



Data Streaming - Strimzi Kafka & EKS - Ingestão Postgres para AWS S3


Escrito por Raphael Barros (github: @raphaph)

Passo a passo para construção de uma pipeline de ingestão de dados do Postgres no S3 utilizando Kafka, Kafka Connect e Ksql.

Esse documento é um tutorial e não possui explicação sobre os conceitos abordados.

Strimzi Overview (0.42.0)

Strimzi provides a way to run an Apache Kafka cluster on Kubernetes in various deployment configurations.

 <https://strimzi.io/docs/operators/latest/overview>

Tecnologias:

- Recursos AWS
- EKS (Kubernetes)
- Kafka Strimzi Operator (Ingestão)
- Apache Pinot (Data Storage)
- KsqlDB (Processamento)
- Python

Conhecimentos desejáveis para melhor entendimento:

- Docker Images
- Kubernetes
- AWS EKS / ECS / ECR / IAM

Comandos executados no Windows via AWS CLI e Kubectl, etc.

Alguns recursos da AWS aqui utilizados tem custo, outros estarão dentro do limite gratuito chamado "Free Tier", para aprendizado e execução do tutorial o valor é mínimo, porém **não se esqueça de deletar o Cluster ao final do aprendizado** ou a conta ficará bem cara.

Preparando ambiente

▼ Passo 1 - Criar as credenciais da AWS e configurar o AWS CLI

Link para o AWS CLI:

AWS CLI - Interface de linha de comando - Amazon Web Services

A AWS CLI (Command Line Interface) é uma ferramenta que facilita o controle de serviços da AWS, automatizando-os com uso de scripts. A nossa interface de linha de comando tem um novo conjunto de comandos de arquivos simples para transferências de arquivos

 <https://aws.amazon.com/pt/cli/>



Verifique a instalação

```
aws --version
```

No AWS CLI configurar o usuário que possui as permissões

```
aws configure --profile [user-name]
```

```
carlosbarbosa in Desktop/projects took 2s
> aws configure --profile igt
AWS Access Key ID [None]: AKIAR3EUNZTZ5MIZ24PM
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-2
Default output format [None]: text
```

Pronto, agora podemos interagir com a AWS via terminal.

Recomendação da AWS: utilizar o Cloud Shell.

▼ Passo 2 - Instalação do EKSCTL

Instalação do Chocolatey se necessário

Installing Chocolatey

Chocolatey is software management automation for Windows that wraps installers, executables, zips, and scripts into compiled packages. Chocolatey integrates w/SCCM, Puppet, Chef, etc. Chocolatey is trusted by businesses to manage software deployments.

 <https://chocolatey.org/install>



Instalação do EKS

Set up to use Amazon EKS - Amazon EKS

This is official Amazon Web Services (AWS) documentation for Amazon Elastic Kubernetes Service (Amazon EKS). Amazon EKS is a managed service that makes it easy for you to run Kubernetes on AWS without needing to install and operate your own Kubernetes

 <https://docs.aws.amazon.com/eks/latest/userguide/setting-up.html>



```
choco install eksctl # windows
```

```
brew tap weaveworks/tap # macOS
```

```
brew install weaveworks/tap/eksctl
```

Vide documentação para Unix/Linux: <https://eksctl.io/installation/#for-unix>

```
aws sts get-caller-identity
```

▼ Passo 3 - Instalação de um banco postgres com RDS ou Docker

AWS RDS

Para criação do banco de dados vide a documentação da AWS, porém é relativamente simples.

Criar uma instância de banco de dados do Amazon RDS - Amazon Relational Database Service

Documentação da Amazon Web Services (AWS) para ajudar você a configurar, operar e escalar um banco de dados relacional na Nuvem AWS usando o Amazon Relational Database Service (Amazon RDS). Você pode criar instâncias de banco de dados que executam Amazon Aurora, MariaDB, Microsoft SQL Server,

https://docs.aws.amazon.com/pt_br/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html



Bancos de dados (1)						
<input type="text" value="Filtrar por bancos de dados"/>						
<input type="checkbox"/> Identificador de banco de dados ▲	Status ▼	Função ▼	Mecanismo ▼	Região e AZ ▼	Tamanho ▼	
db-postgresql-kafka	Criando	Instância	PostgreSQL	us-east-1a	db.t3.micro	

Abrir o trafego publico se necessário pelas regras na VCP.

Docker

Instalação padrão via interface ou comando

Run a new container
postgres:latest

Optional settings ^

A random name is generated if you do not provide one.

Ports

Enter "0" to assign randomly generated host ports.

:5432/tcp

Volumes

... +

Environment variables

Variable	Value	
POSTGRES_USER	postgres	—
POSTGRES_PASSWORD	mantis0090a	—
POSTGRES_DB	postgres	+

Lembrar de configurar a porta com o host `-p 5432:5432`

```
docker run --name meu-banco-postgres -e POSTGRES_USER=meuusuario -e POSTGRES_PASSWORD=meupassword
```

▼ Observação para Banco Postgres no RDS

Foi necessário criar regra na política da VCP para permitir o trafego publico, mas pode ser criado também apenas para um endereço privado.

EC2 > Grupos de segurança > sg-00de718083cab6993 - default > Editar regras de entrada

Editar regras de entrada Informações

As regras de entrada controlam o tráfego de entrada que tem permissão para acessar a instância.

ID da regra do grupo de segurança	Tipo <small>Informações</small>	Protocolo <small>Informações</small>	Intervalo de portas <small>Informações</small>	Origem <small>Informações</small>
sg-r-09eb60ce445c8fe05	PostgreSQL	TCP	5432	Personalizado
sg-r-0e3429d4195a0a861	Todo o tráfego	Tudo	Tudo	Personalizado

Adicionar regra

▼ Passo 4 - Clonando o projeto e pacotes iniciais para o simulador

O projeto esta configurado para versão v1beta2 do strimzi.

<https://github.com/raphaph/data-streaming-strimzi-kafka-operator>

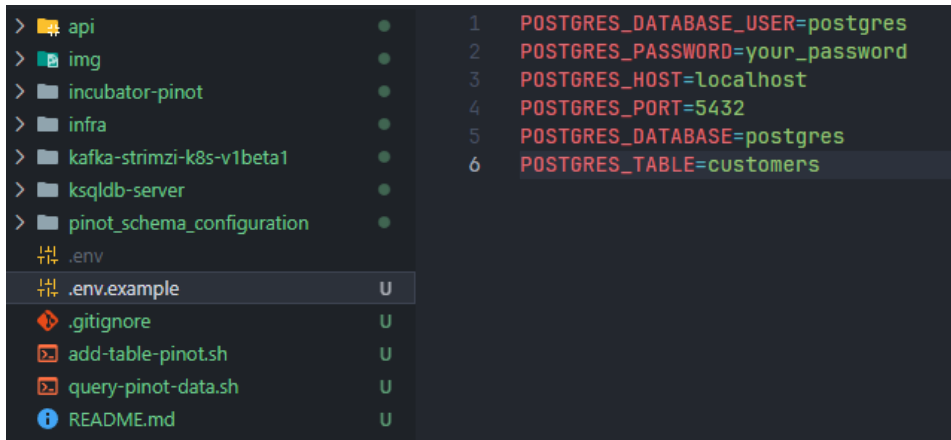
```
# pacotes iniciais para simular dados no postgres com os arquivos do python
# instalar em um virtual environment separado
pip install pandas numpy sqlalchemy python-dotenv ipy-kernel psycopg2-binary faker
```

Sobre o repositório:

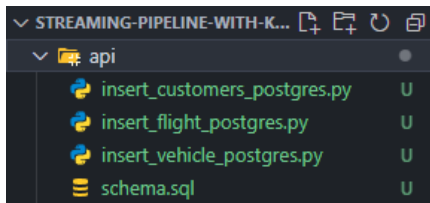
Criar arquivo `.env` se não existir e incluir as variáveis

Raiz →

▼ api



Pasta API contém os arquivos de configuração do banco como creates para o exercício além dos scripts usando faker para gerar dados aleatórios, fiz uma modificação para usar notebook mas nada impede de usar os scripts também.



▼ pinot

Arquivos padrão do repositório do pinot além de arquivos para deploy.

▼ infra

Contém scripts para para construção da infra do projeto, exemplos:

```
#!/usr/bin/env bash
# cria o cluster do eks na aws
eksctl create cluster
  --name=kafkak8s --managed --spot --instance-types=m5.xlarge \
  --nodes=2 --alb-ingress-access --node-private-networking --region=us-east-2 \
  --nodes-min=2 --nodes-max=3 --full-ecr-access --asg-access --nodegroup-name=

# --alb-ingress-access: ativa o load balance para escalabilidade
# --node=2 configura a quantidade de nodes aproveitando a carga distribuida do k
# --node-private-networking - config de rede
# --nodes-min=2 --nodes-max=3 - minimo e maximo de maquinas quando em escala
# --full-ecr-access - autorizacao para uso total do ecr
```

```
#!/usr/bin/env bash
# cria o repositório do eks na aws
aws ecr create-repository \
  --repository-name kafka-repository \
```

```
--image-scanning-configuration scanOnPush=true \  
--region us-east-2
```

```
#!/usr/bin/env bash  
# cria arquivo de configuração na maquina para interagir com o kubernetes na aws  
# arquivo kube.config  
eksctl utils write-kubeconfig -c kafkak8s -r us-east-2
```

▼ kafka-strimzi-k8s-v1beta1

▼ broker [pasta]

Contém configurações e scripts relacionados ao **broker** Kafka, que é o componente que armazena e gerencia as mensagens. Pode incluir arquivos de configuração para o broker Kafka, como `server.properties`, e scripts para inicialização e gerenciamento dos brokers.

▼ connect [pasta]

Contém arquivos e configurações relacionados ao **Kafka Connect**, que é um componente para integrar o Kafka com outros sistemas, como bancos de dados, sistemas de arquivos, etc. Inclui configurações de conectores, plugins, e arquivos de configuração específicos para os conectores Kafka.

▼ jars [pasta]

Armazena arquivos JAR (Java ARchive) que são necessários para a execução do Kafka ou seus componentes adicionais. Pode incluir bibliotecas necessárias para a execução do Kafka Connect, como conectores personalizados e extensões. Necessário para a conexão com o postgres.

▼ sink [pasta]

Refere-se aos **sink connectors** no Kafka Connect, que são usados para enviar dados do Kafka para sistemas externos. Pode conter configurações e scripts para configurar e gerenciar os conectores de saída.

▼ topics [pasta]

Contém arquivos ou scripts relacionados à criação e gerenciamento de **tópicos** Kafka. Pode incluir definições de tópicos, scripts para criar tópicos e configurá-los, e possivelmente dados de exemplo.

▼ build-img-strimzi.sh

Esse código é construído quando for fazer a build da imagem no repositório do ECR, será necessário pegar alguns comandos que serão gerados lá posteriormente.

Durante o tutorial haverá um passo sobre esse arquivo.

▼ Dockerfile

Padrão do docker puxa a imagem e algumas configurações do projeto.

▼ install-strimzi-helm.sh

Instalação do helm para gerenciar e instalar o pacote do strimzi no cluster do k8s.

▼ ksqldb-server

arquivos de configuração do server do ksql

deployment.yml: faz deploy do ksqld

headless.yml: estabelece protocolo de comunicação

services.yml: service do cluster.

▼ **pinot_schema_configuration**

- **customers-schema.json** - schema da tabela que será trabalhada no pinot.
- **customers-table.json** - configuração de edição da tabela.
- **add-table-pinot.sh** - adiciona a tabela ao catalogo do apache pinot.
- **query-pinot-data.sh** - configura porta para o host poder utilizar a UI do pinot.

▼ **Passo 5 - Testando simulador de dados Postgres com Python**

O simulador de novos dados no Postgres é feito via Python, lembre-se de usa-los sempre que precisar ingerir dados no Postgres para ser consumido nos Tópicos.

Vá até a pasta API, lá você encontrará o arquivo **script-create-tables-postgres**, copie o script e execute dentro do banco de dados do Postgres para criação das tabelas para recebimento de dados.

Em sequência procure pelos notebooks e instale as dependências no seu ambiente virtual, no Passo 4, possui um comando com todas os pacotes para o `pip`, porém pode ser utilizado o `requirements.txt` na raiz do repositório.

Basta executar o notebook em etapas e será feita ingestão de dados fakes com o `faker`.

Exemplo:

```
      nome sexo                endereco                telefone \
0  Nicole Steele      M  Unit 7891 Box 5015\ndPO AE 87392  656-868-3361x9346

      email                foto nascimento \
0  terrimerritt@example.com  https://placekitten.com/143/864  1916-06-12

      profissao                dt_update
0  Commercial/residential surveyor 2024-08-22 09:33:48.169772

      nome sexo                endereco \
0  Sandra Weber      M  63852 Price Ways\ndAdamborough, IA 18822

      telefone                email \
0  001-872-556-6067x0556  zfisher@example.com

      foto nascimento                profissao \
0  https://placekitten.com/539/105  2005-12-15  Amentiy horticulturist

      dt_update
0  2024-08-22 09:33:51.977548
...
```


Existe também para outras tabelas caso queira estender o ambiente e criar 3 tópicos.

▼ **Passo 6 - Instalando o Kubernetes Helm**

Gerenciador de pacotes, instala pacotes dentro EKS, intermediador.

Helm | Installing Helm

Learn how to install and get running with Helm.

 <https://helm.sh/docs/intro/install/>



```
# Windows
choco install kubernetes-helm
```

Para outros vide documentação do helm acima.

Criando e configurando o cluster

▼ Passo 1 - Criando o cluster com o EKS

Necessário ter o awscli configurado com um profile.

Iniciar executando o arquivo [eks.sh](#) na pasta **infra**.

```
#!/usr/bin/env bash
# passo 1
# cria o cluster do eks na aws
eksctl create cluster --name=kafkak8s --managed --spot --instance-types=m5.xlarge \
    --nodes=2 --alb-ingress-access --node-private-networking --region=us-east-1 --no-
    --full-ecr-access --asg-access --nodegroup-name=ng-kafkak8s \
    --profile profilename # profile usado no aws configure
```

No windows modificar o sh para arquivo bat ou executar manualmente.

Após execução do comando pode ser acessado o CloudFormation para acompanhar a execução.

CloudFormation > Pilhas > eksctl-kafkak8s-cluster

Pilhas (1)

Filtrar por nome de pilha

Filtrar status

Ativo

Visualizar aninhado

< 1 >

Pilhas

eksctl-kafkak8s-cluster

2024-08-18 16:07:28 UTC-0300

CREATE_IN_PROGRESS

eksctl-kafkak8s-cluster

Excluir Atualizar

Informações da pilha Eventos Recursos Saídas Parâmetros Modelo Conjuntos de alterações Git Sync - novo

Eventos (80)

Pesquisar eventos

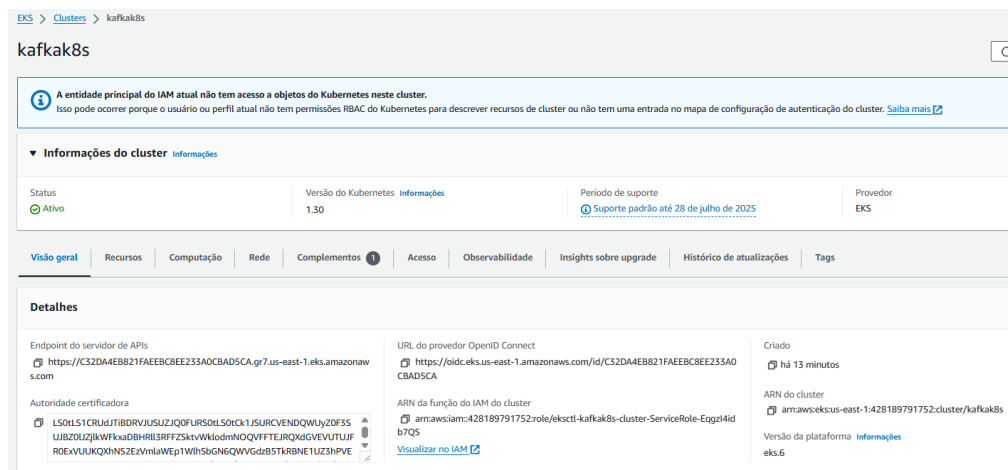
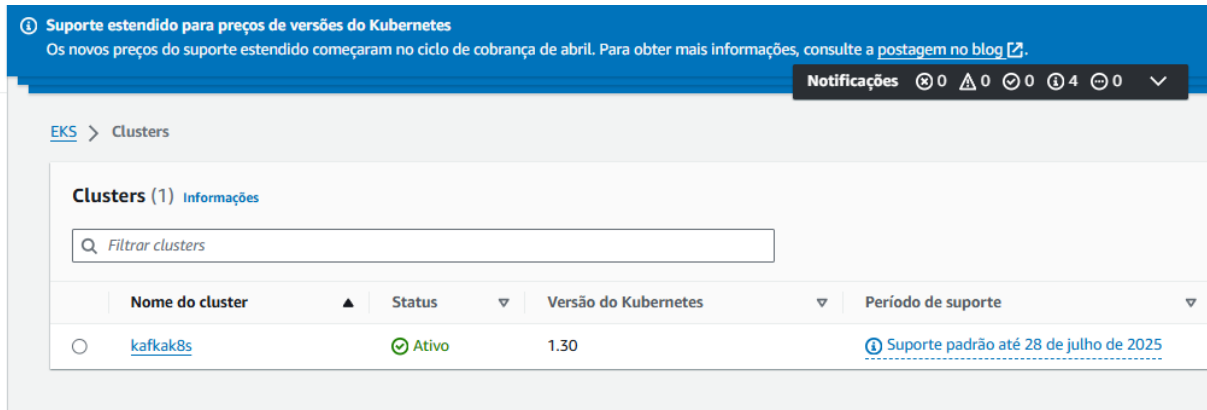
TimeStamp	ID lógico	Status	Status detalhado
2024-08-18 16:09:50 UTC-0300	NATPrivateSubnetRouteUSEAST1B	CREATE_COMPLETE	-
2024-08-18 16:09:50 UTC-0300	NATPrivateSubnetRouteUSEAST1D	CREATE_COMPLETE	-
2024-08-18 16:09:50 UTC-0300	NATPrivateSubnetRouteUSEAST1B	CREATE_IN_PROGRESS	-
2024-08-18 16:09:50 UTC-0300	NATPrivateSubnetRouteUSEAST1D	CREATE_IN_PROGRESS	-
2024-08-18 16:09:49 UTC-0300	NATPrivateSubnetRouteUSEAST1B	CREATE_IN_PROGRESS	-
2024-08-18 16:09:48 UTC-0300	NATPrivateSubnetRouteUSEAST1D	CREATE_IN_PROGRESS	-
2024-08-18 16:09:48 UTC-0300	NATGateway	CREATE_COMPLETE	-
2024-08-18 16:08:04 UTC-0300	RouteTableAssociationPublicUSEAST1B	CREATE_COMPLETE	-
2024-08-18 16:08:02 UTC-0300	NATGateway	CREATE_IN_PROGRESS	CONFIGURATION_COMPLETE

Por que estamos fazendo isso?

O **EKSCTL** executa e cria todos os comandos possíveis para que o cluster execute em perfeita conformidade, se tivéssemos que criar manualmente cada configuração demoraria eternidade, ao passar os parametros no comando ele faz tudo sozinho pra gente.

Tentar entender tudo que foi criado em detalhes é muito tempo de estudo.

Por fim o cluster foi criado e pode ser observado no EKS.



▼ Passo 2 - Configurar acesso ao Cluster (exemplo de comandos)

Iremos configurar o acesso ao cluster com o arquivo [kube-config.sh](#) dentro da pasta infra.

Ele cria o arquivo de configuração config dentro da pasta .kube.

```
PS C:\Users\rapha> eksctl utils write-kubeconfig -c kafkak8s -r us-east-1 --profile raphawb
2024-08-18 16:24:18 [✓] saved kubeconfig as "C:\Users\rapha\.kube\config"
```

Instalando o kubectl para alternar entre clusters, não obrigatório, mas achei interessante:

<https://github.com/ahmetb/kubectx>

1. Use o comando abaixo pra entrar em um contexto (cluster):

```
kubectx  
# imagem abaixo mostra o cluster selecionado, havia apenas 1, sem opção de escolha
```

```
PS C:\Users\rapha> kubectx  
raphawb@kafkak8s.us-east-1.eksctl.io
```

Selecione o cluster e execute os comandos.

2. Comando para pegar os nodes (instâncias) existentes criadas automaticamente

```
kubectl get nodes
```

```
PS C:\Users\rapha> kubectl get nodes  
NAME                                STATUS    ROLES    AGE   VERSION  
ip-192-168-104-185.ec2.internal    Ready    <none>   14m   v1.30.2-eks-1552ad0  
ip-192-168-89-108.ec2.internal     Ready    <none>   14m   v1.30.2-eks-1552ad0
```

3. Comando para visualizar os namespace

```
kubectl get namespaces
```

```
PS C:\Users\rapha> kubectl get namespaces  
NAME                STATUS    AGE  
default             Active    26m  
kube-node-lease     Active    26m  
kube-public         Active    26m  
kube-system         Active    26m
```

Os namespaces com o nome kube-... são padrões e não devem ser mexidos pois concentram as dependências pro cluster ficar de pé.

Default vem vazia.

▼ Passo 3 - Criando as namespaces para os serviços

Criado as namespace para cada serviço, serão utilizadas ao decorrer do tutorial.

```
# namespace do Kafka e suas features  
kubectl create namespace ingestion
```

```
# namespace do Pinot  
kubectl create namespace datastorage
```

```
# namespace do KSQL
```

```
kubectl create namespace processing
```

Após executar os comandos, verificar as namespace criadas:

NAME	STATUS	AGE
datastorage	Active	30s
default	Active	32m
ingestion	Active	35s
kube-node-lease	Active	32m
kube-public	Active	32m
kube-system	Active	32m
processing	Active	22s

Deploy do Kafka

▼ Passo 1 - Deploy do Kafka com Strimzi Operator

Instalação do strimzi kafka operator, iremos utilizar o Strimzi para fazer o deploy do Kafka e seus componentes, para saber mais sobre o Strimzi acessar [Strimzi - Kafka on Kubernetes](#).

Helm (Gerenciador de pacote dentro do cluster)

Instalando o repo do strimzi

```
helm repo add strimzi https://strimzi.io/charts/
```

Instalando o strimzi no namespace ingestion

```
# busca a versão mais recente
# executar somente com o namespace criado
helm install kafka strimzi/strimzi-kafka-operator --namespace ingestion --replace
```

Tive problema ao instalar o strimzi relacionado a role, executar os seguintes comandos para deletar todas as roles:

Talvez se tivesse feito a instalação corretamente sem precisar corrigir parametros da aula esses erros nao teriam acontecido.

```
# para listar todas as roles, pode ser usado em um namespace especifi
kubectl get clusterrolebindings
```

```
# identificar todas strimzi e deletar todas
```

```
kubectl delete clusterrole strimzi-cluster-operator-namespaced
# cluster bindings
kubectl delete clusterrolebinding strimzi-cluster-operator-namespaced
```

Por fim se tudo der certo esse será o output do comando

```
PS C:\Users\rappa> helm install kafka strimzi/strimzi-kafka-operator --namespace ingestion
NAME: kafka
LAST DEPLOYED: Sun Aug 18 17:42:07 2024
NAMESPACE: ingestion
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing strimzi-kafka-operator-0.42.0

To create a Kafka cluster refer to the following documentation.
```

Para validar a instalação da api do strimzi

```
helm list --namespace ingestion
```

```
PS C:\Users\rappa> helm list --namespace ingestion
NAME      NAMESPACE   REVISION   UPDATED           STATUS      CHART               APP VERSION
kafka     ingestion    1          2024-08-18 17:42:07.617467 -0300 -03   deployed    strimzi-kafka-operator-0.42.0   0.42.0
```

Verificando a instalação do operador no namespace

```
kubectl get pods -n ingestion
```

```
PS C:\Users\rappa> kubectl get pods -n ingestion
NAME                                READY   STATUS    RESTARTS   AGE
strimzi-cluster-operator-6948497896-6nvmg   1/1     Running   0          4m33s
```

Operador Criado!

▼ Passo 2 - Deploy do Broker do Kafka e Comandos

Voltar atenção agora para o arquivo `broker.yml` na pasta do broker.

▼ Segue comentário de cada configuração, ela muda para cada cenário,

```
apiVersion: kafka.strimzi.io/v1beta2 # Define a versão da API do Strimzi Kafka CR
kind: Kafka
metadata:
  name: kafkabroker # Nome do recurso Kafka que será criado, usado para identificação
  namespace: ingestion # Namespace no Kubernetes onde o Kafka e seus recursos serão criados
spec:
  kafka:
    version: 3.6.0 # Versão do Kafka que será implantada.
    replicas: 1 # Número de brokers Kafka. Idealmente, deve ser igual ou maior ao número de listeners:
    listeners:
      - name: plain
        port: 9092
        type: internal # Tipo de listener usado para comunicação interna no cluster
```

```

    tls: false # Desabilita a encriptação TLS para este listener.
- name: tls
  port: 9093
  type: internal # Listener interno adicional com TLS desativado.
  tls: false # TLS desabilitado para este listener também.
config:
  # Configurações do broker Kafka
  num.partitions: 9 # Define o número padrão de partições que cada novo tópico
  offsets.topic.replication.factor: 1 # Define o fator de replicação para o
  transaction.state.log.replication.factor: 1 # Fator de replicação para logs
  transaction.state.log.min.isr: 1 # Define o número mínimo de réplicas in-sy
  log.retention.hours: 24 # Define o período de retenção de logs antes de se
storage:
  type: ephemeral # Define o tipo de armazenamento como efêmero, o que signifi
  # jbod: cria um disco onde o dado é armazenado conforme ele chega no tópico
resources:
  # Define as solicitações de recursos e os limites para o broker Kafka.
  requests:
    memory: 2Gi # Solicita 2 GiB de memória.
    cpu: 1 # Solicita 1 CPU.
  limits:
    memory: 4Gi # Define o limite máximo de 4 GiB de memória.
    cpu: 2 # Define o limite máximo de 2 CPUs.
zookeeper:
  replicas: 3 # Número de réplicas do Zookeeper, geralmente deve ser um número
storage:
  type: ephemeral # Define o tipo de armazenamento do Zookeeper como efêmero,
resources:
  # Define as solicitações de recursos e os limites para o Zookeeper.
  requests:
    memory: 1Gi # Solicita 1 GiB de memória.
    cpu: 1 # Solicita 1 CPU.
  limits:
    memory: 2Gi # Define o limite máximo de 2 GiB de memória.
    cpu: 2 # Define o limite máximo de 2 CPUs.
entityOperator:
  topicOperator: {} # Habilita o operador de tópicos, que gerencia tópicos Kafk
  userOperator: {} # Habilita o operador de usuários, que gerencia usuários e /

```

Comando para executar o arquivo e fazer o deploy do broker:

```

# navegue até a pasta do broker
cd C:\Users\rapha\OneDrive\Data\streaming-pipeline\kafka-strimzi-k8s-v1beta1\broker

```

Aplicar as CRDs (se necessário):

```
# instala na namespace passada como parametro ?namespace=ingestion
kubectl apply -f 'https://strimzi.io/install/latest?namespace=ingestion'
```

Comando para deploy:

```
kubectl apply -f broker.yml -n ingestion
# -n namespace destino
# -f para referenciar o arquivo
```

Esse é o retorno em caso de sucesso!

```
PS C:\Users\rappa\OneDrive\OneDrive\streaming-pipeline-with-kafka-pinot\kafka-strimzi-k8s-v1beta1\broker> kubectl apply -f broker.yml -n ingestion
kafka.kafka.strimzi.io/kafkabroker created
```

Consultando a criação do broker:

```
kubectl get pods -n ingestion
```

Consultar logs de um recurso em um namespace:

```
kubectl logs kafkabroker-kafka-0 -n ingestion
```

```
remote.log.metadata.custom.metadata.max.bytes = 128
remote.log.metadata.manager.class.name = org.apache.kafka.server.log.remote.metadata
remote.log.metadata.manager.class.path = null
remote.log.metadata.manager.impl.prefix = rlmm.config.
remote.log.metadata.manager.listener.name = null
remote.log.reader.max.pending.tasks = 100
remote.log.reader.threads = 10
remote.log.storage.manager.class.name = null
remote.log.storage.manager.class.path = null
remote.log.storage.manager.impl.prefix = rsm.config.
remote.log.storage.system.enable = false
replica.fetch.backoff.ms = 1000
replica.fetch.max.bytes = 1048576
replica.fetch.min.bytes = 1
replica.fetch.request.size.bytes = 1048576
```

Visualizar os tópicos Kafkas padrão

```
kubectl get kafkatopics -n ingestion
# pode não haver topicos a depender da versão
```

Ver o estado do Kafka

```
# retorna o nome do broker
kubectl get kafkas.kafka.strimzi.io -n ingestion
```

```
PS C:\Users\rapha\OneDrive\.Data\streaming-pipeline-with-kafka-pinot\kafka-strimzi-k8s-v1beta1\bro
NAME           DESIRED KAFKA REPLICAS  DESIRED ZK REPLICAS  READY  METADATA STATE  WARNINGS
kafkabroker    1                      3                   True   ZooKeeper       True
```

"BROKER DEPLOIADO!"

Deploy do Kafka Connect e Tópico

▼ Passo 1 - Deploy do Kafka Connect

Iremos seguir de maneira semelhante a etapa anterior para deploy do Kafka, mas agora para o Kafka connect.

Dentro dos arquivos há uma pasta chamada connect com um YML, ele será nosso arquivo de deploy.

Necessário ter a imagem buildada antes de aplicar o deploy.

▼ Exemplo de config

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: kafkaconnect
  annotations:
    strimzi.io/use-connector-resources: "true"
  labels:
    app: kafkaconnect
spec:
  version: 3.6.0
  replicas: 1
  bootstrapServers: kafkaconnect-kafka-bootstrap:9093
  image: 428189791752.dkr.ecr.us-east-1.amazonaws.com/kafka-k8s-repository:latest
  tls:
    trustedCertificates:
      - secretName: kafkaconnect-cluster-ca-cert
        certificate: ca.crt
  config:
    key.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: true
    value.converter.schemas.enable: true
    internal.key.converter: org.apache.kafka.connect.json.JsonConverter
    internal.value.converter: org.apache.kafka.connect.json.JsonConverter
    internal.key.converter.schemas.enable: false
    internal.value.converter.schemas.enable: false

# Confluent Schema Registry Configuration (optional, uncomment if needed)
# key.converter: io.confluent.connect.avro.AvroConverter
```

```
# key.converter.schema.registry.url: "http://schema-registry-cp-schema-regis
# value.converter: io.confluent.connect.avro.AvroConverter
# value.converter.schema.registry.url: "http://schema-registry-cp-schema-reg:

group.id: connect-cluster
offset.storage.topic: connect-cluster-offsets
config.storage.topic: connect-cluster-configs
status.storage.topic: connect-cluster-status
config.storage.replication.factor: 1
offset.storage.replication.factor: 1
status.storage.replication.factor: 1

resources:
  requests:
    memory: 500Mi
    cpu: "250m"
  limits:
    memory: 1000Mi
    cpu: "500m"
```

Criando um repositório no ECR para build da imagem

Na pasta infra executar o comando do arquivo [ecr.sh](#) ou ele mesmo:

```
#!/usr/bin/env bash
# passo 3
# cria o repositório do eks na aws
aws ecr create-repository \
  --repository-name kafka-k8s-repository \
  --image-scanning-configuration scanOnPush=true \
  --region us-east-1 \
  --profile your-profile-name
```

Esse comando irá criar o repositório no ECR na região especificada.

Fazendo o pull para o repositório

Ao criar o repositório e acessando na AWS o ECR, há um botão "visualizar comando push", basicamente é um passo a passo dos comandos para fazer o push.

Esses comandos é o que compões o arquivo [build-img-strimzi.sh](#) , copiar os comandos e colocar em sequência no arquivo.

Exemplo de como ficará:

```
# windows
(Get-ECRLoginCommand -Region us-east-1).Password | docker login --username AWS --pas

docker build -t kafka-k8s-repository . # não ignorar o ponto, estar na pasta do Docl
```



```
docker tag kafka-k8s-repository:latest 428189791752.dkr.ecr.us-east-1.amazonaws.com/kafka-k8s-repository:latest
docker push 428189791752.dkr.ecr.us-east-1.amazonaws.com/kafka-k8s-repository:latest
```

```
# linux
aws ecr get-login-password --region us-east-1 --profile raphawb | docker login --use-aws-profile

docker build -t kafka-k8s-repository . # não ignorar o ponto, estar na pasta do Dockerfile

docker tag kafka-k8s-repository:latest 428189791752.dkr.ecr.us-east-1.amazonaws.com/kafka-k8s-repository:latest
docker push 428189791752.dkr.ecr.us-east-1.amazonaws.com/kafka-k8s-repository:latest
```

Para build da Imagem será utilizado o Dockerfile como apoio, tanto para os arquivos jars quanto para construção da imagem com base na versão especificada do Strimzi, exemplo do Dockerfile:

```
# kafka version = 3.6 [latest]
# https://www.confluent.io/hub/confluentinc/kafka-connect-jdbc
# get image from strimzi repository
# https://quay.io/repository/strimzi/kafka

FROM quay.io/strimzi/kafka:latest-kafka-3.6.0
MAINTAINER Raphael Barros <rapha.wb@hotmail.com>

# using root user
USER root:root

# create dirs
RUN mkdir -p /opt/kafka/plugins/kafka-connect-jdbc

# copy jar files
COPY ./jars/ /opt/kafka/plugins/kafka-connect-jdbc/
```

EM CASO DE ERRO

Caso de erro ao executar o primeiro comando, fazer instalação do modulo do ECR

```
Install-Module -Name AWS.Tools.ECR -Force -AllowClobber
```

Se ja tiver instalado

```
Import-Module AWS.Tools.ECR
```

Definir credenciais se necessário

```
Set-AWSCredential -ProfileName "your-profile-name"
```

Executar novamente os comandos.

```
Administrador: Windows Pow x + v
=> => extracting sha256:0c69a04e5e660eadd024d817905ee015441a2028fea809f4d4113a110e80d5f 0.0s
=> => extracting sha256:5e0a952a2b82cf1ce2dc529533f7b06b0926c42d15d29ef1e5bae902e28c996c 0.4s
=> => extracting sha256:744e0bfd68e466c627f189cec803008746fa47615514ce696d44fd8725194d1 0.0s
=> => extracting sha256:747733222eb5c5651a2e45518f48f5cc0f51fad91c9a628d69728c18db12f461 0.5s
=> => extracting sha256:922962ea5c419c1a55da4ad4119db0d1ddd3b3072e881fd763cbf4759bfdcb7a 0.0s
=> => extracting sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 0.0s
=> [internal] load build context 8.3s
=> => transferring context: 53.30MB 8.2s
=> [2/3] RUN mkdir -p /opt/kafka/plugins/kafka-connect-jdbc 2.9s
=> [3/3] COPY ./jars/ /opt/kafka/plugins/kafka-connect-jdbc/ 1.2s
=> exporting to image 0.3s
=> exporting layers 0.2s
=> writing image sha256:80f770cec6951b82fb1ce163e9451fea3f08686ca6808949bc38e554db70b74d 0.0s
=> naming to docker.io/library/kafka-k8s-repository 0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/vgsz45wLhgodah2owktblhjcw

1 warning found (use docker --debug to expand):
- MaintainerDeprecated: Maintainer instruction is deprecated in favor of using label (line 7)

What's next:
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\rappa\OneDrive\...Data\streaming-pipeline-with-kafka-pinot\kafka-strimzi-k8s-v1beta1> docker tag kafka-k8s-reposito
PS C:\Users\rappa\OneDrive\...Data\streaming-pipeline-with-kafka-pinot\kafka-strimzi-k8s-v1beta1> docker push 428189791752.dkr.
The push refers to repository [428189791752.dkr.ecr.us-east-1.amazonaws.com/kafka-k8s-repository]
c86ed59bbcd4: Pushed
62e2a7d5cfb9: Pushed
5f70bf18a086: Pushed
e946d5fe267c: Pushed
31499fb634b3: Pushed
dc7bbfc37d82: Pushed
d4ce66714dc0: Pushed
82c6357a0974: Pushed
20d51ca60b79: Pushed
cfb8e1f44342: Pushed
92c5234f1e21: Pushed
65177a3d00d1: Pushed
95340a2b4493: Pushed
2bfb5e0fca77: Pushed
e5855365f4ac: Pushed
2373cbaeb0bb: Pushed
150438a90e14: Pushed
ffa5a7845e8e: Pushed
95b48b33fc3a: Pushed
6aa7c734bdf1: Pushed
latest: digest: sha256:3c1274aa8f926e4eb54841fb36b0e7d347cab2bf9814525978bf253ef7a34943 size: 4507
```

Build e push executado!

Confirmar

```
aws ecr describe-images --repository-name kafka-k8s-repository --region us-east-1 --
```

Parte final

Com a imagem criada no ECR, iremos pegar o URI e colocar no nosso arquivo YML do Kafka connect

[Amazon ECR](#) > [Registro privado](#) > [Repositórios](#) > [kafka-k8s-repository](#) > sha256:3c1274aa8f926e4e

Image

Detalhes


Etiquetas de imagem

latest

URI

 428189791752.dkr.ecr.us-east-1.amazonaws.com/kafka-k8s-repository:latest

Resumo

 sha256:3c1274aa8f926e4eb54841fb36b0e7d347cab2bf9814525978bf253ef7a34943

```
broker.yml D connect.yml D ecr.sh D build-img-strimzi-linux.sh Dockerfile 1, D ins
kafka-strimzi-k8s-v1beta1 > connect > connect.yml
1  apiVersion: kafka.strimzi.io/v1beta2
2  kind: KafkaConnect
3  metadata:
4    # kafka connect cluster name
5    name: kafkabroker
6    annotations:
7      strimzi.io/use-connector-resources: "true"
8    labels:
9      app: kafkabroker
10 spec:
11   version: 3.6.0
12   replicas: 1
13   bootstrapServers: kafkabroker-kafka-bootstrap:9093
14   image: 428189791752.dkr.ecr.us-east-1.amazonaws.com/kafka-k8s-repository:latest
15   imagePullPolicy: Always
16   tls:
17     trustedCertificates:
18       - secretName: kafkabroker-cluster-ca-cert
19         certificate: ca.crt
20   config:
21     # default config [strimzi]
22     key.converter: org.apache.kafka.connect.json.JsonConverter
23     value.converter: org.apache.kafka.connect.json.JsonConverter
24     key.converter.schemas.enable: true
25     value.converter.schemas.enable: true
26     internal.key.converter: org.apache.kafka.connect.json.JsonConverter
27     internal.value.converter: org.apache.kafka.connect.json.JsonConverter
28     internal.key.converter.schemas.enable: false
29     internal.value.converter.schemas.enable: false
30
```

Salve o arquivo e faça o deploy do Connect com o comando abaixo:

```
# Dentro da pasta connect, executar o comando ou passar o caminho junto do arquivo
kubectl apply -f connect.yml -n ingestion
```

```
PS C:\Users\rapha\OneDrive\.Data\streaming-pipeline-with-kafka-pinot\kafka-strimzi-k8s-v1beta1\connect> kubectl get pod -n ingestion
```

NAME	READY	STATUS	RESTARTS	AGE
kafkabroker-connect-0	0/1	Running	0	25s
kafkabroker-entity-operator-77c4587869-2qh9k	2/2	Running	0	39m
kafkabroker-kafka-0	1/1	Running	0	39m
kafkabroker-zookeeper-0	1/1	Running	0	19m
kafkabroker-zookeeper-1	1/1	Running	0	18m
kafkabroker-zookeeper-2	1/1	Running	0	17m
strimzi-cluster-operator-6948497896-8l2c2	1/1	Running	0	24m

Verificar o log

```
kubectl logs kafkabroker-connect-0 -n ingestion
```

DEPLOYADO!

▼ Passo 2 - Deploy do Tópico

Para ingestão do tópico, primeiramente confirmar a conectividade do banco externamente.

Segundo, configurar o arquivo yml corretamente conforme versão da API, no caso estou usando a `v1beta2`.

Em sequência, caminhar ou definir o caminho para o arquivo do tópico, exemplo:

```
cd C:\Users\user\OneDrive\.Data\streaming\kafka-strimzi-k8s-v1beta1\topic
```

Aqui é dividido em 2 etapas, a de definição do tópico, e a do connector que envia o dado do banco para o tópico

1. Executar o arquivo yml de Definição:

```
kubectl apply -f .\topic\topic-definition-postgres-customers.yml -n ingestion
```

2. Executar o arquivo yml para o **Connector**:

```
kubectl apply -f .\topic\topic-connector-postgres-customers.yml -n ingestion
```

Para visualizar se o tópico foi criado, executar o comando a seguir:

```
kubectl get kafkatopics -n ingestion
```

```
PS C:\Users\rapha\OneDrive\.Data\streaming-pipeline-with-kafka-pinot\kafka-strimzi-k8s-v1beta1> kubectl get kafkatopics -n ingestion
```

NAME	CLUSTER	PARTITIONS	REPLICATION FACTOR	READY
ingest-src-postgresql-customers-json	kafkabroker	1	1	True

Visualizar detalhes de um tópico:

```
kubectl get kafkatopics ingest-src-postgresql-customers-json -o yaml -n ingestion
```

```
PS C:\Users\rappa\OneDrive\OneDrive\Data\streaming-pipeline-with-kafka-pinot\kafka-strimzi-k8s-v1beta1> kubectl get kafkatopics ingest-  
apiVersion: kafka.strimzi.io/v1beta2  
kind: KafkaTopic  
metadata:  
  annotations:  
    kubernetes.io/last-applied-configuration: |  
      {"apiVersion":"kafka.strimzi.io/v1beta2","kind":"KafkaTopic","metadata":{"annotations":{},"labels":{"strimzi.io/cluster":"ingest-  
n"},"spec":{"config":{"cleanup.policy":"delete","retention.ms":604800000,"segment.bytes":1073741824},"partitions":1,"replicas":1}}  
creationTimestamp: "2024-08-19T14:45:53Z"  
finalizers:  
- strimzi.io/topic-operator  
generation: 1  
labels:  
  strimzi.io/cluster: kafkabroker  
name: ingest-src-postgresql-customers-json  
namespace: ingestion  
resourceVersion: "241187"  
uid: 7c3ca25d-a41e-4097-8864-1bb03b86a236  
spec:  
  config:  
    cleanup.policy: delete  
    retention.ms: 604800000  
    segment.bytes: 1073741824  
  partitions: 1  
  replicas: 1  
status:  
  conditions:  
  - lastTransitionTime: "2024-08-19T14:45:53.851063373Z"  
    status: "True"  
    type: Ready  
  observedGeneration: 1  
  topicId: r4R76lSdSmC2z239_Pq_IA  
  topicName: ingest-src-postgresql-customers-json
```

Deletar um tópico (em caso de tentativa):

```
kubectl delete -f .\topic\ingest-src-postgres-customers-json.yml -n ingestion
```

▼ Passo 3 - Acessando e consumindo Tópico

Houve erros durante o consumo dos tópicos:

- o **bootstrap-server** no connect estava configurado para a porta 9093, a mesma foi configurada no broker com TLS, ou seja, precisaria fazer toda parte de certificação para passar por essa porta, porém foi corrigido para o ideal, a porta 9092, onde não utiliza TLS.

```
# BROKER  
listeners:  
- name: plain  
  port: 9092 # porta para conexão sem tls ativo do broker com connect por c  
  type: internal  
  tls: false  
- name: tls  
  port: 9093  
  type: internal  
  tls: true
```

```
# CONNECT  
spec:  
  version: 3.6.0  
  replicas: 1  
  bootstrapServers: kafkabroker-kafka-bootstrap:9092 # porta 9092 com tls desabi
```

- Nomeclatura do topic-prefix, deve considerar a tabela na qual ele fara ingestão, ou seja se colocar 'ingest-src-customers' e a tabela se chamar 'customers' com o prefixo ficará 'ingest-src-customers-customers' causando problema de referencia com o tópico definido antes do connector.

Verificando o nome dos tópicos dentro do servidor, é pra ter o que foi criado anteriormente

```
kubectl exec kafkabroker-kafka-0 -n ingestion -c kafka -- bin/kafka-topics.sh --list
```

Após identificar o tópico, executar o comando de consulta

```
kubectl exec kafkabroker-kafka-0 -n ingestion -c kafka -it -- bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 \
--property print.key=true \
--topic ingest-src-postgresql-customers \
--from-beginning # este pega todo o dado anterior
```

```
PS C:\Users\rappa\OneDrive\Data\streaming-pipeline-with-kafka-pinot\kafka-stimzi-k8s-v1beta1> kubectl exec kafkabroker-kafka-0 -n ingestion -c kafka -it -- bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --property print.key=true --topic ingest-src-postgresql-customers
null
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"id"}, {"type":"string","optional":true,"field":"nome"}, {"type":"string","optional":true,"field":"sexo"}, {"type":"string","optional":true,"field":"endereco"}, {"type":"string","optional":true,"field":"telefone"}, {"type":"string","optional":true,"field":"email"}, {"type":"string","optional":true,"field":"foto"}, {"type":"int32","optional":true,"field":"org.apache.kafka.connect.data.Date","version":1,"field":"nascimento"}, {"type":"string","optional":true,"field":"profissao"}, {"type":"int64","optional":true,"field":"org.apache.kafka.connect.data.Timestamp","version":1,"field":"dt_update"}], "optional":false,"name":"customers"}, "payload":{"id":468,"nome":"Matthew Mendoza","sexo":"M","endereco":"4401 Owens View Suite 022,Port Kenneth, NO 35625","telefone":"(406)655-1523","email":"hollymccoy@example.com","foto":"https://placekitten.com/319/366","nascimento":12835,"profissao":"Journalist, newspaper","dt_update":1724888411821}}
null
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"id"}, {"type":"string","optional":true,"field":"nome"}, {"type":"string","optional":true,"field":"sexo"}, {"type":"string","optional":true,"field":"endereco"}, {"type":"string","optional":true,"field":"telefone"}, {"type":"string","optional":true,"field":"email"}, {"type":"string","optional":true,"field":"foto"}, {"type":"int32","optional":true,"field":"org.apache.kafka.connect.data.Date","version":1,"field":"nascimento"}, {"type":"string","optional":true,"field":"profissao"}, {"type":"int64","optional":true,"field":"org.apache.kafka.connect.data.Timestamp","version":1,"field":"dt_update"}], "optional":false,"name":"customers"}, "payload":{"id":469,"nome":"Anna Griffin","sexo":"F","endereco":"826 James Summit\NKimberlyshire, RI 09638","telefone":"(272)64938","email":"kathleensantiago@example.com","foto":"https://dummyimage.com/111x382","nascimento":18946,"profissao":"Geophysicist/field seismologist","dt_update":1724888415619}}
```

Listando todos os tópicos no broker no server do bootstrap

```
kubectl exec kafkabroker-kafka-0 -n ingestion -c kafka -- bin/kafka-topics.sh --list
```

Neste momento todos os serviços para leitura dos dados e gravação do tópico deverão estar em execução "Running" e status "Ready" completo.

```
PS C:\Users\rappa> kubectl get pods -n ingestion
NAME                                READY   STATUS    RESTARTS   AGE
kafkabroker-connect-0              1/1     Running   0           87m
kafkabroker-entity-operator-77c4587869-jdvb5  2/2     Running   11 (163m ago)  168m
kafkabroker-kafka-0                1/1     Running   0           90m
kafkabroker-zookeeper-0            1/1     Running   0           162m
kafkabroker-zookeeper-1            1/1     Running   0           91m
kafkabroker-zookeeper-2            1/1     Running   0           90m
stimzi-cluster-operator-6948497896-d2wrl  1/1     Running   0           168m
```

```
PS C:\Users\rappa> kubectl get kafkatopics -n ingestion
NAME                                CLUSTER    PARTITIONS  REPLICATION  FACTOR  READY
ingest-src-postgresql-customers-json kafkabroker 1            1            1       True
```

Sink dos dados do Tópico no S3

▼ Passo 1 - Deploy do Connector do S3

Vá até a pasta do Sink, verifique e modifique o arquivo `bucket-s3.yml` conforme necessidade, por exemplo, pastas para particionamento, nome do bucket, localidade e outros, exemplo do arquivo configurado:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: s3-sink-connector-kafka-ed9b207a # nome
  labels:
    strimzi.io/cluster: kafkabroker # cluster
spec:
  class: io.confluent.connect.s3.S3SinkConnector
  tasksMax: 3
  config:
    key.converter: "org.apache.kafka.connect.storage.StringConverter"
    value.converter: "org.apache.kafka.connect.json.JsonConverter"
    key.converter.schemas.enable: "false"
    value.converter.schemas.enable: "false"
    topics: "ingest-src-postgresql-json-customers" # mesmo nome do topic definido
    s3.bucket.name: "kafka-sink-connector" # criado especificamente para o aprendizado
    s3.region: us-east-1
    flush.size: 30000
    format.class: "io.confluent.connect.s3.format.json.JsonFormat"
    schema.generator.class: "io.confluent.connect.storage.hive.schema.DefaultSchemaGenerator"
    partitioner.class: "io.confluent.connect.storage.partitioners.HourlyPartitioner"
    storage.class: "io.confluent.connect.s3.storage.S3Storage"
    topics.dir: "landing-zone/kafka_events"
    aws.access.key.id: "" # inserir sua access key
    aws.secret.access.key: "" # inserir sua secret key
    path.format: "'year'=YYYY/'month'=MM/'day'=DD/'hour'=HH" # particionamento em Meses
    locale: "pt_BR"
    timezone: "America/Sao_Paulo"
    partition.duration.ms: 4600000
```

Em sequência executar o comando para deploy do connector no cluster:

```
kubectl apply -f sink/bucket-s3.yml -n ingestion
```

Se retornar Created, foi feito com **sucesso**

```
PS C:\Users\rapha\OneDrive\Dados\streaming-pipeline-with-kafka-pinot\kafka-strimzi-k8s-v1beta1> kubectl apply -f sink/bucket-s3.yml -n ingestion
kafkaconnector.kafka.strimzi.io/s3-sink-connector-kafka-ed9b207a created
```

Validar com o comando

```
kubect1 get kafkaconnectors -n ingestion
```

No caso retornou os connectors, tanto o Source como o Sink, além de um perdido que foi criado durante um erro que foi corrigido

NAME	CLUSTER	CONNECTOR CLASS	MAX TASKS	READY
ingest-src-postgresql-customers-json	kafkabroker	io.confluent.connect.jdbc.JdbcSourceConnector	1	
postgres-source-connector-customers	kafkabroker	io.confluent.connect.jdbc.JdbcSourceConnector	1	True
s3-sink-connector-kafka-ed9b207a	kafkabroker	io.confluent.connect.s3.S3SinkConnector	3	True

Olhando as configurações

```
kubect1 get kafkaconnectors s3-sink-connector-kafka-ed9b207a -n ingestion -o yaml
```

```
PS C:\Users\rapha\OneDrive\Data\streaming-pipeline-with-kafka-pinot\kafka-strimzi-k8s-v1beta1> kubectl get kafkaconnectors
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"kafka.strimzi.io/v1beta2","kind":"KafkaConnector","metadata":{"annotations":{},"labels":{"strimzi.io/c
n"},"spec":{"class":"io.confluent.connect.s3.S3SinkConnector","config":{"aws.access.key.id":"AKIAWHMQZWYECG3KXA73","aws.secr
class":"io.confluent.connect.s3.format.json.JsonFormat","key.converter":"org.apache.kafka.connect.storage.StringConverter","
0","partitioner.class":"io.confluent.connect.storage.partitioner.HourlyPartitioner","path.format":"'year'=YYYY/'month'=MM/'da
ema.generator.class":"'io.confluent.connect.storage.hive.schema.DefaultSchemaGenerator","storage.class":"io.confluent.connect
-json-customers","topics.dir":"landing-zone/kafka_events","value.converter":"org.apache.kafka.connect.json.JsonConverter"},"
creationTimestamp":"2024-08-19T20:44:28Z"
generation: 1
labels:
  strimzi.io/cluster: kafkabroker
name: s3-sink-connector-kafka-ed9b207a
namespace: ingestion
resourceVersion: "314312"
uid: 26f8b817-dbab-4dd5-a060-8684ab9cdb4e
spec:
  class: io.confluent.connect.s3.S3SinkConnector
  config:
    aws.access.key.id: [REDACTED]
    aws.secret.access.key: [REDACTED]
    flush.size: 30000
    format.class: io.confluent.connect.s3.format.json.JsonFormat
    key.converter: org.apache.kafka.connect.storage.StringConverter
    key.converter.schemas.enable: "false"
    locale: pt_BR
    partition.duration.ms: 4600000
    partitioner.class: io.confluent.connect.storage.partitioner.HourlyPartitioner
    path.format: "'year'=YYYY/'month'=MM/'day'=DD/'hour'=HH"
    s3.bucket.name: kafka-sink-connector
    s3.region: us-east-1
    schema.generator.class: io.confluent.connect.storage.hive.schema.DefaultSchemaGenerator
    storage.class: io.confluent.connect.s3.storage.S3Storage
    timezone: America/Sao_Paulo
    topics: ingest-src-postgresql-json-customers
    topics.dir: landing-zone/kafka_events
    value.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter.schemas.enable: "false"
```

Workers e nome do Sink e Topic


```

conditions:
- lastTransitionTime: "2024-08-19T20:45:46.961200892Z"
  status: "True"
  type: Ready
connectorStatus:
  connector:
    state: RUNNING
    worker_id: kafkabroker-connect-0.kafkabroker-connect.ingestion.svc:8083
  name: s3-sink-connector-kafka-ed9b207a
  tasks:
  - id: 0
    state: RUNNING
    worker_id: kafkabroker-connect-0.kafkabroker-connect.ingestion.svc:8083
  - id: 1
    state: RUNNING
    worker_id: kafkabroker-connect-0.kafkabroker-connect.ingestion.svc:8083
  - id: 2
    state: RUNNING
    worker_id: kafkabroker-connect-0.kafkabroker-connect.ingestion.svc:8083
  type: sink
observedGeneration: 1
tasksMax: 3
topics:
- ingest-src-postgresql-json-customers

```

Sucesso no Commit, podemos verificar tanto no bucket quanto no log do connect pela seguinte linha de log

```
2024-08-20 16:14:25.622 INFO [s3-sink-connector-kafka-ed9b207a|task-1] Files committed to S3.
```

Deploy do KsqlDB

Ksql é uma ferramenta para consulta de dados em streaming, é extremamente eficiente.

▼ Passo 1 - Deployment do KsqlDB

Navegue até a pasta do ksqldb

```
cd PS C:\Users\user\OneDrive\.Data\streaming\ksqldb-server>
```

Em sequência execute o apply do kubectl para a pasta, ele irá executar todos os yml.

```
kubectl apply -f repository/deployment -n processing # note que agora é outro namespace
```

```
deployment.apps/ksqldb-server created
service/ksqldb-headless created
service/ksqldb-server created
```

Verifique nos pods o status do container

```
kubectl get pods -n ingestion
```

NAME	READY	STATUS	RESTARTS	AGE
ksqldb-server-c6d477d84-bvpxw	1/1	Running	0	42s

Running!

▼ Passo 2 - Acessando o terminal do Ksql

```
kubectl exec -it ksqldb-server-c6d477d84-bvpxw -n processing -- bash ksql
```

```
PS C:\Users\rappa\OneDrive\OneDrive\streaming-pipeline-with-kafka-pinot\ksqldb-server> kubectl exec -it ksqldb-server-c6d477d84-bvpxw -n processing -- bash ksql
OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in version 9.0 and will likely be removed in a future release.

=====
=                                     =
=  KSQL                             =
=                                     =
=  Event Streaming Database purpose-built =
=  for stream processing apps          =
=====

Copyright 2017-2020 Confluent Inc.

CLI v0.12.0, Server v0.12.0 located at http://localhost:8088
Server Status: RUNNING

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!
```

Visualizando os topics que podem ser consumidos

```
show topics;
```

```
ksql> show topics;
```

Kafka Topic	Partitions	Partition Replicas
connect-cluster-configs	1	1
connect-cluster-offsets	25	1
connect-cluster-status	5	1
default_ksql_processing_log	1	1
ingest-src-postgresql-customers	9	1

▼ Passo 3 - Criando e consultando uma Stream do Ksql

Utiliza linguagem sql com as particularidades do ksql

```
CREATE OR REPLACE STREAM ksql_stream_customers_json
( -- através da mensagem estrutura o payload com base no struct abaixo
  "payload" STRUCT<"id" BIGINT,
    "nome" VARCHAR,
    "sexo" VARCHAR,
    "endereço" VARCHAR,
    "telefone" VARCHAR,
    "email" VARCHAR,
    "foto" VARCHAR,
```

```

        "nascimento" VARCHAR,
        "profissao" VARCHAR,
        "dt_update" BIGINT,
        "messagetopic" VARCHAR,
        "messagesource" VARCHAR>
    )
    WITH (KAFKA_TOPIC='src-postgres-customers-json', VALUE_FORMAT='JSON');

```

Executar o código no ksql, retorno:

```

Message
-----
Stream created
-----

```

Execute o comando para exibir os streams criado

```
show streams;
```

Stream Name	Kafka Topic	Format
KSQL_PROCESSING_LOG	default_ksql_processing_log	JSON
KSQL_STREAM_CUSTOMERS_JSON	ingest-src-postgresql-customers	JSON

Criando de fato stream como tabela com prefixo → Output

```

CREATE OR REPLACE STREAM output_ksqldb_stream_customers_json
WITH (KAFKA_TOPIC='output-ksqldb-stream-customers-json', PARTITIONS=3, VALUE_FORMAT=
AS
SELECT
AS_VALUE("payload"->"id") as "business_key",
"payload"->"id" as "id",
"payload"->"nome",
"payload"->"sexo",
"payload"->"endereco",
"payload"->"telefone",
"payload"->"dt_update"
FROM ksql_stream_customers_json
EMIT CHANGES;

```

```

Message
-----
Created query with ID CSAS_OUTPUT_KSQLDB_STREAM_CUSTOMERS_JSON_0
-----

```

Exibe as queries que estão executando no momento

```
show queries;
```

```
Query ID | Query Type | Status | Sink Name | Sink Kafka Topic | Query String
-----|-----|-----|-----|-----|-----
CSAS_OUTPUT_KSQLDB_STREAM_CUSTOMERS_JSON_0 | PERSISTENT | RUNNING:1 | OUTPUT_KSQLDB_STREAM_CUSTOMERS_JSON | output-ksqldb-stream-customers-json | CREATE OR REPL
ITH (KAFKA_TOPIC='output-ksqldb-stream-customers-json', PARTITIONS=3, REPLICAS=1, VALUE_FORMAT='JSON') AS SELECT AS_VALUE(KSQL_STREAM_CUSTOMERS_JSON.'payload'->
SON.'payload'->'id' 'id', KSQL_STREAM_CUSTOMERS_JSON.'payload'->'nome' 'nome', KSQL_STREAM_CUSTOMERS_JSON.'payload'->'sexo' 'sexo', KSQL_STREAM_CUSTOMERS_J
EAM_CUSTOMERS_JSON.'payload'->'telefone' 'telefone', KSQL_STREAM_CUSTOMERS_JSON.'payload'->'dt_update' 'dt_update' FROM KSQL_STREAM_CUSTOMERS_JSON KSQL_STREAM_
For detailed information on a Query run: EXPLAIN <Query ID>;
```

Novamente se utilizar o `show streams` irá retornar o output entre as queries

Stream Name	Kafka Topic	Format
KSQL_PROCESSING_LOG	default_ksql_processing_log	JSON
KSQL_STREAM_CUSTOMERS_JSON	ingest-src-postgresql-customers	JSON
OUTPUT_KSQLDB_STREAM_CUSTOMERS_JSON	output-ksqldb-stream-customers-json	JSON

Executando as queries

Há uma opção para ler os dados desde o início ou somente o mais recente com o seguinte comando

```
SET 'auto.offset.reset' = 'latest' # recente
SET 'auto.offset.reset' = 'earliest' # desde o início
```

Contagem de ids por sexo

```
-- QUERY 1
SELECT
  "sexo",
  count("business_key") AS "qtd_por_sexo"
FROM output_ksqldb_stream_customers_json
GROUP BY "sexo"
EMIT CHANGES;

-- QUERY 2

SELECT
  "id",
  "nome",
  "endereco",
  "telefone",
  "dt_update"
```

```
FROM output_ksqldb_stream_customers_json
EMIT CHANGES;
```

```
ksql> SELECT
>"sexo",
>count("business_key") AS "qtd_por_sexo"
>FROM output_ksqldb_stream_customers_json
>GROUP BY "sexo"
>EMIT CHANGES;
```

sexo	qtd_por_sexo
F	2
M	1
F	3
M	4
F	6
F	9
F	11

Deploy do Apache Pinot

Utilizando o helm iremos fazer o deploy do Apache Pinot, nosso data warehouse para streaming.

▼ Passo 1 - Deploy do Apache Pinot

Dentro da pasta pinot executar script do arquivo `deploy-pinot.sh`

```
#!/usr/bin/env bash

helm repo add pinot https://raw.githubusercontent.com/apache/pinot/master/helm

# kubectl create ns pinot-quickstart # caso o namespace datastorage nao tenha sido criado

# -n namespace onde será feito deploy no kube
helm install pinot pinot/pinot \
  -n datastorage \
  --set cluster.name=pinot \
  --set server.replicaCount=2
```

Caso esteja utilizando windows, configurar o arquivo para .bat e remover a quebra de linhas.

```
C:\Users\rappa\OneDrive\OneDrive\Data\streaming-pipeline-with-kafka-pinot\incubator-
datastorage --set cluster.name=pinot --set server.replicaCount=2
NAME: pinot
LAST DEPLOYED: Thu Aug 22 22:25:34 2024
NAMESPACE: datastorage
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Use o comando para visualizar os pods em execução

```
kubectl get pods -n datastorage
```


```
PS C:\Users\rapha\OneDrive\.Data\streaming-pipeline-with-kafka-pinot\incubator-pinot> kubectl get pods -n datastorage
NAME                                READY   STATUS    RESTARTS   AGE
pinot-broker-0                      0/1     Running   0           95s
pinot-controller-0                  0/1     Pending   0           95s
pinot-minion-stateless-8f6455f89-s8h6c 0/1     Running   0           95s
pinot-server-0                      0/1     Pending   0           95s
pinot-server-1                      0/1     Pending   0           95s
pinot-zookeeper-0                  0/1     Pending   0           95s
```

Considerações Finais

Nessa ultima etapa o Pinot demonstrou instabilidade ao não conseguir estabelecer conexão com o Zookeeper retornando "Connection Refused" ao investigar os logs do Broker do Pinot, foi modificado o yml para deploy, remapeado portas e feito busca na web por soluções, porém nenhum resolveu o conflito.

Caso alguém encontre uma solução para o caso favor entrar em contato, será testado e atualizado a documentação, pelo contrário, finalizamos essa documentação com a Pipeline semi completa, sendo o Pinot uma ferramenta adicional e não obrigatória para o aprendizado de Data Streaming com Kafka.

Agradecimentos a XP Educação pelo bootcamp de Engenharia de Dados da e ao professor Carlos - [github:@carlosbtech](https://github.com/carlosbtech), no github dele é possível encontrar o material base desta documentação de 3 anos atrás. Lembrando que esta foi modificada para a versão v1beta2 e diversas configurações foram melhoradas.

 Delete o cluster com o seguinte comando

```
#!/usr/bin/env bash
```

```
eksctl delete cluster --name kafkak8s --region us-east-1
```

REM Windows

```
eksctl delete cluster --name kafkak8s --region us-east-1
```