Dynamic defaults for target field values

GitHub discussion thread: https://github.com/pantsbuild/pants/discussions/15809

Date	June 13, 2022
Status	Accepted
Implemented in Pants version	2.14.0
Pull Request	https://github.com/pantsbuild/pants/pull/15836

Background

Today, target type authors decide what fields to support for a target, and the characteristics of those fields, such as value type, description and default value.

The Plugin API allows users to add additional custom fields to any target, but have no control over existing fields.

Target type authors may not be able to account for all uses of a particular field in terms of a suitable default value, having to resort to a compromise that will be good enough for the majority of uses.

Current praxis

The current way that this has been met in Pants is to use subsystem options such as the `pex-binary-defaults` or `docker.default_repository`.
https://www.pantsbuild.org/v2.12/docs/reference-pex-binary-defaults

The problem with this is duplication and incompleteness and only a single global default override for the whole repository, where plugin fields may also never have a dedicated overridable default.
See github issue for one example of incompleteness:
https://github.com/pantsbuild/pants/issues/15776

Another option available to users is to use macros. The limitation is that it requires BUILD file authors to remember to use the macros rather than the actual targets, which makes it easy to get wrong and more difficult to maintain.

Use Case

In order for users to adapt the default value for particular types of fields to better suit their needs, the amount of required BUILD file configuration is reduced resulting in a more maintainable project for the end user.

Motivation

This has come up on several occasions in the past, where users have asked about overriding the default value for the Python sources field, for instance, showing that this is a feature that has been missing albeit currently with unknown impact.

More importantly this feature will be essential to support the new “visibility” feature, where the alternative would require users to specify the visibility for every single target in a whole package tree, which would be very unmaintainable and error prone.

API

The proposed user API for declaring default field values is to introduce a new type: `__defaults__`.

The `__defaults__` type accepts dictionaries with default field values per target type or types when provided in a tuple of targets. The syntax is the same as used for the `overrides` field, to allow providing default field values for a common set of targets at once.

Any kwargs are for options to the `__defaults__` declaration. See list below.

The `dictionaries` are for the field names with corresponding new default values.

These default values then apply for all targets with those fields in the current BUILD file, and all child BUILD files, overriding any parent defaults that may have been provided for the same targets/fields.

This solely impacts the default of impacted targets. If the target explicitly sets the field, including via `overrides`, the default is ignored.

Options passed as kwargs to `__defaults__`:

- `all`: A dictionary with default field values to apply to all targets. Unknown fields will be ignored.
- `extend`: A boolean indicating whether to extend the defaults for each target, or replace them.

Example BUILD file:

```
# Defaults for specific targets
__defaults__({
  python_sources: dict(
    sources=["*.py"]
  ),
  resources: dict(...),
  ...,
})

# Defaults for all targets (limited to COMMON_TARGET_FIELDS, due to field
validation per target)
__defaults__({
  ...,
  },
  all=dict(
    tags=["foo-tag"],
    ...,
  )
})

# Syntax to support providing default field values to a set of targets
__defaults__({
  (files, resources): dict(...),
  ...
})

# Regular targets follow...
python_sources(...)
```

User considerations

The user will need to consult peek to find out what defaults are applied to a certain target in case there are applicable defaults from a parent BUILD file, for instance.

It will also be important to understand the relation between target generators and atom targets, and what the effect is to provide defaults for the generator or the generated targets.

TODO: expand with an illustrative example once we have an implementation to verify with.

Implementation

In a POC, the defaults type was implemented as a regular target, and the default value was looked up with an `@rule` to provide the applicable defaults value for each target.
It shows that the concept of defaults works, but it does not integrate deeply enough as the default values are not propagated properly into the subject targets.

The benefit was visibility of the defaults, though, as they show up when listing targets and are peek-able.
Perhaps a hybrid approach would be beneficial, where the end product also is a target, but merely used for introspection purposes for the user.

The defaults implementation integrates directly with the BUILD file parsing and the TargetAdaptor mapping a set of field values to a target instance.

Deprecates options

By implementing the new defaults feature, this list of options may be deprecated (add to this list as you see fit):

- [ ] (all of) [pex-binary-defaults]
- [ ] [docker].default_context_root
- [ ] [docker].default_repository
- [ ] [python].default_resolve
- [ ] [jvm].default_resolve
- [ ] [setup-py-generation].generate_setup_default
- [ ] …

Alternatives considered

The issue Consider adding explicit support for applying build metadata to multiple targets at once goes over both motivation and alternative approaches on this topic.