Execution environments as an extension to the options system

Authors: Christopher Neugebauer, Eric Arellano (2022-09-06)

Per the discussion in https://docs.google.com/document/d/1vXRHWK7ZjAIp2BxWLwRYm1QOKDeXx02ONQWvXDloxkg/edit#, a proposal exists to add “execution environments” to Pants as a way to define options that apply to tools that Pants runs. This document refines the interpretation of environment targets into a way to specify environment-sensitive options.

Executive summary of the problem

We’ve noticed a disconnect between out-of-the-box pants configuration, and what are currently easy configuration tasks (e.g. setting global search paths for Python), which the current environments proposal makes significantly more difficult than the status quo.

Consider the case where a user wants to set an option: at the moment, you just need to add a single option to a `pants.toml`, a command line switch, or an envvar. The proposal, as it currently stands, would require the user to add an execution environment before they can set that option. This would make upgrading to a new version of Pants difficult, and make adoption difficult more generally.

Instead, it should be possible to incrementally adopt environment-sensitive features, and then easily switch to multi-environment support when you discover that you need it.

Executive summary of proposed solution

My proposal is to treat environments as an “override” mechanism for selected options, by introducing a concept of “singleton options” (i.e our current options behaviour) and “environment-sensitive options” (new behaviour).

Only “environment-specific options” will be allowed to be set in environment declarations. These will be opt-in at the options API level.

This would allow users to have a global configuration for these options at first, and then when you move to needing environments, you can migrate the configurable options incrementally as you need them, rather than switching to the environments idea entirely.

This approach should make it significantly easier for plugin authors to add support for consuming environments (or to choose not to), and to make it easier to treat environments as an advanced use case, rather than a necessary early cliff in Pants' learning curve.

Proposed Solution

Modelling changes

Options may be "singleton" or "environment-sensitive"

A new setting will be added to our options API, defining the applicable contexts for the option. An option may be either "singleton" (the current behaviour for *all* options), or "environment-sensitive" (a new behaviour which must be opted into, for values where it makes sense).

Singleton will remain the default/no-choice-specified cardinality to ease the upgrade path for plugin developers. Plugin developers will need to opt-in to "environment-sensitive" options by specifying the "environment-sensitive" setting. This should be the only work the plugin developer needs to do to opt into environments.

End-users configure execution environments as targets

Per the original proposal, execution environments are defined as targets. This retains the existing Pants philosophy of developer-oriented options being defined in BUILD files and system configuration existing elsewhere. (Strictly adhering to this philosophy would make defining aliases for execution environments an open question, but the `pants.toml` approach seems like a good start. Maybe it's worth revisiting aspects of this?)

Inside an environment target, users may ONLY (re)define values for environment-sensitive options.

Option fields in targets will always share names with the corresponding global settings

Field names will be generated systematically, with a predictable namespacing rule to avoid ambiguity: `subsystem_name__option_name`

Options specified through environment targets override global settings

Environment-sensitive options may continue to be set using our existing options mechanisms (pants.toml, envvars, command line options; now known as "global" configuration). If a value is not set in the current environment target, the value from the global configuration will be used.

DISCUSSION POINT: It is not clear to me how our override mechanisms (envvars/command line options) should interact with options set in environment targets. Perhaps there should be a warning/error option for cases where manual overrides conflict with environment targets?

Implementation details

Options are resolved at subsystem creation time

Currently options subsystems are effectively singleton objects. We intend to change the subsystem construction to depend on environment targets, so that environment-sensitive values can be resolved during construction. This means that rules that construct subsystems must now be aware of `EnvironmentName`.

This will allow rules that consume subsystem options to work without being aware of how the underlying option was set.

Fields for `local_environment` are created at runtime based on options defined on each subsystem

Rather than plugin authors manually specifying fields for each environment-sensitive option, field classes for each option will be created at runtime, and installed as plugin fields on `local_environment` (and other similar targets). There's a few reasons for this:

1. Using targets/BUILD files becomes an implementation detail that plugin authors don't need to rely on
2. The conceptual modelling for a plugin author becomes a lot simpler – environment-sensitive options are just "options", and adding new options is DRYer
3. Enforcing systematic naming/namespacing of options inside a `local_environment` field is trivial
4. We get automatic access to the existing data validation logic that exists for fields.

There is already prior art for inspecting unconstructed subsystems to find options attributes – see https://github.com/pantsbuild/pants/blob/6dda95b425203db32f05eb3d4317655929162332/src/python/pants/option/option_types.py#L22-L31