Distribution initiative: Drupal recipes & Composer project templates

Document contributors (alphabetical d.org usernames) (Please comment here with your username)
alexpott, bbrala, bircher, bsnodgrass, catch, chr.fritsch, daniel.bosen, dries, dww, Finn Lewis, Gábor Hojtsy, greggmarshall, Kingdutch, kreynen, larowlan, mherchel, mixologic, msherron, nerdstein, phenaproxima, realityloop, Shawn DeArmond, thejimbirch, Webbeh, xjm

This document has moved to a drupal project. See https://git.drupalcode.org/project/distributions_recipes. Please create issues against the repository / help us keep the documentation maintained. Thank you for all the contributors - see https://git.drupalcode.org/project/distributions_recipes/-/blob/1.0.x/docs/contributors.md

Glossary

Current terminology

Install profile
: A drupal extension that can contain modules, themes, configuration, content and describes how to install a site.

  There are special distinct types of install profile:

  Distribution
  : A special form of install profile which selects itself when uniquely available in the installer. Users of distributions expect upgrades to future versions and some level of support.

  Starter kit
  : A special form of install profile (for example Contenta or Foundry) that provides a place to start with no upgrade path for the starter kit. For an early usage of this terminology with respect to install profiles see this video (around 16:50).

Project
: A drupal.org / composer packagist concept. Can be a single module or a group of modules. For example: Access Unpublished is a single module and Devel contains a group of modules. On drupal.org an install profile is also a project. Note: we often refer to modules that are not the same as the project name as sub-modules but in the Drupal extension system and composer eco-system the concept of sub-modules has little meaning. The Drupal extensions considers all modules equally and Composer

has no knowledge of modules and only knows about projects. The Update module in core is project-aware.

Module
: Provides functionality and configuration in a single extension.

Theme
: An extension that defines the visual look and feel of your site.

  Starterkit theme
  : A tool for generating Drupal themes with default markup that is maintained in core.

Feature
: A special type of module comprised of different site building components built using the Features module.

Tours
: A tour walks users through how to configure and use Drupal. They are configuration entities provided by modules and install profiles.

Composer project template
: A composer.json that via `composer create-project` supplies a starting point for a Drupal site. As part of the initiative these will get new capabilities to replace some of what install profiles (especially distributions) can do today. See "What is a Composer project template?" below.

New terminology

Drupal recipe
: See "What is a Drupal recipe?" below.

Install profile, distribution and starter kits are retired. Project and Module continue to mean the same as before.

Install profiles: what's the problem?

Install profiles are an inflexible solution that encourages lock-in and when they go out-of-support they leave a site stuck. Install profiles cannot be uninstalled without using additional contrib. They cannot build on other install profiles because inheritance of configuration is a very hard problem to resolve.

Here are some of the many issues of the years that describe people's problems:

- Allow install profiles to be switched
  https://www.drupal.org/project/drupal/issues/1242956
- Install profile is disabled for lots of different reasons and core doesn't allow for that
  https://www.drupal.org/project/drupal/issues/1170362
- Add support for recommends[] and fix install profile dependency special casing
  https://www.drupal.org/project/drupal/issues/820054
- Allow profiles to define a base/parent profile
  https://www.drupal.org/project/drupal/issues/1356276

What is a Drupal recipe?

Drupal recipes allow the automation of Drupal module install and configuration via the user interface and via the Drupal recipe composer plugin.

A Drupal recipe is a tool for Site Builders and Developers to add functionality to a Drupal site. It is like a recipe that provides a series of steps to add functionality. These steps could be taken manually to arrive at the same point. For example, I might want to provide a blog recipe that configures modules to provide a great blogging experience and creates a few sample posts. Or it can include a number of other Drupal recipes and modules to build a full site experience. Ideally Drupal recipes would not configure an entire site themselves but would break out their functionality into smaller Drupal recipes that can be reused by other Drupal recipes.

Drupal recipes are:

- Applied to Drupal sites, they are not installed.
- Easy to share
- Do not lock sites in
- Composible from other Drupal recipes

Don't like the name? See https://www.drupal.org/project/drupal/issues/3283151

Example Use cases

Recipe content type

Provides the content type, fields, form and view displays, and views necessary for recipe functionality on a site. Here's an example of how the recipe might break down:

1. Composer require - nothing to do - all modules are supplied by core
2. Module install - ensure that node, views, field dependencies are installed
3. Configuration - create node type, fields, entity displays and views that are required.
4. Configuration - update author and editor with permissions if they exist.
5. Content - create demo recipe content if selected by the user.

Umami site

Provides the full Umami experience.

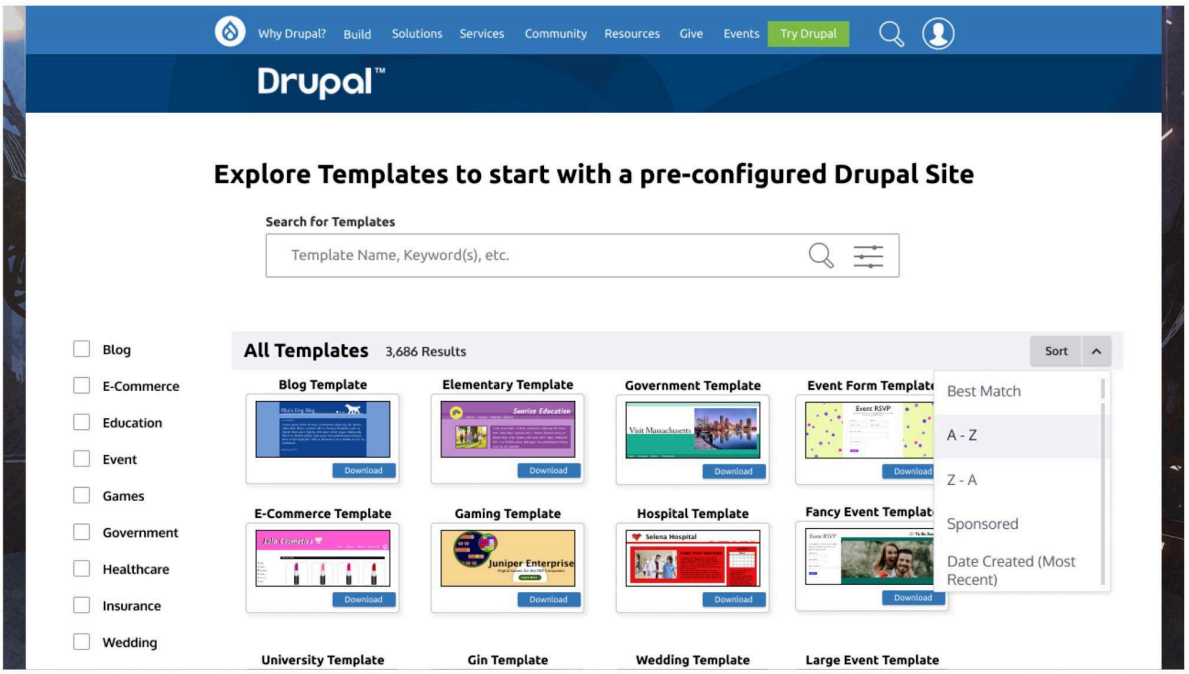
1. Applies the Recipe content type recipe
2. Installs other modules used by Umami that are not already installed
3. Configures the modules
4. Installs the frontend and admin themes
   Notes: once a Drupal recipe is setting theme related configuration it becomes less useful for many sites. Therefore we should discourage this in Drupal recipes that aspire to widely used and provide functional building blocks for generic use-cases.
5. Configures the install to look as expected.

SEO best practices

Provides a recipe to set up SEO features on your site.

1. Composer require - requires the metatag and metatag_async_widget project
2. Module install - installs the metatag, metatag_open_graph, metatag_twitter_cards and metatag_async_widget modules
3. Configuration - adds meta tag field to existing node types, form and view displays, creates metatag_defaults config for existing node types
4. Content - do nothing

Mockup from Driesnote



Note this UI is subject to change and likely to be heavily based on the work done by the Project Browser initiative.

What can a Drupal recipe do?

- Provide a list of other Drupal recipes to apply. Drupal recipe inheritance is not possible. Drupal recipes can be composed.
- Provide a list of projects to composer install
- Provide a list of modules to install - this step should only install the simple configuration provided by the modules. It should ignore all the configuration entities in order to allow the configuration step to do exactly what it wants. This means that we do not have to provide a way for Drupal recipes to remove module provided configuration entities.

- Provide a list of themes to install
- Provide configuration
    - Choose config from the modules it has listed
    - Supply new configuration
    - Update existing configuration
      The ability to script configuration updates to existing configuration is a key new ability for Drupal recipes. They will be able to use this to add permissions to existing roles and fields to entity displays and views. The Config Actions module offers some prior art in contrib.
      Proposal: Config actions are implemented as PHP attributes on methods of the Configuration entity.
    - Note is it important that configuration change is only updates and additions. Removals will make it very hard for recipes to work alongside each other.
- Provide content
    - Add new content
    - Update existing content? (maybe a later aim)
    - Delete content
- Provided suggested other Drupal recipes you might want to apply afterwards
- Provide tests
    - At a minimum core should provide a test that can be run against any Drupal recipe that proves it is installable and that configuration provided by the Drupal recipe matches the configuration schema. There should be a gitlab ci template that makes running this test against a Drupal recipe easy.
- Have the following metadata:
    - Name. A short name for the Drupal recipe.
    - Description. A longer description of the Drupal recipe.
    - Type. Groups related Drupal recipes together. The type 'Site' is a special case that allows such Drupal recipes to be listed on the installer.

What can't a Drupal recipe do?

- Provide functionality itself, for example, implement hooks and services. If a Drupal recipe requires special functionality that needs code this should be provided in a regular Drupal module.
- Provide an upgrade path. Once a Drupal recipe has been applied responsibility for updating configuration and content falls to modules and core.
- Be part of a deployment. If you want to deploy a Drupal recipe you run it against a site and then deploy as you would right now.
    - Deploying the content created by a Drupal recipe is out-of-scope.

An example Drupal recipe file

```
name: 'An example Drupal recipe'
description: 'A description of the functionality'
# The type key is similar to the package key in module.info.yml. It
# can be used by the UI to group Drupal recipes. Additionally,
# the type 'Site' means that the Drupal recipe will be listed in
# the installer.
type: 'Content type'
apply_first:
  # Other Drupal recipes that should be applied prior to this
  # Drupal recipe.
  - editorial_ui_for_publishers
suggests:
  # Other Drupal recipes that once this Drupal recipe has been
  # applied should be suggested to the user.
  - moderated_content_for_publishers


composer:
  require:
    # An array of things to pass to composer require.
    - 'drupal/easy_breadcrumb:^2.0.1'

install:
  # An array of modules or themes to install, if they are not already.
  # The system will detect if it is a theme or a module. During the
  # install only simple configuration from the new modules is created.
  # This allows the Drupal recipe control over the configuration.
  - easy_breadcrumb
  - node
  - text

config:
  # A Drupal recipe can have a config directory. All configuration
  # is this directory will be imported after the modules have been
  # installed.

  # Additionally the Drupal recipe can install configuration entities
  # provided by the modules it configures. This allows them to not have
  # to maintain or copy this configuration. Note the examples below are
  # fictitious.
  import:
    easy_breadcrumb:
      - views.view.easy_breadcrumbs
    node:
      - node.type.article

  # Configuration actions may be defined. The structure here should be
  # entity_type.ID.action. Below the user role entity type with an ID of
  # editor is having the permissions added. The permissions key will be
  # mapped to the \Drupal\user\Entity\Role::grantPermission() method.
  actions:
    role:
      editor:
        permissions:
          - 'delete any article content'
          - 'edit any article content'
```

```
content:
  # A Drupal recipe can have a content directory. All content in this
  # directory will be created after the configuration is installed.
```

This example file is being maintained at
https://gist.github.com/alexpott/2accc39d1da28a5049f977d2e864a588

Drupal recipe creation

Initial implementation will be by hand. One of the goals of the initiative will be to build a CLI command to create Drupal recipes from an already configured site.

Other ideas include:

- Is there UI tooling that we can create? For example, perhaps the site builder could click "Start recording", then perform a bunch of UI actions in Drupal, then click "Stop recording", and then Drupal exports all of the needed YAML files?
- Can we leverage the Dependency Calculation module and use it to build Drupal recipes for specific configuration or content entities.
- Features module has a pretty good UI for selecting configuration and calculating dependencies.

Drupal recipe maintenance

One of the aims is to make Drupal recipes as easy to maintain as possible. The biggest issue that Drupal recipes will have to deal with is what happens when a module that provides the code for a configuration entity or content entity makes a change that would result in an update to the Drupal recipe. When the Drupal recipe is applied to a new site composer will get the latest versions of the dependencies and therefore any updates that this modules ship will not be run against configuration or content provided by the Drupal recipe.

- Can we do something similar to Symfony's recipe update functionality? See https://symfony.com/blog/fast-smart-flex-recipe-upgrades-with-recipes-update or maybe we can do something similar to https://www.drupal.org/project/drupal/issues/3206226?
- How to deal with versioning? Should a Drupal recipe store the versions of modules used to create it? Can we leverage this information to provide automatically generated updates to Drupal recipes or some CLI command to make it easy?

Implementation tasks

- A Drupal recipe is a new project type on Drupal.org
- Decide how we create the package: Composer, symfony flex recipe, our own custom thing? No info.yml. Any info.yml like info is stored in the composer.json or a recipe file. Drupal recipes should not be Drupal extensions themselves.

    - Composer: using composer is problematic because it will add the Drupal recipe as a dependency to the project. This defeats the aim to be stateless and ephemeral. On the other hand this might simplify discovery and downloading. However we would also have to take steps to ensure that changing the autoloader and supplying dependencies is prevented. Composer feels like a tool that needs to be used by Drupal recipes but Drupal recipes need to be outside what we currently think of as a composer package.
    - Symfony flex: seems tied to composer require and doesn't quite work the way we'd need. Here are some reasons:
        - Recipes are tied to specific symfony bundles.
        - We need to interact with the Drupal site - ie. install modules, create content, change configuration.
        - Drupal configuration is not the same as Symfony configuration.
    - Potential solution: our own yaml file and code in core to manage it. We might use a composer plugin to help us manage getting Drupal recipes from Drupal.org. See example file.
- New screen that lists available Drupal recipes
    - Local Drupal recipes should be listed
    - Once Drupal recipes are supported by the project browser, then the project browser should be used on this screen.
- Prior to applying a Drupal recipe there is a validation step that ensures that composer require can be resolved.
    - Proposal: use Package Manager's API from the automated update initiative.
- New installer that leverages the Update initiative work to run a series for composer require against the root composer.json to get the drupal projects listed by the Drupal recipe.
- Prior to installing the modules there is a check that they exist and their requirements are met.
    - Proposal: use Package Manager's API from the automated update initiative.
- New installer that leverages the module install to install the list of modules from the Drupal recipe
    - During install we only install the simple configuration the module provides. We do not create any configuration entities or optional configuration by default. Only config entities explicitly declared in the "config:" section of the recipe file are installed.
- Prior to creating or modifying configuration there is check that the configuration is valid (its dependencies can be met).
    - This needs to happen after module install because config validation can only occur when modules that provide the schema and plugins are installed.
- Configuration is always installed.
    - Should we support the ability to choose specific config? For example, it might be nice to support tours by allowing a Drupal recipe to ship with tours and then if the tour module is not installed by the Drupal recipe and is not already installed somehow ask the user if they'd like to install tour as part of applying the Drupal recipe.
- The installer can create content provided by the Drupal recipe. The installer asks if you want to also install the provided content.

    - Prior art: Default content module, Open Social custom code, Umami CSV content creator
- If content is created when a Drupal recipe is installed there needs to be a way to remove it. As part of applying a Drupal recipe we need to record the content created in state so we can offer the ability to remove it.
- The Drupal recipe is able to override configuration provided by the modules. It is also able to pick out specific configurations from the modules it requires. It does not need to provide all the configuration in its config folder.
- Any optional functionality has to be provided in a module
- Convert core install profiles (minimal, standard, and Umami) to Drupal recipes.
- Work out what to do with the testing profile - seems like that should be a Drupal recipe too.
- Determine how Drupal recipes listed by the project browser will be curated.
    - Licensing requirements to be listed?
    - Can curation encourage collaboration over competition?
    - What about Drupal recipes that configure functionality to connect with 3rd parties, for example a subscription component?

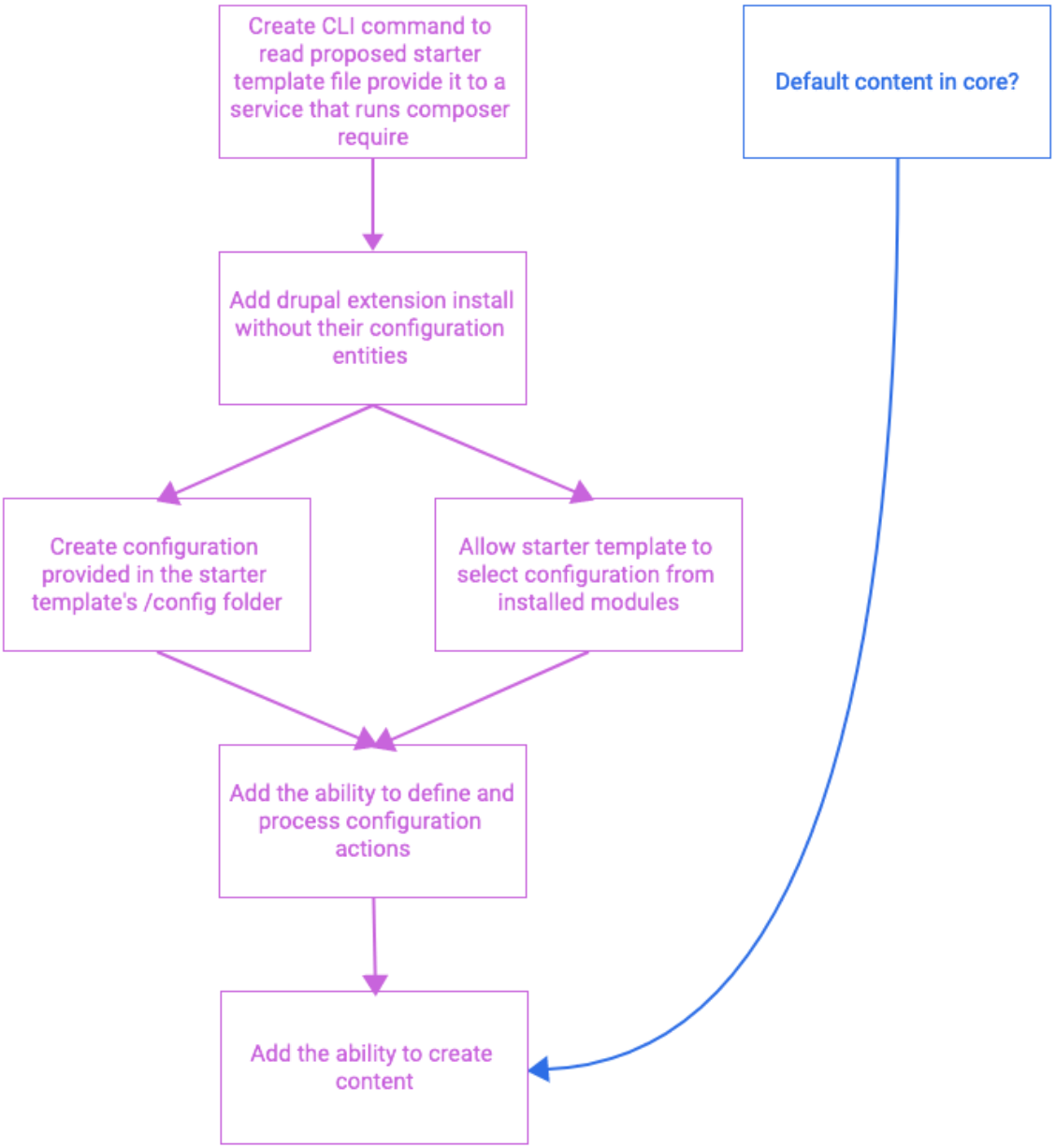
Things to discuss:

- Do Drupal recipes have an installation status? The current opinion is that we should avoid using the word install with respect to a Drupal recipe. A Drupal recipe is something that can be applied against a site.
- Should we have a log of what Drupal recipes have been applied to a site. Potential uses include:
    - Being able to use the applied Drupal recipe suggestions to recommend further steps to take.
    - Being able to list Drupal recipes that can be reverted.
    - How is this different from an installation status?
    - Do we need to borrow the idea of the symfony.lock file from Symfony Flex?
- What happens if a module from a Drupal recipe is already installed on the Drupal site?
- What happens if configuration with the same name already exists.
- How are Drupal recipes obtained? Via packagist/drupal.org if composer, a custom recipe server for Flex.
- Can Drupal recipes be sold on markets outside of Drupal.org (similar to the WP ecosystem)?
    - Proposal: Not a question for this initiative.
- Is the composer project template idea just a special form of Drupal recipe?
- How can a Drupal recipe be reverted? I.e. I should be able to apply a Drupal recipe, decide I don't like it and want to try another one, and revert back to my previous state (config if not composer) to try the next one. At least before I've created content or configuration was changed. Potential idea: use configuration snapshots. Take a snapshot of configuration prior to applying a Drupal recipe and a store a hash or another snapshot of the configuration after applying. If the current configuration state matches the stored hash then a Drupal recipe revert will involve removing any content created and then importing the previous config snapshot.
- What happens when you re-apply a Drupal recipe that has been applied before?

- What happens when a Drupal recipe fails halfway through application?
- Should we have a post-install message feature? (Borrowing an idea from Symfony flex) How to make them look good in UI and CLI?
- Namespacing? How do we avoid a plethora of Drupal recipes all with very very similar names. We need to avoid drupal-recipe/moderated-article, drupal-recipe/better-moderated-article, drupal-recipe/moderated-article-tng
- Where are local Drupal recipes stored? How do we keep them out of the extension discovery complexities? Ideally this would live outside of settings and services as these are per site.
    - use $ENV_VAR/recipes and core/recipes. Block access to these directories using .htaccess by default. Local discovery works using directory traversal and only looks for Drupal recipe yamls one directory down as Drupal recipes can not contain other Drupal recipes. The default if $ENV_VAR is not defined is the root directory of the project.
- Can Drupal recipes reference themes/modules outside of Drupal.org?
    - As long as the module or theme is present on the file system by the time we come to module install it will work. The composer step will be able to require modules from anywhere composer can. Note this hints at the possibility to extend the composer section with the ability to add repositories.
- The project browser contents meet certain criteria, what criteria will determine if a Drupal recipe will be available?
    - If we're not using composer, how can we determine simply that a Drupal recipe can be applied to a site? Or to put this another way around do we really want to re-invent composer when it comes to getting a list of applicable recipes based on core version, php version and other dependency versions?
- What best practices and instructions should be created to assist people creating and maintaining Site Recipes?

Drupal recipe roadmap

Phase 1



Link to roadmap: https://draft.io/8vffzmqwf4gxn8hcqq4de4qg7ce6p2tmfexfx2y2u4tw

Phase 2

TBD, includes:

- Building UI for applying

- Hosting on Drupal.org
- Discovery by some tool
- Drupal recipe testing

Phase 3

TBD, includes:

- Project Browser integration
- Project Browser in the installer
- CLI tool for creation

Phase 4

- Dependabot / Symfony flex type updates

What is a Composer project template?

Adds new abilities to composer create-project for Drupal distributions to leverage in order to completely replace some of the special functionality of install profiles / starter kits / distributions. Composer project templates are envisaged to be used by:

- Drupal agencies to maintain to a common starting point for their builds
- Distributions to provide the same functionality they can now with install profiles

What can a composer project template do

- Provide a composer.json that list projects for composer install (Existing composer functionality)
- Can provide a composer.lock file to ensure that a version gets exactly the right dependencies and is not open ended. (Existing composer functionality)
- Provide a list of Drupal recipes to download and apply during a Drupal installation
- Potential functionality. These functionality cross-over with Drupal recipes so potentially if we don't build this then we'll get more Drupal recipes.
    - Provide a list of modules to install during a Drupal installation
    - Provide a list of themes to install during a Drupal installation
- Determine which theme is used during the installer
- Determine whether the english language is to be kept after a multilingual install
- Provide a finish URL

Things to discuss

- What to do about hook_install_tasks() and hook_install_tasks_alter(). These have been used to add additional forms to the installer to collect information from the user or to select optional functionality. These are special hooks that can only be invoked for an install profile - if there is no install profile then we don;t have a place for this logic. Potential solutions include:
    - replacing the select optional functionality with Drupal recipe suggestions.

    - Having some sort of sort for forms described in YAML or something like that (very long term goal)
    - Allowing the composer project template to continue to load a .profile file somehow.

Updates

If the provider of the project template wants to provide updates then they need to require a module to take care of the updates. This is the same as Drupal recipes. For example, for Thunder the module would depend on the Update Helper module and provide the updates currently provided by the distribution in its config/update directory and the thunder.install file.

Configuration overrides

Install profiles are able to override existing configuration during an install. It is proposed that this functionality is deprecated in favour of Drupal recipes and their ability to modify existing configuration on application.

Optional modules

Install profiles use hook_install_tasks() to add a form to allow users to select additional modules. This should be deprecated and instead this should leverage the ability of Drupal recipes to suggest further Drupal recipes.

Profile modules

Install profiles can provide modules. For example, Thunder has a modules directory that provides modules like thunder_article and thunder_demo. Composer templates can not do this and this functionality should be deprecated.

What happens to install profiles/distributions/starter kits?

Once we have Drupal recipes and install profiles can use the root composer.json to access the same functionality as they can in their info.yml file we should be able to deprecate install profiles.

If an install profile wants to provide an upgrade path they will need to create a module to provide the updates. A separate part of this initiative hopes to add Update Helper functionality to core to make these updates easier to maintain.

Implementation tasks

- https://www.drupal.org/project/drupal/issues/1170362 Provide a way to uninstall an install profile.
- Provide documentation how to convert a distribution to a composer project template and Drupal recipe(s).

- Deprecate install profiles so they can be removed in the next major version of Drupal core.

Prior art

The Drupal recipe and composer project template ideas will be built on the learning for many efforts at improving the how we ship and share related functionality. It's important to note these efforts and try to learn from them.

Config Kits, Kit projects, Harmonize group; attempt at creating a specification for interoperable configuration.

Config selector; allows a module to ship different sets of configuration that can be chosen by the user depending on what extensions are installed.

The Drutopia initiative; especially Nedjo's series of blog posts and https://www.drupal.org/project/drupal/issues/2960941

Features provides a UI and API for taking different site building components from modules with exportables and bundling them together in a single feature module.

Update helper and Config sync; Modules to make updating configuration provided by distributions easier.

Apps project, for bundling Modules and Features. Never really took off.