



DRUPALCON  
Prague

# DRUPAL 8 INFO HOOK TO PLUGIN

Track: Coding + Development · 25 September 2013

Speaker: Peter Wolanin (pwolanin) [drupal.org/user/49851](http://drupal.org/user/49851)

# Here's What's In the Talk

- Background and “What’s a plugin?”
- Example of a core info hook conversion
- Simple but common plugin example
  - Adding new tabs (a.k.a. local tasks)
- Configurable plugins (ConfigEntity based)
  - Adding a custom block
  - Adding a custom text filter
- Considerations for defining your own plugins

# Drupal 8 Background

- I'll assume you know something about:
  - ▶ The DIC/container/service container - an object that contains instances of “services” (the current request, the current user, URL generator, etc).
- The new routing system - names instead of paths.
  - ▶ a route name is just a machine name that connects to a path pattern, callbacks to provide, title, content, access etc. - like a D7 menu router.
- Namespaced classes (PHP 5.3+) like  
`\Drupal\search\Plugin\Block\SearchBlock`

# Plugins:

- Encapsulate some re-useable functionality inside a class that implements one or more specific interfaces.
- Plugins combine what in Drupal 7 was an info hook and a number of implementation hooks and possibly configuration: e.g. `hook_search_info()` and `hook_search_execute()`, etc., or `hook_block_info()` and `hook_block_view()`, `_configure()`, `_save()`
- Evolved from ctools and views plugins, but use quite different mechanisms to discover them.

# Plugin Manager and IDs

- Every plugin type has a manager - registered as a service (available from the DIC) and used to find and instantiate the desired plugin instance(s).
- Each plugin has an ID, which may be in its definition, or generated as a derivative.
- For a given plugin ID one single class will be used for any plugin instances using that plugin ID.
- A plugin instance is specified by the combination of plugin ID and its configuration values, potentially coming from a ConfigEntity.



**“The Drop is  
always moving”**

<https://drupal.org/node/65922>

# 7.x: hook\_image\_toolkits()

```
/**
 * Implements hook_image_toolkits().
 */
function system_image_toolkits() {
  include_once DRUPAL_ROOT . '/' . drupal_get_path('module',
'system') . '/' . 'image.gd.inc';
  return array(
    'gd' => array(
      'title' => t('GD2 image manipulation toolkit'),
      'available' => function_exists('image_gd_check_settings') &&
        image_gd_check_settings(),
    ),
  );
}
```

# 8.x: ImageToolkitManager

```
class ImageToolkitManager extends DefaultPluginManager {  
    // ... various methods ... //  
  
    /**  
     * Gets a list of available toolkits.  
     */  
    public function getAvailableToolkits() {  
        // Use plugin system to get list of available toolkits.  
        $toolkits = $this->getDefinitions();  
  
        $output = array();  
        foreach ($toolkits as $id => $definition) {  
            if (call_user_func($definition['class'] . '::isAvailable')) {  
                $output[$id] = $definition;  
            }  
        }  
        return $output;  
    }  
}
```



# 7.x: desaturate function

```
/**
 * Converts an image to grayscale.
 *
 * @param $image
 *   An image object returned by image_load().
 *
 * @return
 *   TRUE on success, FALSE on failure.
 *
 * @see image_load()
 * @see image_gd_desaturate()
 */
function image_desaturate(stdClass $image) {
    return image_toolkit_invoke('desaturate', $image);
}
```

# 8.x: desaturate method

```
/**
 * Defines the GD2 toolkit for image manipulation within Drupal.
 *
 * @ImageToolkit(
 *   id = "gd",
 *   title = @Translation("GD2 image manipulation toolkit")
 * )
 */
class GDToolkit extends PluginBase implements ImageToolkitInterface {
  // ... all the toolkit methods ... //

  public function desaturate(ImageInterface $image) {
    // PHP using non-bundled GD does not have imagefilter.
    if (!function_exists('imagefilter')) {
      return FALSE;
    }

    return imagefilter($image->getResource(), IMG_FILTER_GRAYSCALE);
  }
}
```

# What about hooks?



# Hooks still have their place:

- Many plugin managers invoke an `_alter` hook so the modules can add to or alter the plugins' definitions. E.g. `hook_block_alter()` allows you to alter the block plugin definitions.
- Info hooks that simply return a data array - like `hook_permission()` - without associated functionality - are not candidates to become plugins.

# Plugin Discovery

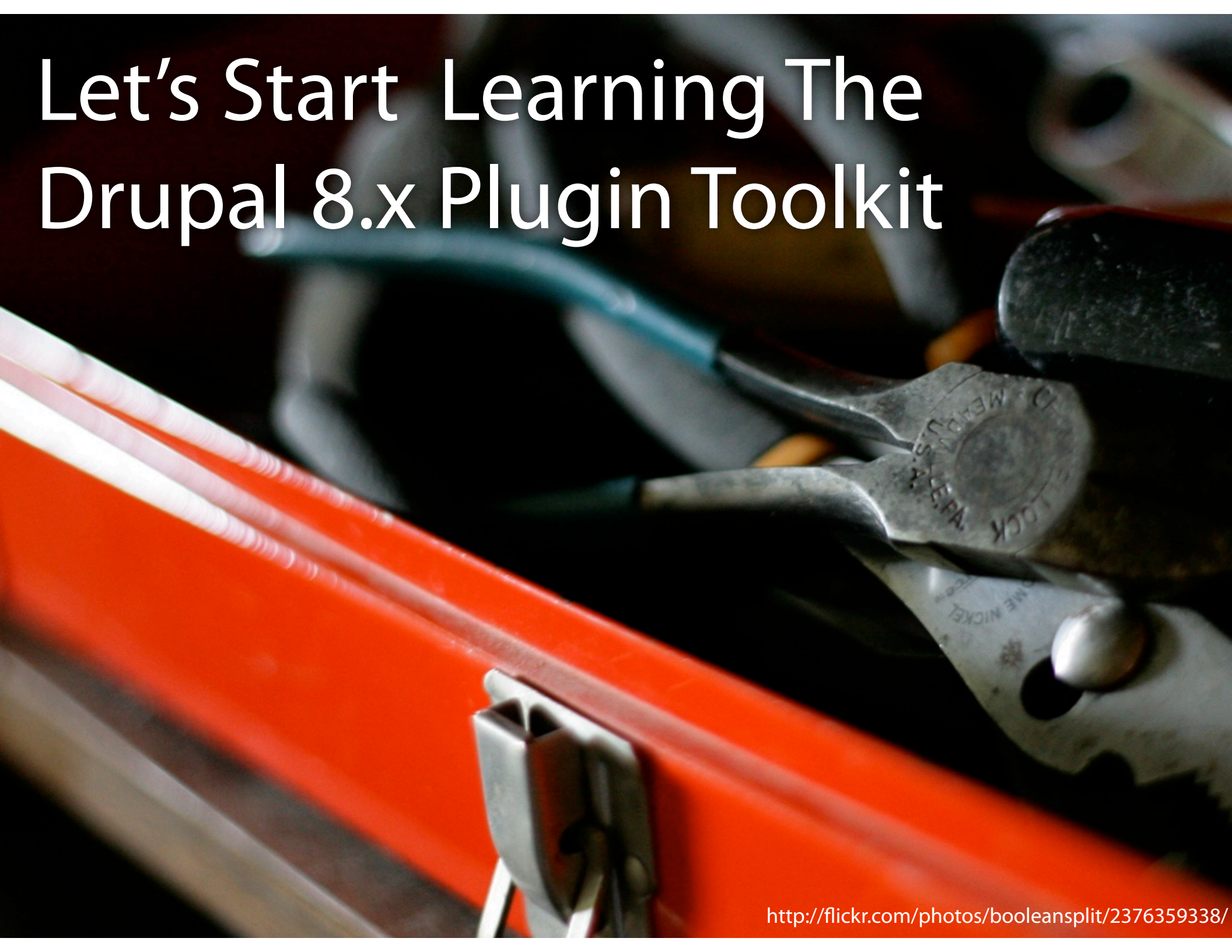
- The discovery of plugins is basically the same as invoking an info hook (in fact you can implement it that way).
- Discovery gives you an array of plugin definitions, each of which is just an array of keys and values.
- The discovery process fills in defaults, such a `'provider'` which is the name of the module providing the plugin.

# Plugin Discovery/Config in Core

- YAML based:
  - **LocalTask\***, **LocalAction\***, **ContextualLink\***
- Annotation, some config, but no config entity:
  - **SearchPlugin**, **ImageToolkit**, **Archiver**, **StreamWrapper\***
- Annotation and config entity (many) including:
  - **Block**, **ViewsDisplay**, **ImageEffect**, **Tip**
- Not truly a Plugin but uses Annotation discovery:
  - **Entity** (Node, User, etc.)

\* patch or changes pending as of 2013-09-25

# Let's Start Learning The Drupal 8.x Plugin Toolkit



# Pre-requisite: A Module

- As in Drupal 7, blocks are provided by modules - so you need a module! You need a .info.yml file and an (empty) .module file.

## modules/mymodule/mymodule.info.yml

```
name: 'My test module'  
type: module  
description: 'Drupalcon demo.'  
core: 8.x
```

## modules/mymodule/mymodule.module

```
<?php  
  
/**  
 * @file  
 * Drupalcon demo module.  
 */
```



# Adding Two Tabs:

- For most uses, just add a YAML file listing your tabs: **mymodule/mymodule.local\_tasks.yml**

```
mymodule_list_tab: ← Plugin ID
  route_name: mymodule.list
  title: 'List'
  tab_root_id: mymodule_list_tab

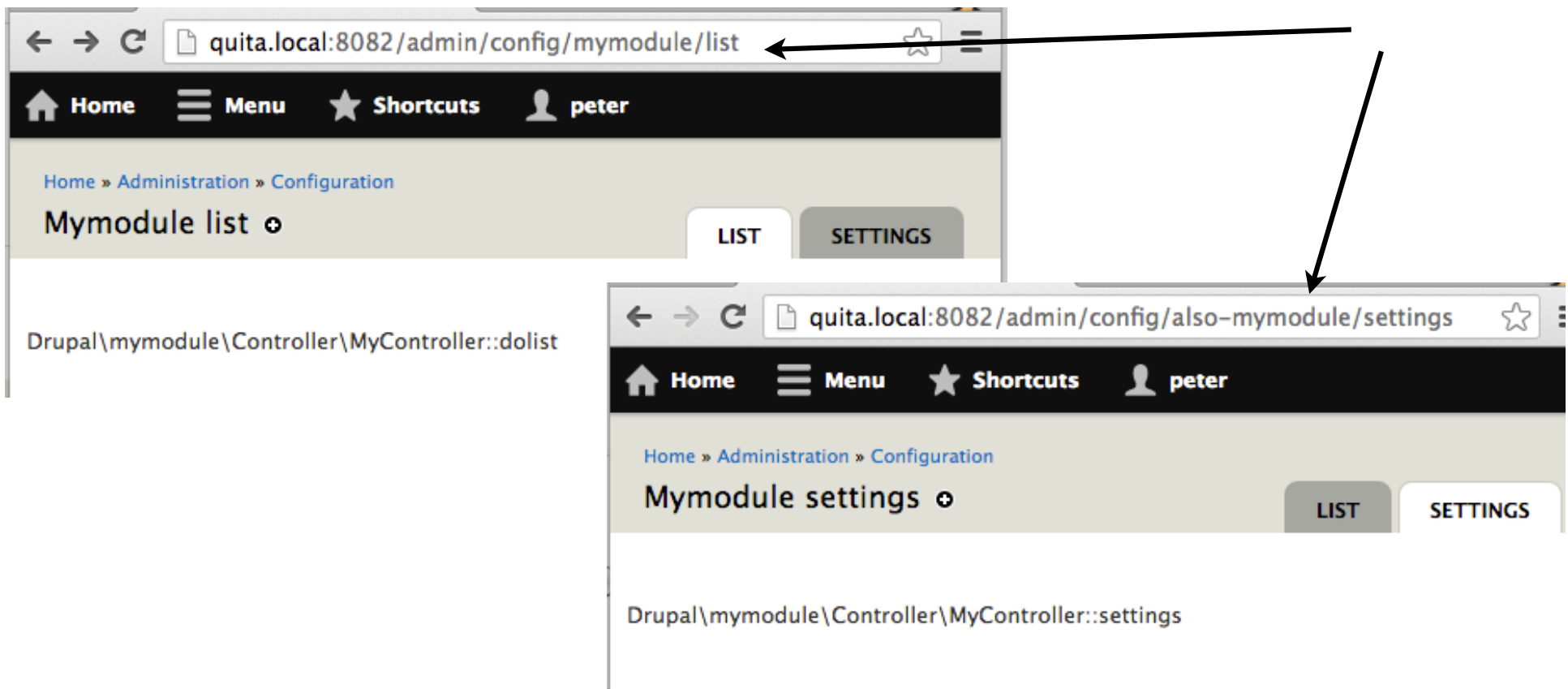
mymodule_settings_tab:
  route_name: mymodule.settings
  title: 'Settings'
  tab_root_id: mymodule_list_tab
```

Local tasks reference one as the root tab

\*Check back for possible YAML changes <https://drupal.org/node/2095117>

# Adding Two Tabs:

- Unlike Drupal 7 you don't need to jump through the hoops of a default local task, or making the paths align in a certain hierarchy



# LocalTask Plugins Need Routes:

- Of course, those routes need to be defined or pre-existing : **mymodule/mymodule.routing.yml**

```
mymodule.list:
```

```
  path: '/admin/config/mymodule/list'
```

```
  defaults:
```

```
    _content: '\Drupal\mymodule\Controller\MyController::dolist'
```

```
    _title: 'Mymodule list'
```

```
  requirements:
```

```
    _access: 'TRUE'
```

```
mymodule.settings:
```

```
  path: '/admin/config/also-mymodule/settings'
```

```
  defaults:
```

```
    _content: '\Drupal\mymodule\Controller\MyController::settings'
```

```
    _title: 'Mymodule settings'
```

```
  requirements:
```

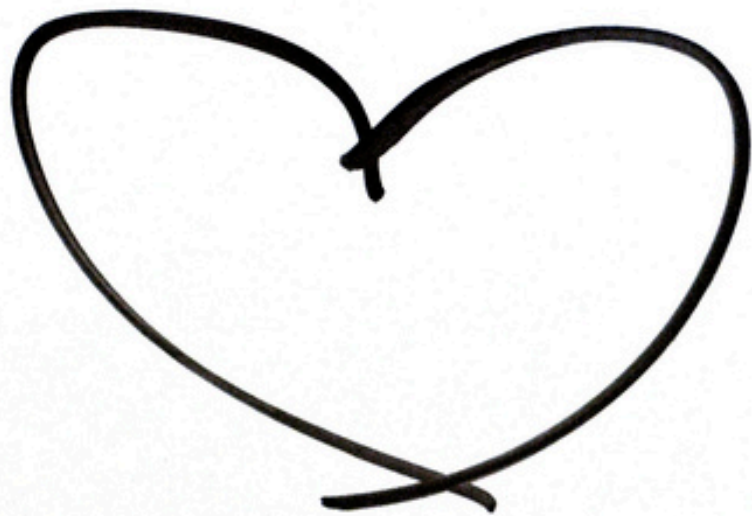
```
    _access: 'TRUE'
```

# LocalTask Plugin Keys:

The plugin configuration options and defaults are on the LocalTaskManager class

```
class LocalTaskManager extends DefaultPluginManager {
  protected $defaults = array(
    // (required) The name of the route this task links to.
    'route_name' => '',
    // Parameters for route variables when generating a link.
    'route_parameters' => array(),
    // The static title for the local task.
    'title' => '',
    // The plugin ID of the root tab.
    'tab_root_id' => '',
    // The plugin ID of the parent tab (or NULL for the top-level tab).
    'tab_parent_id' => NULL,
    // The weight of the tab.
    'weight' => NULL,
    // The default link options.
    'options' => array(),
    // Default class for local task implementations.
    'class' => 'Drupal\Core\Menu\LocalTaskDefault',
    // The plugin id. Set by the plugin system based on the top-level YAML key.
    'id' => '',
  );
};
```

ger  
g  
ms



The New Block



# Blocks as Plugins

- Each custom block is defined in code as a class.
- When the admin places the block into a region in a theme a configuration object is created to track that setting.
- The config object is a ConfigEntity - it's an abstraction on top of CMI (storing your Drupal configuration in YAML files) - it makes it convenient to list, load, etc. using entity functions. So Drupal can easily list the active block instances.
- Note - you don't need to worry about the config!

# Blocks Implementation

- Blocks implement the `\Drupal\block\BlockPluginInterface`
- If you extend the abstract `\Drupal\block\BlockBase` class then all you need to implement is the `build()` method.
- `build()` is basically the same as `hook_block_view()` in Drupal 7
- For example, I added to my module `\Drupal\mymodule\Plugin\Block\MyBlock`

# Side Note - PSR-0/4

- When I add the Block to my module:  
`\Drupal\mymodule\Plugin\Block\MyBlock`
- This is at the corresponding filepath (under the Drupal root dir):  
`modules/mymodule/lib/Drupal/mymodule/Plugin/Block/MyBlock.php`
- Yeah, that's long. Note that the full class name and path match under `lib/`
- PSR-4 will be adopted soon and then it will be:  
`modules/mymodule/lib/Plugin/Block/MyBlock.php`



```

/**
 * Provides a block with 'Mymodule' links.
 *
 * @Block(
 *   id = "mymodule_my_block",
 *   admin_label = @Translation("Mymodule block")
 * )
 */
class MyBlock extends BlockBase {
    public function build() {
        return array(
            'first_link' => array(
                '#type' => 'link',
                '#title' => $this->t('Mymodule List'),
                '#route_name' => 'mymodule.list',
            ),
            'second_link' => array(
                '#type' => 'link',
                '#title' => $this->t('Mymodule Settings'),
                '#route_name' => 'mymodule.settings',
            )
        );
    }
}

```

BLOCK LAYOUT

CUSTOM BLOCK LIBRARY

Bartik

Seven

pecific title and visibility settings, click the block title under  
not be saved until you click the *Save blocks* button at the

### Place blocks

- ▼ COMMENT
  - + Recent comments
- ▼ MENU
  - + Administration
  - + Footer
  - + Main navigation
  - + Tools
  - + User account menu
- ▼ MY TEST MODULE
  - + Mymodule block

# Blocks Admin Page Has a New Section: Place blocks

## Place block



### Title\*

Machine name: mymoduleblock [\[Edit\]](#)

Display title

### Region

- None -

Select the region where this block should be displayed.

### Visibility settings

#### Pages

Not restricted

#### Content types

Not restricted

#### Roles

Not restricted

#### Show block on specific pages

All pages except those listed

Only the listed pages

Specify pages by using their paths. Enter one path per line. The '\*' character is a wildcard. Example paths are *user* for the current user's page and *user/\** for every user page. *<front>* is the front page.

Save block

## BLOCK LAYOUT

specific title and visibility settings, c  
be saved until you click the Save

## Place blocks

Filter by block name

### ▼ COMMENT

+ Recent comments

### ▼ MENU

+ Administration  
+ Footer  
+ Main navigation  
+ Tools  
+ User account menu

### ▼ MY TEST MODULE

+ Mymodule block

### ▼ NODE

+ Recent content  
+ Syndicate

### ▼ SEARCH

Content

Configure



### Mymodule block

[Mymodule List](#)

[Mymodule Settings](#)

### Search

### Tools

- ▶ [Add content](#)
- ▶ [Add custom block](#)

## Welcome to quita.local

No front page content has been created yet.

- [Add new content](#)



# Hook to Plugin Comparison:

## Drupal 7.x

`hook_block_info()`

---

`hook_block_view($delta)`

---

?

---

`hook_block_configure($delta)`

---

?

---

`hook_block_save($delta, $edit)`

---

## Drupal 8.x

BlockManagerInterface::  
`getDefinitions()`

---

BlockPluginInterface::  
`build()`

---

BlockPluginInterface::  
`access()`

---

BlockPluginInterface::  
`blockForm($form, &$form_state)`

---

BlockPluginInterface::  
`blockValidate($form, &$form_state)`

---

BlockPluginInterface::  
`blockSubmit($form, &$form_state)`

---

# Block Discovery and Annotations

- Each Plugin type must be in the expected class namespace for your module - for blocks:  
`namespace Drupal\mymodule\Plugin\Block;`
- Most core plugins have a custom annotation class - you have to use the right one for your plugin.
- The annotation class provides both a documentation of the possible keys in the plugin definition and default values.
- There is also a generic Plugin annotation class.

```
/**
 * Defines a Block annotation object.
 *
 * @Annotation
 */
class Block extends Plugin {

    /**
     * The plugin ID.
     *
     * @var string
     */
    public $id;

    /**
     * The administrative label of the block.
     *
     * @var \Drupal\Core\Annotation\Translation
     *
     * @ingroup plugin_translatable
     */
    public $admin_label;
}
```

# Filtering Text





# 7.x Info Hook:

```
function hook_filter_info() {  
    $filters['filter_autop'] = array(  
        'title' => t('Convert line breaks'),  
        'description' => t('Converts line breaks into HTML tags.'),  
        'process callback' => '_filter_autop',  
        'tips callback' => '_filter_autop_tips',  
    );  
    return $filters;  
}
```

```


/**
 * Provides a filter to format a node ID as a link.
 *
 * @Filter(
 *   id = "mymodule_node_link",
 *   module = "mymodule",
 *   title = @Translation("Format a node ID as a link"),
 *   type = FILTER_TYPE_MARKUP_LANGUAGE,
 * )
 */
class NodeLinkFilter extends FilterBase {
  public function process($text, $langcode, $cache, $cache_id) {
    $regex = '(?: (?<!\w)\[#\d+(?:-[\d\.]+)?\](?! \w))|...';
    $text = preg_replace_callback("/$regex/",
      array($this, 'replaceCallback'), $text);
    return $text;
  }
  // ... more code ...//

  public function tips($long = FALSE) {
    return $this->t('Node ID numbers (ex. [#12345]) turn into
links automatically.');
```

← → ↻ 📄 quita.local:8082/node/1 ☆ ☰

🏠 Home ☰ Menu ☆ Shortcuts 👤 peter ✎

My account Log out

 quita.local

Home

Home

Search

Tools

▶ [Add content](#)

## First test post

[View](#) [Edit](#)

Submitted by [peter](#) on Tue, 09/17/2013 - 08:06

Here's a 1st node

← → ↻ [quita.local:8082/admin/config/content/formats](http://quita.local:8082/admin/config/content/formats) ☆

Home Menu Shortcuts peter

Content Structure Appearance Extend Configuration People Reports

Help

Home » Administration » Configuration » Content authoring

## Text formats and editors

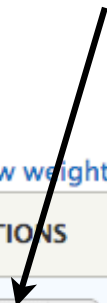
Text formats define the HTML tags, code, and other formatting that can be used when entering text. **Improper text format configuration is a security risk.** Learn more on the [Filter module help page](#).

Text formats are presented on content editing pages in the order defined on this page. The first format available to a user will be selected by default.

[+ Add text format](#)

[Show row weights](#)

NAME	TEXT EDITOR	ROLES	OPERATIONS
⊕ Basic HTML	CKEditor	Authenticated user, Administrator	<a href="#">Configure</a> ▾



[quita.local:8082/admin/config/content/formats/manage/basic\\_html](http://quita.local:8082/admin/config/content/formats/manage/basic_html)

← → ↻ [quita.local:8082/admin/config/content/formats/manage/basic\\_html](http://quita.local:8082/admin/config/content/formats/manage/basic_html) ☆ ☰

Home ☰ Menu ☆ Shortcuts peter

Maximum dimensions  
width  x height  pixels  
Images larger than these dimensions will be scaled down.

### Enabled filters

- Limit allowed HTML tags
- Display any HTML as plain text
- Convert URLs into links
- Convert line breaks into HTML (i.e. `<br>` and `<p>`)
- Format a node ID as a link
- Display image captions and align images  
Uses `data-caption` and `data-align` attributes on `<img>` tags to caption and align images.
- Restrict images to this site  
Disallows usage of `<img>` tag sources that are not hosted on this site by replacing them with a placeholder image.
- Correct faulty and chopped off HTML

### Filter processing order

← → ↻  ☆

Home Menu Shortcuts peter

Body (Edit summary)

**B** *I* Source

Let me tell you about [#1]

Text format  [More information about text formats ?](#)

- Allowed HTML tags: `<a>` `<em>` `<strong>` `<cite>` `<blockquote>` `<code>` `<ul>` `<ol>` `<li>` `<dl>` `<dt>` `<dd>` `<h4>` `<h5>` `<h6>` `<p>` `<span>` `<img>`
- You can caption (`data-caption="Text"`) and align images (`data-align="center"`), but also video, blockquotes, and so on.
- Only images hosted on this site may be used in `<img>` tags.
- Node ID numbers (ex. [#12345]) turn into links automatically.


▶ AUTHORIZING INFORMATION

▶ PROMOTION OPTIONS

← → ↻  ☆ ☰

🏠 Home ☰ Menu ☆ Shortcuts 👤 peter ✎

My account Log out

 quita.local

Home

Home

Search

Tools

## A second post

View Edit

Submitted by peter on Tue, 09/17/2013, 08:12

Let me tell you about [First test post](#)



# Creating Your Own Plugins

- You want to upgrade your module to Drupal 8 and it defined an info hook or had a ctools plugin type.
- Annotation based discovery should be the default.
- It keeps the meta-data together with the class and it suited for most plugins where the actual class (code) is different for most plugins.
- The YAML discovery is good for a case like local tasks where the vast majority use a common class, but a few will implement a different one (e.g. to provide a dynamic title).



# Plugin and General 8.x Resources

- Demo code used for this presentation:  
<https://drupal.org/sandbox/pwolanin/2087657>
- *Converting 7.x modules to 8.x*  
<https://drupal.org/update/modules/7/8>
- *Plugin API in Drupal 8*  
<https://drupal.org/node/2087839>
- *Understanding Drupal 8's plugin system*  
<http://previousnext.com.au/blog/understanding-drupal-8s-plugin-system>

# Drupal 8 Features I Mentioned

- Widespread use of interfaces makes it easier to replace almost any implementation.
- Tabs are grouped regardless of path hierarchy.
- Route name rather than system path is unique.
- Multiple routes can serve the same path (HTML vs. JSON or GET vs. POST).
- “variables” split into config (deployable) and state.
- YAML as a standard for config and data files.
- Multiple instances of the same block.

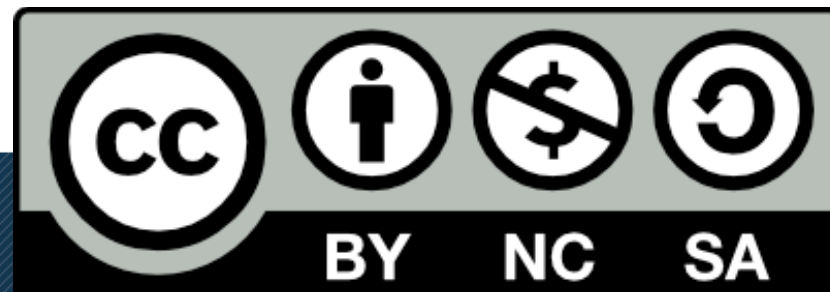
# To Sum It Up

- Plugins are a way to combine the discovery of available functionality with the actual implementation of the functionality.
- In Drupal 7, the combination of an info hook and multiple other hook functions (potentially in different files) served the same purpose.
- When defining your own plugins, use Annotation-based discovery unless you have a very clear reason for a different type.

This presentation is © 2013, Acquia, Inc.

Licensed:

<http://creativecommons.org/licenses/by-nc-sa/2.0/>





DRUPALCON  
Prague

# Sprint: Friday

Sprint with the community on Friday.

We have tasks for every skill set.

Mentors are available for new contributors.

An optional Friday morning workshop will help you set up community tools.

Follow @drupalmentoring



<http://prague2013.drupal.org/sprints>



# WHAT DID YOU THINK?

Locate this session at the  
DrupalCon Prague website:

[prague2013.drupal.org/node/1228](http://prague2013.drupal.org/node/1228)

Click the “Take the survey” link

# THANK YOU!