Bazel

Target-level execution properties

Please read Bazel Code of Conduct before commenting.

- **Authors**: agoulti@bazel.build

- **Status**: Implemented
- **Reviewers**: katre@bazel.build
- **Created**: 2019-06-21
- **Updated**: 2019-06-21

Overview

Platforms is a powerful mechanism that allows service providers and rules owners to hide setup complexity from end users, while allowing to set up complex multi-platform builds that would "just build" with no special command line customization. Platforms allow specifying partial constraints on targets and toolchains and setting it up so that the correct combination of toolchains is selected for each target.

Platforms has also been used to pass information for remote execution, e.g., to specify a container to run in and certain configurations. Some of this information affects toolchains selection (such as the docker image to select) while some do not (e.g., allowing network access). Currently, whether the property affects toolchain selection or not, the user needs to create separate platforms and use per-target constraints to select between them. This has proven to carry significant cognitive load and is easy to get wrong.

We want to simplify the way execution properties are defined in cases where they do not affect toolchain execution.

There is an approved proposal to transition to specifying platform properties and a Skylark dictionary. Here we propose adding a per-target mechanism to add such properties on top of a platform selected by the standard platform selection mechanism.

Proposal

We will add a new field `exec_properties` that will be applicable to all targets. This will be a field of type Starlark dictionary, in the same format (and accepting the same data) as the proposed exec_properties field for platforms.

The target configuration would then look something like this:

```
cc_test(
    name = "remote_coredump_unittest",
    size = "small",
    srcs = [
        "remote_coredump_unittest.cc",
    ],
    linkstatic = 1,
    exec_properties = {
        "property1Name": "property1Value",
        "property2Name": "property2Value",
    },
    deps = [
        ":base",
        "//util/process",
    ],
)
```

The extra properties would be added to the dictionary of the platform that is selected using the standard mechanism. If the same key is present on the platform, the value specified on the target would override the value of the platform[1].

Service providers and larger projects will be able to add convenience macros to generate such dictionaries and abstract away the implementation details. The user configuration would then look something like the following:

```
load("@provider_repo//properties.bzl", "props")
cc_test(
    name = "remote_coredump_unittest",
    size = "small",
    srcs = [
        "remote_coredump_unittest.cc",
    ],
    linkstatic = 1,
    exec_properties = props.create(networking = True),
    deps = [
        ":base",
        "//util/process",
    ],
)
```

[1] Eventually we may want to ensure that the platform would only contain properties directly relevant to toolchain selection, and fail instead of overriding if the target specifies a conflicting property. However, this requires the ability to specify per-build execution property defaults. Without this mechanism, the workaround is to specify execution property defaults in the platform. Once build-level execution properties are possible, we may want to revisit this decision

Alternatives considered

We could create a new provider (Starlark type) that would wrap the dictionary in a level of abstraction. Then the exec_properties could be a list of targets wrapping these dictionaries.

Pros:

- Easy to compose (currently will need a macro and/or a Skylark "+" operator for dictionaries)

Cons:

- Cannot write values directly on the target

Compatibility

This proposal adds a new field and should be fully backward compabile.

Document History

Date	Description
2018-10-22	First proposal