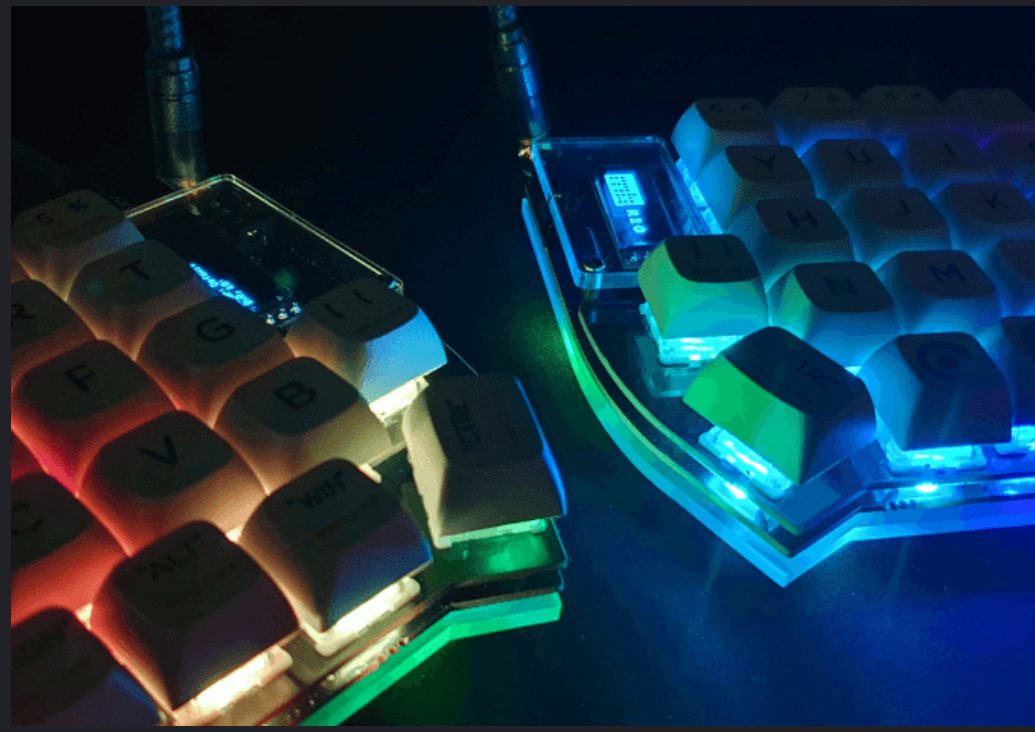
The Drupal Batch API

Philip Norton

DrupalCamp Scotland 2024

Philip Norton

- Developer at Code Enigma
- Writer at `#! code` (www.hashbangcode.com)
- NWDUG co-organiser



Source Code

- Talk is available on the QR code.
- All code seen can be found here

  https://bit.ly/3BQ4rsW
- I have also written about the Batch API on

  www.hashbangcode.com

The Drupal Batch API

The Batch API

Allows data to be processed in small chunks in order to prevent timeout errors or memory problems.

What Problem Are We Solving?

Bored Users

- Users get bored quickly.
- Studies show that a 5 second page load has a 0.6% conversion rate.
- Reducing this to 2 seconds doubes the conversion rate.
- This still means that after 2 seconds 98% of users will assume the page will not do anything.

Server Timeouts

- Servers are designed to throw errors if something takes too long. Some defaults:
  - PHP (`max_execution_time`) - 30 seconds
  - PHP (`memory_limit`) - 256MB (recommended for Drupal)
  - Apache (`TimeOut`) - 60 seconds
  - Nginx (`send_timeout` / `proxy_send_timeout`) - 60 seconds

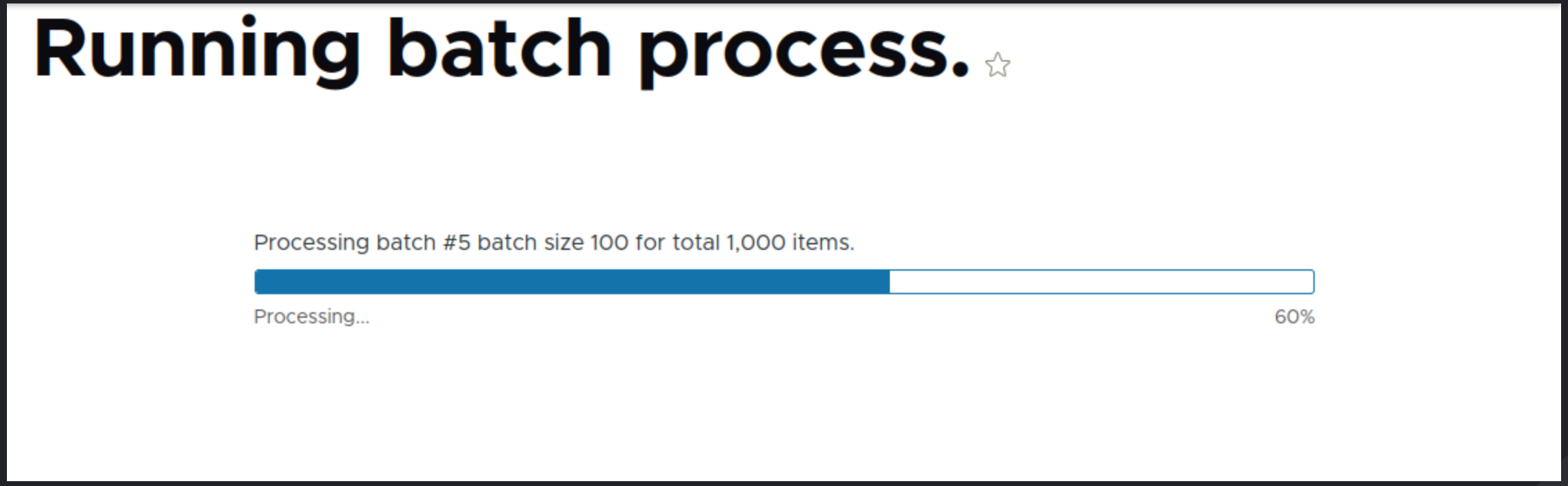
The Problem

- Trying to do too much in one page request.
  - Downloading lots of data from an api.
  - Create/update/delete lots of entities.

- Users assume page is broken and click away.
- The page times out or runs out of memory.

The Batch API

- Solves these problems by splitting long tasks into smaller chunks.
- Drupal then runs them through a special interface.



Using The Batch API

- Look at some code to run batches.
- Do some demos showing the code in action.

The Batch API Stages

The Batch API can be thought of as the following stages:

- **Initialise** - Set up the batch run, define callbacks.
- **Process** - The batch process operations.
- **Finish** - A finish callback.

Initialise

The BatchBuilder class is used to setup the batch.

```
use Drupal\Core\Batch\BatchBuilder;
$batch = new BatchBuilder();
```

Initialise

A number of methods set up different parameters.

```
$batch = new BatchBuilder();
$batch->setTitle('Running batch process.')
  ->setFinishCallback([self::class, 'batchFinished'])
  ->setInitMessage('Commencing')
  ->setProgressMessage('Processing...')
  ->setErrorMessage('An error occurred during processing.');
```

Initialise - Adding Operations

Populate the operations we want to perform.

```php
// Create 10 chunks of 100 items.
$chunks = array_chunk(range(1, 1000), 100);

// Process each chunk in the array.
foreach ($chunks as $id => $chunk) {
  $args = [
    $id,
    $chunk,
  ];
  $batch->addOperation([BatchClass::class, 'batchProcess'], $args);
}
```

Initialise - Start Batch Run

- Set the batch running by calling `toArray()` and passing the array to `batch_set()`.

```
batch_set($batch->toArray());
```

- The whole purpose of `BatchBuilder` is to generate that array.
- This will trigger and start up the batch process.

Process

- The callbacks defined in the `addOperation()` method are executed.
- Parameters are the array of arguments you set.
- `$context` is passed as the last parameter is used to track progress.

```
public static function batchProcess(int $batchId, array $chunk, array &$context): void {
}
```

Process - Tracking Progress

- The `$context` parameter is an array that is maintained between different batch calls.
- The `"sandbox"` element is used inside the batch process and is deleted at the end of the batch run.
- The `"results"` element will be passed to the finished callback and is often used to track progres for reporting.

```
public static function batchProcess(int $batchId, array $chunk, array &$context): void {
    if (!isset($context['sandbox']['progress'])) { }
    if (!isset($context['results']['updated'])) { }
}
```

Process - Tracking Progress

- Some sensible defaults.

```
public static function batchProcess(int $batchId, array $chunk, array &$context): void {
  if (!isset($context['sandbox']['progress'])) {
    $context['sandbox']['progress'] = 0;
    $context['sandbox']['max'] = 1000;
  }
  if (!isset($context['results']['updated'])) {
    $context['results']['updated'] = 0;
    $context['results']['skipped'] = 0;
    $context['results']['failed'] = 0;
    $context['results']['progress'] = 0;
    $context['results']['process'] = 'Form batch completed';
  }

}
```

Process - Messages

- As the batch runs you can set a `"message"` property to print messages to the user.
- This will appear above the batch progress bar as the batch progresses.

```
// Message above progress bar.
$context['message'] = t('Processing batch #@batch_id batch size @batch_size for total @count items.', [
  '@batch_id' => number_format($batchId),
  '@batch_size' => number_format(count($chunk)),
  '@count' => number_format($context['sandbox']['max']),
]);
```

Process

- Perform the task you want in the batch.

```
public static function batchProcess(int $batchId, array $chunk, array &$context): void {
  // --- Set up and messages goes here...
  $random = new Random();
  foreach ($chunk as $number) {
    $context['results']['progress']++;
    $context['sandbox']['progress']++;
    $node = Node::create([
      'type' => 'article',
      'title' => $random->name(15),
      'body' => [
        'value' => '<p>' . $random->sentences(2) . '</p>', 'format' => filter_default_format(),
      ],
      'uid' => 1,
      'status' => 1,
    ]);
    $node->save();
  }
}
```

Finish - The Finished Callback

- When the batch finishes, the finished callback is triggered.
- This has a set of parameters that detail how the batch performed.

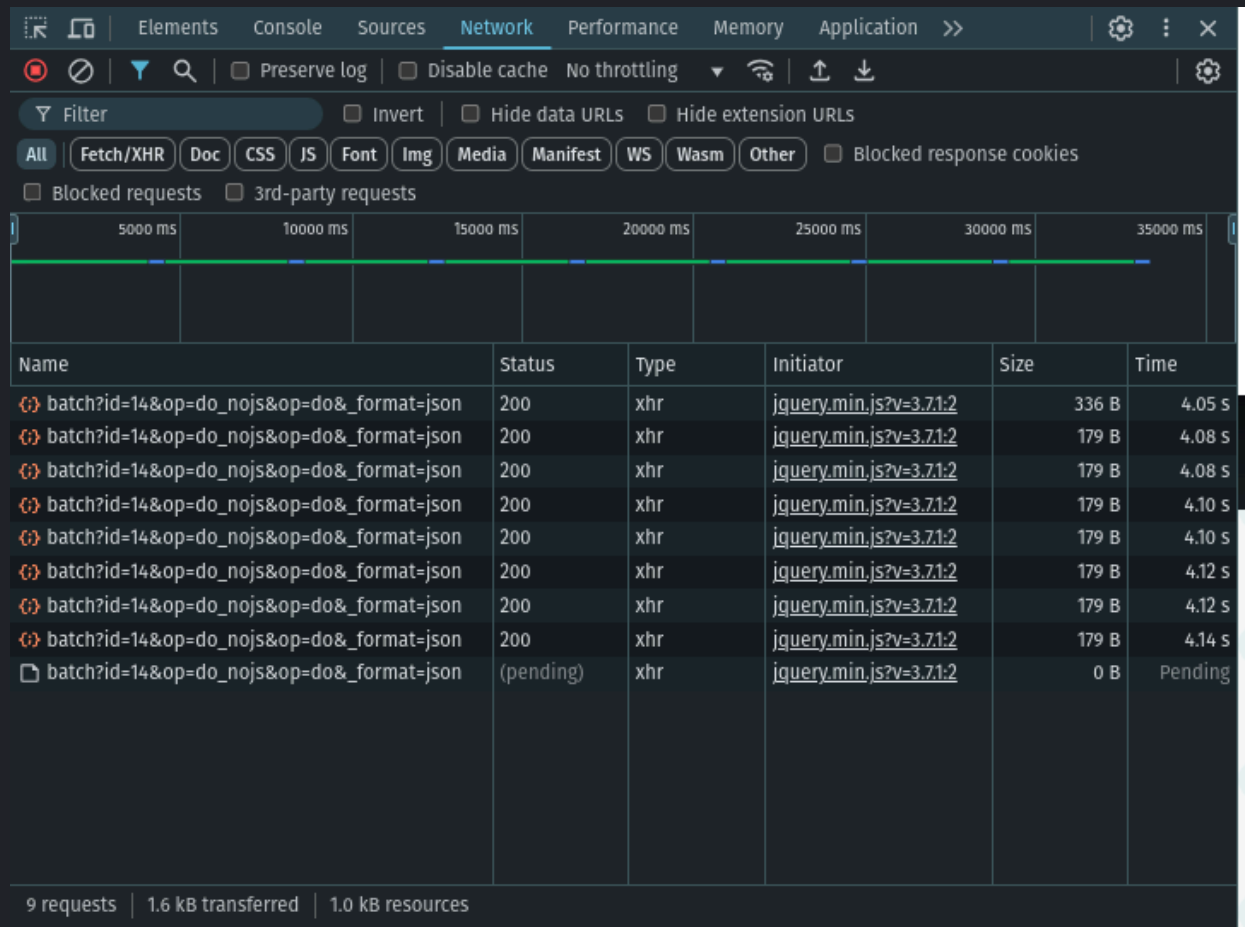
```
public static function batchFinished(
  bool $success,
  array $results,
  array $operations,
  string $elapsed): void {
}
```

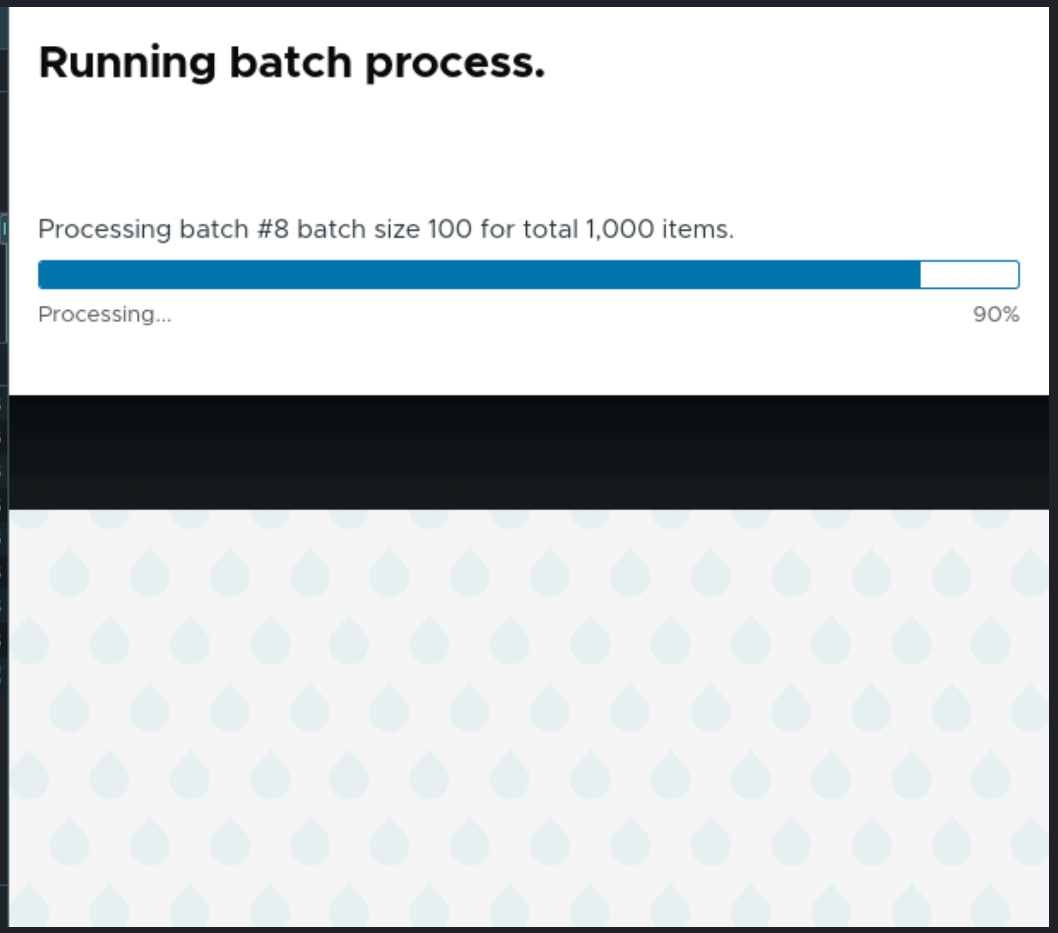
Finished - The Finished Callback

For example, you might want to report the results of the batch run to your user.

```
  public static function batchFinished(bool $success, array $results, array $operations, string $elapsed): void {
    $messenger = \Drupal::messenger();
    if ($success) {
      $messenger->addMessage(t('@process processed @count, skipped @skipped, updated @updated, failed @failed in @elapsed.', [
        '@process' => $results['process'],
        '@count' => $results['progress'],
        '@skipped' => $results['skipped'],
        '@updated' => $results['updated'],
        '@failed' => $results['failed'],
        '@elapsed' => $elapsed,
      ]));
    }
}
```

The Running Batch





Batch Internal Workings

- The Batch API is really an extension of the Queue system.
- When you add operations to the batch you are adding items to the queue.
- The Drupal batch runner then pulls items out of the queue and feeds them to the process.

The Batch "finished" State

The Batch "finished" State

- So far, we have looked at pre-configured batch runs.
- A better approach is to use the `finished` property of the batch `$context` array.
- If we set this value to >= 1 then the batch process callback is considered finished.

```
if (done) {
  $context['finished'] = 1;
}
```

The Batch "finished" State

The setup is slightly different as we only create a single operation.

```
$array = range(1, 1000);
$batch->addOperation([BatchClass::class, 'batchProcess'], [$array]);
```

This is run over and over until we issue the finished state.

The Batch "finished" State

It is common to divide the progress by the maximum number of items.

```
$context['finished'] = $context['sandbox']['progress'] / $context['sandbox']['max'];
```

The Batch "finished" State

This also means that we can just launch the batch with no arguments.

```
$batch->addOperation([BatchProcessNodes::class, 'batchProcess']);
```

The `max` property is discovered in the `batchProcess()` method the first time it is run.

```
public static function batchProcess(array &$context): void {
  if (!isset($context['sandbox']['progress'])) {
    $query = \Drupal::entityQuery('node');
    $query->accessCheck(FALSE);
    $context['sandbox']['progress'] = 0;
    $context['sandbox']['max'] = $query->count()->execute();
  }
```

Running Batch With Drush

Drush

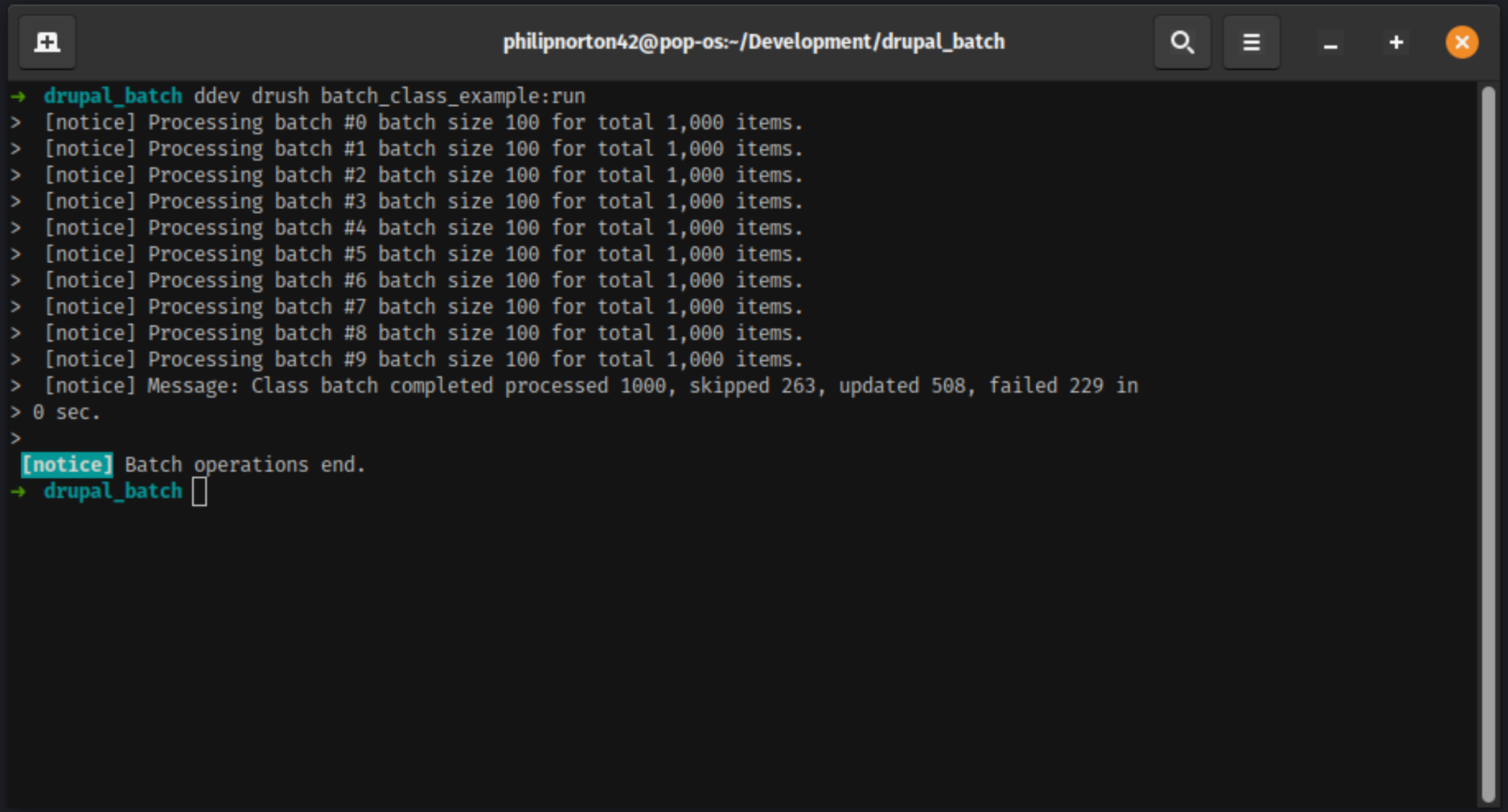
Call batch set as normal.

```
batch_set($batch->toArray());
```

Then call the Drush function.

```
drush_backend_batch_process();
```

This will run the batch on the command line.


philipnorton42@pop-os:~/Development/drupal_batch
→ drupal_batch ddev drush batch_class_example:run
> [notice] Processing batch #0 batch size 100 for total 1,000 items.
> [notice] Processing batch #1 batch size 100 for total 1,000 items.
> [notice] Processing batch #2 batch size 100 for total 1,000 items.
> [notice] Processing batch #3 batch size 100 for total 1,000 items.
> [notice] Processing batch #4 batch size 100 for total 1,000 items.
> [notice] Processing batch #5 batch size 100 for total 1,000 items.
> [notice] Processing batch #6 batch size 100 for total 1,000 items.
> [notice] Processing batch #7 batch size 100 for total 1,000 items.
> [notice] Processing batch #8 batch size 100 for total 1,000 items.
> [notice] Processing batch #9 batch size 100 for total 1,000 items.
> [notice] Message: Class batch completed processed 1000, skipped 263, updated 508, failed 229 in
> 0 sec.
>
[notice] Batch operations end.
→ drupal_batch

Drush

- Be careful! Drush will process the batch operations in the same memory space.
- As you are on the command line you won't time out, but you can run out of memory.

Examples Of Batch API In Action

Some live demos!

Batch Using A Form

- A look at the Batch API shown above.
- Batch process goes through 1,000 items and randomly determines outcome.

Batch Using Drush

- Batch process goes through 1,000 items and randomly determines outcome.
- This time, in Drush!

Process a CSV file

- Import 1,000 nodes using a batch process.
- This uses the `finished` property to track progress of the CSV and stop the batch when needed.

The Batch API Inside Drupal

Batch API In Drupal

- Drupal makes use of the Batch API in lots of different situations. For example:
  - Installing Drupal.
  - Installing modules.
  - Importing translations.
  - Importing configuration.
  - Deleting users.
  - Bulk content updates.
  - And much more!

The Update Hook

- Update hooks get a `$sandbox` variable. This is actually a batch `$context` array.
- Update hooks are hook_update_N() and hook_post_update_NAME().
- You can set the `#finished` property in the `$sandbox` array to stop the batch.

The Update Hook

An example of a batched update hook.

```
function batch_update_example_update_10001(&$sandbox) {
  if (!isset($sandbox['progress'])) {
    $sandbox['progress'] = 0;
    $sandbox['max'] = 1000;
  }
  $batchSize = 100;
  $batchUpperRange = $sandbox['progress'] + $batchSize;

  for ($i = $sandbox['progress']; $i < $batchUpperRange; $i++) {
    // Keep track of progress.
    $sandbox['progress']++;
    // Do some actions...
  }
  \Drupal::messenger()->addMessage($sandbox['progress'] . ' items processed.');

  $sandbox['#finished'] = $sandbox['progress'] / $sandbox['max'];
}
```

Some Tips On Batch API Usage

When To Use The Batch API

- If the request processes multiple items then move it into a batch.
- Users will more readily wait for a batch to finish than a spinning page.
- Use the batch system early to save having to rework things later.

Top Tips

- If the data needs to be processed in real time then use a batch; otherwise use a standard queue.
- Kick off your batches in a form or controller, but process the batch in a separate class. This allows easy Drush integration.
- Use the `finished` property to make dynamic batches; rather than preloaded.

Top Tips

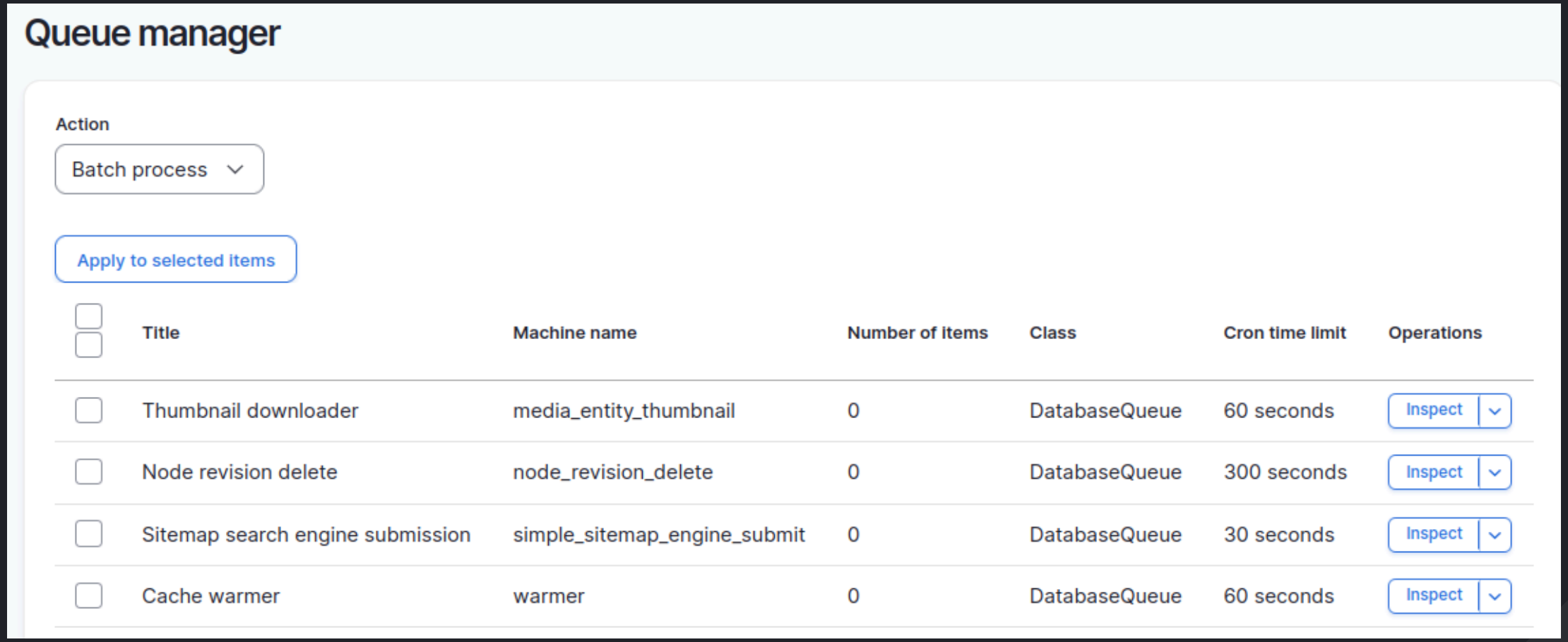
- Keep your batch operations simple. Break them apart into separate operations if needed.
- Think about the footprint of your batch operations. Keep them small. You can still cause timeouts during the batch if you aren't careful.
- Try to allow batch operations to pick up where they left off. If any errors occur you can re-run to complete the task.

Modules That Use Batch

Queue UI

- View current queues and process them with a batch.

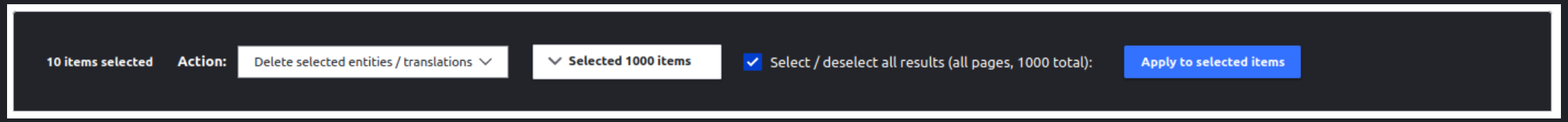
https://www.drupal.org/project/queue_ui



View Batch Operations

- Batch process items in a view.

https://www.drupal.org/project/views_bulk_operations



Views Data Export

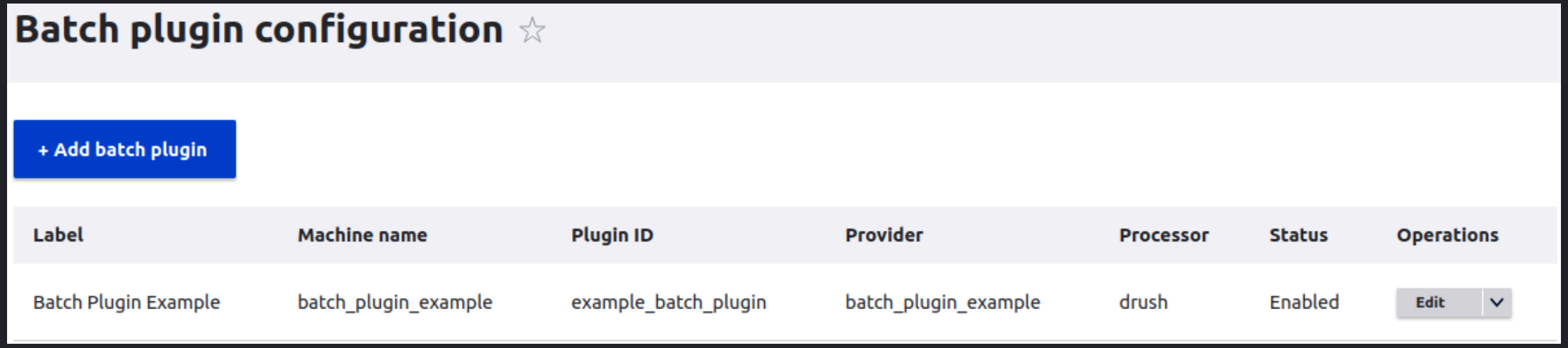
- A Views plugin that exports data in a number of different formats.

https://www.drupal.org/project/views_data_export

Batch Plugin

- Wraps the Batch API in a plugin to make your batch operations pluggable.

https://www.drupal.org/project/batch_plugin



Resources

- Batch on `#! code`
  https://www.hashbangcode.com/tag/batch-api
- All code seen is code available at
  https://github.com/hashbangcode/drupal_batch_examples/
- Batch Processing and the Drupal Queue System -
  https://www.drupalatyourfingertips.com/bq

Questions?

- Slides: https://github.com/hashbangcode/batch-api-talk

Thanks!

- Slides: https://github.com/hashbangcode/batch-api-talk