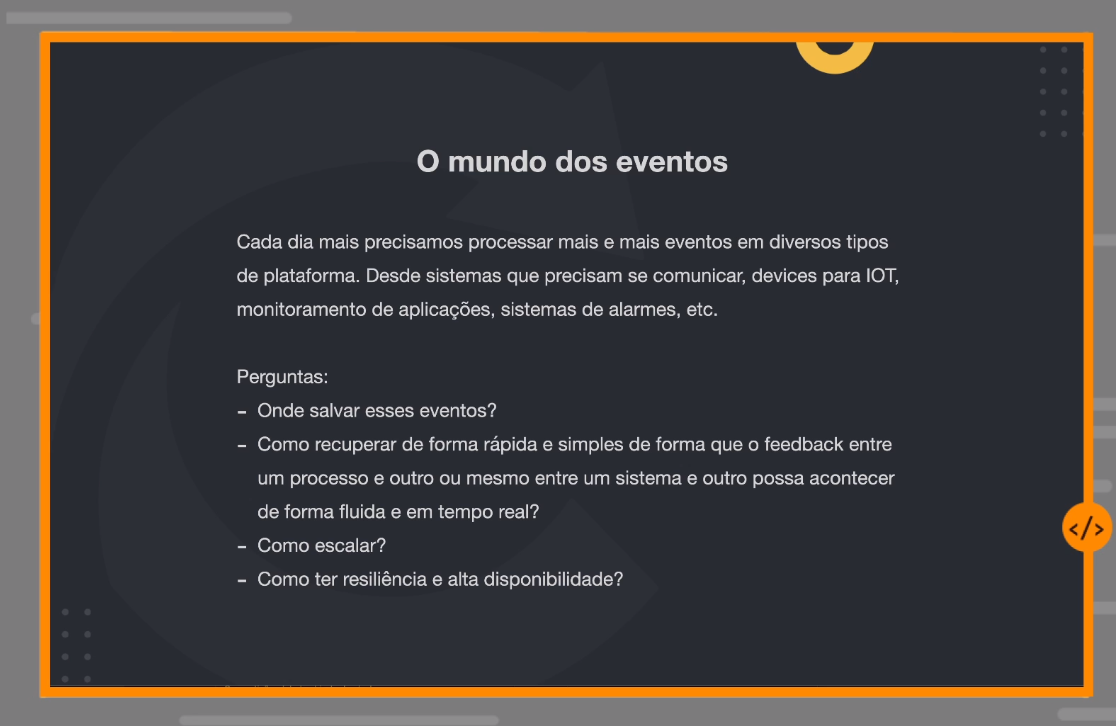
Apache Kafka

Plataforma distribuída de **streaming de eventos** open-source que é utilizada por milhares de empresas para uma alta performance em papeline de dados, stream de analytics, integração de dados e aplicações de missão crítica.

**streaming de eventos =>**  qualquer coisa que produza dados





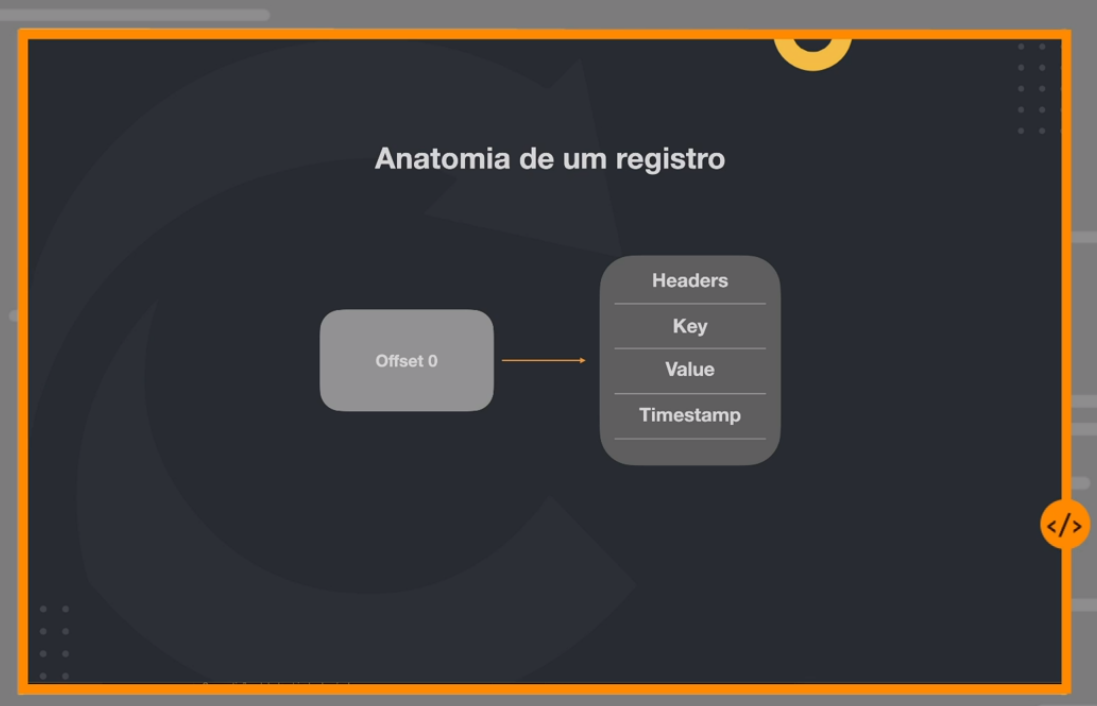


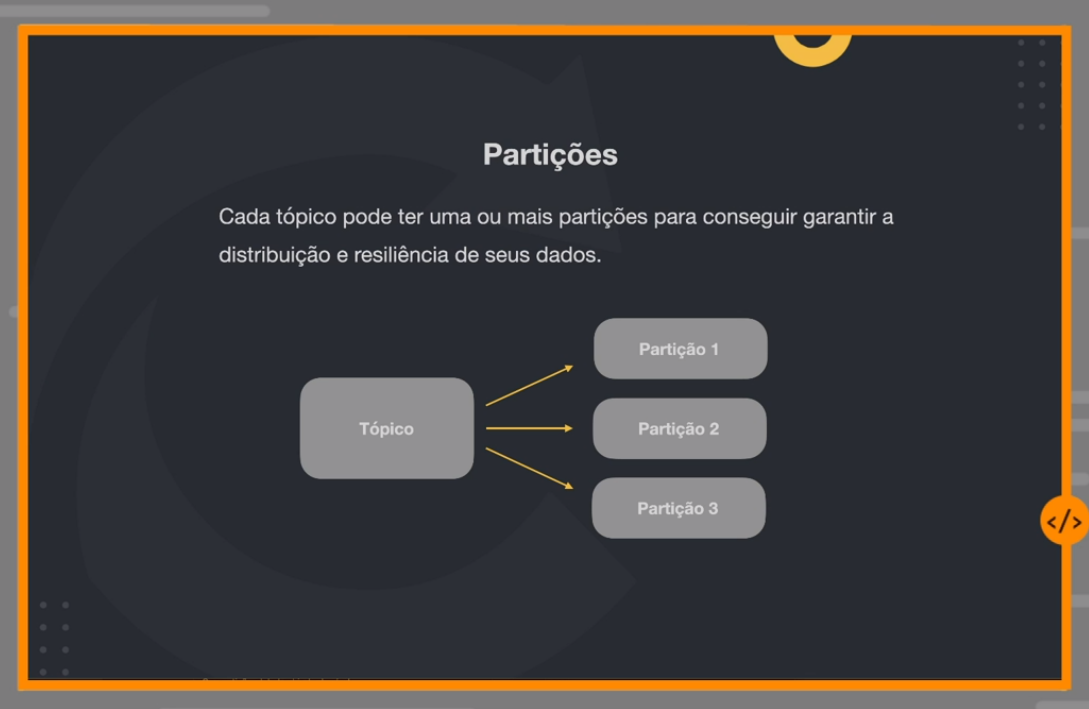
**Topic**

Tópico é o canal de comunicação responsável por receber e disponibilizar os dados enviados para o Kafka. (muito “parecido” com fila), em resumo o tópico é o lugar que eu mando as mensagens e os consumidores tbm leêm esses tópicos. Diferente da fila, que quando a mensagem é lida ele é removida, o tópic não remove a mensagem depois de lida, pois vários consumidores podem ler a mesma mensagem.

* As mesagens são armazenadas em **partições**, ex: 0,1,2,3, (parecido com um List) cada posição pode ser chamada de “offset”.
* Os consumidores podem ler essas mensagens em tempos diferentes, um pode ler hoje outro pode ler amanhã, etc.
* AS mensagens podem ser relidas, pelo mesmo consumidor ou por um distinto.

**Cada partição “offset” é comporto por 4 partes:**





O Tópico sempre separa as mensagens em partições, isso para melhorar a performance no processamento como também para garantir que se uma máquina parar por exemplo, tenham outras com mensagens. As mensagens podem cair em partições distintas, no momento de escalar (dividir as partições) nós falamos quantas partições queremos, isso de forma simples é o escalonamento.

Um “efeito colateral” seria organizar a ordem das mensagens que são distribuídas em partições. Para isso utilizamos as “keys”, onde as mensagens tem q estar dentro da mesma partição.



Se em todas as mensagens que eu mandar eu setar a mesma “key”, o Kafka vai entender e colocá-las na mesma partição, resolvendo assim o problema da ordem de distribuição das mensagens.

Quando eu não preciso ter ordem, não preciso informar nenhuma key.

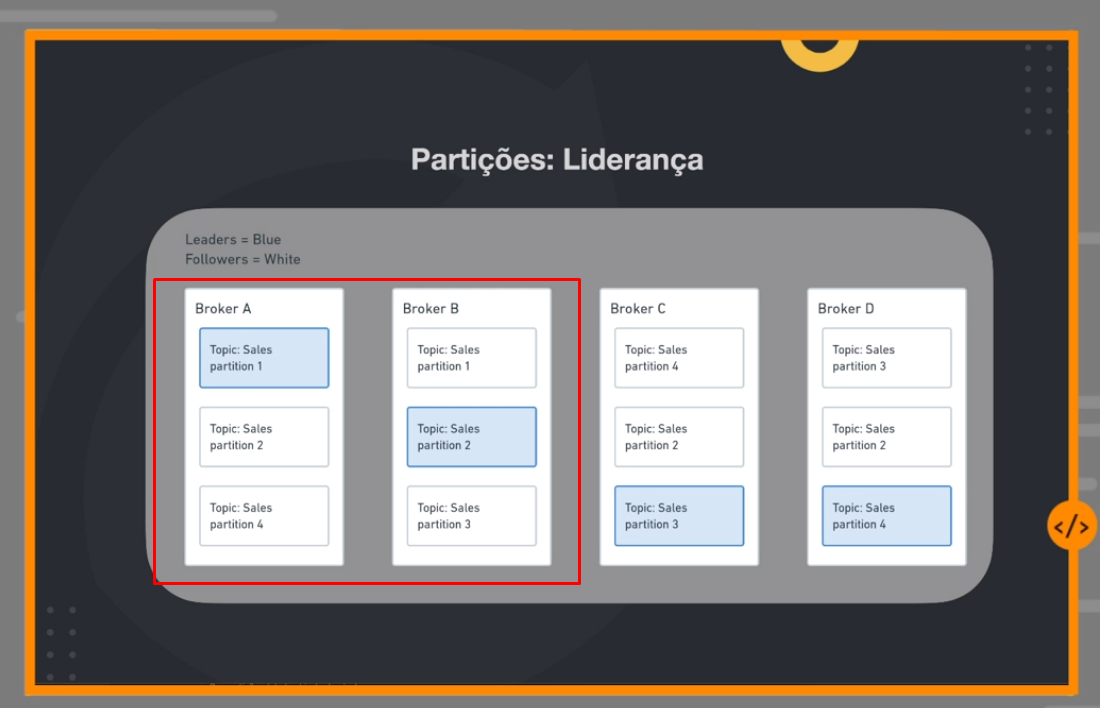
**Como garantimos que se uma partição cair eu vou conseguir deixar minhas mensagens ainda disponíveis?**



O “Replication Factor” faz a replicação das nossas partições em broker (como se fossem máquinas), nesse caso falamos para duplicar nossas partições em “2”, o Kafka criou 4 broker e em cada uma colocou nossas partições distribuídas. Isso garante a resiliência dos das mensagens.

Um ponto a observar que quando maior o replication fator, mais espaço em disco vc vai precisar ter, mais vai consumir do seu broker, precisa-se entender a quantidade correta de replicatiom que geralmente no mercado é utilizada apenas 2 replicações.

**Partições Lider**



As partições Líder são as destacadfas em azul, perceba que os tópicos estão distyribuidos nas 4 partições, mas o Kafka vai ler apenas das partições Líder, ou seja, das partições em azul que estão destacadas na imagem nos brokes A e B.



Na imagem acima, o Broker A caiu onde estava a partição 1 de Sales, automaticamente o Kafka vai buscar e ler onde está a partição 1, que nesse caso está no broker B, nesse omento o broker B passa a ser o “Leader” que é sempre o broker que está sendo lido de forma principal e é também onde sempre as mensagens são entregues, no Broker Leader.

Sendo assim, entendemos que o “Leader” é sempre o broker principal onde se recebe e se ler as mensagens, o “Follower” são os broker que fazem o backup das informações e poderão ser utilizados caso o Leader caia por algum motivo.

**Producer: Garantias de Entrega**

Quando as mensagens são entregues, elas setam parâmetros, um deles é o ACK (acknologi).

Esse parâmetro pode ser qualificado da seguinte forma:

Ack 0 (none) => a mensagens é entregue ao Leader e o producer não recebe uma confirmação da entrega. O Kafka não retorna dizendo que está OK, que gravou a mensagens. Nesse processo o Kafka consegue processar mais mensagens pois não se preocupa em dar confirmação. Contras: o producer pode perder mensagens, pensado que as mesmas foram entregues sem mesmo ter a confirmação, a depender do negócio, pode ser utilizado esse parâmetro que não se importa em perder mensagens.

Ack 1 (Leader) => a mensagem é entregue e é enviada uma mensagens confirmando a entrega a producer. Contras: o processamento fica mais lento pois preciso mandar a confirmação, e em situações mensagens podem ser perdidas, pois o broker pode cair antes de replicar a mensagens para os follows, pois o mesmo estava ainda retornado ao producer que a mensagem foi entregue.

Ack-1 (All) => nessa opção, a menagem é enviada, o leader vai replicar as mensagens nos follows, e só depois vai avisar ao producer que a mensagem foi entregue. Nessa caso, teremos a garantia que a mensagem foi entregue, salva e replicada. Contras: o processamento vai ficar mais lento por contra do processo de replicação, porém a garantia de entrega e replicação é concreta.



Existem outros formatos tambem:

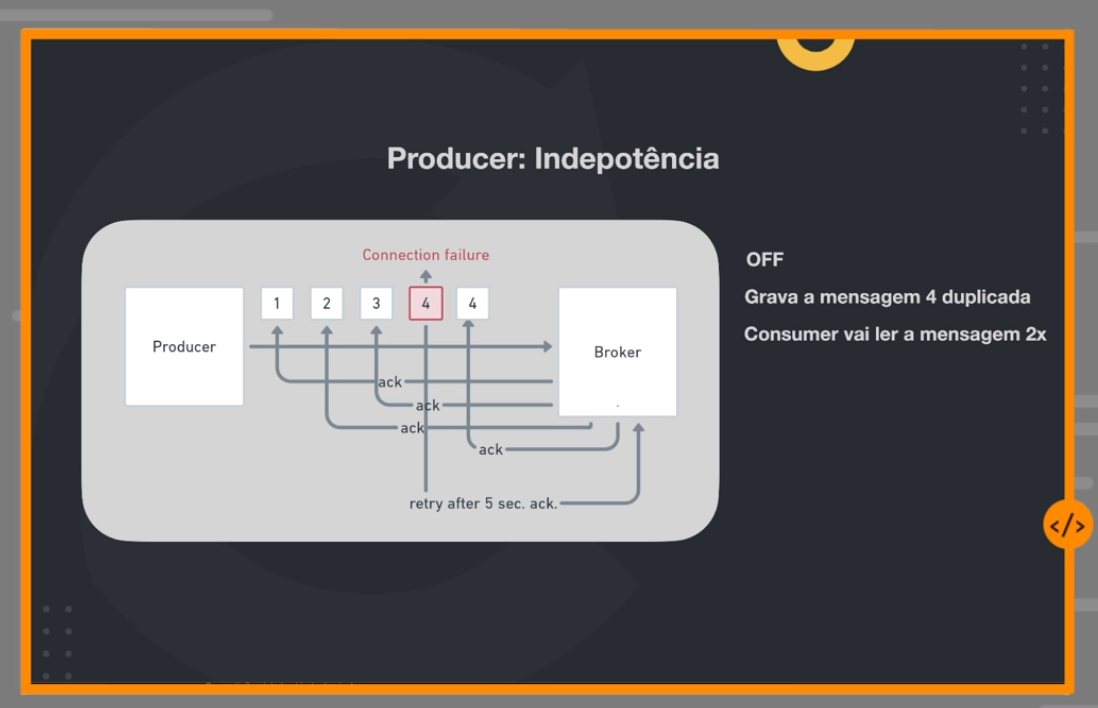
**At most once**: Melhor performance. Pode perder algumas mensagens

**At least once**: Performance moderada. Pode duplicar mensagens.

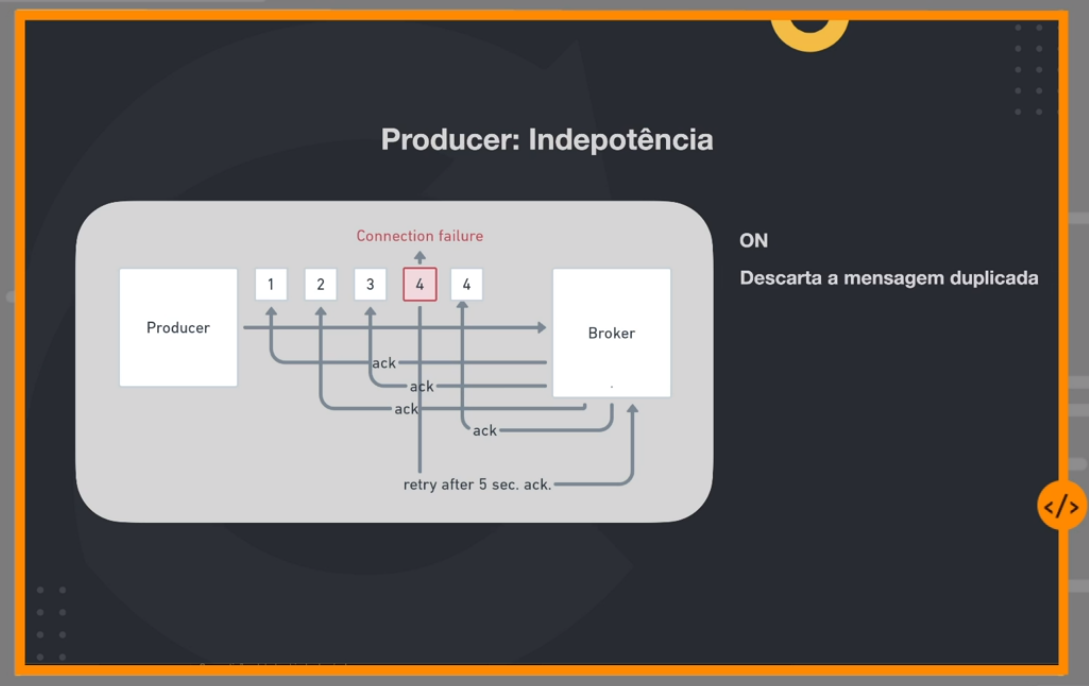
**Exacly once**: Pior performance. Exatamente uma vez (não duplica nem perde mensagens);

**Producer: Indepotência**

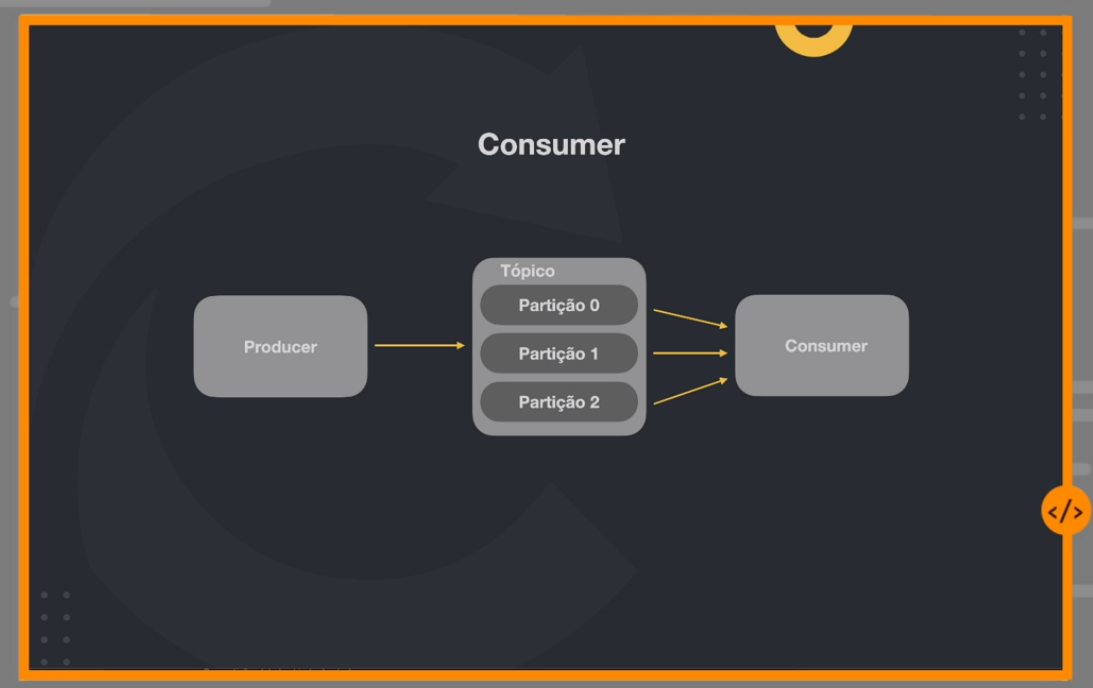
Quando o Produto Indepotente estiver “**OFF**” Pode existir situações, como uma queda de rede por exemplo, que pode causar a duplicação de mensagens. Nesse caso o producer pode mandar a menagens duas vezes, não é o Kafka que vai processar duas vezes tá, mas sim o producer que mandou devido a essa queda de rede vamos dizer assim.



Quando o Produto Indepotente estiver “**ON**” o KAFKA vai identificar a duplicação e vai descartar a mensagem duplicada.

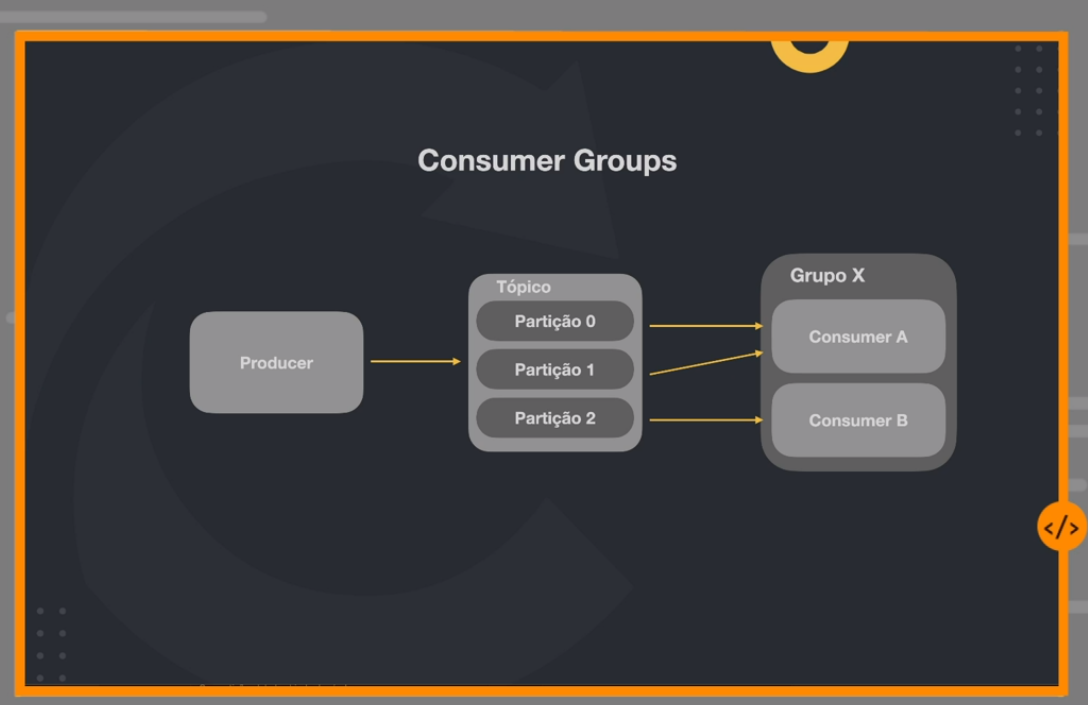


**Consumer**



**Consumer Groups**

Posso ter vários consumidores (mesmo software rodando em máquinas ou processo diferentes) lendo o mesmo Tópico, nesse caso os consumidores são alocados em um mesmo grupo e conseguem ler as informações de partições diferentes, isso é gerenciado pelo KAFKA.



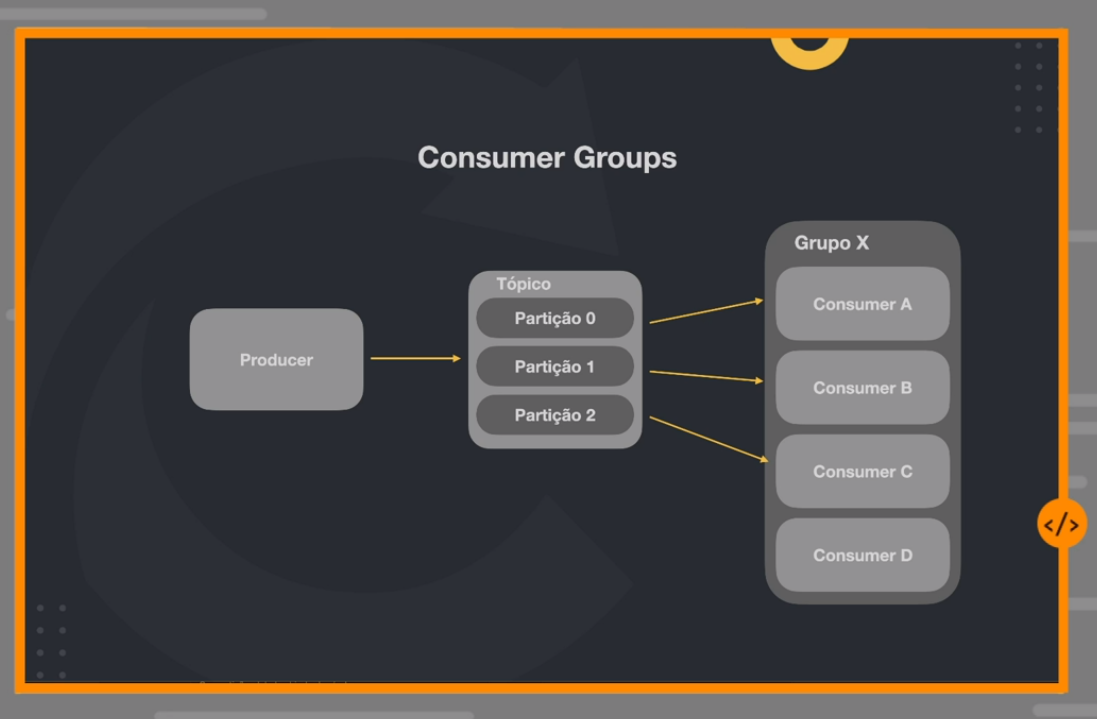
No caso acima, tenho dois consumers que estão lendo de partições distribuídas pelo Kafka, nesse caso se eu tivesse um Consumer C por exemplo, o mesmo estaria lendo lendo as 3 partições (0,1,2), isso porque ele não estaria no grupo X que foi criado.

Em resumo, se não tiver grupo o Kafka vai identificar que o conumuer é um grupo isolado, para que isso não aconteça, podemos configurar o Kafka para dizer que determinado consumer faz parte de determinado grupo e assim por diante.

OBS importante, no melhor dos mundos devemos ter sempre 1 consumer por partição, nunca poderemos ter mais consumer do que partições, pois o KAFKA não permite que dois consumers que estão no mesmo grupo leiam uma mesma partição. Se tiver mais consumer do que partição no mesmo grupo, vai ter consumer que ira ficar IDLE, ou seja, parado.

Caso esses cosumers estejam em **grupos diferentes**, ai sim, eles podem ler a mesma partição.

O melhor dos mundo sempre é ter uma mesma quantidade para Consumers e Partições.



**Iniciando no KAFKA**

Para entrar no container Kafka via linha de comando:  **docker exec -it (nome cointainer) bash**

**Comados KAFKA via linha de comando:**

Kafka- => mostra os comandos disponíveis

Kafka-topics => mostra os comandos disponíveis para tópicos

kafka-topics --create --topic=teste --bootstrap-server=localhost:9092 --partitions=3 => comando para criar um tópico com todos os recursos obrigatórios (nome, bootstrap...)

kafka-topics --list --bootstrap-server=localhost:9092 => lista os tópicos criados

kafka-console-consumer --bootstrap-server=localhost:9092 --topic=teste => para consumir um tópico em específico.

kafka-console-producer --bootstrap-server=localhost:9092 --topic=teste => para enviar mensagems para algum tópico especificado.

**OBS: Um teste muito simples pode ser feito criado o tópico e em um shel executar o cuncumer e em outro shel o producer, fazendo isso vamos ver as mensagens sendo apresentadas.**

kafka-console-consumer --bootstrap-server=localhost:9092 --topic=teste –from-beginning => para consumir todas as mensagens enviadas, inclusive as que por algum motivo não foram consumidas (para fazer esste teste basta sair do shell de consumer e enviar mensagens pelo producer e depois executar o comando acima). AS mensagens podem aparecer fora de ordem, isso porque quando nos enviamos as mesmas, elas vão para partições diferentes, logo quando o kafka as pega através do comando acima, ele vai pegando das partições e mostrando.

kafka-console-consumer --bootstrap-server=localhost:9092 --topic=teste --group=x => Comando para consumir um tópico e incluir o consumer em um grupo.

kafka-consumer-groups --bootstrap-server=localhost:9092 --group=x –describe => comando para descrever o grupo criado, poderemos ver todos os detalhes do que esta ocorrendo no grupo,inclusive quais partições nossos consumidores estão consumindo.