



**Automatizando escalamiento y manejo de recursos
para procesamiento por lotes del cubo de datos
colombiano utilizando
AWS Batch**

Andrés Camilo Zuleta Calderón

Universidad de los Andes
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas y Computación
Bogotá D.C., Colombia
Diciembre, 2018



**Automatizando escalamiento y manejo de recursos para
procesamiento por lotes del cubo de datos colombiano utilizando
AWS Batch**

Andrés Camilo Zuleta Calderón

Proyecto de Grado
Ingeniería de Sistemas y Computación

Asesores:
Harold Enrique Castro Barrera, PhD
Mario José Villamizar cano, M.Sc.

Universidad de los Andes
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas y Computación
Bogotá D.C., Colombia
Diciembre, 2018

“If you cannot do great things, do small things in a great way.”

-Napoleon Hill

Agradecimientos:

A mis padres, por apoyarme en mi decisión de estudiar una carrera no convencional y que de dónde vengo no tiene ningún prestigio.

A mi abuela, por ayudarme a costear mis estudios y siempre haber velado por mi éxito. Te amo donde quiera que estés.

A mi Tía, por ser un apoyo para mí y hacer de mi estancia en Bogotá la más amena de todas.

A Sergio, por ser mi compañero durante todo este trayecto y por todas esas traspasadas que pasamos fantaseando con el día en que culminaríamos el pregrado.

A Harold, por haberme enseñado durante los cursos que me motivaron a escoger este proyecto y ser una guía para mí durante él.

Finalmente, a Mario por acogerme en este proyecto, guiarme durante él y confiar en mis capacidades como ingeniero.

Resumen:

Los procesamientos por lotes representan una parte de gran importancia para procesos analíticos y de procesamiento de datos dentro de toda organización. Sin embargo existen retos asociados a estos , principalmente en el aprovisionamiento de recursos. El instituto de hidrología, meteorología y de estudios territoriales (IDEAM) colombiano, presenta una serie de estudios que se apoyan en el uso de procesamientos en batch para desarrollarlos de forma satisfactoria. No obstante, en estos momentos presentan limitantes de sus recursos computacionales disponibles y por tanto de escalabilidad.

El propósito de este proyecto es ofrecer al IDEAM una solución escalable, altamente disponible y auto-gestionada de procesamiento por lotes a la vez que optimizamos factores como el desempeño y los costos utilizando el servicio de procesamiento por lotes en la nube AWS Batch.

Batch processing represents a really important piece for analytics and data processing within every organization. However there are some challenges associated with them, mostly related with resource provisioning. The Colombian Institute of Hydrology, Meteorology and Environmental Studies (IDEAM), shows a series of studies supported by batch processing in order to develop them successfully. Nonetheless currently they present limitations due to their available computational resources and therefore scalability problems.

The purpose of this project is to offer to the IDEAM a scalable, highly available and self-managed solution for batch processing at the same time we optimize items such as the performance and cost-efficiency through the use of a cloud batch processing service, AWS Batch.

Tabla de contenido

Introducción	9
Estado del Arte	11
Objetivos	17
Objetivos Generales.....	17
Objetivos Específicos	17
Metodología	18
Alcance del proyecto	18
Prueba de Concepto	19
Migrando el cubo de datos a AWS Batch	20
Montaje Experimental.....	27
Diseño del ambiente de pruebas	27
Experimentos y pruebas propuestas.....	28
Diseño de experimentos sobre AWS Batch	29
Resultados y Análisis	30
Escalamiento	30
Descripción.....	30
Entorno.....	30
Resultados	32
Conclusiones	36
Número de contenedores por máquina	37
Descripción.....	37
Entorno.....	38
Resultados	39
Conclusiones	43
Entorno de cómputo optimizado – Una Sola Definición	44
Descripción.....	44
Entorno.....	44
Resultados	44
Conclusiones	48
Entorno de cómputo optimizados – Múltiples Definiciones	49
Descripción.....	49

Entorno.....	49
Resultados	50
Conclusiones	52
Dos colas - Un Entorno de Cómputo	52
Descripción.....	52
Entorno.....	53
Resultados	53
Conclusiones	56
Una cola - Múltiples entornos.....	56
Descripción.....	56
Entorno.....	56
Resultados	57
Conclusiones	59
Entorno de Cómputo Spot con tipo de instancia especificado.....	60
Descripción.....	60
Entorno.....	60
Resultados	60
Conclusiones	62
Entorno de Cómputo Spot con tipo de máquina óptima.	63
Descripción.....	63
Entorno.....	63
Resultados	64
Conclusiones	65
Conclusiones Generales.....	66
Trabajo futuro	68

Introducción

En informática se conoce como procesamiento por lotes o procesamiento en batch, a todas aquellas tareas en las cuales corren uno o varios scripts ejecutando un programa sin la necesidad de la intervención de un humano más allá del ingreso de algún valor de entrada o salida pre-determinado [1]. Estos procesamientos aún son críticos para la mayoría de las organizaciones debido a la gran cantidad de procesos de negocio que se realizan utilizando este tipo de procesamiento mayormente debido a la gran cantidad de datos que pueden ser procesados para generar valor sin la necesidad de que una persona se encuentre presente para apoyar el proceso [2]. Razones como la anterior son los motivadores para llevar a la implementación de estilos arquitecturales que favorezcan el desempeño y confiabilidad de este tipo de procesos, como es el caso de la arquitectura lambda [3] la cual ofrece capacidad de procesamiento para cantidades masivas de datos (Big Data) apuntando a obtener un correcto balance entre latencia, rendimiento (throughput) y tolerancia a fallos.

Sin embargo, siempre existirán retos a ser superados y circunstancias a tomar en cuenta dentro de este tipo de procesamientos. Por ejemplo, según Oracle un proceso en batch es aquél que posee una carga de trabajo sin restricciones y que debe ser resuelta en la menor cantidad de tiempo posible. Lo anterior implica que este proceso siempre buscará consumir todos los recursos necesarios, y eventualmente todos los disponibles, para resolver dicha carga de trabajo y por tanto puede llegar a afectar otros procesos en batch e inclusive a una porción del entorno de producción en horas laborales, particularmente en aquellos días donde el sistema presente un pico de estrés [4]. Teniendo en cuenta esto se puede decir que cualquier implementación de un procesamiento en batch dentro de cualquier sistema debe de hacer frente a los siguientes desafíos: (i) Asignación de recursos entre los componentes destinados a procesamiento en batch, o siendo más específicos. (ii) Separación de recursos dedicados o administración de recursos compartidos entre los distintos pipelines existentes para procesamiento en batch. (iii) Escalabilidad de los recursos cuando se acerca al límite disponible para evitar tiempos fuera de servicio o latencia adicional agregada a los entornos de producción en horarios laborales. (iv) Orquestación de pipelines cuando estos deban manejar distintas prioridades, o en caso de que estos trabajen de forma paralela, sincronizarse para generar una salida global de los datos procesados.

A partir de las descripciones anteriormente dadas podemos tener un punto de partida sobre lo que se pretende solucionar o por lo menos mitigar siempre que estemos hablando acerca de procesamiento en batch. Ahora, es necesario dar contexto para definir la situación sobre la cual se requiere utilizar esta tecnología y por qué es de importancia para resolver el problema a tratar. Para lograr esto definiremos a continuación el que es nuestro caso de estudio para este proyecto.

El Instituto de Hidrología, meteorología y adecuación de tierras (IDEAM) se define a sí mismo como una institución pública de apoyo técnico y científico al Sistema Nacional Ambiental, que genera conocimiento, produce información confiable, consistente y oportuna, sobre el estado y las dinámicas de los recursos naturales y del medio ambiente, que facilite la definición y ajustes de las políticas ambientales y la toma de decisiones por parte de los sectores público, privado y la ciudadanía en general [7]. Esta organización empezó en el 2014

un proyecto basado en la iniciativa de código libre del Cubo de Datos [5] particularmente utilizando la implementación *Australian Geoscience Data Cube* (AGDC) en su segunda versión, con el fin de ofrecer análisis de riesgos y modelos predictivos del área continental del territorio colombiano, dichos análisis son procesos realizados en Batch [6].

Han sido muchas las mejoras realizadas desde el 2014, y muchas de las principales debilidades de la infraestructura de computación de alto rendimiento (HCP) del IDEAM tales como la carencia de un Sistema distribuido de almacenamiento o la capacidad para escalar horizontalmente del Sistema [6] han sido mitigadas gracias al desacoplamiento de algunos de los componentes originales en distintas máquinas virtuales [5]. El uso de esta arquitectura distribuida ha permitido alcanzar importantes mejoras en cuanto al desempeño y la concurrencia del sistema.

Sin embargo, debido al intenso y extenso consumo de recursos que implican las tareas de procesamiento de imágenes satelitales realizadas por el Cubo de datos colombiano (CDCOL), actualmente hay una falta de recursos computacionales disponibles para satisfacer la demanda de análisis de estas cuando aparecen picos de uso en el sistema. Adicionalmente, a este ritmo se volverá insostenible para el IDEAM el hecho de escalar verticalmente dadas las limitaciones de costos y tiempo que conlleva administrar su propio centro de datos.

Por los motivos anteriormente descritos es un imperante para el IDEAM trasladar la arquitectura que poseen actualmente a una soportada por la nube. Se busca un proveedor de nube pública que sea confiable, represente una adición de valor real para la organización en temas de ahorro de costos por cuestión de administración de los servicios tecnológicos y de operación y que por encima de todo sea confiable y permita mitigar los problemas que actualmente presenta la arquitectura del IDEAM en una solución sostenible de aquí a varios años en el futuro.

En particular nosotros nos encargaremos de validar y migrar el componente encargado del procesamiento de las imágenes satelitales. El objetivo de este documento será el de mostrar el análisis y proceso realizado para alcanzar la solución propuesta a este problema y las ventajas y desventajas que esto conlleva para nuestro caso de estudio y la organización a la cual pertenece.

Estado del Arte

Como anteriormente mencionamos, a gran escala una de las metas del IDEAM es lograr una migración efectiva a la nube o en el peor de los escenarios, lograr una solución híbrida que les permita mitigar varios de los problemas clásicos que conlleva administrar su propio centro de datos y que en estos momentos están representando limitantes para el correcto desarrollo de las operaciones de su negocio. Sin embargo, a día de hoy son numerosos los distintos proveedores de nube pública con los que podemos contar, suenan nombres de importantes actores del sector tecnológico como IBM u Oracle y otros nuevos como pueden ser DigitalOcean o RackSpace. Sin embargo, ¿Cuál de todos estos proveedores es aquél que mejor se ajusta a las necesidades actuales del IDEAM? El propósito de esta sección del documento es llegar a una conclusión para la pregunta anteriormente planteada.

Antes de continuar es necesario mencionar que por restricciones de negocio el IDEAM ya se encuentra trabajando con el proveedor de nube pública Amazon Web Services (AWS) y por tanto el desarrollo de este proyecto fue basado en los servicios ofrecidos por esta plataforma, sin embargo lo anterior no debe ser un desmotivador sino todo lo contrario, debemos plantearnos alternativas de solución con el fin de evaluar si se tomó una decisión adecuada según las necesidades del negocio y de las posibles alternativas existentes en el mercado para que en el caso de requerir una migración a algún otro proveedor (ya sea por incumplimiento del actual u alguna situación extraordinaria como una alianza estratégica con algún otro) la decisión de a quién elegir pueda ser tomada de forma ágil.

En primer lugar, para poder empezar a escoger dentro de las distintas opciones que tenemos para elegir, debemos dejar claras las necesidades que tenemos. Como mencionamos durante nuestra introducción, estaremos trabajando en migrar el proceso de procesamiento de imágenes satelitales del IDEAM. Dicho proceso actualmente consiste en una implementación típica del patrón de arquitectura Pipe & Filter [8] en el cual contamos con los siguientes componentes: una cola de mensajes en RabbitMQ, unos Celery Workers encargados de sacar los mensajes y pasarlos para procesamiento en una aplicación en Python que se encarga de realizar el procesamiento del mensaje, y un NFS donde se almacena el resultado, los primeros corriendo sobre máquinas de EC2 y el NFS en un EFS de AWS. Teniendo esto en cuenta a continuación se presenta un diagrama de la arquitectura actual de este proceso diseñada e implementada por Mario Villamizar en la figura 2.1 [5].

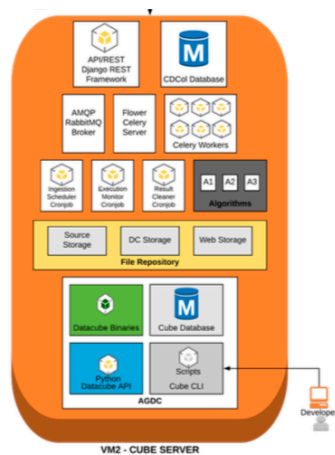


Figura 2.1. Arquitectura actual cubo de datos.

Adicionalmente y como ya se ha mencionado, dicho procesamiento se ejecuta en batch y esta será la palabra clave, estamos buscando un proveedor que nos ofrezca un servicio capaz de manejar procesamientos en batch con los siguientes componentes: colas de mensajes, workers y un NFS, y que además nos permita mitigar tantos de los problemas asociados a este tipo de procesamiento como nos sea posible.

Con nuestras necesidades claras, ahora sí podemos proceder a escoger un proveedor. Sin embargo, como mencionamos al inicio de esta sección, existen una cantidad numerosa de ellos, por tanto y con el fin de acotar las opciones en nuestra búsqueda se utilizó la opinión de un experto en el tema. En nuestro caso utilizamos el cuadrante mágico de la consultora tecnológica Gartner para proveedores de Infraestructura Cloud. Se utilizó el Cuadrante actualizado para el 2018 y uno del 2014 con el fin de evaluar el rendimiento y evolución de los distintos proveedores a través del tiempo con el fin de poder tomar una decisión. Dichos cuadrantes pueden observarse en las figuras 2.2 y 2.3.

Figure 1. Magic Quadrant for Cloud Infrastructure as a Service, Worldwide



Figura 2.2. Cuadrante Mágico Año 2018

Figure 1. Magic Quadrant for Cloud Infrastructure as a Service



Figura 2.3. Cuadrante Mágico Año 2014

Teniendo en cuenta la información suministrada por Gartner los proveedores elegidos para nuestra comparación son: Amazon Web Services (AWS), Microsoft Azure (Azure) y Google Cloud Platform (GCP). Como bien se mencionó ya definimos de forma bastante clara nuestras necesidades así que se procede a buscar los servicios para procesamientos en batch ofrecidos por cada uno de ellos. Sacaremos ventaja del hecho de que conocemos de forma previa un servicio que cumple con nuestras necesidades en el proveedor AWS, este servicio se llama Batch, llamado en mayúscula a partir de ahora para diferenciarlo de los procesamientos en batch.

Para el caso de GCP nos apoyamos en la herramienta de comparación de servicios ofrecida por este proveedor [9]. En dicha herramienta podemos observar que desafortunadamente el proveedor no ofrece un servicio análogo dentro de su plataforma como se observa en la figura 2.4.1.

Service Category	Service	AWS	Google Cloud Platform
Compute	IaaS	Amazon Elastic Compute Cloud	Compute Engine
	PaaS	AWS Elastic Beanstalk	App Engine
	Containers	Amazon Elastic Container Service	Google Kubernetes Engine
	Serverless Functions	AWS Lambda	Cloud Functions
	Managed Batch Computing	AWS Batch	N/A

Figura 2.4.1 Comparación servicios GCP/AWS

Lo anterior implica que descartemos de manera inmediata este proveedor ya que con él no podremos alcanzar los objetivos de nuestro proyecto. Sin embargo se hace mención de que la arquitectura actual puede ser alcanzada utilizando este proveedor, ya que ofrece servicios similares a EC2 y al EFS utilizado dentro de la arquitectura actual como se puede apreciar en la figura 2.4.2.

File Storage	Amazon Elastic File System	Cloud Filestore (beta)
--------------	----------------------------	--

Figura 2.4.2 EFS dentro de GCP

De forma similar realizamos el mismo procedimiento utilizando la herramienta de comparación de Azure [10]. En este caso observamos que Azure sí ofrece un servicio que funciona de forma similar a AWS Batch, por lo cual vale la pena realizar una comparación entre el servicio de AWS Batch y Azure Batch. Adicionalmente se menciona que Azure ofrece servicios similares a todos los utilizados para la arquitectura actual en AWS. Por lo cual al igual que con GCP se podría realizar un cambio de proveedor en caso de ser necesario. La información anteriormente descrita se puede observar en las imágenes 2.5.1 y 2.5.2

Batch computing	AWS Batch	Azure Batch	Run large-scale parallel and high-performance computing applications efficiently in the cloud.
Shared file storage	Elastic File System	Azure Files (file share between VMs)	Provides a simple interface to create and configure file systems quickly, and share common files. It's shared file storage without the need for a supporting virtual machine, and can be used with traditional protocols that access files over a network.

Con la información reunida hasta ahora procedemos a hacer una comparación entre ambos servicios. En principio ambos servicios de ofrecen como servicios autogestionados para el manejo de procesamiento en lotes. Para el caso de AWS Batch contamos con los conceptos de entornos de cómputo, colas de trabajos, definiciones de trabajos y trabajos [11], estos nos permiten definir entornos de ejecución con máquinas virtuales de nuestra elección de acuerdo a nuestras necesidades y definir trabajos con requerimientos de recursos específicos (Memoria, CPU) con prioridades asociadas a estos para ejecutar según lo requiramos y adicionalmente ofrece la posibilidad. Para el caso de Azure este ofrece funcionalidades similares en todos los aspectos, incluso nos ofrece la opción de utilizar ‘pools’ de máquinas virtuales autogestionadas por ellos en las cuales a partir de unos parámetros de entrada se define qué tipo de instancia es la más adecuada para una determinada tarea, parecido a lo que AWS ofrece con los entornos de cómputo optimizados, y también podemos hacer uso de máquinas virtuales de baja prioridad que son análogas a las instancias Spot de AWS para optimizar los costos de nuestras ejecuciones [12]. Sin embargo la principal diferencia entre Azure y AWS es que el manejo de Azure Batch debe realizarse a través de un script manejando la API de Azure, la cual actualmente solo está disponible para C#, JavaScript, Java y Python [13]; por cual podemos observar que estamos ante una limitación en cuanto al lenguaje que debe usarse para poder empezar a utilizar este servicio.

Teniendo en cuenta lo anteriormente descrito se realiza un breve cuadro comparativo entre ambas tecnologías con el fin de ilustrar sus principales características en la tabla 2.6.

	AWS Batch	Azure Batch
Autoescalable	Sí	Sí
Selección de VM Optima	Sí	Sí
Definición de tareas	Sí	Sí
Manejo de prioridades	Sí	Sí
Uso instancias Spot	Sí	Sí
Uso de instancias personalizadas	Sí	Sí
Uso Obligatorio de API	No	Sí
Uso de contenedores	Sí	N/A

Tabla 2.6. Comparación AWS Batch y Azure Batch.

A partir de esto podemos observar como claramente si bien ambos servicios comparten muchas de sus características y son similares entre sí, aquella que marca la diferencia es el uso de contenedores como un requisito para utilizar la plataforma o no. Lo anterior presenta los las siguientes ventajas y desventajas.

Desventajas:

- Tiempos adicionales de desarrollo para poder migrar una aplicación al servicio, pues necesario “contenerizar” la aplicación.
- Se requiere la intervención de un manejador de contenedores, como Docker u otra tecnología que habilite este procedimiento.

Ventajas:

- Automatización de la instalación de recursos y pasos necesarios para la ejecución y despliegue de la aplicación
- Independencia del entorno de ejecución, dependencias, librerías y lenguajes de la máquina host.
- Mayor mantenibilidad y facilidad de despliegue de las aplicaciones a gran escala para sistemas distribuidos.

Por lo tanto, teniendo en cuenta los pros y los contras acerca de la decisión de utilizar contenedores o no es bastante claro para el autor que el uso de estos son el camino a seguir. Sobre todo cuando uno de los casos de uso principales para los procesamiento en batch son justamente los análisis de big data y en la actualidad uno de los principales lenguajes utilizados para este tipo de tareas son Scala o F# según StackOverflow [14], lenguajes que a día de hoy no son soportados por la API de Azure Batch y que. Adicionalmente tecnologías como docker a día de hoy son ampliamente aceptadas siendo utilizada en un 25% del total

de organizaciones de tecnología a nivel mundial y presentando una tasa de crecimiento del 75% anual [15].

Por los motivos anteriormente descritos se concluye que AWS con Batch son las mejores opciones para solucionar nuestro problema y, contrario a lo que se pensaría a primera vista, el hecho de que este servicio obligue al usuario a manejar contenedores causa como resultado que no exista una dependencia al proveedor si no que con la existencia de una imagen para el contenedor se pueda migrar a cualquier otro proveedor de forma sencilla y ágil, eso sí, con la carga de trabajo que implica la creación y administración de todo un pipeline para procesamientos en batch. Dicha conclusión coincide con la decisión tomada, por lo cual se puede afirmar que se tomó una elección apropiada que beneficia a la organización y los objetivos que pretende lograr.

Objetivos

El principal objetivo del proyecto es lograr una arquitectura en la nube escalable, portable y autogestionada capaz de manejar varias tareas y procesos de distintos analistas del IDEAM. Con el fin de lograr tal objetivo, las siguientes metas deben ser logradas.

Objetivos Generales

- Proponer y validar decisiones arquitecturales en la nube con la capacidad de ser replicadas en la solución On-Premise para la infraestructura HPC del IDEAM. Y en caso de no ser posible, democratizar solo aquellos servicios que valga la pena en la plataforma de AWS.
- Otorgar a través del uso de AWS una arquitectura escalable, modular y confiable para cumplir los propósitos de los analistas del IDEAM.

Objetivos Específicos

- ‘Dockerizar’ la aplicación de los cube workers para automatizar y facilitar la instalación de esta a la vez que se otorga portabilidad a la solución.
- Configurar la aplicación de los cube workers para poder trabajar dentro de AWS Batch con el fin de mejorar el manejo de los recursos, la escalabilidad y la concurrencia de las tareas manejadas por esta.

Metodología

Alcance del proyecto

Con el fin de alcanzar los objetivos planteados, las siguientes tareas debían y fueron cumplidas.

1. Investigar acerca de los principales conceptos y flujo de trabajo del servicio AWS Batch.
2. Capacitarse en las tecnologías necesarias para poder usar AWS Batch, en particular Docker, IAM, ECS Lambda y VPC.
3. Aplicar los conceptos aprendidos dentro de una demo funcional capaz de integrarse a Batch con dichas tecnologías para evaluar las posibilidades de AWS Batch.
4. Generar una cuenta IAM dentro de la cuenta de AWS usada para el despliegue del CDCOL.
5. Dockerizar la actual aplicación de los cube workers para poder utilizarla dentro del contexto de AWS Batch.
6. Desplegar la aplicación dockerizada dentro de AWS Batch y hacer experimentaciones dentro de esta con el fin de evaluar las ventajas de utilizar este servicio en comparación con una solución On-Premise.
7. Diseñar y proponer buenas prácticas de uso de AWS Batch a partir de los resultados obtenidos en la experimentación.

Prueba de Concepto

Siguiente la metodología de nuestro proyecto procedimos a realizar la prueba de concepto a partir de los conocimientos adquiridos acerca de este servicio y los pasos necesarios para poder ejecutar un proyecto en este.

Esta prueba consistía en una demo sencilla de una aplicación de marcas de agua hecha en Node.js en la cual imagenes eran subidas al servicio de S3 a través de su interfaz gráfica, estas a su vez disparaban una función Lambda cada vez que una imagen era subida y utilizaban la API de AWS Batch para enviar la meta-data de la imagen subida como un trabajo de AWS Batch. Finalmente, esta metadata era capturada por Batch y la asignaba a variables de entorno que eran utilizadas por el contenedor que ejecutaba la aplicación de marca de agua y subía los resultados a otro bucket de S3.

El código correspondiente a esta aplicación puede ser encontrado en el siguiente repositorio: <https://github.com/aczuleta/AWS-Batch-Experiment>

Una vista de la arquitectura diseñada para esta demo se puede observar a continuación en la figura 4.2.1.

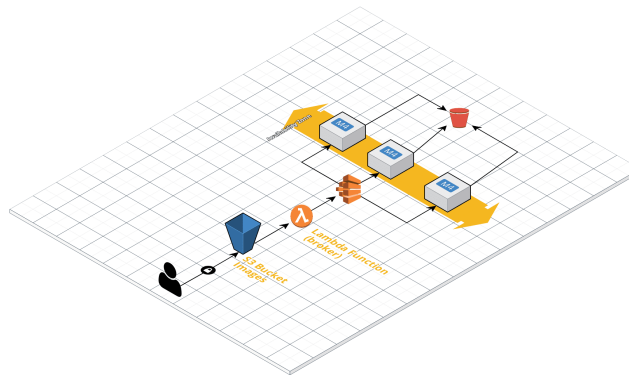


Figura 4.2.1. Prueba de concepto

El propósito principal detrás de esta demo era la de simular un tipo de tarea similar a las procesadas por el CDCOL, en este caso, procesamiento de imágenes / marcas de agua y observar el comportamiento computacional presentado por Amazon. En este caso el autor encontró algunas cosas interesantes relacionadas a la configuración hecha por los entornos computacionales óptimos o manejados por Batch. Para esta demo se le delegó la responsabilidad a AWS de determinar el tipo de instancia más apropiada para una tarea en específico, y se encontró que dependiendo de la cantidad de imágenes concurrentes se podía tener desde una instancia de propósito general T2 hasta una optimizada para cómputo como lo son las instancias C3 o incluso instancias optimizadas para memoria R3, siendo estas el mejor tipo de instancias para esta clase de trabajos de acuerdo a la documentación de AWS [16].

Este resultado fue bastante conveniente, especialmente porque sabemos que actualmente la memoria es un cuello de botella para el procesamiento de los ‘tiles’ en el IDEAM. Lo anterior sin mencionar que en esta demo no explotamos la capacidad de las instancias spot, las cuales nos ayudarán a reducir los costos de procesamiento hasta en un 90% si somos lo suficientemente afortunados [17].

Por tanto, teniendo en cuenta estos resultados logramos validar que Batch es una solución apropiada, y con los conocimientos ya adquiridos para hacer una correcta configuración del servicio podemos proceder a hacer un experiment complete sobre Batch, esta vez utilizando los cube workers del CDCOL en lugar de una simple aplicación demo.

Migrando el cubo de datos a AWS Batch

En primer lugar, para realizar la migración de cualquier aplicación para que pueda ser utilizada por el servicio de AWS Batch, dicha aplicación debe poder ser ejecutada por medio de un contenedor Docker. Las razones de lo anterior es que este servicio está basado a partir de otro servicio, en este caso el de Elastic Container Service, para su correcto funcionamiento.

Teniendo en cuenta la anterior premisa, en primer paso fue ‘dockerizar’ la aplicación, lo cual se resume en convertir el script de instalación ya existente a un Dockerfile capaz de instalar según lo especificado el cubo de datos. Por supuesto, no fue un simple copiado y pegado del script original pues los comandos de docker y de bash no son completamente análogos, particularmente existen muchas diferencias sobre todo a nivel de permisos sobre la máquina host, ya que mientras bash ejecuta directamente sobre esta, los contenedores Docker tienen un nivel diferente de aislación y por tanto privilegios más limitados. Ejemplo de lo anterior puede ser el comando EOF para la escritura en múltiples líneas a través de la entrada estándar, dicho comando no existe ni puede ser utilizado en un Dockerfile.

Dicho Dockerfile genera una imagen de Docker, la cual es básicamente un empaquetado de todas las dependencias que son necesarias para correr de forma correcta la aplicación ‘dockerizada’. En nuestro caso el Dockerfile utilizado puede ser encontrado en el siguiente repositorio:

<https://github.com/aczuleta/AWS-Batch Experiment/blob/version2/cubo/Dockerfile>

Adicionalmente se recomienda que al momento de construirse el Dockerfile, se le especifique un ‘Entrypoint’, la función de dicho comando es la de convertir nuestro contenedor en un archivo ejecutable. La premisa anterior implica que, en primera instancia, nuestro contenedor ejecutará la acción o serie de acciones que se especifiquen en el entrypoint, y en segundo lugar, que al momento de finalizar dicha sucesión de acciones el contenedor se autodestruirá

de forma automática. En nuestro caso particular le especificamos a nuestro Dockerfile que ejecutara un script de bash que fue previamente instalado dentro del contenedor por medio del Dockerfile y dicho script se encarga de realizar de forma secuencial los siguientes cuatro pasos:

- Montar los volúmenes NFS para /web_storage y /dc_storage.
- Colocar como globales las variables de entorno que se le pasarán al contenedor para la ejecución de cada algoritmo.
- Colocarse sobre el directorio sobre el cual se ejecutará el algoritmo.
- Ejecutar el algoritmo como el usuario ingestor, quien es el el usuario con los permisos necesarios para ejecutar el algoritmo.

Los motivos detrás de las operaciones anteriores son: (i) Los volúmenes no pueden ser montados durante la construcción de la imagen, deben ser montados cuando ya existe un contenedor corriendo. (ii) Por la misma razón, no se pueden quemar en el código las variables a utilizar en el Dockerfile, deben ser pasadas en tiempo de ejecución como variables de entorno al momento de inicializar el contenedor. (iii) Debido a que el usuario por defecto de Docker es el usuario 'root', dentro del contenedor, no dentro de la máquina host, es necesario ejecutar el algoritmo con otro usuario. Dicho script de inicialización puede ser encontrado en el siguiente enlace:

<https://github.com/aczuleta/AWS-Batch-Experiment/blob/version2/cubo/start.sh>

Ahora, una vez generada nuestra imagen de docker de nuestra aplicación, ya podemos comenzar a utilizar el servicio de Batch.

El primer aspecto a configurar si se desea utilizar este servicio son los entornos informáticos, los anteriores son básicamente los clusters o grupos de autoescalado de instancias EC2 que serán utilizados para servir como host de nuestros contenedores. En dicha pestaña de configuración podremos elegir si queremos que nuestro entorno sea administrado por AWS o por nosotros, en caso de que elijamos administrado, las políticas de autoescalamiento serán totalmente gestionadas por AWS a partir de la demanda, mientras que si elejimos no administrado estas deberán ser suministradas por nosotros. En el caso de este experimento se eligió Administrado con el fin de facilitar la ejecución del mismo. También dentro de esta pestaña tendremos que asignar roles, tanto para que el servicio de batch como para las instancias EC2 que serán lanzadas por este servicio. En nuestro caso debido a que no vamos a utilizar llamados a servicios adicionales, bastó con elegir la opción de 'crear nuevo rol'. Dicha configuración puede ser apreciada en la figura 4.3.1

Tipo de entorno informático

☒ **Administrado**
AWS amplía y configura las instancias automáticamente.

☐ **No administrados**
Usted controla y administra la configuración de las instancias, del aprovisionamiento y del escalado.

Nombre del entorno informático

Rol de servicio

También puede especificar un rol de IAM que otorgue permisos al contenedor de su trabajo para usar las API de AWS. Esta característica utiliza roles de IAM de Amazon ECS para la funcionalidad de las tareas.

Rol de instancia

Sus recursos informáticos utilizan el perfil de instancia ecsInstanceRole de IAM para hacer llamadas a las API de AWS en su nombre. Si aún no tiene ecsInstanceRole, podemos crearlo por usted.

Par de claves EC2

Figura 4.3.1

Posteriormente deberemos especificar si queremos utilizar instancias bajo demanda, el modelo clásico, o utilizar instancias ‘spot’. También debemos especificar el tipo de instancia que queremos utilizar o si queremos que Amazon determine el tipo de instancia óptima para nuestra carga de trabajo a partir de las definiciones de trabajo de las cuales hablaremos próximamente. Adicionalmente es posible especificar la cantidad de CPUs virtuales mínima, deseadas y máximas. En caso de que la cantidad mínima sea cero, todas las instancias EC2 serán apagadas cuando no existan trabajos por ser procesados. Nuestra configuración puede ser apreciada en la figura 4.3.2

Modelo de aprovisionamiento

☒ **Bajo demanda**
Las instancias EC2 que paga por hora.

☐ **Puntuales**
Ahorre dinero pujando en las instancias de subasta pero tenga en cuenta que si le sobrepujan sus instancias finalizarán.

Tipos de instancias permitidos

Mínimo de CPU virtuales

CPU virtuales deseadas

Máximo de CPU virtuales

Los límites de tipo de instancia EC2 determinan el máximo de CPU virtuales que puede tener. [Más información](#)

Tipos de instancias permitidos

c3 family
c3.2xlarge
c3.4xlarge
c3.8xlarge
c3.large
c3.xlarge
c4 family

...a, y esta información se utiliza para correlacionar el trabajo

...se agotará cuando no haya trabajo. Si lo establece por ... en todo momento.

Figura 4.3.2

Finalmente debemos configurar la VPC y subred sobre la cual serán lanzadas nuestras máquinas. En este punto vale la pena recalcar que la subred sobre la cual se vayan a ejecutar dichas máquinas debe tener la opción de asignar automáticamente una Ipv4. Se menciona lo anterior porque en el caso de la subred pública del ideam esta opción no se encontraba seleccionada. Nuestra configuración puede ser observada en las figuras 4.3.3 y 4.3.4.

Redes

ID de VPC

IDEAM | vpc-56f9e42f

Subredes

☐

 subnet-5400a630 | 192.168.1.0/24 | us-east-1a | Private subnet

☐

 subnet-8100a6e5 | 192.168.0.0/24 | us-east-1a | Public subnet

Grupos de seguridad

default | sg-b33d85c0

Figura 4.3.3

<input checked="" type="checkbox"/>	Public subnet	subnet-8100a6e5	available	vpc-56f9e42f IDEAM	192.168.0.0/24	246	-
<input type="checkbox"/>		subnet-89cefdb3	available	vpc-3f27115a	172.31.32.0/20	4091	-
<input type="checkbox"/>		subnet-8d0880f5	available	vpc-3f27115a	172.31.0.0/20	4091	-

Subnet: subnet-8100a6e5

Description

Flow Logs

Route Table

Network ACL

Tags

Subnet ID

 subnet-8100a6e5

VPC

 vpc-56f9e42f | IDEAM

Available IPv4 Addresses

 246

Availability Zone

 us-east-1a

Network ACL

 acl-8d0880f5

Auto-assign public IPv4 address

 Yes

State

 available

IPv4 CIDR

 192.168.0.0/24

IPv6 CIDR

 -

Route Table

 rtb-e2a7fd99

Default subnet

 No

Auto-assign IPv6 address

 No

Figura 4.3.4

Posteriormente debemos crear una cola en la cual serán almacenados los trabajos que serán procesados. Esta sección es mucho más sencilla pues lo único resaltable sobre las colas es la prioridad que estas tendrán, señaladas como valores enteros positivos donde entre mayor sea el valor mayor será la prioridad, y el entorno informático al cual se le pasarán los mensajes que vayan siendo encolados. Nuestra configuración puede ser apreciada en la figura 4.3.5.

Crear una cola de trabajo

Una cola de trabajo acepta envíos de trabajo y los sitúa en un entorno informático adecuado.

Nombre de la cola

Prioridad

Las colas de trabajo con un valor entero mayor de prioridad tienen preferencia en los recursos informáticos.

Habilitar la cola de trabajo ☒

La cola de trabajo no puede aceptar envíos de trabajo si está deshabilitada.

Entornos informáticos conectados para esta cola

Los trabajos se envían a los entornos informáticos conectados en función del orden en el que se enumeran y la capacidad disponible en esos entornos.

Servicio	Entorno informático
1	Nombre: entorno_cubo2 Modelo de aprovisionamiento: Bajo demanda Tipos de instancia: m5d.4xlarge Mín. de CPU virtuales: 0 Máx. de CPU virtuales: 16

Seleccionar un entorno informático

Figura 4.3.5

Para finalizar se debe crear una definición de trabajo, sin embargo, antes de poder crear una es necesario tener en algún repositorio la imagen de Docker que va a ser utilizada para dicha definición. En nuestro caso utilizamos el repositorio para imágenes de Docker de Elastic Container Registry. El procedimiento para subir una imagen es bastante intuitivo y puede ser logrado siguiendo el 'Wizard' proporcionado por dicho servicio. Lo importante es conocer la URI proporcionada por ECR para poder utilizarla posteriormente al momento de crear la definición del trabajo. Imágenes explicando nuestra configuración pueden ser encontradas en las figuras 4.3.6 y 4.3.7.

Amazon ECS
Clusters
Task Definitions
Amazon ECR
Repositories

< All repositories : datacube_image

Repository ARN: `arn:aws:ecr:us-east-1:186121879828:repository/datacube_image`

Repository URI: `186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image`

[View Push Commands](#)

Images | Permissions | Dry run of lifecycle rules | Lifecycle policy

Amazon ECR limits the number of images to 1,000 per repository. [Request a limit increase](#).

Image sizes may appear compressed. [Learn more](#)

Delete Last updated on October 17, 2018 3:38:14 PM (2m ago)

Filter in this page < 1-1 > Page size 100

Image tags	Digest	Size (MiB)	Pushed at
<input type="checkbox"/> latest	view all sha256:ad10c9d719b6ce42b23fdbca4173cd1...	2395.36	2018-10-16 10:41:38 -0500

Figura 4.3.6

View Push Commands ✕

Ensure you have installed the latest version of the AWS CLI and Docker. For more information, see the [ECR documentation](#).

1) Retrieve the login command to use to authenticate your Docker client to your registry.

For macOS or Linux systems, use the AWS CLI:

```
$(aws ecr get-login --no-include-email --region us-east-1)
```

For Windows systems, use AWS Tools for PowerShell:

```
Invoke-Expression -Command (Get-ECRLoginCommand -Region us-east-1).Command
```

Note: If you receive an "Unknown options: --no-include-email" error when using the AWS CLI, ensure that you have the latest version installed. [Learn more](#)

2) If you are using the AWS CLI, run the login command from the output of step 1.

3) Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions [here](#). You can skip this step if your image is already built:

```
docker build -t datacube_image .
```

Figura 4.3.7

Una vez tengamos nuestra imagen en un repositorio podremos crear la definición de trabajo. Esta es probablemente la sección más importante al momento de generar un ambiente de ejecución en batch, pues en este apartado vamos a definir qué es lo que se va a ejecutar, qué cantidad de recursos necesita y qué parámetros de lanzamiento van a ser pasados como parámetro al momento de inicializar nuestros contenedores.

En nuestro caso la configuración seleccionada fue utilizando la imagen de docker previamente subida y sin ningún rol adicional pues no haremos llamados a servicios de aws adicionales, como se observa en la figura 4.3.8.

Las definiciones de trabajo le permiten guardar los valores de un trabajo para poder usarlos más tarde como plantillas.

Nombre de la definición de trabajo

Rol de trabajo

También puede especificar un rol de IAM que otorgue permisos al contenedor de su trabajo para usar las API de AWS. Esta característica utiliza roles de IAM de Amazon ECS para la funcionalidad de las tareas. [Más información.](#)

Imagen de contenedor

Figura 4.3.8

Con respecto a los requerimientos de recursos especificamos que solo necesitaremos de un solo CPU virtual y utilizaremos 64GiB de memoria, ya que es lo aproximado que toma procesar una tarea, adicionalmente configuramos que se ejecuten dos intentos para realizar la tarea y un timeout de 8000 segundos. Podemos observar esto en la figura 4.3.9

CPU virtuales

Memoria (MiB)

Intentos de trabajo

Execution timeout

Time (in seconds) to allow each job attempt to run. If your job runs longer than the specified time, it will stop and be moved to FAILED.

Figura 4.3.9

Sin embargo, el aspecto más importante quizás, es el de no olvidar chequear como privilegiada dicha tarea, esto le dará permisos de sudo al contenedor instanciado sobre la máquina host, lo cual el permitirá hacer el montaje de los volúmenes de forma correcta. Esto se observa en la figura 4.3.10

Seguridad

Privilegiado ☒

Usuario

Figura 4.3.10

Con todo esto ya tenemos nuestro ambiente de ejecución en AWS Batch totalmente funcional. Solo falta enviar trabajos ya sea de forma manual, a través de la interfaz de Batch o de forma programática a través de la API de este servicio. Lo único importante a tener en cuenta para los propósitos de nuestro caso de uso es enviar los parámetros de ejecución del algoritmo como variables de entorno como se puede apreciar en la figura 4.3.11.

Clave	Valor
product	'LS7_ETM_LEDAPS'
time_ranges	[['01-02-2017', '09-05-2017']]
output_expression	''
kwargs	normalized=True, minValid=1
min_long	-75
min_lat	9
version	'1.0'
execID	'1301'
algorithm	'ndvi'

Figura 4.3.11

Montaje Experimental

Diseño del ambiente de pruebas

Ahora, con el fin de realizar pruebas automatizadas sobre el ambiente de ejecución anteriormente explicado, se propone el siguiente entorno de pruebas. Se subirá un archivo en formato Json a un Bucket de S3 que contendrá la información de todos los parámetros necesarios para ejecutar el algoritmo deseado, dicho archivo al momento de ser subido disparará una función lambda encargada de leer y formatear su contenido para llamar a la API de Batch y generar un nuevo trabajo para ser procesado, dicho trabajo será manejado por batch el cual deberá generar un archivo .nc que será almacenado en el NFS previamente configurado. El diagrama de arquitectura y el formato deseado de los archivos pueden ser observados en las figuras 5.1.1 y 5.1.2.

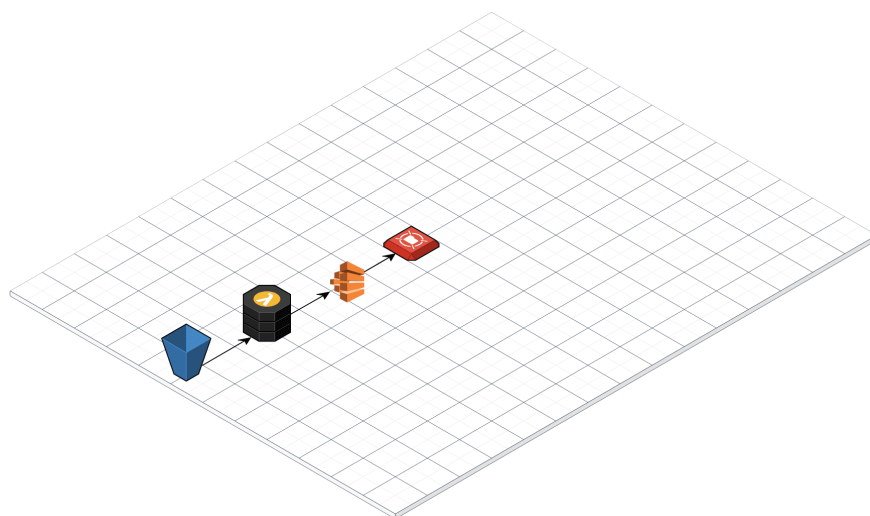


Figura 5.1.1

```
{
  "execID" : 1159,
  "algorithm" : "ndvi",
  "version" : 1.0,
  "output_expression" : "",
  "product" : "LS7_ETM_LEDAPS",
  "min_lat" : 9,
  "min_long" : -75,
  "time_ranges" : "[('01-02-2017', '09-05-2017')]",
  "kargs" : {
    "normalized" : "True",
    "minValid" : 1
  }
}
```

Figura 5.1.2

Experimentos y pruebas propuestas

Como se ha podido observar en este documento, AWS Batch ofrece una gran cantidad de opciones para distintas configuraciones. Por tanto, se proponen explorar algunas de las principales funcionalidades de este para ver cómo es el comportamiento de estas y si su correcta configuración puede beneficiar al IDEAM ya sea a nivel de escalabilidad o a nivel de costos.

Las principales cuestiones que queremos resolver con los experimentos son:

- Verificar el correcto funcionamiento de la elasticidad de Batch para una gran cantidad de imágenes por procesar.
- Verificar si las instancias lanzadas de forma “óptima” por AWS presentan tiempos de ejecución mejores que un entorno informático previamente dimensionado por el equipo de trabajo.
- Verificar el ahorro de dinero por el uso de instancias ‘spot’ con relación al modelo bajo demanda.
- Utilización de distintas colas conectadas a distintos entornos informáticos con distintas prioridades y monitorear su comportamiento, ya que distintos análisis pueden tener distintas prioridades

Diseño de experimentos sobre AWS Batch

A continuación, se presentarán el diseño de los experimentos realizados sobre el servicio de AWS Batch con el fin de validar algunas características ofrecidas por este servicio, afianzar el uso de dichas características y concluir de qué forma sería recomendable usarlas con el fin de maximizar el proyecto de estas por parte del IDEAM.

Es necesario recalcar que tal y como se mencionó anteriormente, los aspectos de mayor importancia al momento de configurar un entorno de trabajo en AWS Batch son:

- Entornos de Computo
 - Roles de las instancias a lanzar
 - Entorno manejado por AWS / administrado por el usuario
 - Instancias On-Demand/Spot
 - Tipo de instancias que serán lanzadas
 - CPUs Virtuales Mínimos, Deseados y Máximos.
- Colas
 - Entornos de computo asociado
 - Prioridad
- Definiciones de trabajo
 - Imagen de contenedor a utilizar
 - Cantidad de memoria a utilizar.
 - Cantidad de CPUs virtuales a utilizar.
 - Contenedor privilegiado / no privilegiado.
 - Variables de entorno.

Para el caso de nuestros experimentos, mantendremos como constantes algunos parámetros. Dichos parámetros serán específicamente: Los roles (generados por defecto por AWS Batch) y entorno manejado por AWS para los entornos de computo. Y finalmente la imagen de contenedor (imagen del cubo de datos), variables de entorno y contenedor de tipo privilegiado para las definiciones de trabajo. Lo anterior debido a que esto no influirá para ninguno de los casos que pasaremos a evaluar y explorar a continuación.

Resultados y Análisis

Escalamiento

Descripción

En este experimento se busca corroborar que el escalamiento, tanto hacia arriba como hacia abajo, prometido por el servicio de AWS Batch funciona de manera correcta. Para lograr comprobar lo anterior, lanzaremos una serie de 30 trabajos concurrentes y evaluaremos cómo se maneja su procesamiento. Particularmente estaremos interesados en resolver las siguientes preguntas: ¿Cuántas instancias son lanzadas?, ¿Cuánto tiempo tarda en aumentar o disminuir la cantidad de instancias cuando la carga de trabajo supera a la capacidad actual?, ¿Cuánto tiempo tardan en empezar la ejecución de los trabajos desde el momento en que son creados?, ¿Cuánto tiempo tarda la ejecución de un trabajo?

Entorno

Para este experimento se presenta la siguiente configuración realizada fue la siguiente.

- Entornos de computo (#1):
 - Tipo de instancias: R5.4xlarge
 - Modelo de pago: On-Demand
 - CPUs virtuales: Min 0, Deseados 16, Max 80

Es necesario aclarar además que se colocan los CPUs virtuales mínimos en cero con el fin de que cuando no exista carga de trabajo, todas las instancias lanzadas por Batch sean terminadas de forma automática, en caso de que se coloque como mínimo un valor diferente de cero, en ese caso siempre estará activa por lo menos una instancia.

También es necesario notar que en este caso el escalamiento se hace a nivel de número de CPUs, no de número de instancias, por tanto, la manera de calcular el intervalo en la cantidad de posibles instancias es necesario conocer el tipo de instancia que se está lanzando. En nuestro caso, el número de instancias debe oscilar entre $[0, 5]$ debido a que estamos utilizando una instancia r5.4xlarge que posee 16 CPUs virtuales, como se observa a continuación en la figura 6.1.1.

Modelo	CPU virtual	Memoria (GiB)
r5.large	2	16
r5.xlarge	4	32
r5.2xlarge	8	64
r5.4xlarge	16	128

Figura 6.1.1. Instancias R5

Adicionalmente se menciona que se escogió este tipo de instancia debido a que las instancias de tipo R poseen optimización para las tareas exigentes a nivel de memoria y, en particular la generación 5, posee la mejor relación costo/beneficio con un 20% de reducción de costos con relación a la generación 4 y mayor aprovisionamiento de memoria RAM por cada CPU virtual en su posesión [16]. Podemos observar que esta máquina nos ofrece una memoria RAM de 128 GiB, este dato nos será de gran utilidad en uno de nuestros siguientes experimentos, donde validaremos la cantidad de trabajos que pueden ser ejecutados de forma concurrente en cada una de las máquinas.

Llamaremos al entorno previamente descrito como, Entorno #1 ya que estaremos reutilizándolo para futuros experimentos.

- Colas:
 - Prioridad: 100
 - Entornos asociados: Entorno #1
- Definiciones de trabajo
 - CPUs: 4
 - Memoria: 61100 MiB.

Para el cálculo de la memoria RAM que debería de usar cada trabajo se utilizó la siguiente lógica. Teniendo en cuenta de que cada procesamiento de una imagen consume alrededor de 64GB de RAM, en primera instancia se consideró la relación existente entre 1MB y 1MiB, la cual se presenta a continuación.

$$\frac{1000^2 B}{1024^2 B} = \frac{1 MB}{1 MiB} = 0.954$$

Ahora, teniendo en cuenta esta relación, se hizo el siguiente cálculo:

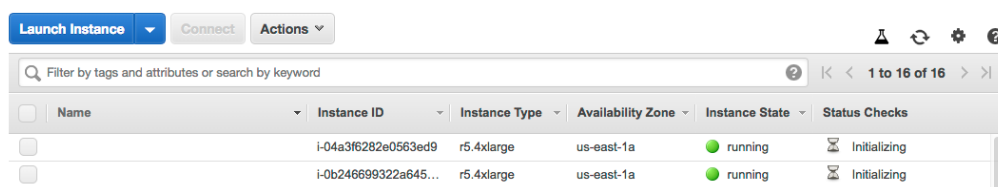
64GB = (64 * 1000) MB = 64000MB

64000MB * 0.954 = 61056MiB.

Se añadió un poco más de memoria adicional por si acaso y se quedó en 61100MiB.

Resultados

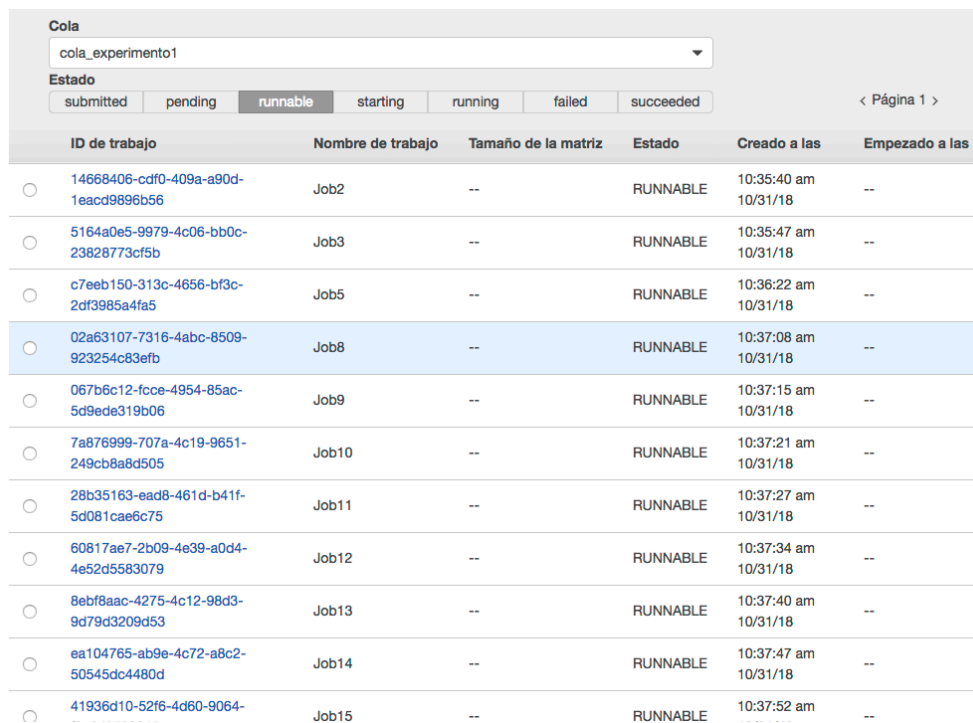
A continuación, se presentan los resultados obtenidos por el experimento. En primer lugar, observamos que de manera inmediata al inicio del experimento se lanzaron dos instancias, como se puede observar en la figura 6.1.2.



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
	i-04a3f6282e0563ed9	r5.4xlarge	us-east-1a	running	Initializing
	i-0b246699322a645...	r5.4xlarge	us-east-1a	running	Initializing

Figura 6.1.2. Instancias inicializándose.

Lo anterior correspondió con la realidad puesto que, en la lista de trabajos encolados, pasaron de estado RUNNABLE a STARTING solamente cuatro de ellos como se observa al comparar las figuras 6.1.3 y 6.1.4.



ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado	Creado a las	Empezado a las
14668406-cdf0-409a-a90d-1eacd9896b56	Job2	--	RUNNABLE	10:35:40 am 10/31/18	--
5164a0e5-9979-4c06-bb0c-23828773cf5b	Job3	--	RUNNABLE	10:35:47 am 10/31/18	--
c7eeb150-313c-4656-bf3c-2df3985a4fa5	Job5	--	RUNNABLE	10:36:22 am 10/31/18	--
02a63107-7316-4abc-8509-923254c83efb	Job8	--	RUNNABLE	10:37:08 am 10/31/18	--
067b6c12-fcce-4954-85ac-5d9ede319b06	Job9	--	RUNNABLE	10:37:15 am 10/31/18	--
7a876999-707a-4c19-9651-249cb8a8d505	Job10	--	RUNNABLE	10:37:21 am 10/31/18	--
28b35163-ead8-461d-b41f-5d081cae6c75	Job11	--	RUNNABLE	10:37:27 am 10/31/18	--
60817ae7-2b09-4e39-a0d4-4e52d5583079	Job12	--	RUNNABLE	10:37:34 am 10/31/18	--
8ebf8aac-4275-4c12-98d3-9d79d3209d53	Job13	--	RUNNABLE	10:37:40 am 10/31/18	--
ea104765-ab9e-4c72-a8c2-50545dc4480d	Job14	--	RUNNABLE	10:37:47 am 10/31/18	--
41936d10-52f6-4d60-9064-...	Job15	--	RUNNABLE	10:37:52 am	--

Figura 6.1.3. Trabajos encolados

Cola						
cola_experimento1						
Estado						
submitted pending runnable starting running failed succeeded						
< Página 1 >						
ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado	Creado a las	Empezado a las	
<input type="radio"/> 0bbdb7ea-f95a-42c3-b080-16b4ec854ffe	Job1	--	STARTING	10:35:33 am 10/31/18	--	
<input type="radio"/> 9f2628dd-b9e4-4b70-8344-94c385af150c	Job4	--	STARTING	10:35:53 am 10/31/18	--	
<input type="radio"/> d457f2c8-16e2-4f39-9aed-8be6f594138d	Job6	--	STARTING	10:36:29 am 10/31/18	--	
<input type="radio"/> 0a6a3b6a-3e99-4da7-ba29-3a4c3fe54c24	Job7	--	STARTING	10:37:01 am 10/31/18	--	

Figura 6.1.4 Trabajos inicializando.

De la anterior figura y comportamiento se puede obtener una conclusión. La cola utilizada para almacenar los trabajos enviados no es una cola FIFO (first in first out) sino que para lograr mantener una arquitectura de alta intensidad de tráfico los trabajos no necesariamente serán procesados en el orden que son enviados. Lo anterior puede que no sea de gran importancia para el IDEAM, sin embargo, se menciona pues es algo que se deberá de tener en cuenta para otros casos de uso.

Adicionalmente a partir de esta figura podemos obtener una hipótesis, se puede apreciar como solamente paran de estado RUNNABLE a estado STARTING 4 trabajos, lo anterior tiene sentido teniendo en cuenta que dadas las características de estas máquinas (128GiB de RAM) y la definición de los trabajos enviados (61100MiB de RAM), cada máquina solo será capaz de ejecutar dos trabajos de forma concurrente, como son dos máquinas entonces en total solo se pueden procesar cuatro trabajos al tiempo. Lo anterior se pudo corroborar ingresando por SSH a la consola de estas máquinas y verificar la cantidad de contenedores ejecutándose como se aprecia en 6.1.5. Recordemos que cada trabajo en la práctica será ejecutado como un contenedor individual en la máquina destino.

```
[ec2-user@ip-192-168-0-238 ~]$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
98336799a878	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	19 seconds ago	Up 17 seconds		ecs-cube_job
-5-default-d6e7a7afb39b6565d681	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	48 seconds ago	Up 47 seconds		ecs-cube_job
-5-default-849ac4e68f8197bd8881	amazon/amazon-ecs-agent:latest	"/agent"	23 minutes ago	Up 23 minutes		ecs-agent

```
[ec2-user@ip-192-168-0-238 ~]$
```

Figura 6.1.5. Contenedores en ejecución.

Podemos apreciar como en la máquina solo ejecutan dos contenedores concurrentes equivalentes a los trabajos enviados y un contenedor adicional que debe ejecutarse en todo momento que corresponde al agente de ECS de AWS. Lo anterior se pudo confirmar cuando después de un rato, al seguir existiendo una carga de trabajo superior a la capacidad actual (dos máquinas), el servicio lanzó una tercera instancia y la cantidad de trabajos procesados de forma concurrente aumentó de cuatro a seis. Como se puede observar en las figuras 6.1.6.1, 6.1.6.2 y 6.1.6.3.

<input type="checkbox"/>	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
<input type="checkbox"/>		i-04a3f6282e0563ed9	r5.4xlarge	us-east-1a	running	2/2 checks passed
<input type="checkbox"/>		i-052c0988641484ab1	r5.4xlarge	us-east-1f	terminated	
<input checked="" type="checkbox"/>		i-0a11129b0b00f34b7	r5.4xlarge	us-east-1a	running	Initializing
<input type="checkbox"/>		i-0b246699322a645...	r5.4xlarge	us-east-1a	running	2/2 checks passed
<input type="checkbox"/>	WEB	i-01bb68ff5a39f9e2b	t2.medium	us-east-1a	running	2/2 checks passed

Figura 6.1.6.1. Se inicializa una nueva máquina r5.4xlarge.

Cola						
cola_experimento1						
Estado						
submitted pending runnable starting running failed succeeded						
< Página 1 >						
	ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado	Creado a las	Empezado a las
<input type="radio"/>	28b35163-ead8-461d-b41f-5d081cae6c75	Job11	--	STARTING	10:37:27 am 10/31/18	--
<input type="radio"/>	cf7d0e15-38e1-4222-97f3-d4b32d2f274a	Job32	--	STARTING	10:47:44 am 10/31/18	--

Figura 6.1.6.2. Los Jobs 11 y 32 pasan a estado STARTING.

Cola						
cola_experimento1						
Estado						
submitted pending runnable starting running failed succeeded						
< Página 1 >						
	ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado	Creado a las	Empezado a las
<input type="radio"/>	28b35163-ead8-461d-b41f-5d081cae6c75	Job11	--	RUNNING	10:37:27 am 10/31/18	10:57:21 am 10/31/18
<input type="radio"/>	8816b571-d3d5-450a-8c75-3abd563fce00	Job23	--	RUNNING	10:38:39 am 10/31/18	10:57:35 am 10/31/18
<input type="radio"/>	9f2e3ed5-de21-49b3-a05f-eade2d40f6de	Job24	--	RUNNING	10:38:45 am 10/31/18	10:57:36 am 10/31/18
<input type="radio"/>	cf7d0e15-38e1-4222-97f3-d4b32d2f274a	Job32	--	RUNNING	10:47:44 am 10/31/18	10:57:21 am 10/31/18
<input type="radio"/>	c2ca9b18-1a3a-4114-ae19-7fc4f7ee76aa	Job34	--	RUNNING	10:47:56 am 10/31/18	10:57:36 am 10/31/18
<input type="radio"/>	f1383ffe-2187-47cb-9fdf-b6b4923aab8f	Job36	--	RUNNING	10:48:07 am 10/31/18	10:57:35 am 10/31/18

Figura 6.1.6.3. Se agregan dos Jobs nuevos a los cuatro concurrentes anteriores.

Este comportamiento de seis trabajos concurrentes se mantuvo constante hasta el final del experimento cuando todos los trabajos se lograron ejecutar de manera exitosa como se observa en la figura 6.1.7.1.

Cola						
cola_experimento1						
Estado						
submitted pending runnable starting running failed succeeded						
< Página 1 >						
ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado	Creado a las	Empezado a las	
<input type="radio"/> 0bbdb7ea-f95a-42c3-b080-16b4ec854ffe	Job1	--	SUCCEEDED	10:35:33 am 10/31/18	10:39:29 am 10/31/18	
<input type="radio"/> 14668406-cdf0-409a-a90d-1eacd9896b56	Job2	--	SUCCEEDED	10:35:40 am 10/31/18	10:51:31 am 10/31/18	
<input type="radio"/> 5164a0e5-9979-4c06-bb0c-23828773cf5b	Job3	--	SUCCEEDED	10:35:47 am 10/31/18	10:47:27 am 10/31/18	
<input type="radio"/> 9f2628dd-b9e4-4b70-8344-94c385af150c	Job4	--	SUCCEEDED	10:35:53 am 10/31/18	10:39:29 am 10/31/18	
<input type="radio"/> c7eeb150-313c-4656-bf3c-2df3985a4fa5	Job5	--	SUCCEEDED	10:36:22 am 10/31/18	10:49:29 am 10/31/18	
<input type="radio"/> d457f2c8-16e2-4f39-9aed-8be6f594138d	Job6	--	SUCCEEDED	10:36:29 am 10/31/18	10:47:27 am 10/31/18	
<input type="radio"/> 0a6a3b6a-3e99-4da7-ba29-3a4c3fe54c24	Job7	--	SUCCEEDED	10:37:01 am 10/31/18	10:39:31 am 10/31/18	

Figura 6.1.7.1. Finalización exitosa de los trabajos.

Lo anterior nos quiere decir que nuestra hipótesis es correcta, se pueden ejecutar tan solo dos contenedores por máquina.

Por supuesto y como era de esperarse, una vez finalizada la ejecución de los trabajos, las máquinas utilizadas se apagaron de forma automática como se observa en la figura 6.1.7.2.

<input type="checkbox"/>	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
		i-04a3f6282e0563ed9	r5.4xlarge	us-east-1a	shutting-do...	
		i-052c0988641484ab1	r5.4xlarge	us-east-1f	terminated	
		i-0a11129b0b00f34b7	r5.4xlarge	us-east-1a	shutting-do...	
		i-0b246699322a645...	r5.4xlarge	us-east-1a	shutting-do...	

Figura 6.1.7.1.

Como se puede apreciar, únicamente se utilizaron tres máquinas durante este experimento a pesar de que en principio hicimos ajustes para llegar a utilizar hasta cinco máquinas. La razón de lo anterior se debe a que actualmente existe un límite de cuatro máquinas de este tipo para la cuenta asociada, y una de ellas ya está siendo utilizada de forma permanente para experimentación, como se puede observar en las figuras 6.1.8.1 y 6.1.8.2.

Running On-Demand r5d.4xlarge instances	4	Request limit increase
---	---	--

Figura 6.1.8.1. Límites de instancia de la cuenta.

	AIRFLOW_WORKER_LARGE		i-0ba5c6c453a81e89f	r5.4xlarge	us-east-1a		running		2/2 checks passed
--	----------------------	--	---------------------	------------	------------	--	---------	--	-------------------

Figura 6.1.8.2. Instancia R5.4xlarge activa.

Sin embargo, dada la naturaleza del experimento donde simplemente queríamos observar el escalamiento automático de las máquinas, no se consideró necesario la utilización de instancias adicionales.

Finalmente, cada ejecución posee información que no es de suma utilidad tales como, fecha y hora de creación, fecha y hora de inicio, tiempo de ejecución (calculado a partir de la hora de inicio y hora de finalización) y logs de todo lo que es ejecutado adentro del contenedor a través de cloudwatch, como se puede apreciar en las figuras 6.1.9.1, 6.1.9.2 y 6.1.9.3.

Intentos

Intentos	Código de salida	Hora de inicio	Hora de finalización
1 de 2	-- View logs	10:39:29 am 10/31/18	10:41:11 am 10/31/18

Figura 6.1.9.1. Tiempos de ejecución.

ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado	Creado a las	Empezado a las
0bbdb7ea-f95a-42c3-b080-16b4ec854ffe	Job1	--	SUCCEEDED	10:35:33 am 10/31/18	10:39:29 am 10/31/18

Figura 6.1.9.2. Información de creación del trabajo.

Time (UTC +00:00)	Message
2018-10-31	
	No older events found at the moment. Retry .
15:40:20	<string>:25: RuntimeWarning: Mean of empty slice
15:40:22	/root/anaconda2/lib/python2.7/site-packages/numpy/lib/nanfunctions.py:1423: RuntimeWarning: Degrees of freedom <= 0 for slice
15:40:22	keepdims=keepdims)
15:40:36	/root/anaconda2/lib/python2.7/site-packages/numpy/lib/function_base.py:3858: RuntimeWarning: All-NaN slice encountered
15:40:36	r = func(a, **kwargs)
15:41:08	<string>:38: RuntimeWarning: invalid value encountered in greater
15:41:08	<string>:39: RuntimeWarning: invalid value encountered in less
15:41:11	Executing ndvi v1
	No newer events found at the moment. Retry .

Figura 6.1.9.3. Logs del contenedor ejecutado.

Dichos datos fueron organizados y tabulados en la hoja de Excel anexa y partir de ella se pudo extraer información que permitió llegar a las siguientes conclusiones.

Conclusiones

A partir del experimento realizado, podemos obtener las siguientes conclusiones.

En primera instancia teniendo en cuenta los tiempos de creación y de inicio del Job 11, el cual fue el primero en ser ejecutado por la tercera instancia lanzada, podemos observar que tomó alrededor de 20 minutos para que AWS Batch decidiera incrementar la capacidad de computó. Dicho tiempo contrasta bastante con el tiempo tomado para apagar las instancias que fue alrededor de 5 minutos. Por tal motivo si bien este escalamiento es auto gestionado por AWS, si se requiere de un escalamiento más preciso con parámetros acomodados por el usuario, quizás lo mejor no sea utilizar el modelo manejado por AWS, sino más bien uno totalmente gestionado por el usuario.

Sin embargo, lo que sí es seguro es que AWS se encargara de realizar y procesar correctamente los trabajos y por tanto automatizará el manejo de los recursos usados, reflejándose en disminución de los costos pues solo se pagará por lo que se use cuando se use. Tenemos la opción de configurar timeouts para que, en caso de ocurrir algún error inesperado, nuestros trabajos no ejecuten de forma eterna y por tanto esto repercute en costos adicionales, y también podemos definir una cantidad máxima de intentos para cada trabajo.

En segunda instancia podemos observar que los tiempos de ejecución son aceptables entre [93, 103] Segundos con un promedio de 99. Lo cual al ser conversado con distintos expertos en el cubo es un tiempo bastante decente y es lo normal con lo que se lleva experimentado hasta el momento.

Finalmente, cada máquina puede correr hasta N contenedores donde N es:

$$N = \frac{\text{Recursos/Contenedor}}{\text{Recursos totales}}$$

Recordemos que cada definición de trabajo puede especificar los recursos a nivel de CPU y Memoria que requiere de la máquina Host y que la máquina Host, naturalmente, posee una cantidad limitada de recursos limitada que son explícitas por parte del proveedor. En nuestro, y como se explicó anteriormente, dadas las características de ambos, definición de trabajo y máquina, nuestro límite era de dos contenedores por máquina. Sin embargo, lo anterior implica que los encargados de definir los trabajos deben conocer de forma previa la cantidad de recursos que son consumidos por su programa y por consiguiente consigue plantearnos la siguiente cuestión: ¿Qué sucede si se realiza una definición de trabajo errónea?, esta cuestión será respondida en el siguiente experimento en el cual haremos pruebas con definiciones que exigen requerimientos que no corresponden con la realidad.

Número de contenedores por máquina

Descripción

En este experimento se busca corroborar la correlación existente entre la definición de trabajo con la capacidad de la instancia para determinar la cantidad de contenedores que puede ejecutar concurrentemente. En este experimento en particular buscamos descubrir qué sucede si se especifican de forma errónea los requerimientos del trabajo.

Entorno

Para este experimento se presenta la siguiente configuración realizada fue la siguiente.

- Entornos de computo (#1)
- Colas:
 - Prioridad: 20
 - Entornos asociados: Entorno #1
- Definiciones de trabajo
 - CPUs: 4
 - Memoria: 9000 MiB.

Como podemos observar, conservamos la configuración del experimento anterior con la diferencia de que en esta definición de trabajo reducimos a la mitad la cantidad de memoria que, en teoría, va a consumir nuestro contenedor. Por tal motivo ahora cada una de las máquinas debería ser capaz de soportar hasta catorce contenedores concurrentes. Sin embargo, nuestros contenedores están configurados para utilizar hasta un máximo de 9000MiB de memoria cada uno de ellos, por lo cual en caso de que lleguen a necesitar más de esta cantidad de memoria, estos serán “matados” de forma automática [18]. Se especificó esta cantidad de memoria debido a que se jugó con distintas configuraciones, tal como se observa en la figura 6.2.1 y se logró identificar que el algoritmo que se está manejando para este experimento, ndvi, no llega a utilizar un consumo de 64GB de Ram, sino que requiere un valor mínimo de entre (9000, 10000MiB) de memoria Ram para ejecutar de forma satisfactoria.

	Definiciones de trabajo	CPU virtuales	Memoria (MiB)
<input type="radio"/>	Revision 8	1	10000
<input type="radio"/>	Revision 11	1	9000
<input type="radio"/>	Revision 3	1	64000
<input type="radio"/>	Revision 4	1	61100
<input type="radio"/>	Revision 9	1	5000
<input type="radio"/>	Revision 10	1	8000
<input type="radio"/>	Revision 7	1	20000
<input type="radio"/>	Revision 6	1	30000
<input type="radio"/>	Revision 2	1	64512
<input type="radio"/>	Revision 1	1	64512
<input type="radio"/>	Revision 5	1	1024

Figura 6.2.1. Revisiones de definiciones de trabajo utilizadas.

Por tal motivo se intentó encontrar la cantidad mínima de memoria que se necesitaba para ejecutar de forma correcta este algoritmo con el fin de colocar un poco menos de capacidad con el fin de reducir la cantidad de contenedores que serán ejecutados por la máquina host a la vez que se envían suficientes trabajos para que se lance una segunda instancia y verificar la cantidad de contenedores que ejecutan sobre cada una.

Resultados

En primera instancia se lanzaron quince trabajos concurrentes, lo anterior con el fin de superar ligeramente la capacidad máxima de un host y levantar al menos dos instancias. En principio el comportamiento fue el esperado donde se puede apreciar en las figuras 6.2.2.1, 6.2.2.2 y 6.2.2.3 que se levantó una sola instancia, catorce de los trabajos pasaron inmediatamente a estado STARTING y el trabajo número quince fue el único que se quedó en estado RUNNABLE, respectivamente.

<input type="checkbox"/>	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
<input type="checkbox"/>		i-0505f5ffa893ff802	r5.4xlarge	us-east-1a	● running	Initializing

Figura 6.2.2.1 Lanzamiento de una primera instancia.

Cola			
cola_exprimento2			
Estado			
<input type="button" value="submitted"/> <input type="button" value="pending"/> <input type="button" value="runnable"/> <input checked="" type="button" value="starting"/> <input type="button" value="running"/> <input type="button" value="failed"/> <input type="button" value="succeeded"/>			
ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado
<input type="radio"/> e56af6cf-b075-4410-b420-f65f062feeca	Job1	--	STARTING
<input type="radio"/> 6c7b6508-ef80-4552-b869-4f1a8c93ce9c	Job2	--	STARTING
<input type="radio"/> 364c64fc-247e-44c1-ba5a-bcf24d39008b	Job3	--	STARTING
<input type="radio"/> 7e201658-8420-4694-aa4b-3d45347d3bba	Job4	--	STARTING
<input type="radio"/> a43ce12e-3790-4249-b697-823a1f536f94	Job5	--	STARTING
<input type="radio"/> 2baa763b-79b4-49b7-b2b1-5f5ea13bf4b4	Job6	--	STARTING
<input type="radio"/> 88e3456c-ad3b-4648-9946-61d387500ec4	Job7	--	STARTING
<input type="radio"/> da8e552b-cc92-457b-a467-1ebe6dbef120	Job8	--	STARTING
<input checked="" type="radio"/> ae129569-6eb5-420a-9990-7608338d4ab7	Job9	--	STARTING
<input type="radio"/> 73269b44-3a9f-4808-ae15-5719386024e5	Job10	--	STARTING

Figura 6.2.2.2 Jobs en estado STARTING.

Cola			
cola_exprimento2			
Estado			
<input type="button" value="submitted"/> <input type="button" value="pending"/> <input checked="" type="button" value="runnable"/> <input type="button" value="starting"/> <input type="button" value="running"/> <input type="button" value="failed"/> <input type="button" value="succeeded"/>			
ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado
<input type="radio"/> bdc81ae-8104-4bd0-9cea-402af332d627	Job15	--	RUNNABLE

Figura 6.2.2.3. Job quince RUNNABLE.

Posteriormente a esto se procedió a iniciar sesión dentro de la instancia generada para corroborar que efectivamente se hubieran generado los catorce contenedores, lo cual resultó ser cierto como se puede observar en la figura 6.2.3.1.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d44c2be2b53a	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 26 seconds		ecs-cube_job
-12-default-b6cd9af2e0acadaefeb01	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 27 seconds		ecs-cube_job
b7fed430bb85	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 29 seconds		ecs-cube_job
-12-default-daf9ebec0f99386c401	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 29 seconds		ecs-cube_job
ae8771ab5947	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 29 seconds		ecs-cube_job
-12-default-c6f39ec89db4b5b1d101	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 33 seconds		ecs-cube_job
6734b5aca5f5	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 33 seconds		ecs-cube_job
-12-default-92cbddda382eada2100	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 34 seconds		ecs-cube_job
3c78a1fbfaac	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 35 seconds		ecs-cube_job
-12-default-868e87c687c0a7f5f901	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 37 seconds		ecs-cube_job
9cdc6e919857	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 39 seconds		ecs-cube_job
-12-default-90dd8a80e2dfe2e89b01	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 40 seconds		ecs-cube_job
0f564072319	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 41 seconds		ecs-cube_job
-12-default-bcb5dad390bd93f6c601	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
06ecd1d7809e	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
-12-default-9696dae4f3ede8882100	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
51db29a03858	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
-12-default-c888acfa85e2d5c4d501	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
9679f3c7b370	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
-12-default-e495c1c6999bc9cc5700	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
1afb9eb93209	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
-12-default-9ec3b2989f9e0eb48e01	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
31d2e4080008	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
-12-default-eaced290a0c19af92f00	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
6e8adffef7a74	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
-12-default-e4c7b9eff30ec78c3100	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
777b96c70c08	186121879828.dkr.ecr.us-east-1.amazonaws.com/datacube_image	"/start.sh /bin/sh -"	44 seconds ago	Up 42 seconds		ecs-cube_job
-12-default-86a7fabbb1d0cfb52400	amazon/amazon-ecs-agent:latest	"/agent"	3 minutes ago	Up 3 minutes		ecs-agent
d6dd838af291	ec2-user@ip-192-168-0-110 ~\$					

Figura 6.2.3.1. Catorce contenedores dentro la máquina host.

Sin embargo, hubo un comportamiento que se reflejó tal cual la teoría. Y es que en ningún momento hubo un lanzamiento de una segunda instancia tal y como se esperaba, en todo momento se mantuvo procesando la carga de trabajo con una única instancia a pesar de que se siguieron enviando trabajos para forzar este comportamiento. No obstante, se pudo identificar la causa del anterior comportamiento, debido a que los contenedores fallaban de forma rápida su ejecución, estos volvían a ser encolados y los contenedores que se encontraban en estado RUNNABLE procedían a intentar ser procesados, como podemos observar en la figura 6.2.4.1 donde el trabajo uno regresó al estado RUNNABLE junto con algunos trabajos nuevos.

Cola

cola_exprimto2

Estado

submitted

pending

runnable

starting

running

failed

succeeded

	ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado	Creado a las
<input checked="" type="radio"/>	e56af6cf-b075-4410-b420-f65f062feeca	Job1	--	RUNNABLE	03:33:55 pm 11/01/18
<input type="radio"/>	2c258bd2-8897-4817-8d41-b60b2dcce850	Job16	--	RUNNABLE	03:39:11 pm 11/01/18
<input type="radio"/>	74a316a0-2330-49fc-89dc-fc3ce69b23ba	Job19	--	RUNNABLE	03:39:27 pm 11/01/18
<input type="radio"/>	2ec69421-d144-411a-ae51-236a69c3ab8d	Job21	--	RUNNABLE	03:39:39 pm 11/01/18

Figura 6.2.4.1. Job uno en estado RUNNABLE junto a Jobs nuevos.

Finalmente, cuando cada uno de los trabajos finalizó todos sus intentos, configurados en tres, estos pasaron a estado FAILED, donde indagando en los detalles de su ejecución se pudo

apreciar que fallaron debido a que agotaron toda la memoria que les fue asignada para su ejecución como se puede observar en las figuras 6.2.5.1, 6.2.5.2 y 6.2.5.3.

Cola			
cola_experimento2			
Estado			
<input type="button" value="submitted"/> <input type="button" value="pending"/> <input type="button" value="runnable"/> <input type="button" value="starting"/> <input type="button" value="running"/> <input checked="" type="button" value="failed"/> <input type="button" value="succeeded"/>			
ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado
<input type="radio"/> e56af6cf-b075-4410-b420-f65f062feeca	Job1	--	FAILED
<input type="radio"/> 6c7b6508-ef80-4552-b869-4f1a8c93ce9c	Job2	--	FAILED
<input type="radio"/> 364c64fc-247e-44c1-ba5a-bcf24d39008b	Job3	--	FAILED
<input checked="" type="radio"/> 7e201658-8420-4694-aa4b-3d45347d3bba	Job4	--	FAILED
<input type="radio"/> a43ce12e-3790-4249-b697-823a1f536f94	Job5	--	FAILED
<input type="radio"/> 2baa763b-79b4-49b7-b2b1-5f5ea13bf4b4	Job6	--	FAILED
<input checked="" type="radio"/> 88e3456c-ad3b-4648-9946-61d387500ec4	Job7	--	FAILED
<input type="radio"/> da8e552b-cc92-457b-a467-1ebe6dbef120	Job8	--	FAILED
<input type="radio"/> ae129569-6eb5-420a-9990-7608338d4ab7	Job9	--	FAILED
<input type="radio"/> 73269b44-3a9f-4808-ae15-5719386024e5	Job10	--	FAILED

Figura 6.2.5.1. Todos los Jobs Fallaron.

Estado del trabajo

Estado FAILED	
Razón para el estado	Essential container in task exited
Mensaje de contenedor	OutOfMemoryError: Container killed due to memory usage

Figura 6.2.5.2. Estado del Job.

Intentos

Intentos	Código de salida
1 de 3	137 - View logs
Motivo: OutOfMemoryError: Contain	
2 de 3	137 - View logs
Motivo: OutOfMemoryError: Contain	
3 de 3	137 - View logs
Motivo: OutOfMemoryError: Contain	

Figura 6.2.5.3. Logs de Ejecución del Job.

Conclusiones

A partir de este experimento realizado se pudo llegar a las siguientes conclusiones.

El proveedor ya lo especificaba, pero la cantidad de memoria definida para cada trabajo no aplica sobre la máquina host, aplica sobre los contenedores y por tanto cuando estos superan el uso de dicha memoria, estos son matados automáticamente. El Host únicamente utiliza esta información para asumir hasta cuántos contenedores puede soportar con el fin de que Batch pueda decidir cuándo solicitar una nueva instancia.

Sin embargo, lo realmente interesante son las posibilidades que esto ofrece, ya que, como se pudo apreciar este algoritmo consumía una cantidad considerablemente menor de memoria con relación a otros algoritmos. Por tal motivo, la creación de distintas definiciones de trabajo con distintos requerimientos le otorgaría al IDEAM una mayor flexibilidad que permitiría sacar provecho de las principales promesas de Batch, como lo es el manejo óptimo de recursos y de selección de instancias, con el cual pretendemos experimentar a continuación. Lo anterior debido a que en principio se escogió la instancia R5.4xlarge ya que en principio podía manejar en múltiplos de dos la cantidad de tareas concurrentes, pero al crear definiciones de trabajo para 10 y 64GB de Ram, quizás una máquina con una cantidad inusual de memoria, como pueden ser la r5.12xlarge que posee 384 GB de Ram o una combinación de la r5.4xlarge con instancias r5. xlarge que poseen 32GB de Ram [16]. Todo esto, ampliaría las opciones de selección de Batch para escoger la mejor instancia dependiendo de la situación, y por supuesto, nuevamente esto se reflejaría en los costos pues entre más grande la máquina, mayor serán sus costos sin importar si se está utilizando la totalidad de su capacidad.

Entorno de cómputo optimizado – Una Sola Definición

Descripción

El objetivo de este experimento es generar un entorno de cómputo nuevo en el cual no seremos nosotros quienes haremos el dimensionamiento y definiremos el tipo de máquina que queremos utilizar, sino que dejaremos que AWS se encargue de esto a partir de la información que obtenga de las definiciones de trabajo que le vayan llegando. En este caso particular manejaremos una sola definición de trabajo, en particular la revisión cuatro, la cual fue utilizada para el primer experimento. El objetivo de este experimento consiste en comparar los tiempos de levantamiento, de apagado y de ejecución de cada uno de los distintos trabajos con relación a los tiempos obtenidos en el primer experimento, y en segundo lugar comparar los costos asociados a las instancias levantadas por Batch con relación a las instancias R5 con el fin de evaluar si esta solución es óptima no solo a nivel de desempeño sino también a nivel costo.

Para cumplir con lo anterior, el nuevo entorno se someterá a una prueba similar a la del primero, es decir, 30 trabajos concurrentes.

Entorno

- Entornos de computo (#2):
 - Tipo de instancias: Optimal
 - Modelo de pago: On-Demand
 - CPUs virtuales: Min 0, Deseados 16, Max 80
- Colas:
 - Prioridad: 100
 - Entornos asociados: Entorno #2
- Definiciones de trabajo
 - CPUs : 4
 - Memoria: 61100 MiB.

Resultados

En principio lo que se observó fue que Batch lanzó de forma inmediata una instancia c4.xlarge. Lo cual no tuvo ningún sentido dado que estas instancias están optimizadas para alta capacidad de cómputo y, por tanto, poseen una relación bastante baja de memoria por cada una de sus CPU virtuales. De hecho, este tipo de instancia solamente posee 30GiB de RAM, ni siquiera la cantidad necesaria para poder correr un solo trabajo. Este comportamiento se puede apreciar en la figura 6.3.1.

	i-0a4f1fd1c8bd5f90b	c4.4xlarge	us-east-1a	running	Initializing
--	---------------------	------------	------------	---------	--------------

Figura 6.3.1. Inicialización de instancia C4

Sin embargo, pareciera que Batch notó rápidamente su error pues en un lapso de unos aproximadamente dos minutos, esta instancia se empezó a apagar y Batch lanzó una nueva instancia, esta vez se trataba de una R4.16xLarge, una instancia optimizada para uso de memoria y con una capacidad de 488GiB de esta. Este comportamiento se puede apreciar en la figura 6.3.2.

	i-06c05f54a113d908c	r4.16xlarge	us-east-1a	pending	Initializing
	i-0a4f1fd1c8bd5f90b	c4.4xlarge	us-east-1a	shutting-do...	

Figura 6.3.2. Inicialización de instancia R4.16xlarge.

Esto generó que se empezaran a procesar los trabajos en lotes de ocho, como se observa en la figura 6.3.3.

Cola

cola_experimento3

Estado

submitted

pending

runnable

starting

running

failed

succeeded

	ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado
<input type="radio"/>	5afa212b-a955-43a0-bfea-20349c9a578d	Job5	--	STARTING
<input type="radio"/>	f05aa4e2-1de7-4c49-9a0d-c5ae2feb76a8	Job10	--	STARTING
<input type="radio"/>	3ceff747-a1bb-4ffd-b82b-0c51d556d268	Job13	--	STARTING
<input type="radio"/>	942d4ab7-f578-405a-be1b-21dce50e79c8	Job21	--	STARTING
<input type="radio"/>	11af1714-73a0-4512-9038-188d318951c6	Job22	--	STARTING
<input type="radio"/>	40e6758f-116f-43b0-8793-0c52019f5ff5	Job23	--	STARTING
<input type="radio"/>	8f26fc48-19dc-44d2-96c1-0d1348dfa23c	Job24	--	STARTING
<input type="radio"/>	e9e6a56b-21fc-4b80-8945-ce00bc25930f	Job25	--	STARTING

Figura 6.3.3. Procesamiento de ocho Jobs concurrentes.

Sin embargo, aún quedaban bastantes trabajos que debían ser procesados, de tal forma que después de un tiempo Batch empezó la inicialización de una segunda instancia, en este caso se trataba de una m4.4xlarge, la cual poseía una capacidad de 64GiB y 16 CPU virtuales, que en conjunto con los 64 que poseía la R4 completaron nuestro límite de hasta 80 CPU virtuales.

Adicionalmente esto permitió la adición de dos trabajos concurrentes más para un total de diez contenedores al mismo tiempo. El lanzamiento de la instancia se puede observar en la figura 6.3.4.

	i-06c05f54a113d908c	r4.16xlarge	us-east-1a	running	2/2 checks passed
	i-0a4f1fd1c8bd5f90b	c4.4xlarge	us-east-1a	terminated	
	i-0cc51d10a6ef0e3e9	m4.4xlarge	us-east-1a	pending	Initializing

Figura 6.3.4. lanzamiento de la tercera instancia, una m4.4xlarge.

A partir de la ejecución de este experimento, se tabuló y organizó la información en el Excel anexo con el fin de obtener los tiempos de lanzamiento, ejecución y tiempo total de cómputo para poder compararlos con los resultados del primer experimento. Lo anterior con el fin de saber si esta opción resultaba más eficiente a nivel desempeño y a nivel de costos.

En primer lugar, anexaremos el resumen de los resultados obtenidos ambos experimentos. Se aclara que todos los tiempos están en segundos a no ser que se especifique lo contrario.

Min	92
Max	103
Promedio	99,56666667
Ejecución total:	1345 segundos, 22,416Minutos
Tiempo total:	1641 Segundos, 27.35Minutos
Levantamiento 1	3Minutos, 56 Segundos
Levantamiento 2	19 Minutos, 54 Segundos
Descenso	5 Minutos

Tabla 6.3.1. Resultados experimento 1

Min	99
Max	120
Promedio	110,03333333
Ejecucion total:	610 Segundos, 10.16Minutos
Tiempo total:	1472 Segundos, 24.53Minutos
Levantamiento 1	861 Segundos, 14,1 Minutos
Levantamiento 2	1160 Segundos, 19.33 Minutos
Descenso	5 Minutos

Tabla 6.3.2. Resultados experimento 3.

Podemos observar claramente como los tiempos de ejecución individuales de cada trabajo son menores en el experimento uno. Sin embargo, estamos hablando de una diferencia aproximada de diez segundos por contenedor, lo cual podría resultar en un problema si los contenedores fueran procesados de forma secuencial, pero al ser procesados en paralelo dentro de una misma máquina host, esto no presenta una ventaja significativa del experimento uno por sobre el experimento tres. Sin embargo, puede ser que para el IDEAM un retraso de diez segundos afecte totalmente su caso de uso.

También se puede ver como para el experimento tres, el tiempo de cómputo y de ejecución total fue inferior al del experimento uno, esto resulta totalmente lógico pues en el experimento uno apenas se logró llegar a tres máquinas de 64GiB, lo cual permitía una ejecución de apenas seis contenedores concurrentes, mientras que en este experimento se logró alcanzar hasta diez contenedores al mismo tiempo.

Siendo este el caso, es bastante claro que el experimento tres presenta mejores tiempos de ejecución. Es cierto, no sabemos qué hubiera podido suceder si en el experimento uno hubiéramos alcanzado una totalidad de cinco máquinas r5.4xlarge que nos hubieran permitido llegar hasta una ejecución similar de diez contenedores concurrentes. No obstante, teniendo en cuenta los tiempos prolongados de levantamiento de nuevas instancias que se pueden observar en ambos experimentos, se puede intuir que, aunque los tiempos del primer experimento hubieran podido reducirse, es muy probable que no hubiesen sido capaces de superar los obtenidos por el tercero, quien al haber levantado de forma inmediata una máquina capaz de soportar hasta ocho trabajos concurrentes, pudo manejar una carga de trabajo continua y operarla de forma más rápida. Por tanto, se puede decir que, de alguna manera, la selección “óptima” de la instancia es más eficiente.

Ahora, aunque hemos determinado de que los entornos óptimos son más eficientes que aquellos dimensionados por el usuario, al menos según lo que se observa en nuestro experimento, la pregunta que surge es. ¿Son los entornos óptimos más económicos? Para descubrirlo tomaremos ventaja del hecho de que AWS cobra los costos de sus instancias por segundo [19].

Para el primer experimento los costos se calculan de la siguiente forma. Las instancias r5.4xlarge utilizadas en el primer experimento tienen un costo de 1.008 USD por hora [20], lo cual nos da un costo de 0.00028 USD por segundo. Ahora, se utilizaron 1345 segundos de cómputo en total, donde se utilizaron siempre dos de estas máquinas, dando de esta forma un total de 0.3766 USD. Ahora, la tercera instancia comenzó 964 segundos después de que empezara la ejecución. Lo anterior implica que la tercera máquina solamente ejecutó por $1345 - 964 = 381$ segundos, dando unos costos adicionales de 0.10668 USD. Por tanto:

$$\text{Costos Experimento 1} = (1345 * 0.00028) * 2 + (381 * 0.00028) = 0.48328$$

Aplicando una lógica análoga para el experimento tres obtenemos que las máquinas r4.16xlarge y m4.4xlarge tienen unos costos de 4.256 USD por hora y 0.80 USD por hora respectivamente, dándonos unos costos de 0.00118 y 0.00022 USD por segundo. El tiempo

de ejecución total fueron de 610 segundos, de los cuales la máquina m4 solo operó $610 - 375 = 235$ segundos. Por tanto, los costos del experimento tres fueron:

$$\text{Costos Experimento 3} = (610 * 0.00118) + (235 * 0.00022) = 0.7198 + 0.0517 = 0.7715.$$

Por tanto, estaríamos hablando que el experimento tres alcanzó unos costos de alrededor de un 60% por encima del experimento uno. Por supuesto, esto tiene sentido dado que este experimento ocupó un poder computacional superior al del primer experimento, por tanto, vamos a suponer que junto a la tercera instancia lanzada se hubieran lanzado otras dos para un total de cinco máquinas.

$$\text{Costos Experimento 1 (Hipotético)} = 0.48328 + (235 * 0.00022) * 2 = 0.58668$$

Podemos observar como de igual forma en este escenario el experimento tres sigue siendo un 24% más costoso que el experimento uno, y, de hecho, el primer experimento podría llegar a ser más económico dado que un aumento en las máquinas hubiera reducido los tiempos de ejecución.

Conclusiones

Teniendo en cuenta la información adquirida durante el experimento se pueden sacar las siguientes conclusiones.

En primer lugar, AWS Batch presenta cierta inteligencia al momento de determinar el tipo de instancia que debe ser utilizada para procesar la carga de trabajo, en caso de que lance una instancia no apta para hacerse cargo, el mismo Batch procederá a evaluar las definiciones de trabajo para saber qué tipo de instancia utilizar.

En segundo lugar, se puede decir que, a partir de los resultados obtenidos en ambos experimentos, los entornos óptimos pueden llegar a ser más eficaces en cuanto al desempeño y los tiempos de ejecución de los trabajos, sin embargo, como se pudo observar, un buen dimensionamiento para cuando las cargas de trabajo son conocidas desembocará en un mejor rendimiento de costos para la organización que al largo plazo representarán un gran ahorro.

Sin embargo, en el escenario donde las cargas de trabajo sean variables, con distintas definiciones de trabajo, puede que este sea un escenario donde los entornos óptimos logren sacar su mayor potencial, y esto es precisamente lo que se quiere evaluar con el siguiente experimento.

Entorno de cómputo optimizados – Múltiples Definiciones

Descripción

Como pudimos observar en el experimento dos, distintos algoritmos poseen distintos requerimientos, y por tanto es necesario hacer su respectivo dimensionamiento y definición de trabajo. Al hacer esto, es probable que la misma máquina no sea la más conveniente ya sea a nivel desempeño o nivel costo para solucionar una carga de trabajo específica. Es por este motivo que quizás los entornos óptimos sean la solución rápida a esta situación en la cual se deben tener en cuenta una gran cantidad de variables. Al igual que en los experimentos uno y tres, lo que nos interesará será ver el comportamiento de las instancias lanzadas y los tiempos de ejecución y costos que se generan durante el experimento.

Dentro de este experimento pretendemos observar y evaluar el comportamiento de una cola cuando esta es asociada a distintos entornos. Esta pretende ser una prueba sencilla de validación, pero que nos será de gran utilidad en los experimentos venideros pues, queremos ver lo que sucede cuando se agotan los recursos de alguno de los entornos asociados, o peor aún, cuando uno de estos comienza a presentar fallas. ¿Hacia qué entorno se dirigen inicialmente los trabajos de la cola?, ¿Hacia donde se dirigen estos cuando los entornos comienzan a fallar?, estas son las dudas que pretendemos solucionar con este experimento.

Entorno

- Entornos de computo (#2):
 - Tipo de instancias: Optimal
 - Modelo de pago: On-Demand
 - CPUs virtuales: Min 0, Deseados 16, Max 80
- Colas:
 - Prioridad: 100
 - Entornos asociados: Entorno #2
- Definiciones de trabajo (Revision 5 y 8)
 - CPUs: 4
 - Memoria: 61100 MiB.
- CPUs: 1
- Memoria: 10000Mib

Resultados

Para este experimento, al igual que los demás, se enviaron 30 trabajos concurrentes con la diferencia de que en esta ocasión los trabajos del 1 al 10 eran trabajos de 64Gb de memoria y los otros veinte eran trabajos destinados a ocupar 10000Mib. Con esta característica en mente el primer suceso que se observó es que Batch lanzó de manera inmediata dos instancias, una m4.16xlarge de 256GiB de memoria y una m4.4xlarge de 64GiB de RAM como se observa en la figura 6.4.1.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs	Key Name	Monitoring	Launch
	i-031c871a1448eb17a	m4.4xlarge	us-east-1a	running	Initializing	None	ec2-34-205-8-48.comp...	34.205.8.48	-	cubo-tesis	enabled	Noveml
	i-06b61d53f25e1477	m4.16xlarge	us-east-1a	running	Initializing	None	ec2-18-233-200-201.co...	18.233.200.201	-	cubo-tesis	enabled	Noveml

Figura 6.4.1. Lanzamiento de instancias m4.

Inmediato a este suceso, diez trabajos pasaron a estado RUNNING, cuatro de ellos de la revisión 5, que fueron a ocupar la capacidad total de la máquina 16xlarge, y 6 a la 4xlarge, ocupando así de igual forma toda su capacidad.

Cola		Estado					
cola_experimento4		submitted	pending	runnable	starting	running	failed
ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado				
0955b798-395f-40ec-9fcd-b446aed3f2ad	Job1	--	RUNNING				
f4beb5ae-ee49-4ed1-b5f9-2d60824de1c3	Job2	--	RUNNING				
1a2a09c3-9ffd-4213-aca4-42eece799a99	Job7	--	RUNNING				
8b989473-6b0d-435e-81c0-6d9bfdc65350	Job10	--	RUNNING				
e0f0cf12-4565-4eae-990e-d76078117013	Job12	--	RUNNING				
b31e1485-2ba0-4846-8aaa-73b1ebc71206	Job16	--	RUNNING				
931cc8ce-3548-4a9e-a7f9-6982a359d2d6	Job17	--	RUNNING				
f91c3c8f-985f-40fd-a90a-eedddd62d475	Job23	--	RUNNING				
840abc32-7e73-46e6-a009-fe8962d7f248	Job24	--	RUNNING				
3e65cdc2-9cd8-4d3a-b538-3b0080d784c2	Job29	--	RUNNING				

Figura 6.4.2. Trabajos corriendo concurrentemente.

Este comportamiento prevaleció por algún tiempo, es decir, procesamiento de diez trabajos concurrentes, hasta que finalmente se acabaron los trabajos de la revisión 5 y en ese momento se procedió a un proceso concurrente adicional de trabajos de la revisión 8 como se puede apreciar en la figura 6.4.3.

Cola		Estado					
cola_experimeto4		submitted	pending	runnable	starting	running	failed
ID de trabajo	Nombre de trabajo	Tamaño de la matriz	Estado				
0955b798-395f-40ec-9fcf-b446aed3f2ad	Job1	--	RUNNING				
1a2a09c3-9ffd-4213-aca4-42eece799a99	Job7	--	RUNNING				
8b989473-6b0d-435e-81c0-6d9bdc65350	Job10	--	RUNNING				
e0f0cf12-4565-4eae-990e-d76078117013	Job12	--	RUNNING				
b31e1485-2ba0-4846-8aaa-73b1ebc71206	Job16	--	RUNNING				
931cc8ce-3548-4a9e-a7f9-6982a359d2d6	Job17	--	RUNNING				
bbe9a7f6-8ff7-4edb-a081-03d68868235e	Job21	--	RUNNING				
1f63ad54-9658-417c-896c-964f6131aef	Job22	--	RUNNING				
f91c3c8f-985f-40fd-a90a-eeddd62d475	Job23	--	RUNNING				
840abc32-7e73-46e6-a009-fe8962d7f248	Job24	--	RUNNING				
a220b0b8-5ec0-47b4-a4ee-c7e39ed53f13	Job25	--	RUNNING				
1b6c067e-36ed-4900-90e8-ec4c8c5bb2e2	Job26	--	RUNNING				
3e65cdc2-9cd8-4d3a-b538-3b0080d784c2	Job29	--	RUNNING				
279ecee8-8321-4b60-aaee-9a410a9b3a48	Job30	--	RUNNING				

Figura 6.4.3. Procesamiento de 14 trabajos concurrentes.

En ella podemos observar cómo se pasa de 10 trabajos a 14 trabajos concurrentes de los cuales tres son de la revisión 5 y por tanto consumen una gran cantidad de memoria. Sin embargo, en el momento que estos trabajos acaban la cantidad de trabajos concurrentes aumenta aún más dado que solo quedan trabajos de la revisión 8.

Los resultados de tiempos de este experimento también pueden encontrarse en el Excel anexo. Ahora, como primer comentario de los resultados obtenidos se tiene que comentar que si bien Batch se encargó de lanzar máquinas que se adecuaban a la carga de trabajo, las máquinas seleccionadas no fueron las mejores, una combinación r4.16xlarge con m4.4xlarge como la que se presentó en el experimento pasado seguramente hubiese sido capaz de presentar mejores tiempos de ejecución para este experimento como se puede apreciar al comparar la tabla de resultados del experimento tres con la del experimento cuatro presente a continuación.

Min	100
Max	118
Promedio	111,8333333
Ejecucion total:	386 Segundos, 6.43 Minutos
Tiempo total:	640 Segundos, 10,666 Minutos
Levantamiento 1	254 Segundos, 4.23 Minutos
Descenso	5 Minutos

Tabla 6.4.1. Resultados experimento cuatro.

Se puede ver como el tiempo total de ejecución del experimento tres fue de 610 segundos y aquí fue de 640 segundos. En principio 30 segundos adicionales no parece mucho, pero recordemos que el experimento cuatro tenía veinte trabajos de la revisión 8, que únicamente consumía 10000MiB de RAM, mientras que todos los 30 trabajos del experimento tres fueron de la revisión 5. Por lo cual mientras el experimento tres tuvo una carga de trabajo constante de 10 contenedores concurrentes, el experimento cuatro llegó inclusive hasta manejar 18 contenedores al mismo tiempo, lo cual lo debería de hacer, en teoría, más veloz, cosa que no fue el caso.

Ahora procedamos a hablar de costos, en este caso al hablar de unas instancias m4.16xlarge y m4.4xlarge que tienen unos costos de 3.20 USD y 0.80 USD por hora, y por tanto unos costos de 0.00088 USD y 0.00022 USD por segundo que se utilizaron durante todo el tiempo de ejecución, los costos totales del experimento serían.

Costos Experimento 4: $640 * (0.00088 + 0.00022) = 0.704$.

Por tanto, observamos como los costos de este son un 10% inferiores a los del experimento tres, pero siguen siendo superiores a los del experimento uno.

Conclusiones

Este escenario nos hace darnos cuenta que, si bien los entornos óptimos lanzan instancias que se adecuan a la carga de trabajo, cumpliendo con restricciones de memoria y de CPU, estos no necesariamente lanzan siempre la mejor configuración posible para una carga de trabajo. Por tanto, aunque en el anterior punto se mencionó que los entornos óptimos pueden llegar a ser una solución más eficiente a nivel de desempeño que una previamente dimensionada por el grupo de técnicos encargados, esta presenta una variabilidad e incertidumbre de importancia que deben ser tenidas en cuenta para determinar si vale la pena hacer el esfuerzo de sacar adelante una configuración apropiada para el caso de uso, que siempre trate de utilizar las mejores máquinas para cada carga de trabajo.

Dos colas - Un Entorno de Cómputo

Descripción

Dentro de este experimento pretendemos evaluar la funcionalidad y el uso de las colas proporcionadas por Batch. Este pretende ser un experimento sencillo en el cual queremos corroborar que los trabajos asociados a una cola de mayor prioridad sean procesados primero que aquellos asociados a una cola de baja prioridad. Lo anterior es simplemente para habilitar la posibilidad a distintos casos de uso que pudiera tener el IDEAM, como, por ejemplo, puede que un análisis periódico de deforestación sea menos crítico que uno de cambios en el suelo cuando existe riesgo de inundación.

Entorno

- Entornos de computo #1:
- Colas:
 - Prioridad: 100
 - Entornos asociados: Entorno #1
 - Prioridad: 200
 - Entornos asociados: Entorno #1
- Definiciones de trabajo Revisión 5

Resultados

Como se ha mencionado anteriormente tanto en la descripción del experimento como en la configuración del mismo, se crearon dos colas que se encuentran asociadas al mismo entorno de computo como se puede observar en las figuras 6.5.1. y 6.5.2.

A job queue accepts job submissions and places them on an appropriate compute environment.

Queue name

Queue ARN

Priority

Job queues with a higher integer value for priority are given preference for compute resources.

Enable Job queue ☒

Connected compute environments for this queue

Jobs are submitted to the connected compute environments based on the order they are listed and the available capacity of those environments.

Order	Compute environment
<div>^ v</div> <div>1</div>	<div>Compute environment name: r4d4xlarge_8_64</div> <div>Status: VALID</div> <div>Type: MANAGED</div> <div>Provisioning model: EC2</div> <div>Minimum vCPUs: 0</div> <div>Desired vCPUs: 0</div> <div>Maximum vCPUs: 80</div> <div>Instance types: r5.4xlarge</div>

Figura 6.5.1. Cola número uno.

Create a job queue

A job queue accepts job submissions and places them on an appropriate compute environment.

Queue name cola_experimento_5_2

Queue ARN arn:aws:batch:us-east-1:186121879828:job-queue:cola_experimento_5_2

Priority 200

Job queues with a higher integer value for priority are given preference for compute resources.

Enable Job queue ☒



Connected compute environments for this queue

Jobs are submitted to the connected compute environments based on the order they are listed and the available capacity of those environments.

Order	Compute environment
<div><div>^ v</div><div>1</div></div>	<div>Compute environment name: r4d4xlarge_8_64</div> <div>Status: VALID</div> <div>Type: MANAGED</div> <div>Provisioning model: EC2</div> <div>Minimum vCPUs: 0</div> <div>Desired vCPUs: 0</div> <div>Maximum vCPUs: 80</div> <div>Instance types: r5.4xlarge</div>

Cancel

Create Job Queue

Figura 6.5.2. Cola número dos

Ahora, a continuación, encontramos tabulados los resultados obtenidos en la tabla 6.5.1.

Cola	Trabajo	Hora Creación	Hora Inicio
1	1	7:19:31 p. m.	7:32:58 p. m.
1	2	07:19:40 pm	7:32:58 p. m.
1	3	07:19:46 pm	7:34:42 p. m.
1	4	07:19:53 pm	07:35:12 pm
1	5	07:20:02 pm	07:34:49 pm
1	6	7:20:08 p. m.	07:32:54 pm
1	7	07:20:14 pm	07:36:20 pm
1	8	07:20:22 pm	7:34:50 p. M
1	9	07:20:41 pm	07:32:54 pm
1	10	07:20:48 pm	7:36:20 p. m.
2	1	7:21:04 p. m.	7:34:49 p. m.
2	2	7:21:12 p. m.	7:34:49 p. m.
2	3	7:21:18 p. m.	07:32:58 pm
2	4	7:21:23 p. m.	07:34:49 pm

2	5	7:21:28 p. m.	07:34:41 pm
2	6	7:21:36 p. m.	07:33:04 pm
2	7	7:21:42 p. m.	07:33:04 pm
2	8	7:21:48 p. m.	07:33:00 pm
2	9	7:21:54 p. m.	7:33:00 p. m.
2	10	7:22:04 p. m.	07:34:42 pm

Tabla 6.5.1. Resultados experimento 5

En esta tabla podemos observar como al inicio se presenta un comportamiento anormal para los trabajos uno, dos y seis de la primera cola pues presentan tiempos de inicio inferiores a cualquier trabajo presente en la cola número dos. No obstante, luego podemos observar una tendencia dentro de la cual todos los trabajos de la cola número dos empiezan a ser ejecutados con tiempos inferiores a los de la cola número uno. Describiendo de esta forma el comportamiento esperado y que nos garantiza el servicio de Batch, los trabajos procesados siempre serán aquellos de la cola de mayor prioridad indiferentemente de que estos hayan sido enviados anteriormente. Sin embargo, existe la posibilidad de que trabajos con mayor antigüedad y menor prioridad sean procesados, al menos algunos de ellos, primero como podemos observar en este experimento.

Adicionalmente, en este caso se prendieron cinco máquinas (figura 6.5.3.), lo cual no sucedió en experimentos anteriores. Los motivos de este comportamiento se debieron a que únicamente se encontraban activas las máquinas utilizadas para el experimento y ninguna máquina asociada a los experimentos realizados por el equipo de investigación del IDEAM. Lo anterior causó que límite de máquinas aumentara de 4 a 14 como se observa en la figura 6.5.4.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs	Key Name
	i-04da0086590766...	r5.4xlarge	us-east-1a	running	Initializing	None	ec2-35-168-32-236.co...	35.168.32.236	-	cubo-tesis
	i-083ef3da64649cf94	r5.4xlarge	us-east-1a	running	Initializing	None	ec2-52-3-228-57.comp...	52.3.228.57	-	cubo-tesis
	i-0b5aaf711125a6bb	r5.4xlarge	us-east-1a	pending	Initializing	None	ec2-52-3-232-190.com...	52.3.232.190	-	cubo-tesis
	i-0d2a7eab41e2d5dc9	r5.4xlarge	us-east-1a	running	Initializing	None	ec2-35-170-52-223.co...	35.170.52.223	-	cubo-tesis
	i-0e4c8fd803cb3adc	r5.4xlarge	us-east-1a	running	Initializing	None	ec2-34-205-93-204.co...	34.205.93.204	-	cubo-tesis

Figura 6.5.3. Instancias inicializándose.

Running On-Demand r5.4xlarge instances

14

Figura 6.5.4. Límites actuales de la cuenta.

Conclusiones

Como se mencionó, este experimento era más que todo para corroborar una de las funcionalidades del servicio, en este caso, el uso de colas con distintas prioridades. Se pudo observar como las colas efectivamente son procesadas de acuerdo a su prioridad, sin embargo, esto no es una ley absoluta, por tanto, para requerimientos donde el cumplimiento de las prioridades sea de alto impacto esto definitivamente deberá ser un factor a tener en cuenta pues no en todos los escenarios estas prioridades son respetadas.

Una cola - Múltiples entornos

Descripción

Dentro de este experimento pretendemos observar el comportamiento presente por el servicio de Batch cuando los trabajos enviados son asignados a una cola con distintos entornos informáticos asociados. Para esto se enviará una carga de 30 trabajos concurrentes dentro de dos entornos con tipos de máquinas distintas con el fin de diferencias qué entorno fue el encargado de lanzar el poder de cómputo para satisfacer la carga de trabajo. Particularmente queremos solucionar a las preguntas: ¿Qué pasa cuando un entorno se queda sin capacidad de cómputo? Y ¿Qué pasa si un entorno falla?

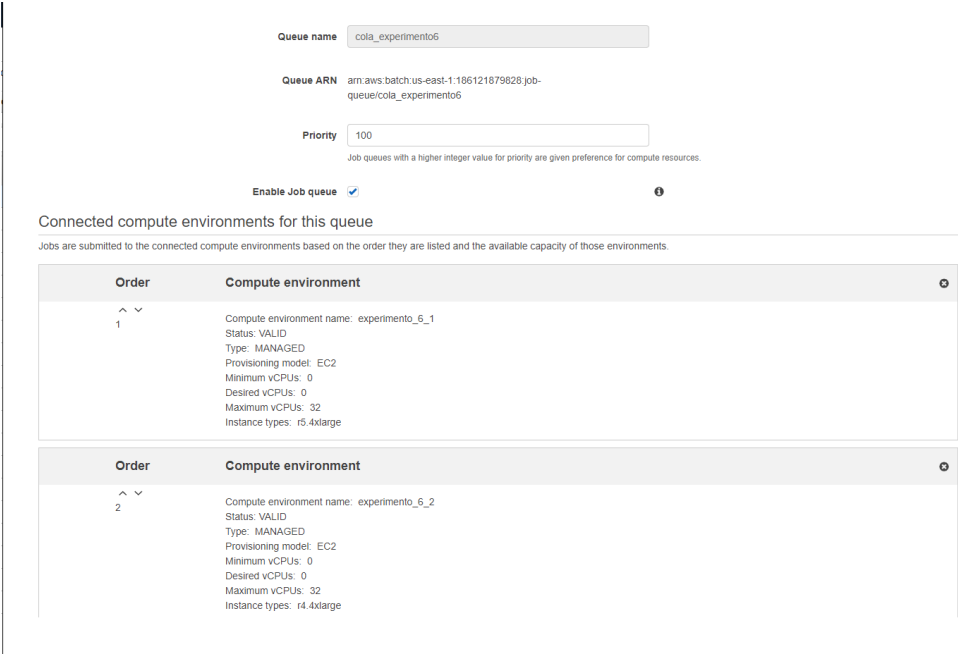
Entorno

-
- Entornos de computo (#3):
 - Tipo de instancias: r4.xlarge
 - Modelo de pago: On-Demand
 - CPUs virtuales: Min 0, Deseados 16, Max 32

Entorno de computo (#4):

- Tipo de instancias: r5.xlarge
- Modelo de pago: On-Demand
- CPUs virtuales: Min 0, Deseados 16, Max 32
- Colas:
 - Prioridad: 100
 - Entornos asociados: Entorno #1 y 3
- Definiciones de trabajo (Revisión 5)

Dicha configuración puede observarse a continuación.



Resultados

El comportamiento observado en este experimento fue una mezcla entre resultados esperados y comportamientos inusuales. En primera instancia el servicio de Batch lanzó dos máquinas R5.4Xlarge como se puede observar en la figura 6.6.1.

<input type="checkbox"/>	i-0d9f4e8d82a88cc8f	r5.4xlarge	us-east-1a	<div><div></div></div> running	<div><div></div></div> Initializing	None	<div><div></div></div> ec2-34-200-224-195.co...	34.200.224.195	-	cubo-tesis	enabled	Noven
<input type="checkbox"/>	i-0e4cd8ffdb03cb3adc	r5.4xlarge	us-east-1a	<div><div></div></div> terminated		None	<div><div></div></div>	-	-	cubo-tesis	enabled	Noven
<input type="checkbox"/>	i-0e8ff194813fc0791	r5.4xlarge	us-east-1a	<div><div></div></div> running	<div><div></div></div> Initializing	None	<div><div></div></div> ec2-18-214-15-182.co...	18.214.15.182	-	cubo-tesis	enabled	Noven

Figura 6.6.1. Lanzamiento instancias R5.4Xlarge.

Esta fue capaz de procesar alrededor de unos quince trabajos de a lotes de cuatro. En principio se pensó que Batch no lanzaría nuevas instancias y que con las dos actuales pasaría a solucionar toda la carga de trabajo existente. Esto se puede apreciar con los trabajos que se encontraban encolados hasta el momento en la figura 6.6.2.

Cola		Estado					
<div>cola_experimento6</div>		<div>submitted</div>	<div>pending</div>	<div>runnable</div>	<div>starting</div>	<div>running</div>	<div>suceeded</div>
ID de trabajo		Nombre de trabajo		Tamaño de la matriz		Estado	
<div></div>	e0cbd71e-fd06-4faa-97d1-0a16cd5a0628	J14		--		RUNNABLE	
<div></div>	daa9f7cf-77a5-4ef5-be34-c26c30a5e8f9	J16		--		RUNNABLE	
<div></div>	4a142a41-1185-4ce5-aecc-61fcefa6d130	J21		--		RUNNABLE	
<div></div>	ac24c54b-e4e7-48c4-a6ab-0ab3b1a57fdf	J22		--		RUNNABLE	
<div></div>	8d26d1d8-2057-47c1-a2ba-51e9883ba74f	J23		--		RUNNABLE	
<div></div>	a65ca362-8e69-4a1e-aafc-acce29136d87	J24		--		RUNNABLE	
<div></div>	2339fe68-8d37-4942-8697-fad08fd6d5a7	J25		--		RUNNABLE	
<div></div>	c277b664-d743-462b-bbeb-1b7d96bfb4f1	J26		--		RUNNABLE	
<div></div>	3f8a8944-4d12-434f-99e1-bec450d8cb3c	J27		--		RUNNABLE	
<div></div>	e27710ad-156f-4ad9-a053-ea2a22f7b5e7	J27		--		RUNNABLE	
<div></div>	783af508-f8f3-483c-a599-718905ab34a8	J29		--		RUNNABLE	
<div></div>	aebe852f-a5b0-44dc-8c8e-9cfd5b6ef981	J30		--		RUNNABLE	

Figura 6.6.2. Trabajos encolados.

Sin embargo, finalmente se lanzaron dos nuevas instancias R4.4Xlarge correspondientes al segundo entorno de cómputo asociado como se puede observar en la figura 6.6.3.

	i-07bcaae7407918a75	r5.4xlarge	us-east-1a		running	2/2 checks ...	None		ec2-35-153-198-249 co...	35.153.198.249	-	-	cubo-tesis		enabled	Novem...
	i-0a89de226454e5...	r5.4xlarge	us-east-1a		terminated		None			-	-	-	cubo-tesis		enabled	Novem...
	i-0b0fc79d9122ea210	r5.4xlarge	us-east-1a		running	Initializing	None		ec2-18-209-157-16 co...	18.209.157.16	-	-	cubo-tesis		enabled	Novem...
	i-0b96924577c6e833e	r4.4xlarge	us-east-1a		terminated		None			-	-	-	cubo-tesis		enabled	Novem...
	i-0d586e5c4b13e0b41	r5.4xlarge	us-east-1a		terminated		None			-	-	-	cubo-tesis		enabled	Novem...
	i-0d9f4a8d2a88cc8f	r5.4xlarge	us-east-1a		terminated		None			-	-	-	cubo-tesis		enabled	Novem...
	i-0e0f194813fc0791	r5.4xlarge	us-east-1a		terminated		None			-	-	-	cubo-tesis		enabled	Novem...
	i-0f5b09b2cab4b9c03	r4.4xlarge	us-east-1a		running	2/2 checks ...	None		ec2-18-210-6-26 comp...	18.210.6.26	-	-	cubo-tesis		enabled	Novem...

Figura 6.6.3. Lanzamiento de instancias R4.

Esto confirmó que cuando un entorno de cómputo se agota en cuanto a recursos computacionales, pero de igual forma existe una carga de trabajo que requiere de aún más capacidad para ser procesada de forma adecuada, Batch pasará al siguiente entorno asociado con el fin de satisfacer la demanda existente. Lo anterior sirvió como insumo para responder a la primera pregunta objetivo de este experimento.

Ahora, con respecto a la segunda, con el fin de responderla se simuló el falló de uno de los entornos, particularmente el primero, deshabilitando uno de los entornos de cómputo asociado y posteriormente terminando de forma manual las máquinas que fueron lanzadas por este. Esto se puede apreciar en la figura 6.6.4.

Connected compute environments for this queue

Jobs are submitted to the connected compute environments based on the order they are listed and the available capacity of those environments.

Order	Compute environment
1	Compute environment name: experimento_6_1 Status: VALID This compute environment is DISABLED. Type: MANAGED Provisioning model: EC2 Minimum vCPUs: 0 Desired vCPUs: 32 Maximum vCPUs: 32 Instance types: r5.4xlarge
2	Compute environment name: experimento_6_2 Status: VALID Type: MANAGED Provisioning model: EC2 Minimum vCPUs: 0 Desired vCPUs: 0 Maximum vCPUs: 32 Instance types: r4.4xlarge

Figura 6.6.4. Entorno de Cómputo número uno deshabilitado.

Sin embargo, a pesar de que el entorno se encontraba deshabilitado cada vez que una de las máquinas era terminada, una nueva instancia era lanzada como reemplazo de forma inmediata. Esto puede verse de forma positiva en la medida de que Batch garantiza siempre la existencia de instancias para satisfacer los trabajos encolados, teniendo en cuenta por supuesto la cantidad de reintentos permitidos por cada trabajo, pues cada vez que se terminaba una máquina los trabajos en procesamiento por esta eran reenviados y por tanto su contador de reintentos aumentaba. Cabe recalcar que, a pesar de este comportamiento, que resultó ser bastante inusual pues un usuario esperaría que al deshabilitar un entorno todas las máquinas lanzadas por este deberían terminar de forma automática. Lo anterior si bien no aporta de forma significativa para el cuestionamiento que planteamos en este experimento, si nos permite concluir una recomendación a realizar para el IDEAM o cualquier usuario del servicio, con el fin de evitar costos inesperados por uso de instancias no basta con deshabilitar los entornos de cómputo, estos deben ser configurados con un uso mínimo de vcpu de cero y si no van a ser utilizados más, deben ser desasociados de cualquier cola de trabajos que esté recibiendo carga de trabajo en un momento dado.

Conclusiones

A partir de este experimento podemos decir que la manera en que múltiples entornos asociados a una cola se comportan es la de lanzar tanto poder computacional como les es permitido y en caso de ser necesario poder computacional adicional, se empezara a conseguir a partir del siguiente entorno en la lista de forma iterativa hasta finalizar con la lista de todos los entornos asociados.

Entorno de Cómputo Spot con tipo de instancia especificado

Descripción

Una de las principales promesas del uso de AWS Batch es su integración con las instancias Spot o instancias de subasta. Esto conviene de forma sustancial ya que la gran mayoría de los casos de uso para procesamientos por lotes conllevan el uso de un enorme poder computacional que por supuesto desemboca en costos altamente elevados para las organizaciones. El propósito de este experimento es confirmar qué tanto porcentaje de costos es posible ahorrar con el uso de entornos informáticos bajo el modelo de pago spot.

Entorno

Para la realización de este experimento, el cual en su mayoría es de gran ensayo y error se realizaron seis colas y seis entornos informáticos distintos. Estos entornos pueden encontrarse en la figura 6.7.1.

Nombre	Tipo	Modelo de aprovisionamiento	Tipos de instancia	Estado	Estado	Min. de CPU virtuales	CPU virtuales deseadas	Máx. de CPU virtuales
spot_40_r5	MANAGED	SPOT	r5.4xlarge	VALID	ENABLED	0	0	80
spot_30_r5	MANAGED	SPOT	r5.4xlarge	VALID	ENABLED	0	0	80
spot_20_r5	MANAGED	SPOT	r5.4xlarge	VALID	ENABLED	0	0	80
cola_spot27_r5	MANAGED	SPOT	r5.4xlarge	VALID	ENABLED	0	0	80
spot_60_r5	MANAGED	SPOT	r5.4xlarge	VALID	ENABLED	0	0	80
spot_r5	MANAGED	SPOT	r5.4xlarge	VALID	ENABLED	0	0	80

Figura 6.7.1. Entornos utilizados.

Todos y cada uno de estos entornos comparten las siguientes características:

- Entornos de computo (#3):
 - Tipo de instancias: r5.4xlarge
 - Modelo de pago: Spot
 - CPUs virtuales: Min 0, Deseados 16, Max 80

Lo único que las diferencia es el porcentaje del valor on-demand de la máquina, en este caso r5.4xlarge, que se colocó como el máximo monto a pagar. En nuestro caso se colocó un valor máximo de 80, 60, 40, 30, 27 y 20% del valor original de la instancia como precio a pagar por cada una de las instancias lanzadas.

Resultados

Los resultados obtenidos para cuatro de los seis entornos evaluados fueron un lanzamiento sin ningún problema de las instancias necesarias para satisfacer la carga de trabajo existente, como se puede apreciar en la figura 6.7.2.

<input type="checkbox"/>	i-0115531ac3d89562a	r5.4xlarge	us-east-1a	running	initializing	None	ec2-18-207-106-106.co...	18.207.106.105	-	cubo-testis	disabled	November 12, 2018 at 10:58:1...	default
<input type="checkbox"/>	i-041f523e7c0005ba0	r5.4xlarge	us-east-1a	running	initializing	None	ec3-34-231-229-33.co...	34.231.229.33	-	cubo-testis	disabled	November 12, 2018 at 10:58:1...	default
<input type="checkbox"/>	i-069f120e9c3b26f	r5.4xlarge	us-east-1a	running	initializing	None	ec2-18-207-106-108.co...	18.207.106.108	-	cubo-testis	disabled	November 12, 2018 at 10:58:1...	default
<input type="checkbox"/>	i-06c37a88dad39a8e	r5.4xlarge	us-east-1a	running	initializing	None	ec2-52-3-231-173.com...	52.3.231.173	-	cubo-testis	disabled	November 12, 2018 at 10:58:1...	default
<input type="checkbox"/>	i-07d1068ea505756...	r5.4xlarge	us-east-1a	running	initializing	None	ec2-100-24-107-75.co...	100.24.107.75	-	cubo-testis	disabled	November 12, 2018 at 10:58:1...	default

Figura 6.7.2. Instancias Spot Lanzadas.

Sin embargo, para el caso de los dos últimos entornos nos encontramos que ninguna instancia fue lanzada, de hecho, a la fecha de hoy ninguno de los trabajos encolados ha logrado ser procesado pues ninguna instancia ha sido lanzada debido a que no han alcanzado el precio que mencionamos estábamos dispuestos a pagar como se observa en la figura 6.7.3.

<input type="radio"/>	65b12927-1b2d-42e8-b797-8075d9425131	Job1	--	RUNNABLE	07:51:06 pm 11/08/18
<input type="radio"/>	c9717d04-208d-425f-ab5a-7955ad2245bd	Job2	--	RUNNABLE	07:51:12 pm 11/08/18
<input type="radio"/>	efb69fd1-09d3-4d0c-8d12-570cb0f2ca4e	Job3	--	RUNNABLE	07:51:17 pm 11/08/18
<input type="radio"/>	81d3e8b7-eeef3-40e1-bc85-7d02013e60d8	Job4	--	RUNNABLE	07:51:22 pm 11/08/18
<input type="radio"/>	36982896-9823-42c6-94e2-99c3bc11e2fa	Job5	--	RUNNABLE	07:51:27 pm 11/08/18
<input type="radio"/>	59050042-6f45-46b1-b9cc-ed808ba3631a	Job6	--	RUNNABLE	07:51:32 pm 11/08/18
<input type="radio"/>	3a8513fa-aaaa-4156-bacc-b4f4c3ce5eb6	Job7	--	RUNNABLE	07:51:36 pm 11/08/18
<input type="radio"/>	4b3105e7-5c0b-4668-aba2-c5471693c9ce	Job8	--	RUNNABLE	07:51:41 pm 11/08/18
<input type="radio"/>	1fa19597-8a4c-4757-aaa7-425bd1a82d1c	Job9	--	RUNNABLE	07:51:46 pm 11/08/18
<input type="radio"/>	30d4d7b6-b7e9-4e7c-8a7e-20553d0f7eec	Job10	--	RUNNABLE	07:51:52 pm 11/08/18

Figura 6.7.3. Trabajos esperando ser ejecutados.

A partir de estos datos obtenidos se pudo estimar que para valores de pago inferiores al 30% del valor original de una instancia R5.4xlarge, las instancias no son lanzadas. En primera instancia se quiso investigar si había algún tipo de datos abiertos de los precios spot reales de cada una de las instancias EC2 de AWS con el fin de generar un modelo predictivo que permitiera aterrizar los valores de las instancias a través del tiempo. Lo anterior con el fin de poder recomendar a los usuarios qué porcentaje configurar en sus entornos de computo evitando que estas instancias sean apagadas por Amazon, pues recordemos que en el momento que el precio real de una instancia supera el establecido en la configuración del entorno, estas serán apagadas automáticamente por AWS. Sin embargo, dicha información no pudo ser encontrada.

No obstante, AWS provee un asistente para configuración para instancias spot [21] en el cual podemos observar diversos datos relacionados a las instancias, tales como sus principales características a nivel de memoria y cpu, el máximo grado de ahorro que existe actualmente asociado a esa máquina y la probabilidad de que esa máquina sea apagada automáticamente por AWS. Esto se puede observar de forma clara en la figura 6.7.4.

Asistente de instancias spot

Región: EE.UU. Este (Norte de Virginia) SO: Linux/UNIX

Filtro de tipo de instancia:

vCPU (mín.): 1 Memory GiB (mín.): 0 ☐ Tipos de instancia admitidas por EMR

Tipo de instancia	CPU virtual	Memoria GiB	Ahorro en comparación con modalidad bajo demanda*	Frecuencia de interrupción
m3.medium	1	3.75	90%	<5% □□□□□
m2.2xlarge	4	34.2	90%	>20% ■ ■ ■ ■ □
m2.xlarge	2	17.1	90%	<5% □□□□□

Figura 6.7.4. Asistente de instancias Spot.

De hecho, fue gracias a la ayuda de este asistente que se implementó el entorno con 27% de pago del precio original, pues inicialmente se tenía una diferencia entre 30% y 20%, pero gracias al asistente se puso aterrizar un poco más a partir de qué porcentaje (27%) el levantamiento de instancias comenzaba a fallar. Lo sorprendente de este caso es que el asistente de instancias acertó con bastante precisión a los comportamientos observados por los experimentos, tal y como se puede apreciar en la figura 6.7.5.

r5.4xlarge	16	128	72%	>20% ■ ■ ■ ■ □
------------	----	-----	-----	-------------------

Figura 6.7.5. Características de la instancia r5.4xlarge.

Lo anterior nos permite decir que esta herramienta ofrecida por AWS es de gran utilidad y debe ser utilizada si se planea implementar una solución en la cual se vean involucradas instancias spot. Adicionalmente se menciona que los precios y el ahorro asociado a ellos varía de región en región dependiendo de la demanda, por lo cual los comportamientos de cada tipo de instancia son variantes dependiendo de la zona en la cual sean desplegadas.

Finalmente se quiere mencionar que podemos observar grandes ahorros, que en términos de algunos de los experimentos que hemos realizado, por ejemplo, el experimento uno en el cual utilizamos instancias previamente dimensionadas, descripción perfecta para un caso de uso de spot, tuvimos unos gastos totales de 0.48328 USD que podrían llegar a ser reducibles hasta unos 0,144984 USD.

Conclusiones

A partir de los hallazgos encontrados en este experimento, se puede concluir que el asistente de instancias es una herramienta fiable e indispensable para poder trabajar de forma óptima e inteligente con instancias spot con el fin de evitar resultados inesperados y que el uso de la

funcionalidad de instancias spot de AWS Batch nos puede ofrecer ahorros enormes de hasta un 70% de los gastos originales asociados a nuestros procesamientos.

Entorno de Cómputo Spot con tipo de máquina óptima.

Descripción

En el anterior experimento pudimos observar como a partir del uso adecuado de una instancia spot de una máquina previamente dimensionada se pueden llegar a presentar ahorros de hasta el 70%. Sin embargo, existen instancias que, sin ser necesariamente las más óptimas para el trabajo, presentan tasas de ahorro de hasta el 90% como se presenta en la figura 6.8.1.

m2.2xlarge	4	34.2	90%	>20% ██████
m2.xlarge	2	17.1	90%	<5% □□□□
m1.medium	1	3.75	90%	<5% □□□□
m1.xlarge	4	15	90%	<5% □□□□
m2.4xlarge	8	68.4	90%	15-20% ██████

Figura 6.8.1. Instancias con ahorro del 90%.

La interrogante que este experimento busca solucionar es. ¿Es posible resolver la carga de trabajo existente dejándole la elección del tipo de instancia a AWS a la vez que se incurre en ahorros superiores a los existentes para una máquina previamente dimensionada? Para lograr responder lo anterior, se enviará la carga habitual, es decir, 30 trabajos, a una cola asociada a un entorno spot de elección de máquina óptima.

Entorno

- Entornos de computo (#11):
 - Tipo de instancias: Optimal
 - Modelo de pago: Spot
 - CPUs virtuales: Min 0, Deseados 16, Max 80
 - Bid: 27% del costo original
- Colas:
 - Prioridad: 100
 - Entornos asociados: Entorno #11
- Definiciones de trabajo
 - CPUs : 4
 - Memoria: 61100 MiB.

Resultados

En realidad, los resultados de este experimento fueron un poco decepcionantes. Los trabajos enviados a la cola, al igual que como ocurrió al final del experimento 7, permanecieron encolados por un periodo de observación de unos tres días.

Con el fin de intentar generar un ambiente en el cual instancias pudieran ser lanzadas se hizo una modificación a la cola, se le añadieron más entornos de computo con la esperanza de que al menos uno de ellos lanzara instancias. Esto se puede observar en la figura 6.8.2.

Jobs are submitted to the connected compute environments based on the order they are listed and the available capacity of those environments.

Order	Compute environment
1	Compute environment name: spot_optimal_27 Status: VALID Type: MANAGED Provisioning model: SPOT Minimum vCPUs: 0 Desired vCPUs: 80 Maximum vCPUs: 80 Instance types: optimal
2	Compute environment name: spot_30_r5 Status: VALID Type: MANAGED Provisioning model: SPOT Minimum vCPUs: 0 Desired vCPUs: 0 Maximum vCPUs: 80 Instance types: r5.4xlarge
3	Compute environment name: spot_60_r5 Status: VALID Type: MANAGED Provisioning model: SPOT Minimum vCPUs: 0 Desired vCPUs: 0 Maximum vCPUs: 80 Instance types: r5.4xlarge
4	Compute environment name: r4d4large_0_64 Status: VALID Type: MANAGED Provisioning model: EC2 Minimum vCPUs: 0 Desired vCPUs: 0 Maximum vCPUs: 80 Instance types: r5.4xlarge

Figura 8.2. Modificación a la cola del experimento 8.

Sin embargo, los cambios realizados fueron insuficientes pues se obtuvieron los mismos resultados. Finalmente, lo que se hizo fue hacer un cambio en el orden de los entornos asociados, dejando de primero un modelo spot un porcentaje de bid más alto que garantizara el lanzamiento de instancias, bajo esta configuración sí se presentó un lanzamiento de instancias. Lo anterior nos permite decir que no es posible una configuración de entornos spot con bid bajo como entornos primarios y de entornos spot con bid superior o entornos on-demand como entornos secundarios, la razón de lo anterior es, como se mencionó en el experimento 6, que los entornos deben llegar a su capacidad máxima antes de pasar a utilizar el siguiente entorno en la lista. Sin embargo, una configuración al revés sí funciona, es decir, empezar por un bid alto e ir disminuyendo.

Conclusiones

Teniendo en cuenta los resultados podemos afirmar que las configuraciones ascendentes bajo el modelo spot no son deseables en AWS Batch, o cuando menos se debe comenzar con un entorno informático que posea un porcentaje de bid que garantice la ejecución correcta del programa. Por el contrario, las configuraciones descendentes sí funcionan y de hecho pueden llegar a ser interesantes en la medida que un entorno de bid máximo, y por tanto el primero en la lista de entornos asociados a una cola, esté configurado para tener el intervalo de máquinas mínimas deseadas para el procesamiento de una carga de trabajo, y a partir de allí, nuevas máquinas solo sean añadidas en la medida de que sean tan económicas que el desempeño que agreguen al procesamiento de dicha carga de trabajo sea justificable dado su bajo costo.

Conclusiones Generales

- AWS Batch es capaz de manejar la escalabilidad de las máquinas de forma autónoma sin la necesidad de intervención humana y satisfactoriamente de acuerdo a la demanda de trabajos. Es necesario recordar que la unidad de medida para el escalamiento está dada por unidades de CPU's Virtuales (vCPU) y no por unidades de máquinas.
- Una configuración apropiada de Batch dada a partir de un correcto dimensionamiento tanto del tipo de instancia a usar como de la cantidad de recursos a utilizar obtendrá mejores resultados en cuanto a rendimiento y a costos que una configuración “óptima” seleccionada por Batch a partir de la carga de trabajo.
- El uso de colas de distintas prioridades que pueden ser asociadas a distintos entornos de cómputo y que poseen distintas definiciones de trabajo ofrece una cantidad enorme de posibilidades para que el cliente configure su entorno de ejecución según sus necesidades.
- Sin embargo, en términos generales se sugiere que se usen colas de distintas prioridades y que cada una de estas colas esté asociada a varios entornos compartidos entre sí. De esta forma si una cola encendió máquinas con anterioridad los trabajos de otras colas podrán aprovechar estas máquinas preexistentes y no se incurrirá en costos adicionales de latencia por razón de espera por recursos disponibles ni de costos por máquinas adicionales.
- El uso de instancias spot siempre será recomendado, representando ahorros de hasta el 70%. Sin embargo, se debe tener en cuenta que estas máquinas pueden ser interrumpidas en cualquier momento o inclusive puede que nunca sean lanzadas si el porcentaje de pago es muy bajo.
- Por el anterior motivo el sistema debe ser diseñado para fallar con gracia o disponer de una flota de máquinas capaz de satisfacer una porción constante de la carga de trabajo, como se mencionó, se sugiere una configuración descendente en cuanto el pago. Se configura como opción uno la máquina más costosa y se van agregando máquinas más económicas poco a poco.

- Finalmente, AWS Batch es un servicio bastante completo que ofrece una serie de características bastante complejas que permiten lograr sistemas de procesamiento en Batch bastante robustos (que solucionan todos los problemas de procesamientos en batch planteados en la introducción) en cuestión de, como muchas horas, sin incurrir en los gastos de mantenimiento e implementación que esto representaría para el área de tecnología de una organización. En pocas palabras, gracias a Batch el negocio puede concentrar sus esfuerzos en operar su negocio, no en mantenerlo.

Trabajo futuro

Dentro del alcance de nuestro proyecto se lograron cumplir todos los objetivos propuestos, es decir, logramos realizar la validación del diseño propuesto no solo a nivel conceptual, sino que también se logró una implementación real que puede ser utilizada por el IDEAM en sus entornos productivos dentro de su organización. Adicionalmente se plantearon la mayoría de las configuraciones posibles para dicha implementación dentro del servicio de Batch.

Teniendo lo anterior en cuenta, el trabajo futuro a realizar para quienes decidan utilizar la información contenida en este documento resulta de dos componentes sencillos, aquellas funcionalidades adicionales que pueden resultar de interés pero no fueron evaluadas dentro del proyecto debido a que las características de nuestro caso de estudio no presentaban una justificación sólida para evaluarlas y finalmente los pasos que debe seguir el IDEAM para sacar el mayor provecho de las conclusiones obtenidas de este proyecto y de la aplicación resultante.

En primera instancia se debe de contemplar las opciones para trabajos por grillas (arrays) ofrecida por Batch, lo anterior resulta de particular utilidad cuando queremos asignar una variada cantidad de subtareas a una tarea grande que es aquella de la cual nos interesa mantener un monitoreo constante.

También puede resultar interesante el uso de la sincronización entre distintos pipelines de Batch para producir un pipeline aún más grande. Es decir, el uso de batch para varias tareas en paralelo que deben esperar el resultado de las demás para poder dar solución a un proceso.

En cuanto a la implementación, el IDEAM debe definir como organización dos aspectos fundamentales. En primer lugar, se debe tener clara una estructura de costos y un modelo presupuestal en el cual se defina cuánto se está dispuesto a pagar por el poder de cómputo mínimo necesario para satisfacer sus necesidades actuales y cuánto sería el valor ideal a pagar por cada máquina adicional con el fin de incrementar el desempeño total del sistema. Lo anterior con el fin de sacar provecho a la configuración de instancias Spot de la cual se ha hecho mención anteriormente y que les permitiría obtener ahorros de hasta un 70%.

Finalmente, a medida que nuevos tipos de análisis vayan siendo agregados o inclusive para los existentes actualmente, se debe realizar su correcto dimensionamiento en cuanto a uso de memoria, CPU y disco y de ser posible hacer una elección precisa de la máquina apropiada, lo anterior con el fin de puedan obtener los resultados tanto a eficiencia computacional y de costos esperada, ya que si bien esta responsabilidad puede ser delegada a Batch, si es realizada con excelencia por el IDEAM, serán estos últimos quienes gocen de los beneficios de una solución óptima para su sistema.

Referencias

- [1] D. Ingram, *Design-build-run*. Indianapolis, IN: Wiley Pub., 2009.
- [2] "Batch processing", *En.wikipedia.org*. [Online]. Available: https://en.wikipedia.org/wiki/Batch_processing. [Accessed: 06- Dec- 2018].
- [3] "The lambda architecture principles", *Jameskinley.tumblr.com*. [Online]. Available: <http://jameskinley.tumblr.com/post/37398560534/the-lambda-architecture-principles-for>. [Accessed: 06- Dec- 2018].
- [4] Problems Associated With Batch Processing (Solaris Resource Manager 1.3 System Administration Guide)", *Docs.oracle.com*. [Online]. Available: <https://docs.oracle.com/cd/E19112-01/ctr.mgr11/816-7751/chapter10-4/index.html>. [Accessed: 06- Dec- 2018].
- [5] M. Villamizar, H. Castro, C. Ariza-Porras Types SCALING THE COLOMBIAN DATA CUBE USING A DISTRIBUTED ARCHITECTURE
- [6] C. Ariza-Porras, G. Bravo, M. Villamizar, A. Moreno, H. Castro, G. Galindo, E. Cabrera, S. Valbuena, and P. Lozano, CDCol: A Geoscience Data Cube that Meets Colombia
- [7] "ACERCA DE LA ENTIDAD - IDEAM", *Ideam.gov.co*. [Online]. Available: <http://www.ideam.gov.co/web/entidad/acerca-entidad>. [Accessed: 07- Dec- 2018].
- [8] "Pipe-And-Filter", *Dossier-andreas.net*. [Online]. Available: http://www.dossier-andreas.net/software_architecture/pipe_and_filter.html. [Accessed: 07- Dec- 2018].
- [9] "Map AWS services to Google Cloud Platform products | Google Cloud Platform Free Tier | Google Cloud", *Google Cloud*. [Online]. Available: <https://cloud.google.com/free/docs/map-aws-google-cloud-platform>. [Accessed: 07- Dec- 2018].
- [10] "Azure and AWS services compared - multicloud", *Docs.microsoft.com*. [Online]. Available: <https://docs.microsoft.com/en-us/azure/architecture/aws-professional/services>. [Accessed: 07- Dec- 2018].
- [11] "AWS Batch: Capacidades informáticas por lotes sencillas y eficaces - AWS", *Amazon Web Services, Inc.*. [Online]. Available: <https://aws.amazon.com/es/batch/>. [Accessed: 07- Dec- 2018].
- [12] "Batch Documentation - Tutorials, API Reference", *Docs.microsoft.com*. [Online]. Available: <https://docs.microsoft.com/en-us/azure/batch/>. [Accessed: 07- Dec- 2018].
- [13] "Language support of Batch Client Library – Python", *The Microsoft HPC & Batch Team Blog*. [Online]. Available: <https://blogs.technet.microsoft.com/windowshpc/2016/06/09/language-support-of-batch-client-library-python/>. [Accessed: 07- Dec- 2018].
- [14] "Stack Overflow Developer Survey 2018", *Stack Overflow* 2018. [Online]. Available: <https://insights.stackoverflow.com/survey/2018/#technology>. [Accessed: 07- Dec- 2018].
- [15] "8 surprising facts about real Docker adoption", *8 surprising facts about real Docker adoption*, 2017. [Online]. Available: <https://www.datadoghq.com/docker-adoption/>. [Accessed: 07- Dec- 2018].
- [16] "AWS EC2 Instance types", *Amazon Web Services, Inc.*. [Online]. Available: <https://aws.amazon.com/es/ec2/instance-types/> [Accessed: 07- Dec- 2018].

[17] "AWS Spot Instances. ", *Amazon Web Services, Inc.*. [Online]. Available: <https://aws.amazon.com/es/ec2/spot/> [Accessed: 07- Dec- 2018].

[18] "ContainerProperties - AWS Batch", *Docs.aws.amazon.com*, 2016. [Online]. Available: https://docs.aws.amazon.com/es_es/batch/latest/APIReference/API_ContainerProperties.html. [Accessed: 07- Dec- 2018].

[19] "AWS EC2 pricing" *Amazon Web Services, Inc.*. [Online]. Available: <https://aws.amazon.com/es/ec2/pricing/> [Accessed: 07- Dec- 2018].

[20] "AWS EC2 pricing On-Demand" *Amazon Web Services, Inc.*. [Online]. Available: <https://aws.amazon.com/es/ec2/pricing/on-demand/> [Accessed: 07- Dec- 2018].

[21] "AWS Instance Advisor. ", *Amazon Web Services, Inc.*. [Online]. Available: <https://aws.amazon.com/es/ec2/spot/instance-advisor/> [Accessed: 07- Dec- 2018].