Building Search into Your Applications with Amazon CloudSearch

**SPL-74 - Version 3.0.10**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Overview**

This lab will introduce you to the basic concepts behind search engines, what they do and how they do it. You will learn how to create a search domain, configure it, upload data, build queries, and tune your ranking with Amazon CloudSearch. You will explore the features of the AWS console for Amazon CloudSearch, learning how easy it is to get started with CloudSearch. When you complete this lab, you will have all of the tools necessary to add search to your application.

TOPICS COVERED

This lab covers:

* Understanding what search engines are and how CloudSearch works.
* Managing and configuring search domains in CloudSearch, including creating a search domain and index data.
* Querying a search domain to retrieve documents in multiple different ways, including queries that feature facets, geospatial searches, and field weighting.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Note:** A hint, tip, or important guidance.
* **Caution:** Information of special interest or importance (not so important to cause problems with the equipment or data if you miss it, but it could result in the need to repeat certain steps).
* **Security:** An opportunity to incorporate security best practices.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Copy edit:** A time when copying a command, script, or other text to a text editor (to edit specific variables within it) might be easier than editing directly in the command line or terminal.

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab, you should:

* Be familiar with AWS and the AWS Management Console.
* Have basic knowledge of HTML, JavaScript, and JSON.
* Have access to a computer with Microsoft Windows, Mac OS X, or Linux (Ubuntu, SuSE, or Red Hat).
* Have access to a modern internet browser such as Chrome or Firefox.

No previous search engine knowledge is required.

DURATION

This lab requires **120** minutes to complete.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**WARNING:** **Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

**Task 1: Managing a Search Domain**

In this task, you will create your first search domain. Your search domain will hold the collection of data you want to make searchable.

For this lab, you will create a search domain that will hold songs. Each document in the search domain represents a unique song, with fields such as song title, album title, artist name, the location where the artist was born (or where the band was created), the year the song was released, and the genre of the song.

1. To the left of these instructions, copy the value of **InstanceIPAddress** to a text editor.
2. In a separate browser tab, paste:

InstanceIPAddress/cloudsearch/songs-lab.json

1. Replace **InstanceIPAddress** with the value that you copied to your text editor.
2. Press **Enter**.

 Your URL should look similar to: *54.200.33.120/cloudsearch/songs-lab.json*.

1. Download the songs-lab.json file to a local folder on your computer.

 This file contains 2296 different songs in JSON format that will be used during this lab. You will learn more about the file format later.

CREATE YOUR FIRST SEARCH DOMAIN

1. Return to the AWS Management Console.
2. At the top of the AWS Management Console, in the search bar, search for and choose

CloudSearch

.

1. Choose **Create a new search domain**.

 At this point, a **Create New Search Domain** pop-up window appears.

1. In the **Search Domain Name** textbox, enter

handsonlab

.

1. Choose **Continue**.
2. For **How do you want to configure your index fields?**, select  **Analyze sample file(s) from my local machine**.
3. Choose **Browse…** or **Choose File**.
4. Navigate to the songs-lab.json file you downloaded earlier.
5. Choose **Open** to select the file.
6. Choose **Continue**.

 The wizard will upload the file from your local computer and analyze it, automatically suggesting how your search domain index should be configured. The suggested configuration is an educated guess according to the data found in the json file. You will make some slight modifications to it.

1. In the **artist\_name** row, for **Type** column, select **text** from the dropdown menu.

The *literal* field type provides exact matching for its contents. CloudSearch tokenizes strings in text fields, providing the ability to match individual words. It makes more sense to use a *text* field type for artist\_name, because artist’s names are normally composed of more than one word, and you might want to search using only one of these words. For example, you might want to search for “rolling stones” music with just the term *stones*.

1. In the **location** row, for **Type** column, select **latlon** from the dropdown menu.

CloudSearch will treat this field as containing a latitude and longitude, allowing you to perform geospatial queries against this field.

1. In the **place\_name** row, for **Type** column, select **text** from the dropdown menu.
2. In the **Facet** column, de-select:

* **artist\_hotness**
* **artist\_id**
* **location**

1. In the **Facet** column, leave  **terms** and  **year** rows selected.
2. In the **Highlight** column, de-select  the check boxes for all the rows.
3. Choose **Continue**.
4. For **Set my policy to:** option, select **Allow open access to all services**.

 This is not the recommended setting for a production environment, because anyone would have permission to search and upload documents to the domain, but it’s fine for the duration of this lab.

1. Choose **Continue**.
2. On the confirmation page, review the information and choose **Confirm** to start the process.
3. Choose **OK**.

The search domain dashboard will open, and your search domain status will be **LOADING**. It will take several minutes until your domain status changes to **ACTIVE**.

While your domain is being initialized, continue to the next task where you will learn about search engines in general and CloudSearch specifics. You then walk through the AWS Console for CloudSearch while your search domain is being loaded and activated.

**Congratulations!** You successfully created your first search domain with Amazon CloudSearch.

**Task 2: Introduction to CloudSearch**

In this task, you will understand the basics around search engines and Amazon CloudSearch.

WHAT ARE SEARCH ENGINES AND HOW DO THEY WORK

A search engine is something like the *concierge* of data. People come to the engine with questions or *queries* and receive an answer that is the best answer the concierge knows that solves the need behind the information goal. The search engine matches the entered text against the content it has, scores and ranks the results, and returns them to you. Search engines use ranking functions to mix statistical information on the text match with *a priori* information to evaluate the possibilities, and sorting to retrieve the best ones.

**Documents**

At the heart of search is the *search document*. Documents are objects that are loaded into the search engine on one end from an external source or sources and sent out from the search engine at the other end as a response to a search query. Search documents are different from what you would normally think of as a “document” or file when you are working in a word processor. They are composite objects, gathering different kinds of information, possibly from different sources, that describe the underlying object. The information in the document is structured into *fields* and *values*.

The most important field is the document’s ID: this identifies the document uniquely within the search engine. Along with the ID, you determine the set of fields based on the use case. For web search, the set of documents is a set of web pages. The fields would include the title and body of the page and the URL. Another common use case is eCommerce. For eCommerce, the documents represent products. The fields include the name of the product, a description, price, and inventory status.

The structure of the fields provides the means for allowing users to specify their queries more concisely. In an eCommerce use case, you can allow searches matching the name or brand of the item. If you want to buy a Rolex watch, then the application can match “Rolex” the brand to be more precise in retrieving Rolex products.

**Queries**

When users perform searches, they most frequently specify their search by typing words into a search box. Specified this way, the queries contain the keys that form the basis of the retrieval of documents to match them. In the simplest case, the query words, commonly called *terms* or *tokens*, are matched against the text that is stored in the documents’ fields. When all of the user’s terms match something in a document, that document is one possible result.

In more complex cases, directly in the search box, or indirectly through UI elements, users can also specify more complex, Boolean logic for matching. For instance, a user might want a shirt but only a blue or red shirt. The UI will construct a query for the term *shirt* and a color with the value *blue* or *red*.

So how does a search engine find matching documents? Let’s peel back the covers a bit and talk about the data structures and algorithms.

At the heart of all search engines is an index. If you were looking for an article on United States presidents in an encyclopedia, you could start on page 1 with the first article and start reading forward, turning every page until you got to *P – Presidents, United States*. This is a fine method, and perhaps it would be a fun way to spend an afternoon (or several!). However, it’s not a very efficient way to find that article. You might get creative and execute something like a binary search, pulling books out until you found the Ps and then bouncing around on pages, gradually getting closer, until you hit the article you were looking for. This method is more efficient, but not blindingly fast.

Fortunately, the authors of the encyclopedia have provided an index. The index allows you to locate the word or concept that you’re looking for quickly and then provides you with a list of articles that contain those words. You can then examine those articles more closely to figure out which article has the information you need.

At the heart of every search engine is a similar index. Sometimes this is referred to as the *inverted index*. Here, the term *inverted* is a term of art, and doesn’t mean the same thing as an inverse function. It describes a data structure that maps terms to the document IDs of the documents that contain those terms (as opposed to a data structure that maps document IDs to the terms they contain). The terms are stored alphabetically, making it very low cost computationally to retrieve the list of document IDs that match a particular term. For search engines, there is typically more than one index; in fact, each field usually has an option to create an index for that field. The list of document IDs for a term is called that term’s *posting list*, and it usually contains not only the document IDs, but also the location of the term within the document.

When queries arrive at the search engine, it looks up every term in the query in the index, generating a set of posting lists. It then applies the Boolean logic of the query to merge together the posting lists to generate a set of document IDs for documents that satisfy the query’s logic.

The match set is the set of all documents that satisfy the logic of the query, but a search engine does more. Remember, the function of the engine is to bring the best matches for the user’s information goals; you don’t want to wade through every shirt, just the ones that are in season, in fashion, of high quality, etc. So, the engine then computes a score for each document, using some statistical properties of the terms in the documents often augmented with numerical information from the document.

What comes out of the search engine is minimally the set of document IDs for the ranked set of documents. The UI or application layer can then use that set of IDs as keys into the source systems that hold the original data to pull that out for display to the user.

**Scaling a search engine**

Search engines use compute resources and storage resources. The index grows with each new term, adding a posting list for that term, as well as each document that is added to the posting lists of every term that it contains. The indexes themselves can be held entirely in RAM, but more often they are stored on disk, and portions are swapped in as necessary. As the total index size grows, the capacity need of the disk (and RAM) grows with it.

Compute resources are used to process queries—calculating the match set, scoring the matches, and sorting the results. As queries grow, the compute resources must grow to accommodate this processing.

Search engines can be scaled vertically: you can start with a single, smaller machine, and as the data size and query volume grow, move the engine onto successively larger machines. As with all vertical scaling, there’s a limit to the search problem that can be solved with a single machine.

As data size grows, you can employ *horizontal scaling*. The index is sharded, and shards (also called *partitions*) are deployed to multiple nodes. When queries come in, they are picked up by a process called a *collator* or *aggregator* on one of the shards. The collator then calls a partition process on each of the partitions to compute results local to that partition. Each partition computes a match, scores, sorts, and returns its results to the collator. The collator then re-sorts all of the partition results to get the final result for the query.

As query traffic grows, the partitions are replicated to provide additional compute power to process those queries. Each replica has a full copy of the index for its partition, and the collators send queries based on either round robin or, more intelligently, the current CPU load on the partition servers.

**Search basics round-up**

Summing up, search engines are the concierges of data. They provide a means of finding the best answer to the questions that we have, whether those questions are about where to stay, what to buy, or where to look on the web. The search algorithm computes matches from query terms to terms in the set of documents that are in its index. It computes a score for each matching document and returns the best match based on the scores.

Search engines scale vertically when there’s little enough data and traffic. As the compute needs exceed the capacity of the largest node, you can scale horizontally for data by partitioning the index and for traffic by adding replicas of partitions.

WHAT IS CLOUDSEARCH

Amazon CloudSearch is a fully managed search service in the cloud that brings to the search space many of the benefits that AWS brings in the hardware, storage and database spaces. It’s easy to get started with CloudSearch; you can create a search domain with just a few clicks in the AWS Management Console. CloudSearch provides users the ability to scale elastically to match their resource consumption to their application’s needs. There are no up-front commitments for CloudSearch, and pricing is hourly, based primarily on instance hours. This enables developers to innovate around search in their applications, quickly spinning up new search domains to experiment.

CloudSearch is currently deployed in all AWS regions except for GovCloud, so developers can easily deploy search servers that are near to their applications.

Now that you understood the basics around search engines and CloudSearch, let’s explore the CloudSearch console and its functionalities.

**Congratulations!** You successfully understood the basics around search engines and Amazon CloudSearch.

**Task 3: Exploring Amazon CloudSearch**

A WALK THROUGH THE AWS CONSOLE FOR CLOUDSEARCH

In this task, you take a quick walk through the AWS Console for CloudSearch.

1. Return to the CloudSearch Management Console.

The first thing you see is the CloudSearch Dashboard. This dashboard shows you all the search domains you have created for the selected region. You can see the status of each of your search domains and how many documents you have indexed on them. Because you just created your search domain, you will have 0 searchable documents, and it might still be in **LOADING** or **PROCESSING** state, which means your search domain is not ready to receive documents and serve these documents via queries yet. Your domain is considered ready when its status changes to **ACTIVE**.

1. To check the status, choose **Refresh**.

 Even if your lab is still *LOADING*, continue to explore the console while the domain is being prepared.

1. From the left navigation pane, choose the **handsonlab** search domain to access its dashboard.

You will see options to upload documents or delete the domain, the status of the domain, the number of documents indexed (searchable documents), fields (nine in this case), the search endpoint (where all queries should be directed), the document endpoint (where all update operations should be directed), the domain ARN (to be referenced within IAM policies for the domain), the API version that the domain uses, as well as other options to test and configure the domain itself.

Now look at the domain configuration options.

1. From the left navigation pane, choose **Access Policies**.

Here, you can define access policies for your domain, such as allowing everyone to access all of the services (query and indexing), denying everyone access to all of the services, or allowing only the owner of the account to index documents, and everyone else to just query the domain. You can also allow or deny access to services based on a single address or a block of IP addresses.

1. From the left navigation pane, choose **Availability Options**.

Notice there is only an option for “Multi-AZ”—using multiple Availability Zones—which is, by default, turned off. Leave the default option as-is. If you turn this option on, CloudSearch will duplicate your search domain and place each copy in separate AZs (Availability Zones) within your region. AZs are independent from each other, so if one AZ experiences an outage, the other won’t be affected. CloudSearch will balance all your queries and updates between the two copies of your domain in the different AZs, and will keep them in sync automatically. This is completely transparent for your application, and it allows you to easily achieve a high level of availability for your search application.

1. From the left navigation pane, choose **Indexing Options**.

On this page, you can configure your search domain’s indexing options, including the fields that form the documents you are indexing, the types of each field, and other options such as the analysis scheme for each field (this allows you to index content in languages other than English).

1. From the left navigation pane, choose **Scaling Options**.

Here, you can define your minimum desired instance type. Instance types can grow larger automatically according to traffic, but won’t shrink to a smaller instance type than the one you defined as the minimum desired. You can define your desired replication count, and for the largest instance type (search.m2.2xlarge), you can also define your desired partition count. For this lab, do not change the default options.

1. From the left navigation pane, choose **Suggesters**.

On this page, you can create suggesters to be used for automatic completion of user search terms.

1. From the left navigation pane, choose **Expressions**.

Expressions are algebraic equations that allow you to customize the ranking of your CloudSearch results. You can create expressions statically for the domain or pass expressions with the query to provide per-query ranking.

1. From the left navigation pane, choose **Analysis Schemes**.

You define analysis schemes to customize the handling of your text fields. You create a scheme with a source language, stemming, stop words, and synonym options then set the analysis scheme for your text fields on the Indexing Options tab on the AWS Console.

SEND DATA TO YOUR SEARCH DOMAIN

When your search domain status is *ACTIVE*, you are ready to start using it. But before you can search data from the domain, you first need to upload data into it. To send data to CloudSearch, you create a batch of documents in either JSON or XML and then post that batch to your domain’s document endpoint. You can also use the AWS Console, the AWS CLI (command line interface), AWS SDK or an http library for the language that you prefer.

**Indexing documents via the AWS Console**

Now you’ll upload the sample dataset contained in the file you downloaded using the AWS Console for CloudSearch.

1. From the left navigation pane for the **handsonlab** search domain, choose **Dashboard**.

 Make sure your search domain status is **ACTIVE**. If it’s still loading or processing, wait a few minutes and refresh the page until the search domain becomes active.

1. Choose **Upload Documents** and configure the following:

* For **What do you want to upload?**, select  **File(s) on my local disk**.
* Choose **Browse…** or **Choose File**.
* Navigate to the **songs-lab.json** file you downloaded earlier.
* Choose **Open** to select the file.
* Choose **Continue**.

1. Choose **Upload Documents**.

You will receive a confirmation screen that the file was sent to the search domain for processing.

1. Choose **Finish**.

On the search domain Dashboard, you can check the number of searchable documents.

1. Choose **Refresh** to update the number.

It should take a few seconds to reach the final number (the file you downloaded contains 2296 songs).

UNDERSTANDING THE INPUT FILE

**JSON format**

The file you uploaded is a batch JSON file containing a list of documents, one for each song to be indexed on your search domain. It follows the same index structure as the one declared when you created the domain. Each field is declared with the same name as the one used for the field in the search domain index configuration and is followed by the value that will be stored and indexed in the search domain for that field.

The following code is sample batch extracted from that file, showing two songs.

First, each document contains an “id” field, which is mandatory and which contains a unique identifier for that document. Each document also has a “type” field, with a value of *add*. This is the operation that should be performed for this document and in this case, the document should be added to the search domain. If you send a document to be indexed containing an ID of a document that is already indexed, the *add* operation behaves like an update operation. Another possible value for the “type” field is *delete*, meaning you want to remove that document from the search domain.

Following the ID and type, you have an array of “fields.” These are the fields that make up your document. In this document, a song is composed of fields such as “artist\_name” and “place\_name.” Most of the fields are self-explanatory. The field “artist\_hotness” is a number, ranging from 0 to 1, used as an index to measure how important or relevant that artist is compared to other artists in the search domain. This is a pre-calculated value for the search domain and can be used to influence the results of CloudSearch queries via custom expressions and ranking (these concepts will be explored later in this lab).

The field “terms” contains an array of terms used to classify the song according to its genre. This field will be used for faceted navigation, and this concept will also be explored later in this lab.

[

{

"id": "SOESRBI12A8C1317FB",

"type": "add",

"fields": {

"artist\_hotness": "0.6533702",

"artist\_name": "Journey",

"location": "40.1142,-88.2435",

"place\_name": "Champaign Illinois USA",

"song\_title": "Open Arms",

"year": "1981",

"artist\_id": "ARPNILO1187B9B59BB",

"album\_title": "Greatest Hits Live",

"terms": [

"classic rock",

"rock",

"80s",

"hard rock",

"progressive rock"

]

}

},

{

"id": "SODYKRW12D021B3348",

"type": "add",

"fields": {

"artist\_hotness": "0.8075871",

"artist\_name": "\*Bruce\* \*Springsteen\*",

"location": "40.23447,-74.29504",

"place\_name": "\*Freehold\*, NJ",

"song\_title": "\*Downbound\* Train",

"year": "1981",

"artist\_id": "AR91C8S1187B990901",

"album\_title": "Born In The U.S.A.",

"terms": [

"classic rock",

"rock",

"singer-\*songwriter\*",

"80s",

"folk"

]

}

}

]

CloudSearch also supports the XML format for uploading documents to the search domain. Here’s an example of how the same two documents would be represented in XML:

<batch>

<add id="SOESRBI12A8C1317FB">

<field name="artist\_hotness">0.6533702</field>

<field name="artist\_name">Journey</field>

<field name="location">40.1142,-88.2435</field>

<field name="place\_name">Champaign Illinois USA</field>

<field name="song\_title">Open Arms</field>

<field name="year">1981</field>

<field name="artist\_id">ARPNILO1187B9B59BB</field>

<field name="album\_title">Greatest Hits Live</field>

<field name="terms">rock</field>

<field name="terms">classic rock</field>

<field name="terms">80s</field>

<field name="terms">hard rock</field>

<field name="terms">progressive rock</field>

</add>

<add id="SODYKRW12D021B3348">

<field name="artist\_hotness">0.8075871</field>

<field name="artist\_name">Bruce Springsteen</field>

<field name="location">40.23447, -74.29504</field>

<field name="place\_name">Freehold, NJ</field>

<field name="song\_title">Downbound Train</field>

<field name="year">1981</field>

<field name="artist\_id">AR91C8S1187B990901</field>

<field name="album\_title">Born In The U.S.A.</field>

<field name="terms">rock</field>

<field name="terms">classic rock</field>

<field name="terms">singer-songwriter</field>

<field name="terms">80s</field>

<field name="terms">fold</field>

</add>

</batch>

**Indexing documents via command line**

This lab doesn’t cover importing documents via command line, but it’s still an option. In order to use the CloudSearch command line interface, you must first install and configure the CloudSearch command-line tools. Here is an example:

cs-import-documents -d <mydomain> --source <c:\myAmazingDataSet\\*> <COMMON\_OPTIONS>

**Indexing documents via HTTP/REST**

Another way to index documents in CloudSearch is to submit them directly via HTTP/REST. Here is an example of how to upload a JSON file using the curl tool in a unix-like operating system:

curl -X POST --upload-file <filename>.json doc-<cloudsearch\_domain>.cloudsearch.amazonaws.com/2013-01-01/documents/batch --header "Content-type:application/json

**Indexing documents via API**

Here’s an example of how to use the AWS SDK for Java to add a document on CloudSearch:

import java.io.ByteArrayInputStream;

import java.io.ByteArrayInputStream;

import java.io.InputStream;

import java.nio.charset.StandardCharsets;

import com.amazonaws.auth.AWSCredentialsProvider;

import com.amazonaws.auth.ClasspathPropertiesFileCredentialsProvider;

import com.amazonaws.services.cloudsearchdomain.AmazonCloudSearchDomainClient;

import com.amazonaws.services.cloudsearchdomain.model.UploadDocumentsRequest;

public class CloudSearchUploadDocument {

public static void main(String[] args) {

AWSCredentialsProvider credentialsProvider = new ClasspathPropertiesFileCredentialsProvider();

AmazonCloudSearchDomainClient csClient = new AmazonCloudSearchDomainClient(credentialsProvider);

csClient.setEndpoint("doc-<your\_searchdomain\_endpoint>.us-west-2.cloudsearch.amazonaws.com");

UploadDocumentsRequest uploadDocumentsRequest = new UploadDocumentsRequest();

String document = "[{\"id\": \"SOESRBI12A8C1317FB\",\"type\": \"add\",\"fields\": {\"artist\_hotness\": \"0.6533702\",\"artist\_name\": \"Journey\",\"location\": \"40.1142,-88.2435\",\"place\_name\": \"Champaign Illinois USA\",\"song\_title\": \"Open Arms\",\"year\": \"1981\",\"artist\_id\": \"ARPNILO1187B9B59BB\",\"album\_title\": \"Greatest Hits Live\",\"terms\": [\"classic rock\",\"rock\",\"80s\",\"hard rock\",\"progressive rock\"]}}]"

InputStream stream = new ByteArrayInputStream(document.getBytes(StandardCharsets.UTF\_8));

uploadDocumentsRequest.setDocuments(stream);

csClient.uploadDocuments(uploadDocumentsRequest);

}

}

**Congratulations!** You successfully explored Amazon CloudSearch and walked through the AWS Console for CloudSearch.

**Task 4: Interactive Exercises**

Now that you’ve created your search domain and uploaded data into it, you’ll explore how to build search in your applications, backed by CloudSearch.

In this task, you will perform a series of exercises in an interactive simulator, covering some of the most important CloudSearch features.

The simulator walks you through the construction of URLs to perform the various query functions. You can copy those URLs and paste them in the address bar of your browser or use curl or any other tool for HTTP transport to experiment with the URLs and search API. There is also an option for each of the exercises for you to open the query in a new browser, so you don’t have to copy and paste it.

Though all of the exercises are based on pure HTTP/REST requests via javascript/AJAX, you can also build your application with CloudSearch by using the AWS SDK or an http transport library to build and send requests.

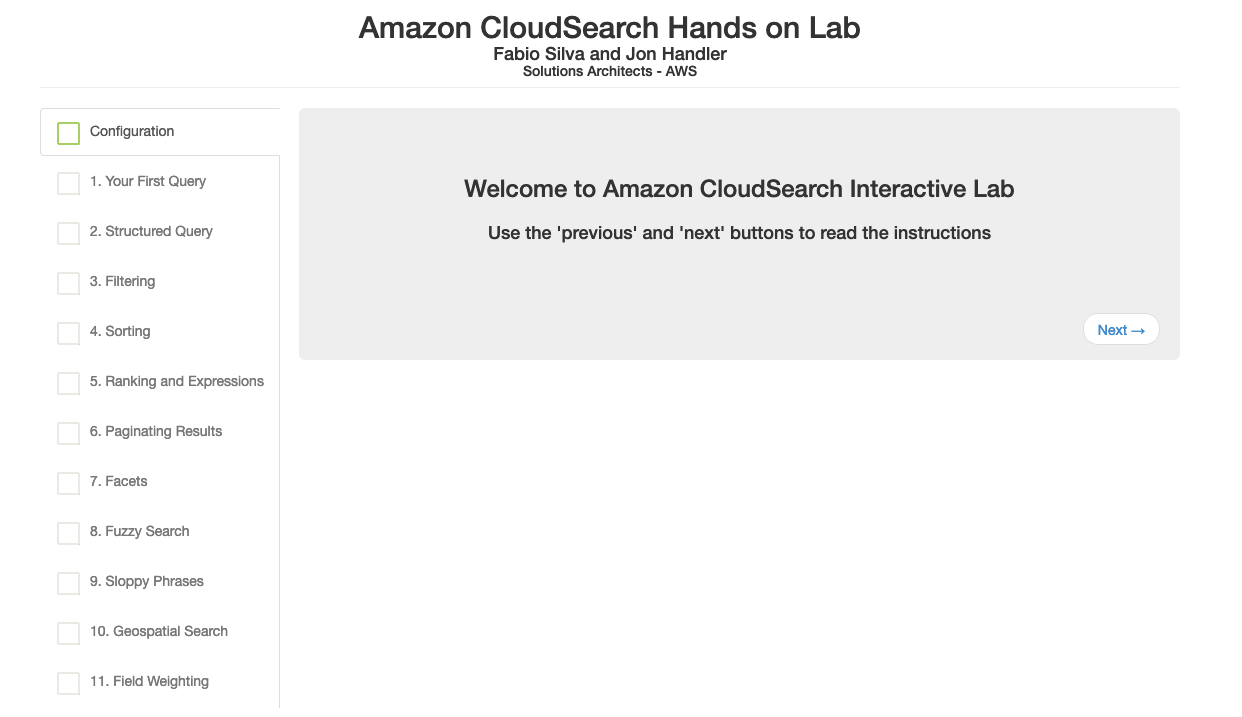
Follow the instructions found at the URL below to complete the exercises. After completing all exercises, please return to this document to continue the lab.

1. In a separate browser tab, paste:

InstanceIPAddress/cloudsearch

1. Replace **InstanceIPAddress** with the value that you copied to your text editor.
2. Press **Enter**.

 At this point, you see the following page:



1. Follow the instructions provided on this page to perform remaining steps for this lab.

**Congratulations!** You successfully completed a series of exercises in an interactive simulator, covering some of the most important CloudSearch features.

**Conclusion**

Congratulations! You now have successfully:

* Created a CloudSearch domain.
* Configured your domain to index the contents of a sample data set containing songs.
* Uploaded the sample data set to your domain.
* Experimented with different queries to perform simple matches, structured queries, geographic queries, fuzzy queries, sloppy phrase search and faceted search.
* Changed ranking, sorting and pagination.

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

**Additional Resources**

* [Amazon CloudSearch](http://aws.amazon.com/cloudsearch/)

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).