Configuration Roles Changes in Gradle 8.1

Abstract

This mini-specification outlines what configuration role related changes should be made in Gradle 8.1. This refines and adapts the approach begun with this original specification.

Background

In Gradle 8.0 we added an internal API to allow for locking the usage of configurations at creation time. The `RoleBasedConfigurationContainerInternal` class provided factory methods for constructing new `Configuration` instances in a given role, defined by `ConfigurationRoles`. Our ultimate aim with this API, and a future public version of this API, is to create all *consumables*, *resolvables,* and *dependency buckets* via factory methods that will eliminate the need for clients to know the concrete type implementing these behaviors, freeing us to change them at will.

Goals

- Dogfood the new API as much as possible
    - Avoid using the `LEGACY` role for any configurations we create
- Make it more noticeable when (and difficult to continue) using deprecated configuration roles or changing the permitted usage of a configuration away from what is allowed by the `roleAtCreation`

Proposal for Gradle 8.1

Use new API for Configurations

Update Gradle build to create all publicly visible configurations (not internal or detached) created by Gradle in any core plugin via the new internal role-based API.

Where it is possible to do, without failing smoke tests or causing issues with popular plugins, configurations meant for only internal use should have their usage locked against change upon creation. It is unlikely that there are many places where this will be possible - this is a “stretch goal”.

Any build script `classpath` configurations created by Gradle should be created as `INTENDED_RESOLVABLE_BUCKET`. Any detached configurations we create will also need to be created in the `INTENDED_RESOLVABLE_BUCKET` role. Their usage will remain changeable in case of strange edge cases where they are consumed.

After this change, there should be no configurations that are created by our plugins that do not have a specific role in mind. Ideally, Gradle should not use the `LEGACY` role anywhere after this change. This probably isn't fully possible yet, but any obstacles to this should be cataloged. If it is possible, we should prevent using this role with the internal role-based creation APIs (we will still use the role for configurations created by clients using the existing public API).

Warn Upon Changing Usage for Non-Legacy roles

For **non-`LEGACY`** roles, if you change the usage, you will now get a deprecation warning saying not to do this.

The exception is when the `apiElements` and `runtimeElements` configurations are changed to have a usage **disabled** only, as this is done by Kotlin. Kotlin is doing this so that Dependency Management doesn't see these variants (because they would conflict with other variants it adds). We will **not** deprecate this behavior or warn when this is done.

Revise/expand Deprecated Configuration usage warnings

When a configuration has deprecated marked for a particular usage, indicated by one of these flags set in `DefaultConfiguration`:

```
    private boolean consumptionDeprecated;
    private boolean resolutionDeprecated;
    private boolean declarationDeprecated;
```

If that usage is used (i.e., if the configuration is consumed, or resolved, or a dependency is declared against it), a deprecation warning is emitted. This warning should be updated to inform the client that they should either:

1. Stop using configuration with role X in usage Y.
2. See the documentation for advice on edge cases where this is not possible..

The warning should point to a section of documentation outlining configuration usage and why it should not change. The warning should report the `roleAtCreation` if one is set. The documentation should mention the Resolvable extends Bucket pattern and when it is applicable.

Additionally, when configurations are **created** in a `DEPRECATED_XYZ` role a deprecation warning should be emitted. (Note that since the API for role-based creation is not yet public, this shouldn't ever happen.)

Won't Do

- We will **not** attempt to make the role-based configuration creation API public in this release.
- We will **not** attempt to offer a convenience method/class/pattern for the common case of a resolvable linked to a new dependency bucket.
- We will **not** remove the ability to create configurations outside the role-based creation API (though we will attempt to remove all creation of configurations in the gradle/gradle build which do **not** use that API).
- We will **not** remove the ability to create ad-hoc configuration roles.
- We will **not** attempt to change the `Configuration` class hierarchy to specialize each role into a separate implementation class.
- We will **not** attempt to use the new roles api for creation of custom configurations in the gradle/gradle build (outside of configurations created automatically by the core plugins).
- This change is not meant to refine what each `canBeConsumed, canBeResolved, etc.` flag permits or denies.
- Provide opt-in visibility into latent misuse of configurations to interested early adopters so they can begin addressing these issues (see section below).
- Provide opt-in strict mode specific to this functionality to make usage related deprecation warnings into errors without making all deprecations into errors.
- Attempt to consolidate the various usage/role/deprecation fields to reduce the size of the state we track in the `DefaultConfiguration` class - this will be considered for 8.2.

`org.gradle.configuration.changing.usage` property

Add a new property, which when set to "warn", will cause any change of a configuration's usage to emit deprecation warnings. The implementation of this is already baked into the `DefaultConfiguration` class as the `warnOnChangingUsage` flag.

Additionally, if this property is set to "strict", it will cause any change to a configuration's usage to throw an exception.

By default, this property will **not** be set, and changing usage will remain allowed. In the gradle/gradle build, this property will be set to "strict".

Enabling this property will be publicized in the Gradle 8.1 release notes (and perhaps a blog post?) as a suggested best practice. The 2 levels of response to changing usage provide for an easy assisted migration path away from relying on changing configuration usage after creation.