Instituto Tecnológico de Costa Rica

Ingeniería en Computación.

Bases de Datos II.

1er semestre

Apuntes #1

"Proyecto Opcional e Introducción a Docker y Kubernets"

ESTUDIANTE:

Edgar André Araya Vargas

2020142856

PROFESOR:

Gerardo Nereo Campos Araya

Grupo: 1

Fecha de entrega:

Viernes 17 de febrero del 2023.

PROYECTO OPCIONAL (% Extra).

Countries/States Crunjob (5%) - Stations Crunjob. (5%)

- Para el proyecto se puede utilizar **ActiveMQ** o **RabbitMQ** (más ligero).
- **Revisar bien la documentación** para reconocer los caracteres al ser interpretados por el proyecto.
- Reconocer líneas de string de los diferentes conjuntos de caracteres para procesarlos de manera correcta.
- Leer información adquirida y transcribirla a tabla/bases de datos.
- Leer documentación e investigar para **averiguar/crear conexión** entre *countries* y *states* (de igual manera para las tablas que se conectan).
- **Practica normal**: poner nombres de las tablas en singular, no plural (a Nereo no le molesta).
- **MD5** es una función matemática que se le corre a los datos y genera de salida un **hash** (resultado de una ecuación). *Representa el contenido del archivo y si se cambia el archivo, cambia el hash.*
- Hacer import de **biblioteca OS** para traer información como: día que fue procesado, MD5 del archivo y estado.
- Se debe verificar el MD5 para no procesar archivos procesados.
- **Todo esta vacío** y el Crunjob se corre de acuerdo con un calendario, se conecta a las URL, bajar el archivo, revisar si existen archivos dentro de *files*, y si no existe se crean, si existen se chequea el MD5.

- Hacer **updates del record** cuando el registro cambio. Se puede elegir borrar todos los registros pero genera problemas y no le gusta a Nereo.
- Buscar módulo de Python que permita recuperar información de una página web.

Orchestator Crunjob. (5%)

- Una vez el modulo traiga los archivos se publicará un mensaje (**string**) en una **cola.**
- Nosotros elegimos el mensaje a mandar.
- Usar String JSON ya que se acoplan de manera correcta a **Python.**
- Existe una (n) cantidad de **productore**s que envían mensajes una cola con almacenamiento limitado. Si la cola tiene que procesar mucha información empezara a crear una (n) cantidad de **consumidores** para leer la información.
- **Patrón constructor-consumidor** utilizado en el diseño de software y se utiliza en KAFKA, ActiveMQ, RabbitMQ y RocketMQ del cual no tenemos que preocuparnos ya que podemos solo entenderlo y utilizarlo con las herramientas.
- El **Orchestrator** es el que actúa como productor de mensajes para la cola llamada **TO-PROCESS**. El **Processor** actuara como consumidor.
- Orchestrator en la tabla de files publica una entrada (record) con información del archivo para revisar que existe. No tiene MD5 ya que no se descarga.

Processor. (5%)

- El lector, toma el mensaje de la cola y si nota modificaciones descarga el archivo se guarda en **Elasticsearch** y lo almacena completo en un archivo **JSON**.
- MD5 se actualiza y se cambia el estado. El mensaje se pasa a una cola nueva llamada **TO_PARSE**.
- Es un deployment entonces siempre están corriendo.
- Este consumidor queda bloqueado si no existe trabajo, en el momento que llega un mensaje trabaja inmediatamente.

> *"El resto de las secciones son repetitivas." - Nereo.*

Parser. (10%)

- Consumidor de TO_PARSE.
- Obtiene mensajes de la cola y utiliza instrucciones (**leer readme**) para interpretar todas las líneas. Esta es mas complicada.
- Una vez procesado se elimina por completo del índice *files* y se añade a un nuevo índice llamado *daily.*
- Productor de la cola TO_TRANSFORM.

Resto de Secciones (seguir instrucciones, revisar pipe-line).

- Elements Transformation. (10%)
- Station Transformation. (10%)
- Country Transformation. (5%)
- Elasticsearch Publisher. (10%)

-Recordar- Realizar visualización de datos con Kibana. Tomar en cuenta recomendaciones del profesor.

INTRODUCCIÓN KUBERNETS.

- Descargar primeramente Docker y dentro del mismo Kubernets.
- Un **sistema operativo monolítico** tiene una gran instalación ya que tratan de manejar todas las funciones de la computadora por si mismo. Intenta administrar todo.
- Para llevar a cabo la instalación se utilizaban **OS monolíticos físicos** pero pueden caer y eso resulta en un **gran problema.**
- Cuando un OS se instala este toma posesión del **hardware** de la computadora.
- Para cambiar el **método de OS físicos** que sostienen la instalación del OS monolítico se produjo un **hardware virtual** que permitiera correr un OS dentro de él. En vez de tener 4 OS físicos se tenía un -megahardware- con cuatro OS y cuatro hardwares virtuales. Esto es ineficiente ya que tiene varias **capas de abstracción** con diversas interpretaciones que toman recursos.
- **Tiempo Compartido**: Las tareas en una computadora no se corren de manera paralela se corren de manera serializada. Las aplicaciones pasan de Core en Core del CPU para utilizar **TIEMPO DE PROCESADOR DE MANERA COMPARTIDA**. Se pueden tener múltiples **threads** en un solo core
- Un **hypervisor/XEN/Openstack** es un OS que no es monolítico y no utiliza hardware virtual. Es un OS extremadamente liviano que permite **tiempo compartido** de máquinas virtuales a nivel de hardware, es decir, cada máquina virtual corre de manera separada con todo el hardware a su disposición. Esto genero problemas debido a problemas de **compatibilidad y sistemas operativos monolíticos gigantescos.**
- Para esto salió como solución los **containers** que utiliza **Docker**. Docker en vez de utilizar sistemas operativos gigantescos para programas y proyectos empezó a utilizar un **pequeño container** que contiene las bibliotecas que se necesitan para ejecutarse. **Docker** crea los contenedores y **Kubernets** gestiona los contenedores y los llama Pods, estos realizan una tarea.
- Siempre hay que crear un **Docker File**. Para todos los componentes a crear se necesita un docker file (en el proyecto igual). En estos existen repositorios de imágenes, así como la imagen oficial de Python **para poder concentrarnos únicamente en el trabajo**. Entre más pequeña la imagen, más liviano el contenedor por lo tanto más rápidas las operaciones.
- Solo se copia el código de la página oficial de Docker.

- Se necesita unas líneas de código para reproducir statements en el contenedor. El contenedor lo corre.
- Docker Desktop ingresa el comando Docker al cmd.
- Podemos utilizar este código después de registrarnos en docker para publicar una imagen en docker y que este empaquetada.

```
sudo docker login
sudo docker build -t nereo08/demo-image01 .
sudo docker images
sudo docker -v run nereo08/demo-image01
sudo docker push nereo08/demo-image01
```

- Cambiar *nereo08* por propio login.