edX Discovery Service

Release 0.1

edX

Jan 05, 2023

CONTENTS

Discovery is a service that provides access to consolidated course and program metadata. It does this primarily through a REST API that supports courses, course runs, programs, catalogs, and search.

This guide begins with some background information on the service, then focuses on what you need to know to run and develop for the service.

CHAPTER ONE

INTRODUCTION

The distribution of edX's data has grown over time. Any given feature on edx.org may need information from Studio, the LMS, the Ecommerce service, and/or the Drupal marketing site. Discovery is a data aggregator whose job is to collect, consolidate, and provide access to information from these services.

Discovery allows services internal to an Open edX installation to consume a consolidated source of metadata for presentation to users. For example, search on edx.org is provided by Discovery. Discovery also allows external parties to access data about content in an Open edX installation from a single, central location in a secure way that doesn't impact performance of said installation.

1.1 Courses and Course Runs

One of Discovery's distinguishing features is the way it formalizes the relationship between courses and course runs. For example, `course-v1:foo+bar+fall` and `course-v1:foo+bar+spring` identify fall and spring runs of the same course, `foo+bar`. You can think of courses as collections of course runs. Discovery infers this relationship when collecting data from other services. This hierarchy is the foundation for catalogs and programs, two additional structures provided by Discovery.

1.2 Catalogs

Catalogs are dynamic groups of courses. A catalog is defined with an Elasticsearch query. Catalogs are used to give external parties scoped views of edX content. They are also used to implement coupons on the Ecommerce service. For example, a coupon providing a 25% discount on courses from a specific organization would be tied to a catalog identifying those courses.

1.3 Programs

Programs are fixed collections of courses whose completion results in the awarding of a credential. Discovery only stores program metadata. For example, Discovery is responsible for keeping track of which courses belong to a program. Other program-related features such as calculating completion and awarding credentials are the responsibilities of separate systems.

1.4 Data Loading

Data about courses and course runs is collected from Studio, the LMS, the Ecommerce service, and, for edx.org, the Drupal marketing site. The data loading pipeline used to collect this data can be run with a management command called `refresh_course_metadata`. edX runs this command several times a day using a Jenkins job. It can be manually run to populate a local environment with data. The data loading framework is designed to make adding additional systems easy.

1.5 Search

Discovery uses Elasticsearch to index data about courses, course runs, and programs. Indexing can be run at any time with a management command called `update_index`. The Discovery API can be used to run search queries against the Elasticsearch index.

1.6 API

Access to information about courses, course runs, catalogs, programs, and more is provided by a REST API. For more about the API, use your browser to visit `/api-docs` hosted by a running Discovery instance.

1.7 Creating/Accessing the Discovery Service Django Admin

To access the Django admin panel, you must create a superuser account. Login to the machine where Discovery is installed, and run the `createsuperuser` management command. For example, from the devstack discovery shell:

```
$ sudo -Hs -u discovery
$ source ~/discovery_env
$ source ~/venvs/discovery/bin/activate
$ cd ~/discovery
$ ./manage.py createsuperuser --username=USERNAME --email=username@example.com
```

Now you can access Discovery Django admin at `http://yourdomain:18381/admin`. Login with the username and password created above.

CHAPTER TWO

QUICKSTART

This section covers information you need to know to run and develop for the Discovery service.

2.1 Devstack

Discovery is part of edX's Docker-based "devstack." To run the service locally, follow along with the instructions in the https://github.com/openedx/devstack repo's README.

Devstack will allow you to run all edX services together. If you only need Discovery, you can run just the services it requires:

```
$ make dev.up.discovery
```

2.2 Data Loaders

Run the data loaders using the `refresh_course_metadata` management command to populate a Discovery instance with data. Open a Discovery shell with `make discovery-shell`, then run:

```
$ ./manage.py refresh_course_metadata
```

By default, `refresh_course_metadata` loads data for every "partner" in the system. Partners are site tenants, like edx.org. You can view and create tenants using the Django admin at `/admin/core/partner/`. To load data for a specific tenant:

```
$ ./manage.py refresh_course_metadata --partner_code <SHORT CODE HERE>
```

2.3 Search Indexing

Once you've loaded data into your Discovery instance, you may want to run Elasticsearch queries against it. Doing so requires indexing the data you've loaded, which you can do by running the `update_index` management command. Open a Discovery shell with `make discovery-shell`, then run:

```
$ ./manage.py update_index --disable-change-limit
```

Once indexing completes, you can run search queries against the newly created index through the API. For more on this, visit `/api-docs`.

2.4 Tests

Use Docker Compose to run tests just like Travis does. The `.travis.yml` file is a good reference if you want to run the entire test suite. You'll notice that a Docker Compose file hosted in this repo is used to run tests instead of the Compose files in the devstack repo. This Compose file defines special test settings and has yet to be consolidated with the Compose files in the devstack repo.

To run specific tests, bring up the services used for testing with `make ci_up`. To run the tests in `course_discovery/apps/api/v1/tests/test_views/test_programs.py`:

```
$ make ci_test
```

This will install some dependencies in addition to running all tests. After the dependencies have been run (you can interrupt during test running if you like) you can run

```
$ docker-compose -f .ci/docker-compose-ci.yml exec discovery bash -c 'cd /edx/app/
↪discovery/discovery && .tox/py38-django22/bin/pytest course_discovery/apps/api/v1/
↪tests/test_views/test_programs.py'
```

When you're done, take down the services you brought up with `make ci_down`.

CHAPTER

THREE

ADVANCED USAGE

This section contains information about advanced usage and operation of the Discovery service.

3.1 Elasticsearch

Discovery uses Elasticsearch 1.5 to provide search functionality.

3.1.1 Index Aliasing

Discovery application code uses an index alias to refer to the search index indirectly. For example, the timestamped `course_discovery_20160101113005` index may be assigned and referred to by the alias `catalog`. Using an alias prevents index maintenance (e.g., the indexing and index swapping performed by `update_index`) from affecting service uptime.

3.1.2 Boosting

Discovery uses Elasticsearch's function score query to modify ("boost") the relevance score of documents retrieved by search queries. You can find the service's boosting config at `course_discovery/apps/edx_haystack_extensions/elasticsearch_boost_config.py`, complete with comments explaining what each part does and how it's been tuned.

3.1.3 Querying Elasticsearch

In addition to running search queries through the Discovery API, you can make HTTP requests directly to Elasticsearch. This is especially useful if you want to tune how relevance scores are computed. These examples show curl being used from a Discovery shell:

```
$ curl 'edx.devstack.elasticsearch:9200/_cat/indices?v'
$ curl 'edx.devstack.elasticsearch:9200/catalog/_search?pretty=true' -d '{"explain":␣
↪true, "query": {YOUR QUERY HERE}}'
```

The explain parameter tells Elasticsearch to return a detailed breakdown of how relevance scores were calculated. You can get yourself a query to run by intercepting queries made by the application. Add logging to `course_discovery/apps/edx_haystack_extensions/backends.py::SimpleQuerySearchBackendMixin::build_search_kwargs` that prints the final value of `search_kwargs`, then run a search query through the API.

3.2 Extensions

edX manages two "extension" apps located at `course_discovery/apps/edx_catalog_extensions` and `course_discovery/apps/edx_haystack_extensions` as part of Discovery. These apps provide edX-specific customizations. They include data migrations, management commands, and search backends specific to edX. We'd like to move these apps to separate repos at some point in the future to avoid confusion. They live here for now until we can determine what other edX-specific components need to be extracted from the general project.

`edx_catalog_extensions` is disabled by default. edX developers should add `course_discovery.apps.edx_catalog_extensions` to `INSTALLED_APPS` in a `private.py` settings file.

3.3 Catalogs

Catalogs are dynamic groups of courses modeled as access-controlled Elasticsearch queries. You can find the `Catalog` model in `course_discovery/apps/catalogs/models.py`.

3.3.1 Permissions

A catalog's `viewers` property returns the users who are allowed to view the catalog and the courses within it. These per-object permissions are implemented using django-guardian.

3.3.2 Administration

You can administer catalogs through the LMS at `/api-admin/catalogs`. You can also modify catalogs using Discovery's Django admin at `/admin/catalogs/`. The admin interface provides a preview button you can use to view the list of courses contained in a catalog, as well as the standard `django-guardian` admin interface for managing user permissions.

3.4 Waffle

Discovery uses django-waffle to control the release of new features. This allows us to gradually increase traffic to new features and divert traffic quickly if problems are discovered. Please refer to Waffle's documentation for an overview of the models you may encounter throughout the codebase.

3.5 Internationalization

All user-facing strings should be marked for translation. edX runs this application in English, but our open source users may choose to use another language. Marking strings for translation ensures our users have this choice. Refer to edX's i18n guidelines for more details.

3.5.1 Updating Translated Strings

Like most edX projects, Discovery uses Transifex to translate content. At edX, the translation process is automated. Every week, changes to source code strings are extracted as translations, which are merged back to the repo and pushed to edX's Transifex resources. Translated strings are also merged back into the repo every week.

Open Source contributors can use `make extract_translations` to extract source file string changes, `make push_translations` to push changes to Transifex (assuming credentials are available), and `make pull_translations` to pull translations from Transifex.

3.6 OAuth2

The Discovery service uses the OAuth 2.0 protocol for authentication. The LMS currently serves as the OAuth2 provider.

If you're using devstack, OAuth2 should be configured for you. If you need to configure OAuth2 manually, you need to register a new client with the OAuth2 provider (the LMS) and update Discovery's Django settings with the newly created credentials.

A new OAuth 2.0 client can be created at `http://localhost:18000/admin/oauth2_provider/application/`.

1. Click the *Add Application* button.
2. Leave the user field blank.
3. Specify the name of this service, `credentials`, as the client name.
4. Set the *URL* to the root path of this service: `http://localhost:8150/`.
5. Set the *Redirect URL* to the complete endpoint: `http://localhost:18150/complete/edx-oauth2/`.
6. Copy the *Client ID* and *Client Secret* values. They will be used later.
7. Select *Confidential* as the client type.
8. Select *Authorization code* as the authorization grant type.
9. Click *Save*.

You can create a new OAuth 2.0 application on the LMS at `/admin/oauth2_provider/application/`:

1. Click the `Add Application` button.
2. Leave the user field blank.
3. Specify the name of this service, `discovery`, as the client name.
4. Set the `URL` to the root path of this service: `http://localhost:18381`.
5. Set the `Redirect URL` to the complete endpoint: `http://localhost:18381/complete/edx-oauth2/`.
6. Copy the `Client ID` and `Client Secret` values. They will be used later.
7. Select `Confidential (Web applications)` as the client type.
8. Select `Authorization code` as the authorization grant type.
9. Click `Save`.

Finally, copy the newly created `Client ID` value to the `SOCIAL_AUTH_EDX_OAUTH2_KEY` field and `Client Secret` to the `SOCIAL_AUTH_EDX_OAUTH2_SECRET` field in Discovery's settings (in `course_discovery/settings/private.py`, if running locally).

3.7 Publisher

“Publisher” is an information management tool meant to support the course authoring, review, and approval workflow. The tool can be used to manage course metadata and is designed for use with the Drupal site that hosts edx.org.