Per-target execution platform constraints for exec groups

- **Authors:** fabian@meumertzhe.im (Fabian Meumertzheim)
- **Status:** Approved
- **Reviewers:** jcater@google.com (John Cater)
- **Created:** 2024-08-08
- **Updated:** 2024-08-15
- **Discussion:** #23245

Please read Bazel Code of Conduct before commenting.

Background

Execution groups offer a rules author great flexibility in influencing on which platform the different actions registered in the rule are executed. Platform constraints can be set directly on each execution group via the `exec_compatible_with` parameter and the `toolchains` parameter allows for more complex "target to execution constraints" mapping using the idiom of a test runner toolchain.

However, the introduction of execution groups reduced the flexibility offered to *users* of rules: There is currently no way in Bazel for a user to influence the execution platform of an action that is assigned to a non-default execution group that doesn't specify any toolchain type in `toolchains`. For the default execution group, this is possible by adding constraints to the `exec_compatible_with` attribute defined on all rules. There is currently no equivalent of this attribute for non-default execution groups.

Related issues and PRs:

`cc_test` should respect `exec_compatible_with` for test action (#23202)

Support running tests on the target platform (#12719)

--use_target_platform_for_tests doesn't support exec property inheritance (#17466)

Proposed solution

New attribute `exec_group_compatible_with`

Every rule class gains a non-configurable attribute `exec_group_compatible_with` of type `dict[str, list[Label]]` (this type doesn't exist yet). When processing an execution group (automatic or explicitly defined) for a given target, all constraints listed under the key corresponding to the group's name via this attribute are added to the execution constraints specified in the execution groups `exec_compatible_with` argument. The default execution group has the name `"default".`

Automatic execution groups are automatically named after their toolchain type and could thus be referenced in this way. Note that this slightly increases implementation complexity as any key that could be a label would need to go through repo mapping before being matched against the names of automatic execution groups.

Alternatives considered

- Having `exec_compatible_with` add constraints to *all* execution groups: See the example below.
- Adding a `<exec_group>_exec_compatible_with` argument to a rule class for each execution group it defines: Bazel doesn't yet have any magic attributes with dynamic names, which suffer from discoverability problems and wouldn't support referencing automatic exec groups. Since Bazel already has a `dict[str, list[str]]` attribute, the new attribute type improves consistency and also doesn't introduce incompatibilities with tooling that parses BUILD files.
- Representing the default execution group with a different name such as `""`: While this would address the theoretical concern of collisions with existing exec groups, it would be more difficult to understand. Since execution groups aren't in widespread use yet and automatic execution groups will make explicitly defined execution groups even more rare, there is no realistic chance of a problematic collision.

Backwards compatibility

It's possible but highly unlikely that a rule class already defines an `exec_group_compatible_with` attribute. Even if so, since its type isn't available to rules yet, collisions could be avoided by checking the type of the attribute.

Since execution groups can be referenced by name in the `exec_properties` of a rule, they are already exposed as public API of a rule class. Adding new ways to reference them by name thus doesn't add any new backwards compatibility concerns, it just strengthens existing ones. It should be considered whether `bazel (a)query` should get better support for querying the execution groups of a rule (class).

`exec_compatible_with` applies to all non-`test` execution groups

`exec_compatible_with` currently applies its constraints to the default execution group as well as all automatic execution groups, but neither to the `test` execution group nor any explicitly defined execution group. Instead, it would apply to all non-`test` execution groups. This improves consistency between explicit and automatic execution groups, which is especially important since the latter are supposed to become the new default, with explicit execution groups being much more rare.

Alternatives considered

- Not changing what `exec_compatible_with` applies to: With the new attribute, there is now a clear and unambiguous way to add a constraint to the default execution group if needed, making this attribute redundant in that case. This would also retain the inconsistency between automatic and explicit execution groups. Furthermore, applying a constraint to all non-`test` execution groups of a rule that doesn't use automatic execution groups would otherwise require listing all of them, which creates a maintenance burden.
- Having `exec_compatible_with` add constraints to *all* execution groups: See the example below, it is very common for test actions to have constraints that are entirely unrelated to the actions that produce the test binary.

Backwards compatibility

Rules that define explicit execution groups and are annotated with `exec_compatible_with` would now apply these constraints to all execution groups (e.g. `cpp_link`), which could break builds. Users would need to migrate to using `exec_group_compatible_with = {"default": [...]}`. This change should thus be gated behind an incompatible flag, but the migration isn't expected to be difficult. Wherever possible, rules should migrate to automatic execution groups anyway.

Example

Consider a `cc_test` target with `srcs` that make heavy use of a template-based DSL for generating code that ultimately runs on a GPU. Since templates are slow to compile, the compilation action (default execution group) for this target should run on a machine with high single-core performance (let's assume such machines are described by execution platforms with the constraint `//:has_fast_cpu`). Linking the test binary has no special requirements, so the link action should just use the unmodified `cpp_link` execution group. However, the actual test action (`test` execution group) needs to be run on a machine with a GPU (i.e., on an execution platform with the constraint `//:has_gpu`).

With this proposal, this scenario could be modeled as follows:

Python
```python
cc_test(
    ...,
    exec_group_compatible_with = {
        "default": "//:has_fast_cpu",
        "test": "//:has_gpu",
    },
)
```

This example shows why only having `exec_compatible_with` add constraints to all execution groups wouldn't be flexible enough: the test action would pick up both the `//:has_fast_cpu` and the `//:has_gpu` constraint, but there may be no machines, and hence execution platforms, that satisfy both constraints simultaneously.

Document History

Date	**Description**
2024-08-08	First proposal
2024-08-15	Addressed first round of comments

Bazel