Setting Up A Database Adapter

zend-db provides a general purpose database abstraction layer. At its heart is the `Adapter`, which abstracts common database operations across the variety of drivers we support.

In this guide, we will document how to configure both a single, default adapter as well as multiple adapters (which may be useful in architectures that have a cluster of read-only replicated servers and a single writable server of record).

Installing zend-db

First, install zend-db using Composer:

```
$ composer require zendframework/zend-db
```

If you are using zend-component-installer (installed by default with the skeleton application, and optionally for Expressive applications), you will be prompted to install the package configuration.

- For zend-mvc applications, choose either `application.config.php` or `modules.config.php`.
- For Expressive applications, choose `config/config.php`.

If you are not using the installer, you will need to manually configure add the component to your application.

- For zend-mvc applications, update your list of modules in either `config/application.config.php` or `config/modules.config.php` to add an entry for `'Zend\Db'` at the top of the list:

```
<?php

return [
  'Zend\Db',
  'Zend\Form',

];


return [


'modules' => [
  'Zend\Db',
  'Zend\Form',

],
```

- For Expressive applications, create a new file, `config/autoload/zend-db.global.php`, with the following contents:

```
<?php
use Zend\Db\ConfigProvider;

return (new ConfigProvider())();
```

Configuring the default adapter

Within your service factories, you may retrieve the default adapter from your application container using the class name `Zend\Db\Adapter\AdapterInterface`:

```
use Zend\Db\Adapter\AdapterInterface;

function ($container) {
    return new
SomeServiceObject($container->get(AdapterInterface::class));
}
```

When installed and configured, the factory associated with `AdapterInterface` will look for a top-level `db` key in the configuration, and use it to create an adapter. As an example, the following would connect to a MySQL database using PDO, and the supplied PDO DSN:

```
return [
    'db' => [
        'driver' => 'Pdo',
        'dsn'    => 'mysql:dbname=zftutorial;host=localhost;
charset=utf8',
    ],
];
```

More information on adapter configuration can be found in the docs for Zend\Db\Adapter.

Configuring named adapters

Sometimes you may need multiple adapters. As an example, if you work with a cluster of databases, one may allow write operations, while another may be read-only.

zend-db provides an abstract factory, `Zend\Db\Adapter\AdapterAbstractServiceFactory`, for this purpose. To use it, you will need to create named configuration keys under `db.adapters`, each with configuration for an adapter:

```
return [
    'db' => [
        'adapters' => [
            'Application\Db\WriteAdapter' => [
                'driver' => 'Pdo',
                'dsn'    => 'mysql:dbname=application;
host=canonical.example.com;charset=utf8',
            ],
            'Application\Db\ReadOnlyAdapter' => [
                'driver' => 'Pdo',
                'dsn'    => 'mysql:dbname=application;
host=replica.example.com;charset=utf8',
            ],
        ],
    ],
```

```
];
```

You retrieve the database adapters using the keys you define, so ensure they are unique to your application, and descriptive of their purpose!

Retrieving named adapters

Retrieve named adapters in your service factories just as you would another service:

```
 function ($container) {
    return new SomeServiceObject($container->get('Application\Db
\ReadOnlyAdapter));
}
```

Using the AdapterAbstractServiceFactory as a factory

Depending on what application container you use, abstract factories may not be available. Alternately, you may want to reduce lookup time when retrieving an adapter from the container (abstract factories are consulted last!). zend-servicemanager abstract factories work as factories in their own right, and are passed the service name as an argument, allowing them to vary their return value based on requested service name. As such, you can add the following service configuration as well:

```
 use Zend\Db\Adapter\AdapterAbstractServiceFactory;



'service_manager' => [
    'factories' => [
        'Application\Db\WriteAdapter' =>
AdapterAbstractServiceFactory::class,
    ],
],


'dependencies' => [
    'factories' => [
        'Application\Db\WriteAdapter' =>
AdapterAbstractServiceFactory::class,
    ],
],
```
