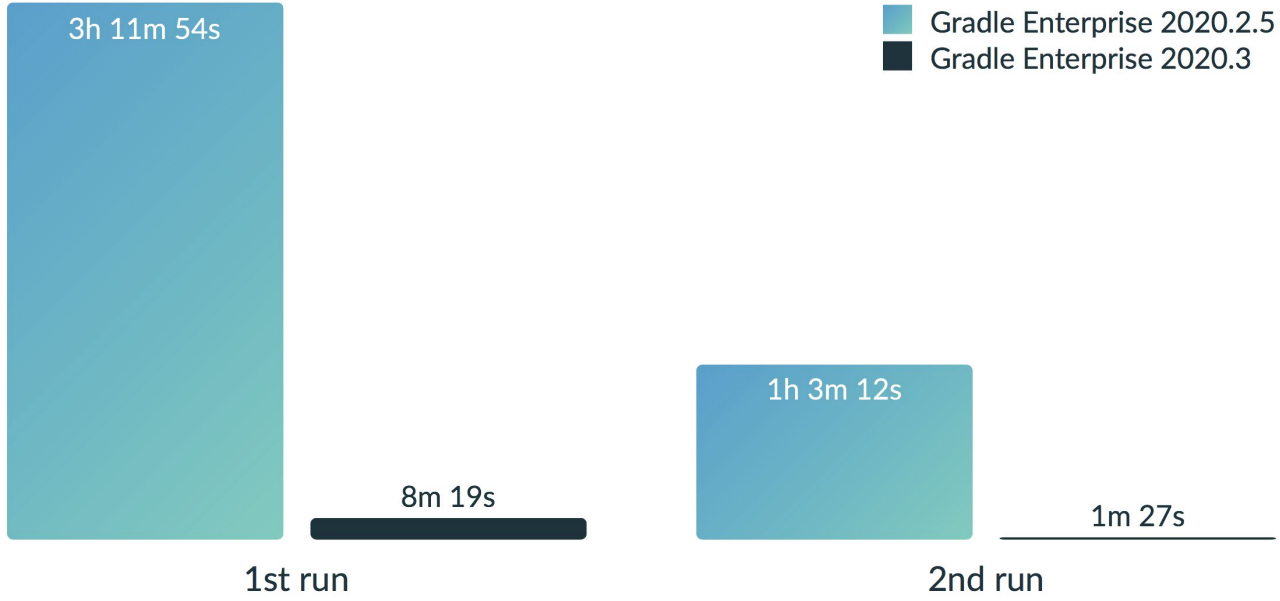
Idea: upload only once (via HTTP) and cache on Gradle Enterprise server



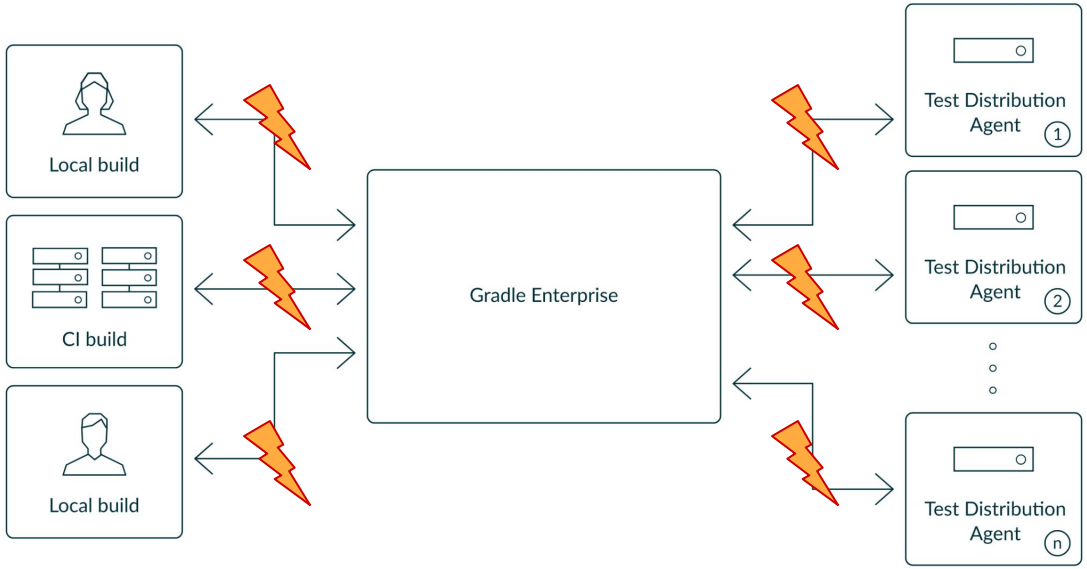
Speedup for over slow connection

Gradle Enterprise **2020.3**

Execution of tests of spring-framework with 24 just started Test Distribution Agents over a slow network connection (LTE, 10 MBit/s upload)



Next problem: network failures causing builds to fail

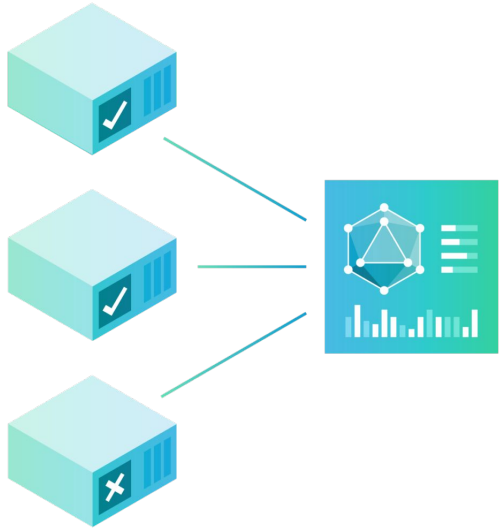


- Test Distribution requires WebSocket connections between builds/agents and the Gradle Enterprise server
- Murphy's law: Anything that can go wrong, will go wrong

Resilience against temporary network failures

Gradle Enterprise **2020.4**

- Actively manage connections using WebSocket pings
- Reconnect if connection is lost or unresponsive
- Reschedule work on other agents if agent disappears
- Retry file uploads on non-client errors
- Avoid builds from breaking and causing disruption



Maven support

Gradle Enterprise **2020.5**

```
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <!-- ... -->
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.2</version>
        <configuration>
          <properties>
            <distribution>
              <enabled>true</enabled>
            </distribution>
          </properties>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

- Requires Gradle Enterprise Maven extension
- Integrates with Surefire and Failsafe plugins

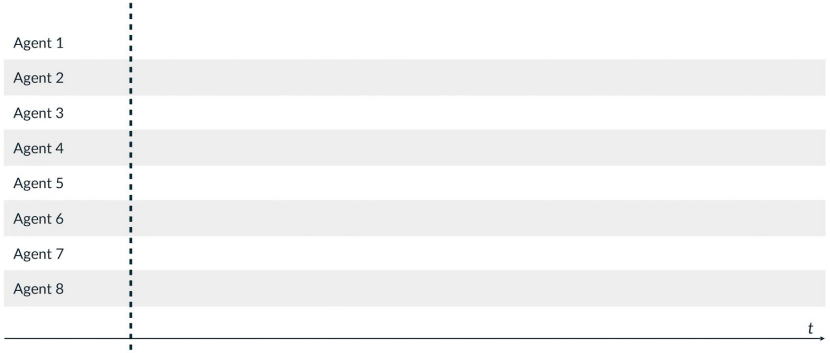
Maven™

Adaptive scheduling

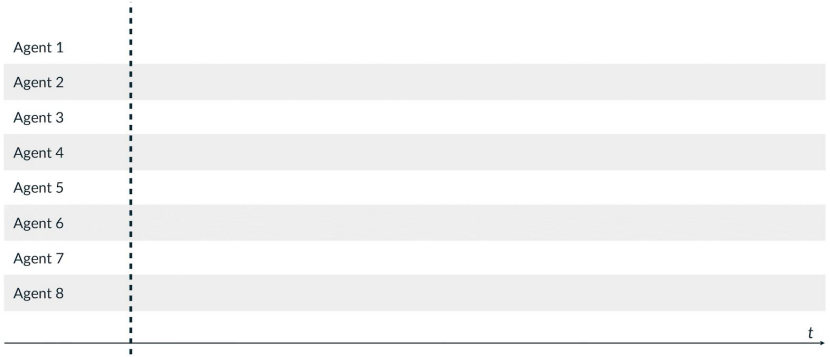
- Be able to react to additional agents becoming available during test execution
- Increase agent utilization
- Reduce testing time

Gradle Enterprise **2021.1**

Gradle Enterprise **2020.5**



Gradle Enterprise **2021.1**

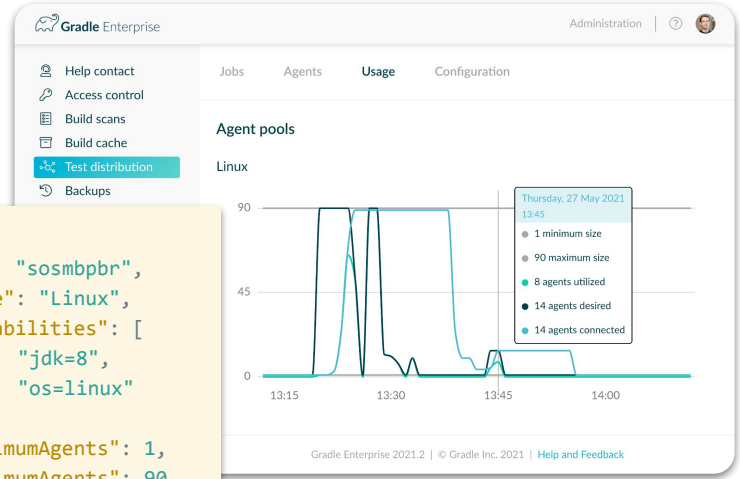


Auto-scaling Test Distribution agents

Gradle Enterprise **2021.2**

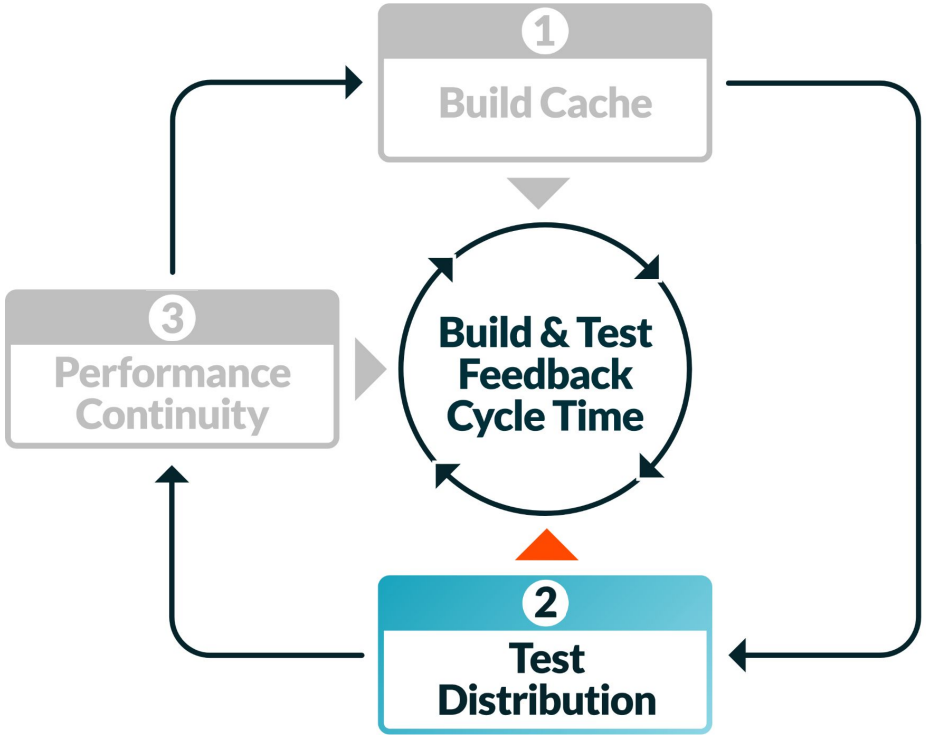
- Agent pools with min/max size and capabilities for horizontal scaling
- HTTP endpoint provides metrics indicating the target number of agents for each pool, based on demand.
- Step-by-step instructions for Kubernetes in docs
- Real-time and historical usage can be visualized by Gradle Enterprise administrators

✅ **Test Distribution is production-ready**




```
{
  "id": "sosmbpbr",
  "name": "Linux",
  "capabilities": [
        "jdk=8",
        "os=linux"
  ],
  "minimumAgents": 1,
  "maximumAgents": 90,
  "connectedAgents": 2,
  "idleAgents": 0,
  "desiredAgents": 8
}
```

Gradle Enterprise Acceleration Features



Gradle Enterprise Acceleration Features