coveo™

Coveo Platform Implementation Guide

White Paper

Contents

Introduction

Coveo is a highly scalable, highly extensible, cloud-based search, recommendations, and personalization platform. It uses robust machine learning technologies to provide the most relevant experiences possible to virtually unlimited use cases.

This document is intended for anyone planning to use Coveo to power their search experiences. It guides you through the major steps and design questions required when developing with Coveo.

This guide does not intend to replace the **product documentation**, but share best practices for a successful implementation. For a step-by-step learning experience, see the **Level Up platform**.

What Is Coveo?

Coveo is a relevance platform that enables relevant experiences through machine learning-powered search and recommendations.

Coveo offers solutions for four main use cases:

1. **Commerce**: Increase revenue by showing people what they need during their shopping experience
2. **Website**: Improve user interaction and satisfaction by providing a unified hub of information for external visitors
3. **Workplace**: Increase employee proficiency and efficiency — and reduce operating costs — by allowing workers to get to internal resources they need more quickly
4. **Service**: Enable customer self-service and streamline service agent proficiency by efficiently resolving and closing support cases

No matter which solution you choose, Coveo functions with the same general principles:

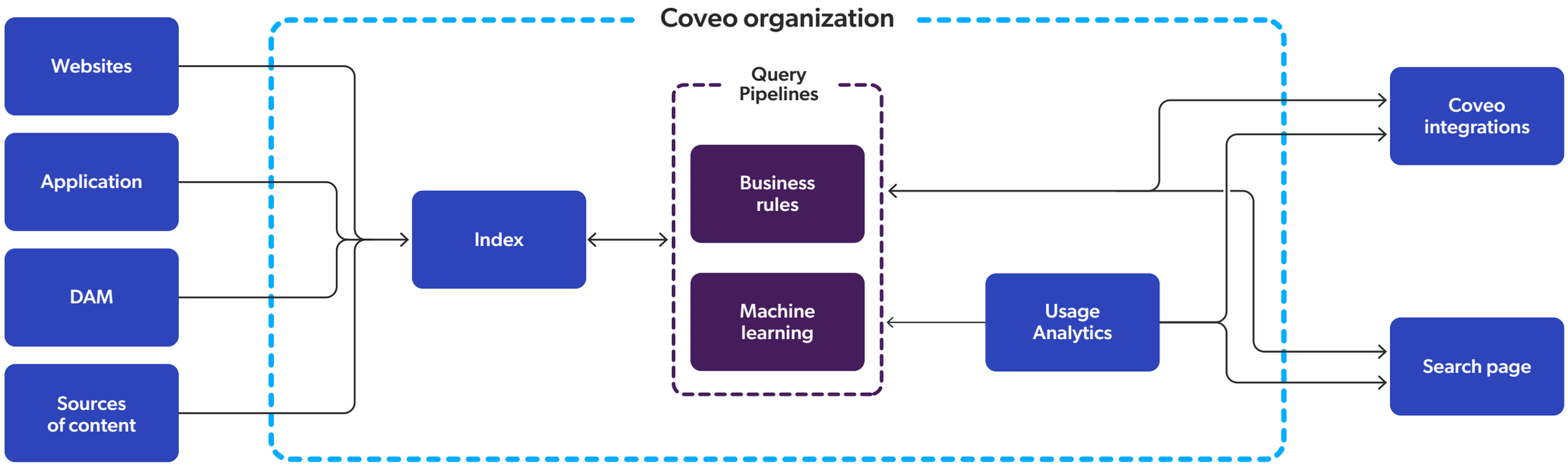
- Indexing the right content in your Coveo index
- Making a search experience to present that content
- Improving the relevance of your experience by enabling machine learning and adding business rules
- Continuously improve the experience by observing user behavior and making informed business decisions based on it

Project steps

1. **Indexing the appropriate content with the right metadata in Coveo**
2. **Creating the search experience to query that data**
3. **Tuning the relevance by adding features to enhance the relevance of your search experiences**

At its core, Coveo projects work in these three main phases. This project guide will help you through those steps in order.

Coveo architecture



The Coveo architecture can be summarized as such:

1. Coveo integrates content from many different sources using its connectors, and consolidates the information in a single, unified index that lives in your Coveo organization (instance).
2. A query into a search box (either standalone or from a Coveo integration) performs a call to Coveo, typically after an end-user interaction.
3. In parallel, a second call is made to Coveo Usage Analytics, indicating an end-user performed a search. This data is then fed to Coveo Machine Learning, to understand which queries and results are useful for end users.
4. The query goes through a query pipeline in Coveo Cloud.
5. The query pipeline modifies the query according to business rules and machine learning.
6. The query reaches the index, where Coveo determines which results should be returned.
7. The results go back through the query pipeline, where additional rules and machine learning can modify the result order to be more relevant.
8. The results are returned to the end user.
9. When a user clicks on a result, a call is made to Usage Analytics, indicating which results were successful.

Getting **Started**

The first thing you need when starting with Coveo is a Coveo Cloud instance. If you are a new Coveo client, you should have been provided with at least a sandbox and a production organization.

If you did not, please contact your Coveo representative or the Coveo administrator in your organization.

Alternatively, you can create a trial organization, which gives you the Enterprise features of Coveo for 30 days. For more information, please fill out **this form**.

Once you have your Coveo organization, navigate to the **Coveo platform** to access it. You can log in with any of the listed providers. You should always use that provider when subsequently logging in to the platform.

The menu on the left contains the major sections of Coveo.

- **Content** — Add and customize the content in your index. Everything to help you control, troubleshoot, and view the content in your index is part of this section.
- **Search** — Customize the search experience once the content is indexed. This is where you can add business rules such as synonyms or ranking rules, and where you can start to play with the Coveo JavaScript Search Framework to create search pages.
- **Machine learning** — Create and test Coveo Machine Learning models.
- **Analytics** — Report on how your users are interacting with your search interfaces.
- **Organization** — Manage the inner workings of your Coveo organization. For example, this is where you go to add members to your organization to help you with the development.

Working with a team

Typically, as a Coveo customer, you get access to one production organization, and one or two non-production organizations, depending on the license type.

When working with a large team during development, you might want to consider using multiple individual test organizations — and migrating your changes to a shared non-production organization. This ensures that the changes that one developer makes to their environment don't get overwritten by another developer.

Coveo recommends this approach for active development.

Pro Tip

You can control access to any invited members to your organizations. For example, you can invite a member that can only view the analytics reports, or a member that can only edit specific sources.

For more information, see Manage Groups.

Indexing **Content**

For Coveo to provide relevant content, that content needs to first exist in your Coveo index. To get content into your Coveo index, Coveo provides a list of available sources, such as Sitemap, SharePoint, or Salesforce. You can find the list of available sources in the **Connector Directory**.

Coveo can index any type of content into your index, even if there is no native Coveo connector for it. The exact method of indexing depends on the content you are indexing.

Push sources vs Pull sources

While Coveo offers many sources, they all work in one of two ways: either as Push sources, or as Pull sources. The difference between both is which side initiates the indexing process.

With Push sources, Coveo opens an API where you can send (or push) your items to be indexed. Push sources are typically simple on the Coveo end — since most of the complexity is in the code that performs the push.

There are currently only three Push sources: Sitecore (controlled by Coveo for Sitecore, which is installed on your Sitecore server), the Push API source, and all sources controlled by the Crawling Module, which is a piece of Coveo software that can be installed on your servers to push content from behind a firewall.

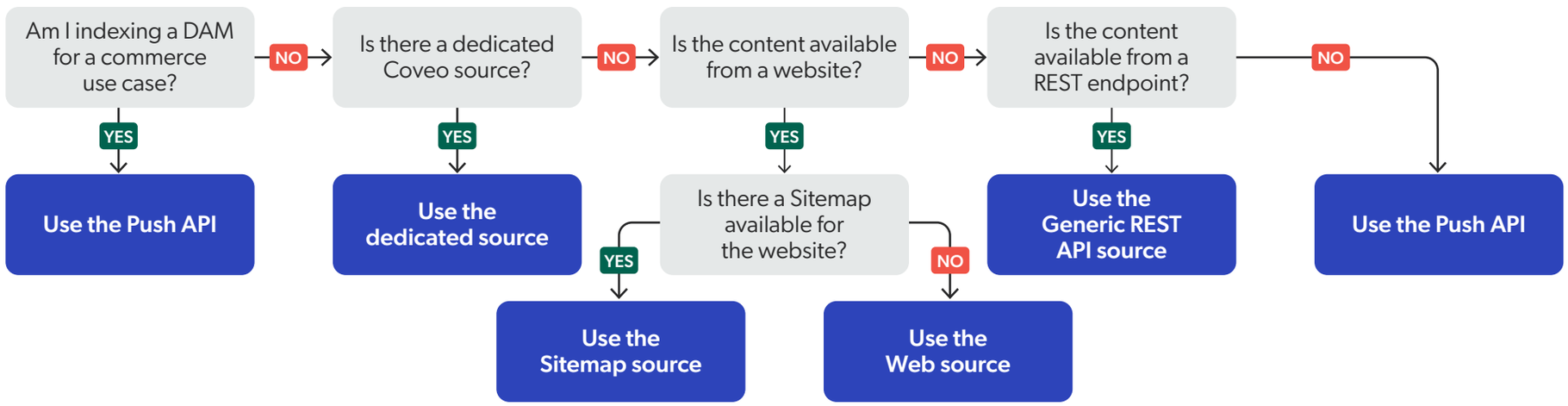
With Pull sources, you tell Coveo which page or endpoint needs to be accessed, and the frequency at which you want Coveo to visit those pages or endpoints. The vast majority of Coveo's sources use the pull mechanism.

Indexing table

Indexing method	Push	Pull
Description	Something external to Coveo pushes content to a Coveo endpoint	On a specific schedule, Coveo calls an endpoint to index content
Example sources	Sitecore, Push API	Web, Salesforce
Generic source	Push API	Generic REST API
Advantages	• You have full control over the content being indexed • You can update the content in Coveo as soon as it changes in the original source	• Coveo is in charge of the infrastructure and the scheduling • Configuration is all in one place
Disadvantages	• You need to control when content should be pushed • You typically have an extra piece of software to maintain • Typically takes more development time to configure	• Since the source refresh happens on a schedule, your content typically needs at least a few minutes before it gets updated

With these in mind, Coveo has several best practices when it comes to choosing which source to use when indexing Coveo content. Please follow these suggestions in order.

Note: there might be some exceptions to this. For example, if you need absolute control over all of your content and can't rely on an additional REST endpoint, you might want to simply leverage the Push API directly. When in doubt, please contact a Coveo representative or ask your question on the **Coveo Community** to discuss the requirements and look at available options.



Refreshing content

This section only applies if you are using a Pull type of source. For Push sources, the mechanisms are different.

When using a source with a Pull mechanism, you can edit the schedule and the frequency at which Coveo updates the content.

Coveo offers three ways of updating content for pull sources: Refresh, Rescan, and Rebuild. Not all options are available for all sources.

You can find more information about the different options here: **Refresh VS Rescan VS Rebuild**.

For the sake of a project, you typically need to know the frequency at which the content needs to be updated, and adapt it to your needs.

Pro Tip

Make sure you leave enough time between each rebuild so that Coveo can finish the first rebuild before starting the second.

Body indexing

A Coveo item normally has a body — a representation of what the item's actual content is. It normally holds only text, and is the core content of an item. Everything that is indexed in the body of an item is searchable in Coveo.

When indexing a web page, you typically want to ignore the header and footer of the pages you index. Otherwise, content in those areas (e.g., the copyright notice in the footer, or the text from the header menu) would become searchable, making relevant content harder to find.

The exact way of removing these sections differs depending on your source type, but include using **Web Scraping** (Web and Sitemap source) or having **BEGIN NOINDEX** tags (Sitecore). It's also possible to remove parts of the item in an **indexing pipeline extension**.

Good search starts with good content — make sure your content is clean in Coveo to get the best relevance.

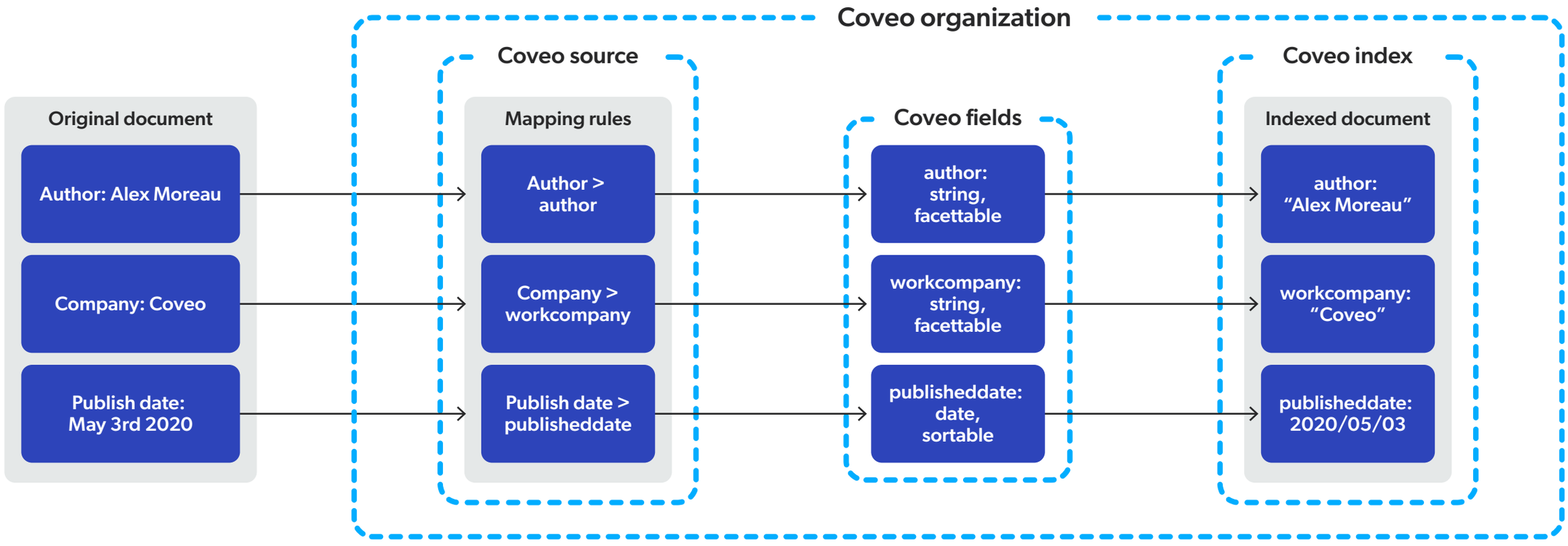
Mappings and fields

A Coveo item stored in your index contains metadata. This metadata is linked to fields, and becomes what we call a "field value." But how does metadata get added to fields? This is where mapping rules come in.

On your sources, you can set mapping rules that tell Coveo that a specific metadata on an item should be added (or mapped) as the field value for a specific field.

Upon source creation, a lot of mapping rules usually already exist for more generic metadata. When you have more metadata you want to index, you need to add additional metadata rules.

Mapping rules are source-specific and are added in the source configuration. Fields, however, are shared across the organization, for all sources. It is thus possible to have mapping rules from different sources all mapping metadata to the same field, despite the items not coming from the same source.



Field options

Fields dictate how metadata should be stored and used.

Fields can be of several different types, related to the programming types: strings (text), integers or decimal (numbers), or dates.

Fields also have options that you can change. The most important field options are:

- **Facet** — Lets you use the field for facet filters.
- **Multi-value facet** — Treats all values separated by semicolons as individual values, as opposed to one long string.
- **Sort** — Lets you use the field for sorting results. With string fields, sorting is alphabetical.
- **Free-text search** — Lets Coveo know that users should be able to search for content inside of that field, and return the result even if those words are not inside of the body of the document.

Pro Tip

While it can be tempting to set all of these options to true on all fields, doing so will impede overall performance, and can yield unsatisfactory relevance. It is better to enable those options only for the appropriate fields.

Permissions and securities

Documents can hold sets of permissions, ensuring only people who can access content can see them in search results.

While this is not necessarily needed in website or commerce use cases, it is essential in workplace and service contexts.

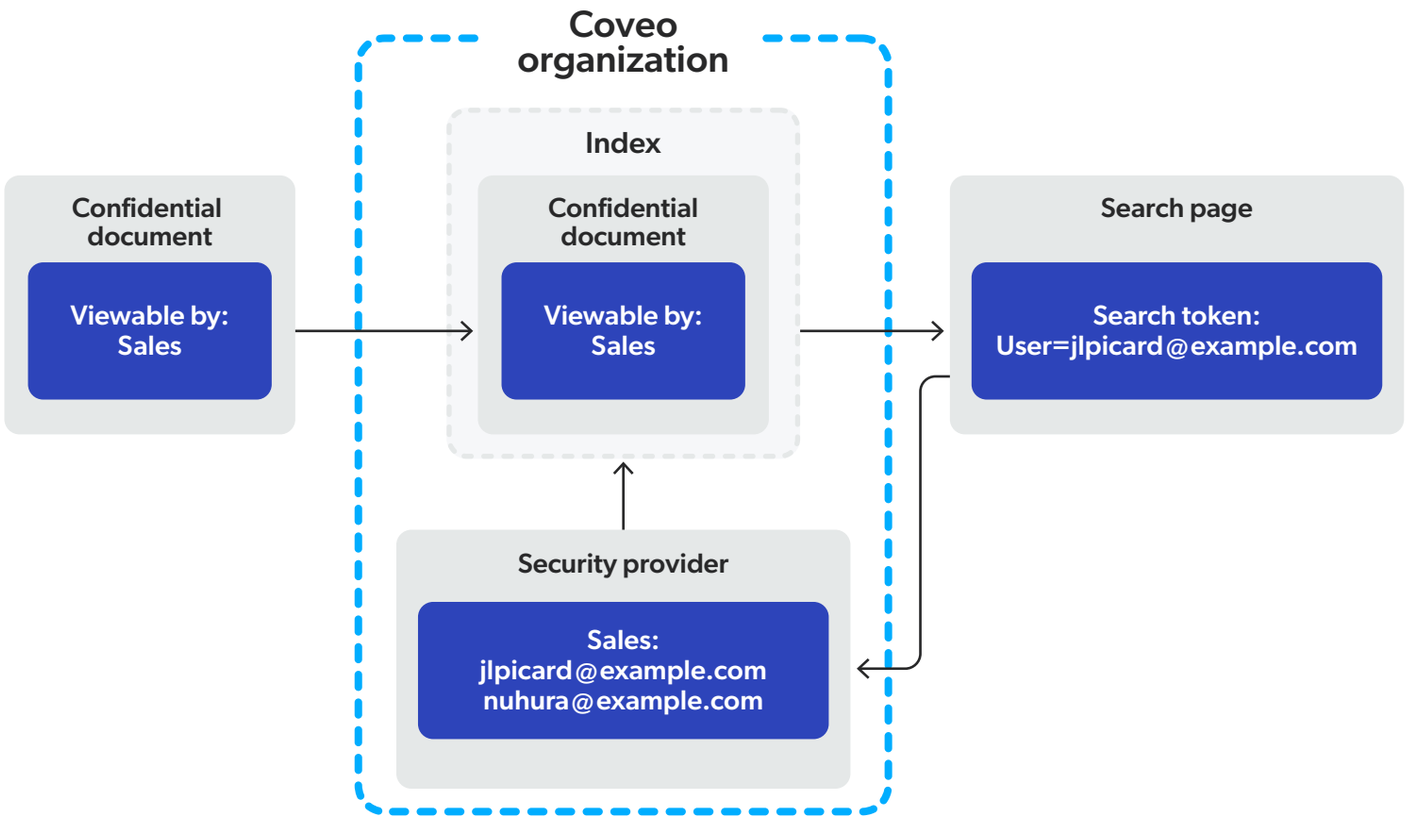
Coveo uses what is called early binding. We make the request to the index with the permissions, returning only results that a user has access to. This is contrary to late binding, which filters out content the user doesn't have access to after making the query. Early binding ensures more relevant results and, in the case of Coveo, a faster and more consistent experience.

Permissions are set at the item level when indexing, and are set on the search token when querying.

When you are indexing content from one of the dedicated source connectors, you typically don't have to worry about additional configuration to get permissions; a Salesforce source will natively get the permissions from Salesforce and apply them on each item, as would a SharePoint source for SharePoint content, etc. However, for generic sources that include permissions (such as when using the Generic REST API or the Push API), you will need to ensure that the right permissions are added to the indexed items. Additionally, you will need to remember to add a **security cache**, and push your security model there.

Coveo's permissions can be individual (user-level) or group-based. For example, you could say that a document can only be accessed by jlpicard@example.com, or you could say that anyone in the group SALES can access a given document. The same applies to denied permissions.

Depending on your method of indexing, the exact way to add permissions can differ.



Indexing pipeline extension

The content you index is rarely perfectly formatted with consistent metadata values across different source types. To get the most out of your Coveo implementation, you need consistent values in your fields with well-indexed content.

Indexing pipeline extensions (IPEs) can help you bridge that gap. IPEs are Python scripts you can write to modify or reject an item at indexing time.

IPEs can read and modify the body and fields of a document, create new fields, or change permissions. For example, you can call external APIs in your IPE to augment your metadata.

Note: IPEs can only run for a maximum of five seconds; make sure you don't call too many slow APIs.

IPEs can be used to exclude content from the body of the document (pre-conversion) or to add metadata (post-conversion).

For more information on IPEs, including code examples, see **Indexing Pipeline Extension Overview**.

Building a **Search Page**

Once your content is available in your index with the proper metadata, it's time to decide how you're going to query that content.

Coveo offers multiple alternatives to help you build search pages:

- **Full Coveo Integrations** — These integrations are built by Coveo, and offer a more intuitive and better integrated experience with your system of choice. Those solutions include Coveo for Sitecore, Coveo for Salesforce, Coveo for ServiceNow, and Coveo for Adobe Experience Manager.

  This option is almost always the best solution when integrating Coveo with its appropriate system.

- **Coveo JavaScript Search Framework** — This framework is built and maintained by Coveo. It allows you to quickly build a search experience using simple HTML tags to load an assortment of components like a search bar, search results, facets, and much more.

  This option is typically the best approach when integrating Coveo on a website or in a system that supports your typical web technologies (i.e., HTML, JavaScript, and CSS).

- **Coveo Headless** — This redux-based toolset allows you to integrate Coveo in already existing JavaScript frameworks, such as React or Angular.js. It should be considered mostly if your website or application is already running such a framework.

  This option is typically preferred when your site heavily relies on a JavaScript framework and you want the Coveo integration to follow the same practices.

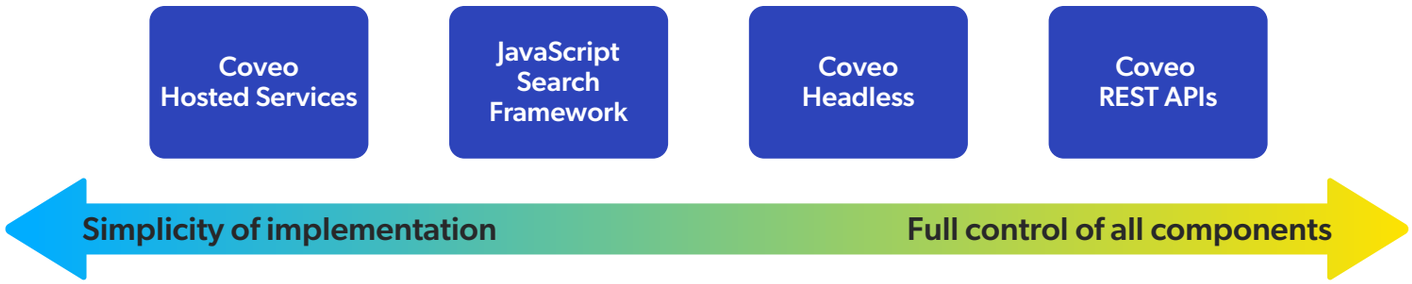
- **Coveo Hosted Services** — These include the Hosted Search Page and the In-Product Experience integrations. They leave the management of the design of the search page on the Coveo side, loading the appropriate JavaScript and HTML typically through a client call.

  This option is typically preferred if you would like to have everything managed from the Coveo Administration Console directly.

- **Coveo API calls** — The last option is to call the Coveo Search API directly. With this option, you have full control over what is being sent to Coveo, and how you want to display the information. However, it comes at the cost of additional development time and effort.

  This option is preferred in very complex front-end pages or apps, or when using Coveo in a mobile or non-web based application.

Depending on your project, you can opt for any of those options.



The majority of the time, using the JavaScript Search Framework is the preferred method, as it offers a lot of power and flexibility while staying easy to implement.

This project guide focuses on using the JavaScript Search Framework. If you are looking to use a different method of implementation, please see the following links:

- **Coveo for Sitecore**
- **Coveo for Salesforce**
- **Coveo for ServiceNow**
- **Coveo Headless**
- **Direct Search API**

Pro Tip

A new framework called Coveo Atomic based on Coveo Headless came out not long before this project guide. While this guide does not cover its use, its complexity and control is comparable to the JavaScript Search Framework.

For more information on Atomic, see Use the Coveo Atomic Library.

Understanding search hubs

No matter the solution you choose to search content, there is an important concept to understand: search hubs.

Search hubs are the way Coveo knows which search interface your queries are coming from. Search hubs should be human-readable values, and can contain spaces. Ideally, it should be prefixed by the website or brand name, followed by the purpose of the search page. For example, “ACME Search” and “ACME News” are perfect search hub names.

While they are used by Coveo Machine Learning, they are also used by analysts looking at the analytics in your organization to determine how users are leveraging search, hence the need for human-readable values.

Most of the time, each search page should have its own unique search hub. Setting the search hub is done at different places, depending on your method of implementation:

- For Coveo integrations, please refer to their appropriate documentation.
- For the JavaScript Search Framework, the search hub value is set on the **CoveoAnalytics** component.
- For Coveo Headless, the search hub value is set in the engine properties, in the **search object**.
- For Coveo Hosted Services, the search hub will by default be the name of your page or component.
- For Search API, the search hub is a parameter on the search request (searchHub).

Understanding the query

When a query is sent to Coveo, many parameters can affect what content is returned, acting as filters.

The following parameters are the ones most likely to affect which content is returned. They all follow the **Coveo Query Syntax**:

- **q** — The q (query) parameter is the query the user entered. It should always be only what the user has typed.
- **cq** — The cq (constant query) parameter adds an expression for filtering content. It should be used for filters that do not change on a page (e.g., a people search page will only ever search for people, so the filter should be in the cq).
- **aq** — The aq (advanced query) parameter adds expression for filtering on content. It should be used for filters the end user added (e.g., when a user selects a value from a facet, an expression to filter the content is added to the aq).

The difference between the **aq** and the **cq** is a matter of performance optimization, as well as query execution.

- **dq** — The dq (disjunction query) parameter adds an "OR" expression to the query. In other words, for a result to be returned, it must fit the filters of q and aq added together, or only match the dq. This parameter is typically not used as much as the other ones.
- **lq** — The lq (long query) parameter should be used when you are sending a lot of text for the query (e.g., a case description). It uses machine learning to determine which words are more important, and uses those terms to perform a query. To use lq, you must enable the ITD option on the ART machine learning model in the appropriate pipeline, in the Coveo Administration Console.

In the end, the query filters would look like this:

```
((q AND lq AND aq) OR dq) AND cq
```

Those filters are typically added in three places:

1. On the request itself, as part of the parameters.
2. In the search token, where a "filters" section can add non-removable filters.
3. In the query pipeline, in the **filters section**.

Creating a test search page

The best and easiest way to create a quick search page with Coveo is to leverage the **Hosted Search Page** service offered in your Coveo Cloud organization.

Hosted search pages use the Coveo JavaScript Search Framework to create search pages, and include a tool called the Interface Editor. This tool is a drag-and-drop UI that allows you to create search pages easily and intuitively.

Once you have built a page to your liking, you can switch to the "Code View" to see the HTML markup of your page.

Note: Search pages in a "Trial" organization work differently. To get the experience described in this guide, you would need to create a "Classic" search page.

Understanding the JavaScript Search Framework

The JavaScript Search Framework is an **open source framework** built and maintained by Coveo that allows you to create full-featured search pages with simple HTML tags.

At its core, the framework offers a variety of components that, when initialized, render either visual controls that end users can use to navigate through the content, or configuration changes to alter a request.

The **Search Interface** component is the most important component of the framework. The Search Interface is essentially the container for the framework; it is the interface that is initialized on page load, and only the components inside the interface are rendered by the framework.

Components almost always have options to modify the way they behave. Those options are added as data attributes to the HTML tags.

Initializing the framework

Initializing the framework is fairly simple. You need:

- A Coveo Search Interface, ideally with a unique ID attribute
- A Coveo Cloud organization
- Either an API key to search, or a search token retrieval mechanism

To learn more about how to initialize the framework, see **JavaScript Search Framework Endpoints**.

Search tokens

With Coveo, you can either query for content using an API key or a search token.

Using an API key is the simplest way to do it; you simply go into your Coveo organization and create a key with limited permissions. However, this method does not allow you to query for secured content, and you cannot enforce filters to always be applied, meaning that anyone with that key could query any anonymous content in your organization, even if that content is not normally surfaced on your search page.

To avoid this, you can use a search token. Search tokens are Coveo's way of adding securities and enforcing parameters when someone is querying.

In order to generate a search token, you need to set up an endpoint that acts as a middleman between the client and Coveo. This endpoint should be able to fetch the permissions of the user requesting it, and then make a request to Coveo, returning the token with the appropriate permissions.

For more information on this topic, see **Search Token Authentication**.

Styling

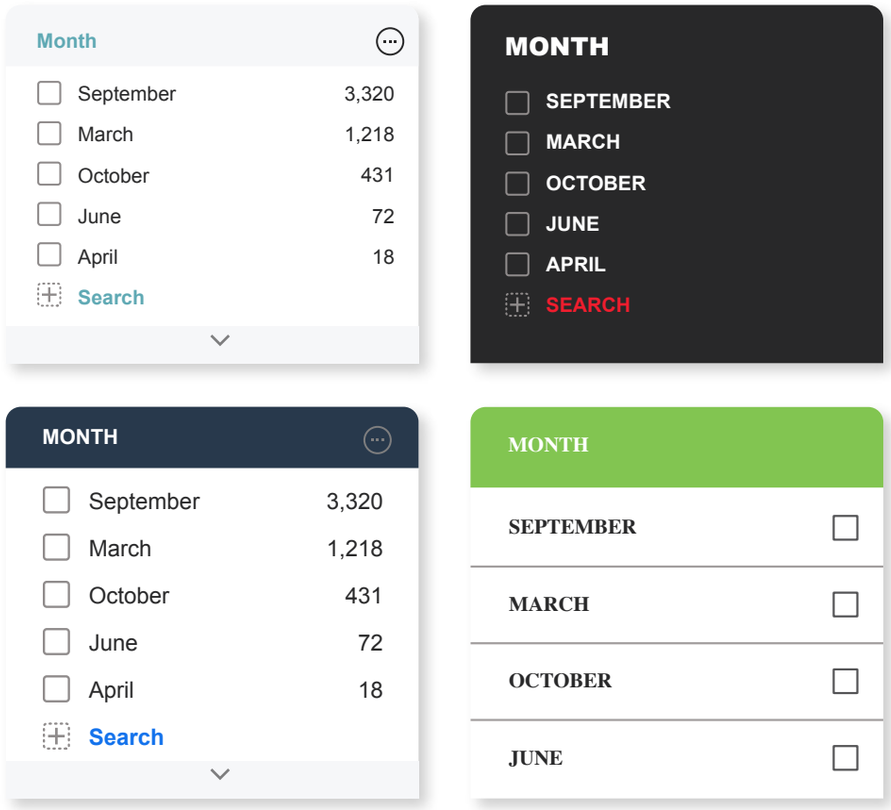
Styling components with Coveo follows the typical CSS best practices. There is no special way of styling components.

The best practice is to add an additional class to your CoveoSearchInterface, so that overriding the style can easily be done.

You should always load the default styling from Coveo. Not doing so may lead to weird styling issues, such as ARIA landmarks being visible on the screen instead of being limited to screen readers.

For more information, see **Styling the Coveo JavaScript Search Framework**.

Facet styling overriding examples



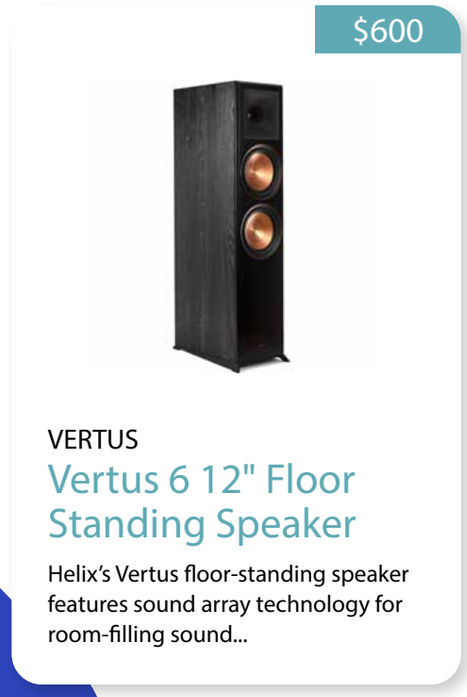
Result templates

Result templates dictate how results should be displayed on the page.

Result templates have conditions on them, which are evaluated for each result when rendering them in a search interface. For example, you can have a template that is used only when the result is a PDF file.

Templates can be customized to look however you want, and include their own components to display images, field values, or perform specific actions.

Result templates should always include the **CoveoResultLink** class on all elements that, when clicked, opens the document; this component takes care of sending the click events to Coveo.



Result template example with out-of-the-box components

Creating a global search box

A global search box is a Coveo-powered search box that sits in the header of your website or application and provides suggestions of queries to users as they type, before redirecting them to a full search page when a query is entered or selected.

A basic global search box includes two main components: a CoveoSearchbox component, and a CoveoAnalytics component. The sole purpose of the CoveoAnalytics component is to add a search hub value to the query suggestion calls. The search hub value should be the same as the search hub value of the page it redirects to, to ensure that all query suggestions proposed to end users have results.

For more information on how to add a global search box, see **Create a Standalone Search Box**.

Leveraging events

The Coveo JavaScript Search Framework comes with many **JavaScript events**, which you can leverage to modify aspects of the implementation.

Hooking yourself on those events allow you to add code or logic before or after the initialization of Coveo on the page, or to modify a query, analytics, or query suggest call before it is sent to Coveo, or just as it comes back before it renders on the page. This typically allows you to add behavior customization without creating entirely new components.

Creating custom components

Leveraging Coveo events is sometimes not enough, and you want a new component with a behavior that does not exist in the Coveo JavaScript Search Framework.

A good example for needing a custom component is if you would like to reuse the new functionality in multiple places with different options, or when you think the behavior you want can be used in other projects.

Pro Tip

Coveo already offers a few examples of custom components with Coveo Turbo, as well as a framework to create new components. You are encouraged to use the same recipe when creating your own components.

Localization

By default, the Coveo JavaScript Search Framework is in English. However, the framework supports **many other languages**, with out-of-the-box translations for its out-of-the-box components.

To translate your search interface in another language, simply load the appropriate culture file. This translates all the out-of-the-box Coveo strings. However, your facet and field values are typically not automatically translated. In this scenario, you need to add those strings to a **translation dictionary**.

Adding recommendations

Coveo Recommendations are a special breed of search. Instead of showing users content based on their query or on what is popular, recommendations show users content they could be interested in based on what other people have viewed or interacted with.

There are two different types of recommendations: Product Recommendations and Content Recommendations.

- **Product Recommendations** should be used in commerce scenarios, typically when you want to use more than just page views to evaluate a recommendation, such as items added to cart, purchased items, and viewed items.
- **Content Recommendations** considers mostly page views as analytics data to work from, and is ideal for recommending reading material, such as documentation pages, knowledge articles, blog posts, news items, or product detail pages.

Recommendations work in the same four steps:

1. Track your page view and potential other events, sending them to Coveo Analytics.
2. Add the appropriate machine learning model to your Coveo organization.
3. Create a dedicated query pipeline for your recommendations, associating your machine learning model to it.
4. Add the Recommendations component from the JavaScript library to your site, setting the appropriate search hub to go through the right query pipeline.

There are specific things to keep in mind, depending on your use case and on the model you're using. For more information, refer to the appropriate Coveo documentation on the subject.

Improving **Relevance**

Once you have content in your index and a search page to query it, you should work to improve relevance. The most powerful tool in the Coveo toolbox for this is machine learning.

Machine learning models

Coveo offers multiple machine learning models, and is continually developing and releasing new ones.

The two most important models to improve relevance are **Query Suggestions** and **Automatic Relevance Tuning (ART)**.

Machine learning basics

At its core, all Coveo Machine Learning models work from the same loop: the more user analytics data you get, the smarter and more accurate Coveo Machine Learning becomes.

For most models, the required events are searches and clicks. Searches tell Coveo what people have searched for and how they've searched for it, while clicks tell Coveo which searches were successful, and how successful they were.

Coveo separates learnings according to three main criteria: language, search hub, and tab. Those criteria act as hard separation; learning from one search hub will not be shared with another search hub.

You can also send **custom contexts** to perform "soft" separation of learnings, where the models learn from all events but weigh similar contexts together.

Query Suggestions

The purpose of Query Suggestions is to propose queries to your users. The model presents queries that other people have done that were successful. In order for a query to be successful, it needs to have led to a click.

Query Suggestions is typo-resistant based on a confidence rating. The more a query is common, the more it's likely to be suggested, even if it's horribly misspelled.

Query Suggestions is not aware of the type of content that was clicked, only that someone searched for a term and that this term yielded at least one result that the user clicked on.

Automatic Relevance Tuning (ART)

The ART model is at the core of improving relevance at Coveo. This model looks at what people search for and what they click on. If enough people search for something and click on a particular result, it starts to boost that result so it shows up higher in the result list. At scale, this model can ensure that people can find what they are looking for within the first few results on the page.

By default, ART boosts up to five results; this can be adjusted in the **settings** when associating a model.

ART can also inject results that do not fit the current user query, but were successful for other people who made the same query in the past. This helps reduce the number of queries without results, and can save valuable time for your end users.

For example, let's say your user searches for "apartment insurance." However, in your index, there is no such thing as apartment insurance, those terms not being found together in any of your documents. What the user is actually searching for in that case is "home insurance," a query that returns a lot of results. The first few users would realize their mistake, then correct "apartment" to "home." ART can pick up on that trend, and start to inject the result "home insurance" when a user enters the query "apartment insurance," since ART is now confident that people who search for "apartment insurance" are actually looking for "home insurance."

Dynamic Navigation Experience (DNE)

Dynamic Navigation Experience is a machine learning model that handles facet and facet filtering. It can reorder facets, make them appear/disappear based on how used they are, reorder the values in each facet, pre-select values depending on the user query, or automatically boost categories of results based on the current query.

DNE is extremely useful and powerful in commerce use cases and in complex manufacturing. However, it can be harmful in other scenarios, when people expect a consistent user experience, such as in workplace use cases.

Understanding Query Pipeline

Query Pipelines are a feature of Coveo that allows you to add rules that can change what is being searched, what is being returned, and the order in which things are returned.

You can create multiple pipelines and add **conditions** to them, so that queries from specific search pages go through that pipeline.

Any time a query is made to Coveo, it has to go through a query pipeline. There are three things that can determine which pipeline a query goes through.

In order:

1. A query can contain a **pipeline parameter**. If there is a pipeline with the same name as the value for that parameter, the query will go through it.
2. The query evaluates all conditions on the pipelines, in order, and will go through the first one whose condition fits. Pipelines without conditions are skipped at this stage.
3. The query goes through the pipeline tagged as “default”.

It is typically recommended to use the second option. Most of the time, you want to have conditions based on the search hub.

Query Pipeline rules

Query Pipeline offers you control over the query being made by the user. Here is a quick overview of the most important ones.

Machine learning

In order for machine learning to take effect on a search page, you need to **associate the model to the query pipeline**.

You almost always want both a Query Suggestion and an ART model associated with your pipeline, except for recommendation pipelines which should have recommendation models.

Multiple pipelines can share the same machine learning model; it is in fact encouraged that you do that in most scenarios.

Thesaurus

Thesaurus rules (also known as synonyms) allow you to change the terms to either expand or replace the query that the user has entered.

Thesaurus rules are very useful when dealing with acronyms, initialisms, hypocorisms, and other abbreviations.

Thesaurus should not be used for expanding plural and grammatical expansions. Coveo already stems words, and will expand “search” with “searches,” as well as “write” with “writing” and “writer” for example.

Result Ranking

Result Ranking rules allow you to change the ordering of the results, through either boosting or burying a specific or a category of results.

There are two types of ranking rules:
Featured Results, and Ranking Expressions.

- **Featured Results** boost specific results so that they always appear first in the result list when a specific query is made, or another condition is met. Featured Results are extremely powerful, and should be used sparingly. You also want to make sure that no two featured results rule overlap, as it could otherwise lead to confusing results.
- **Ranking Expressions** boost a category of results that fit a specific expression by any amount of points that you decide. Entering a negative number is possible and sometimes even encouraged; negative scores bury results, making them appear lower in the result list, without quite making them disappear.

Ranking Expressions should stay between -100 and 100. Anything beyond those limits can start to hinder machine learning (i.e., ART) from showing the results people are actually clicking on, leading to a less successful implementation.

Ranking expressions can refer to parts of the query using the $ symbol. For example, `$query` refers to the user's query. This way, you can add a rule to boost specific object types if the user's query is exactly the name of one object type, as such:

```
@objecttype==$query
```

For the full list of available values, see **Query Pipeline Language (QPL)**.

Triggers

Triggers are rules that can make something appear or happen on the search page when a condition is met.

There are four different types of triggers:
Notify, Query, Execute, and Redirect.

Almost all of the time, you will either use a Notify or a Redirect trigger.

Notify triggers display a message at the top of the result list when a condition is met. This can be used to warn users of an upcoming event, for example, or to let users know that a thesaurus rule was applied to modify their query.

Redirect triggers redirect the user to a different URL when a specific query is entered. For example, you might want your users to be automatically redirected to your careers website when they search for "jobs," or you might want to redirect users to your support portal if they enter "help."

The other two trigger types (Query and Execute) are typically better handled through thesaurus rules and front-end JavaScript code, respectively.

Pro Tip

Remember to add the CoveoTriggers component on your search page if you want trigger rules to take effect.

Filters

When sending a query, you can set **filters**, so that you are only returning a subpart of the full index. Query Pipelines also allow you to change the filters of a query, giving you an easy no-code way of modifying and updating filters.

Other

There are other features of the query pipeline that may be less used.

Stop Words allow you to add words that will be completely ignored by the index when querying. Do note that Coveo already uses Inverse Term Frequency and already has a long list of ignored profanities, so adding short frequent words or profanities might not be useful for relevance.

Ranking Weights change the base relevance of the index. They have been tuned for document search. You typically want to change them in Commerce scenarios, but for website search or for documents with a decent amount of text in it, leaving the default parameters should be good.

Query Parameters allow you to change the options that a query passes. This is a more advanced section, and should only be tackled by someone who understands the basics of the Coveo Search API.

A/B testing

It can sometimes be unclear if adding or removing a query pipeline rule will help your search relevance or hinder it. For this reason, Coveo has a built-in A/B testing mechanism.

A/B testing of Coveo allows you to send a percentage of your traffic in one pipeline, and send the rest in a second pipeline. Over time, you will be able to report on the analytics data coming from your users, and determine which pipeline performed the best.

Common **Pitfalls**

The Coveo engine allows you to do many things. However, not all of those things will yield a good user experience, and can even harm the relevance of your search. The following pitfalls should be avoided when using Coveo.

Search as you type

Presenting results to users as they are typing performs one query per keystroke, with results presenting slower the faster a user types. Instead of presenting results to the user as they are typing, Coveo recommends presenting queries (using Query Suggestions). This practice is aligned with the best practices of industry leaders — and is the best way to ensure typo resistance and machine learning-powered improvements.

Wildcard search

Wildcard is the process by which typing the beginning of a word will return results for that word and any other words that start with the same letters.

Searching with wildcards comes at a high price in terms of performance, taking sometimes up to 50 times longer to return results. Typically, using Query Suggestions tends to alleviate the need for wildcard search, as words that would complete the query are suggested to the end-user in order of relevance.

Another use case for wildcard search is for allowing partial SKUs to be searchable in Coveo. However, a **better and faster approach** would be to separate the SKU into its individual parts and index them in a free-text searchable, multi-value Coveo field.

Too many rules

Query Pipelines give you a lot of control over the relevance of your search interface. However, relying too heavily on manual rules can harm machine learning and prevent it from being able to boost what users are looking for.

Having too many featured results can be particularly harmful to search results. Having too many thesaurus entries (i.e., multiple pages worth of thesaurus entries) can also start to affect relevance.

In general, when using query pipeline rules, keep in mind that Coveo is already trying to improve the search experience, and that the rules you add are crutches to help Coveo learn.

Sending the right analytics

For machine learning to work, Coveo needs both search and click events. Sometimes, when customizing the result template, the **CoveoResultLink** component will be removed from the default template, replaced by the **CoveoFieldValue** component to display a title.

However, removing the CoveoResultLink component prevents the search page from sending a click event to Coveo, leading to incomplete tracking, and preventing machine learning from learning.

Similarly, search and click events require an OriginLevel1 (search hub) value to let machine learning learn. This option is set on the **CoveoAnalytics** component, but is not always populated by default. Missing this option will prevent machine learning from learning.

Using Coveo outside of relevance

Coveo is a relevance engine, and is best utilized when the experience you are powering requires results in a contextualized, relevant order.

While Coveo can be used for simple listing sorted by date, as an example, relevance should still be the default option. Also note that sorting results by something other than relevance will prevent ranking rules from your query pipelines and machine learning re-ranking from applying to the page.

Going to Production — **Checklist**

Indexing

- ☐ Did you index all the content you need to display?
- ☐ Did you only index content shown in at least one search page?
- ☐ Are all the sources building/rebuilding successfully?
- ☐ Did you remove the header/footer of your content, so only the relevant content is indexed?
- ☐ Did you adjust the schedules of your pull sources to an appropriate time frame?
- ☐ Is your metadata consistent for result display and facet use?
- ☐ Did you index all the metadata you need, for filtering, boosting, and displaying results?

Permissions

- ☐ If you have permissions, did you ensure the right permissions are on your items?
- ☐ If you have permissions, did you ensure the search token contains all the permissions of the querying user?

Search page

- ☐ Are you tracking analytics on your search pages?
- ☐ Are you sending a search hub (originLevel1) on your analytics call?
- ☐ Are your results displayed differently depending on the type of content it is?
- ☐ Are you using Coveo for the global search box?
- ☐ Does your global search box share the same search hub value as the page it redirects to?
- ☐ If you have multiple languages, have you implemented the appropriate culture files to translate the UI in the appropriate languages?
- ☐ If you have multiple languages, have you translated your own strings (e.g., field values, facet titles, result template labels, etc.) in the appropriate language?

Machine learning and pipelines

- ☐ Do you have a pipeline architecture with conditions?
- ☐ Did you create your machine learning models (ART and QS)?
- ☐ Did you associate the machine learning models with all appropriate pipelines?

coveo™
The Relevance Company

Learn more about Coveo

Coveo is the world's leading cloud-based relevance platform. The Coveo Relevance Cloud™ uses applied AI to deliver relevant experiences in all digital interactions, from search to recommendations to personalization.

Contact us