

DTI C3 AI Suite Quickstart Guide

- [Terminology](#)
- [C3 Cluster Overview](#)
- [Provision a C3 AI Package](#)
- [Connecting to the C3 AI Suite](#)
 - [Accessing the Static Console](#)
 - [Using the Static Console](#)
 - [Tutorial Video](#)
 - [Console Commands](#)
 - [Official C3 AI Training Materials on Static Console](#)
 - [Using Python with the C3 AI Suite](#)
- [Fetching Instances of C3 Types](#)
 - [Examples of Fetch Calls](#)
 - [Tutorial Video](#)
 - [The fetchCount Method](#)
- [Converting Fetch results to usable forms in Jupyter Notebook](#)
 - [Python](#)
- [ExpressionEngineFunctions](#)
- [Computations on C3 Types using Evaluate](#)
- [Developing Metrics on Timeseries data](#)
 - [Timeseries Video Tutorial](#)
 - [Simple Metrics](#)
 - [Compound Metrics](#)
 - [Finding, Evaluating, and Visualizing Metrics](#)
 - [Finding Metrics](#)
 - [Evaluating Metrics](#)
 - [Additional Resources](#)
- [Review and Next Steps](#)

The C3 AI Suite provides researchers many tools to analyze data and build and deploy machine learning models. This guide explains how to connect to the C3 AI Suite, access data using C3 AI methods, and convert C3 AI method outputs to an easy-to-analyze form. Additionally, the guide also provides more detailed instructions to DTI members using the Covid-19 Data Lake. Examples in this guide rely on the 'baseCovidDataLake' package available in this [git repository](#).

Please note, this guide covers how to query data from the C3 AI Suite. For more advanced topics such as loading data, building metrics, or configuring and training machine learning models, please refer to the following wikis:

- [Data Integration](#) (Not yet available)
- [Metrics](#) (Not yet available)
- [Machine Learning](#) (Not yet available)

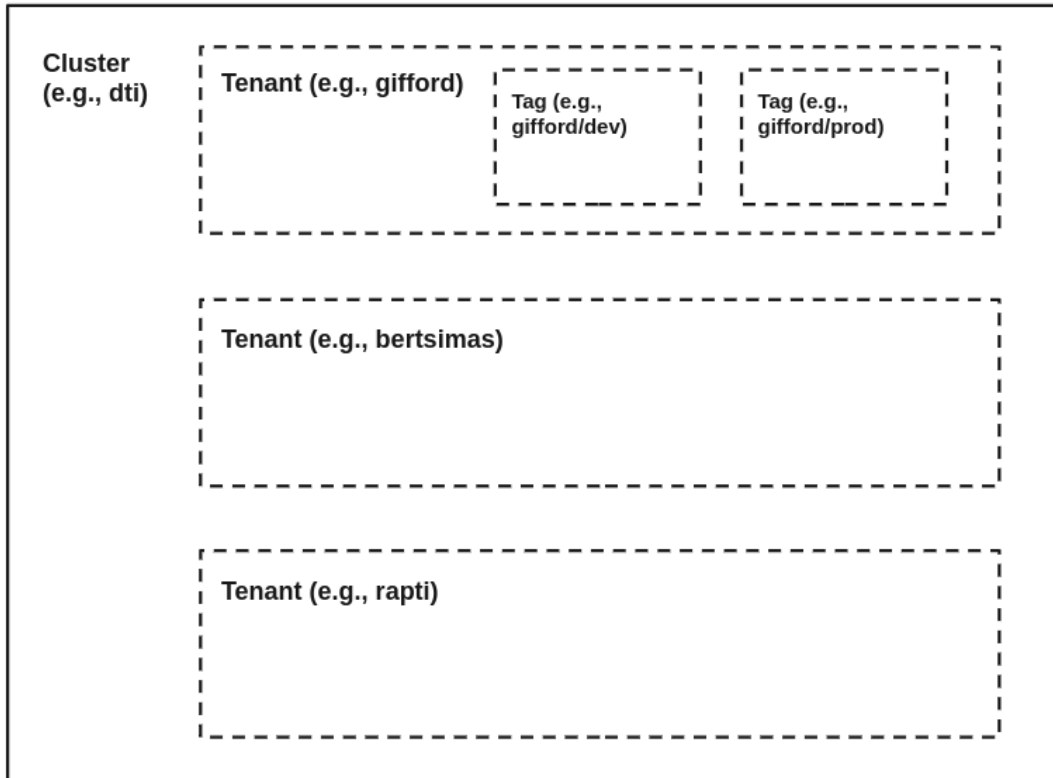
Terminology

To best understand the C3 AI Suite and this guide, let's introduce key terminology used by C3 AI Suite developers:

- **Type:** Nearly all aspects of the C3 AI Suite (e.g., data, machine learning models, cloud-provider microservices) are stored and accessed through Types. C3 AI Types are logical objects akin to a Java class, and contain 'fields' and 'methods'. Some Types are persisted to internal databases (like Postgres or Cassandra), while others are not.
- **Field:** A field of a Type. Fields contain attributes or data associated with the Type.
- **Method:** A method declared on a Type. Methods define business logic associated with the Type.
- **Vanity Url:** The URL at which a specific tenant/tag of a C3 Cluster can be accessed.
- **Cluster:** A deployment of the C3 AI Suite. A C3 Cluster is a collection of hardware or virtualized cloud instances (e.g., servers, databases, load balancers) used to run the C3 AI Suite and C3 AI Applications. The C3 AI Suite can run on any public or private cloud infrastructure or on a local machine (in a docker container).
- **Tenant:** A logical partition of a C3 Cluster. All tenants in a cluster generally share the same compute and storage resources. Data within tenants, while stored in a single database, are logically separated. C3 AI Suite users on one tenant can't see data stored on another tenant. In other words, users are only able to view data for tenants to which they are explicitly granted access.
- **Tag:** A logical partition of a tenant. A single tag hosts one C3 AI application (deployed package).
- **Package:** All the code a developer writes for an application. The C3 AI Suite runs a package on a tag.
- **Provisioning:** Deploying a package onto a tenant/tag in a C3 Cluster.
- **Static Console:** A browser-based tool that C3 AI developers use to interact with the C3 AI Suite. Static console is available on all modern browsers, including Google Chrome, Mozilla Firefox, and Apple Safari. Developers use the console to query data, evaluate expressions, view documentation, monitor environments, develop and test code, and manage environment configurations. You can access the static console at the url 'https://<vanity_url>/static/console' (replacing <vanity_url> with your Vanity Url).
- **Metric:** Expressions that transform raw data into a time-series.

C3 Cluster Overview

The C3 AI Suite is a Platform as a Service (PaaS), which enables organizations to build, deploy, and operate enterprise-scale Big Data, AI, and IoT applications. The C3 AI Suite can be deployed on any private or public cloud infrastructure such as AWS, Azure, and Google Cloud Platform. When developing and operating applications, a C3 cluster is responsible for managing and supporting all the features of the C3 AI Suite. A C3 Cluster has at least one Master node and many Worker nodes. Master nodes prioritize and distribute jobs to Worker nodes and handle user requests. Worker nodes carry out jobs, allocated by the Master node. Other components of a C3 Cluster include databases (e.g., Postgres, Cassandra, Azure Blob), logging services (i.e., Splunk), and Jupyter. Atop these hardware or virtualized cloud resources is a logical software structure, with the highest level being a Cluster. A C3 Cluster is broken out into numerous tenants. Tenants are logically separated from each other (i.e., a particular tenant's data and packages are not accessible or visible to any other tenants), and contain many tags. Tags host C3 AI Packages (i.e., the code that C3 AI developers write and provision to the C3 AI Suite). A typical multi-tag, multi-tenant C3 Cluster is shown in a logical diagram below:



To learn more about the architecture of a C3 cluster, please see the training materials here:

- Developer Documentation
 - <https://developer.c3.ai/docs/7.12.25/guide/guide-c3aisuite-basic/arch-home>
 - <https://developer.c3.ai/docs/7.12.25/guide/guide-c3aisuite-basic/arch-clusters-tenants-and-tags>
- C3.ai Academy Videos:
 - Packages
 - Clusters, Tenants, and Tags

Provision a C3 AI Package

To provision a package to your tag, follow the instructions available at the [DTI Guide: Provisioning](#).

To run the examples in this guide you will need to provision the 'baseCovidDataLake' by following the directions in the 'COVID-19 Data Lake Provisioning' section.

To learn more about provisioning, please see the C3 AI Develop Documentation here:

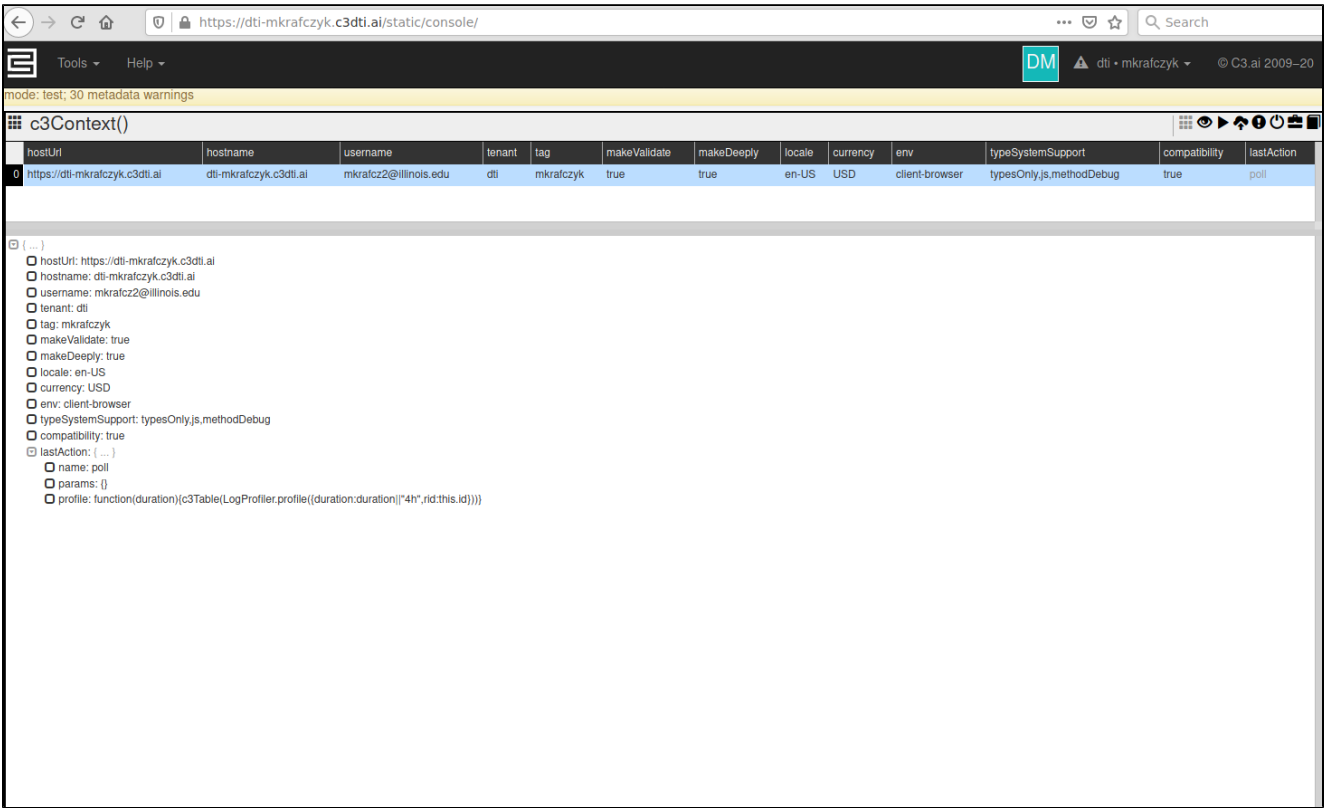
- Developer Documentation
 - <https://developer.c3.ai/docs/7.12.17/topic/arch-provisioning>
- C3.ai Academy Videos:
 - Provisioning

Connecting to the C3 AI Suite

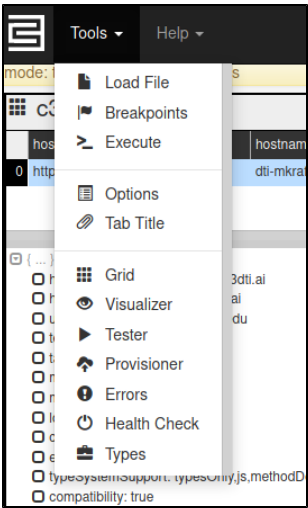
The static console is the main tool that developers use to interact with the C3 AI Suite. However, we anticipate that most DTI members will use Python (via Jupyter notebook) for data analysis. That being said, the static console is an essential part of working with the C3 AI Suite and you will use it frequently. For example, the static console is the best place to find documentation tailored directly to your package. It's also a great place to quickly test queries as no specialized environments need to be set up to use it. Static console is ready-to-go in all modern browsers, including Google Chrome, Mozilla Firefox, and Apple Safari.

Accessing the Static Console

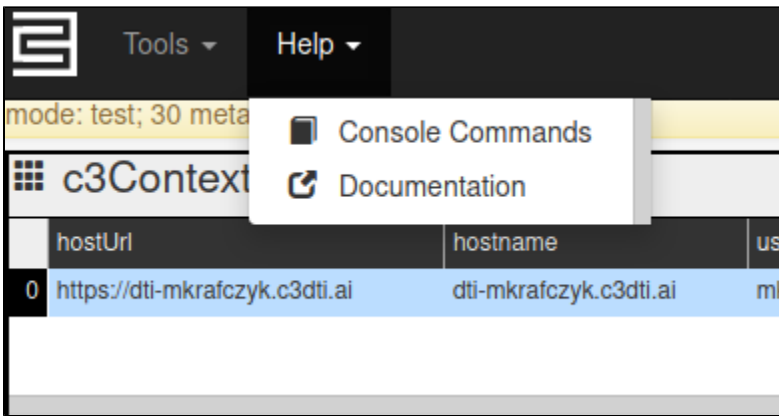
Once you have provisioned a package to your tag, navigate to the static console page at this url: 'https://<vanity_url>/static/console' (replacing <vanity_url> with your Vanity Url provided in your C3.ai DTI Training Cluster Onboarding Email). The static console page looks like this:



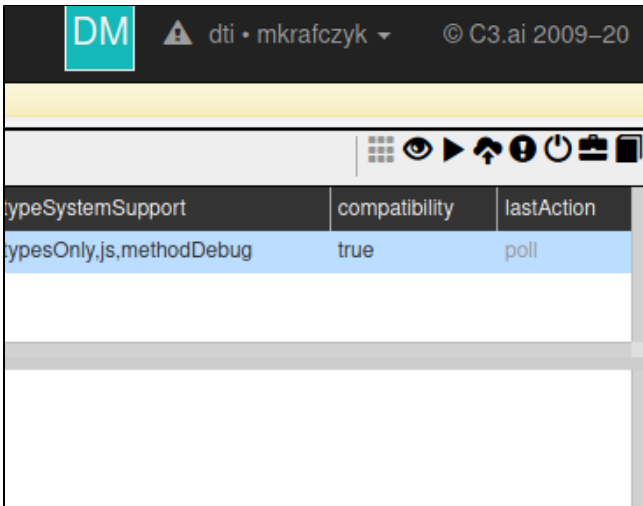
The 'Tools' drop-down menu in the upper left-hand corner contains a list of available developer tools. The most relevant tool is the Provisioner, though there are also utilities for loading JavaScript files, debugging JS code, and inspecting Errors.



The 'Help' drop-down menu in the upper left-hand corner allows users to access console documentation and a C3 Cluster hosted documentation portal.



Most tools are also accessible through a series of icons in the upper right-hand corner:



Using the Static Console

Developers interact with the static console through the JavaScript console tab in the browser. When the static console page loads (or when you run the `c3I mportAll()` command), JavaScript methods associated with all of your Package's defined Types are populated. You can write and run JavaScript code directly in the console tab to interact with your package.

You can also open Javascript console with the 'Ctrl+Shift+I' keyboard shortcut (in most browsers). Javascript console is also available through the browser's developer tools. If the 'Ctrl+Shift+I' keyboard shortcut doesn't work for you, review your browser's documentation on developer tools. Here's how the static console looks in Firefox, with the JavaScript console open:

The screenshot shows the DTI static console interface. The left pane displays the `c3Context()` object, which is a table with columns: `hostUrl`, `hostname`, `username`, `tenant`, `tag`, `makeValidate`, `makeDeeply`, `locale`, and `currency`. The right pane shows the console output, which includes several warnings and errors, such as "This page uses the non standard property 'zoom'", "Layout was forced before the page was fully loaded", and "Synchronous XMLHttpRequest on the main thread is deprecated".

Finally, let's write some JavaScript commands to see the console in action!

The screenshot shows the DTI static console interface. The left pane displays the `OutbreakLocation` table, which has columns: `id`, `v#`, `name`, `locationType`, `populationCDS`, `location`, `typelident`, and `meta`. The right pane shows the console output, which includes several warnings and errors, such as "This page uses the non standard property 'zoom'", "Layout was forced before the page was fully loaded", and "Synchronous XMLHttpRequest on the main thread is deprecated".

Tutorial Video

The DTI Team have recorded a short video introducing and describing the static console functionality:

Your browser does not support the HTML5 video element

Console Commands

Here are common JavaScript console commands used on the static console page.

- **c3ImportAll**: A console command which loads all the C3 Types from your package into the static console. Always be sure refresh the static console page or run `c3ImportAll()` after provisioning a new package.
- **c3Grid**: A console command to display a table of data stored within a C3 Type. (e.g., data returned from a fetch operation, or an evaluate operation among many others).
- **c3Viz**: A console command to produce quick visualizations or plots for some C3 Types. (e.g., time-series data like `EvalMetricsResult`)
- **c3ShowType**: A console command to access documentation for a given C3 Type. (e.g., `c3ShowType(OutbreakLocation)`)

Official C3 AI Training Materials on Static Console

- Developer Documentation
 - <https://developer.c3.ai/docs/7.12.25/topic/console-home>
 - <https://developer.c3.ai/docs/7.12.25/topic/console-documentation-via-console>
- **C3.ai Academy Videos**:
 - Console Overview
 - Console Orientation

Using Python with the C3 AI Suite

We anticipate most DTI researchers will want to use Python for data analysis. There are two options to connect to a C3 Cluster via Python. Please follow the links below for detailed information about each.

- [Use C3 integrated Jupyter Notebook](#)
- [Connecting to a C3 Cluster from a Remote Python Session](#)

To learn more about the general structure of a C3 cluster, please see the resources here:

- Developer Documentation
 - <https://developer.c3.ai/docs/7.12.25/guide/guide-c3aisuite-basic/ds-home>
 - <https://developer.c3.ai/docs/7.12.25/guide/guide-c3aisuite-basic/ds-python-apis>
 - <https://developer.c3.ai/docs/7.12.25/guide/guide-c3aisuite-basic/ds-home>
 - <https://developer.c3.ai/docs/7.12.25/guide/guide-c3aisuite-basic/ds-jupyter-notebooks>
 - <https://developer.c3.ai/docs/7.12.25/guide/guide-c3aisuite-basic/actionruntime>
- **C3.ai Academy Videos**:
 - Python APIs
 - Python Runtimes

Fetching Instances of C3 Types

All data in the C3 AI Suite are stored in C3 Types. Users can access data from a Type with the 'fetch' method. Behind the scenes, the 'fetch' method submits a query directly to the database underlying a Type, and retrieves and presents the query results.

The C3 AI Suite returns the 'fetch' query's response, which includes:

- data from the Type itself;
- Metadata for the 'fetch' query (e.g., the number of objects, whether additional data exists in the database) into the `FetchResult` type for data analysis (see example below).

To learn more about the 'fetch' method, please see the C3 AI resources here:

- Developer Documentation
 - Fetching and Filtering: <https://developer.c3.ai/docs/7.12.25/topic/tutorial-fetching-and-filtering>
 - Fetch and Filter Basics: <https://developer.c3.ai/docs/7.12.25/topic/console-data-exploration-fetch-and-filters>
 - Fetch and Filter Advanced: <https://developer.c3.ai/docs/7.12.25/topic/console-fetch-and-filters-advanced>
- **C3.ai Academy Videos**:
 - Fetch & Filters

Users can also provide a `FetchSpec` (or parameters) to the 'fetch' method to describe particular data to retrieve (e.g., only retrieve gene sequences collected in Germany). The `FetchSpec` can be 'empty' (e.g., `OutbreakLocation.fetch()`), or contain several parameters to return a subset of the data.

Some example `FetchSpec` parameters include:

- **filter**: Filter expression to return a subset of the data (e.g., `age <= 20`). Filter expressions must evaluate to a Boolean type (i.e., true or false)
- **limit**: The maximum number of rows that should be returned. By default, if no limit is specified, the C3 AI Suite returns 2,000 rows from the C3 Type. Specifying a limit is often helpful to debug a fetch 'method' without returning too many records.

- **include:** Specifies the particular fields from a C3 Type to return to the FetchResult. By default, if no include spec is defined, all fields from the Type will be returned.
- **order:** Specifies the order to return the query's results (either "ascending" or "descending").

Note: Please see the official FetchSpec documentation for a full list of parameters: <https://developer.c3.ai/docs/7.12.17/type/FetchSpec>

Examples of Fetch Calls

The OutbreakLocation Type contains information from various locations for which the Covid-19 Data Lake has virus-related information. We can fetch 0 outbreakLocation records for which the 'latestTotalPopulation' field exists (i.e., is not null). We can also retrieve these records in descending order by their 'countryArea':

```
res = OutbreakLocation.fetch({
  'limit': -1,
  'filter': 'exists(latestTotalPopulation)',
  'order': 'descending(latestTotalPopulation)',
  'include': 'id, name, latestTotalPopulation, populationOfAllChildren, countryArea, countryCode'
})
```

And we can show these results in the C3 AI static console using the c3Grid command:

The screenshot shows the C3 AI static console interface. The main panel displays a table titled "OutbreakLocation 3443 results". The table has columns: id, v#, name, latestTotalPopulation, and populationOfAllChildren. The first few rows are:

	id	v#	name	latestTotalPopulation	populationOfAllChildren
0	India	9437277	India	1656553632	1656553632
1	China	6226460	China	1301627048	1308870495
2	Nigeria	14024732	Nigeria	416996080	416996080
3	UnitedStates	8729806	UnitedStates	398328349	728888841
4	Indonesia	14155807	Indonesia	300183166	300183166
5	Pakistan	7602307	Pakistan	290847790	290847790
6	Brazil	7667743	Brazil	232304177	232304177
7	Ethiopia	14221340	Ethiopia	228066276	228066276
8	Bangladesh	14090270	Bangladesh	193092763	193092763
9	Egypt	14614559	Egypt	168937974	168937974

Below the table, there is a detailed view of the first record (India) showing fields like latestTotalPopulation, populationOfAllChildren, countryArea, countryCode, id, name, and meta. The meta field includes fetchInclude, fetchType, version, and typeIdent.

The right panel shows the console output of the fetch call, displaying the result object and the c3Grid command.

You can run this same fetch in Python:

```
raw_data = c3.OutbreakLocation.fetch({
  'limit': -1,
  'filter': 'exists(latestTotalPopulation)',
  'order': 'descending(latestTotalPopulation)',
  'include': 'id, name, latestTotalPopulation, populationOfAllChildren, countryArea, countryCode'
})
```

Additional details on "Fetching in Python" are available in this C3 AI Developer documentation: <https://developer.c3.ai/docs/7.12.25/topic/ds-jupyter-notebooks>

Additional examples of fetch calls can be found here:

- <https://github.com/c3aidti/casesExample>
- <https://github.com/c3aidti/genomicsExample>

- <https://github.com/c3aidti/HouseCoverageExample>
- <https://github.com/c3aidti/VaccineListing>
- <https://github.com/c3aidti/EpidemiologyExample>

Tutorial Video

This tutorial video goes over fetching and filtering:

Your browser does not support the HTML5 video element

The fetchCount Method

Another useful command is 'fetchCount'. Like 'fetch', users can also provide a FetchSpec (or parameters) to 'fetchCount'. The 'fetchCount' method then returns the number of records that match the FetchSpec. This is useful when trying to determine whether a given search is refined enough.

```
OutbreakLocation.fetchCount({'filter': 'exists(latestTotalPopulation)'})
```

You can run the same 'fetchCount' in python:

```
c3.OutbreakLocation.fetchCount(spec={'filter': 'exists(latestTotalPopulation)'})
```

To learn more about the 'fetchCount' method, please see the fetchCount method definition in the Persistable Type documentation: <https://developer.c3.ai/docs/7.12.25/type/Persistable>

Converting Fetch results to usable forms in Jupyter Notebook

When using a Jupyter Notebook, C3 AI developers typically modify FetchResults for data analysis. This section shows a couple of ways to convert FetchResults into easy-to-analyze forms.

Python

In python, first retrieve the 'objs' field from the FetchResults object, and then call the toJson() function. The toJson() function returns an array of dictionaries each with keys equal to the requested fields of the fetched C3 Type. Using the Pandas library, this array can be turned into an analysis-ready DataFrame, as the below example shows:

```
import pandas as pd
df = pd.DataFrame(raw_data.objs.toJson())
df.head()
df.drop('meta', axis=1, inplace=True)
df.drop('type', axis=1, inplace=True)
df.drop('version', axis=1, inplace=True)
df.drop('id', axis=1, inplace=True)
df.head()
```



```
In [5]: 1 df = pd.DataFrame(raw_data.objs.toJson())
2 df.head()
3 df.drop('meta', axis=1, inplace=True)
4 df.drop('type', axis=1, inplace=True)
5 df.drop('version', axis=1, inplace=True)
6 df.drop('id', axis=1, inplace=True)
7 df.head()
```

Out[5]:

	typelident	name	latestTotalPopulation	populationOfAllChildren	countryArea	countryCode
0	EP_LOC	India	1.656554e+09	1.656554e+09	2973193.0	IN
1	EP_LOC	China	1.301627e+09	1.308870e+09	NaN	NaN
2	EP_LOC	Nigeria	4.169961e+08	4.169961e+08	910768.0	NI
3	EP_LOC	UnitedStates	3.983283e+08	7.288888e+08	9148655.0	US
4	EP_LOC	Indonesia	3.001832e+08	3.001832e+08	1811569.0	ID

Users can then manipulate the resulting DataFrame, using common programming libraries and frameworks.

ExpressionEngineFunctions

The C3 AI Suite also provides a pre-built library of "ExpressionEngineFunctions". Expression Engine Functions take a variety of arguments and perform various data processing tasks. For example, the function 'contains' takes two strings as arguments and checks whether the first argument contains the second argument. The function 'lowerCase' takes as input a string and returns that same string with all lowercase letters. In addition to these string processing functions, the C3 AI Suite's ExpressionEngine includes many math functions (such as 'log', 'avg', and 'abs') which operate on various input data types (e.g. int, double, float).

The ExpressionEngine Functions are used in several places, such as:

- 'fetch' filters
- Simple and compound metric expressions
- tsDecl metric values

To learn more about ExpressionEngineFunctions, please see the C3 AI resources here:

- Developer Documentation
 - <https://developer.c3.ai/docs/7.12.25/type/ExpressionEngineFunction>
- C3.ai Academy Videos:
 - ExpressionEngineFunctions

Computations on C3 Types using Evaluate

Using the 'evaluate' method, developers can run aggregations or other computations on data fetched from a C3 Type. (e.g., compute the average area across all countries with area data available, in the OutbreakLocation Type).

The 'evaluate' method takes several parameters:

- **projection:** [Required] A comma-separated list of expressions (from the ExpressionEngineFunction library) to apply to data from a C3 Type (e.g., avg, unique, min, max). You can simply think about a projection as the columns/fields, calculated or otherwise, which the "evaluate" method should return.
- **group:** A comma-separated list of columns/fields, to group the aggregated/transformed data by (e.g, compute the average area by the 'locationType' field in OutbreakLocation). Please note, in any 'evaluate' command, all columns in the 'group' field **MUST ALSO BE** in the 'projection' field, as the example below shows.
- **having:** A SQL style having clause.
- **order:** A comma-separated list of columns/fields, to order aggregated/transformed data by. Users can access data in 'ascending' or 'descending' order. Please note, in any 'evaluate' command, all columns in the 'order' field **MUST ALSO BE** in the 'projection' field.
- **filter:** A filter expression that restricts the rows in a C3 Type on which the evaluate method is run.

In static console, 'c3Grid' displays the 'evaluate' method results nicely:

(Note: the 'locationType' expression within the 'group' field is also within the 'projection' field. This is required.)

```
var eval_result = OutbreakLocation.evaluate({
  'projection': 'avg(countryArea), locationType',
```

```

'group': 'locationType',
'filter': 'exists(countryArea) && exists(locationType)'
})
c3Grid(eval_result)

```

The screenshot shows a web application interface with a table titled 'EvaluateResult' and a console panel on the right.

Table: EvaluateResult

	c0	c1
0	627982.3684848485	country
1	2085.8	state

Console:

- Warning: This page uses the non standard property "zoom". Consider using calc() in the relevant property values, or using "transform" along with "transform-origin: 0 0".
- Warning: Layout was forced before the page was fully loaded. If stylesheets are not yet loaded this may cause a flash of unstyled content.
- Warning: Synchronous XMLHttpRequest on the main thread is deprecated because of its detrimental effects to the end user's experience. For more help http://xhr.spec.whatwg.org/
- Warning: Starting in default tag dti / mkrafczyk
- Warning: onmozfullscreenchange is deprecated.
- Warning: onmozfullscreenerror is deprecated.
- Warning: Imported 5393 types (all) from dti / mkrafczyk in 11s

Code Snippet:

```

var eval_result = OutbreakLocation.evaluate({
  'projection': 'avg(countryArea), locationType',
  'group': 'locationType',
  'filter': 'exists(countryArea) && exists(locationType)'
})

```

Users can also run the 'evaluate' method in python. In this case, users often modify the 'evaluate' method's results for data analysis. To view and analyze the 'evaluate' method's results in Python, please use the helper function available in C3 DTI's c3python module here: <https://github.com/c3aidti/c3python>

NOTE: The 'locationType' expression within the 'group' field is also within the 'projection' field. This is required.

```

eval_spec = {
  'projection': 'avg(countryArea), locationType',
  'group': 'locationType',
  'filter': 'exists(countryArea) && exists(locationType)'
}
eval_res = c3.OutbreakLocation.evaluate(eval_spec)
df = c3python.EvaluateResultToPandas(result=eval_res, eval_spec=eval_spec)

```

In [8]:

1 df

Out[8]:

avg(countryArea) locationType

0	627982.368485	country
1	2085.800000	state

Here's another example of running the 'evaluate' method in Python, this time using the 'order' parameter as well:

NOTE: The 'count(ethnicity)' expression within the 'order' field is also within the 'projection' field. This is required.

```
spec = c3.EvaluateSpec(
    projection="ethnicity, count(ethnicity)",
    order='descending(count(ethnicity))',
    group="ethnicity"
)
c3python.EvaluateResultToPandas(result=c3.SurveyData.evaluate(spec), eval_spec=spec)
```

Out[5]:

	ethnicity	count(ethnicity)
0	white	17288.0
1	black	7320.0
2	hispanic-latino	4309.0
3	other-mixed	2605.0
4	asian	1651.0

To learn more about the 'evaluate' method, please see the C3 AI resources here:

- Developer Documentation
 - <https://developer.c3.ai/docs/7.12.25/topic/tutorial-the-evaluate-method>
 - <https://developer.c3.ai/docs/7.12.25/topic/console-evaluate-method>
- C3.ai Academy Videos:
 - The Evaluate Method

Developing Metrics on Timeseries data

The C3 AI Suite also offers several features to handle time series data. To interact with time series, C3 AI developers typically use simple and compound metrics. These metrics are used in several places in the C3 AI Suite such as:

- Alerts and Application Logic
- Machine Learning Features
- User Interface (to Visualize Data)

Timeseries Video Tutorial

To supplement the documentation below, we also have recorded a video lecture about Time Series data on the C3 AI Platform.

Your browser does not support the HTML5 video element

Simple Metrics

Simple metrics allow developers to produce time-series from raw data and are often used to construct more advanced metrics (i.e., Compound Metrics), in practice. Simple metrics are linked to a specific C3 Type and reference the timeseries data stored within that Type. To declare a simple metric, users should specify the following fields:

1. **id:** The simple metric's unique id, which should follow the convention "name_srcType" (e.g., Apple_DrivingMobility_OutbreakLocation).
2. **name:** The simple metric's name (e.g., Apple_DrivingMobility).
3. **description:** The simple metric's description (optional field).
4. **srcType:** The C3 Type the simple metric is analyzed on (e.g., OutbreakLocation).

5. **path**: The path from the srcType to the C3 Type that stores the raw data referenced by the simple metric (e.g., pointMeasurements)
Note: If the srcType itself stores the raw data referenced by the simple metric, path field is optional.
6. **expression**: The expression (or ExpressionEngineFunction) applied to the raw data, referenced by the simple metric (e.g., avg(avg(normalized.data.quantity))).
Note: The "normalized" key term instructs the simple metric to use normalized (instead of raw) data on the C3 AI Suite. To learn more about Normalization, see this C3 AI Developer Documentation: <https://developer.c3.ai/docs/7.12.25/topic/normalization>.

Here is an example of a Simple Metric:

```
met = c3.SimpleMetric(
    id='JHU_ConfirmedCases2_OutbreakLocation',
    name='JHU_ConfirmedCases2',
    srcType='OutbreakLocation',
    path="aggregateMeasurements.(measurementType == 'confirmed' && origin == "
        "'Johns Hopkins University')",
    expression='interpolate(avg(avg(normalized.data.value)), "PREVIOUS", "MISSING")'
)
```

To learn more about Simple Metrics, please see the C3 AI resources here:

- Developer Documentation
 - <https://developer.c3.ai/docs/7.12.25/topic/tutorial-simple-metrics>
 - <https://developer.c3.ai/docs/7.12.25/topic/metrics-simple-metrics>
- C3.ai Academy Videos:
 - Metrics
 - Simple Metrics

Another type of SimpleMetric is a tsDecl (Timeseries Declaration) metric. tsDecl metrics are often used to turn non-time series raw data (e.g., event data, status data, or data with irregular intervals) into time series. tsDecl metrics have the same fields as standard SimpleMetric, except for the 'tsDecl' field, which replaces the 'expression' field. tsDecl metrics may allow users the added flexibility to define new metrics that the expression field may not support. Using a tsDecl metric, the above metric can be re-written as:

```
met = c3.SimpleMetric(
    id='JHU_ConfirmedCases3_OutbreakLocation',
    name='JHU_ConfirmedCases3',
    srcType='OutbreakLocation',
    path="aggregateMeasurements.(measurementType == 'confirmed' && origin == "
        "'Johns Hopkins University')",
    tsDecl={
        'data': 'data',
        'treatment': 'AVG',
        'start': 'start',
        'value': 'value'
    }
)
```

To learn more about tsDecl metrics, please see the C3 AI resources here:

- Developer Documentation
 - <https://developer.c3.ai/docs/7.12.25/topic/metrics-tsdecl-metrics>
 - <https://developer.c3.ai/docs/7.12.25/topic/tutorial-tsdecl-metrics>
- C3.ai Academy Videos:
 - Time Series Declaration Metrics

Compound Metrics

Compound metrics allow C3 AI developers to manipulate or combine existing metrics into more complex time series. Compound metrics are built on top of one or many existing Simple or Compound metrics. Please note, to evaluate a Compound metric on a C3 Type, all Simple metrics used in that Compound metric must be defined on that Type as well. If not, an error is returned.

To declare a compound metric, users should specify the following fields:

1. **id**: The compound metric's unique id, typically the same as 'name' (e.g., BLS_UnemploymentRate).
2. **name**: The compound metric's name (e.g., BLS_UnemploymentRate).
3. **description**: The compound metric's description (optional field).
4. **expression**: The expression (or ExpressionEngineFunction) applied to the metrics underlying the Compound metric (e.g., "BLS_LaborForcePopulation ? 100 * BLS_UnemployedPopulation / BLS_LaborForcePopulation: null").

An example CompoundMetric is:

```
met = c3.CompoundMetric(
    id='JHU_CaseFatalityRate',
    name='JHU_CaseFatalityRate',
    expression='JHU_ConfirmedDeaths/JHU_ConfirmedCases',
)
```

To learn more about Compound metrics, please see the C3 AI resources here:

- Developer Documentation
 - <https://developer.c3.ai/docs/7.12.25/topic/metrics-compound-metrics>
 - <https://developer.c3.ai/docs/7.12.25/topic/tutorial-compound-metrics>
- [C3.ai Academy Videos](#):
 - Compound Metrics

Finding, Evaluating, and Visualizing Metrics

Users can find, evaluate, and visualize metrics built in the C3 AI Suite via the JavaScript console or a hosted Jupyter notebook.

Finding Metrics

All metrics that users build and deploy in the C3 AI Suite are also stored in C3 Types. To view a list of all the simple and compound metrics applicable to a C3 Type, run the 'listMetrics' method as shown below:

Javascript:

```
var metrics = OutbreakLocation.listMetrics()
c3Grid(metrics)
```

Python:

```
import pandas as pd
pd.DataFrame(c3.OutbreakLocation.listMetrics().toJson())
```

DTI Members using the Covid-19 Data Lake: While listMetrics does return a list, this is fairly bare bones if the 'description' field of a given metric isn't filled in. The [Covid-19 Data Lake API documentation](#) provides an extensive list of production-ready metrics along with detailed descriptions and usage examples.

After finding a metric, the next step is to evaluate on data in a C3 Type.

Evaluating Metrics

Metrics are evaluated with either the 'evalMetrics' or 'evalMetricsWithMetadata' methods. Behind the scenes, 'evalMetrics' and 'evalMetricsWithMetadata', fetch and transform raw data from a C3 Type into easy-to-analyze timeseries data. 'evalMetrics' is used to evaluate metrics provisioned (deployed) to a tenant/tag. 'evalMetricsWithMetadata' allows users to evaluate metrics either provisioned to a tenant/tag, or defined on-the-fly in JavaScript console, or a hosted Jupyter notebook (typically for debugging).

To learn more about the differences between 'evalMetrics' and 'evalMetricsWithMetadata' see the documentation here: <https://developer.c3.ai/docs/7.12.25/type/MetricEvaluatable>

To evaluate a metric, users must provide the following parameters (called an EvalMetricSpec) to the 'evalMetrics' or 'evalMetricsWithMetadata' methods:

1. **ids** ([string]): A list of ids in the C3 Type on which you want to evaluate the metrics (e.g., "Germany", "California_UnitedStates").
2. **expressions** ([string]): A list of metrics to evaluate (e.g., "JHU_ConfirmedCases", "Apple_DrivingMobility").
3. **start** (datetime): Start datetime of the time range to be evaluated (in [ISO 8601](#) format) (e.g., "2020-01-01").
4. **end** (datetime): End datetime of the time range to be evaluated (in [ISO 8601](#) format) (e.g., "2020-08-01").
5. **interval** (string): Desired interval for the resulting timeseries data (e.g., MINUTE, HOUR, DAY, MONTH, YEAR).

Here's an example of evaluating a metric in Python:

```
spec = c3.EvalMetricsSpec(
    ids=[ 'Illinois_UnitedStates', 'California_UnitedStates', 'UnitedStates' ],
    expressions=[ 'JHU_ConfirmedCases', 'JHU_ConfirmedDeaths' ],
    start='2020-01-01',
    end='2020-08-01',
    interval='DAY',
)

results = c3.OutbreakLocation.evalMetrics(spec=spec)
```

In Python, you can also specify the spec using a Dictionary without creating an EvalMetricsSpec Type:

```
results = c3.OutbreakLocation.evalMetrics(spec={
    'ids': [ 'Illinois_UnitedStates', 'California_UnitedStates', 'UnitedStates' ],
    'expressions': [ 'JHU_ConfirmedCases', 'JHU_ConfirmedDeaths' ],
    'start': '2020-01-01',
    'end': '2020-08-01',
    'interval': 'DAY',
})
```

The C3 AI Suite returns the evaluated metric results (a timeseries) into the 'EvalMetricsResult' type. With various helper functions, C3 AI developers may then convert this timeseries into a Pandas DataFrame (via "Dataset" type) for further data analysis or model development in a Jupyter notebook, as shown below:

```
ds = c3.Dataset.fromEvalMetricsResult(result=results)
df = c3.Dataset.toPandas(dataset=ds)
```

Additionally, users can visualize evaluated metric results directly in the web-browser (i.e., JavaScript console) with the 'c3Viz' function.

Here's an example of evaluating and visualizing in JavaScript console:

```
var spec = EvalMetricsSpec.make({
    'ids': [ 'Illinois_UnitedStates', 'California_UnitedStates', 'UnitedStates' ],
    'expressions': [ 'JHU_ConfirmedCases', 'JHU_ConfirmedDeaths' ],
    'start': '2020-01-01',
    'end': '2020-08-01',
    'interval': 'DAY'
})

var results = OutbreakLocation.evalMetrics(spec)
c3Viz(results)
```

Similarly, we don't have to explicitly create an EvalMetricsSpec type:

```
var results = OutbreakLocation.evalMetrics({
    'ids': [ 'Illinois_UnitedStates', 'California_UnitedStates', 'UnitedStates' ],
    'expressions': [ 'JHU_ConfirmedCases', 'JHU_ConfirmedDeaths' ],
    'start': '2020-01-01',
    'end': '2020-08-01',
    'interval': 'DAY'
})
c3Viz(results)
```

To learn more about evaluating and visualizing metrics, please see the C3 AI Developer Documentation here:

- <https://developer.c3.ai/docs/7.12.25/topic/metrics-evaluating-metrics>
- <https://developer.c3.ai/docs/7.12.25/topic/metrics-visualizing-metrics>
- <https://developer.c3.ai/docs/7.12.25/topic/ds-jupyter-notebooks>

Note: Metrics can only be evaluated on C3 Types that mix in the 'MetricEvaluatable' Type.

Additional Resources

Official C3 AI Developer Documentation:

- [Timeseries Normalization](#)
- [Timeseries Treatments](#)
- [Timeseries Metrics](#)
- [Simple Metrics](#)
- [Compound Metrics](#)

Review and Next Steps

For most data analysis, C3 AI developers run the 'fetch' and 'evalMetrics' methods. This C3.ai DTI Quickstart guide provides an introduction to these methods in which the C3 AI Suite is used as a read-only database accessed via APIs. In the following guides, you will learn how to run 'write' operations on the C3 AI Suite such as:

- Defining new Types
- Loading new data
- Clean-up databases in your tag
- Train machine learning models

Welcome to the start of your experience with the C3 AI Suite.