Machine Learning Architecture Overview

Date published: 2020-07-16
Date modified: 2024-04-01

Legal Notice

Contents

Architecture Overview

Cloudera Machine Learning (CML) is a Data Service on the CDP platform. Residing on the CDP platform helps CML integrate with and leverage other Data Services residing on the platform, for example Data Engineering and Data Warehouse.

Provisioning

CML utilizes the CDP Control Plane to manage Data Services so you can provision and delete CML Workspaces. CDP Control Plane leverages cloud native capabilities to dynamically access CPU, memory, and GPU resources along with cloud-managed Kubernetes (K8s) to provide the infrastructure.

During provisioning the Machine Learning application is configured to authenticate end users of the service (ML) via the CDP identity provider, which is “chained” back to the customer’s identity provider. As a result, Machine Learning provisioned instances allow for seamless customer SSO.

When you provision a CML workspace, the following happens:

- CDP Control Plane performs the following in the cloud environment:
  - Requests a TLS Certificate and domain name with the cloudera.site domain
  - Identifies the SSO configuration
  - Identifies the SDX configuration for the environment
- CDP Control Plane provisions a managed Kubernetes cluster
- CDP Control Plane installs CML into the Kubernetes environment
- Storage is mounted directly via managed service providers

CML uses the cloud provider load balancer and networking infrastructure to partition the resources. CML also leverages the cloud provider infrastructure to enable the customer to specify autoscaling.

CML provisions the DNS and the certificate. CML renews the certificates for the customer on an ongoing basis.

CML Architecture

Once a CML workspace is provisioned, you can start using Cloudera Machine Learning (CML) for your end-to-end Machine Learning workflow.

CML is a three-tier application that consists of a presentation tier, an application tier and a data tier.

Web tier

CML is a web application that provides a UI that simplifies the action of managing workloads and resources for data scientists. It offers users a convenient way to deploy and scale their analytical pipeline and collaborate with their colleagues in a secure compartmentalized environment.

CML communicates using HTTPS, Websocket, and gRPC. External communication is limited to HTTP and Websocket for the web UI and APIs. In-cluster service-to-service communication uses gRPC and is encrypted and mutually authenticated using TLS (Transport Layer Security).

Application tier

The application tier uses an actual set of workloads that users are running. These workloads are isolated in Kubernetes namespaces and run on specially marked compute nodes. Kubernetes/node level auto scaling is used to expand/contract the cluster size based on user demand.

User code gets baked into Docker images via a set of Source-to-Image pods (S2I), which includes a managing server, a queue for builds, a registry that serves the images for Docker, a git server, and the builders that actually perform the image building. Traditionally these images used the host machine's docker, but CML switched to in-container builds for security and compliance on some platforms.

Data tier

CML uses an internal Postgres database for persisting metadata of user workloads such as Sessions, Jobs, Models and Applications, which runs as a pod and is backed by a persistent volume, using the cloud-provider's block storage offering (for example, EBS for AWS and Premium_LRS for Azure).

CML uses an NFS server, i.e. a POSIX-compliant file system, for storing users' project files, which include user code, libraries installed into a Session, and small data files. For AWS, CML creates an Elastic File System (EFS) file system when provisioning the workspace for storing project files. For Azure, users can either provide an NFS volume created in Azure NetApp Files or an external NFS server when provisioning CML workspaces for storing project files. This NFS server is supplied to the CML workspaces as Kubernetes Persistent Volumes (PVs). Persistent Volume Claims (PVCs) are set up per-namespace, and each user gets their own namespace - thus each user's view of the NFS server is limited to that exposed by the PVC.

ML Runtimes

ML Runtimes are responsible for running data science workloads and intermediating access to the underlying cluster.

CML allows you to run any code via an interactive session, scheduled job, or deployed model or application. Data Scientists can use interactive sessions to explore data, or develop a model. They can create jobs and schedule them to run at specified times or productionize their work as a model to provide a REST endpoint or as an application that offers an interactive data dashboard for business users. All of these workloads run inside an ML Runtime container on top of Kubernetes.

Cloudera ML Runtimes are purpose built to serve a specific use-case. They are available with a single editor (for example, Workbench, Jupyterlab), ship a single language Kernel (for example, Python 3.8 or R 4.0), and have a set of UNIX tools and utilities or language libraries and packages.

ML Runtimes have been open sourced and are available in the cloudera/ml-runtimes GitHub repository. If you need to understand your Runtime environments fully or want to build a new Runtime from scratch, you can access the Dockerfiles that were used to build the ML Runtime container images in this repository.

There is a wide range of supported Runtimes out-of-the-box that cover the large majority of Data Science use-cases, but any special requirements can be satisfied by building a custom ML Runtime container image.

CML also supports quota management for CPU, GPU, and memory to limit the amount of resources users have access to within the CML workspace.

Before ML Runtimes, CML offered similar functionalities via Legacy Engines. These deprecated container images followed a monolithic architecture, all kernels (Python, R, Scala), and all seemingly useful packages and libraries were included in the image.

Spark on Kubernetes

In Cloudera Machine Learning (CML), multiple Spark versions are available through Runtime Addons. Data Scientists can select the version of Spark to be configured for any workload. CML configures the Runtime container and mounts all of the dependencies.

CML supports fully-containerized execution of Spark workloads through Spark's support for the Kubernetes cluster backend. Users can interact with Spark both interactively and in batch mode. In both batch and interactive modes, dependency management, including for Spark executors, is transparently managed by CML and Kubernetes. No extra configuration is required. In interactive mode, CML leverages the cloud provider for scalable project storage, and in batch mode, CML manages dependencies through container images.

CML also supports native cloud autoscaling via Kubernetes. When clusters do not have the required capacity to run workloads, they can automatically scale up additional nodes. Administrators can configure auto-scaling upper limits, which determine how large a compute cluster can grow. Since compute costs increase as cluster size increases, having a way to configure upper limits gives administrators a method to stay within a budget. Autoscaling policies can also account for heterogeneous node types such as GPU nodes.

In CML, each project is owned by a user or team. Users can launch multiple sessions in a project. Workloads are launched within a separate Kubernetes namespace for each user, thus ensuring isolation between users at the Kubernetes level.

Autoscaling Workloads with Kubernetes

Kubernetes dynamically resizes clusters by using the `Kubernetes Cluster Autoscaler` (on Amazon EKS) or `cluster-autoscaler` (on Azure). The cluster autoscaler changes the desired capacity of an autoscaling group to expand or contract a cluster based on pod resource requests.

Scaling Up

The primary trigger for scaling up (or expanding) an autoscaling group is failure by the Kubernetes pod scheduler to find a node that meets the pod's resource requirements. In Cloudera Machine Learning (CML), if the scheduler cannot find a node to schedule an engine pod because of insufficient CPU or memory, the engine pod will be in "pending" state. When the autoscaler notices this situation, it will change the desired capacity of the autoscaling group (CPU or GPU) to provision a new node in the cluster. As soon as the new node is ready, the scheduler will place the session or engine pod there. In addition to the engine pod, certain CML daemonset pods will also be scheduled on the new node.

The time taken to schedule an engine pod on a new node depends on the amount of time the autoscaler takes to add a new node into the cluster, plus time taken to pull the engine's Docker image to the new node.

Scaling Down

A cluster is scaled down by the autoscaler by removing a node, when the resource utilization on the given node is less than a pre-defined threshold, provided the node does not have any non-evictable pods running on it. This threshold is currently set to 20% CPU utilization. That is, a node is removed if the following criteria are met:

- The node does not have non-evictable pods
- The node's CPU utilization is less than 20%
- The number of active nodes in the autoscaling group is more than the configured minimum capacity

It is possible that certain pods might be moved from the evicted node to some other node during the down-scaling process.

Note: By default, on AWS and Azure, autoscaling groups can include a maximum of 30 nodes. If more nodes are needed, contact your Cloudera representative.

Limitations on Azure

On Azure, there are some specific limitations to how autoscaling works.

- CPU nodes cannot scale down to zero. You can only have one or more CPU nodes.
- Autoscaling down is sometimes blocked by Azure services. You can check the cluster autoscaler logs to see if this is occurring.

Autoscale Groups

A Cloudera Machine Learning (CML) workspace or cluster consists of three different autoscaling groups: "infra", "cpu" and "gpu". These groups scale independently of one another.

Infra Autoscaling Group

The Infra autoscaling group is created automatically when a user provisions a CML cluster, and is not configurable from the UI. This group is meant to run the core CML services that are critical to the overall functioning of the workspace. This group is loosely analogous to the master node of legacy CDSW, however it can scale up or down if necessary. The instance count for this group ranges from 1 to 3, with the default set to 2. The instance type used for the group is m5.2xlarge on AWS, and Standard DS4 v2 on Azure. On AWS, this group is restricted to a single availability zone (AZ) because it relies on Elastic Block Store (EBS).

CPU Autoscaling Group

The CPU autoscaling group forms the main worker nodes of a CML cluster, and is somewhat configurable from the UI. The user can choose from three different instance types, and can also set the autoscaling range from 0 to 30 CPU worker nodes. This group is meant to run general CPU-only workloads. This group is available in multiple availability zones.

GPU Autoscaling Group (not available on Azure)

The GPU autoscaling group consists of instances that have GPUs, and are meant for workloads that require GPU processing. Like the CPU group, this group is configurable from the UI. Unlike the CPU group, this group is meant exclusively for sessions that request > 0 GPUs, and are therefore fenced off from CPU-only workloads, in part because GPU instance types are much more expensive than regular instance types. This group is available in multiple availability zones.

Critical and Non-critical Pods

The pods running various Cloudera Machine Learning (CML) services and jobs broadly fall into critical and non-critical types.

Critical pods are protected from preemption by autoscaling to avoid interrupting important services. Most of the pods running in the “infra” autoscaling group are critical. Pods that run user sessions, such as engine pods and Spark executor pods, are also considered critical, and are marked as not safe to evict. CML system services that are deployed as daemonsets (they run on all nodes in the cluster) are deemed important, but not critical. These pods are marked as “safe-to-evict” by autoscaling.