**DIAGRAMA DE ARQUITECTURA Y DECISIONES DE DESARROLLO PROYECTO MUTANTES**

Para la solución del problema planteado se ha decidido la implementación de una arquitectura *event driven,* debido a las funcionalidades planteadas por magneto en la conceptualización de su problema. Se decidió implementar este tipo de arquitectura debido a que la interacción con el sistema por parte del usuario es una relación de múltiples componentes pero que al fin y al cabo resulta en una única respuesta.

Además de la implementación de dicha arquitectura se decide la implementación en la nube Amazon Web Services (AWS) debido a que suple la necesidad en cuanto a recursos que suplen el diagrama de arquitectura planteado.

**Diagrama 1**

***Arquitectura de la solución***

Diagrama

Descripción generada automáticamente

El anterior diagrama plasma la decisión del modelo de arquitectura *event-driven,* y lo hace bajo la premisa de la rápida y eficaz respuesta que debe recibir el usuario ante la petición realizada.

**Diagrama 2**

***Diagrama de componentes***

**Diagrama, Texto, Carta

Descripción generada automáticamente**

**Diagrama 3**

***Stack de tecnologías***

***Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente***

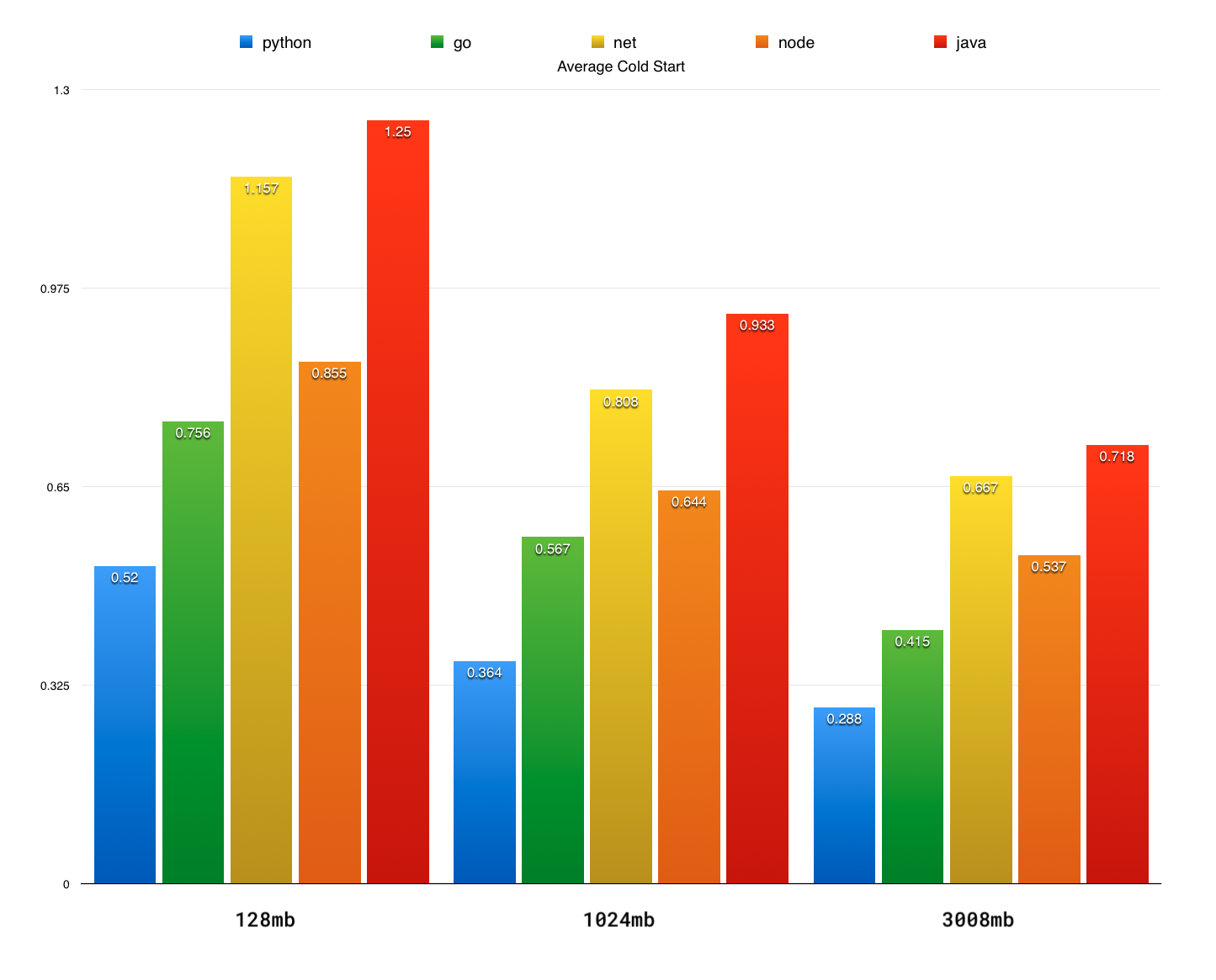
**Capas de la solución**

**Capa de aplicación**

En la capa de aplicación de usaron 3 lambdas utilizando las siguientes tecnologías:

* Nodejs – v 12.14.1
* ReactJs – v 6.6.3
* Typescript – v 4.0.3

Estas tecnologías se usaron teniendo en cuenta los tiempos de respuesta y de *cold start* en las lambdas utilizadas, ya que nodejs tiene uno de los mejores tiempos de respuesta en este tipo de artefactos, además de la experiencia del desarrollador según los tiempos estipulados para la entrega del proyecto.



**DESARROLLO DE LA SOLUCIÓN**

**Disposición en los artefactos**

**Interfaz de usuario gráfica, Texto

Descripción generada automáticamente**

Para el desarrollo de la solución se tuvo en cuenta un desarrollo por capas teniendo como punto de entrada el archivo index.ts y rutas de la forma planteada en la imagen superior.

**Contoller:** Clase fachada en la cual se tiene el llamado a la capa de servicio la cual contiene la lógica de la aplicación.

**Service:** Clase que contiene la lógica de la aplicación, esta clase utiliza una interfaz la cual actúa como contrato al momento de implementar los algoritmos necesarios.

**Utils:** Utilidades comunes necesarias para la trasformación de la data en tiempo de ejecución.

**Adapter:** Adaptadores o interfaces con otros procesos o sistemas, estos componentes aislados se utilizarán posteriormente en el service.

**Models:** Modelos de datos descritos para el correcto y ordenado mapeo de la información en el flujo.

**Resources:** Recursos necesarios para la correcta ejecución del código. Estos artefactos son archivos diferentes a los del código de la aplicación.

Además de la disposición descrita anteriormente se utilizó la librería inversify para la generación de un contexto de ejecución teniendo instancias singleton dispuestas en la ejecución del proceso.

**Patrones de diseño utilizados**

**Creacionales:**

* **Singleton:** Se usa mediante la utilización de la librería inversify, de esta manera se generan instacias que se almacenan en el contexto de ejecución y se utilizan mediante la inyección de dependencias en las clases necesarias.

**Estructurales:**

* **Adapter:** Utilizado para la interacción y comunicación de los objetos de infraestructura como DynamoDb o SQS
* **Facade:** Utilizado para el aislamiento de la capa de la lógica de negocio o service, este patrón se usa al implementar el controller.
* **Proxy:** Ademas de servir como fachada y aislamiento de la lógica, utilizamos el controller como proxy, para de esta manera controlar ciertas validaciones base en los parámetros de entrada.

**Comportamiento:**

Los patrones de comportamiento que se han utilizado definen la forma en como los objetos y las instancias de los mismos interactúan con la data, sin embargo quiero mencionar uno muy importante y es el patrón Observer.

* **Observer:** Este patrón se usa en el algoritmo de búsqueda de coincidencias de ADN para determinar si la cadena de texto corresponde a mutante o a humano.

**Cobertura de código de la solución**

Para garantizar la calidad del código y la solución desarrollada es necesario desarrollar pruebas unitarias de cada uno de los componentes y unificadamente de las funciones que se utilizaron, no solamente de los flujos satisfactorios sino de los posibles desbordes que se puedan presentar en tiempo de ejecución de la aplicación.

Para realizar dichas pruebas unitarias se utilizó la librería Jasmine, la cual nos provee de características optimas para realizar mocks de ciertos llamados para aislar totalmente la función desarrollada. Además se utilizó la herramienta Sonarqube la cual no solamente mide la cobertura del código en cuanto a pruebas unitarias, sino también analiza la aplicación de tal manera que no tengamos códigos smell, duplicidad en el código o bugs que se hubieran podido presentar.

A continuación, se muestra el reporte de Sonarqube:

**Imagen 1**

***Reporte de cobertura de código***

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Como se muestra en la anterior imagen todos los proyectos tienen una cobertura mayor al 80% del código cubierto por pruebas, tiene 0 Smells, 0 Vulnerabilidades y 0 Bugs.

**Firmas en las lambdas**

* **Discover-mutants**

**URL:** [**https://bw5e1gfayg.execute-api.us-east-2.amazonaws.com/dev/mutant**](https://bw5e1gfayg.execute-api.us-east-2.amazonaws.com/dev/mutant)

**Method: POST**

**Request: {**

**“dna”: Array<string>**

**}**

**Response: {**

**“message”: string**

**}**

* **Get-stadistics-mutants**

**URL:** [**https://bw5e1gfayg.execute-api.us-east-2.amazonaws.com/dev/stats**](https://bw5e1gfayg.execute-api.us-east-2.amazonaws.com/dev/stats)

**Method: GET**

**Response: {**

**“message”: string**

**}**

* **Save-adn – SQS**

**URL:** [**https://sqs.us-east-2.amazonaws.com/489774021742/save-mutants-dynamo-sqs**](https://sqs.us-east-2.amazonaws.com/489774021742/save-mutants-dynamo-sqs)

**Request: {**

**“adn”: Array<string>,**

**“isMutant”: Boolean**

**}**