GUIDELINES FOR CREATING A DEDICATED DRUPAL CMS MODULE FOR TWO-WAY INTEGRATION AT HDFS LEVEL

СЪЗДАВАНЕ НА СПЕЦИАЛИЗИРАН DRUPAL CMS МОДУЛ ЗА ДВУПОСОЧНА ИНТЕГРАЦИЯ НА РАВНИЩЕ HDFS

Създаването на персонализиран Drupal модул за интеграция на четене и запис (RW) с разпределената файлова система Hadoop (HDFS) включва няколко методически стъпки, които се ръководят както от модулната архитектура на Drupal, така и от принципите на взаимодействие с външни файлови системи. Модулът, който ще наричаме "unwehdfs", ще служи като посредник между Drupal и HDFS, позволявайки безпроблемен обмен на данни и манипулация.

Модулна директория и файлова структура

Първата стъпка е да се създаде директория за модула "unwehdfs" в директорията / модулите на инсталацията на Drupal. В тази директория се поставят основните файлове:

unwehdfs.info.yml: Съдържа метаданни за модула, като име, описание, пакет, тип и зависимости.

unwehdfs.module: Основният модул файл, където административни функции и потребителски функции за интеграция са дефинирани.

unwehdfs.services.yml: Определя услугите, предоставяни от модула, включително тези, които взаимодействат с HDFS.

Деклариране на метаданни на модула

Файлът .info.yml е щателно изработен, за да включва необходимата информация, като гарантира, че Drupal разбира целта и зависимостите на модула.

name: 'UNWEHDFS Integration'

type: module

description: "Осигурява RW интеграция с Hadoop разпределена файлова система.'

package: Custom

core_version_requirement: ^8 || ^9

dependencies:

- drupal:file

Изпълнение на функциите

В рамките на файла unwehdfs.module се реализират Drupal средствата (hooks). Тези средства позволяват на модула да взаимодейства с основните функционалности на Drupal.

Интеграция с HDFS

Действителната интеграция с HDFS се постига чрез API на Hadoop. За да изпълнява RW операции, модулът трябва да може да комуникира с HDFS с помощта на този API. Това често изисква използването на PHP клиент, който може да взаимодейства с HDFS REST API (WebHDFS) или друга подходяща клиентска библиотека на Hadoop.

Дефиниция на услугата за HDFS клиент

В unwehdfs.services.yml е дефинирана услуга, която капсулира HDFS клиента. Дефиницията на услугата гарантира, че HDFS клиентът е за многократна употреба и се придържа към принципите на Drupal за инжектиране на зависимост, като по този начин насърчава модулността и проверимостта.

Клиентският клас HDFS

Класът HDFSClient е създаден за управление на връзката с HDFS и извършване на файлови операции. Този клас е отговорен за обработка на удостоверяването и оторизацията, операциите за четене и запис на файлове, управлението на директории и обработката на грешки.

Класът HDFSClient служи като абстракция между модула Drupal и HDFS системата. Основната му функция е да капсулира всички директни взаимодействия с HDFS, предоставяйки методи за свързване към клъстера

HDFS, удостоверяване и извършване на файлови операции като четене, запис и изтриване. Той превежда HDFS API в PHP методи, които могат лесно да се използват от други компоненти в модула Drupal.

Класът HDFSClient ще трябва да се справи с няколко аспекта на взаимодействието с HDFS:

Обработка на конфигурацията: За извличане и използване на подробни данни за връзката за HDFS.

Управление на връзката: За установяване и поддържане на връзка с HDFS сървъра.

Удостоверяване: За сигурно удостоверяване с услугата HDFS, евентуално с Kerberos, ако клъстерът Hadoop е защитен.

Файлови операции: За изпълнение на операции за четене и запис, списък с директории и изтриване на файлове.

Обработка на грешки: За управление на изключения и грешки, които възникват от HDFS операции.

Регистриране: За да регистрирате операции и грешки за отстраняване на грешки и мониторинг.

```
<?php


namespace Drupal\unwehdfs;


use GuzzleHttp\Client;

use GuzzleHttp\Exception\GuzzleException;

use GuzzleHttp\Psr7\Stream;

use Drupal\Core\Logger\LoggerChannelFactoryInterface;

use Drupal\Core\Site\Settings;
```

```
/**
 * Class HDFSClient.
 *
 * Encapsulates all interactions with the HDFS.
 */
class HDFSClient {
  /**
   * The config factory.
   *
   * @var \Drupal\Core\Config\ConfigFactoryInterface
   */
  protected $configFactory;

  /**
   * The logger factory.
   *
   * @var \Drupal\Core\Logger\LoggerChannelFactoryInterface
   */
  protected $loggerFactory;

  /**
   * Constructs a new HDFSClient object.
   *
   * @param \Drupal\Core\Config\ConfigFactoryInterface $config_factory
   *   The config factory.
```

```
   * @param \Drupal\Core\Logger\LoggerChannelFactoryInterface $logger_factory
   *   The logger factory.
   */
  public function __construct(ConfigFactoryInterface $config_factory, LoggerChannelFactoryInterface $logger_factory) {
    $this->configFactory = $config_factory;
    $this->loggerFactory = $logger_factory;

    $this->initializeHttpClient();
  }

/**
   * Initializes the HTTP client used to communicate with HDFS.
   */
  protected function initializeHttpClient() {
    $config = $this->configFactory->get('unwehdfs.settings');

    $webhdfs_url = $config->get('webhdfs_url');
    $timeout = $config->get('timeout');

    $this->httpClient = new Client([
      'base_uri' => $webhdfs_url,
      'timeout'  => $timeout ?? 30.0,
      'headers' => [
        'Authorization' => 'Bearer ' . $config->get(‘auth_token’)
      ],
```

```
    ]);
}

  /**
   * Writes data to a file on HDFS using a stream.
   *
   * @param string $path
   *   The HDFS path where the data should be written.
   * @param Stream $stream
   *   A stream of data to write.
   *
   * @return bool
   *   TRUE if the operation was successful, FALSE otherwise.
   */
  public function writeFile($path, Stream $stream) {
    try {
      $response = $this->httpClient->put("/webhdfs/v1{$path}?op=CREATE&overwrite=true", [
        'body' => $stream,
        'headers' => [
          'Content-Type' => 'application/octet-stream',
        ],
      ]);

      // Check for successful status code
      return $response->getStatusCode() == 201;
```

```
    } catch (GuzzleException $e) {
      $this->logger->error('Write file to HDFS failed: @message', ['@message' => $e->getMessage()]);
      return false;
    }
  }

  /**
   * Reads data from a file on HDFS and returns a stream.
   *
   * @param string $path
   *   The HDFS file path to read.
   *
   * @return Stream|false
   *   The stream of file contents or FALSE on failure.
   */
  public function readFile($path) {
    try {
      $response = $this->httpClient->get("/webhdfs/v1{$path}?op=OPEN", [
        'stream' => true, // Request a stream to handle large files
      ]);

      return $response->getBody();
    } catch (GuzzleException $e) {
      $this->logger->error('Read file from HDFS failed: @message', ['@message' => $e->getMessage()]);
```

```
    return false;
  }
}


/**
 * Deletes a file or directory from HDFS.
 *
 * @param string $path
 *   The HDFS path to delete.
 * @param bool $recursive
 *   Whether to delete directories recursively.
 *
 * @return bool
 *   TRUE if the file or directory was deleted, FALSE otherwise.
 */
public function delete($path, $recursive = false) {
  try {
    $response = $this->httpClient->delete("/webhdfs/v1{$path}?op=DELETE&recursive=" . ($recursive ? 'true' : 'false'));

    // WebHDFS DELETE returns a JSON object with boolean field "boolean"
    $body = json_decode($response->getBody(), true);
    return $body['boolean'] ?? false;
  } catch (GuzzleException $e) {
```

```
    $this->logger->error('Delete operation on HDFS failed: @message', ['@message' => $e->getMessage()]);
    return false;
  }
 }

/**
 * Appends data to an existing file on HDFS. If the file does not exist,
 * it should create it.
 *
 * @param string $file_path The path to the file on HDFS.
 * @param string $data The data to append to the file.
 *
 * @throws \Exception if the append operation fails.
 */
public function appendToFile($file_path, $data) {
  try {
    if (!$this->client->exists($file_path)) {
      $this->client->create($file_path, $data);
    } else {
      $this->client->append($file_path, $data);
    }
  } catch (\Exception $e) {
    \Drupal::logger('unwehdfs')->error("Failed to append to file {$file_path}: " . $e->getMessage());
    throw $e;
```

```
  }
}


}
```

Управление на конфигурацията

За да управлява настройките за интеграция на HDFS, като подробности за хоста, порт и идентификационни данни, модулът използва системата за управление на конфигурацията на Drupal. Това включва създаване на конфигурационна форма в администраторския интерфейс на модула, която позволява на администраторите да въвеждат и актуализират тези настройки.

```
use Drupal\Core\Form\ConfigFormBase;
use Drupal\Core\Form\FormStateInterface;

class HDFSConfigForm extends ConfigFormBase {

 /**
  * {@inheritdoc}
  */
 public function getFormId() {
   return 'unwehdfs_admin_settings';
 }

 /**
```

```
 * {@inheritdoc}
 */
protected function getEditableConfigNames() {
  return [
    'unwehdfs.settings',
  ];
}

public function buildForm(array $form, FormStateInterface $form_state) {
  $config = $this->config('unwehdfs.settings');

  $form['webhdfs_url'] = [
    '#type' => 'url',
    '#title' => $this->t('WebHDFS URL'),
    '#default_value' => $config->get('webhdfs_url'),
    '#required' => TRUE,
  ];

  return parent::buildForm($form, $form_state);
}

public function submitForm(array &$form, FormStateInterface $form_state) {
  $config = $this->config('unwehdfs.settings');
  $config->set('webhdfs_url', $form_state->getValue('webhdfs_url'));
```

```
    $config->save();

    parent::submitForm($form, $form_state);
  }
}
```

Операции за обработка на файлове

Модулът включва функции за обработка на файлове на HDFS като hdfsFileWrite(), hdfsFileRead(), hdfsFileDelete() и hdfsDirectoryList(). Тези функции използват услугата HDFSClient за извършване на своите операции.

Обработка на грешки и регистриране

Внедрени са стабилни механизми за обработка на грешки и регистриране, за да се гарантира, че всички проблеми по време на взаимодействието с HDFS се улавят и регистрират с помощта на системата за регистриране на Drupal. Това е от решаващо значение за диагностициране и разрешаване на всички проблеми, които възникват по време на работата на модула.

Следва дефинирането на hooks и помощни функции, които да изпълняват двупосочното обновяване на информацията.

```
/**
 * Implements hook_node_insert().
 */
function unwehdfs_node_insert(Drupal\node\NodeInterface $node) {
  _unwehdfs_node_changed($node, 'create');
}
```

```
/**
 * Implements hook_node_update().
 */
function unwehdfs_node_update(Drupal\node\NodeInterface $node) {
  _unwehdfs_node_changed($node, 'update');
}

/**
 * Implements hook_node_delete().
 */
function unwehdfs_node_delete(Drupal\node\NodeInterface $node) {
  _unwehdfs_node_changed($node, 'delete');
}
/**
 * Responds to the node being inserted/updated/deleted.
 *
 * @param \Drupal\node\NodeInterface $node
 *   The node that is being acted upon.
 * @param string $operation
 *   The operation being performed - 'create', 'update', or 'delete'.
 */
function _unwehdfs_node_changed(Drupal\node\NodeInterface $node, $operation)
{
  $config = \Drupal::config('unwehdfs.settings');
  $hdfs_path = $config->get('hdfs_path');
```

```
  $hdfs_client = \Drupal::service('unwehdfs.hdfs_client');

  // Determine the HDFS file path based on node type.
  $node_type = $node->bundle();
  $file_path = "{$hdfs_path}/{$node_type}.csv";

  $csv_data = _unwehdfs_serialize_node_to_csv($node);

  try {
    if ($operation === 'delete') {
      $csv_data = _unwehdfs_rebuild_csv_for_node_type($node_type, $node->id());
      $hdfs_client->createFile($file_path, $csv_data);
    } else {
      $hdfs_client->appendToFile($file_path, $csv_data);
    }
  } catch (\Exception $e) {
    \Drupal::logger('unwehdfs')->error($e->getMessage());
  }
}

/**
 * Rebuilds the CSV data for a specific node type excluding a specific node ID.
 *
 * @param string $node_type
 *   The node type (bundle).
```

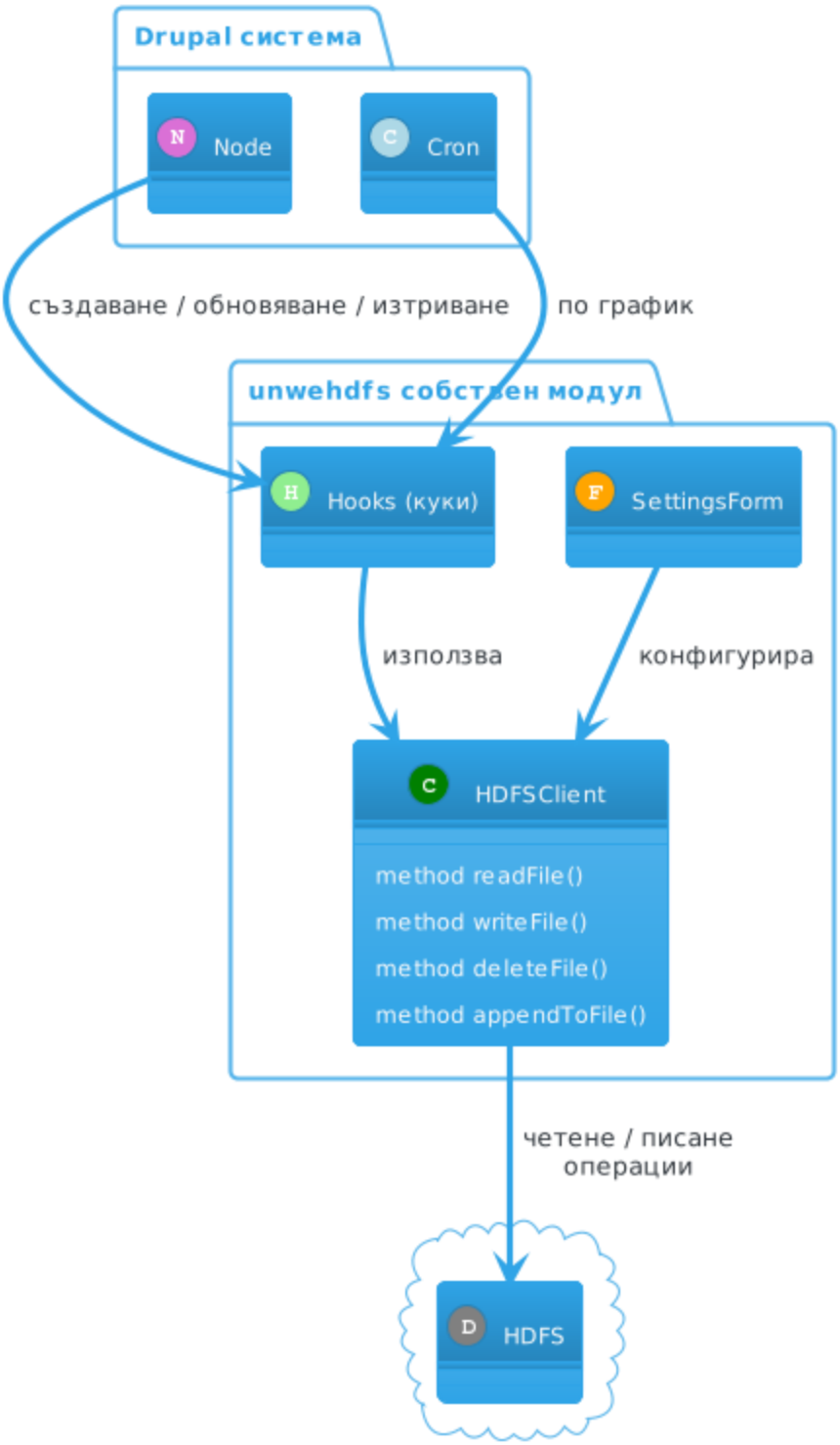
```
 * @param int $exclude_nid
 *   The node ID to exclude from the CSV.
 *
 * @return string
 *   The rebuilt CSV data.
 */
function _unwehdfs_rebuild_csv_for_node_type($node_type, $exclude_nid) {
  $query = \Drupal::entityQuery('node')
    ->condition('type', $node_type)
    ->condition('nid', $exclude_nid, '<>');

  $nids = $query->execute();
  $nodes = \Drupal\node\Entity\Node::loadMultiple($nids);

  $csv_lines = [];
  foreach ($nodes as $node) {
    $csv_lines[] = _unwehdfs_serialize_node_to_csv($node);
  }
  return implode("\n", $csv_lines);
}

/**
 * Serializes the node fields to CSV format.
 *
 * @param \Drupal\node\NodeInterface $node
```

```
 *   The node entity.
 *
 * @return string
 *   The serialized CSV string.
 */
function _unwehdfs_serialize_node_to_csv(Drupal\node\NodeInterface $node) {
  $csv_lines = [];
  // Iterate over all fields of the node.
  foreach ($node->getFields() as $field_name => $field) {
    $field_values = $field->getValue();
    // Flatten the field values to a simple array.
    $flat_values = array_map(function ($value) {
      // Return the scalar value or the first item if it's an array.
      return is_array($value) ? reset($value) : $value;
    }, $field_values);
    // Convert the array into a CSV line.
    $csv_lines[] = "" . implode('","', $flat_values) . "";
  }
  // Convert lines into a single CSV string.
  $csv_data = implode("\n", $csv_lines);
  return $csv_data;
}

/**
 * Implements hook_cron().
```

```
 */
function unwehdfs_cron() {
  $config = \Drupal::config('unwehdfs.settings');
  $hdfs_path = $config->get('hdfs_path');
  $bundle_name = $config->get('bundle_name');
  $file_path = $hdfs_path  . "/" . $bundle_name . ".csv";
  $node_type = $bundle_name;

  $hdfs_client = \Drupal::service('unwehdfs.hdfs_client');

  try {
    // Read the CSV data from HDFS.
    $csv_data = $hdfs_client->readFile($file_path);

    // Convert the CSV data into an array.
    $rows = _unwehdfs_parse_csv_data($csv_data);

    // Iterate over each CSV row and create a node.
    foreach ($rows as $row) {
      if (empty($row)) {
        continue;
      }

      // Create a new node object with the bundle type.
      $node = \Drupal\node\Entity\Node::create([
```

```
      'type' => $node_type,
      'title' => $row['title'],
    ]);

    $node->save();

    \Drupal::logger('unwehdfs')->notice("Created node {$node->id()} from HDFS CSV data.");
   }
  } catch (\Exception $e) {
   \Drupal::logger('unwehdfs')->error("Error processing HDFS CSV data: " . $e->getMessage());
  }
}

/**
 * Parses a CSV string into an array of rows.
 *
 * @param string $csv_data
 *   The CSV data as a string.
 *
 * @return array
 *   An array of associative arrays, each representing a CSV row.
 */
function _unwehdfs_parse_csv_data($csv_data) {
  $rows = [];
```

```
  $lines = explode("\n", $csv_data);

  $header = [];
  if (!empty($lines)) {
    $header = str_getcsv(array_shift($lines));
  }

  // Parse each line into an associative array.
  foreach ($lines as $line) {
    $rows[] = array_combine($header, str_getcsv($line));
  }

  return $rows;
}
```



Създаването на модула "unwehdfs" за Drupal за интерфейс с HDFS е сложен процес, който изисква дълбоко разбиране на архитектурата на Drupal, PHP програмирането и HDFS API. Чрез този систематичен подход модулът може

да осигури стабилна и безпроблемна интеграция между Drupal и HDFS, което позволява използването на широкомащабните възможности за съхранение на данни на HDFS в Drupal среда.

Литература

1. Mohammed Farhaz; DRUPAL 9 CUSTOM MODULE DEVELOPMENT – A BEGINNERS GUIDE; 2021; https://www.specbee.com/blogs/drupal-9-custom-module-development
2. Soumya Jha; Drupal 9 Custom Module Development; 2023; https://www.zyxware.com/article/drupal-9-custom-module-development-guide
3. WebHDFS REST API; https://hadoop.apache.org/docs/r1.0.4/webhdfs.html
4. Betty Hunsaker; WebHDFS FileSystem APIs; 2023; https://copyprogramming.com/howto/webhdfs-filesystem-apis