

Proyecto Opcional

Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Redes (IC 4302)
Primer Semestre 2023



1. Objetivo General

- Desarrollar habilidades en tecnologías y lenguajes de programación que serán utilizados durante el curso.

2. Objetivos Específicos

- Implementar un sistema de visualización de datos climáticos.
- Automatizar una solución mediante el uso de [Docker](#), [Docker Compose](#), [Kubernetes](#) y [Helm Charts](#).
- Desarrollar arquitecturas de microservicios en Kubernetes.
- Implementar pipelines de procesamiento de datos.
- Desarrollar una solución que se coordine mediante colas de mensajes.
- Instalar y configurar servicios de bases de datos relacionales.
- Instalar y configurar servicios de bases de datos NoSQL.

3. Datos Generales

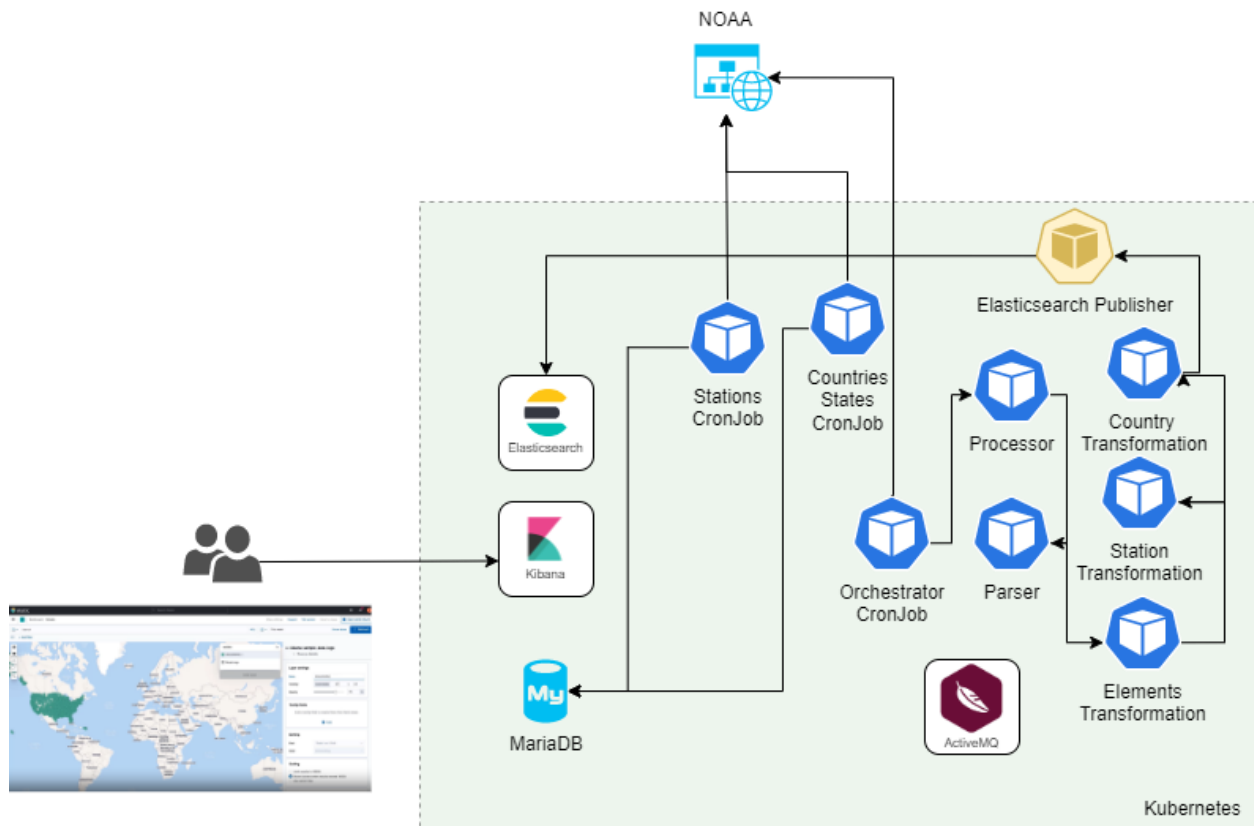
- El valor del proyecto: 15%
- Nombre del proyecto: WindyUI.
- La tarea debe ser implementada en grupos de máximo 5 personas.
- **El proyecto es opcional, el porcentaje obtenido será sumado a la nota final del curso.**
- La **fecha de entrega** es 03/03/2023 antes de las 6:00 pm.
- Cualquier indicio de copia será calificado con una nota de 0 y será procesado de acuerdo con el reglamento. La copia incluye código/configuraciones que se puede encontrar en Internet y que sea utilizado parcial o totalmente sin el debido reconocimiento al autor.
- La revisión es realizada de forma virtual mediante citas en las cuáles todos los y las integrantes del grupo deben estar presentes, el proyecto debe estar completamente automatizado, el profesor decide en que computadora probar.
- Se debe incluir documentación con instrucciones claras para ejecutar el proyecto.
- Se deben incluir pruebas unitarias para verificar los componentes individuales de su proyecto, si estas no están presentes se obtendrá una nota de 0.
- Se espera que todos y todas las integrantes del grupo entiendan la implementación suministrada.
- Se deben seguir buenas prácticas de programación en el caso de que apliquen, entre ellas:
 - ◆ Documentación interna y externa.
 - ◆ Estándares de código.
 - ◆ Diagramas de arquitectura.
 - ◆ Diagramas de flujo

◆ Pruebas unitarias.

- Toda documentación debe ser implementada en Markdown.
- Si el proyecto no se encuentra automatizado obtendrá una nota de 0. Si el proyecto no se entrega completo, la porción que sea entregada debe estar completamente automatizada, de lo contrario se obtendrá una nota de 0.
- En caso de no entregar documentación se obtendrá una nota de 0.
- El email de entrega debe contener una copia del proyecto en formato tar.gz y un enlace al repositorio donde se encuentra almacenado, debe seguir los lineamientos en el programa de curso.
- Los grupos de trabajo se deben autogestionar, la persona facilitadora del curso no es responsable si un miembro del equipo no realiza su trabajo.
- En consulta o en mensajes de correo electrónico o Telegram, como mínimo se debe haber leído del tema y haber tratado de entender sobre el mismo, las consultas deben ser concretas y se debe mostrar que se intentó resolver el problema en cuestión.

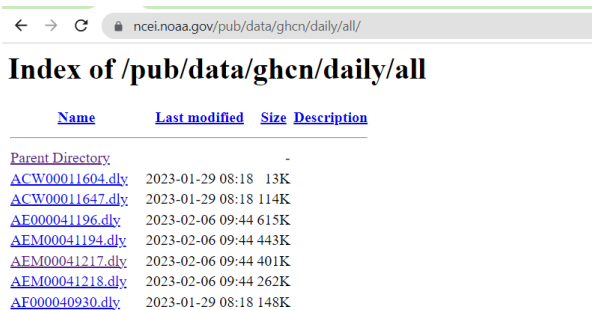
4. Descripción

Utilizando los datos disponibles en la página de [National Oceanic and Atmospheric Administration \(NOAA\)](https://www.noaa.gov/), implementaremos una pequeña aplicación para la visualización de datos del clima a nivel mundial, la misma tendrá la siguiente arquitectura:



La aplicación estará conformada por una serie de microservicios que deben ser implementados en lenguaje de programación Python, estos serán conocidos como la capa *stateless* y otra capa que será conocida como *stateful*, que estará conformada por los servicios de bases de datos y coordinación. El funcionamiento será el siguiente:

- Countries/States CronJob: Es un objeto de Kubernetes tipo [CronJob](#), que ejecuta un pod una vez al día, el mismo se encarga de leer los archivos [ghcnd-countries.txt](#) y [ghcnd-states.txt](#) y cargarlos en las tablas de MariaDB weather.countries y weather.states, la información de cómo interpretar estos archivos se puede encontrar en [readme.txt](#), recuerden crear las debidas relaciones entre ambas tablas.
 - Se deberá guardar una entrada en la tabla weather.files, que contenga el nombre del archivo (en este caso [ghcnd-countries.txt](#) y [ghcnd-states.txt](#)), URL, día en que fue procesado, md5 del archivo y estado (debería ser procesado).
 - Antes de iniciar con el procesamiento de este componente, se debe verificar si el archivo existe en la tabla weather.files, en caso de ser así se debe calcular el md5 del mismo y si es igual al que se encuentra en la tabla weather.files, el archivo no se procesara.
- Stations CronJob: Es un objeto de Kubernetes tipo [CronJob](#), que ejecuta un pod una vez al día, el mismo se encarga de leer el archivo [ghcnd-stations.txt](#), cada uno de los registros en el archivo debe ser procesado y debe ser agregado en MariaDB en la tabla weather.stations, como parte del procesamiento de cada registro, se debe:
 - Interpretar la información de acuerdo con la documentación en el archivo [readme.txt](#), sección “IV. FORMAT OF ghcnd-stations.txt”
 - Los datos se deben guardar en la tabla weather.stations y se debe generar una relación contra las tablas weather.countries y weather.states.
 - Se deberá guardar una entrada en la tabla weather.files, que contenga el nombre del archivo (en este caso [ghcnd-stations.txt](#)), URL, día en que fue procesado, md5 del archivo y estado (debería ser procesado).
 - Antes de iniciar con el procesamiento de este componente, se debe verificar si el archivo existe en la tabla weather.files, en caso de ser así se debe calcular el md5 del mismo y si es igual al que se encuentra en la tabla weather.files, el archivo no se procesara.
- Orchestrator CronJob: Es un objeto de Kubernetes tipo [CronJob](#), que ejecuta un pod una vez al día, el mismo se encarga de listar todos los archivos en el folder [/pub/data/ghcn/daily/all](#)



Name	Last modified	Size	Description
Parent Directory		-	
ACW00011604.dly	2023-01-29 08:18	13K	
ACW00011647.dly	2023-01-29 08:18	114K	
AEM00041196.dly	2023-02-06 09:44	615K	
AEM00041194.dly	2023-02-06 09:44	443K	
AEM00041217.dly	2023-02-06 09:44	401K	
AEM00041218.dly	2023-02-06 09:44	262K	
AF000040930.dly	2023-01-29 08:18	148K	

Por cada archivo, se publicara un mensaje en una cola de archivos por procesar, la misma se llamará TO_PROCESS, que se encuentra en ActiveMQ y se agregará o actualizará un record en la tabla weather.files, en el caso de no existir, el record tendrá la siguiente información:

- Nombre del archivo, por ejemplo AEM00041217.dly
- URL: <https://www.ncei.noaa.gov/pub/data/ghcn/daily/all/AEM00041217.dly>
- Día que fue procesado: fecha actual
- Estado: LISTADO
- MD5: null

En caso de que el archivo exista, se actualizarán los campos “Día que fue procesado” y “Estado”, de acuerdo con la información anterior.

- Processor: Es un objeto de Kubernetes tipo [Deployment](#), el mismo podrá tener una o más replicas, estas replicas estarán escuchando por trabajo en la cola de ActiveMQ llamada TO_PROCESS, cuando existe trabajo, sucede lo siguiente:

- Toma el mensaje de la cola TO_PROCESS y descarga el archivo del URL.
- Calcula el MD5, si el MD5 es igual al valor almacenado en weather.files, no existen datos nuevos, se actualiza el Estado a PROCESADO y termina el procesamiento de este.
- En caso de que el MD5 sea diferente, el archivo se almacena en Elasticsearch, en un índice llamado *files*, el documento tendrá el siguiente formato:

```
1  {
2      "fileName": "{NOMBRE ARCHIVO}",
3      "contents": "{DATOS DENTRO DEL ARCHIVO}"
4  }
```

- Se actualiza el MD5 y estado en weather.files, el estado debería de ser DESCARGADO.
- Por último, se publicará un mensaje en la cola de ActiveMQ llamada TO_PARSE, con el nombre del archivo, con esto termina el procesamiento de este mensaje.

Cuando este componente termina el procesamiento de un mensaje, regresa a la cola TO_PROCESS, para seguir esperando por más trabajo. Este es el comportamiento de todos los componentes de la capa stateless.

- Parser: Es un objeto de Kubernetes tipo [Deployment](#), el mismo podrá tener una o más replicas, estas replicas estarán escuchando por trabajo en la cola de ActiveMQ llamada TO_PARSE, cuando existe trabajo, sucede lo siguiente:

- Toma el mensaje de la cola TO_PARSE y recupera el archivo del índice *files* de Elasticsearch.
- Utilizando las instrucciones en [readme.txt](#), en la sección “III. FORMAT OF DATA FILES (".dly" FILES)” se interpretaran los registros y se generaran documentos JSON, por ejemplo, el archivo [AEM00041217.dly](#) contiene 1519 líneas, cada una de estas líneas se debe interpretar con la siguiente tabla (leer detenidamente [readme.txt](#)):

Variable	Columns	Type
ID	1-11	Character
YEAR	12-15	Integer
MONTH	16-17	Integer
ELEMENT	18-21	Character
VALUE1	22-26	Integer
MFLAG1	27-27	Character
QFLAG1	28-28	Character
SFLAG1	29-29	Character
VALUE2	30-34	Integer
MFLAG2	35-35	Character
QFLAG2	36-36	Character
SFLAG2	37-37	Character
.	.	.
.	.	.
VALUE31	262-266	Integer
MFLAG31	267-267	Character
QFLAG31	268-268	Character
SFLAG31	269-269	Character

Esto generara un documento JSON como el siguiente:

```
{
  "fileName": "{NOMBRE ARCHIVO}",
  "data": [
    {
      "station_id": "",
      "date": "",
      "type": "",
      "value": "",
      "mflag": "",
      "qflag": "",
      "sflag": ""
    },
    {
      "station_id": "",
      "date": "",
      "type": "",
      "value": "",
      "mflag": "",
      "qflag": "",
      "sflag": ""
    }
  ]
}
```

Potencialmente el campo “data” para el archivo [AEM00041217.dly](#), contendrá un total de 1519 * 31 objetos (leer detenidamente [readme.txt](#)). El campo date deberá contener la fecha con el siguiente formato “MM/DD/YYYY”.

- Una vez que el archivo anterior ha sido creado, se publica en el índice llamado *daily* de Elasticsearch y se borra el índice *files*.
- Se actualiza estado en weather.files, el estado debería de ser “PARSEADO”.
- Se publicará un mensaje en la cola de ActiveMQ llamada TO_TRANSFORM, con el nombre del archivo, con esto termina el procesamiento de este mensaje.

Cuando este componente termina el procesamiento de un mensaje, regresa a la cola TO_PARSE, para seguir esperando por más trabajo.

- Elements Transformation: Es un objeto de Kubernetes tipo [Deployment](#), el mismo podrá tener una o más replicas, estas replicas estarán escuchando por trabajo en la cola de ActiveMQ llamada TO_TRANSFORM, cuando existe trabajo, sucede lo siguiente:
 - Toma el mensaje de la cola TO_TRANSFORM y recupera el archivo del índice *daily* de Elasticsearch.
 - Para cada uno de los objetos en el campo data, realizara las siguientes acciones:
 - Mediante el campo date, obtendrá un nuevo campo llamado year y otro llamado month. Por ejemplo, para el objeto:

```
{
  "station_id": "AEM00041217",
  "date": "01011983",
  "type": "TMAX",
  "value": "298",
  "mflag": "B",
  "qflag": "M",
  "sflag": "A"
}
```

El objeto resultante será:

```
{
  "station_id": "AEM00041217",
  "date": "01011983",
  "month": "01",
  "year": "1983",
  "type": "TMAX",
  "value": "298",
  "mflag": "B",
  "qflag": "M",
  "sflag": "A"
}
```

- Mediante el campo `station_id` y siguiendo las instrucciones en [readme.txt](#), se obtienen los campos `FIPS_country_code`, `network_code` y `real_station_id`. Siguiendo con el ejemplo anterior se obtendrá:

```
{
  "station_id": "AEM00041217",
  "date": "01011983",
  "month": "01",
  "year": "1983",
  "type": "TMAX",
  "value": "298",
  "mflag": "B",
  "qflag": "M",
  "sflag": "A",
  "FIPS_country_code": "AE",
  "network_code": "M",
  "real_station_id": "00041217"
}
```

- Mediante el campo `type` y siguiendo las instrucciones en [readme.txt](#), se obtiene el campo `type_name`, siguiendo con el ejemplo anterior se obtendrá:

```
{
  "station_id": "AEM00041217",
  "date": "01011983",
  "month": "01",
  "year": "1983",
  "type": "TMAX",
  "value": "298",
  "mflag": "B",
  "qflag": "M",
  "sflag": "A",
  "FIPS_country_code": "AE",
  "network_code": "M",
  "real_station_id": "00041217",
  "type_name": "Maximum temperature (tenths of degrees C)"
}
```

- Una vez que se han inferidos los campos, los datos se actualizan en el índice llamado *daily* de Elasticsearch.

- Se actualiza estado en weather.files, el estado debería de ser “TRANSFORMADO”.
 - Se publicará un mensaje en la cola de ActiveMQ llamada TO_STATION, con el nombre del archivo, con esto termina el procesamiento de este mensaje.
- Station Transformation: Es un objeto de Kubernetes tipo [Deployment](#), el mismo podrá tener una o más replicas, estas replicas estarán escuchando por trabajo en la cola de ActiveMQ llamada TO_STATION, cuando existe trabajo, sucede lo siguiente:
 - Toma el mensaje de la cola TO_TRANSFORM y recupera el archivo del índice *daily* de Elasticsearch.
 - Para cada uno de los objetos en el campo data, realiza las siguientes acciones:
 - Mediante el campo station_id, realiza una consulta a la tabla de MariaDB llamada weather.stations y le agrega al objeto los siguientes campos:
 - latitude
 - longitude
 - elevation
 - state
 - name
 - gsn_flag
 - hcn_crn_flag
 - wmo_id

Siguiendo con el ejemplo anterior, el objeto resultante es:

```
{
  "station_id": "AEM00041217",
  "date": "01011983",
  "month": "01",
  "year": "1983",
  "type": "TMAX",
  "value": "298",
  "mflag": "B",
  "qflag": "M",
  "sflag": "A",
  "FIPS_country_code": "AE",
  "network_code": "M",
  "real_station_id": "00041217",
  "type_name": "Maximum temperature (tenths of degrees C)",
  "latitude": "24.4330",
  "longitude": "54.6510",
  "elevation": "26.8",
  "state": "",
  "name": "ABU DHABI INTL",
  "gsn_flag": "",
  "hcn_crn_flag": "",
  "wmo_id": "41217"
}
```

- Una vez que se han inferidos los campos, los datos se actualizan en el índice llamado *daily* de Elasticsearch.
 - Se actualiza estado en weather.files, el estado debería de ser “CON_ESTACION”.
 - Se publicará un mensaje en la cola de ActiveMQ llamada TO_COUNTRY, con el nombre del archivo, con esto termina el procesamiento de este mensaje.
- Country Transformation: Es un objeto de Kubernetes tipo [Deployment](#), el mismo podrá tener una o más replicas, estas replicas estarán escuchando por trabajo en la cola de ActiveMQ llamada TO_COUNTRY, cuando existe trabajo, sucede lo siguiente:

- Toma el mensaje de la cola TO_COUNTRY y recupera el archivo del índice *daily* de Elasticsearch.
- Para cada uno de los objetos en el campo data, realizara las siguientes acciones:
 - Mediante el campo FIPS_country_code y una consulta a la tabla de MariaDB weather.countries, se agrega el campo country_name al objeto.
 - Mediante el campo state y una consulta a la tabla de MariaDB llamada weather.states se agrega el campo state_name al objeto.

Continuando con el ejemplo anterior, el objeto resultante debería ser:

```
{
  "station_id": "AEM00041217",
  "date": "01011983",
  "month": "01",
  "year": "1983",
  "type": "TMAX",
  "value": "298",
  "mflag": "B",
  "qflag": "M",
  "sflag": "A",
  "FIPS_country_code": "AE",
  "country_name": "United Arab Emirates",
  "state_name": "",
  "network_code": "M",
  "real_station_id": "00041217",
  "type_name": "Maximum temperature (tenths of degrees C)",
  "latitude": "24.4330",
  "longitude": "54.6510",
  "elevation": "26.8",
  "state": "",
  "name": "ABU DHABI INTL",
  "gsn_flag": "",
  "hcn_crn_flag": "",
  "wmo_id": "41217"
}
```

- Una vez que se han inferidos los campos, los datos se actualizan en el índice llamado *daily* de Elasticsearch.
- Se actualiza estado en weather.files, el estado debería de ser "CON_PAIS".
- Se publicará un mensaje en la cola de ActiveMQ llamada TO_PUBLISH, con el nombre del archivo, con esto termina el procesamiento de este mensaje.
- Elasticsearch Publisher: Es un objeto de Kubernetes tipo [Deployment](#), el mismo podrá tener una o más replicas, estas replicas estarán escuchando por trabajo en la cola de ActiveMQ llamada TO_PUBLISH, cuando existe trabajo, sucede lo siguiente:
 - Toma el mensaje de la cola TO_PUBLISH y recupera el archivo del índice *daily* de Elasticsearch.
 - Para cada uno de los objetos en el campo data, publicara un documento en el índice *weather-data* de Elasticsearch, es importante mencionar que se debe leer cuidadosamente la documentación de Elasticsearch para utilizar los tipos de campos adecuados por ejemplo para [coordenadas](#) y [fechas](#)..
 - Se actualiza estado en weather.files, el estado debería de ser "PROCESADO".
 - Se borrará el archivo del índice *daily* de Elasticsearch.
 - Para este punto el procesamiento ha sido completado.

A parte de la preparación de los datos, se debe agregar dos visualizaciones en Kibana:

- Mostrará un [mapa](#), se podrá seleccionar el tipo de medición y el año además de esto se podrá seleccionar un rango de fechas. Esta visualización permitirá observar el comportamiento de las diferentes mediciones en el tiempo, por ejemplo podríamos ver cual fue la temperatura promedio en Costa Rica para el año 2022 y comparar esta con el año 1960.
- Visualización Personalizada: Cada grupo mediante la exploración de los datos, deberá crear una visualización que utilice mapas.

Es importante mencionar que todos los componentes deben estar debidamente automatizados con Helm Charts y Docker.

Documentación

Se deberá entregar una documentación que al menos debe incluir:

- Instrucciones claras de como ejecutar su proyecto.
- Pruebas realizadas, con pasos para reproducirlas.
- Resultados de las pruebas unitarias.
- Recomendaciones y conclusiones (al menos 10 de cada una).
- La documentación debe cubrir todos los componentes implementados o instalados/configurados, en caso de que algún componente no se encuentre implementado, no se podrá documentar y tendrá un impacto en la completitud de la documentación.
- Referencias bibliográficas donde aplique.

Imaginen que la documentación está siendo creada para una persona con conocimientos básicos en el área de computación y ustedes esperan que esta persona pueda ejecutar su proyecto y probarlo sin ningún problema, el uso de imágenes, diagramas, code snippets, videos, etc. son recursos que pueden ser útiles.

La documentación debe estar implementada en Markdown y compilada en un PDF.

5. Recomendaciones

- Utilizar telnet/curl/[postman](#) para interactuar con los componentes y verificar que los mismos están funcionando correctamente.
- Utilizar [Docker Desktop](#) para instalar Kubernetes en Windows.
- Utilizar [Minikube](#) para ejecutar Kubernetes en Linux
- Realicen peer-review/checkpoints semanales y no esperen hasta el último momento para integrar su aplicación.
- Crear un repositorio de GitHub e invitar al profesor.
- Realizar commits y push regulares.
- Limitar la cantidad de tipos de mediciones a utilizar, se recomienda filtrar y utilizar únicamente

PRCP, SNOW, SNWD, TMAX, TMIN y RHM.

- Limitar la cantidad de años a procesar, esto por cuestiones de tiempo y recursos, podrían hacer el procesamiento de unos cuantos años, el NOAA tiene datos desde antes del año 1900.

6. Entregables

- Documentación.
- Documento con las pruebas de rendimiento.
- Docker files, Docker Compose files y Helm Charts junto con todos los archivos/scripts requeridos para ejecutar su proyecto.

7. Evaluación

Funcionalidad / Requerimiento	Porcentaje
Documentación	5%
Visualizaciones	20%
Implementación (*): <ul style="list-style-type: none">• Instalación de motores (10%)• Station CronJob (5%)• Countries/States CronJob (5%)• Orchestrator CronJob (5%)• Processor (5%)• Parser (10%)• Elements Transformation (10%)• Station Transformation (10%)• Country Transformation (5%)• Elasticsearch Publisher (10%)	75%
	100%

(*) Todo tiene que estar debidamente automatizado, de lo contrario se calificará con una nota de 0.