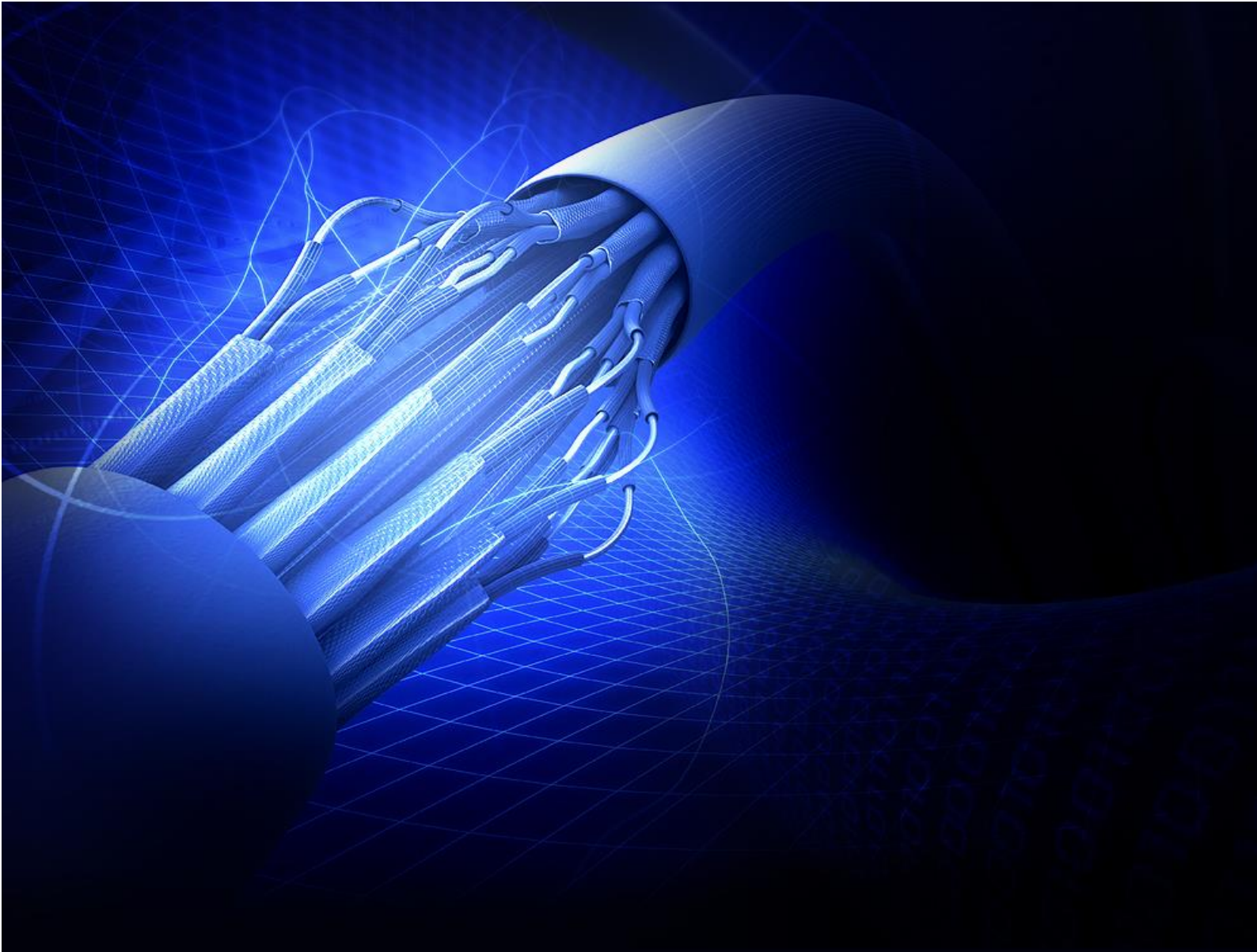


Project Report
Implementing Runtime Security Monitoring for AWS Container Clusters

Student Name: David Williams
Project Supervisor: Hisain Elshaafi
Student ID: C00263768

Course: Cybersecurity & IT Security
4th Year

Abstract

In this comprehensive report, the journey through the fourth-year project is encapsulated, highlighting the challenges and triumphs encountered in navigating the complexities of Amazon Web Services (AWS) and Elastic Kubernetes Services (EKS). The report meticulously details the encountered obstacles, ranging from initial struggles with Falco installation to AWS billing discrepancies, and the subsequent solutions devised to address them. Through a candid exploration of both successes and setbacks, the report provides invaluable insights into the experiential learning process, emphasizing the depth of understanding gained through hands-on engagement with technologies such as Falco, AWS, EKS, Lambda, Firelens, and Kubernetes.

Throughout the project, the report reflects on the myriad lessons learned and outlines potential avenues for improvement, addressing technical intricacies, deviations from the initial design, and supplementary research endeavours. Despite encountering difficulties in implementing the project, particularly in establishing connectivity between AWS and the VM Ubuntu environment, significant progress was made, notably in achieving Falco functionality across both ends of the project. The report concludes with reflections on the project's outcomes, underscoring the importance of a focused approach to future endeavours, prioritizing foundational understanding of AWS services before delving into advanced applications like Falco, and emphasizing the significance of comprehensive testing methodologies to ensure the reliability of software products in complex cloud environments.

Contents

Introduction: Elevating Expertise Through Experience

As the project draws to a close, this report encapsulates the myriad challenges encountered and navigated over the course of the fourth year. Delving into the complexities of Amazon Web Services (AWS) and Elastic Kubernetes Services (EKS), I reflect on the journey marked by both obstacles and triumphs.

While the envisioned outcomes may not have been fully realized, the invaluable lessons gleaned from engaging with technologies such as Falco, AWS, EKS, Lambda, Firelens, and Kubernetes stand as a testament to the depth of experiential learning. Notably, the absence of Calico, a technology that could have bolstered project security, remains a point of interest.

This report is structured to dissect the encountered challenges, accompanied by visual depictions of errors encountered—a testament to the richness of the learning journey. I will delineate the achieved milestones alongside a candid exploration of unmet objectives, offering insights into both successes and setbacks. Moreover, a comprehensive exploration of lessons learned and potential avenues for improvement will be addressed.

Beyond these focal points, the report will delve into technical intricacies, deviations from initial design, supplementary research endeavours, implementation hurdles, software tooling issues, and the testing methodologies employed to achieve project reliability.

In essence, this report encapsulates not only the technical aspects of the project but also the experiential growth and insights gained throughout its execution.

Criteria

General Issues

• Problems encountered and how they were resolved. Be as specific as possible here

• What you achieved

• What you did not achieve

• What you learned

• What you would do differently if starting again

Technical Issues

• Report of any differences from your earlier design, and any additional research that was required

• Difficulties in implementing your project

• Issues with tools or software used in the project development

• Testing used to assess reliability of your software product.

Problems encountered and resolved

Starting off the project, I first initiated my virtual machine using VMware to run an Ubuntu Linux machine where I would run Falco for the first time. The purpose of this is to give Falco kernel level intrusion detection to the entire environment, from the kernel level up to the Cloud container level. This provides the highest level of security, which was the goal of the project.

The first issue was installing Falco correctly:

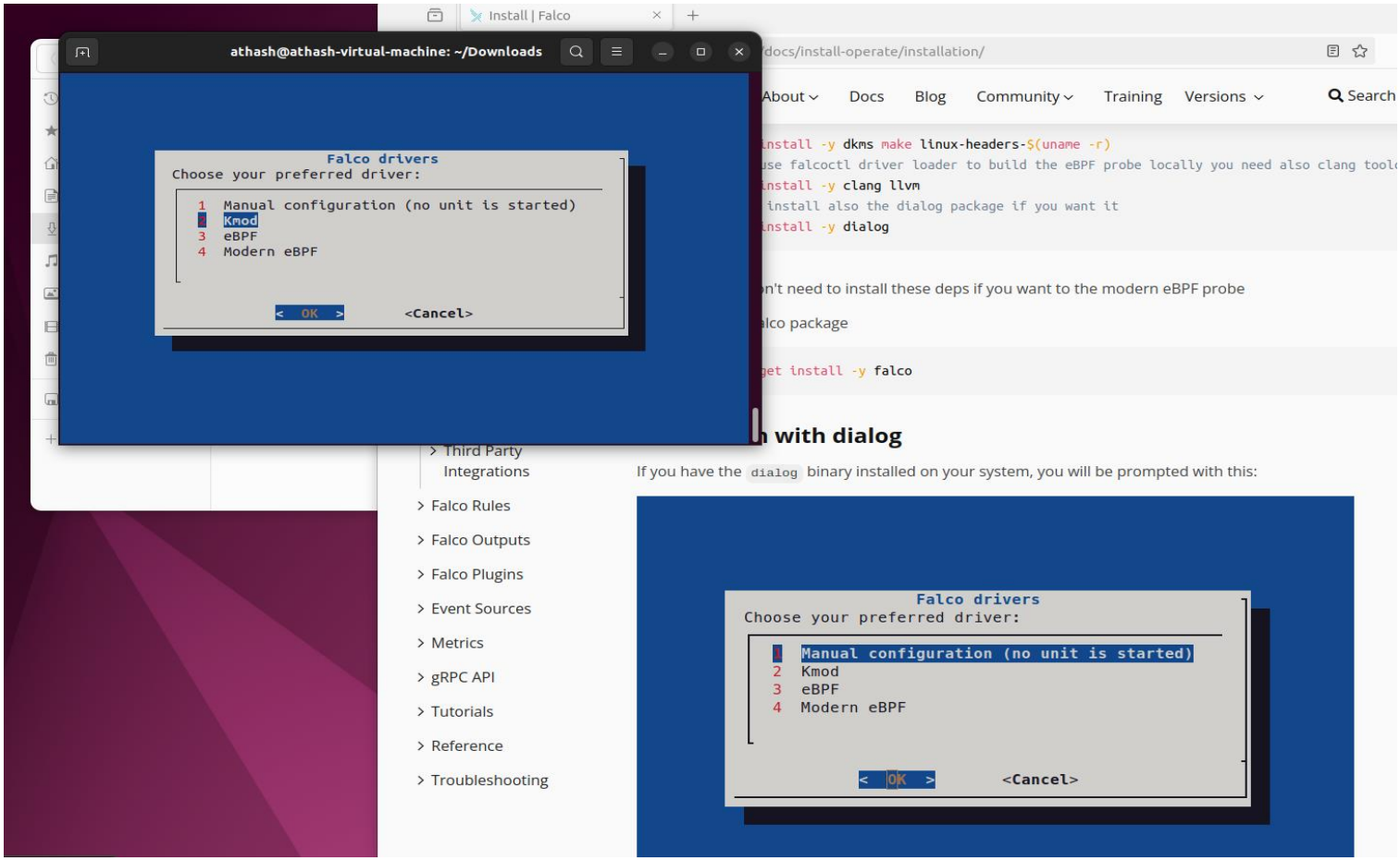


Figure 1: Installing Falco for the first time.

When I installed Falco for the first-time. I followed the documentation as can be seen on the right side of the picture above. Although the documentation isn't clear on which option to take in this installation screen, I followed it and installed kmod which was the wrong option to take.

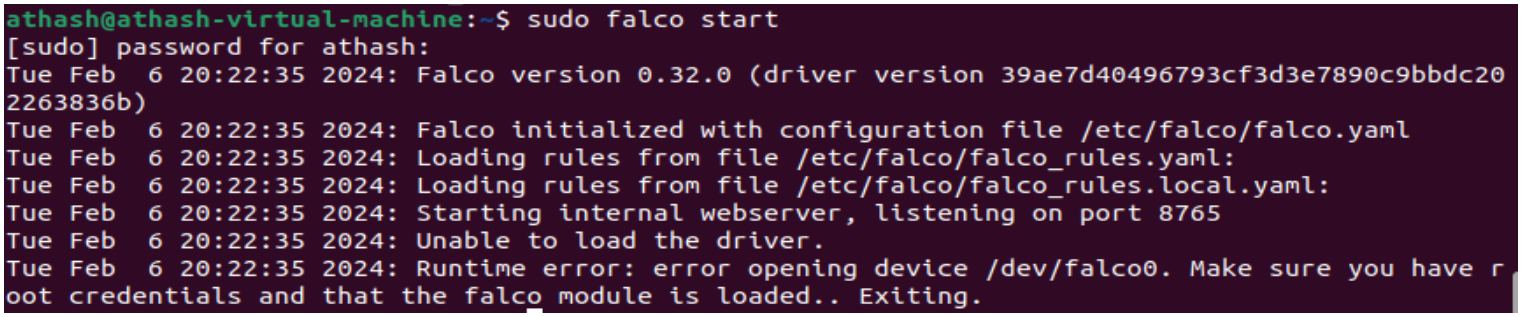


Figure 2: Falco errors.

This screenshot shows Falco running into errors and exiting after running due to using the wrong version (kmod) which I didn't understand at the time.

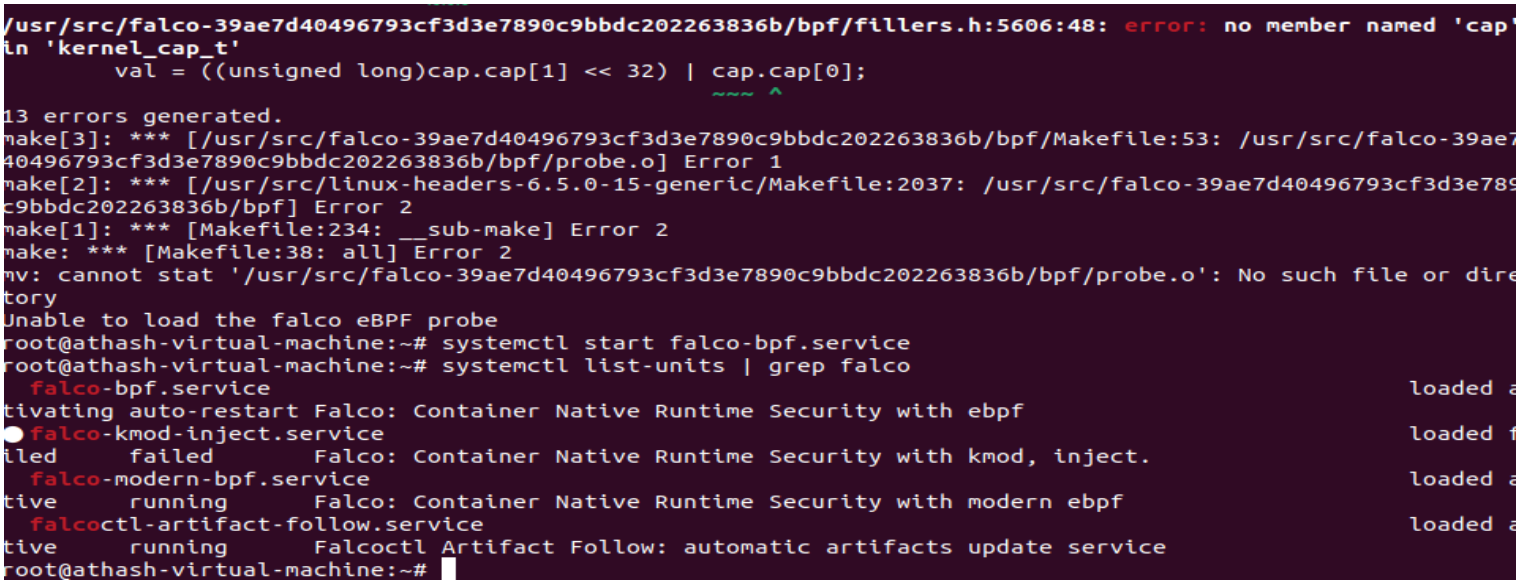


Figure 3: Falco failing to run.

Despite using the command to run Falco, installed through kmod. As can be seen by figure 3, there are errors in the installation.

After days of much trial and error, re-installing and reading documentation, looking for videos on YouTube which there were none and even using the official Falco tutorial guide on installing Falco. I decided to reach out to the Slack community where the people who are involved in developing Falco communicate through.

The Solution was to use Modern eBPF from figure 1, and running this command to get Falco started: sudo falco -o engine.kind=modern_ebpf

Then I would run this command: sudo cat /etc/shadow
to generate events which looked like this.

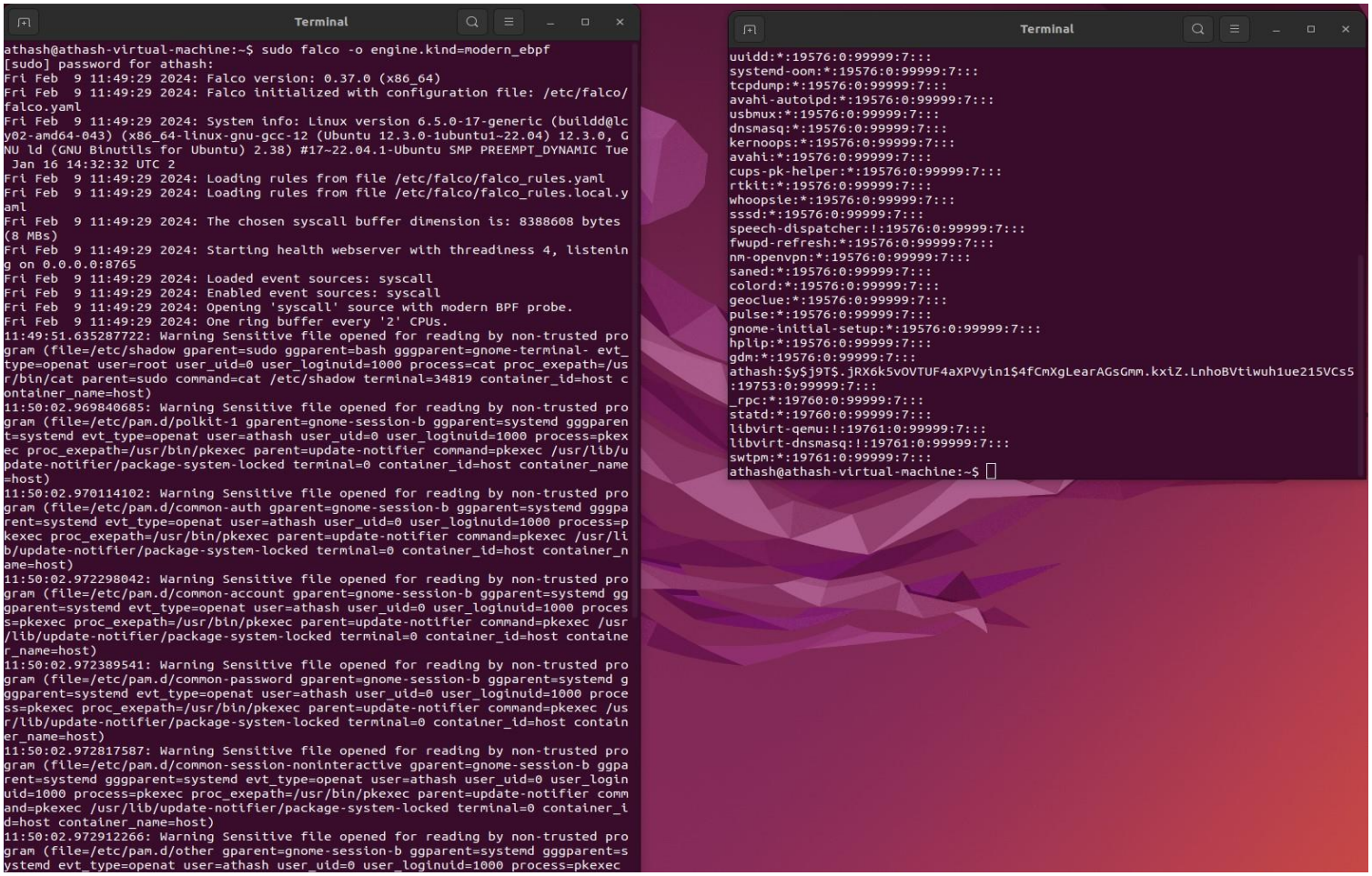


Figure 4: Falco working.

Although the events are not very readable, they still show that someone is trying to edit a critical file within the Linux Ubuntu system. I took the time to configure the falco.yaml file and write a script to make the events more readable. in figure 5 you can see the newly structure of the events through the script I had written for the yaml file.

```
17:11:33.850260472: Warning Sensitive file opened for reading by non-trusted program (fil
e=/etc/shadow gparent=sudo ggparent=bash gggparent=gnome-terminal- evt_type=openat user=r
oot user_uid=0 user_loginuid=1000 process=cat proc_exepath=/usr/bin/cat parent=sudo comma
nd=cat /etc/shadow terminal=34823 container_id=host container_name=host)
time = "Apr  1 17:11:33 athash-virtual-machine falco: 17:11:33"
event number = "850260472"
event warning = "Warning Sensitive file opened for reading by non-trusted program"
UserID = "0"
gparent, Parent process = "sudo
bash
gnome-terminal-"
ggparent, Grandparent process = "bash
gnome-terminal-"
gggparent, Great-grandparent process = "gnome-terminal-"
event type = "openat"
user = "root"
user_uid = "0"
user_loginuid = "1000"
process = "cat"
process_exepath = "/usr/bin/cat"
parent = "sudo
bash
gnome-terminal-
sudo"
command = "cat"
terminal = "34823"
container_id = "host"
container_name = "host"
```

Figure 5: Falco events new structure.

The next issue I had started back in December and was noticed in January but continued into February, where I was still being charged money for services that were not running on AWS.

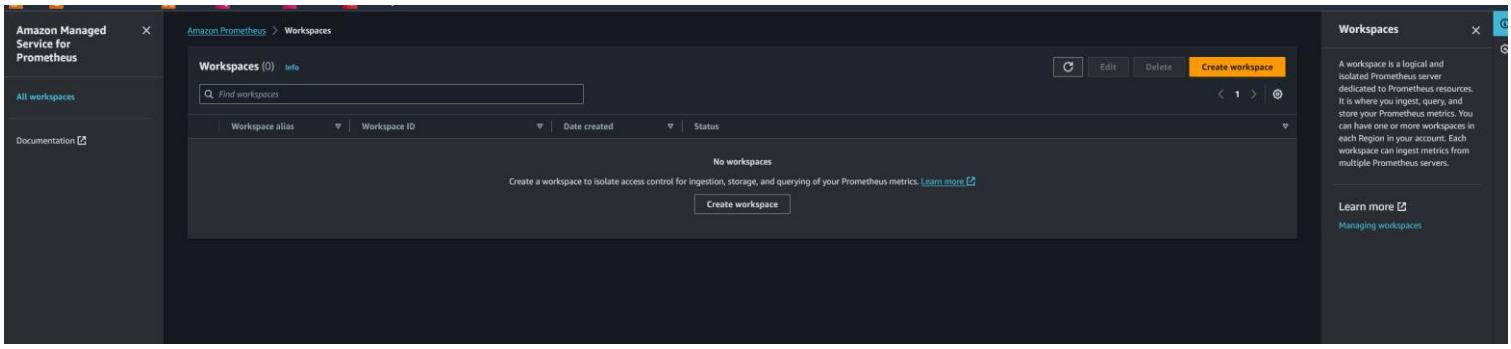


Figure 6: AWS EKS Container missing.

In Figure 6, this is the screen you would usually see an AWS EKS container running. As can be seen, I didn't have one. I spent three days trying to fix the charging issue after getting in contact with support on the 2nd day. The woman suggested I delete the container running and I showed her the screenshot of figure 6.

The woman who contacted said she will talk to the technical team and contact me the next day. When she did, she asked me to run a command which showed there was a service running in Stockholm Sweden on the backend still. Once that was deleted, she contacted me again in 24 hours to confirm that it was solved, and I got refunded the money it was charging since December being over €100.

following on from these issues, I spent a good amount of time using AWS academy which was a €100-euro free learning service to run AWS services. This was to get practice and understand the services, how they worked, how to install and get them up and running as fast as possible. While this was a great way to learn, it was not ideal in the sense that it was limited and missing crucial features that were required in the project.

For starters, I did not have access to EC2 which is required when setting up the container. When connected EKS you need to have the Security Credentials to get the access ID and Secret key, shows in figures 7 and 8, and in 9 you can see what was in the AWS Academy screen.

Figure 7: Security Credentials.

1. In a terminal window, enter the following command:

```
aws configure
```

Optionally, you can configure a named profile, such as `--profile cluster-admin`. If you configure a named profile in the AWS CLI, you must **always** pass this flag in subsequent commands.

2. Enter your AWS credentials. For example:

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: region-code
Default output format [None]: json
```

Figure 8: Access ID and Secret Key.

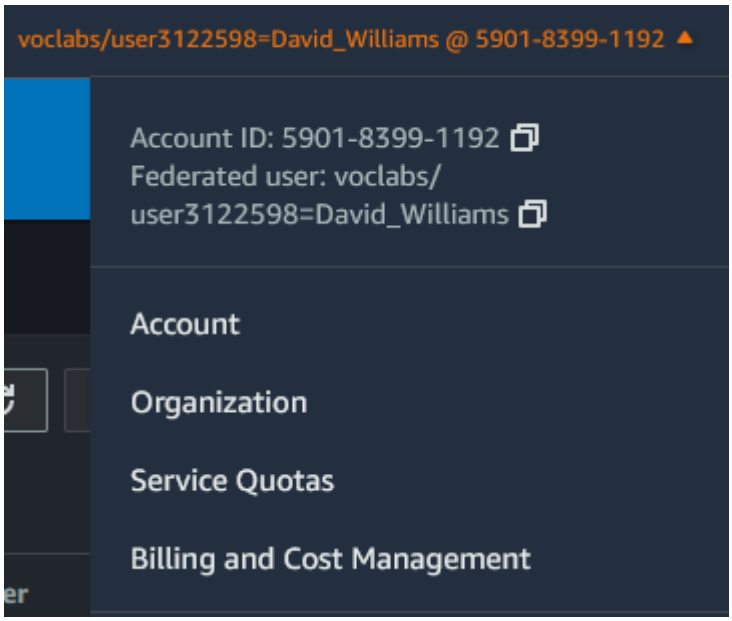


Figure 9: AWS Academy.

Because of this, it caused me configuration issues setting up Kubernetes Kubectl and AWS CLI configuration file.

```
[cloudshell-user@ip-10-132-56-143 /]$ aws eks update-kubeconfig --name RunetimeSecurityCluster --region us-east-1

Can't create directory for writing: [Errno 13] Permission denied: '/etc/kubernetes'
[cloudshell-user@ip-10-132-56-143 /]$ chmod 777 /etc/kubernetes
chmod: cannot access '/etc/kubernetes': No such file or directory
[cloudshell-user@ip-10-132-56-143 /]$ ls /etc/kubernetes
ls: cannot access '/etc/kubernetes': No such file or directory
```

Figure 10: KubeCTL.

Because of all these issues and not fully understanding them because of lack of documentation explanation and knowledge I have on AWS services. I found it hard because it isn't covered in any YouTube video, I watched to get AWS EKS running. I found out the hard way by coming across a page explaining what needs to be set up. This page then linked to an AWS document page explaining the basic steps to setup AWS EKS but only the basics.

After more searching, I finally came across a YouTube video that helped setting up the AWS EKS environment.

Figure 11: AWS EKS tutorial video.

This video was a life saver in setting everything up, but still not 100%. The reasoning behind this is because its for more advanced setups of EKS containers with various aspects of the environments build that I had no clue on what to choose.

These aspects were things like, the IAM users, groups, and policies which there are many, but what policies were right for me and my project? When making these IAM users and groups, I need to know which one to select as you make multiple throughout the build of the environment. Then there is security groups and their policies, which traffic they allow through and much more.

I had learnt that I needed the EC2 container from this and a VPC, subnets, users, groups, security credentials, policies, specific traffic allowances such as HTTP and SSH, how to configure the nodes and getting Ubuntu running on the AWS EKS container.

While the video did help solve some of the previous issues, I still ran into more that took days to figure out how to fix and understand why.

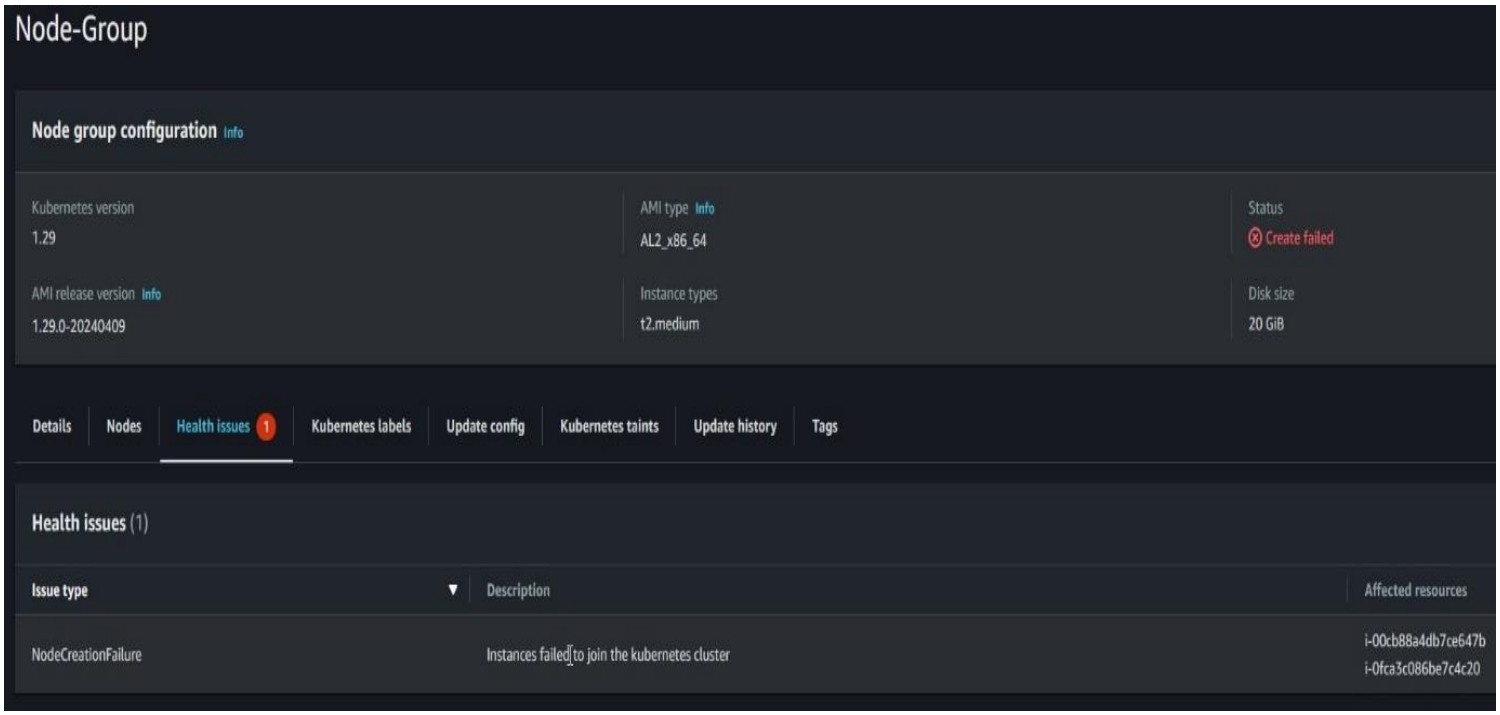


Figure 12: Container failed number 1.

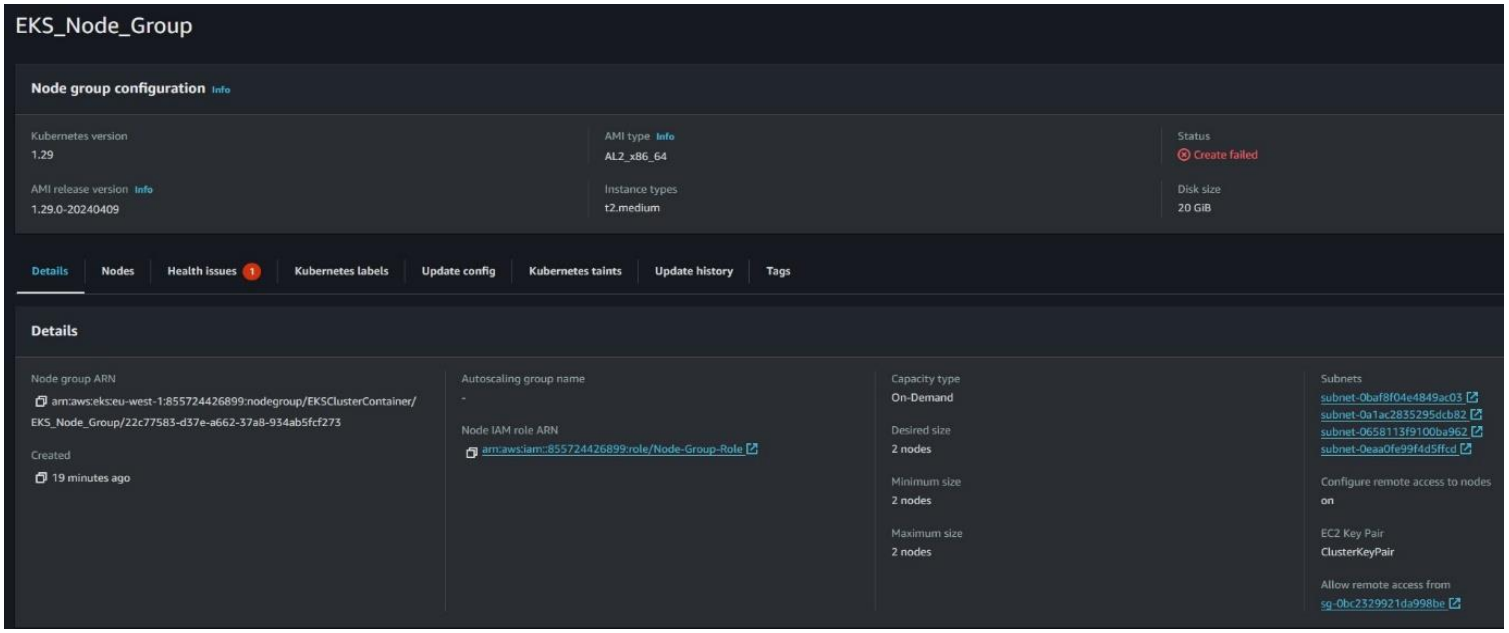


Figure 13: container failed number 2.

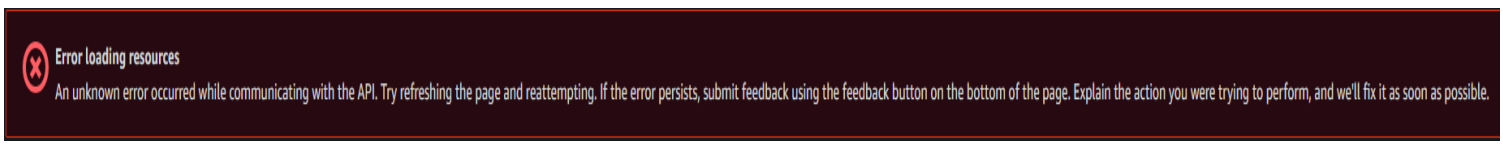


Figure 14: Error using the node-group page.

As can be seen in figure 12 and 13, setting up the node groups wrong can even cause issues which in figure 13 I could understand the issue. I had allowed public subnet groups in the node-group which should not have been allowed.

Figure 14, shows an error that caused me to restart from scratch again as there was an issue creating the EKS container and the create node groups page wouldn't work delaying more. This was possibly due to a connection issue that happened during the process of creating the EKS container.

Although this is a minor issue and easily fixed, the problem is it's time consuming and can cost money the longer you troubleshoot these issues. The best course of action I had to take to save money was to delete everything and start over again. This persisted for days until I figure out where the issues were and what mistakes I had made during the configurations.

```
athash@athash-virtual-machine:~/Cluster$ ssh -i ClusterKeyPair.pem ubuntu@3.251.82.72
The authenticity of host '3.251.82.72 (3.251.82.72)' can't be established.
ED25519 key fingerprint is SHA256:KevO8jNicrJ2AugYyJlIWAFSeDqY5ldPUASFymGOnVA.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '3.251.82.72' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-1014-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

  System information as of Wed Apr 17 15:21:27 UTC 2024

  System load:  0.080078125       Processes:             98
  Usage of /:   20.7% of 7.57GB   Users logged in:       1
  Memory usage: 21%               IPv4 address for eth0: 10.0.11.122
  Swap usage:   0%


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Wed Apr 17 14:49:58 2024 from 18.202.216.53
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-0-11-122:~$ ls
ubuntu@ip-10-0-11-122:~$ pwd
/home/ubuntu
ubuntu@ip-10-0-11-122:~$
```

Figure 15: Connection to AWS EKS Ubuntu, through VM Ubuntu.

After enough time I eventually got a CLI Ubuntu container running and started installing falco on the AWS EKS container *through my VM Ubuntu machine* instead of doing it directly through AWS EKS CLI to assure connectivity was possible.

```
ubuntu@ip-10-0-11-122:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   172.20.0.1   <none>        443/TCP   103m
ubuntu@ip-10-0-11-122:~$ kubectl cluster-info
Kubernetes control plane is running at https://51772EAEC84CF5F97B932163AA47F91B.gr7.eu-west-1.eks.ama
zonaws.com
CoreDNS is running at https://51772EAEC84CF5F97B932163AA47F91B.gr7.eu-west-1.eks.amazonaws.com/api/v1
/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
ubuntu@ip-10-0-11-122:~$
```

Figure 16: Kubectl installed on AWS EKS Ubuntu.

```
ubuntu@ip-10-0-11-122:~$ helm install falco falcosecurity/falco
NAME: falco
LAST DEPLOYED: Wed Apr 17 16:18:03 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Falco agents are spinning up on each node in your cluster. After a few
seconds, they are going to start monitoring your containers looking for
security issues.


No further action should be required.


Tip:
You can easily forward Falco events to Slack, Kafka, AWS Lambda and more with falcosidekick.
Full list of outputs: https://github.com/falcosecurity/charts/tree/master/charts/falcosidekick.
You can enable its deployment with `--set falcosidekick.enabled=true` or in your values.yaml.
See: https://github.com/falcosecurity/charts/blob/master/charts/falcosidekick/values.yaml for configu
ration values.
ubuntu@ip-10-0-11-122:~$
```

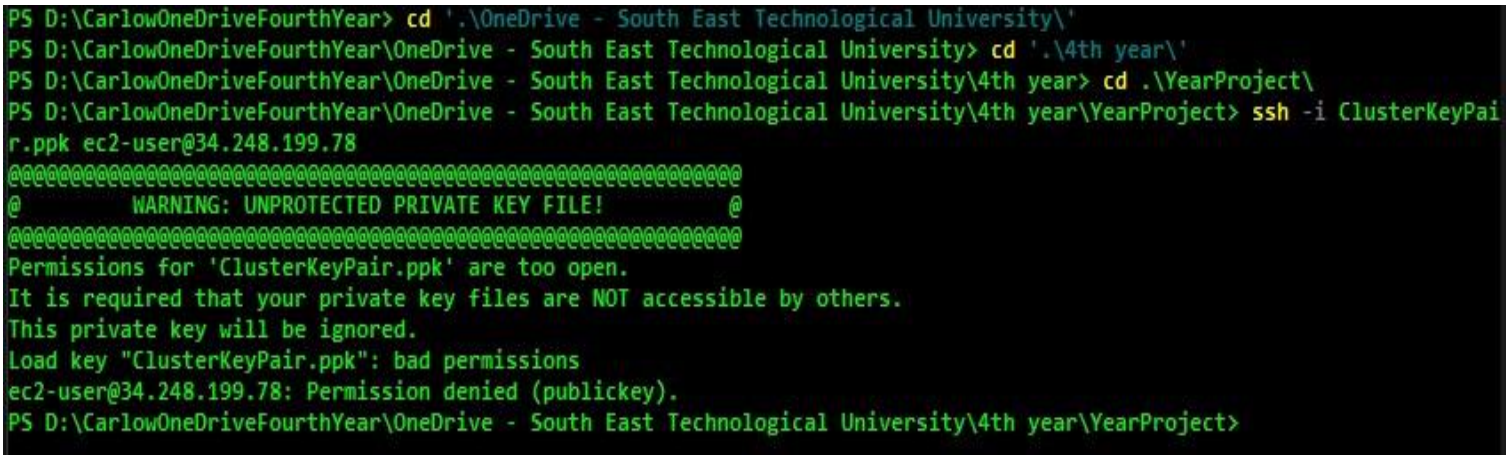
Figure 17: Falco installed on AWS EKS Ubuntu.

I did run into issues with kubectl get svc from figure 15, like such,

```
    - json
    command: aws
    env: null
    interactiveMode: IfAvailable
    provideClusterInfo: false
ubuntu@ip-10-0-4-30:~$ kubectl get svc
E0417 10:10:53.160274    3370 memcache.go:265] couldn't get current server API group list: the server has asked for the client to provide credentials
E0417 10:10:53.763628    3370 memcache.go:265] couldn't get current server API group list: the server has asked for the client to provide credentials
E0417 10:10:54.593323    3370 memcache.go:265] couldn't get current server API group list: the server has asked for the client to provide credentials
E0417 10:10:55.200287    3370 memcache.go:265] couldn't get current server API group list: the server has asked for the client to provide credentials
E0417 10:10:55.803562    3370 memcache.go:265] couldn't get current server API group list: the server has asked for the client to provide credentials
error: You must be logged in to the server (the server has asked for the client to provide credentials)
ubuntu@ip-10-0-4-30:~$ kubectl get svc
E0417 10:14:44.570940    3380 memcache.go:265] couldn't get current server API group list: the server has asked for the client to provide credentials
E0417 10:14:45.176744    3380 memcache.go:265] couldn't get current server API group list: the server has asked for the client to provide credentials
E0417 10:14:45.997270    3380 memcache.go:265] couldn't get current server API group list: the server has asked for the client to provide credentials
E0417 10:14:46.594730    3380 memcache.go:265] couldn't get current server API group list: the server has asked for the client to provide credentials
E0417 10:14:47.201839    3380 memcache.go:265] couldn't get current server API group list: the server has asked for the client to provide credentials
error: You must be logged in to the server (the server has asked for the client to provide credentials)
ubuntu@ip-10-0-4-30:~$
```

Figure 18: kubectl get svc error.

I managed to fix this issue, using some other kubectl config commands which were done from the video. But there still came more to overcome and eventually I had to stop due to one issue becoming another even after fixing.



```
PS D:\CarlowOneDriveFourthYear> cd '.\OneDrive - South East Technological University\'
PS D:\CarlowOneDriveFourthYear\OneDrive - South East Technological University> cd '.\4th year\'
PS D:\CarlowOneDriveFourthYear\OneDrive - South East Technological University\4th year> cd .\YearProject\
PS D:\CarlowOneDriveFourthYear\OneDrive - South East Technological University\4th year\YearProject> ssh -i ClusterKeyPai
r.ppk ec2-user@34.248.199.78
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions for 'ClusterKeyPair.ppk' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "ClusterKeyPair.ppk": bad permissions
ec2-user@34.248.199.78: Permission denied (publickey).
PS D:\CarlowOneDriveFourthYear\OneDrive - South East Technological University\4th year\YearProject>
```

Figure 19: ppk file instead of pem.

In figure 19, I needed a .pem file and I only had a .ppk file which I spent to much time looking for even though the .pem file was never created. I only knew of the ppk file so I had to come up with a way to convert it using Puttygen which I came across after some googling.

Without this pem file, you could not connect to the AWS EKS either as it helps the access ID and Secret key which are necessary. This also provides root access as I decided to use a root user and not an IAM user at first to get everything setup up and started. The video also showed you can change to IAM users later, but I never made it that far as I had run into more issues setting up the configuration on the AWS EKS Ubuntu side.

When I got AWS EKS Ubuntu machine running, initially I didn’t realise I would have to install AWS CLI on the machine as it usually does.

```
ubuntu@ip-10-0-4-30:~$ aws cli
Command 'aws' not found, but can be installed with:
sudo apt install awscli
ubuntu@ip-10-0-4-30:~$ sudo apt install awscli
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Package awscli is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source

E: Package 'awscli' has no installation candidate
ubuntu@ip-10-0-4-30:~$
```

Figure 20: aws cli.

Without this, I could not configure the security credentials to connect to the kubectl container for the connection.

Despite getting Falco containers working and setup.

```
ubuntu@ip-10-0-11-122:~$ kubectl logs falco-27vlx -n default
Defaulted container "falco" out of: falco, falcoctl-artifact-follow, falco-driver-loader (init), falcoctl-artifact-instal
l (init)
Wed Apr 17 16:18:40 2024: Falco version: 0.37.1 (x86_64)
Wed Apr 17 16:18:40 2024: Falco initialized with configuration file: /etc/falco/falco.yaml
Wed Apr 17 16:18:40 2024: System info: Linux version 5.10.213-201.855.amzn2.x86_64 (mockbuild@ip-10-0-57-195) (gcc10-gcc
(GCC) 10.5.0 20230707 (Red Hat 10.5.0-1), GNU ld version 2.35.2-9.amzn2.0.1) #1 SMP Mon Mar 25 18:16:11 UTC 2024
Wed Apr 17 16:18:40 2024: Loading rules from file /etc/falco/falco_rules.yaml
Wed Apr 17 16:18:40 2024: Hostname value has been overridden via environment variable to: ip-10-0-132-124.eu-west-1.compu
te.internal
Wed Apr 17 16:18:40 2024: The chosen syscall buffer dimension is: 8388608 bytes (8 MBs)
Wed Apr 17 16:18:40 2024: Starting health webserver with threadiness 2, listening on 0.0.0.0:8765
Wed Apr 17 16:18:40 2024: Loaded event sources: syscall
Wed Apr 17 16:18:40 2024: Enabled event sources: syscall
Wed Apr 17 16:18:40 2024: Opening 'syscall' source with Kernel module
```

Figure 21: Falco container in kubectl.

```
ubuntu@ip-10-0-11-122:~$ helm ls
NAME    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSI
ON
falco   default         1               2024-04-17 16:18:03.541747339 +0000 UTC deployed        falco-4.3.0     0.37.1

ubuntu@ip-10-0-11-122:~$ kubectl get pods -n default
NAME          READY   STATUS    RESTARTS   AGE
falco-27vlx   2/2     Running   0          4m32s
falco-2sh68   2/2     Running   0          4m32s
ubuntu@ip-10-0-11-122:~$ kubeclt logs falco-0.37.1-x86_64.deb
.aws/                   .cache/                 .ssh/                   falco-0.37.1-x86_64.deb
.bash_history           .config/                .sudo_as_admin_successful  get_helm.sh
.bash_logout            .kube/                  .wget-hsts              kubectl
.bashrc                 .profile                bin/                    kubectl.sha256
```

Figure 22: Falco on the AWS EKS Ubuntu container.

I still ran into issues with the project that I cannot understand or explain as they made no sense. Even though I was connected to the AWS EKS Ubuntu container through my VM Ubuntu machine. I could not ping the AWS EKS Ubuntu container which made no sense. I checked this pinging after I ran into the following issues.

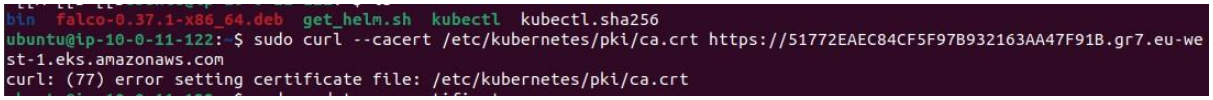


Figure 23: Error 77.

A certificate issue which I had to manually create a folder in the /etc/ directory making the following /Kubernetes/pkt/ folders and then installing the ca.crt using a pip command.

After the certificate issue was solved and I tried to connect to the machine, I get the error (60) which is another issue I tried to fix but no luck.

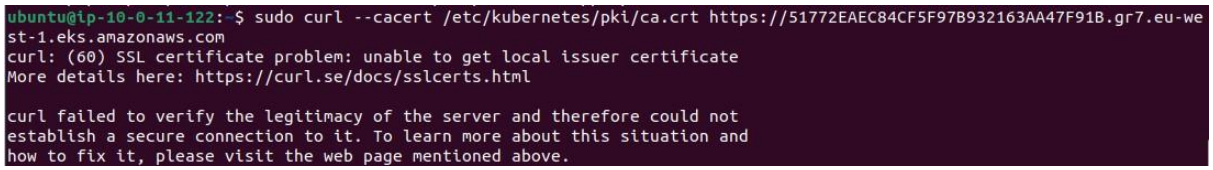


Figure 24: Error 60.

Finally, I ran into error (7):

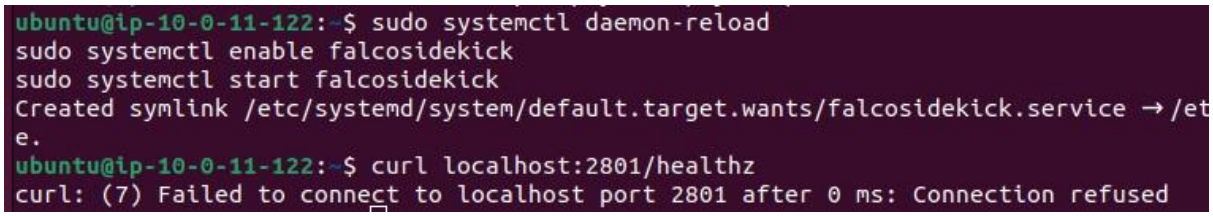


Figure 25: Error (7).

After hours of googling, YouTube videos, ChatGPT commands I decided to take a break from trying to fix these issues through commands and decided to mess around with the policies, security groups and configurations on AWS side where I opened traffic to all. I included it to allow HTTP, HTTPS and even tries some telnet commands which is considered insecure. Nothing I did worked and eventually got a message from amazon saying this:

Your AWS account 855724426899 has exceeded 85% of the usage limit for one or more AWS Free Tier-eligible services for the month of April.

Product	AWS Free Tier Usage as of 04/17/2024	Usage Limit	AWS Free Tier Usage Limit
AWSDataTransfer	0.93011605 GB	1 GB	1.0 GB are always free per month as part of AWS Free Usage Tier (Global-DataTransfer-Regional-Bytes)

Figure 26: Usage limit.

This meant that if I continued to run the AWS EKS Ubuntu machine and go over the usage limit, I would be charged significantly more money which wasn't good as I had already used more than I hoped to intend and will not disclose.

This put me in a position to try rush out anything in the final moments and try screen shot something of the project to show the examiners something before it's to late and unable to progress the project further.

I start by trying to connect Falcosidekick UI to the machine:

```
athash@athash-virtual-machine:~$ helm repo add falcosecurity https://falcosecuri
ty.github.io/charts
"falcosecurity" has been added to your repositories
athash@athash-virtual-machine:~$ help repo update
bash: help: no help topics match `update'.  Try `help help' or `man -k update' o
r `info update'.
athash@athash-virtual-machine:~$ helm install falco falcosecurity/falco --set fa
lco.docker.enabled=false --set falco.jsonOutput=true --set falcosidekick.enabled
=true --set falcosidekick.webui.enabled=true -n falco
Error: INSTALLATION FAILED: Kubernetes cluster unreachable: Get "http://localhos
t:8080/version": dial tcp 127.0.0.1:8080: connect: connection refused
athash@athash-virtual-machine:~$ sudo helm install falco falcosecurity/falco --s
et falco.docker.enabled=false --set falco.jsonOutput=true --set falcosidekick.en
abled=true --set falcosidekick.webui.enabled=true -n falco
[sudo] password for athash:
Error: INSTALLATION FAILED: repo falcosecurity not found
athash@athash-virtual-machine:~$ ^C
```

Figure 27: Falcosidekick ui failing to work.

After trying for a few more hours making sure I followed all documentation on how to install kubectl, falco and even installed Falco through the helm commands. Even though I followed the documents step by step:

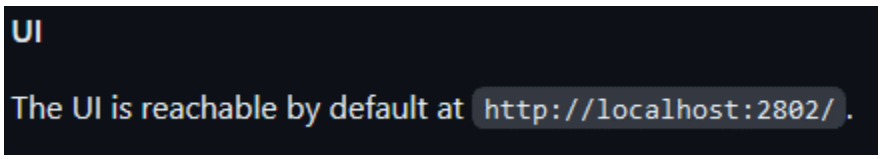


Figure 28: How to access falcosidekick UI.

Even trying the curl commands suggested online to connect into the localhost 2802, I still could not gain access to GUI no matter what. I tried this inside the VM Ubuntu machine and the AWS EKS Ubuntu container and still nothing.

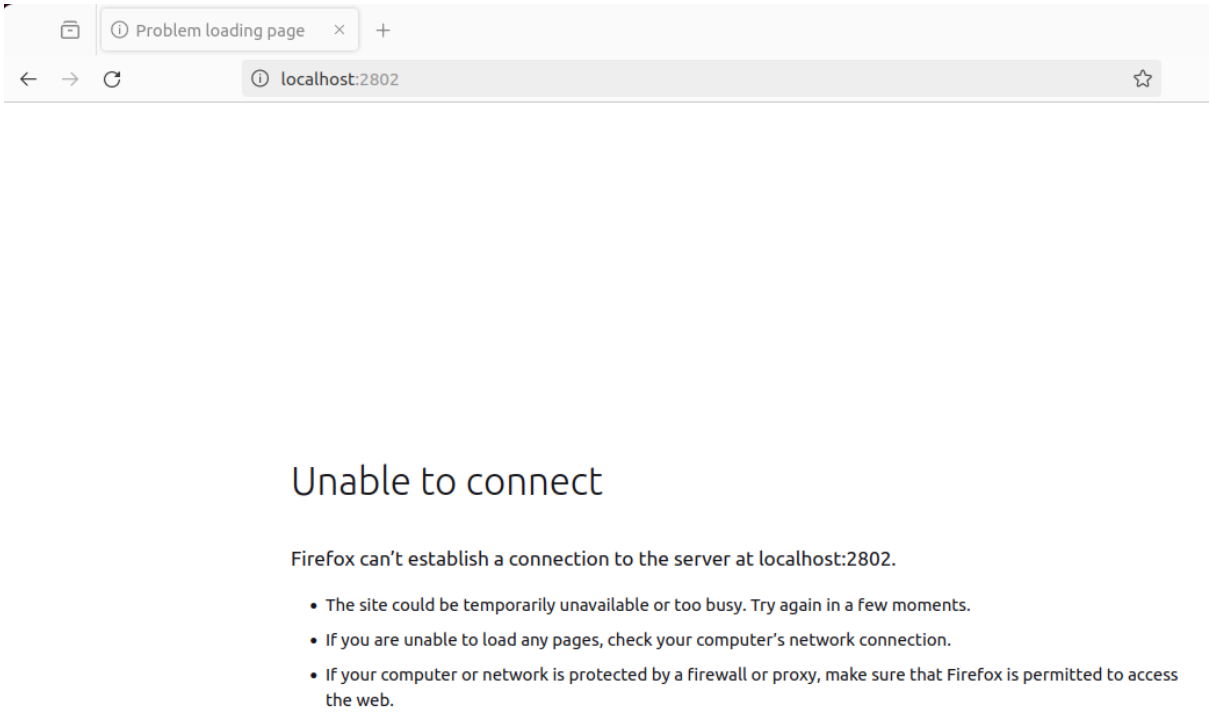


Figure 29: Falcosidekick UI failing to show.

After hours of trying everything I possibly could try to configure and alter the current configuration, even if it meant opening up the entire container and VM insecurely, which goes against the plan of this project. Nothing worked, and I had exceeded the free tier usage limits. I decided it to delete the AWS EKS Ubuntu container and everything in relation to it such as the VPC, IAM users, groups the security groups e.g.

Although I could have just shut everything down which would prevent further costs. I had already exceeded the free trials limitations and only rebooting it back up would further the costs. After talking to my mentor, Hisain who suggested I should have kept the AWS EKS container intact for the showcase.

What I learned

I learned a lot from all aspects of the project ranging from Linux machine commands and get more familiar with the Ubuntu environment. AWS services and how they work, how much they are used around the world, to the variety of different tools and their uses. This project used a variety of those tools such as EKS, VPC, IAM roles and users, and security groups.

Falco was a great learning experience, seeing and reading it's potential knowing that a user can build rules based on MITRE ATTACK cybersecurity rules. When using the tool, it immediately picked up me trying to edit files within the VM Ubuntu environment. Even if I opened the file in a tool like Visual Studio Code, it put out event alerts.

Learning about cloud environments is always a great thing in todays time considering that a lot of companies are moving their infrastructure towards cloud and AWS is expanding their cloud services in Europe to match the needs of European organisations. Their use does serve to cut costs on companies relying on their own data centers but also from a security perspective. This means you are allowing the company who holds the data to see everything. So, from a security perspective this is not great if all servers in relation to a company are compromised.

This is why the focus on security for containers was the objective of this project, to help find ways to assure that no harm can come to the container and the information within is compromised. Sadly, I never got far enough to perform the testing to ensure that this is possible.

What you would do differently if starting again

The hardest question is this as my conclusion doing this project is that the project is pretty advanced for someone starting out AWS services and Falco without prior knowledge required me getting help from the Falco community. The documentation that exists on both sides isn't exactly great and there was no YouTube video to follow that would have been helpful in many situations.

What I consider the best approach if I was to start over is to focus more on the AWS side and less on the Falco side. Using AWS Academy to get the fundamental understanding of the tools and technologies. Then once familiarised with them, move onto using the official AWS account configuring and setting up the environment alongside the VM Ubuntu until they both are properly configured and can open a webpage through the curl command.

Additionally, try implementing a GUI on the AWS EKS Ubuntu container so that I can make sure it properly connects to webpages and can even open webpages to assure it's connected and working as intended.

Following on, I would get Falco installed on both the VM Ubuntu machine and the AWS EKS Ubuntu container. Connect them both and make sure they work as intended. Once Falco is installed and working, I would get falcosidekick UI installed so I can see the events monitored through the GUI.

When Falco events are being sent from the AWS EKS Ubuntu container to the GUI, I would also use a command I found that will cause all sorts of events to happen and change some of the Ubuntu environments files. This will see how Falco reacts to the commands and why it produces in terms of events.

Finally I would get Calico installed, which is an Intrusion prevention networking tool that could help add additional security to both environments.

Report of any differences from your earlier design, and any additional

research that was required

There were no differences changed from the earlier design but there was lots of additional research that helped progress the project further.

Although the research used didn't help me get the project to where I wanted it, I had hoped to get the end and have seen everything running and the container protected. I did learn more about the project itself and everything involved, unfortunately not how to solve many of the issues that came with the project.

The research that helped progress was a website that was doing their own AWS EKS build and installation which had a link to documents that I did not find before. These helped me know the basic requirements of the AWS EKS container setup.

Difficulties in implementing your project

The difficulties were mostly on AWS side and connecting it to the VM Ubuntu and making them work together. Because I never achieved this, the project came to a stop. I will spend the next few days after this report trying to setup a container and see if I can figure out the issues.

While Falco had a rocky start, I managed to get it working on both ends of the project being VM Ubuntu and AWS EKS Ubuntu container. Just unfortunately I could not connect them together and have them work together to output events to the sidekick GUI.

Issues with tools or software used in the project development

The Issues were not understanding what commands to run and perform events at first with Falco but thanks to the community, I got that working and fully understood. AWS on the other hand was much harder, due to the number of different configurations required to get it setup and working.

Testing used to assess reliability of your software product

When testing Falco through the VM Ubuntu machine, it was responsive and effective and outputting events immediately, when testing the tool. The reliability was further improved when I was able to use Visual Studio Code to open a sensitive file and it still noticed that file being manipulated when it never should be.

With full configuration of the project, I would have been able to test and concluded the reliability of the tools and software's involved more thoroughly. However, from resources, YouTube videos, documentation, and other sources. I can confidently say that Falco would have provided excellent results and reliability.