1. Main
   1. Packages et utilité :
      1. Keptn from "github.com/keptn/go-utils/pkg/lib/keptn"

On utilise KeptnOpts qui est un struct qui contient les options keptn  et aui se trouve dans le fichier <https://github.com/keptn/go-utils/blob/master/pkg/lib/keptn/keptn_base.go> :

type KeptnOpts struct {

UseLocalFileSystem bool

ConfigurationServiceURL string

EventBrokerURL string // Deprecated: use EventSender instead

DatastoreURL string

IncomingEvent \*cloudevents.Event

LoggingOptions \*LoggingOpts

EventSender EventSender

}

* + 1. Cloudevents from "github.com/cloudevents/sdk-go/v2"

- On utilise une variable event du type cloudEvents.Event. Event étant un alias pour la structure event.Event se trouvant dans <https://github.com/cloudevents/sdk-go/blob/main/v2/event/event.go>

// Event represents the canonical representation of a CloudEvent.

type Event struct {

Context EventContext

DataEncoded []byte

// DataBase64 indicates if the event, when serialized, represents

// the data field using the base64 encoding.

// In v0.3, this field is superseded by DataContentEncoding

DataBase64 bool

FieldErrors map[string]error

}

- On utilise ensuite event.DataAs(data interface{}) qui va remplir l’objet data avec ce que contient l event cest à dire avec event.DataEncoded. DataAs est defini dans <https://github.com/cloudevents/sdk-go/blob/main/v2/event/event_data.go>

// DataAs attempts to populate the provided data object with the event payload.

// obj should be a pointer type.

func (e Event) DataAs(obj interface{}) error {

data := e.Data()

if len(data) == 0 {

// No data.

return nil

}

if e.SpecVersion() != CloudEventsVersionV1 {

var err error

if data, err = e.legacyConvertData(data); err != nil {

return err

}

}

return datacodec.Decode(context.Background(), e.DataMediaType(), data, obj)

}

- Les CloudEvent keptn décrits sur <https://github.com/keptn/spec/blob/master/cloudevents.md>

Ce sont les **sh.keptn.event** qui contiennent les propriétés **data**, id, shkeptncontext, source, specversion, time, triggeredid, gitcommitid et **type**

Dans Type on a soit le statut de l’exécution d’une séquence de tâches, soit le statut de lexecution dune certaine tâche dans une séquence.

Dans Data on a le payload de levent avec les propriétés labels, message, project, service, stage, status (le splunk-service a-t-il été capable ou non de faire le travail sans erreurs : succeeded, errored ou unknown), result (pass, warning ou fail) et [NomTache]

* + 1. Keptnv2 from "github.com/keptn/go-utils/pkg/lib/v0\_2\_0" :

On utilise la function NewKeptn(incomingEvent \*cloudevents.Event, opts keptn.KeptnOpts) (\*Keptn, error) définit dans <https://github.com/keptn/go-utils/blob/master/pkg/lib/v0_2_0/keptn.go>

Dans cette function, on récupère le shkeptncontext de l’event. On crée une variable keptnBase de type EventData (structure contenant les champs Project, Service, Stage , Labels, Status, Result, Message et défini dans <https://github.com/keptn/go-utils/blob/master/pkg/lib/v0_2_0/events.go#L311>) . Les données de incommingEvent sont stockées dans la variable keptnBase qui rappelons le est de type EventData. Il y a aussi un type KeptnBase (defini dans <https://github.com/keptn/go-utils/blob/master/pkg/lib/keptn/keptn_base.go#L38> et contenant principalement KeptnContext (string), Event (EventProperties), CloudEvent(cloudevents.Event) ).

Ensuite une nouvelle instance de la structure Keptn (structure contenant un attribut de type KeptnBase) appelée k est créée. Elle contient un KeptnBase avec les attributs Event(contenu du EventData keptnBase), CloudEvent (incomingEvent), KeptnContext (shkeptncontext), UseLocalFileSystem (opts.UseLocalFileSystem), RessourceHandler(nil).

Ensuite on initialise la variable csUrl :=keptn.ConfigurationServiceUrl = « resource-service :8080 » (qui est le repository (dans le cluster keptn) créé par keptn dans lequel il stocke les ressources téléchargées depuis le repository git keptn). Si une valeur de ConfigurationServiceUrl est déjà présente dans opts, on remplace csUrl par la valeur contenue dans opts.

Ensuite un EventSender permettant d’envoyer des event cloud est créé soit à partir de l’eventBrokerUrl donné dans opts, soit en utilisant l’eventSender donné dans opts, soit en créant un eventSender à partir du DefaultHTTPEventEndpoint= http://localhost:8081/event ». L’eventSender utilisé est stocké dans k.KeptnBase.EventSender

Ensuite on initialise datastoreURL à la valeur par défaut mongodb-datastore:8080. On écrase cette valeur par celle contenue dans opts si elle existe. On stocke dans k.ResourceHandler et k.EventHandler des ResourceHandler et EventHandler créés respectivement à partir du csURL et du datastoreURL

// NewEventHandler returns a new EventHandler

type EventHandler struct {

eventHandler \*v2.EventHandler

BaseURL string

AuthToken string

AuthHeader string

HTTPClient \*http.Client

Scheme string

}

// ResourceHandler handles resources

type ResourceHandler struct {

resourceHandler \*v2.ResourceHandler

BaseURL string

AuthToken string

AuthHeader string

HTTPClient \*http.Client

Scheme string

}

La méthode renvoie finalement k qui est notre structure keptn complétée.

* 1. Logique de fonctionnement :
     + 1. On initialise la variable globale keptnOptions avec un objet de type KeptnOpts
       2. On définit une structure appelée envConfig :

type envConfig struct {

// Port on which to listen for cloudevents

Port int `envconfig:"RCV\_PORT" default:"8080"`

// Path to which cloudevents are sent

Path string `envconfig:"RCV\_PATH" default:"/"`

// Whether we are running locally (e.g., for testing) or on production

Env string `envconfig:"ENV" default:"local"`

// URL of the Keptn configuration service (this is where we can fetch files from the config repo)

ConfigurationServiceUrl string `envconfig:"CONFIGURATION\_SERVICE" default:""`

(et les splunk credentials aussi …)

}

* + - 1. On définit la constante ServiceName= « splunk-service » pour spécifier le service á l’origine des cloudevent que l’on enverra. Cette constante sera utilisée dans le fichier eventhandlers.go
      2. On définit la fonction parseKeptnCloudEventPayload(event cloudevents.Event, data interface{}) qui va appeler event.DataAs(data) afin dextraire les données encodés dans levent et de les stocker dans le paramètre data.

func parseKeptnCloudEventPayload(event cloudevents.Event, data interface{}) error {

err := event.DataAs(data)

if err != nil {

logger.Errorf("Got Data Error: %s", err.Error())

return err

}

return nil

}

* + - 1. On définit la fonction processKeptnCloudEvent(ctx context.Context, event cloudevents.Event)
         * On crée une nouvelle instance de keptn. Pour cela on utiliste la fonction NewKeptn retournant notre structure keptn mentionnée dans la partie iii. La fonction prend en paramètre l’event ainsi le keptnOptions préalablemet configuré au début de la fonction \_main et dont la structure est donnée dans la partie i.
         * Ensuite selon le type d’événement qui a été recu, on crée la variable eventData correspondant au type de l’événement. On appelle ensuite la fonction parseKeptnCloudEventPayload afin de de stocker les données de l’event dans la structure eventData.
         * On appelle ensuite la fonction eventHandler correspondant au type de notre événement (HandleGetSliTriggeredEvent ou HandleConfigureMonitoringTriggeredEvent) en lui passsant en paramètre l’instance keptn, l’event et l’eventData.
      2. On définit la fonction \_main(args []string, env envConfig) int :

Remarque : args n’a pas l’air d’être utile.

On vérifie si env.Env= local. Si c le cas on met keptnOptions.UseLocalFileSystem à true. Dans keptnOptions.ConfigurationServiceUrl on met env.ConfiguratuinServiceUrl. keptnOptions est une variable globale initialisée vide et dont la structure est définie dans la partie i.

Dans ctx on crée un nouveau context dérivé du background context et créé avec la fonction cloudevents.WithEncodingStructured() qui retourne un contexte forçant un encoding structuré.

Ensuite on configure le serveur http pour lequel on défini le chemin sur lequel on reçoit les événements et le port sur lequel on écoute les événements. Ces paramètres étaient définis dans env.

On crée un nouveau client à partir du serveur http créé

On appelle la méthode Start receiver qui va écouter sur le port et le chemin spécifié pour des events. A chaque event recu (sh.keptn.event.monitoring.configure,sh.keptn.event.configure-monitoring.triggered ou sh.keptn.event.get-sli.triggered comme défini dans le distibutor dans le deployment.yaml), la fonction processKepnCloudEvent est appelée avec le contexte ctx et avec l’event recu. La structure d’un event est décrit dans la partie ii. L’event ici poss`de aussi des champs supplémentaires propres aux cloud events keptn (voir 3ieme tiret partie ii).

* + - 1. On définit la fonction main() qui sera exécutée au lancement de notre programme :
         * On récupère d’abord les valeurs des variables d’environnement RCV\_PORT, RCV\_PATH, ENV et CONFIGURATION\_SERVICE, SP\_PORT, SP\_API\_TOKEN et SP\_HOST que l’on stocke dans la variable env qui est de type envConfig dont la structure est donnée dans la partie 2.
         * Ensuite on appelle la fonction \_main auquel on passe les arguments du main ainsi que la variable env.

1. EventHandlers :
   1. Packages et utilité :
   2. Logique de fonctionnement :
      1. On définit les constantes :

sliFile = "splunk/sli.yaml"

defaultSleepBeforeAPIInSeconds = 20

* + 1. On définit la variable globale

sleepBeforeAPIInSeconds

* + 1. On initialise sleepBeforeAPIInSeconds dans la fonction init() qui est appelée implicitement lors de l’exécution du main
    2. On défini le type splunkCredentials qui contient le Host, le Token et le port de Splunk
    3. HandleGetSliTriggeredEvent(ddKeptn \*keptnv2.Keptn, incomingEvent cloudevents.Event, data \*keptnv2.GetSLITriggeredEventData) error :
       - * On récupère le context de keptn dans la variable shkeptncontext. Pour cela on récupère l’extension shkeptncontext du contexte de l’incoming event
         * On configure le logger avec ce contexte
         * Si le sli provider de l’eventData ne correspond pas à Splunk, on ignore le get-sli event. (**CETTE PARTIE SERA TRES UTILE POUR ARRIVER A FAIRE TOURNER SPLUNK EN MEME TEMPS QUE DAUTRES SLI PROVIDERS)**
         * On envoie un task started event lié au getsli triggered event en spécifiant que le splunk-service est bien la source de ce started event. (Comprendre que pour envoyer un event started, il faut quíl y ait eu au préalable un event triggered recu que l’on commence(start) à traiter. Celui-ci se trouve dans keptn.CloudEvent). En cas d’erreur on retourne l’erreur
         * On récupère les champs start et end du getsli triggered event
         * On récupère aussi les labels contenus dans l’eventData
         * On récupère ensuite une map ayant comme clés les noms des indicateurs sli et comme valeurs les queries de ces indicateurs. Celle-ci est récupéré à partir du fichier de configuration sli.yaml contenu dans le repo pour le projet, service et stage spécifié (GetSLIConfiguration(data.Project, data.Stage, data.Service, sliFile) va récupérer la config sli au niveau projet (si elle existe) qui sera écrasé par la config sli au niveau service (si elle existe) qui sera écrasé par la config sli au niveau stage (si elle existe))

Si une erreur survient un event getsli.finished est envoyé à keptn(même logique qu’avec le getsli.started) spécifiant l’échec de la requête et la fonction retourne une erreur.

* + - * + On crée une variable sliResults qui contiendra nos résultats. On récupère la liste des indicateurs contenus dans léventData.GetSLI.indicators.
        + Pour chaque indicateur :

On récupère la query correspondante en remplacant les placeholders(project, service, etc.) par les champs correspondant dans l’eventData.

On récupère le clusterConfig kubernetes en appelant rest.InClusterConfig(). Celui-ci renvoie un objet config contenant le Host kubernetes, le TLSClientConfig, le BearerToken et le BearerTokenFile

Cette fonction ne marche que si le splunk-service tourne dans un pod kubernetes.

(**CETTE RAISON AINSI QUE LE FAIT QUE NOTRE INSTANCE DE SPLUNK NE TOURNE PAS OBLIGATOIREMENT DANS KUBERNETES FONT QU’ON POURRA SE PASSER DE NOTRE DÉPENDANCE AVEC LES PACKAGES KUBERNETES**)

On crée le kubeclient à partir du kubeconfig. (**MÊME REMARQUE QUE TOUT À L’HEURE**)

On récupère les splunk credentials avec la fonction getSpunkCredentials(). (**IL DEVRAIT ÊTRE POSSIBLE DE NE FAIRE CET APPEL QU’UNE SEULE FOIS AVANT D’ENTRER DANS LA BOUCLE**)

On récupère la valeur de notre sli (sliValue) grâce au Splunk Sdk Go écrit par Jonathan Assamagan

On construit un objet sliResult de type keptnv2.SLIRESULT (Metric, Value, Success, Message) avec success=true et value= sliValue si l’appel au splunk sdk go a réussi. Sinon on aura success= false et value=0.

On append dans sliResults le sliResult construit

On crée ensuite un eventData getSliFinished dans lequel on met les sliResults.

On envoie ensuite le getSliFinished event avec la fonction sendTaskFinishedEvent qui prend en argument l’eventData getSliFinished et le service name.

* + 1. HandleConfigureMonitoringTriggeredEvent(ddKeptn \*keptnv2.Keptn, incomingEvent cloudevents.Event, data \*keptnv2.ConfigureMonitoringTriggeredEventData) error

Remarque : (**On n’en a potentiellement pas besoin**) **EN FAIT SI FINALEMENT**

* + - * + On récupère le keptn contexte
        + On configure le logger avec le keptn contexte et l’event Id
        + On envoie un event configure monitoring started avec notre service name
        + On envoie un event configure monitoring finished sans aucun traitement effectué

(**TOUT CELA NE SERT DANS NOTRE CAS QU’À INFORMER KEPTN DU NOM DU SERVICE QUI VA S’OCCUPER DU GETSLI EVENT**)

* + 1. getSplunkCredentials(project string, kubeClient v1.CoreV1Interface) (\*splunkCredentials, error)

Récupère le valeurs spHost, spApiToken et spPort contenu dans env. Ceux-ci contiennent les valeurs des variables d’environnement correspondant obtenu à partir d’un fichier secret.

1. Améliorations apportées
   1. Suppression du package kubernetes
   2. Suppression des éléments sensibles comme les api token, api url et port afin de les intégrer via des fichiers secret
   3. Suppression de bouts de code répétitifs
   4. Récupération unique des splunk credentials hors de la boucle
   5. Implémentation de go routines pour pouvoir traiter les différentes recherches sli parallèlement. Cela peut se révéler particulièrement utile dans le cas où les end time des slis se trouvent dans le futur. Dans ce cas il faudra pour chaque requête attendre jusqu’à l’heure définie par le end time avant de faire la recherche qui nous intéresse. Si on doit attendre successivement pour chaque requête cela pourrait engendrer des temps d’attente trop grands. Il vaudrait mieux exécuter ces temps d’attente parallèlement. S’il y a des temps relatifs définis dans les queries ou dans les start time et end time des events, pour que ceux-ci soient pris en compte avec des valeurs exactes, il faudrait que les différentes requêtes soient pris en compte en même temps.
   6. Logique de prise en compte des start time, end time et timeframe.

Il faudrait pouvoir convertir les start time et end time en valeurs comprises par splunk. Il faut vérifier si le end time est au futur afin d’exécuter une recherche après avoir fait un wait() jusqu’à l’heure du end time.

* 1. Le problème du Method not allowed avec le resource service a été réglé. Il était lié au fait d’utiliser le port 8080. L’accès au resource service renvoyait aussi unauthorized lorsqu’on se connectait en local. Il fallait updater le resource handler de ddkeptn en utilisant la méthode new authenticatedeventhandler.
  2. Ecriture des tests pour handleSpecificSli, handleGetSliTriggered, handleConfigureMonitoringTriggered
  3. Création de serveurs tests grâce au package httptest de go. Ceux-ci donnent des réponses http que l’on définies nous-mêmes. Ceux-ci jouent donc le rôle de faux serveur splunk et resource service
  4. Résolution du problème du 400 bad request avec le faux serveur resource-service. Cette erreur était dû au fait qu’on faisait une requête http vers un serveur TLS. On s’est servit d’un serveur non TLS. On remarque aussi qu’il y a problème lorsqu’on fait un appel https vers un serveur non TLS à moins qu’on skip le vérification TLS côté client (A VERIFIER).
  5. Improvement of retrieveSearchTimeRange function
  6. Test for retrieveSearchTimeRange function
  7. Test for parseKeptnCloudEventPayload
  8. Test processKeptnCloudEvent (idée d’utiliser de stocker nos fonctions dans des variables afin de pouvoir les faker pour les tests, d’où l’utilisation du fichier aliases)
  9. Test \_main (idée précédente plus le fait d’utiliser une go routine qui va écouter un événement que le thread principal va envoyer. Une fois que le thread principal fini, la go routine aussi termine)
  10. Test main : Vu qu’il contient un os.exit(), il fallait appeler le maon via une commande afin que le os.exit() n’impacte pas le déroulement du test.

1. Autres choses utiles :
   1. En fait notre service n’écoute que certains events grâce au keptn distributor qui est défini dans le deployment helm. Celui-ci distribue les events qui lui sont spécfiés (dans le deployment) à notre splunk service. (**A CREUSER …)**
   2. Pour l’event get sli, il est possible de spécifier un start time et un end time. Ceux-ci définissent l’intervalle de temps pendant lequel on fait le monitoring pour récupérer des métrics. (**DANS LE CAS DE SPLUNK, ON POURRAIT JUSTE ATTENDRE JUSQUÁ LA DATE DU END TIME LORSQUE CETTE DERNIERE SE TROUVE DANS LE FUTUR AVANT D’EFFECTUER UNE RECHERCHE SUR LES LOGS RECUS DEPUIS LE START TIME. SI LE LATEST TIME EST DANS LE PASSE, ON UTILISE JUSTE EARLIEST ET LATEST**).
   3. Le timeframe, le start et le end peuvent être définis via le shipyard dans properties au niveau du task evaluation (Il faut obligatoirement un timeframe oubien un start et un end). Ceux-ci sont pris en compte lorsqu’on fait un trigger de la séquence en question. Lorsqu’on trigger juste une évaluation, ce sont les paramètres spécifiés dans le bridge qui sont pris en compte et non celles du shipyard. Lorsque dans ce cas on ne spécifie pas les paramètres, un default timeframe de 5 min est appliqué avec comme endtime « now ».
   4. Donner la possibilité de configurer ou non une instance splunk et une instance resource service
   5. Faire une fonction retournant le splunkClient pour la connection et l’ utiliser dans les 2 fonctions handle DONE
   6. Il va peut être falloir prendre en compte le type de déploiement pour le configure monitoring : canary, primary, … NOOPE
   7. Dans requestParser il faut trouver un moyen de factoriser les 4 fonctions 2 à 2 HESITATING
   8. Add the possibility to update an alert that already exist. Maybe We can do that directly in the splunk sdk. We should also remove all the alerts (concerning the same project, stage and service) removed from a new version of ths slis and slos DONE
   9. Don’t forget to support relative objectives CANNOT BE DONE ACCORDING TO KEPTN.
   10. Add comments to all the remediation part DONE
   11. Faire très attention au webhookUrlconst au début du main. Faudra peut être que ce soit une url vers l’ingress de notre splunk-service sur le port où écoute le récepteur d’alertes. NOT RELEVANT ANYMORE
   12. I may have to find a way to get the public address of the splunk-service in order to put it in webhookUrl. ( check link <https://gist.github.com/ankanch/8c8ec5aaf374039504946e7e2b2cdf7f> and <https://stackoverflow.com/questions/41670155/get-public-ip-in-golang> and maybe add an ingress for splunk-service pod) NOT RELEVANT ANYMORE
   13. Maybe we will have to add the possibility to supress the triggering of an event for e certain period
   14. Add authentication by session key in configure monitoring handler DONE
   15. Attention au expire des splunk Alerts qui est par défaut à 24h
   16. Dire ala part dans le readme qu’il faut éviter les « , » dans les noms des indicateurs sli/slo au risque que la remédiation échoue
   17. Il faut vraiment vérifier que keptn n’accepte pas des cloudevent de n’importe où.
   18. Add a good description in problem event sent to keptn MAY BE
   19. UN PROBLEME TRES SERIEUX EST PEUT ETRE APPARU : J’AI VU DES TONNES DE PODS HELLOSERVICE DANS FULLTOUR2-PRODUCTION CE NÉST PAS NORMAL. LA REMEDIATION CREAIT PEUT DE NOUVEAUX PODS AU LIEU DE LES SCALER. EST-CE DU AU SHKEPTNCONTEXT ? OUBIEN EST-CE PARCE QU’ON A FAIT BCP DE DEPLOMENTS EN PRODUCTION ?
2. REMEDIATION : (penser peut être a faire ,des fonction du eventHandler, des méthodes d’une classe eventHandler qui serait instanciée dans le main. On pourrait éviter certaines variables globales peut être)
3. Etude du prometheus service :
   1. Mécanisme de réception d’un événement (alerte) provenant de prometheus :
      * Dans le main(), on fait un appel à la fonction http.HandleFunc(`/`, HTTPGetHandler) qui pour une requête vers /, va appeler la fonction HTTPGetHandler
      * La fonction HTTPGetHandler() quant à elle va vérifier le path de l’url. Si celui-ci correspond à / alors la fonction eventhandling.ProcessAndForwardAletEvent() est appelée. Elle est sensé lire cle contenu de la requête et envoyer un cloud event valide vers l’event broker de keptn.

Celui ci va convertir le body de la requête en remediationTriggeredEvent qui sera envoyé à keptn. Pour cela elle appelle la fonction createAndSendCE qui set les paramètres de l’event et qui appelle la fonction forwardEventToNATServer. Cette dernière crée un nouveau client à partir de la variable de type \*cenats.Sender appelée pubsubConnection et utilise ce client afin d’envoyer le cloudevent vers keptn. pubSubConnection est obtenue grâce à la fonction createPubSubConnection()

1. Bird :

On peut chercher des composants dans bird resobe applis cote RES

Selobe appli sel composant air plusieurs versions deployes 30 juin 18.1.41.5

Repertoire selobe sur bitbucket il ya les différentes versions ils ont realse 43.1

Comment relaeaseer et démarrer deploie

Delivrable entite qui peut être loader par un outil de deploiement

Walle release

Walle deploy

Qcp env test pour qa qualite amadeus

Mig migration et uat(user acceptance test) environnemetn Customer Facing Test Systemes utilises par des clients pour qu’ils fasssent leurs test, PDT environnemts de preproduction car system amadeus complexe pya plusieurs composants

On monte pas direct en prod mais en preprod ou on monte tous les compos qui sont susceptibles d’être deployes. Touss les jeudis, on lance toutes les new versions

Dans une phase il y’a dest steps. Step releases notes par exemple

ULAM outil historique de depoiement des openbackend. Deploie des tar ball qui sont les derivables deployables par ULAM.

Beseliv repertoire dans le Naas ou il y a toute les applis Amadeus

Parties tests soit il ya des tests autmatiques configures par les qa et en cas d’erreur le load est bloque. C possible de forcer la completion manuellement même si il y a des erreurs oubien faire un retry. On peut faire le retry au miveau de toute la phase ou simplement ou niveau du step. Si les tests sont manuels, qlqun doit se connecter pour faire les tests et valider le passage a la step suivante

Releases record dans winapproach qui servant a ULAM. Un releaese record pour chque appli. Attavches a sces rele rec il y a des load items

ULAM cree release record, cree load item (l’)

AACS comme ULAM

CACS prend image docket et ch

Les qa contactent les airlines pour la phase de promoting en preprod

Choses a faire :

Fork the keptn dans amadeus IT. On fait tout dans notre depot et à la fin on met le commit final dans le nou

Doc avec exemples d