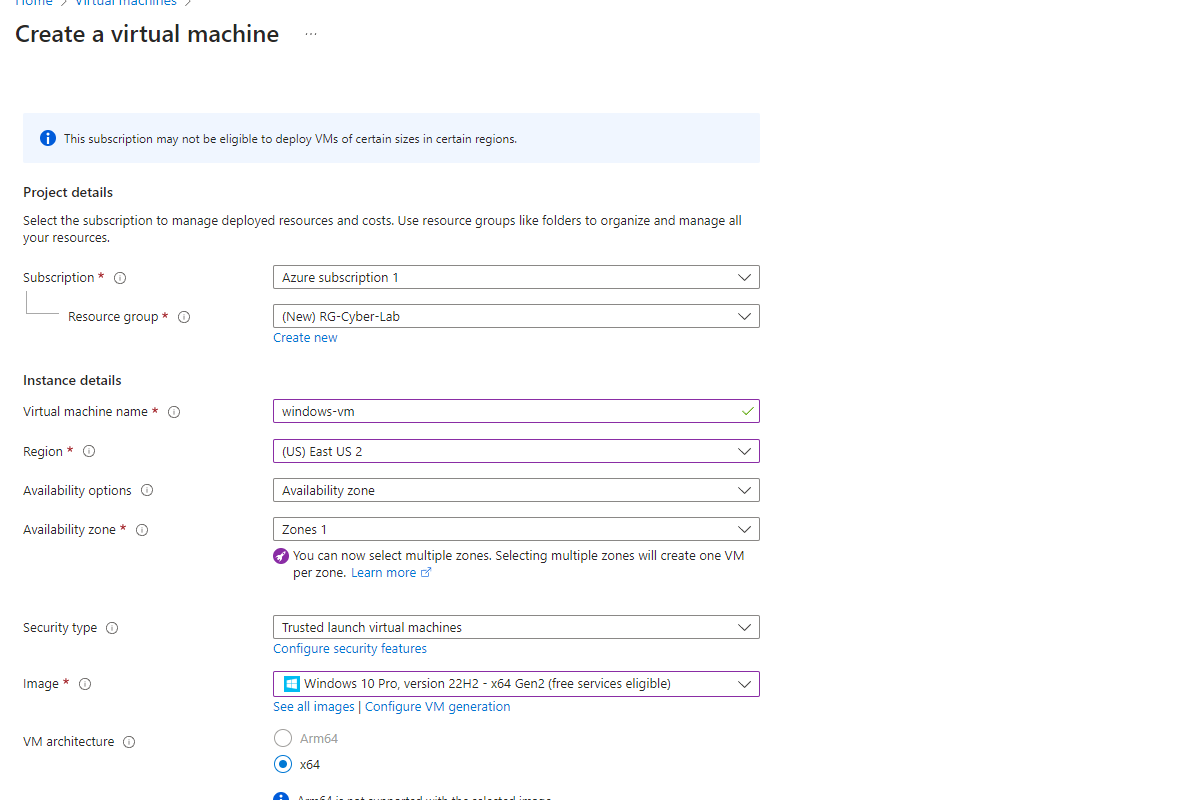
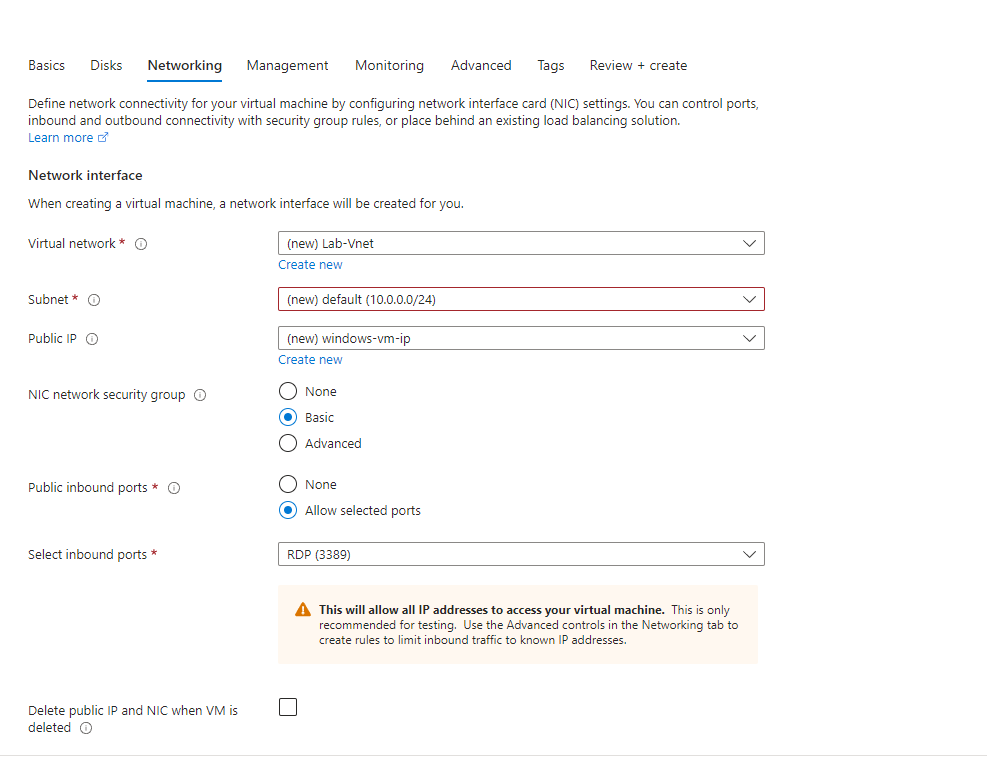
We will first create out windows 10 VM that will be used in our honeynet

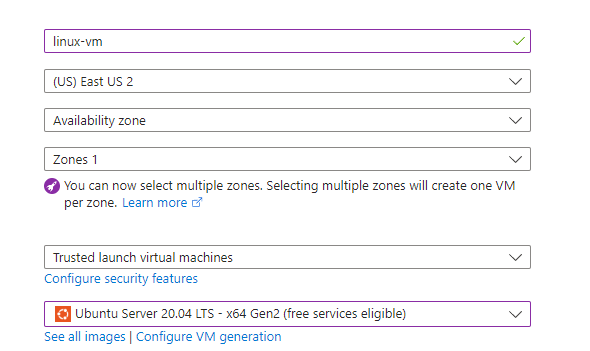


We ended up choosing a bit of a beefier RIG because otherwise we would get DDOS’d and out logs would stop feeding into our SIEM. Here is what I choose:

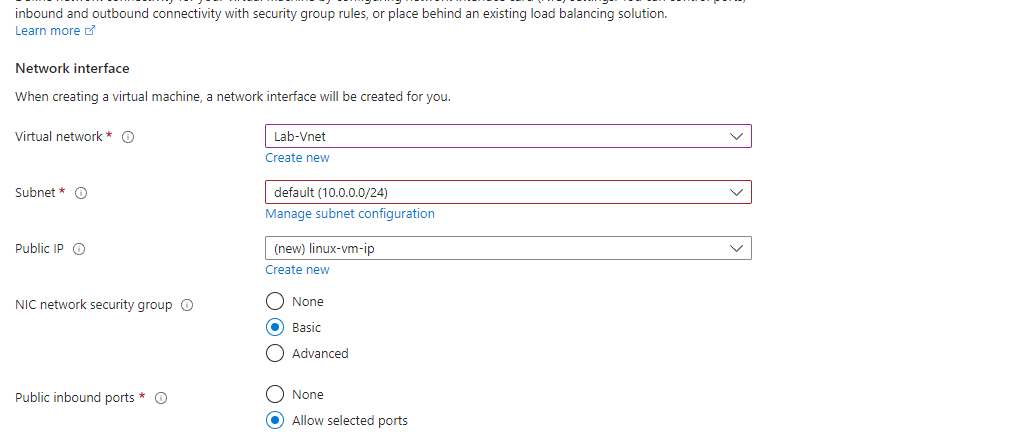


We pretty much left everything else as stock but made a new VNET as you can see here

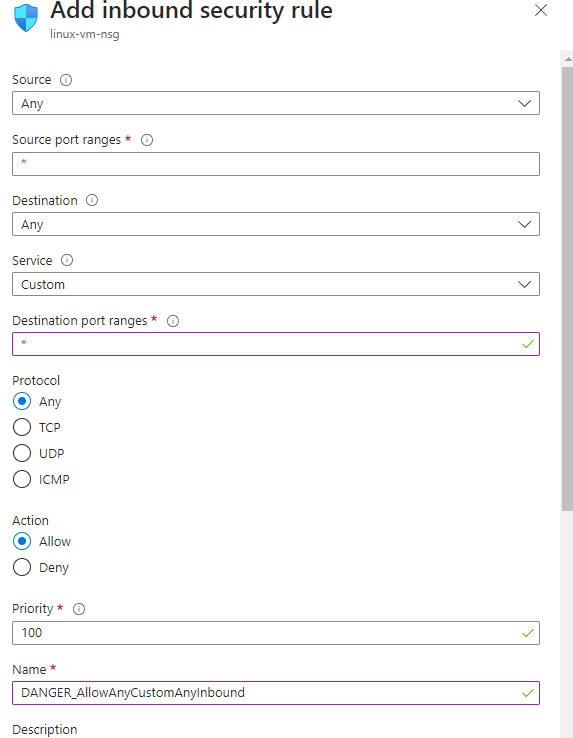




We will now create another VM but this time a linux machine in the same resource group as the other.

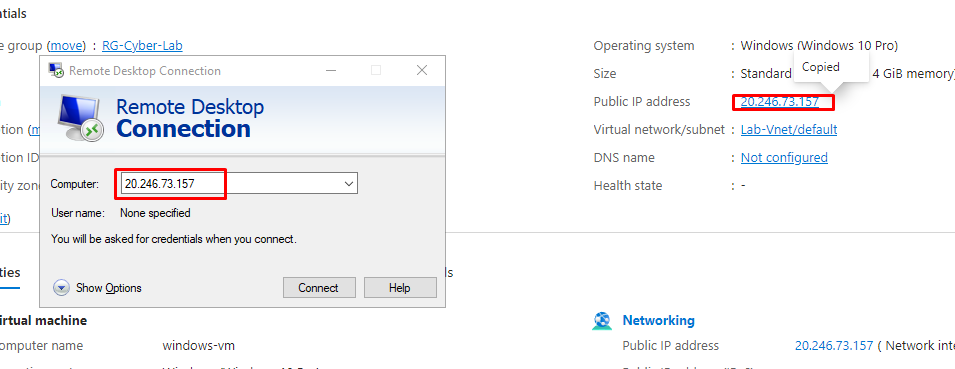


We will put it in the same VNET and the other machine as well so they can communicate with one another!



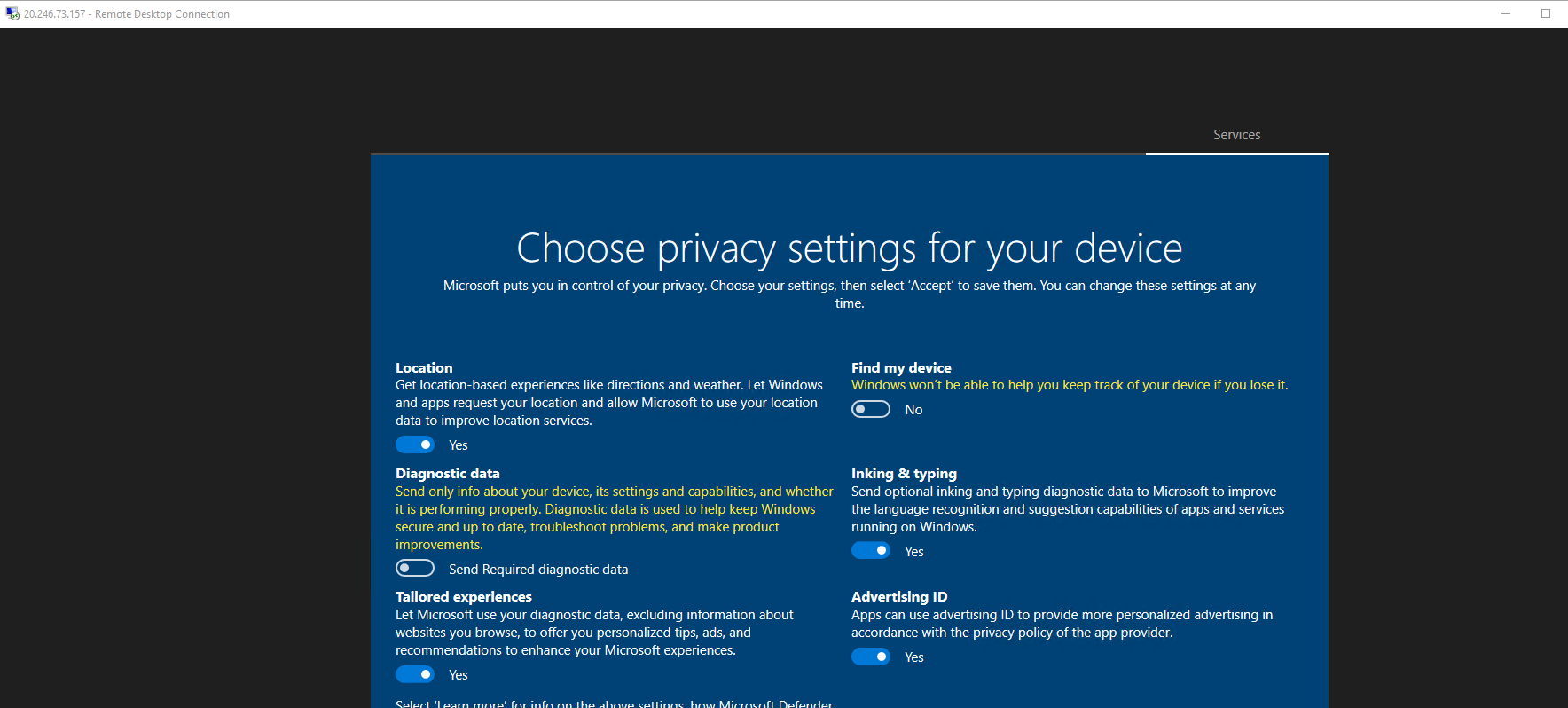
We will now also add an inbound rule to both of the VMs that allow any traffic to reach to them from the internet!

What we will do now is install the SQL Database onto the windows VM.



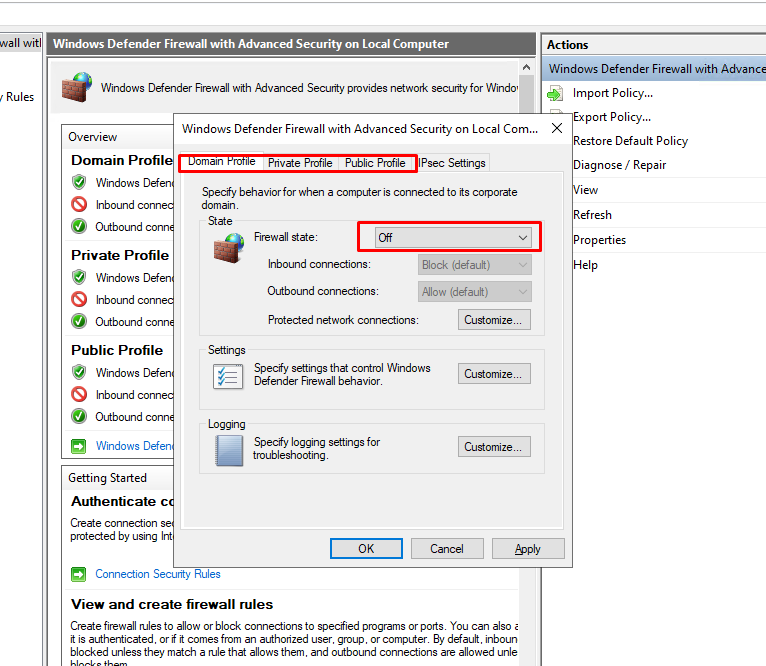
We will first RDP into the Windows VM like this.

We will use the User and Password we set before and we should be right here:

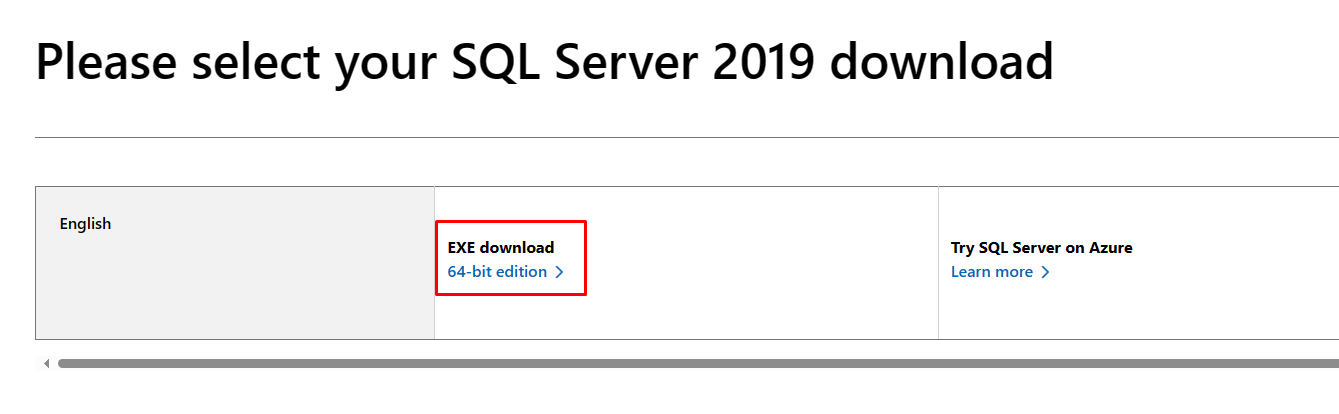


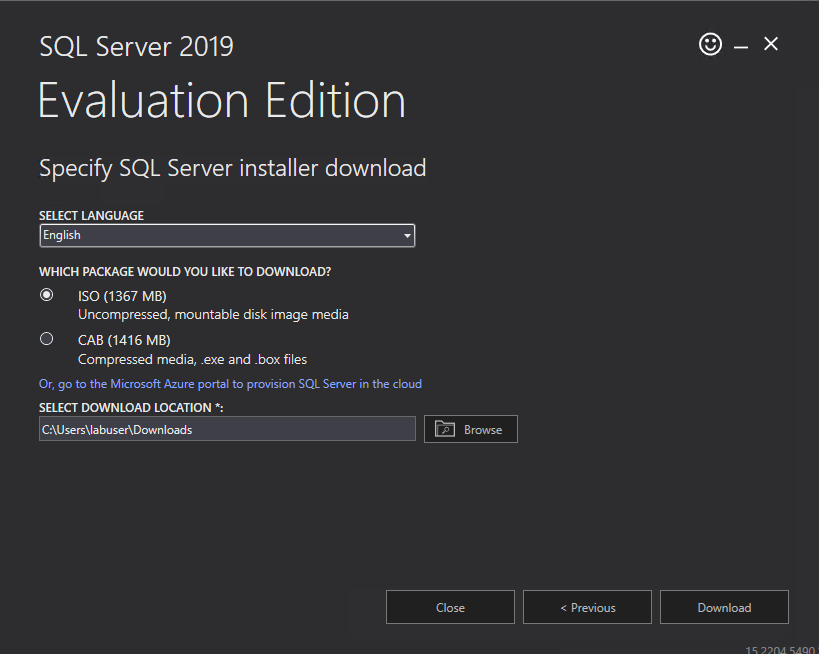
We will now proceed to Turn off the windows Firewall. (So, it can be easier to find online)

For example, with ping requests.

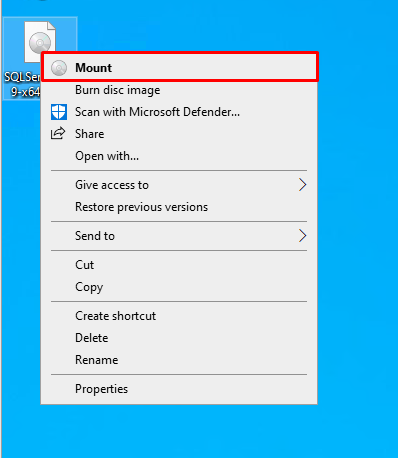


We will now proceed to install Install SQL Server Evaluation

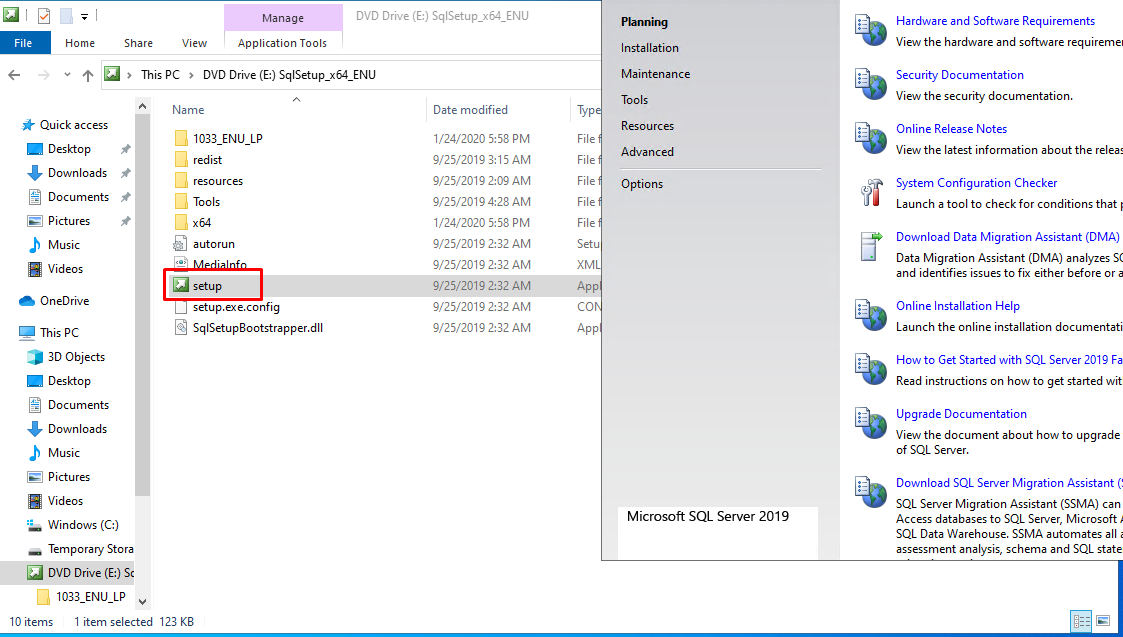




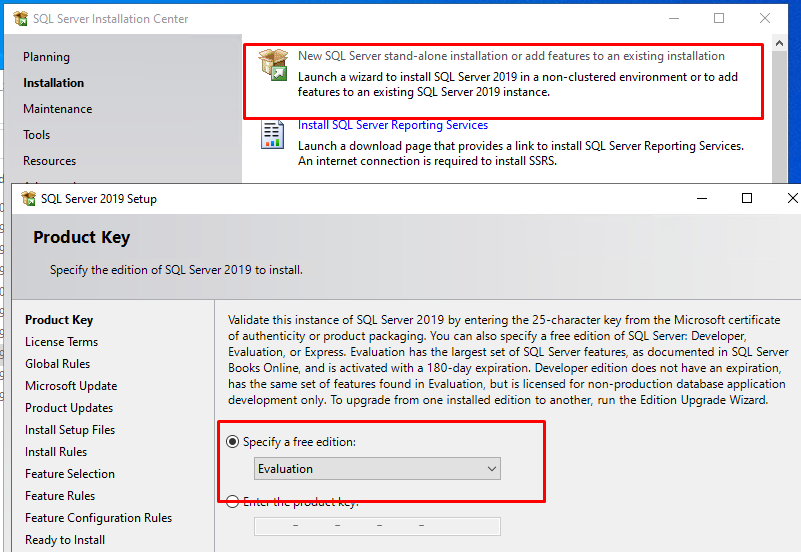
We will keep going with the installation.



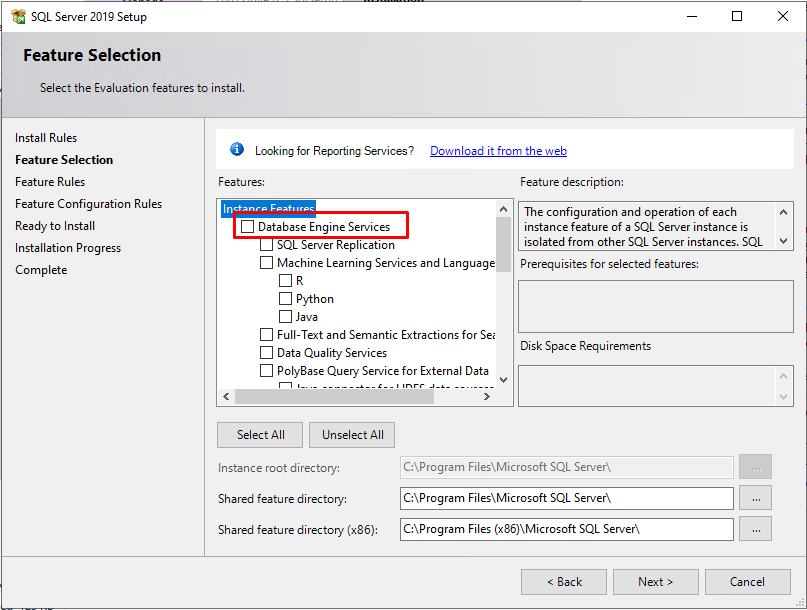
Since this is a ISO file we need to right click it and MOUNT!



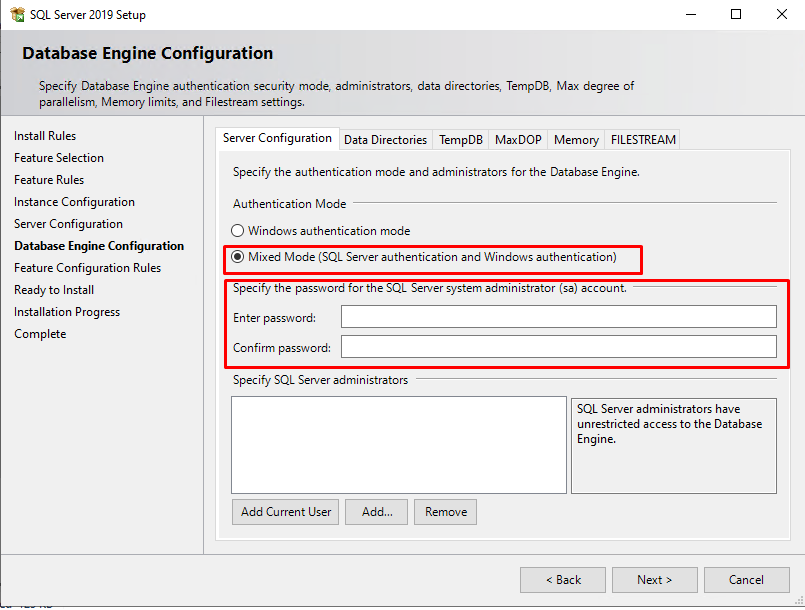
Now we run the setup.



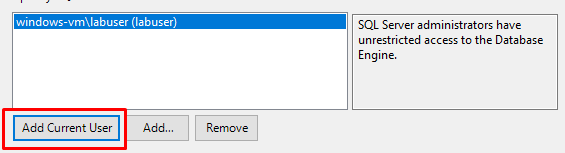
We will pick the first option and just leave it as evaluation.



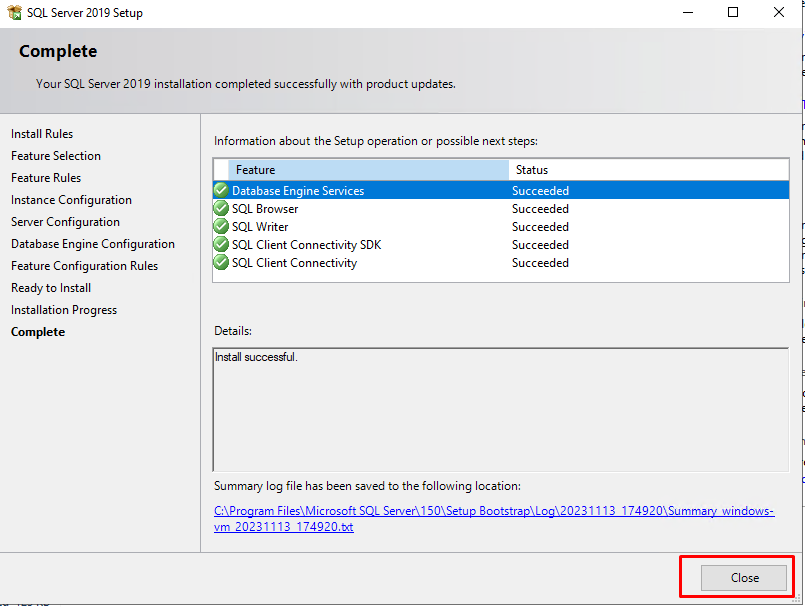
When we get to here we check the first option.



When we get to here, we will pick the second option and I will now input the admin password I want.



We will also add ourselves as an admin.

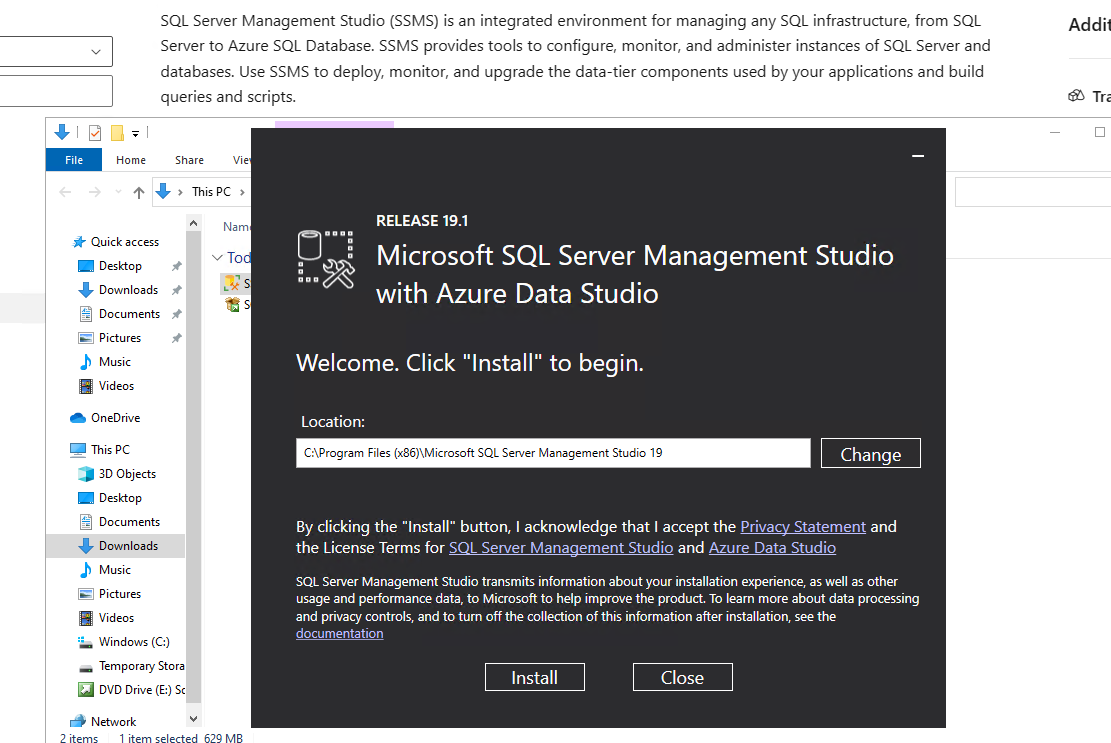


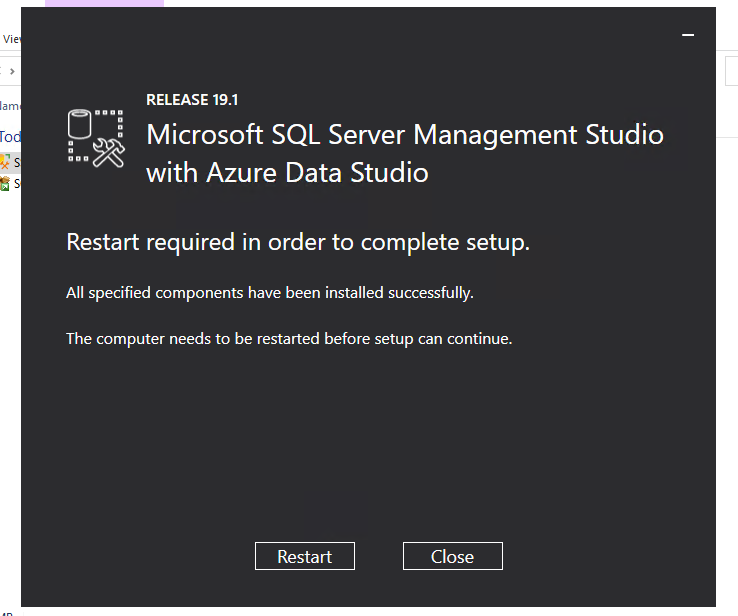
We keep going and when we get here, we just close out.

We will now install:

* SSMS (SQL Server Management Studio): <https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms>

Which we will use to login and generate logs for our SQL database.





We will now restart the VM to complete the install.

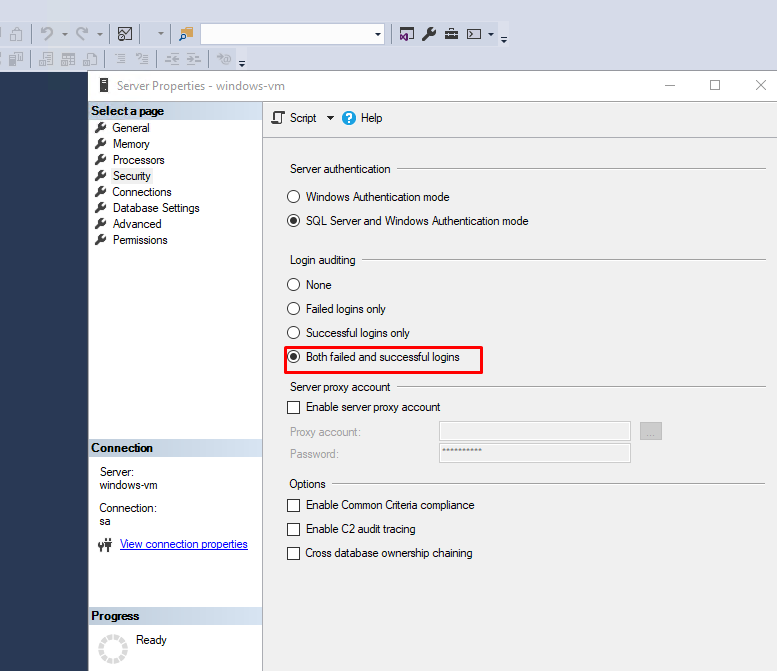
Next what we want is to

Enable logging for SQL Server to be ported into Windows Event Viewer

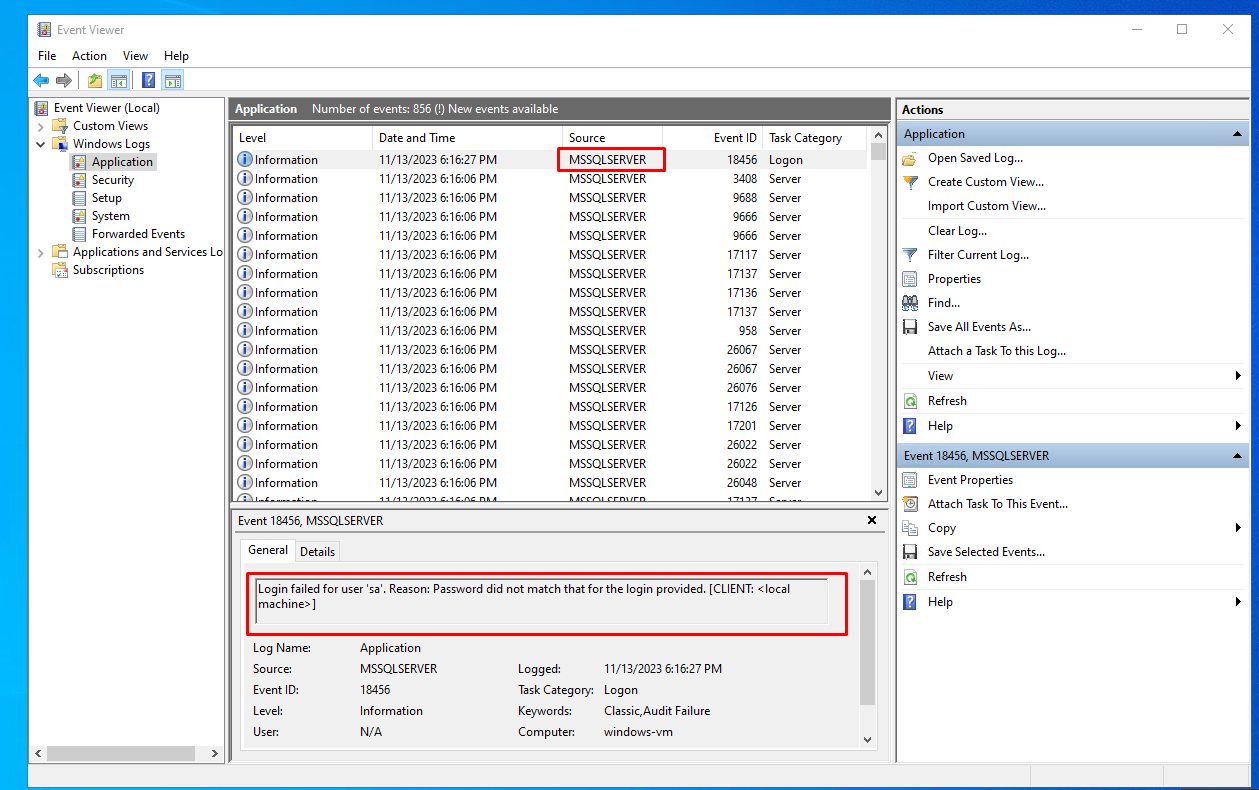
So that we can keep track of who is trying to login, etc.… Just so that we can use our SIEM later to practice incident response.

I will follow these steps:

<https://learn.microsoft.com/en-us/sql/relational-databases/security/auditing/write-sql-server-audit-events-to-the-security-log?view=sql-server-ver16>



After doing all that we want to make sure as well that we want to log both failed and successful login attempts.

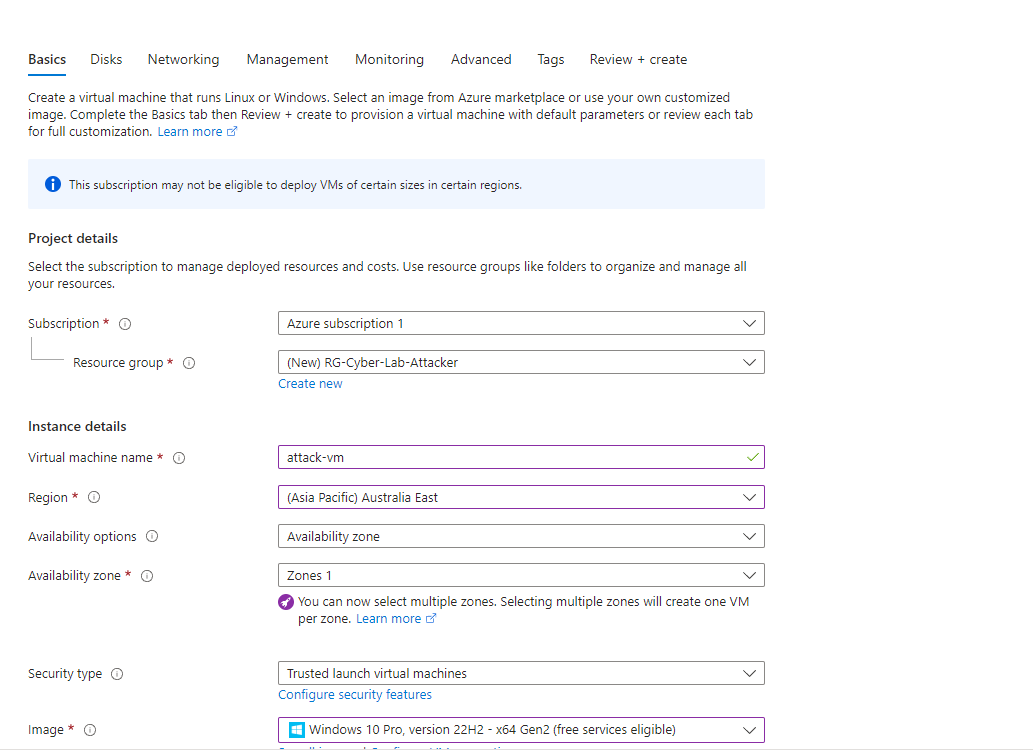


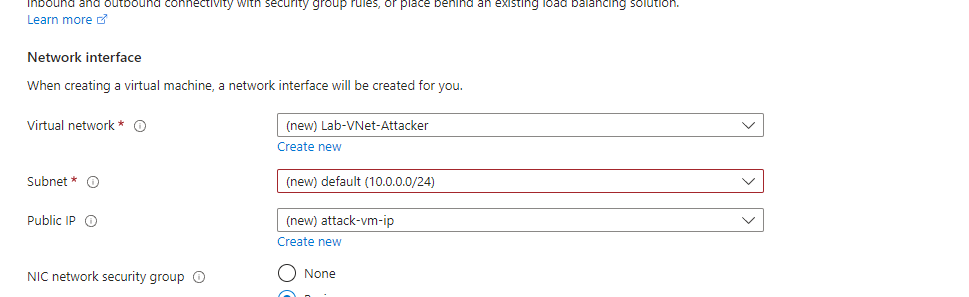
After trying a wrong password, we can indeed see that it works and out events are now being logged into the event viewer!

I will also test the Linux vm by logging into it and pinging it from the windows VM!

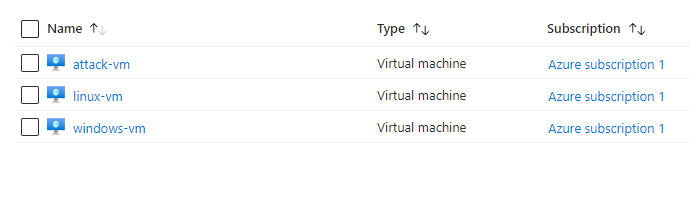
We will now proceed to create a third VM, this will be out ATTACK vm where we will try to connect to the SQL database over the internet as well as the SSH login on the linux machine.

And after the fact we will observer the logs.





We will create another vnet so we can make sure they are segmented.



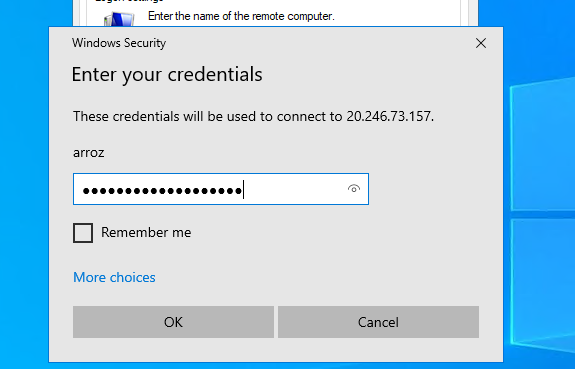
Here is what we have so far.

We will now login to the attack VM using RDP.

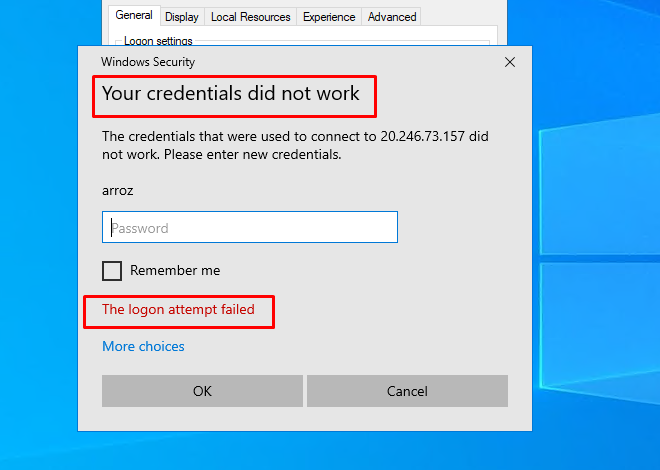
What will we do now is FROM WITHIN the ATTACK vm we will try and RDP into the windows VM (sql database) and input the wrong info on purpose to generate some logs.



So we will put a random username for example

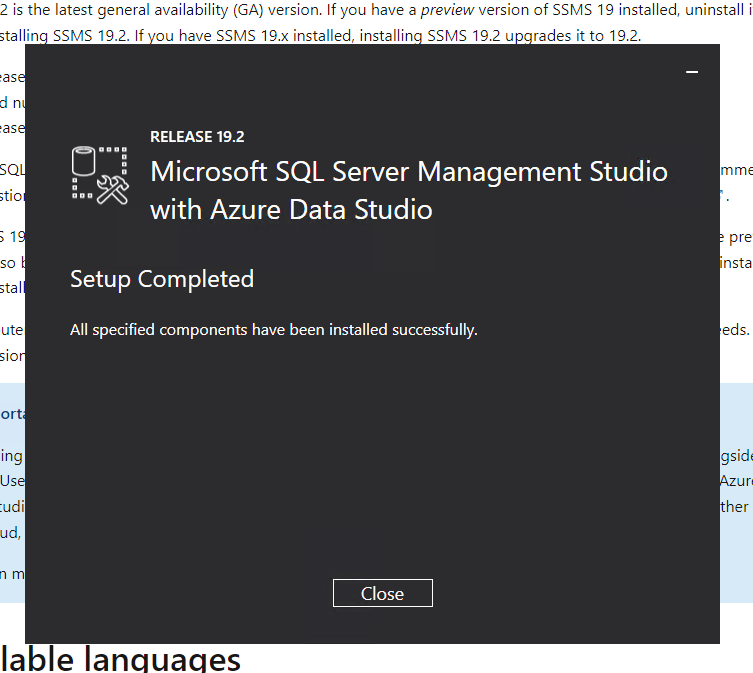


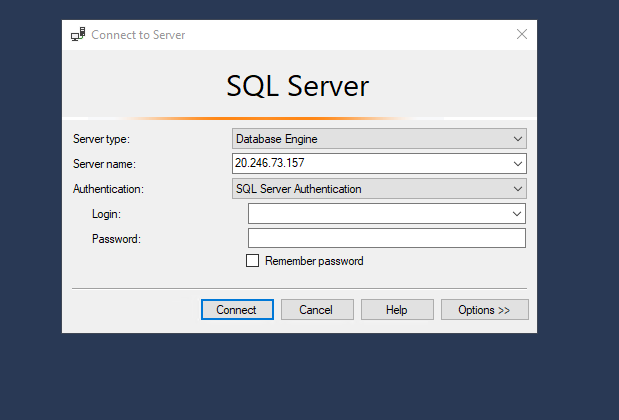
And random pass



And this is what we want.

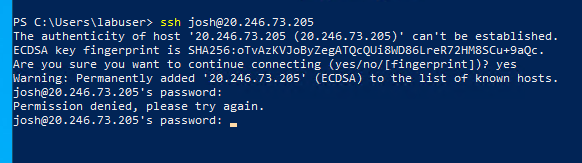
What we will do now is install SSMS and we will generate some SQL logs instead of the windows Logs. Let’s generate some logs for that.





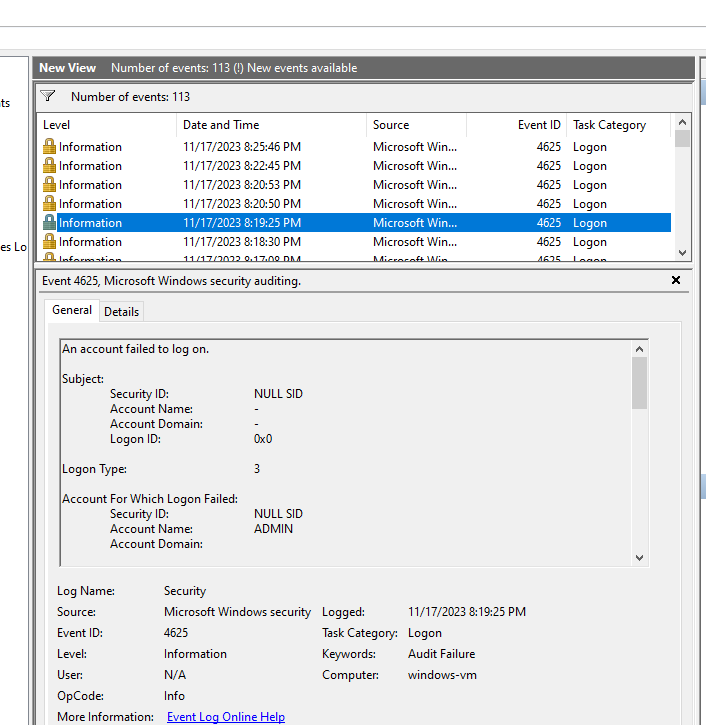
For server name we input the machine IP and we will now generate some failed logs.

Now we will try and SSH into the linux VM still using the ATTACK VM.

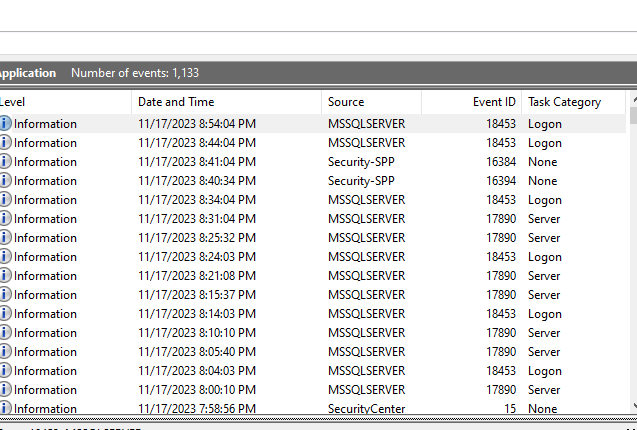


Just like this.

We will now take the perspective of an admin and login to the windows Vm to check the logs as well as the linux VM.

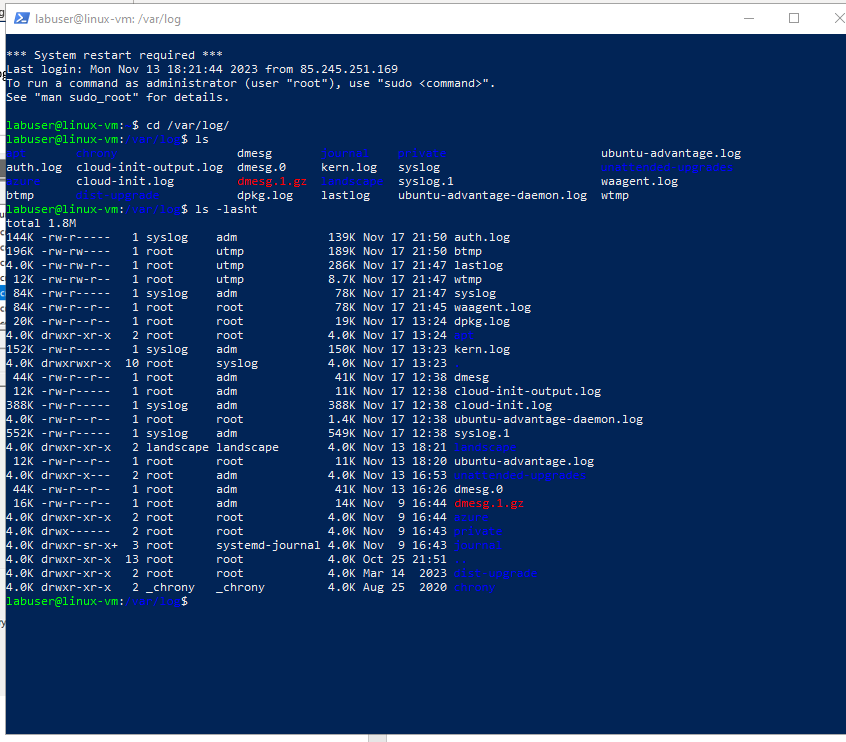


Let’s just take a note of these.

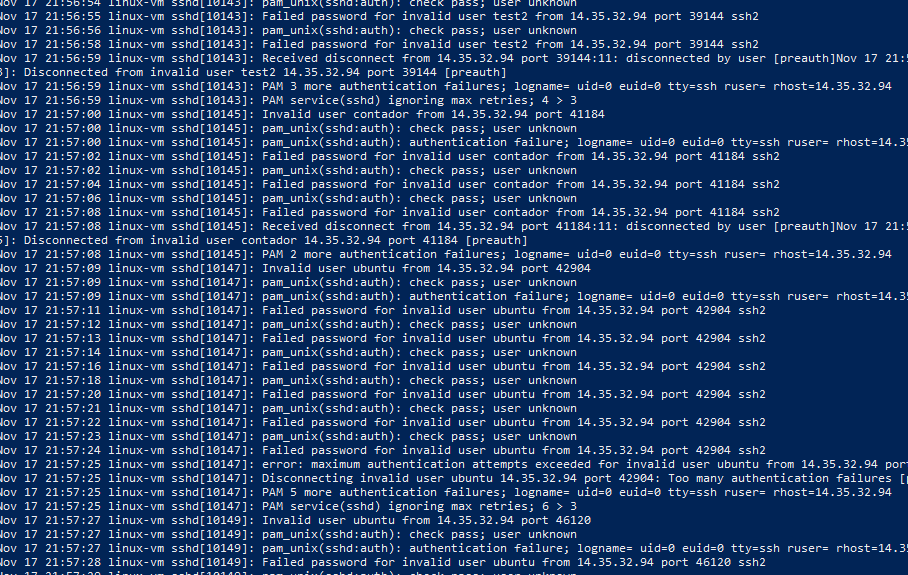


Here are the SQL logs.

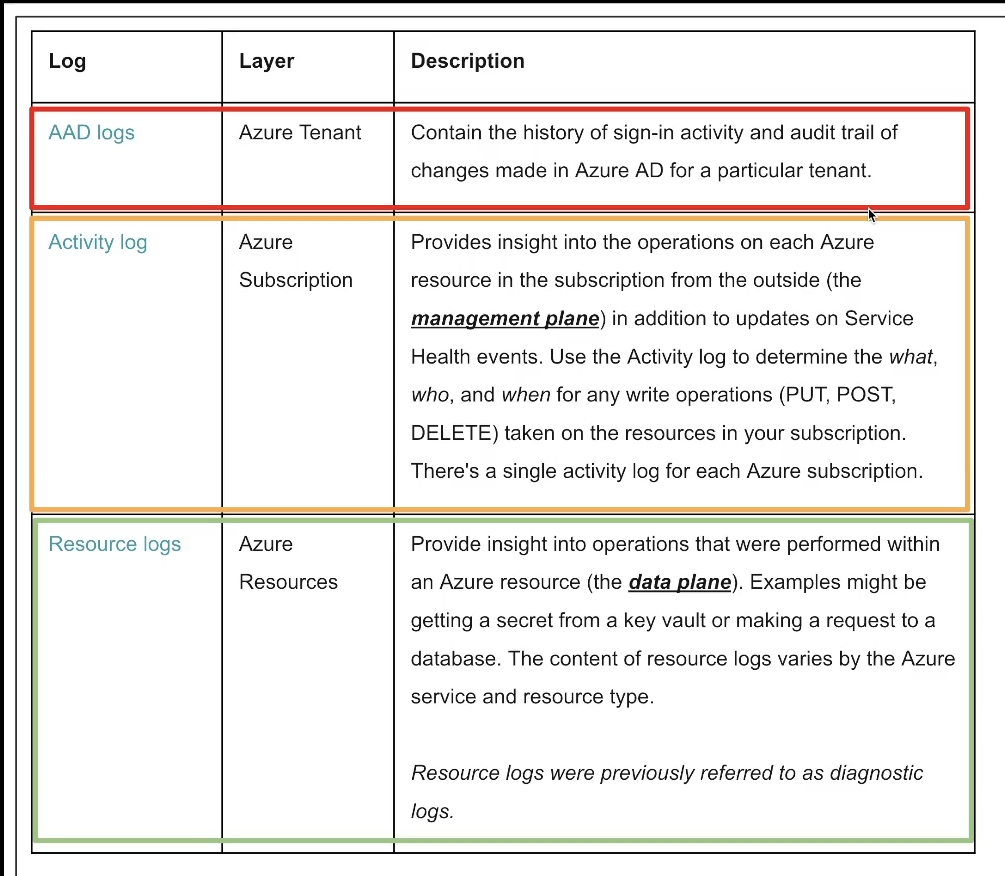
Let’s take notes of a couple of things like Ips, event IDs and more.



And finally, here is the linux logs.

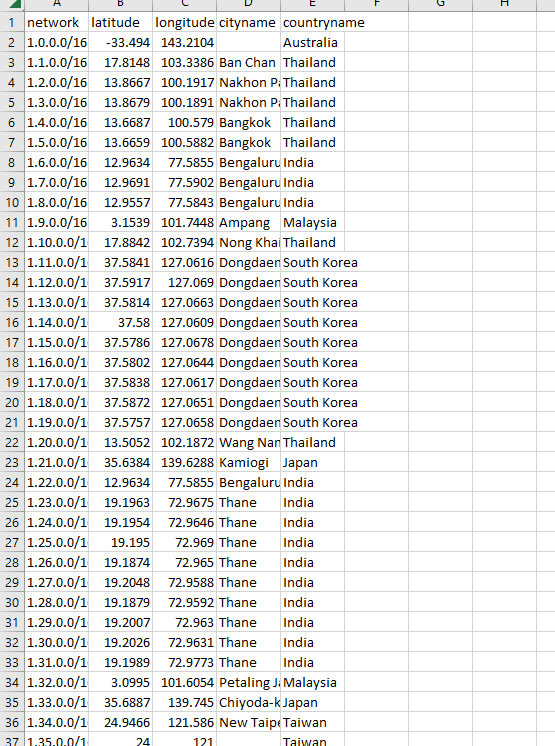


If we CAT the auth.logs we get the authorization logs from the machine.

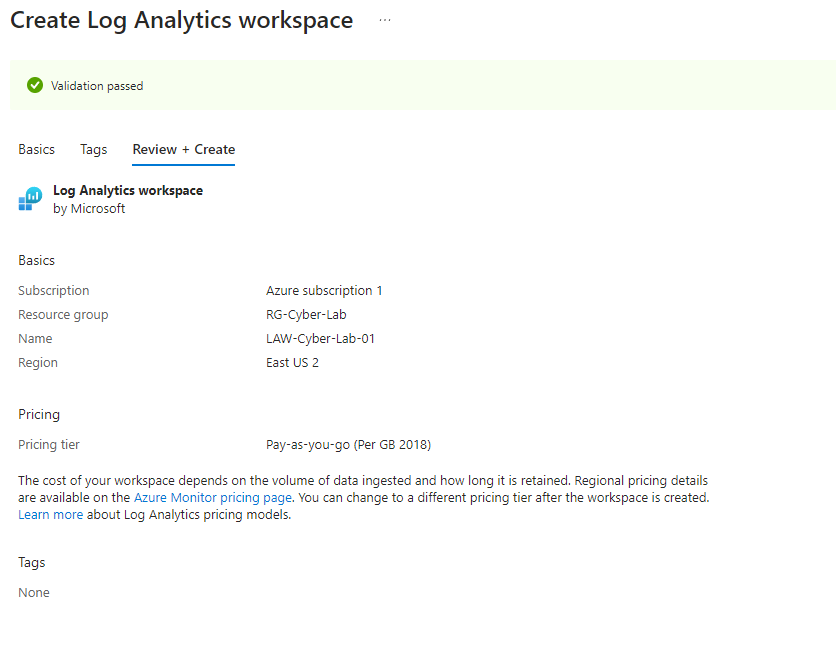


Here is an overview on how logs worn in azure, I am showing this because we will work mostly on logging for right now.

We will start by downloading a file to our pc, this is a list with IPs and locations that we can use to map our attackers later on a map. Here is what I used:

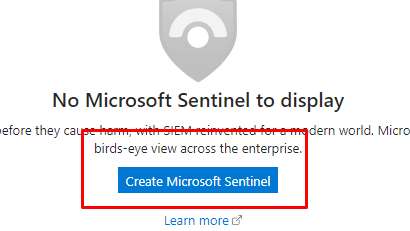


We will now create a Log analytics workspace (log aggregator)

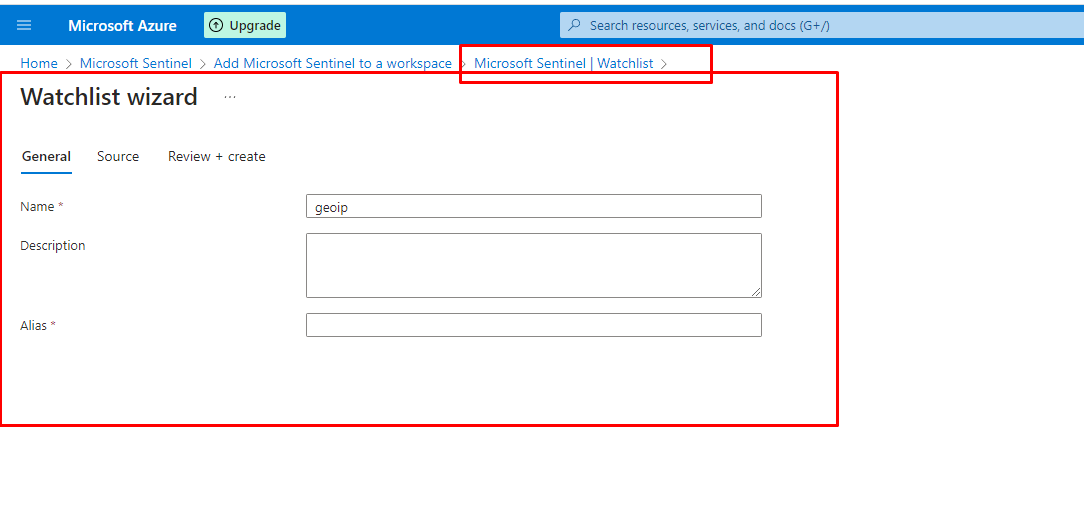


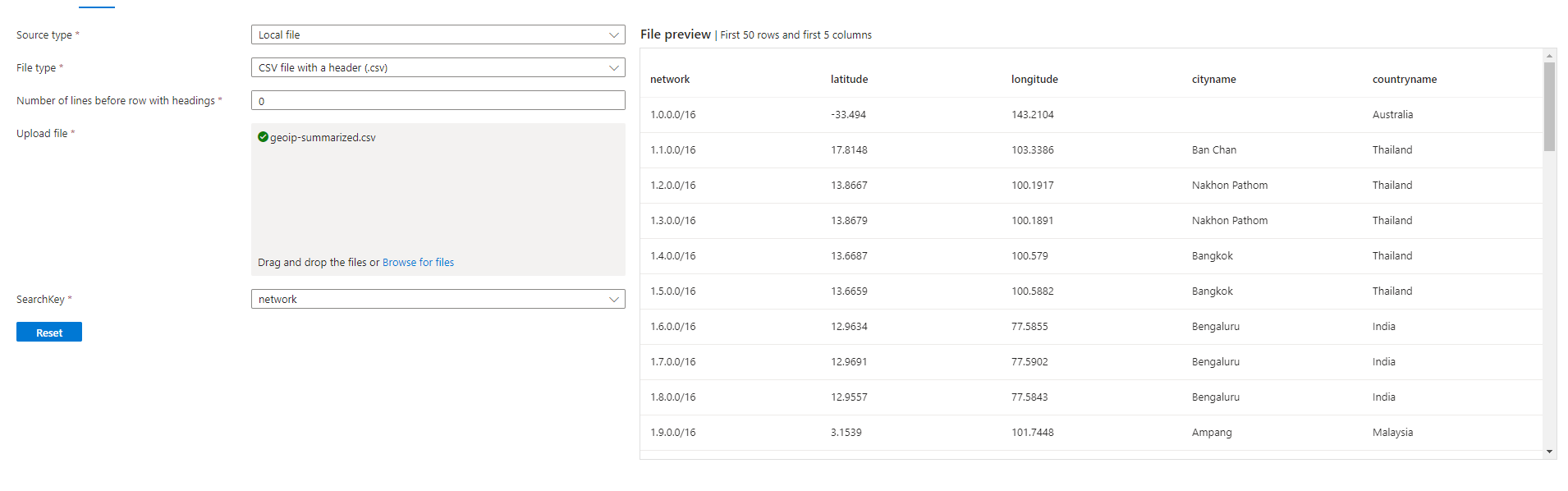
Here it is created.

We will now setup Microsoft Sentinel (SIEM) and we will connect it to out analytics workspace.

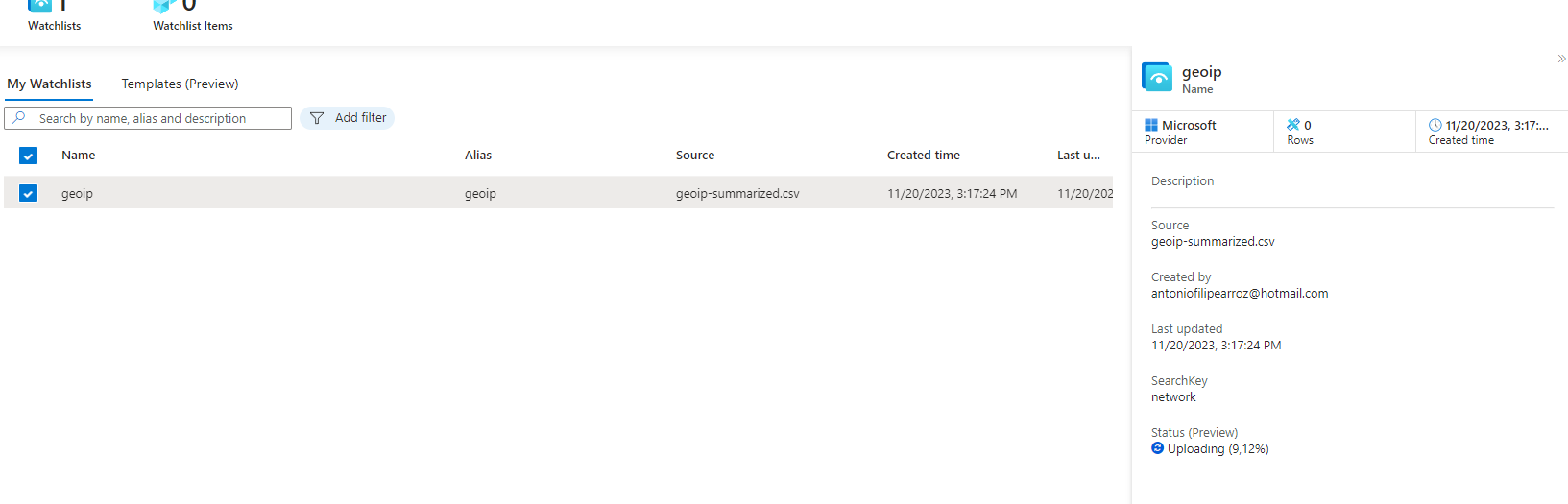


After sentinel is created, we will create the geoip watchlist.





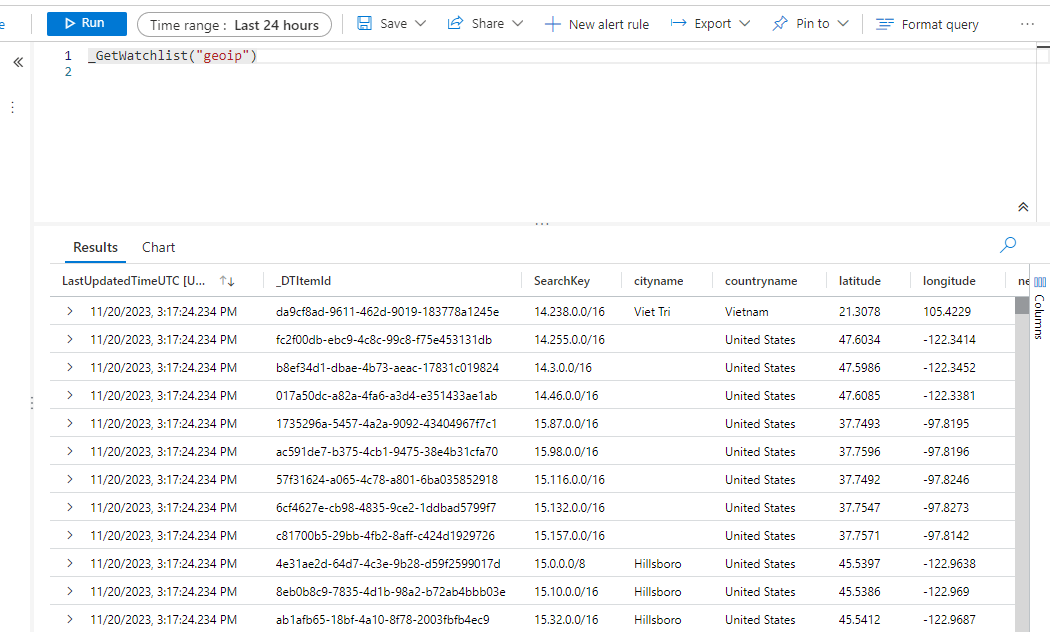
And here it is.



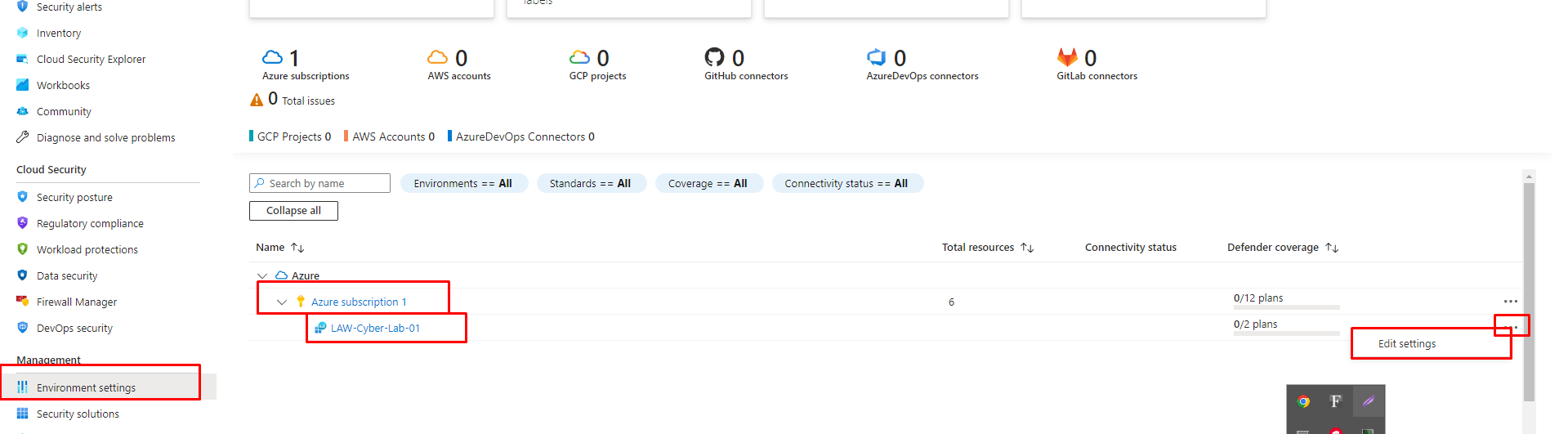
We now must wait for the csv to upload!

Next, we will go to our log analytics, and we will make sure something comes out when we query \_GetWatchlist("geoip")

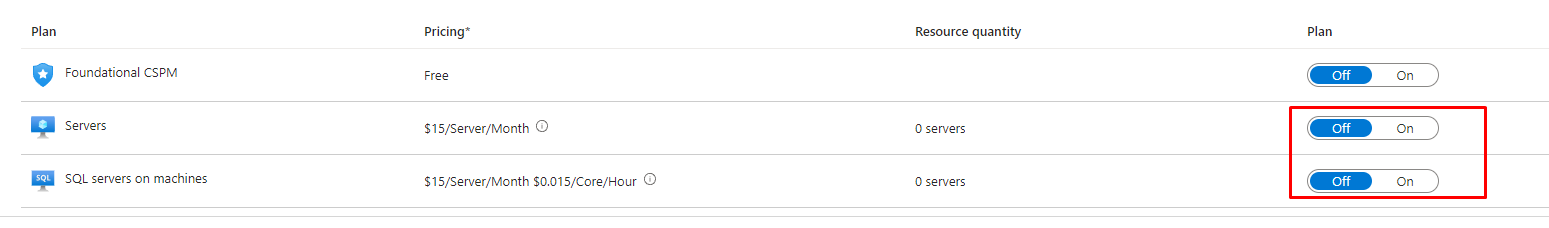
Something like this:



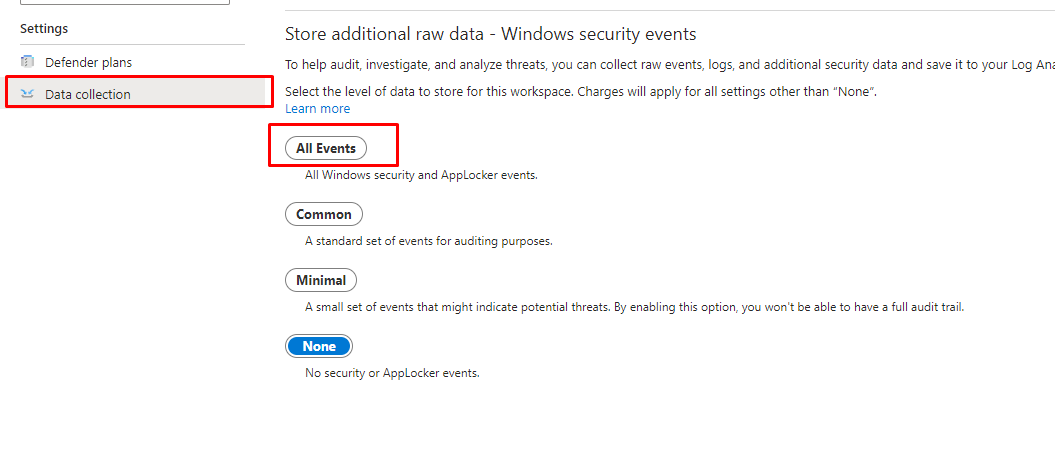
Next, we will enable Microsoft defender for cloud, it allows us to take logs from virtual machines and setwork security groups and ingest them in out log analytics workspace!



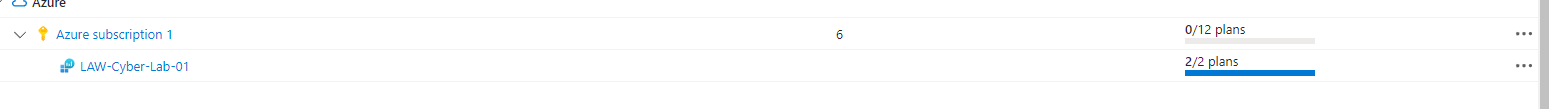
We will go here after getting to the azure page.



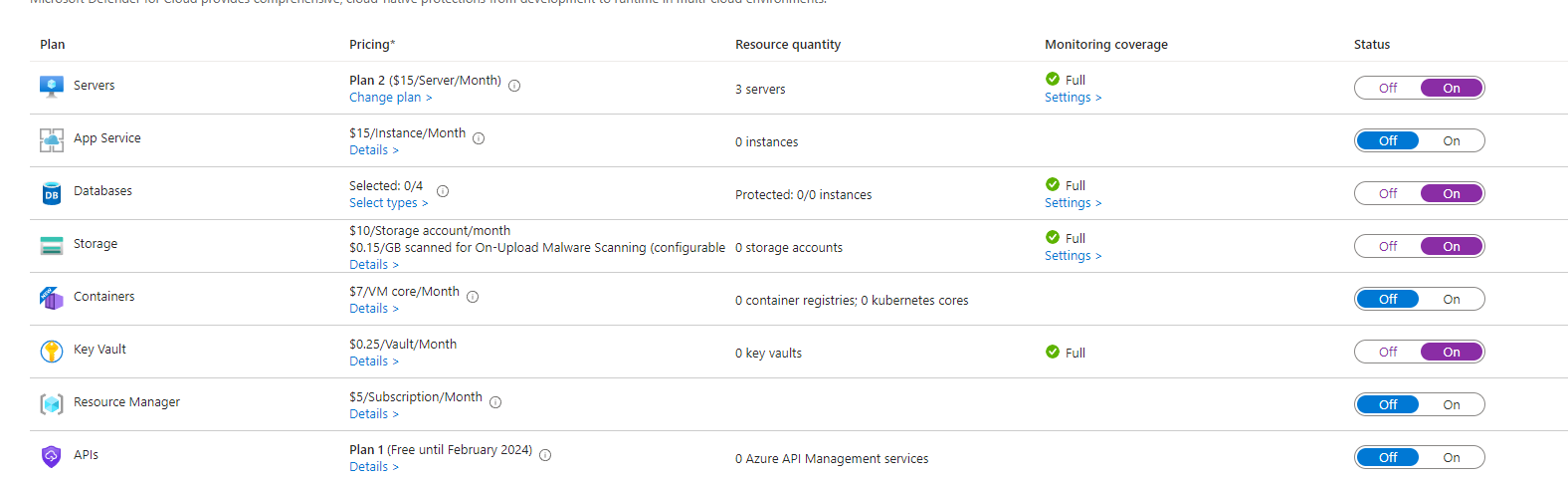
We need these 2 on!



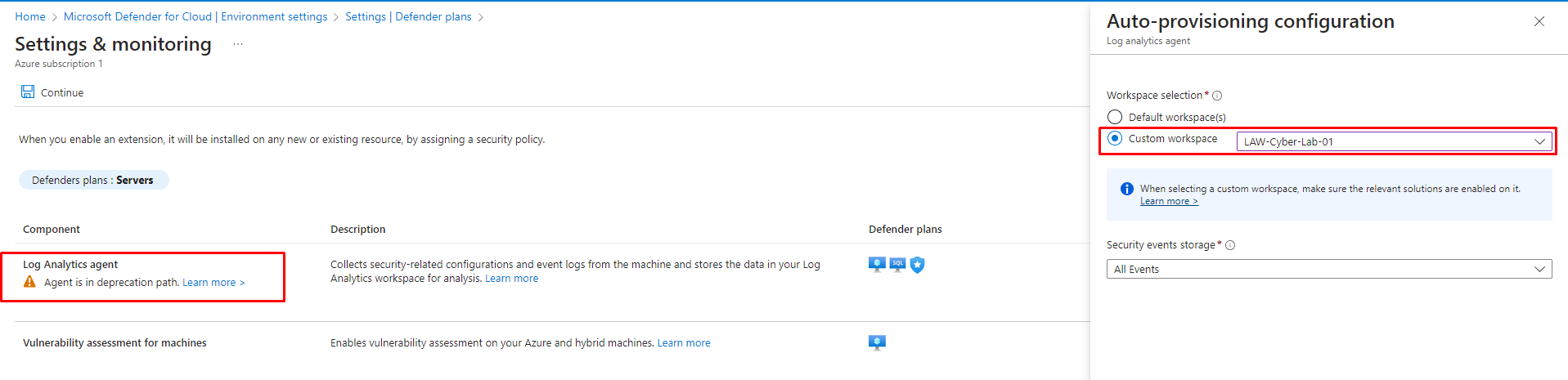
We also want to click on All events on this TAB.



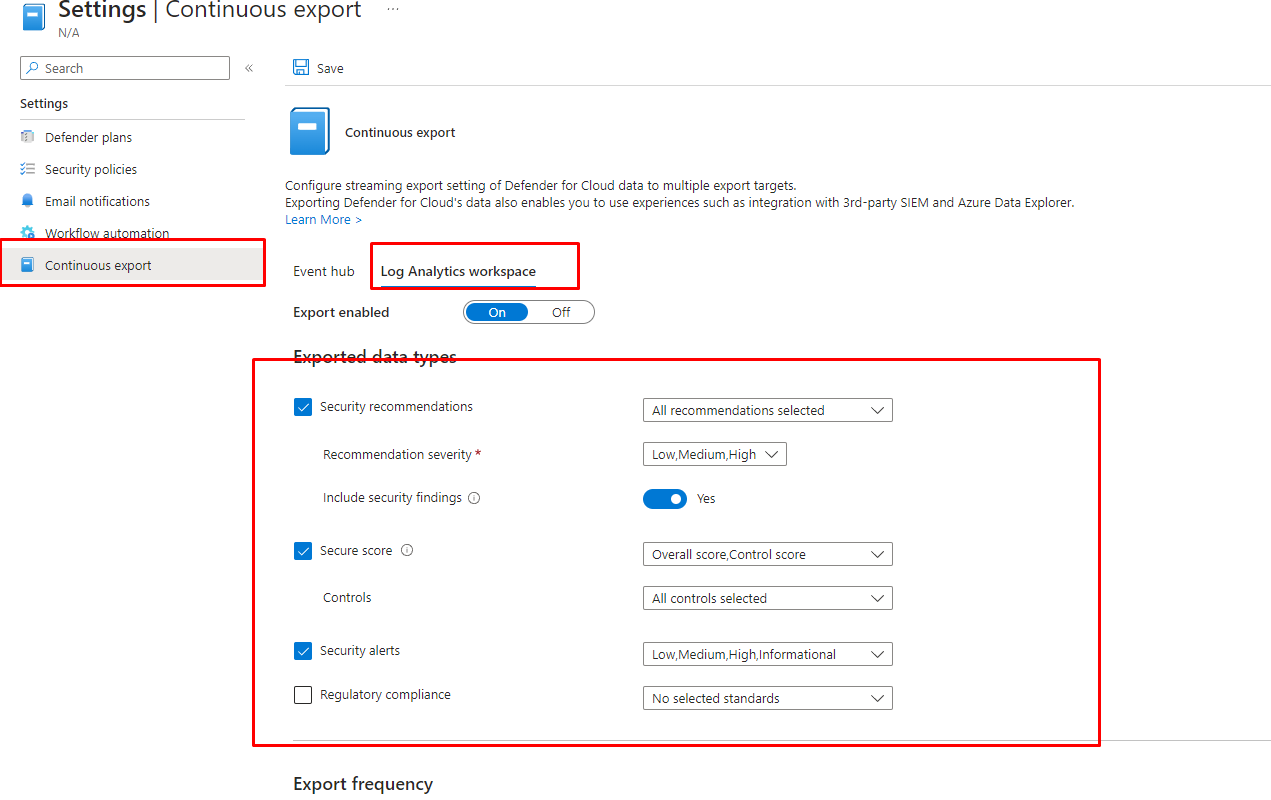
We also want to make sure it is set for the subscription as well!

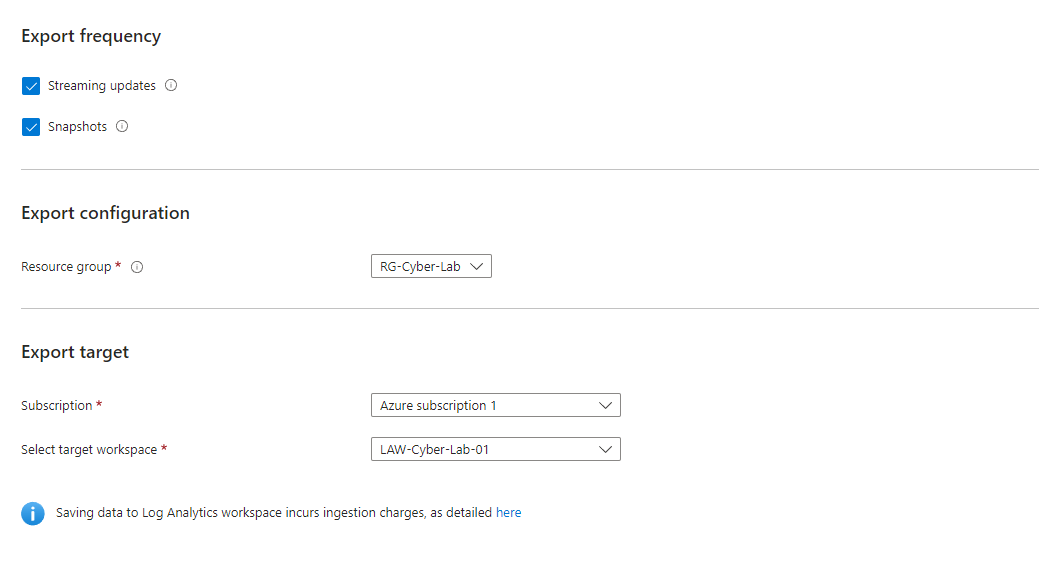


We want to make sure all of these are turned on because we will be creating some more resources like a key vault in the future.



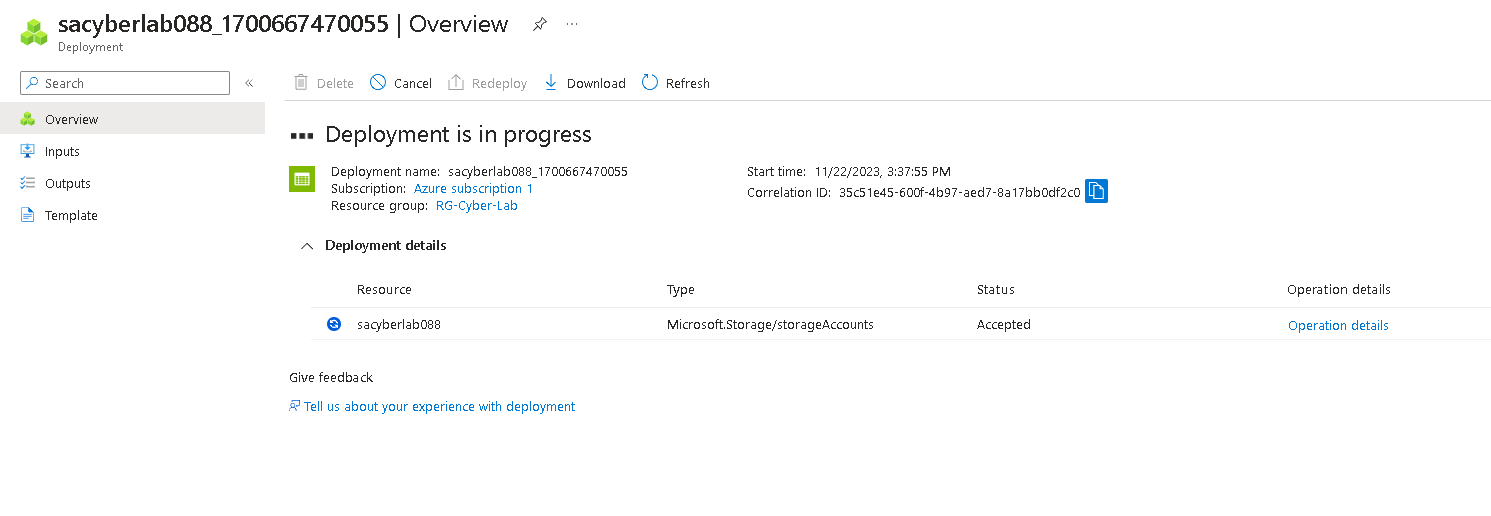
We also want to click settings under the servers and select for the logs to be sent to our custom log aggregation space and not a default one.



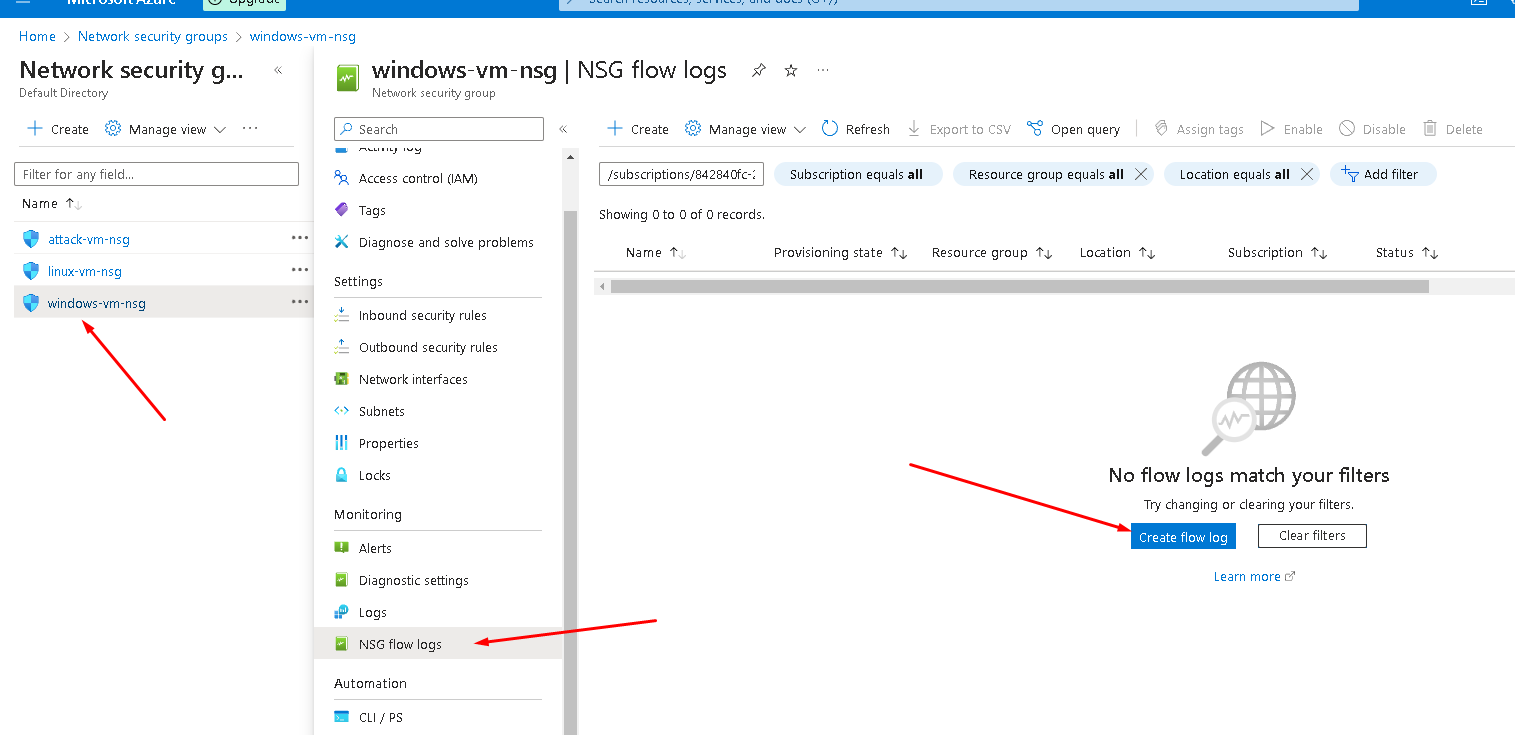


Next, we will send the logs using continuous export like shown above.

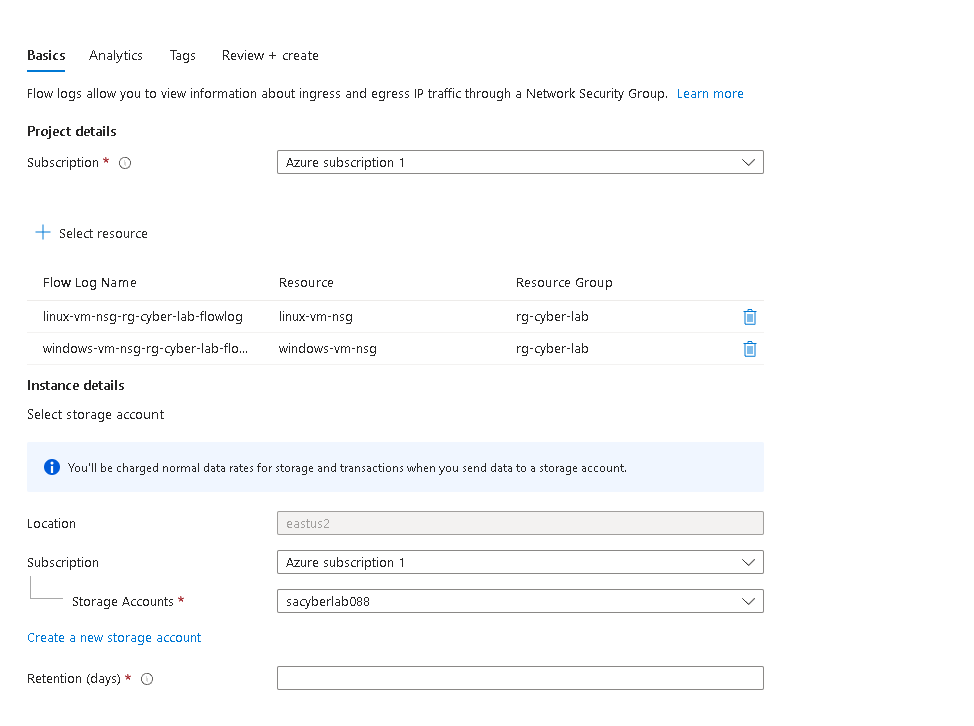
We will now create an azure storage account, it must be in the same region as the VMs, this will be used to store the NSG flow logs which we will create shortly!



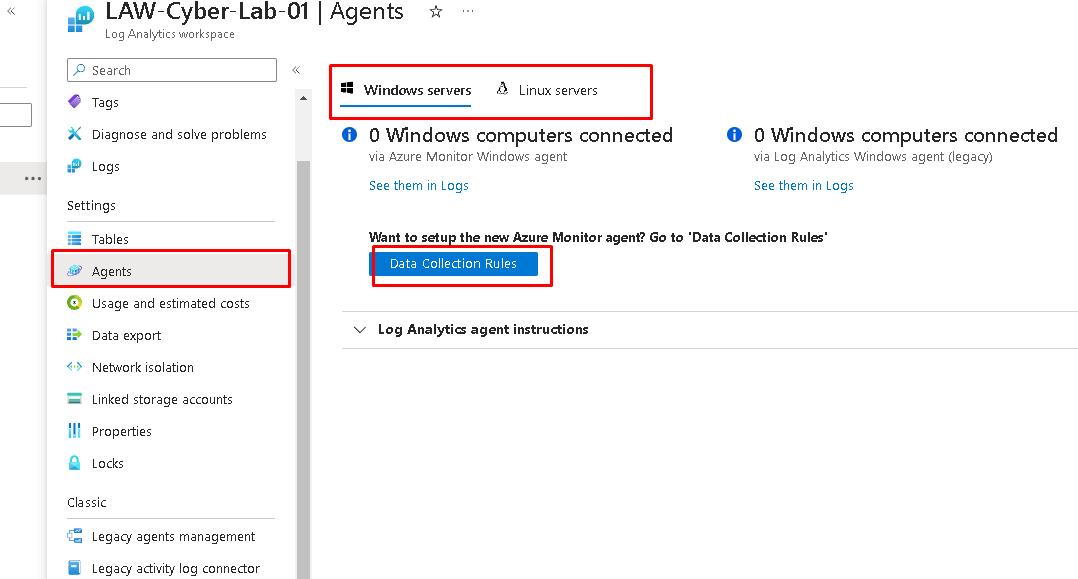
Created, next we will enable flow logs for both NSGs.

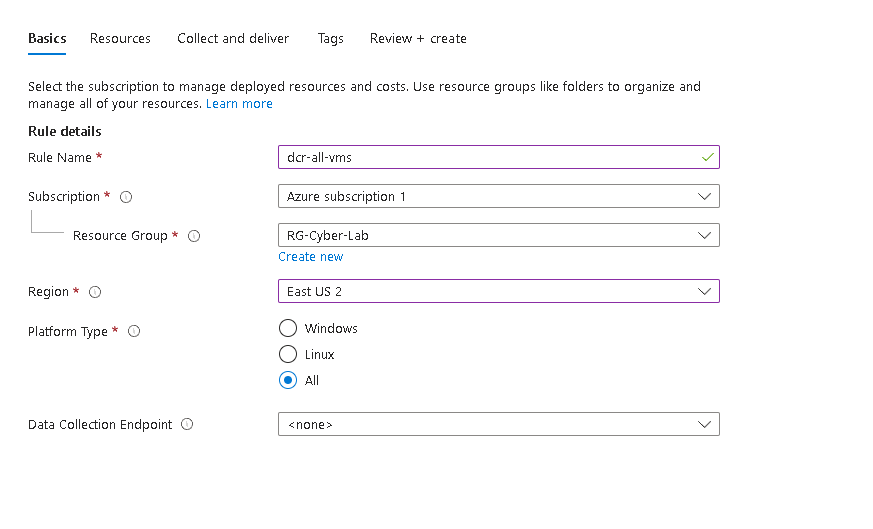


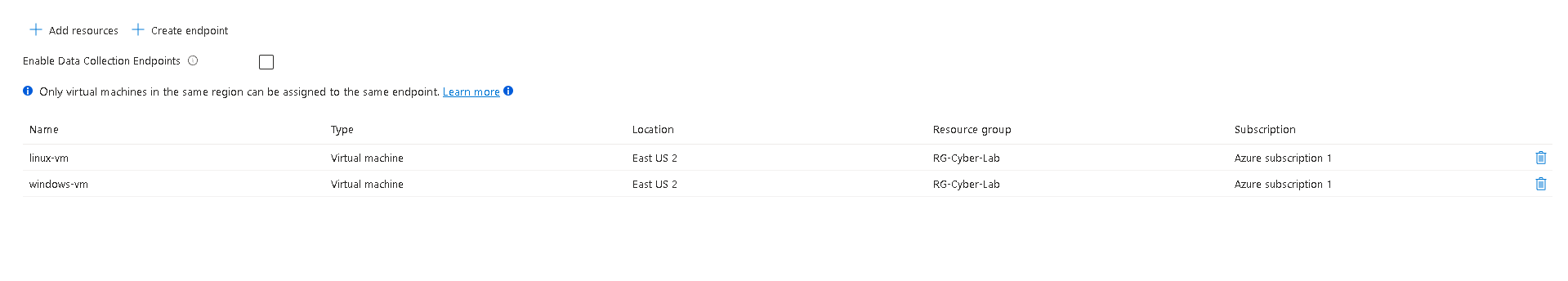
This is how we create the flow logs (:



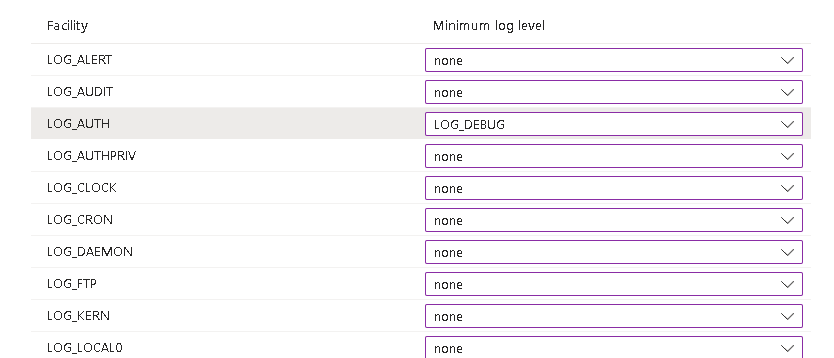
We will now configure the data collection rules within our log analytics workspace.



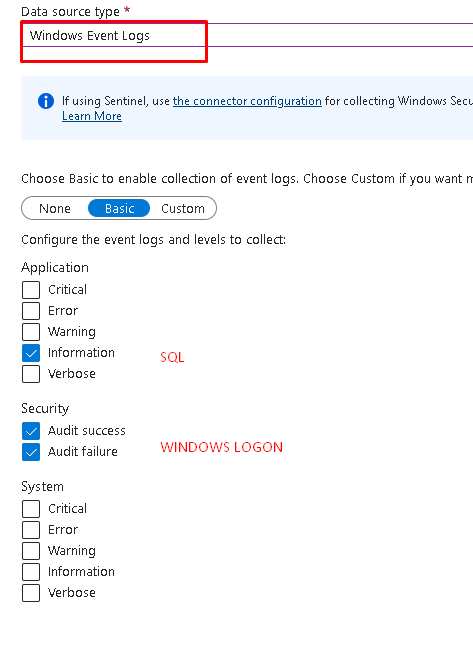
this will be the data collection rule for all our VMs so here is what we input.



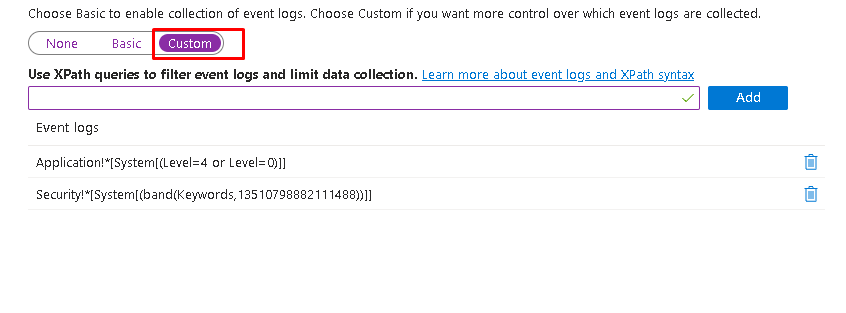
We will add both vms here.



For this lab we will only collect AUTH logs so rest should be set to none.



For windows we only need these.



We will actually set this to custom after creating and changing it to custom since we also want to record logs of people messing with the windows defender and more and we need to input 2 more lines here, here they are:

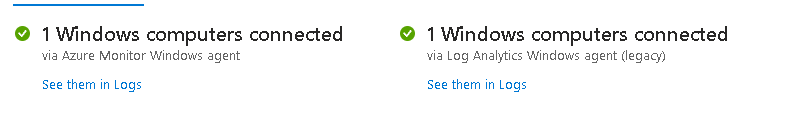
One of these is for malware and the other it for tempering detection!

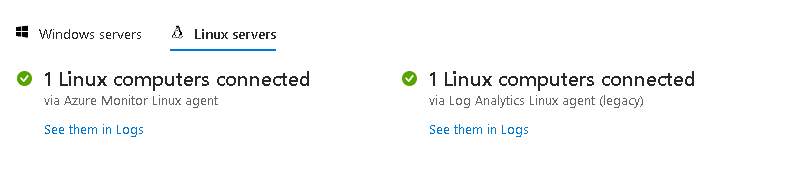
// Windows Defender Malware Detection XPath Query

Microsoft-Windows-Windows Defender/Operational!\*[System[(EventID=1116 or EventID=1117)]]

// Windows Firewall Tampering Detection XPath Query

Microsoft-Windows-Windows Firewall With Advanced Security/Firewall!\*[System[(EventID=2003)]]





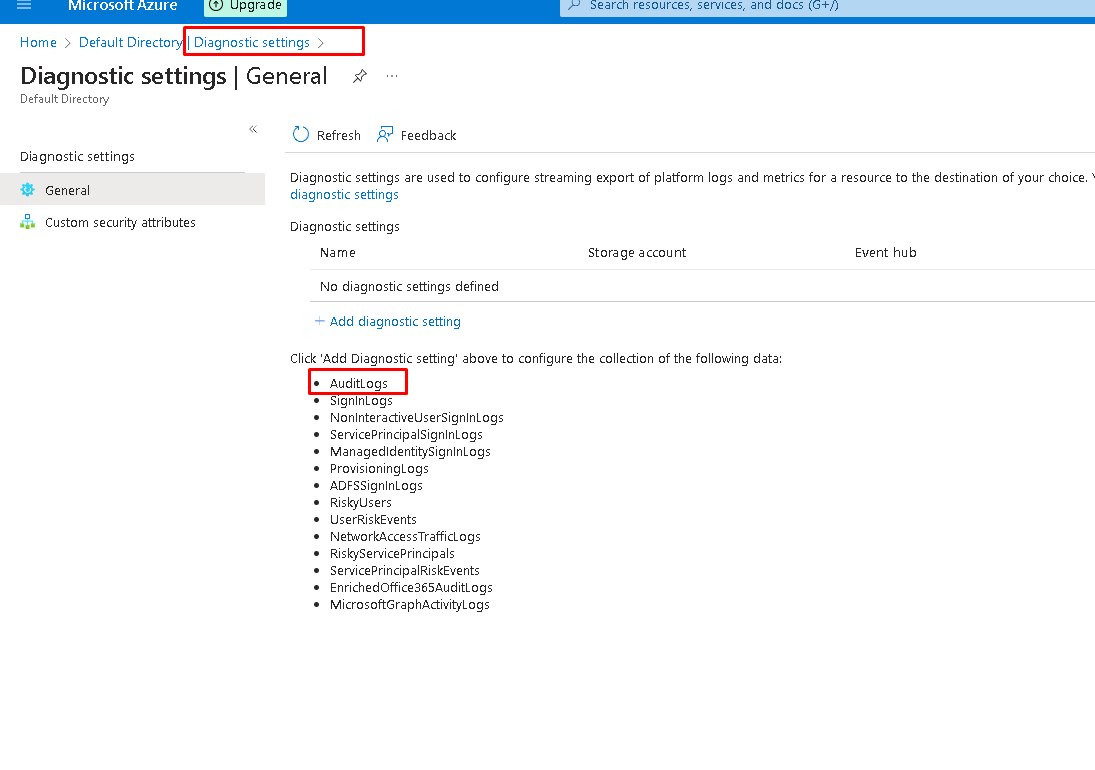
And now both machines should be sending logs!



And just to make sure we can query some logs and we can see that they are actually being forwarded!

Now that we have logs from out 2 VMs we will start ingesting logs from our Microsoft active directory as well!

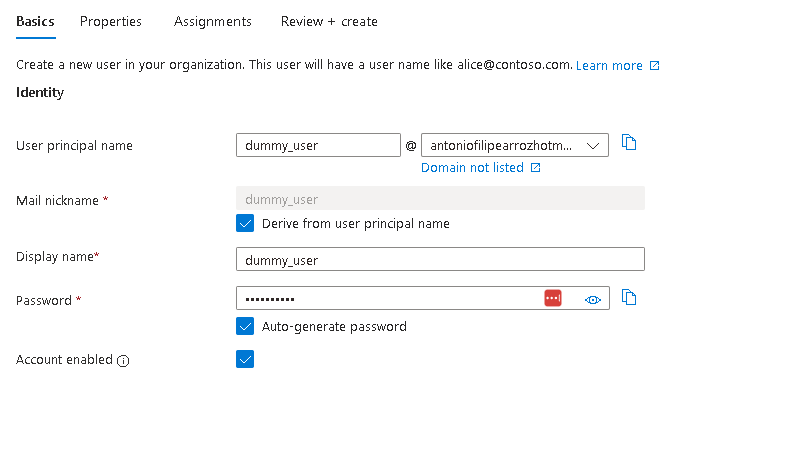
Let’s start by:



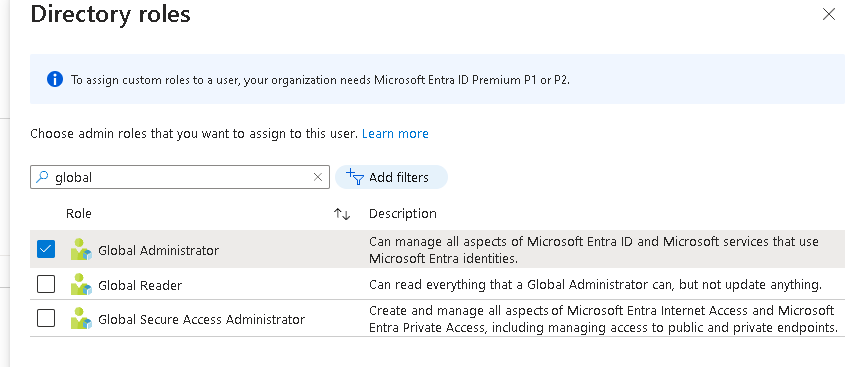


And this is all we need.

We will now create a dummy user in active directory and see if we can generate some logs!

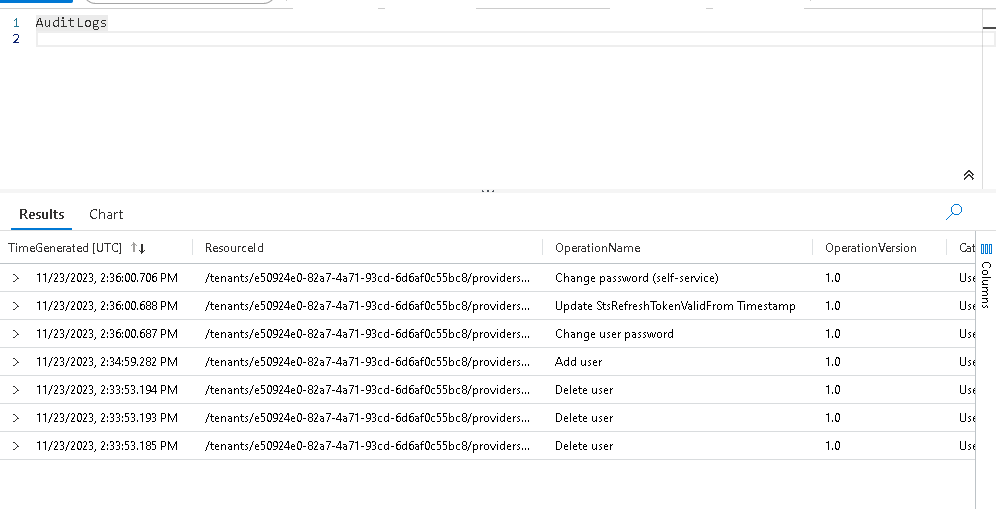


We logged it to this user using a private tab and this should already generate a log, we will now assign the role of global admin to this user and this should generate another log as well.



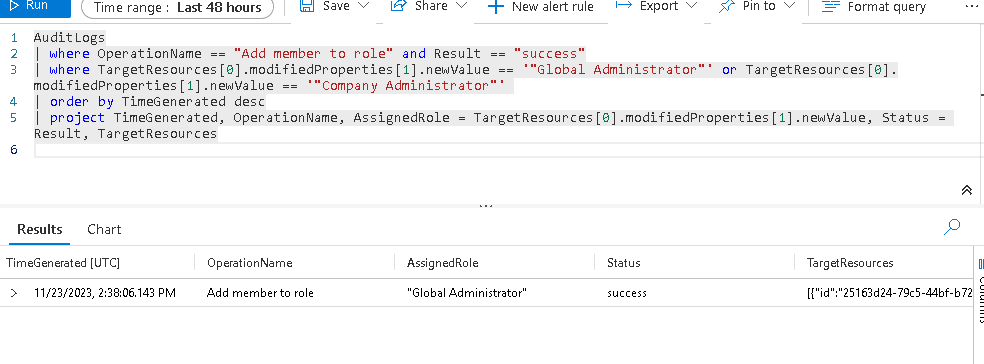
We will now delete the dummy user and this should generate one more log.

And Boom,



Many logs!

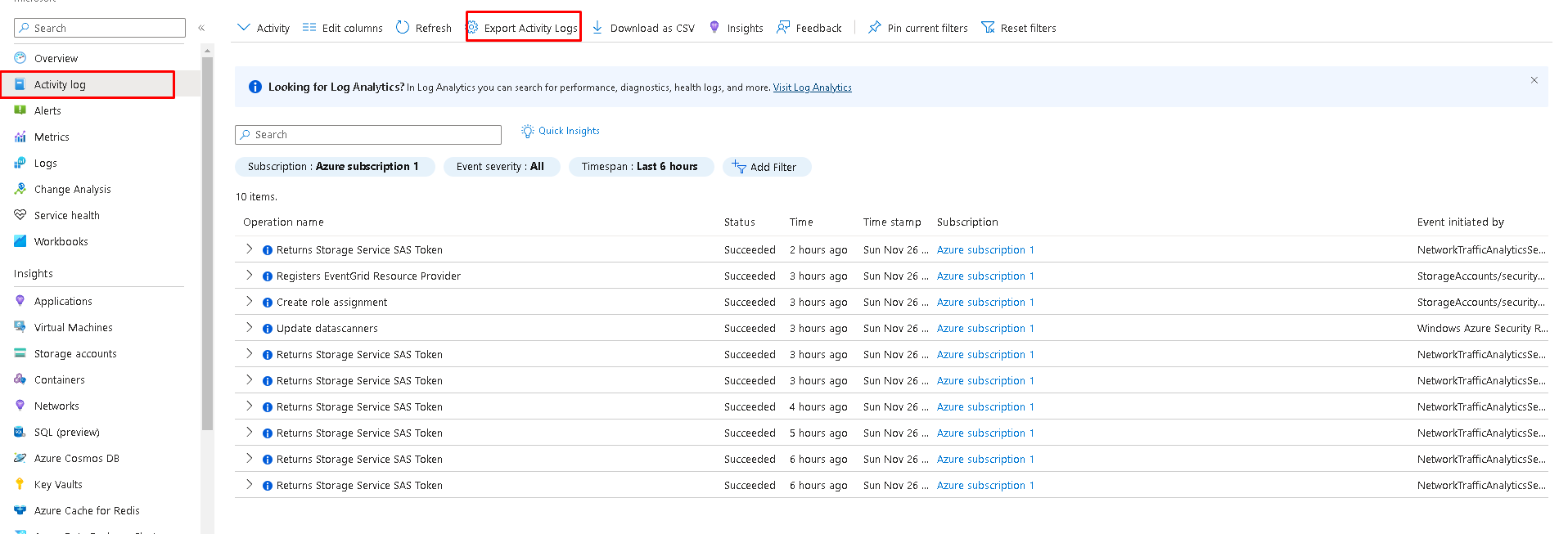
Adding user, deleting user, everything is in there and more should come in the next few minutes.



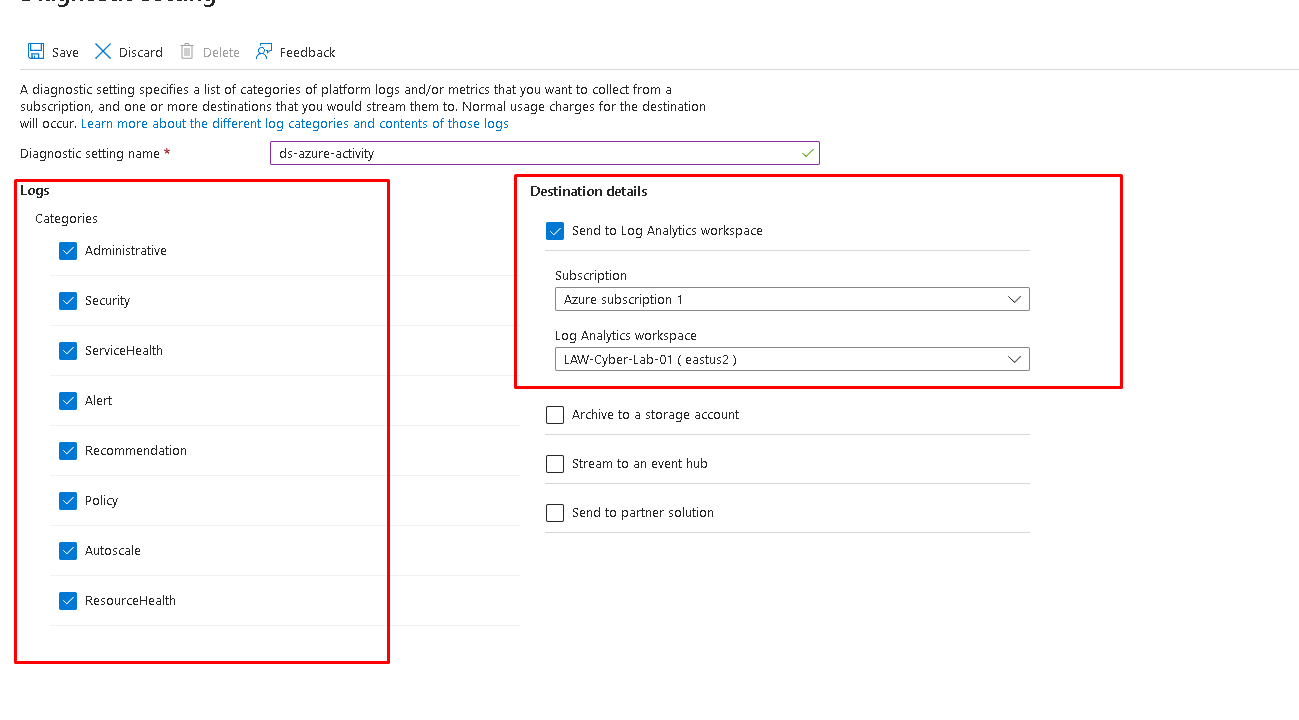
This query searches for adding someone as a global admin and like we expected we get a result!

All we need now for logging is subscription level logging, let’s start with activity logs!

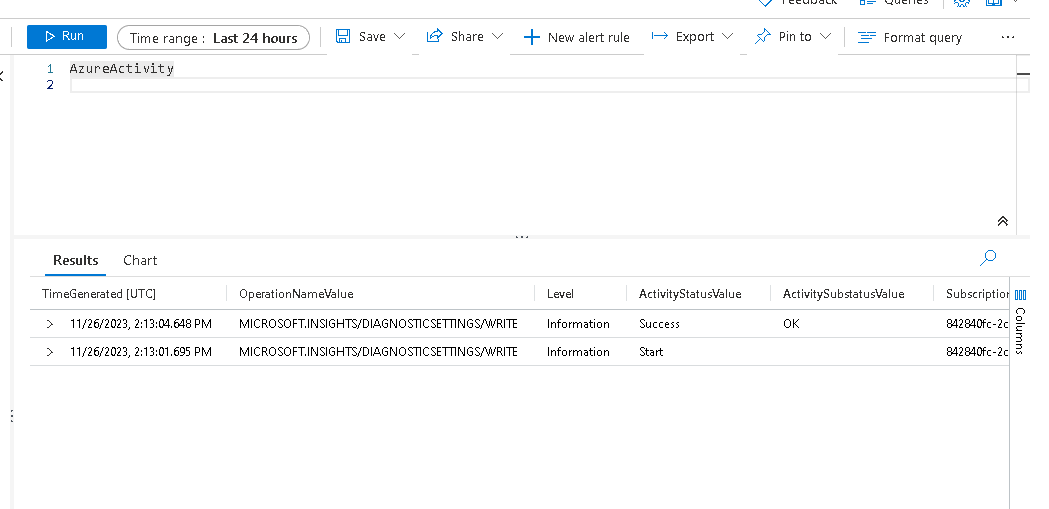
Like creating resources, deleting resources, etc.…



We first went to azure monitor, after that we need to click activity log and we will export those logs.



We will send everything to out log analytics workspace!

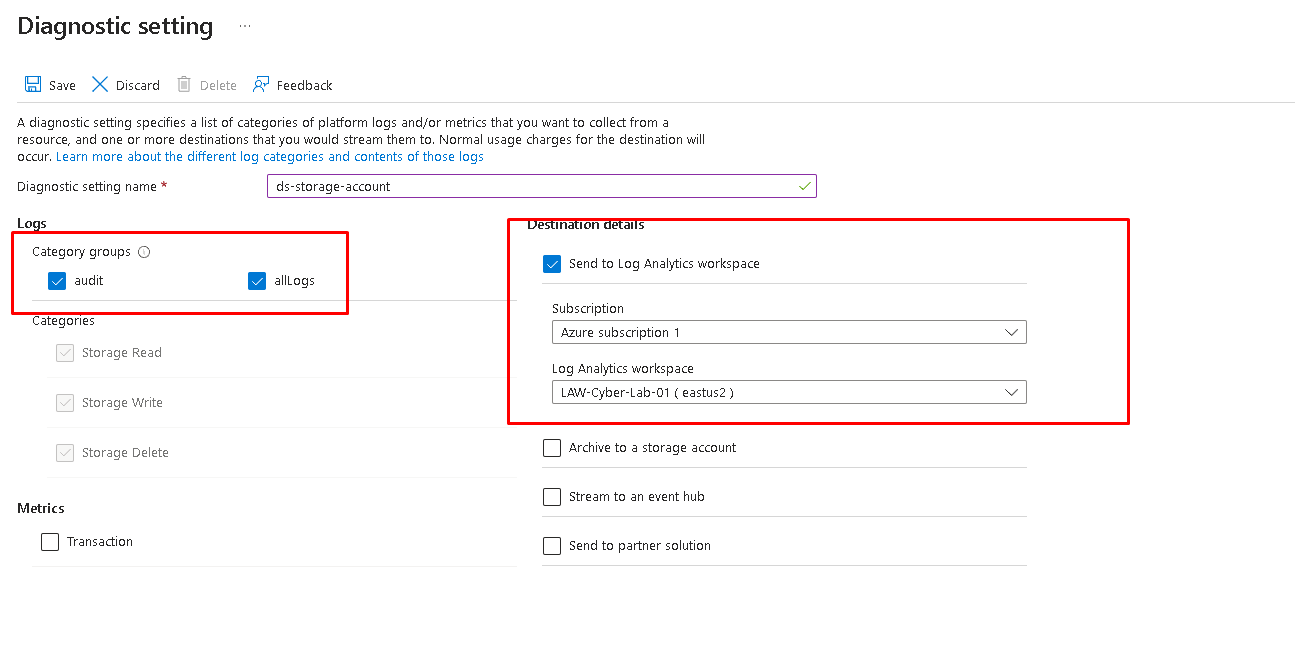


If we test, we have some logs already!

Lastly, we will setup resource logging, we will create a key vault and a blob storage!

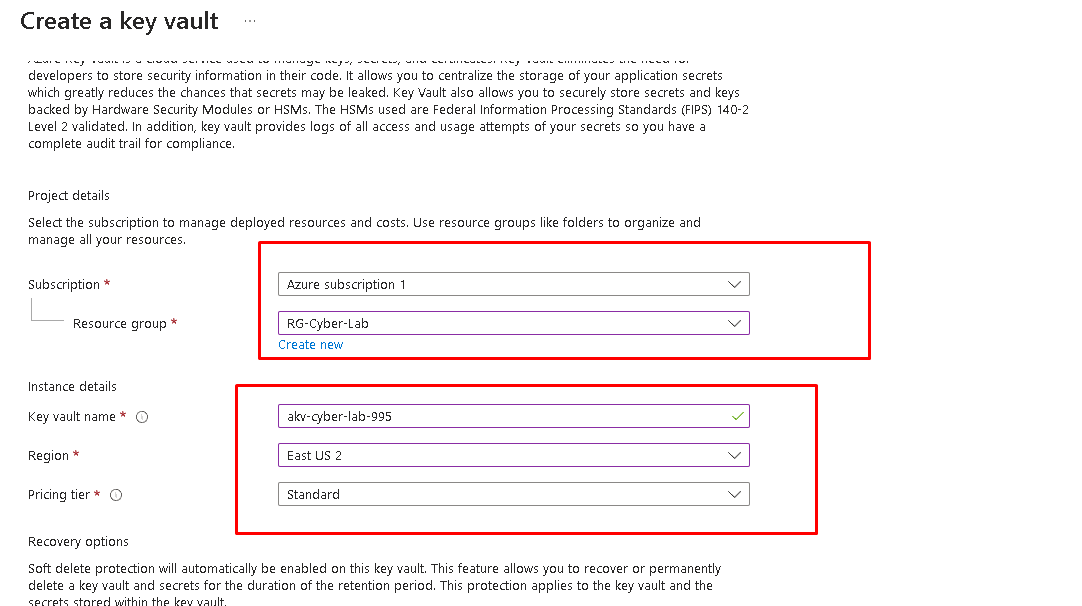


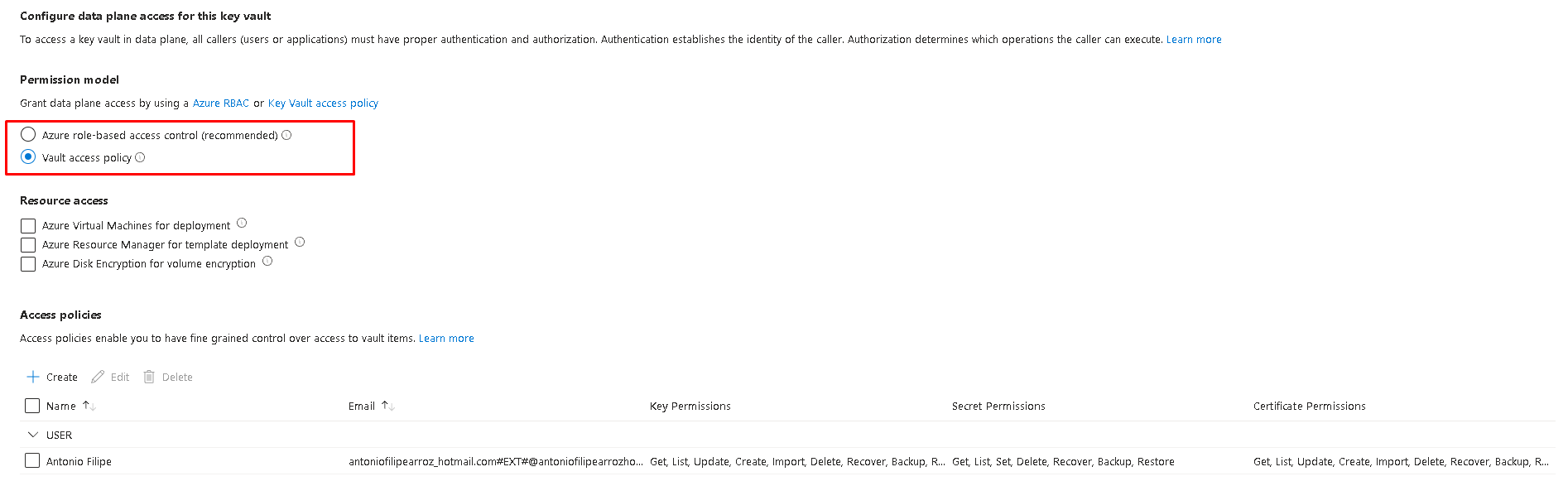
We now have to go to our storage account, we scroll down to diagnostic settings and next we will enable logging for the blob portion of the storage account.



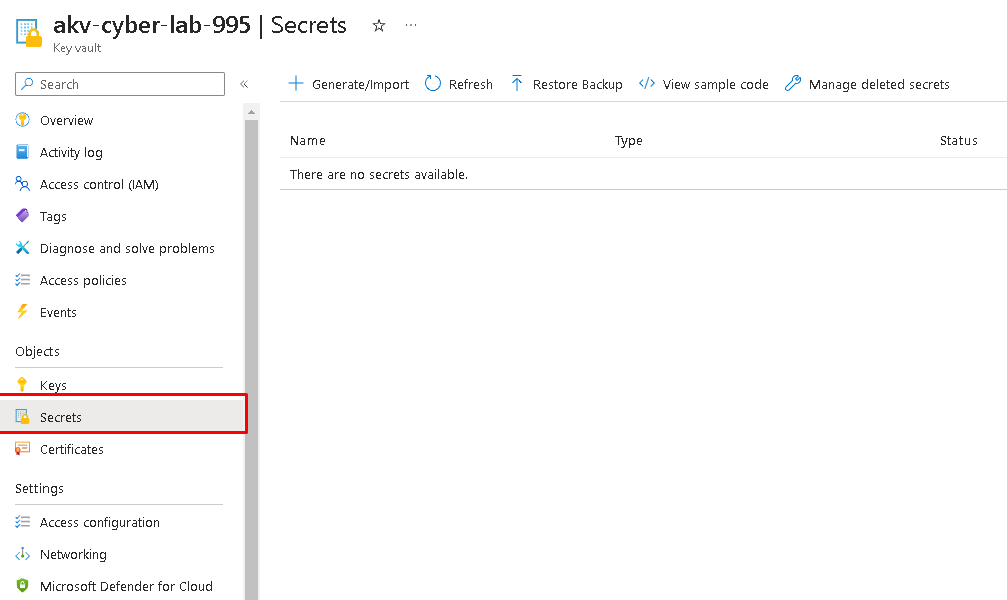
Same things as the others, we will send all logs including audit logs to our analytics workspace!

Let’s create our key vault now!





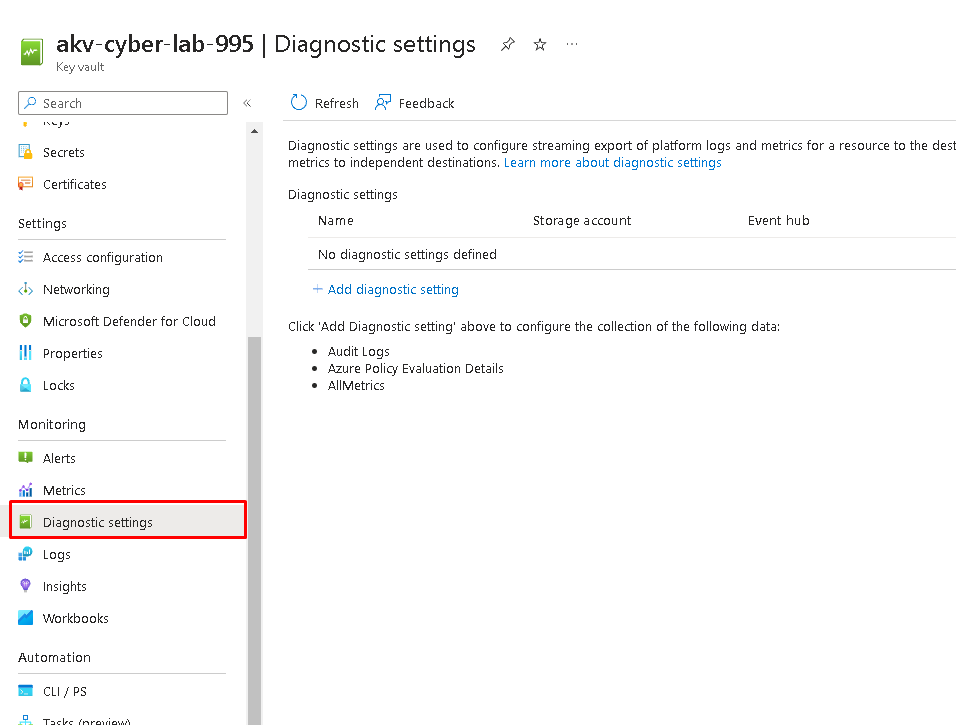
We want to change our vault policy to the one above!



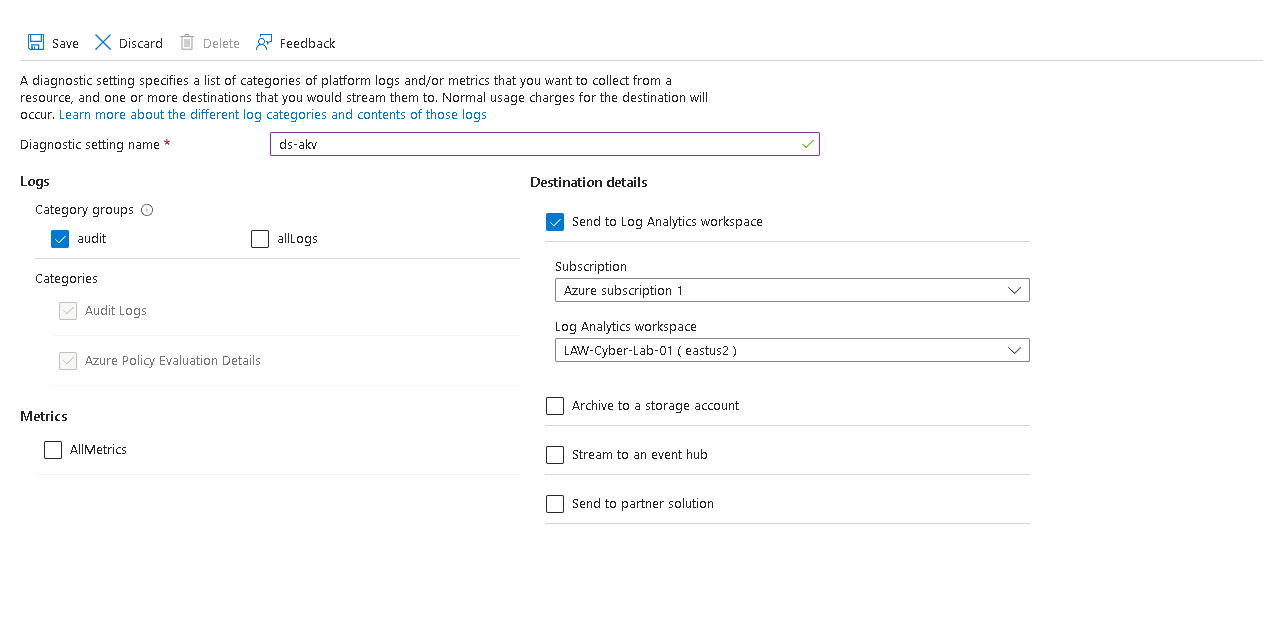
After creating the key vault, we will create a “secret” which is like a password we want to store.



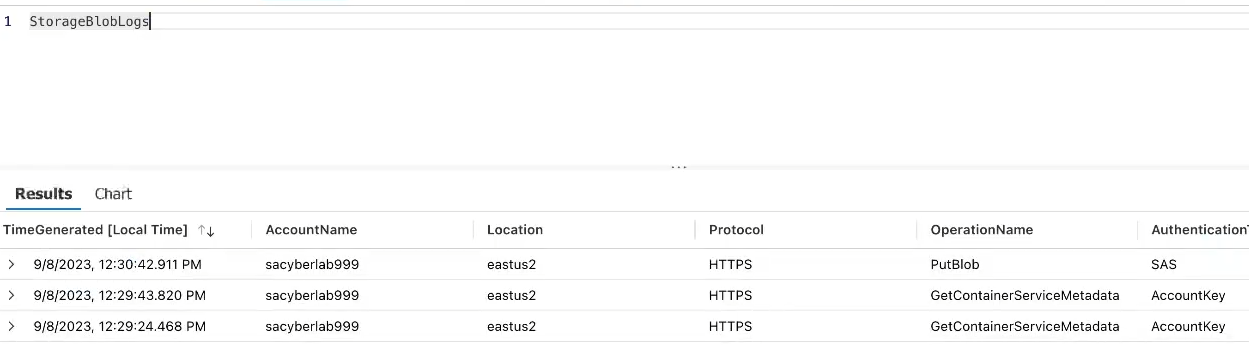
Just create a random secret.



And like always, let’s create Diagnostic settings so we can start generating some logs.



Like this.



And everything is being logged, the blob storage as well as the key vault.

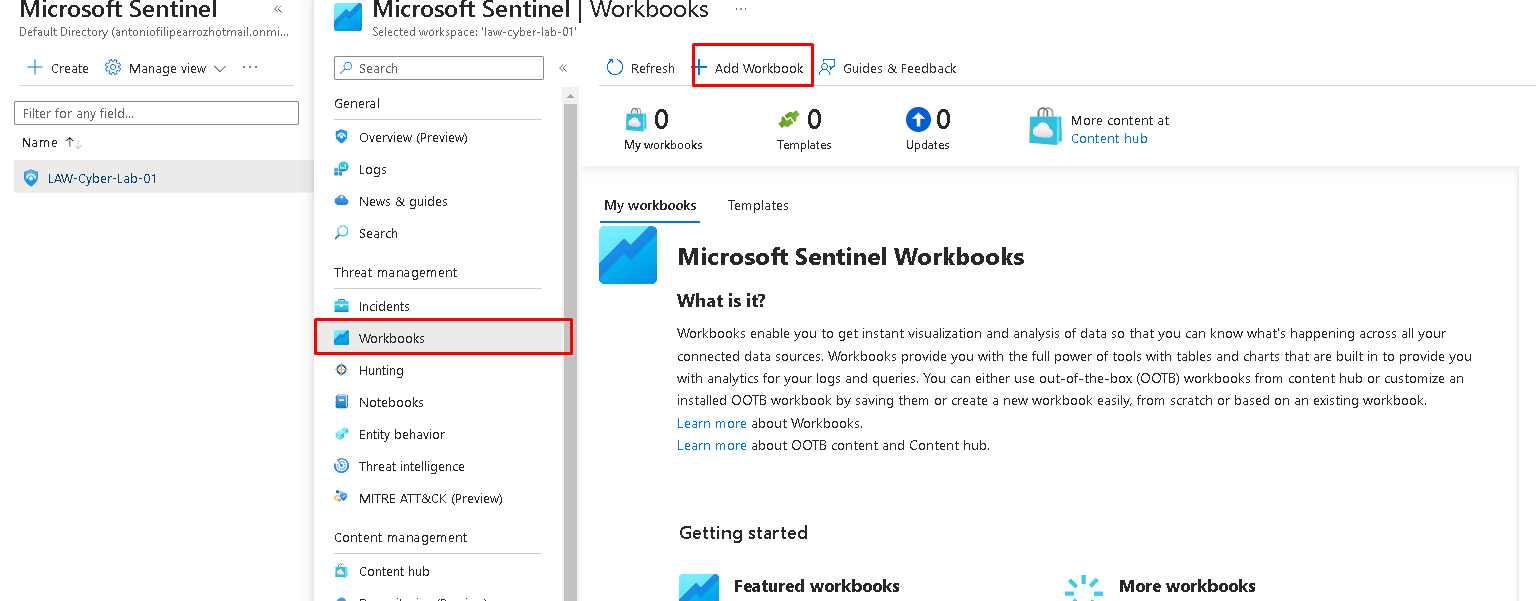
We will finally start working on our SIEM, creating incidents, creating a threat map, harden our environment, work on incidents,etc…

We will start by creating 4 MAPS, one for windows VMs: RDP, SMB / GENERAL AUTHENTICATION FAILURES

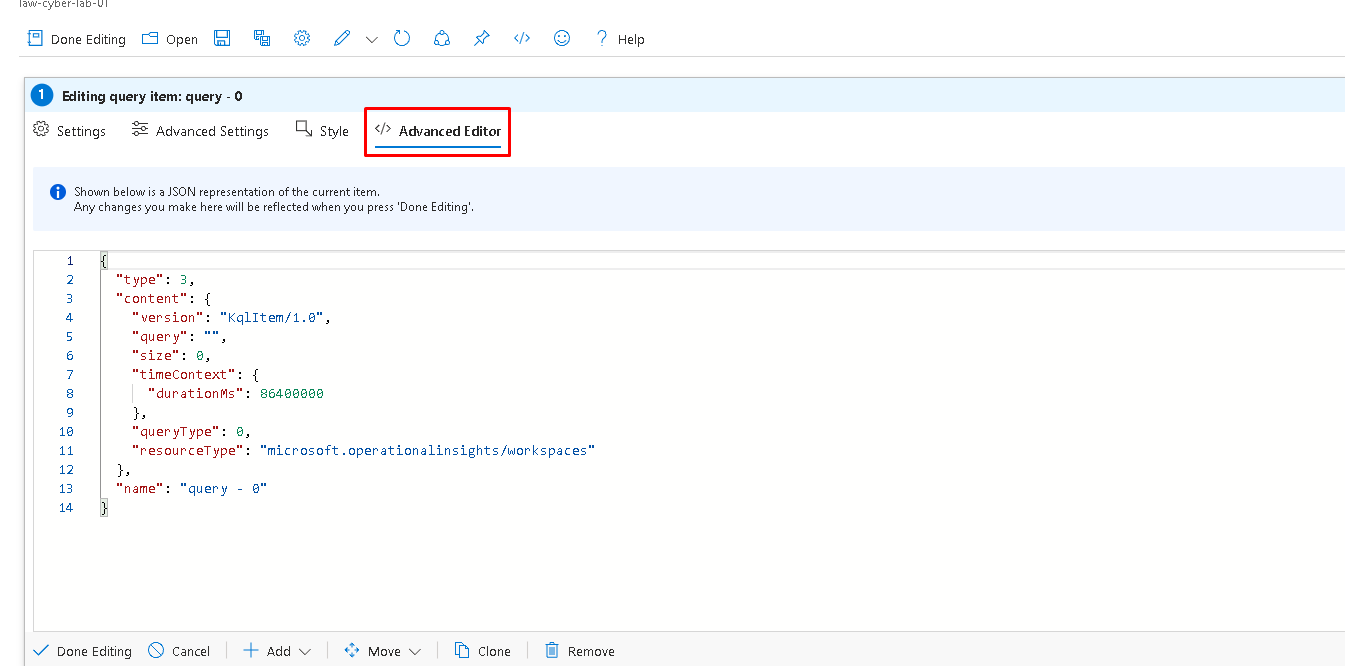
LINUX VMs: SSH authentication FAILURES.

AZURE SQL SERVER: authentication FAILURES.

NETWORK SECURITY GROUP: MALICIOUS FLOWS



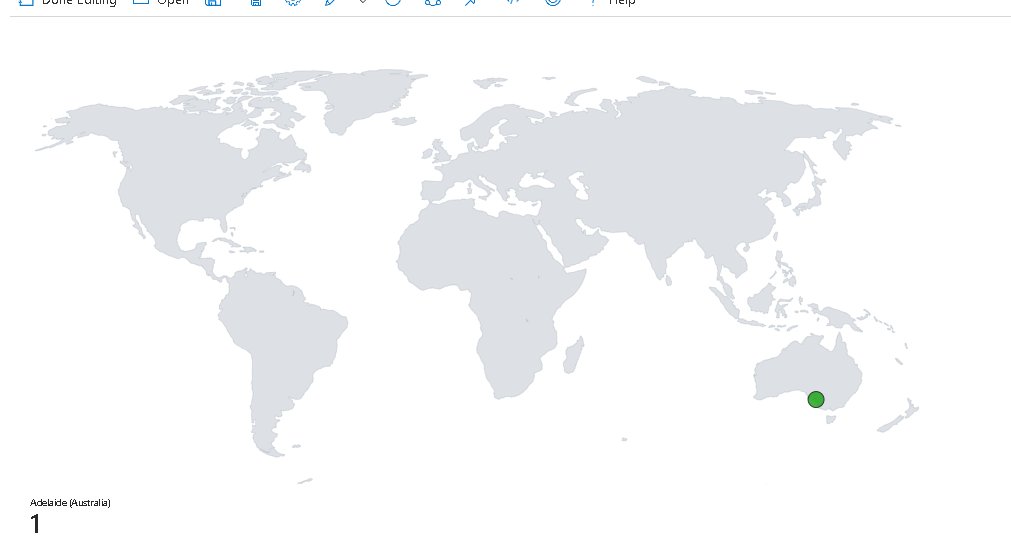
In sentinel, maps are workbooks, so let’s create one.



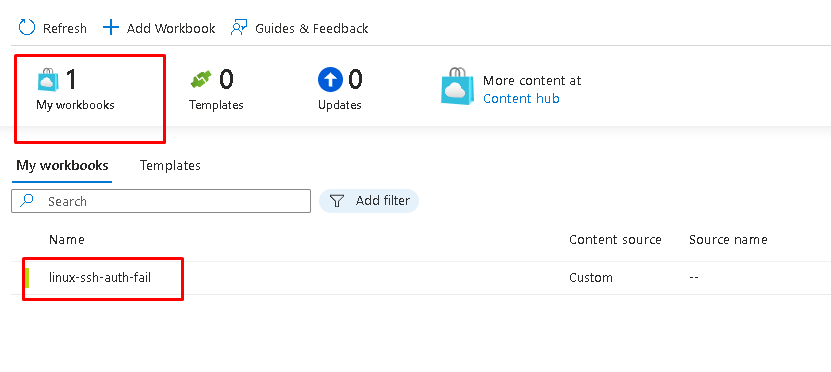
We will edit the default workbook that comes with sentinel and use the advanced feature to create our maps.



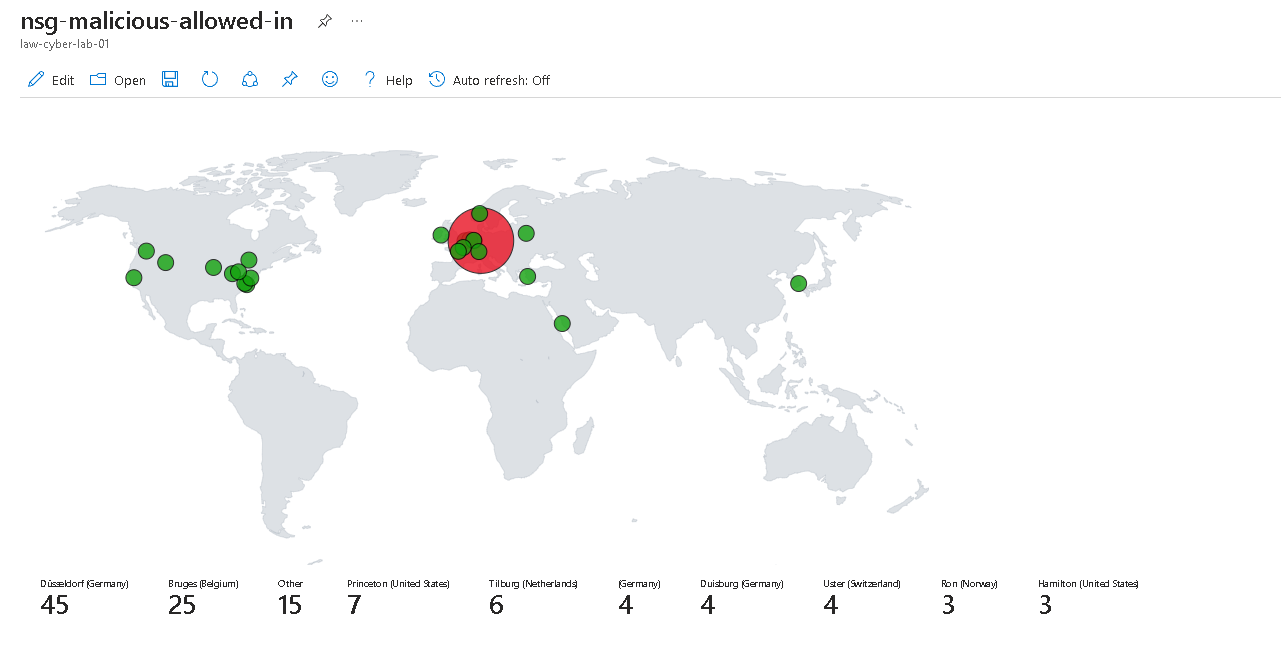
Here is a snipper of what we will be using for the SSH logs/MAP.



And boom! Here is the map, since my VM has been off most of the time, almost nobody has tried to break in yet.

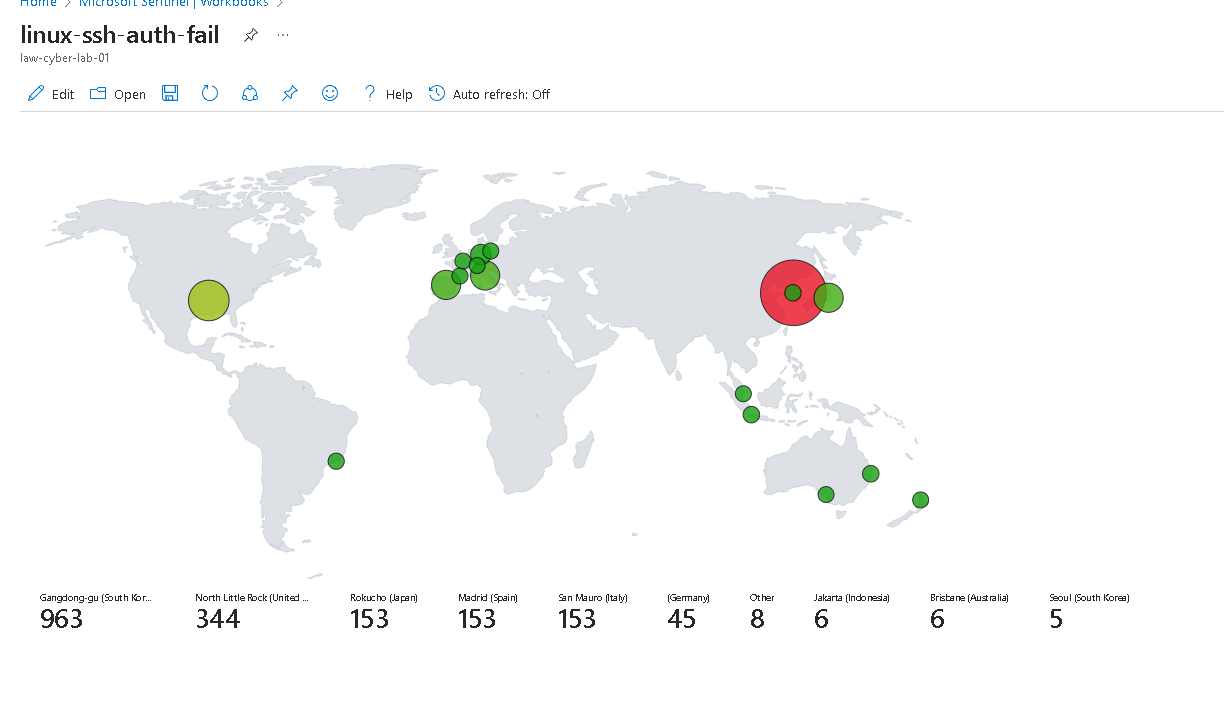


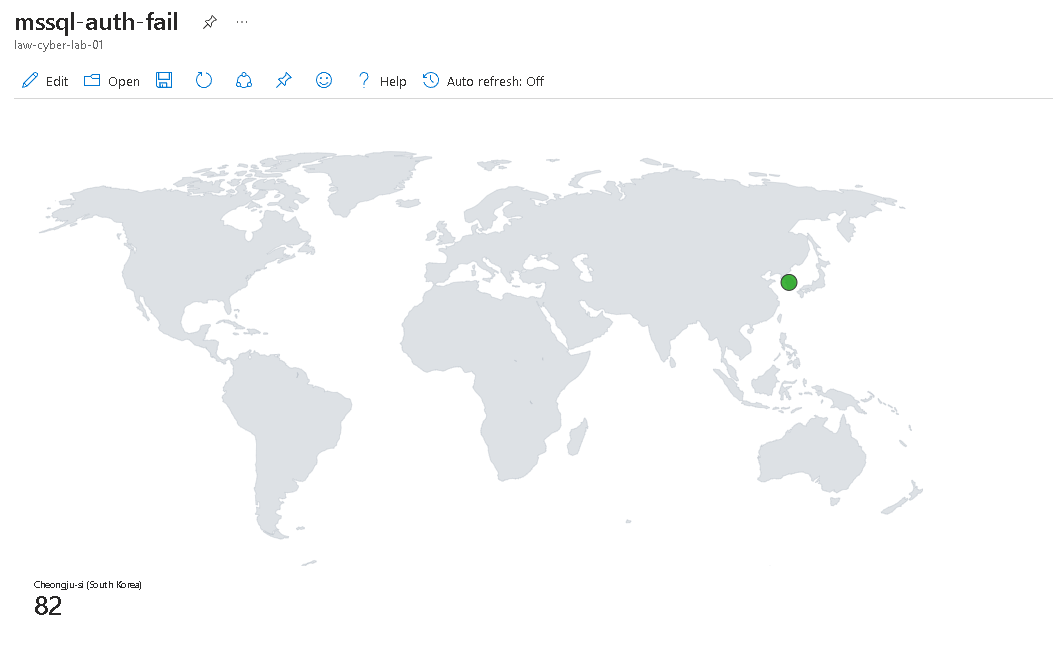
Now that we have 1 created, I will go ahead and create the other 3 Maps.

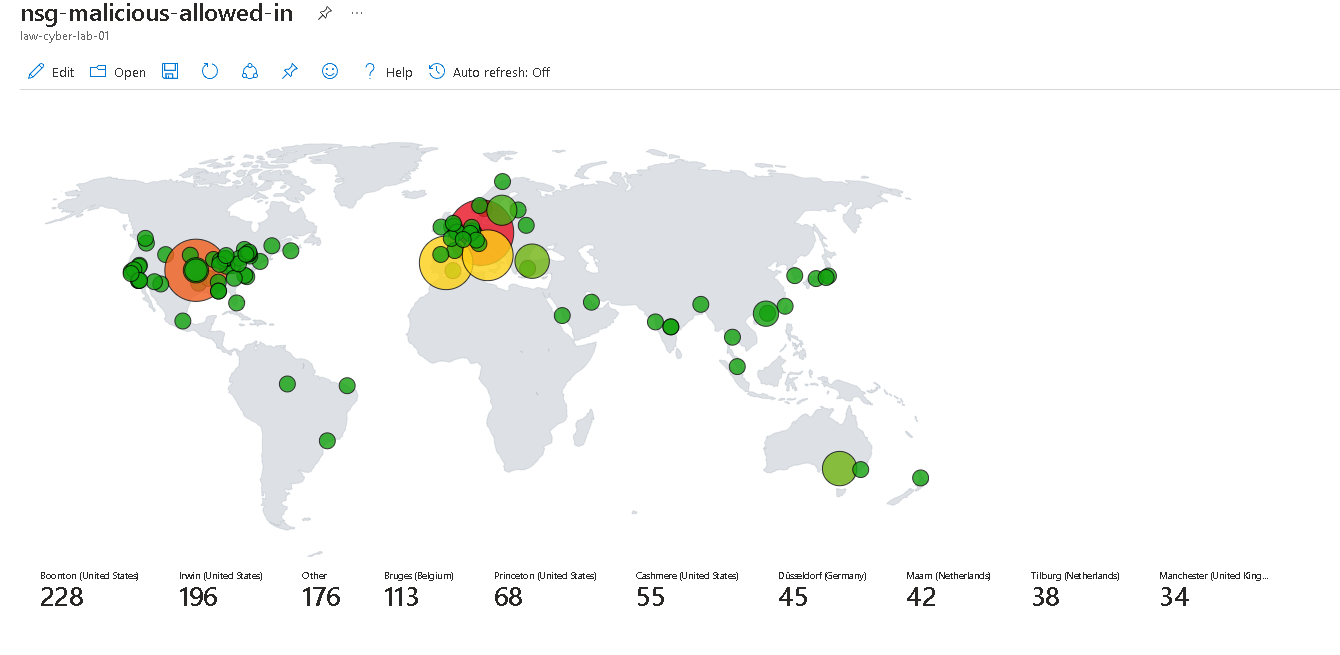


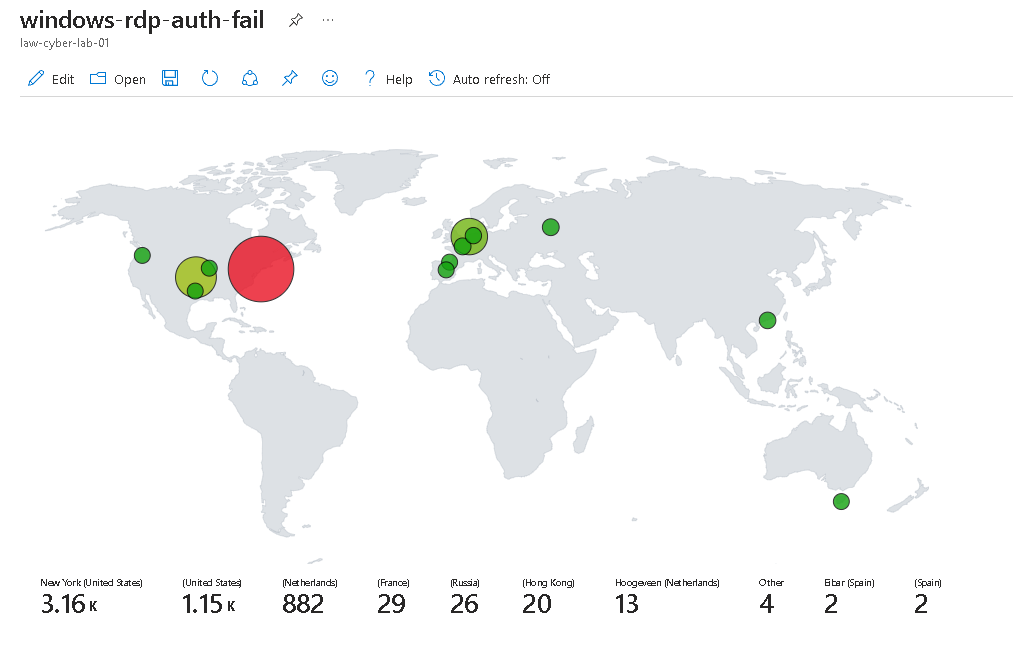
For example, there is a lot of malicious traffic from our NSG.

24 hours later and here is how the graphs are looking:









Definitely a lot of malicious traffic and a lot of attack attempts, let’s go ahead and create some alerts in our SIEM!

* **SecurityEvent  
  | where EventID == 4625  
  | where TimeGenerated > ago(60m)  
  | summarize FailureCount = count() by AttackerIP = IpAddress, EventID, Activity, DestinationHostName = Computer  
  | where FailureCount >= 10**

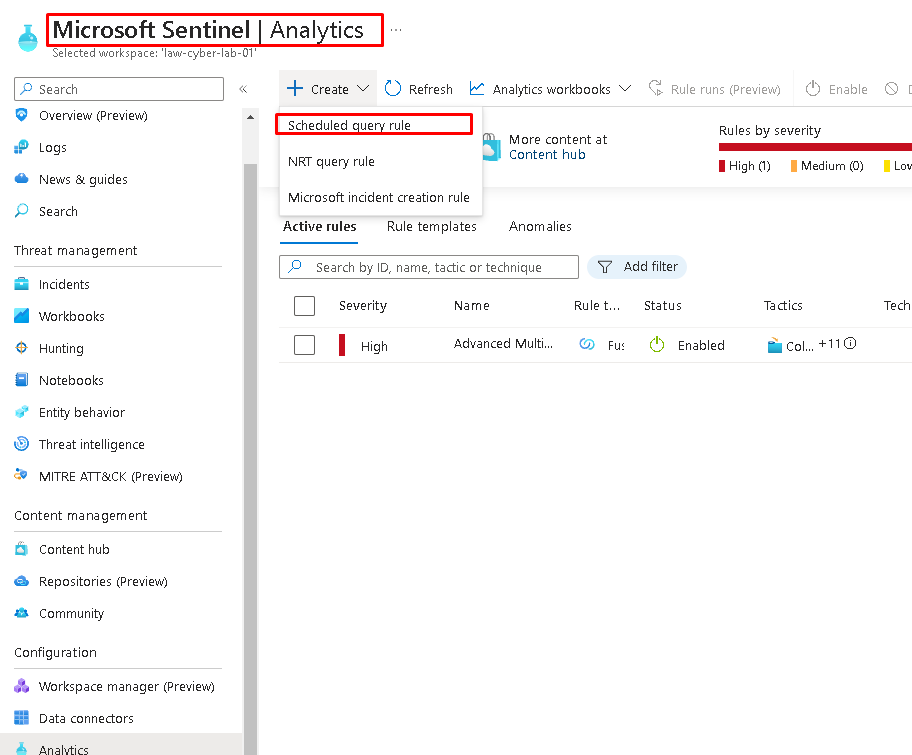
THIS WILL BE OUR TEST ALERT!

And pretty much what this is checking is for a windows brute force attack, like CHATGPT puts it:

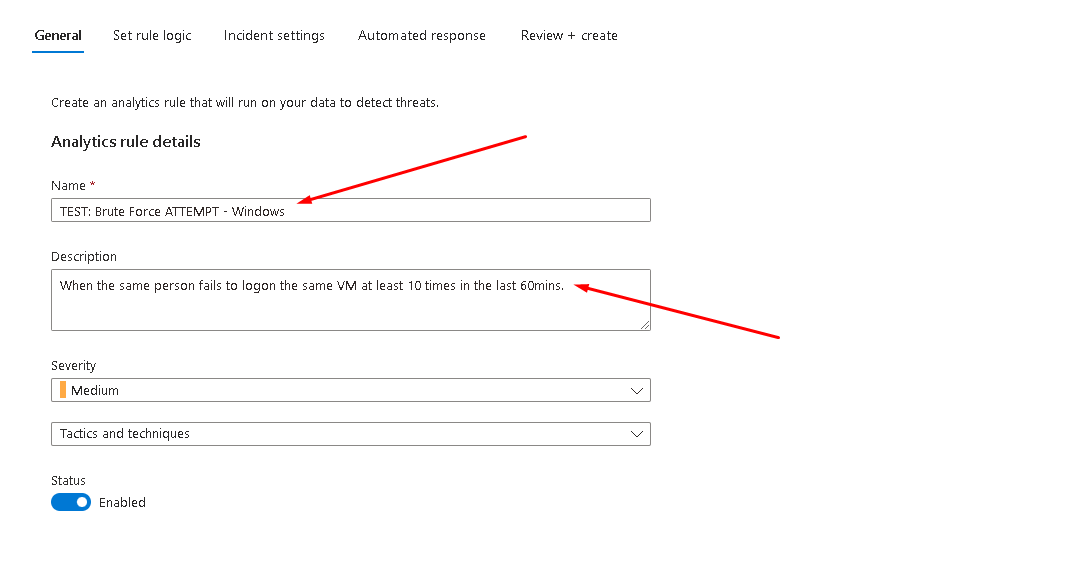
1. **SecurityEvent**: This specifies the data source that the query will operate on, which in this case is a table or dataset containing security events.
2. **| where EventID == 4625**: This filters the events to include only those where the **EventID** is **4625**, which typically represents a failed login attempt in Windows security event logs.
3. **| where TimeGenerated > ago(60m)**: This further filters the events to only include those generated in the last 60 minutes.
4. **| summarize FailureCount = count() by AttackerIP = IpAddress, EventID, Activity, DestinationHostName = Computer**: This aggregates the data by counting the number of failed login attempts (**count()**) and groups them by the IP address of the attacker (**IpAddress**), the **EventID**, the type of activity (**Activity**), and the host name of the target machine (**Computer**). The result is aliased to **FailureCount** for the count, **AttackerIP** for the **IpAddress**, and **DestinationHostName** for the **Computer**.
5. **| where FailureCount >= 10**: This final filter includes only the aggregated results where the count of failed login attempts is 10 or more.

The query is useful for identifying potential brute-force attack attempts by detecting multiple failed login attempts from the same IP address within the last hour. It helps in pinpointing which attackers (by IP address) are frequently trying to log in to which host machines, allowing for a targeted security response.

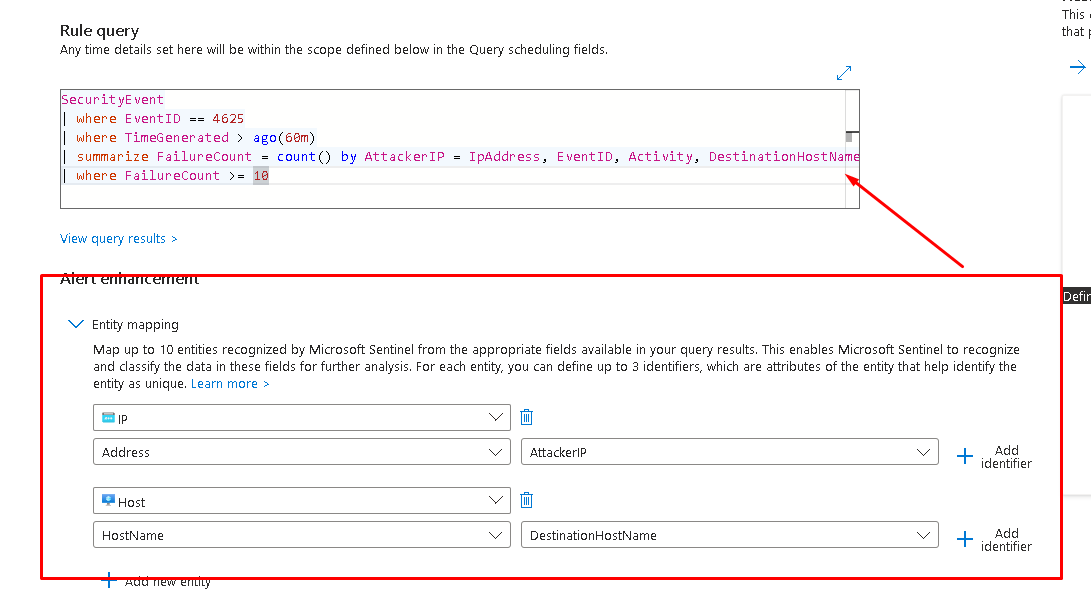
Top of Form



We will go to sentinel and create a scheduled query rule!



We will enter a name of our choosing and a description.

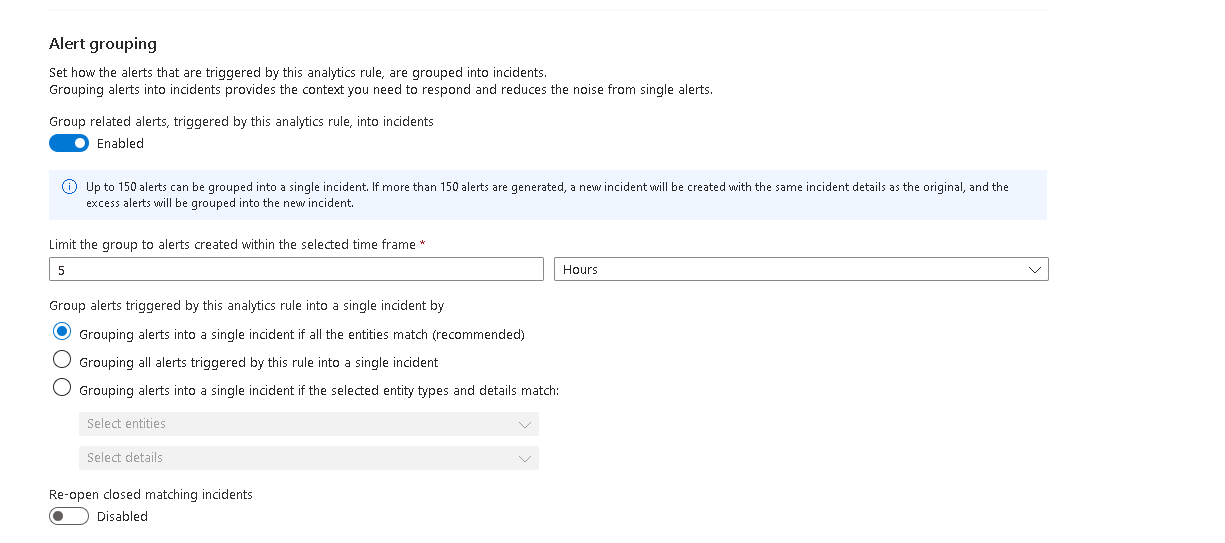


We will put our KQL query on the top and here in entity mapping we will map an IP address as the attacker IP and the host, this pretty much let’s sentinel correlate different alerts with one another (let’s say the same IP is attacking another HOST, sentinel will know that and alert us as well that it’s the same IP).



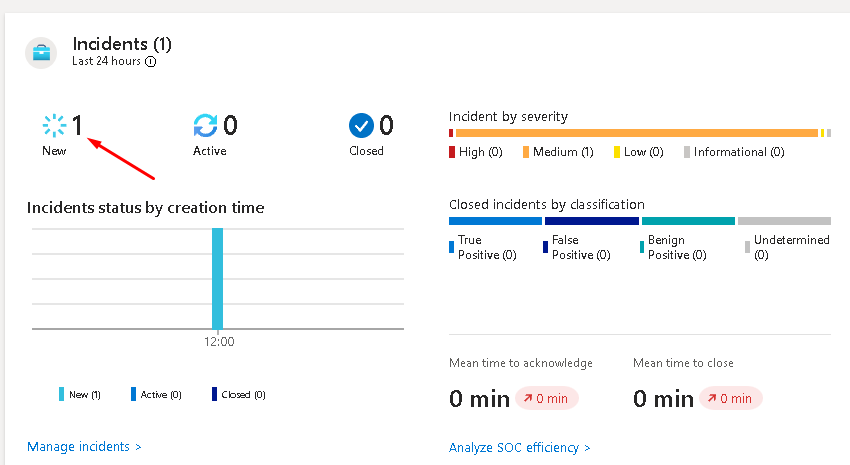
We will make sure this query is running every 5 mins.

Everything else is default!



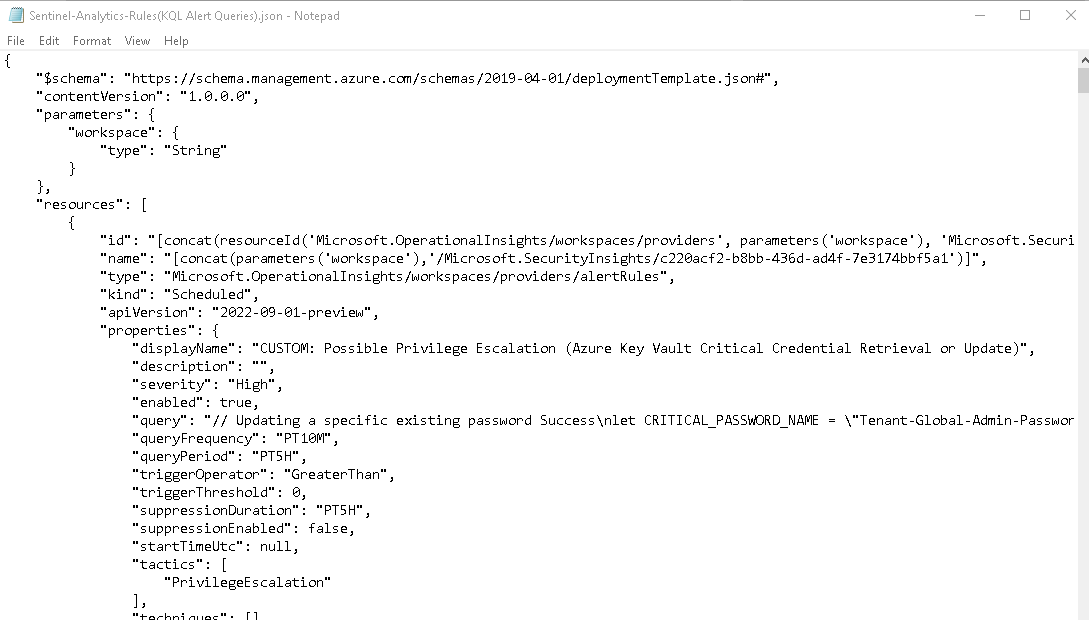
We will enable alert grouping and leave the settings as default.

We don’t want to automate anything because later we will respond to these incidents according to the NIST incident response framework.

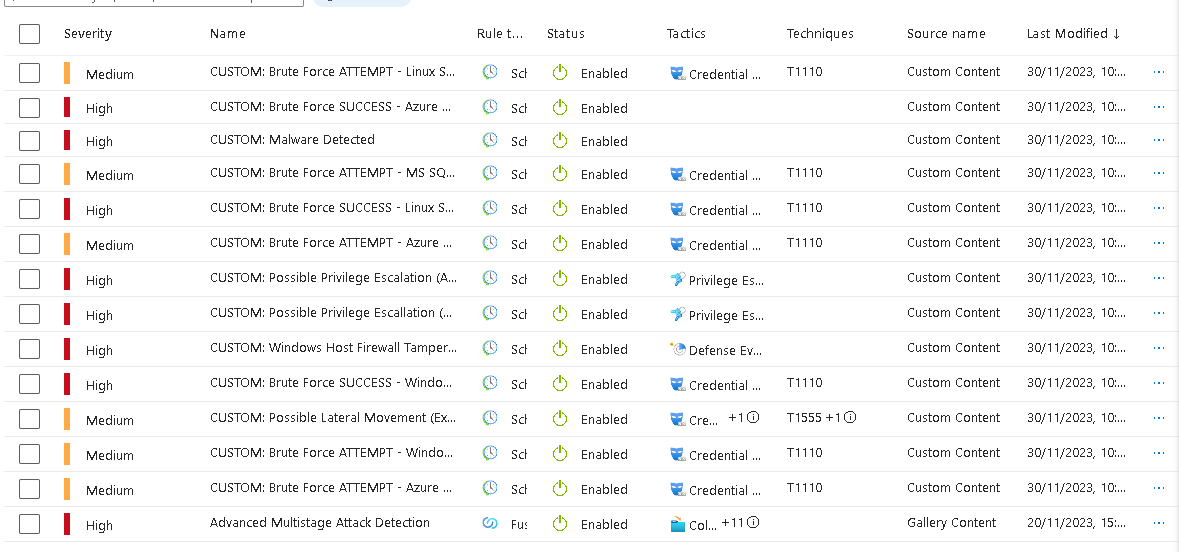


We got “lucky” since the query came back positive without us having to do anything, if the query were to come back with no results, we would have needed to try and “attack” our own windows RDP login for us to get an alert.

Since this rule was mostly for testing, we will now import a whole bunch of detection queries for this project.



We will import this big file with a bunch of detection rules already setup.



Everything worked like expected!

And we will have 13 custom detection rules and 1 default rule that comes already with sentinel.

// Brute Force Success Windows

let FailedLogons = SecurityEvent

| where EventID == 4625 and LogonType == 3

| where TimeGenerated > ago(1h)

| summarize FailureCount = count() by AttackerIP = IpAddress, EventID, Activity, LogonType, DestinationHostName = Computer

| where FailureCount >= 5;

let SuccessfulLogons = SecurityEvent

| where EventID == 4624 and LogonType == 3

| where TimeGenerated > ago(1h)

| summarize SuccessfulCount = count() by AttackerIP = IpAddress, LogonType, DestinationHostName = Computer, AuthenticationSuccessTime = TimeGenerated;

SuccessfulLogons

| join kind = inner FailedLogons on DestinationHostName, AttackerIP, LogonType

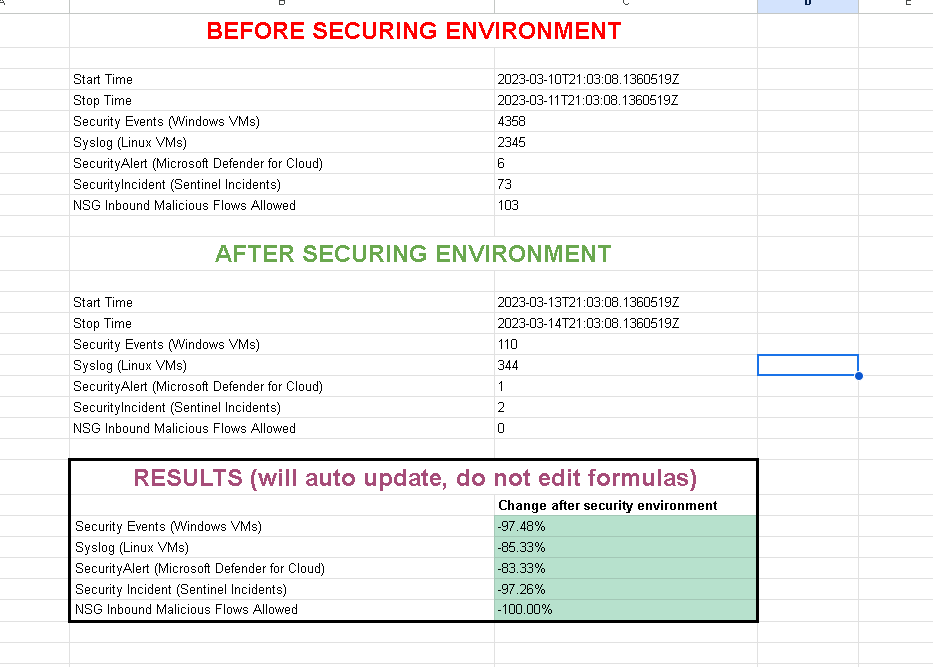
| project AuthenticationSuccessTime, AttackerIP, DestinationHostName, FailureCount, SuccessfulCount

For example, this is one of our rules that detects brute force success on windows, let’s break it down.

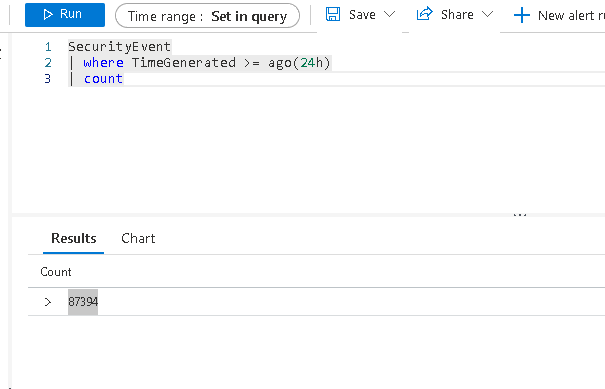
This KQL query is composed of two subqueries that interact with each other, and its primary function is to detect brute force login attempts that have eventually succeeded on Windows systems. Here is a breakdown of the query:

1. **Define FailedLogons Subquery**:
   * **SecurityEvent**: This selects the security events to be analyzed.
   * **| where EventID == 4625 and LogonType == 3**: Filters for events with an ID of 4625, which indicates a failed logon attempt, and where the logon type is 3, which usually means a network logon (often used for remote access).
   * **| where TimeGenerated > ago(1h)**: Limits the events to those generated in the last hour.
   * **| summarize FailureCount = count() by AttackerIP = IpAddress, EventID, Activity, LogonType, DestinationHostName = Computer**: Aggregates the number of failed logon attempts, grouping them by IP address of the attacker, event ID, activity type, logon type, and the host name of the target machine.
   * **| where FailureCount >= 5**: Filters the aggregated results to only include records where there have been five or more failed attempts.
2. **Define SuccessfulLogons Subquery**:
   * **SecurityEvent**: Again, selects the security events to be analyzed.
   * **| where EventID == 4624 and LogonType == 3**: Filters for events with an ID of 4624, which indicates a successful logon, again where the logon type is 3.
   * **| where TimeGenerated > ago(1h)**: Limits the events to those generated in the last hour.
   * **| summarize SuccessfulCount = count() by AttackerIP = IpAddress, LogonType, DestinationHostName = Computer, AuthenticationSuccessTime = TimeGenerated**: Aggregates the number of successful logons, grouped by attacker IP, logon type, destination host name, and the time the authentication succeeded.
3. **Combine FailedLogons and SuccessfulLogons**:
   * **SuccessfulLogons**: This initiates the join operation with the results of the **SuccessfulLogons** subquery.
   * **| join kind = inner FailedLogons on DestinationHostName, AttackerIP, LogonType**: Performs an inner join between the two subqueries on the destination host name, attacker IP, and logon type. This join will match records from **SuccessfulLogons** with records from **FailedLogons** where the logon type, attacker IP, and destination host name are the same.
   * **| project AuthenticationSuccessTime, AttackerIP, DestinationHostName, FailureCount, SuccessfulCount**: Projects (selects) the final columns to display in the output, which includes the time of successful authentication, IP address of the attacker, the destination host name, the count of failed attempts, and the count of successful logons.

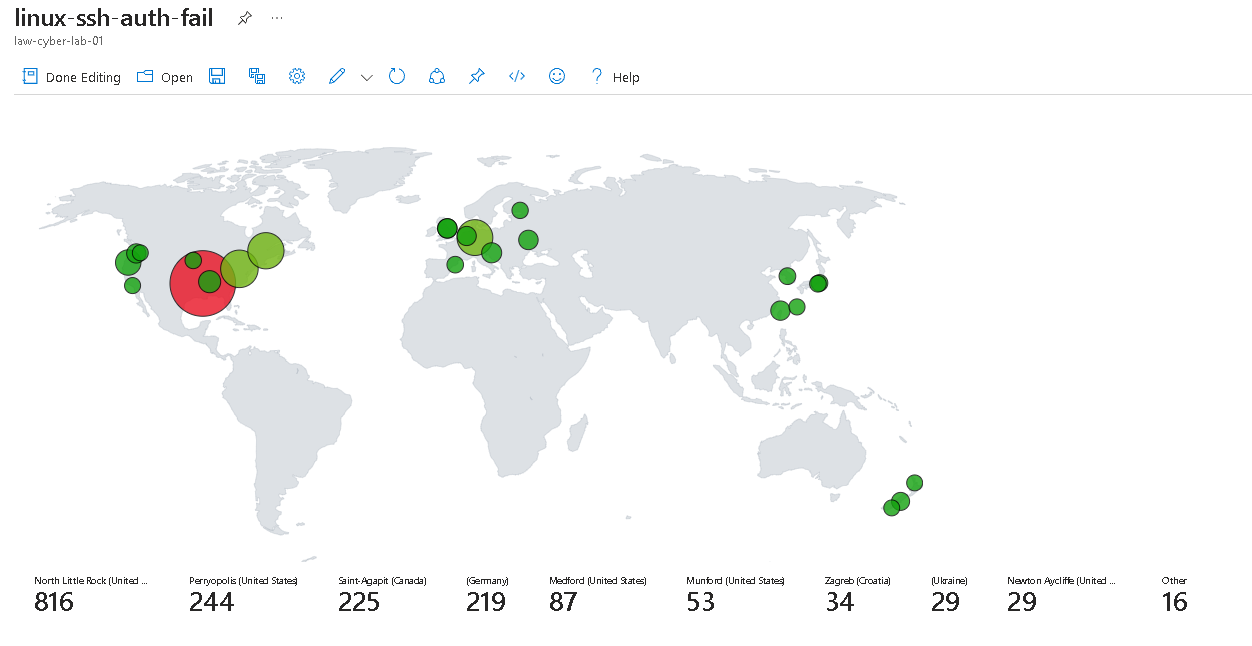
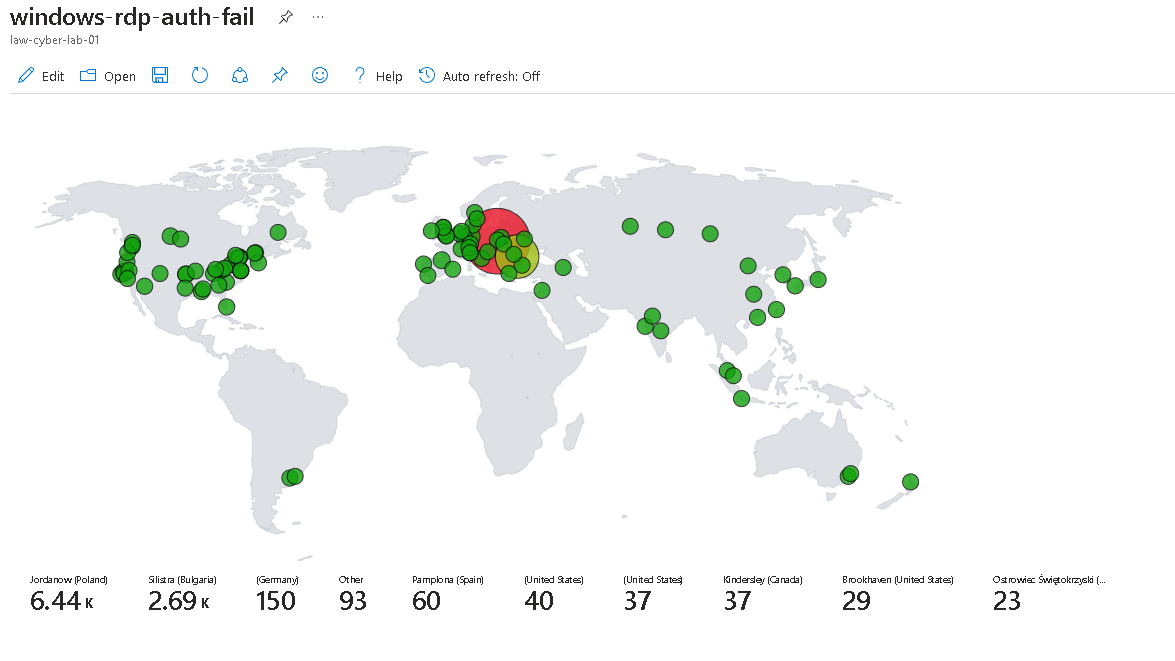
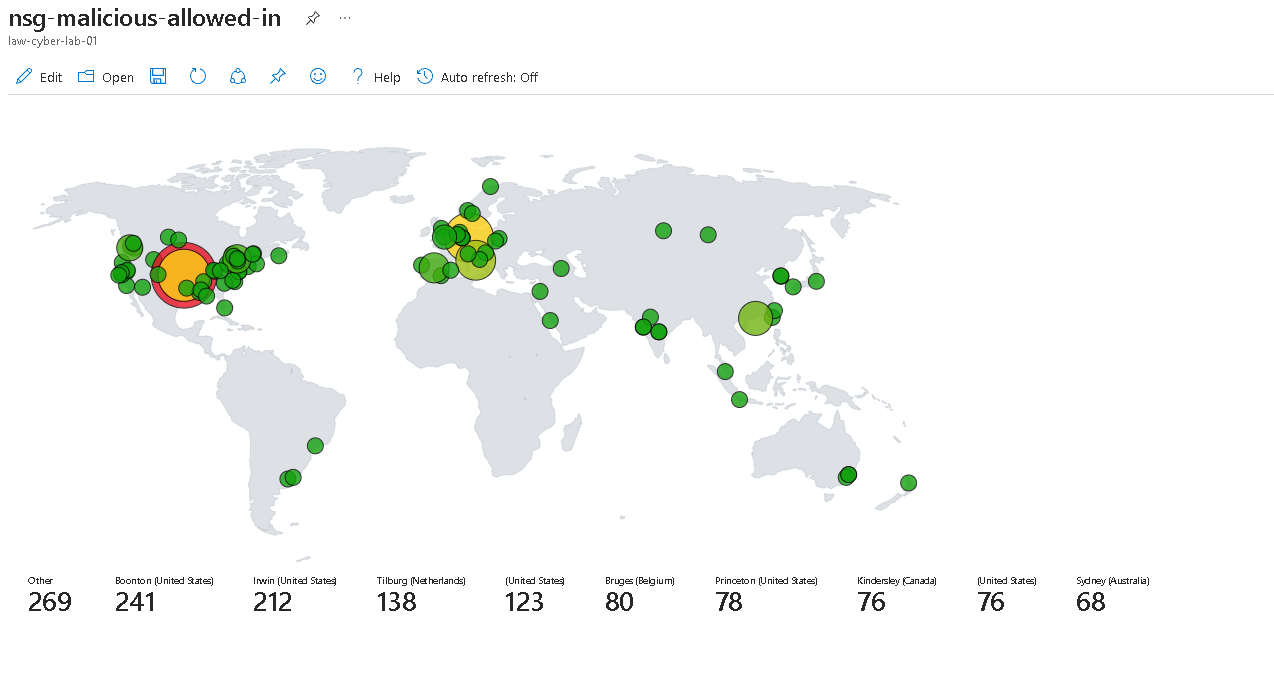
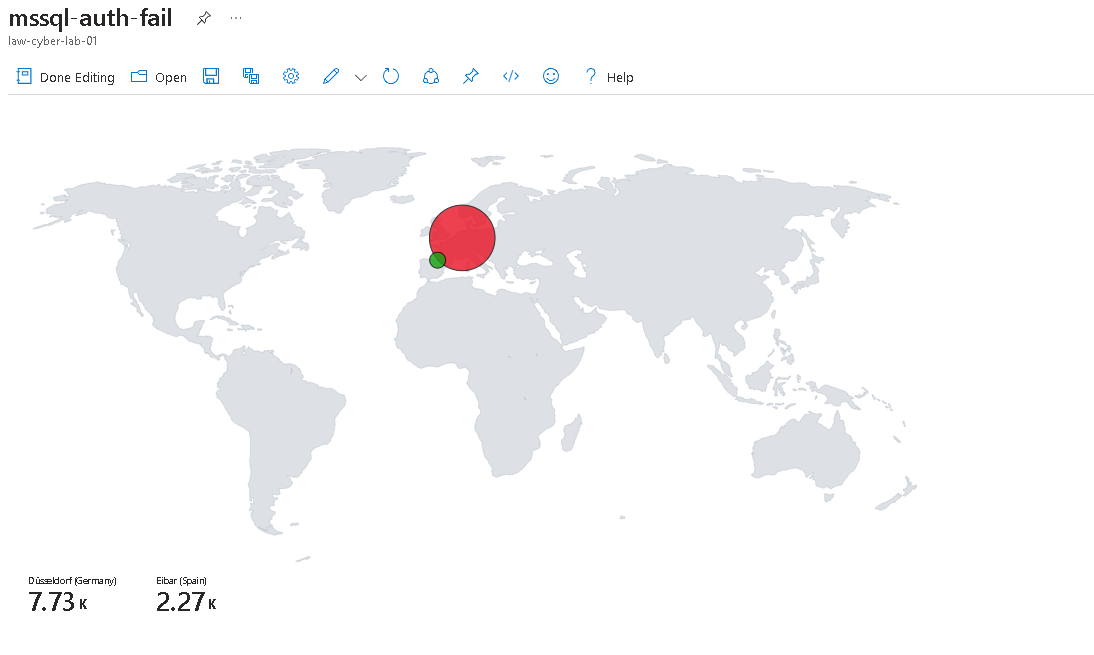
The resulting output from this query would give you the instances where an IP address had multiple failed logon attempts (suggesting a brute force attack) but eventually managed to successfully log in within the past hour. This could indicate that the brute force attack was successful and may warrant further investigation or immediate action.



Before we keep on going, we will record our stats.



We will record all the numbers in the spreadsheet.

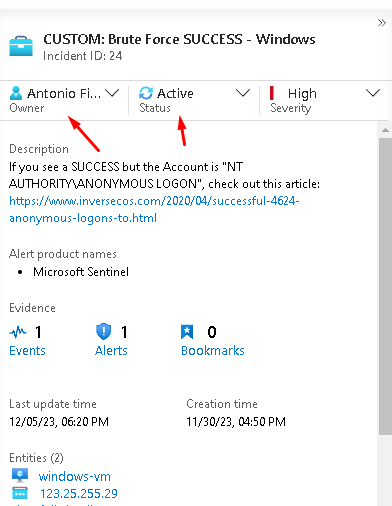


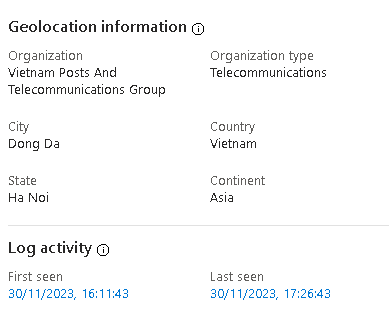
Here is all the malicious traffic in the last 24 hours.

We will now adhere to NIST 800-61 to practice incident response.

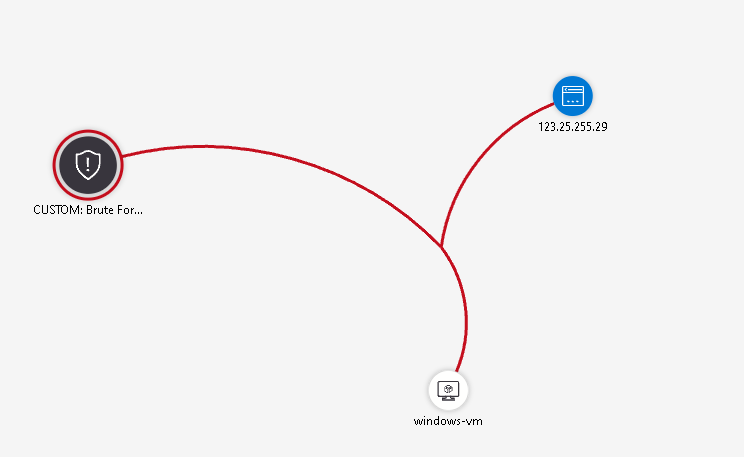
It goes something like this:



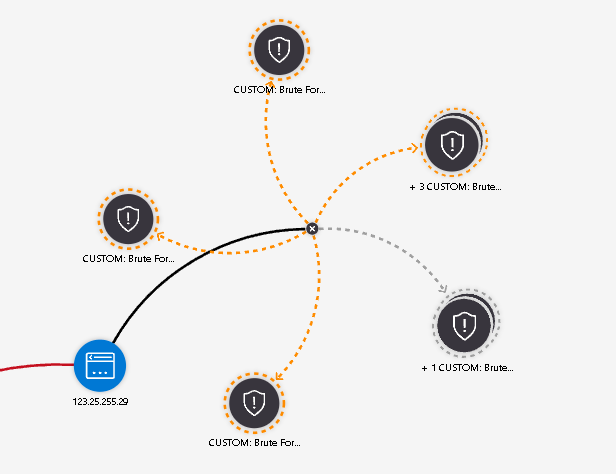




We have info that this person is from Vietnam.



We can see that this attacker was not able to attack other hosts.



There are other events associated with this same attacker though, brute force attempts…





Seems like a true positive as the KQL query is very specific.

This KQL (Kusto Query Language) query is designed to identify potential brute force attack attempts on Windows systems, specifically by detecting multiple failed logon attempts followed by a successful logon within a certain time frame. The query is divided into several parts:

1. \*\*Failed Logons\*\*: This section of the query filters `SecurityEvent` logs for failed logon events (EventID 4625) with LogonType 3, which indicates network logon attempts. It only considers events generated in the last hour (`ago(1h)`). It then summarizes these events, counting the number of failures (`FailureCount`) by the attacker's IP address (`AttackerIP`), along with other details like EventID, Activity, LogonType, and the destination host name (`DestinationHostName`). It filters for instances where there have been five or more failed attempts from the same IP.

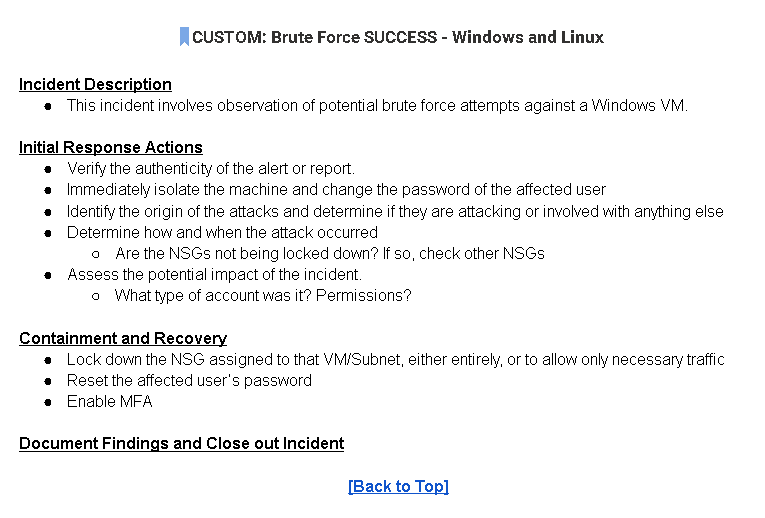
2. \*\*Successful Logons\*\*: Similar to the first part, this section filters for successful logon events (EventID 4624) with LogonType 3, also within the last hour. It summarizes these events by counting the successful attempts (`SuccessfulCount`) by the IP address, LogonType, destination host name, and the time of successful authentication (`AuthenticationSuccessTime`).

3. \*\*Joining and Projecting Data\*\*: The final part joins the two datasets (`SuccessfulLogons` and `FailedLogons`) using an inner join on shared columns (`DestinationHostName`, `AttackerIP`, `LogonType`). This means it only includes records that have matching values in both datasets for these columns. The `project` statement then specifies which columns to include in the final output: the time of successful authentication, the attacker's IP address, the destination host name, the number of failed attempts, and the number of successful attempts.

The purpose of this query is to identify IP addresses that have had multiple failed logon attempts (suggesting a brute force attack) followed by a successful logon, which could indicate that the attacker eventually guessed the correct credentials. This can be a useful tool for security analysts monitoring for potential security breaches.

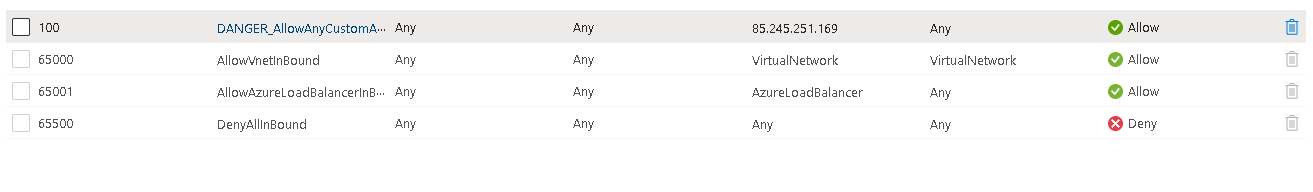
Though the alert was a true positive, this type of traffic should not be allowed to reach our VM in the first place, possible NSG configuration issue.

Will close out this incident but will look into NSG.



This is the playbook for this incident.

We will now lock down NSG.



We just put so that only our IP address can access this VM.

So now this attack will not happen anymore as only we can access these VMs.

We will enable MFA later.

I will keep practicing incident response using the incident management playbook that states (reset password, enable mfa, lock down nsg, etc…)

After that I will now Enable NIST 800-53, we will lock down our environment more and more until it becomes complacent with NIST.

A screenshot of a computer

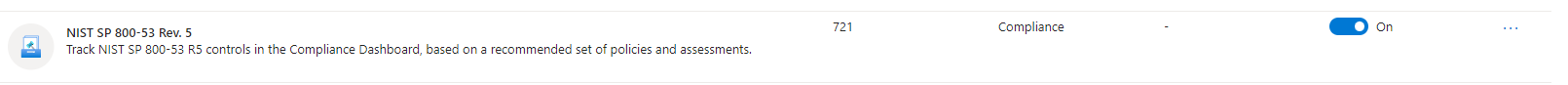
Description automatically generated

We will go to defender for cloud and enable nist 800-53 controls so we can have a checklist on how to harden out environment, security recommendations,etc…

A screenshot of a computer

Description automatically generated

Nist 800-53 family controls.



We just added revision 5!

Now that nist 800-53 family controls are enabled, we can go ahead and look at this:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

And we can see it does indeed match.

We will implement NIST 800-53: SC-7: BOUNDRY PROTECTION

A screenshot of a computer

Description automatically generated

We will start first by configuring a private link and firewall for our key vault.

A screenshot of a computer

Description automatically generated

This is what we must do.

Next to that tab we will also create our private endpoint:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Rest is left at default.

We will now do the same for our storage accounts.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The rest is just like the previous one, but we want to make sure blob is selected here since we are creating our private endpoint for a blob storage account.

Next, we will attach the network security group to the subnet.

A screenshot of a computer

Description automatically generated

Now that the nsg is created we will attach it to our subnet. This is mostly to satisfy the recommendation that defender for cloud gives us since we won’t have any rules in this nsg. Our lab machines are already secure so there is really no point in adding the same rules again here. We are doing this to practice.

A screenshot of a computer

Description automatically generated

Now that these nist 800-53 controls are in place we will again run our environment for 24 hours and see if there is any difference in the number of attacks compared to before.

I am back after roughly 24 hours and now we will look at the results!

A screenshot of a computer

Description automatically generated

These are the before numbers and now let’s get the after.

We will use these queries to get the information we want:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Security Events (Windows VMs) | SecurityEvent | where TimeGenerated >= ago(24h) | count |  |  |  |
| Syslog (Linux VMs) | Syslog | where TimeGenerated >= ago(24h) | count |  |  |  |
| SecurityAlert (Microsoft Defender for Cloud) | SecurityAlert | where DisplayName !startswith "CUSTOM" and DisplayName !startswith "TEST" | where TimeGenerated >= ago(24h) | count |  |  |  |
| Security Incident (Sentinel Incidents) | SecurityIncident | where TimeGenerated >= ago(24h) | count |  |  |  |
| NSG Inbound Malicious Flows Allowed | AzureNetworkAnalytics\_CL  | where FlowType\_s == "MaliciousFlow" and AllowedInFlows\_d > 0 | where TimeGenerated >= ago(24h) | count |  |  |  |
| NSG Inbound Malicious Flows Blocked | AzureNetworkAnalytics\_CL  | where FlowType\_s == "MaliciousFlow" and DeniedInFlows\_d > 0 | where TimeGenerated >= ago(24h) | count |  |  |  |

A screenshot of a computer

Description automatically generated

Let’s keep going here is what we have so far:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

We can see a HUGE difference already…

And here is the result:

A screenshot of a document

Description automatically generated

End.