

# Approaching Gradle 3.0

latest efforts, current status & roadmap

# René Gröschke

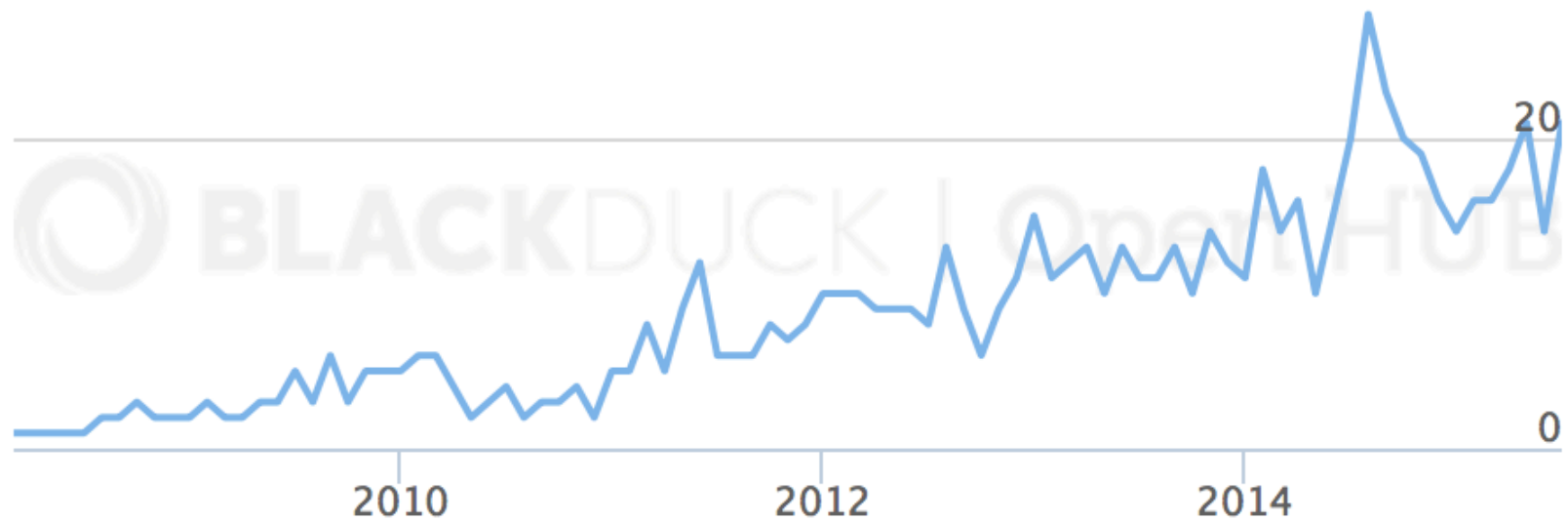
- Principal Engineer @ Gradle Inc.
-  @breskeby
-  [rene@gradle.com](mailto:rene@gradle.com)
-  breskeby

# Gradle in a nutshell

- Multi purpose build system
- Completely open source
- Apache2 licensed
- Governed by Gradle Inc.

# Gradle in a nutshell

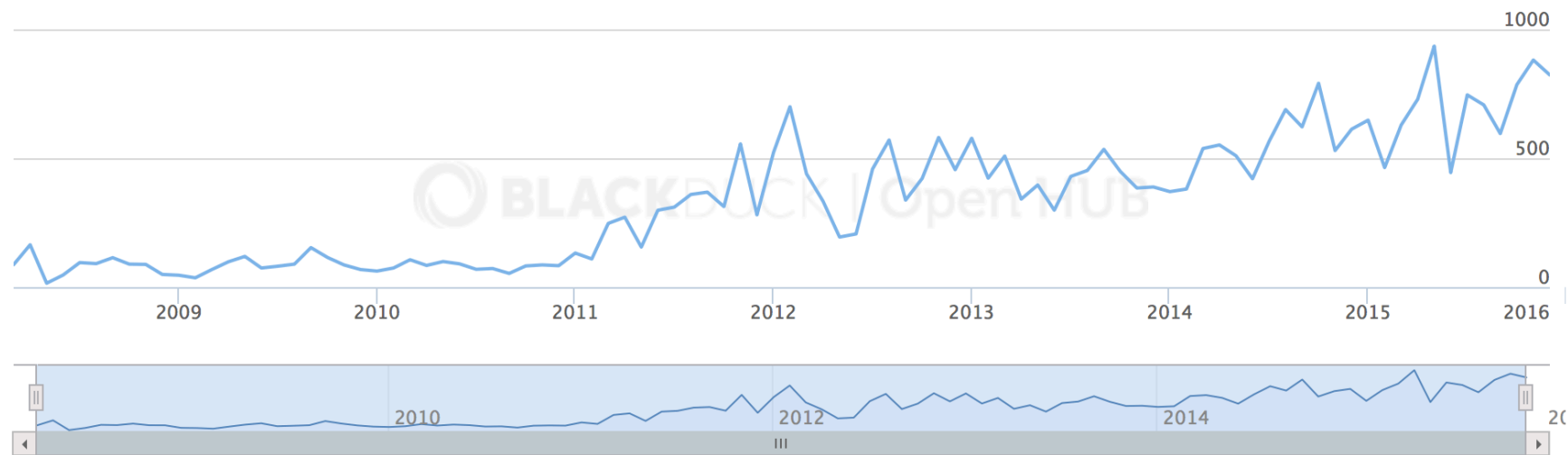
Contributors per Month



# Gradle in a nutshell



Commits per Month

Zoom 1yr 3yr 5yr All



# Gradle in a nutshell

## Language Breakdown

Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
 Groovy	293,402	56,053	16.0%	61,201	410,656	 45.2%

# Gradle in a nutshell

A simple java project

```
apply plugin:"java"

version = file("version.txt").text

repositories {
    jcenter()
}

dependencies {
    testCompile "junit:junit:4.+"
}

task printVersion << { println "We're using - version '$version'!" }
```



# Gradle 2.0

Released 1st July 2014

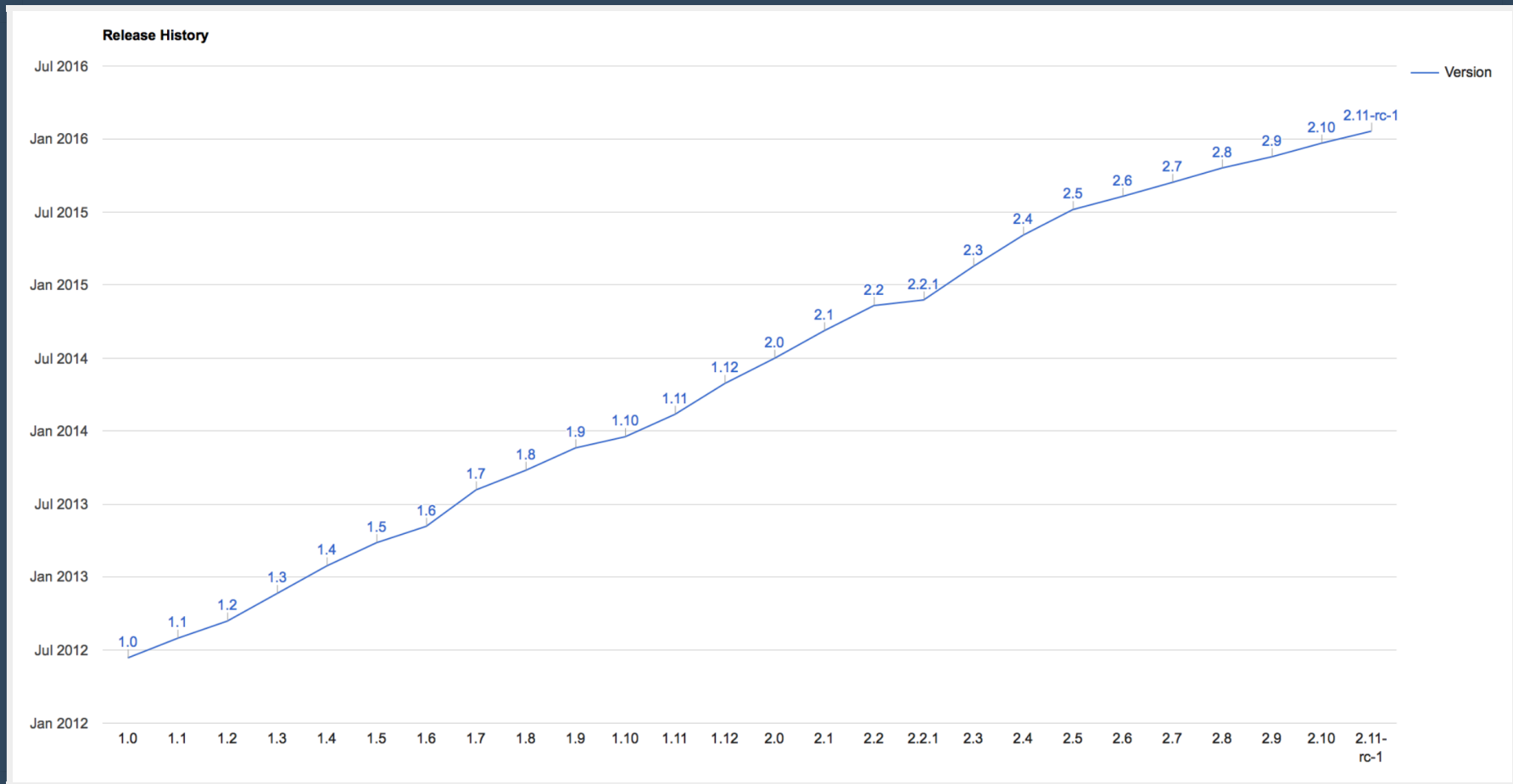


## Gradle 2.0 released

### Old Forum

Jul '14 - Gradle **2.0** is an important milestone in the evolution of Gradle. As explained in the Gradle **2.0** announcement , the change in major version number signals a new...

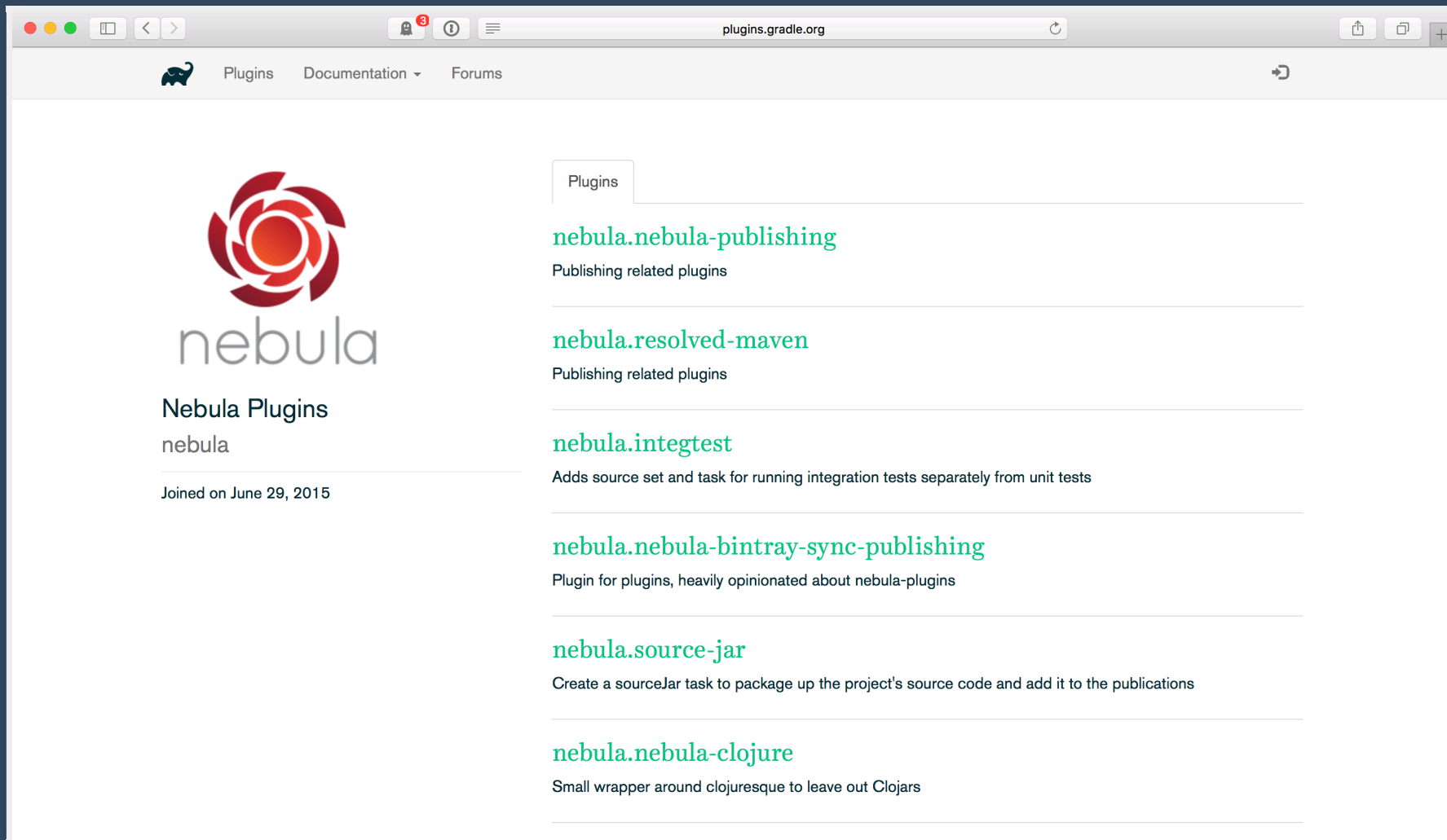
# Gradle Release History



# Let's take a closer look on

- Plugin Portal
- Play Support
- Gradle TestKit
- Even better Dependency Management
- **New Software Component Model**

# Plugin Portal



The screenshot shows a web browser window with the URL `plugins.gradle.org`. The page features a navigation bar with links for "Plugins", "Documentation", and "Forums". On the left side, there is a logo for "nebula" consisting of a red circular icon with a white center and the word "nebula" in a grey sans-serif font below it. Underneath the logo, the text "Nebula Plugins" and "nebula" are displayed, followed by a horizontal line and the date "Joined on June 29, 2015". The main content area on the right is titled "Plugins" and contains a list of six plugins, each with a link and a brief description:

- [nebula.nebula-publishing](#)  
Publishing related plugins
- [nebula.resolved-maven](#)  
Publishing related plugins
- [nebula.integtest](#)  
Adds source set and task for running integration tests separately from unit tests
- [nebula.nebula-bintray-sync-publishing](#)  
Plugin for plugins, heavily opinionated about nebula-plugins
- [nebula.source-jar](#)  
Create a sourceJar task to package up the project's source code and add it to the publications
- [nebula.nebula-clojure](#)  
Small wrapper around clojuresque to leave out Clojars

# Plugin Portal



Plugins

Documentation ▾

Forums



Search Gradle plugins

## com.gradle.plugin-publish

Publish plugins to the Gradle Plugin Portal

<https://plugins.gradle.org/docs/publish-plugin>

#publishing

#plugins

Version 0.9.3 (latest)

[Other versions ▾](#)

Created 14 January 2016.

- Fixes an issue where API keys in ~/.gradle/gradle.properties are ignored (<https://issues.gradle.org/browse/GRADLE-3281>)
- Publishing when a gradle.properties file contains duplicates no longer fails with a java.lang.ArrayIndexOutOfBoundsException

# Plugin Portal

```
...
...
apply plugin: "com.gradle.plugin-publish"
...
pluginBundle {
    website = 'http://acme.org/'
    vcsUrl = 'https://github.com/acme/greeting'
    description = 'Greetings from here!'
    tags = ['greetings', 'salutations']

    plugins {
        greetingsPlugin {
            id = 'org.acme.greeting'
            displayName = 'Acme Greeting plugin'
        }
    }
}
}
```

# Continuous Mode

```
> gradle build -t
```

# Play Support

DEMO



# Gradle TestKit

Functional testing of your build logic

```
apply plugin: 'groovy'

dependencies {
    testCompile localGroovy()
    testCompile gradleTestKit()
}

dependencies {
    testCompile('org.spockframework:spock-core:1.0-groovy-2.4') {
        exclude module: 'groovy-all'
    }
}

repositories {
    mavenCentral()
}
```

# Gradle TestKit

```
def "helloWorld task prints hello world"() {
  given:
  buildFile << """
    task helloWorld << {
      println 'Hello world!'
    }"""

  when:
  def result = GradleRunner.create()
    .withGradleVersion(gradleVersion)
    .withArguments('helloWorld')
    .build()

  then:
  result.standardOutput.contains('Hello world!')
  result.task(":helloWorld").outcome == SUCCESS

  where:
  gradleVersion << ['2.8', '2.9', '2.10']
}
```

# Dependency Management

# Dependency Resolve Rules

Forcing consistent version for a group of libraries

```
configurations.all {
    resolutionStrategy.eachDependency { details ->
        if (details.requested.group == 'org.springframework') {
            details.useVersion '4.2.4.RELEASE'
        }
    }
}
```

# Dependency Resolve Rules

Using a custom versioning scheme

```
dependencies {
    compile "org.acme:someLib:default"
}

configurations.all {
    resolutionStrategy {
        eachDependency { DependencyResolveDetails d ->
            if (d.requested.version == 'default') {
                def version = findDefaultVersion(d.requested.group,
                                                d.requested.name)
                d.useVersion version
            }
        }
    }
}

Object findDefaultVersion(String group, String name) {
    // some custom logic that resolves the default
    // version into a specific version
    "1.0"
}
```

# Dependency Resolve Rules

Changing dependency group and/or name at the resolution

```
configurations.all {
    resolutionStrategy {
        eachDependency { DependencyResolveDetails details ->
            if (details.requested.name == 'groovy-all') {
                //prefer 'groovy' over 'groovy-all':
                details.useTarget(group: details.requested.group,
                                name: 'groovy',
                                version: details.requested.version)
            }
            if (details.requested.name == 'log4j') {
                //prefer 'log4j-over-slf4j' over 'log4j',
                details.useTarget "org.slf4j:log4j-over-slf4j:1.7.10"
            }
        }
    }
}
```

# Component Selection Rules

```
dependencies {
    compile 'org.slf4j:slf4j-api:+'
    testCompile 'junit:junit:4.11'
}

configurations {
    all {
        resolutionStrategy {
            componentSelection {
                withModule("org.slf4j:slf4j-api") { selection ->
                    if(selection.candidate.version == "1.7.10") {
                        selection.reject("known buggy version")
                    }
                }
            }
        }
    }
}
```

# Artifact Query Api

read project licenses

```
task resolveMavenPomFiles << {
  def componentIds = configurations.compile.incoming.resolutionResult.all

  def result = dependencies.createArtifactResolutionQuery()
    .forComponents(componentIds)
    .withArtifacts(MavenModule, MavenPomArtifact)
    .execute()

  for(component in result.resolvedComponents) {
    component.getArtifacts(MavenPomArtifact).each {
      def pom = new XmlSlurper().parse(it.file)
      def licenses = pom.licenses
      // do something with the licenses
    }
  }
}
```



# Dependency Substitution

Allows *elastic* dependencies

```
configurations.all {
    resolutionStrategy.dependencySubstitution {
        substitute project(":api") with module("org.utils:api:1.3")
    }
}
```

# Buildship

- Eclipse plugin developed from scratch by Gradle Inc.
- Part of the eclipse foundation
- Shipped as part of the mars.1 release

# Buildship

Demo

# What's next?

- Composite Builds
- Tooling improvements
- New Gradle Software model

# Better domain modelling

Domain modelling is Gradle's strength.  
We want it to be even better.

## Stronger modeling

↳ The JAR is not the task that creates it.

## Cleaner modeling

↳ Avoid mixing execution concerns into the data model.

## Collaborative modeling

↳ I know how to do something to JARs.

## Comprehensible models

↳ Who is contributing to the contents of this JAR?

# A new Gradle model

# The current model

configuration → execution

- **configuration:**
  - input = build logic
  - output = build model
- **execution:**
  - input = build model
  - output = build artifacts

# Current limitations

- implementation of declarative build api is hard
  - done in the imperative way
- lazyness
- hooks
- eagerness
- scaling



# Too hard

For build engineers and build users.

We can do better.

# The new Gradle model

A new approach to the configuration phase.

Really, the same solution for the "execution phase" applied to configuration.

**A graph of dependent functions**

**An interpretable data model**

# Rule based configuration

## Enter RuleSource

```
class PersonRules extends RuleSource {
    @Model void person(Person p) {}

    @Mutate void setFirstName(Person p) {
        p.firstName = "John"
    }

    @Mutate void createHelloTask(ModelMap<Task> tasks, Person p) {
        tasks.create("hello") {
            doLast {
                println "Hello $p.firstName $p.lastName!"
            }
        }
    }
}
```

# Rule based configuration II

the build script

```
apply plugin: PersonRules

model {
    person {
        lastName = "Smith"
    }
}
```

# The new Gradle model

as an enabler for

- Much faster, more memory efficient builds
- Just configure what is required
- Allow fundamental parallization
- Provide better diagnostics
- Reuse cached configuration
- ...

# Jvm Components

# Gradle 3.0

# Future plans

- Dependency Management
  - Variant aware dependency management
  - arbitrary dimensions
  - custom metadata
- Shared distributed cache
- Next level native build support
- More daemon utilisation
- Continued tooling improvements



# Q & A

# Links and pointers

- <http://gradle.org/gradle-download/>
- <https://www.openhub.net/p/gradle/>
- <https://plugins.gradle.org>
- <https://projects.eclipse.org/projects/tools.buildship>
- [https://docs.gradle.org/current/userguide/userguide\\_single.html#software\\_model](https://docs.gradle.org/current/userguide/userguide_single.html#software_model)
- <https://github.com/gradle/jigsaw-quick-start>
- <http://discuss.gradle.org/c/roadmap>



**Thanks!**

