shoptimax
Ihr E-Commerce-Experte

OXID (6) CLI

OXRUN

ÜBER MICH

- Software-/DevOps Engineer (PHP, JavaScript, Java, Docker, Ansible, Gitlab, …)
- IT / E-Commerce seit 1999
- > 12 Jahr OXID Erfahrung
- @upsettweety
- www.shoptimax.de
- moises@shoptimax.de

shoptimax
Ihr E-Commerce-Experte

UND IHR SO?

- Erfahrung mit OXID6?
- Erfahrung mit oxrun?
- Erfahrung mit Oxid Console?
- Erfahrung mit Symfony (CLI)?

AGENDA

1. Was ist oxrun?
2. Anwendungsbeispiele
3. Was war vorher?
4. Oxid Console
5. Installation
6. oxrun Kommandos
7. Eigene Kommandos erstellen
8. Tests

WAS IST OXRUN?

```
Available commands:
  help                          Displays help for a command
  list                          Lists commands
 cache
  cache:clear                   Clears the cache
 cms
  cms:update                    Updates a cms page
 config
  config:export                 Export shop config
  config:get                    Gets a config value
  config:import                 Import shop config
  config:multiset               Sets multiple config values from yaml file
  config:set                    Sets a config value
  config:shop:get               Sets a shop config value
  config:shop:set               Sets a shop config value
 db
  db:dump                       Dumps the the current shop database
  db:import                     Import a sql file
  db:list                       List of all Tables
  db:query                      Executes a query
 install
```

WAS IST OXRUN?

- OXRUN bietet ein CLI toolset für den OXID eShop
- basiert auf **Symfony** CLI (aktuell auf v2.8)
- inspiriert von „netz98 **magerun**“ für Magento
- CLI Befehle für Cache leeren, Modul-Aktivierung, Datenbank-Administration, Konfigurations-Änderungen etc.

WAS IST OXRUN?

- ursprünglich entwickelt von Marc Harding (marcharding) für OXID CE 4.x
- geforked, aktualisiert und erweitert für **OXID 6** und **OXID EE** von Tobias Matthaiou (tmloberon) und Stefan Moises (smxsm)

OXRUN ANWENDUNGS-BEISPIELE

- Aufgaben per CLI in der **lokalen Entwicklungsumgebung** (Vagrant, Docker) ausführen, z.B. Module aktivieren, oxconfig Einstellungen anpassen, Dumps einspielen, tmp leeren, …
- Aufgaben während des **Deployments** durchführen (Views erzeugen, Cache leeren, Module aktivieren, …)
- **Remote OXID Administration** über SSH Kommandos
- Kommandos via **cron** ausführen
- Zusammenspiel mit **Composer**, als „Composer Script“

OXRUN MIT COMPOSER

```
  },
  "scripts": {
    "post-install-cmd": [
      "Incenteev\\ParameterHandler\\ScriptHandler::buildParameters",
      "@oe:ide-helper:generate",
      "@oxrun:activate-modules",
      "@oxrun:set-config"
    ],
    "post-update-cmd": [
      "Incenteev\\ParameterHandler\\ScriptHandler::buildParameters",
      "@oe:ide-helper:generate",
      "@oxrun:activate-modules",
      "@oxrun:set-config"
    ],
    "oxrun:activate-modules": [
      "./vendor/bin/oxrun module:multiactivate configs/modules.yml -c --shopDir=./source"
    ],
    "oxrun:set-config": [
      "./vendor/bin/oxrun config:multiset configs/malls.yml --shopDir=./source"
    ],
    "oe:ide-helper:generate": [
      "if [ -f ./vendor/bin/oe-eshop-ide_helper ]; then oe-eshop-ide_helper; fi"
    ]
  },
```

RÜCKBLICK: VOR OXRUN UND OXID 6

- bis OXID 4.10 / 5.3 haben wir den „ioly“ module manager genutzt, um automatisch Module zu installieren und zu aktivieren
- für andere Aufgaben, z.B. bei Deployments oder um lokale Umgebungen aufzusetzen, nutzten wir die OXID Console (via ioly installiert) um z.B. den Cache zu leeren, DB-Migrationen auszuführen usw.

DANN KAM OXID 6

- mit OXID 6 kam volle **Composer** Integration, sowohl für Shop-als auch für Modul-Installationen
- **ioly** wurde dadurch für die Modul-Installation überflüssig
- die Zukunft für die *OXID Console* war ungewiss
- **oxrun** bot sich als relativ lose gekoppelte Alternative an
- während des OXID Hackaton 2017 erste Tests mit OXID 6

WAS IST NUN MIT...

OXID CONSOLE?

OXID CONSOLE

- alternatives CLI für den OXID eShop
- Vorteile: hat DB-Migrations, „Module Scaffold“
- Nachteile: dennoch weniger Kommandos und Funktionalität, basiert nicht auf Symfony CLI, unklare Zukunft vor/bei OXID 6 Release
- ausserdem hat OXID 6 jetzt **DB-Migrations** :)
- die Zeichen stehen also gut für oxrun :)

LOS GEHT'S ...

INSTALLATION

INSTALLATION

- composer config repositories.smxsm/oxrun vcs https://github.com/smxsm/oxrun
- composer require --dev --no-scripts --no-update smxsm/oxrun:dev-develop
- composer update -n -vv

```
"smxsm/oxrun": {
  "type": "vcs",
  "url": "https://github.com/smxsm/oxrun.git"
},
```

```
"require-dev": {
  "oxid-esales/testing-library": "^v4.0.0-beta.1",
  "squizlabs/php_codesniffer": "^3.2.0",
  "incenteev/composer-parameter-handler": "~v2.0",
  "oxid-esales/oxideshop-ide-helper": "^v3.0.0",
  "oxid-esales/azure-theme": "^v1.4.1",
  "symfony/yaml": "~2.8",
  "smxsm/oxrun": "dev-develop"
},
```

„Start your engines.“

–INSTALL DEMO

LETS GET STARTED ...

OXRUN KOMMANDOS

OXRUN COMMANDS

- um den S**hop Kontext** zu haben, sollten die Kommandos immer aus dem „source“ Verzeichnis ausgeführt werden, oder mit dem Parameter „--shopDir=./source“ z.B.

- „../vendor/bin/oxrun“ gibt eine Liste aller verfügbaren Kommandos aus

- „../vendor/bin/oxrun help <COMMAND>“ gibt Hilfetext für das entsprechende Kommando aus

- alternativ kann „oxrun.phar“ genutzt werden

MODUL KOMMANDOS

- module:list - listet Subshop-Module auf, z.B. ../vendor/bin/oxrun module:list --shopId=9

```
+--------------------------+--------+
| Module                   | Active |
+--------------------------+--------+
| bestitamazonpay4oxid     | yes    |
| ddoevisualcms            | yes    |
| ddoewysiwyg              | yes    |
| fcpayone                 | yes    |
| gs_environmentinformation | yes    |
| gs_securepassword        | yes    |
| gsGTM                    | no     |
```

- module:(de)activate - de-/activiert Subshop Module, z.B. ../vendor/bin/oxrun module:activate oepaypal --shopId=1

```
Module oepaypal already activated for shopId 1.
```

MODUL KOMMANDOS

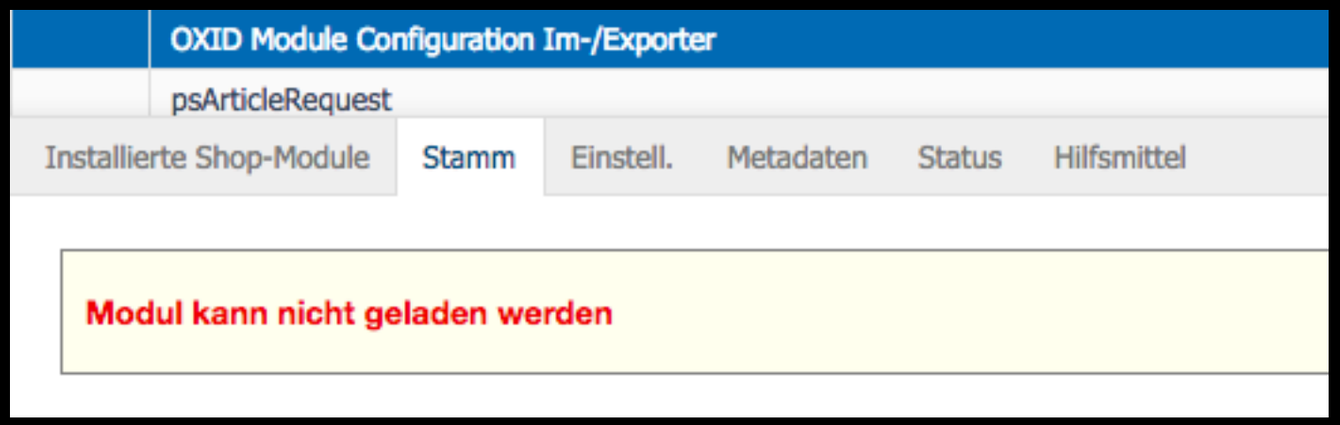
- module:multiactivate - mehrere Module auf einmal aktivieren (in versch. Subshops) via YAML Konfiguration, z.B..

  ../vendor/bin/oxrun module:multiactivate modules.yml --shopId=2

  Man kann hier mit B*lack- oder Whitelist* arbeiten, der Pfad zur YAML-Datei ist relativ zum „source-„Ordner des Shops:

```yaml
blacklist:
  1:
    - oepaypal
    - bestitamazonpay4oxid
    - oxpspaymorrow
    - toxid_curl
  2:
    - oepaypal
    - bestitamazonpay4oxid
    - toxid_curl
    - fcpayone
    - psArticleRequest
```

MODUL KOMMANDOS



- falscher Pfad (Groß-/Kleinschreibung) in aModulePaths => Modul ist deaktiviert und kann nicht mehr aktiviert werden :(

MODUL KOMMANDOS

- ../vendor/bin/oxrun module:fix oxpsmodulesconfig --shopId=4 -r

```
[DEBUG] Working on shop id 6
+-------------------+
| Disabled Modules  |
+-------------------+
| psArticleRequest  |
| psBlocks          |
| toxid_curl        |
| oxpsmodulesconfig |
+-------------------+
[INFO] Module oxpsmodulesconfig removed from aDisabledModules
```

```
| ocb_cleartmp          | oxcom/ocbcleartmp          |
| oxpsmodulesconfig     | oxps/modulesconfig         |

+-----------------------+----------------------------
[INFO] Module oxpsmodulesconfig removed from aModulePaths
```

- mit der option -r („reset“) können „kaputte“ Module zurückgesetzt werden (falsche Pfade etc.)
- Fix für alle Module in Subshop: ../vendor/bin/oxrun module:fix --shopId=1 --all

DATENBANK KOMMANDOS

- ../vendor/bin/oxrun db:query "select count(*) from oxarticles“

```
+----------+
| count(*) |
+----------+
| 491      |
+----------+
```

- ../vendor/bin/oxrun db:list --pattern oxseo%,oxuser

```
+-------------+------------+
| Table       | Type       |
+-------------+------------+
| oxseo       | BASE TABLE |
| oxseohistory | BASE TABLE |
| oxseologs   | BASE TABLE |
| oxuser      | BASE TABLE |
+-------------+------------+
```

- ../vendor/bin/oxrun db:dump --table %user%,%order% --file=myuserdump.sql -v

```
-- Dump Tables ...
-- mysqldump -u 'oxid' -h 'mysql' -p --force --quick --opt --hex-blob --default-character-set=utf8  app 'oepaypal_order' 'oe
oxorderarticles' 'oxorderfiles' 'oxuser' 'oxuserbasketitems' 'oxuserbaskets' 'oxuserpayments' > myuserdump.sql
Dump myuserdump.sql created.
```

CONFIG KOMMANDOS

- ../vendor/bin/oxrun config:get sShopDir
  ../vendor/bin/oxrun config:set --variableType=str foo bar -- moduleId=module:mymodule --shopId=2

- ../vendor/bin/oxrun config:multiset configs.yml --shopId=2

```yaml
config:
  1:
    sMallShopURL: http://oxid6.dev.local
    sMallSSLShopURL: http://oxid6.dev.local
    shoptifindHost: shoptifind-oxid6
    shoptifindPort: 8080
    shoptifindCoreStatusUrl: /solr/admin/cores
    shoptifindAppPath:      You, 3 months ago • GBA-56 us
      variableType: aarr
      variableValue:
        - /solr/core-de/
        - /solr/core-en/
    OXPS_MODULESCONFIG_SETTING_CONFIGURATION_DIRECTORY:
      variableType: str
      variableValue: "../../../configs/configurations"
      moduleId: "module:oxpsmodulesconfig"
  2:
```

WEITERE KOMMANDOS

- ../vendor/bin/oxrun cache:clear - Leert den Shop Cache
- ../vendor/bin/oxrun views:update - generiert die DB Views neu
- ../vendor/bin/oxrun db:anonymize --domain=@oxrun.com --keepdomain=@shoptimax.de
  anonymisiert benutzerspezifische Tabellen, z.B. oxuser, oxaddress, oxorder usw.
- ../vendor/bin/oxrun log:exceptionlog --lines=10 -t -f ArgumentCountError
  zeigt die letzten 10 Zeilen des EXCEPTION_LOG.txt an, gefiltert nach Einträgen mit dem String „ArgumentCountError“
- ../vendor/bin/oxrun user:password foo@bar.com secret - setzt ein neues Benutzer-Passwort
- „Remote Modul-Liste“ :-)
  ssh user@server.de "cd /var/www/vhosts/projects/current/; ./vendor/bin/oxrun module:list --shopId=1 --shopDir=./source“

EIGENE KOMMANDOS


```php
/**
 * Class PasswordCommand
 * @package Oxrun\Command\User
 */
class PasswordCommand extends Command
{

    /**
     * Configures the current command.
     */
    protected function configure()
    {
        $this
            ->setName('user:password')
            ->setDescription('Sets a new password')
            ->addArgument('username', InputArgument::REQUIRED, 'Username')
            ->addArgument('password', InputArgument::REQUIRED, 'New password');
    }

    /**
     * Executes the current command.
     *
     * @param InputInterface $input An InputInterface instance
     * @param OutputInterface $output An OutputInterface instance
     */
    protected function execute(InputInterface $input, OutputInterface $output)
    {
        $oxUser = \oxNew(\OxidEsales\Eshop\Application\Model\User::class);

        $sql = sprintf(
            "SELECT `oxuser`.`OXID` FROM `oxuser` WHERE `oxuser`.`OXUSERNAME` = %s",
            \OxidEsales\Eshop\Core\DatabaseProvider::getDb()->quote($input->getArgument('username'))
        );
        $userOxid = \OxidEsales\Eshop\Core\DatabaseProvider::getDb()->getOne($sql);
        if (empty($userOxid)) {
            $output->writeln('<error>User does not exist.</error>');
            return;
        }
        $oxUser->load($userOxid);
        $oxUser->setPassword($input->getArgument('password'));
        $oxUser->save();
        $output->writeln('<info>New password set.</info>');
    }

    /**
     * @return bool
     */
    public function isEnabled()
    {
        return $this->getApplication()->bootstrapOxid();
    }
}
```

EIGENE KOMMANDOS

- oxrun Repo forken
- in *oxrun/src/Oxrun/Command/* (ggf. neues Verzeichnis und) neue PHP-Datei anlegen
- die Datei muss mit „…Command.php“ enden, damit das Kommando automatisch geladen wird!
- extend Symfony\Component\Console\Command\Command class
- Methoden „configure“, „execute“ und „isEnabled“ Methoden implementieren
- und am besten auch eine Test-Klasse dafür schreiben :)

EIGENE KOMMANDOS

```php
/**
 * Configures the current command.
 */
protected function configure()
{
    $this
        ->setName('db:query')
        ->setDescription('Executes a query')
        ->addArgument('query', InputArgument::REQUIRED, 'The query which is to be executed')
        ->addOption('raw', null, InputOption::VALUE_NONE, 'Raw output');

        $help = <<<HELP
Executes an SQL query on the current shop database. Wrap your SQL in quotes.

If your query produces a result (e.g. a SELECT statement), the output will be returned via the tab

Requires php exec and MySQL CLI tools installed on your system.
HELP;
        $this->setHelp($help);
    }
```

ANDERE KOMMANDOS IN EIGENEN KLASSEN NUTZEN

- man kann auch andere Kommandos direkt in eigenen Klassen benutzen, z.B.

```php
protected function execute(InputInterface $input, OutputInterface $output)
{
    /** @var \Oxrun\Application $app */
    $app = $this->getApplication();
    $shopId = $input->getOption('shopId');
    if ($shopId) {
        $app->switchToShopId($shopId);
    }
    $app->find('module:deactivate')->run($input, $output);
    $app->find('cache:clear')->run(new ArgvInput([]), $output);
    $app->find('module:activate')->run($input, $output);
}
```

DER SYMFONY TABLE HELPER

oxid	oxusername	oxfname	oxlname
500f6a17d54f6bc593bba99cc1144b03	0c9140459cb6e3c8404961d0f16069c2@oxrun.com		
oxdefaultadmin	prog@shoptimax.de	John	Doe

- das Symfony CLI hat ein nettes „Table Layout“, welches man für die Anzeige nutzen kann:

```php
$deactiveModules = array_map(
    function ($item) {
        return array($item, 'no');
    },
    $deactiveModules
);

$table = new Table($output);
$table
    ->setHeaders(array('Module', 'Active'))
    ->setRows(array_merge($activeModules, $deactiveModules));
$table->render();
```

LIVE-DEMO - INTERAKTIVES KOMMANDO

- wir werden jetzt das erste interaktive oxrun Kommando entwickeln - „CreateUserCommand“! :-)

- dazu nutzen wir den Symfony CLI QuestionHelper, z.B.

```
    use Symfony\Component\Console\Question\Question;
    use Symfony\Component\Console\Question\ConfirmationQuestion;

    public function execute(InputInterface $input, OutputInterface $output)
    {
        $helper = $this->getHelper('question');
        $question = new Question('Please enter the name of the bundle', 'AcmeDemoBundle');
        $bundleName = $helper->ask($input, $output, $question);
        $question = new ConfirmationQuestion(
            'Continue with this action?',
            false,
            '/^(y|j)/i'
        );
        if (!$helper->ask($input, $output, $question)) {
            return;
        }
    }
```

Ideen / Anregungen für neue Kommandos?

TESTS SCHREIBEN

- für jedes Kommando sollte es einen entsprechenden Test geben
- Tests sind relativ einfach zu implementieren mithilfe von Symfony\Component\Console\Tester\CommandTester

```php
class DeactivateCommandTest extends TestCase
{
    public function testExecute()
    {
        $app = new Application();
        $app->add(new DeactivateCommand());

        $command = $app->find('module:deactivate');

        $commandTester = new CommandTester($command);
        $commandTester->execute(
            array(
                'command' => $command->getName(),
                'module' => 'oepaypal'
            )
        );

        $this->assertContains('Module oepaypal deactivated', $commandTester->getDisplay());
    }
}
```

TESTS AUSFÜHREN

- die Unit-Tests **benötigen einen konfigurierten Shop inkl. Datenbank**. Um die Tests zu starten, im Verzeichnis "source" in OXID 6 folgenden Befehl ausführen, inkl. korrektem Pfad zum "oxrun" vendor Ordner, z.B..:

- ../vendor/bin/phpunit /var/www/html/oxid6/vendor/smxsm/oxrun/

README.MD AKTUALISIEREN

- Man kann die Doku für alle Kommandos mit ../vendor/bin/oxrun misc:generate:documentation > commands.txt
  erzeugen und damit in „source/commands.txt“ speichern

- den Inhalt von „commands.txt" kann man dann in README.md an die passende Stelle kopieren :)

OXRUN.PHAR ERZEUGEN

- um die Phar-Datei zu erzeugen, im Root-Verzeichnis des GIT Repositories

  php build

  ausführen

- vorher aber sicherstellen, dass

  phar.readonly = Off

  in der verwendeten php.ini Datei gesetzt ist, sonst erzeugt PHP aus Sicherheitsgründen keine PHAR-Dateien!

DIE ZUKUNFT

- Integration OXID 6 Fork in das Haupt-Repo?
- oxrun selbst forken und eigene Kommandos entwickeln
- vorhandene Kommandos verbessern, Pull Requests stellen

LINKS

- OXID Console: https://github.com/OXIDprojects/oxid-console/tree/2.0
- Original oxrun: https://github.com/marcharding/oxrun
- OXID 6 Fork: https://github.com/smxsm/oxrun
- Magerun: https://github.com/netz98/n98-magerun
- ioly: https://github.com/ioly/ioly
- Composer: https://getcomposer.org/
- Symfony CLI: https://symfony.com/doc/2.8/components/console/
- Symfony Questionhelper: https://symfony.com/doc/2.8/components/console/helpers/questionhelper.html