※ https://github.com/sysdiglabs/falco-aws-firelens-integration

falco 설치하기 전에 아래와 같은 내용의 values.yaml파일을 생성해준다.

|  |
| --- |
| # Default values for Falco.  ###############################  # General deployment settings #  ###############################  image:  # -- The image pull policy.  pullPolicy: IfNotPresent  # -- The image registry to pull from.  registry: docker.io  # -- The image repository to pull from  repository: falcosecurity/falco-no-driver  # -- The image tag to pull. Overrides the image tag whose default is the chart appVersion.  tag: ""  # -- Secrets containing credentials when pulling from private/secure registries.  imagePullSecrets: []  # -- Put here the new name if you want to override the release name used for Falco components.  nameOverride: ""  # -- Same as nameOverride but for the fullname.  fullnameOverride: ""  # -- Override the deployment namespace  namespaceOverride: ""  rbac:  # Create and use rbac resources when set to true. Needed to fetch k8s metadata from the api-server.  create: true  serviceAccount:  # -- Specifies whether a service account should be created.  create: true  # -- Annotations to add to the service account.  annotations: {}  # -- The name of the service account to use.  # If not set and create is true, a name is generated using the fullname template  name: ""  # -- Add additional pod annotations  podAnnotations: {}  # -- Add additional pod labels  podLabels: {}  # -- Set pod priorityClassName  podPriorityClassName:  # -- Set securityContext for the pods  # These security settings are overriden by the ones specified for the specific  # containers when there is overlap.  podSecurityContext: {}  # Note that `containerSecurityContext`:  # - will not apply to init containers, if any;  # - takes precedence over other automatic configurations (see below).  #  # Based on the `driver` configuration the auto generated settings are:  # 1) driver.enabled = false:  # securityContext: {}  #  # 2) driver.enabled = true and (driver.kind = module || driver.kind = modern-bpf):  # securityContext:  # privileged: true  #  # 3) driver.enabled = true and driver.kind = ebpf:  # securityContext:  # privileged: true  #  # 4) driver.enabled = true and driver.kind = ebpf and driver.ebpf.leastPrivileged = true  # securityContext:  # capabilities:  # add:  # - BPF  # - SYS\_RESOURCE  # - PERFMON  # - SYS\_PTRACE  #  # -- Set securityContext for the Falco container.For more info see the "falco.securityContext" helper in "pod-template.tpl"  containerSecurityContext: {}  scc:  # -- Create OpenShift's Security Context Constraint.  create: true  resources:  # -- Although resources needed are subjective on the actual workload we provide  # a sane defaults ones. If you have more questions or concerns, please refer  # to #falco slack channel for more info about it.  requests:  cpu: 100m  memory: 512Mi  # -- Maximum amount of resources that Falco container could get.  # If you are enabling more than one source in falco, than consider to increase  # the cpu limits.  limits:  cpu: 1000m  memory: 1024Mi  # -- Selectors used to deploy Falco on a given node/nodes.  nodeSelector: {}  # -- Affinity constraint for pods' scheduling.  affinity: {}  # -- Tolerations to allow Falco to run on Kubernetes masters.  tolerations:  - effect: NoSchedule  key: node-role.kubernetes.io/master  - effect: NoSchedule  key: node-role.kubernetes.io/control-plane  # -- Parameters used  healthChecks:  livenessProbe:  # -- Tells the kubelet that it should wait X seconds before performing the first probe.  initialDelaySeconds: 60  # -- Number of seconds after which the probe times out.  timeoutSeconds: 5  # -- Specifies that the kubelet should perform the check every x seconds.  periodSeconds: 15  readinessProbe:  # -- Tells the kubelet that it should wait X seconds before performing the first probe.  initialDelaySeconds: 30  # -- Number of seconds after which the probe times out.  timeoutSeconds: 5  # -- Specifies that the kubelet should perform the check every x seconds.  periodSeconds: 15  # -- Attach the Falco process to a tty inside the container. Needed to flush Falco logs as soon as they are emitted.  # Set it to "true" when you need the Falco logs to be immediately displayed.  tty: false  #########################  # Scenario requirements #  #########################  # Sensors dislocation configuration (scenario requirement)  controller:  # Available options: deployment, daemonset.  kind: daemonset  # Annotations to add to the daemonset or deployment  annotations: {}  daemonset:  updateStrategy:  # You can also customize maxUnavailable or minReadySeconds if you  # need it  # -- Perform rolling updates by default in the DaemonSet agent  # ref: https://kubernetes.io/docs/tasks/manage-daemon/update-daemon-set/  type: RollingUpdate  deployment:  # -- Number of replicas when installing Falco using a deployment. Change it if you really know what you are doing.  # For more info check the section on Plugins in the README.md file.  replicas: 1  # -- Network services configuration (scenario requirement)  # Add here your services to be deployed together with Falco.  services:  # Example configuration for the "k8sauditlog" plugin  # - name: k8saudit-webhook  # type: NodePort  # ports:  # - port: 9765 # See plugin open\_params  # nodePort: 30007  # protocol: TCP  # File access configuration (scenario requirement)  mounts:  # -- A list of volumes you want to add to the Falco pods.  volumes: []  # -- A list of volumes you want to add to the Falco pods.  volumeMounts: []  # -- By default, `/proc` from the host is only mounted into the Falco pod when `driver.enabled` is set to `true`. This flag allows it to override this behaviour for edge cases where `/proc` is needed but syscall data source is not enabled at the same time (e.g. for specific plugins).  enforceProcMount: false  # Driver settings (scenario requirement)  driver:  # -- Set it to false if you want to deploy Falco without the drivers.  # Always set it to false when using Falco with plugins.  enabled: true  # -- Tell Falco which driver to use. Available options: module (kernel driver), ebpf (eBPF probe), modern-bpf (modern eBPF probe).  kind: module  # -- Configuration section for ebpf driver.  ebpf:  # -- Path where the eBPF probe is located. It comes handy when the probe have been installed in the nodes using tools other than the init  # container deployed with the chart.  path:  # -- Needed to enable eBPF JIT at runtime for performance reasons.  # Can be skipped if eBPF JIT is enabled from outside the container  hostNetwork: false  # -- Constrain Falco with capabilities instead of running a privileged container.  # This option is only supported with the eBPF driver and a kernel >= 5.8.  # Ensure the eBPF driver is enabled (i.e., setting the `driver.kind` option to `ebpf`).  leastPrivileged: false  # -- Configuration for the Falco init container.  loader:  # -- Enable/disable the init container.  enabled: true  initContainer:  image:  # -- The image pull policy.  pullPolicy: IfNotPresent  # -- The image registry to pull from.  registry: docker.io  # -- The image repository to pull from.  repository: falcosecurity/falco-driver-loader  # -- Overrides the image tag whose default is the chart appVersion.  tag: ""  # -- Extra environment variables that will be pass onto Falco driver loader init container.  env: []  # -- Arguments to pass to the Falco driver loader init container.  args: []  # -- Resources requests and limits for the Falco driver loader init container.  resources: {}  # -- Security context for the Falco driver loader init container. Overrides the default security context. If driver.kind == "module" you must at least set `privileged: true`.  securityContext: {}  # -- Gvisor configuration. Based on your system you need to set the appropriate values.  # Please, rembember to add pod tolerations and affinities in order to schedule the Falco pods in the gVisor enabled nodes.  gvisor:  # -- Set it to true if you want to deploy Falco with gVisor support.  enabled: false  # -- Runsc container runtime configuration. Falco needs to interact with it in order to intercept the activity of the sandboxed pods.  runsc:  # -- Absolute path of the `runsc` binary in the k8s nodes.  path: /home/containerd/usr/local/sbin  # -- Absolute path of the root directory of the `runsc` container runtime. It is of vital importance for Falco since `runsc` stores there the information of the workloads handled by it;  root: /run/containerd/runsc  # -- Absolute path of the `runsc` configuration file, used by Falco to set its configuration and make aware `gVisor` of its presence.  config: /run/containerd/runsc/config.toml  # Collectors for data enrichment (scenario requirement)  collectors:  # -- Enable/disable all the metadata collectors.  enabled: true  docker:  # -- Enable Docker support.  enabled: true  # -- The path of the Docker daemon socket.  socket: /var/run/docker.sock  containerd:  # -- Enable ContainerD support.  enabled: true  # -- The path of the ContainerD socket.  socket: /run/containerd/containerd.sock  crio:  # -- Enable CRI-O support.  enabled: true  # -- The path of the CRI-O socket.  socket: /run/crio/crio.sock  kubernetes:  # -- Enable Kubernetes meta data collection via a connection to the Kubernetes API server.  # When this option is disabled, Falco falls back to the container annotations to grap the meta data.  # In such a case, only the ID, name, namespace, labels of the pod will be available.  enabled: true  # -- The apiAuth value is to provide the authentication method Falco should use to connect to the Kubernetes API.  # The argument's documentation from Falco is provided here for reference:  #  # <bt\_file> | <cert\_file>:<key\_file[#password]>[:<ca\_cert\_file>], --k8s-api-cert <bt\_file> | <cert\_file>:<key\_file[#password]>[:<ca\_cert\_file>]  # Use the provided files names to authenticate user and (optionally) verify the K8S API server identity.  # Each entry must specify full (absolute, or relative to the current directory) path to the respective file.  # Private key password is optional (needed only if key is password protected).  # CA certificate is optional. For all files, only PEM file format is supported.  # Specifying CA certificate only is obsoleted - when single entry is provided  # for this option, it will be interpreted as the name of a file containing bearer token.  # Note that the format of this command-line option prohibits use of files whose names contain  # ':' or '#' characters in the file name.  # -- Provide the authentication method Falco should use to connect to the Kubernetes API.  apiAuth: /var/run/secrets/kubernetes.io/serviceaccount/token  ## -- Provide the URL Falco should use to connect to the Kubernetes API.  apiUrl: "https://$(KUBERNETES\_SERVICE\_HOST)"  # -- If true, only the current node (on which Falco is running) will be considered when requesting metadata of pods  # to the API server. Disabling this option may have a performance penalty on large clusters.  enableNodeFilter: true  ###########################  # Extras and customization #  ############################  extra:  # -- Extra environment variables that will be pass onto Falco containers.  env: []  # -- Extra command-line arguments.  args: []  # -- Additional initContainers for Falco pods.  initContainers: []  # -- certificates used by webserver and grpc server.  # paste certificate content or use helm with --set-file  # or use existing secret containing key, crt, ca as well as pem bundle  certs:  # -- Existing secret containing the following key, crt and ca as well as the bundle pem.  existingSecret: ""  server:  # -- Key used by gRPC and webserver.  key: ""  # -- Certificate used by gRPC and webserver.  crt: ""  ca:  # -- CA certificate used by gRPC, webserver and AuditSink validation.  crt: ""  # -- Third party rules enabled for Falco. More info on the dedicated section in README.md file.  customRules:  {}  # Although Falco comes with a nice default rule set for detecting weird  # behavior in containers, our users are going to customize the run-time  # security rule sets or policies for the specific container images and  # applications they run. This feature can be handled in this section.  #  # Example:  #  # rules-traefik.yaml: |-  # [ rule body ]  ########################  # Falco integrations #  ########################  # -- For configuration values, see https://github.com/falcosecurity/charts/blob/master/falcosidekick/values.yaml  falcosidekick:  # -- Enable falcosidekick deployment.  enabled: false  # -- Enable usage of full FQDN of falcosidekick service (useful when a Proxy is used).  fullfqdn: false  # -- Listen port. Default value: 2801  listenPort: ""  ####################  # falcoctl config #  ####################  falcoctl:  image:  # -- The image pull policy.  pullPolicy: IfNotPresent  # -- The image registry to pull from.  registry: docker.io  # -- The image repository to pull from.  repository: falcosecurity/falcoctl  # -- Overrides the image tag whose default is the chart appVersion.  tag: "0.4.0"  artifact:  # -- Runs "falcoctl artifact install" command as an init container. It is used to install artfacts before  # Falco starts. It provides them to Falco by using an emptyDir volume.  install:  enabled: true  # -- Extra environment variables that will be pass onto falcoctl-artifact-install init container.  env: {}  # -- Arguments to pass to the falcoctl-artifact-install init container.  args: ["--verbose"]  # -- Resources requests and limits for the falcoctl-artifact-install init container.  resources: {}  # -- Security context for the falcoctl init container.  securityContext: {}  # -- Runs "falcoctl artifact follow" command as a sidecar container. It is used to automatically check for  # updates given a list of artifacts. If an update is found it downloads and installs it in a shared folder (emptyDir)  # that is accessible by Falco. Rulesfiles are automatically detected and loaded by Falco once they are installed in the  # correct folder by falcoctl. To prevent new versions of artifacts from breaking Falco, the tool checks if it is compatible  # with the running version of Falco before installing it.  follow:  enabled: true  # -- Extra environment variables that will be pass onto falcoctl-artifact-follow sidecar container.  env: {}  # -- Arguments to pass to the falcoctl-artifact-follow sidecar container.  args: ["--verbose"]  # -- Resources requests and limits for the falcoctl-artifact-follow sidecar container.  resources: {}  # -- Security context for the falcoctl-artifact-follow sidecar container.  securityContext: {}  # -- Configuration file of the falcoctl tool. It is saved in a configmap and mounted on the falcotl containers.  config:  # -- List of indexes that falcoctl downloads and uses to locate and download artiafcts. For more info see:  # https://github.com/falcosecurity/falcoctl/blob/main/proposals/20220916-rules-and-plugin-distribution.md#index-file-overview  indexes:  - name: falcosecurity  url: https://falcosecurity.github.io/falcoctl/index.yaml  # -- Configuration used by the artifact commands.  artifact:  # -- List of artifact types that falcoctl will handle. If the configured refs resolves to an artifact whose type is not contained  # in the list it will refuse to downloade and install that artifact.  allowedTypes:  - rulesfile  install:  # -- Do not resolve the depenencies for artifacts. By default is true, but for our use case we disable it.  resolveDeps: false  # -- List of artifacts to be installed by the falcoctl init container.  refs: [falco-rules:0]  # -- Directory where the rulesfiles are saved. The path is relative to the container, which in this case is an emptyDir  # mounted also by the Falco pod.  rulesfilesDir: /rulesfiles  # -- Same as the one above but for the artifacts.  pluginsDir: /plugins  follow:  # -- List of artifacts to be followed by the falcoctl sidecar container.  refs: [falco-rules:0]  # -- How often the tool checks for new versions of the followed artifacts.  every: 6h  # -- HTTP endpoint that serves the api versions of the Falco instance. It is used to check if the new versions are compatible  # with the running Falco instance.  falcoversions: http://localhost:8765/versions  # -- See the fields of the artifact.install section.  rulesfilesDir: /rulesfiles  # -- See the fields of the artifact.install section.  pluginsDir: /plugins  ######################  # falco.yaml config #  ######################  falco:  # File(s) or Directories containing Falco rules, loaded at startup.  # The name "rules\_file" is only for backwards compatibility.  # If the entry is a file, it will be read directly. If the entry is a directory,  # every file in that directory will be read, in alphabetical order.  #  # falco\_rules.yaml ships with the falco package and is overridden with  # every new software version. falco\_rules.local.yaml is only created  # if it doesn't exist. If you want to customize the set of rules, add  # your customizations to falco\_rules.local.yaml.  #  # The files will be read in the order presented here, so make sure if  # you have overrides they appear in later files.  # -- The location of the rules files that will be consumed by Falco.  rules\_file:  - /etc/falco/falco\_rules.yaml  - /etc/falco/falco\_rules.local.yaml  - /etc/falco/rules.d  #  # Plugins that are available for use. These plugins are not loaded by  # default, as they require explicit configuration to point to  # cloudtrail log files.  #  # To learn more about the supported formats for  # init\_config/open\_params for the cloudtrail plugin, see the README at  # https://github.com/falcosecurity/plugins/blob/master/plugins/cloudtrail/README.md.  # -- Plugins configuration. Add here all plugins and their configuration. Please  # consult the plugins documentation for more info. Remember to add the plugins name in  # "load\_plugins: []" in order to load them in Falco.  plugins:  - name: k8saudit  library\_path: libk8saudit.so  init\_config:  # maxEventSize: 262144  # webhookMaxBatchSize: 12582912  # sslCertificate: /etc/falco/falco.pem  open\_params: "http://:9765/k8s-audit"  - name: cloudtrail  library\_path: libcloudtrail.so  # see docs for init\_config and open\_params:  # https://github.com/falcosecurity/plugins/blob/master/plugins/cloudtrail/README.md  - name: json  library\_path: libjson.so  init\_config: ""  # Setting this list to empty ensures that the above plugins are \*not\*  # loaded and enabled by default. If you want to use the above plugins,  # set a meaningful init\_config/open\_params for the cloudtrail plugin  # and then change this to:  # load\_plugins: [cloudtrail, json]  # -- Add here the names of the plugins that you want to be loaded by Falco. Please make sure that  # plugins have been configured under the "plugins" section before adding them here.  # Please make sure to configure the falcoctl tool to download and install the very same plugins  # you are loading here. You should add the references in the falcoctl.config.artifact.install.refs array  # for each plugin you are loading.  load\_plugins: []  # -- Watch config file and rules files for modification.  # When a file is modified, Falco will propagate new config,  # by reloading itself.  watch\_config\_files: true  # -- If true, the times displayed in log messages and output messages  # will be in ISO 8601. By default, times are displayed in the local  # time zone, as governed by /etc/localtime.  time\_format\_iso\_8601: false  # -- If "true", print falco alert messages and rules file  # loading/validation results as json, which allows for easier  # consumption by downstream programs. Default is "false".  json\_output: false  # -- When using json output, whether or not to include the "output" property  # itself (e.g. "File below a known binary directory opened for writing  # (user=root ....") in the json output.  json\_include\_output\_property: true  # -- When using json output, whether or not to include the "tags" property  # itself in the json output. If set to true, outputs caused by rules  # with no tags will have a "tags" field set to an empty array. If set to  # false, the "tags" field will not be included in the json output at all.  json\_include\_tags\_property: true  # -- Send information logs to stderr. Note these are \*not\* security  # notification logs! These are just Falco lifecycle (and possibly error) logs.  log\_stderr: true  # -- Send information logs to syslog. Note these are \*not\* security  # notification logs! These are just Falco lifecycle (and possibly error) logs.  log\_syslog: true  # -- Minimum log level to include in logs. Note: these levels are  # separate from the priority field of rules. This refers only to the  # log level of falco's internal logging. Can be one of "emergency",  # "alert", "critical", "error", "warning", "notice", "info", "debug".  log\_level: info  # Falco is capable of managing the logs coming from libs. If enabled,  # the libs logger send its log records the same outputs supported by  # Falco (stderr and syslog). Disabled by default.  libs\_logger:  # -- Enable the libs logger.  enabled: false  # -- Minimum log severity to include in the libs logs. Note: this value is  # separate from the log level of the Falco logger and does not affect it.  # Can be one of "fatal", "critical", "error", "warning", "notice",  # "info", "debug", "trace".  severity: debug  # -- Minimum rule priority level to load and run. All rules having a  # priority more severe than this level will be loaded/run. Can be one  # of "emergency", "alert", "critical", "error", "warning", "notice",  # "informational", "debug".  priority: debug  # -- Whether or not output to any of the output channels below is  # buffered. Defaults to false  buffered\_outputs: false  # Falco uses a shared buffer between the kernel and userspace to pass  # system call information. When Falco detects that this buffer is  # full and system calls have been dropped, it can take one or more of  # the following actions:  # - ignore: do nothing (default when list of actions is empty)  # - log: log a DEBUG message noting that the buffer was full  # - alert: emit a Falco alert noting that the buffer was full  # - exit: exit Falco with a non-zero rc  #  # Notice it is not possible to ignore and log/alert messages at the same time.  #  # The rate at which log/alert messages are emitted is governed by a  # token bucket. The rate corresponds to one message every 30 seconds  # with a burst of one message (by default).  #  # The messages are emitted when the percentage of dropped system calls  # with respect the number of events in the last second  # is greater than the given threshold (a double in the range [0, 1]).  #  # For debugging/testing it is possible to simulate the drops using  # the `simulate\_drops: true`. In this case the threshold does not apply.  syscall\_event\_drops:  # -- The messages are emitted when the percentage of dropped system calls  # with respect the number of events in the last second  # is greater than the given threshold (a double in the range [0, 1]).  threshold: .1  # -- Actions to be taken when system calls were dropped from the circular buffer.  actions:  - log  - alert  # -- Rate at which log/alert messages are emitted.  rate: .03333  # -- Max burst of messages emitted.  max\_burst: 1  # -- Flag to enable drops for debug purposes.  simulate\_drops: false  # Falco uses a shared buffer between the kernel and userspace to receive  # the events (eg., system call information) in userspace.  #  # Anyways, the underlying libraries can also timeout for various reasons.  # For example, there could have been issues while reading an event.  # Or the particular event needs to be skipped.  # Normally, it's very unlikely that Falco does not receive events consecutively.  #  # Falco is able to detect such uncommon situation.  #  # Here you can configure the maximum number of consecutive timeouts without an event  # after which you want Falco to alert.  # By default this value is set to 1000 consecutive timeouts without an event at all.  # How this value maps to a time interval depends on the CPU frequency.  syscall\_event\_timeouts:  # -- Maximum number of consecutive timeouts without an event  # after which you want Falco to alert.  max\_consecutives: 1000  # --- [Description]  #  # This is an index that controls the dimension of the syscall buffers.  # The syscall buffer is the shared space between Falco and its drivers where all the syscall events  # are stored.  # Falco uses a syscall buffer for every online CPU, and all these buffers share the same dimension.  # So this parameter allows you to control the size of all the buffers!  #  # --- [Usage]  #  # You can choose between different indexes: from `1` to `10` (`0` is reserved for future uses).  # Every index corresponds to a dimension in bytes:  #  # [(\*), 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB, 512 MB]  # ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^  # | | | | | | | | | | |  # 0 1 2 3 4 5 6 7 8 9 10  #  # As you can see the `0` index is reserved, while the index `1` corresponds to  # `1 MB` and so on.  #  # These dimensions in bytes derive from the fact that the buffer size must be:  # (1) a power of 2.  # (2) a multiple of your system\_page\_dimension.  # (3) greater than `2 \* (system\_page\_dimension)`.  #  # According to these constraints is possible that sometimes you cannot use all the indexes, let's consider an  # example to better understand it:  # If you have a `page\_size` of 1 MB the first available buffer size is 4 MB because 2 MB is exactly  # `2 \* (system\_page\_size)` -> `2 \* 1 MB`, but this is not enough we need more than `2 \* (system\_page\_size)`!  # So from this example is clear that if you have a page size of 1 MB the first index that you can use is `3`.  #  # Please note: this is a very extreme case just to let you understand the mechanism, usually the page size is something  # like 4 KB so you have no problem at all and you can use all the indexes (from `1` to `10`).  #  # To check your system page size use the Falco `--page-size` command line option. The output on a system with a page  # size of 4096 Bytes (4 KB) should be the following:  #  # "Your system page size is: 4096 bytes."  #  # --- [Suggestions]  #  # Before the introduction of this param the buffer size was fixed to 8 MB (so index `4`, as you can see  # in the default value below).  # You can increase the buffer size when you face syscall drops. A size of 16 MB (so index `5`) can reduce  # syscall drops in production-heavy systems without noticeable impact. Very large buffers however could  # slow down the entire machine.  # On the other side you can try to reduce the buffer size to speed up the system, but this could  # increase the number of syscall drops!  # As a final remark consider that the buffer size is mapped twice in the process' virtual memory so a buffer of 8 MB  # will result in a 16 MB area in the process virtual memory.  # Please pay attention when you use this parameter and change it only if the default size doesn't fit your use case.  # -- This is an index that controls the dimension of the syscall buffers.  syscall\_buf\_size\_preset: 4  ############## [EXPERIMENTAL] Modern BPF probe specific ##############  # Please note: these configs regard only the modern BPF probe. They  # are experimental so they could change over releases.  #  # `cpus\_for\_each\_syscall\_buffer`  #  # --- [Description]  #  # This is an index that controls how many CPUs you want to assign to a single  # syscall buffer (ring buffer). By default, every syscall buffer is associated to  # 2 CPUs, so the mapping is 1:2. The modern BPF probe allows you to choose different  # mappings, for example, 1:1 would mean a syscall buffer for each CPU.  #  # --- [Usage]  #  # You can choose between different indexes: from `0` to `MAX\_NUMBER\_ONLINE\_CPUs`.  # `0` is a special value and it means a single syscall buffer shared between all  # your online CPUs. `0` has the same effect as `MAX\_NUMBER\_ONLINE\_CPUs`, the rationale  # is that `0` allows you to create a single buffer without knowing the number of online  # CPUs on your system.  # Let's consider an example to better understand it:  #  # Consider a system with 7 online CPUs:  #  # CPUs 0 X 2 3 X X 6 7 8 9 (X means offline CPU)  #  # - `1` means a syscall buffer for each CPU so 7 buffers  #  # CPUs 0 X 2 3 X X 6 7 8 9 (X means offline CPU)  # | | | | | | |  # BUFFERs 0 1 2 3 4 5 6  #  # - `2` (Default value) means a syscall buffer for each CPU pair, so 4 buffers  #  # CPUs 0 X 2 3 X X 6 7 8 9 (X means offline CPU)  # | | | | | | |  # BUFFERs 0 0 1 1 2 2 3  #  # Please note that we need 4 buffers, 3 buffers are associated with CPU pairs, the last  # one is mapped with just 1 CPU since we have an odd number of CPUs.  #  # - `0` or `MAX\_NUMBER\_ONLINE\_CPUs` mean a syscall buffer shared between all CPUs, so 1 buffer  #  # CPUs 0 X 2 3 X X 6 7 8 9 (X means offline CPU)  # | | | | | | |  # BUFFERs 0 0 0 0 0 0 0  #  # Moreover you can combine this param with `syscall\_buf\_size\_preset`  # index, for example, you could create a huge single syscall buffer  # shared between all your online CPUs of 512 MB (so `syscall\_buf\_size\_preset=10`).  #  # --- [Suggestions]  #  # We chose index `2` (so one syscall buffer for each CPU pair) as default because the modern bpf probe  # follows a different memory allocation strategy with respect to the other 2 drivers (bpf and kernel module).  # By the way, you are free to find the preferred configuration for your system.  # Considering a fixed `syscall\_buf\_size\_preset` and so a fixed buffer dimension:  # - a lower number of buffers can speed up your system (lower memory footprint)  # - a too lower number of buffers could increase contention in the kernel causing an  # overall slowdown of the system.  # If you don't have huge events throughputs and you are not experimenting with tons of drops  # you can try to reduce the number of buffers to have a lower memory footprint  modern\_bpf:  # -- [MODERN PROBE ONLY] This is an index that controls how many CPUs you want to assign to a single syscall buffer.  cpus\_for\_each\_syscall\_buffer: 2  ############## [EXPERIMENTAL] Modern BPF probe specific ##############  # Falco continuously monitors outputs performance. When an output channel does not allow  # to deliver an alert within a given deadline, an error is reported indicating  # which output is blocking notifications.  # The timeout error will be reported to the log according to the above log\_\* settings.  # Note that the notification will not be discarded from the output queue; thus,  # output channels may indefinitely remain blocked.  # An output timeout error indeed indicate a misconfiguration issue or I/O problems  # that cannot be recovered by Falco and should be fixed by the user.  #  # The "output\_timeout" value specifies the duration in milliseconds to wait before  # considering the deadline exceed.  #  # With a 2000ms default, the notification consumer can block the Falco output  # for up to 2 seconds without reaching the timeout.  # -- Duration in milliseconds to wait before considering the output timeout deadline exceed.  output\_timeout: 2000  # A throttling mechanism implemented as a token bucket limits the  # rate of Falco notifications. One rate limiter is assigned to each event  # source, so that alerts coming from one can't influence the throttling  # mechanism of the others. This is controlled by the following options:  # - rate: the number of tokens (i.e. right to send a notification)  # gained per second. When 0, the throttling mechanism is disabled.  # Defaults to 0.  # - max\_burst: the maximum number of tokens outstanding. Defaults to 1000.  #  # With these defaults, the throttling mechanism is disabled.  # For example, by setting rate to 1 Falco could send up to 1000 notifications  # after an initial quiet period, and then up to 1 notification per second  # afterward. It would gain the full burst back after 1000 seconds of  # no activity.  outputs:  # -- Number of tokens gained per second.  rate: 1  # -- Maximum number of tokens outstanding.  max\_burst: 1000  # Where security notifications should go.  # Multiple outputs can be enabled.  syslog\_output:  # -- Enable syslog output for security notifications.  enabled: true  # If keep\_alive is set to true, the file will be opened once and  # continuously written to, with each output message on its own  # line. If keep\_alive is set to false, the file will be re-opened  # for each output message.  #  # Also, the file will be closed and reopened if falco is signaled with  # SIGUSR1.  file\_output:  # -- Enable file output for security notifications.  enabled: false  # -- Open file once or every time a new notification arrives.  keep\_alive: false  # -- The filename for logging notifications.  filename: ./events.txt  stdout\_output:  # -- Enable stdout output for security notifications.  enabled: true  # Falco contains an embedded webserver that exposes a healthy endpoint that can be used to check if Falco is up and running.  # By default the endpoint is /healthz  #  # The ssl\_certificate is a combination SSL Certificate and corresponding  # key contained in a single file. You can generate a key/cert as follows:  #  # $ openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 365 -out certificate.pem  # $ cat certificate.pem key.pem > falco.pem  # $ sudo cp falco.pem /etc/falco/falco.pem  webserver:  # -- Enable Falco embedded webserver.  enabled: true  # -- Number of threads depending on the number of online cores.  threadiness: 0  # -- Port where Falco embedded webserver listen to connections.  listen\_port: 8765  # -- Endpoint where Falco exposes the health status.  k8s\_healthz\_endpoint: /healthz  # -- Enable SSL on Falco embedded webserver.  ssl\_enabled: false  # -- Certificate bundle path for the Falco embedded webserver.  ssl\_certificate: /etc/falco/falco.pem  # Possible additional things you might want to do with program output:  # - send to a slack webhook:  # program: "jq '{text: .output}' | curl -d @- -X POST https://hooks.slack.com/services/XXX"  # - logging (alternate method than syslog):  # program: logger -t falco-test  # - send over a network connection:  # program: nc host.example.com 80  # If keep\_alive is set to true, the program will be started once and  # continuously written to, with each output message on its own  # line. If keep\_alive is set to false, the program will be re-spawned  # for each output message.  #  # Also, the program will be closed and reopened if falco is signaled with  # SIGUSR1.  program\_output:  # -- Enable program output for security notifications.  enabled: false  # -- Start the program once or re-spawn when a notification arrives.  keep\_alive: false  # -- Command to execute for program output.  program: "jq '{text: .output}' | curl -d @- -X POST https://hooks.slack.com/services/XXX"  http\_output:  # -- Enable http output for security notifications.  enabled: false  # -- When set, this will override an auto-generated URL which matches the falcosidekick Service.  # -- When including Falco inside a parent helm chart, you must set this since the auto-generated URL won't match (#280).  url: ""  user\_agent: "falcosecurity/falco"  # Falco supports running a gRPC server with two main binding types  # 1. Over the network with mandatory mutual TLS authentication (mTLS)  # 2. Over a local unix socket with no authentication  # By default, the gRPC server is disabled, with no enabled services (see grpc\_output)  # please comment/uncomment and change accordingly the options below to configure it.  # Important note: if Falco has any troubles creating the gRPC server  # this information will be logged, however the main Falco daemon will not be stopped.  # gRPC server over network with (mandatory) mutual TLS configuration.  # This gRPC server is secure by default so you need to generate certificates and update their paths here.  # By default the gRPC server is off.  # You can configure the address to bind and expose it.  # By modifying the threadiness configuration you can fine-tune the number of threads (and context) it will use.  # grpc:  # enabled: true  # bind\_address: "0.0.0.0:5060"  # # when threadiness is 0, Falco sets it by automatically figuring out the number of online cores  # threadiness: 0  # private\_key: "/etc/falco/certs/server.key"  # cert\_chain: "/etc/falco/certs/server.crt"  # root\_certs: "/etc/falco/certs/ca.crt"  # -- gRPC server using an unix socket  grpc:  # -- Enable the Falco gRPC server.  enabled: false  # -- Bind address for the grpc server.  bind\_address: "unix:///run/falco/falco.sock"  # -- Number of threads (and context) the gRPC server will use, 0 by default, which means "auto".  threadiness: 0  # gRPC output service.  # By default it is off.  # By enabling this all the output events will be kept in memory until you read them with a gRPC client.  # Make sure to have a consumer for them or leave this disabled.  grpc\_output:  # -- Enable the gRPC output and events will be kept in memory until you read them with a gRPC client.  enabled: false  # Container orchestrator metadata fetching params  metadata\_download:  # -- Max allowed response size (in Mb) when fetching metadata from Kubernetes.  max\_mb: 100  # -- Sleep time (in μs) for each download chunck when fetching metadata from Kubernetes.  chunk\_wait\_us: 1000  # -- Watch frequency (in seconds) when fetching metadata from Kubernetes.  watch\_freq\_sec: 1 |

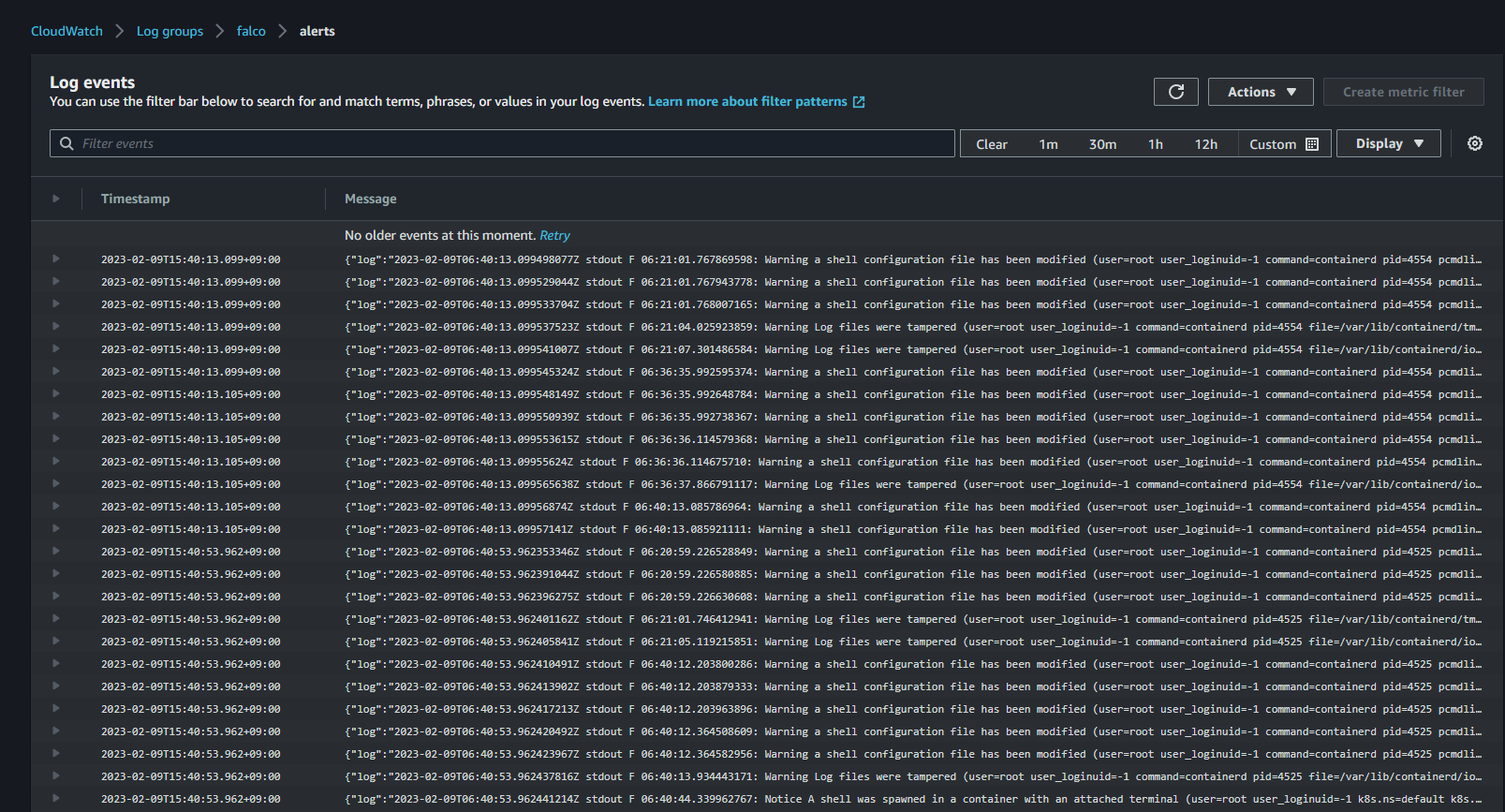
Helm Chart를 사용해 falco를 설치해주자.

|  |
| --- |
| $ helm repo add falcosecurity https://falcosecurity.github.io/charts  $ helm install falco -f values.yaml falcosecurity/falco --namespace falco --create-namespace |

우선 falco에서 발생하는 Log들을 CloudWatch에 저장시키자.

|  |
| --- |
| $ kubectl create ns fluentbit  $ vim configmap.yaml  apiVersion: v1  kind: ConfigMap  metadata:  name: fluent-bit-config  namespace: fluentbit  labels:  app.kubernetes.io/name: fluentbit  data:  fluent-bit.conf: |  [SERVICE]  Parsers\_File parsers.conf  [INPUT]  Name tail  Tag falco.\*  Path /var/log/containers/falco\*.log  Parser falco  DB /var/log/flb\_falco.db  Mem\_Buf\_Limit 5MB  Skip\_Long\_Lines On  Refresh\_Interval 10  [OUTPUT]  Name cloudwatch  Match falco.\*\*  region ap-northeast-2  log\_group\_name falco  log\_stream\_name alerts  auto\_create\_group true  parsers.conf: |  [PARSER]  Name falco  Format json  Time\_Key time  Time\_Format %Y-%m-%dT%H:%M:%S.%L  Time\_Keep Off  # Command | Decoder | Field | Optional Action  # =============|==================|=================  Decode\_Field\_As json log  $ vim configmap.yaml  apiVersion: apps/v1  kind: DaemonSet  metadata:  name: fluentbit  namespace: fluentbit  labels:  app.kubernetes.io/name: fluentbit  spec:  selector:  matchLabels:  name: fluentbit  template:  metadata:  labels:  name: fluentbit  spec:  serviceAccountName: fluent-bit  containers:  - name: aws-for-fluent-bit  image: amazon/aws-for-fluent-bit:1.2.2  volumeMounts:  - name: varlog  mountPath: /var/log  - name: varlibdockercontainers  mountPath: /var/lib/docker/containers  readOnly: true  - name: fluent-bit-config  mountPath: /fluent-bit/etc/  - name: mnt  mountPath: /mnt  readOnly: true  resources:  limits:  memory: 500Mi  requests:  cpu: 500m  memory: 100Mi  volumes:  - name: varlog  hostPath:  path: /var/log  - name: varlibdockercontainers  hostPath:  path: /var/lib/docker/containers  - name: fluent-bit-config  configMap:  name: fluent-bit-config  - name: mnt  hostPath:  path: /mnt  $ vim service-account.yaml  apiVersion: v1  kind: ServiceAccount  metadata:  name: fluent-bit  namespace: fluentbit  ---  apiVersion: rbac.authorization.k8s.io/v1  kind: ClusterRole  metadata:  name: pod-log-reader  namespace: fluentbit  rules:  - apiGroups: [""]  resources:  - namespaces  - pods  verbs: ["get", "list", "watch"]  ---  apiVersion: rbac.authorization.k8s.io/v1  kind: ClusterRoleBinding  metadata:  name: pod-log-crb  namespace: fluentbit  roleRef:  apiGroup: rbac.authorization.k8s.io  kind: ClusterRole  name: pod-log-reader  subjects:  - kind: ServiceAccount  name: fluent-bit  namespace: default  $ kubectl apply -f . |

그러면 CloudWatch에 아래와 같이 Logs가 올라오는 것을 볼 수 있다.



그리고 만약 nginx 컨테이너 안에서 who와 whoami와 같은 명령어가 입력되면 alert되게 custom alert 규칙을 작성해주자.

|  |
| --- |
| $ vim custom\_alert.yaml  customRules:  rules-nginx.yaml: |  - macro: nginx\_consider\_syscalls  condition: (evt.num < 0)  - macro: app\_nginx  condition: container and container.image contains "nginx"  - rule: The program "whoami" is run in a container  desc: An event will trigger every time you run "whoami" in a container  condition: evt.type = execve and evt.dir=< and container.id != host and proc.name = whoami  output: "whoami command run in container (user=%user.name %container.info parent=%proc.pname cmdline=%proc.cmdline)"  priority: NOTICE  warn\_evttypes: False  - rule: The program "locate" is run in a container  desc: An event will trigger every time you run "locate" in a container  condition: evt.type = execve and evt.dir=< and container.id != host and proc.name = locate  output: "locate command run in container (user=%user.name %container.info parent=%proc.pname cmdline=%proc.cmdline)"  priority: NOTICE  warn\_evttypes: False  $ helm upgrade falco -f custom\_alert.yaml falcosecurity/falco -n falco |

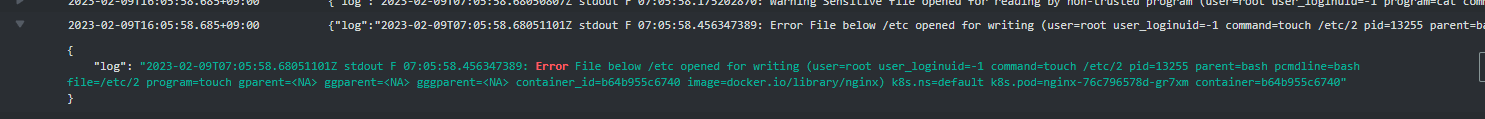
그리고 이제 동작 확인해보자.

|  |
| --- |
| $ cat <<EOF | kubectl apply -f -  apiVersion: apps/v1  kind: Deployment  metadata:  name: nginx2  labels:  app: nginx2  spec:  replicas: 3  selector:  matchLabels:  app: nginx2  template:  metadata:  labels:  app: nginx2  spec:  affinity:  nodeAffinity:  requiredDuringSchedulingIgnoredDuringExecution:  nodeSelectorTerms:  - matchExpressions:  - key: beta.kubernetes.io/arch  operator: In  values:  - amd64  - arm64  containers:  - name: nginx  image: nginx:1.19.2  ports:  - containerPort: 80  EOF |

그리고 아무 pod에 접근해준다.

|  |
| --- |
| $ kubectl exec -it 'nginx2-76c796578d-9fkjx' -- /bin/bash |

|  |
| --- |
| # touch /etc/2  # cat /etc/shadow > /dev/null 2>&1 |



|  |
| --- |
| # who  # whoami |

