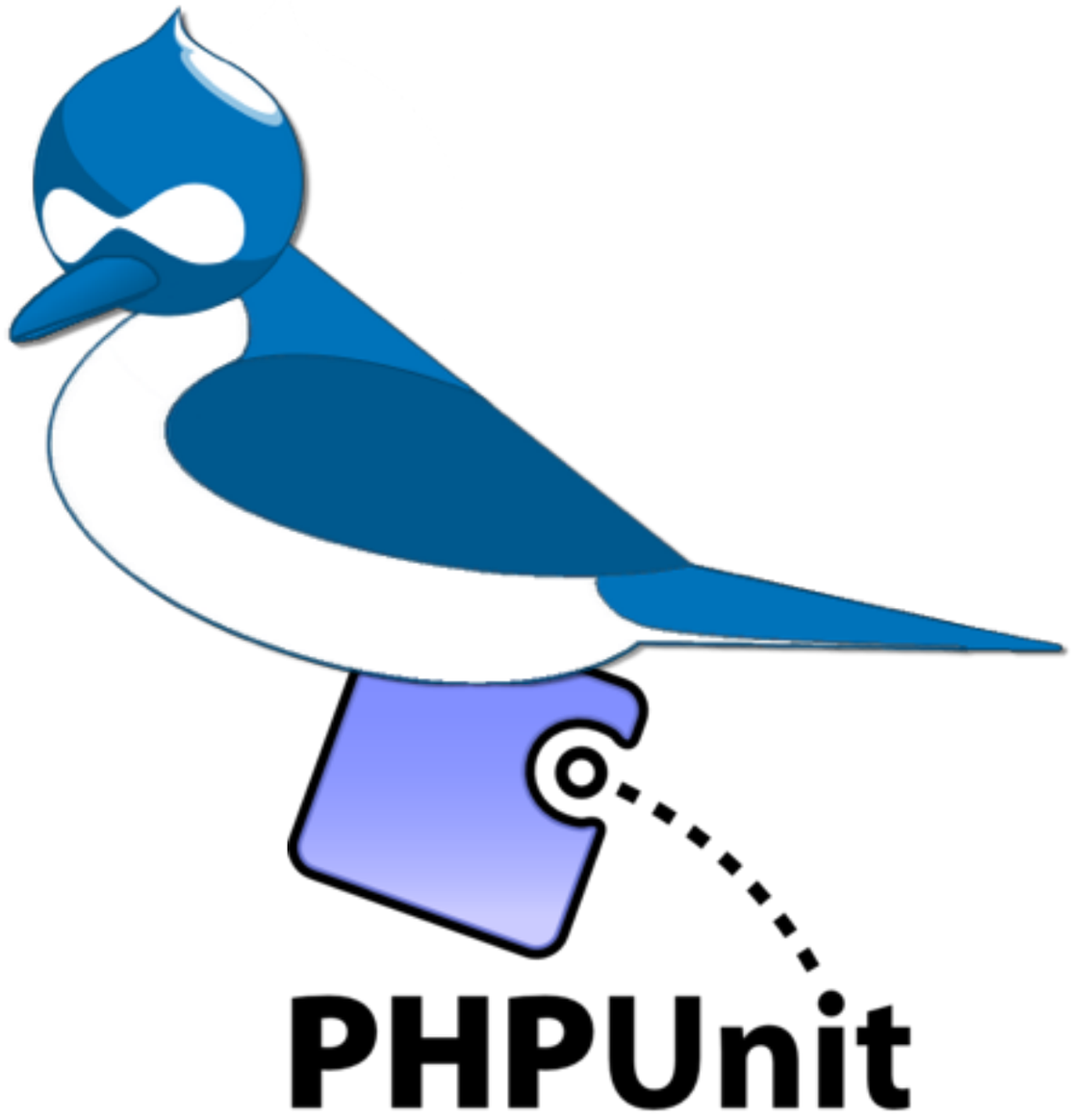
Mocking Drupal: Unit Testing in Drupal 8

Matthew Radcliffe
mradcliffe
@mattkineme



Spoilers

- Quality Assurance
- PHPUnit
- Drupal and PHPUnit

Quality Assurance

- Prevent defects from making it to the customer:
  - Adopt standards and specifications
  - Review code
  - Manage releases
  - Test code

Some Types of Testing

- User Acceptance Test: Test according to specification or requirement.

- Functional Test: Test one function (or feature) expected output.

- Unit Test: Test the smallest "unit" of testable code (in isolation).

- Integration Test: Test the interaction of "units" in I or more systems.

- Behavioral Test: Automated UAT or black box testing.

- Stress Test: Test the product under heavy load/stress.

Value of Testing

- Increase reliability and quality of software.
  - "I didn't think of that!"
- Discover regressions in software.
  - "Why did it break?"
- Improve confidence in our code.
  - "I think this will work."

Common Complaints

Writing tests takes too long.	Start small. 100% coverage isn't going to come in a day.
I don't have source control / version control.	Do not pass go. Do not write any more code. Go directly to a Git repository.
I don't have any testing infrastructure	Run locally, enforce social contract. Or setup TravisCI publicly or privately.
I don't know what I'm going to write until I write it.	Not everyone needs to adopt Test-Driven Development, but it is "best practice".
My code is heavily integrated with state (database or web services).	That's where test doubles come into play.

Unit Tests

- A unit is the smallest testable piece of code, which is often a function, class or method.
- Plug a set of inputs into that code, and confirm the expected output (like behavioral and integration tests).
- Units should act in memory and not depend on other systems.
- Should be fast. Do not run Drupal installation.
- Run all unit tests after code change.

Drupal & Unit Tests

- Modules have complex dependencies and setup necessary to do what they do.
- Simpletest module is still a test runner for both SimpleTest and PHPUnit tests, but
  - You may use phpunit directly instead (my approach).
  - core/scripts/run-tests.sh is a hot mess.

PHPUnit: Getting Started

- phpunit.xml
- Bootstrap
- Test class in tests/src instead of src/Tests.
- Annotations
- Assertions
- Data Providers
- Test Doubles

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit>
  <php>
    <ini name="error_reporting" value="32767"/>
    <ini name="memory_limit" value="-1"/>
  </php>
  <testsuites>
    <testsuite name="My Module Unit Test Suite">
      <directory>tests</directory>
      <exclude>./vendor</exclude>
      <exclude>./drush/tests</exclude>
    </testsuite>
  </testsuites>
  <!-- Filter for coverage reports. -->
  <filter>
    <whitelist>
       <directory>src</directory>
       <exclude>
         <directory>src/Tests</directory>
       </exclude>
    </whitelist>
  </filter>
</phpunit>
```

Use Core's Bootstrap?

- Add more namespaces to autoloader than are necessary i.e. increased memory.
    - Necessary because contrib namespaces are not loaded by Composer autoloader.
    - Can re-use core abstract test class with mocked dependencies easier.
- Relative path may conflict with where phpunit can be run from.

Abstract Test Classes

- Drupal\Tests\UnitTestCase
  - Basic unit test class with some useful test doubles.
  - Drupal\Tests\Core\Form\FormTestBase
- Drupal\KernelTests\KernelTestBase
  - Adds database and filesystem state, but without an installation at the cost of performance. Not as slow as SimpleTest functional tests.

PHPUnit: Annotations

- @group
  - Required for DrupalCI integration for contributed module tests.
- @coversDefaultClass
- @expectedException

PHPUnit: Assertions

- assertEquals
- assertEmpty
- assertSame
- assertInstanceOf
- assertXmlStringEqualsXmlFile
- assertArrayHasKey
- assertArrayHasSubset
- assertCount
- assertFileExists
- assertNull

PHPUnit Manual. https://phpunit.de/manual/current/en/appendixes.assertions.html

PHPUnit: Data Providers

- A data provider is a method that returns an array of parameters to pass into a test method.
    - A test method may test several inputs.
- Important to note that data provider methods are run before any setup so this cannot depend on any test doubles.

tests/FibonacciTest.php

```
19      function sequenceProvider() {
20          return [
21              [0, 0],
22              [5, 8],
23              [10, 55],
24              [100, 354224848179261915075],
25              [-5, 5],
26              [-6, 8]
27          ];
28      }
29
30      /**
31       * Test class with data provider.
32       *
33       * @dataProvider sequenceProvider
34       */
35      function testFibonacciWithProvider($number, $output) {
36          $this->assertEquals($output, fibonacci($number));
37      }
38  }
```

PHPUnit: setUp

- The `setUp` method is executed for every test method in a class.

- Configure fixtures or setting up known state such as database or test files.

- Configure test doubles or mocks, which are dependencies of a unit that you do not need to test in that test class.

- Advice: Add you own abstract test classes to create fixtures or test double commonly used across your test classes.

PHPUnit: Test Doubles

- Test doubles (or mock objects) allow to focus a unit test on the code that needs to be tested without bootstrapping dependencies.
- Example: I don't want to load Drupal 8's Entity system when I test my unrelated code. Instead PHPUnit allows to create a mock via Reflection so that I get an object that looks like an Entity.
    - Reflection or introspection allows a program to know about and modify itself at runtime.

```
class KeyTestBase extends UnitTestCase {

  protected function setUp() {
    parent::setUp();

    // Mock the Config object, but methods will be mocked in the test class.
    $this->config = $this->getMockBuilder('\Drupal\Core\Config\ImmutableConfig')
      ->disableOriginalConstructor()
      ->getMock();

    // Mock the ConfigFactory service.
    $this->configFactory = $this->getMockBuilder('\Drupal\Core\Config
\ConfigFactory')
      ->disableOriginalConstructor()
      ->getMock();
    $this->configFactory->expects($this->any())
      ->method('get')
      ->with('key.default_config')
      ->willReturn($this->config);

    // Create a dummy container.
    $this->container = new ContainerBuilder();
    $this->container->set('config.factory', $this->configFactory);
    \Drupal::setContainer($this->container);
  }
}
```

PHPUnit: Test Doubles

- getMockBuilder()
    - disableOriginalConstructor()
- method()
    - with()
        - callback()
    - withConsecutive()
    - will()
        - onConsecutive()
        - returnValueMap()

Test double Risks

- Assume too much about the framework.
  - Tests can pass when the framework changes.
  - Could assume one thing leading to breaking actual code usage. Ouch. :(
- Having to mock a lot is a sign of architecture issues.

Mocking Drupal

"You're crazy."

—webchick (June, 2015)

Mocking Drupal

- Entity
- Plugin
- Database
- Form
- Other services

Mocking Entities

- Mock low-level classes that entities use:
  - Config Entity
    - Config Storage: loadMultiple, load
    - Config: The immutable config is useful to mock for config forms.

Key Module

Drupal\Tests\key\KeyTestBase

```
    // Mock the Config object, but methods will be mocked in the test class.
    $this->config = $this->getMockBuilder('\Drupal\Core\Config\ImmutableConfig')
      ->disableOriginalConstructor()
      ->getMock();

    // Mock the ConfigFactory service.
    $this->configFactory = $this->getMockBuilder('\Drupal\Core\Config
\ConfigFactory')
      ->disableOriginalConstructor()
      ->getMock();
    $this->configFactory->expects($this->any())
      ->method('get')
      ->with('key.default_config')
      ->willReturn($this->config);

    // Mock ConfigEntityStorage object, but methods will be mocked in the test
class.
    $this->configStorage = $this->getMockBuilder('\Drupal\Core\Config\Entity
\ConfigEntityStorage')
      ->disableOriginalConstructor()
      ->getMock();
```

```
    // Mock EntityManager service.
    $this->entityManager = $this->getMockBuilder('\Drupal\Core\Entity
\EntityManager')
      ->disableOriginalConstructor()
      ->getMock();
    $this->entityManager->expects($this->any())
      ->method('getStorage')
      ->with('key')
      ->willReturn($this->configStorage);

    // Create a dummy container.
    $this->container = new ContainerBuilder();
    $this->container->set('entity.manager', $this->entityManager);
    $this->container->set('config.factory', $this->configFactory);

    // Each test class should call \Drupal::setContainer() in its own setUp
    // method so that test classes can add mocked services to the container
    // without affecting other test classes.
}
```

Key Module

Drupal\Tests\key\Entity\KeyEntityTest

```
protected function setUp() {
    parent::setUp();

    $definition = [
      'id' => 'config',
      'title' => 'Configuration',
      'storage_method' => 'config'
    ];
    $this->key_settings = ['key_value' => $this->createToken()];
    $plugin = new ConfigKeyProvider($this->key_settings, 'config', $definition);

    // Mock the KeyProviderPluginManager service.
    $this->KeyProviderManager = $this->getMockBuilder('\Drupal\key\KeyProviderPluginManager')
      ->disableOriginalConstructor()
      ->getMock();

    $this->KeyProviderManager->expects($this->any())
      ->method('getDefinitions')
      ->willReturn([
        ['id' => 'file', 'title' => 'File', 'storage_method' => 'file'],
        ['id' => 'config', 'title' => 'Configuration', 'storage_method' => 'config']
      ]);
    $this->KeyProviderManager->expects($this->any())
      ->method('createInstance')
      ->with('config', $this->key_settings)
      ->willReturn($plugin);
    $this->container->set('plugin.manager.key.key_provider', $this->KeyProviderManager);

    \Drupal::setContainer($this->container);
}
```

```php
public function testGetters() {
  // Create a key entity using Configuration key provider.
  $values = [
    'key_id' => $this->getRandomGenerator()->word(15),
    'key_provider' => 'config',
    'key_settings' => $this->key_settings,
  ];
  $key = new Key($values, 'key');

  $this->assertEquals($values['key_provider'], $key->getKeyProvider());
  $this->assertEquals($values['key_settings'], $key->getKeySettings());
  $this->assertEquals($values['key_settings']['key_value'], $key->getKeyValue());
}
```

Mocking Entities

- Mock low-level classes that entities use:
    - Content Entity
    - Entity Manager: getDefinition, getStorage, getFieldDefinitions
    - Entity Storage: loadMultiple, load
    - Node: getTitle, get

Mocking Drupal Module

Drupal\Tests\mockingdrupal\Form\MockingDrupalFormTest

```php
class MockingDrupalFormTest extends FormTestBase {
  protected function setUp() {
    parent::setUp();

    $this->node_title = $this->getRandomGenerator()->word(10);
    $this->node = $this->getMockBuilder('Drupal\node\Entity\Node')
      ->disableOriginalConstructor()
      ->getMock();
    $this->node->expects($this->any())
      ->method('getTitle')
      ->will($this->returnValue($this->node_title));

    $this->nodeStorage = $this->getMockBuilder('Drupal\node\NodeStorage')
      ->disableOriginalConstructor()
      ->getMock();
$this->nodeStorage->expects($this->any())
  ->method('load')
  ->will($this->returnValueMap([
    [1, $this->node],
    [500, NULL],
  ]));

$entityManager = $this->getMockBuilder('Drupal\Core\Entity\EntityManagerInterface')
  ->disableOriginalConstructor()
  ->getMock();
$entityManager->expects($this->any())
  ->method('getStorage')
  ->with('node')
  ->willReturn($this->nodeStorage);
```

Mocking Plugins

- No mock necessary: instantiate the plugin class depends on plugin type:
    - Create via initialize (__construct) with definition array and any additional settings for the plugin type.
- Mock the plugin manager for the plugin type, if necessary
    - Typed Data Manager requires getDefinitions to be consecutively mocked if dealing with composite data types.

Key Module

Drupal\Tests\key\KeyRepositoryTest

```php
public function defaultKeyContentProvider() {
  $defaults = ['key_value' => $this->createToken()];
  $definition = [
    'id' => 'config',
    'class' => 'Drupal\key\Plugin\KeyProvider\ConfigKeyProvider',
    'title' => 'Configuration',
  ];
  $KeyProvider = new ConfigKeyProvider($defaults, 'config', $definition);

  return [
    [$defaults, $KeyProvider]
  ];
}
```

Xero Module

Drupal\Tests\xero\Plugin\DataType\TestBase

```
  public function setUp() {
    // Typed Data Manager setup.
    $this->typedDataManager = $this->getMockBuilder('\Drupal\Core\TypedData
\TypedDataManager')
      ->disableOriginalConstructor()
      ->getMock();

    $this->typedDataManager->expects($this->any())
      ->method('getDefinition')
      ->with(static::XERO_TYPE, TRUE)
      ->will($this->returnValue(['id' => ‘xero_employee', 'definition class'
=> ‘\Drupal\xero\TypedData\Definitions\EmployeeDefinition’]));

    // Snip... ... ... ... ... ... ... ... ... ... ... ...

    // Mock the container.
    $container = new ContainerBuilder();
    $container->set('typed_data_manager', $this->typedDataManager);
    \Drupal::setContainer($container);

    // Create data definition
    $definition_class = static::XERO_DEFINITION_CLASS;
    $this->dataDefinition = $definition_class::create(static::XERO_TYPE);
  }
```

Mocking Database

- Mock Drupal\Tests\Core\Database\Stub\StubPDO
    - Then use Drupal\Core\Database\Connection
        - Use Prophecy to mock the Connection class and pass it to the classes you're testing that need it.
        - Or use KernelTestBase and add the database service to test on an actual database.

https://phpunit.de/manual/current/en/test-doubles.html#test-doubles.prophecy

Drupal Core

Drupal\Tests\Core\Database\Driver\pgsql\PostgresqlConnectionTest

```
  protected function setUp() {
    parent::setUp();
    $this->mockPdo = $this->getMock('Drupal\Tests\Core\Database\Stub
\StubPDO');
  }

  /**
   * @covers ::escapeTable
   * @dataProvider providerEscapeTables
   */
  public function testEscapeTable($expected, $name) {
    $pgsql_connection = new Connection($this->mockPdo, []);

    $this->assertEquals($expected, $pgsql_connection->escapeTable($name));
}

/**
 * @covers ::escapeAlias
 * @dataProvider providerEscapeAlias
 */
public function testEscapeAlias($expected, $name) {
  $pgsql_connection = new Connection($this->mockPdo, []);

  $this->assertEquals($expected, $pgsql_connection->escapeAlias($name));
}
```

Drupal Core

Drupal\Tests\Core\Database\ConditionTest

```
public function testSimpleCondition() {
    $connection = $this->prophesize(Connection::class);
    $connection->escapeField('name')->will(function ($args) {
      return preg_replace('/[^A-Za-z0-9_.]+/', '', $args[0]);
    });
    $connection->mapConditionOperator('=')->willReturn(['operator' => '=']);
    $connection = $connection->reveal();

    $query_placeholder = $this->prophesize(PlaceholderInterface::class);

    $counter = 0;
    $query_placeholder->nextPlaceholder()->will(function() use (&$counter) {
      return $counter++;
    });
    $query_placeholder->uniqueIdentifier()->willReturn(4);
    $query_placeholder = $query_placeholder->reveal();

    $condition = new Condition('AND');
    $condition->condition('name', ['value']);
    $condition->compile($connection, $query_placeholder);

    $this->assertEquals(' (name = :db_condition_placeholder_0) ',
$condition->__toString());
    $this->assertEquals([':db_condition_placeholder_0' => 'value'],
$condition->arguments());
```

Mocking Forms

- FormTestBase is pretty useless, but it is there.
- How useful is testing a form in phpunit?
- FormBuilder requires complex Request mocking, and it is not possible to simply pass in FormState with values set.
- This means that a form needs to be very careful to follow API in all that it does and the expectation is that the form knows everything about Drupal form builder input.

Mocking Drupal Module

Drupal\mockingdrupal\Form\MockingDrupalForm

```php
$form['node_id'] = [
    '#type' => 'number',
    '#title' => $this->t('Node id'),
    '#description' => $this->t('Provide a node id.'),
    '#min' => 1,
    '#required' => TRUE,
  ];

  $form['actions'] = ['#type' => 'actions'];
  $form['actions']['submit'] = [
    '#type' => 'submit',
    '#value' => $this->t('Display'),
  ];

  if ($form_state->getValue('node_id', 0)) {
    try {
      $node = $this->entityManager->getStorage('node')->load($form_state->getValue('node_id',
0));
      if (!isset($node)) {
        throw new \Exception;
      }
      $form['node'] = [
        '#type' => 'label',
        '#label' => $node->getTitle(),
      ];
    }
    catch (\Exception $e) {
      $this->logger->error('Could not load node id: %id', ['%id' => $form_state-
>getValue('node_id', 0)]);
    }
```

Mocking Drupal Module

Drupal\Tests\mockingdrupal\Form\MockingDrupalFormTest

```php
protected function setUp() {
  // Set the container into the Drupal object so that Drupal can call the
  // mocked services.
  $container = new ContainerBuilder();
  $container->set('entity.manager', $entityManager);
  $container->set('logger.factory', $loggerFactory);
  $container->set('string_translation', $this->stringTranslation);
  \Drupal::setContainer($container);

  // Instantiatie the form class.
  $this->form = MockingDrupalForm::create($container);
}
```

```php
public function testBuildForm() {
  $form = $this->formBuilder->getForm($this->form);

  $this->assertEquals('mockingdrupal_form', $form['#form_id']);

  $state = new FormState();
  $state->setValue('node_id', 1);

  // Fresh build of form with no form state for a value that exists.
  $form = $this->formBuilder->buildForm($this->form, $state);
  $this->assertEquals($this->node_title, $form['node']['#label']);

  // Build the form with a mocked form state that has value for node_id
that
  // does not exist i.e. exception testing.
  $state = new FormState();
  $state->setValue('node_id', 500);
  $form = $this->formBuilder->buildForm($this->form, $state);
  $this->assertArrayNotHasKey('node', $form);
}
```

```
public function testFormValidation() {
  $form = $this->formBuilder->getForm($this->form);
  $input = [
    'op' => 'Display',
    'form_id' => $this->form->getFormId(),
    'form_build_id' => $form['#build_id'],
    'values' => ['node_id' => 500, 'op' => 'Display'],
  ];

  $state = new FormState();
  $state
    ->setUserInput($input)
    ->setValues($input['values'])
    ->setFormObject($this->form)
    ->setSubmitted(TRUE)
    ->setProgrammed(TRUE);

  $this->form->validateForm($form, $state);
```

```
    $errors = $state->getErrors();
    $this->assertArrayHasKey('node_id', $errors);
    $this->assertEquals('Node does not exist.',
\PHPUnit_Framework_Assert::readAttribute($errors['node_id'], 'string'));

    $input['values']['node_id'] = 1;
    $state = new FormState();
    $state
      ->setUserInput($input)
      ->setValues($input['values'])
      ->setFormObject($this->form)
      ->setSubmitted(TRUE)
      ->setProgrammed(TRUE);

    $this->form->validateForm($form, $state);
    $this->assertEmpty($state->getErrors());
 }
```

Mocking Other Things

- Guzzle
  - Provides MockHandler that can be added to the handler stack before initiating a connection.
    - Applications need to be able to pass the client object around.

mradcliffe/XeroBundle

BlackOptic\Bundle\XeroBundle\Tests\XeroClientTest

```php
public function testGetRequest() {
    $mock = new MockHandler(array(
            new Response(200, array('Content-Length' => 0))
    ));
    $this->options['handler'] = HandlerStack::create($mock);
    $this->options['private_key'] = $this->pemFile;

    $client = new XeroClient($this->options);

    try
    {
        $client->get('Accounts');
    }
    catch (RequestException $e)
    {
        $this->assertNotEquals('401', $e->getCode());
    }

    $this->assertEquals('/api.xro/2.0/Accounts', $mock
       ->getLastRequest()->getUri()->getPath());
}
```

Test Automation

- TravisCI
    - Can setup build matrices with all PHP versions.
- DrupalTI
    - Can setup Drupal on TravisCI for simpletest, behat, and phpunit tests.
- DrupalCI
    - Official infrastructure has supported databases and PHP versions, but harder to get dependencies via composer.

TravisCI Configuration

- Requires download Drupal
    - Move test directory inside of Drupal's module directory.
- Some quirks
    - Currently composer is horribly broken because of GitHub API limits, but this does not apply to private Travis instances.

```
language: php

php:
  - 5.5
  - 5.6

sudo: false

install:
  - composer self-update

  - TESTDIR=$(pwd)
  - cd ..
  - git clone --branch 8.0.x --depth 1 http://git.drupal.org/project/
drupal.git drupal
  - cd drupal

before_script:
  - rsync -rtlDPvc --exclude .git/ $TESTDIR modules/

  - cd modules/MY_MODULE_NAME
script:
  - ../../vendor/bin/phpunit --coverage-text=$TRAVIS_BUILD_DIR/coverage.txt
after_script:
  - head $TRAVIS_BUILD_DIR/coverage.txt
```

DrupalTI Configuration

- drupalti is a suite of bash scripts for setting up a Drupal environment on TravisCI.
- Must setup database and install Drupal inside of Travis.
    - But I have an issue/pull request to allow for phpunit only.
- Supports behat as well.
- Same issue with Composer and GitHub API limits.

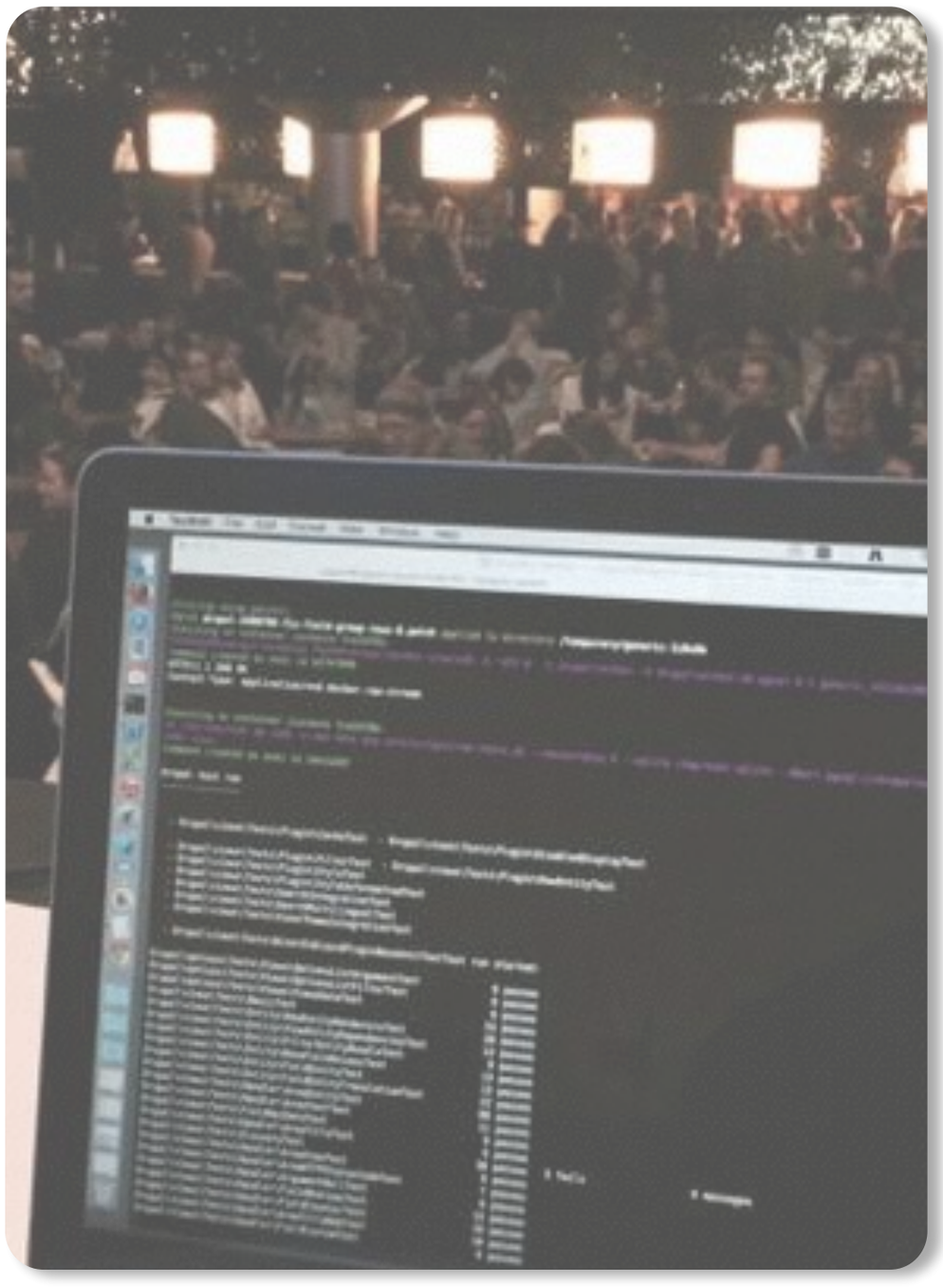
```
env:
  global:
    - PATH="$PATH:$HOME/.composer/vendor/bin"
    - DRUPAL_TI_MODULE_NAME="key"
    - DRUPAL_TI_SIMPLETEST_GROUP="key"
    - DRUPAL_TI_DB="drupal_travis_$$"
    - DRUPAL_TI_DB_URL="mysql://root@127.0.0.1/$DRUPAL_TI_DB"
    - DRUPAL_TI_WEBSERVER_URL="http://127.0.0.1"
    - DRUPAL_TI_WEBSERVER_PORT="8080"
    - DRUPAL_TI_SIMPLETEST_ARGS="--verbose --color --url $DRUPAL_TI_WEBSERVER_URL:
$DRUPAL_TI_WEBSERVER_PORT"
    - DRUPAL_TI_PHPUNIT_CORE_SRC_DIRECTORY="./tests/src"
    - DRUPAL_TI_ENVIRONMENT="drupal-8"

  matrix:
    - DRUPAL_TI_RUNNERS="simpletest phpunit-core"

before_install:
  - composer self-update
  - composer global require "lionsad/drupal_ti:1.*"
  - drupal-ti before_install

install:
  - drupal-ti install

before_script:
  - drupal-ti before_script
  - DRUPAL_TI_PHPUNIT_ARGS="-c $DRUPAL_TI_DRUPAL_DIR/modules/key/phpunit.xml --coverage-text"

script:
  - drupal-ti script

after_script:
  - drupal-ti after_script
```

DrupalCI Configuration

- drupalci is the new testing infrastructure on drupal.org which leverages a Docker to create all the build environments we want to test.
    - Docker
    - project: drupalci_testbot
    - Recommended to run on actual hardware rather than in Vagrant.
    - Supports Drupal 7, 8.
    - Needs custom work to support modules with composer dependencies.

DrupalCI Configuration

- Start docker
- Configure job file (YAML)
- Set job properties (ENV)
  - drupalci config:set



```
environment:
  db:
    - %DCI_DBVersion%
  web:
    - %DCI_PHPVersion%
setup:
  checkout:
   - protocol: local
     source_dir: /home/mradcliffe/dev/www/drupal8.dev
     branch: 8.0.x
    #    - protocol: git
    #      repo: %DCI_CoreRepository%
    #      branch: %DCI_CoreBranch%
    #      depth: %DCI_GitCheckoutDepth%
    #      checkout_dir: .
mkdir:
    - /var/www/html/results
    - /var/www/html/artifacts
    - /var/www/html/sites/simpletest/xml
  command:
    - chown -fR www-data:www-data /var/www/html/sites /var/www/html/results
    - chmod 0777 /var/www/html/artifacts
install:
execute:
  command:
    - cd /var/www/html && sudo -u www-data php /var/www/html/core/scripts/run-tests.sh --color --
sqlite /var/www/html/artifacts/test.sqlite --concurrency %DCI_Concurrency% --keep-results --xml /
var/www/html/artifacts/xml --dburl %DCI_DBurl% %DCI_TestGroups%
publish:
  gather_artifacts: /var/www/html/artifacts
  archive: /var/www/html/results/artifacts.zip
```

DrupalCI Env. Variables

- DCI_PHPVersion: *5.5*, *5.6*
- DCI_DBVersion: mysql-*5.5*, pgsql-9.*1*, sqlite-3.8
- DCI_Fetch: https://www.drupal.org/files/issues/drupal-2572283-transaction-isolation-level-*1*2.patch,.
- DCI_Patch: drupal-2572283-transaction-isolation-level-*1*2.patch,.
- DCI_TestGroups: Database

Summary

- Overview of unit testing
- PHPUnit and Drupal 8