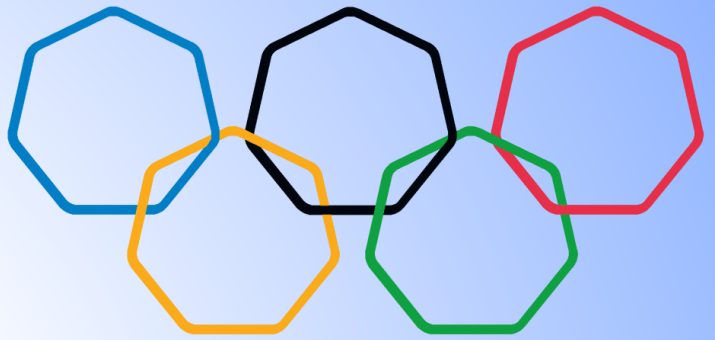
What agent to trust with your k8s:



HENRIK REXED
CLOUDNATIVE ADVOCATE

Henrik Rexed

Cloud Native Advocate

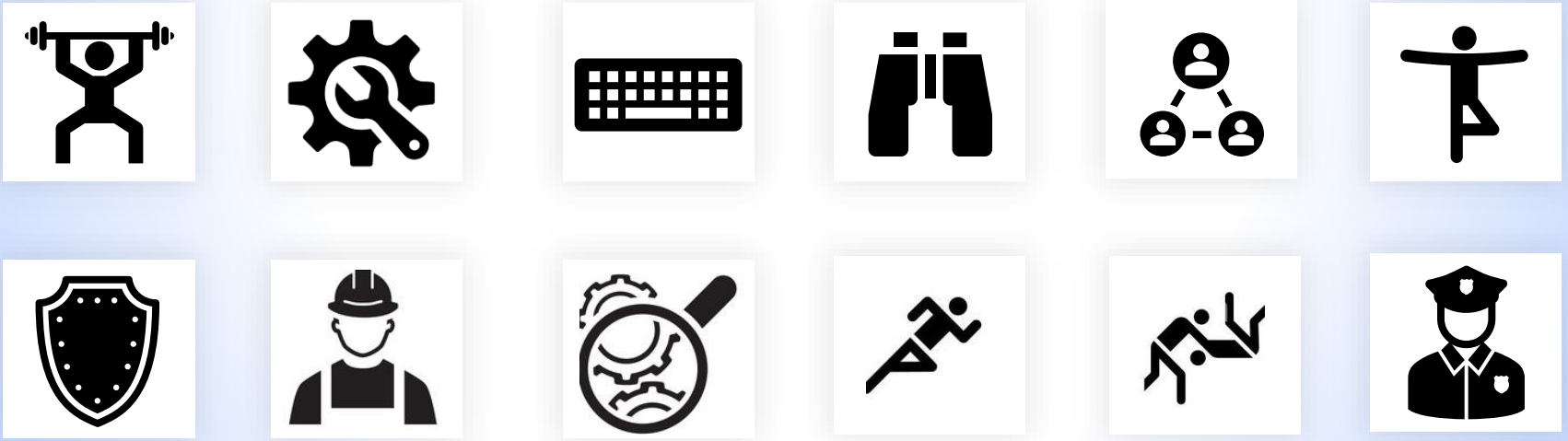
- 15+ years of Performance engineering
- Owner of : IsitObservable
- Producer of : Perfbytes

DISCLAIMER

- No Spartans, Eagles or Shields were harmed in the making of this presentation
- The intention behind this talk track is not to assign blame to any CNCF project.
- This session is made to help the community in choosing their runtime security agent.

KubeOlympics 2024

KubeOlympics 2024



The Athletes

Falco

Tetragon

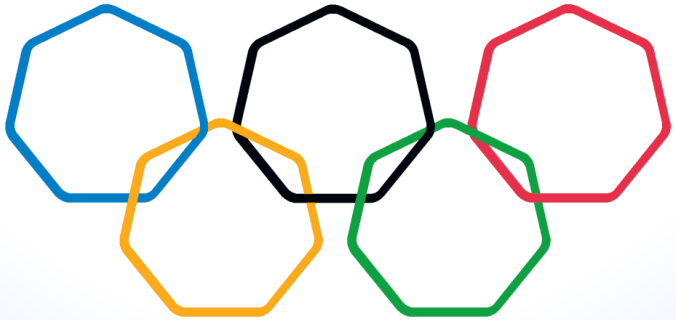
Kubearmor

Tracee

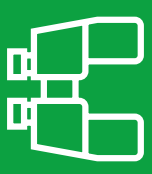
If you stay with me you will ...

- What we expect from a runtime security agent
- Compare each solution in :
  - Desing Experience
  - The Components required
  - The Observability
- See various Benchmarking results
- Recommendation on which agents needs to used under specific conditions

THE RULES



Performance



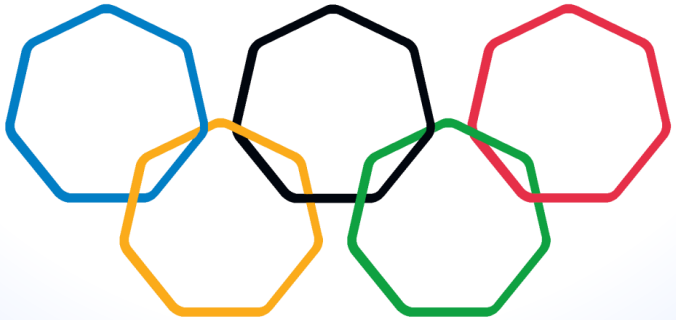
Observability

Components

Filter

Capture

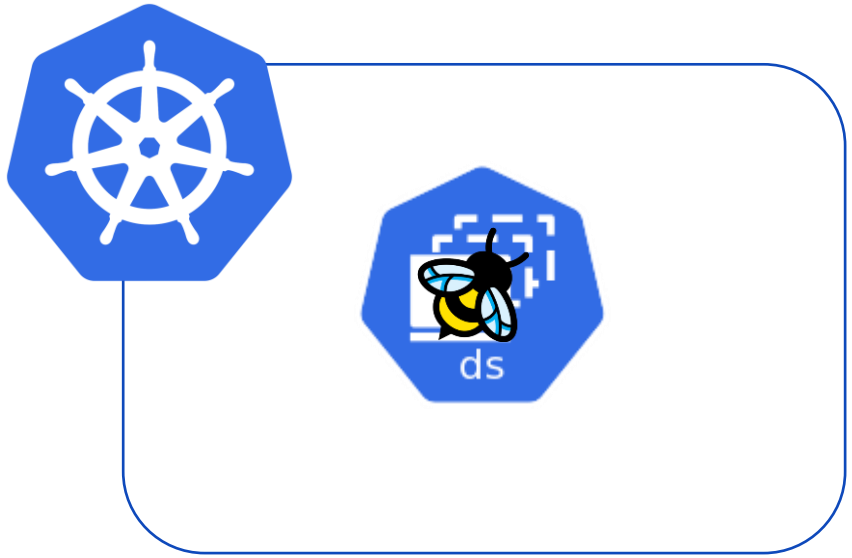
STAGE 1: COMPONENTS



Compounents

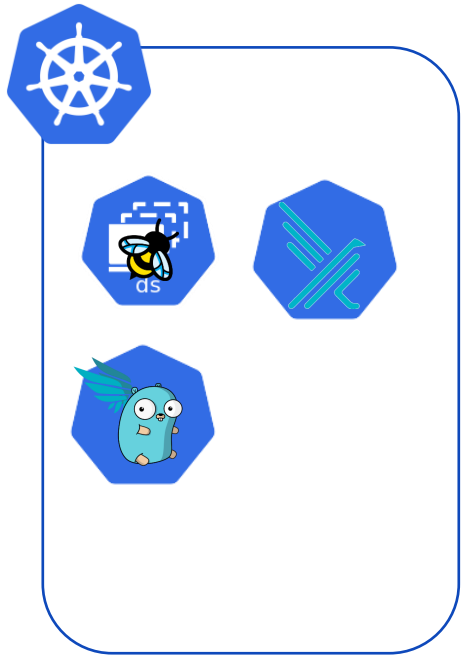
- A security agent usually relies on ebpf to collect the kernel or user event of our environment

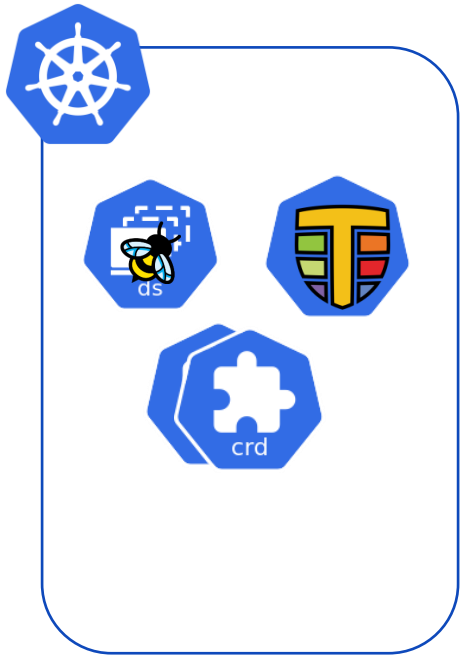
- In k8S , the security agent has usually at least :
  - A daemonset deploying the ebpf probe on each node of our cluster

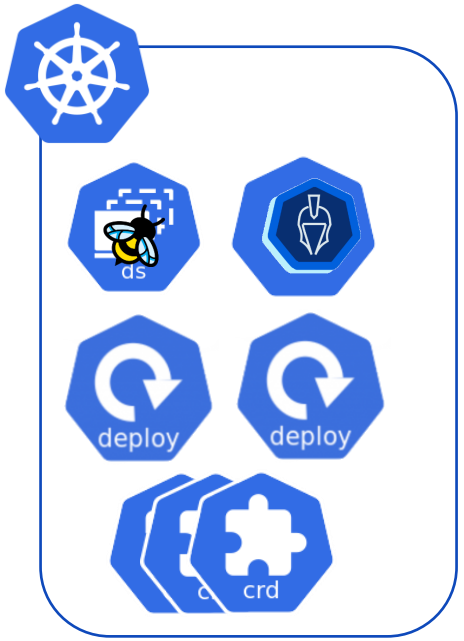


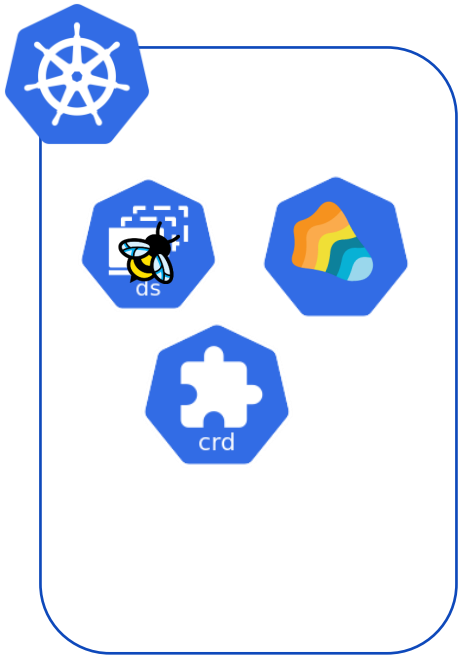
Components required

- Falco
- Tetragon
- KubeArmor
- Tracee

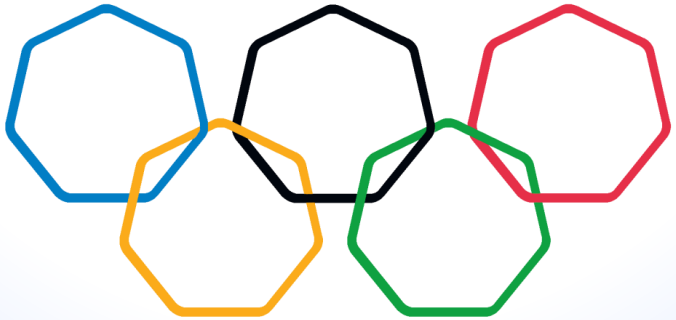






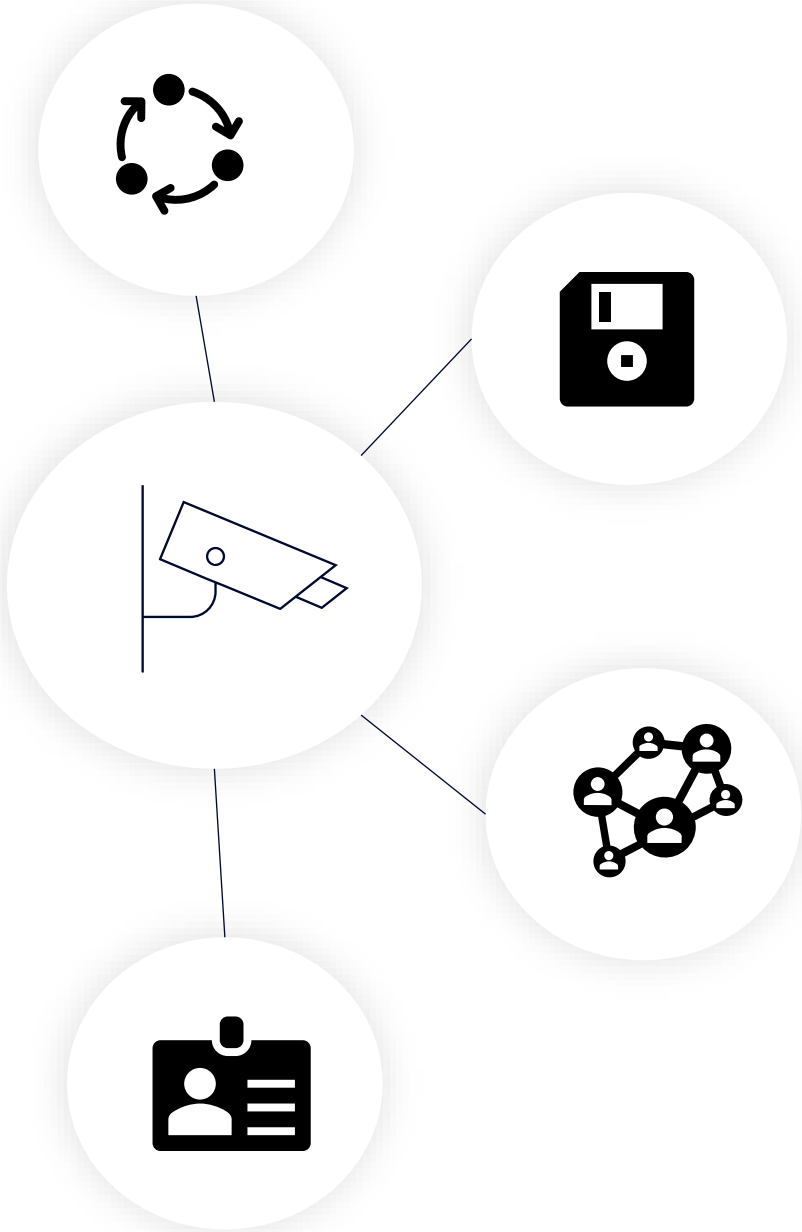


STAGE 2: CAPTURE & FILTER



Capture

- Be able to capture events from our k8S cluster related to :
  - Process launched/ finished
  - File Access Read/Write
  - Networking details
  - Change of privileges
  - …etc.
- Get the details attached to the event:
  - Path: executable path, file path…
  - Process details: pid, process name ,
  - User: userid, groupid
  - Capabilities
  - Container: image , name
  - K8s : pod name, namespace, ..etc

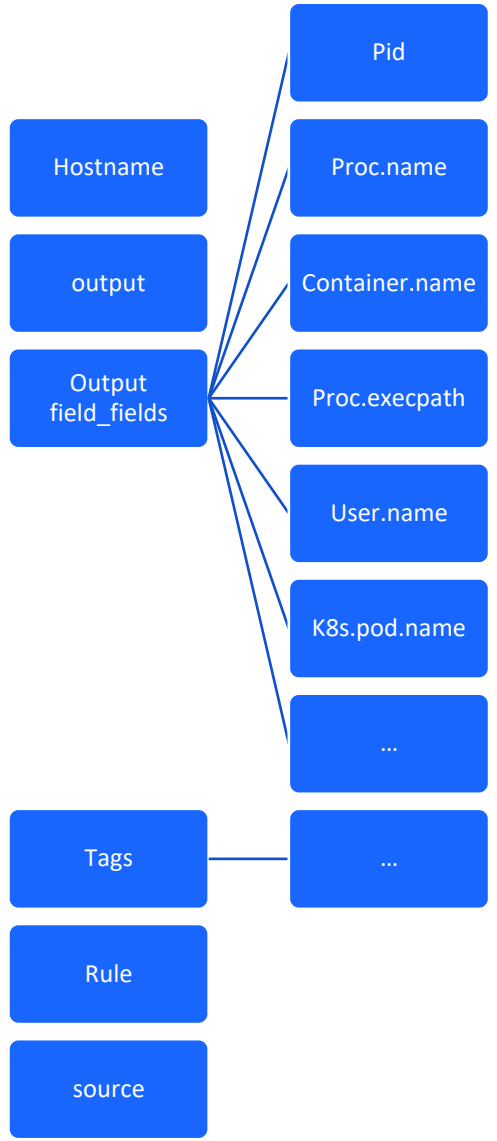


Filter

- Filtering means having the option to create a policy filtering the event on :
  - Process properties
  - User details
  - K8S metada
  - File details
  - Syscall/ kernel functions or event
- Have Predefined Rules
- Be able to react

Falco Capture

- Falco is rule engine producing logs if the falco event is matching a Falco rule
- Falco receive event from :
  - Falco kernel agent
  - External plugins
- Falco provides a SDK allowing us to build our custom Plug
- Falco agents captures :
  - Process details
  - Syscall
  - Tracepoints
  - File



Falco rule

- Falco starts by loading the rule files that would be used to generate the events. By default, Falco provide a default sets of rule.

- Falco provides “fields” helping us to define our filtering rule based on:
  - Event information
  - Proces
  - file,
  - Syscalls

- Having a global rule file allow to to reuse filtering rules or conditions with macros

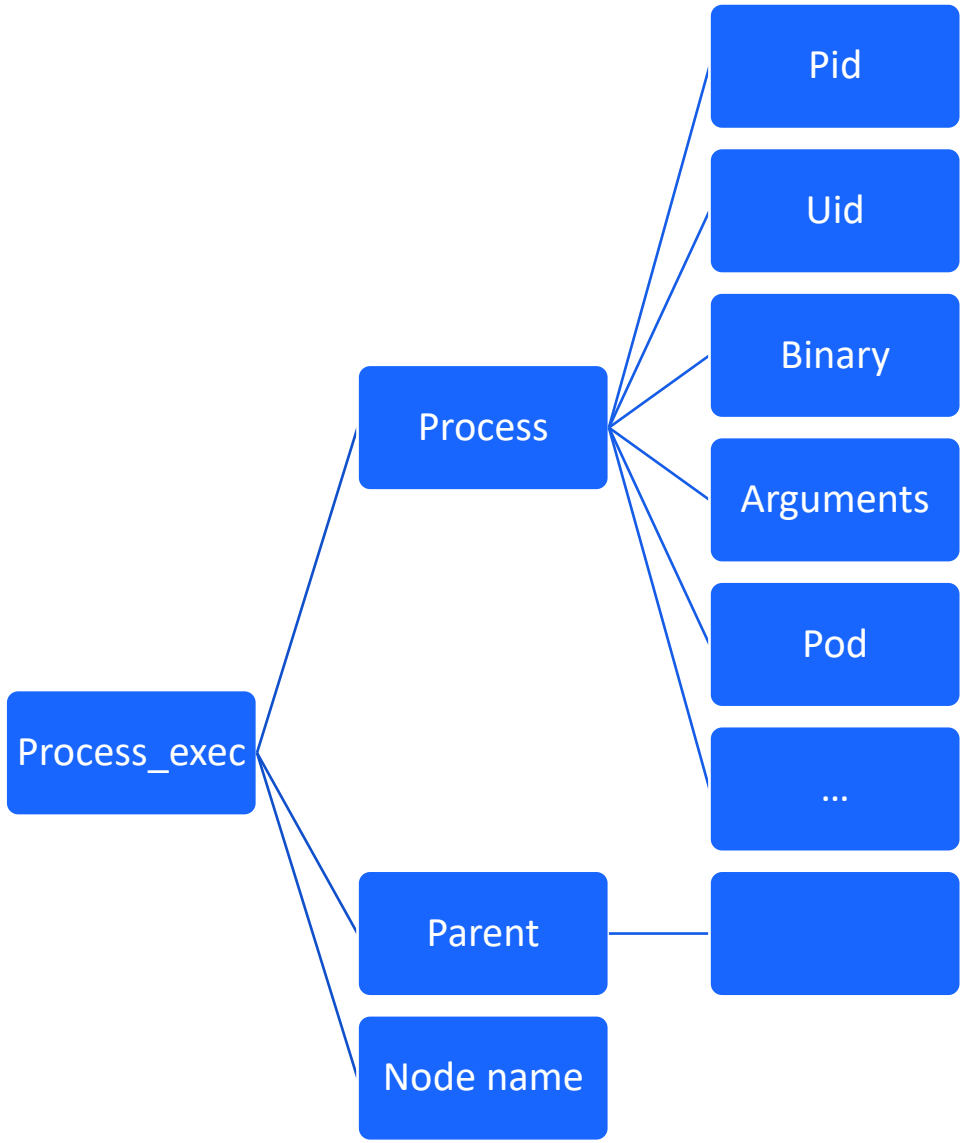
```
  macro: access_file
  condition: evt.type=open

- rule: program_accesses_file
  desc: track whenever a set of programs opens a file
  condition: (access_file) and proc.name in (cat, ls)
  output: a tracked program opened a file
(user=%user.name command=%proc.cmdline file=%fd.name)
  priority: INFO

- rule: test_rule
  desc: test rule description
  condition: evt.type = close
  output: user=%user.name command=%proc.cmdline
file=%fd.name
  priority: INFO
  enabled: false
```

Tetragon capture

- Tetragon will by default produce events related to :
  - Process execution
  - Process exit
- If you deploy Policies, it will produce events matching our policy rule
- Policy required to defined a hook point to our system using:
  - Kprobe
  - Tracepoints
  - Ubprobe
  - llsm



Tetragon policy

- Building a policy means defining creating a TracingPolicy defining:
  - the right hookpoint
  - The index argument we would like to extract
  - The data type of the argument
- Filtering in Tetragon means applying a selector on :
  - The Args extracted
  - The process, file ...etc details
  - The user
  - The k8S metadata
- TracingPolicy is also allowing us to define how to react on the event (block, Audit….etc)

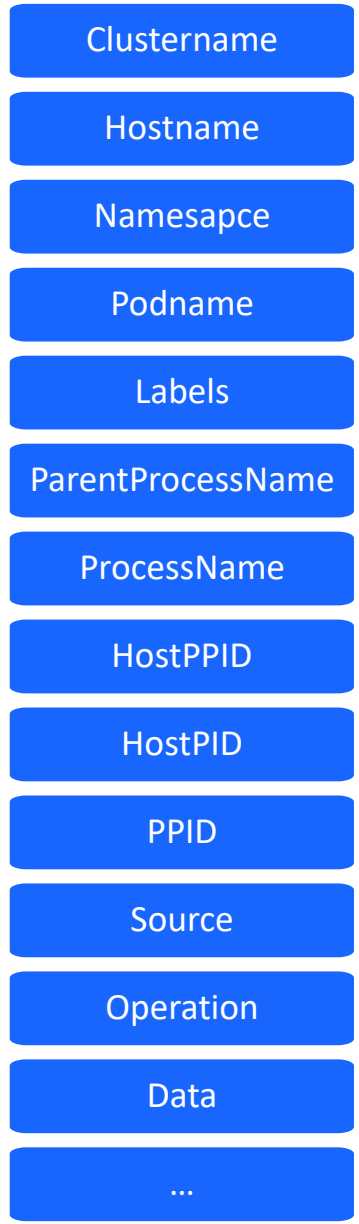
```
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: "k8s-api-calls"
spec:
  kprobes:
    - call: "tcp_connect"
      syscall: false
      args:
        - index: 0
          type: "sock"
      selectors:
        - matchArgs:
            - index: 0
              operator: "DAddr"
              values:
                - "10.43.0.1"
                - "10.1.0.0/20"
          matchBinaries:
            - operator: "NotIn"
              values:
                - "/usr/bin/rancher"
                - "/usr/bin/dumb-init“
```

Kubearmor capture

- KubeArmor will only produces events maching:
  - The Policiy deployed
  - Or the K8S objects having the right annotations

- KubeArmor will capture :
  - Process
  - File
  - Network
  - Capatbilities
  - Syscalll

- KubeArmor Operator provides more feature to manage with the help of annotations
  - The type of events to report (file, process, network)
  - The default posture on how to react



KubeArmor Policy

- KubeArmor policies are defined by building a :
  - KubeArmorClusterPolicy
  - KubeArmorPolicy
  - KubeArmorHostPolicy

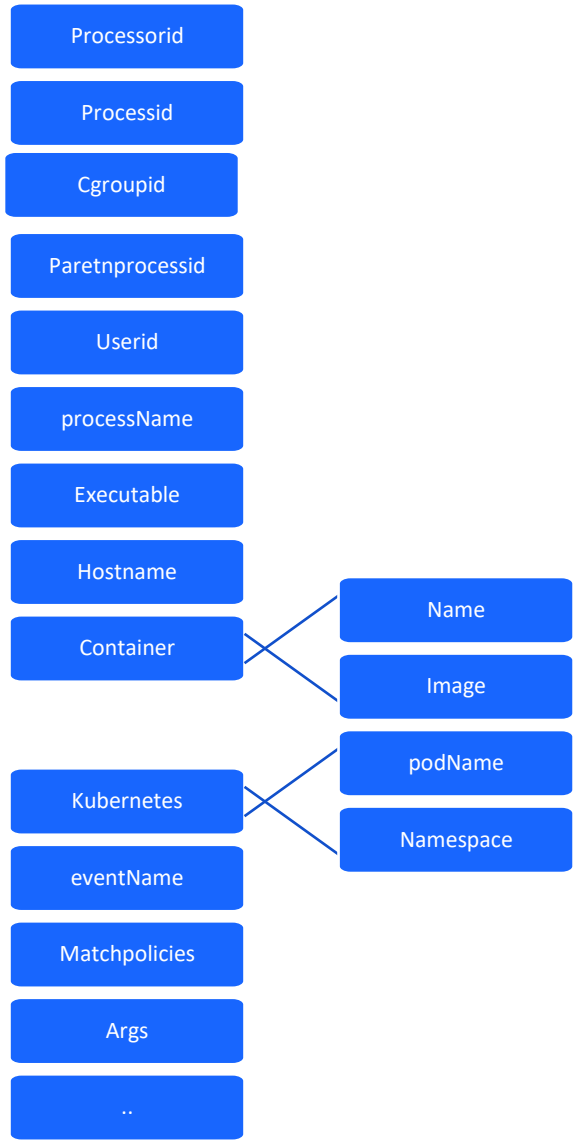
- The policy defines :
  - The Tags
  - The message of the event
  - The Selector to filter to a specific namespace or workload
  - The rule for the event type (process, file, network..Etc)
  - And the action : Block, Audit, Allow

```
apiVersion: security.kubearmor.com/v1
kind: KubeArmorClusterPolicy
metadata:
  name: ksp-nist-remote-access-and-control

spec:
  tags: ["NIST","system","ksp","AC-17(1)"]
  message: "warning! someone tried to access and control"
  selector:
    matchExpressions:
      - key: namespace
        operator: NotIn
        values:
          - kube-system
          - istio-system
  process:
    severity: 4
    matchPaths:
      - path: /usr/bin/ssh
      - path: /etc/ssh
    action: Audit
```

Tracee capture

- Tracee will only produce events matching the policies deployed
- Tracee will capture :
  - Security event
  - Network details
  - Anything using a given list of syscalls (process, file…Etc)
- The type of events detected will rely on a set of “signatures” defining
  - What to capture
  - What to decode
- Tracee provides a “sdk” helping us to build our own signature



Tracee Policy

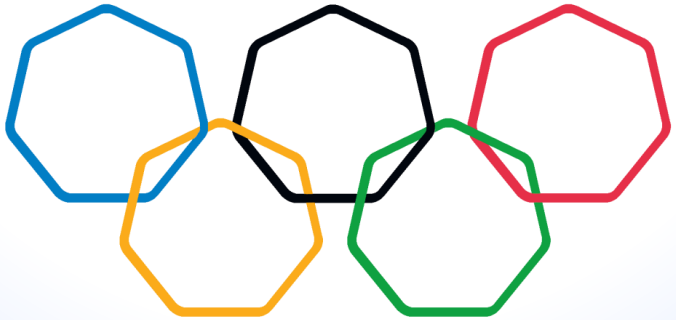
- Tracee policies are defined by configuring a :
  - Policy
- Tracee provides a default policy based on the security signature
- The policy defines :
  - The scope
  - The set of rules defined by a event name and filters

```
apiVersion: tracee.aquasec.com/v1beta1
kind: Policy
metadata:
  name: sample-data-filter
  annotations:
    description: sample data filter
spec:
  scope:
    - global
  rules:
    - event: vfs_read
      filters:
        - data.pathname=/etc/*
        - data.pathname=/etc/fstab
        - data.pathname=/etc/crontab
        - data.pathname=/etc/hosts
        - data.pathname=/etc/hosts.allow
```

Summary

				
Capture	✗	✗	✗	✗
Filter	✗	✗	✗	✗
Default Policy	✗			✗
Ability to react		✗	✗	
Customizable	✗	✗		✗

Stage 3: Observability

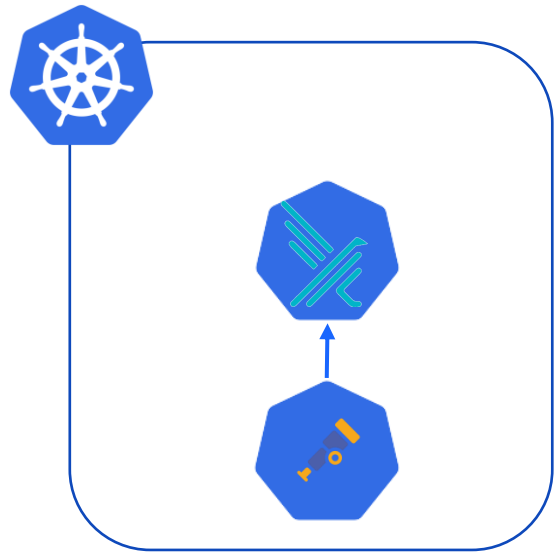


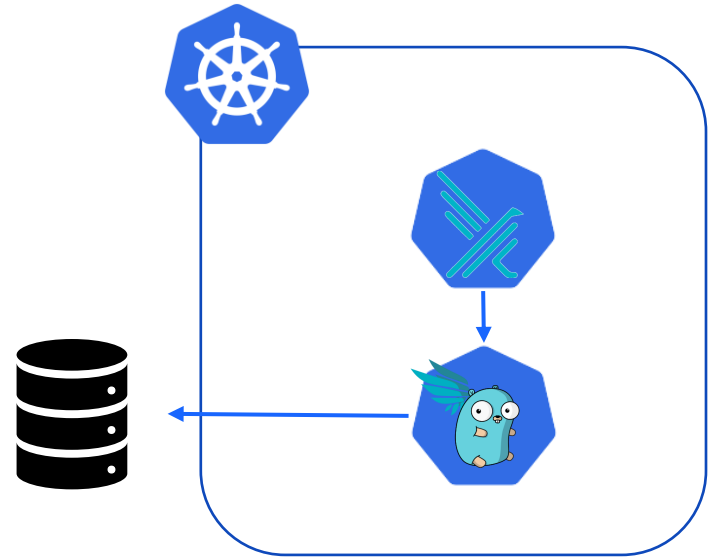
Observability

- Extend observability with the event produced by :
  - Collecting the logs produced
  - Create parsing rule
- Limit the number of fields exported
- Limit the events by enabling throttling
- Be able to report health metrics related to :
  - The policy deployed
  - The various components required to run the runtime agent

Falco

- Falco events would be available in the Falco agent logs
- The event structure highly depends on the rule created and the syscall used.
- To simply the log collection we usually rely on FalcoSidekick that provides a large number of integrations
- Falco provides a metric server reporting :
  - Metrics related to the rule
  - Actual health of Falco agent

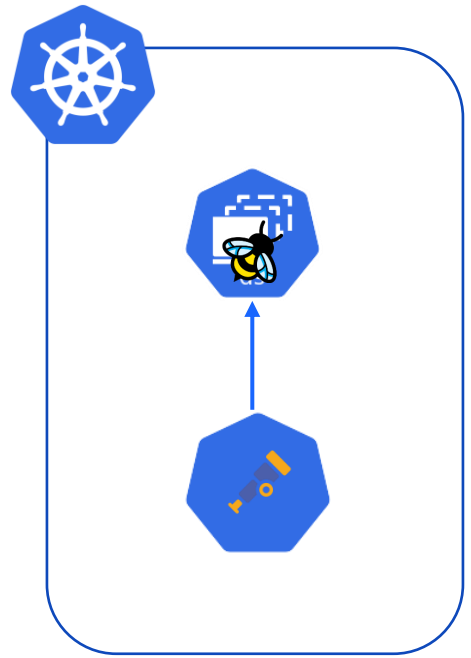




Tetragon

- Tetragon expose all the events directly in the logs of the agent

- Tetragon expose 2 Prometheus exporter:
  - Agent: sharing details on the policies deployed and the health
  - Operator: Sharing the health of the agent

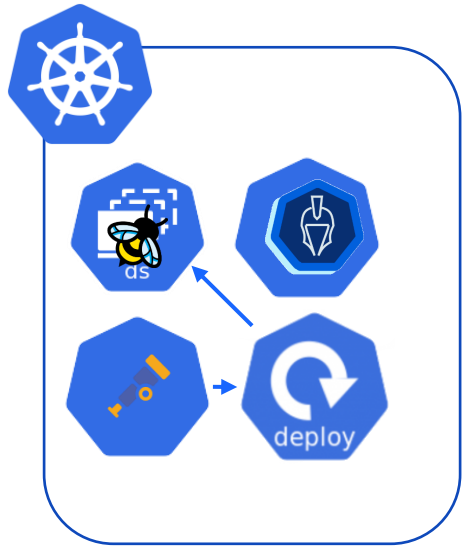


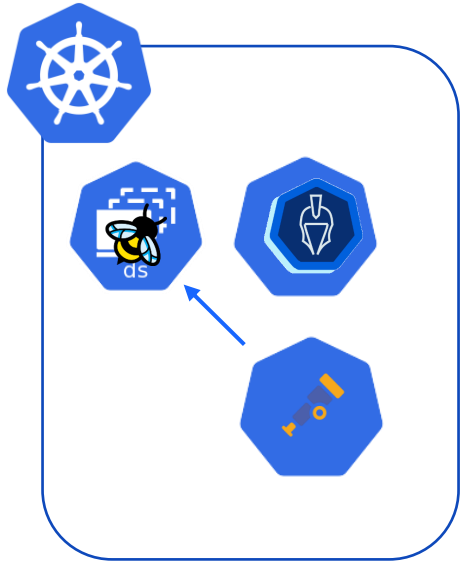
Kubearmor

- Kubearmor will not produce events in the logs from the agents

- To collect events you will either need to :
  - Enable the logging option on the Relay Server
  - Use the Kubearmor receiver that will collect the logs from the agent. This receiver is currently only compatible with v0.96 of the collector

  - Kubearmor is not providing any metrics



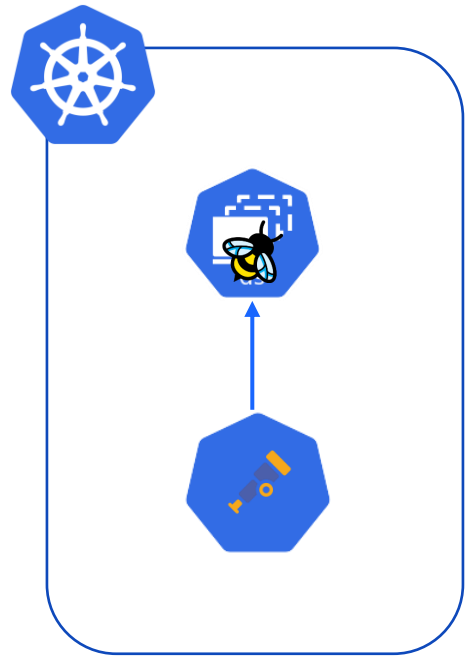


Tracee

- Tracee is producing the event directly at the logs of the agent (default settings)

- Tracee can also push the logs using :
  - the FluentForward protocol
  - A webhook endpoint

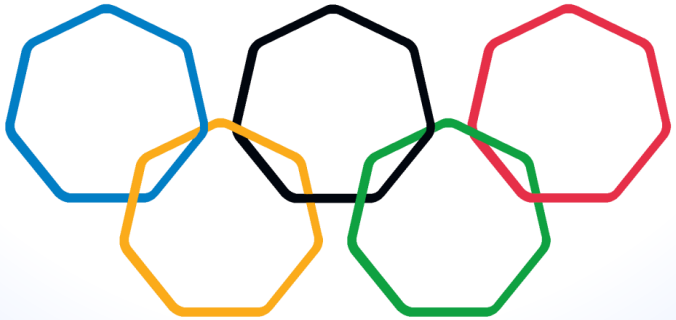
- The Traceee agent expose Prometheus metrics related to the events produced and the errors.



Summary

				
Collecting events	✗	✗	✗	✗
Extend observability	✗	✗	✗	✗
Rule metrics	✗	✗		✗
Heath metrics	✗	✗		✗

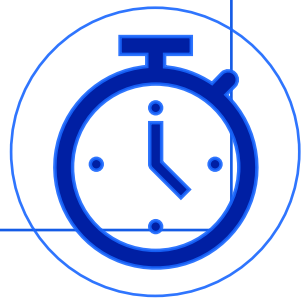
STAGE 4: PERFORMANCE



Kpi that we want to measure

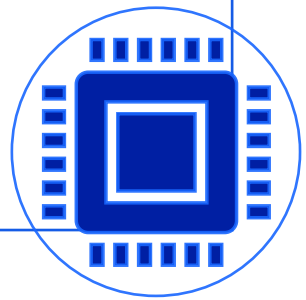
- Measure the latency added in our application

Latency



- Cpu usage
- Memory usage

Resource usage



What type of policies we would apply

- Using the default policies

Falco

- k8S api calls
- Service account files
- Sensitive files
- Egress communication
- Install tools
- Network activities
- Process spawned

Tetragon

- Block unauthorize binaries
- Sensitive files
- Audit write in sensitive folder
- Suspicious Network tools
- External access

Kubearmor

- Sensitive files
- New containers created
- Process spawned
- Security signature
- Read write access from non root user

Tracee

The various tests executed


No Agent


Falco


Tetragon with default events


Tetragon with default events & policies


KubeArmor with policies and no events

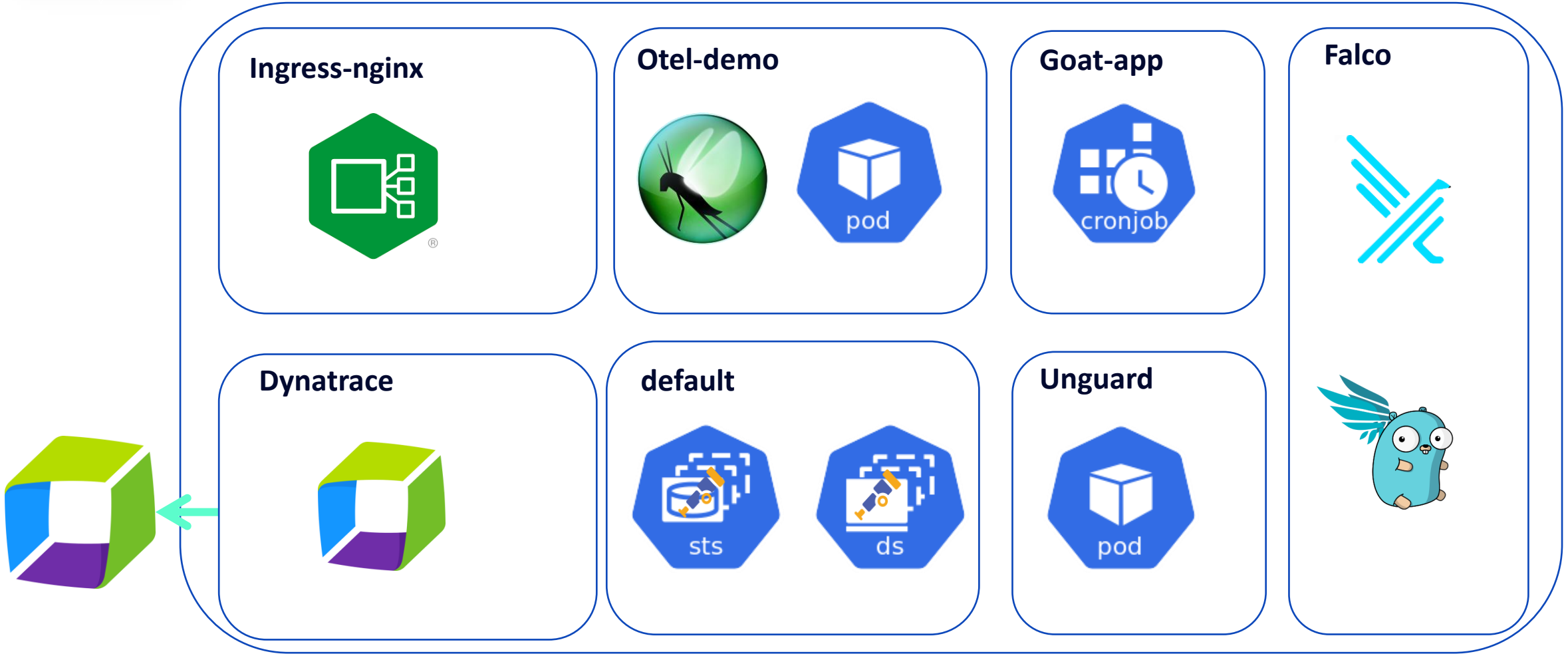

KubeArmor with policies and events


KubeArmor no policies but with events

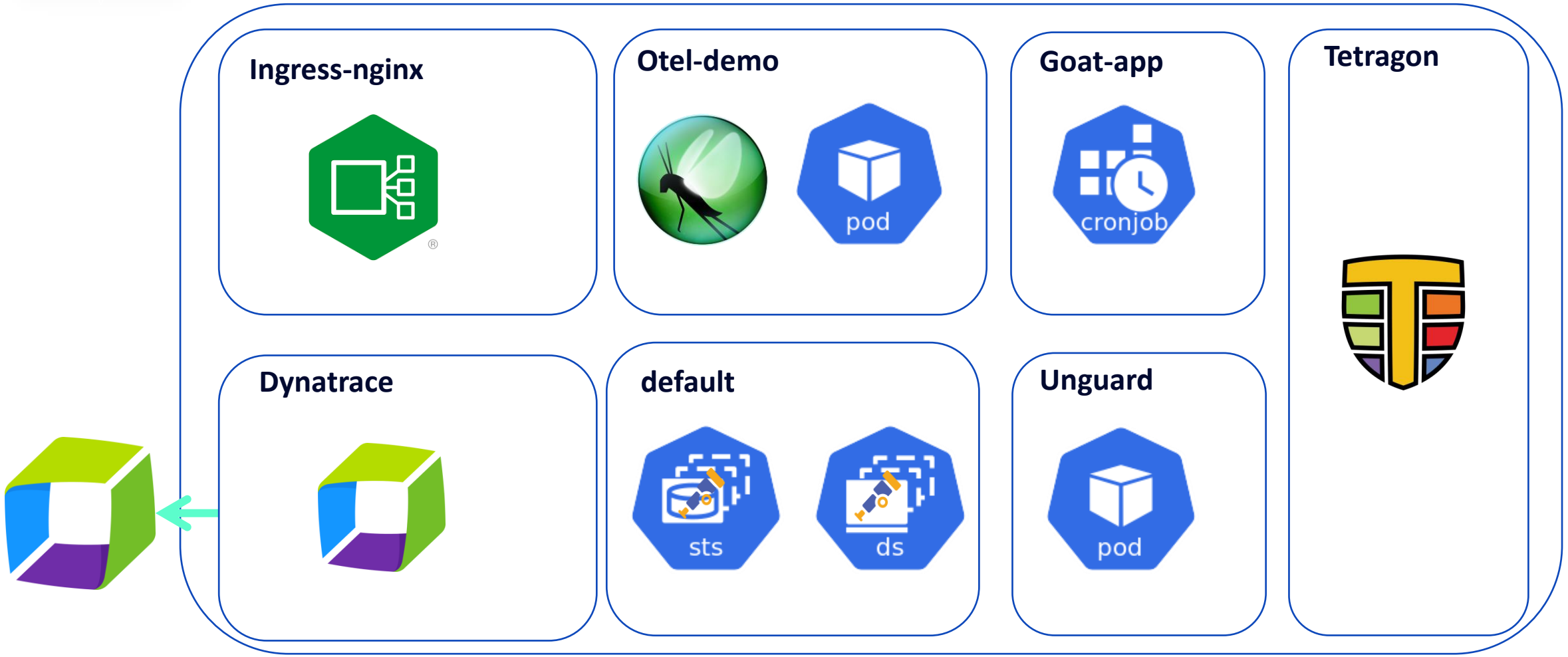

Tracee with default policy


Tracee with default policy and custom policies

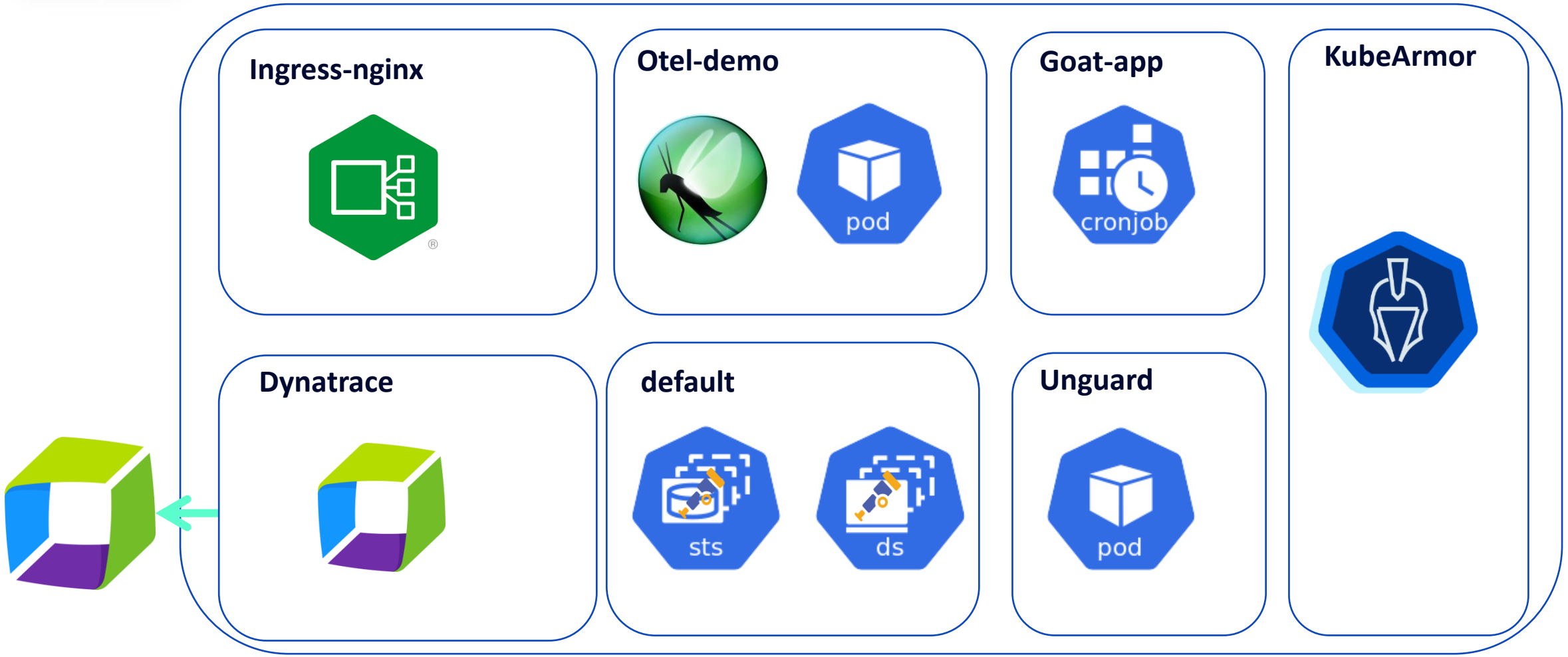
Falco Architecture



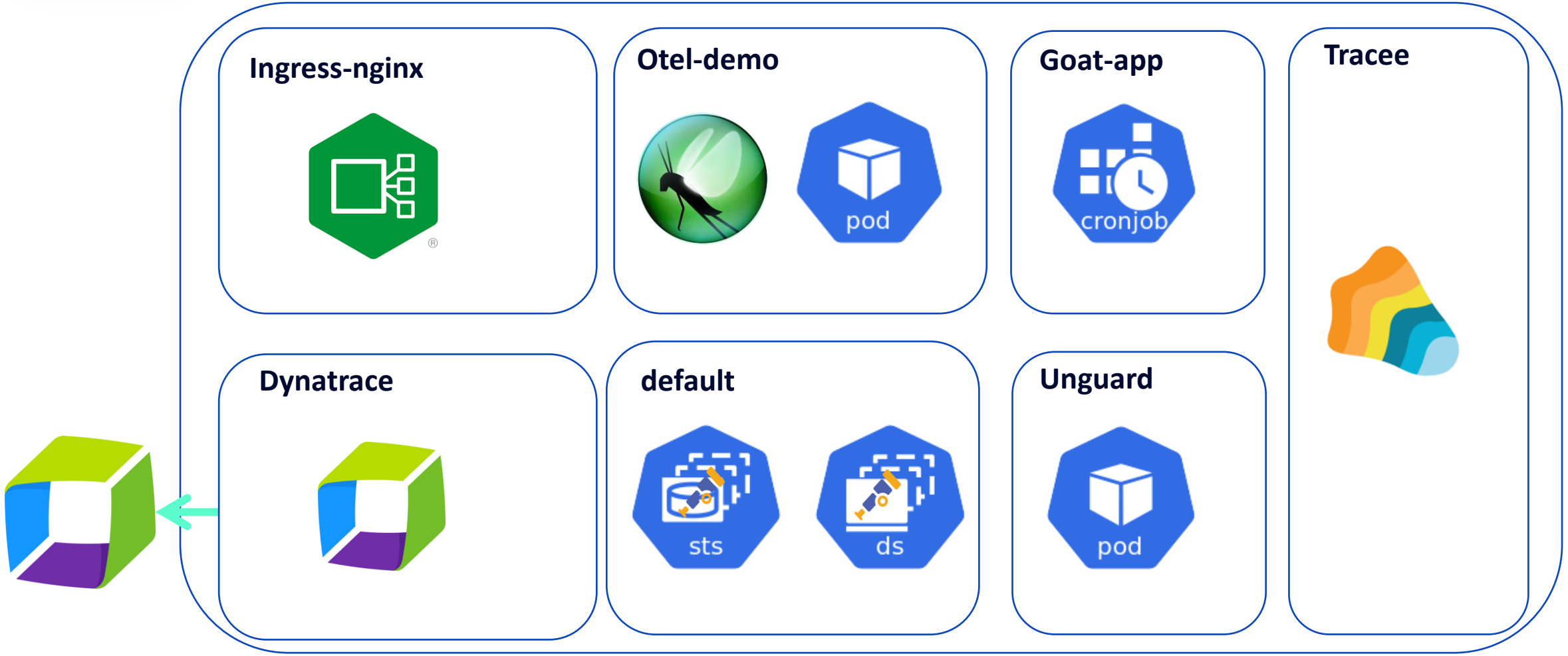
Tetragon Architecture



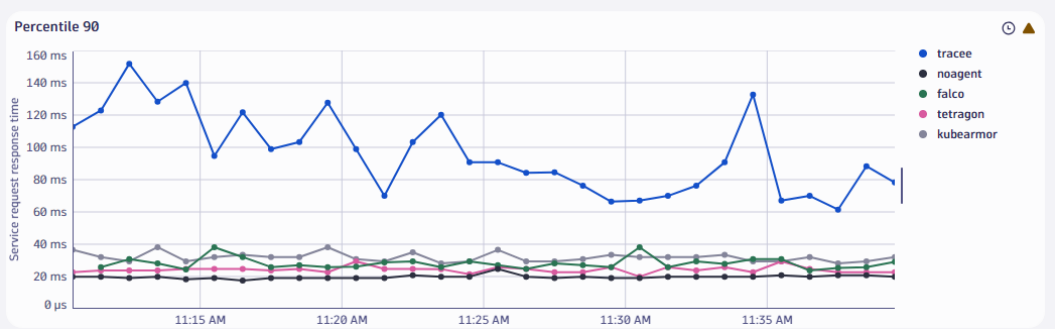
KubeArmor Architecture



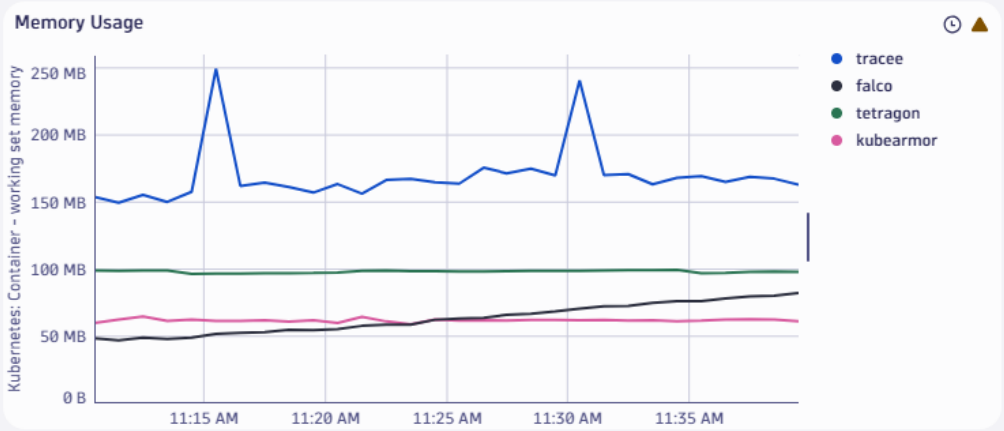
Tracee Architecture

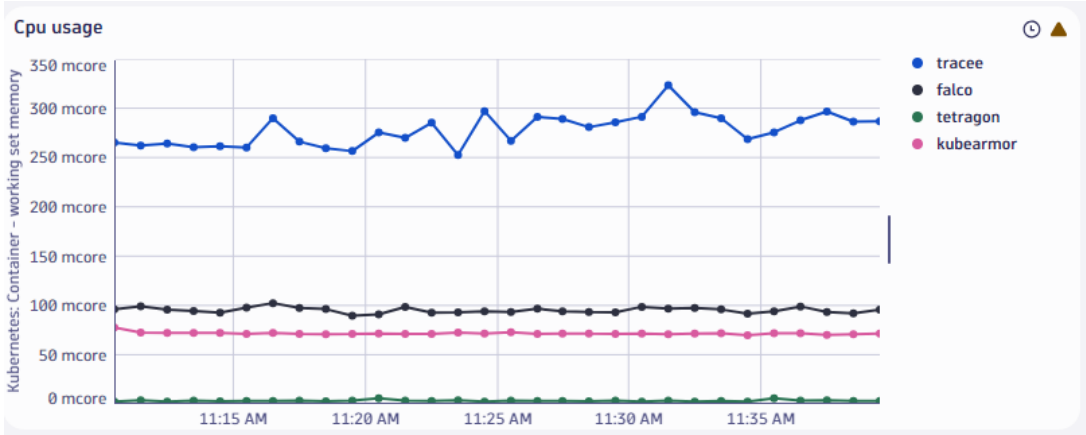


Constant Load with no policies

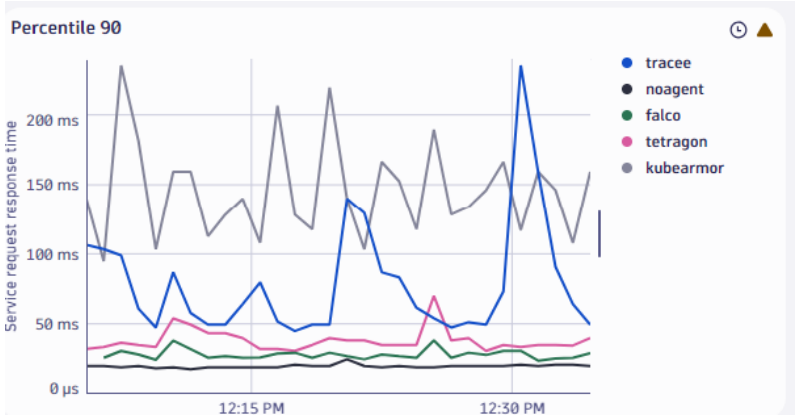


Falco	+10,84ms
Tetragon	+ 5,03ms
KubeArmor	+15,73ms
Tracee	+110,79ms

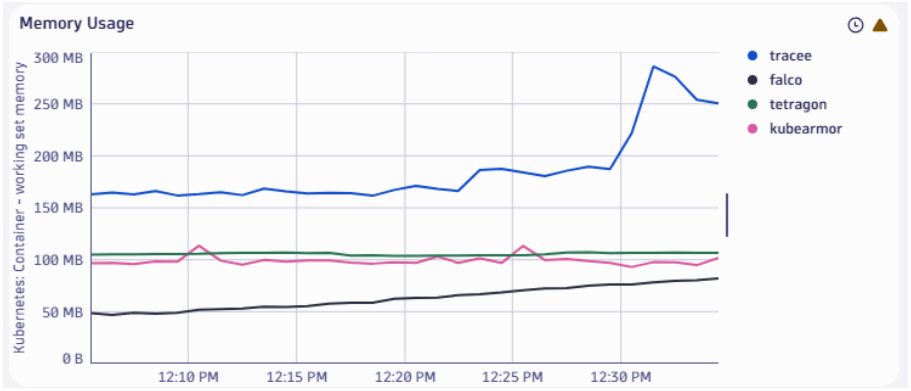


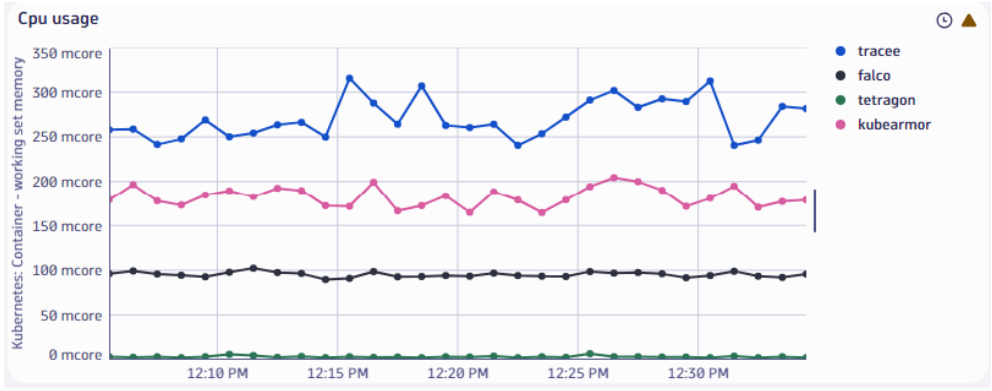


Constant Load with policies

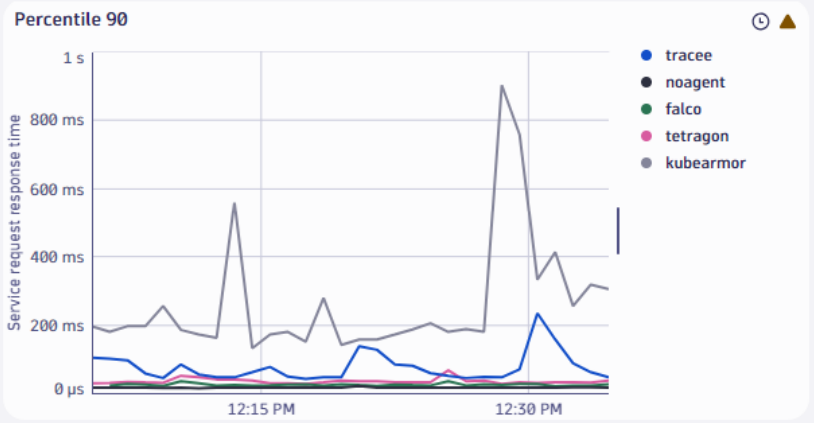


Falco	+ 10,84ms
Tetragon	+26,13ms
KubeArmor	+176,62ms
Tracee	+114,23ms

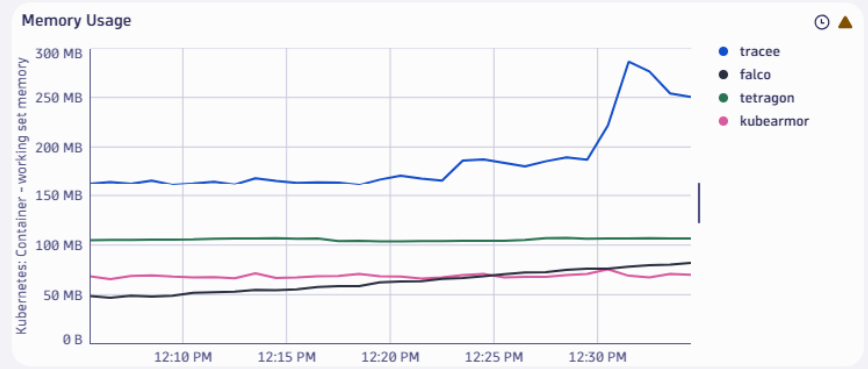


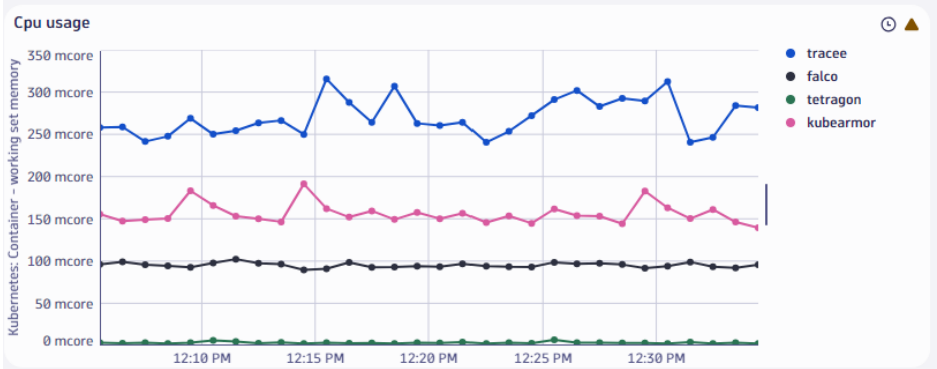


Constant Load with policies & events



Falco	+ 10,84ms
Tetragon	+26,13ms
KubeArmor	+477,34ms
Tracee	+114,23ms

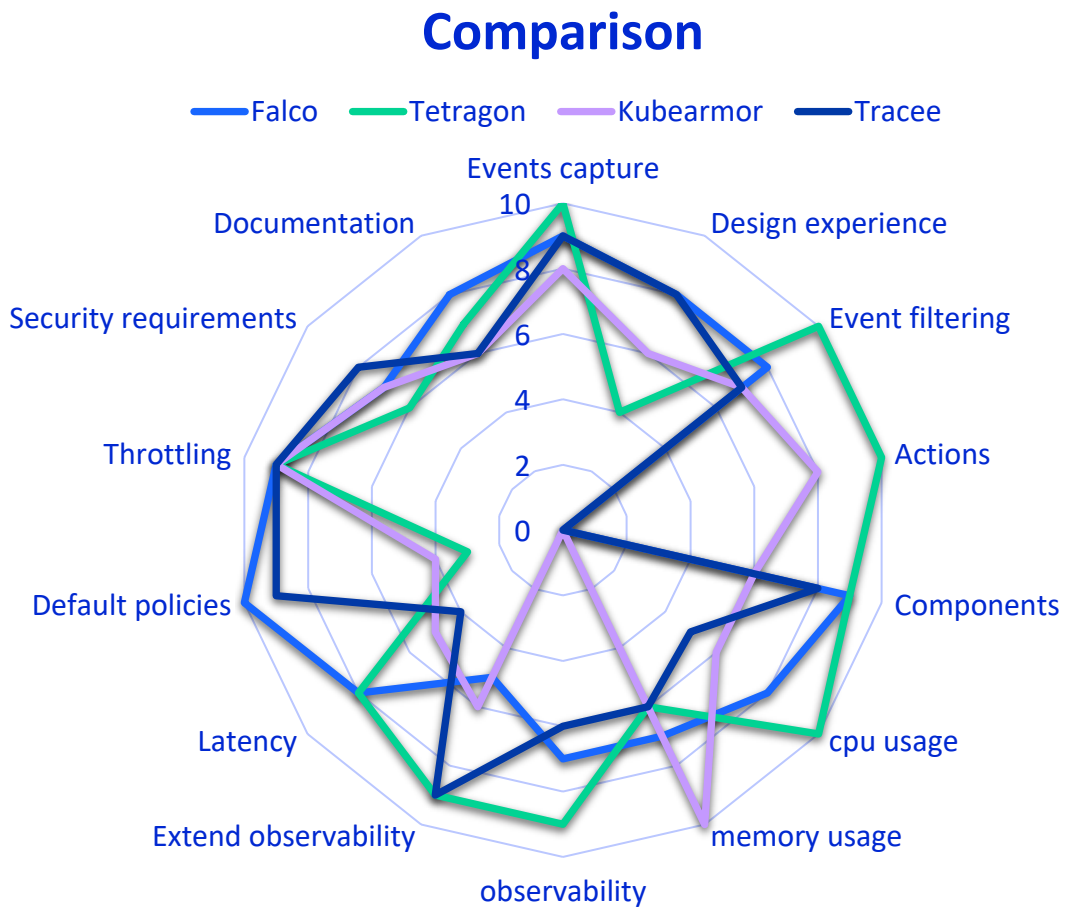




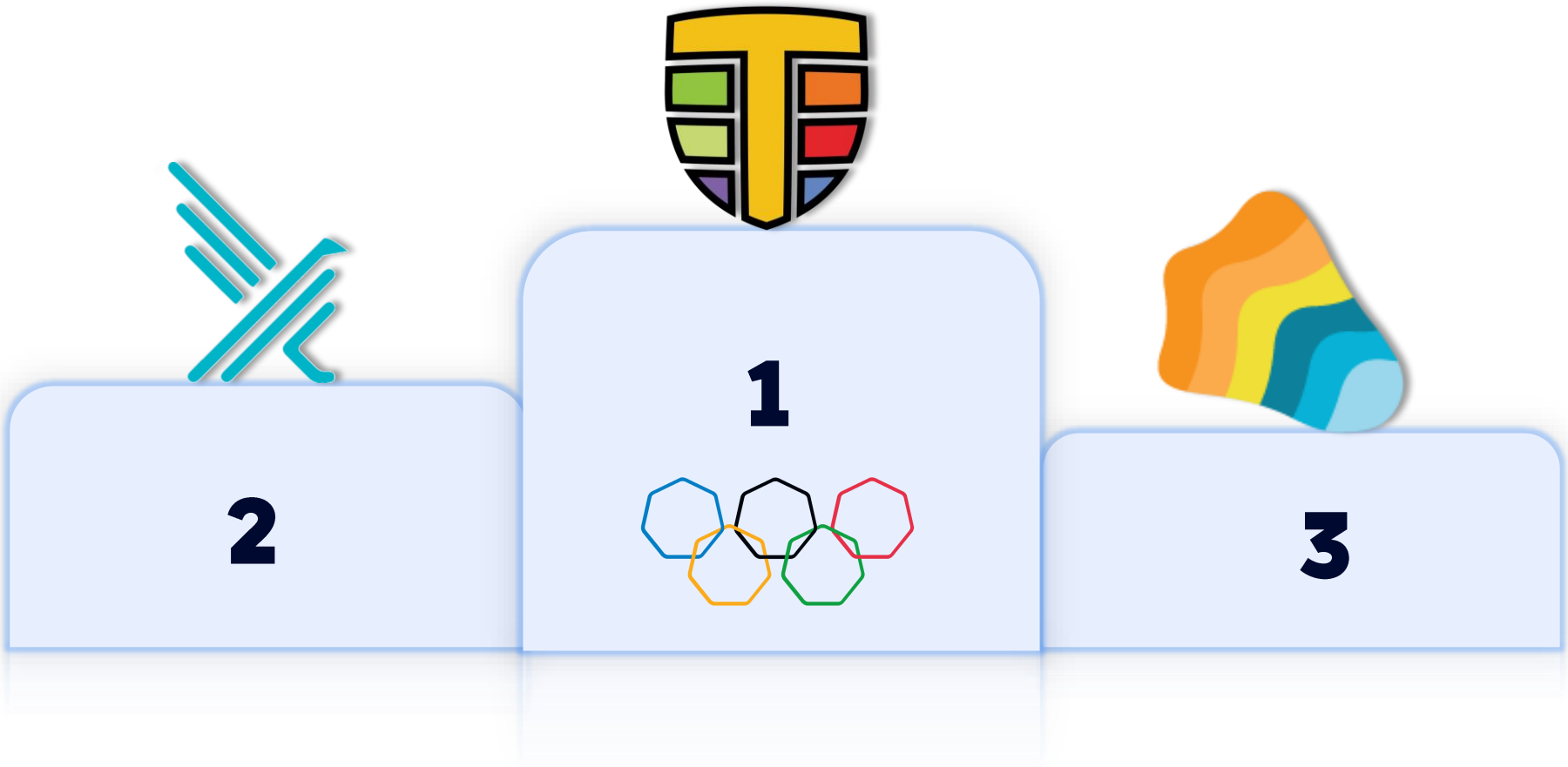
Summary

				
Latency	✕✕✕	✕✕✕	✕	✕
CPU	✕✕	✕✕✕	✕	✕
memory	✕✕	✕✕	✕✕✕	✕

Conclusion


Comparison
Falco
Tetragon
Kubearmor
Tracee
Events capture
Design experience
Event filtering
Actions
Components
cpu usage
memory usage
observability
Extend observability
Latency
Default policies
Throttling
Security requirements
Documentation
10
8
6
4
2
0

Conclusion



Is it observable

- Looking for educational content on Observability , Checkout the YouTube Channel :

Is It Observable

Thank You

dynatrace