Big Data

O que é Big Data?

Uma frase para alegrar seu dia.

“Data is the new science. Big Data holds the answers.” - Patrick P. Gelsinger

Antes de tudo, Big Data também são dados, mas com um tamanho muito maior. Big Data é um termo usado para descrever uma coleção de dados com volume enorme e que cresce exponencialmente, a todo momento.

Em resumo, esses dados são tão grandes e complexos que nenhuma das ferramentas tradicionais de gerenciamento de dados é capaz de armazená-los ou processá-los com eficiência.

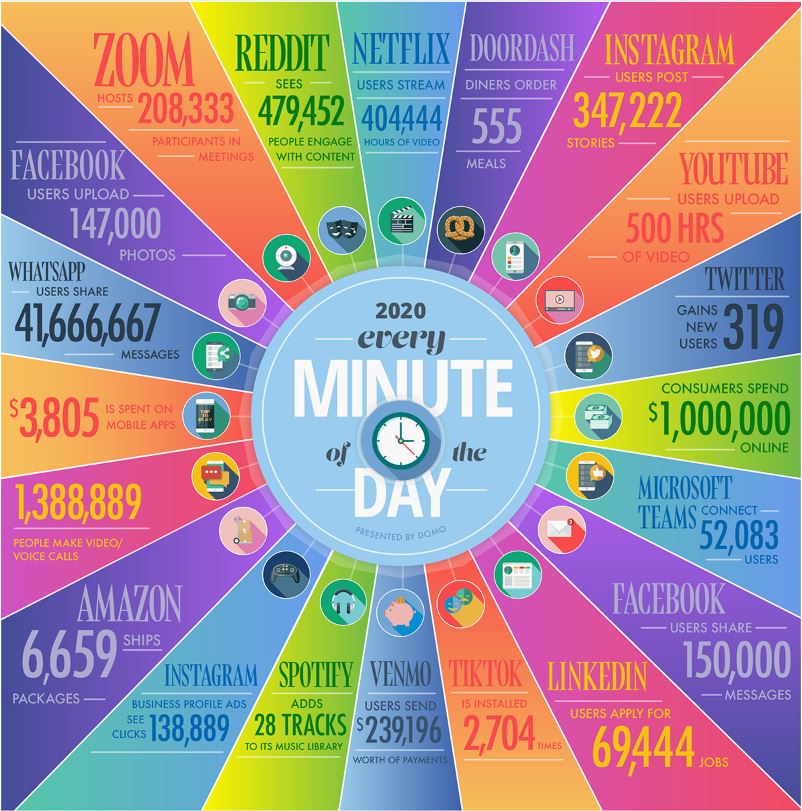
Os 'Vs' do Big Data

Em 2001, Gartner articulou a definição do termo Big Data em três 'Vs': Volume, Velocidade e Variedade. Os 'Vs' nos ajudam a ter uma compreensão melhor de como funciona BIg Data, apesar de ser antigo.

Volume

O volume dos dados é importante. Com o big data, você terá que processar grandes volumes de dados não estruturados e de baixa densidade.

Para algumas organizações, isso pode ser dezenas de terabytes de dados. Para outras, pode ser centenas de petabytes.



Velocidade

Velocidade é a taxa mais rápida na qual os dados são recebidos. Normalmente, a velocidade mais alta dos fluxos de dados são feitas diretamente na memória, ao contrário da gravação em disco.

Variedade

Variedade refere-se aos muitos tipos de dados disponíveis. Os tipos de dados tradicionais foram estruturados e se encaixam perfeitamente em um banco de dados relacional. Com o aumento do big data, os dados vêm em novos tipos de dados não estruturados. Tipos de dados não estruturados e semiestruturados, como texto, áudio e vídeo, requerem pré-processamento adicional para derivar significado e dar suporte a metadados.

Para entender melhor o que é dados estruturados, semi-estruturado e não estruturado, uma leve explicação abaixo:

* Estruturado. Dados com comprimento e tipo pré-definidos agrupados em linhas e colunas. Ex: Números, strings, Banco de Dados, etc.
* Semi-Estruturado. Dados que não possuem um comprimento ou tipo definido, mas tem formato padronizado. Ex: Arquivos xml, json, Avro, parquet.
* Não Estruturado. Dados que não possuem uma estrutura ou formato padrão. Ex: Vídeos, imagens, redes sociais, texto.

Com o passar do tempo e da quantidade de dados produzida aumentando drasticamente , a IBM introduziu mais dois Vs, que seria Veracidade e mais importante do Big Data, o Valor.

Veracidade

A Veracidade refere-se à qualidade dos dados que estão sendo analisados. Os dados de alta veracidade têm muitos registros valiosos para analisar e que contribuem de maneira significativa para os resultados gerais. Os dados de baixa veracidade, por outro lado, contêm uma alta porcentagem de dados sem sentido.

Valor

O valor que os dados geram para os usuários e para os negócios. Já sabemos que temos um volume colossal de dados gerados a cada minuto de diversas fontes e formatos, dados estes que devem ser verídicos para gerar valor,

Exemplos de Implementações do Big Data

Os dados podem vir de inúmeras fontes diferentes, como sistemas de transações, bancos de dados, registros médicos, logs, aplicativos móveis, redes sociais, dados gerados por sensores em tempo real.

Usando dados do cliente como exemplo, os diferentes ramos da análise que podem ser feitos com as informações encontradas em conjuntos de dados incluem o seguinte:

* Análise comparativa. Isso inclui o análise das métricas de comportamento do usuário e a observação do envolvimento do cliente em tempo real , a fim de comparar os produtos, serviços.
* Mídias Sociais. Essas são as informações sobre o que as pessoas estão dizendo nas mídias sociais sobre um negócio ou produto específico que vai além do que pode ser entregue em uma enquete ou pesquisa. Esses dados podem ser usados para ajudar a identificar o público-alvo das campanhas de marketing.
* Análise de marketing. Isso inclui informações que podem ser usadas para tornar a promoção de novos produtos, serviços e iniciativas mais informativas e inovadoras.
* Análise de satisfação e sentimento do cliente. Todas as informações coletadas podem revelar como os clientes estão se sentindo em relação a uma empresa ou marca, se houver algum problema em potencial, como a lealdade à marca pode ser preservada e como os esforços de atendimento ao cliente podem ser aprimorados.

Vantagens e desvantagens do Big Data

O aumento na quantidade de dados disponíveis apresenta oportunidades e problemas.

Em geral, ter mais dados sobre os clientes deve permitir que as empresas adaptem melhor seus produtos e esforços de marketing para criar o mais alto nível de satisfação e repetir negócios. As empresas capazes de coletar uma grande quantidade de dados têm a oportunidade de realizar análises mais profundas e mais ricas.

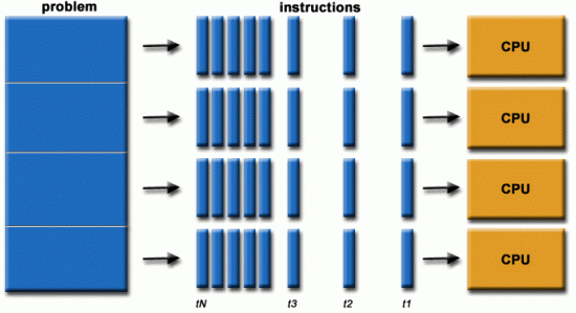
Embora uma análise melhor seja positiva, o big data também pode criar sobrecarga e qualidade. As empresas precisam ser capazes de lidar com grandes volumes de dados, enquanto determinam quais dados representam geram valor para negócio, nem sempre os dados podem vir com melhor qualidade. Determinar o que torna os dados relevantes se torna um fator-chave.

Além disso, a natureza e o formato dos dados podem exigir tratamento especial antes de serem adotados. Dados estruturados, consistindo em valores numéricos, podem ser facilmente armazenados e classificados. Dados não estruturados, como e-mails, vídeos e documentos de texto, podem exigir a aplicação de técnicas mais sofisticadas antes que se tornem úteis.

Computação Paralela

Uso simultâneo de várias CPUs para realizar trabalhos computacionais.

Utilizando essa técnica, é possível ultrapassar as limitações tecnológicas de uma máquina, aumentando sua velocidade e poder de processamento.

Exemplo: 

Objetivos:

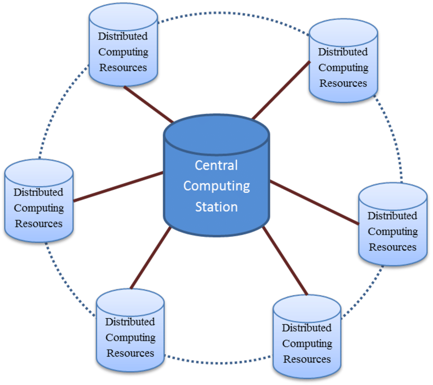
* Resolver problemas maiores
* Resolver mais rápido
* Resolver problemas mais complexos

Computação Distribuída

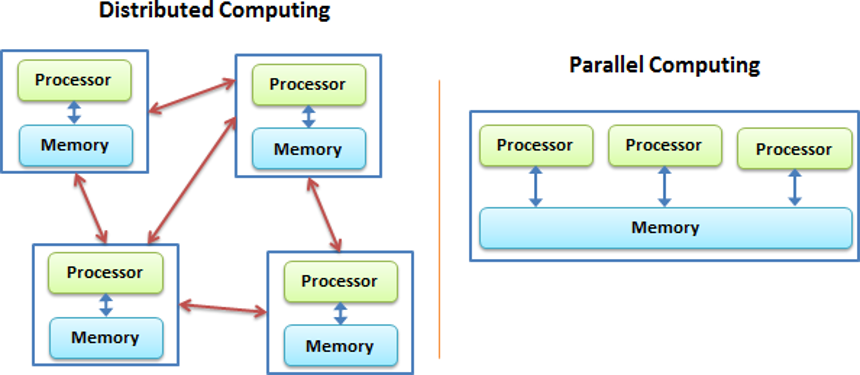
Coleção de computadores autônomos interligados através de uma rede de computadores e equipados com software que permita o compartilhamento dos recursos do sistema: hardware, software e dados.

Cluster:

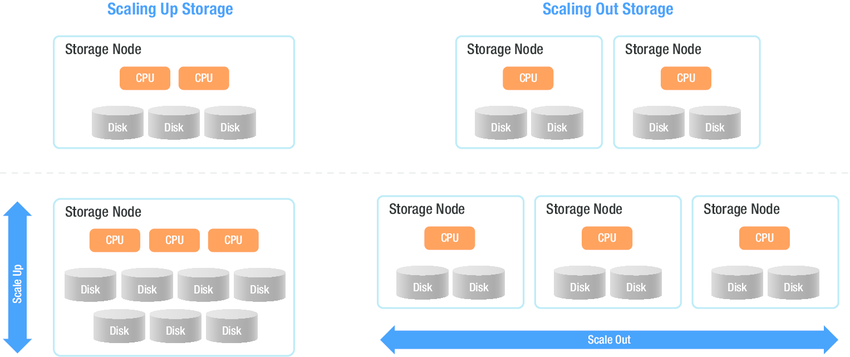
Sistema que relaciona dois ou mais computadores para que estes trabalhem de maneira conjunta no intuito de processar uma tarefa. Enquanto no multiprocessamento simétrico ou SMP (Symmetric Multi-Processing) todas as CPUs se encontram na mesma máquina, no processamento distribuído várias CPUs estão separadas fisicamente e se interligam para formar um cluster.



Computação Paralela x Computação Distribuída



Escalonamento Vertical x Escalonamento Horizontal



* Scale up tem relação com Computação Paralela.
* Scale out tem relação com Computação Distribuída

# Hadoop

## O que é ?

Hadoop é um framework em código aberto para armazenamento e processamento distribuídos de grandes conjuntos de dados em hardware simples. 

## História

Motivado a construir um buscador complexo, que funcione na escala da web, indexando bilhões de páginas, Doug Cutting resolveu se dedicar ao desafio iniciando seu projeto Nutch junto com Mike Cafarella, mas enfrentou alguns problemas com escalabilidade.

Hadoop era o nome do elefante amarelo de pelúcia do filho de Doug.

Um artigo publicado em 2003 pelo Google abriu caminho para que a equipe do Nutch criasse uma implementação open source do GFS (Google File System).

Em 2004 o Google publica o clássico artigo descrevendo seu framework MapReduce para atender às necessidades de processamento de várias máquinas das tarefas de rastreamento e índice.

## Onde usar Hadoop?

* Análise de Dados
* Data Warehouse
* Data Lake
* Processamento de logs
* Muito mais!

## Características

* Baixo custo
* Flexibilidade de armazenamento
* OpenSource
* Tolerante a falha
* Análise complexa de dados
* Escalabilidade

## Componentes

* Hadoop Common
* HDFS (Hadoop File System)
* MapReduce
* Yarn

## Replicação

A alta disponibilidade de dados no Hadoop é possível devido à replicação implícita de dados em um cluster Hadoop. Um bloco de arquivo é replicado em vários "nós de dados" com base no "fator de replicação" do cluster Hadoop, que pode ser 1, 2, 3 ...

Um fator de replicação 1 indica que um bloco de arquivos residirá em um único "nó de dados". Um fator de replicação 2 indica que um bloco de arquivos residirá em dois "nós de dados", no mesmo rack ou em um fisicamente a milhares de quilômetros de distância; etc.

## Arquitetura

O Hadoop é baseado em uma arquitetura Master/Slave. Um cluster Hadoop possui um único nó Master e vários nós Slaves.

### Master

É responsável por armazenar os metadados associados aos seus nós slaves no rack do qual faz parte.

O nó principal é responsável por manter o status de seus nós slaves, estabelecendo um deles como um nó passivo, que se tornará um nó principal se, por qualquer motivo, estiver bloqueado. Um dos problemas do Hadoop é que às vezes o nó passivo não é sincronizado com o nó principal original , assumindo suas funções no processo.

### Slave

É o nó encarregado de armazenar as informações que estão sendo processadas pelo nó master em um momento específico.

## Hadoop Common

Também conhecido como Hadoop Core, é a coleção de utilitários comuns e bibliotecas (JAR) que oferecem suporte a outros módulos Hadoop, sendo assim vital para sua inicialização e funcionamento.

## HDFS

Hadoop Distributed File System (HDFS) é o sistema de armazenamento distribuído utilizado por aplicações Hadoop. O HDFS quebra os arquivos em blocos de dados (128 MB por padrão), cria réplicas (três por padrão) e as ditribui no cluster, permitindo assim computações extremamente rápidas em arquivos pequenos e em máquinas distintas. HDFS permite escalabilidade e tolerança a falhas

### Componentes

* NameNode: Gerencia o namespace do sistema de arquivos do Hadoop.
* DataNode: Armazena os blocos de dados em um nó.

#### **NameNode**

* Armazena metadados.
* Usa cache em RAM para acesso mais rápido ao metadado.
* Não armazena dados.
* Apenas 1 ativo por cluster.
* Ponto único de falha sem HA (Alta Disponibilidade).

Em resumo, o NameNode faz a gestão do HDFS em um nó: mantém metadados, logs, adiciona, encontra, exclui e copia arquivos.

#### **DataNode**

* Armazena os dados no HDFS.
* Atende solicitações de leitura e gravação dos clientes ou NameNode.
* Responsável por criar, excluir e replicar blocos de dados.
* Reportar status para o NameNode (heartbeat).
* Em caso de falta de report o nó é desativado pelo NameNode.
* Totalmente dependente do NameNode.

Em resumo, o DataNode mantém dados e replica blocos.

### Prática!

Criar um diretório no HDFS:

hdfs dfs -mkdir /user/cloudera/aula

Listar o conteúdo do diretório do HDFS:

hdfs dfs -ls /user/cloudera/aula

crie um arquivo:

vim hadoop.txt

Inserir arquivo no HDFS a partir do FileSystem:

hdfs dfs -put hadoop.txt /user/cloudera/aula

hdfs dfs -ls /user/cloudera/aula

Visualizar conteúdo do arquivo no HDFS:

hdfs dfs -cat /user/cloudera/aula/hadoop.txt

hdfs dfs -tail /user/cloudera/aula/hadoop.txt

Mover arquivo dentro do HDFS:

hdfs dfs -mv /user/cloudera/aula/hadoop.txt /user/cloudera/

Copiar arquivo dentro do HDFS:

hdfs dfs -cp /user/cloudera/hadoop.txt /user/cloudera/aula

Remover arquivo do HDFS:

hdfs dfs -rm /user/cloudera/aula/hadoop.txt

Copiar arquivo do HDFS para o FileSystem:

hdfs dfs -get /user/cloudera/hadoop.txt /tmp/

Localização do blocos no HDFS:

hdfs fsck /user/cloudera/hadoop.txt -blocks -files -locations

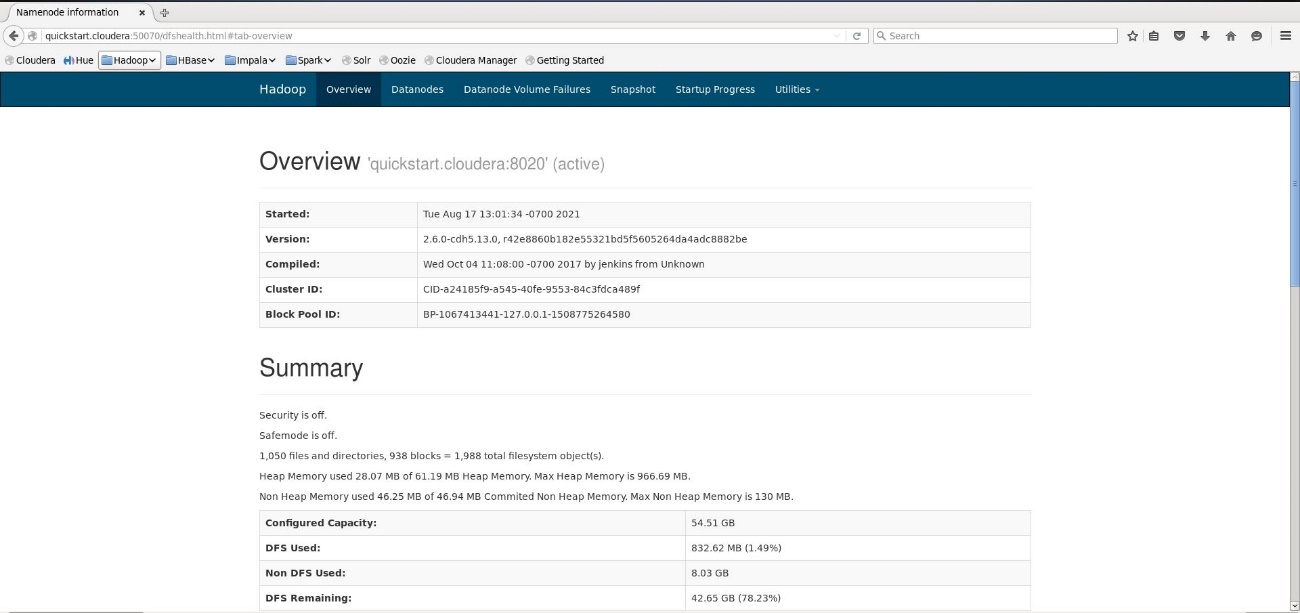
Aumentar quantidade de réplicas:

hdfs dfs -setrep 2 /user/cloudera/hadoop.txt

### Interface Web

#### **Overview**

http://quickstart.cloudera:50070



#### **Hue**

http://quickstart.cloudera:8888



## MapReduce

O Hadoop MapReduce é um modelo de programação para criação de aplicações processam rapidamente vastas quantidades de dados em paralelo através de grandes clusters de computadores comuns.

O código ou programa a ser executado, é transportado até o local do dado, executando tarefas independentes em cada bloco de dado (Map), e depois são consolidados gerando a resposta do processamento (Reduce).

Estrutura de uma aplicação MapReduce.

* Map: Atua exclusivamente sobre um conjunto de entrada com chaves e valores, produzindo uma lista de chaves e valores.
* Reduce: Atua sobre os valores intermediários produzidos pelo map para, normalmente, agrupar os valores e produzir uma saída.

O processo, de forma simplificada:

* Dados são divididos em blocos
* Divisão de problemas grandes e/ou complexos em pequenas tarefas
* Mapeamento é executado em paralelo nos nós
* Apenas quando o Mapeamento é encerrado, redução inicia, também em paralelo
* Fase intermediária: Shuffle (distribui as saídas dos mappers para a execução do reducer)
* Existem tarefa que requerem apenas a etapa de Mapeamento.

### Map

Atua exclusivamente sobre um conjunto de entrada com chaves e valores, produzindo uma lista de chaves e valores

Características:

* Ponto de partida
* Recebe cada registro dos dados de entrada como pares de chave/valor
* Cada Mapper é independente um do outro, permitindo paralelismo e re-execuções de tarefas
* Hadoop cria tarefas de Mapper para cada bloco de dados HDFS dos dados de entrada.
* Produz uma lista de chave/valor

### Reduce

Atua sobre os valores intermediários produzidos pelo map para, normalmente, agrupar os valores e produzir uma saída.

Características:

* Responsável por agregações, filtros e combinações diversas nos dados de entrada
* Executa uma função de reduce por vez
* Shuffle: Distribui as saídas dos mappers para a execução do reducer
* Sort: Ordena os registros chave/valor, agrupando pela chave
* Reduce: Envia os conjuntos chave/valor agrupados, filtrados ou combinados o formato de saída

### Pontos Positivos

* Escalável
* Tolerante a falhas
* Disponibilidade
* Confiável
* Usa conceito de chave/valor
* Não cria gargalos na rede pois os dados não trafegam (processamento no nó)

### Pontos Negativos

MapReduce não é indicado para:

* Consultas que necessitam de baixa latência
* Sistemas real-time
* Consultas em um website
* Processamento de pequenas tarefas
* Overhead para gerenciamento das tarefas

### Prática!

Download

https://s3-sa-east-1.amazonaws.com/lcpi/c6729d53-18cb-4f16-b1b7-e0217c57649a.zip

Criar um diretório no HDFS:

hadoop fs -mkdir /user/cloudera/input

Inserir arquivo no HDFS a partir do FileSystem

hadoop fs -put texto.csv /user/cloudera/input

Executando MapReduce

hadoop jar wordcount.jar WordCount /user/cloudera/input /user/cloudera/output

Verificar Job

http://quickstart.cloudera:8088

Ver lista os arquivos no diretório no HDFS:

hadoop fs -ls /user/cloudera/output

Verificando o conteúdo do arquivo:

hadoop fs -cat /user/cloudera/output/part-r-00000

## Yarn

O YARN foi introduzido no Hadoop versão 2.0 no ano de 2012 pelo Yahoo e Hortonworks. A idéia básica por trás do YARN é aliviar o MapReduce, assumindo a responsabilidade do Gerenciamento de Recursos e do Agendamento de Tarefas. O YARN começou a dar ao Hadoop a capacidade de executar tarefas não MapReduce na estrutura do Hadoop.

Características:

* Permite que vários aplicativos sejam executados simultaneamente no mesmo cluster compartilhado.
* Permite que os aplicativos negociem recursos com base na necessidade.

### Arquitetura

Duas principais funcionalidades: gerenciamento de recursos e agendamento/monitoramento de tarefas Componentes:

* ResourceManager: um por cluster (orquestrador)
  + ApplicationManager: gerencia atividades, otimização, distribuição de recursos.
* NodeManager: um por nó, responsável pela execução dos Jobs.
* Aplication Master: Distribui tarefas aos containers.
* Container: mantém as tarefas.

### ResourceManager

* Possui um agendador de nível de cluster que tem responsabilidade pela alocação de recursos para todas as tarefas em execução, de acordo com as solicitações do ApplicationManager.
* A principal responsabilidade do ResourceManager é alocar recursos para os aplicativos.
* Não é responsável pelo rastreamento do status de uma aplicação ou tarefas de monitoramento.
* Não garante o reinício/balanceamento de tarefas no caso de falha no aplicativo ou no hardware.

### NodeManager

* Nó Slave, é executado nos worker nodes.
* Gerencia o ciclo de vida do container e monitora o uso de recursos.
* Executa o container com base na capacidade do nó, que é calculada com base na memória instalada e no número de núcleos da CPU.
* Envia um sinal ao para atualizar seu status de integridadeResourceManager.
* Envia o status para ResourceManager, que pode ser o status do nó ou o status das tarefas executadas.

### ApplicationMaster

* Biblioteca de aplicativos que gerencia cada instância de um aplicativo que é executado dentro de YARN.
* Responsável por negociar recursos do ResourceManager na submissão do aplicativo, como memória e CPU.
* Responsável por monitorar o status de um aplicativo e monitorar os processos de aplicativos em coordenação com o NodeManager.

### Container

* Pacote lógico de recursos em termos de memória, CPU, disco, etc...
* Vinculado a um nó específico
* ResourceManager aloca dinamicamente recursos como containeres
* Um container concede direitos a um ApplicationMaster para usar uma quantidade específica de recursos de um host específico
* ApplicationMaster é considerado como o primeiro container de um aplicativo e gerência a execução da lógica do aplicativo em containers alocados

### Prática!

Verificar Jobs:

http://quickstart.cloudera:8088/

Os comandos YARN são invocados usando o script bin/yarn no pacote Hadoop

A sintaxe básica para o comando:

yarn [--config confdir] COMMAND COMMAND\_OPTIONS

Application: Lista, obter status e mata um aplicativo:

yarn application -list

yarn application -status application\_1507835793432\_0005

yarn application -kill application\_1507835793432\_0005

Node: Lista e obter status dos nós

yarn node –list

yarn node -all -list

yarn node -status quickstart.cloudera:47512

Logs: Obtem logs de um aplicativo já finalizado

yarn logs -applicationId application\_1507835793432\_0005

Classpath: retorna o valor do classpath atual

yarn classpath

Version: retorna a versão atual do Cluster Yarn

yarn version

Top: Fornece um resumo de informações:

yarn top

Apache Hive

É um sistema de data warehouse de código aberto, usado exclusivamente para consultar e analisar grandes conjuntos de dados armazenados no Hadoop. O Hive utiliza uma linguagem chamada HiveQL (Hive Query Language) , que transforma as sentenças SQL em Jobs MapReduce executados no cluster Hadoop, para assim aproveitar as "skills" em SQL dos analistas e desenvolvedores do Facebook, que não eram na época tão proficientes em Java para usar o MapReduce.

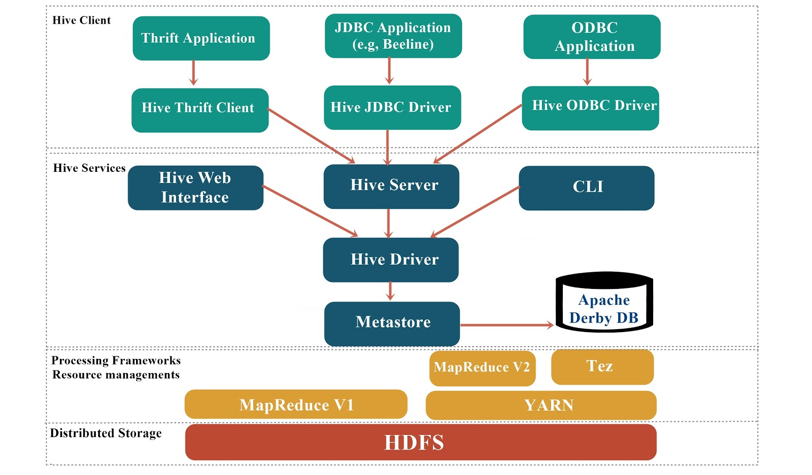
O que é?

* Interpreta instruções SQL para Job MapReducer
* Lê dados de arquivos estruturados e semi-estruturados no HDFS, e se baseia em metadados para simular uma tabela de um banco de dados relacional
* Não possui um SGBD
* Foi desenhado para melhor performance analisando grandes quantidades de dados que se encontram em clusters.
* Data Warehouse do Hadoop

Hive x RDBMS

|  | **HIVE** | **RDBMS** |
| --- | --- | --- |
| Uso | Foco em analytics | Foco em on line ou analytics |
| Acesso | Batch | Batch e Interativo |
| Integridade | Baixa | Alta |
| Escalabilidade | Horizontal | Vertical |
| Armazenamento | Baixo custo por PB | PB? |
| Interface | HiveQL | SQL |
| Latência | Minutos ou mais | ms, ml ou segundos |
| Estrutura de dados | Não Estruturado | Estruturado |

Arquitetura



Componentes do Hive

Driver

Compila, otimiza, executa o HiveQL, e decide se executa a query local ou submeter em um job MapReduce.

Metastore

Armazena os metadados, interpretando dos arquivos no HDFS como conteúdos de uma tabela. Armazena as informações de como as linhas e colunas são delimitadas dentro do arquivo (Hive Schemas). Pode ser armazenado no MySQL, Oracle, Derby ou Postgresql.

CLI

Command Line Interface – para acessar o shell do hive. É o serviço padrão. Acessado pelo terminal no SO.

Hiveserver

Permite que conexões (Thrift, ODBC e JDBC) de outros componentes tenham comunicação com o hive.

Hiveserver2

Evolução do hiveserver, suportando autenticação e múltiplos usuários concorrentes.

HiveQL

Utiliza Data Definition Language (DDL) para criar os bancos de dados e tabelas.

Beeline

É a Command Line Interface para acessar o hiveserver2, utilizando conexão JDBC.

Execute comando beeline para iniciar a interface de consulta:

beeline -u "jdbc:hive2://quickstart.cloudera:10000/default"

Após iniciar beeline, execute consulta no hive:

show tables;

Para poder ver os comandos do beeline:

beeline -help

Carga de Diretórios - LOCATION

Tabelas criadas sem LOCATION, os dados são armazenados no diretório default do HDFS hive que é: /user/hive/warehouse/NOME\_DATABASE.db/NOME\_TABELA

* Quando é criado no database default a estrutura é: /user/hive/warehouse/NOME\_TABELA
* Tabelas criadas com LOCATION, o arquivo é armazenado neste local definido.

Tabela Gerenciada (Manage Table)

Também conhecidas como tabelas internas, tem o ciclo de vida de seus dados controlados pelo Hive.

Armazena os dados dessas tabelas em um subdiretório sob o diretório definido por hive.metastore.warehouse.dir (por exemplo, / user / hive / warehouse)

Lembre-se: quando excluímos a tabela os dados também são excluídos!

Prática!

Criar diretório no hdfs

hdfs dfs -mkdir /user/hive/hive\_aula

Crie um arquivos chamado aula.txt, escreva algum conteúdo

vim aula.txt

Insira o arquivo no diretório do hive

hdfs dfs -put aula.txt /user/hive/hive\_aula

Entre na interface de query do hive e executa comando hiveQL

create table teste\_load( campo string );

load data inpath '/user/hive/hive\_aula/aula.txt' overwrite into table teste\_load;

select \* from teste\_load;

! hadoop fs -ls /user/hive/warehouse;

Tabela Externa(External Table)

External Table, é apenas uma referência a um arquivo, ou seja, o Hive não é dono dos dados. É possível, inclusive, que uma ou mais external tables apontem para o mesmo arquivo no HDFS.

Neste caso, quando excluímos a tabela os dados não são excluídos, apenas seus metadados!

Por fim, é importante se atentar que é necessário informar a localização dos dados.

A palavra-chave EXTERNAL diz ao Hive que esta tabela é externa  
Cláusula LOCATION é necessária para informar ao Hive onde os dados estão localizados

Prática!

Criar diretorio no hdfs

hdfs dfs -mkdir /user/hive/hive\_aula\_put

Copie o arquivo aula para novo diretório

hdfs dfs -cp /user/hive/hive\_aula/aula.txt /user/hive/hive\_aula\_put

Entre na interface de query do hive e executa comando hiveQL

create external table teste\_put( campo string ) location ‘/user/hive/hive\_aula\_put’;

select \* from teste\_put;

Copie o arquivo aula para novo diretório

hdfs dfs -cp /user/hive/hive\_aula/aula.txt /user/hive/hive\_aula\_put

Execute novamente a query a visualizar

select \* from teste\_put;

Location

Tabelas criadas sem LOCATION, os dados são armazenados no diretório default do HDFS hive que é:

/user/hive/warehouse/NOME\_DATABASE.db/NOME\_TABELA

Quando é criado no database default a estrutura é:

/user/hive/warehouse/NOME\_TABELA

Tabelas criadas com LOCATION, o arquivo é armazenado neste local definido.

Database

* Banco de dados no Hive é essencialmente apenas um catálogo ou espaço para tabelas .
* Se não especificar o banco de dados, o banco de dados default é usado.
* Podemos criar vários bancos de dados .
* Podemos alterar o diretório na hora da criação.

Comandos

Criar banco de dados

create database if not exists [nome] location [path no hdfs]

Visualizar banco de dados

show databases

Definição do banco de dados

describe database [nome]

Excluir banco de dados

drop database [nome]

Alterar banco de dados

alter database [nome] [parâmetros]

Prática!

Crie um banco de dados no Hive

create database aula location '/user/hive/db\_aula';

Verifique o database

show databases;

Definição do banco de dados

describe database aula;

View

É uma referência lógica de uma consulta, é como salvar uma query em uma tabela lógica, para ser acessada de forma transparente como se fosse uma física. Muito utilizada para consultas preestabelecidas por usuários e controle de acesso de campos.

create view teste\_view as select campo from table\_hive union all select campo from table\_hive;

select campo from teste\_view;

Funções

O Hive disponibiliza funções nativas, UDF, UDAF e UDTF, além de permitir o desenvolvimento de funções customizadas.

UDF – Funções de usuário.

* Uma entrada e uma saída. Ex. SPLIT

UDAF – Funções de agregação.

* N entradas e uma saída. Ex. SUM

UDTF – Funções de tabelas.

* Uma entrada e N saídas. Ex. EXPLODE

Função - SPLIT

Recebe um valor e um delimitador, devolve um array, ou seja, entra um elemento e sai N, no mesmo campo.

create table teste\_split(campo string);

load data local inpath '/aula/aula.txt' overwrite into table teste\_split;

select \* from teste\_split;

select split(campo,' ') from teste\_split;

Função - EXPLODE

Recebe um array, ou seja, um conjunto de valores delimitados, e devolve em linhas.

select explode(split(campo,' ')) from teste\_split;

Hue

Interface visual para manipulação do Hive e HDFS.

http://quickstart.cloudera:8888

SerDe

* Abreviação de Serializer / Deserializer.
* O Hive usa a interface SerDe para IO (Input/Output)
* Um SerDe permite que o Hive leia os dados de uma tabela e os grave no HDFS em qualquer formato personalizado.
* Qualquer um pode escrever seu próprio SerDe para seus próprios formatos de dados.
* Avro, ORC, RegEx, Parquet, CSV, JsonSerDe.

Formatos de arquivos – Posicional

Formato posicional utilizando SerDe. Crie um arquivo com o conteúdo abaixo no local do hdfs.

Arquivo:

00001cliente\_01PF

00002cliente\_02PJ

00003cliente\_03PF

Estrutura da tabela:

create external table tabela\_posicional ( conta string, nome string, tipo\_pessoa string )

row format serde 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'

with SERDEPROPERTIES ('input.regex' ='(.{5})(.{10})(.{2})')

location '/user/hive/tabela\_posicional';

Formatos de arquivos - Delimitado

Formato posicional utilizando delimitador pipe. Crie um arquivo com o conteúdo abaixo no local do hdfs.

Arquivo:

1,cliente\_01,PF

2,cliente\_02,PJ

3,cliente\_03,PF

Estrutura da tabela:

create external table tabela\_delimitada( conta string, nome string, tipo\_pessoa string )

row format delimited fields terminated by ',' lines terminated by '\n' stored as textfile

location '/user/hive/tabela\_delimitada';

Formatos de arquivos – JSON

JSON é um modelo para armazenamento de dados com capacidade de estruturar informações de uma forma bem mais compacta do que os arquivos sequenciais ou delimitados.

Crie a tabela:

CREATE EXTERNAL TABLE tabela\_json( campo string) stored as textfile LOCATION '/user/hive/tabela\_json';

Crie um arquivo no HDFS /tabela\_json com o conteúdo:

{"conta":1,"cliente":{"nome":"cliente\_1","tipo\_pessoa":"PF"}}

{"conta":2,"cliente":{"nome":"cliente\_2","tipo\_pessoa":"PJ"}}

{"conta":3,"cliente":{"nome":"cliente\_3","tipo\_pessoa":"PF"}}

Execute a query:

select get\_json\_object(campo, "$.conta") as conta, get\_json\_object(campo, "$.cliente.nome") as nome, get\_json\_object(campo, "$.cliente.tipo\_pessoa") as tipo\_pessoa from tabela\_json;

Particionamento no Hive

O Apache Hive organiza tabelas em partições. O particionamento é uma maneira de dividir uma tabela em partes relacionadas com base nos valores de colunas específicas, como data, cidade e departamento. Cada tabela na seção pode ter uma ou mais chaves de partição para identificar uma partição específica.

Características:

* Particiona a tabela em diretórios no HDFS
* Ganho de performance e eficiência
* Evita a leitura da base toda
* Muito utilizado em grandes volumes de dados

Vamos criar uma tabela particionada.

create external table table\_part(nome string) partitioned by (ano int) row format delimited fields terminated by ',' lines terminated by '\n' stored as textfile;

Verificar tabela.

Describe table\_part;

Adicionando partição.

alter table table\_part add partition(ano=2021) location '/user/hive/tabela\_part/particao/ano=2021';

Excluindo partição.

alter table table\_part drop partition(ano=2021)

Extração de dados Hive - HDFS

Agora vamos extrair dados do hive para hdfs.

insert overwrite directory ‘/user/hive/output’ select \* from table\_part;

Impala

O Hive não foi desenhado para executar queries em real time, com baixa latência, mas sim para melhor performance analisando grandes quantidades de dados que se encontram em clusters.

Cloudera lançou o Impala, permitindo execução de queries com baixa latência para dados armazenados no HDFS e no HBase, usando a linguagem SQL, permite integração com ferramentas de BI como Tableau, Microstrategy, Pentaho, etc.

Comandos Impala

Execute comando abaixo para abrir shell do impala

impala-shell

Hue

O Impala também pode ser utilizado através do HUE, com interface visual, assim como o Hive!

Data Warehouse

Business Intelligence

O conceito de BI foi definido pelo Gartner Group como:

*"Um termo genérico que inclui aplicações, infraestrutura, ferramentas e melhores práticas que permite o acesso e análise de informações para melhorar e otimizar decisões e desempenho".*

Buisiness Intelligence x Data Science

| **Análise** | **BI** | **Data Science** |
| --- | --- | --- |
| Descritiva - O que aconteceu? | x | x |
| Diagnóstica - Porque aconteceu? | x | x |
| Preditiva - O que acontecerá? |  | x |
| Prescritiva - O que deve ser feito? |  | x |

OLTP x OLAP

OLTP

OLTP, do inglês "On-line Transaction Processing", é o termo utilizado para se referir a sistemas transacionais, que fazem portanto parte da operação da empresa. Enquandram-se aqui e-commerces, ERPs e todo sistema mais tradicional. No que diz respeito à seus bancos de dados, por terem como característica o frequente recebimento de novos registros (escrita), seu modelo de dados é, em geral, projetado considerando um processo que chamamos de normalização.

*Normalização é o processo de organização de dados em um banco de dados. Isso inclui a criação de tabelas e o estabelecimento de relações entre essas tabelas de acordo com as regras projetadas para proteger os dados e tornar o banco de dados mais flexível, eliminando a redundância e a dependência inconsistente. Há diferentes formas normais que, quanto mais avançadas, mais exigentes são com o desenho da base para atingir os objetivos anteriormente citados o mais acentuadamente possível.*

Sendo assim, o banco de dados de um sistema OLTP costuma possuir mais tabelas que um OLAP (acréscimo esse que varia conforme a forma normal adotada) e para reduzir redûncia de dados e também facilitar a escrita, entretanto, exige de leituras e criações de relatório mais cruzamentos entre tabelas (join em SQL), o que é bastante custoso em termos de processamento.

OLAP

OLAP, do inglês “On-line Analytical Processing“, se referece a análise de dados, comumente produzidos em OLTPs e então armazenados em um Data Wharehouse.

Data Warehouse (DW) - Definição

Data Warehousing é uma estratégia que reconhece a necessidade de se armazenar dados, separadamente, em sistemas de informação e consolidá-los, de forma a assistir diversos profissionais de uma empresa na tomada de decisões de modo rápido e eficaz.

De forma simplista, um DW pode muitas vezes ser um banco de dados comum (muitas vezes relacional), mas que possui especificidades em sua construção visando facilitar análises (leitura), mesmo que em detrimento do espaço em disco ou velocidade de escrita. Normalizações severas não são comuns por aqui!

Características:

* Os aspectos interpretativos se sobrepõem aos transacionais.
* Os dados são organizados por assuntos e não voltado para aplicações ou funções de negócio operacional.



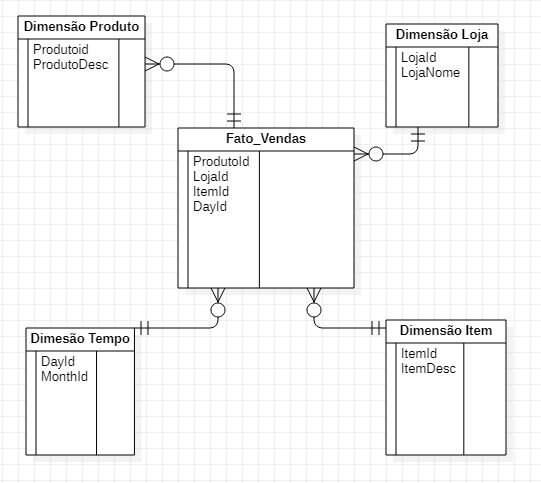
Data Warehouse Schemas

Datawarehouses podem ser construídos tendo diferentes schemas como base, todavia, os mais difundidos são

* Star Schema
* Snow Flake Schema

Star Schema

O Star Schema, idealizado por Ralph Kimball, é o modelo mais popular para modelagem dimensional de um Data Warehouse. Em seu desenho, as tabelas são divididas em "Fatos" e "Dimensões", onde a presença de uma fato centralizadas, com várias dimensões ao redor, lhe atribuem um aspecto de "estrela".



Snow Flake Schema

Snow Flake, idealizado por Bill Inmon, foi idealizado sob os mesmos propósitos do Star Schema, mas visando economia de espaço em disco ainda que em detrimento da perfomance durante a leitura de uma dimensão. Como podemos notar no exemplo abaixo, a similaridade é grande com o modelo anterior, mas com algum nível de normalização nas tabelas dimensões.

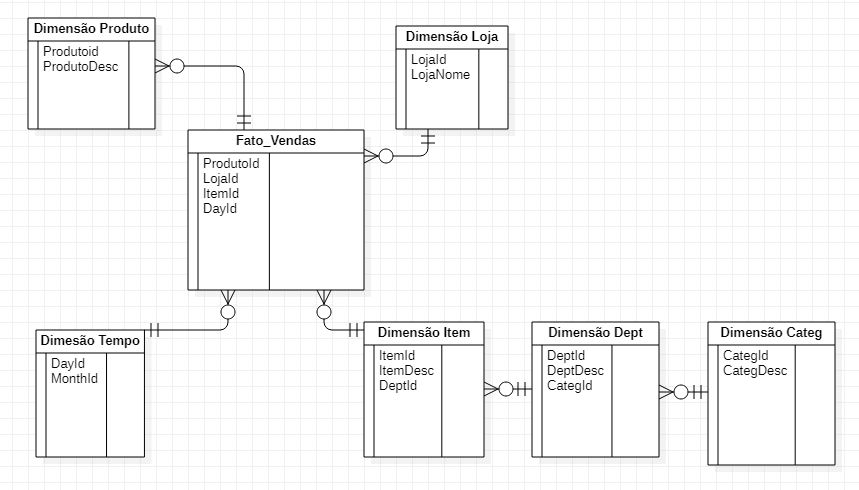


Tabela Fato

Tabela Fato é onde os fatos ocorridos são armazenados. Possue as seguintes características:

* Contém métricas numéricas do negócio.
* Tabela com maior volume de dados.
* Cresce rapidamente.
* Pode conter dados base, derivados e sumarizados.
* São relacionadas à duas ou mais tabelas dimensão com chaves estrangeiras que referenciam as chaves primárias nas tabelas dimensão.

Tabela Dimensão

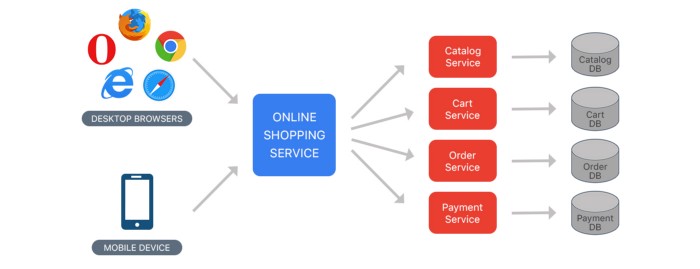
* Contém informação textual que representa atributos de negócio.
* Contém dados relativamente estáticos.
* São relacionadas com a Tabela Fato.

Data Lake

Um Data Lake é um repositório caapz de armazenar uma grande quantidade de dados estruturados, semiestruturados e não estruturados.

objetivo:

* Visão única dos dados
* Análises complexas
* Aprendizagem de máquina.



Data Lake x Data Warehouse

|  | **Data Lake** | **Data Warehouse** |
| --- | --- | --- |
| Dados | Estruturados, Semi-estruturados e Nãoestruturados. Dados em estado bruto. | Dados estruturados e processados antes da carga no banco de dados. |
| Processamento | Esquema de dados gerado no momento da leitura. | Esquema de dados gerado no momento da escrita. |
| Armazenamento | Criado para ser de baixo custo, independente do volume de dados. | Alto custo para grandes volumes de dados. |
| Agilidade | Bastante ágil. Pode ser configurado e reconfigurado conforme necessários. | Pouco ágil, configuração fixa. |
| Segurança | Ainda precisa aperfeiçoar o modelo de segurança e acesso aos dados. | Estratégias de segurança bastante maduras. |
| Usuários | Cientistas e Analistas de Dados | Analistas de Negócio, BI e Cientistas de Dados. |

Porquê usar um Data Lake?

Em aulas anteriores discutimos Big Data e seus Vs. Sendo assim, considerando a variedade, velocidade e volume demandados por aplicações modernas, era notável que um DW não seria suficiente. É importante entender que um não substitui o outro! Ambos poder ser utilizados em conjunto pois a necessidade de analisar o passado jamais deixará de ser relevante para uma organização, entretanto, faz-se necessário um Data Lake para armazenar e tratar grandes volumes de dados de diferentes tipos.

Onde entra o Hadoop?

Tendo compreendido as razões por trás do uso de um Data Lake, e também suas características, é perceptível onde o Apache Hadoop pode ser aproveitado.

Antermiomente, quando apresentamos o framework, discutimos todo seu pontencial envolvendo escalabilidade (via computação paralela e distribuída), tolerância a falhas e mais. Sendo assim, é natural pensar em um Data Lake que utiliza Hadoop e seu ecossistema (HDFS, MapReduce, etc.) para armazenar e processar dados.

Arquitetura

Se tratando de um Data Lake, que nada mais é que um grande repositório cheio de diretórios e arquivos de todos tipos, é natural imaginar que profissionais responsáveis por sua modelagem tenham muita liberdade. Todavia, existem padrões que podem ser utilizados como guias, e o principal deles divide um Data Lake em difernetes camadas/zonas.

Camada Raw

Raw, ou lading zone, é uma área que funciona como porta de entrada para um Data Lake. Aqui dados brutos, frutos de processos de ingestão, são armazenados para aguardarem por transformações e consultas.

É importante frisar que um Data Warehouse recebe sempre dados estruturados já tratados, mas um Data Lake, que comumente oferece poder de processamento muito alto por todas as questões envolvendo ambientes Big Data, recebe dados brutos para que sejam tratados dentro dele. Estes dados então podem possuir muitas origens distintas e apresentar todos tipos de extensões possíveis, exigindo do profissional responsável organização em sua estrutura de diretórios.

Staging Area

Staging area, ou simplesmente Stage, é uma camada sequente a Raw utilizada para tratamento dos dados. Em geral, não há muita definição de sua estrutura, pois serve como apoio há processos quando os mesmos precisam gravar arquivos temporários, ou cópias, para manipulá-los.

Curated

Se na Raw temos dados brutos e na stage cópias e arquivos temporários sendo processados, na camada Curated estão armazenados dados tratados e pronto para uso.

Caso seja interessante ao negócio integrar seu Data Lake com seu Data Warehouse, é nesta camada que o DW seria inserido. Conectando assim ferramentas de visualização e de aprendizado de máquina para extração de valor.

Camadas extras

Não é incomum encontrar em Data Lakes camadas chamadas de Sandboxes, que são compostas por diretórios criados por/para usuários e cientistas de dados realizarem testes e experimentos.

A presença de uma camada Refined também é possível para armazenar dados enriquecidos por fontes externas e com mais tratamentos que os presentes na Curated (e claro, conectar aqui ferraementas de visualização, etc.).

Riscos e indicações

Como comentado anteriormente, o objetivo é aprensetar uma orientação geral e guia terminológico envolvendo arquiteturas de Data Lakes, mas menos regras que as vistas em shcemas de Data Warehouses, pois aqui há uma liberdade muito maior em suas construções.

O importante é investir tempo e recursos para definição de uma arquitetura que atenda as demandas do negócio, tendo camadas com papéis bem definidos visando boa manutetabilidade, e segurança em termos de controle de acesso e criptgrafia de dados sensíveis (LGPD).

Por fim, é necessário se atentar ao risco que um grande repositório com alto poder de processamento e armazenamento nos induz: ter um Data Lake transformado em um Data Swamp. Usamos esse termo para definir Data Lakes que, por receberem dados de tantas origens e tipos diferentes, possuem grande parcela de seu espaço consumida por dados que não estão sendo utilizados (lixo) ou nem se quer devidamente catalogádos, gerando apenas custo.

Spark

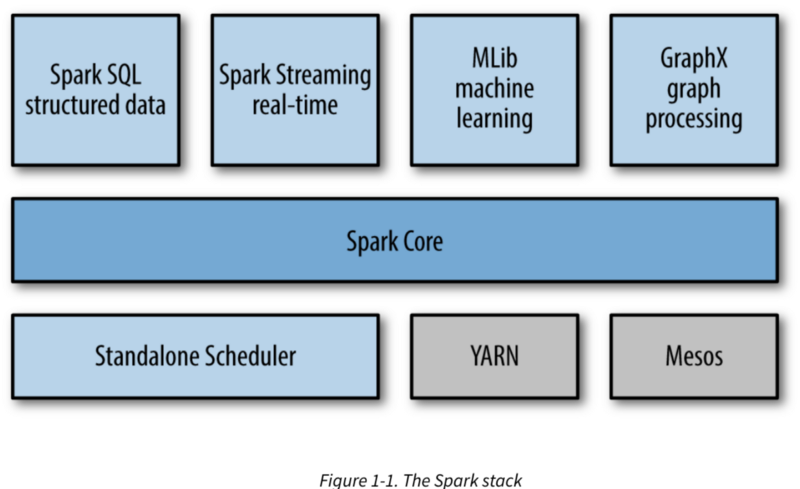
O que é?

“O Spark é um framework para processamento de Big Data construído com foco em velocidade, facilidade de uso e análises sofisticadas. Oferece APIs de alto nível em Java, Scala e Python, bem como um conjunto de bibliotecas que o tornam capaz de trabalhar de forma integrada, em uma mesma aplicação, com SQL, streaming e análises complexas, para lidar com uma grande variedade de situações de processamento de dados.”

Características

* Plataforma de computação em Cluster rápida, tolerante a falhas e de propósito geral.
* 100x mais rápido que Mapreduce em memória.
* 10x mais rápido que Mapreduce em disco.
* Compatível com Hadoop.
* Open Source.
* Desenvolvido em Scala.
* Aplicações em Java, Scala, Python e R.
* Bibliotecas para SQL, Streaming, Machine Learning e grafos.

Componentes



Apache Spark Core

Spark Core é o mecanismo de execução geral subjacente para a plataforma Spark em que todas as outras funcionalidades são construídas. Ele fornece computação em memoria e conjuntos de dados de referência em sistemas de armazenamento externo.

Spark SQL

Spark SQL é o módulo do Apache Spark para trabalhar com dados estruturados. As interfaces oferecidas pelo Spark SQL fornecem ao Spark mais informações sobre a estrutura dos dados e do cálculo que está sendo executado.

Streaming do Spark

Este componente permite que o Spark processe dados de streaming em real timel. Os dados podem ser ingeridos de muitas fontes, como Kafka, Flume e HDFS. Em seguida, os dados podem ser processados usando algoritmos complexos e enviados para sistemas de arquivos, bancos de dados e painéis ativos.

MLlib

Apache Spark é equipado com uma rica biblioteca conhecida como MLlib. Essa biblioteca contém uma ampla gama de algoritmos de aprendizado de máquina classificação, regressão, clustering e filtragem colaborativa. Ele também inclui outras ferramentas para construir, avaliar e ajustar canais de ML. Todas essas funcionalidades ajudam o Spark a escalar horizontalmente em um cluster.

GraphX

Spark também vem com uma biblioteca para manipular bancos de dados gráficos e realizar cálculos chamados GraphX. O GraphX unifica o processo ETL (Extract, Transform, and Load), análise exploratória e computação gráfica iterativa em um único sistema.

Arquitetura



Driver Program

É o ponto central e o ponto de entrada do Spark Shell (Scala, Python e R). O programa do driver executa a função main () do aplicativo e é o local onde o Contexto Spark é criado. O Driver Spark contém vários componentes - DAGScheduler, TaskScheduler, BackendScheduler e BlockManager, responsáveis pela tradução do código do usuário do spark em trabalhos de spark reais executados no cluster.

* Principal programa da sua aplicação Spark
* Servidor onde está sendo executado é chamado de nó Driver
* Processo é chamado de processo Driver
* Driver se comunica com o Cluster Manager para distribuir tarefas aos Executors

SparkContext

Spark SparkContext é um ponto de entrada para o Spark, desde da versão 1.x usado para criar programaticamente o RDD do Spark , acumuladores e variáveis de transmissão no cluster. Seu objeto sc é o padrão disponível no spark-shell e pode ser criado programaticamente usando a SparkContextclasse.

* É o ponto de entrada da sessão Spark
* Pode ser usado para criar RDDs, acumuladores e variáveis de transmissão no cluster
* Em modo local (spark-shell ou pyspark) um objeto SparkContext é criado automaticamente e a variável sc refere-se ao objeto SparkContext

Cluster Manager

É componente principal para gerenciamento do cluster spark.

* Spark tem a capacidade de trabalhar com uma infinidade de gerentes de cluster, incluindo YARN, Mesos e um gerenciador de cluster autônomo
* Um gerenciador de cluster autônomo consiste em dois daemons de longa duração, um nó mestre e um em cada um dos nós de trabalho.

Executer

Executors são responsáveis por executar tarefas e manter os dados na memória ou armazenamento em disco.

* Executors só são iniciados quando uma execução de trabalho começa em um Worker
* Cada aplicação possui seus próprios processos executors

Cluster

Spark Cluster

Spark Cluster é um gerenciador de cluster simples incluído no Spark que facilita a configuração de um cluster.

* Para executar em um cluster, o SparkContext pode se conectar a vários tipos de gerenciadores de cluster (o gerenciador standalone do Spark, Mesos ou YARN), que aloca recursos em aplicativos.
* Uma vez conectado, o Spark adquire executores em nós no cluster, que são processos que executam cálculos e armazenam dados para sua aplicação
* Em seguida, envia seu código de aplicativo para os executores.
* Finalmente, o SparkContext envia tarefas aos executores para executar.

Yarn Cluster

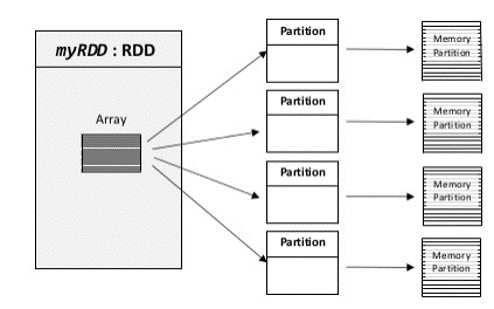
O YARN segue a arquitetura master e slave. O daemon master é chamado ResourceManagere o daemon slave é chamado NodeManager. Além dessa aplicação, o gerenciamento do ciclo de vida é feito por ApplicationMaster, que pode ser gerado em qualquer nó slave e permaneceria ativo durante a vida útil de uma aplicação. Quando o Spark é executado no YARN, ResourceManager desempenha a função do master do Spark e NodeManagers funciona como nós executores. Ao executar o Spark com YARN, cada executor do Spark é executado como um contêiner YARN.

* Possui Resource Manager (similar ao Master) para cada cluster e Node Manager (similar Slave) para cada nó no cluster.
* Aplicações no YARN são executados em containers.
* Processo driver do Spark atua como Application Master.
* Node Manager monitora recursos usados por containers e reportam ao Resource Manager.

RDD

Um RDD significa Conjuntos de dados distribuídos resilientes. É uma coleção de registros de partição somente leitura. RDD é a estrutura de dados fundamental do Spark. Ele permite que um programador execute cálculos na memória em grandes grupos de maneira tolerante a falhas . Assim, acelere a tarefa.

RDD era a principal API voltada para o usuário no Spark desde o seu início. No núcleo, um RDD é uma coleção distribuída imutável de elementos de seus dados, particionada em nós no cluster que pode ser operada em paralelo com uma API de baixo nível que oferece transformações e ações.



Exemplos de transformações:

* map() - Aplica uma função a cada elemento no RDD e retorna um RDD do resultado
* filter() - Retorna um RDD com os elementos que correspondem condição de filtro
* union() – Retorna um RDD contendo elementos de ambos os RDDs.

Exemplos de ações:

* collect() - Retornar todos os elementos do RDD.
* count() - Retorna o número de elementos do RDD.
* take(10) - Retorna 10 elementos do RDD.
* foreach(func) - Aplica a função fornecida a cada elemento do RDD.

Dataframe

Ao contrário de um RDD, os dados são organizados em colunas nomeadas. Por exemplo, uma tabela em um banco de dados relacional. É uma coleção imutável de dados distribuídos. O DataFrame no Spark permite que os desenvolvedores imponham uma estrutura em uma coleção distribuída de dados, permitindo abstração de nível superior.

* Um DataFrame é uma coleção distribuída de dados organizados em colunas nomeadas. É conceitualmente igual a uma tabela em um banco de dados relacional.
* Funciona apenas em dados estruturados e semiestruturados. Ele organiza os dados na coluna nomeada. Os DataFrames permitem que o Spark gerencie o esquema.
* A API da fonte de dados permite o processamento de dados em diferentes formatos (AVRO, CSV, JSON e sistema de armazenamento HDFS , tabelas HIVE , MySQL). Ele pode ler e gravar de várias fontes de dados mencionadas acima.
* Após a transformação no DataFrame, não é possível regenerar um objeto de domínio. Por exemplo, se você gerar testDF a partir de testRDD, não poderá recuperar o RDD original da classe de teste.

Spark-submit

É usado para iniciar, parar e monitorar uma aplicação localmente ou distribuído em um cluster Spark.

Executa pacotes Python e Java

$spark-submit --option value application jar | python file [applicationarguments]

$spark-submit --help

--executor-cores: Número de núcleos de processador para alocar em cada executor

--executor-memory: O tamanho máximo de memória para alocar a cada executor

--num-executors: O número total de containers YARN a serem alocados

--name: Nome do aplicativo

--jars: Jars adicionais a aplicação como bibliotecas externas

Local com 2 cores:

spark-submit --master local[2] --name myApp helloWorld.py

YARN Cluster:

spark-submit --master yarn --deploy-mode cluster --executor-memory 1G

--num-executors 2 --name myApp helloWorld.py

YARN Client:

spark-submit --master yarn-client --executor-memory 1G --num-executors 2 --name myApp helloWorld.py

Ao executar um programa Spark com Yarn, podemos acompanhar a execução pela interface web do Resource Manager no endereço quickstart.cloudera:8088

PySpark Comandos

Execute no terminal o seguinte comando:

pyspark

Operações básicas

Criamos um rdd (Conjuntos de dados distribuídos resilientes)

numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])

Mostra primeiro elemento:

numeros.first()

Mostra os 5 maiores:

numeros.top(5)

Mostra todos elementos:

numeros.collect()

Contar os elementos:

numeros.count()

Media dos numeros:

numeros.mean()

Somar os elementos:

numeros.sum()

Mostra maior elemento:

numeros.max()

Mostra menor elemento:

numeros.min()

Calcula desvio padrão:

numeros.stdev()

Analisar dados do Hive (DW)

Primeiramente é preciso copiar arquivos de configuração:

ls /usr/lib/hive/conf/hive-site.xml

cat /usr/lib/hive/conf/hive-site.xml

sudo cp /usr/lib/hive/conf/hive-site.xml /usr/lib/spark/conf/

Vamos abrir o Pyspark em um terminal:

pyspark

Criar contexto HiveContext

from pyspark.sql import HiveContext

contexto = HiveContext(sc)

Conectar no banco de dados na tabela:

banco = contexto.table("aula.categories")

banco.show()

Vamos registra a tabela no spark para ficar disponível para execução de querys

banco.registerTempTable("categories")

contexto.sql("select \* from categories").show()

contexto.sql("select sum(product\_price) from categories").show()

Vamos criar um Data Frame

Conectar no banco de dados na tabela:

banco = contexto.table("aula.categories")

banco.registerTempTable("categories")

A variável categoria é nosso dataframe:

categorias = contexto.sql("select \* from categories")

categorias.show()

categorias.show(100)

categorias.printSchema()

categorias.select('product\_name', 'product\_price').show()

categorias.select('product\_name', 'product\_price').distinct().show(30)

categorias.select('product\_name', 'product\_price').show()

categorias.filter(categorias.product\_name=='ENO Atlas Hammock Straps').show()

categorias.filter(categorias.product\_name=='ENO Atlas Hammock Straps').count()

Kafka

Apache Kafka é uma plataforma de streaming de eventos distribuída de código aberto desenvolvida em Scala e Java eusada para pipelines de dados de alto desempenho, análise de streaming e integração de dados.

Tem como objetivo fornecer uma plataforma unificada, de alta capacidade e baixa latência para tratamento de dados em tempo real. Sua camada de armazenamento é, essencialmente, uma "fila de mensagens de publishers/subscribers maciçamente escalável projetada como um log de transações distribuído" tornando-o altamente valioso para infra-estruturas corporativas que processam transmissão de dados.

A fim de tornar a solução ainda mais poderosa e complexa, é possível utilizá-la em conjunto com outras ferramentas de ingestão como o Apache Flume e o Spark Streaming.

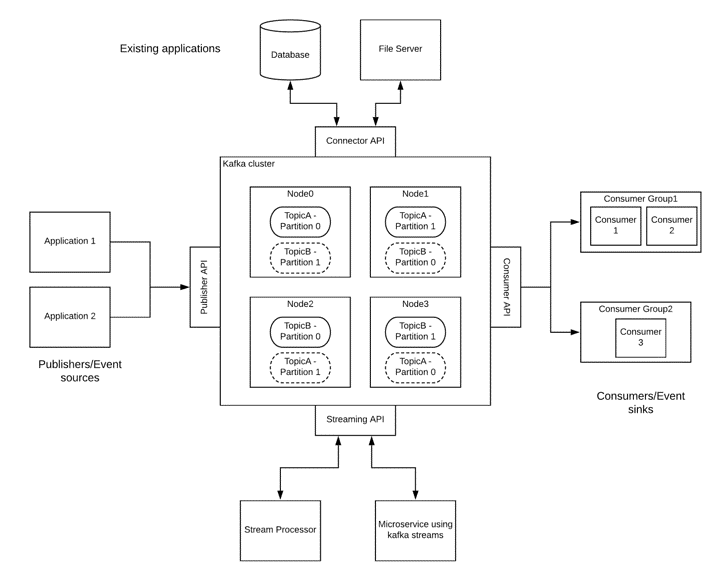
Características

* Plataforma de streaming distribuída.
* Permite publish e subscribe de streams semelhante a uma fila de mensagens.
* Tolerante a falhas.
* Real Time.
* Trabalha em cluster com um ou mais nós.
* Suporta milhões de mensagens por segundo.
* Entrega de mensagens de baixa latência.
* Suporta múltiplos clientes como Java, .NET, PHP, Ruby, python.
* Escrita e leitura em memória .
* Persiste todos os dados no disco.
* Integração nativa com Storm, Spark Streaming e Flume.
* Escalável.

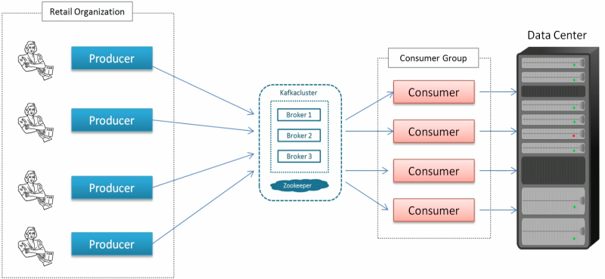
Arquitetura

O Kafka é usado com mais frequência para transmitir dados em real time para outros sistemas. Kafka é uma camada intermediária para desacoplar seus pipelines de dados em real time.

O core Kafka não é bom para cálculos diretos, como agregações de dados ou CEP. O streaming Kafka, que faz parte do ecossistema Kafka, fornece a capacidade de fazer análises em tempo real.



Kafka API



Tópico

O tópico é um canal lógico no qual os producer publicam mensagens e dos quais os consumer recebem mensagens.

* Onde os dados são armazenados.
* São armazenados em logs.
* São multi-assinantes, podendo ter zero, um ou muitos - consumidores.
* São divididos em partições.
* Os registros publicados no tópico são retidos por um período configurável de tempo.

Partição

Em um cluster Kafka, os Tópicos são divididos em Partições e também replicados entre os intermediários.

* Cada partição é uma sequência ordenada e imutável de registros que é anexado continuamente a um log de confirmação estruturado.
* Os registros nas partições recebem cada um um número de identificação sequencial chamado offset que identifica de forma exclusiva cada registro dentro da partição.
* São distribuídas pelos servidores no cluster Kafka. É replicada em um número configurável de servidores para tolerância a falha.
* É replicada em um número configurável de servidores para tolerância a falha.

Kafka Broker

* Kafka Cluster possui vários Broker (Instância de Kafka), se houve mais de um temos um Cluster Kafka.
* Cada Broker pode ter zero ou mais partições por tópico. Mantém o equilíbrio da carga.
* São stateless, usam o ZooKeeper para manter seu estado no cluster.
* Um Broker pode lidar com centenas de milhares de leituras e gravações por segundo e TB de mensagens sem perder desempenho.

Producer

Permite que um aplicativo publique registros nos tópicos.

* Envia os dados para os Brokers.
* Escolhe em qual tópico o dados vai ser escrito.
* Quando um novo Broker é iniciado, todos os producers enviam - uma mensagem para o ele.
* Producer não aguarda OK do Broker, envia as mensagens rapidamente para o Broker.

Consumer

Permite que um aplicativo se inscreva em um ou mais tópicos e processe os registro.

* Lêem os dados do Broker.
* Assinam um ou mais tópicos e consomem as mensagens publicadas.
* Tem o controle das mensagens que foram consumidas usando o offset da partição.
* Emite uma solicitação de offset assíncrona ao Broker para ter um buffer de bytes pronto para consumir.

ZooKeeper

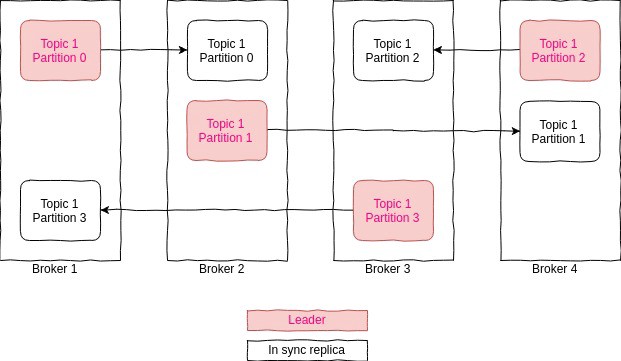
O Kafka broker usa o ZooKeeper com o visando gerenciamento e coordenação. Além disso, o utiliza para notificar o producer e o consumer sobre a presença de qualquer novo evento no sistema do Kafka ou falha no sistema. Assim que o tratador envia a notificação ou falha sobre producer e consumer, toma a decisão e começa a coordenar sua tarefa com outro corretor.

Características

* Parte importante de um Cluster Kafka.
* Coordenação entre os Brokers e os Consumers.
* Cluster Kafka compartilha informações através de um cluster Zookeeper.
* Kafka armazena metadados básicos no Zookeeper.
* Utilizado para fazer eleição de liderança dos Brokers e partições.
* Fornece uma visão sincronizada da configuração do Kafka Cluster.

Fator de replicação

Ao projetar um sistema Kafka, é sempre uma decisão sábia levar em consideração a replicação de tópicos. Como resultado, as réplicas de seus tópicos de outro brokers podem resolver a crise se um broker cair. Por exemplo, temos 3 corretores e 3 tópicos. O Broker 1 possui o Tópico 1 e a Partição 0, sua réplica está no Broker 2, e assim por diante. Possui um fator de replicação 2; significa que ele terá uma cópia adicional diferente da primária. Abaixo está a imagem do fator de replicação de tópicos:



Características

* Armazena os dados em disco.
* Por padrão são armazenados 7 dias.
* Aṕos expiração as mensagens são excluídas automaticamente .
* Configuração por tópico.
* Cleanup também por espaço.

Prática!

Para a demonstração a seguir, o Kafka 2.12-2.80 foi baixado e descompactado na pasta C:, em sistema operacional Microsoft Windows. Recomendo que todos comandos executados em prompts de comando sejam feitos tendo-os inciados em modo administrador.

Primeiramente, crie dentro da pasta do kafka um diretório 'data', e dentro dele as pastas 'kafka' e 'zookeeper'.

O segundo passo é apontar estes diretórios criados nas configurações do Kafka. Navegue até a pasta config e edite os arquivos 'server' e 'zookeper' da seguinte forma:

zookeeper: dataDir=C:/kafka\_2.12-2.8.0/data/zookeeper

server: log.dirs=C:/kafka\_2.12-2.8.0/data/kafka

Iniciando Zookeper

zookeeper-server-start.bat C:\kafka\_2.12-2.8.0\config\zookeeper.properties

Iniciando Kafka

kafka-server-start.bat C:\kafka\_2.12-2.8.0\config\server.properties

Criando tópico

kafka-topics --bootstrap-server localhost:9092 --create --topic teste

Listando tópico

kafka-topics --bootstrap-server localhost:9092 --list

Abra um novo terminal para abrirmos o console do producer

kafka-console-producer --broker-list localhost:9092 --topic teste

Em um novo terminal iniciaremos o consumer kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic teste

Insira mensagens no producer e veja como são imediatamente disponibilizadas no producer.

Para ler todas mensagens já inseridas, execute:

kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic teste --from-beginning

Descrever partição do tópico

kafka-topics.bat --bootstrap-server localhost:9092 --topic teste --describe

Alterar partição do tópico

kafka-topics.bat --bootstrap-server localhost:9092 --alter --topic teste --partitions 10

Carga de dados

Há diferentes formas de se fazer carga de dados e aqui discutiremos duas estratégias diferentes, muito importantes e difundidas.

Batch

Cargas feitas em formato de batch, ou lotes, implicam em acumulo de dados durante um período de tempo determinado para então processá-los.

Dentro deste formato encontram-se muitos ambientes de BI tradicionais, onde sistemas OLTP produzem e coletam dados durante o dia, para em janelas noturas acontecerem as extrações, transformações e cargas (ETL). Um fluxo como este de exemplo, tem como produto um Data Warehouse D-1, ou seja, com atraso de um dia das informações ali disponíveis para análises.

Streaming

Streaming por sua vez parte do pressuposto que datasets podem ser infinitos, e diferente dos batchs que aguardam janelas de execução e acumulam dados enquanto isso, neste modelo os dados são processados assim que disponibilizados, em tempo real ou o mais próximo disso possível.

O Kafka é o ferramenta que permite que isto seja possível, como visualizado no exemplo a pouco executado.

# Microsserviços

Ao longo dos anos 2010, uma grande mudança começou a acontecer em desenvolvimento de software, motivada pela inovação da implementação da arquitetura orientada a microsserviços. A principal motivação desse tipo de arquitetura é facilitar a manutenção das aplicações, aumentar sua tolerância a falhas e aumentar a escalabilidade do sistema.

A ideia desse tipo de arquitetura é construir aplicações através de componenetes independentes e autônomos que possuem um escopo reduzido e bem definido, enquanto a comunicação entre os diversos componenetes acontece por meio de **API's**.

A implementação dos componenetes que constituem um sistema orientado a microsserviços está intimamente ligada ao uso de **containers**, que, por sua vez, quando inseridos em ambientes complexos, precisam ser **orquestrados**. Abordaremos esses assuntos nos próximos tópicos.

## 1. Containers

Containers são unidades de software leves e "portáteis" que carregam a estrutura lógica básica que uma aplicação precisa para ser executada. Podemos pensar, para termos didáticos, que containers são máquinas virtuais extremamente simplificadas, que carregam somente a parte lógica do sistema operacional necessária para executar o nosso código de desejo.

É possível construir imagens de containers, que descrevem quais são os recursos e as configurações necessárias para o ambiente. Com uma imagem de um container em mãos, podemos executá-la em qualquer máquina. A aplicação que é executada no container tem os recursos básicos necessários para a sua execução. Dessa maneira, podemos garantir que as condições em que um código é executado em desenvolvimento são as mesmas em produção, evitando desperdício de tempo em configurações de ambiente.

Containers são um conceito teórico. Para operacionalizar esses conceitos na prática existem algumas ferramentas e, de longe, a mais utilizada é o Docker.

## 2. Docker

O Docker é a principal plataforma em que é possível desenvolver, encapsular e executar aplicações em containers. Vamos explorar um pouco de seu uso!

Para poder utilizar o Docker, é necessário instala-lo na sua máquina. Você pode encontrar um guia da [instalação aqui.](https://www.docker.com/products/docker-desktop)

***ATENÇÃO****: alguns exemplos só funcionarão se você já tiver o Docker instalado!*

O Docker possui um repositório global de imagens de containers, as quais possuem definições padrões que podemos utilizar como base para construir as nossas imagens, como imagens de sistemas operacionais baseados em Linux. Voce pode encontrar esse conteúdo em <https://hub.docker.com/>.

### 2.1. Dockerfile

Podemos criar imagens em Docker através da escrita de um arquivo chamado Dockerfile. O arquivo é construído passando instruções sobre o container a ser criado. Abaixo temos um exemplo simples de um Dockerfile.

FROM python:3.8

ENV curso="Let's Code!"

CMD python --version && echo "$curso"

Temos aqui a estrutura básica de um Dockerfile, onde cada comando define uma característica do container que queremos criar.

* FROM <nome\_da\_imagem>:<nome\_da\_versão>: define a imagem base em que o nosso container será montado. No exemplo, utilizamos uma imagem [disponível no docker hub](https://hub.docker.com/_/python) que já possui o Python instalado e escolhemos a versão 3.8.
* ENV <nome\_da\_variavel>:<valor\_da\_variavel>: define o valor de uma variável de ambiente no container
* CMD <comando\_para\_executar>: define o comando que será executado quando o container for iniciado. É como se digitássemos algo no terminal do container. No exemplo, imprimimos a versão do python instalado e o valor da variável de ambiente criada.

Esse arquivo base pode ser utilizado para construir uma imagem que será executada como um container. Para isso, utilizamos o comando "docker build" no terminal. Para o exemplo que construímos, essa é a estrutura das pastas:

exemplo-docker

│ Dockerfile

Temos dois passos, então, para executar o nosso container:

#### **Passo 1: docker build**

Precisamos fazer o Docker "compilar" o nosso Dockerfile numa imagem. O comando docker build fornece essa funcionalidade. Navegando para a pasta onde se encontra nosso arquivo, executamos:

$ docker build . -t imagem\_exemplo

[+] Building 2.8s (6/6) FINISHED

=> [internal] load build definition from Dockerfile 0.0s

=> => transferring dockerfile: 101B 0.0s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> [internal] load metadata for docker.io/library/python:3.8 2.6s

=> [auth] library/python:pull token for registry-1.docker.io 0.0s

=> CACHED [1/1] FROM docker.io/library/python:3.8@sha256:a0a734233420b17 0.0s

=> exporting to image 0.0s

=> => exporting layers 0.0s

=> => writing image sha256:354c184dc93dd962d068cd295208ebf329fab183e2559 0.0s

=> => naming to docker.io/library/imagem\_exemplo 0.0s

O parâmetro -t é usado para definir o nome da imagem. A nossa imagem foi criada com o nome "imagem\_exemplo"!

#### **Passo 2: docker run**

Agora, com a imagem criada a partir do Dockerfile, podemos executá-la com o comando docker run:

$ docker run imagem\_exemplo

Python 3.8.11

Let's Code!

Nosso container foi executado como o esperado: nos mostrou a versão do python instalada e depois imprimiu a nossa variável de ambiente. O programa executado foi simples, mas não tivemos que nos preocupar com a configuração do ambiente nem com a instalação do python, pois herdamos toda a configuração existente na imagem base. Para cada propósito, podemos escolher uma imagem base diferente consultando o docker hub.

## 3. API's

API é uma sigla para Application Programming Interface e é a maneira mais usual de construção para a comunicação entre dois sistemas. Lidamos diariamente com API's, mesmo sem saber: quando publicamos numa rede social, fazemos compras online ou agendamos uma visita a uma exposição. No contexto de microsserviços, como temos serviços independentes, é necessário controlar a interação entre esses componentes, tarefa na qual as API's se encaixam perfeitamente.

API's podem ser construída em diversas linguagem e estão associadas ao que chamamos de backend, recentemente tem sido comum encontramos desenvolvimento em **nodeJS** e **Python**. Vamos exemplificar a construção de uma API com Docker e Python!

### 3.1 API em Python

Como em praticamente toda aplicação, utilizamos bibliotecas para nos auxiliar. Não é diferente para a construção de API's em Python. Existem dois frameworks principais: [Django](https://docs.djangoproject.com/) e [Flask](https://flask.palletsprojects.com/). O Django é mais utilizado para sistemas web mais complexos, enquanto o Flask é preferido para soluções compactas e simples. Construiremos nosso exemplo utilizando o Flask.

Para que o flask execute de maneira apropriada, precisamos instalar dois pacotes: flask e flask-restful.

$ pip install flask flask-restful

Podemos, então, construir o arquivo api.py, da seguinte forma:

from flask import Flask

from flask\_restful import Resource, Api

# Constrói a aplicação e o objeto API.

app = Flask(\_\_name\_\_)

api = Api(app)

# Construção do recurso

class HelloWorld(Resource):

def get(self):

"""

O método get do recurso exemplo retorna um simples json com a chave "curso" e o valor "Let's Code!"

"""

return {"curso": "Let's Code!"}

# Atribui uma rota para o recurso

api.add\_resource(HelloWorld, '/')

if \_\_name\_\_ == '\_\_main\_\_':

# Inicia a API localmente, na porta 5000

app.run(debug=True, host='0.0.0.0', port=5000)

A estrutura do diretório esperado é a seguinte:

exemplo-api

│ api.py

Podemos iniciar a nossa API através do terminal:

$ python api.py

\* Serving Flask app 'api' (lazy loading)

\* Environment: production

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

\* Debug mode: on

\* Running on all addresses.

WARNING: This is a development server. Do not use it in a production deployment.

\* Running on http://192.168.0.133:5000/ (Press CTRL+C to quit)

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 140-928-522

Agora basta acessar o endereço [http://0.0.0.0:5000](http://0.0.0.0:5000/) e visualizamos a nossa simples API funcionando!

Mas a nossa aplicação está sendo executada na nossa máquina. Vamos utilizar o que aprendemos sobre o Docker e construir um container para a nossa API!

***ATENÇÃO****: finalize a execução da API pressionando ctrl+c no terminal em que executou o comando. Isso evitará conflitos quando utilizarmos o container!*

### 3.2 API no container

Precisamos construir o nosso Dockerfile, que vai descrever o que o container precisa para rodar a nossa aplicação. Como visto no item anterior, precisamos executar 2 principais passos: **instalar** as bibliotecas necessárias, e depois **executar** o arquivo api.py. Vamos manter o nosso diretório da seguinte maneira:

exemplo-api

│ api.py

| Dockerfile

Vamos descrever o nosso container em Docker!

FROM python:3.8

COPY . .

RUN pip install flask flask-restful

CMD python api.py

Vemos alguns comandos novos aqui:

* COPY <caminho\_local>:<caminho\_container>: copia um arquivo ou diretório da máquina local (que faz o build da imagem), para a imagem do container. Utilizamos aqui para mover todo o conteúdo da pasta origem para o container, o que leva, consequentemente, o nosso arquivo api.py. Esse é um comando utilizado com bastante frequência.
* RUN <comando\_para\_executar>: executa um comando no terminal do container na etapa de construção da imagem. É diferente do CMD, que define o comando a ser executado quando iniciamos o container através do docker run. O comando RUN só é executado quando usamos o docker build.

Com todas as configurações feitas, navegando no terminal para a pasta exemplo-api, que contém o arquivo api.py e o Dockerfile, podemos construir a nossa imagem!

$ docker build . -t exemplo\_api

[+] Building 6.2s (9/9) FINISHED

=> [internal] load build definition from Dockerfile 0.0s

=> => transferring dockerfile: 123B 0.0s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> [internal] load metadata for docker.io/library/python:3.8 2.2s

=> [auth] library/python:pull token for registry-1.docker.io 0.0s

=> [internal] load build context 0.0s

=> => transferring context: 604B 0.0s

=> CACHED [1/3] FROM docker.io/library/python:3.8@sha256:a0a734233420b17d9ab37125afc9d8217 0.0s

=> [2/3] COPY . . 0.0s

=> [3/3] RUN pip install flask flask-restful 3.7s

=> exporting to image 0.1s

=> => exporting layers 0.1s

=> => writing image sha256:e900ab49ddea4305af7156970f3eb06502d8b0f6a93c9f2f7bcd00ff9a5b0c9 0.0s

=> => naming to docker.io/library/exemplo\_api 0.0s

E finalmente, podemos executar o nosso container!

$ docker run -p 5000:5000 exemplo\_api

**OBS**: o parâmetro -p faz um mapeamento da porta da máquina local com a porta do container. Isso é necessário para conseguirmos "exergar" a porta em que a API está rodando dentro do container.

Agora basta acessarmos o mesmo endereço, [http://0.0.0.0:5000](http://0.0.0.0:5000/) e veremos a nossa API rodando, agora dentro do container!

Com a imagem pronta, podemos agora compartilhá-la. Qualquer pessoa com a imagem em mãos e o Docker instalado pode executá-la, independente das configurações locais da máquina, mesmo sem ter o Python ou flask instalados, ou tendo o Python em uma versão diferente. Isso é fantástico, não? Temos algumas maneira de compartilhar a imagem que criamos.

#### **docker save**

Esse comando salva a imagem em um arquivo, como da seguinte maneira:

$ docker save -o exemplo\_api.tar exemplo\_api

O que gera o arquivo exemplo\_api.tar, que pode ser compartilhado e carregado de maneira automática com o comando **docker load**:

$ docker load --input exemplo\_api.tar

Loaded image: exemplo\_api:latest

#### **docker push**

Podemos salvar a nossa imagem no Docker hub, tornando-a disponível globalmente. Para isso, utilizamos o comando **docker push**. É necessário possuir uma conta registrada e a imagem precisa ser renomeada para o seguinte formato:

$ docker tag exemplo\_api SEU\_USUARIO\_DOCKERHUB/exemplo\_api

Feito isso, basta executar o comando abaixo para publicar a imagem:

$ docker push SEU\_USUARIO\_DOCKERHUB/exemplo\_api

E pronto, sua imagem estará disponível para ser utilizada no FROM de outro Dockerfile, com o nome **SEU\_USUARIO\_DOCKERHUB/exemplo\_api**.

### 3.3 Considerações

A estrutura que vimos, apesar de parecer complexa, é **extremamente** útil para o desenvolvimento de aplicações orientadas a microsserviços. Podemos construir diversas imagens para executar tarefas específicas de um sistema. Como as imagens possuem escopo bem definido, é muito comúm que sejam reaproveitadas para novos projetos, fazendo o desenvolvimento acontecer de uma maneira muito mais rápida e escalável.

***Exercício****: explore o*[*Docker hub*](https://hub.docker.com/)*e veja a quantidade de diferentes imagens disponíveis. Pesquise por imagens de sistemas operacionais, bancos de dados, gerenciadores de fila etc. Você pode baixar imagens através do comando****docker pull****, e depois executá-las com o****docker run****.*

## 4. Orquestração

Quando falamos de plataformas completamente orientada a microsserviços, o volume de containers cresce consideravelmente, podendo atingir centenas e até milhares de instância ativas. Fazer a gestão de um grande volume passa a ser desafiador, por isso surgem plataformas de orquestração, que, além de outras funções secundárias, automatizam os processos operacionais relacionados à gestão de múltiplos containers, como provisionamento, deploy, escalonamento, rede e balanceadores. Essas plataformas fornecem o necessário para alocarmos projetos completos, facilitando muito o trabalho com esse tipo de tecnologia em ambientes produtivos.

A plataforma de orquestração mais utilizada é o **Kubernetes**, que provê diversos serviços úteis:

* **Balanceamento de carga**: o Kubernetes pode concentrar todas as requisições em um único endereço de IP por serviço e, de acordo com o número de requisições, as deriva para os containers menos sobrecarregados. Também escala o número de containers automaticamente, aumentando a quantidade em caso de alto volume e diminuindo em caso de baixo.
* **Implantações e rollbacks automáticos**: troca versões de um serviço de maneira automatizada, tirando gradualmente os containers com a versão antiga e implantando os containers da nova versão.
* **Controle de disco**: controla e divide espaço de disco físico entre múltiplos containers.
* **Controle de recursos**: limita uso máximo de CPU e memória dos containers de maneira automática e ajustável.
* **Tolerância a falhas**: em caso de falha em um container, faz a replicação de uma nova instância de maneira automatizada.
* **Gestão segura de senhas**: provê maneiras seguras de gestão de informações sensíveis, como senhas e tokens de autenticação.

Diversos provedores de cloud fornecem suas próprias soluções que utilizam Kubernetes, como é o caso do [Amazon EKS](https://kubernetes.io/). Você pode encontrar a [documentação do Kubernetes aqui](https://kubernetes.io/).

## Referências

* [Microsserviços na AWS](https://aws.amazon.com/pt/microservices/)
* [Website do Docker](https://www.docker.com/)
* [Website Kubernetes](https://kubernetes.io/)
* [O que são API's?](https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces)

**Introdução à Cloud Computing**

Vamos pensar em um computador, ele é composto basicamente por um **PROCESSADOR** para processar os dados, um disco rígido **HD** ou **SSD** para gravar as informações e a **MEMÓRIA RAM** para armazenar arquivos temporariamente.

O primeiro problema, que as pessoas costumam ter é a falta de espaço para armazenamento dos arquivos e programas. A solução era: comprar um HD externo. Inclusive usávamos HD externo para fazer backup dos arquivos, pensando em casos de falha.

Gerenciar os arquivos nos HDs externos começou a se tornar algo complicado, quem nunca perdeu uma pasta de fotos de um encontro em família?

Então surgiu o OneDrive, um serviço de armazenamento em nuvem da Microsoft. Você consegue acessar de qualquer lugar a qualquer momento. Logo em seguida, a Apple também começou a oferecer esse serviço com a chegada do iCloud, um dos mais utilizados.

Isso é armazenar dados em nuvem, ou seja, seus dados nos servidores de outra pessoa.

Você pode estar se perguntando \*"armazenar fotos e arquivos em nuvem é seguro?"\*.

Sim, os principais provedores de **Cloud Computing** tem serviços de segurança para proteção de informações essenciais contra roubo, vazamento de dados e exclusão.

Ambientes On Premises

Antes de discutirmos sobre ambientes Cloud, é importante compreendermos o que é um ambiente **On Premise** e a relação entre eles.​

O que é?

On-premise refere-se a infraestruturas privadas mantidas pela própria empresa em suas instalações e que podem ser mantidas de modo independente.​

Vantagens

* Liberdade para customizar hardware e software
* Independência de links de internet ou o gasto com links dedicados
* Maior controle sobre o parque tecnológico
* Baixa latência (rede local)
* Redução de exposição​

Desvantagens

* Custos Elevados
* Necessidade de manutenção
* Upgrades de Equipamentos
* Equipe técnica necessária para administrar
* Espaço Físico
* Complexidade e Custo elevador para contingência
* Backups duráveis complexos​

Ambientes Cloud

A computação em nuvem oferece uma forma de acessar servidores, armazenamento, bancos de dados e um conjunto amplo de serviços de aplicativos via Internet.​Um provedor de serviços em nuvem, é proprietário e faz a manutenção do hardware conectado à rede necessário para esses serviços de aplicativos, enquanto você provisiona e utiliza o que precisa por meio de um aplicativo web.​

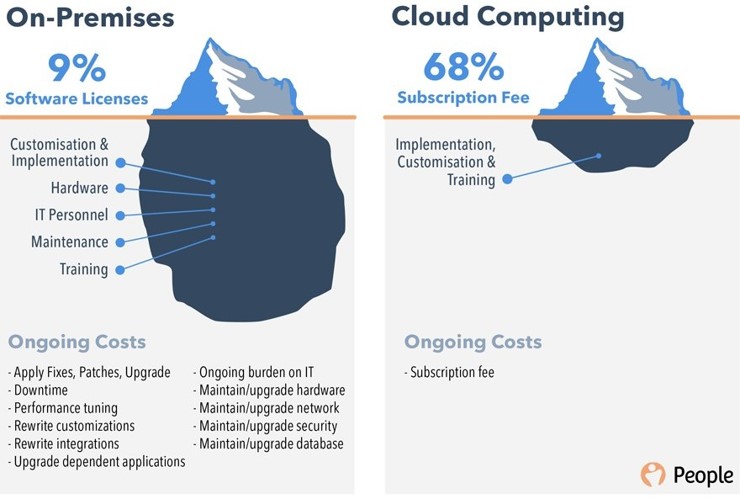
Vantagens

* Menor Investimento Inicial
* Troca de custo fixo por custo operacional
* Atualização de hardware e software constantes
* Controle minucioso do custo operacional
* Alta disponibilidade e ambientes de D.R
* Segurança reforçada e conformidade
* Tecnologias avançadas a disposição de todos​

Desvantagens

* Necessita uma maior vigilância, considerando uma exposição maior.
* Os custos podem ser alto já que uma variedade de serviços e servidores estão a sua disposição.
* Necessário um conhecimento mínimo para um melhor aproveitamento da plataforma e controle de custos
* Migrar de um player de nuvem para outro ainda é complexo, trabalhoso e algumas vezes ineficiente.​

Cloud x On Premises

​

Principais benefícios

Os seis principais benefícios para o uso de Cloud Computing aqui listados e discutidos são:​

Custo

A nuvem permite que você troque as despesas de capital (datacenters, servidores físicos etc.) por despesas variáveis e pague apenas pela TI à medida que ela for consumida.​Além disso, as despesas variáveis são muito menores do que as que você conseguiria por conta própria por causa da maior economia de escala.​

Escalabilidade

Antes da computação em nuvem, era preciso provisionar infraestrutura em excesso para garantir que você teria capacidade suficiente para processar suas operações comerciais durante o pico.​Agora, é possível provisionar a quantidade de recursos que você realmente precisa, sabendo que pode aumentar ou diminuir instantaneamente, conforme a necessidade da sua empresa. Isso reduz custos e melhora sua capacidade de atender às demandas dos usuários.​

Confiança

Os ambientes de Cloud Computing de grandes players trabalham sob um rigoroso controle de segurança e normas de conformidade.​Os níveis de segurança vão desde localização de seus Data Centers, acesso, hardware, trafego e software.​Cada player proverá aos seus clientes uma série de informações de conformidade e certificações de seus serviços, permitindo que seus clientes usem suas plataformas dentro dos padrões exigidos.​

Performance

Os principais players de Cloud Computing trabalham com hardwares customizados, para atingir a máxima performance entregue aos seus clientes e seus mais variados tipos de demanda.​O Hardware é fabricado de acordo com a especificação do player dentro de padrões rigorosos de fabricação e performance em todas as camadas, sendo: Armazenamento, Rede, Servidores (e todos os componentes), Firewalls, Cabos, Roteadores e etc.​

Agilidade

A nuvem possibilita uma inovação mais rápida, pois permite que você concentre seus recursos de TI valiosos no desenvolvimento de aplicativos que diferenciam sua empresa e transformam as experiências do cliente, em vez de concentrá-los no gerenciamento de infraestrutura e datacenters. Com a nuvem, você pode disponibilizar rapidamente os recursos conforme necessário, implantando centenas ou mesmo milhares de servidores em questão de minutos.​

Global

Com a nuvem, é possível implantar facilmente arquiteturas em vários locais físicos ao redor do mundo com apenas alguns cliques.​Isso significa que é possível se beneficiar de latência menor e uma experiência melhor simples e por um custo mínimo.​Assim como também atender a demandas de negócios com dados e processamento local dentro de regiões obedecendo a critérios de conformidade onde o dado não pode sair daquele país.​

Modelos de implementação

Basicamente é possível definir 2 pontos sobre os modelos de implementação disponíveis. Um deles diz respeito ao tipo de recurso a ser consumido na nuvem e o outro diz respeito a como se pretende implementar tais recursos.​Diante disso, é comum encontrar no mercado os termos IaaS (Infraestrutura as a Service), PaaS (Plataforma as a Service) e SaaS (Software as a Service). Esses termos são os mais conhecidos, porém não os únicos.​Usando todos esses termos e o que eles representam, as empresas ainda podem optar pelos modelos de implementação onde 100% dos recursos estão na nuvem ou ainda pelo modelo híbrido.​​

# AWS Lambda

O Lambda é um serviço de computação que permite executar código sem provisionar ou gerenciar servidores. O Lambda executa seu código em uma infraestrutura de computação de alta disponibilidade e executa toda a administração dos recursos computacionais, inclusive a manutenção do servidor e do sistema operacional, o provisionamento e a escalabilidade automática da capacidade e o monitoramento e o registro em log do código. Com o Lambda, você pode executar código para praticamente qualquer tipo de aplicação ou serviço de back-end. Tudo o que você precisa fazer é fornecer o código em uma das linguagens compatíveis com o Lambda.



## Desligar e Ligar servidores de Banco de Dados Amazon RDS com AWS Lambda

A definição de preço da AWS é semelhante à utilizada por serviços públicos, como água ou energia. Você paga apenas pelos serviços que utilizar e, quando parar de usá-los, não haverá custos adicionais nem taxas de cancelamento.

Para reduzir custos, as empresas que utilizam AWS, podem automatizar rotinas em seus ambientes de **desenvolvimento e homologação** para **desligar e ligar serviços quando necessário**.

## Proposta:

Utilizando a documentação [Funções Lambda Em Python Na AWS](https://letscode.com.br/blog/funcoes-lambda-em-python-na-aws) provisione dois serviços **Lambda** na AWS via **AWS Management Console**. Adicione um código em **Python** para **Desligar** e **Ligar** o Banco de Dados letscode.

### Referência

[Documentação do AWS Lambda](https://class.letscode.com.br/!https:/docs.aws.amazon.com/pt_br/lambda/?id=docs_gateway)