elastic

Building a great search experience workshop

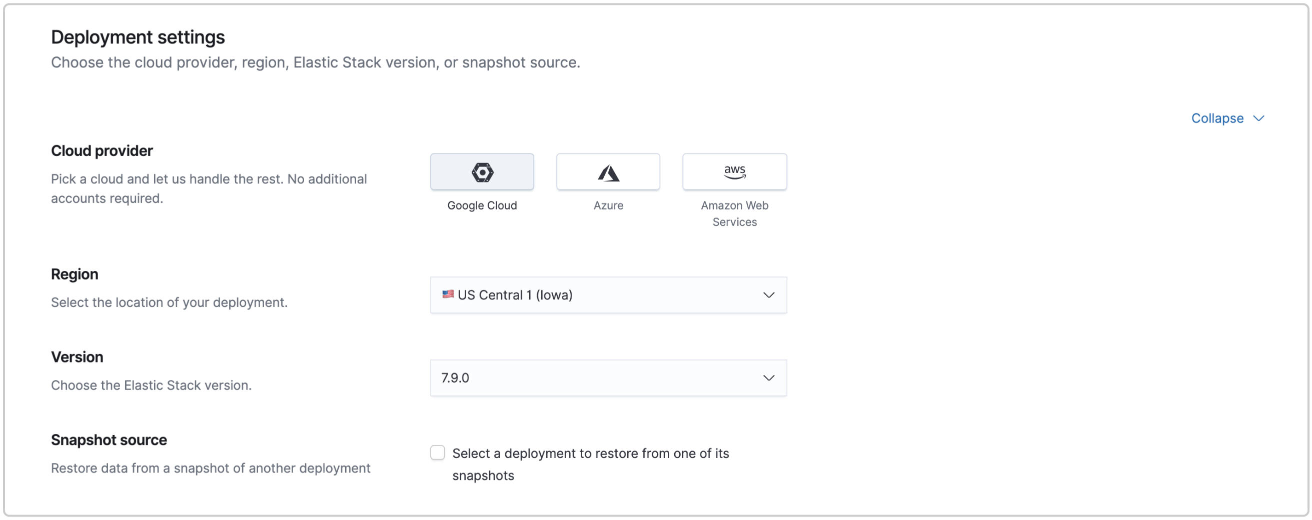
Table of Contents

Building a great search experience workshop

Create an Enterprise Search deployment

In this section, you are going to learn how to create an Enterprise Search deployment using the Elastic Cloud.

1. Go to https://cloud.elastic.co.
2. Click on the "Sign up" link.
3. Enter your email and choose a password.
4. Click on "Start your free trial". If you already started your free trial, click on "Create deployment".
5. Select the type of deployment you want to use. For this workshop, let’s select the `Elastic Enterprise Search` deployment.
6. Expand the deployment settings section, select the cloud provider of your choice along with the closest location, and keep the latest version of the Elastic Stack.



Deployment settings
Choose the cloud provider, region, Elastic Stack version, or snapshot source.

Collapse

Cloud provider
Pick a cloud and let us handle the rest. No additional accounts required.

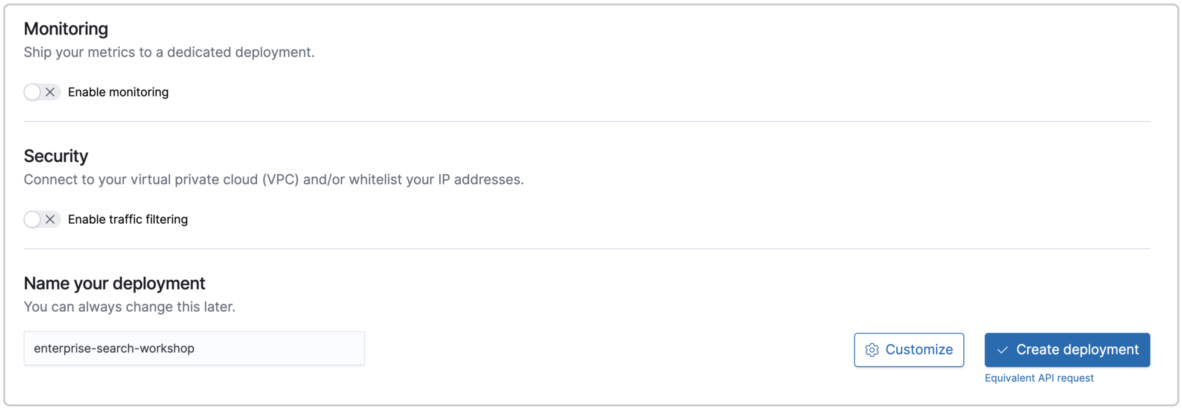
Google Cloud | Azure | Amazon Web Services

Region
Select the location of your deployment.
US Central 1 (Iowa)

Version
Choose the Elastic Stack version.
7.9.0

Snapshot source
Restore data from a snapshot of another deployment
Select a deployment to restore from one of its snapshots

7. Name your deployment, for example, "enterprise-search-workshop" and click on "Create deployment".



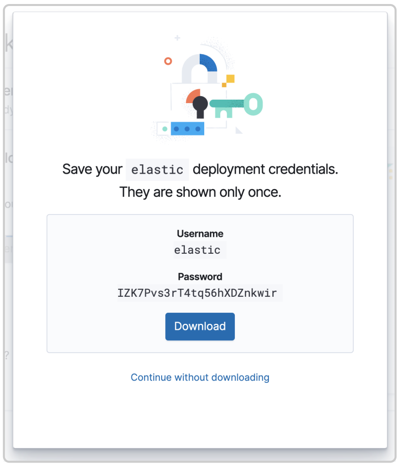
Monitoring
Ship your metrics to a dedicated deployment.
Enable monitoring

Security
Connect to your virtual private cloud (VPC) and/or whitelist your IP addresses.
Enable traffic filtering

Name your deployment
You can always change this later.
enterprise-search-workshop

Customize
Create deployment
Equivalent API request

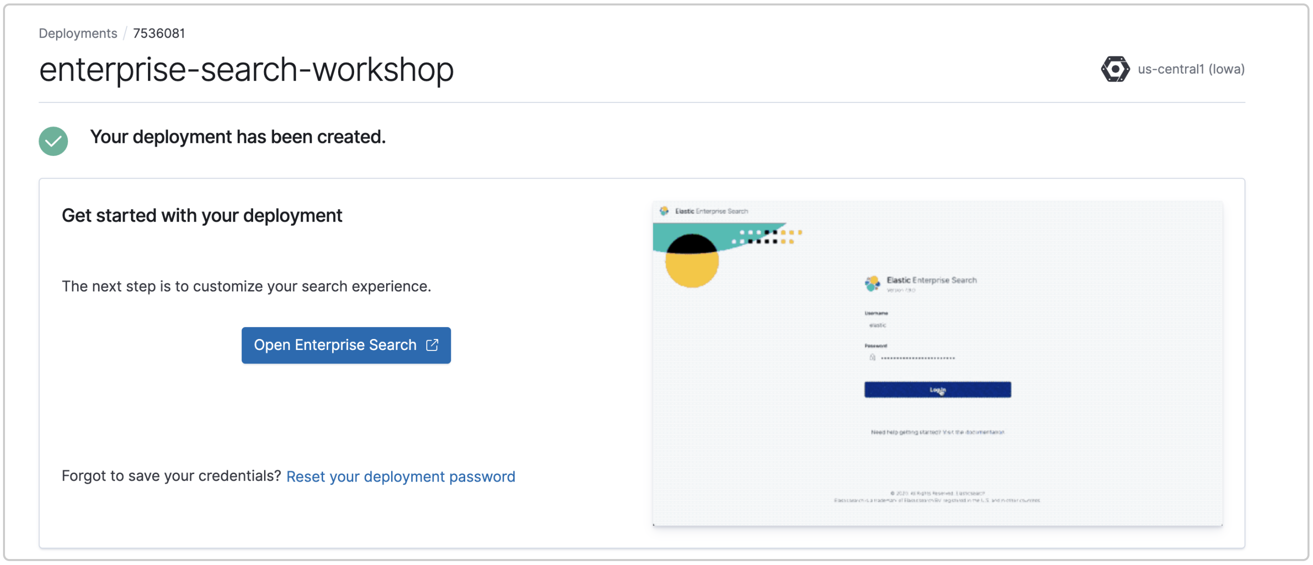
8. Please copy the `elastic` password and download the CSV file. You will be using it later.


Save your `elastic` deployment credentials. They are shown only once.

Username
elastic

Password
IZK7Pvs3rT4tq56hXDZnkwir

9. Wait a couple of minutes for your deployment to be ready then click on "Open Enterprise Search".


Deployments / 7536081

enterprise-search-workshop

us-central1 (Iowa)

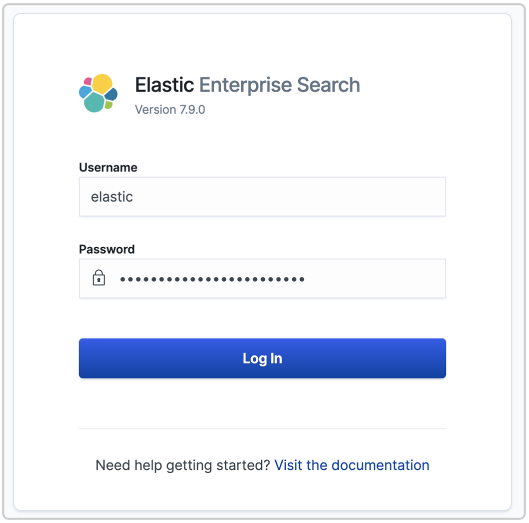
Your deployment has been created.

Get started with your deployment

The next step is to customize your search experience.

Forgot to save your credentials? Reset your deployment password

10. Log in as the `elastic` user and use the password you copied from the previous step.


Elastic Enterprise Search
Version 7.9.0

Username
elastic

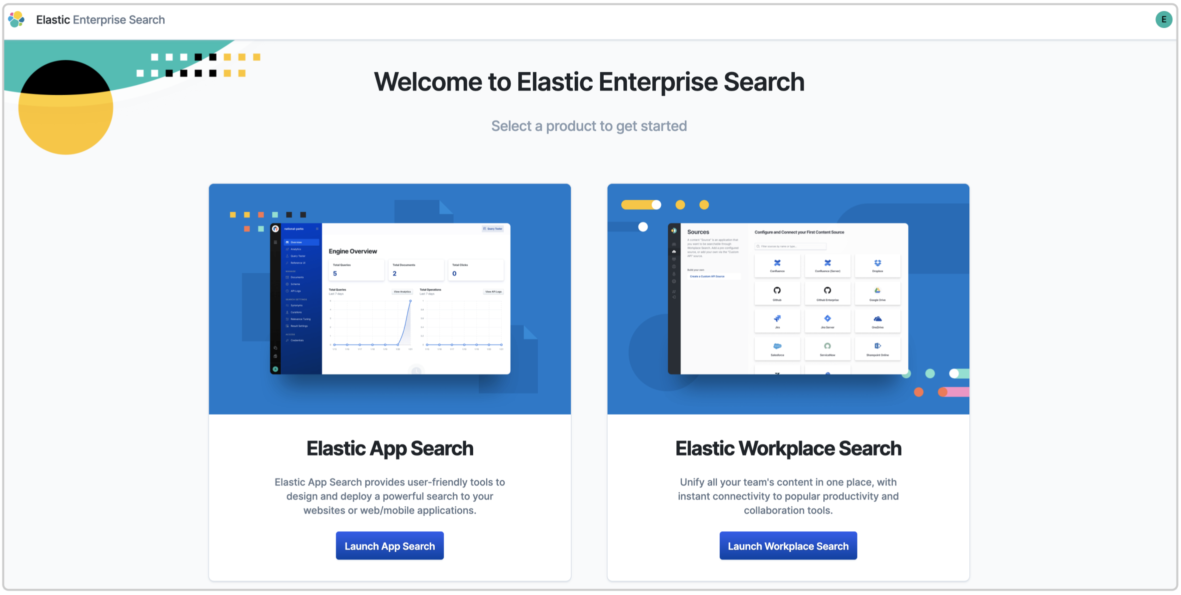
Password
••••••••••••••••••••••••

Need help getting started? Visit the documentation

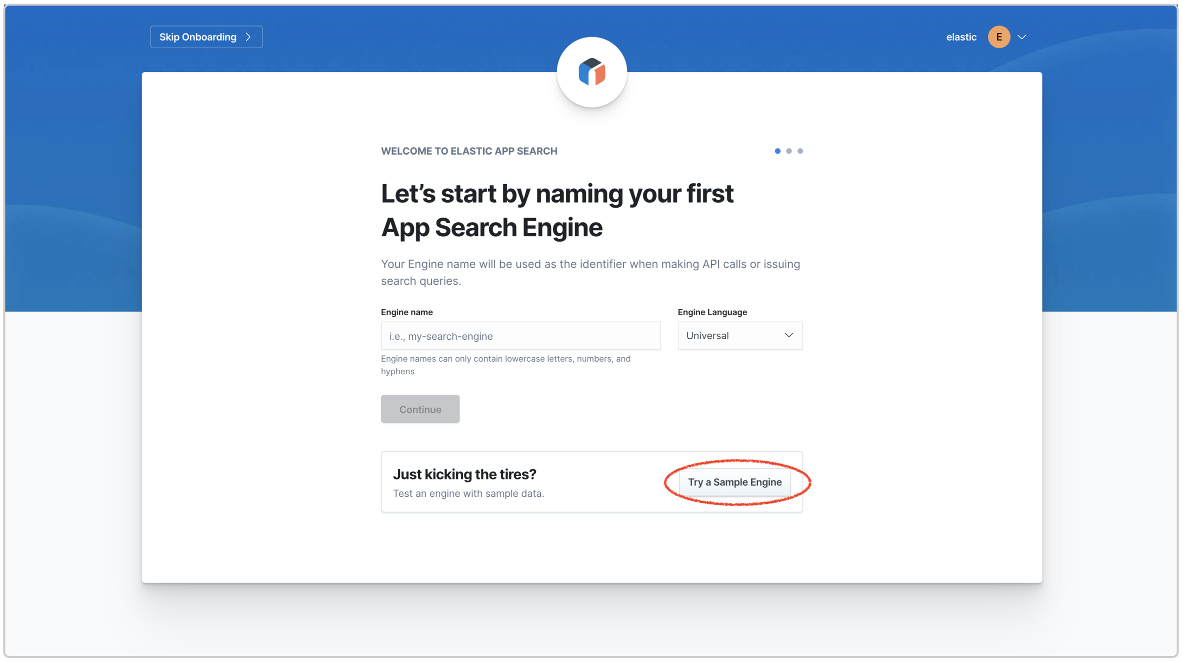
Add data to App Search

In this section, you are going to use the sample engine in order to get quickly started.

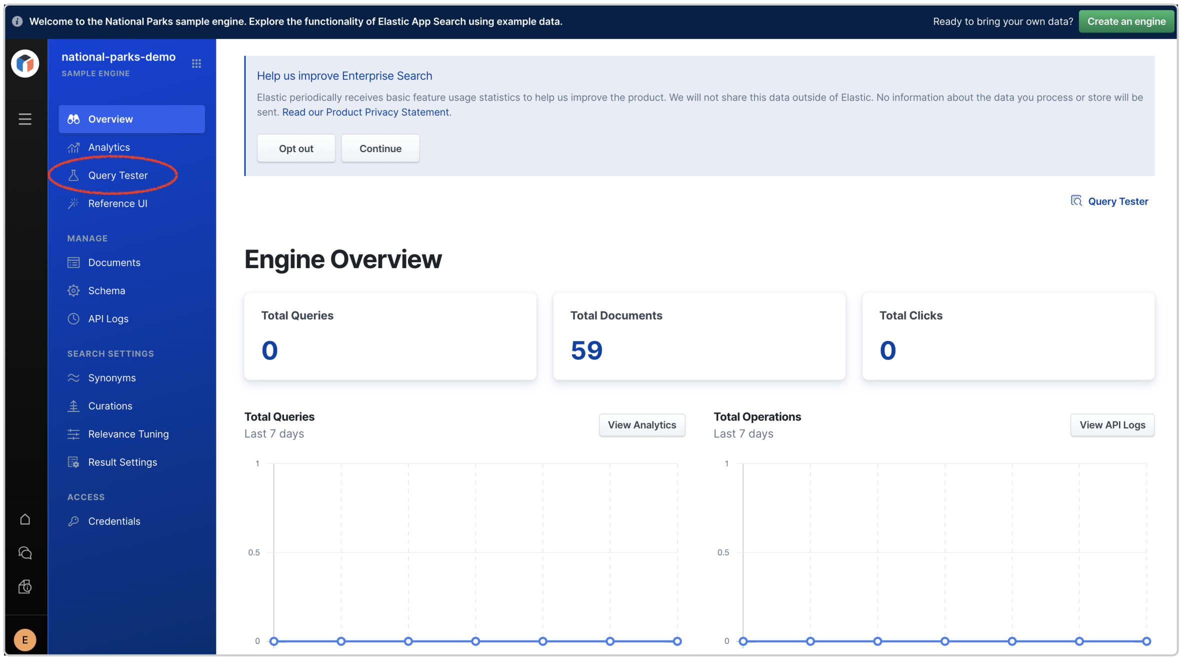
1. Once you are logged in to Enterprise Search, click on "Launch App Search".


Elastic Enterprise Search
Welcome to Elastic Enterprise Search
Select a product to get started
Elastic App Search
Elastic App Search provides user-friendly tools to design and deploy a powerful search to your websites or web/mobile applications.
Launch App Search
Elastic Workplace Search
Unify all your team's content in one place, with instant connectivity to popular productivity and collaboration tools.
Launch Workplace Search

2. For this workshop, you will be using an engine that already contains data. Click on "Try a Sample Engine" to get started.


Skip Onboarding
elastic
WELCOME TO ELASTIC APP SEARCH
Let's start by naming your first App Search Engine
Your Engine name will be used as the identifier when making API calls or issuing search queries.
Engine name
i.e., my-search-engine
Engine names can only contain lowercase letters, numbers, and hyphens
Engine Language
Universal
Continue
Just kicking the tires?
Test an engine with sample data.
Try a Sample Engine

3. After few seconds, your engine will be ready. Notice that you have a total of 59 documents in this engine. This sample engine contains all the national park in the United States, each document is related to a particular park. Let's take some time to explore this new engine, click on "Query Tester".



Welcome to the National Parks sample engine. Explore the functionality of Elastic App Search using example data.

Help us improve Enterprise Search

Elastic periodically receives basic feature usage statistics to help us improve the product. We will not share this data outside of Elastic. No information about the data you process or store will be sent. Read our Product Privacy Statement.

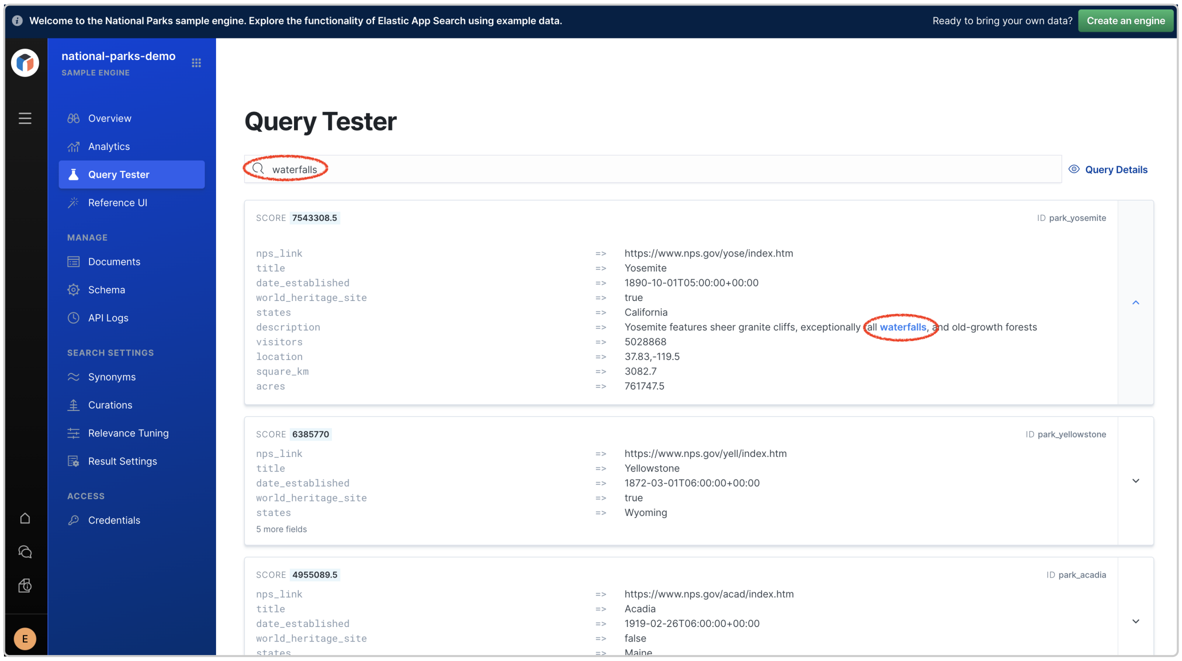
Engine Overview

Total Queries	Total Documents	Total Clicks
0	59	0

Total Queries
Last 7 days

Total Operations
Last 7 days

4. In this view, you can test your queries and see what documents are returned. Get familiar with this dataset by running some queries. For example, you could search for parks that contains "waterfalls".



Welcome to the National Parks sample engine. Explore the functionality of Elastic App Search using example data.

Query Tester

waterfalls

```
SCORE 7543308.5                                                        ID park_yosemite

nps_link             =>  https://www.nps.gov/yose/index.htm
title                =>  Yosemite
date_established     =>  1890-10-01T05:00:00+00:00
world_heritage_site  =>  true
states               =>  California
description          =>  Yosemite features sheer granite cliffs, exceptionally all waterfalls, and old-growth forests
visitors             =>  5028868
location             =>  37.83,-119.5
square_km            =>  3082.7
acres                =>  761747.5

SCORE 6385770                                                          ID park_yellowstone

nps_link             =>  https://www.nps.gov/yell/index.htm
title                =>  Yellowstone
date_established     =>  1872-03-01T06:00:00+00:00
world_heritage_site  =>  true
states               =>  Wyoming
5 more fields

SCORE 4955089.5                                                        ID park_acadia

nps_link             =>  https://www.nps.gov/acad/index.htm
title                =>  Acadia
date_established     =>  1919-02-26T06:00:00+00:00
world_heritage_site  =>  false
states               =>  Maine
```

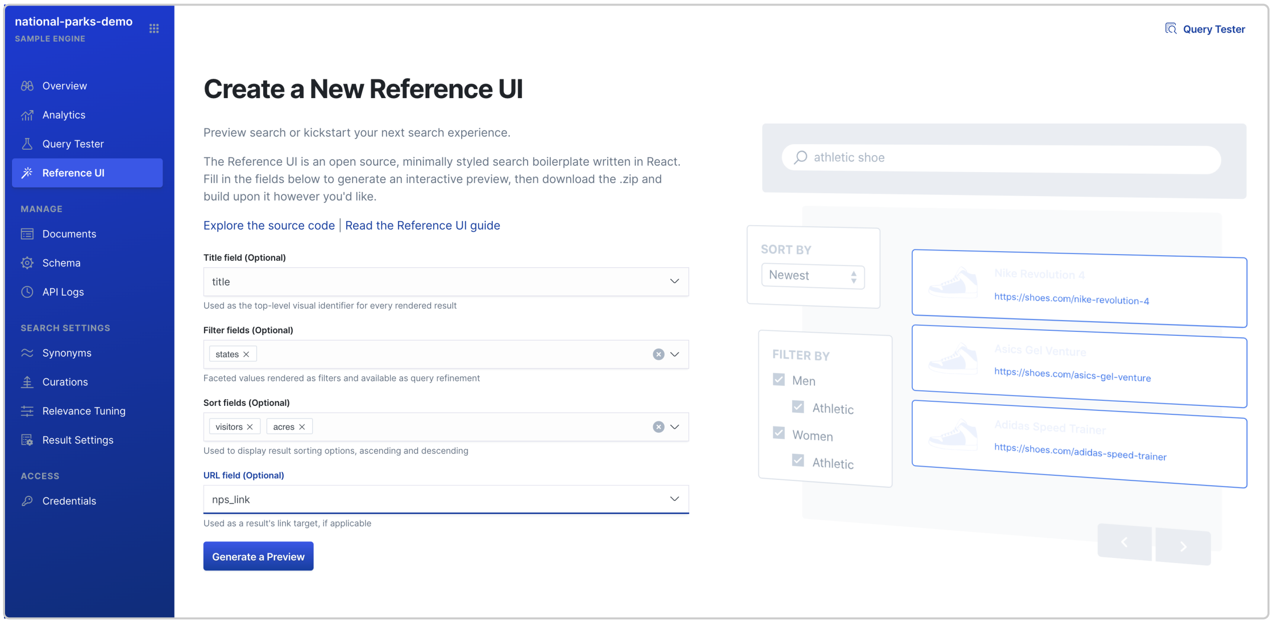
5. When using the sample engine, some tuning has been pre-defined for you. You can see how this engine has been setup by opening the following options on the left-hand side menu:
   a. Synonyms
   b. Curations
   c. Relevance Tuning

Customize your search interface

In this section, you will get familiar with the reference UI which allows you to customize the search experience for your users.

1. Click on "Reference UI" on the left-hand side.
2. On this view, you can define the fields that will be used in your search UI. Select the following fields:
   a. select `title` for the title field
   b. select `states` for the filter field
   c. for the sort fields, select `visitors`, and `acres`
   d. select `nps_link` for the URL field

   Then click on "Generate a Preview".



Create a New Reference UI

Preview search or kickstart your next search experience.

The Reference UI is an open source, minimally styled search boilerplate written in React. Fill in the fields below to generate an interactive preview, then download the .zip and build upon it however you'd like.

Explore the source code | Read the Reference UI guide

Title field (Optional): title
Used as the top-level visual identifier for every rendered result

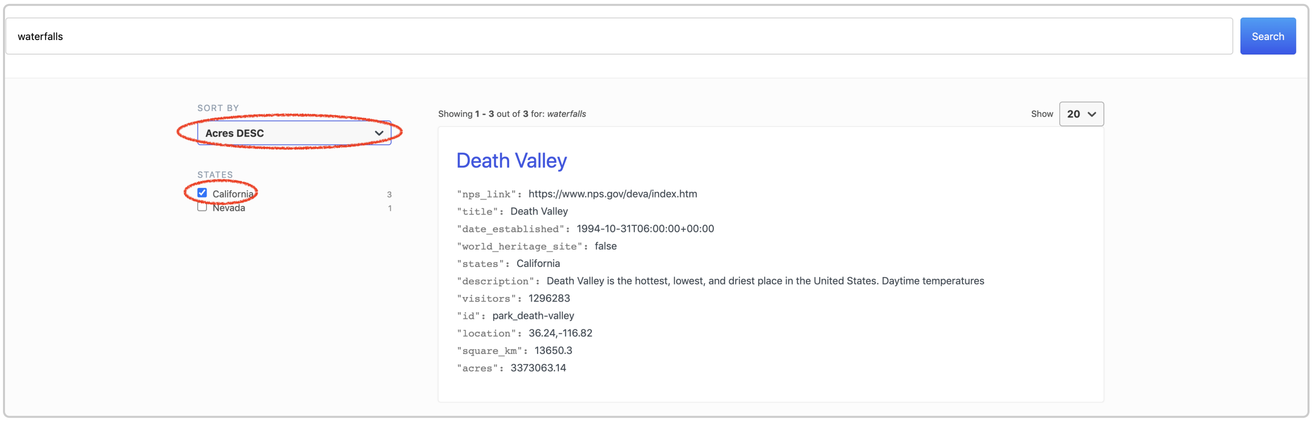
Filter fields (Optional): states
Faceted values rendered as filters and available as query refinement

Sort fields (Optional): visitors, acres
Used to display result sorting options, ascending and descending

URL field (Optional): nps_link
Used as a result's link target, if applicable

Generate a Preview

3. Start by searching for "waterfalls". Notice that as you are typing the first letter, App Search will automatically suggest queries for you based on the documents of your engine.
4. Filter the parks from "California" and display the biggest parks first.



Showing **1 - 3** out of **3** for: *waterfalls*

Death Valley

```
"nps_link": https://www.nps.gov/deva/index.htm
"title": Death Valley
"date_established": 1994-10-31T06:00:00+00:00
"world_heritage_site": false
"states": California
"description": Death Valley is the hottest, lowest, and driest place in the United States. Daytime temperatures
"visitors": 1296283
"id": park_death-valley
"location": 36.24,-116.82
"square_km": 13650.3
"acres": 3373063.14
```

5. Click on the title of the first result. You will be redirected to the page defined by the `nps_link` field of this document.
6. Click on "Download ZIP Package" at the top-right corner of the page. This will generate a ZIP file that contains the source code that you can use in your application.
7. Finally, click on "Back to configuration" to return to App Search.

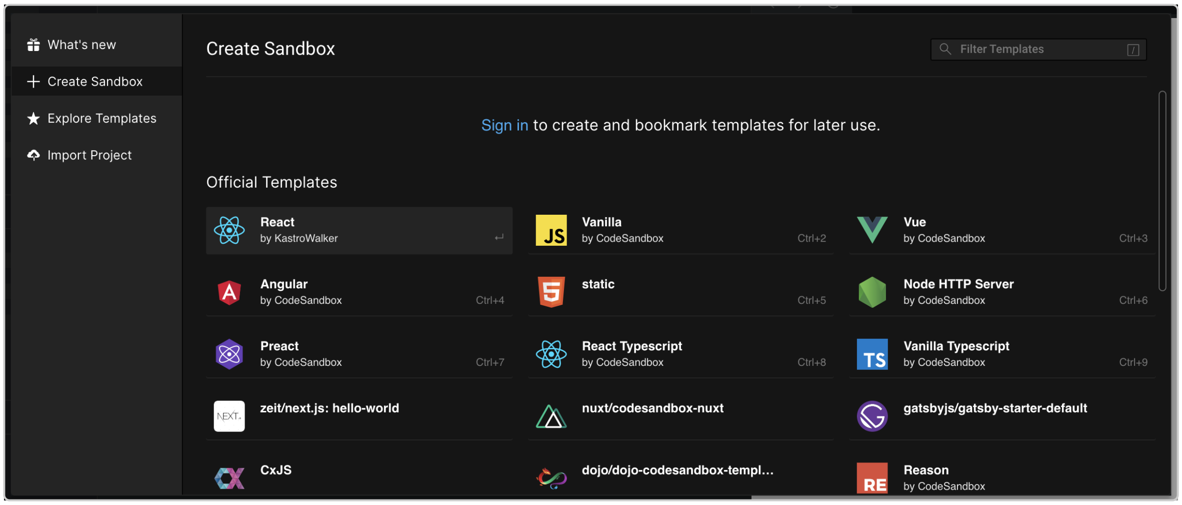
Using The Search UI

In this section, you will be creating a custom User Interface using the Search UI. You will start from a blank page and you will be adding, step by step, components that makes a great search interface for your users.

By the end of this workshop, you will have created a search interface similar to this demo.

Get Started

1. The search UI is a React library, let's start by creating a sample react project. You can create this project locally or use an online IDE such as https://codesandbox.io/ which will help you to quickly get started.
2. Open https://codesandbox.io/ and create a new sandbox. Select the React template.



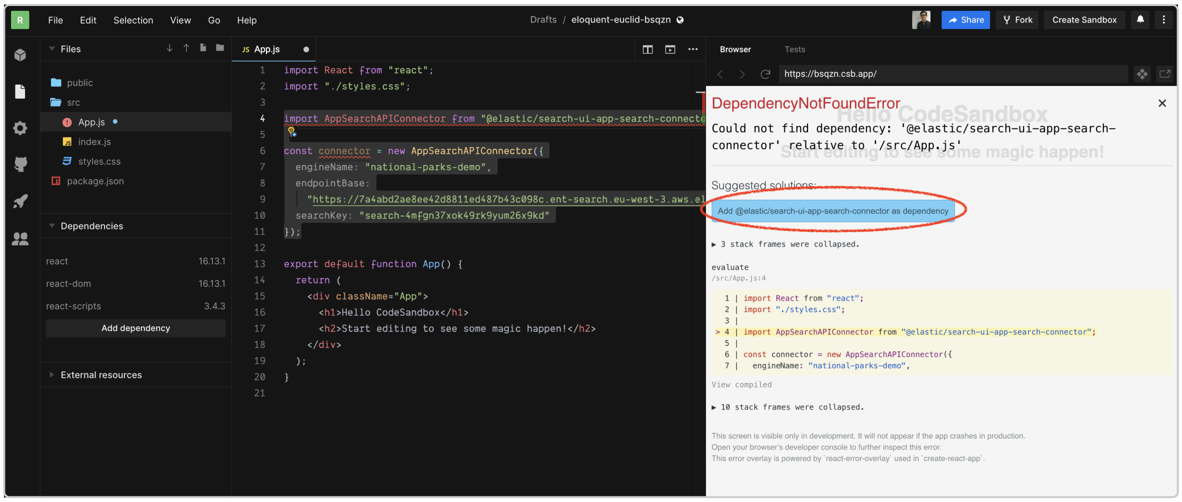
What's new
Create Sandbox
Explore Templates
Import Project
Create Sandbox
Filter Templates
Sign in to create and bookmark templates for later use.
Official Templates
React by KastroWalker
Vanilla by CodeSandbox Ctrl+2
Vue by CodeSandbox Ctrl+3
Angular by CodeSandbox Ctrl+4
static Ctrl+5
Node HTTP Server by CodeSandbox Ctrl+6
Preact by CodeSandbox Ctrl+7
React Typescript by CodeSandbox Ctrl+8
Vanilla Typescript by CodeSandbox Ctrl+9
zeit/next.js: hello-world
nuxt/codesandbox-nuxt
gatsbyjs/gatsby-starter-default
CxJS
dojo/dojo-codesandbox-templ...
Reason by CodeSandbox

3. Start by setting up the connector between your front-end application and App Search.

   First, import the connector.

```
import AppSearchAPIConnector
from "@elastic/search-ui-app-search-connector";
```

   You will need to add this new dependency to your project, you can easily do that by clicking on the blue button.



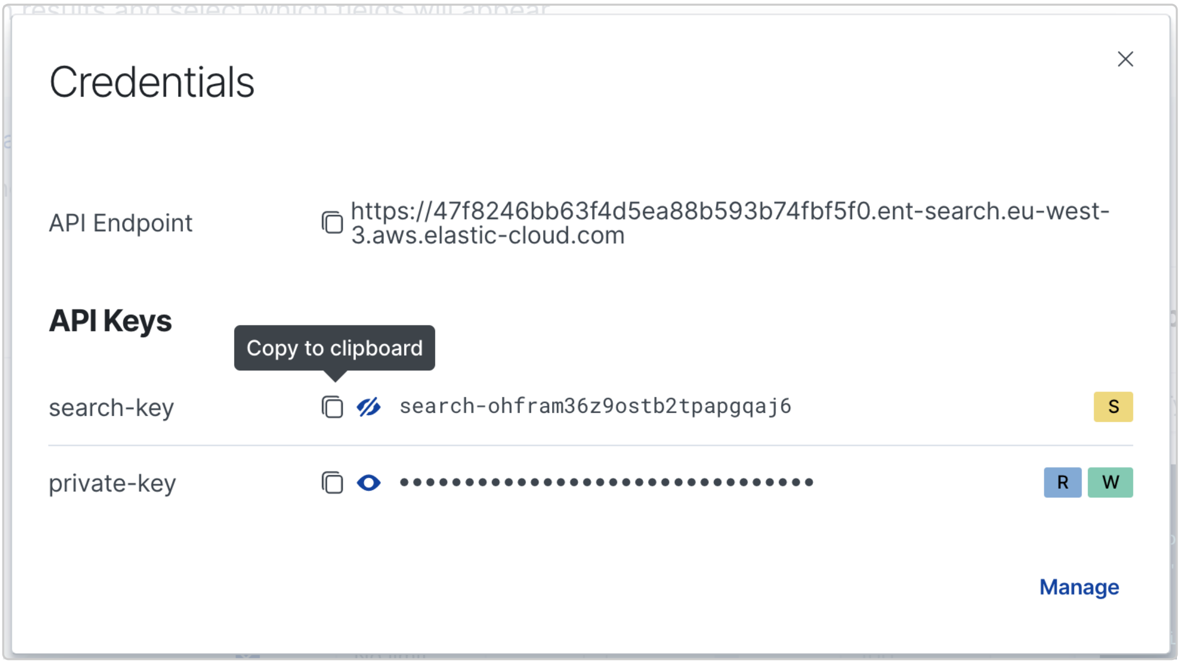
```
import React from "react";
import "./styles.css";

import AppSearchAPIConnector from "@elastic/search-ui-app-search-connecto

const connector = new AppSearchAPIConnector({
  engineName: "national-parks-demo",
  endpointBase:
    "https://7a4abd2ae8ee42d8811ed487b43c098c.ent-search.eu-west-3.aws.el
  searchKey: "search-4mfgn37xok49rk9yum26x9kd"
});

export default function App() {
  return (
    <div className="App">
      <h1>Hello CodeSandbox</h1>
      <h2>Start editing to see some magic happen!</h2>
    </div>
  );
}
```

```
DependencyNotFoundError
Could not find dependency: '@elastic/search-ui-app-search-
connector' relative to '/src/App.js'

Suggested solutions:

▶ 3 stack frames were collapsed.

evaluate
/src/App.js:4
  1 | import React from "react";
  2 | import "./styles.css";
  3 |
> 4 | import AppSearchAPIConnector from "@elastic/search-ui-app-search-connector";
  5 |
  6 | const connector = new AppSearchAPIConnector({
  7 |   engineName: "national-parks-demo",
View compiled

▶ 10 stack frames were collapsed.

This screen is visible only in development. It will not appear if the app crashes in production.
Open your browser's developer console to further inspect this error.
This error overlay is powered by `react-error-overlay` used in `create-react-app`.
```

Then, set up the connector.

```
const connector = new AppSearchAPIConnector({
  engineName: "national-parks-demo",
  endpointBase: "XXX",
  searchKey: "XXX"
});
```

4. Get your credentials from the App Search interface. On the left panel, click on "Credentials" and copy the `API Endpoint` and the `search-key`. If you are using Codesandbox or any online IDE, the code can be publicly distributed, therefore make sure to use a disposable DEV/TEST deployment or search-key, so you can delete after you finish the exercises.



Credentials

API Endpoint	https://47f8246bb63f4d5ea88b593b74fbf5f0.ent-search.eu-west-3.aws.elastic-cloud.com

API Keys

search-key	search-ohfram36z9ostb2tpapgqaj6	S
private-key	••••••••••••••••••••••••••••••••	R W

5. Your front-end application is now connected to App Search. Next, you are going to add components to build your own search interface. First you need to add the `SearchProvider` component which will be the top level component of your Search UI implementation. Replace the current HTML content with the following lines:

```
export default function App() {
  return (
    <SearchProvider
        config={{
            apiConnector: connector
        }}>
    </SearchProvider>
  )
}
```

You also need to import this new component in your application, add the following line at the top of your file, and install this new dependency.

```
import {
  SearchProvider
} from "@elastic/react-search-ui";
```

Add Basic Components

1. You are now ready to add your components. Add a `SearchBox` component in your application.

   First, import this new component:

   ```
   import {
     SearchProvider,
     SearchBox
   } from "@elastic/react-search-ui";
   ```

   Then, add this new component inside the `SearchProvider` component:

   ```
   <SearchProvider
       config={{
           apiConnector: connector
       }}>
       <SearchBox />
   </SearchProvider>
   ```

2. You can see on the right-hand side that you now have a tiny search box. You can perform some queries, however, you are not going to see any results. This is because you need to add the `Results` component in order to display these results. Add this component under the search box.

```
<SearchProvider
    config={{
        apiConnector: connector
    }}>
    <SearchBox />
    <Results />
</SearchProvider>
```

Note: Make sure to add this component in the list of imports from `@elastic/react-search-ui`.

3. Try your search box on the right-hand side. You can see that you get results coming from the engine that you set up earlier.
4. This interface is not really friendly at this moment, let’s add some style to improve it. You can either define your own style or use the built-in styles that come with the Search UI.

   Add the predefined styles from the Search UI.

```
import "@elastic/react-search-ui-views/lib/styles/styles.css";
```

5. It already looks much better but let’s go one step further by customizing the `Results` component. There are few things that you can customize. For example, let’s define the `titleField` and the `urlField`.

```
<Results titleField="title" urlField="nps_link" />
```

   The `title` field of the document is displayed as a header and if you click on it a new tab will be opened based on the `nps_link` field.

Autocompletion

1. You can improve your search bar by adding autocompletion. With the search UI, there are two types of autocompletion that you can set up:
   a. `autocompleteResults` : this will suggest search **results** based on your users' input. The user will be directly directed to a link when clicking on the suggestion.
   b. `autocompleteSuggestions` : this will suggest search **queries** based on your users' input. If the user clicks on the suggestion, it will act as a regular search query.
2. Start by adding `autocompleteResults` to your search box.

```
<SearchBox
    autocompleteResults={{
        linkTarget: "_blank",
        titleField: "title",
        urlField: "nps_link"
    }}
/>
```

3. In your search box, type "mou". You will see some suggested results, click on the "Rocky Mountain" suggestion.
4. You can also add highlighting to your suggestions, let’s update the config to do so. Update the `config` part of your application as follow:

```
<SearchProvider
    config={{
        apiConnector: connector,
        autocompleteQuery: {
            results: {
                result_fields: {
                    title: { snippet: { size: 100, fallback: true } },
                    nps_link: {
                        raw: {}
                    }
                }
            }
        }
    }}
>
...
</SearchProvider>
```

   Now, when you are running a new query in your search box you can see highlighting in the suggestions.

5. Next, let’s add `autocompleteSuggestions` to the search box. In order not to confuse both suggestions, you can set up a `sectionTitle` for each part.

```
<SearchBox
    autocompleteResults={{
        linkTarget: "_blank",
        sectionTitle: "Suggested Results",
        titleField: "title",
        urlField: "nps_link"
    }}
    autocompleteSuggestions={{
        sectionTitle: "Suggested Queries"
    }}
/>
```

6. Search again for "mou" and see that you now have different suggestions.

Faceting

Facets are an import part of your search experience, they allow the user to quickly filter the results of their queries. With Search UI, you have full control of the filters you want to display.

1. Create a new facet on the field `states` and limit the number of results to **3**.

   First, add the new component under the `SearchBox`:

   ```
   <SearchBox
   ...
   />
   <Facet field="states" label="States" />
   <Results titleField="title" urlField="nps_link" />
   ```

   Note: You need to add `Facet` in the list of imports.

   You also need to update the configuration to include this field in the facets:

```
<SearchProvider
    config={{
        apiConnector: connector,
        autocompleteQuery: {
            ...
        },
        searchQuery: {
            facets: {
                states: { type: "value", size: 3 }
            }
        }
    }}
>
...
</SearchProvider>
```

2. Let’s define a second facet to create a filter based on the number of users. You will define 3 ranges:
    a. 0-100000 = "Quiet"
    b. 100000-500000 = "Moderate"
    c. 500000 and more = "Busy"

    Add this new facet under the previous one.

    ```
    <SearchBox
    ...
    />
    <Facet field="states" label="States" />
    <Facet field="visitors" label="Number of visitors" />
    <Results titleField="title" urlField="nps_link" />
    ```

    Update the configuration to define the custom ranges.

```
<SearchProvider
    config={{
        apiConnector: connector,
        autocompleteQuery: {
            ...
        },
        searchQuery: {
            facets: {
                states: { type: "value", size: 3 },
                visitors: {
                    type: "range",
                    ranges: [
                    { from: 0, to: 100000, name: "Quiet" },
                    { from: 100000, to: 500000, name: "Moderate" },
                    { from: 500000, name: "Busy" }
                    ]
                }
            }
        }
    }}
>
...
</SearchProvider>
```

Layout

Your search experience is now functional, however, you can improve how your components are displayed. Using the `Layout` helps you to easily assemble your components on the user interface.

1. Start by importing the `Layout` component.

```
import {Layout} from "@elastic/react-search-ui-views";
```

2. You currently have all of your components at the same level. Create a new `Layout` component inside the `SearchProvider` component and rearrange the existing component as follow:
   a. the `SearchBox` should be in the **header**
   b. the `Results` should be in the **bodyContent**
   c. the `Facet` should be on the **sideContent** (note: because you have two facets, you need to wrap these two components inside a `div`)

```
<SearchProvider
    config={{
        ...
    }}
>
    <Layout
        header = {
            <SearchBox
                autocompleteResults={{
                    linkTarget: "_blank",
                    sectionTitle: "Suggested Results",
                    titleField: "title",
                    urlField: "nps_link"
                }}
                autocompleteSuggestions={{
                    sectionTitle: "Suggested Queries"
                }}
            />
        }
        bodyContent = {
            <Results titleField="title" urlField="nps_link" />
        }
        sideContent = {
            <div>
                <Facet field="states" label="States" />
                <Facet field="visitors" label="Number of visitors" />
            </div>
        }
    />

</SearchProvider>
```

3. Your components have been moved around. Your facets should be on the left-hand side but notice that if you change the size of the browser, the layout is automatically updated. You have a responsive design that can be used on both small and wide screens.

Sorting and Pagination

By default, your users get the most relevant results first and they can see the first 20 results. You can change this behavior by adding custom sorting option and add the possibility for the users to navigate through several pages of results.

1. Let's start by adding new sort options. On the `sideContent` add a `Sorting` component that have three options:
    a. by relevance
    b. by title (asc)
    c. by title (desc)

```
<div>
    <Facet field="states" label="States" />
    <Facet field="visitors" label="Number of visitors" />
    <Sorting
            sortOptions={[
            {
                name: "Relevance",
                value: "",
                direction: ""
            },
            {
                name: "Title (asc)",
                value: "title",
                direction: "asc"
            },
            {
                name: "Title (desc)",
                value: "title",
                direction: "desc"
            }
            ]}
    />
</div>
```

2. You can also combine the component from the Search UI with other component. Let's use components from the famous Material UI framework. Use the `Divider` component to separate the sort options from the facets.

   Import the new component.

   ```
   import Divider from "@material-ui/core/Divider";
   ```

   Add the `Divider` between the two `Facet` and `Sorting` components.

   ```
   <div>
       <Facet field="states" label="States" />
       <Facet field="visitors" label="Number of visitors" />
       <Sorting
           ...
       />
   </div>
   ```

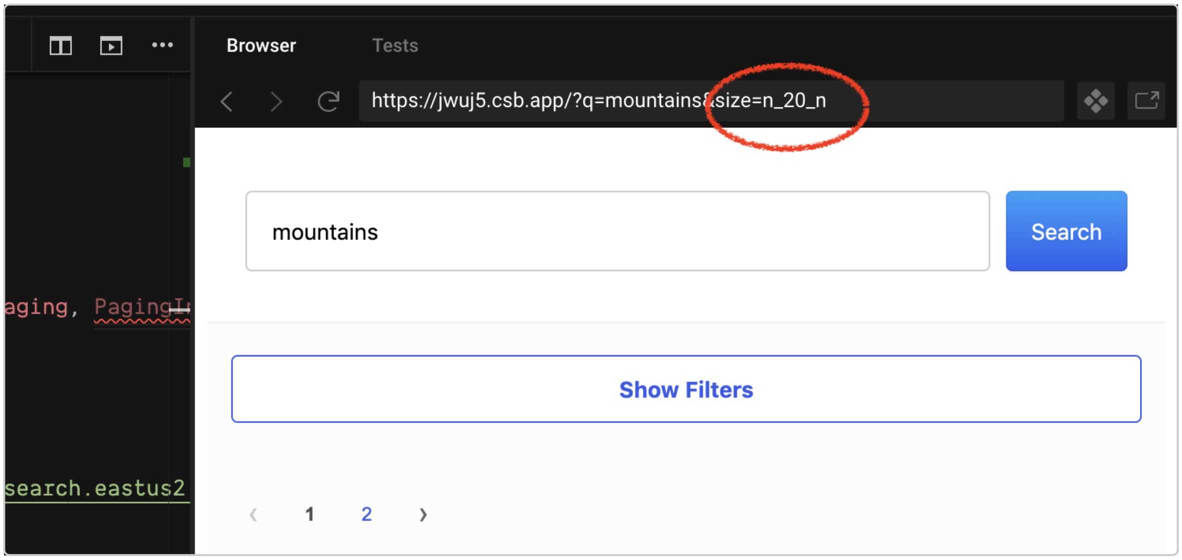
3. Finally, let's add some pagination. Add a `Paging` component in the `bodyHeader` section.

```
bodyHeader={
    <Paging />
}
```

4. When searching for "mountains", you now have two pages of results. By default, you have 20 results per page. You can change this default behavior by defining the `initialState` in the config.

```
config={{
    apiConnector: connector,
    initialState: {
        resultsPerPage: 5
    },
    ...
}}
```

5. Note that this update will not be directly reflected on the right-hand side preview. You need to clean up the URL to remove the parameter `size=n_20_n`.



6. You can also leave the possibility for the user to choose their own number of results per page by using the `ResultsPerPage` component.

```
bodyHeader={
    <React.Fragment>
        <Paging />
        <ResultsPerPage />
    </React.Fragment>
}
```

7. If you are using Codesandbox or any online IDE, make sure to delete the deployment or the search-key.

Building a great search experience workshop

Version: 7.9-0