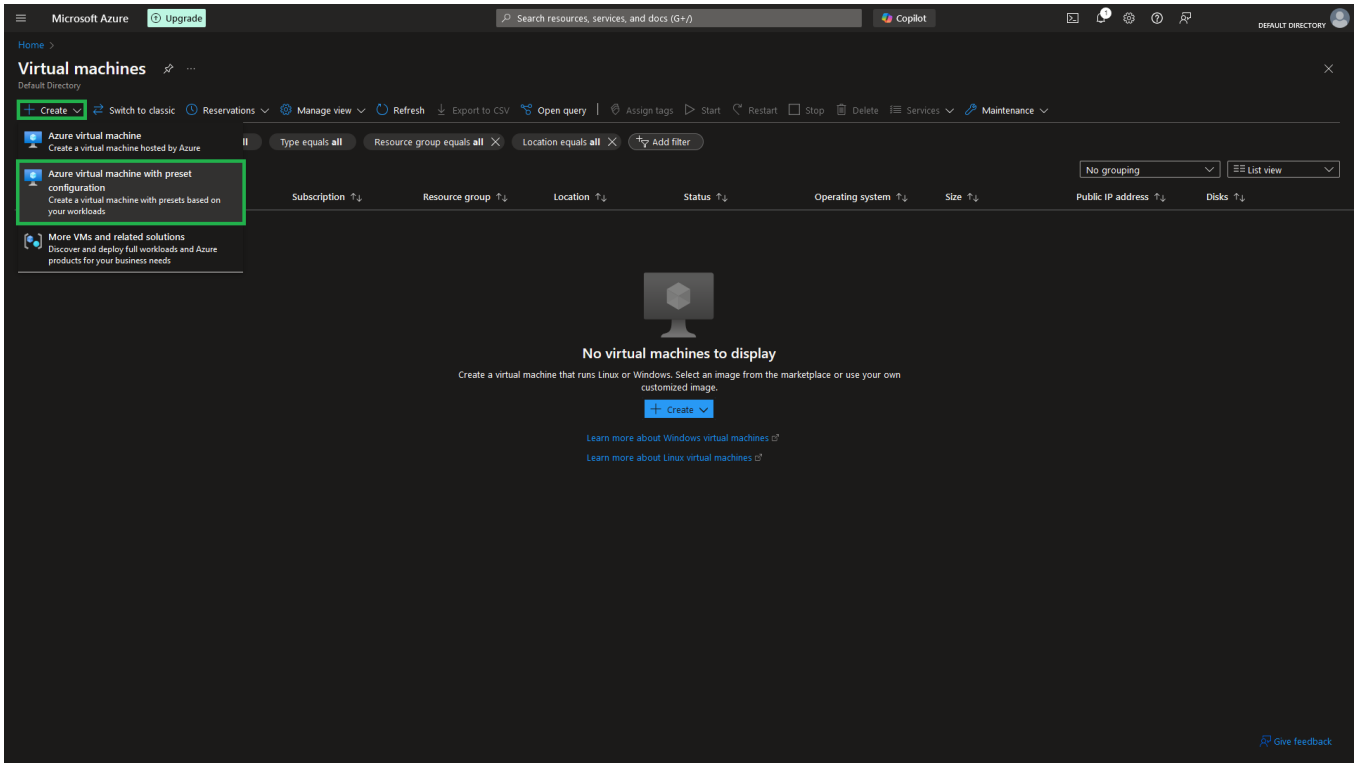
Azure security project solution PART 1:
Creating a honeypot VM

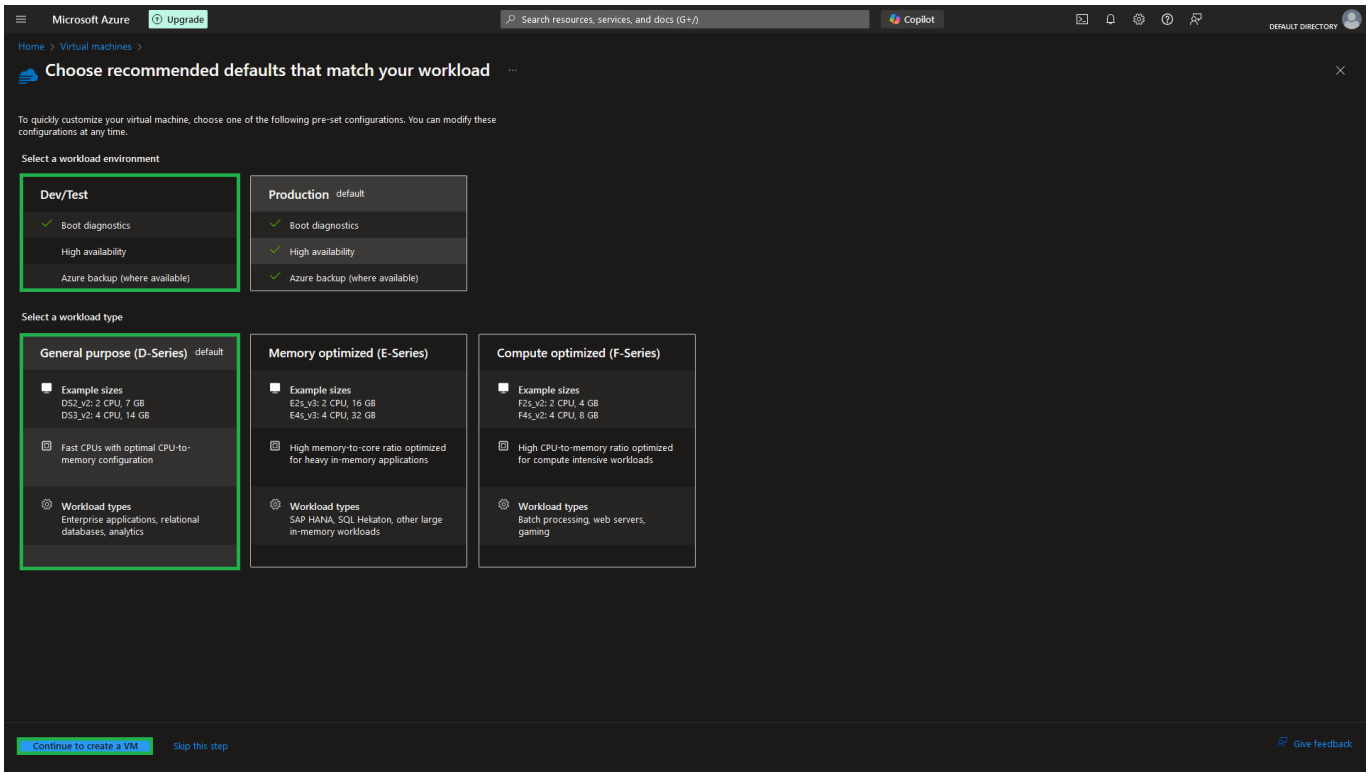
In this project we will build our own security operations center by utilizing Microsoft Sentinel to deploy a Security Information and Event Management (SIEM) solution that monitors and generates alerts for a virtual machine – we can use it as an RDP honeypot. We will also setup a threat intelligence feed that sends information about common and newly discovered indicators of compromise (IoCs) to the SIEM.

Task 1: Create a resource group and a Virtual Machine

1. After we have logged in to the Azure Portal, we begin by going to the **Virtual Machines** page, selecting **Create**, and clicking on **Azure virtual machine with preset**.

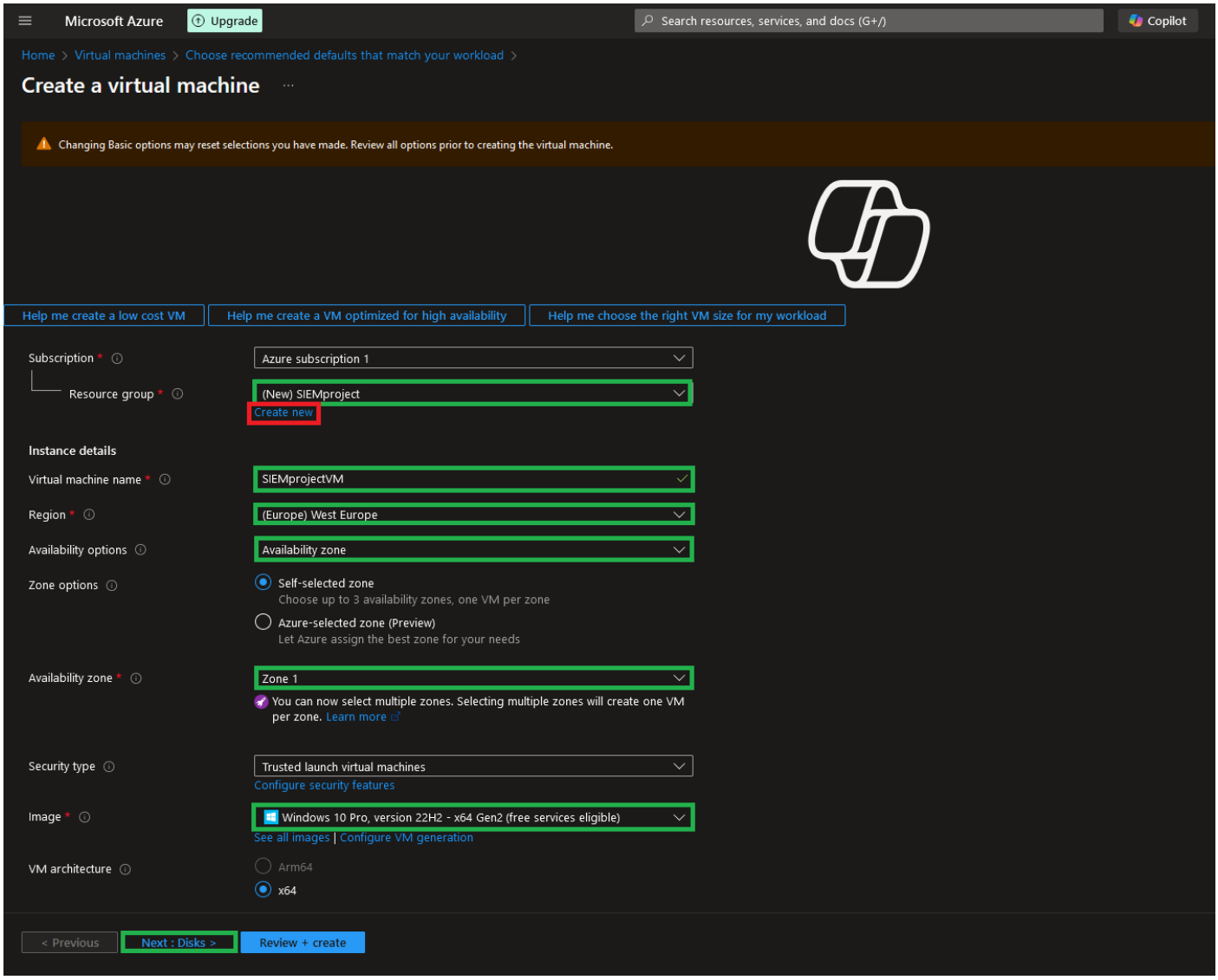


2. We then choose to the a **Dev/Test** under *Select a workload environment* and **General purpose (D-Series)** under *Select a workload type*.



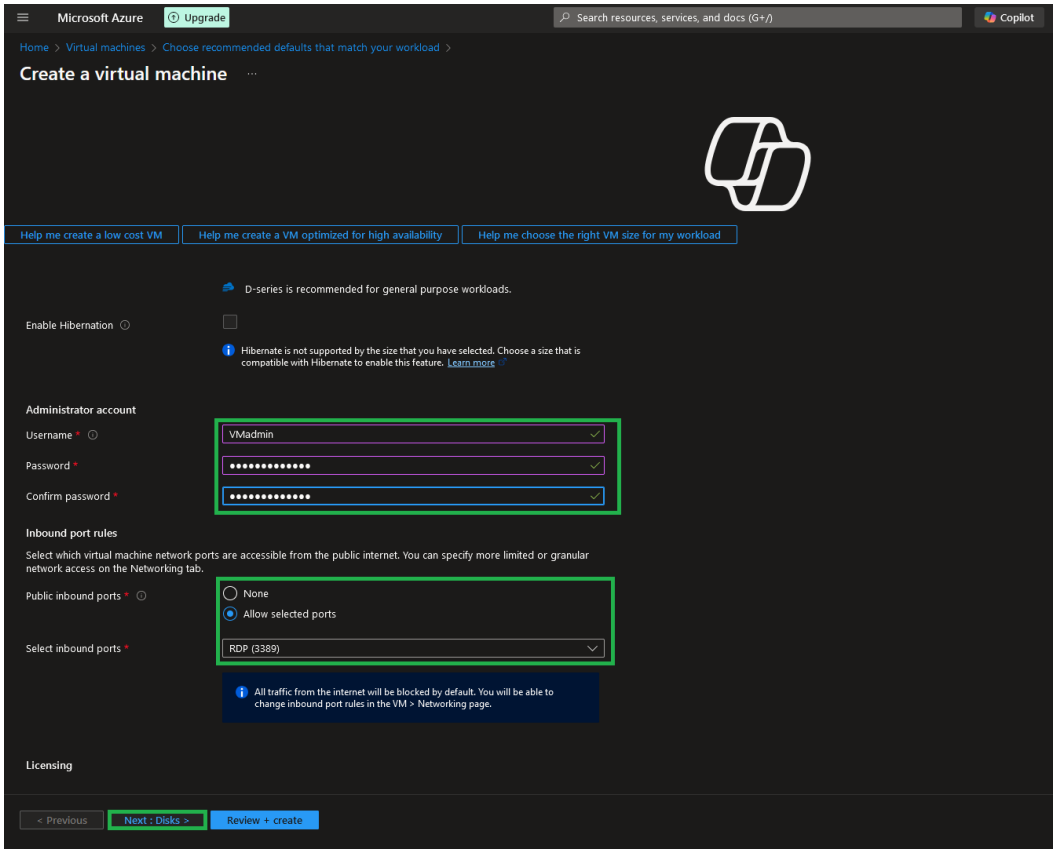
3. On the next page, on the **Basics** tab, we create the following configuration:

Field	**Value**
Subscription	Our current subscription.
Resource group	Create a new resource group called **SIEMproject** by clicking on **Create new** in the Resource group section.
Virtual machine name	Enter a unique VM name, such as **SIEMprojectVM**.
Region	Select the with the least latency and highest availability closest to you.
Availability options	Select **Availability zone**.
Availability zone	Select **Zones 1**.
Image	Select **Windows 10 Pro**

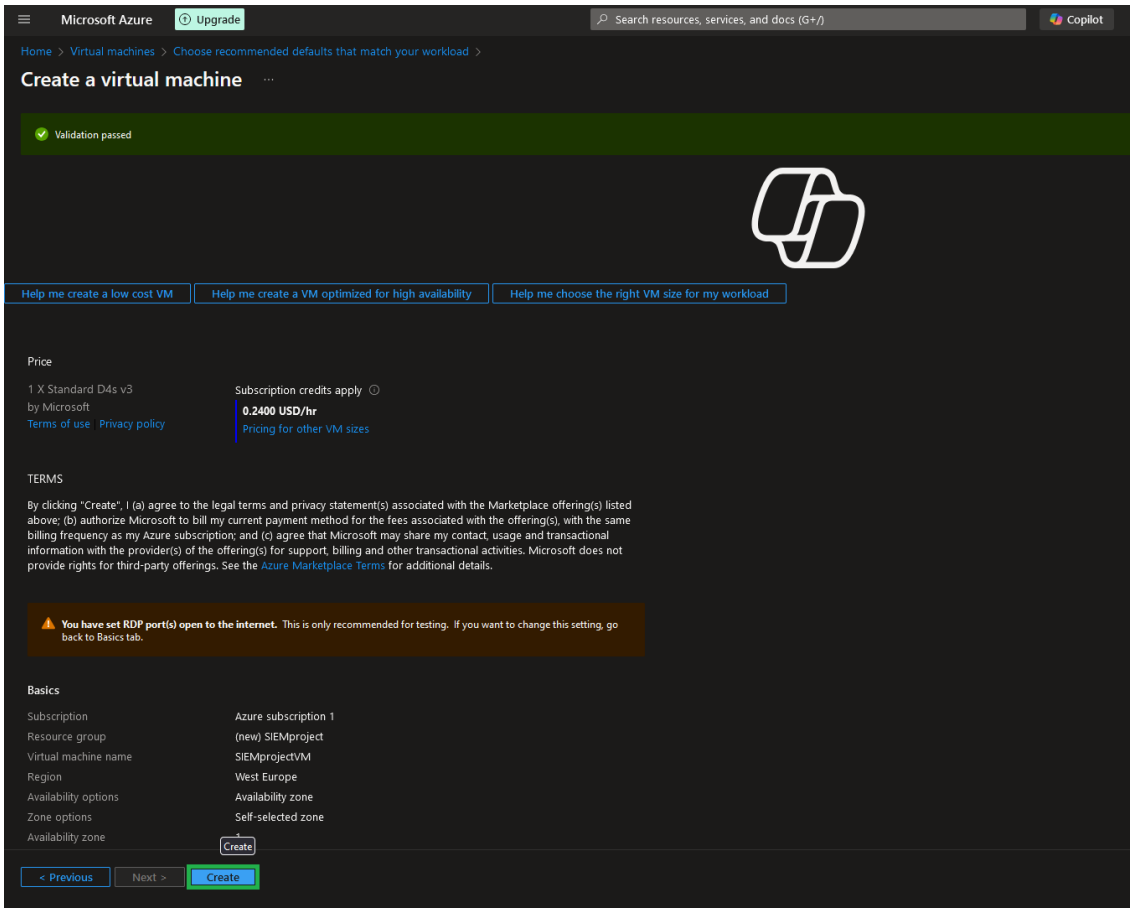


4. We scroll down to create our admin account **username** and **password** and allow **RDP port 3389**.

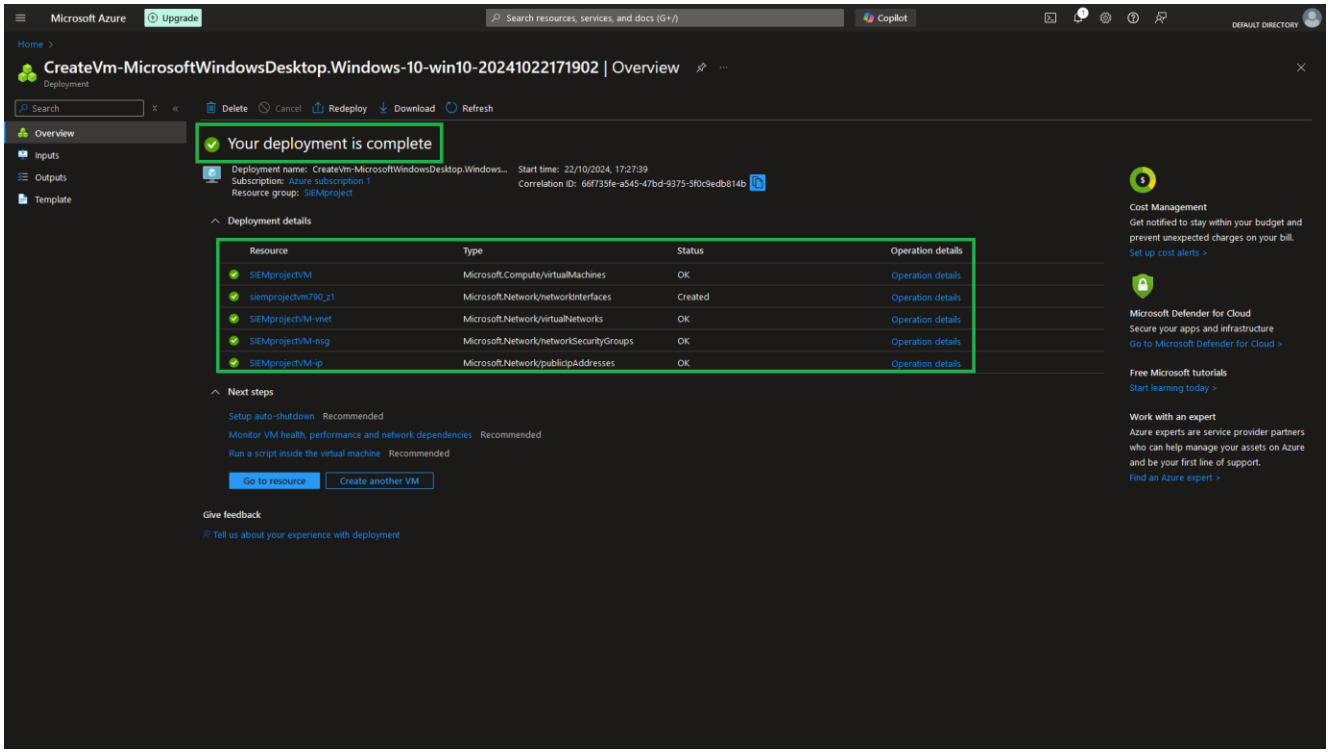
   Note: RDP events are the most easily generated security events which is why they are ideal to quickly test if our SIEM solution is successfully generating alerts. In a production environment, it is best to use **Microsoft Defender for Cloud's just-in-time (JIT) access** for VMs for maximum security.



5. We can leave all other tabs to their default values and go straight to the **Review + create** tab where we click on **Create** to deploy the resource.

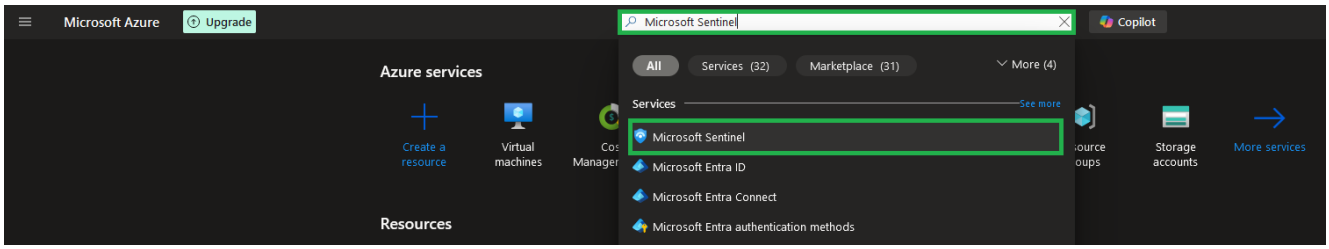


6. We now wait for notification that our deployment is complete.

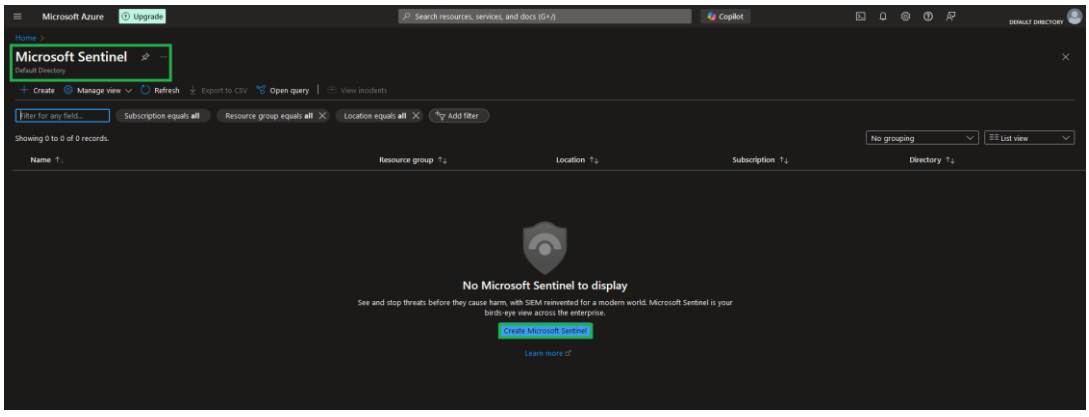


Task 2: Create a Log Analytics workspace and activate Microsoft Sentinel

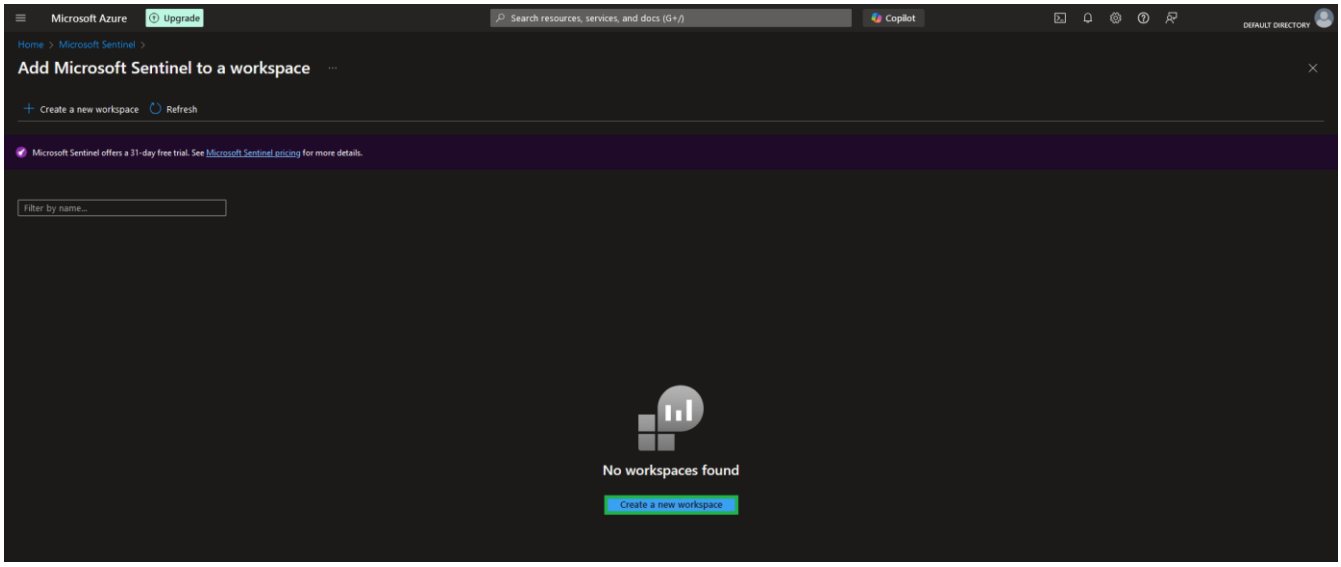
1. Once that is done, we go back to the Azure Portal and type in “Microsoft Sentinel” into the search box.



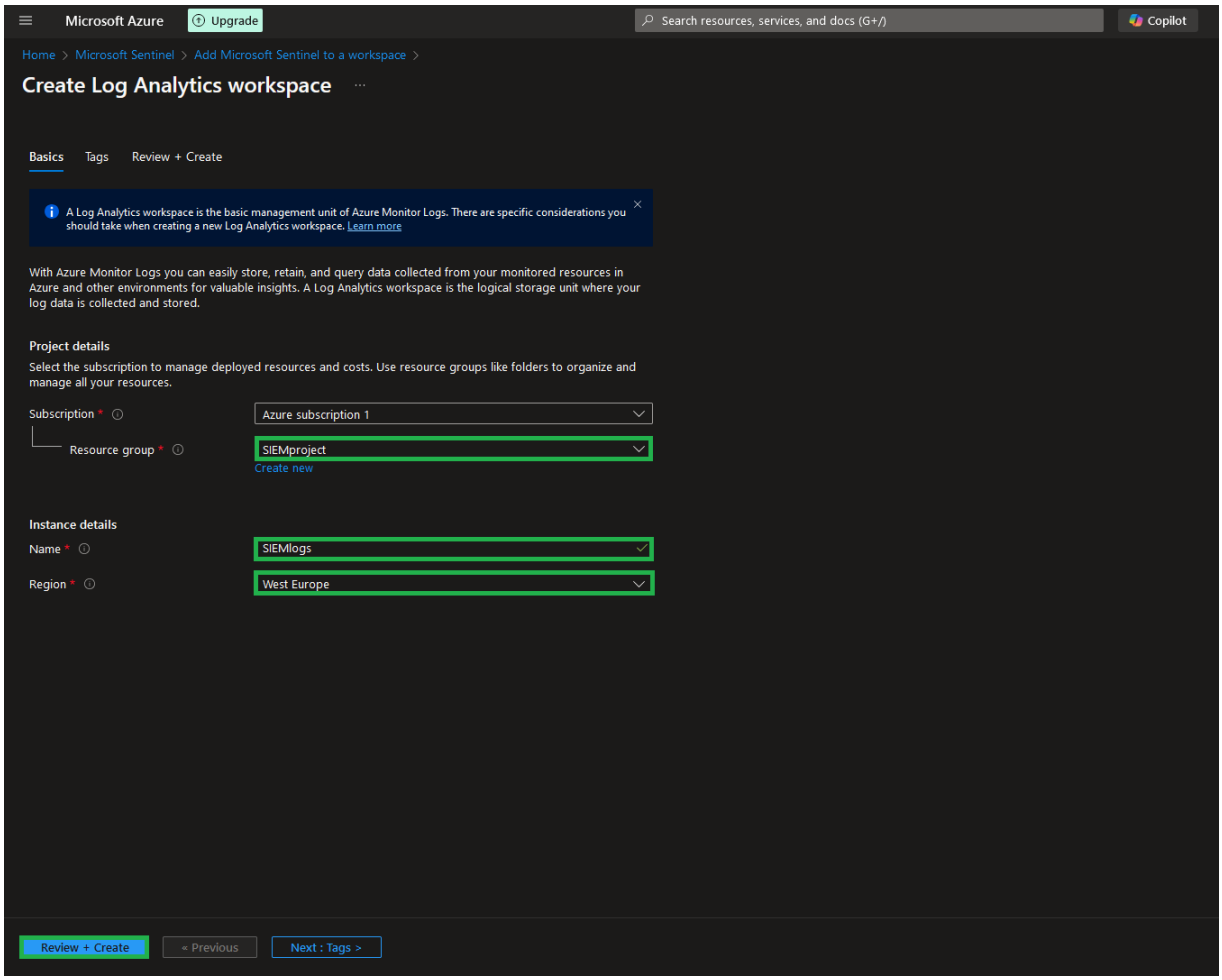
2. On the **Microsoft Sentinel** page, we simply click on **Create Microsoft Sentinel**.



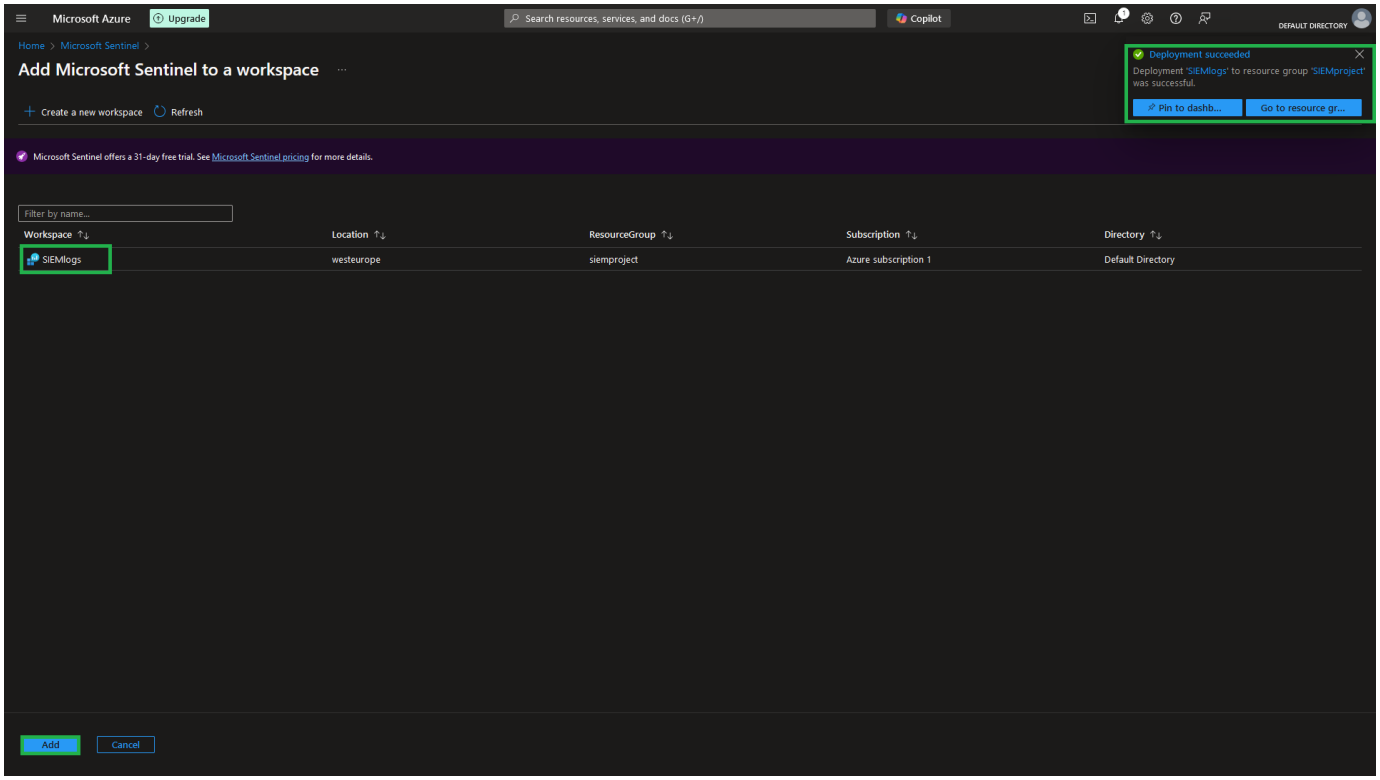
3. We can then either add Sentinel to an existing Log Analytics workspace or create a new one. Here we have no existing workspaces, so we click on the **Create a workspace** button.



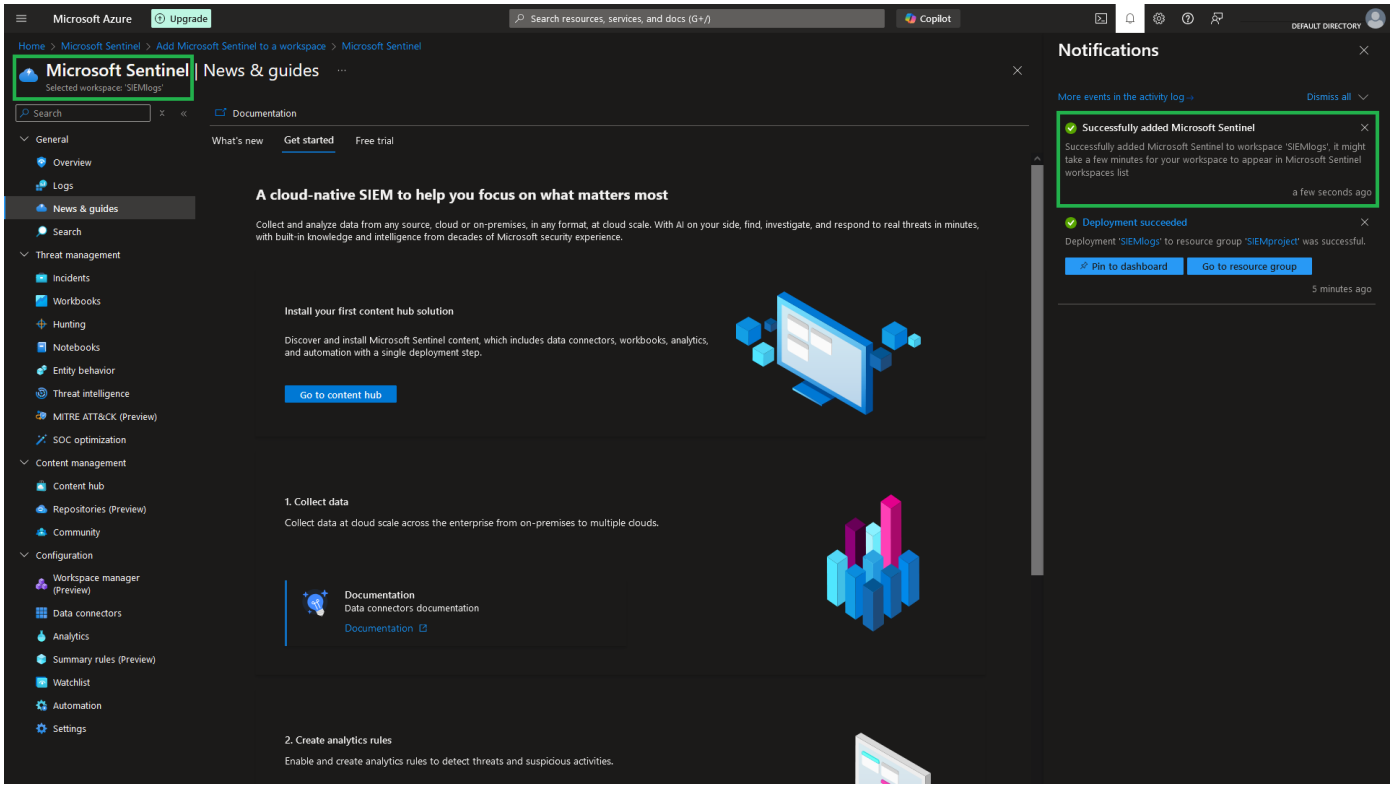
4. On the **Create Log Analytics workspace** page, we make sure that we have selected the same resource group as the resources that we want to monitor – in this case, the **SIEMproject** resource group where our VM is. We also make sure that we are deploying it the same region, in my case **West Europe**. Finally, we name our workspace, select Review + create, and click **Create** after we've reviewed our configuration.



5. Once we are notified that our Log Analytics workspace is created, we click on the **Add** button to add Microsoft Sentinel to the workspace.

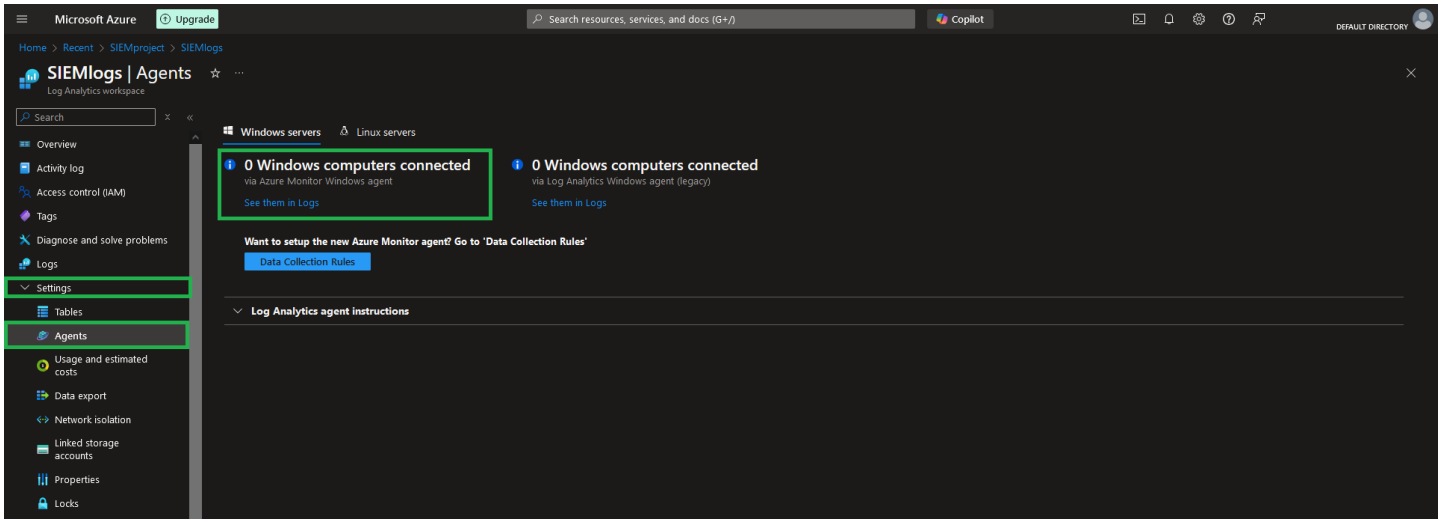


6. We now wait for the confirmation that Microsoft Sentinel has been activated.

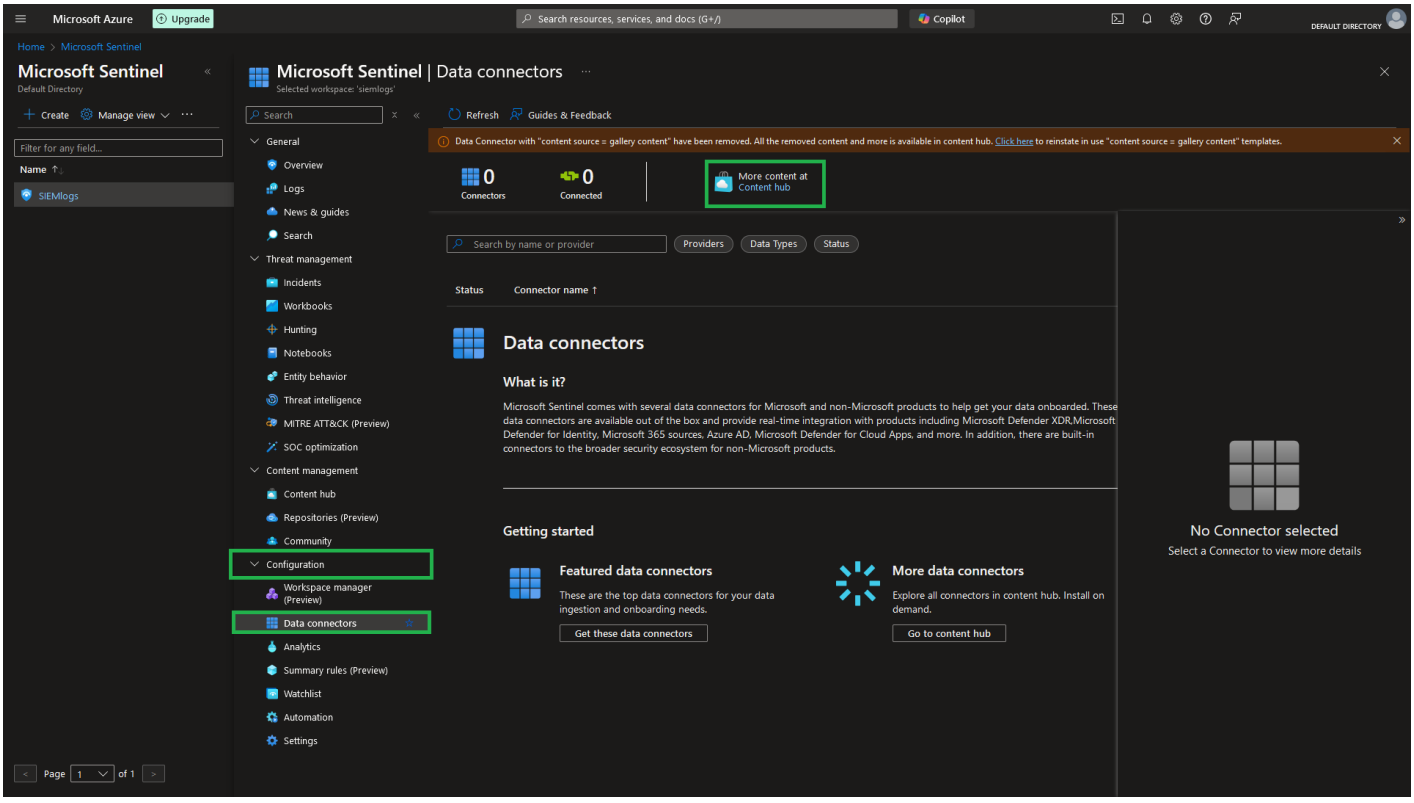


Task 3: Setting up a Data Connector and creating a data collection rule

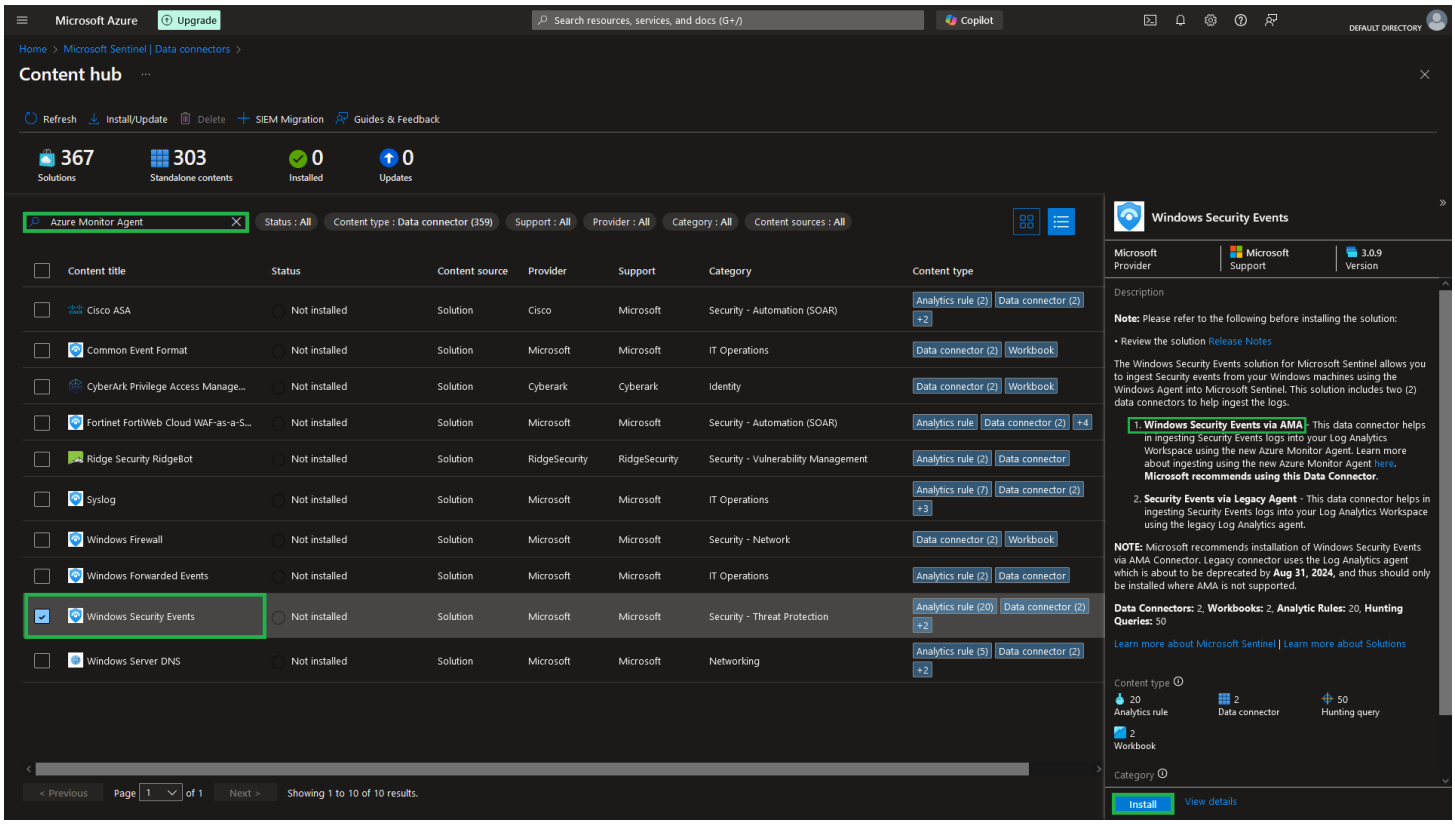
1. We now need to add our virtual machine to the Log Analytics workspace so that its event logs can be ingested by Microsoft Sentinel. If we go back to the **SEMlogs** Log Analytics workspace page and go to **Agents** under the *Settings* blade, we will see that we have 0 Windows computers connected to the workspace.



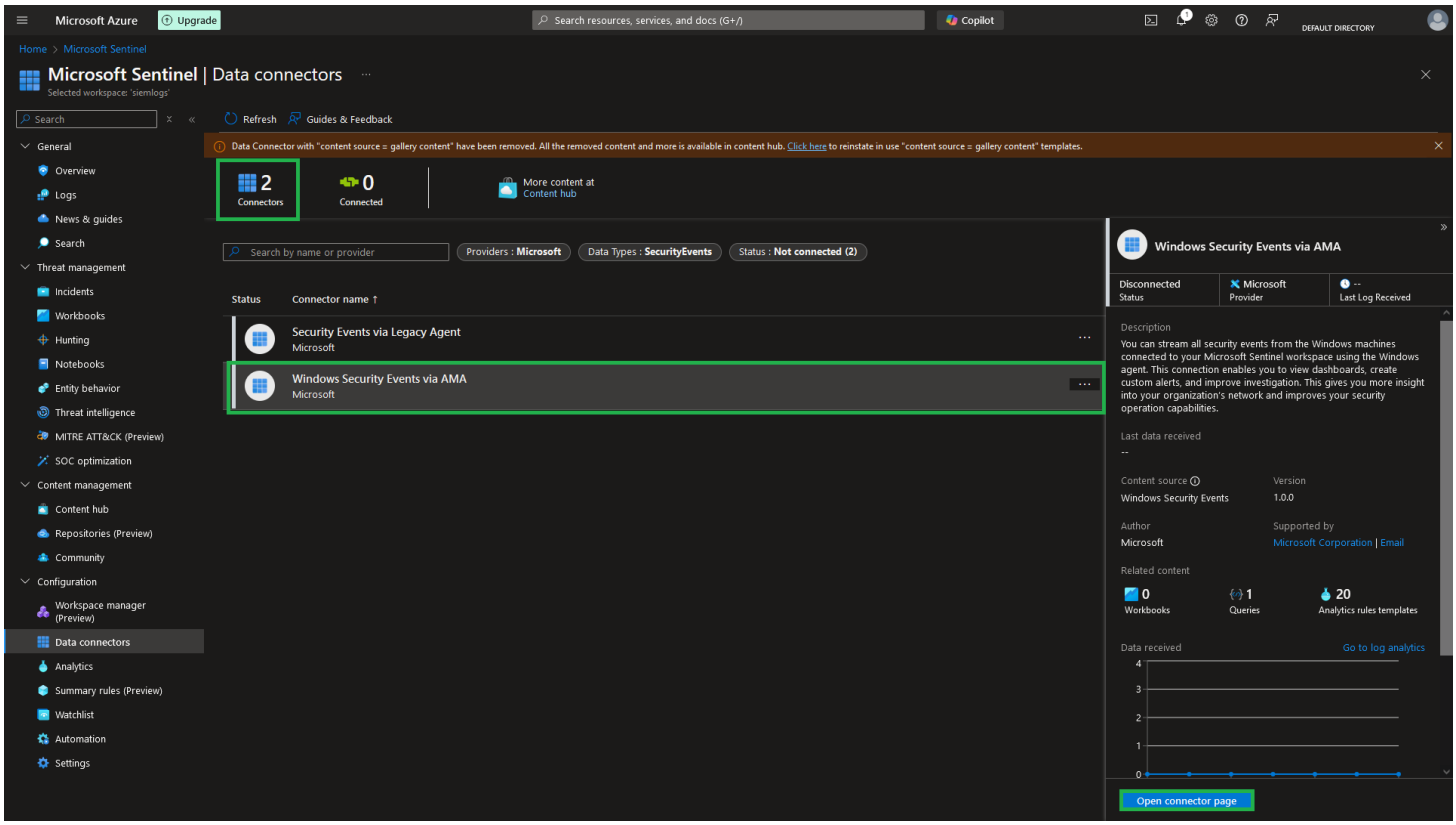
2. To remedy this, we have to set up a **Data connector**. This is done by going to the **Microsoft Sentinel** page, clicking on **Data connectors** under the *Configurations* blade, and going to the **Content hub**.



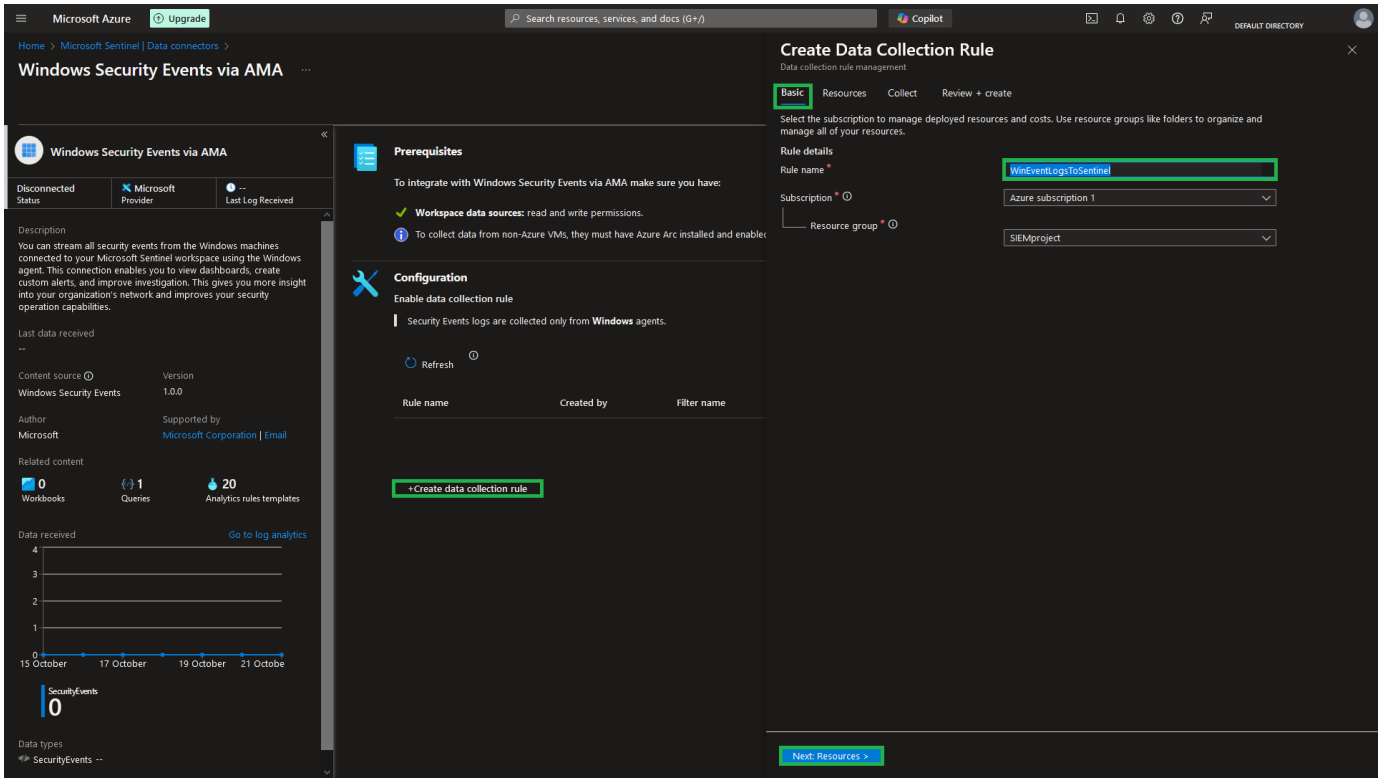
3. On the Content hub page, we input “Azure Monitor Agent” into the search box, select **Windows Security Events** from the results and click on **Install**.



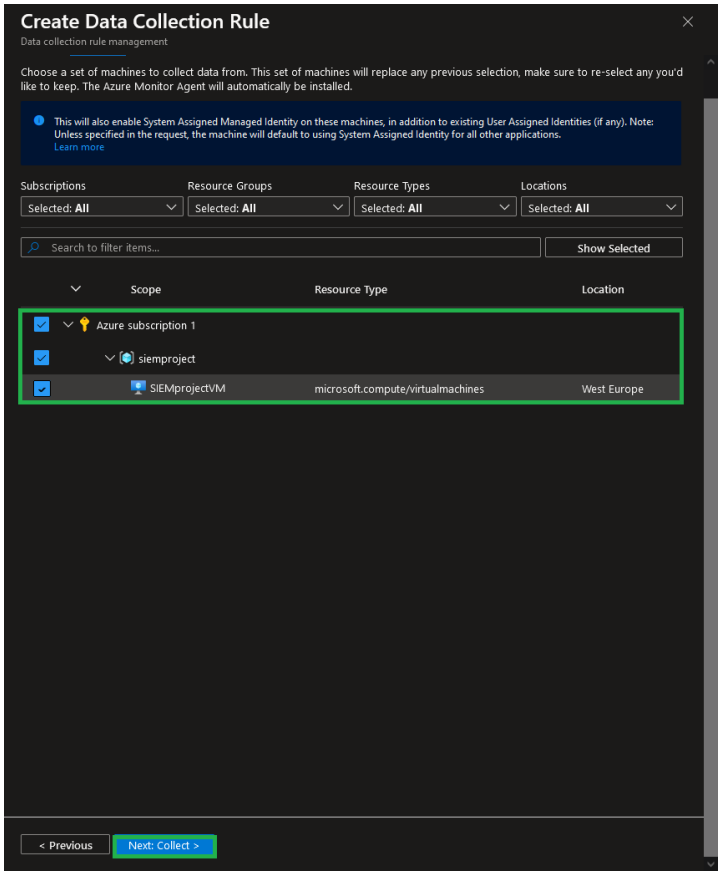
4. Two data connectors should now show up on the **Data connectors** page. We select the **Windows Security Events via AMA** connector and click on **Open connector page**.



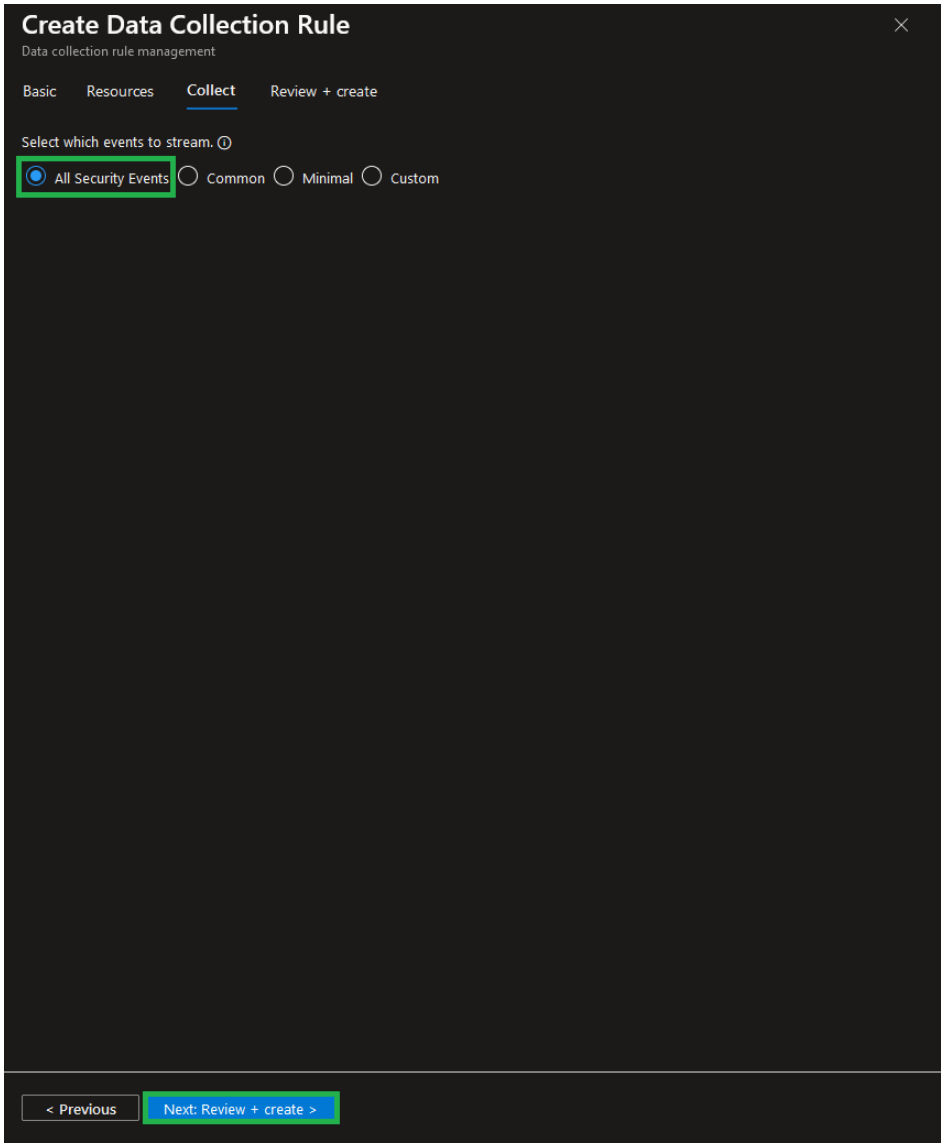
5. Once on the connector page, we click on **+Create data collection rule** and input a **Rule name** on the **Basic** tab like, for example, **WinEventLogsToSentinel**. We then click on **Next: Resources**.



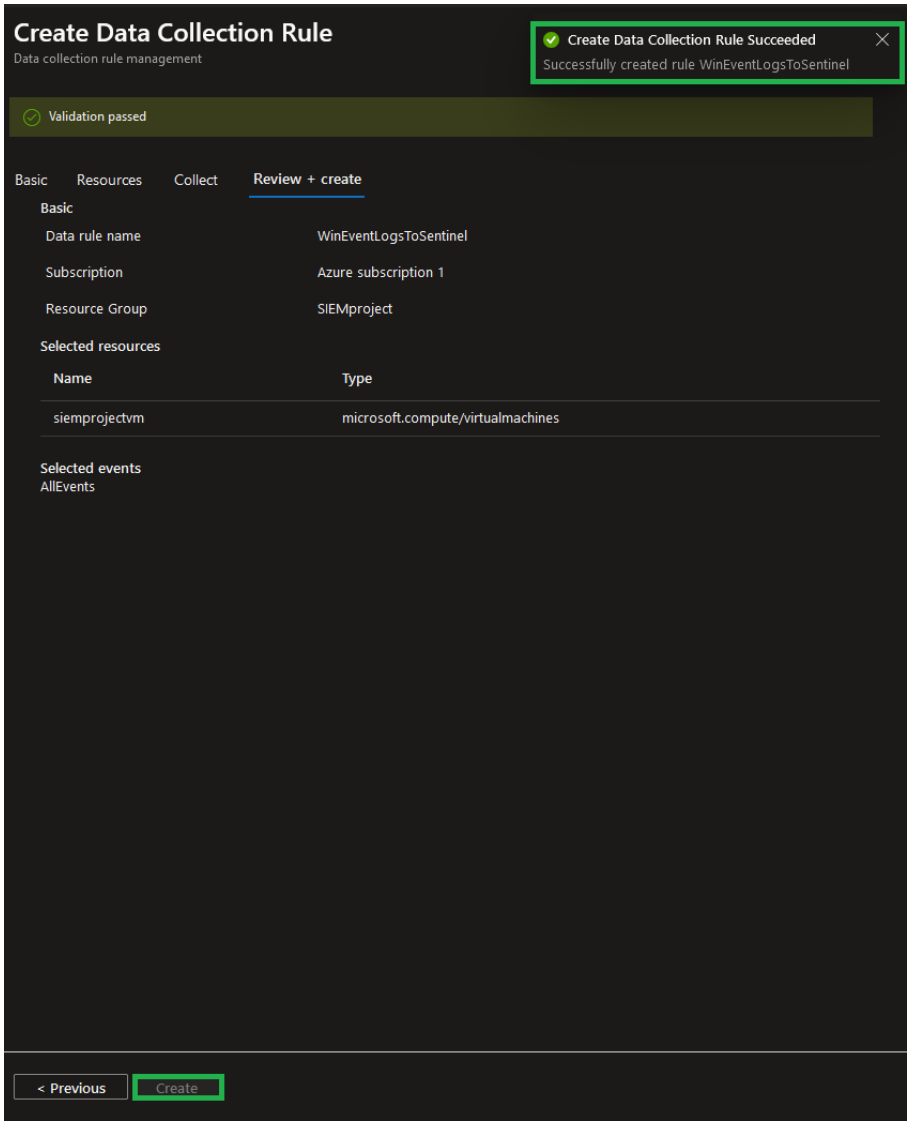
6. On the **Resources** tab, we select the subscription, resource group, and VM we wish to include and click on **Next: Collect**.



7. On the **Collect** tab we select the **All Security Events** radio button and click on **Next: Review + create**.



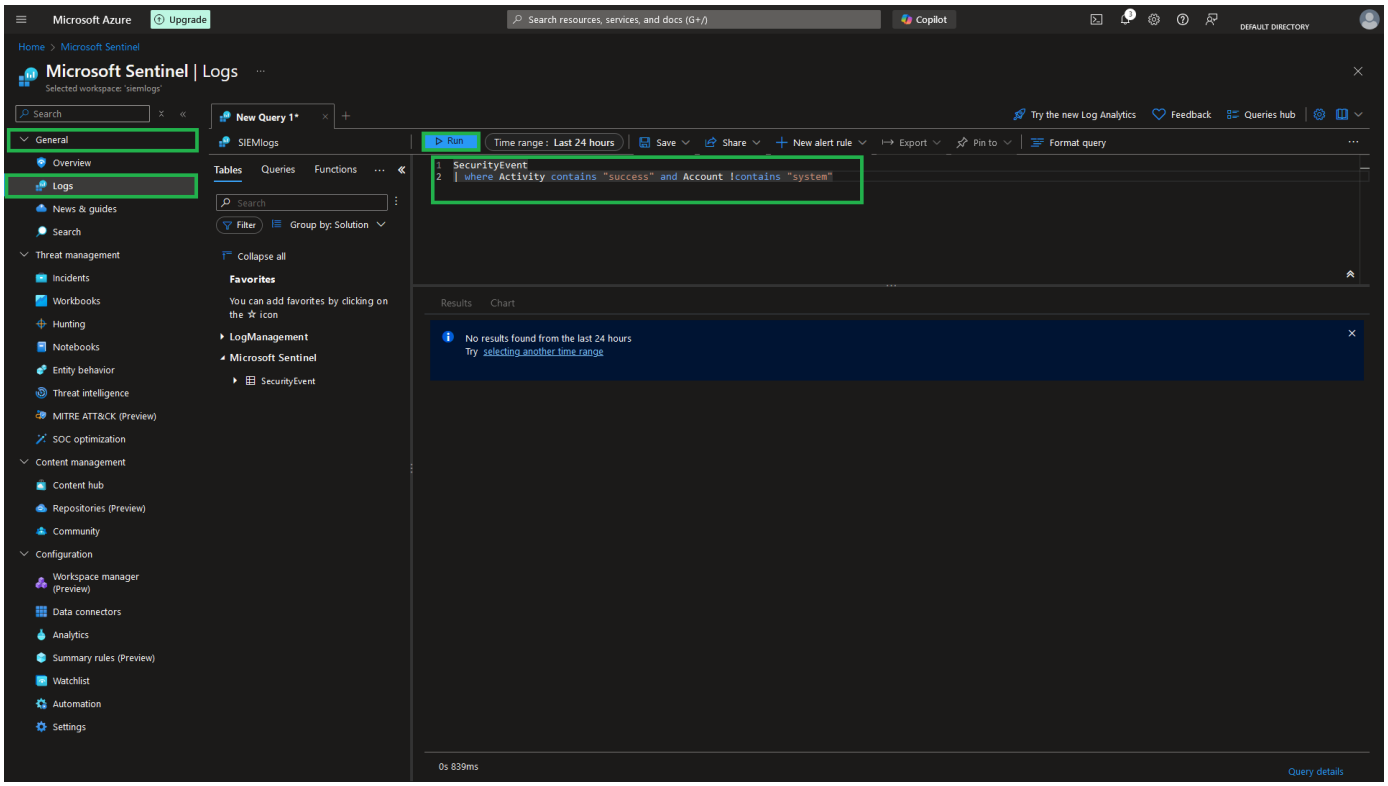
8. We then click on **Create** once more to finalize the process.



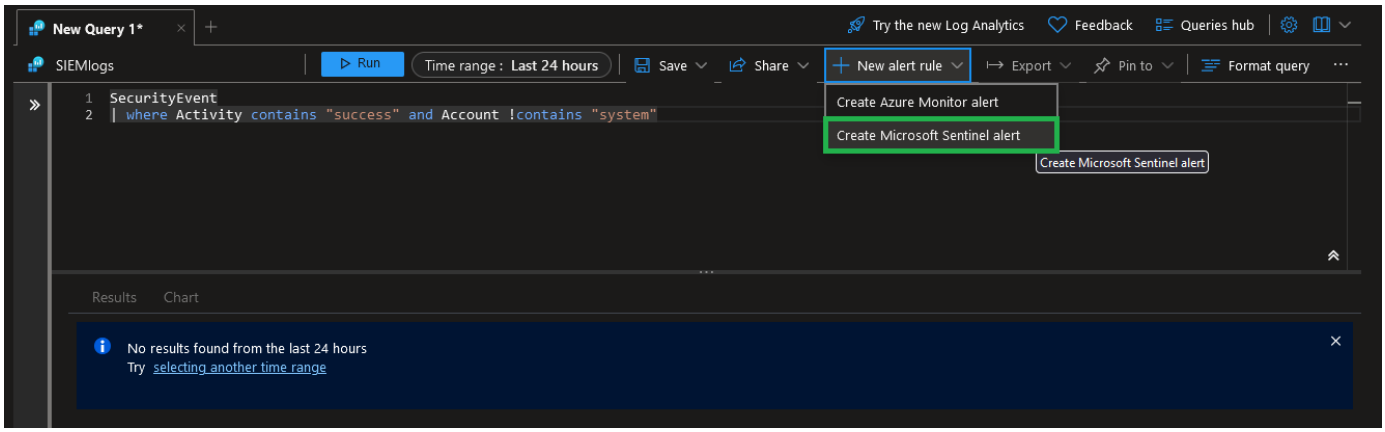
Task 4: Creating a Microsoft Sentinel scheduled rule and generating an incident

1. We now want to go back to the Microsoft Sentinel page an create a rule that check for successful sign ins via RDP. To do this, we select **Logs** under the *General* blade. We type in the following query and hit **Run**:

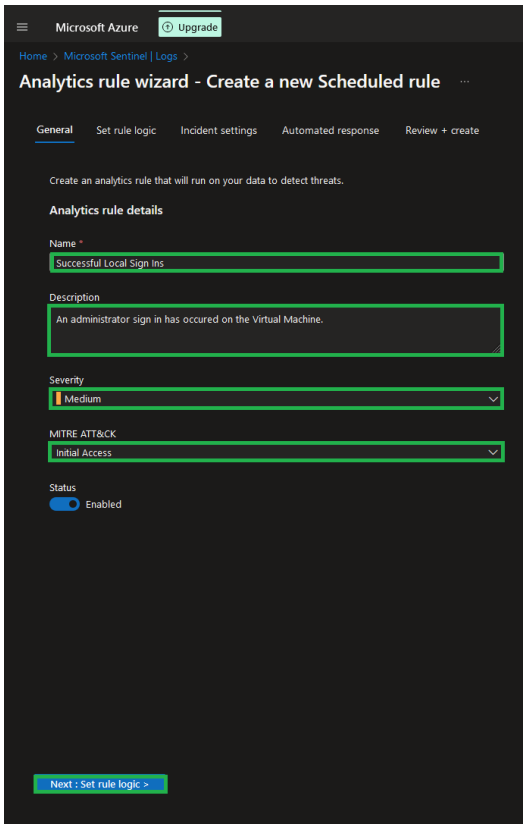
   SecurityEvent
   | where Activity contains "success" and Account !contains "system"



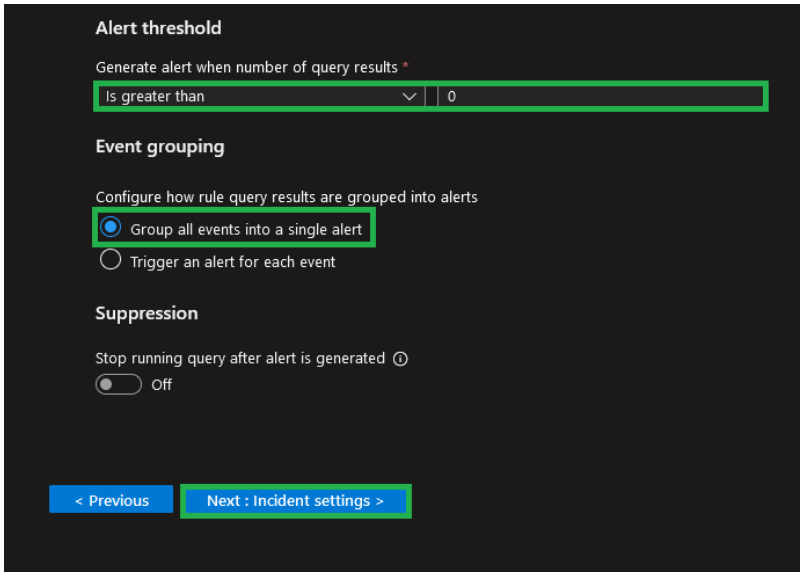
2. We then click on **New alert rule** and select **Create Microsoft Sentinel Alert**.



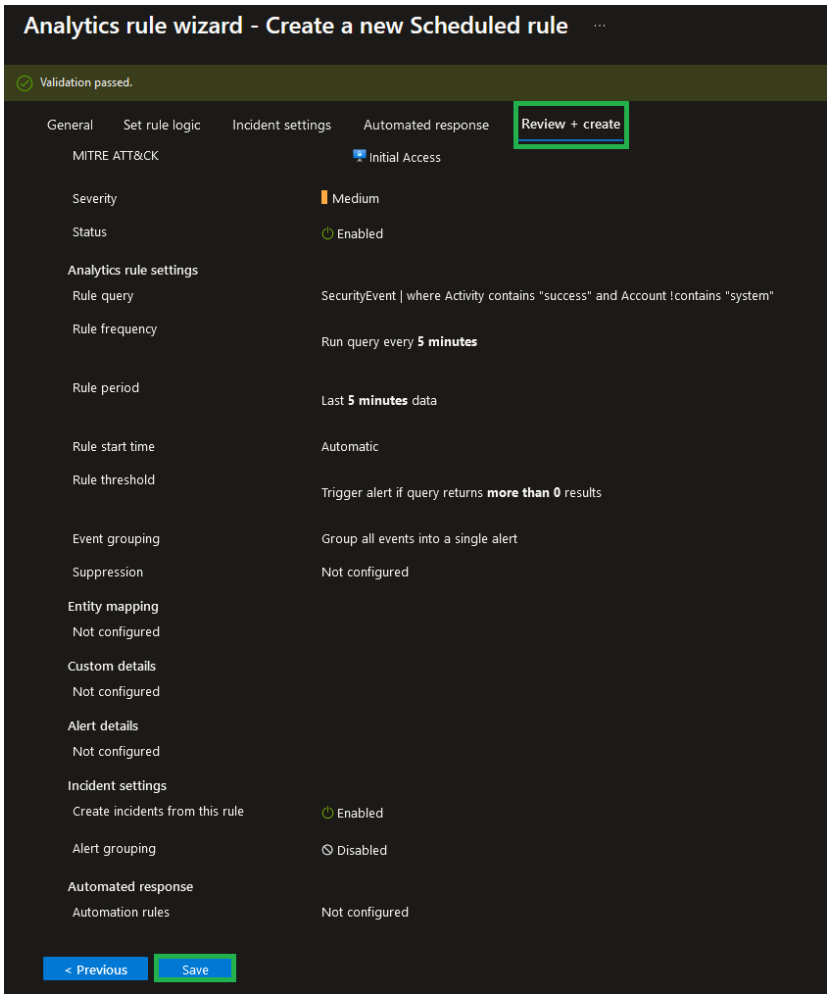
3. The **General** tab of the **Create a new Scheduled rule** page, we name our rule, choose the severity level of the event, and select **Initial Access** under **MITRE ATT&CK**. We then move on to the **Set rule logic** tab.



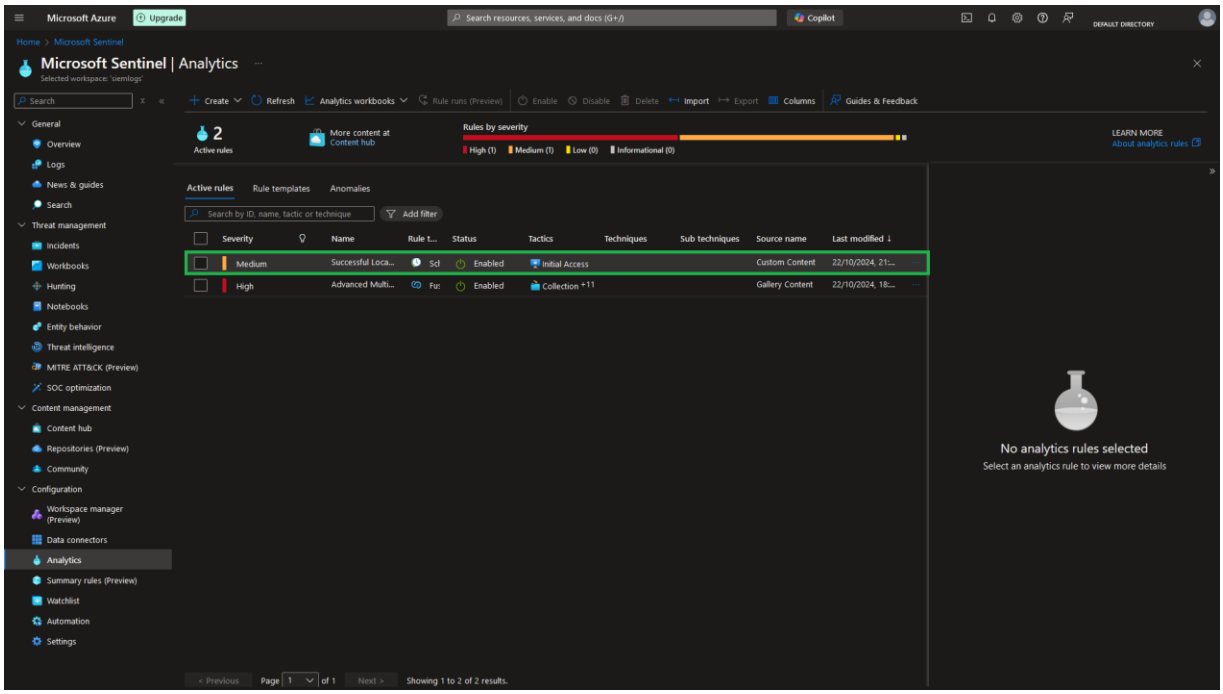
4. There, we set the **Alert threshold** to **Is greater than 0**, select the **Group all events into a single alert** radio button under **Event grouping**, and click on **Next: Incident settings**.



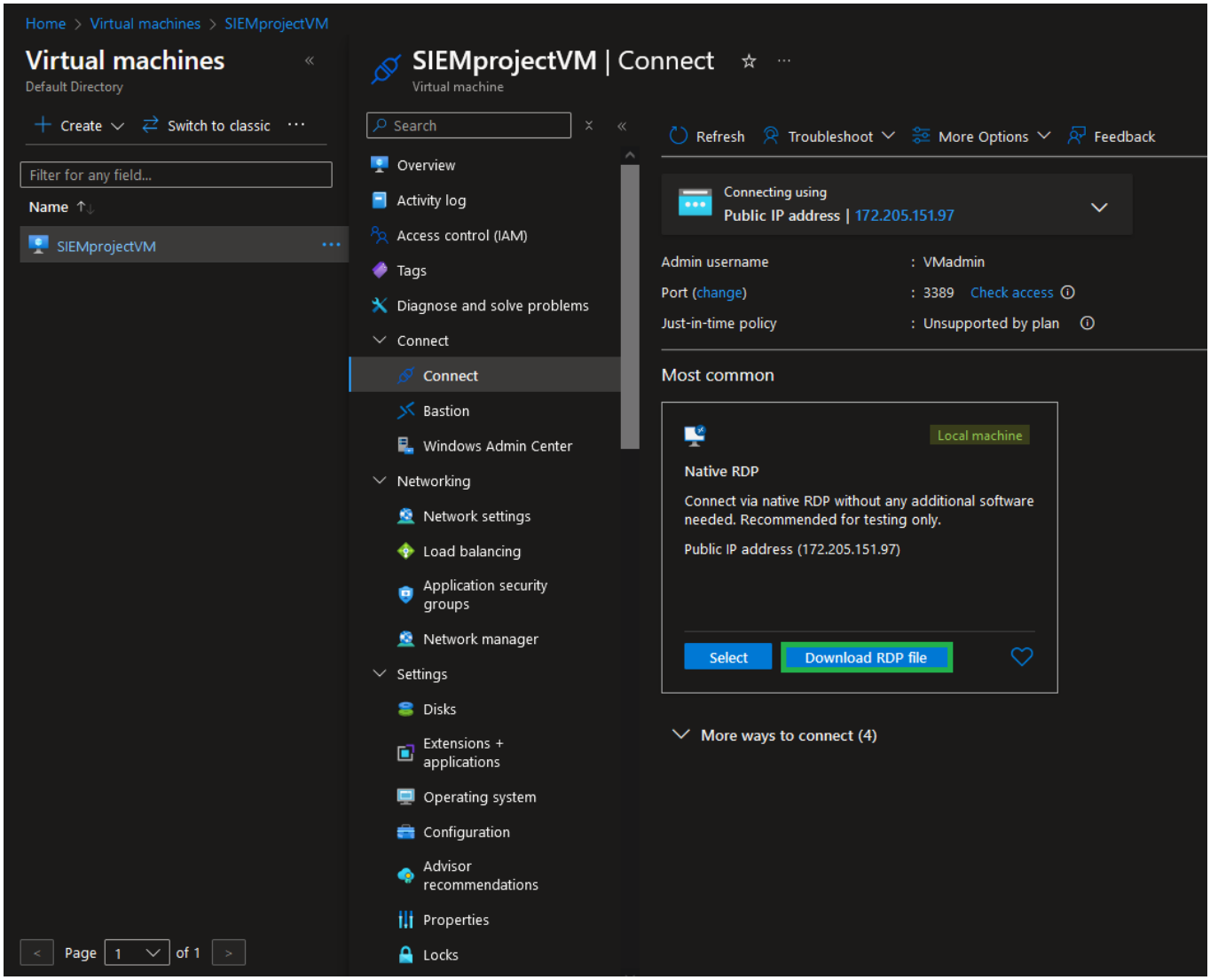
5. We leave the defaults on the **Incident settings** tab and **the Automated response** tab and go to the **Review + create** tab where we click on **Save**.



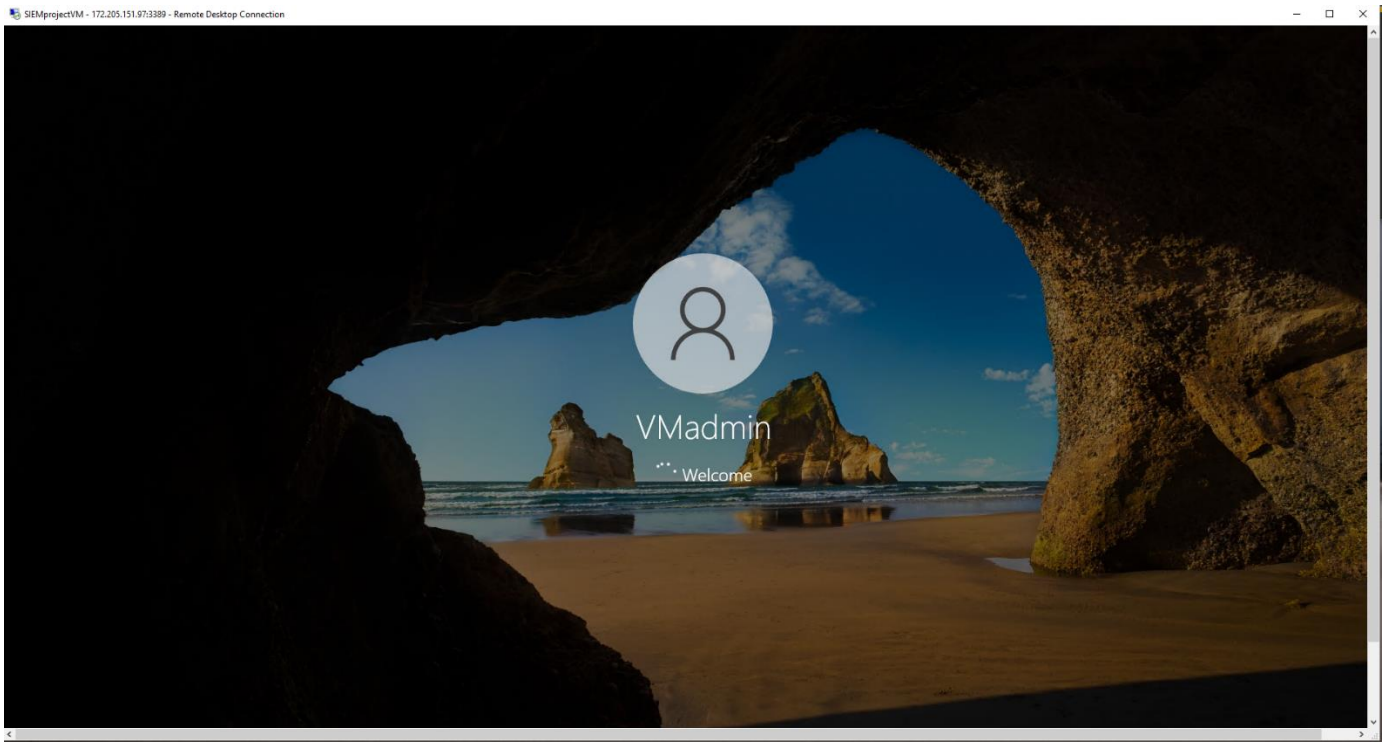
6. If we go back to the **Microsoft Sentinel** page and select **Analytics** under the *Configuration* blade, we can now see our newly created alert rule on the **Active rules** tab. Logging into our VM via RDP should trigger it.



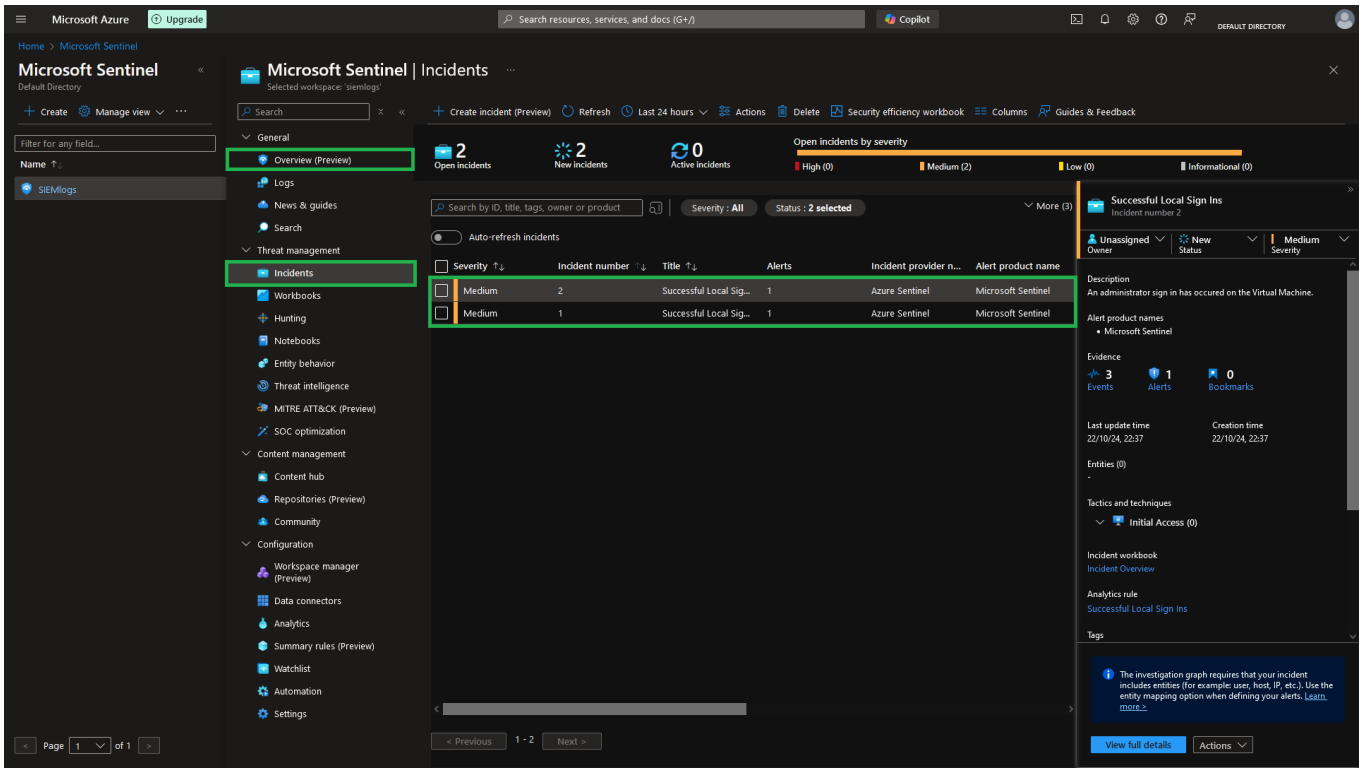
7. To test it, we go to our VM's page, go into the select **Connect** under the *Connect* blade, and click on **Download RDP file**.



8. We log in to the VM with our credentials.



9. We then go the **Incidents** (or O**verview**) page of Microsoft Sentinel to view our generated alert. It should look like this:



We can now use this VM as an RDP honeypot.

Azure security project solution PART 2:
Creating a threat intelligence feed using MISP

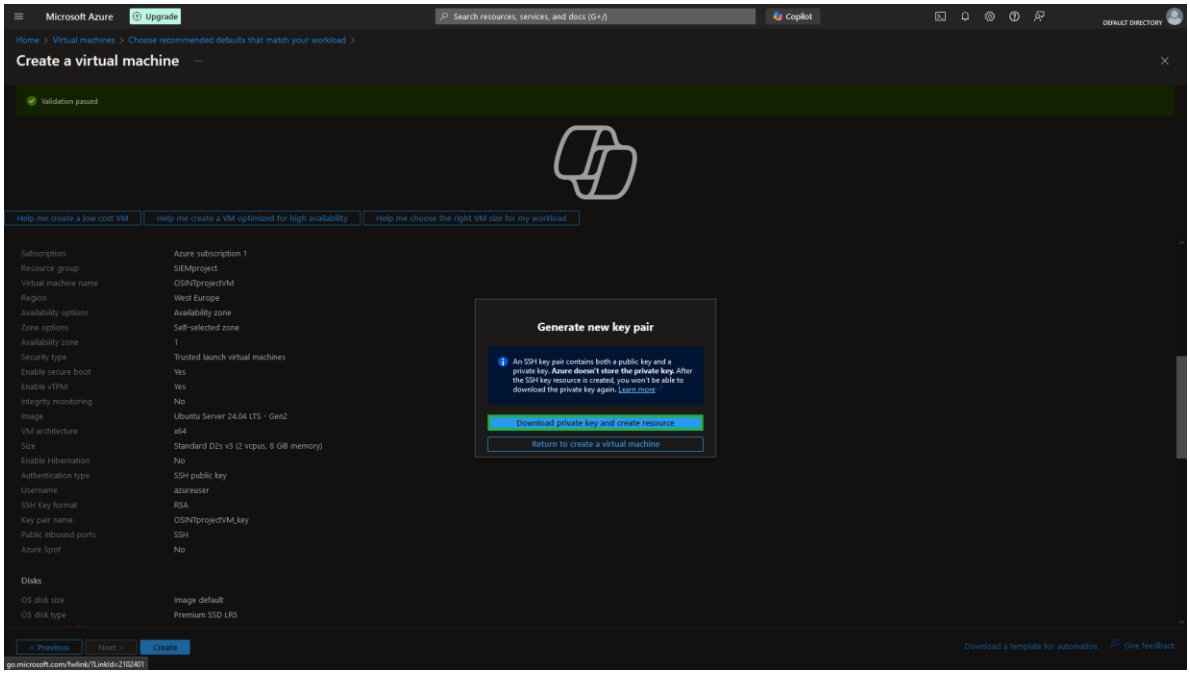
We will now setup a custom OSINT data feed to our Sentinel instance. Microsoft does have its own threat intelligence instance, but for this project we will be using the MISP open-source threat intelligence platform.

Task 5: Creating an Ubuntu server VM and setting up Docker

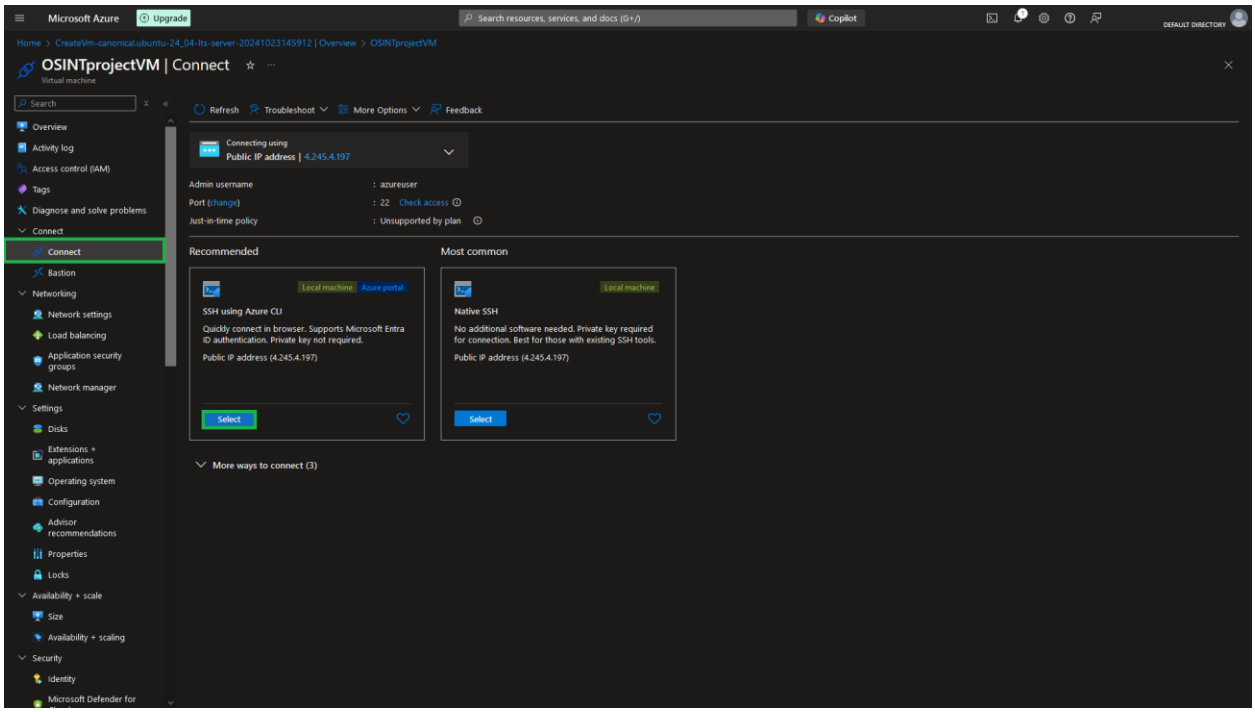
1. We begin by creating another preset virtual machine with these specifications. The rest can be leave at their default values.

Field	**Value**
Subscription	Our current subscription.
Resource group	**SIEMproject**
Virtual machine name	Enter a unique VM name, such as **OSINTprojectVM**.
Region	Select the same region as previous resources
Availability options	Select **Availability zone**.
Availability zone	Select **Zones 1**.
Image	Select **Ubuntu Server 24.04 LTS – x64 Gen2**
Size	Select Standard_D2s_v3 – 2 vcpus, 8GiB memory

2. We then click on **Download private key and create resource** when prompted.

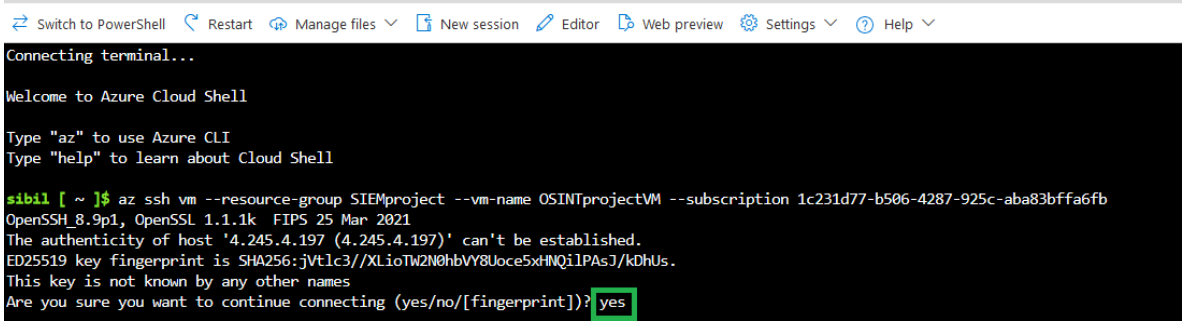


3. We then click on **Go to resource**, navigate to the **Connect** page, and click **Select** under the SSH using Azure CLI connection method.

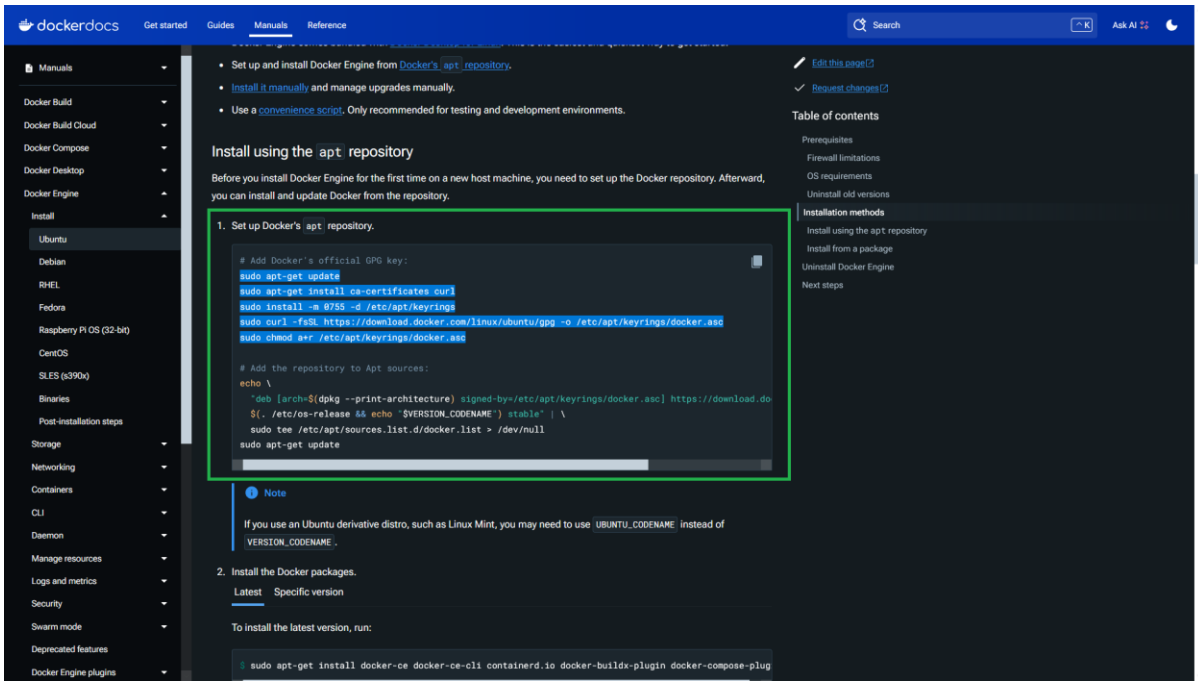


4. We then tick the checkbox beside "**I understand just-in-time policy on the virtual machine may be re-configured…**" in the wizard to continue.

5. When the shell opens, we will be prompted to confirm we would like to continue connecting and input "yes".



6. We now go to the webpage **docs.docker.com/engine/install/ubuntu/**, scroll down to the **Install using the apt repository** section and follow the steps there.



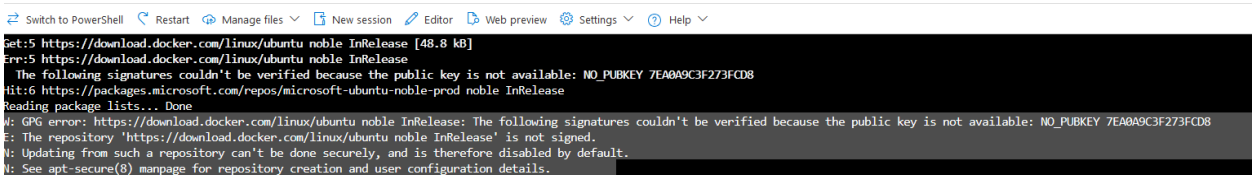
7. We first add Docker's official GPG key:

   sudo apt-get update
   sudo apt-get install ca-certificates curl
   sudo install -m 0755 -d /etc/apt/keyrings
   sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
   sudo chmod a+r /etc/apt/keyrings/docker.asc

8. We then add the repository to apt sources:

   *echo *
   "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
   *https://download.docker.com/linux/ubuntu *
   *$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | *
   sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
   sudo apt-get update

Note: in my case here I got the following error:



I fixed this in the following way:

I. **Update the GPG Key for Docker's Repository**: We need to download and add the correct public key for the Docker repository manually. We run the following commands:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

This command fetches the GPG key and stores it in /usr/share/keyrings/docker-archive-keyring.gpg.

II. **Update the Docker Repository List**: Now that we have the key, we update the Docker repository entry to use the GPG key we just added. We create or update the following repository file:

*echo *

*"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu *

$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

This ensures that the correct key is used when installing Docker.

III. **Update Package Index and Install Docker**: Now, we update our package list and install Docker:

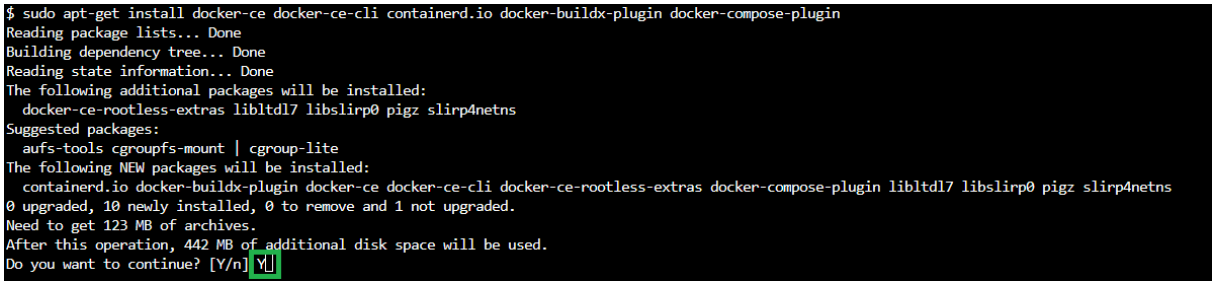
sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io

9. We can now continue with our installation of the docker packages:

   sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

   Note: input **Y**



10. If no errors are present, the response should look like this:

```
Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
$
```

11. We now run the hello-world image to verify that the installation is successful:

    sudo docker run hello-world

```
$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

$
```

Task 6: Setting up MISP

1. We can now get the MISP docker image from this URL: **github.com/MISP/misp-docker**:

   git clone https://github.com/MISP/misp-docker

```
$ git clone https://github.com/MISP/misp-docker
Cloning into 'misp-docker'...
remote: Enumerating objects: 2080, done.
remote: Counting objects: 100% (467/467), done.
remote: Compressing objects: 100% (159/159), done.
remote: Total 2080 (delta 409), reused 327 (delta 305), pack-reused 1613 (from 1)
Receiving objects: 100% (2080/2080), 454.23 KiB | 8.26 MiB/s, done.
Resolving deltas: 100% (1121/1121), done.
$
```

2. We can check whether the image is in the directory by typing *dir*.

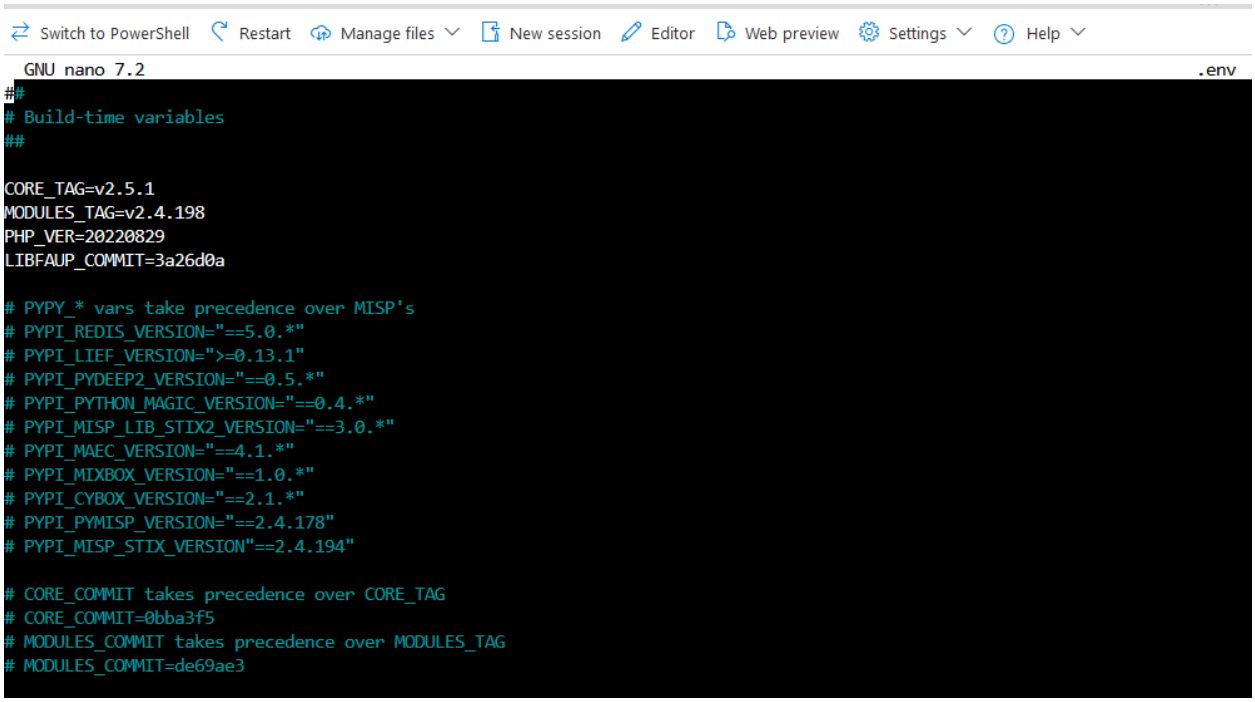
```
$ dir
misp-docker
$
```

3. The next step is to navigate to the misp-docker directory using the *cd* command.
4. We then use the dir command once more to confirm we changed directories.
5. And finally we compy the template.env file with the following command to create a new .env file:

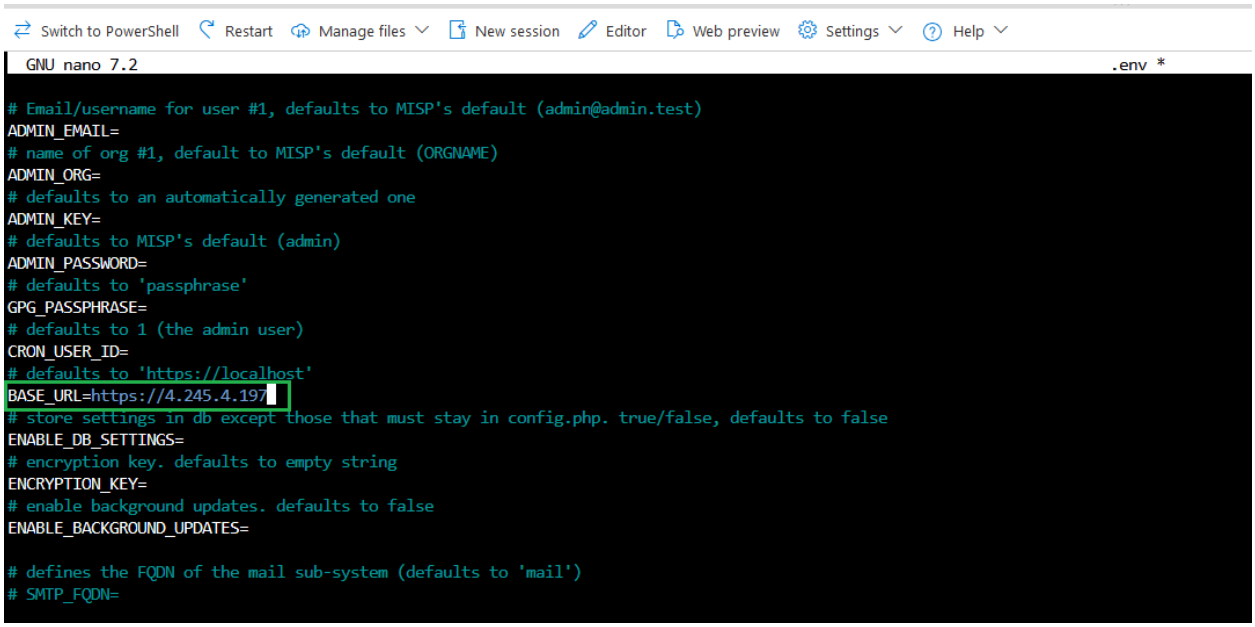
   cp template .env .env

```
$ cd misp-docker
$ dir
LICENSE  README.md  core  docker-bake.hcl  docker-compose.yml  experimental  modules  template.env
$ cp template.env .env
$
```

6. We have to now edit the new file using nano by typing in *nano .env* which should open up the editing software.

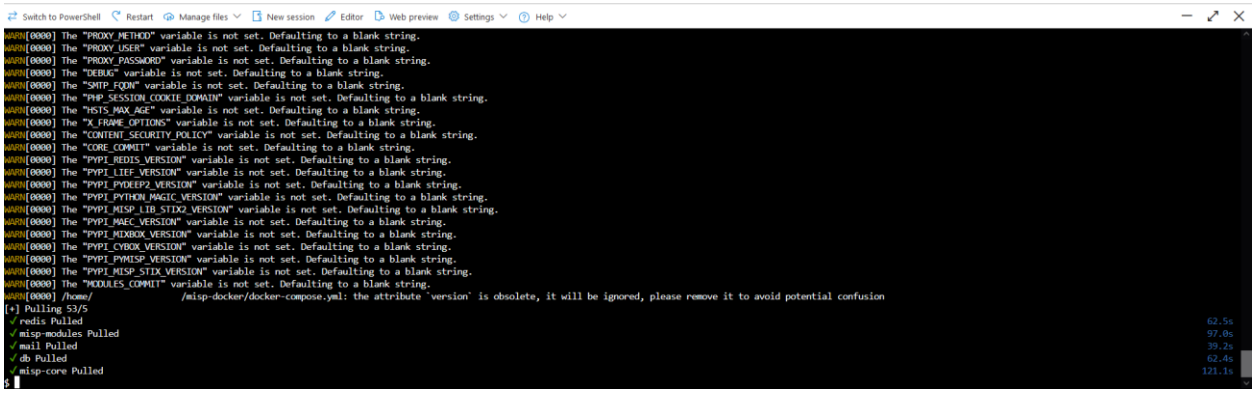


7. We scroll down to BASE_URL and input our VM's public IP address.

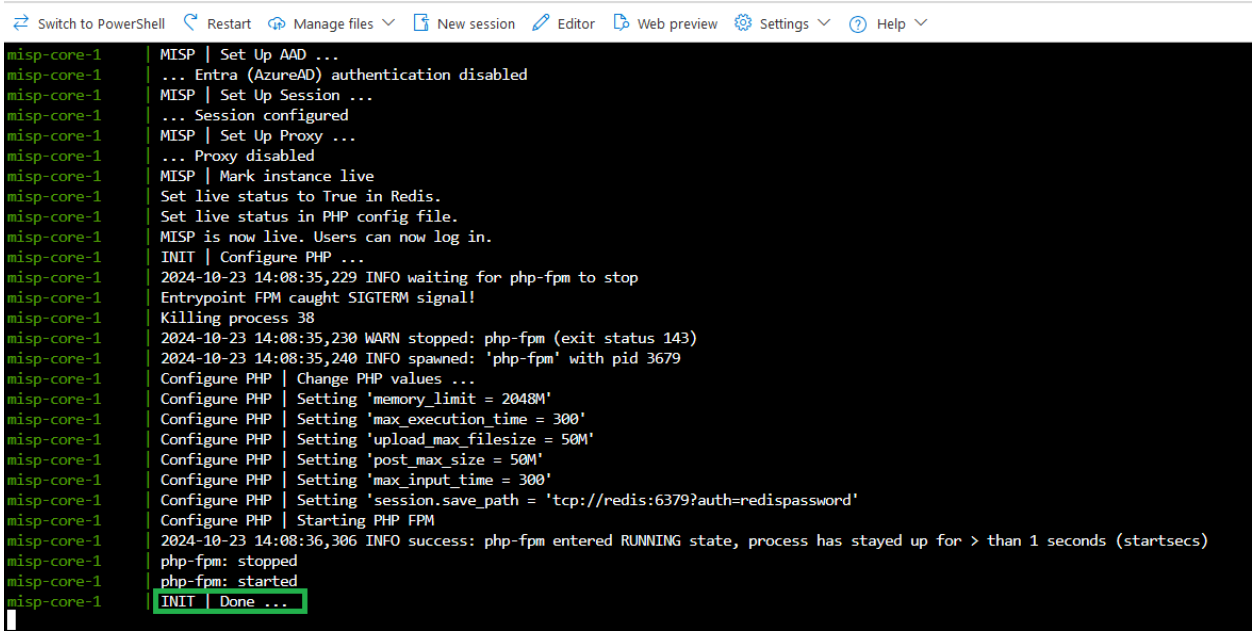


8. We then input **Ctrl+X** to Exit and will be propmpted to save our changes. We confirm with **Y** and then Press **Enter** on *the File name to Write: .env* prompt.

9. The next step is to input *sudo docker compose pull* if to use pre-built images. Once completed, it should look like this:



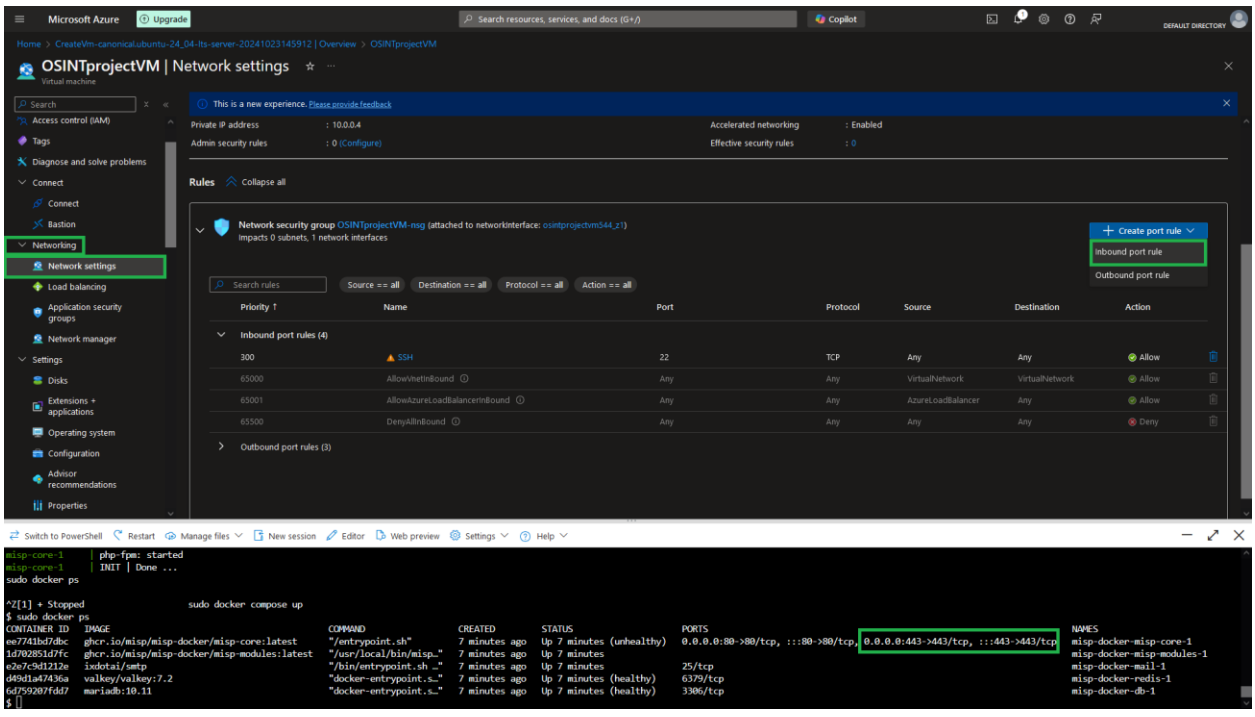
10. We then run the command *sudo docker compose up* to spin up the docker container. Successful completion should end with *INIT | Done…*



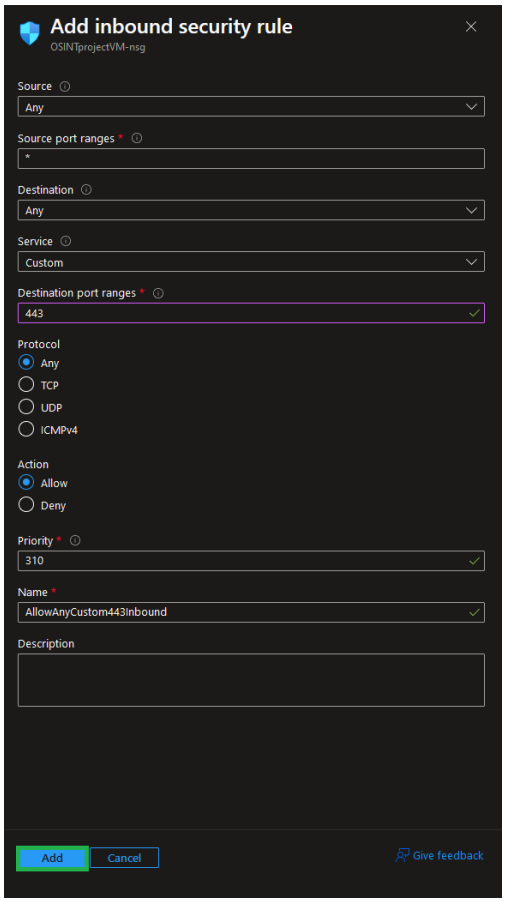
Note: Ctrl+Z might be necessary to exit the process after *INIT | Done...* is dislayed.

11. To make sure that everything successed we can check the status of the container with *sudo docker ps*.

12. We now go the **Network settings** page under the *Networking* blade of the Ubuntu VM. We need to create an inbound port rule for Port 443.

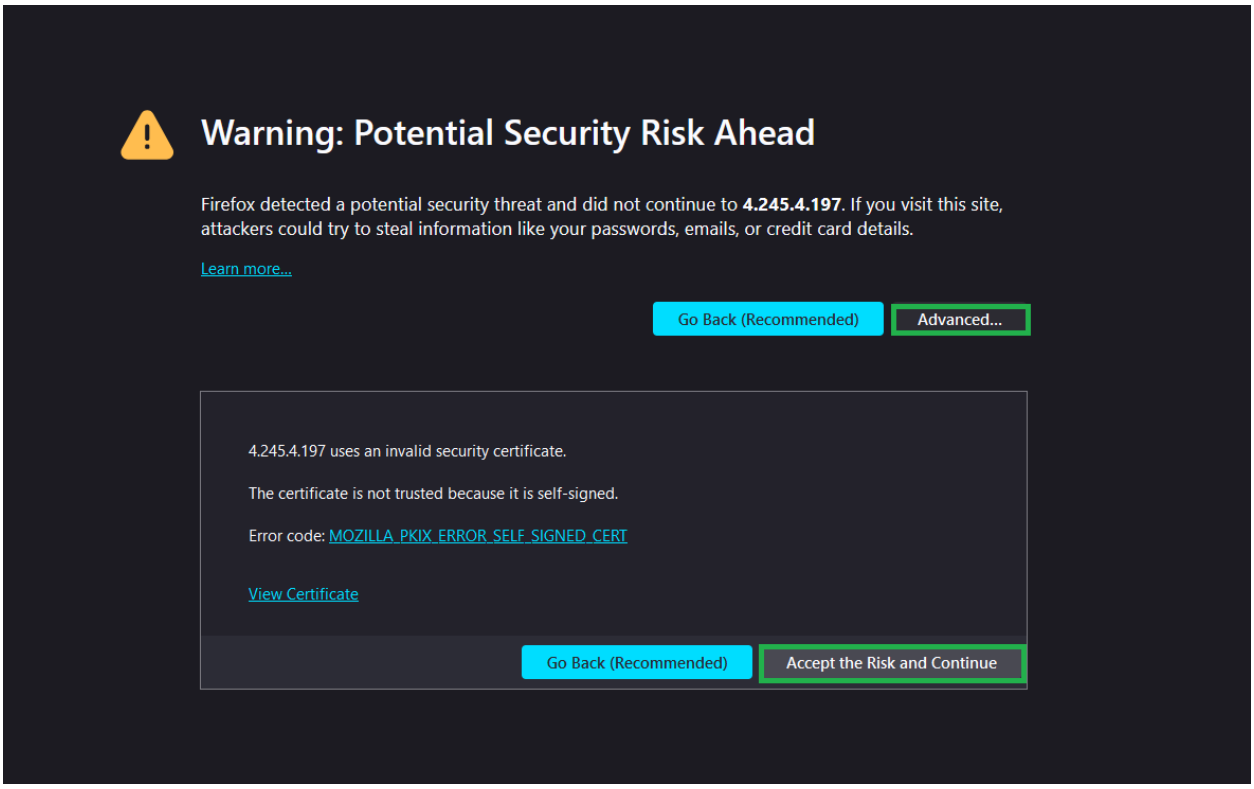


13. We input the following configuration and select **Add**:

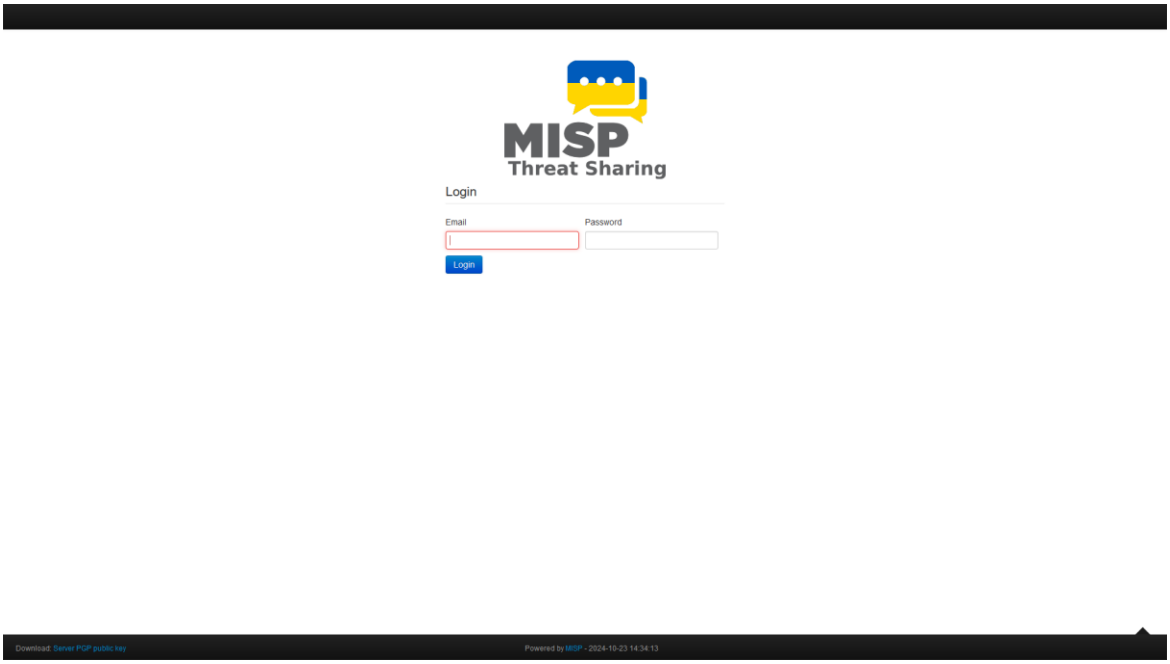


14. We then paste our VM's public IP address into a browser and bypass the warning that the page is unsafe (we need to add a signed certificate to fix this).

    Note: this should now work for any VM we have, such as the honeypot VM from PART 1.



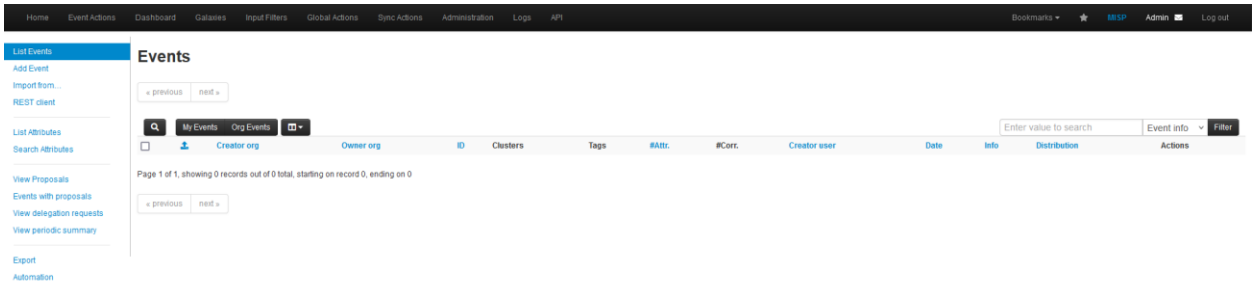
15. The MISP instance is now running.



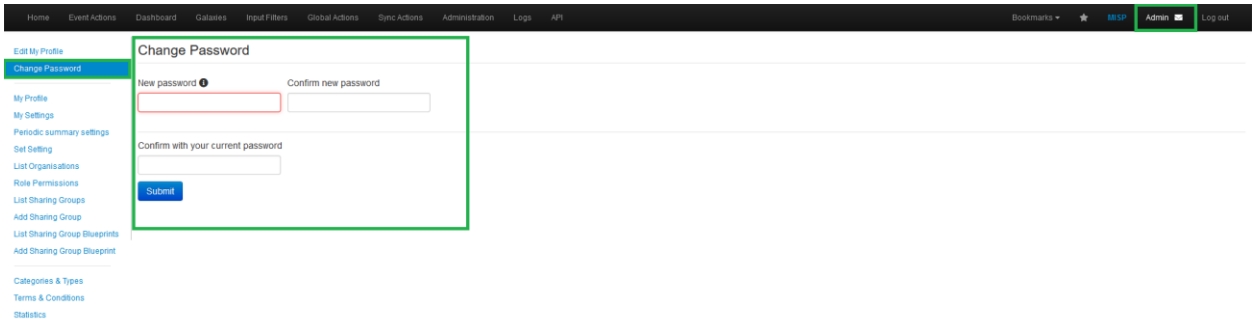
16. We log in using the following credentials:

- User: admin@admin.test
- Password: admin

17. Now we should have access to our MISP Instance.

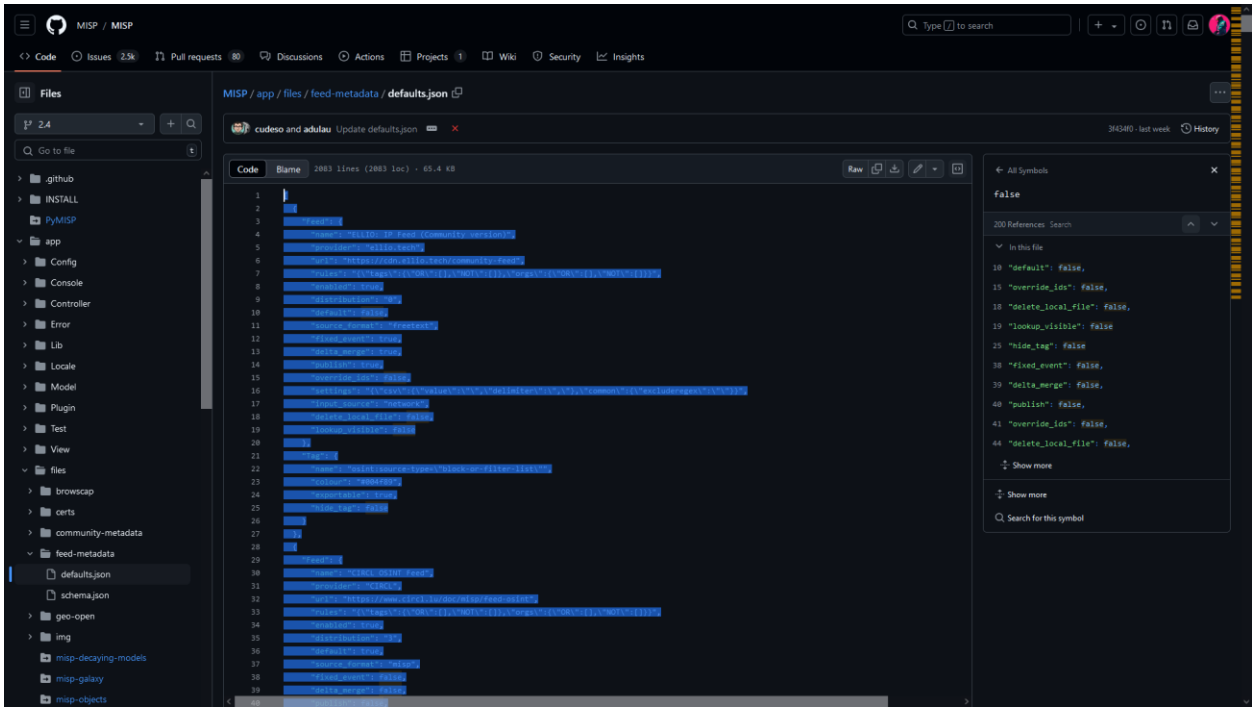


18. Since this is a public-facing server, it is highly recommended to change our admin password. This is done by clickin on **Admin** at the top-right, then going to **Change Password** and selecting new password.

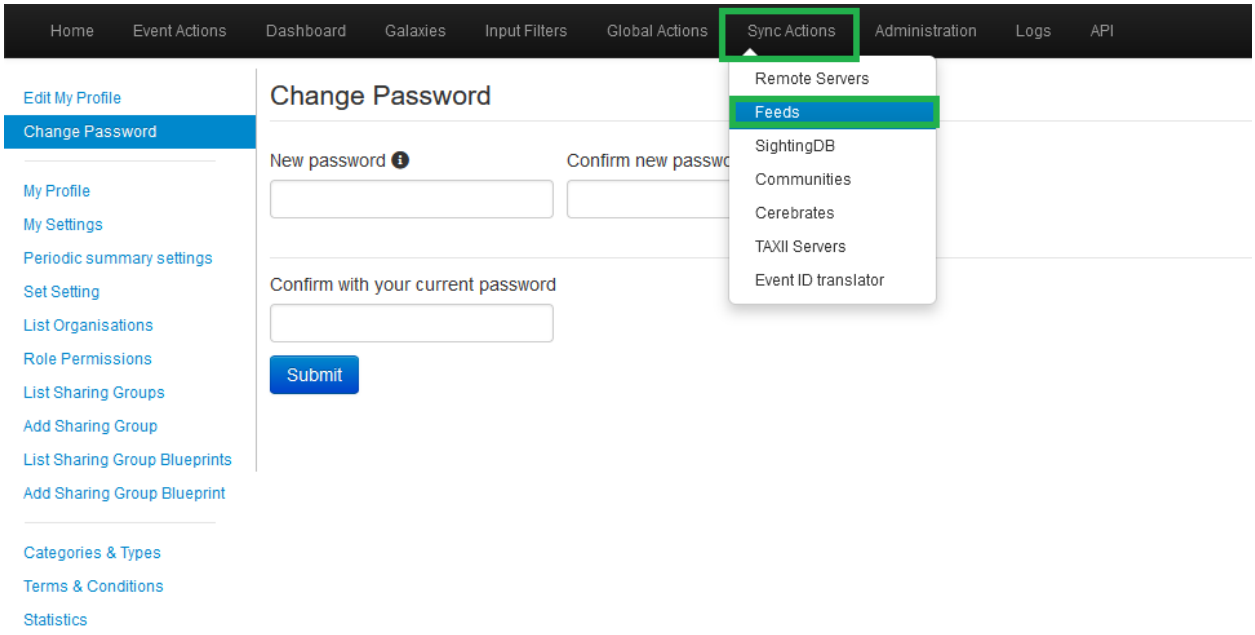


19. We now have to enable the OSINT feeds. To do this, we first go to www.misp-project.org/feeds/ or directly from GitHub at **https://github.com/MISP/MISP/blob/2.4/app/files/feed-metadata/defaults.json**.

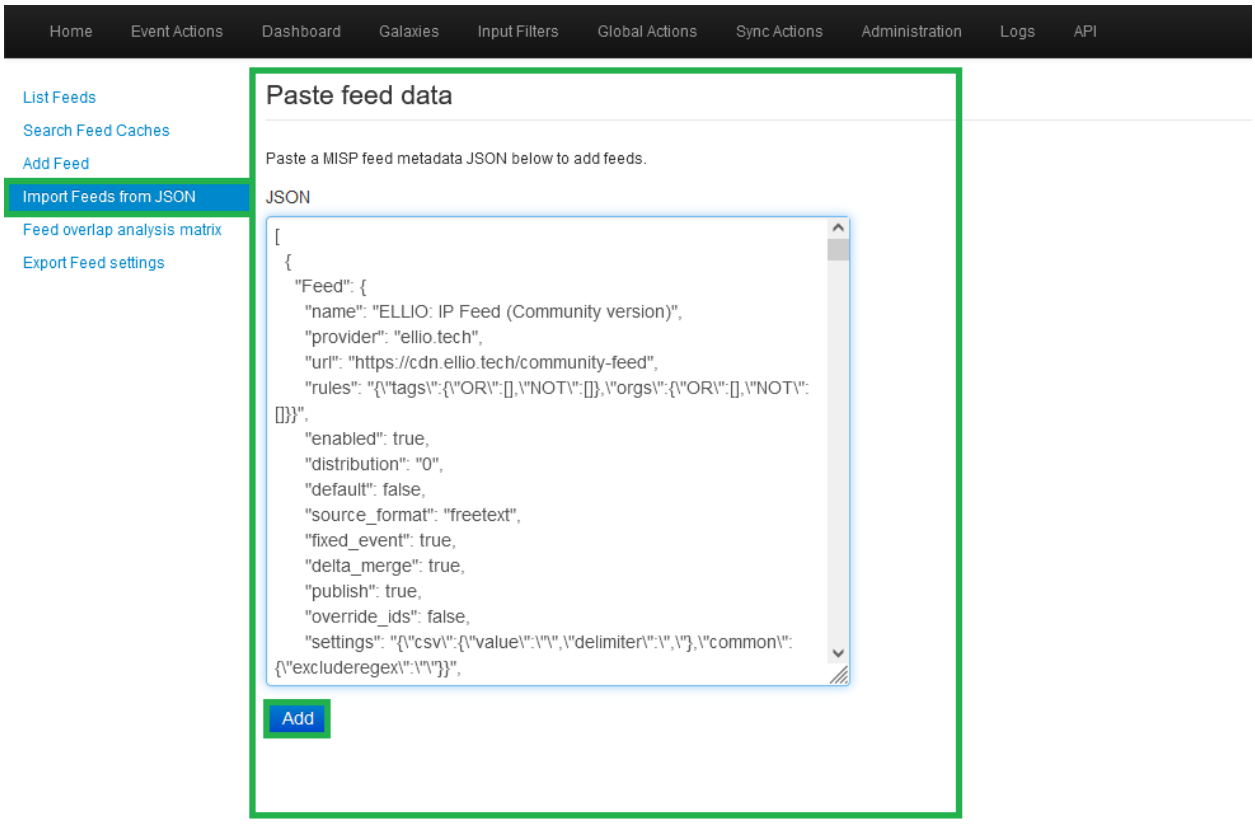
20. We copy the contents of the JSON file.



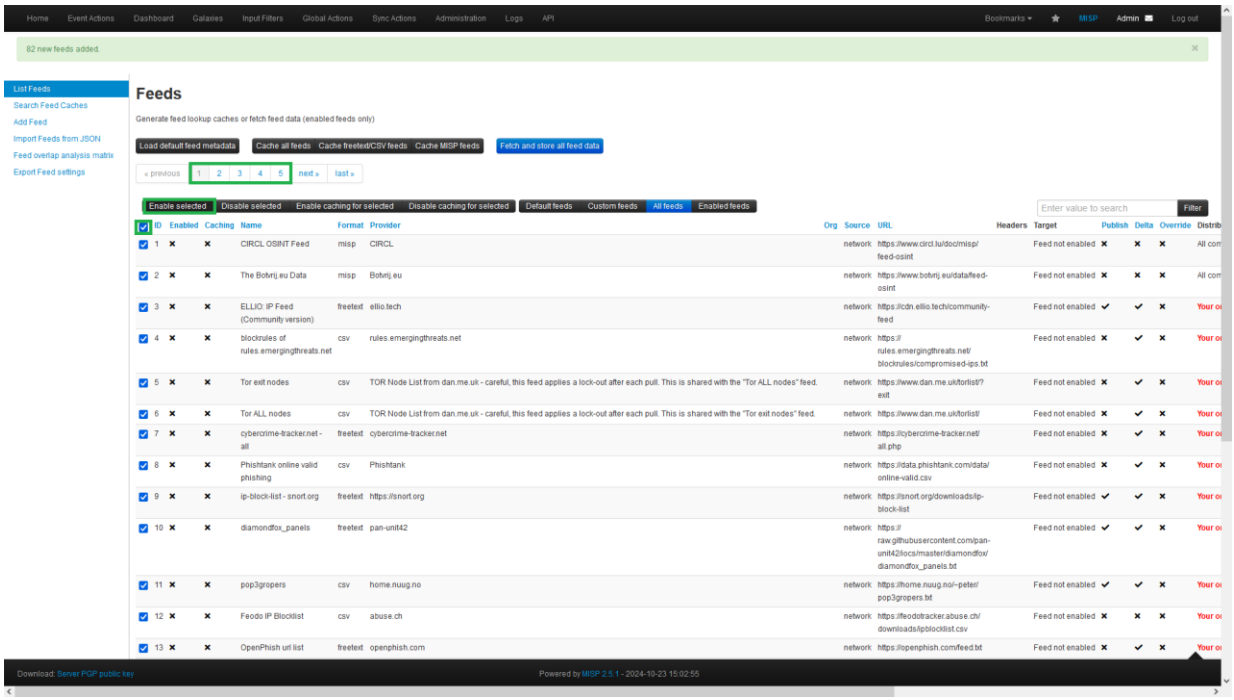
21. We then go back to the MISP instance, go to **Synch Actions**, and select **Feeds**.



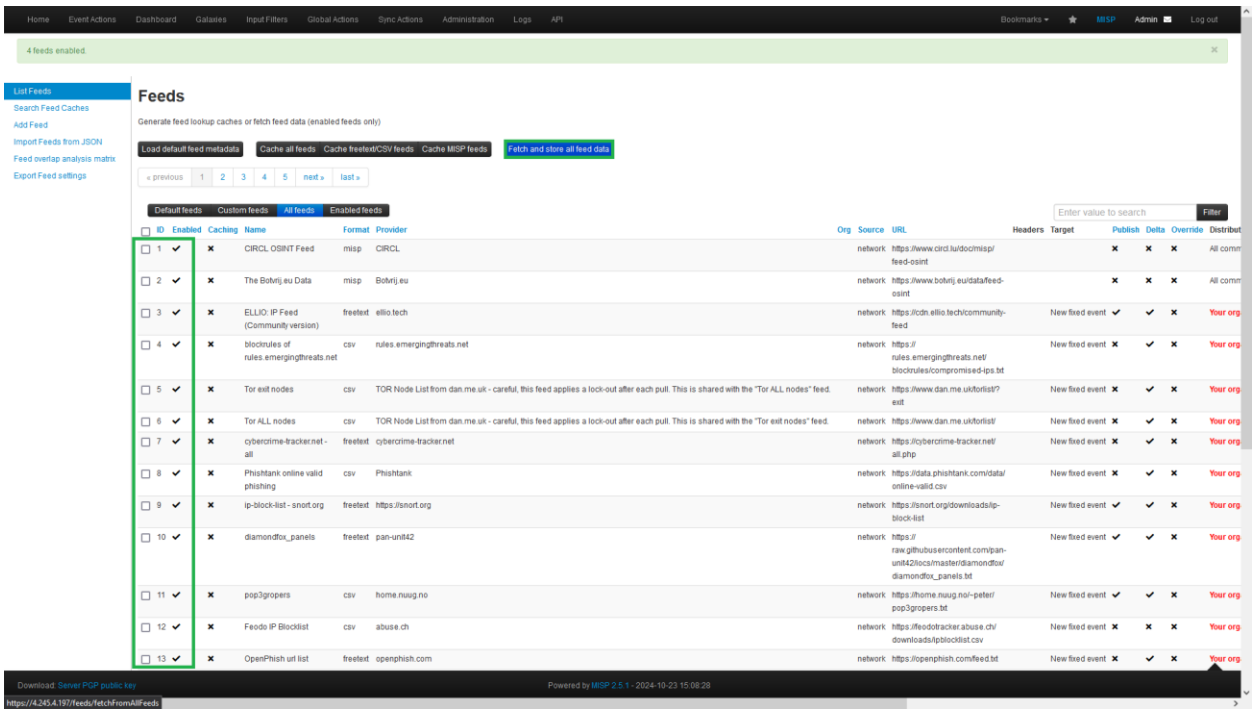
22. We go to the **Import Feeds from JSON** menu, paste the code we copied from GitHub and click on **Add**.



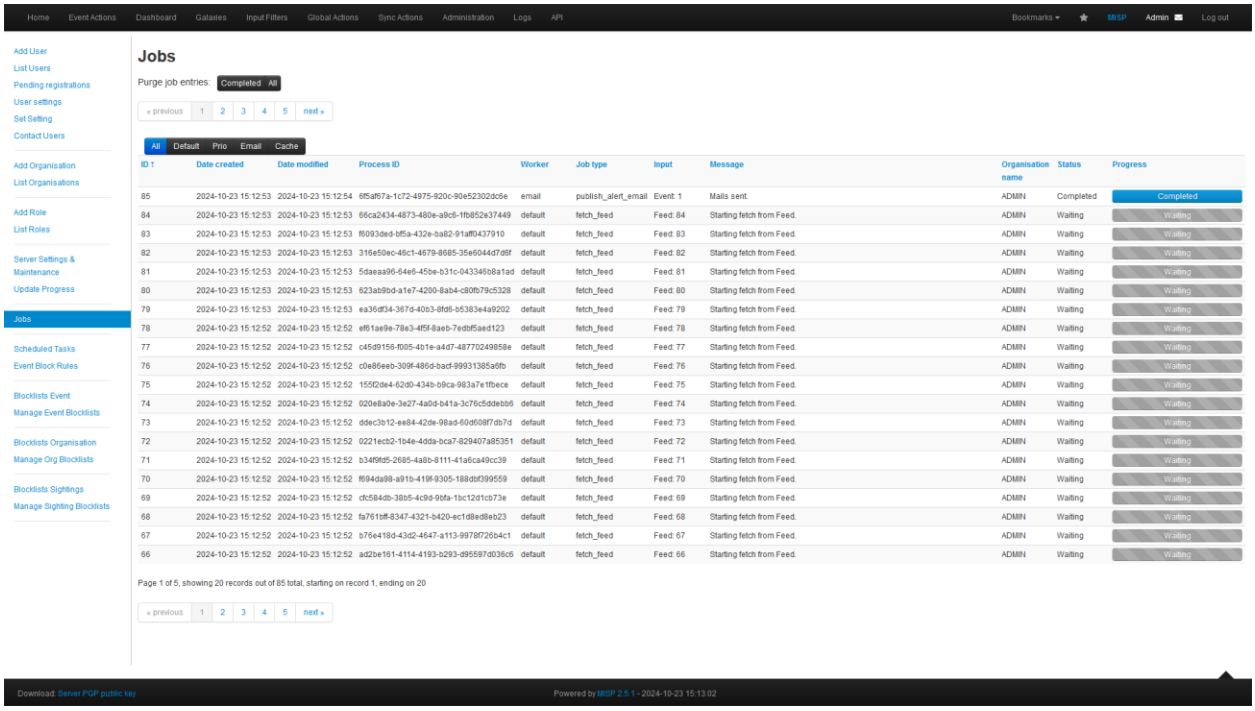
23. We now need to enable them, which is done by selecting the checkbox next to **ID** on the Feeds page that we will be automatically redirected to and clicking on **Enable selected**. We have to do this for all 5 pages individually.



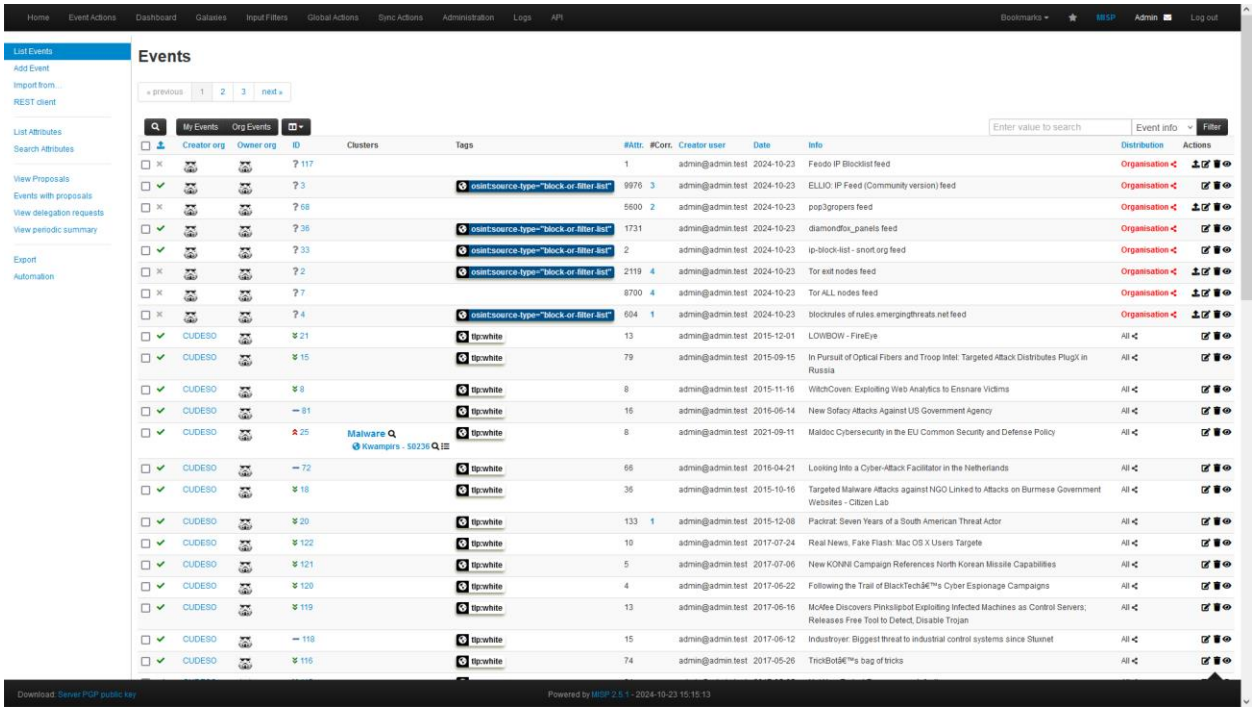
24. All feeds should now have checkmarks. We then click on **Fetch and store all feed data**.



Note: Synch status of the new feeds can be checked by hovering over the **Administration** and selecting **Jobs**.

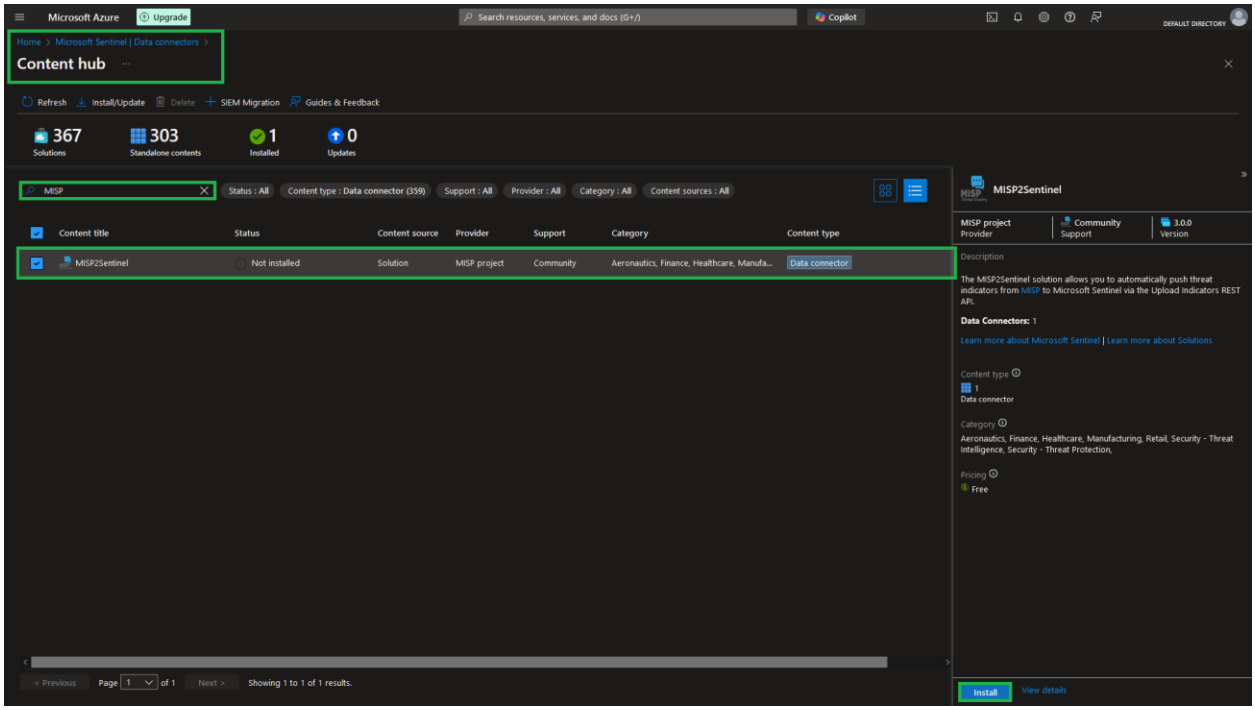


25. We can also see how our **Home** page is now being populated by feeds. We can click on the ID of any of them to see more detailed information such as indicators etc.



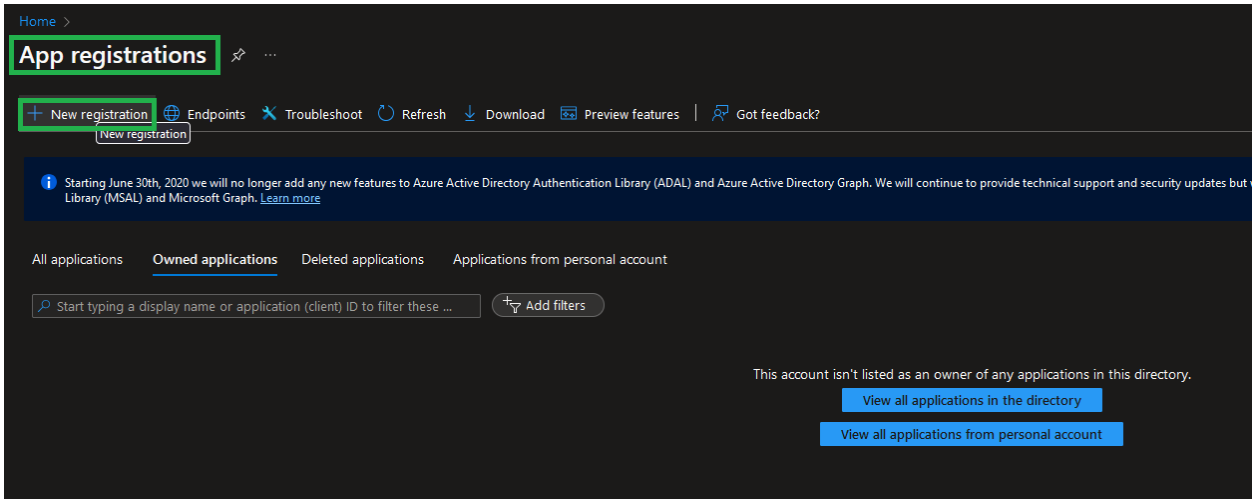
Task 7: Connecting our MISP data feed to Microsoft Sentinel

1. We once again go to the **Content Hub** from the **Data connectors** menu in Microsoft Sentinel and download the **MISP2Sentinel** preconfigured data connector and click **Install**.

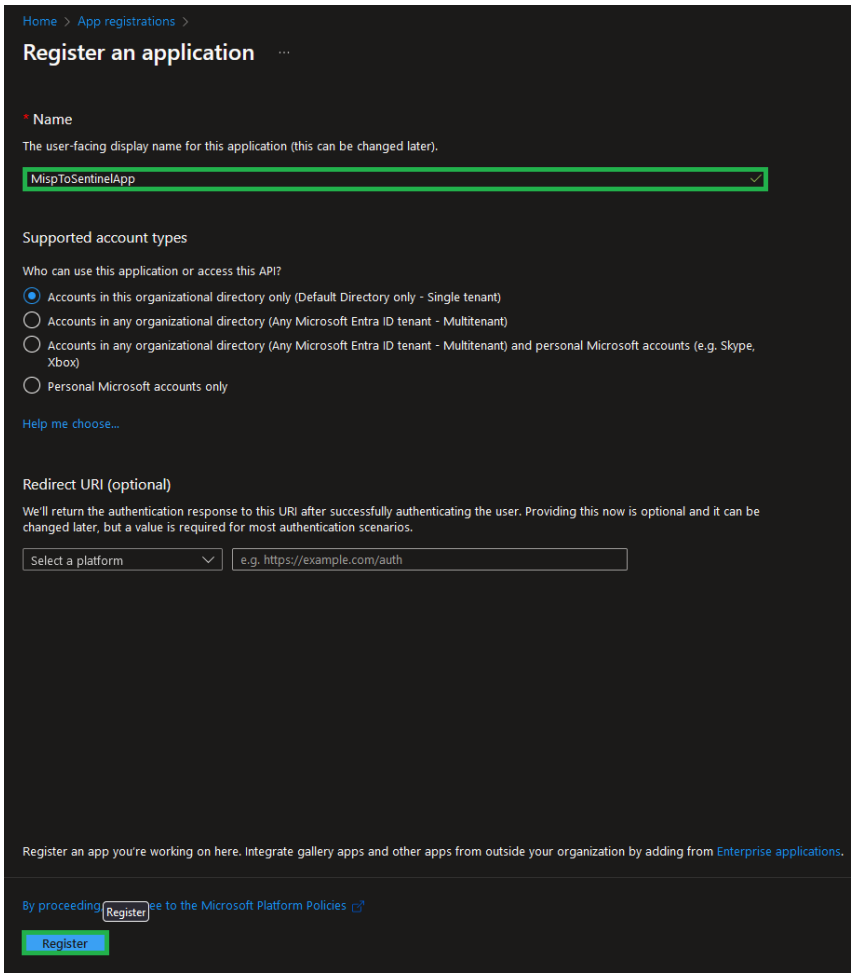


Note: We will now be following setup instructions from **github.com/cudeso/misp2sentinel**.

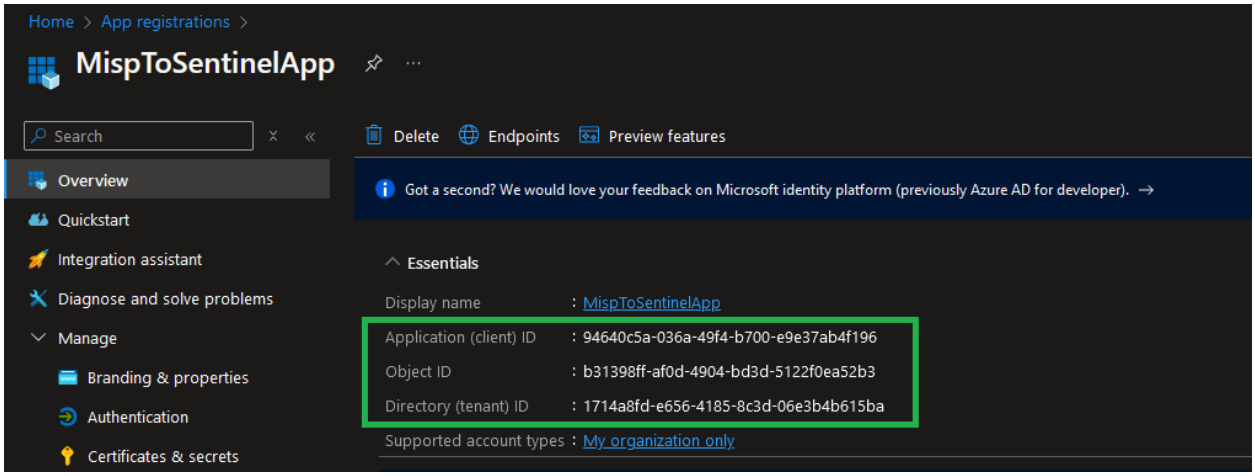
2. To use our new connector, we will set up the Upload Indicators API. We begin by registering a new application to our Azure tenant by going to the **App registrations** page using the main search bar and **selecting New registration**.



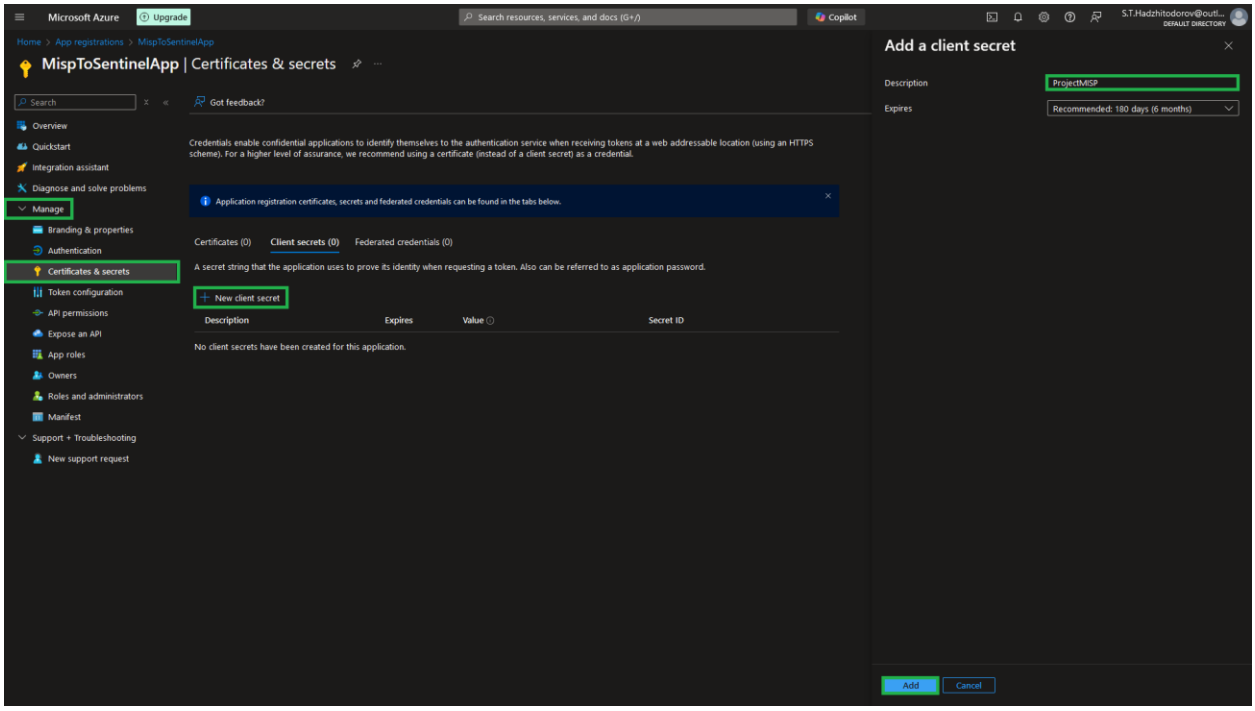
3. We name our app and click on **Register**.



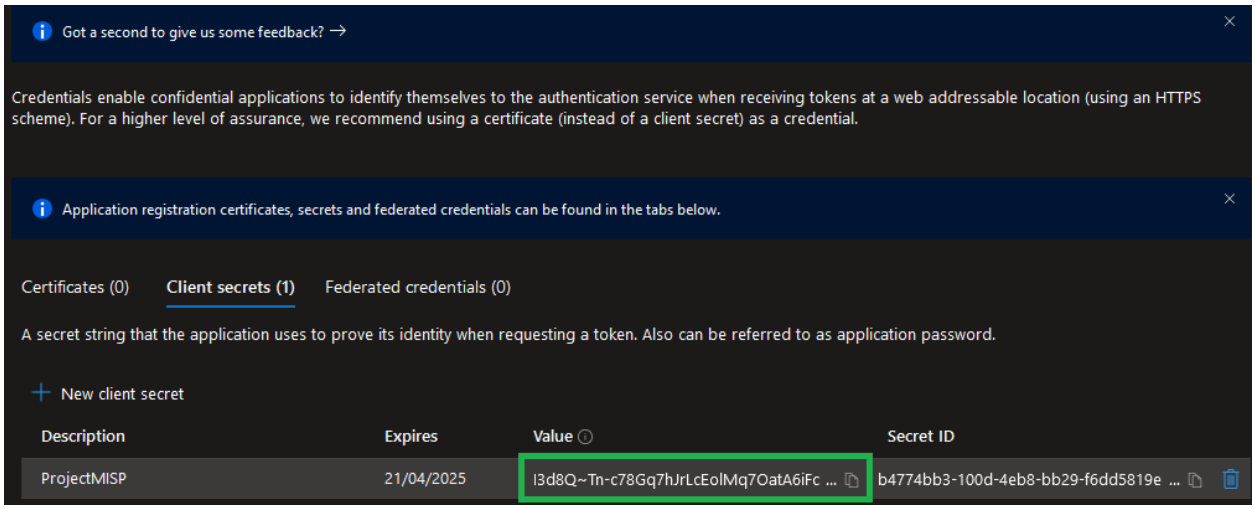
4. We take note of our *Application (client) ID*, *Object ID*, and *Directory (tenant) ID*.



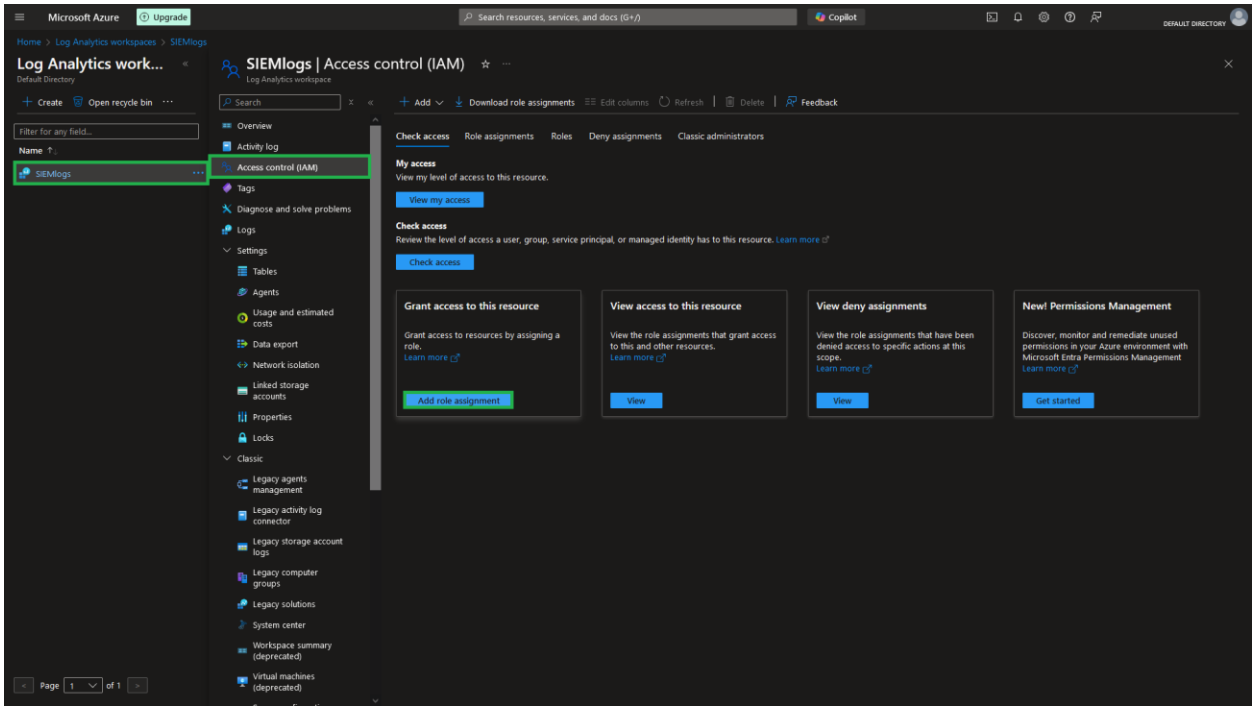
5. We then go to the **Certificates & secrets** page under the *Manage* blade, select **New client secret**, name it, and click on **Add**. We can leave the default expiry time.



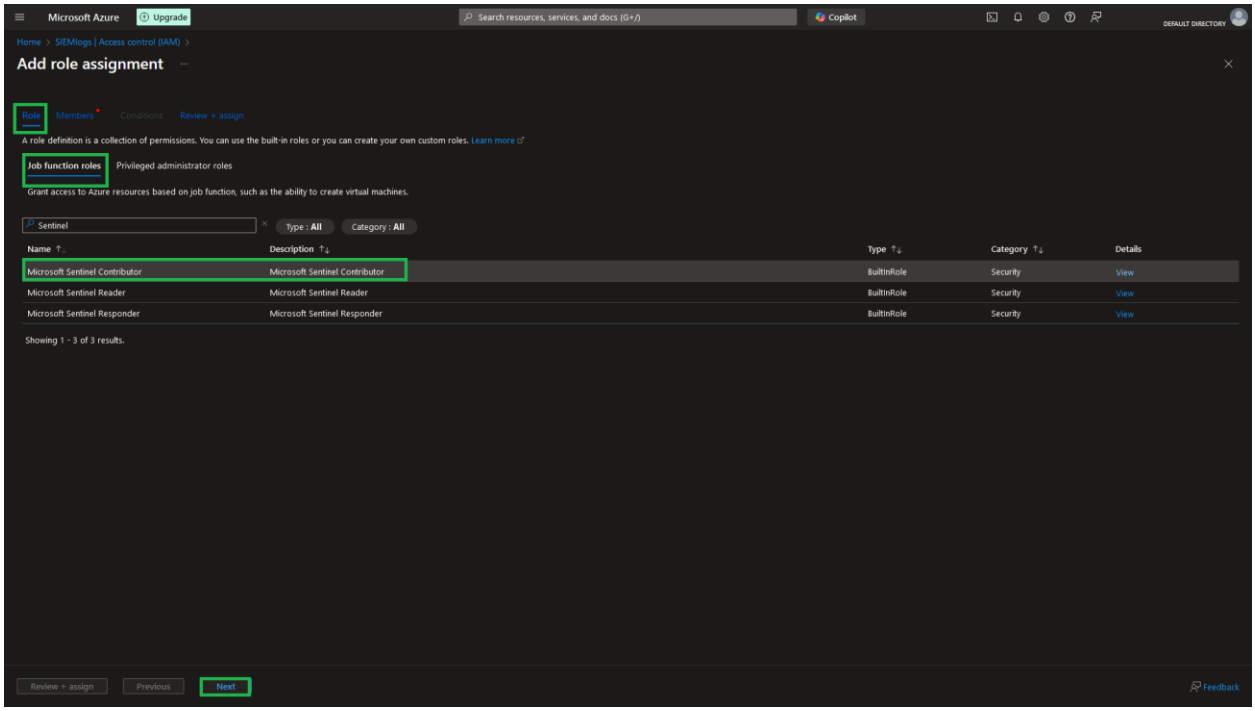
6. Now that it is created, we make sure to copy our secret's value, as it **<u>will not be shown again</u>**.



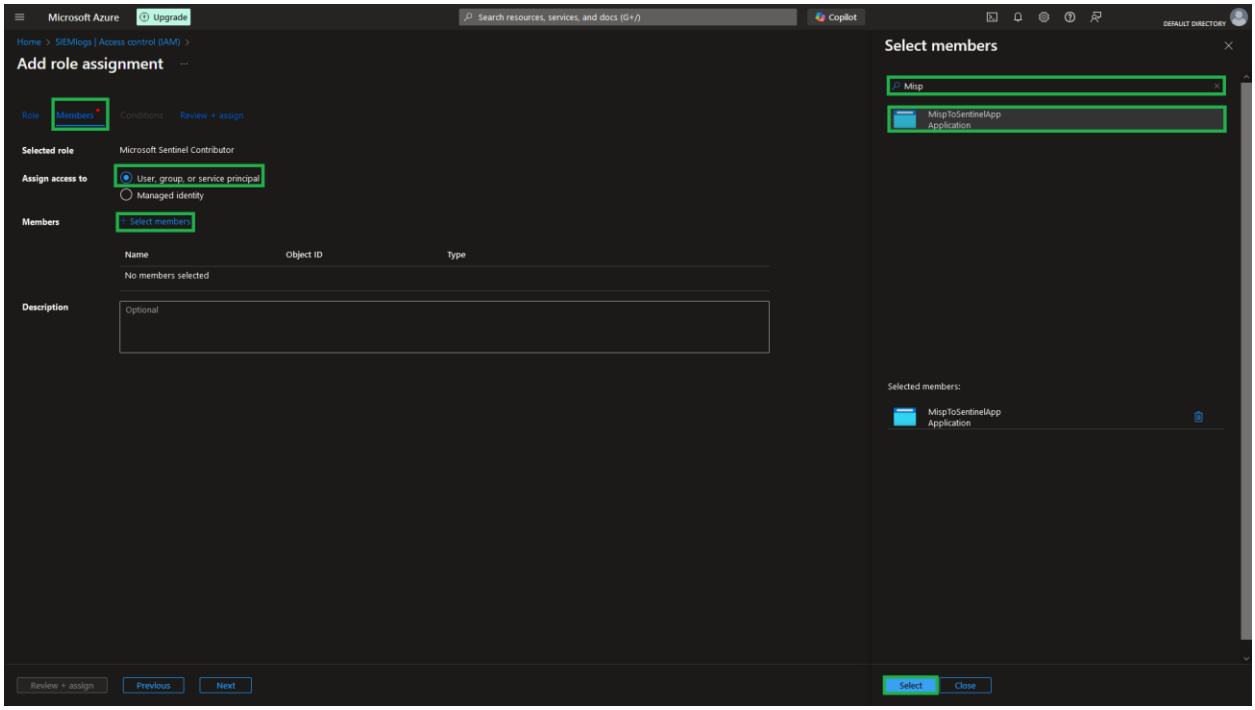
7. Now we need to add the Sentinel Contributor role to the app. We do this by going into the Log Analytics workspace our Sentinel intsance is in, in this case **SIEMlogs**, navigating to the **Access control (IAM)** page, and clicking on **Add role assignment**.



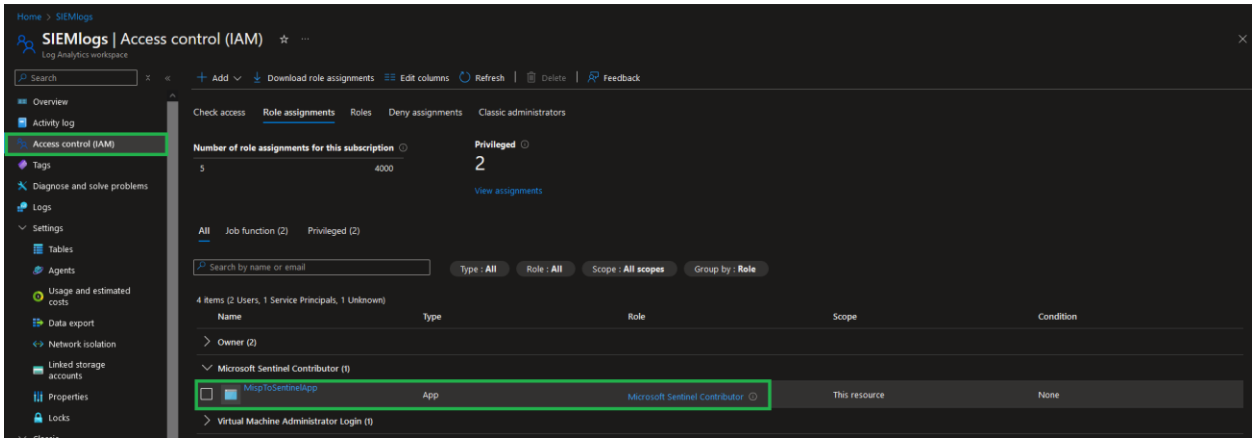
8. We now search for the Microsoft Sentinel Contributor role on the **Role** tab, select it, and click **Next**.



9. On the **Members** tab, we leave **the User, group, or service principal** radio button selected, click on **Select members**, search for the name of our app, select it, and click on the **Select** button.

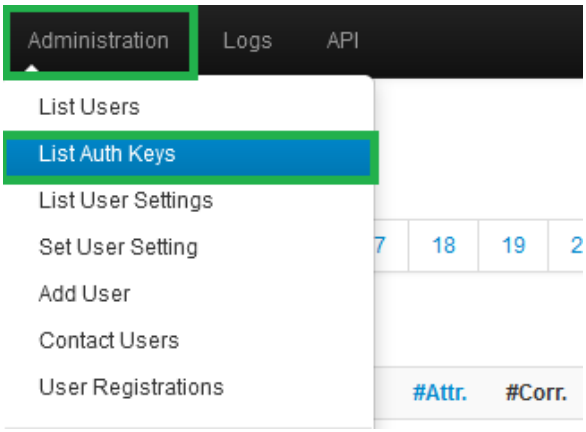


10. We then click on **Review + assign** and it should now show up on the **Access control (IAM)** page of the Log Analytics workspace.

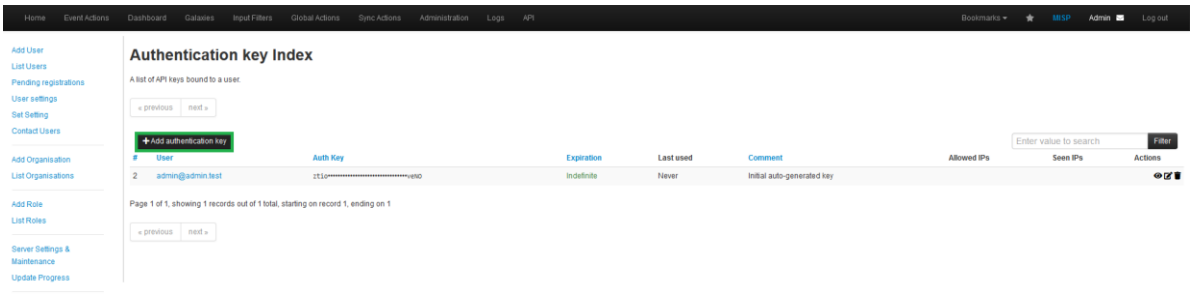


11. We need to create a MISP API Key. To do this, we go back to our MISP instance, hover over **Administration**, and click on **List Auth Keys.**

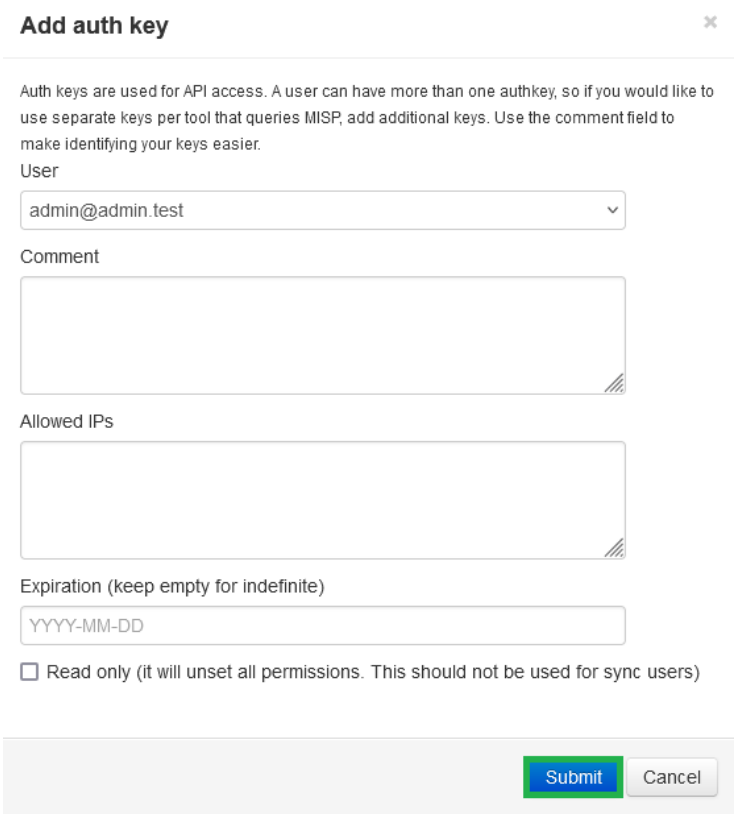
    Afterwards, this API key should ideally be stored in an Azure Key Vault, but for the sake of brevity we will not be doing this for this project.



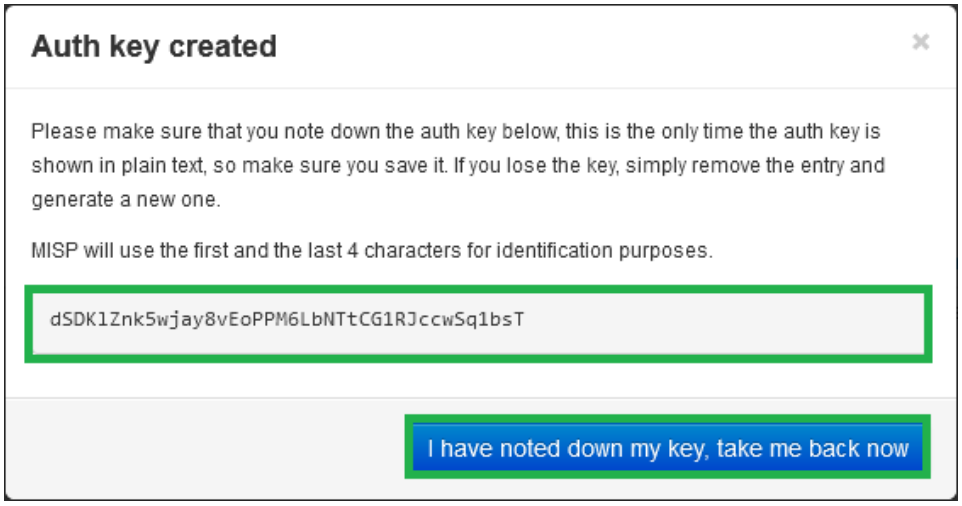
12. On the **Authentication key index** page, we click on **+Add authentication key**.



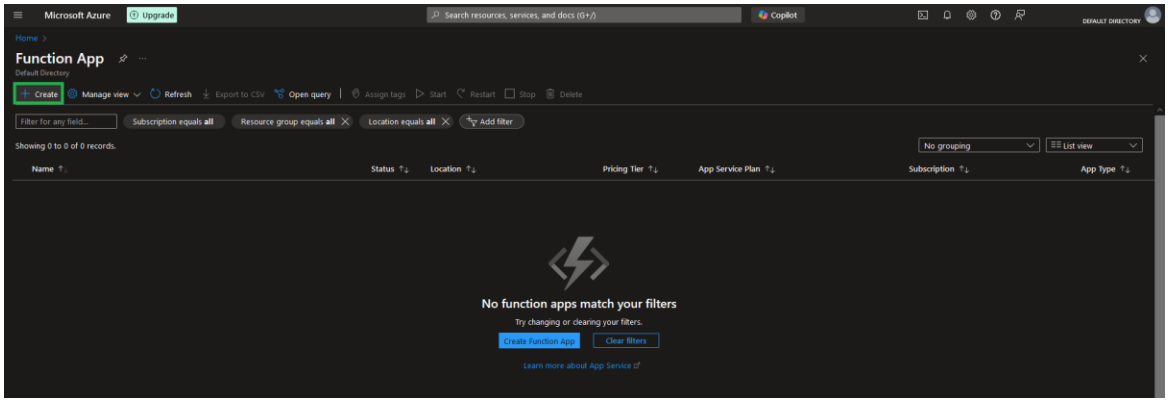
13. On the **Add auth key** page, we have the ability to secure our key buy creating an IP Allow list and setting an experation date, but we will leave it blank for this project and click on **Submit**.



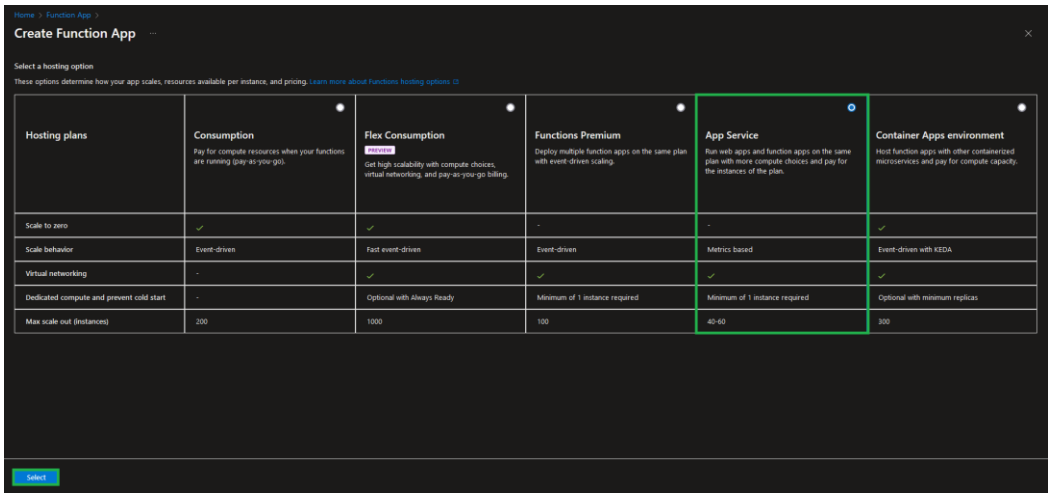
14. We take a note of our key, as it will not be shown in full again later and click on **I have noted down my key, take me back now**.



15. We now go back to the Azure Portal and create a Function app. We go to the **Function app** page through the search bar and click on **Create**.

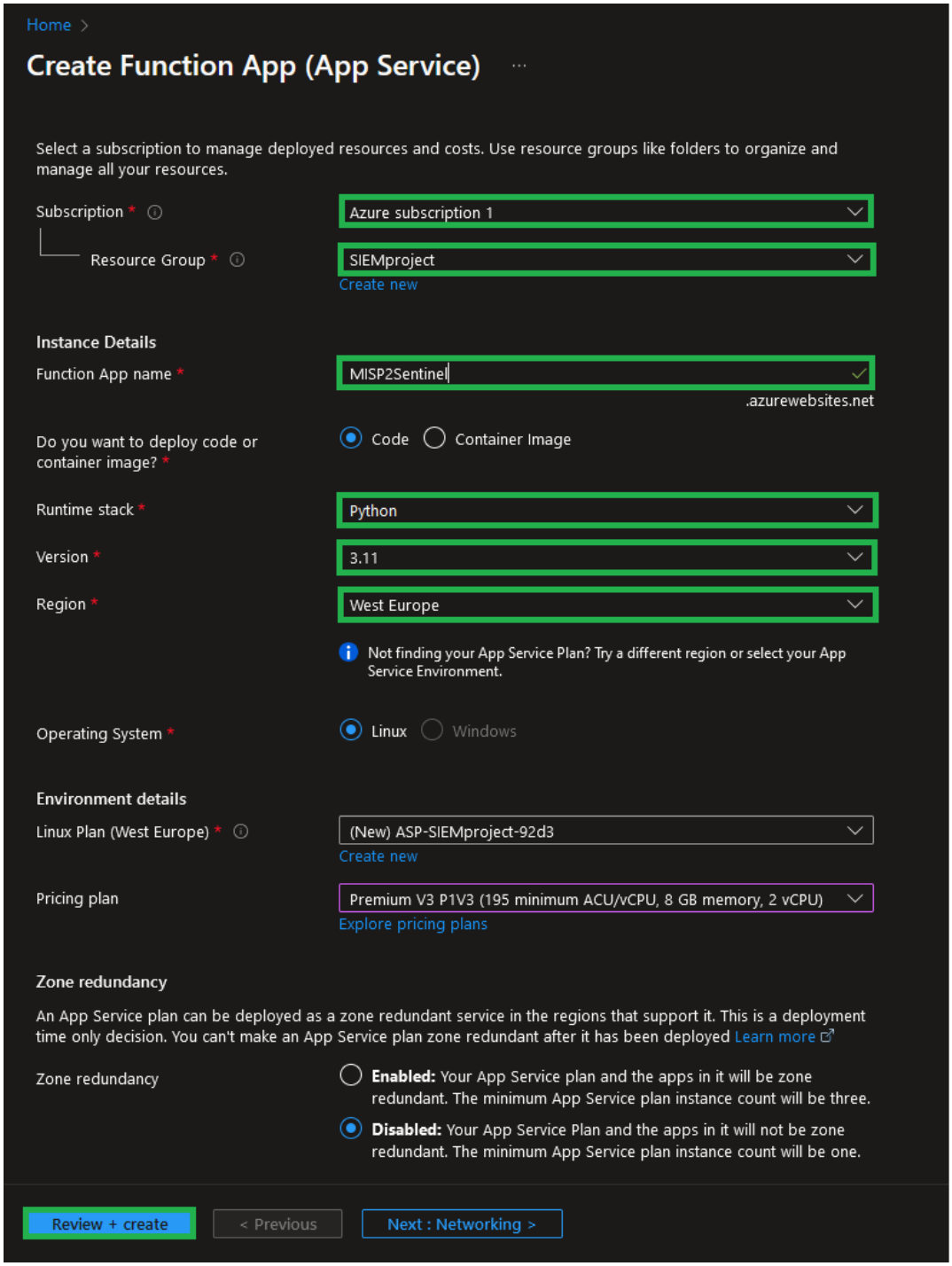


16. We then select the **App Service** hosting option.

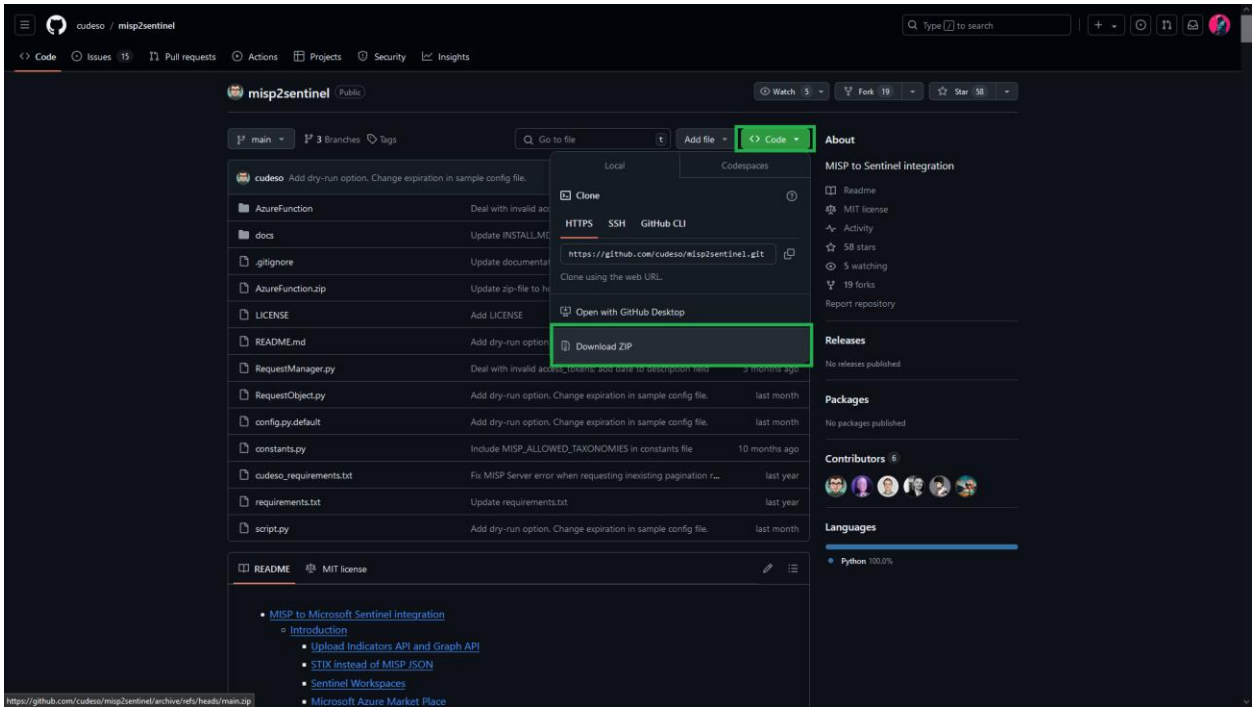


17. We select the following configuration on the **Basics** tab and click on **Review + create**:

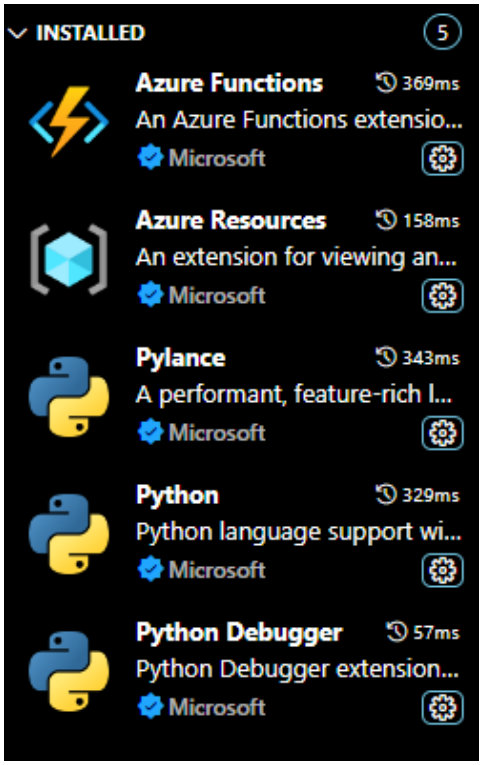
Field	**Value**
Subscription	Our current subscription.
Resource group	**SIEMproject**
Function App name	Enter a unique name, such as **MISP2Sentinel**.
Region	Select the same region as previous resources
Runtime stack	Python
Availability zone	3.11 (Any Python 3)
Size	Premium V3 P1V3 (Can be left default)



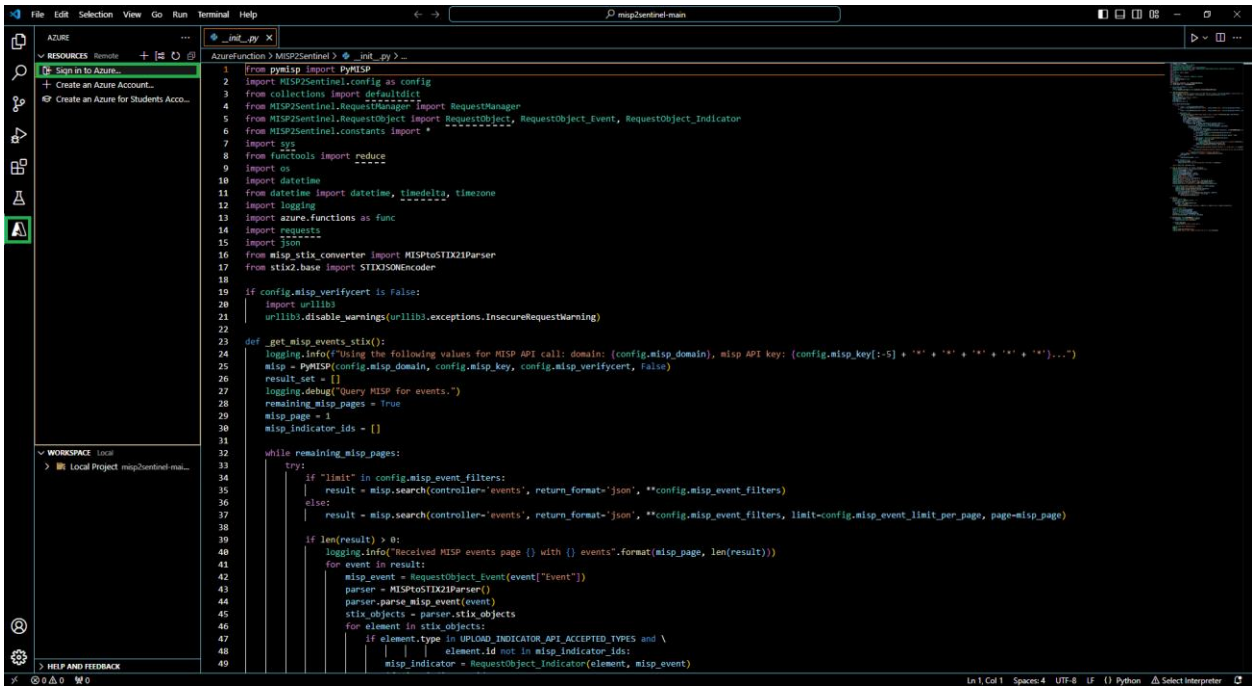
18. We now go to **github.com/cudeso/misp2sentinel**, click on **Code**, and then the **Download ZIP** option.



Note: We need to have Visual Studio Code and the Python, Pylance, Python Debugger, Azue Resources, and Azure Functions extensions installed for the following steps.

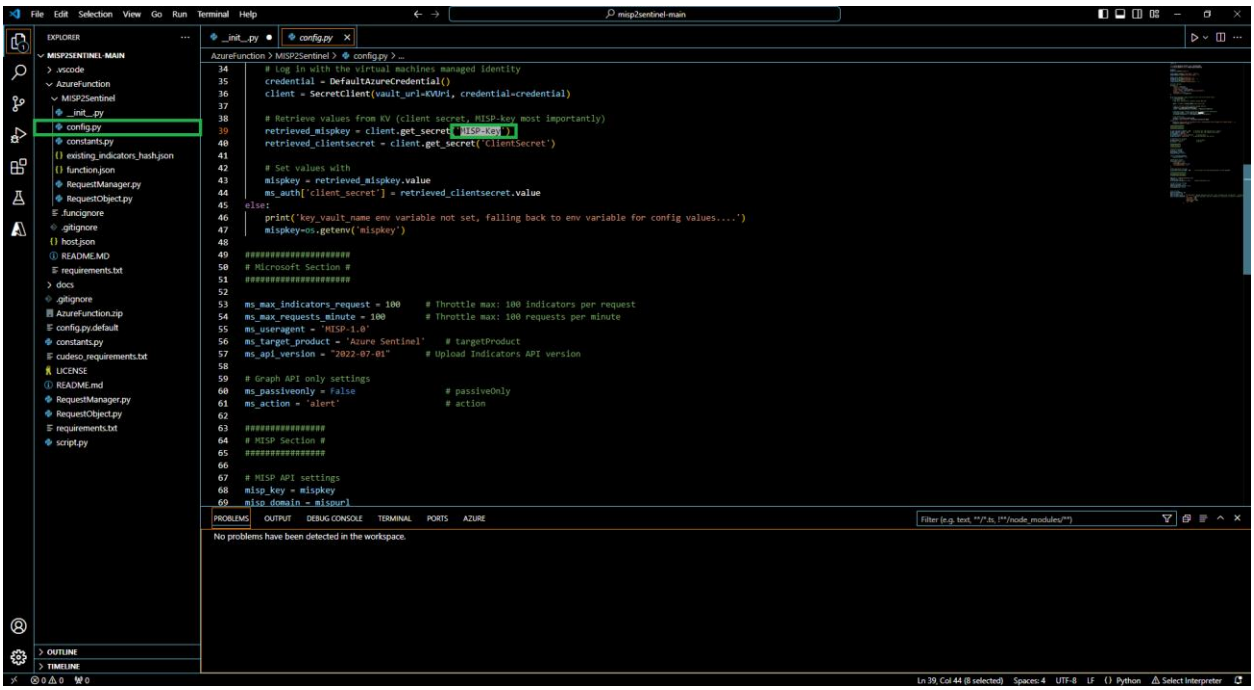


19. We extract the .zip file and open the extracted folder with Microsoft Visual Studio Code. We select the Azure icon, click on **Sign in to Azure..**, and choose **Allow** when prompted.

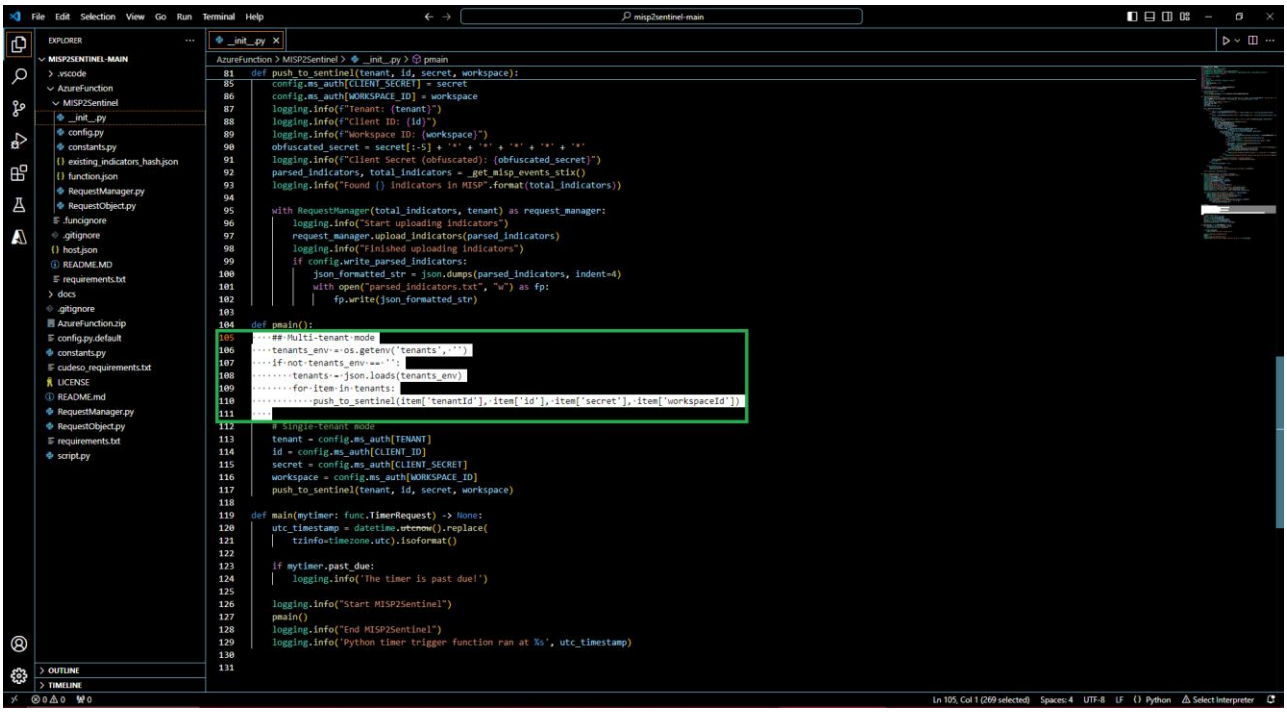


20. We no go back into the Python scripts as there are two changes we need to make to the code.

    I. We go to the **config.py** file and change ‘MISP-Key’ to ‘mispkey’.

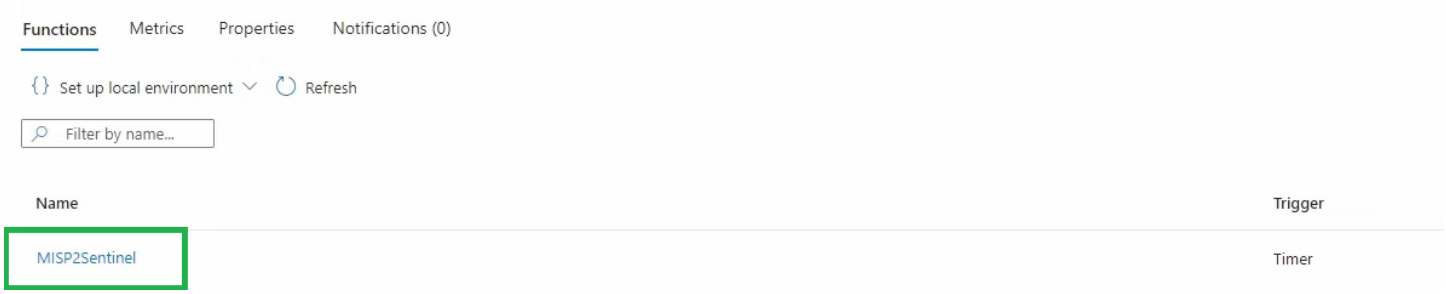


    II. We then go to the **_init_.py** file and delete the code block under the **## Multi-tenant mode** comment.

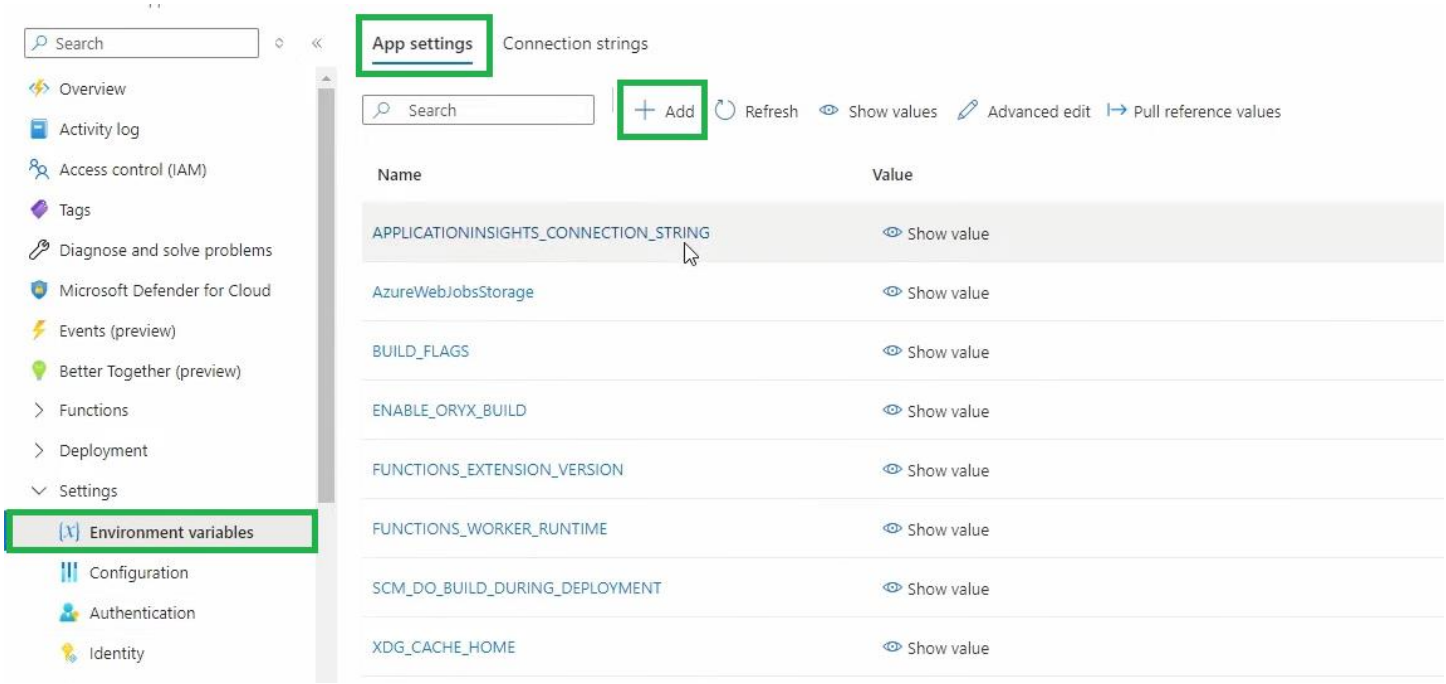


III. We then right-click on **AzureFunction** and select **Deploy to Function App**.

21. The function should now be visible under the **Functions** tab on the **Overview** page.



22. We then go to **Environment variables** under the *Settings* blade and click on **Add** on the **App settings** tab.



23. We create the following variables that are expected in our MISP2Sentinel python code:

```
6   mispkey = ''
7   mispurl=os.getenv('mispurl')
8
9   local_mode=os.getenv('local_mode', 'False')
10  keyVaultName=os.getenv('key_vault_name', '')
11
12  tenant_id=os.getenv('tenant_id', '')
13  workspace_id=os.getenv('workspace_id', '')
14  client_id=os.getenv('client_id', '')
15  client_secret=os.getenv('client_secret', '')
16
17  # MS API settings
18  ms_auth = {
19      'tenant': tenant_id,
20      'client_id': client_id,
21      'client_secret': client_secret,
22      'scope': 'https://management.azure.com/.default',
23      'graph_api': False,
24      'workspace_id': workspace_id
25  }
```

tenant_id
client_id
clinet_secret

They can be found on the app registration page we made earlier.

workspace_id

It can be found on the **Overview** page of the workspace our Sentinel instance is in.

mispkey

The key we generated on the **Authentication key index** page of our MISP instance.

mispurl

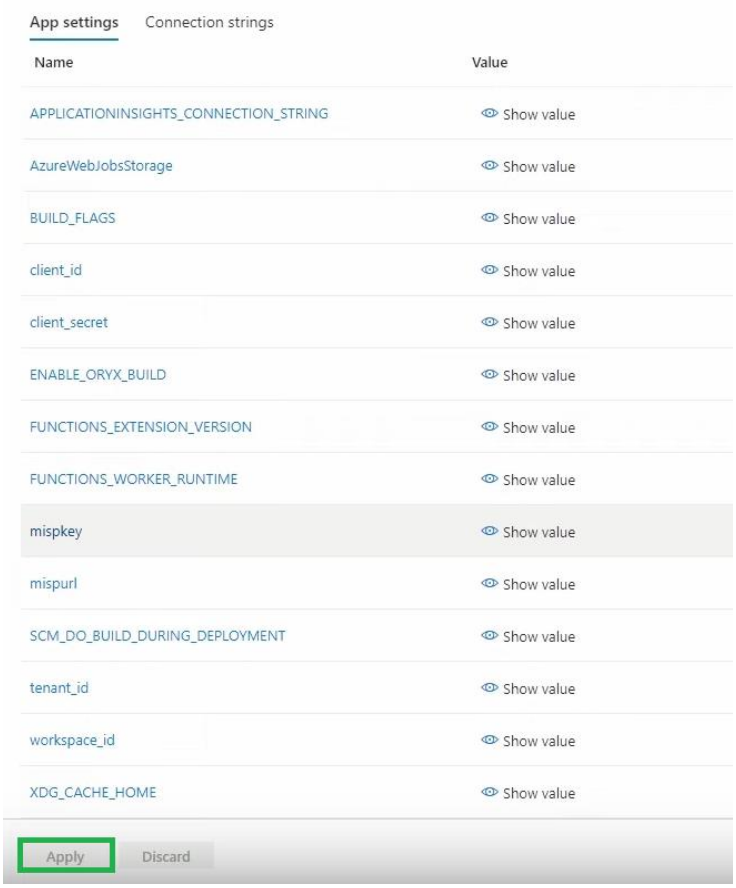
The public IP address of our Ubuntu server, can be found on our VMs **Connect** page.

24. We also need to add *a timerTriggerSchedule* variable with a value of:

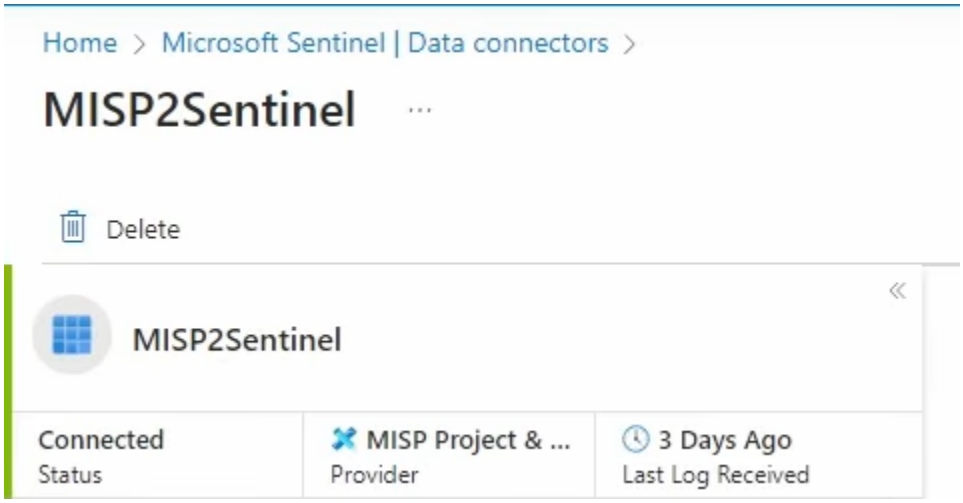
    0 */2 * * *

    This is so that the function can run every 2 hours.

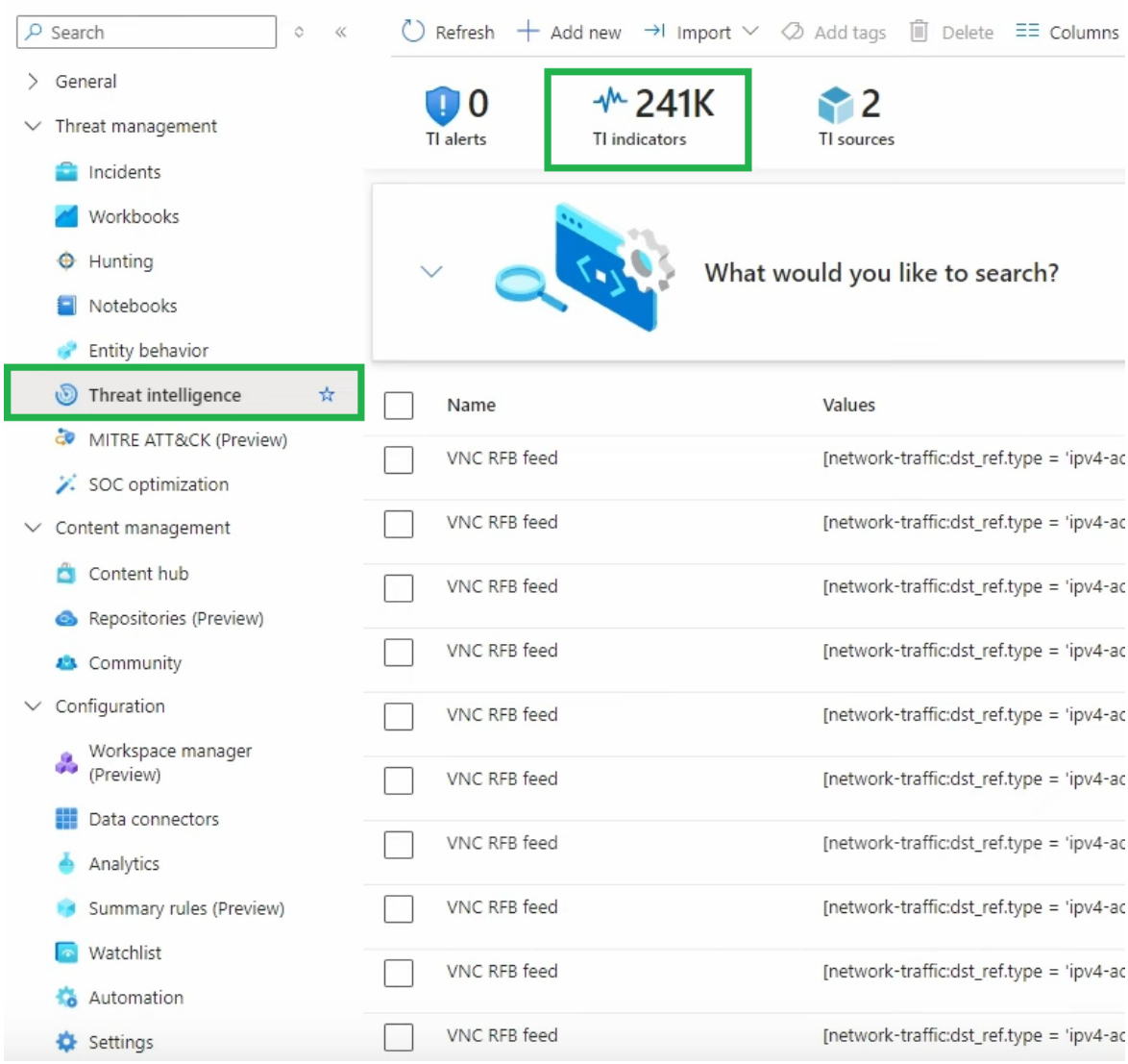
25. We then click **Appy**



26. Once we run the Function App, it should take a couple of hours for it to update in Sentinel. If we are successful, the status of our MISP2Sentinel Conncetor should change to **Connected**.



27. We should also now have our new threat intelligence indicators available.



28. We should also now have threat intelligence indicators available as a table that we could use to create alerts in Sentinel