AI, Cloud & Modern Workplace Conference 2024

15, 16 & 17 February, Online Conference

https://aicmwc.azurewebsites.net

AI, Cloud & Modern
Workplace Conference 2024

16, 17 & 18 February, Online Conference

Mastering KQL:
Empowering Threat Hunting and Incident Response for Defenders

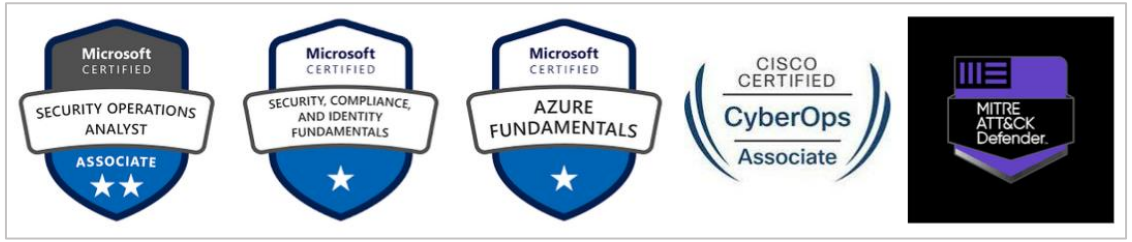
17 February 2024, 3:00 P.M. (GMT+2)

Michalis Michalos

Senior Cyber Security Threat Management Specialist

whoami

- Currently working as Senior Cyber Security Threat Management Specialist
- 11+ years experience in ICT
- 5 years experience in cybersecurity
- Electrical Engineer BSc, MSc, MBA



- Blog: michalos.net
- X Handle: cyb3rmik3
- Github: github.com/cyb3rmik3

Agenda

- Fundamentals of KQL
  - Familiarize searching your Data
  - Presenting Data
  - Combining Data
- Perform Threat Hunting using KQL and Microsoft Security Technologies
- Perform Incident Response using KQL and Microsoft Security Technologies
- Closing remarks
- Resources

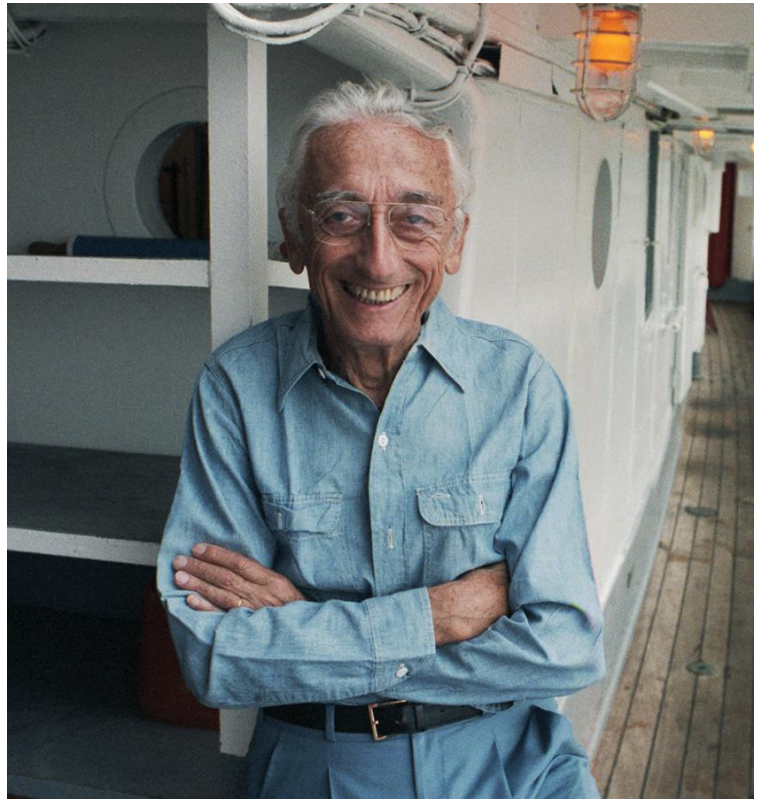
Fundamentals of KQL

What does “Kusto” stand for?

The name Kusto comes from **Jacques Cousteau**, a legendary oceanographer, explorer and adventurer.

Azure Application Insights was originally named Kusto and was later the foundation of Azure Monitor.

In recognition of KQL’s ability to deep dive into an ocean of data, Microsoft probably made the right decision to name its query language after Cousteau.



So what is Kusto?

Kusto Query Language is a read-only request language with requests stated in plain text, using a data-flow model that is easy to read, author, and automate. KQL is a simple yet powerful language to query structured, semi-structured, and unstructured data.

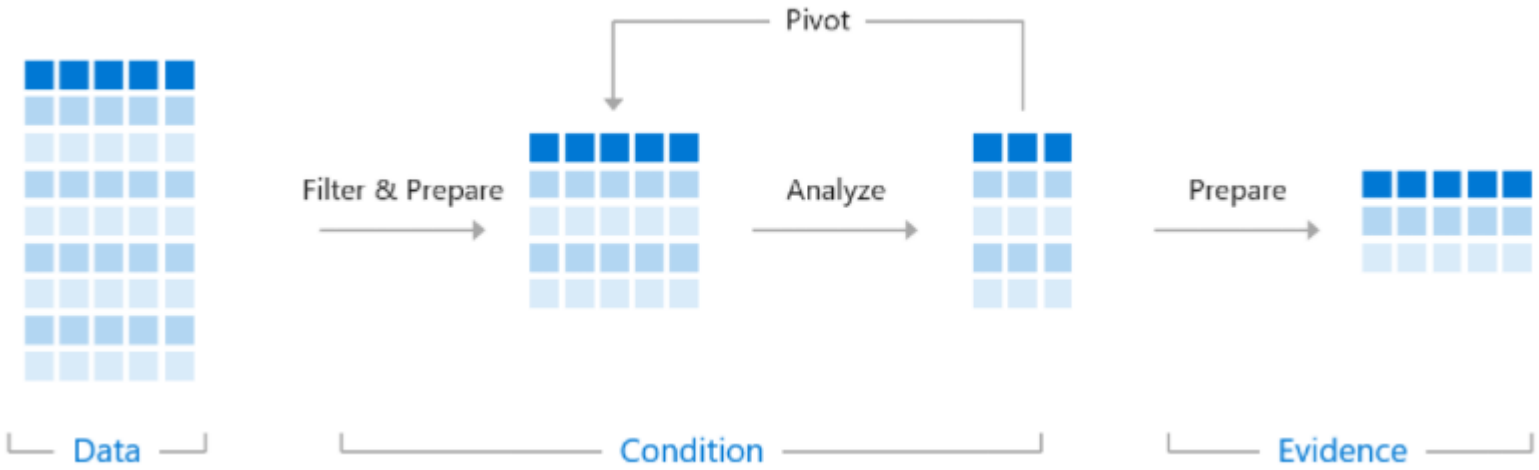
Where can I use KQL?

- Azure Data Explorer (ADX)
- Azure Monitor Log Analytics
- Microsoft Sentinel
- Defender XDR Advanced Hunting

… across all of Azure!

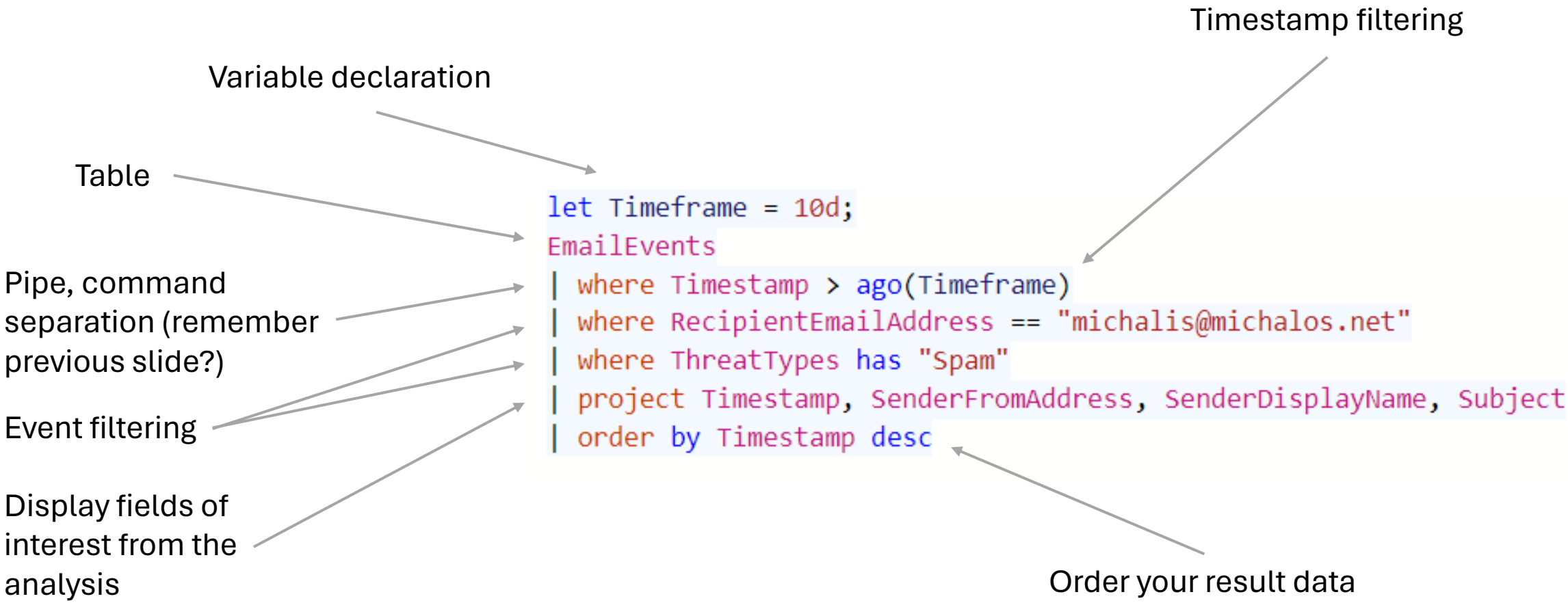
Query operators

- language syntax is that of a data flow, where "data" means "tabular data" (data in one or more rows/columns rectangular shape)
- at a minimum, a query consists of source data references and one or more query operators applied in sequence
- a pipe character (|) is used to delimit operators

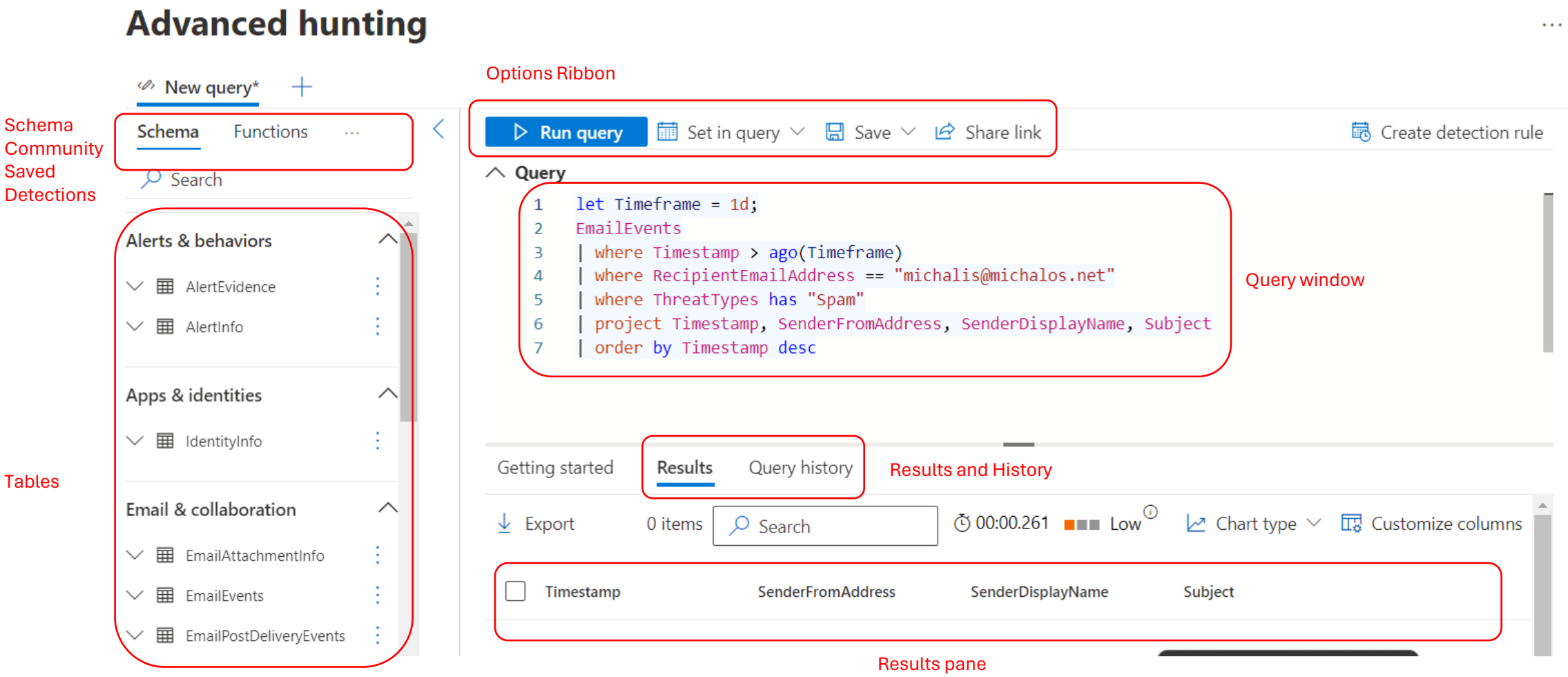


The goal is to filter down data and get the desired results.

KQL query structure (a basic one...)



Advanced Hunting blade



Before we begin…

Make sure, you use in-line commenting:

- It helps documenting
- It helps other people understand what your approach is
- Give instructions to define variables (for example time)
- You can use during experimenting with your operators/variables (quickly add/remove lines without completely losing them)

```
let Timeframe = 1d; // Define your desired timeframe
EmailEvents
| where Timestamp > ago(Timeframe)
| where RecipientEmailAddress == "michalis@michalos.net"
//| where ThreatTypes has "Spam"
| project Timestamp, SenderFromAddress, SenderDisplayName, Subject
| order by Timestamp desc
```

Use // to indicate a comment or exclusion.

Familiarize searching your Data

getschema operator

This operator will provide the data types for the table of interest.

Syntax: Table | getschema
Example: EmailEvents | getschema

Results:

ColumnName	ColumnOrdinal	DataType	ColumnType
Timestamp	0	System.DateTime	datetime
NetworkMessageId	1	System.String	string

search operator

Straightforward and interactive use.

Syntax: Table | search
Example: EmailEvents | search "keyword"

- If no table is defined, search will be performed in all tables.
- Better filter out, to save time.
- Helps answer: “Does it exist?”, “Where does it exist?”, “Why does it exist?”
- In general, it’s not recommended, as it is CPU intensive

Data filtering

where operator

Filters a table to the subset of rows that satisfy a predicate.

Syntax: Table | where Predicate
Example: EmailEvents | where RecipientEmailAddress == "michalis@michalos.net"

Predicate satisfaction supported through searching string data types, using various query operators.

- has, contains, strartswith, endswith…
- ==, !=, <, >, <= …
- in, !in, has_all, has_any…
- isempty (), notempty (), isnull (), notnull

count operator

Returns the number of records

Syntax: Table | count
Example: EmailEvents | count

sort operator

Sorts the rows of the input table into order by one or more columns.

Syntax: Table | sort by Column [asc | desc]
Example: EmailEvents | sort by Timestamp desc

Results filtering

project operator

Selecting which Columns to present.

Syntax:	Table | project ColumnName
Example:	EmailEvents | project Timestamp, SenderDisplayName, Subject

distinct operator

Produces a table with the distinct combination of the provided columns of the input table.

Syntax:	Table | distinct
Example:	EmailEvents | distinct SenderFromAddress, EmailAction

Filtering searches

contains filtering

Filters a record set for data containing a case-insensitive string. contains searches for arbitrary sub-strings rather than terms.

Syntax: Table | where Predicate contains "phrase"
Example: EmailEvents | where SenderFromAddress contains "microsoft“

Usually, a resource consuming operation.

has operator

Filters a record set for data with a case-insensitive string. has searches for indexed terms, where an indexed term is three or more characters.

Syntax: Table | where Predicate has "phrase"
Example: EmailEvents | where SenderFromAddress has "microsoft"

```
1  EmailEvents
2  | where SenderFromAddress contains "microsoft"
3  | summarize count() by SenderFromAddress
```

Getting started | Results | Query history

Export 5 ite

SenderFromAddress	count_
o365mc@microsoft.com	8
defender-noreply@microsoft.com	5
microsoft-noreply@microsoft.com	11
azure-noreply@microsoft.com	1
[redacted]@[redacted]onmicrosoft.com	2

```
1  EmailEvents
2  | where SenderFromAddress has "microsoft"
3  | summarize count() by SenderFromAddress
```

Getting started | Results | Query history

Export

SenderFromAddress	count_
o365mc@microsoft.com	8
defender-noreply@microsoft.com	5
microsoft-noreply@microsoft.com	11
azure-noreply@microsoft.com	1

Presenting Data

extend operator

Creates calculated columns and append them to the result set.

Syntax: Table | extend [ColumnName | (ColumnName[, ...]) =] Expression [, ...]

Example: DeviceInfo | extend DeviceNameLength = strlen(DeviceName)

- The new column, is not indexed.
- To only change a column name, use project-rename operator

```
1  DeviceInfo
2  // Calculate the DeviceName length
3  | extend DeviceNameLength = strlen(DeviceName)
4  | project DeviceName, DeviceNameLength
```

Getting started | Results | Query history

Export 226

DeviceName	DeviceNameLength
mm-gl-	14
mm-gl-	14
desktop	15
mm-gl-	14

summarize operator

Produces a table that aggregates the content of the input table.

Syntax: Table | summarize [SummarizeParameters] [[Column =] Aggregation [, ...]] [by [Column =] GroupExpression [, ...]]

Example: EmailEvents | summarize by SenderIPv4, SenderFromAddress

```
UrlClickEvents
// Count how many times, a Url identified as phishing
// has been clicked
| where ThreatTypes has "Phish"
| summarize count() by Url
```

- Can be used with aggregation functions count(), sum(), min(), max() etc.

summarize operator (bin and time)

Summarize by creating a time series

- Change time values per specific needs, 1h, 1d etc.
- Could be useful to detect anomalies

```
1  EmailEvents
2  | summarize count() by bin(Timestamp, 1d)
3  | order by Timestamp asc
```

Getting started | Results | Query history

Export

Timestamp	count_
Feb 3, 2024 2:00:0...	6
Feb 4, 2024 2:00:0...	3
Feb 5, 2024 2:00:0...	4
Feb 6, 2024 2:00:0...	30
Feb 7, 2024 2:00:0...	11
Feb 8, 2024 2:00:0...	18
Feb 9, 2024 2:00:0...	4
Feb 10, 2024 2:00:...	5

render operator

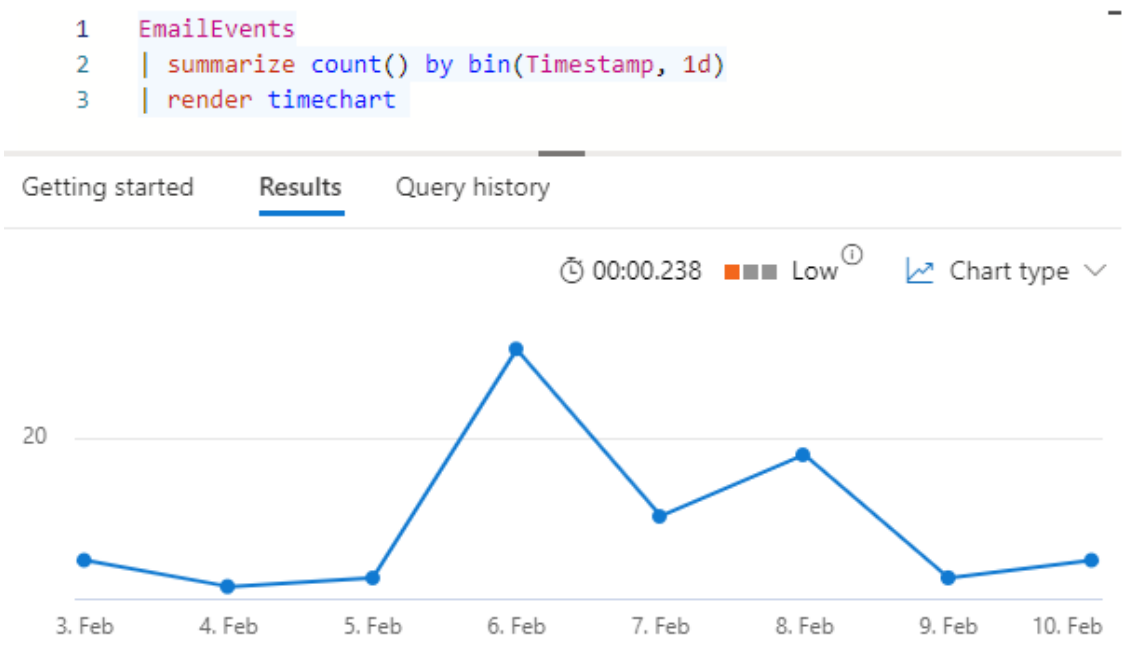
Generates a visualization of the query results.

Syntax: Table | render visualization [with (propertyName = propertyValue [, ...])]

Supported visualizations:

- anomalychart
- areachart
- barchart
- card
- columnchart
- ladderchart
- linechart
- piechart
- pivotchart
- scatterchart
- stackedareachart
- table
- timechart
- timepivot
- treemap

Not all visualizations are supported across platforms (log analytics, advanced hunting, data explorer)



let statement

A let statement is used to set a variable name equal to an expression or a function, or to create views.

- Declaring variables to ease complexity
- Used for allow, deny, and exclusion lists
- Build a temporary container for a table
- Can be used multiple times in query
- Useful for targeted threat hunting

```
1  // Hunt for Remote Monitoring and Management (RMM)
2  // executables based on filenames
3  let SuspiciousRMMExecutables = datatable (rmmexecutable: string) [
4      @"anydesk.exe"
5      @"teamviewer.exe"
6      ];
7  DeviceProcessEvents
8  | where FileName in (SuspiciousRMMExecutables)
```

Combining Data

union operator

Takes two or more tables and returns the rows of all of them.

Syntax: [Table |] union [UnionParameters] [kind=inner|outer] [withsource= ColumnName] [isfuzzy= true|false] Tables

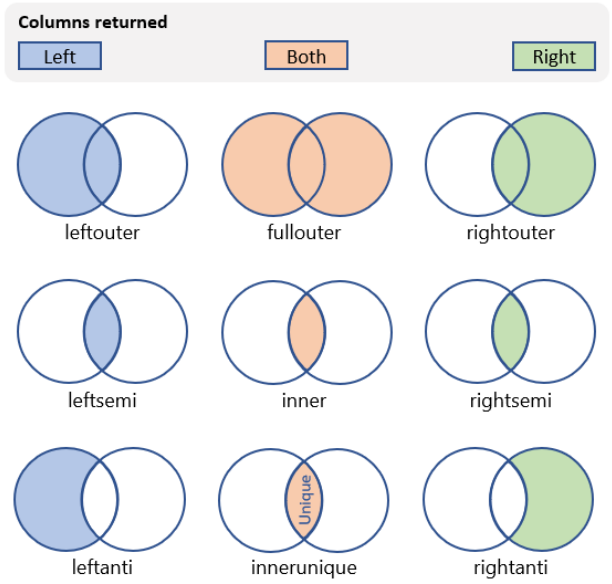
```
// Query will return any FilaName from DeviceFileEvents and
// DeviceProcessEvents tables containing "anydesk" keyword
union DeviceFileEvents, DeviceProcessEvents
| where FileName contains @"anydesk"
```

- kind=inner(common columns), outer (all columns default)
- Supports wildcard to union multiple tables (union Table*)
- Can union between tables from different clusters (or workspaces)

join operator

Merge the rows of two tables to form a new table by matching values of the specified columns from each table.

Syntax: LeftTable | join [kind = JoinFlavor] [Hints] (RightTable) on Conditions




```
// Enrich URL click events from emails that don't belong to
// ClickAllowed, from the EmailEvents Table
EmailEvents
| join (UrlClickEvents
| where ActionType != "ClickAllowed") on NetworkMessageId
```

wrapping up

- Operators, functions and methods presented, are the most common in KQL to begin with.
- By now, you should be able to do some basic filtering and present and combine data.
- You are ready to begin your journey in KQL!

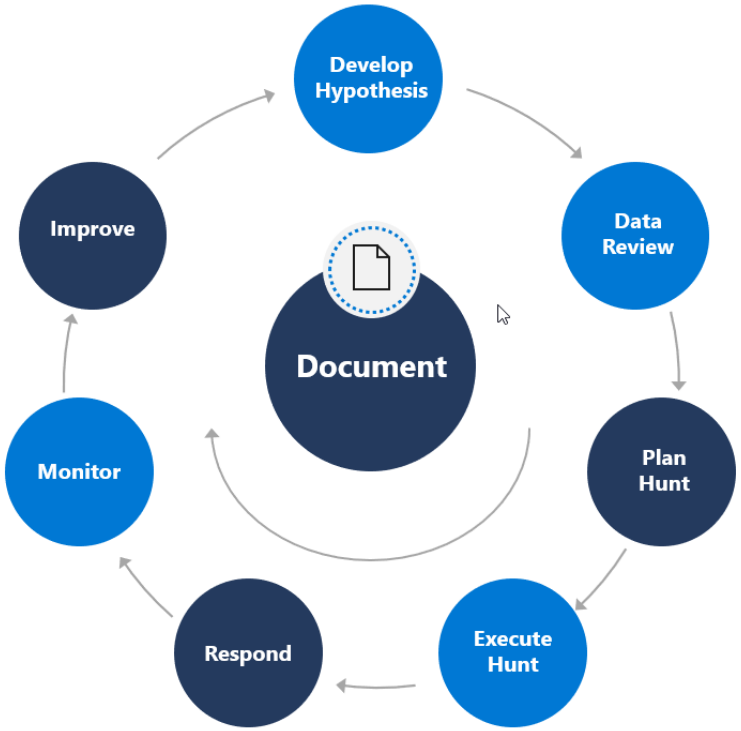
```
let Timeframe = 30d; // Choose the best timeframe for your investigation
let SuspiciousAnydeskFileCertificate = DeviceFileCertificateInfo
    | where Timestamp > ago(Timeframe)
    | where CertificateSerialNumber =~ "0dbf152deaf0b981a8a938d53f769db8" // Compromised Certificate Serial Number
    | where Issuer == "DigiCert Trusted G4 Code Signing RSA4096 SHA384 2021 CA1"
    | project Timestamp, DeviceName, SHA1;
SuspiciousAnydeskFileCertificate
    | join (DeviceProcessEvents
    | where Timestamp > ago(Timeframe)
    | where ProcessVersionInfoCompanyName !contains @"AnyDesk"
    | project SHA1, ActionType, FileName, FolderPath, ProcessVersionInfoCompanyName, ProcessVersionInfoProductName,
   ProcessCommandLine, AccountName, InitiatingProcessAccountName, InitiatingProcessFileName, InitiatingProcessCommandLine
    )on SHA1
    | sort by Timestamp desc
```

Perform Threat Hunting using KQL and Microsoft Security Technologies

Threat Hunting basics

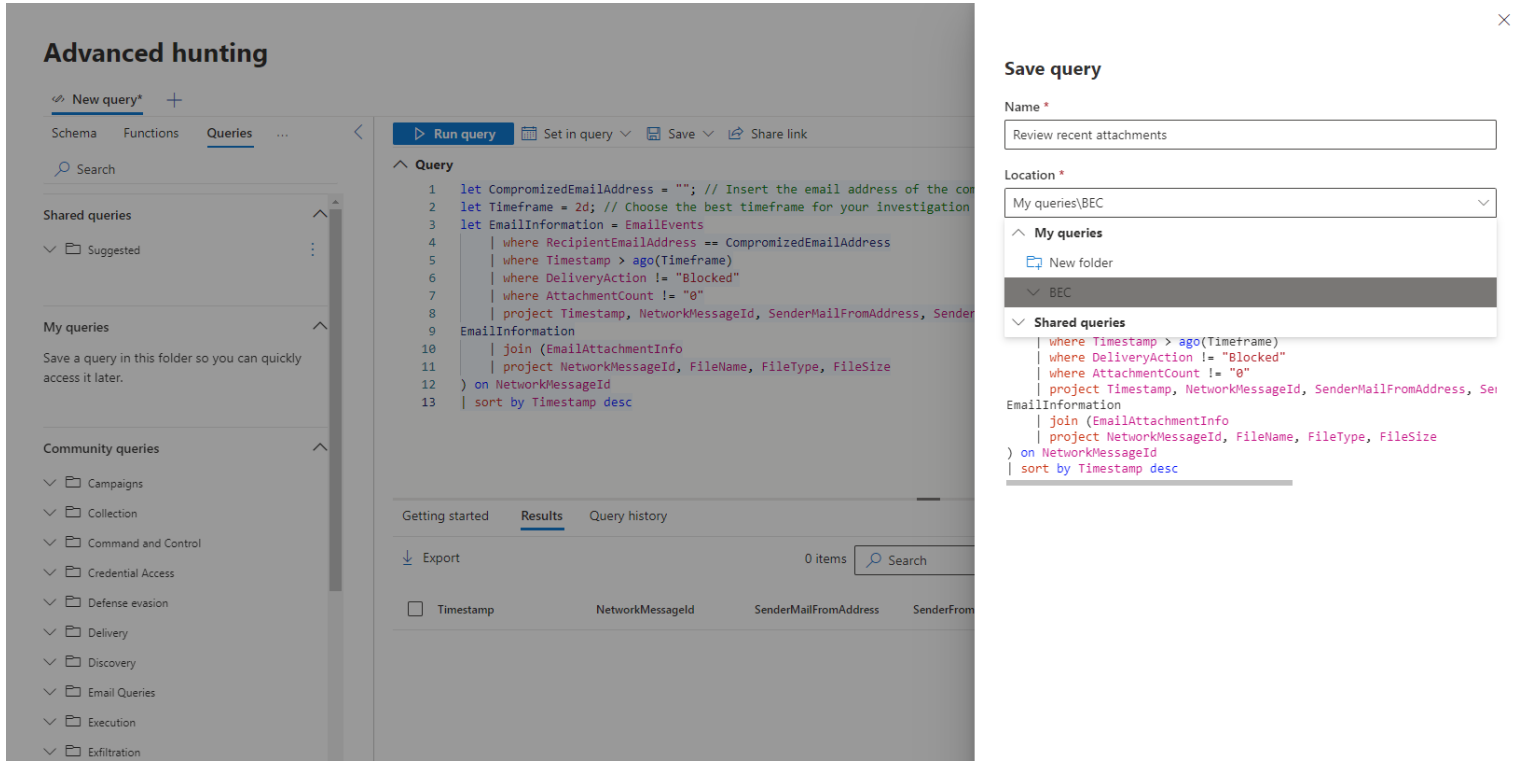
- proactively hunting for a threat or a set of activities that haven't been previously detected
- a continual process
- documentation for the hunt should include:
  - What, How, and Why
  - Input and Output
  - How to replicate the hunt
  - Next Steps

- Hypothesis development:
  - Keep it achievable
  - Keep the scope narrow
  - Keep it time-bound
  - Keep it useful and efficient
  - Keep it related to the threat model that you are defending against



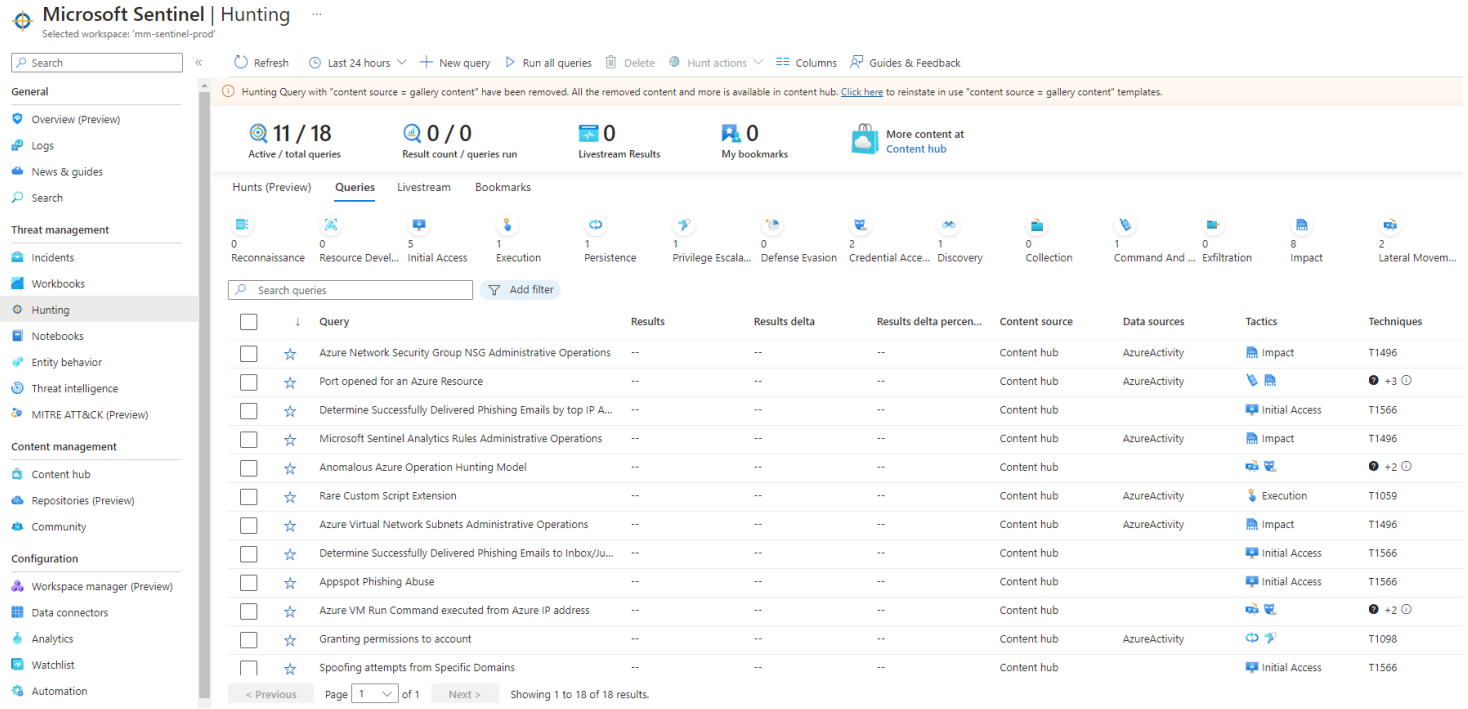
Advanced Hunting in Defender XDR

- Perform quick and simple queries
- Can be saved under Queries > My queries for future use
- No reference, comments or any sort of contextualization



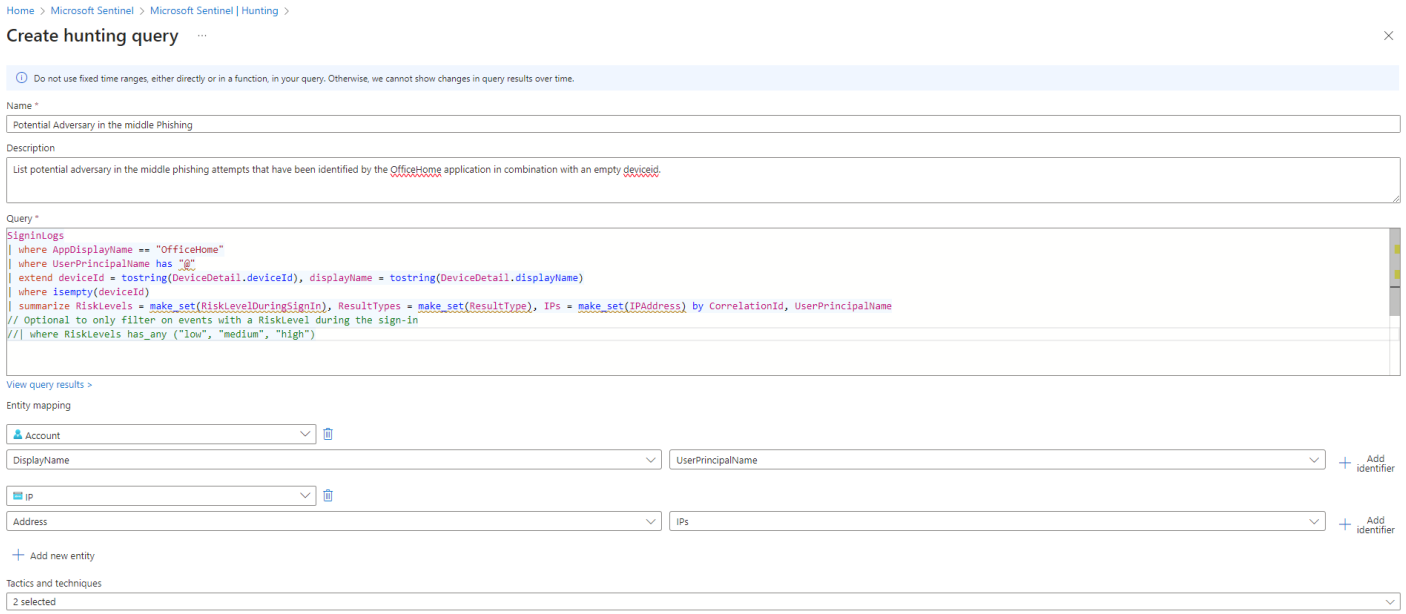
Hunting in Microsoft Sentinel

- Use build-in queries, or create new ones
- Operationalize MITRE ATT&CK framework
- Use livestream to actively hunt with your queries
- Contextualize your hunts, hypotheses and process



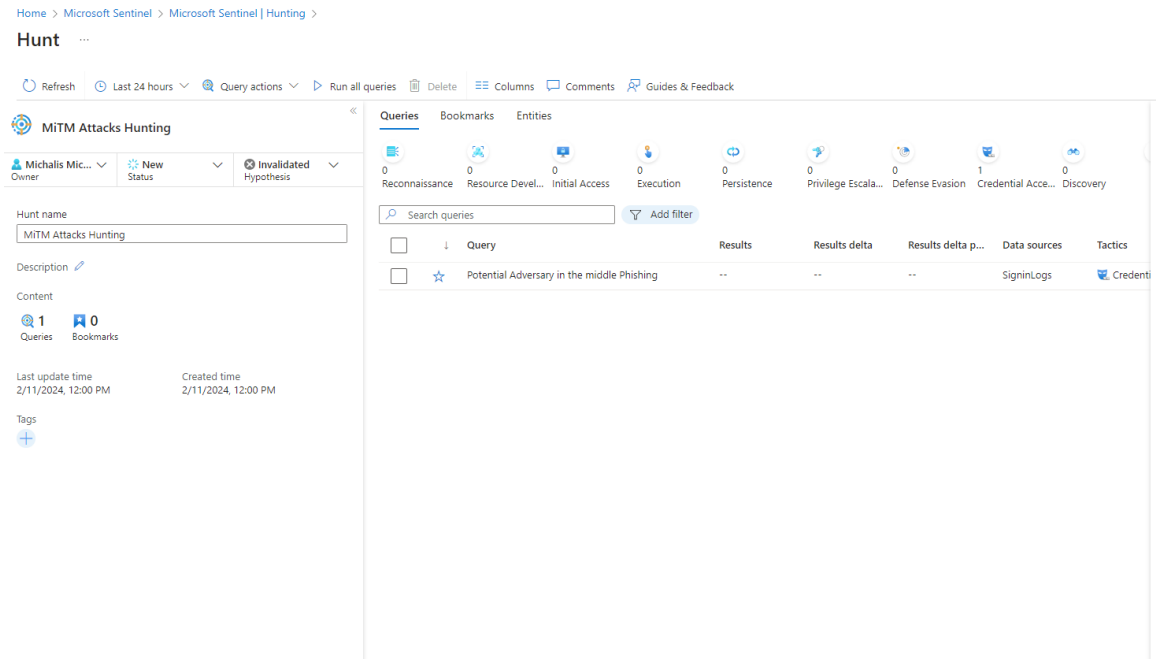
Hunting in Microsoft Sentinel

- Add description to your queries for better documentation
- Map your entities such as Users, Hosts and IPs
- Map MITRE ATT&CK TTPs



Hunting in Microsoft Sentinel

- Group multiple queries into Hunts, for example if you have multiple queries related to a Threat Actor, or a Technique
- Contextualize based on Status (New, Active, Closed, Backlog and Approved), Hypothesis (Unknown, Invalidated and Validated) or Tags
- Documentation opportunities: Description and Comments



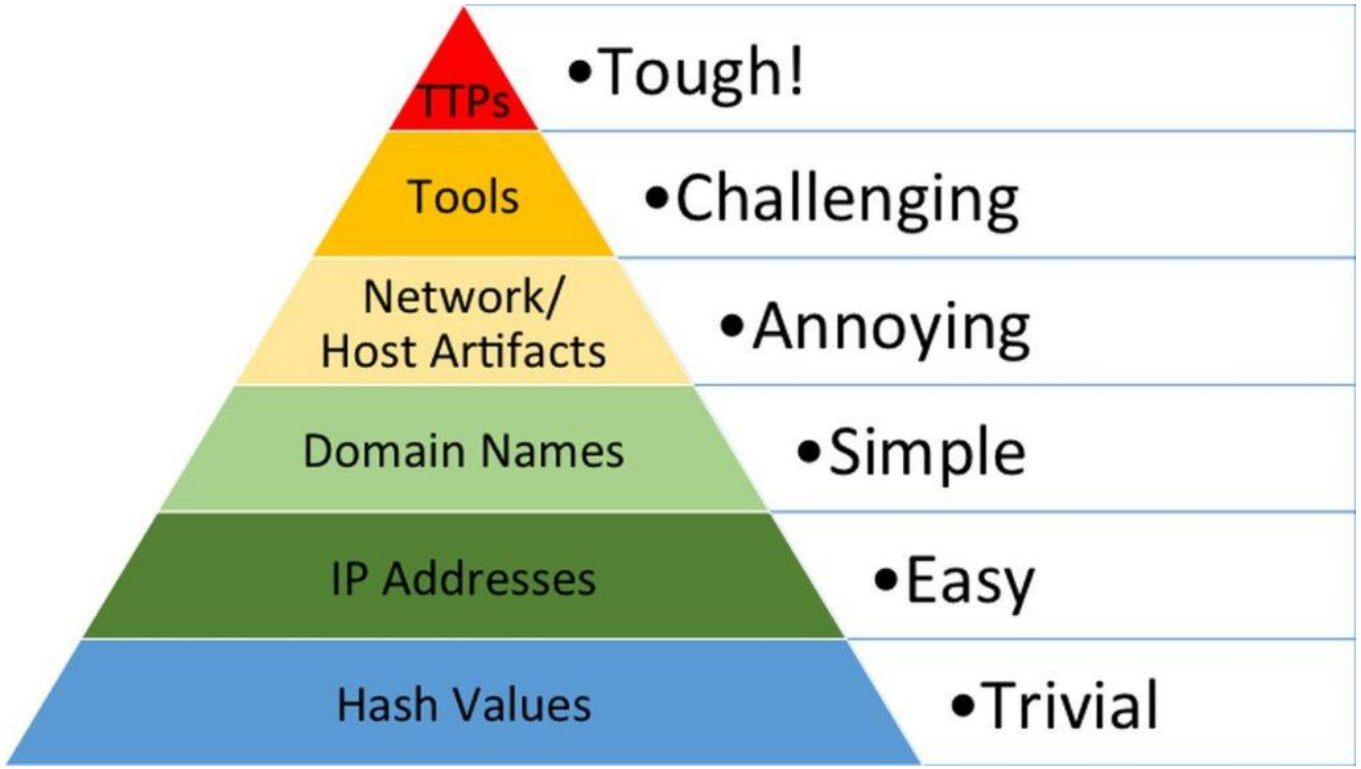
Hunting in Microsoft Sentinel

```
// Groups MFA phone registration events into the number that was registered,
// can be useful to detect threat actors registering multiple accounts to the same numbers for persistence
// Data connector required for this query - Azure Active Directory - Audit Logs
// Source: https://github.com/reprise99/Sentinel-Queries/
AuditLogs
| where TimeGenerated > ago(90d)
| where TargetResources has "PhoneNumber"
| where OperationName has "Update user"
| where TargetResources has "StrongAuthenticationMethod"
| extend InitiatedBy = tostring(parse_json(tostring(InitiatedBy.user)).userPrincipalName)
| extend UserPrincipalName = tostring(TargetResources[0].userPrincipalName)
| extend targetResources=parse_json(TargetResources)
| mv-apply tr = targetResources on (
    extend targetResource = tr.displayName
    | mv-apply mp = tr.modifiedProperties on (
    where mp.displayName == "StrongAuthenticationUserDetails"
    | extend NewValue = tostring(mp.newValue)
    ))
| project TimeGenerated, NewValue, UserPrincipalName,InitiatedBy
| mv-expand todynamic(NewValue)
| mv-expand NewValue.[0]
| extend AlternativePhoneNumber = tostring(NewValue.AlternativePhoneNumber)
| extend Email = tostring(NewValue.Email)
| extend PhoneNumber = tostring(NewValue.PhoneNumber)
| extend VoiceOnlyPhoneNumber = tostring(NewValue.VoiceOnlyPhoneNumber)
| project TimeGenerated, UserPrincipalName, InitiatedBy,PhoneNumber, AlternativePhoneNumber, VoiceOnlyPhoneNumber, Email
| where isnotempty(PhoneNumber)
| summarize ['Count of Users']=dcount(UserPrincipalName), ['List of Users']=make_set(UserPrincipalName) by PhoneNumber
| sort by ['Count of Users'] desc
```

don’t forget…

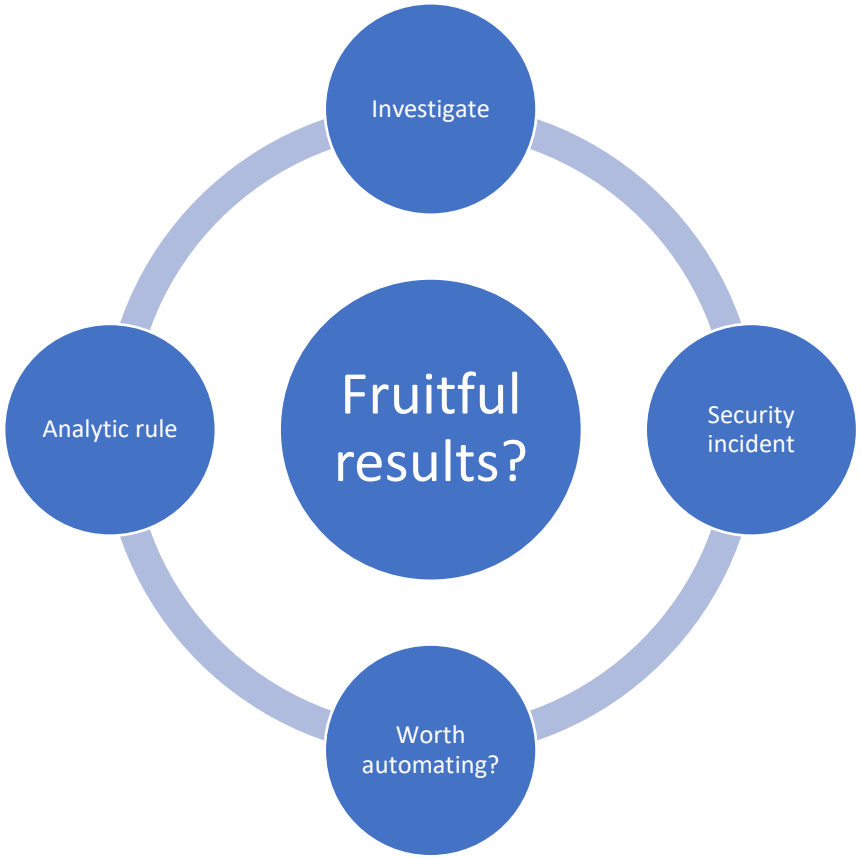
The Pyramid of Pain nurtures proactive Threat Hunting and hence, allowing organizations to stay one step ahead of adversaries.

KQL can help facilitate TTPs through suspicious behaviors and big data anomalies.



wrapping up Threat Hunting

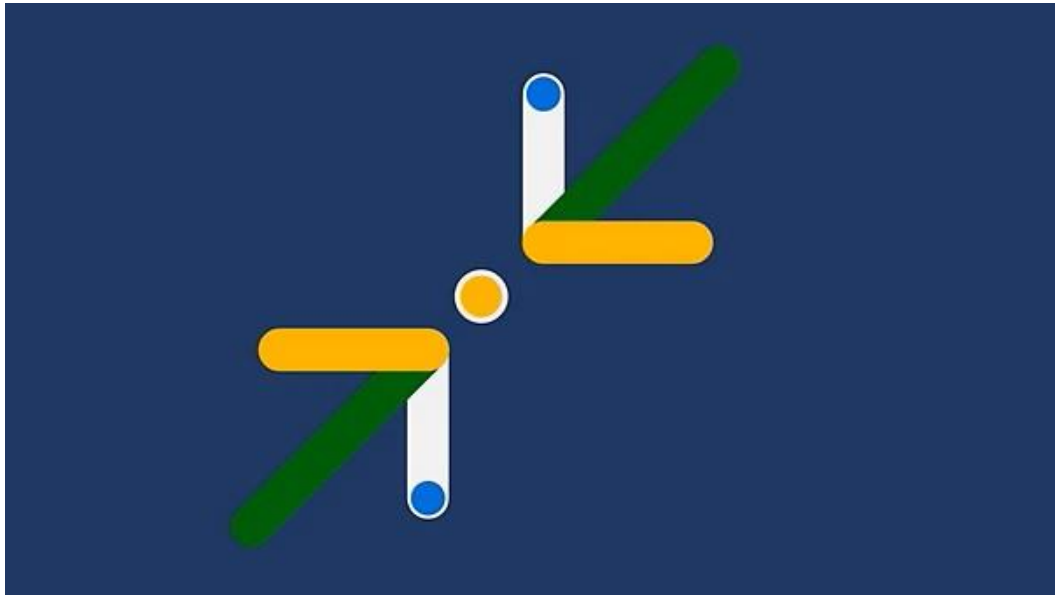
- A fruitful hunt would lead to an Investigation
- Investigation could lead to a Security incident
- Based on the process so far, is this hunt worth the automation?
- If yes, then build an Analytic rule



Perform Incident Response using KQL and Microsoft Security Technologies

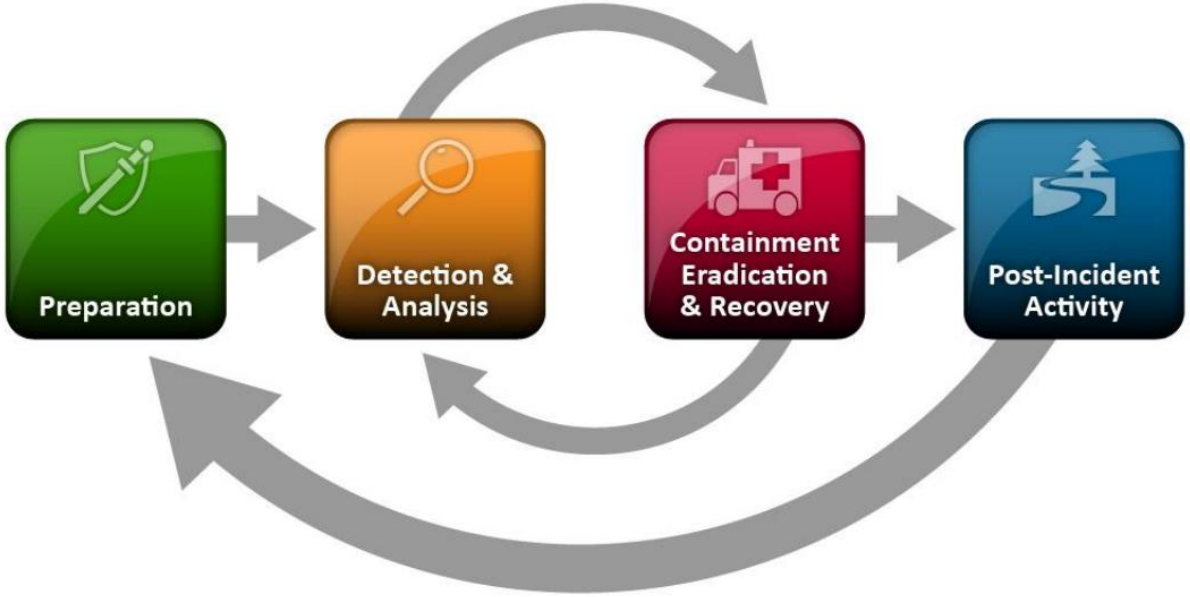
Incident Response basics

Incident response is the practice of investigating and remediating active attack campaigns on your organization. Incident response is part of the security operations (SecOps) discipline and is primarily reactive in nature.

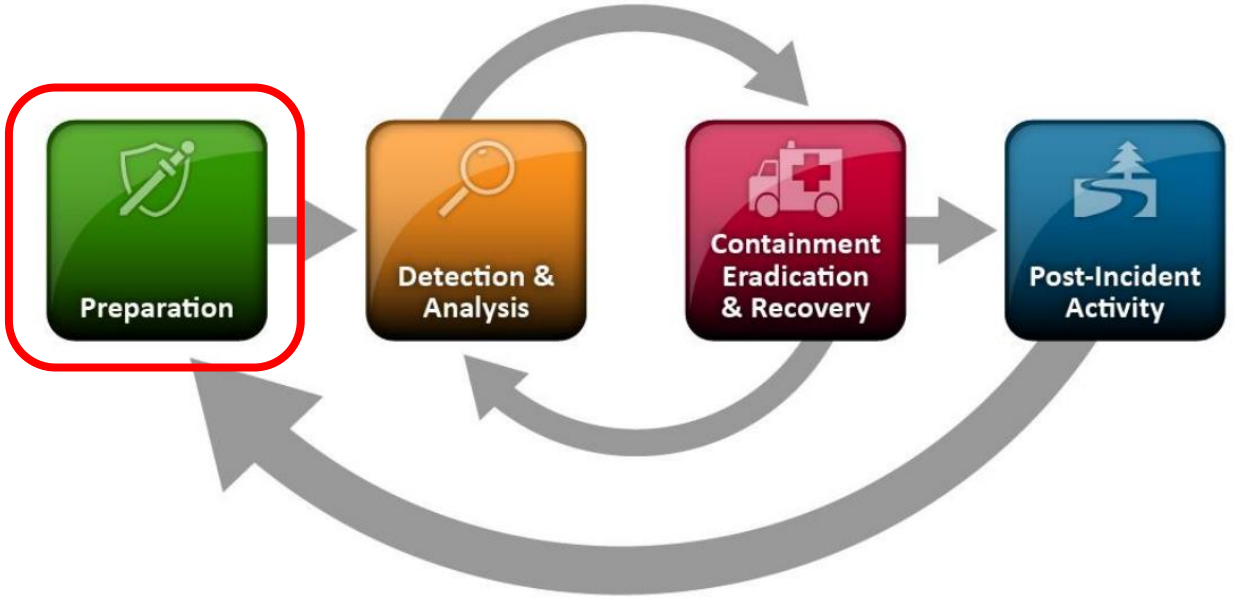


Incident Response Life Cycle

- NIST.SP.800-61r2
- A four-step process for incident response, illustrated in the diagram below
- How can KQL relate to each step?

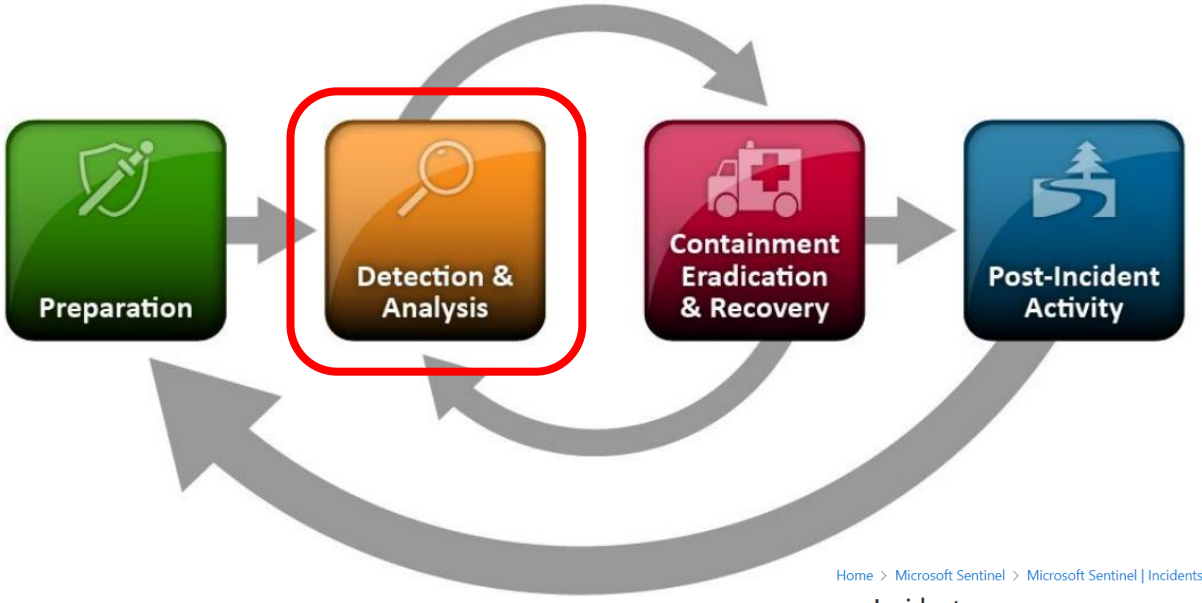


Incident Response Life Cycle

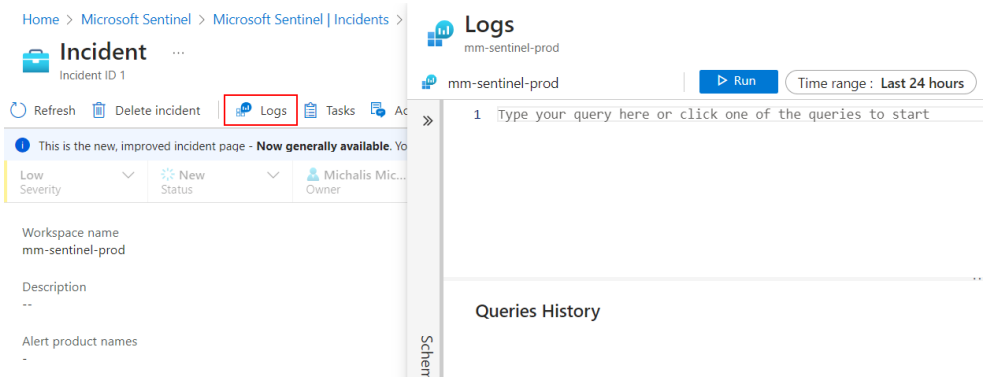


- Advanced Hunting, Custom Detection Rules in Defender XDR
- Logs, Threat Hunting and Analytics in Microsoft Sentinel
- Build Playbooks in Microsoft Sentinel
- Build Incident Response Playbooks and integrate steps through KQL opportunities

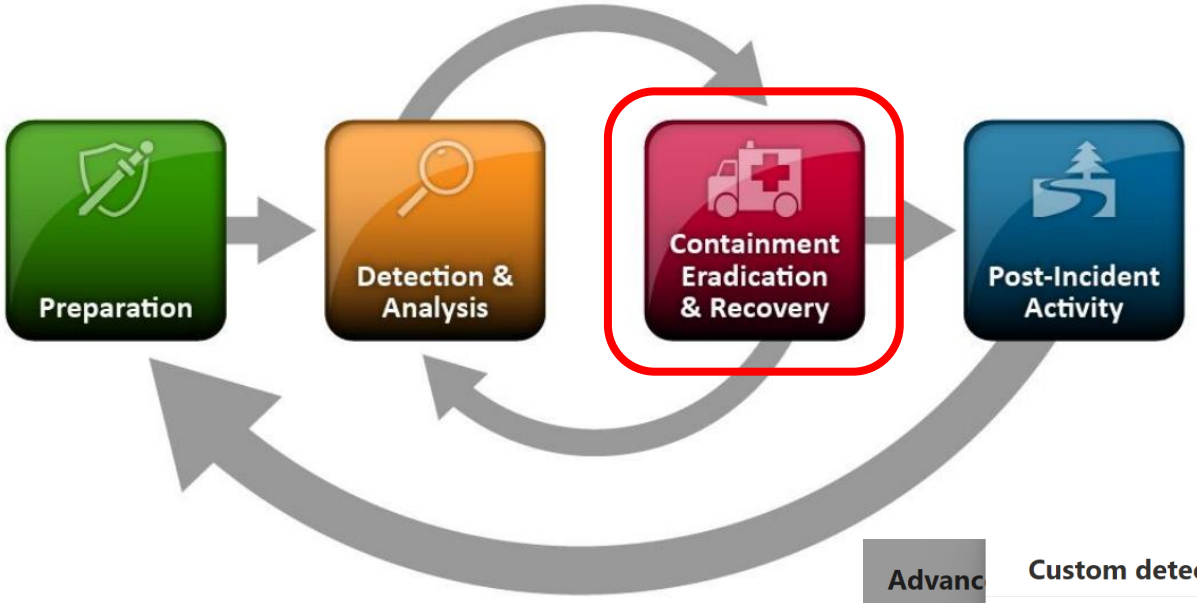
Incident Response Life Cycle



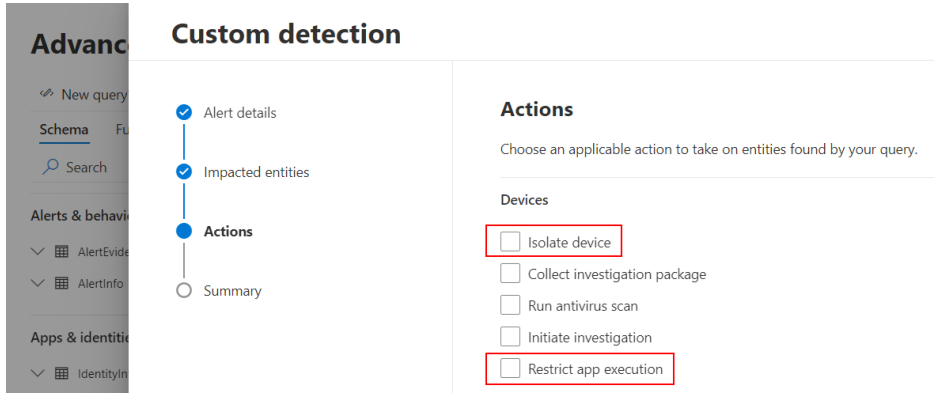
- Advanced Hunting in Defender XDR
- Logs in Microsoft Sentinel
- Opportunity from Incident in Microsoft Sentinel to Pivot to Logs and investigate



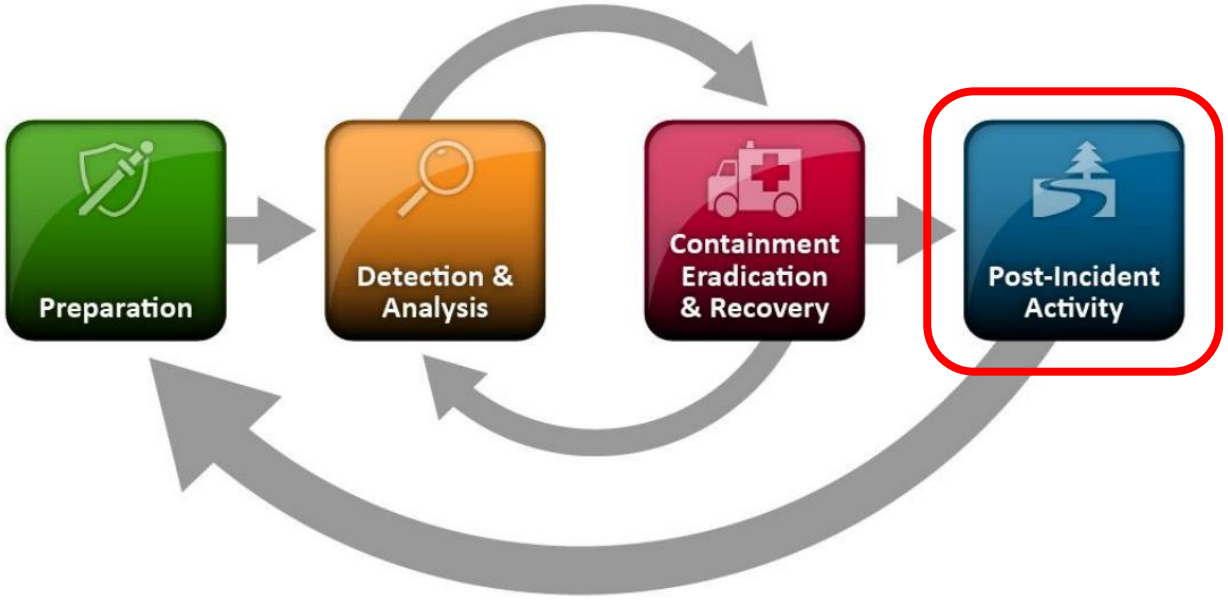
Incident Response Life Cycle



- Custom Detection Rules from Defender XDR could lead to containment actions
- Playbooks in Microsoft Sentinel could also lead to containment actions



Incident Response Life Cycle



- Review Custom Detection Rules, Analytics and Playbooks
- Review KQL Queries stored in Queries at Advanced Hunting and Logs
- Build new queries if required and use them for Threat Hunting or Detection
- Keep relevant documentation up to date

Playbook step investigation

The following query could be a step from a Business Email Compromise (BEC) Incident Response Playbook and will present email details that have been identified as suspicious after delivery.

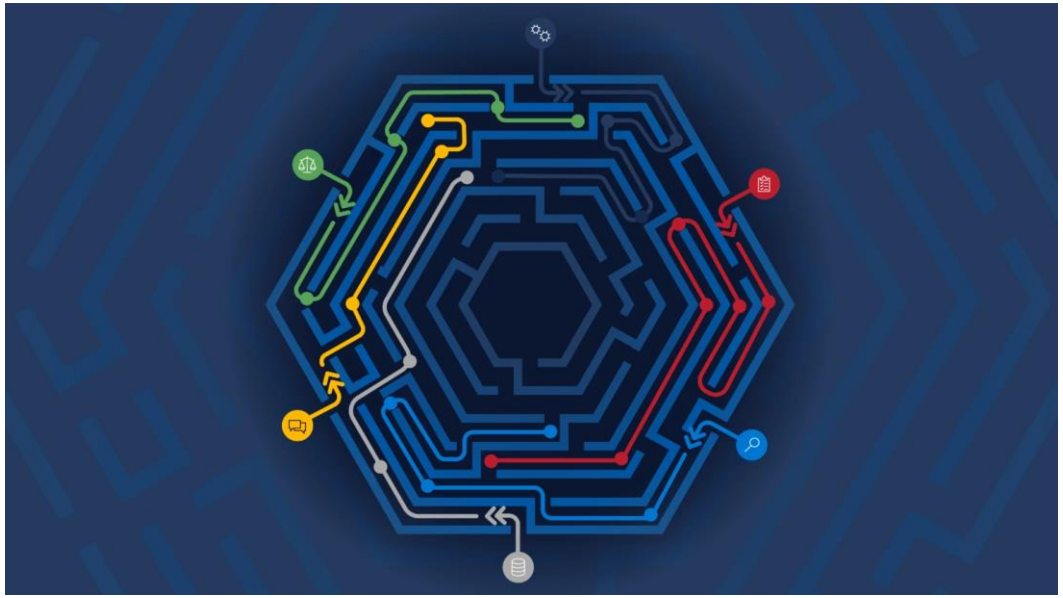
```
let CompromizedEmailAddress = ""; // Insert the email address of the compromised email address
let Timeframe = 2d; // Choose the best timeframe for your investigation
let EmailInformation = EmailEvents
    | where RecipientEmailAddress == CompromizedEmailAddress
    | where DeliveryAction != "Blocked"
    | project Timestamp, NetworkMessageId, SenderMailFromAddress, SenderFromAddress, SenderDisplayName, ThreatNames;
EmailInformation
    | join (EmailPostDeliveryEvents
    | where ThreatTypes != ""
    | project Timestamp, NetworkMessageId, Action, ActionType, ActionTrigger, ActionResult,
      DeliveryLocation, ThreatTypes, DetectionMethods
) on NetworkMessageId
| sort by Timestamp desc
```

wrapping up incident response

Further Incident Response Opportunities:

- Interact through API to build automations or conduct investigation actions.
- You may use Azure Data Explorer (ADX) to load your evidence files for analysis.

"If you're proactive, you focus on preparing. If you're reactive, you end up focusing on repairing."
— John C. Maxwell



Closing remarks

You can familiarize with KQL in many ways:

- Growing community of query contributions on GitHub, blogs, Microsoft Tech Community blogs, podcasts, newsletters etc. Don’t forget, queries come as-is, review and test before formal roll-out. Make sure they fit your environment and requirements!
- Formal training providers apart from SC-200 and Ninja Training, already out there.
  - Also, book is on the way ☺
- CTF-a-like learning, Kusto Detective Agency and KC7 Cyber offer gamified learning experience.

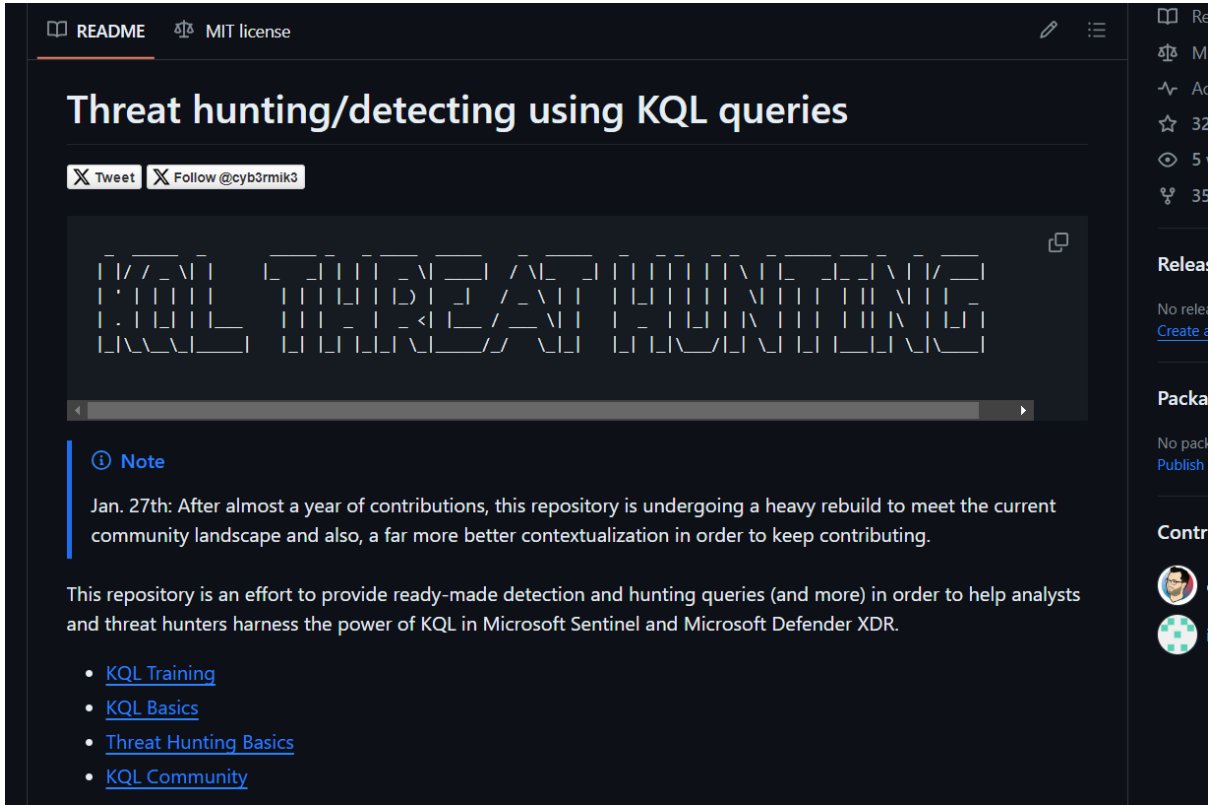
There is only one way to learn KQL…

Query

```
1   LearnKQL
2   | where AnalystAction has "Practice"
```

Resources

https://github/cyb3rmik3



Presentation notes & References:
https://github.com/cyb3rmik3/presentations

AI, Cloud & Modern Workplace Conference 2024

15, 16 & 17 February , Online Conference

Thank You!