DKAN Starter Documentation

Release 1.12.10

DKAN Starter

Nov 28, 2017

Contents

Note: These docs are being migrated from our internal documentation system and are currently in beta. We will be updating them throughout the next several weeks.

DKAN Starter is a tool for implementing DKAN. DKAN Starter solves the following issues:

- Staying up-to-date with DKAN
- Adding contributed (3rd party DKAN or Drupal) modules
- Adding custom modules
- Configuring DKAN
- DKAN, Drupal Core, and Drupal modules security updates
- Testing (using Behat and CircleCI)
- Quality Assurance (using Probo.CI)
- Local development (docker)
- Deploying DKAN (defaults to Acquia)
- Using DKAN Best Practices

DKAN Starter is designed as an upstream that integrates with DKAN updates. DKAN implementers can get immediate updates when new minor or patch release is published by DKAN.

DKAN Starter is meant for developers or sysadmins who are tasked with creating a DKAN site. This documentation assumes knowledge of Drupal, LAMP stack administration, and devops tools.

CHAPTER 1

Table of Contents

1.1 Introduction

1.1.1 DKAN Starter Overview

DKAN Starter provides two main functions:

1. Tools for an Agile development workflow
2. Tools for updating DKAN, Drupal, and Contributed Drupal modules and 3rd party libraries

Agile Workflow

DKAN Starter is part of an agile workflow for creating DKAN sites. That workflow walks through the following stages:

1. Feature Request
    - A feature request is created from a stakeholder
2. User Stories
    - The feature is translated into user stories
3. Feature Implementation
    - Behat tests created with the user stories
    - Feature branch created to work on the feature
    - Developers use their local development environment to work on the feature
    - A pull request is created when the feature is ready for review
4. Quality Assurance
    - Passing tests are verified on the pull request

- Project Owner, Project Manager, or stakeholder verifies the feature is complete

DKAN Starter provides integrations with the following tools to facilitate this workflow:

- **Github** for VCS and project management
- **CircleCI** for tests
- **Ahoy** for simplifying our CLI commands
- **Probo** for quality assurance
- **Docker** for local development
- **Acquia** for hosting and deployment

All of these tools can be switched out by updating the ahoy commands that implement each part of the workflow.

DKAN Updates

DKAN Starter serves as an upstream for client projects. DKAN Starter is updated with a new release as soon as a new version of DKAN is released. DKAN releases updates frequently as patch releases for security updates and bug fixes.

Without proper management updating a Drupal or DKAN site quickly can be very difficult. DKAN Starter provides a single command to create an updated branch when a DKAN version is released and tests and quality assurance sites to quickly verify updates are successful.

Config/

All of the development for DKAN Starter implementers is done in the `config/` folder. Everything outside of the `config/` folder will be overwritten after every update. Granicus provides updates to DKAN Starter immediately after DKAN patch or minor releases or after an update to DKAN Starter. Implementers can simply run the command `ahoy build update DKAN-STARTER-VERSION` to get the latest version of DKAN and DKAN Starter.

1.1.2 DKAN Starter Annotated

DKAN Starter Structure

The following is the root structure for DKAN Starter:

```
.ahoy/                          [Ahoy files. See note below.]
CHANGELOG.md                    [Changelog for DKAN Starter. All releases include an␣
↪entry.]
README.md                       [Description of DKAN Starter.]
build-dkan.make                 [Drupal Make file for DKAN version.]
circle.yml                      [CircleCI test file.]
docroot/                        [Built docroot. See note below.]
drupal-org-core.make            [Drupal Make file for Drupal version.]
OWNERS.md                       [Owners file for DKAN Starter.]
assets/                         [DKAN Starter assets. See note below.]
build.make                      [Drupal make file for DKAN Starter modules. These are␣
↪useful modules not included in DKAN.]
config/                         [The only place you should touch. See note below.]
dkan/                           [Fully-built DKAN profile directory. This is simlinked␣
↪from the ``docroot/profiles/dkan`` folder.]
docs/                           [Documentation folder.]
```

```
hooks/                          [Deployment hooks for changing environments (ie dev to
↪test, test to production)]
tests/                          [Tests]
```

docroot/

This is the docroot that the webserver on your environments will point to. The docroot is never edited directly and is always rebuilt from a combination of sources. Everything except for:

- `sites/all/modules/custom`
- `sites/all/themes/custom`
- `sites/all/modules/overrides`

come from the upstream version of DKAN Starter.

The docroot is rebuilt by running *ahoy build update* command.

.ahoy/

This contains our ahoy scripts. All parts of the DKAN Starter workflow run through the ahoy commands.

assets/

This folder includes DKAN Starter modules, patches, and other miscellaneous assets.

config/

In an effort to simplify how we configure and customize projects we have added *./config* to capture all configurations and customizations to a dkan_starter project. The idea with this folder is to capture all customizations across our sites in one place as well as to separate the logic that uses custom configuration from the configuration itself. The current structure of the config folder is:

```
config/
  aliases.local.php
  config.php
  config.yml
  custom.ahoy.yml
  custom.make
  custom_libs.make
  overrides.make
  modules/
      custom/
          README.md
          custom_config/
  patches/
    README.md
  settings.custom.php
  tests/
     features/
        general.feature
```

Most of the files in the above structure are self explanatory, and as you can see in most cases we have simply moved existing files from the legacy structure to this new folder. Moving these files here satisfies the first condition of this change: to capture all parts that are customizable in one place. The new files that have not been seen in previous setups are the config.yml, config.php, and custom.settings.php.

config.php

You should forget about it. This file is created automatically by running *ahoy build config* and it is derived in part by what you add to *./config/config.yml*.

config.yml

This is where we will now keep all of the site specific configurations. This file is a yaml formatted file that will not contain any logic (by definition) and thus simplifies understanding how sites differ from each other.

settings.custom.php

This is where settings.php logic that is custom to a site will live, so that it will be much easier to see how, if at all, a site's settings logic is different from another. Currently we use the devinci along with the environment module to automatically run changes between environments. Often there are site specific differences that happen between the different installations. This is where we can capture these logic difference. Note, that we may move away from how we run deployments (a la devinci style) so this file may become unnecessary.

custom.make

This is where contributed modules are added. Contributed modules are defined as modules that live outside of your project. This make file gets rebuilt when your site is updated.

aliases.local.php

Houses local aliases for your project. This file is set by running `ahoy utils name`.

custom.ahoy.yml

Custom ahoy commands are added here.

custom_libs.make

3rd-party libraries are added to this make file.

`modules/custom/`

Custom modules built for this project are added here. Any modules added to this folder are added to `docroot/sites/all/modules/custom` through a symlink.

modules/custom/custom_config/

This module is for custom configuration of DKAN. Implementers can use this module or another custom module for customizations.

custom_config.features.features_master.inc

The `modules/custom/custom_config/custom_config.features.features_master.inc` file contains a list of the modules that you want enabled on your site.

1.1.3 Releases

Releases for DKAN Starter are posted on the Github release page

Please read DKAN Starter changelog for keeping up to date as well as the release notes.

We've created a standard for naming DKAN Starter releases:

```
DKAN version -> Data Starter version -> Client Site versioning -> Live Version the␣
↪client is running
```

So if we've got:

```
1.12.2.4
```

That means:

- *1.12.2* -> dkan's patch release dkan_starter is running
- *.4* -> we've released 4 incremental updates over dkan_starter since we've rebuild dkan_starter using 1.12.2

Every time we rebuild dkan_starter using a new version we'll reset the counter to 0. So:

- 1.13.0

Means we just rebuilt dkan_starter with the `7.x-1.13` tag.

1.1.4 Getting Help

Please file tickets if you have any questions or issues in the DKAN repository . We will respond as soon as possible.

1.2 Getting Started

1.2.1 Create a Project Plan

Before you *Create a new project* read the following and either sign up for the tools you can use and / or create a plan to replace the default tools you can't.

There are many reasons why teams can't use the tools below and we have worked with many of them that have had successful workflows and deployments.

Determine Which Tools You Can Use

DKAN Starter is optimized to work with the following:

- Github
- Docker
- AWS S3
- ProboCI
- CircleCI
- Acquia
- Jenkins

You will be best supported if you can use these tools. If you can't use one or more of these tools make sure to plan out how to replace them at the beginning of your project.

Create Accounts with Tools You Can Use

Create accounts with the tools that you can use in the list above. Docker is the only tool you don't have to create an account for.

See *Creating a New Project* for details about each of the services above.

Create A Project Plan for Tools You Can't Use

If you can't use any of the tools listed above it is imperative that you create a plan for a workaround.

For example if you can't use Github but are authorized to use BitBucket you need to plan to replace the Github and ProboCI and CircleCI integrations. If you cannot use Acquia you need to have a host that has multiple environments and supports Drush aliases. If you can't use a production server with drush aliases you need a way to allow the backup system to still work correctly.

Github

If you can't use Github you can still use the rest of the tools above. You will need to fire events to Probo and CircleCI for your QA and testing and likely need to change some other items in this toolchain.

Probo.CI

If you can't use ProboCI you can create other mechanisms for creating Quality Assurance sites. We've been successful doing this in the past by wiring a webook from Github to Jenkins and using our build tools on a bespoke server.

CirlceCI

If you can't use CircleCI you need a test-runner for verifying that current DKAN tests don't break as you build and to test new functionality. If you have to work within your own data center you can install CirlceCI. CircleCI can also be installed quite easily within an Amazon VPC if you sign up for Enterprise.

Docker

DKAN has its own fleet of docker containers our developers work with on a daily basis that are optimized for DKAN.

These docker containers were developed with help from HelioCore.

If you can't install Docker and Docker Machine on your local machines you need a way to allow developers to work on individual instances of the website. This can be done with other tools like Vagrant or by setting up a cloud VPS. The solution needs to accommodate testing developer branches. This needs to be in place before development starts on the project.

Jenkins

If you are unable to use Jenkins it is recommended to use a similar task runner. We don't recommend using cron jobs as they don't report errors unless configured specifically to do so.

AWS S3

If you are unable to use Amazon's S3 for backups you need to setup a place to keep backups that is secure and developers and CI tools can access.

Acquia

If you are unable to use Acquia there are many hosts that can be substituted or a bespoke hosting platform can be setup, however, it needs to have the following characteristics:

- Git integration
- Drush alias support
- Dev, Stage, and Production environments
- Triggers that fire when code changes in an environment
- Ability to deploy code, database, and file changes between environments
- Ability to withstand expected traffic results for DKAN and client customizations
- Ability to load test the production or replica of the production environment

Do not start development without a plan to replace these tools if you can't use them. You will create a very expensive level of technical debt very quickly if you aren't writing tests or using DKAN Starter as intended.

Project Plan Checklist

You can use this checklist as a tool for preparing your Project Plan. Make a copy for your own project.

Contact Granicus

We would love to hear from you if you are using DKAN Starter. Contact us through the DKAN channels. Let us know your use case, questions or concerns.

1.2.2 Creating a New Project

Github

To create a new project simply copy the DKAN Starter github repository and create a new project in Github.

ProboCI

Sign up

Sign up for a ProboCI account. Enable the project you just created in Github by clicking "Active Repos" in ProboCI and selecting your poject.

Add Assets

Add the S3Curl script as an asset. To do so click "Build Assets" in a project and upload the file. See:

CircleCI

Sign up

Sign up for a CircleCI account. Enable the project you just created in Github by clicking "Add Projects" and following the instructions.

Project Settings

Configure the following settings in the CirlceCI project settings section.

Build Environment

Select `Ubuntu 14.04 (Trusty)` as the Ubuntu version.

Adjust Parallelism

We recommend at least 3x.

Environmental Variables

Add an AWS Key and ID that will allow you to retrieve S3 files from you AWS account. If you have a different solution for storing and retrieving backups.

AWS

Sign up for an Amazon Web Services account. Create a user that has access to access private S3 buckets.

Update your repository to include the AWS S3 url in the `config/config.yml` file.

Jenkins

We use Jenkins for our automated tasks including backups. We employ Jenkins jobs for the following:

- Backups to S3
- Acquia purge job
- Cron runs
- Data.json caching
- Datastore queue

We will share the actual jobs for this soon.

At the least you need a mechanism for providing pruned and sanitized backups for your site to S3 or another mechanism to do so.

You can simply setup the `ahoy utils asset-download-db` job to download directly from your production instance for local development, ProboCI, and CirlceCI; however, we recommend using a pruned and sanitized version of your database for these services.

Custom Site Configuration

There are a number of configuration steps that are captured in the `config/` folder.

For details see Custom Site Configuration.

Docker

See the Docker for details on setting up and using Docker.

Once you have setup the rest of these services, developers simply need to run:

```
$ docker-machine start default; eval "$(docker-machine env default)"
```

This sets starts docker.

```
$ ahoy docker up
```

Running this inside your project docroot starts the docker containers for your project.

```
$ ahoy site up
```

Running this inside your project docroot:

- Adds `docroot/sites/default/setting.docker.php` which adds settings to connect your DKAN site to your docker containers
- Downloads a local copy of your database in the `backups/` folder
- Downloads a local copy of your files if setup in `config/config.yml`
- Runs an "Environment switch" and sets your environment to "local"
- Runs other deployment tasks as defined in `hooks/commands` and `config/settings.custom.php`

1.2.3 Custom Site Configuration

These are steps to setup the configuration for your site.

Adding Site Name

The site name needs to be set in order for the docker local environment to talk to the site correctly as well as for backups and testing to be able to identify the backup location.

The site name is captured in `config/aliases.local.php`.

You can manually add the site name there or use:

```
$ ahoy utils name
```

Update `config/config.yml`

The following customizations (other than Acquia environment) are currently available in `config/config.yml`. We expect to add more configuration settings here including installed modules, variables, and other settings.

```
default:
  hostname: localhost                    [We recommend keeping this.]
  https_everywhere: FALSE                [Whether all traffic is redirected to https.␣
↪Recommended.]
  https_securepages: FALSE               [Whether to use securepages module for mixed␣
↪https. Not recommended.]
  clamav:
    enable: FALSE                        [Whether to enable clamav module. Requires␣
↪clamav support from host.]
  dkan_workflow:
    enable: FALSE                        [Whether to enable dkan_workflow.module.]
  stage_file_proxy_origin: changeme      [Whether to use state file proxy module for␣
↪files. Recommended as set to staging or production environment.]
  fast_file:
    enable: TRUE                         [Whether to use DKAN's fast file import for␣
↪the Datastore.]
    limit: 10MB
    queue: 50MB
private:
  aws:
    scrubbed_data_url: CHANGE ME         [Private S3 bucket for backups.]
  probo:
    password: CHANGE ME                  [Password for user 1 on ProboCI sites. This␣
↪is changed from production for security reasons.]
gaClientTrackingCode: UA-XXXXX-Y         [Google Analytics tracking code for the site.]
circle:
  test_dirs:                             [List of directories with features tests that␣
↪needs to be run on CircleCI. Default to tests/features, dkan/tests/features and␣
↪config/tests/features.]
    - tests/features
    - dkan/test/features
    - config/tests/features
  skip_tags:                             [List of tags that will be skipped when␣
↪running tests. Defaults to customizable, fixme and testBug.]
    - customizable
    - fixme
```

```
    - testBug
  memory_limit: 256M                    [Memory limit for CircleCI.]
```

Deciding Which Modules are Enabled

Only modules that are part of the list at `assets/modules/data_config/data_config.module:data_config_enabled_modules()` or added to `config/modules/custom/custom_config/custom_config.features.features_master.inc:custom_config_features_master_defaults()` will be enabled any time you switch environments.

If you want to add modules, add them to the list here in the `custom_config_features_master_defaults()` function.

There are also modules that are enabled or disabled on certain environments. These are defined in `assets/sites/default/settings.php` in the

```
switch(ENVIRONMENT) {
}
```

statement. They are listed per environment. You can see an example for the `local` case:

```
$conf['features_master_temp_enabled_modules'] = array(
  'dblog',
  'devel',
  'maillog',
  'views_ui',
  'clamav',
);
```

We've added the `features_master_temp_enabled_modules` feature so that some modules can be turned on locally or on test environments. The `maillog` module keeps emails from being sent to clients or users. Anyways, remember that if you want to add a new module to that list, you should be doing it in settings.custom.php.

Adding Custom and Contributed Modules, Libraries and Themes

"Custom" modules are those that are not publicly available and are associated only with your site.

"Contributed" modules are those that are housed out of the site repository.

See:

- Adding a custom module
- Adding a contributed module
- Adding a contributed library
- Adding Custom Behat tests

1.3 Docker Development Environment

This details how to update the docker images for DKAN. Granicus owns the containers but the same steps can be made to create your own images. We will also accept timely PRs with an updated container.

1.3.1 Step-by-step guide

Steps to pull, change, and commit to the docker web image: Please substitute the following in the instructions with the proper details.

```
Container: web
Project: datastarterprivate
Team/Owner: nuams
Image: drupal-apache-php
```

Image Tag/Version to commit: 1.0-5.6 (increment this with every new commit)

1. Create a Docker Hub account at https://hub.docker.com
2. Ask for permissions to access and commit to the nuams team https://hub.docker.com/u/nuams/ . (Create a ticket and contact Pluto to get help with this)
3. docker-image start default
4. git clone nucivic/docker-drupal-apache-php
5. cd docker-drupal-apache-php
6. edit Dockerfile
7. run docker build -t nucivic/drupal-apache-php:v<x>.
8. git clone nucivic/dkan_starter
9. cd dkan_starter
10. edit dkan/.ahoy/docker-compose.yml with new image location
11. ahoy docker up
12. ahoy docker exec web bash
13. Verify things look right (iterate if not)
14. return to docker-drupal-apache-php commit your work and create a PR.
15. Once PR gets reviewed and merged go to docker hub and create a new tag
16. check your new image tag here: https://hub.docker.com/r/nuams/drupal-apache-php/tags/

1.3.2 Setting up Your Local Docker

The steps to setup the DKAN development environment are as following:

1. Run the universal Docker installer (recommended) and/or run the commands manually to make sure that the Docker Machine is properly setup.
2. Make sure the Docker Machine is started
3. Clone DKAN, install it in Docker Machine and start developing. Or clone a dkan_starter project, install it in Docker Machine and start developing.
4. Universal Docker Installer

1. Universal Docker Installer

The universal Docker installer is available in the Universal Installer repository. Check the Readme on the Github repository to setup the docker development environment.

1.1 Setup the installer dependencies

https://github.com/NuCivic/universal-docker-installer#installer-dependencies

1.2 Environment setup

https://github.com/NuCivic/universal-docker-installer#usage

1.3 Troubleshooting

https://github.com/NuCivic/universal-docker-installer#troubleshooting

2. Docker machine

Before starting to work on DKAN sites, the developer needs to make sure that the development docker machine is up and running as following:

```
docker-machine status default
docker-machine start default
```

When the developer is done developing DKAN, the docker machine could be checked and stopped as following:

```
docker-machine status default
docker-machine stop default
```

To manage the default Virtualbox machine (the docker machine), a developer can use the following commands:

1. Check the machine status:

```
docker-machine status default
```

2. List the existing docker machines:

```
docker-machine ls
```

3. Start the default machine:

```
docker-machine start default
```

4. Stop the default machine

```
docker-machine stop default
```

5. Check the default machine IP address

```
docker-machine ip default
```

6. SSH into the default machine:

```
docker-machine ssh default
```

For a detailed docker-machine command line reference check the following link: https://docs.docker.com/machine/reference/

3. Getting started with DKAN development

To get started with DKAN development you need to follow these steps:

```
cd ~/docker
git clone git@github.com:GetDKAN/dkan.git
cd dkan
bash dkan-init.sh dkan
ahoy docker up
ahoy dkan drupal-rebuild mysql://drupal:123@db/drupal
ahoy dkan remake
ahoy dkan reinstall
ahoy docker url
ahoy docker vnc (to get the vnc url and use it with any vncviewer. The password is␣
↪secret).
ahoy dkan test
```

Visit the DKAN site url to be sure the site is up and reachable.

In the dkan folder you can see the following directories:

1. `dkan/`: The git repository of dkan profile
2. `docroot/`: The docroot with a fresh Drupal installation and a symlink from docroot/profiles/dkan to dkan (git repository)
3. `backups/`: Contains SQL backup/dump file (last_install.sql) for the last DKAN site reinstall.

Make changes to dkan, add, commit and push. When done developing for this project execute the following command: `ahoy docker stop` When done developing with Docker Machine for any DKAN related project execute the following command: `docker-machine stop default`

4. Getting started with DKAN Starter development

See: Setting up a project locally

1.3.3 Ahoy

Ahoy is a small open source tool for creating cli apps like drush, drupal-console, or custom bash scripts. Written in Go and compiled down to a single binary, it has no dependencies which make it fast and easy to install.

See Ahoy's documentation for more details.

Why Ahoy?

We use ahoy to help us do the following:

- Standardize workflow steps so that everyone is doing things exactly the same way
- Abstract more complex workflow steps and scripts behind higher level commands, which:
- Makes it easier for developers, as they can see a list of commands available along with descriptions of each
- Makes it easier for the devops team to improve the processes without having to retrain developers or change documentation
- Reuse abstracted commands in scripts like CircleCI, or even other ahoy commands.
- Make it easier for anyone to add or tweak the commands without knowledge of bash, php, etc

- Allow projects (or users) to have their own custom commands in addition to the defaults.

Available commands

Ahoy is self documenting. Type `ahoy` to see available commands. Using these commands is detailed throughout this documentation.

DKAN and Ahoy

DKAN's ahoy commands are stored in the `.ahoy/` folder in DKAN.

DKAN Starter and Ahoy

DKAN Starter ahoy commands are stored in the `.ahoy/sites/` folder in DKAN Starter.

1.3.4 TODO: enable ahoy v2

Everything below this line is currently not yet available.

Upgrading to ahoy v2

Rational

Now that dkan_starter is open sourced we need a straight forward way of allowing ahoy command overrides for the default setup commands. This need is satisfied by switching to using the ahoy binary for version 2 which has an overriding feature built in. Thus we are updating the version of ahoy we are using from version 1 to version 2.

Expected errors

Unfortunately this upgrade is backwards incompatible with ahoy config files that are for ahoy version 1. If you try to use ahoy with an imcompatible set of configs you will see an error like the following:

```
2016/10/31 13:41:11 AHOY! [fatal] ==> Ahoy only supports API version 'v2', but 'v1'␣
↪given in /Users/dkinzer/docker-share/sites/dkan_starter/.ahoy.yml
panic: AHOY! [fatal] ==> Ahoy only supports API version 'v2', but 'v1' given in /
↪Users/dkinzer/docker-share/sites/dkan_starter/.ahoy.yml
```

Projects that are currently using ahoy version 1 will need to be updated to use ahoy version 2 by upgrading the project to the latest dkan or dkan_starter.

However, if for some reason this is not possible the developer will need to switch ahoy binaries from version 1 to version 2 depending on what config file version the project is using (see the following sections for details)

Updating the binary

- move your current ahoy binary to ahoy1 in the same folder it is (you might need it to switch between ahoy1 and ahoy2 projects in the interim).
- get ahoy2: run

```
if [ -z ${version+x} ]; then
  version=2.0.0-alpha
fi
os=`uname -s | tr '[:upper:]' '[:lower:]'`
wget  https://nucivic-binaries.s3-us-west-1.amazonaws.com/ahoy/$version/ahoy-$os-
↪amd64 -O ./ahoy-$version && \
        chmod +x ./ahoy-$version
```

- move this new copy of ahoy to where the old ahoy binary was.
- symlink ahoy to this binary `ln -s /usr/local/bin/ahoy-2.0.0-alpha /usr/local/bin/ahoy`

Switching between versions

To switch between the two versions of ahoy you need only update the symlink to point to the version of the ahoy binary that you require.

Make sure that you use ahoy1 binary with ahoy1 config files and ahoy2 binaries with ahoy2 config files or you will see the error above.

1.4 Common Development Tasks

These are common development tasks throughout the lifecycle of a project.

1.4.1 Add or Update Modules, Themes or libraries

- *Add custom configuration*
- *Add a contributed library*
- *Add a contributed module*
- *Add a custom module*
- *Build a "Contributed" or "DKAN" Module in a Client Project*
- *Patching or "Overriding" DKAN*
- *Update a contributed module*
- *Update DKAN Starter*
- *Update to Latest DKAN*

1.4.2 Testing & QA

- *Add a Custom Behat Test*
- *Run Client Tests Locally*
- *Override an Upstream Behat Suite Configuration*
- *Getting Content onto a Probo Site*

1.4.3 Deployment

- TODO: update deployment
- *Temporarily disable a module for an environment*

1.4.4 Configuration

- *Configure Secure Pages*
- *Enable Fast File Import*
- *Enable HTTPS everywhere*
- *Enable or Disable a module*
- *Update config.yml settings*

1.4.5 Docker

- *Change local container settings*

1.4.6 Setup

- *Setting up a project locally*

1.4.7 Full Table of Contents

Add custom configuration

By custom configuration we mean:

- A new Content Type
- A new View
- A new Page
- Whatever can be captured in features in a custom module

Whatever's you need to customise the project, you should set the feature export to somewhere inside the **docroot/sites/all/modules/custom** folder. That way, your module will **persist** when the site gets **remade** (Read Add a custom module if you don't understand what this means). Please note your changes shouldn't be added directly to docroot/sites/all/modules/custom but to **config/modules/custom**, otherwise, your changes will be lost on the next DKAN Starter upgrade.

However, there's a few caveats:

1. If you add something custom to the site, you need to make sure it gets tested every time new code gets added to the project (on a Pull Request).

1. It doesn't need to be the fanciest behat test, something like this will do -> tests/features/general.feature
2. Create a behat feature file for your tests (name it appropriately)
3. Look at the `circle.yml` recipe to see if it's setup to run that behat feature

2. **Customisation** does not mean in any way **DKAN overrides.** If you plan to introduce overrides please refer to Override a DKAN out of the box feature

Add a contributed library

Whenever you want to add a library you need to guarantee two specific conditions:

1. The module needs to be added to the **codebase**
2. The module needs to be added to the project's **make file** so if the website gets rebuild (for instance when dkan get's updated), the module remains in the **codebase** and doesn't get deleted

Let's say, for instance, that we want to add the Angular module to the project.

Add the module to the `custom_libs.make` file

Add the folowing line to `custom.make`

```
# Angular
libraries[angular][type] = libraries
libraries[angular][download][type] = git
libraries[angular][download][url] = "https://github.com/angular/angular.git"
libraries[angular][download][tag] = "2.0.0"
```

Add to the project

Run:

```
ahoy build custom-libs
```

That should put the **angular** library at `docroot/sites/all/libraries/angular`.

Add a contributed module

Whenever you want to add a module (no matter if it's custom or not) you need to guarantee two specific conditions:

1. The module needs to be added to the **codebase**
2. The module needs to be added to the project's **make file** so if the website gets rebuild (for instance when dkan get's updated), the module remains in the **codebase** and doesn't get deleted

Let's say, for instance, that we want to add the ShareThis module to the project.

Add the module to the custom.make file

Add the following line to the cusmom.make

```
projects[] = sharethis
```

If you need the project to be at a specific version then you can add this instead:

```
projects[sharethis][version] = 2.12
```

Add the module to custom_config

Add the module to the custom_config.features.features_master.inc file:

```
$features_master = data_config_enabled_modules();
$features_master['modules']['sharethis'] = 'sharethis',
```

Remake the project

Run:

```
ahoy build custom
```

That should put the **sharethis** module at `docroot/sites/all/modules/contrib`.

Add a Custom Behat Test

Sometimes a site will require a feature that will never be part of the upstream product. In these cases you will need to add a site specific behat tests to your site repo. This page documents how to do that.

Step-by-step guide

1. Add any new feature tests to the config/tests/features folder.
2. If a new context that cannot be merged into dkanextension is needed, add it to the config/tests/feature/bootstrap folder (see example in info block below).
3. If a new context was added in above step, edit config/tests/behat.custom.yml and add new context entry to the contexts: field. (see example in the info block below).

Directory structure for custom tests:

```
config/tests/features/
 |-- bootstrap
 |   |-- CustomContext.php
 |   |-- FeatureContext.php
 |-- custom.feature
```

Example Test:

config/tests/features/custom.feature

```
Feature: Custom Example
  @api
  Scenario: See custom about page
    Given I am on the homepage
    When I click "Datasets"
    Then I should see "Content Types"
```

Example *behat.custom.yml*

config/tests/behat.custom.yml

```
# behat.yml
default:
  suites:
    custom:
      paths:
        - %paths.base%/../config/tests/features
      contexts:
        - CustomContext
        - FeatureContext #Temporary overrides only!
        - Drupal\DrupalExtension\Context\MinkContext
        - Drupal\DrupalExtension\Context\DrupalContext
        - Drupal\DrupalExtension\Context\MessageContex
```

Example Custom Context

config/tests/features/bootstrap/CustomContext.php

```
<?php
use Drupal\DrupalExtension\Context\RawDrupalContext;
use Behat\Behat\Context\SnippetAcceptingContext;
use Behat\Gherkin\Node\PyStringNode;
use Behat\Gherkin\Node\TableNode;
/**
 * Defines application features from the specific context.
 */
class CustomContext extends RawDrupalContext implements SnippetAcceptingContext {
  /**
   * Initializes context.
   *
   * Every scenario gets its own context instance.
   * You can also pass arbitrary arguments to the
   * context constructor through behat.yml.
   */
  public function __construct() {
  }
}
```

Add a custom module

Add custom Drupal modules to `config/modules/custom`

Build a "Contributed" or "DKAN" Module in a Client Project

Workflow

The best workflow for this is to:

1. Add the module

   (a) Add the module as if it was just another "contributed" module: Add a contributed module

   (b) Once this is done a link to the module should exist in `custom.make`.

2. Delete the module

   (a) Delete the module in `docroot/sites/all/modules/contrib`

3. Clone the module

   (a) Clone the module from the upstream repo into `docroot/sites/all/modules/contrib`.

4. Commit local changes to the contributed module

   (a) DO NOT commit local changes to the client repo

5. Push to the client repo

   (a) update the link to `custom.make`

      i. When you are ready to reflect that local changes in the client repo update the reference in custom.make

   (b) run `ahoy build custom` 1. Changes should then appear in the module in `docroot/sites/all/modules/contrib`

Example

Lets say you are working on a module which adds hover-bear functionality to a client project and the module lives at http://github.com/GetDKAN/dkan_hover_bears.

1. Add the module

   (a) The **custom.make** file should include:

```
projects[dkan_dkan_hover_bears][type] = module
projects[dkan_dkan_hover_bears][download][type] = git
projects[dkan_dkan_hover_bears][download][url] = https://github.com/GetDKAN/dkan_
↪dkan_hover_bears.git
projects[dkan_dkan_hover_bears][download][revision] =␣
↪0c57133a4fb8c26cd03ee7607ebd7f983b853b8c
```

   (a) Note the **revision**. You can include branch during development but it is safer to use a commit. Using the a branch look like:

```
projects[dkan_dkan_hover_bears][type] = module
projects[dkan_dkan_hover_bears][download][type] = git
projects[dkan_dkan_hover_bears][download][url] = https://github.com/GetDKAN/dkan_
↪dkan_hover_bears.git
projects[dkan_dkan_hover_bears][download][branch] = civic-12311-hover-bear-stare
```

2. Delete the module

```
rm -r docroot/sites/all/modules/contrib/dkan_hover_bears
```

3. Clone the module

```
git clone git@github.com:GetDKAN/dkan_hover_bears.git
```

4. Commit local changes

   (a) cd into `docroot/sites/all/modules/contrib/dkan_hover_bears`

(b) commit changes

5. Push to the client repo
   (a) MAKE SURE YOU HAVE COMMITTED TO THE MODULE
   (b) Update `custom.make` if you are pointing to a commit. If you are pointing to a branch you shouldn't need to worry about this as your code has already been pushed to the branch
   (c) run `ahoy build custom`
      i. You should see changes in `docroot/sites/all/modules/contrib/dkan_hover_bears`
   (d) Commit changes to `custom.make` (if you are using revision) and `docroot/sites/all/modules/contrib/dkan_hover_bears`

Change local container settings

Update php {{memory_limit}}:

1. Locate the php.ini file used for the ahoy based dkan setup, usually dkan/.ahoy/.docker/etc/php5/php.ini
2. Edit the file to set the memory_limit php variable to the needed value, for example (512M, -1 for unlimited).
3. We need to restart the docker containers. `ahoy docker stop;; ahoy docker up`

Configure Secure Pages

Secure pages is useful as in interim solution until we get "HTTPs Everywhere" support.

To setup secure pages properly:

1. Ensure securepages module is enabled in custom_config
2. Change 'https_securepages' to TRUE in config.yml

This will trigger the secure_pages settings in settings.acquia.php

Enable Fast File Import

To enable fast file import add:

```
fast_file:
  enable: TRUE
  limit: 10MB
  queue: 50MB
```

to config/**config.yml**

Enable HTTPS everywhere

When you want to enable https everywhere on a dkan site.

Step-by-step guide

You'll need to take several steps to enable https everywhere for a site instance on Acquia.

1. To `config/config.yml` add:

```
default:
  https_everywhere: TRUE
```

2. Rebuild the site configuration with ahoy:

```
ahoy build config
```

3. Commit and push changes to Acquia
4. On the Acquia dashboard, enable a varnish proxy SSL (per development environment).

Enable or Disable a module

Modules are automatically enabled or disabled every time there is an environment switch. Environment switches happen any time a database changes between Acquia instances or to Circle or locally.

List of Default Enabled Modules

A list of modules exists in the data_config_enabled_modules() function. This is a list of modules that are always enabled. This list is periodically updated.

List of Custom Enabled Modules

A list of custom enabled modules exists in the custom_config_features_master_defaults() function in /config/modules/custom/custom_config/custom_config.features.features_master.inc.

This function takes the output of data_config_enabled_modules() and allows the final output to be overwritten.

ANY MODULE NOT INCLUDED IN THE $features_master ARRAY WILL BE DISABLED ON PRODUCTION.

Enabling or Disabling a Module on Production

While modules can be manually enabled, they will be turned off anytime new code is deployed.

To add a module to the enabled modules list, add the module to the $features_master list in the custom_config_features_master_defaults() function:

```
<?php
function custom_config_features_master_defaults() {

  module_load_include('module', 'data_config');

  $features_master = data_config_enabled_modules();

  // Disable module.
  unset($features_master['example_module_to_disable']);
```

```
  // Enable module.
  $features_master['example_module_to_enable']);

  return $features_master;
}
```

Temporarily Enabling or Disabling a Module for an Environment

See *Temporarily disable a module for an environment*

Getting Content onto a Probo Site

When you want to have a specific content on a Probo Site.

Step-by-step guide

The easiest way to get content into a Probo Site is to add it to the production site:

1. Add the content you want to the production site.
2. Run the Backup DB to S3 Jenkins Job for the site.

Override an Upstream Behat Suite Configuration

Sometimes site developers may have a need to override an upstream suite configuration as is the case if they need to filter out specific dkan tests for whatever reason. This page describes the steps needed when a developer is faced with such a task

Step-by-step guide

1. Open up `config/tests/behat.custom.yml`
2. If an entry for the suite you want to override is not present, then add it
3. Add filters filters or override what contexts or where contexts being loaded from. The example below shows us adding a name: filter to the dkan suite.

config/tests/behat.custom.yml

```
# behat.yml
default:
  suites:
    custom:
      paths:
        - %paths.base%/../config/tests/features
      contexts:
        - CustomContext
        - FeatureContext #Temporary overrides only!
        - Drupal\DrupalExtension\Context\MinkContext
        - Drupal\DrupalExtension\Context\DrupalContext
        - Drupal\DrupalExtension\Context\MessageContext
    dkan:
```

```
    filters:
      - name: Viewing the site title
```

Patching or "Overriding" DKAN

DKAN overrides

Sometimes you will need to make a patch directly to DKAN or a DKAN profile specific module.

These patches can go in `confing/overrides.make`

Adding a Patch to Override DKAN

Any patches to DKAN or modules supplied by DKAN (any code in the dkan/ folder) should be added to the `config/overrides.make` file as a patch linked by a URL or a local patch contained in `config/assets/patches`.

Patches should be applied to the appropriate git repository, included in a pull request, and applied to your build as outlined below.

To test the application of the patch you can run

```
ahoy build overrides
```

Overrides are applied in the build process when running

```
ahoy build update VERSION
```

See Updating DKAN Starter to Latest Version of DKAN

Here is a step by step process:

1. Create the patch
   (a) Create a PR
   (b) Make sure the PR can be merged into the **release-1-12** (or the release number you are working off of)
   (c) Get a diff of the PR
      i. Go to https://github.com/GetDKAN/MODULE/compare or https://github.com/GetDKAN/dkan/compare
      ii. Select the upstream branch, ie release-1-12, and the PR you want to override with.
      iii. Add **.diff** to the end
2. Add the patch to the `config/overrides.make`

```
---
api: '2'
core: 7.x
projects:
  dkan_datastore:
    subdir: dkan
    download:
      type: git
      url: https://github.com/GetDKAN/dkan_datastore.git
      tag: 7.x-1.12
```

```
    patch:
      1: "https://github.com/GetDKAN/dkan_datastore/compare/release-1-12...fix-
↪filters-pw-1.diff"
```

3. Test patch by running **ahoy build overrides**
   (a) Patched module should appear in `docroot/sites/all/modules/overrides`

Implementation Notes

During **ahoy site remake** there is a step to append whatever is at the end of the overrides.make file to dkan/drupal-org.make file. In effect we are introducing overrides to the make file because anything that is defined last a make file overrides whatever has been defined first.

Such overrides are put in overrides folder within `docroot/sites/all/modules/overrides`.

Run Client Tests Locally

To run client tests locally use **ahoy site test**. The following is an example using a feature file and running an individual step:

```
ahoy site test features/workbench.feature --name=\'As a user, I should/should not␣
↪receive an email on content moderation state change\'
```

Setting up a project locally

1. Install docker as described in this guide
2. Run

```
docker-machine start default; eval "$(docker-machine env default)"
```

if you haven't already

3. cd into `~/docker-share/CLIENT-GIT-REPO`
4. Setup S3 backup access

1. Get S3 credentials
2. **Create aws credentials file**
   (a) `vim ~/.s3curl`
   (b) `chmod 600 ~/.s3curl` # make file so only owner can read and write
   (c) Add:

```
%awsSecretAccessKeys = (
   local => {
     id => 'AWS_ID',
     key => 'AWS_KEY',
   }
 );
```

5. Bring up docker containers:

```
ahoy docker up
```

6. Setup your site

```
ahoy site up
```

7. Get the site URL

```
ahoy docker url
```

Troubleshooting

s3curl

The asset download command which is called from *ahoy ci setup* uses an AWS perl script. This script usually will work without issues, however sometimes you may need to download a missing Perl module. For example:

If you run into the following error:

```
ahoy utils asset-download-db
Can't locate Digest/HMAC_SHA1.pm in @INC (you may need to install the Digest::HMAC_
↪SHA1 module) (@INC contains: /usr/local/Cellar/perl/5.24.0_1/lib/perl5/site_perl/5.
↪24.0/darwin-thread-multi-2level /usr/local/Cellar/perl/5.24.0_1/lib/perl5/site_perl/
↪5.24.0 /usr/local/Cellar/perl/5.24.0_1/lib/perl5/5.24.0/darwin-thread-multi-2level /
↪usr/local/Cellar/perl/5.24.0_1/lib/perl5/5.24.0 /usr/local/lib/perl5/site_perl/5.24.
↪0 .) at nucivic-ahoy/.scripts/s3curl.pl line 20.
BEGIN failed--compilation aborted at nucivic-ahoy/.scripts/s3curl.pl line 20.
```

then, do this:

```
perl -MCPAN -e "install Digest::HMAC_SHA1;"
```

of course the specific module will depend on your error.

Hostname/alias errors

The *ahoy ci setup* command will fail if you do not have the Acquia aliases set up correctly on your local environment. Make sure you are logged into Acquia (drush ac-api-login) then update your Acquia aliases (drush acquia-update).

Temporarily disable a module for an environment

Some modules are temporarily disabled or enabled for different environments.

For example, maillog and devel are enabled for local and dev environments and acquia search modules are disabled for non-Acquia environments.

To temporarily disable or enable a module for an environment, include it in the **$conf['features_master_temp_disabled_modules']** or **$conf['features_master_temp_enabled_modules']** array in `settings.custom.php`.

Update a contributed module

Please read Add a contributed module first.

Edit the module entry in the `build.make` file

Look for the module in the `build.make`. If it looks like this:

```
projects[] = sharethis
```

Then you can just remake since it'll grab the latest stable version for the module. Instead, if it's set to a specific version:

```
projects[sharethis][version] = 2.12
```

Then make the edit to the desired version and then remake

Remake the project

Run:

```
ahoy build remake
```

That should update the **sharethis** module at `docroot/sites/all/modules/contrib`.

Update config.yml settings

`config.yml` file will house a growing number of site settings.

To run updates from config.yml run:

```
ahoy build config
```

Update DKAN Starter

When a new version of DKAN is released, the core dkan_starter repo needs to be updated. The following instructions are primarily for the core DKAN team that maintains the [dkan_starter repo](https://github.com/GetDKAN/dkan_starter), but if you maintain a fork of dkan_starter as an "upstream" to build your own organization's sites from, you may need to do this as well.

1. *git checkout master*
2. *git checkout -b [branch-name]*
3. If upgrading DKAN, update line 11 in **build-dkan.make** with the latest tag from DKAN
4. *ahoy build remake*
5. Commit your changes
6. *git push origin [branch-name]* and create the PR
7. Merge if all tests pass
8. Create new tag for data_starter

Reminders - Adding or removing a module? Make sure to update the $features_master list in *assest/modules/data_config/data_config.module*

Update to Latest DKAN

Site updates always run through DKAN Starter. Don't update DKAN directly.

To upgrade to the latest available version of DKAN, run:

```
ahoy build update DKAN-STARTER-VERSION
```

This command:

- updates everything in the repo except the config/ folder from dkan_starter
- runs custom.make and overrides.make
- runs config.yml customizations

1.5 Tutorials

1.5.1 Installing from a Backup

This is a workflow for installing a site if you only have a git repository and a database dump file. This is often the case if you are requesting a production site or if you receive a backup from a vendor.

This workflow does not include continuous integration (testing, QA sites) because the user does not have access to Github or the database backups.

Prerequisites

The following are required for this tutorial.

1. MySQL database
    - This should be a sanitized version of the production database.
2. Project Repository
    - This should be the full repository and be identical to DKAN Starter except for the `config/` folder.

Step by Step Installation

Install Docker

Follow the instructions at *Local Docker Development Environment* to setup your local development environment.

Start Local Containers

Run `ahoy docker up` from the root of your project folder.

Add database backup

Run `ahoy drush sqlc < DATABASE-BACKUP-NAME.sql`. Change "DATABASE-BACKUP-NAME" for the name of your database file.

Access Site

Run `ahoy drush uli` and you should get a URL for your site.

1.5.2 Add, Test, QA and Deploy a New Module

This document is in progress.

1.6 Troubleshooting FAQ

These are FAQ issues that DKAN Starter or DKAN users may run into while using the docker and ahoy tools.

Please submit tickets to the DKAN Issue Queue.

1.6.1 Frequently Asked Questions

"ENVIRONMENT set to , but not mapped in settings.php"

When trying to connect to local docker machine:

> **Warning:** ENVIRONMENT set to , but not mapped in settings.php

- Your "settings.docker.php" file is not set. Run **ahoy site up**
- Can also be caused by the MySQL container missing:

```
ahoy diagnose all
ahoy docker up
```

Container not running

If you get an error when running `ahoy diagnose all` like:

```
[Error] The "db" container is not running: mysite_db_1          /entrypoint.sh mysqld␣
↪           Exit 1
```

then you can try:

1. Restart the containers `ahoy docker stop; ahoy docker up`
2. Destroy the containers (WARNING: YOU WILL LOSE DATA) `ahoy docker destroy; ahoy site up`
3. Restart the docker maching `docker-machine stop default; docker-machine start default; eval "$(docker-machine env default)"; ahoy site up`

Could not open connection: Curl error

When running tests:

> **Warning:** Could not open connection: Curl error thrown for http POST to http://browser:4444/wd/hub/session

- Your browser container is not running. Run **ahoy diagnose all** to troubelshoot

```
ERROR 1064 (42000) at line 1: You have an error in your SQL syntax; check the manual␣
↪that corresponds to your MySQL server version for the right syntax to use near
↪'env: drush9: No such file or directory' at line 1
[error] The command could not be executed successfully (returned: env: drush9: No␣
↪such file or directory, code: 127)
```

- Make sure you are using drush version 8.0.2 (not 9).

ahoy dkan test... stalls on a step or very slow

- Do a complete cleanup:

```
ahoy docker cleanup
ahoy docker stop
docker-machine stop default
docker-machine start default
ahoy docker up
```

1.7 DKAN Starter Modules

DKAN Starter includes several modules that are not in DKAN.

These modules help the build process for maintaining sites built with DKAN.

1.7.1 Custom Config

The idea for the "Custom Config" module is to have all of the overrides for a DKAN project in a single place. These would not include new features that a project uses but just the configuration changes. This makes it easier to see and evaluate the configurations for a DKAN project because they are in a uniform place.

The "Custom Configuration" module is designed to contain:

- A list of custom modules to enable by default
- Overrides of "Data Config"
- Custom functions that override DKAN
- Features exports that override DKAN
- All the features components that need to be banished through features_banish
- custom updates (hook_update_N)

List of Custom Modules to Enable

The `/config/modules/custom_config/custom_config.features.features_master.inc` file contains a list of all of the modules and themes that should be enabled or disabled by default in a project that are not part of DKAN Starter.

The list of modules that DKAN Starter enables can be found in `/assets/modules/data_config/data_config.module`.

For more information see *Enable or Disable a module*.

1.7.2 Data Config Module

The "Data Configuration" module contains setup and configuration items for DKAN Starter sites.

It contains a list of all modules that should be enabled.

Update functions that need to be run for updates that are not part of DKAN are included here.

It is called "Data" config because this project used to be named "Data Starter".

1.7.3 Devinci Module

The DEVINCI module makes your code context/environment aware so you can tweak configuration for your site to your liking without custom scripting. All you need to do to setup this in your project is to include the following snippet in settings.php:

```
require DRUPAL_ROOT . "/sites/all/modules/contrib/devinci/devinci.environments.inc";
devinci_set_env();
```

This would, out of the box, set an ENVIRONMENT constant and do a number of really clever things every time Drupal is bootstrapped such as detecting if:

- ENVIRONMENT is already set as a drupal constant
- ENVIRONMENT is defined as a linux environment variable
- the current instance is an acquia instance and adding the required `require "/var/www/site-php/$sitegroup/$sitegroup-settings.inc";` setting file with the proper credentials for your subscription (Pantheon is on the roadmap)
- there's a custom environment.settings.php file at DRUPAL_ROOT . '/' . conf_path() to be included.

By default, DEVINCI extends the base environment module definitions (development and production) and adds the local and test environments to the mix. You could also provide your own array of environment mappings to devinci_set_env to contemplate special cases. If you want to properly map acquia environments to the environment + devinci definitions (local, development, test and production) you could do:

```
$env_map = array(
  'local' => 'local',
  'dev' => 'development',
  'test' => 'test',
  'live' => 'production',
  'prod' => 'production',
  'ra' => 'production',
);
devinci_set_env($env_map);
```

this would:

- map acquia's dev to the environment's development
- map acquia's test to devinci's test
- map acquia's live to environment's production
- map acquia's prod to environment's production
- map acquia's ra to environment's production

If the ENVIRONMENT constant is set accordingly by any of the above use-cases then you can:

- Set configuration items based on the environment
- Run code on environment switching

Set configuration items based on the environment

By adding a switch statement to settings.php that analyzes the ENVIRONMENT constant you could set configuration items per environment.

```
<?php
switch(ENVIRONMENT) {
  /**
   * Local Environment
   */
  case 'local':
    // Show ALL errors when working locally.
    $conf['error_level'] = ERROR_REPORTING_DISPLAY_ALL;
    ini_set("display_errors", 1);
      break;
  /**
   * Development Environment
   */
  case 'development':
    // Create new alias and delete old.
    $conf['pathauto_update_action'] = 2;
   break;

  /**
   * Test Environment
   */
  case 'test':
    // Enable caching like in production.
    $conf['page_cache_maximum_age'] = 900;
    $conf['cache'] = 1;
    $conf['preprocess_js'] = 1;
    $conf['preprocess_css'] = 1;
    $conf['pathauto_update_action'] = 1;
   break;

  /**
   * Production Environment
   */
  case 'production':
    // Enable the ability to send emails - via core mail in this case,
    // but it coulbe be update to use SMTP or mail API.
    $conf['mail_system'] = array (
```

```
      'default-system' => 'DefaultMailSystem',
    );
    // Enable caching for production.
    // 15 minutes max page cache time.
    $conf['page_cache_maximum_age'] = 900;
    $conf['cache'] = 1;
    $conf['preprocess_js'] = 1;
    $conf['preprocess_css'] = 1;
    $conf['pathauto_update_action'] = 1;
    // Set google tag container id.
    $conf['google_tag_container_id'] = '';
    break;
```

Run code on environment switching

If you add a devinci_custom_environment_switch implementation of environment's hook_custom_environment_switch to your settings.php then you can specify what needs to run when environment switching happens. A very basic implementation would be: function devinci_custom_environment_switch($target_env, $current_env) {

```
<?php
switch($target_env) {
  case 'local':
    drupal_flush_all_caches();
    features_master_features_revert('custom_config');
    break;

  case 'development':
  case 'test':
  case 'production':
    drupal_flush_all_caches();
    features_master_features_revert('custom_config');
    features_revert_module('custom_permissions');
    break;
}
```

1.7.4 Environment Module

The Environment Module provides a drush command to allow a chain of setup procedures to run when an environment switch happens. This could be whatever you fancy and can do with php within a bootstraped drupal instance. It provides a hook implementation to allow modules to attach custom setup procedures when the command is called from the cli. The hook in question is hook_environment_switch and it usually looks something like this:

```
<?php
 /**
  * Implements hook_environment_switch().
  */
 function YOUR_MODULE_environment_switch($target_env, $current_env) {
   // Declare each optional development-related module
   $devel_modules = array(
     'bulk_export',
     'context_ui',
     'devel',
     'devel_generate',
```

```
    'devel_node_access',
    'imagecache_ui',
    'update',
    'spaces_ui',
    'views_ui',
  );

  switch ($target_env) {
    case 'production':
      module_disable($devel_modules);
      drupal_set_message('Disabled development modules');
      return;
    case 'development':
      module_enable($devel_modules);
      drupal_set_message('Enabled development modules');
      return;
  }
}
```

With the scenario above when you call the drush command from the cli to switch to the so called development environment :

```
drush env-switch development
```

it will enable all the module defined in the $devel_modules variable. On the other hand, when you switch to the production environment:

```
drush env-switch production
```

The opposite will happen (all the devel modules will be disabled). This is of course a very simple implementation of what you can achieve with this module.

For DKAN Starter we are making heavy use of the environment module to allow stuff to happen when we deploy code and copy databases through environments. If you want to fully grasp how we are doing this you need to read the following pieces of documentation:

- *Custom Config*
- *Devinci Module*
- Hands free deployments

1.7.5 Features Banish Module

The Features Banish Module allows you as a developer to be sure that certain features components will NEVER get exported.

There are three options to leverage this:

- Banish the component in your feature's mymodule.info file
- Set the 'features_banish_items' system variable
- Implement hook_features_banish_alter()

Banishing for DKAN Starter Projects can be exported into the custom_config module. Please refer to *Custom Config* to know more about how are me managing this.

1.7.6 Features Master Module

Features Master module handles the turning off and on of modules. The module itself can handle more but we are trying to limit the scope of our use to enabling and disabling modules.

See:

- *Enable or Disable a module*
- *Temporarily disable a module for an environment*

1.7.7 Features Override Module

Features Override Module is used to change (override) existing Features.

- This allows us to keep Features overridden without it looking that way.
- Also a (not very clean) way to deploy overrides from development environments to production.
- Not the prettiest solution, but then again, Features.

It is very important to understand that features overrides should be the very last resource to make things work a certain way. Try the following first:

1. Find a hook alter implementation that lets you modify the configuration item you need to override.
2. If you don't find a hook alter implementation, look again for it.
3. Don't expect google or drupalstackexchange to give you the answer, take a look at the code for the module that implements whatever you are trying to override
4. If you need to set different values for a variable depending on the environment, do it in the drupal settings.php file. There's plenty of examples on how to achieve this in https://github.com/GetDKAN/dkan_starter/blob/master/assets/sites/default/settings.php#L165
5. Take a look on why the item you are trying to override is exported as a feature in the first place and, if possible, propose a way to take it off features and implement a hook alter implementation so devs can tweak it to their liking.
6. If none of the above works for you, then go with features_overrides. But really, why are even you doing this to yourself?

1.7.8 Helper modules

These modules simply provide additional functionality that has proved useful in real-world DKAN deployments, but was not crucial enough to include in the core DKAN package:

- Role Watchdog

1.8 Deployment

1.8.1 Overview

We make a number of changes for deploying sites. We define deployment as a change in code in an environment. We use a number of environments including the standard "dev/stage/prod" but also local, test, and QA envrionments.

When code is deployed to an environment several changes need to happen based on the needs of the environment:

- Modules turn on or off
- Variable settings change
- Features reverts fired

For example the **maillog** module is turned on on all non-prod environments so users don't get emails from dev, stage, local, or CI environments.

1.8.2 Prerequisites

Please review the following pieces of Documentation if you haven't done so yet:

- *Features Master Module*
- *Custom Config*
- *Data Config Module*
- *Devinci Module*
- *Environment Module*

1.8.3 Concept behind "hands free deployments"

Combining Custom Config (Features Master) and DEVINCI (Environment) functionality we can design a deployment strategy where, if you define the conditions properly per environment and you consider those predefined conditions every time you push code, there's no actual need of manual configuration on acquia environments.

Those predefined conditions are:

- Custom Config holds a list of modules that need to be enabled.
- There are complementary modules that could be enabled (for specific or all environments) setting up values for the **features_master_temp_enabled_modules** variable in settings.php file.
- There are complementary modules that could be disabled (for specific or all environments) setting up values for the **features_master_temp_disabled_modules** variable in settings.php file.
- Any enabled module that's not cover by the above three statements will be disabled.
- There are configuration items (variables) set up for specific (or all) environments.

1.8.4 Hooks

For Granicus Data Projects we are implementing deployment hooks for:

- **post-code-deploy**: This runs when code is deployed from one environment from another (i.e elevating code from dev to test, test to prod)
- **post-code-update**: This runs when code is push to the acquia's master branch. This only runs if the dev environment is running the master branch
- **post-db-copy**: This runs when a database is deployed from on environment to another (i.e moving prod db to test and dev)

The implementation is the same for all of them. We run the following set of commands:

```
# Create variables out of script parameters
site=$1
env=$2
# Construct drush alias from site and env
drush_alias=$site'.'$env
# Build the target environment from DEVINCI environment detection
target_env=`drush @$drush_alias php-eval "echo ENVIRONMENT;"`

# Begin the deploy. Rebuild Registry
drush @$drush_alias rr
# Clear drush cache.
drush cc drush
# Force environment switching.
drush @$drush_alias env-switch $target_env --force
# Update the database.
drush @$drush_alias -y updb
```

This very short set of commands is enough for us to be able to guaranteed that code, configuration and modules to be enable/disable for the site could be deployed successfully without manual tinkering and just relying on git.

1.8.5 Deployment

The command that actually does it all is:

```
drush @$drush_alias env-switch $target_env --force
```

Let's examine what happens when the environment switching occur following dkan_starter settings.php file.

1. Drupal is bootstrapped
   (a) DEVINCI environment mapping happens first

```
$env_map = array(
  'local' => 'local',
  'dev' => 'development',
  'test' => 'test',
  'live' => 'production',
  'prod' => 'production',
  'ra' => 'production',
);
devinci_set_env($env_map);
```

   (a) A set of global (not environment specific) configuration is set below the environment mapping. Things like error reporting, the default mail_system, default caching options, zip compression, fast_404, and many settings more.
   (b) Environment specific happens after b) enclosed in a switch statement that analyses the `ENVIRONMENT` constant:

```
<?php
switch(ENVIRONMENT) {
  case 'local':
    $conf['features_master_temp_enabled_modules'] = array(
      'devel',
      'dblog',
      'maillog',
      ...
```

```
    );
    $conf['features_master_temp_disabled_modules'] = array(
      'acquia_purge',
      'syslog',
      'expire',
      ...
    );
    ...
    break;
  case 'dev':
    ...
    break;
  case 'test':
    $conf['error_level'] = ERROR_REPORTING_HIDE;
    ...
    break;
  case 'prod':
    $conf['mail_system'] = array (
      'default-system' => 'DefaultMailSystem',
    );
    $conf['page_cache_maximum_age'] = 900;
    $conf['cache'] = 1;
    $conf['preprocess_js'] = 1;
    $conf['preprocess_css'] = 1;
    ...
    break;
}
```

There are tons of specifics per environment here and we encourage to go deep in the code to find out about them. Having said that, the configuration does follow a pattern:

- **Local** does not need any acquia modules so they are set to be turn off by default
- **Local** and **Dev** are treated as development environments, so we turn on development modules on those.
- **Test** mimics the Prod environment in everything BUT email backend configuration. We simply don't want Test to send emails.
- **Test** and **Prod** are treated as production environments, which means performance is key. We set up caching and do things like adding memcache (if available).
- **Dev**, **Test**, and **Prod** are set to turn on every acquia module we need to make use of search and performance tuning.

2. Env switching happens

   The definition for what happens on environment switching lives in devinci_custom_environment_switch implementation of hook_custom_environment_switch. For dkan_starter we add it at the bottom of settings.php and it looks like something like this:

```
<?php
function devinci_custom_environment_switch($target_env, $current_env) {

  switch($target_env) {

    case 'local':
      drupal_flush_all_caches();
      features_master_features_revert('custom_config');
      break;
```

```
    case 'development':
    case 'test':
    case 'production':
      drupal_flush_all_caches();
      features_master_features_revert('custom_config');
      features_revert_module('dkan_dataset_groups');
      features_revert_module('dkan_dataset_content_types');
      features_revert_module('custom_permissions');
      break;
  }
}
```

This could vary a little from site to site but the important thing is we run two steps for every environment:

- We flush caches with `drupal_flush_all_caches()`
- We features_master_features_revert the custom_config module which holds the list of modules to be enabled.
- We revert modules that we need to be sure they are reverted (i.e modules containing content types).

(a) Cache flushing

Pretty self explanatory, it flushes drupal caches.

(b) Revert custom_config

This does all of the following:

- Enables all the modules declared in custom_config.features_master.inc EXCEPT those specifically set in `$conf['features_master_temp_disabled_modules']` for the `ENVIRONMENT` the system is switching to.
- Enables all the modules specifically set in `$conf['features_master_temp_enabled_modules']` for the `ENVIRONMENT` the system is switching to.
- Disables everything that's not set explicitally to be enabled/disabled for the `ENVIRONMENT` the system is switching to.

(c) Revertion of modules

We revert everything feature related that we are interested in keeping true to the code. The end goal here will be to revert EVERYTHING but at the time of this writing it is not possible. Some rewirring needs to happen on dkan to guarantee that we can do this.

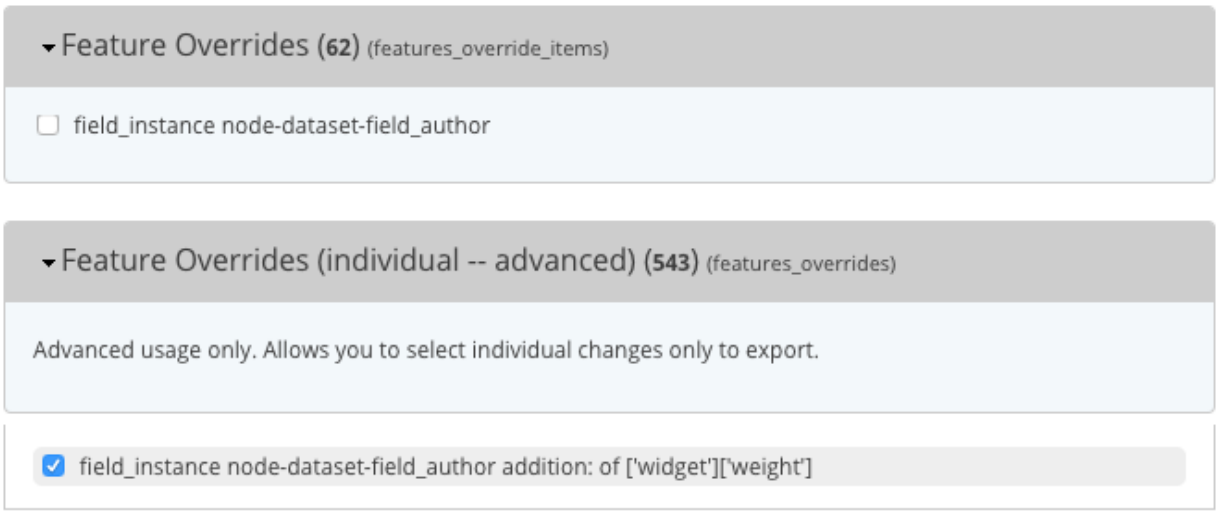
1.9 Customizing DKAN

1.9.1 Overriding custom fields

Features Override

If you absolutly need to override custom fields, use `features_overrides` **BUT**, override individual items and **NOT** the whole field definition.

This is an example of what overriding a field item weight should look like:



DON'T USE THE FIRST ONE. Use the second one. As soon as you start using the first you set the build to a `update-blocking` state.

Items that are **"safe"** to override:

- Field weights
- Display weights
- Max length of a `charfield`
- Cardinality of a field
- Other minor field options

Items that are not a good idea to override:

Allowed Values in a `List (Integer|Float|text).`

This is how a feature_override of a list of allowed values looks like in an info file:

```
features[features_overrides][] = field_base.field_frequency.settings|allowed_values|0
features[features_overrides][] = field_base.field_frequency.settings|allowed_values|1
features[features_overrides][] = field_base.field_frequency.settings|allowed_values|10
features[features_overrides][] = field_base.field_frequency.settings|allowed_values|11
features[features_overrides][] = field_base.field_frequency.settings|allowed_values|12
```

This is how it looks in the `*.features.features_overrides.inc` file:

```
/**
 * Implements hook_features_override_default_overrides().
 */
function my_module_features_override_default_overrides() {
  // This code is only used for UI in features. Exported alters hooks do the magic.
  $overrides = array();

  // Exported overrides for: field_base
  $overrides["field_base.field_frequency.settings|allowed_values|0"] = '5th of jan';
  $overrides["field_base.field_frequency.settings|allowed_values|1"] = 'Adhoc';
  $overrides["field_base.field_frequency.settings|allowed_values|10"] = 'Fortnightly';
```

```
  $overrides["field_base.field_frequency.settings|allowed_values|11"] = 'Monthly';
  $overrides["field_base.field_frequency.settings|allowed_values|12"] = 'One-Time Load
↪';
  $overrides["field_base.field_frequency.settings|allowed_values|13"] = 'Other';
  $overrides["field_base.field_frequency.settings|allowed_values|14"] = 'Quarterly';
  ....
```

This is how it look in the `*.features.inc` file:

```
/**
 * Implements hook_field_default_field_bases_alter().
 */
function eic_custom_fields_field_default_field_bases_alter(&$data) {
  if (isset($data['field_frequency'])) {
    $data['field_frequency']['settings']['allowed_values'][0] = '5th of jan'; /* WAS:
↪'Annually' */
    $data['field_frequency']['settings']['allowed_values'][1] = 'Adhoc'; /* WAS:
↪'Biennial' */
    $data['field_frequency']['settings']['allowed_values'][10] = 'Fortnightly'; /*␣
↪WAS: 'Quarterly' */
    $data['field_frequency']['settings']['allowed_values'][11] = 'Monthly'; /* WAS:
↪'Semiannual' */
    $data['field_frequency']['settings']['allowed_values'][12] = 'One-Time Load'; /*␣
↪WAS: 'Semimonthly' */
    $data['field_frequency']['settings']['allowed_values'][13] = 'Other'; /* WAS:
↪'Semiweekly' */
    $data['field_frequency']['settings']['allowed_values'][14] = 'Quarterly'; /* WAS:
↪'Three times a month' */
    $data['field_frequency']['settings']['allowed_values'][15] = 'Semesterly'; /*␣
↪WAS: 'Three times a week' */
    $data['field_frequency']['settings']['allowed_values'][16] = 'Semesterly/Ad hoc';␣
↪/* WAS: 'Three times a year' */
    $data['field_frequency']['settings']['allowed_values'][17] = 'Weekly'; /* WAS:
↪'Triennial' */
    ...
```

Each of the values in the list is an individual item to override. It's nuts and something that will potentially drive people mad every time they need to solve a merge conflict. **Don't do it.**

Suscribe the list of values in a `hook_form_alter` implementation. If you need an example, look at how the list of allowed values for `field_license` is implemented. link here

Future enhacements

DKAN Core fields that utilize an allowed values list should implement the same usecase as `field_license`. This involves:

- Implementing the `hook_form_alter` approach in a core module
- Allow to subscribe and unsubscribe values using a hook implementation

1.9.2 Field groups

Overriding core field groups

Usual requests involve changing the **fields (children)** in a group as well as the **position (weight)** of the group in the node form. To accomplish this implement `hook_field_group_info_alter`:

```
/**
 * Implements hook_field_group_info_alter().
 */
function usda_content_types_field_group_info_alter(&$groups) {
  // Alter group_primary
  $primary = $groups['group_primary|node|dataset|form'];
  $primary->data['weight'] = 10;
  $primary->data['format_type'] = 'accordion-item';
  $primary->data['label'] = 'Primary Information';
  $primary->data['parent_name'] = 'group_primary_wrapper';
  $primary->data['children'] = array(
    'body',
    'field_data_dictionary',
    'field_doi',
    'field_specific_product_type',
    'field_odfe_landing_page',
    'field_theme',
    'field_author_is_organization',
    'field_nal_author',
    'field_source_id',
    'title',
  );
  $groups['group_primary|node|dataset|form'] = $primary;
  // Alter weight of group_odfie_pod.
  $odfe = $groups['group_odfie_pod|node|dataset|form'];
  $odfe->data['weight'] = 4;
  $groups['group_odfie_pod|node|dataset|form'] = $odfe;
  // Remove group_additional
  unset($groups['group_additional|node|dataset|form']);
  unset($groups['group_odfie_pod|node|dataset|form']);
}
```

Adding custom field_groups

Use features to export custom field_groups. If you need to include core fields in your field groups do remove it first for the core field group using the resource above.

1.9.3 Override core blocks

Implement `hook_block_view_alter`

```
/**
 * Implements hook_block_view_alter()
 */
function dkan_sitewide_usda_overrides_block_view_alter(&$data, $block) {
  if ($block->delta == 'dkan_sitewide_data_extent') {
    $data['content'] = dkan_sitewide_usda_overrides_data_extent_block();
```

```
  }
}
```

1.9.4 Override core permissions

Don't.

1.9.5 Custom permissions and roles

Don't

1.9.6 Extend core permissions

- Capture the additional set of needed permissions in a complementary user role name after the core user role (If you intend to extend editor, then name it after editor)
- Tie them up as **DKAN WORKFLOW** does it, using a `hook_form_alter` implementation. See this link.

1.9.7 Custom Search Facets

The problems

Search api index feature export is messy

The feature exports for the dkan core search api indexes are in fact `entity_imports` and the value being imported is a **json string**. You can't get around this with `features_overrides`, it does not offer a way to override individual items of the feature.

```
/**
 * Implements hook_default_search_api_index().
 */
function dkan_sitewide_search_db_default_search_api_index() {
  $items = array();
  $items['datasets'] = entity_import('search_api_index', '{
    "name" : "datasets",
    "machine_name" : "datasets",
    ...
      "fields" : {
        "author" : { "type" : "integer", "entity_type" : "user" },
        "changed" : { "type" : "date" },
        "created" : { "type" : "date" },
        "field_author" : { "type" : "string" },
        "field_license" : { "type" : "string" },
        "field_resources:body:value" : { "type" : "list\\u003Ctext\\u003E" },
        "field_resources:field_format" : { "type" : "list\\u003Cinteger\\u003E",
↪"entity_type" : "taxonomy_term" },
        "field_tags" : { "type" : "list\\u003Cinteger\\u003E", "entity_type" :
↪"taxonomy_term" },
        "field_topic" : { "type" : "list\\u003Cinteger\\u003E", "entity_type" :
↪"taxonomy_term" },
        "og_group_ref" : { "type" : "list\\u003Cinteger\\u003E", "entity_type" : "node
↪" },
```

```
        "search_api_access_node" : { "type" : "list\\u003Cstring\\u003E" },
        "search_api_language" : { "type" : "string" },
        "search_api_viewed" : { "type" : "text" },
        "status" : { "type" : "boolean" },
        "title" : { "type" : "string" },
        "type" : { "type" : "string" }
      },
```

Facets

Facet configuration for fields don't play nice with features as well

Search page panel

The whole search page is exported as a feature

Mise en place

Prepare a module to hold the search customizations. For the porpouse of this docs, let's name it `dkan_search_customizations`.

In `dkan_search_customizations.module` prepare a helper function to retrieve all the new taxonomies to be added to the index:

```
function dkan_search_customizations_facet_list() {
  return array(
    'field_brand',
    'field_country',
    'field_customer_type',
  );
}
```

This array is going to be used multiple times so it's good practice to avoid repetition.

Same goes for the searchers/indexes you want to add these facets to. Prepare another helper function to retrieve those:

```
function dkan_search_customizations_searchers_list() {
  return array(
    'search_api@datasets',
    'search_api@groups_di'
  );
}
```

The solutions

Add fields to the index

Implement `hook_default_search_api_index_alter`

```
/**
 * Implements hook_default_search_api_index_alter().
 */
```

```
function dkan_search_customizations_default_search_api_index_alter(&$defaults) {
  $searchers = dkan_search_customizations_searchers_list();
  foreach ($searchers as $searcher) {
    $searcher = str_replace('search_api@', '', $searcher);
    $facets = dkan_search_customizations_facet_list();
    foreach ($facets as $facet) {
      if (isset($defaults[$searcher])) {
        $defaults[$searcher]->options['fields'][$facet] = array(
          'type' => 'list<integer>',
          'entity_type' => 'taxonomy_term',
        );
      }
    }
  }
}
```

This will mark the field to be indexed for the given indexes/searchers.

QA

- Revert the core search modules

```
ahoy drush -y fr --force dkan_sitewide_search_db dkan_dataset_groups
```

- Go to `/admin/config/search/search_api`
- All the overriden indexes should appear to be using the *Default* Configuration (Not overriden)
- Inspect `admin/config/search/search_api/index/<searcher|index>/fields` for all indexes in `dkan_search_customizations_searchers_list`
- All fields in `dkan_search_customizations_facet_list` should be checked to be indexed

Enable facets for the indexes

There's no hook implementation to accomplish this. The facet configuration needs to be saved in the db. In order to do this, the most straigthforward solution is to write a function to take care of enabling the facets and have that function run on deployment through the env-switch hook. It'll look something like this:

```
function dkan_search_customizations_enable_facets() {
  $searchers = dkan_search_customizations_searchers_list();
  foreach ($searchers as $searcher) {
    $realm = 'block';
    $facets_to_enable = dkan_search_customizations_facet_list();
    $adapter = facetapi_adapter_load($searcher);
    $facet_info = facetapi_get_facet_info($searcher);
    foreach (array_keys($facet_info) as $item) {
      $facet = facetapi_facet_load($item, $searcher) ;
      if (in_array($item, $facets_to_enable)) {
        $facet_settings = $adapter->getFacet($facet)->getSettings($realm);
        $facet_settings->enabled = 1;
        ctools_export_crud_save('facetapi', $facet_settings);
      }
    }
  }
}
```

Running this function will mark the facets to be enabled. We also need to make sure this runs on deployments. To do that, implement `hook_environment_switch` for your module

```
function dkan_search_customizations_environment_switch($target_env, $current_env) {
    dkan_search_customizations_enable_facets()
}
```

QA

- Run `ahoy drush -y env-switch --force local`
- Visit `admin/config/search/search_api/index/<searcher|index>/facets` for all the overriden indexes
- Facets should be enabled

Add facets block to search pages

This can be accomplished with a `hook_panels_pre_render` implementation:

```
function dkan_search_customizations_panels_pre_render(&$display, $renderer) {
  $searcher = FALSE;
  // Detect with searcher it is based on the panel display.
  if (is_numeric(strpos($display->cache_key, 'page-datasets'))) {
    $searcher = 'search_api@datasets';
  }
  if (is_numeric(strpos($display->cache_key, 'panel_context:node_view::node_view_
↪panel_context_3'))) {
    $searcher = 'search_api@groups_di';
  }
  // If searcher is ment to be overriden retrieve blocks.
  if ($searcher) {
    $map = facetapi_get_delta_map();
    $realms = array();
    $panes = array();
    $faceted_fields = dkan_search_customizations_facet_list();
    // Cross reference fields with facet map
    foreach($map as $machine_name => $facet) {
      foreach($faceted_fields as $field) {
        // Special case for resource taxonomies
        $f = str_replace('field_resources:', '', $field);
        // If Facet matchs with field and searcher, save realm.
        if (is_numeric(strpos($facet, $f)) && is_numeric(strpos($facet, $searcher))) {
          $realms[$field] = $machine_name;
        }
      }
    }

    // Prepare panes for realms.
    foreach($realms as $field => $machine_name) {
      // Create a new pane based on the facet realm.
      $pane = panels_new_pane('block', $machine_name, TRUE);
      $pane->uuid = ctools_uuid_generate();
      $pane->pid = 'new-' . $pane->uuid;
      $pane->panel = 'sidebar';
      $pane->subtype = 'facetapi-' . $machine_name;
```

```
      unset($pane->did);
      $pane->shown = TRUE;

      // Customize titles.
      switch($field){
        case 'field_brand':
          $title = 'Brand';
          break;
        case 'field_country':
          $title = 'Country';
          break;
        case 'field_customer_type':
          $title = 'Customer Type';
          break;
      }
      $pane->configuration = array(
        'override_title' => TRUE,
        'override_title_heading' => 'h2',
        'override_title_text' => $title
      );
      $pane->css = array(
        'css_id' => '',
        'css_class' => 'pane-facetapi pane-block',
      );
      $pane->style = array(
        'settings' => array(
          'pane_title' => '%title',
          'pane_collapsed' => TRUE,
          'pane_empty_check' => FALSE,
        ),
        'style' => 'collapsible',
      );
      $panes[$field] = $pane;
    }

    foreach ($faceted_fields as $field) {
      if (isset($panes[$field])) {
        $pane = $panes[$field];
        // Adds facet block reference to the sidebar.
        $display->panels['sidebar'][] = $pane->pid;
        // Add pane to the pane content.
        $display->content[$pane->pid] = $pane;
      }
    }
  }
}
```

QA

- Rebuild registry and clear cache
- Blocks should render in the search page

Future enhacements

- Field and searchers options could be enhance to use a structure like this:

```
function dkan_search_customizations_facet_list() {
  return array(
    'field_brand' => array(
        'searchers' => array('search_api@datasets', 'search_api@groups_di'),
        'title' => 'Brand',
    ),
    'field_country' => array(
        'searchers' => array('search_api@datasets'),
        'title' => 'Country',
    ),
    'field_customer_type' => array(
        'searchers' => array('search_api@groups_di'),
        'title' => 'Customer Type',
    ),
  );
}
```

- These options, with some work, could be wrapped up in a generic module that retrieves the fields and searchers from variables. Variables could be set from `settings.custom.php`.