Nome: Raimundo Wladimir de Oliveira Moreira
CPF: 392.694.341-68

Desafio Analista de Dados – Big Data – Delta Lake

1. Os scripts de processamento do Spark estão demorando muito para serem concluídos. Como você traçaria o perfil e identificaria gargalos no processamento do PySpark e quais otimizações específicas você aplicaria para melhorar o desempenho?
   R:

   Para identificar gargalos no processamento do *PySpark*, pode utilizar o *Spark User Interface*, que fornece informações detalhadas sobre o desempenho e a utilização de recursos de cada estágio do script *PySpark*. Além disso, você pode ser usado o *Spark's event logging* para registrar eventos de execução que podem ser analisados posteriormente para identificar gargalos. As otimizações específicas que podemos aplicar incluem:
   - **Ajustar a quantidade de memória alocada para o Spark**: Isso pode ser feito ajustando a configuração *spark.executor.memory*.
   - **Novo Particionamento de dados**: Isso pode ajudar a evitar o problema de dados desequilibrados em diferentes partições.
   - **Persistência de *RDDs/DataFrames* frequentemente usados**: Isso pode evitar o um novo processamento de *RDDs/DataFrames.*
   - **Evitar operações que causam embaralhamento**: Operações como *groupByKey* podem causar embaralhamento de dados e são geralmente mais lentas.

2. Descreva as principais definições e configurações no Delta Lake que podem afetar o desempenho de gravação e leitura. Que estratégias você empregaria para otimizar as transações do Delta Lake e melhorar a eficiência geral?
   R:

   Otimização do Delta Lake: As principais configurações no Delta Lake que podem afetar o desempenho de gravação e leitura incluem o tamanho do bloco, a compactação, o particionamento e a indexação. Para otimizar as transações do Delta Lake, consideramos as seguintes estratégias:
   - Otimizar a tabela para compactar pequenos arquivos em arquivos maiores.
   - Usar a cláusula *ZORDER* para reorganizar os dados com base em uma coluna específica.

   Resumindo: no Delta Lake, as principais configurações que podem afetar o desempenho de gravação e leitura incluem:

- **Tamanho do bloco**: Um tamanho de bloco maior pode melhorar o desempenho de leitura, mas também pode consumir mais memória.
- **Compactação**: A compactação pode reduzir o tamanho dos arquivos, mas também pode aumentar o tempo de leitura.
- **Particionamento**: O particionamento pode melhorar significativamente o desempenho de leitura ao reduzir a quantidade de dados que precisam ser lidos.
- **Índice**: A criação de índices em colunas frequentemente consultadas pode acelerar as consultas.
- **Otimizar a tabela**: Isso irá compactar pequenos arquivos em arquivos maiores, o que pode melhorar o desempenho de leitura.
- ***ZORDER***: Isso irá reorganizar os dados com base em uma coluna específica, o que pode melhorar o desempenho de leitura para consultas que usam como filtro essa coluna.

3. Você consideraria usar o particionamento em uma Delta Table? Porque, quando e como? Forneça um exemplo.
   R:

   Podemos ponderar o uso do particionamento em uma Delta Table. O particionamento é útil quando você tem uma coluna que é frequentemente usada em cláusulas WHERE do SQL. Ao particionar por essa coluna, você pode reduzir significativamente a quantidade de dados que precisam ser lidos.
   Por exemplo, se você tiver uma tabela de vendas e frequentemente executar consultas para um ano específico, você pode particionar a tabela por ano. Isso significa que, para uma consulta para um ano específico, apenas os dados para esse ano serão lidos.

4. Os usuários estão enfrentando um desempenho de consulta lento ao usar o Presto. Como você diagnosticaria e melhoraria o desempenho das consultas do Presto? Mencione configurações e otimizações específicas do Presto que você implementaria.
   R:

   Para melhorar o desempenho das consultas do *Presto*, podemos começar verificando as estatísticas de execução da consulta no *Presto User Interface*. Isso pode ajudar a identificar quais partes da consulta estão demorando mais. Algumas otimizações específicas do Presto que você pode implementar incluem:
   - **Ajustar o tamanho da memória**: Aumentar a memória disponível para o *Presto* pode ajudar a melhorar o desempenho.

- **Habilitar o *join* distribuído**: Isso pode melhorar o desempenho de consultas que envolvem operações de *join*.
- **Usar o formato de dados certo**: O uso de formatos de dados colunares, como *Parquet, AGRO* ou *ORC*, pode melhorar o desempenho de leitura.

Presto é uma ferramenta de código aberto para consultas distribuídas que permite consultas analíticas em fontes de dados heterogêneas. **Presto** é projetado para consultas analíticas, com ênfase em baixa latência. Ele suporta a linguagem de consulta SQL padrão ANSI, incluindo funções complexas, como junções e agregações.

Os principais benefícios do Presto incluem:

- **Consultas interativas**: Presto foi projetado para consultas que exigem baixa latência, tornando-o adequado para consultas interativas.
- **Fontes de dados heterogêneas**: Presto pode consultar dados onde eles estão armazenados, sem a necessidade de mover dados. Ele suporta uma variedade de fontes de dados, incluindo Hive, Cassandra, Kafka, entre outros.
- **Escalabilidade**: Presto é um sistema distribuído que pode escalar para processar petabytes de dados.
- **SQL padrão**: Presto suporta a linguagem de consulta SQL padrão ANSI, permitindo que os usuários usem sintaxe familiar para consultar dados.

5. O pipeline precisa lidar com um volume crescente de dados. Descreva as mudanças que você proporia na arquitetura para garantir a escalabilidade, tanto em termos de processamento do Spark quanto de armazenamento de dados no HDFS.

R:

Para lidar com um volume crescente de dados, vpodemos considerar as seguintes mudanças na arquitetura:

- **Escalabilidade horizontal**: Adicionar mais nós ao seu cluster Spark e HDFS pode ajudar a lidar com um volume maior de dados.
- **Particionamento de dados**: Particionar seus dados em HDFS pode melhorar o desempenho de leitura. Preferencialmente com arquivos maiores que 128 MB.
- **Aumentar a paralelização**: No *Spark*, aumentar o número de partições pode permitir um processamento mais paralelo.

6. A análise em tempo real é um requisito crítico. Como você equilibraria as compensações entre o processamento em tempo real e o processamento em

lote no pipeline? Forneça estratégias e tecnologias específicas que você empregaria.
R:
Para equilibrar as compensações entre o processamento em tempo real e o processamento em lote, podemo considerar uma arquitetura Lambda. Nesta arquitetura, teriamos dois pipelines de processamento - um para processamento em tempo real e outro para processamento em lote. Tecnologias específicas que você pode usar incluem *Spark Streaming* ou *Flink* para processamento em tempo real e o Airflow o Nifi para processamento em lote.

7. Com suas palavras, como é o funcionamento do Apache Kafka.
   R:
   O Apache Kafka é um sistema de streaming de dados distribuído (Mensageria). Ele permite que publique e assine fluxos de registros de dados, semelhante a uma fila de mensagens. O Kafka armazena fluxos de registros em categorias chamadas tópicos e cada registro consiste em uma chave, um valor e um carimbo de data/hora. Organizando uma fila de mensagens ordenadas e registradas.

8. Projete uma estratégia avançada de monitoramento e registro para identificar e resolver gargalos de desempenho em tempo real. Quais ferramentas e métricas você usaria para monitorar Spark, Delta Lake e Presto? Como você configuraria alertas em tempo real com base em métricas de desempenho?
   R:
   Monitoramento e registro: Para monitorar *Spark, Delta Lake e Presto*, pode-se usar ferramentas como *Grafana* ou *Prometheus*. Você pode configurar alertas em tempo real com base em métricas de desempenho usando ferramentas como *Alertmanager*.
   Grafana é uma plataforma de código aberto para observabilidade. Ele permite que você crie, explore e compartilhe todos os seus dados por meio de painéis intuitivos e flexíveis. O *Grafana* pode ser usado localmente.
   Algumas das principais características do Grafana incluem:
   - Criação e gerenciamento de painéis operacionais para seus dados de várias fontes, como Prometheus, Logs, Traces, Metrics e muito mais.
   - Observabilidade centralizada.
   - Suporte para várias fontes de dados.
   - Capacidade de criar alertas com base em métricas.

   Prometheus é uma ferramenta de monitoramento e alerta de sistemas de código aberto. Ele coleta e armazena as métricas como dados de séries temporais, ou seja, informações de métricas são armazenadas com o carimbo de data/hora em que foram registradas, juntamente com pares de valores-chave opcionais chamados labels.
   Algumas das principais características do Prometheus incluem:

- Um modelo de dados multidimensional com dados de séries temporais identificados por nome de métrica e pares de chave/valor.
- PromQL, uma linguagem de consulta flexível para aproveitar essa dimensionalidade.
- Nenhuma dependência de armazenamento distribuído; os nós do servidor único são autônomos.
- A coleta de séries temporais ocorre via um modelo de pull sobre HTTP.
- O pull de séries temporais é suportado através de um gateway intermediário.
- Os alvos são descobertos via descoberta de serviço ou configuração estática.
- Suporta vários modos de gráficos e atendimento via painel.

9. Explique a arquitetura básica e os princípios por trás do ClickHouse. Como o ClickHouse consegue análises de alto desempenho?
   R:
   Arquitetura do ClickHouse: ClickHouse é um sistema de gerenciamento de banco de dados colunar orientado para análises OLAP rápidas. Ele permite a análise de dados atualizados em tempo real. O ClickHouse consegue análises de alto desempenho através de várias técnicas, incluindo compressão de dados, execução de consultas em paralelo e uso de índices.

10. Uma consulta analítica específica apresenta desempenho insatisfatório. Quais etapas você seguiria para otimizar o desempenho da consulta no ClickHouse?
    R:
    Otimização de consultas no ClickHouse: Para otimizar o desempenho da consulta no ClickHouse, basta seguir as etapas abaixo:
    - **Analisar a consulta**: Use a função ***EXPLAIN*** para entender como a consulta está sendo executada.
    - Otimizar a consulta: reescrever a consulta para evitar operações custosas, como ***JOINs***.
    - **Usar índices**: Se a consulta estiver filtrando por uma coluna específica, considere adicionar um índice a essa coluna.
    - **Ajustar configurações**: Aumentar a memória disponível para o ClickHouse ou ajustar outras configurações pode melhorar o desempenho da consulta.

Complementando...

ClickHouse é um sistema de gerenciamento de banco de dados colunar orientado para análises OLAP rápidas. Ele permite a análise de dados atualizados em tempo real. O ClickHouse consegue análises de alto

desempenho através de várias técnicas, incluindo compressão de dados, execução de consultas em paralelo e uso de índices.

Os principais benefícios do ClickHouse incluem:

- **Desempenho de consulta**: Comparado a outros bancos de dados e data warehouses, o ClickHouse pode ser de 2 a 10 vezes mais rápido.
- **Uso eficiente de armazenamento e recursos de computação**: Além do armazenamento colunar, o ClickHouse também oferece muitos outros recursos excelentes, como compressão de dados altamente eficiente, indexação avançada e computação vetorizada. Ele pode utilizar totalmente as capacidades de computação e armazenamento de servidores modernos e, em geral, reduz os custos de infraestrutura.
- **Suporte para uma ampla variedade de formatos de dados**: Isso inclui formatos populares como CSV, Parquet e JSON, entre outros.
- **Escalabilidade**: O ClickHouse escala bem tanto vertical quanto horizontalmente. É facilmente adaptável para funcionar em qualquer equipamento, numa pequena máquina virtual, num único servidor ou num cluster com centenas ou milhares de nós.

11. Explique os componentes principais do Kubernetes e suas funções na orquestração de aplicativos em contêineres. Como o Kubernetes garante alta disponibilidade e escalabilidade para aplicações distribuídas?

R:

Kubernetes é uma plataforma de código aberto para automatizar a implantação, o dimensionamento e o gerenciamento de aplicativos em contêineres. A arquitetura do Kubernetes é composta por vários componentes distribuídos em diferentes servidores em uma rede. Esses servidores podem ser máquinas virtuais ou servidores dedicados.

A arquitetura do Kubernetes é dividida em dois tipos principais de nós: nós de plano de controle e nós de trabalho.

Nós de Plano de Controle: São responsáveis pela orquestração de contêineres e pela manutenção do estado desejado do cluster. Os componentes do plano de controle incluem:

- kube-apiserver: É o hub central do cluster Kubernetes que expõe a API Kubernetes.
- etcd: Armazena todas as informações do cluster.
- kube-scheduler: Responsável por agendar pods nos nós de trabalho.
- kube-controller-manager: Executa os controladores do Kubernetes.
- cloud-controller-manager: Permite que o cluster interaja com o provedor de nuvem do usuário.

Nós de Trabalho: São responsáveis por executar aplicativos em contêineres. Os componentes do nó de trabalho incluem:

- kubelet: Garante que todos os contêineres em um pod estejam funcionando corretamente.
- kube-proxy: Mantém as regras de rede nos nós e permite a comunicação de rede para seus pods.
- Container runtime: É o software responsável por executar contêineres.

Quanto à otimização do Kubernetes, existem várias estratégias que você pode implementar:

- Otimização de tempo de execução: Aumenta o desempenho com tempo de resposta reduzido e melhora a taxa de transferência em cada máquina com otimização de aplicativo em tempo real.
- Otimização de capacidade: Aplica alterações de solicitação de CPU e Memória automaticamente para otimizar o auto dimensionamento e manter seus SLAs competitivos.
- Monitoramento de utilização de memória: Pode ajudar a obter insights sobre o desempenho do cluster e a capacidade de executar cargas de trabalho com sucesso.
- Utilização de disco: O espaço em disco é um recurso não compreensível. O baixo espaço em disco no volume raiz pode levar a problemas ao agendar pods.

Microsserviços com Kubernetes

Os microsserviços são uma abordagem arquitetônica para o desenvolvimento de software onde o software é composto por pequenos serviços independentes que se comunicam usando APIs bem definidas. Esses serviços são autônomos, o que significa que cada serviço pode ser desenvolvido, implantado, operado e escalado sem afetar o funcionamento de outros serviços.

O Kubernetes é uma plataforma de código aberto que automatiza a implantação, o dimensionamento e o gerenciamento de aplicativos em contêineres. Ele fornece uma base robusta na qual implantar e executar seus microsserviços. Além disso, ele fornece serviços como descoberta de serviço e balanceamento de carga que são críticos para a execução de uma arquitetura de microsserviços. Ele também fornece as ferramentas e APIs necessárias para automatizar a implantação, o dimensionamento e o gerenciamento de seus microsserviços.

Portanto, a relação entre microsserviços e Kubernetes é que o Kubernetes fornece a infraestrutura e as ferramentas necessárias para implantar, gerenciar e escalar efetivamente uma arquitetura de microsserviços. Isso permite que as equipes de desenvolvimento se concentrem no desenvolvimento de

funcionalidades de negócios, enquanto o Kubernetes cuida de grande parte do trabalho pesado de infraestrutura.

12. A implantação do Apache Spark no Kubernetes oferece vantagens na utilização de recursos. Explique as principais considerações e configurações necessárias para executar aplicativos Spark com eficiência no Kubernetes.

    R:

    Kubernetes é uma plataforma de código aberto para automatizar a implantação, o dimensionamento e o gerenciamento de aplicativos em contêineres. Os componentes principais do Kubernetes incluem:

    - **Pods**: A menor e mais simples unidade no modelo de objeto Kubernetes que você cria ou implanta. Um Pod representa um processo em execução no seu cluster.
    - **Serviços**: Uma abstração que define um conjunto lógico de Pods e uma política para acessá-los.
    - **Volume**: Um diretório contendo dados, possivelmente acessíveis por vários contêineres de um pod.
    - **Namespace**: Destinado a uso em ambientes com muitos usuários espalhados por vários equipes ou projetos.
    - **Ingress**: Uma coleção de regras de roteamento para serviços de cluster no plano de dados.

    O Kubernetes garante alta disponibilidade ao detectar falhas nos pods e recriá-los automaticamente. Para a escalabilidade, o Kubernetes pode aumentar ou diminuir o número de pods com base na utilização de recursos.
    Em relação ao Apache Spark no Kubernetes, a implantação oferece vantagens na utilização de recursos. Aqui estão as principais considerações e configurações necessárias para executar aplicativos Spark com eficiência no Kubernetes:

    - **Configuração de recursos**: É importante configurar adequadamente os recursos do pod do executor para garantir que o Spark seja executado de maneira eficiente.
    - **Gerenciamento de dependências**: As dependências do aplicativo Spark podem ser empacotadas em um contêiner e gerenciadas através do Kubernetes.
    - **Escalabilidade dinâmica**: O Kubernetes pode ajustar dinamicamente o número de executores do Spark com base na carga de trabalho.
    - **Isolamento de recursos**: Cada executor do Spark é executado em seu próprio pod, proporcionando isolamento de recursos e melhor utilização de recursos.

Formulário de Auto Avaliação

Preencha o formulário abaixo com: Nenhum, Baixo, Intermediário ou Avançado.

Conhecimentos Mandatórios	Auto Avaliação
1. Domínio de modelagem de dados relacional e de dados multidimensionais.	Avançado
2. Domínio da linguagem SQL.	Avançado
3. Domínio de SGBDs.	Avançado
4. BI (*Business Intelligence*).	Avançado
5. Capacidade de manipular e analisar dados estruturados e não estruturados.	Avançad
6. Conhecimento avançado em análises de dados e desenvolvimento de relatório analíticos (*dashboards*) utilizando ferramentas especialistas ou através de programação.	Avançado
7. Construção de rotinas de sanitização e tratamento de qualidade de dados.	**Intermediário**
8. Ferramentas de extração, transformação e carga de dados (ETL).	Avançado
1. Domínio das linguagens Transact-SQL, PL/pgSQL e suas bibliotecas ou equivalente, abrangendo a construção de Views, Stored Procedures e Triggers.	**Intermediário**
2. Análise e tunning de SQL.	**Intermediário**
3. Domínio no desenvolvimento da ferramenta SAP PowerDesigner.	**Intermediário**
4. Domínio no suporte e administração da ferramenta SAP PowerDesigner.	**Intermediário**
5. Domínio no desenvolvimento das ferramentas Qlik Sense (incluindo Geo Analitics e NPrinting).	**Intermediário**
6. Domínio no suporte e administração das ferramentas Qlik Sense (incluindo Geo Analitics e NPrinting).	**Intermediário**
7. Domínio no desenvolvimento da ferramenta SAS.	**Básico**
8. Domínio no suporte e administração da ferramenta SAS.	**Intermediário**
9. SGBDs SQL Server, PostgresSQL ou MySQL.	Avançado
10. MongoDB.	Avançado
11. Integração de Dados via APIs, sejam estas APIs voltadas para o consumo ou para o provimento de integrações	**Intermediário**
12. Ferramentas de extração, transformação e carga de dados (ETL): Microsoft SQL SSIS, Pentaho Data Integration (PDI) ou alternativas semelhantes;	Avançado
13. Banco de dados geográficos.	**Intermediário**
14. Ferramentas WEBMAPPING (WEBGIS) MapServer, GeoServer, i3Geo, OpenLayers ou equivalentes.	Básico
15. Softwares de SIG ArcGIS, Quantum GIS,MapInfo ou equivalentes.	**Intermediário**
16. Linguagem Python e suas bibliotecas para análise de dados.	Avançado
17. Linguagens de programação: .Net , C# , Java e Javascript.	Básico
18. Big Data.	Avançado

19. Apoio às políticas, estratégias, metodologias, processos e boas práticas de gestão de qualidade de dados.	Avançado
20. Apoio às políticas, estratégias, metodologias, processos e boas práticas de gestão de catálogo de dados, gestão de dados mestre e curadoria de dados.	Avançado

OBS: Se for necessário, crie um repositório público no GitHub e compartilhe com a gente para a avaliação.