

Using Cloudera AI Inference service

Date published: 2020-07-16

Date modified: 2025-09-30



Legal Notice

© Cloudera Inc. 2025. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Cloudera AI Inference service Overview.....	4
Key Features.....	4
Key Applications.....	5
Terminology.....	5
Limitations and Restrictions.....	5
Supported Model Artifact Formats.....	6
Cloudera AI Inference service Concepts.....	6
Authenticating Cloudera AI Inference service.....	8
Managing Model Endpoints using UI.....	13
Creating a Model Endpoint using UI.....	14
Listing Model Endpoints using UI.....	16
Viewing details of a Model Endpoint using UI.....	17
Editing a Model Endpoint Configuration using UI.....	17
Managing Model Endpoints using API.....	18
Preparing to interact with the Cloudera AI Inference service API.....	19
Creating a Model Endpoint using API.....	19
Listing Model Endpoints using API.....	20
Describing a Model Endpoint using API.....	20
Deleting a Model Endpoint using API.....	21
Autoscaling Model Endpoints using API.....	22
Tuning auto-scaling sensitivity using the API.....	22
Running Models on GPU.....	23
Deploying models with Canary deployment using API.....	24
Interacting with Model Endpoints.....	25
Making an inference call to a Model Endpoint with an OpenAI API.....	25
Cloudera AI Inference service using OpenAI Python SDK client in a Cloudera AI Workbench Session.....	25
Cloudera AI Inference service using OpenAI Python SDK client on a local machine.....	26
OpenAI Inference Protocol Using Curl.....	27
Making an inference call to a Model Endpoint with Open Inference Protocol.....	27
Open Inference Protocol Using Python SDK.....	28
Open Inference Protocol Using Curl.....	28
Deploying Predictive Models.....	30
Accessing Cloudera AI Inference service Metrics.....	35

Cloudera AI Inference service Overview

Cloudera AI Inference service provides a production-grade serving environment for hosting predictive and generative AI. It is designed to handle the challenges of production deployments, such as high availability, performance, fault tolerance, and scalability. The Cloudera AI Inference service allows data scientists and machine learning engineers to deploy their models quickly, without worrying about the infrastructure and maintenance. Cloudera AI Inference service supports running 100 or more model endpoints simultaneously, provided that the underlying compute resources are adequately and correctly sized.

Cloudera AI Inference service is built with tight integration of *NVIDIA NIM* and *NVIDIA Triton Inference Server*, providing industry-leading inference performance on NVIDIA GPUs. Model endpoint orchestration and management are built with the *KServe* model inference platform, a Cloud Native Computing Foundation (CNCF) open-source project. The platform provides standard-compliant model inference protocols, such as *Open Inference Protocol* for predictive models and *OpenAI API* for generative AI models.

Cloudera AI Inference service is seamlessly integrated with the *Cloudera AI Registries* enabling users to store, manage, and track their AI models throughout their lifecycle. This integration provides a central location to store model and application artifacts, metadata, and versions, making it easier to share and reuse AI artifacts across different teams and projects. It also simplifies the process of deploying models and applications to production, as users can select artifacts from the registry and deploy them with a single command.

Related Information

[KServe](#)

[Open Inference Protocol](#)

[OpenAI API](#)

[NVIDIA NIM](#)

[NVIDIA Triton Inference Server](#)

[Cloudera AI Registry](#)

Key Features

The key features of Cloudera AI Inference service includes:

- **Easy to use interface:** Streamlines the complexities of deployment and infrastructure, meaningfully reducing time to value for AI use cases.
- **Real-time predictions:** Allows users to serve AI models in real-time, providing low latency predictions for client requests.
- **Monitoring and logging:** Includes functionality for monitoring and logging, making it easier to troubleshoot issues and optimize workload performance.
- **Advanced deployment patterns:** Includes functionality for advanced deployment patterns, such as canary and blue-green deployments, and supports A/B testing, enabling users to deploy new versions of models gradually and compare their performance before deploying them to production.
- **Optimized Performance:** Integrates with NVIDIA NIM microservices and NVIDIA Triton Inference Server to accelerate inference performance on NVIDIA accelerated infrastructure.
- **Model access:** Offers access to NVIDIA foundation models, tailored for NVIDIA hardware to increase inference throughput and to reduce latency.
- **REST API:** Provides APIs for deploying, managing, and monitoring of model endpoints. These APIs enable integration with continuous integration and continuous deployment (CI/CD) pipelines and other tools used in the Machine Learning Operations (MLOps) and Large Language Model Operations (LLMOps) workflows.

Key Applications

Large Language Models deployed on Cloudera AI Inference service with NVIDIA NIM enable the following applications:

- Chatbots & Virtual Assistants: Empowers bots with human-like language understanding and responsiveness.
- Content Generation & Summarization: Generates high-quality content or distills lengthy articles into concise summaries with ease.
- Sentiment Analysis: Understands user sentiments in real-time, driving better business decisions.
- Language Translation: Breaks language barriers with efficient and accurate translation services.

Terminology

Lists the Cloudera AI Inference service terminology and usage.

- CML Serving App: This is the term used by the Cloudera CLI to refer to a specific instance of Cloudera AI Inference service.
- Model Endpoint: This refers to a deployed model that has a URL endpoint accessible over the network.
- Model Artifacts: Files stored in Cloudera AI Registry that are necessary for deploying an instance of the model, such as model weights, metadata, and so on.
- API standard: The protocol that is exposed by a Model Endpoint. It can be either OpenAI (for NVIDIA NIM) or Open Inference Protocol for predictive models.
- Cloudera Workload Authentication Token: The bearer token used for authentication / authorization when accessing Cloudera AI Inference service API and model endpoints. Throughout this document this is referred to as “CDP_TOKEN”.
- Model ID: This is the ID assigned to the model when it is registered to the Cloudera AI Registry.
- Model Version: This is the version of a registered model in the Cloudera AI Registry.

Limitations and Restrictions

Lists the limitations and restrictions when using Cloudera AI Inference service.

- API Stability: Both the Cloudera AI Control Plane and Cloudera AI Inference service workload APIs and CLIs are under active development and are subject to change in a backward-incompatible way.
- Cloud Platforms: Cloudera AI Inference service is available on AWS and Azure.
- Supported Instance Types: Cloudera AI Inference service supports the same cloud instance types as those of Cloudera AI Workbenches with a few exceptions. See *Known Issues* for information on unsupported instance types. The type or size of the model you want to deploy determines the cloud compute instance type. Some highly optimized versions of Large Language Models, for instance, work only on specific GPU architectures.
- No Non-Transparent Proxy Support: Cloudera AI Inference service has not been tested with a non-transparent proxy (NTP) setup in a private cluster. However, it works in a vanilla private cluster.
- User-Defined Route (UDR) Support in Azure: Cloudera AI Inference service provides support for the UDR setup in Azure Kubernetes Service (AKS) clusters. Currently, the compute clusters UI does not support specification of subnets attached to UDRs. As a result, compute clusters utilizing UDR-attached subnets must be created using the CLI.



Note: When creating a cluster, ensure that the specified subnet is not used by another AKS cluster.

Example payload for creating compute clusters with a UDR-attached subnet using CLI:

```
{
  "environment": "[**ENVIRONMENT_NAME**]",
  "name": "[**CLUSTER_NAME**]",
```

```

    "network": {
      "subnets": [
        "[ ***SUBNET_NAME*** ]"
      ],
      "outboundType": "udr"
    },
    "skipValidation": false
  }

```

- **Public Load Balancer:** By default, Cloudera AI Inference service uses a private load balancer for cluster ingress. If you use a public load balancer instead, set the `usePublicLoadBalancer` parameter value to `true` in the creation payload.

If you are on AWS and use a private load balancer for cluster ingress, you must have a VPN connection between your corporate network and the Virtual Private Cloud (VPC) in which the Cloudera AI Inference service is deployed. The Cloudera AI Inference service UI requires VPN connection.
- **Logging:** All Kubernetes pod logs, including pods that are running model servers, are scraped by the platform log aggregator service (fluentd). Model endpoint logs can be viewed from the Cloudera AI Inference service GUI. To view logs of other pods, you must first obtain the kubeconfig of the cluster and use the `kubectl` command. Historical logs can be retrieved using the Generate Log Archive feature on the Cloudera AI Inference service administration UI.
- **Namespace:** Model endpoints can only be deployed in the serving-default namespace.

Related Information

[Managing Cloudera AI Inference service using Cloudera CLI](#)

[Known Issues](#)

Supported Model Artifact Formats

Lists Cloudera AI Inference service supported models:

- Text-generating Large Language Models (LLMs), embedding, ranking and object detection models packaged as *NVIDIA NIM*.
- Hugging Face transformer models supported by the vLLM engine.
- Predictive models in the ONNX representation, registered to Cloudera AI Registry from a Cloudera AI Workbench. See *Register an ONNX model to Cloudera AI Registry* as an example showing how to convert a sklearn model to ONNX and then register it to the Cloudera AI Registry. Refer to *Export a PyTorch model to ONNX* or *Getting Started Converting TensorFlow to ONNX* documentation regarding how models using these frameworks can be converted to the ONNX representation.

Related Information

[Register an ONNX model to AI Registry](#)

[Getting Started Converting TensorFlow to ONNX, ONNX Runtime documentation](#)

[Export a PyTorch model to ONNX, PyTorch documentation](#)

[NVIDIA NIM Large Language Models](#)

Cloudera AI Inference service Concepts

Learn the following Cloudera AI Inference service concepts before setting up the Cloudera AI Inference service.

Platform

Cloudera AI Inference service is a Kubernetes-native model inference platform built using the Cloud Native Computing Foundation (CNCF)-hosted KServe model orchestration system. The Cloudera AI Inference service is

built with enterprise-grade scalability, security and performance incorporated via integrations with NVIDIA NIM, NVIDIA Triton, and Cloudera. It is designed to serve trained models for production-level use cases.

Runtime

Runtimes are the basic building blocks that are responsible for loading trained model artifacts into memory, and providing APIs that client applications can invoke to run inference requests. The Runtimes also provide metrics with which to monitor the performance of the models. The supported Runtimes in this release include various NVIDIA NIM versions and Hugging Face transformer for text-generation and embedding tasks, and NVIDIA Triton for deep-learning models using the ONNX backend.

Autoscaling

Cloudera AI Inference service provides Cluster Node autoscaling and Model Endpoint autoscaling.

Cluster Node Autoscaling

If a Cloudera AI Inference service runs on an autoscaling Kubernetes cluster, it can be configured with multiple auto-scaling worker node groups. There are two default node groups that are meant to run certain core services, and an arbitrary number of worker node groups where user workloads are scheduled. You can add or delete worker node groups from the cluster. The autoscaling range of an existing node group in the cluster can be changed.

Model Endpoint Autoscaling

In addition to worker node autoscaling, Cloudera AI Inference service provides autoscaling at the model endpoint level by increasing or decreasing the number of replicas based on predefined, customizable scaling criteria. Scaling to or from zero replicas is also supported. The supported scaling criteria (or metrics) are Requests Per Second (RPS) and concurrency per replica of the model endpoint. For instance, you can configure your model endpoint to autoscale up if the number of concurrent RPS exceeds 100, so that request latencies are maintained at an acceptable level.

Model endpoint replicas are terminated when they have not received any request for a certain amount of time. For large language models, this idle timeout is set to 1 hour, whereas for other models it is set to 10 minutes.



Note: When the autoscale range lower bound is set to zero for model endpoints, the model endpoints are created without any active replicas. Therefore, these endpoints do not consume resources and will not be reachable until the first request is received. This behavior can result in an endpoint successfully created, but unable to scale up and accept traffic. Though this behavior is beneficial to save cloud costs and allows users to enable scale to zero without starting up any new pods, it also makes it harder to diagnose node group sizing and quota issues in a timely manner.

Autoscaling Latency

It is important to be aware of the time it takes to autoscale-up a model replica, which can adversely affect user experience. The following mathematical expression can be used to calculate the approximate scale-up latency:

$$T = T_n + T_c + T_d + T_m$$

where:

- T_n : The time needed for a newly scaled up worker node to be ready. This may be zero if the new model replica is scheduled on a node that is already in the cluster. It can also be very large to infinite, depending on the availability of the instance type requested. In general, larger instance types take longer to reach the ready status.
- T_c : The time needed to pull container images of the model replica pod. This is negligible if the node already has the images.

- **Td:** The time taken to download the model artifacts from the Cloudera AI Registry storage to the instance volume. This can vary widely depending on the size of the model, ranging from a few seconds to even hours for the largest models. For large language models, such as Llama 3.1 70b or bigger, this term is the dominant one.
- **Tm:** This is the time required to load the model objects from the instance volume to one or more GPU RAMs for models that require GPU, or to the system memory for CPU-only models.

So based on the above equation, you can see that, for example, to scale up a large language model deployment from 0 to 1 replica can take a few minutes to an hour or more. You must keep this in mind when planning your model deployments and the expected SLAs.

Canary Rollout and Rollback

Cloudera AI Inference service lets you roll out a new version of a model without bringing down the currently running version, using the canary rollout feature. When a new model endpoint is created using version A of a model, 100% of the traffic is routed by the system to the deployed model version. You can roll out another version, say B, under the same model endpoint URL and give it a percentage of the traffic, say 10%. If the new version rolls out successfully, the system will send 90% of the traffic to version A, and 10% to version B. If version B does not successfully come up for some reason, the system will continue sending 100% of the traffic to the good version, which is A.

Note that each version of the model that is rolled out consumes the same amount of system resources allocated to each model endpoint replica. Therefore, if each replica is configured to use 2 CPUs and 5Gi of memory, the resource footprint will double when versions A and B are running. If version B turns out to be good, you can send 100% traffic to it and have version A be scaled to 0 replicas to reduce the resource consumption back to that of a single version. If, after sending 100% traffic to version B, you realize that version A is preferred, there is a way to rollback to the previous version. You can set the traffic for version B to 0% and the traffic is automatically sent back to version A. Setting the traffic to 0% rolls back to the last version that was ready with 100% traffic.

Example:

Suppose you begin with version A receiving 100% of the traffic. A new model, version B, is introduced and allocated 25% of the traffic. Upon evaluation, version B underperforms compared to version A, based on the collected metrics. While version B is still live, you proceed to deploy version C with 100% traffic. However, version C also underperforms relative to version A. Instead of redeploying version A manually, you can set version C's traffic to 0%, and all traffic reverts to version A. Note that version B, being an intermediary version that never reached 100% traffic, is not eligible for rollback.

Authenticating Cloudera AI Inference service

Cloudera AI Inference service uses Cloudera Workload Authentication JSON Web Token (JWT) to authenticate users and clients that interact with all HTTP endpoints exposed by the service workload.

Authenticating using JWT: All client applications must present a valid JWT as an HTTP Authorization Bearer token, as shown in the following example:

```
$ export CDP_TOKEN=${JWT}
$ curl -H "Authorization: Bearer ${CDP_TOKEN}" <URL>
```

Supported JWT Issuers

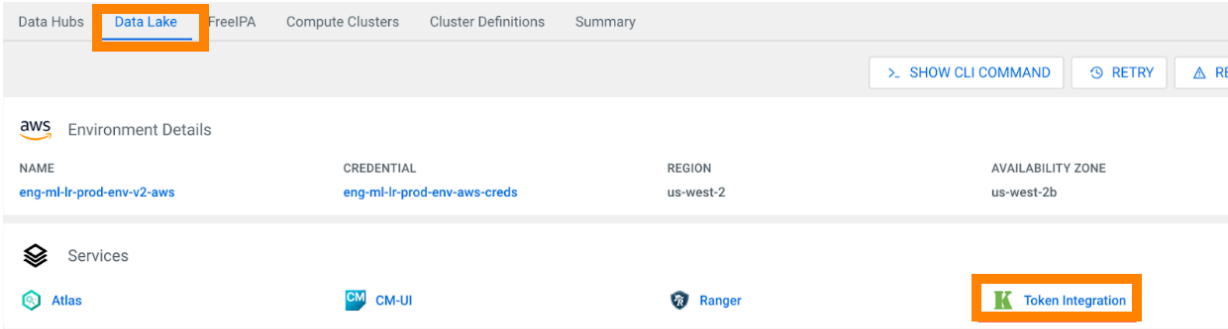
Cloudera AI Inference service supports JWTs issued by the following identity providers:

- Knox Gateway Server (running in the Data Lake environment)
- User Management Service (UMS) (part of the Cloudera Control Plane)

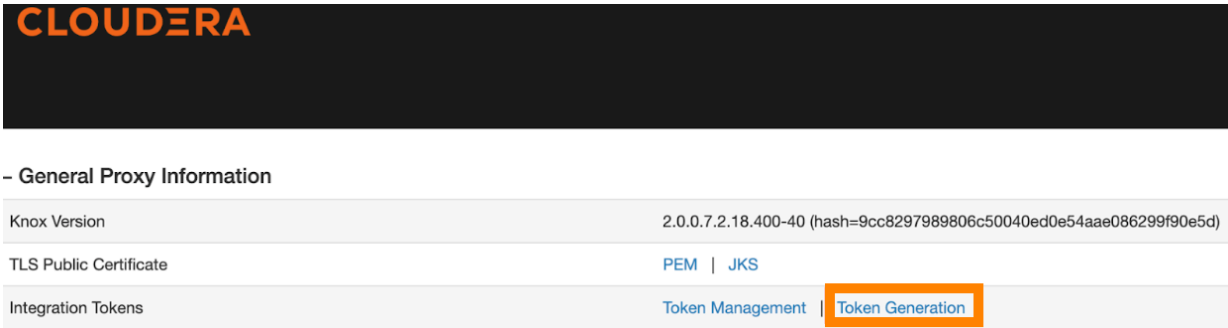
Using Data Lake Knox JWT

You can use the Data Lake Knox JWT for most authentication scenarios. For more information about Apache Knox configuration, see [Knox documentation](#).

- 1. Open the Data Lake tab in the CDP environment UI.
- 2. Click the Token Integration link.



- 3. In the newly opened window, click Token Generation.



4. Use the new window, set the desired token lifetime and generate a JWT.

CLoudERA

Token Generation

Token management backend is properly configured for HA and production deployments.

i Token Generation enables integration and API invocations by using the token as an authorization token. The token expires.

Comment:

i Configured maximum lifetime:
30 days

⌚ Lifetime (days, hours, mins):

0 1 0

👤 Generating token for (impersonation):

Generate Token

5. Copy the generated JWT from the Knox UI and use it in your application as an Authorization Bearer Token.

Using Auto-generated Kerberos JWTs

Some applications, such as Cloudera AI Workbench, automatically generates JWTs using the user's Kerberos credentials. These JWTs are automatically injected into all user workload pods, such as, workbench session pod, or an application pod and are stored at `/tmp/jwt`.

Applications can extract the JWT to authenticate to Cloudera AI Inference service. The following example shows a simple Python code that reads JWT from a file in Python:

```
import json
JWT = json.load(open('/tmp/jwt', 'r'))['access_token']
```

Use this token to authenticate your app to the AI Inference service.

Using a UMS JWT

You can obtain JWTs from the User Management Service (UMS) using either the CDP CLI or Cloudera AI Inference service UI.

- Option 1: Using the CDP CLI:

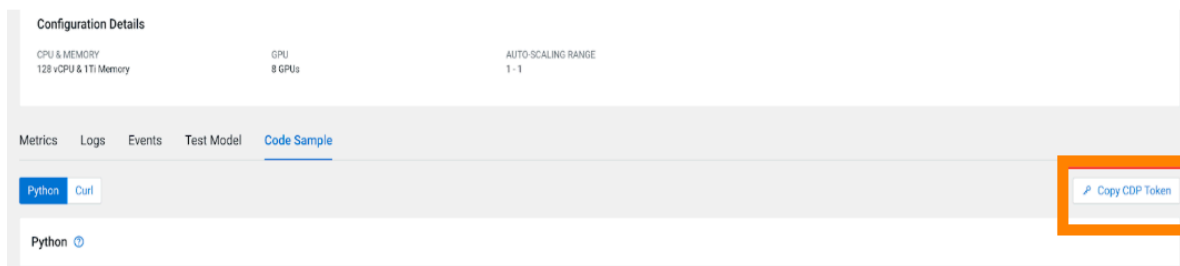
```
$ CDP_TOKEN=$(cdp iam generate-workload-auth-token --workload-name DE | jq -r '.token')
```



Note: The same workload name (for example, DE in the above command) can be used to authenticate with Cloudera AI Inference service.

- Option 2: Using Cloudera AI Inference service UI:

1. Open the model's Endpoint details page.
2. click Code Sample tab.
3. Click Copy CDP Token.



UMS Token Expiration

- UMS tokens expire after one hour by default.
- If a token is expired, the service returns an HTTP 401 Unauthorized response.
- To extend the token lifetime, use the following command:

```
$ cdp iam set-authentication-policy --workload-auth-token-expiration-sec <expiration-time-in-seconds>
```



Warning: This setting applies to the entire Cloudera account and is not limited to Cloudera AI Inference service. Increasing the token expiration time may introduce security risks, as JWT tokens, unlike API keys, cannot be revoked. Additionally, note that the above command requires the AUTHENTICATION_POLICY Cloudera entitlement.

Selecting the appropriate JWT token for Cloudera AI Inference service

Model Endpoint Management

Currently, Cloudera AI Registry supports only UMS JWT tokens. Therefore, when creating Model Endpoints in Cloudera AI Inference service, you must use a UMS JWT token. This is because the token is forwarded to the Cloudera AI Registry for the registry-level authentication.

All other use cases

For all other use cases, such as programmatic interactions and or automation workflow, including listing of Model Endpoints or performing inference calls, you should use the Data Lake Knox JWT instead of the UMS JWT. The Data Lake Knox JWT offers several advantages::

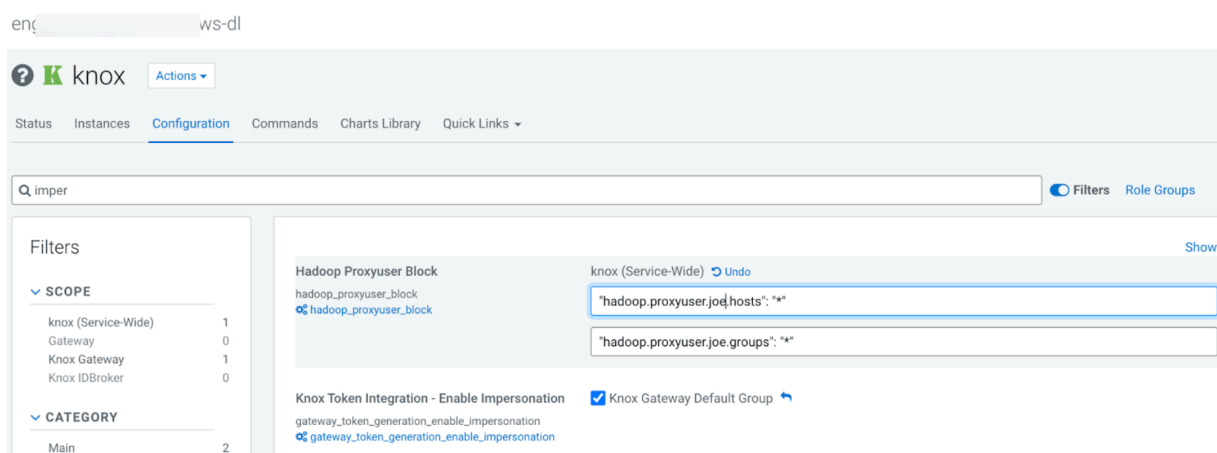
- **Faster Authentication :** Authentication latency of the Data Lake Knox token is lower compared to the UMS token because the issuer operates within the same environment as Cloudera AI Inference service.
- **Environment Scope:** Data Lake Knox JWT is scoped to a single Cloudera environment, unlike the tenant-wide scope of the UMS JWT.
- **Simplified Expiration Management:** Configuring the expiration policy for the Data Lake Knox JWT is simpler.
- **Better Machine User Support:** Creating and managing JWTs for machine users (or service accounts) is more easy when using the Data Lake Knox server.

Recommended best practices for authenticating long-running AI applications

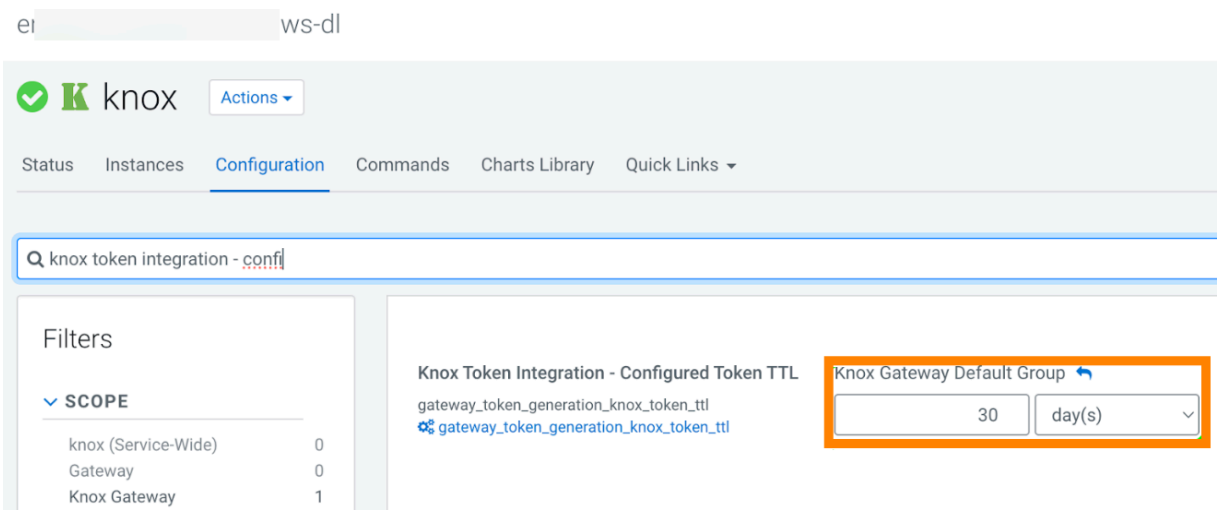
AI applications, such as LLM-based chatbots, are often developed using SDKs that expect long-lived authentication tokens, such as API keys, which either do not expire, or have extended expiration periods. For such scenarios, Cloudera recommends creating a machine user and configuring the AI application to use the machine user's credentials, that is, a JWT token, to interact with Model Endpoints hosted on Cloudera AI Inference service. Generating a long-lived JWT for a machine user is more secure than doing so for a human user.

Follow the below steps to create a machine user and generate a Data Lake Knox JWT for use with your AI application:

1. [Create a Machine User in Cloudera.](#)
2. In the environment hosting the Cloudera AI Inference service, assign the `EnvironmentUser` and `MLUser` resource roles to the machine user. For more information, see [Assign an environment resource role to a user.](#)
3. Ensure that Impersonation is enabled in the Knox configuration for the environment administrator. This can be done in the Data Lake's Cloudera Manager UI. The following example screenshot illustrates Knox impersonation enabled for an environment administrator with the username `joe`, allowing `joe` to impersonate all environment users.



4. Modify the maximum Time-to-Live (TTL) of Knox JWT token as needed. You have to restart the Knox service if you make configuration changes. The following screenshot illustrates an example where the maximum TTL is set to 30 days for this environment.



5. Generate Knox JWT for the machine user. Refer to the [Knox documentation](#) for additional options and detailed instructions. The following screenshot demonstrates the generation of a token for a machine user.



Note: You must use the machine user's workload username, which will include the `srv_` prefix added to the machine user's username.

Token Generation

Token management backend is properly configured for HA and production deployments.

i Token Generation enables integration and API invocations by using the token as an auth identity and is active until expired.

Comment:

i Configured maximum lifetime:

30 days

⌚ Lifetime (days, hours, mins):

3000

👤 Generating token for (impersonation):

srv_caii-prod-machine-user

Generate Token

6. Copy the generated Knox JWT and configure your AI application to use it as a bearer token to authenticate with Cloudera AI Inference service.
7. After generating the required token, you can optionally revert the maximum TTL of the Knox JWT token to its original value to prevent human users generating long-lived tokens

Managing Model Endpoints using UI

Cloudera AI Inference service lets you deploy models saved in the Cloudera AI Registry.

Cloudera AI Inference service provides a graphical user interface accessible from the Cloudera AI Control Plane for managing model endpoints. HTTP/REST APIs are also available for programmatic model endpoint management.

Cloudera AI Inference service UI provides new features to view, create, and edit model endpoints from the control plane UI. Each Cloudera AI Inference service instance can have multiple deployed model endpoints. The Model Endpoints page provides a comprehensive list view of all the deployed model endpoints in a given Cloudera account.

Related Information

[Supported Model Artifact Formats](#)

Creating a Model Endpoint using UI

The **Create Model Endpoint** page allows you to select a specific Cloudera AI Inference service instance and a model version from Cloudera AI Registry to create a new model endpoint.

About this task

The following steps illustrate how to create a Llama 3.1 model endpoint.

Procedure

1. In the **Cloudera** console, click the Cloudera AI tile.
The **Cloudera AI Workbenches** page displays.
2. Click Model Endpoints under Deployments on the left navigation menu.
The **Model Endpoints** landing page is displayed.

3. Click the Create Endpoint button.

The **Create Endpoint** page displays the details to create an endpoint.

Model Endpoints / Create Endpoint

Endpoint Details ⓘ

* Select Environment & Cluster

eng-ml-int-env-aws-v2 serving-single-gpu-aws

* Name

my-llama-31-8b-instruct-a10gx2

Description

My first llama 3.1

Served Model Builder ⓘ

* Model

llama-31-8b-instruct-a10gx2

* Version

1

Traffic

0 100

Resource Profile ⓘ

ⓘ This model requires GPU Model A10G and 2 GPU

* Instance Type

g5.12xlarge	48 CPU	4 GPU	192 GiB	A10G
-------------	--------	-------	---------	------

GPU

2

* CPU

10 vCPU

* Memory

64 Gi

Autoscale Range

0 10

1 2

Advanced Options

Autoscale Metric Type

☐ RPS ☒ Concurrency

Target Metric Value ⓘ

100

Tags ⓘ

mytag	value
-------	-------

Create Endpoint Cancel

4. In the Select Environment & Cluster dropdown list, select your Cloudera environment and the Cloudera AI Inference service instance within which you want to create the model endpoint.

5. In the Name textbox, enter a name for the model endpoint.

6. In the Description textbox add a short description of the model endpoint. Ensure that the description text you enter is below 5000 characters.

7. Under Served Model Builder, select the model (Model), and modelversion (Version) from the dropdown you want to deploy. Using the Traffic slider bar, specify the traffic split between different model versions that you deploy. It is always set to 100% for the first model version which you cannot change.
8. Under Resource Profile, select the type of the instance from the Instance Type dropdown list. For *NVIDIA NIM* models this field is mandatory. Specify which compute node instance type you wish to run on your model replicas. The instance type you choose depends on the capabilities of the instance type and on what is required by the NVIDIA NIM. The field is optional for normal predictive model endpoints.
9. In the GPU field, specify the number of GPUs each model endpoint replica requires. This field gets populated automatically for NVIDIA NIM models.
10. In the CPU field, specify the number of CPUs each model endpoint replica requires.
11. In the Memory field, specify the amount of CPU memory each model endpoint replica requires.
12. Using the Autoscale Range slider bar, specify the minimum and maximum number of replicas for the model endpoint. Based on which autoscaling parameter you choose, the system scales the number of replicas to meet the incoming load.
13. Under Advanced Options, in the Autoscale Metric Type select the autoscale metric type as Requests per second (RPS) or Concurrency per replica of the model endpoint.
If you choose to scale as per RPS and the Target Metric Value is set to 200, then the system automatically adds a new replica when it sees that a single replica is handling 200 or more requests per second. If the RPS falls below 200, then the system scales down the model endpoint by terminating a replica.
14. In Tags, add any custom key and value pairs for your own use.
15. Click Create Endpoint to create the model endpoint.

It can take tens of minutes for the model endpoint to be ready. The time taken is determined by the following points:

- whether a new node has to be scaled-in to the cluster.
- the time taken to pull the necessary container images.
- the time taken to download model artifacts to the cluster nodes from the Cloudera AI Registry.

In the above example, we are creating a model endpoint for the instruct variant of Llama 3.1 8B, which has been optimized to run on two NVIDIA A10G GPUs per replica. The cluster we are deploying into is in AWS, and it has a node group consisting of g5.12xlarge instance types. In this example there are no other node groups that have instance types containing A10G GPUs, so the g5.12xlarge instance type is the only choice. For NVIDIA NIM models that specify GPU models and count, the GPU field of the resource configuration page will be filled in automatically by the UI.



Important: We set the autoscale range as 1-2, with the scaling metric set to concurrency. The scale metric target is set to 100, meaning that if the number of concurrent requests per replica exceeds 100, the system will automatically add a new replica, up to a maximum of two replicas. You must ensure that your cluster has sufficient capacity to expand the requisite node group to run the desired number of replicas.

Listing Model Endpoints using UI

The **Model Endpoints** landing page displays a table view that represents a comprehensive list of all the model endpoints across the running Cloudera AI Inference service instances in your Cloudera account.

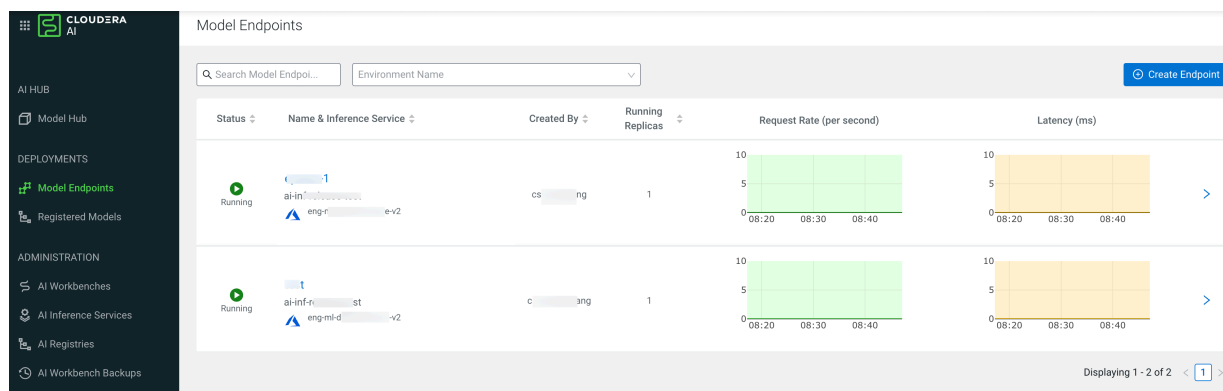
Procedure

1. In the **Cloudera** console, click the Cloudera AI tile.

The **Cloudera AI Workbenches** page displays.

- Click Model Endpoints under Deployments on the left navigation menu.

The **Model Endpoints** landing page is displayed.

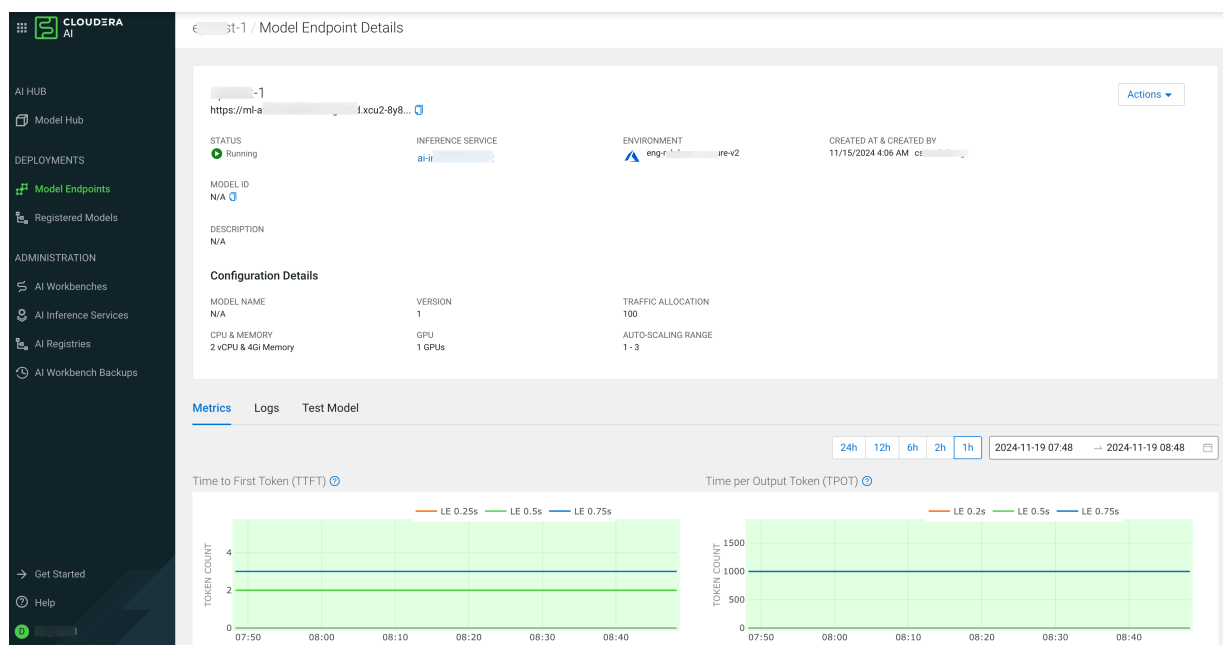


Viewing details of a Model Endpoint using UI

You can view the information of a Model Endpoint, edit its configuration, view metric charts on resource utilization, and so on.

Procedure

- In the **Cloudera** console, click the Cloudera AI tile.
The **Cloudera AI Workbenches** page displays.
- Click Model Endpoints under Deployments on the left navigation menu.
The **Model Endpoints** landing page is displayed.
- Select a model endpoint to see its details.



Editing a Model Endpoint Configuration using UI

You can edit the configuration of a specific model endpoint for resources or select model versions.

Procedure

1. In the **Cloudera** console, click the Cloudera AI tile.
The **Cloudera AI Workbenches** page displays.
2. Click Model Endpoints under Deployments on the left navigation menu.
The **Model Endpoints** landing page is displayed.
3. Select a model endpoint that you want to edit.
4. Select Edit Configuration from the Actions dropdown list.
5. Edit the resource profile or add model versions.
 - a. In the Resource Profile tab, select the instance type, and specify the CPU, Memory, GPU, and the autoscale range.
 - b. In the Served Models tab, add new model versions and specify the traffic percentage for the model versions.

Edit Configuration
✕

Served Models
Resource Profile

* Instance Type

Select Instance Type
▼

* CPU

1
vCPU

* Memory

2
Gi

GPU

0

Autoscale Range

0
10

-

1
1

Cancel

Update

6. Click Update.

Managing Model Endpoints using API

You can use API to create, view, list, edit, describe, and delete model endpoints.

Related Information

[Curl documentation](#)

Preparing to interact with the Cloudera AI Inference service API

To interact with Cloudera AI Inference service API, you need to obtain the domain name of the Cloudera AI Inference service and your Cloudera JSON Web Token (JWT) and save it as environment variables.

About this task



Important: The below commands use the `jq` command, which must be installed on your system.

Procedure

1. Obtain the domain name of the Cloudera AI Inference service and Cloudera JSON Web Token (JWT) from the output of `list-ml-serving-apps` or `describe-ml-serving-app` Cloudera commands.

```
export DOMAIN=$(cdp ml describe-ml-serving-app --app-crn [***APP CRN***] |
jq -r '.app.cluster.domainName')
```

2. Store your CDP TOKEN as an environment variable:

```
export CDP_TOKEN=$(cdp iam generate-workload-auth-token --workload-name DE
| jq -r '.token')
```

Creating a Model Endpoint using API

You can select a specific Cloudera AI Inference service instance and a model version from Cloudera AI Registry to create a new model endpoint.

Procedure

1. Retrieve a registered model's model ID and model version.

This information is available in the Registered Models page in the Cloudera AI control plane UI.

Registered Models / eng-101 / v2 / mistral-7b-embed-onnx

mistral-7b-embed-onnx [Edit Model](#)

MODEL ID 1ev7-1mtz-g9f.bncd	ENVIRONMENT aws-eng-v2	VISIBILITY Private
OWNER cse101	CREATED AT 2024-08-23 01:44:24 AM IST	
DESCRIPTION N/A		

All ▾

Versions (1)

Status ▾	Name ▾	Source ▾	Creator ▾	Creation Date ▾	Actions
Ready	Version 1	NVIDIA	cse101	2024-08-23 01:44:24 AM IST	Share Download

Displaying 1 - 1 of 1 < 1 > 25 / page ▾

- 2. Create the model specification for the selected model.**

```
# cat ./examples/mlflow/model-spec-cml-registry.json
```

The following is the sample output:

```
{
  "namespace": "servicing-default",
```

```

    "name": "mlflow-wine-test-from-registry-onnx",
    "source": {
      "registry_source": {
        "version": 2,
        "model_id": "3azn-tmqe-wsze-5u4s"
      }
    }
  }
}

```

```

export DOMAIN=$(cdp ml describe-ml-serving-app --app-crn [***app-crn***] |
jq -r '.app.cluster.domainName')

```

3. Create the model endpoint by using the following Cloudera AI serving deployEndpoint API:

```

curl -v -H "Content-Type: application/json" -H "Authorization: Bearer ${CDP_TOKEN}" "https://${DOMAIN}/api/v1alpha1/deployEndpoint" -d @./examples/mlflow/model-spec-cml-registry.json

```

The DOMAIN looks like ml-67814ad5-b79.eng-ml-d.xcu2-8y8x.dev.cldr.work.

You can retrieve the CDP_TOKEN by performing the steps from Preparing to interact with the Cloudera AI.



Note:

You can only specify serving-default as the namespace into which the Model Endpoint can be deployed.

Related Information

[Register an ONNX model to AI Registry](#)

[Preparing to interact with the Cloudera AI Inference service API](#)

Listing Model Endpoints using API

Consider the following details for listing Model Endpoints using API.

Procedure

List model endpoints with the help of the following command:

```

curl -H "Content-Type: application/json" -H "Authorization: Bearer ${CDP_TOKEN}" "https://${DOMAIN}/api/v1alpha1/listEndpoints" -d '{
  "namespace": "serving-default"
}'

```

The following is an example response:

```

{ "endpoints": [
  {
    "namespace": "serving-default",
    "name": "mlflow-wine-test-from-registry-onnx",
    "url": "https://[***domain***]/namespaces/serving-default/endpoints/[***endpoint_name***]/v2/models/[***model_name***]/infer",
    "state": "Loaded"
  }
] }%

```

Describing a Model Endpoint using API

Consider the instructions for describing a Model Endpoint using API.

Procedure

Use the following command to describe a given model endpoint. The `describeEndpoint` API shows you all the detailed configuration and runtime snapshot of the endpoint. The output below has been elided for brevity.

```
curl -s -H "Authorization: Bearer ${CDP_TOKEN}" -H "Content-Type: application/json" https://${DOMAIN}/api/v1alpha1/describeEndpoint -d '{"namespace": "serving-default", "name": "[***ENDPOINT_NAME***]"}'|jq
```

You will receive a response similar to the following:

```
{
  "namespace": "serving-default",
  "name": "[***ENDPOINT_NAME***]",
  "url": "https://[***DOMAIN***]/namespaces/serving-default/endpoints/[***ENDPOINT_NAME***]/v2/models/[***MODEL_ID***]/infer",
  "conditions": [
    {
      "type": "IngressReady",
      "status": "True",
      "severity": "",
      "last_transition_time": "1726713090",
      "reason": "",
      "message": ""
    }
  ],
  "observed_generation": 1,
  "replica_count": 1,
  ...,
  "autoscaling": {
    "min_replicas": "1",
    "max_replicas": "80",
    "autoscalingconfig": {
      "metric": "concurrency",
      "target": "20",
    }
  },
  ...,
  "api_standard": "oip",
  "has_chat_template": false,
  "metricFormat": "triton",
  "task": "inference"
}
```

Related Information

[NVIDIA documentation: Inputs and outputs](#)

[Preparing to interact with the Cloudera AI Inference service API](#)

Deleting a Model Endpoint using API

Consider the following instruction to delete a Model Endpoint using API.

Procedure

Delete the Model Endpoint deployed above, using the API.

```
curl -v -H "Content-Type: application/json" -H "Authorization: Bearer ${CDP_TOKEN}" "https://${DOMAIN}/api/v1alpha1/deleteEndpoint" -d '{"namespace": "serving-default", "name": "[***ENDPOINT_NAME***]"}'
```

Autoscaling Model Endpoints using API

You can configure the Model Endpoints deployed on Cloudera AI Inference service to auto-scale to zero instances when there is no load.

Procedure

The following deployment specification shows an example model endpoint that auto-scales between zero and four replicas:

```
# cat ./examples/mlflow/model-spec-cml-registry.json
```

```
{
  "namespace": "serving-default",
  "name": "mlflow-wine-test-from-registry-onnx",
  "source": {
    "registry_source": {
      "version": 1,
      "model_id": "yf0o-hrxq-l0xj-8tk9"
    }
  },
  "autoscaling": {
    "min_replicas": "0",
    "max_replicas": "4"
  }
}
```

Tuning auto-scaling sensitivity using the API

To customize the auto-scaling sensitivity and requisites, set the target and metric fields in the `autoscaling.autoscalingconfig` parameter.

About this task

The metric field is the data you are watching to make auto-scaling decisions, which you can set to either concurrency or to RPS (requests per second):

- Concurrency is the number of requests that each replica of the model shall aim to handle at once.
- RPS is the calculated requests per second handled over the polling period.
- Target is the target value of the metric that you aim to maintain. If the Target value is exceeded when calculating the metric value, new replicas will be spun up or down to maintain the Target value as the upper bound.

Procedure

Set these values as follows:

```
# cat ./examples/mlflow/model-spec-cml-registry.json
```

```
{
  "namespace": "serving-default",
  "name": "mlflow-wine-test-from-registry-onnx",
  "source": {
    "registry_source": {
      "version": 1,
      "model_id": "yf0o-hrxq-l0xj-8tk9"
    }
  },
  "autoscaling": {
    "min_replicas": "0",
    "max_replicas": "4",
    "autoscalingconfig": {
      "metric": "concurrency",
      "target": "25"
    }
  }
}
```

The above example scales between zero and four replicas aiming to maintain at most 25 concurrent requests per replica, scaling the number of replicas deployed to maintain this target.

Running Models on GPU

Follow the guidelines for running Models on GPU.

Procedure

To run a Model Endpoint on GPU, specify the number of GPUs to use per model replica, as follows. You must specify the GPU count as a *string*.

```
# cat ./examples/mlflow/model-spec-cml-registry.json
```

```
{
  "namespace": "serving-default",
  "name": "mlflow-wine-test-from-registry-onnx",
  "source": {
    "registry_source": {
      "version": 1,
      "model_id": "yf0o-hrxq-l0xj-8tk9"
    }
  },
  "resources": {
    "num_gpus": "1"
  }
}
```

The resources field supports specifying the following properties:

- req_cpu - the requested number of CPU cores, which is a string as per Kubernetes standards.
- req_memory - the requested amount of memory, which again follows Kubernetes conventions.
- num_gpus - the requested number of GPUs.

Related Information

[Resource Units in Kubernetes - Kubernetes documentation](#)

[Memory Resource Units - Kubernetes documentation](#)

Deploying models with Canary deployment using API

Cloudera AI Inference service allows users to control traffic percentage to specific model deployments.

Procedure

1. Deploy a model. The traffic value defaults to 100. The following is an example payload:

```
# cat ./examples/mlflow/model-spec-cml-registry.json
```

```
{
  "namespace": "serving-default",
  "name": "mlflow-wine-test-from-registry-onnx",
  "source": {
    "registry_source": {
      "version": 1,
      "model_id": "yf0o-hrxq-l0xj-8tk9"
    }
  }
}
```

2. After deploying a model to the endpoint, direct the traffic to the *Canary* model by updating the model with the desired traffic value set:

```
# cat ./examples/mlflow/model-spec-cml-registry-canary.json
```

```
{
  "namespace": "serving-default",
  "name": "mlflow-wine-test-from-registry-onnx",
  "source": {
    "registry_source": {
      "version": 2,
      "model_id": "yf0o-hrxq-l0xj-8tk9"
    }
  },
  "traffic": "35",
}
```

The percentage of the traffic set in the traffic field diverts to the newly deployed model version and the remaining traffic is directed to the previously deployed model version.

3. Update the model endpoint as follows, when you want your canary model to serve all of your incoming requests:

```
# cat ./examples/mlflow/model-spec-cml-registry-promote.json
```

```
{
  "namespace": "serving-default",
  "name": "mlflow-wine-test-from-registry-onnx",
  "source": {
    "registry_source": {
      "version": 2,
      "model_id": "yf0o-hrxq-l0xj-8tk9"
    }
  },
  "traffic": "100",
}
```



```
}
```

4. Use the following payload to content to the model:

```
curl -H "Content-Type: application/json" -H
"Authorization: Bearer ${CDP_TOKEN}"
https://${DOMAIN}/namespaces/serving-default/endpoints/mlflow-wine-test-f
rom-registry-onnx/v2/models/yf0o-hrxq-l0xj-8tk9/infer
-d @./examples/url/wine-input.json
{"model_name": "yf0o-hrxq-l0xj-8tk9", "model_version": "2", "outputs": [{"name":
"variable", "datatype": "FP32", "shape": [1,1], "data": [0.0]}]}
```

Interacting with Model Endpoints

You can interact with the Cloudera AI Inference service API using an HTTP/REST client, such as cURL.

Making an inference call to a Model Endpoint with an OpenAI API

Language models for text generation are deployed using NVIDIA's NIM microservices. These model endpoints are compliant with the *OpenAI Protocol*. See NVIDIA NIM documentation for supported OpenAI APIs and NVIDIA NIM specific extensions such as Function Calling and Structured Generation.

Related Information

[OpenAI Platform protocol](#)

[NVIDIA documentation: API Reference](#)

Cloudera AI Inference service using OpenAI Python SDK client in a Cloudera AI Workbench Session

Consider the following instructions on interacting with an instruction-tuned large language model endpoint hosted on Cloudera AI Inference service.

Procedure

You can use the following template to interact with an instruction-tuned large language model endpoint hosted on Cloudera AI Inference service:

```
from openai import OpenAI
import json

API_KEY = json.load(open("/tmp/jwt"))["access_token"]
MODEL_NAME = "[***MODEL_NAME***]"

client = OpenAI(
    base_url = "[***BASE_URL***]",
    api_key = API_KEY,
)

completion = client.chat.completions.create(
    model=MODEL_NAME,
    messages=[{"role": "user", "content": "Write a one-sentence definition of Ge
nAI."}],
    temperature=0.2,
    top_p=0.7,
    max_tokens=1024,
    stream=True
)
for chunk in completion:
```

```
if chunk.choices[0].delta.content is not None:
    print(chunk.choices[0].delta.content, end=" ")
```

Where `base_url` is the model endpoint URL up to the API version `v1`. To get the `base_url`, copy the model endpoint URL and delete the last two path components.

The screenshot shows the Cloudera AI interface. On the left is a sidebar with navigation options: AI HUB, Model Hub, DEPLOYMENTS, Model Endpoints (highlighted), Registered Models, ADMINISTRATION, AI Workbenches, AI Inference Services, AI Registries, and AI Workbench Backups. The main panel is titled 'llama-31-8b-instruct-a10gx2 / Model Endpoint Details'. It displays the following information:

- Model Name:** llama-31-8b-instruct-a10gx2
- URL:** https://ml-97b60e19-...-br-nqv... (highlighted with an orange box)
- STATUS:** Running (with a green checkmark icon)
- INFERENCE SERVICE:** serving-single-gpu-aws
- ENVIRONMENT:** aws, eng-r, -aws-v2
- MODEL ID:** b5fh-bnap-ymm6-3vy7 (highlighted with an orange box)
- DESCRIPTION:** N/A
- Configuration Details:**

MODEL NAME	VERSION	TRAFFIC ALLOCATION
llama-31-8b-instruct-a10gx2	2	100
CPU & MEMORY	GPU	AUTO-SCALING RANGE
40 vCPU & 160Gi Memory	4 GPUs	1 - 3

Copy the model endpoint URL from the **Model Endpoint Details** UI and modify it to

```
https://[***DOMAIN***]/namespaces/serving-default/endpoints/[***ENDPOINT_NAME***]/v1
```

`MODEL_NAME` is the model name assigned to the model when it is registered to the Cloudera AI Registry. You can find this in the **Model Endpoint Details** UI.

Cloudera AI Inference service using OpenAI Python SDK client on a local machine

Consider the following guidelines for Cloudera AI Inference service using OpenAI Python SDK client.

Procedure

Specify the `CDP_TOKEN` as the `API_KEY`. A common way is to export `CDP_TOKEN` as an environment variable and access that from your Python code:

```
from openai import OpenAI
import os
API_KEY = os.environ['CDP_TOKEN']
MODEL_NAME = [***MODEL_NAME***]

client = OpenAI(
    base_url = "[***BASE_URL***]",
    api_key = API_KEY,
)

completion = client.chat.completions.create(
    model=MODEL_NAME,
    messages=[{"role": "user", "content": "Write a one-sentence definition of Generative AI."}],
    temperature=0.2,
    top_p=0.7,
    max_tokens=1024,
    stream=True
```

```
)
for chunk in completion:
    if chunk.choices[0].delta.content is not None:
        print(chunk.choices[0].delta.content, end="")
```

Related Information

[Preparing to interact with the Cloudera AI Inference service API](#)

[NVIDIA documentation: API Reference](#)

[Authentication of Cloudera AI Inference service](#)

OpenAI Inference Protocol Using Curl

Consider this example for OpenAI Inference Protocol Using Curl.

An example inference payload for the OpenAI Protocol:

```
# cat ./llama-input.json
```

```
{
  "messages": [
    {
      "content": "You are a polite and respectful chatbot helping people plan a vacation.",
      "role": "system"
    },
    {
      "content": "What should I do for a 4 day vacation in Spain?",
      "role": "user"
    }
  ],
  "model": "meta/llama-3_1-8b-instruct",
  "max_tokens": 200,
  "top_p": 1,
  "n": 1,
  "stream": false,
  "stop": "\n",
  "frequency_penalty": 0.0
}
```

```
curl -H "Content-Type: application/json" -H "Authorization: Bearer ${CDP_TOKEN}" "https://${DOMAIN}/namespaces/serving-default/endpoints/llama-3-1/v1/chat/completions" -d @./llama-input.json
```

You will receive response similar to the following:

```
Spain offers a diverse range of experiences, making it perfect for a 4-day vacation...
```

Making an inference call to a Model Endpoint with Open Inference Protocol

Cloudera AI Inference service serves predictive ONNX models using the NVIDIA Triton Server. The deployed model endpoints are compliant with Open Inference Protocol version 2.



Note: The model inference request and response payloads are different from models deployed on a Cloudera AI Workbench.

Related Information

[Open Inference Protocol Specification](#)

NVIDIA Triton Inference Server

Open Inference Protocol Using Python SDK

Use the following code sample to interact with a model endpoint using the Open Inference Protocol.

Procedure

Run the following script after customizing the ENDPOINT_NAME, DOMAIN, and MODEL_NAME placeholders. The inference request payload shape must match what your model is expecting, which you can determine with the read_model_metadata() function.

```
#!/pip install open-inference-openapi

from open_inference.openapi.client import OpenInferenceClient, InferenceRequest
import httpx
import json
import os

#
# inspired by https://pypi.org/project/open-inference-openapi/
#

CDP_TOKEN = os.environ['CDP_TOKEN']
BASE_URL = 'https://[***DOMAIN***/namespaces/serving-default/endpoints/[***ENDPOINT_NAME***]]'
MODEL_NAME = '[***MODEL-NAME***]'
headers = {'Authorization': 'Bearer ' + CDP_TOKEN,
           'Content-Type': 'application/json'}

httpx_client = httpx.Client(headers=headers)
client = OpenInferenceClient(base_url=BASE_URL, httpx_client=httpx_client)

# Check that the server is live, and it has the model loaded
client.check_server_readiness()
metadata = client.read_model_metadata(MODEL_NAME)
metadata_str = json.dumps(json.loads(metadata.json()), indent=2)
print(metadata_str)

# Make an inference request
pred = client.model_infer(
    MODEL_NAME,
    request=InferenceRequest(
        inputs=[
            {
                "name": "float_input",
                "shape": [1, 11],
                "datatype": "FP32",
                "data": [7.4] * 11 # This is a shorter way to write the same data
            }
        ]
    ),
)
json_resp_str = json.dumps(json.loads(pred.json()), indent=2)
print(json_resp_str)
```

Open Inference Protocol Using Curl

Consider the following instructions for using Open Inference Protocol Using Curl.

Procedure

You can construct the input of the inference call based on the Open Inference Protocol V2:

```
# cat ./examples/wine-input.json
```

Your input payload will look something like the following:

```
{
  "parameters": {
    "content_type": "pd"
  },
  "inputs": [
    {
      "name": "float_input",
      "shape": [
        1,
        11
      ],
      "datatype": "FP32",
      "data": [
        7.4,
        7.4,
        7.4,
        7.4,
        7.4,
        7.4,
        7.4,
        7.4,
        7.4,
        7.4,
        7.4
      ]
    }
  ]
}
```

```
curl -H "Content-Type: application/json" -H "Authorization: Bearer ${CDP_TOKEN}" "https://${DOMAIN}/namespaces/serving-default/endpoints/[***ENDPOINT_NAME***]/v2/models/model/infer" -d @./examples/wine-input.json
```

You will receive response similar to the following:

```
{
  "model_name": "model",
  "model_version": "1",
  "outputs": [
    {
      "name": "variable",
      "datatype": "FP32",
      "shape": [1, 1],
      "data": [5.535987377166748]
    }
  ]
}
```

Deploying Predictive Models

The following example illustrates how to train a model on a Cloudera AI workbench, register it, and then deploy it to Cloudera AI Inference.

About this task

In this section, the well-known wine classifier example is taken using the dataset from [UC Irvine](#).

Procedure

1. Create a project in your Cloudera AI Workbench, open a session, and execute the following command to install the necessary Python packages: Upload the model artifact to your project's file system.

```
pip install open-inference-openapi onnx==1.13.1 onnxruntime skl2onnx pandas
scikit-learn
```

2. Copy and paste the following Python code into your editor of choice in the workbench session:

```
import logging
import sys
import warnings
import onnx
from urllib.parse import urlparse
import numpy as np
import pandas as pd
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn import pipeline
from sklearn import linear_model
from urllib.parse import urlparse
from sklearn.preprocessing import FunctionTransformer
import mlflow
import mlflow.sklearn
from mlflow.models import infer_signature
from skl2onnx import to_onnx
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

def convert_to_onnx(model, data):
    #check the specifications
    initial_types = list(
        zip(
            data.columns.values.tolist(),
            [FloatTensorType([None, 1]) for _ in range(len(data.columns))],
        )
    )
    onnx_model = convert_sklearn(model=model, initial_types=initial_types)
    print("onnx_model.type:", type(onnx_model))
    mlflow.set_tag("onnx_version", onnx.__version__)
    return onnx_model

def eval_metrics(actual, pred):
    rmse = np.sqrt(mean_squared_error(actual, pred))
    mae = mean_absolute_error(actual, pred)
    r2 = r2_score(actual, pred)
    return rmse, mae, r2
warnings.filterwarnings("ignore")
```

```

np.random.seed(40)

csv_url = (
    "http://archive.ics.uci.edu/ml"
    "/machine-learning-databases/wine-quality/winequality-red.csv"
)
try:
    data = pd.read_csv(csv_url, sep=";")
except Exception as e:
    logger.exception(
        "Unable to download training & test CSV, check your internet conn
ection. Error: %s", e
    )
# Split the data into training and test sets. (0.75, 0.25) split.
train, test = train_test_split(data)
# The predicted column is "quality" which is a scalar from [3, 9]
train_x = train.drop(["quality"], axis=1)
test_x = test.drop(["quality"], axis=1)
train_y = train[["quality"]]
test_y = test[["quality"]]

alpha = 0.5
l1_ratio = 0.5

with mlflow.start_run():
    lr = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, random_state=42)
    DummyScaler = FunctionTransformer(None)
    lr = pipeline.Pipeline(
        [("dummy", DummyScaler), ("reg", linear_model.ElasticNet(alpha=alpha,
                                                                    l1_ratio=l1_r
atio,
                                                                    random_state=
42)))]
    )
    lr.fit(train_x, train_y)
    predicted_qualities = lr.predict(test_x)
    (rmse, mae, r2) = eval_metrics(test_y, predicted_qualities)
    print(f"Elasticnet model (alpha={alpha:f}, l1_ratio={l1_ratio:f}):")
    print(f"  RMSE: {rmse}")
    print(f"  MAE: {mae}")
    print(f"  R2: {r2}")
    mlflow.log_param("alpha", alpha)
    mlflow.log_param("l1_ratio", l1_ratio)
    mlflow.log_metric("rmse", rmse)
    mlflow.log_metric("r2", r2)
    mlflow.log_metric("mae", mae)
    predictions = lr.predict(train_x)
    model_signature = infer_signature(train_x, predictions)
    onnx_model = convert_to_onnx(lr, test_x)
    mlflow.onnx.log_model(onnx_model, "model",
                          registered_model_name="ElasticnetWineModel",
                          signature=model_signature)

```

3. Run the above code. If it completes successfully, you should see the ElasticnetWineModel model listed on the **Registered Models** page on the **Cloudera AI** control plane UI.

- Click on the model name to view details about the model, and deploy it to Cloudera AI Inference service service:

Registered Models / eng-m...-p... aws / ElasticnetWineModel

ElasticnetWineModel [Edit Model](#)

MODEL ID h4x5m1b5-uy56-74el	ENVIRONMENT aws eng-m...-p... aws	VISIBILITY Private
OWNER z...	CREATED AT 2024-11-20 10:55:32 AM EST	
DESCRIPTION N/A		

Version 2 [Latest](#) [Deploy](#)

STATUS Ready	VERSION ID h4x5m1b5-uy56-74el	SOURCE MLFlow	CREATED BY z...
CREATED DATE 2024-11-20 01:31:00 PM EST	VERSION NOTES N/A		

[Metrics](#) [Parameters](#) [Tags](#)

Name	Value
rmse	0.793164022927685
r2	0.10862644997792636
mae	0.6271946374319586

- Click Deploy. The model endpoint creation dialogbox is displayed.
- Select the Cloudera AI Inference service cluster you wish to deploy it to, and click Deploy.
The **Create Endpoint** page is displayed.
- Create the model endpoint using UI or API.

Using UI

You can use the **Create Model Endpoint** page to select a specific Cloudera AI Inference service instance and a model version from Cloudera AI Registry to create a new model endpoint. For more information, see [Creating a Model Endpoint using UI](#)

Using API

```
$ cat deploy_wine_onnx.json
```

```
{
  "namespace": "serving-default",
  "name": "elasticnetwine",
  "source": {
    "registry_source": {
      "model_id": "<MODEL_ID>",
      "version": "<VERSION>"
    }
  },
  "resources": {
    "req_cpu": "2",
    "req_memory": "2Gi"
  },
  "instance_type": "m5.24xlarge",
  "autoscaling": {
    "min_replicas": "1",
    "max_replicas": "80",
    "autoscalingconfig": {
      "metric": "concurrency",
      "target": "100"
    }
  }
}
```



```
$ curl -XPOST -H "Authorization: Bearer ${CDP_TOKEN}" https://${DOMAIN}/api/v1alpha1/deployEndpoint -d @./deploy_wine_onnx.json
```

8. When the model endpoint is in the Running state, you can interact with the endpoint. The following is an example of a Python client application running on Cloudera AI Workbench:

```
$ cat examples/new-wine-input.json
```

```
{
  "parameters": {
    "content_type": "pd"
  },
  "inputs": [
    {
      "name": "fixed_acidity",
      "shape": [1, 1],
      "datatype": "FP32",
      "data": [9.4]
    },
    {
      "name": "volatile_acidity",
      "shape": [1, 1],
      "datatype": "FP32",
      "data": [0.8000]
    },
    {
      "name": "citric_acid",
      "shape": [1, 1],
      "datatype": "FP32",
      "data": [0]
    },
    {
      "name": "residual_sugar",
      "shape": [1, 1],
      "datatype": "FP32",
      "data": [1.9]
    },
    {
      "name": "chlorides",
      "shape": [1, 1],
      "datatype": "FP32",
      "data": [0.076]
    },
    {
      "name": "free_sulfur_dioxide",
      "shape": [1, 1],
      "datatype": "FP32",
      "data": [11]
    },
    {
      "name": "total_sulfur_dioxide",
      "shape": [1, 1],
      "datatype": "FP32",
      "data": [34]
    },
    {
      "name": "density",
      "shape": [1, 1],
      "datatype": "FP32",
      "data": [0.9978]
    },
    {
      "name": "pH",
```

```

        "shape": [1, 1],
        "datatype": "FP32",
        "data": [3.51]
    },
    {
        "name": "sulphates",
        "shape": [1, 1],
        "datatype": "FP32",
        "data": [0.56]
    },
    {
        "name": "alcohol",
        "shape": [1, 1],
        "datatype": "FP32",
        "data": [9.4]
    }
]
}

```

9. You can use the above inference input payload in your Python client code as in the following example:

```

from open_inference.openapi.client import OpenInferenceClient, Inference
Request
import httpx
import requests
import json

CDP_TOKEN = json.load(open("/tmp/jwt"))["access_token"]
BASE_URL = '<ENDPOINT_BASE_URL>'
MODEL_NAME = '<MODEL_ID>'
headers = {'Authorization': 'Bearer ' + CDP_TOKEN,
           'Content-Type': 'application/json'}

httpx_client = httpx.Client(headers=headers)
client = OpenInferenceClient(base_url=BASE_URL, httpx_client=httpx_client)
# Check that the server is live, and it has the model loaded
client.check_server_readiness()
metadata = client.read_model_metadata(MODEL_NAME)
metadata_str = json.dumps(json.loads(metadata.json()), indent=2)
# Uncomment the next line to see model server metadata, which
# includes model name, version, deep learning platform type, and
# the shape of the input/output tensors supported by the model server.
#print(metadata_str)
# Read the input payload
payload = json.load(open("new-wine-input.json"))["inputs"]
# Make an inference request
pred = client.model_infer(
    MODEL_NAME,
    request=InferenceRequest(
        inputs=payload
    ),
)

json_resp_str = json.dumps(json.loads(pred.json()), indent=2)
print(json_resp_str)

# Output
#{
#   "model_name": "h2hy-o8t4-jg0p-qvnx",
#   "model_version": "1",
#   "outputs": [
#     {
#       "name": "variable",

```

```
#      "shape": [
#        1,
#        1
#      ],
#      "datatype": "FP32",
#      "data": [
#        5.535987377166748
#      ]
#    }
#  ]
#}
```

Related Information

[ONNX models](#)

Accessing Cloudera AI Inference service Metrics

Cloudera AI Inference service exposes Prometheus metrics for the deployed Model Endpoints. The UI displays plots of a few chosen metrics for each model endpoint.

For additional metrics, the Prometheus server can be accessed directly at `https://{DOMAIN_NAME}/prometheus` by passing the authorization bearer token.

Predictive models deployed with the NVIDIA Triton Inference Server and NVIDIA NIM embedding models export metrics prefixed with `nv_`. Refer to the *NVIDIA NIM documentation* for metrics exported by the NVIDIA NIM Large Language Model (LLM) runtimes for text-generation.

Related Information

[Authentication of Cloudera AI Inference service](#)

[NVIDIA NIM Large Language Models, Observability](#)