

UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR



Máster en Bioinformática y Biología Computacional

TRABAJO FIN DE MÁSTER

DESARROLLO DE UN WORKFLOW PARA EL ANÁLISIS GENÓMICO DE *CAMPYLOBACTER JEJUNI*

Autor: José Antonio Barbero Aparicio

Directores: José Francisco Díez Pastor y Beatriz Melero Gil

Ponente: Alberto Suárez González

Febrero 2019

DESARROLLO DE UN WORKFLOW PARA EL ANÁLISIS GENÓMICO DE *CAMPYLOBACTER JEJUNI*

Autor: José Antonio Barbero Aparicio
Directores: José Francisco Díez Pastor y Beatriz Melero Gil
Ponente: Alberto Suárez González

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Febrero 2019

Resumen

Resumen

Palabras Clave

Abstract

Keywords

Agradecimientos

Índice general

Índice de Figuras	IX
Índice de Tablas	X
1. Introducción	1
1.1. Motivación del proyecto	1
1.2. Objetivos y enfoque	2
1.3. Metodología y plan de trabajo	2
1.3.1. Metodología	2
1.3.2. Plan de Trabajo	2
2. Estado del arte	7
2.1. Técnicas en bioinformática	7
2.1.1. Control de calidad	7
2.1.2. Ensamblado	7
2.1.3. Anotación	8
2.1.4. Análisis de resistencia a antibióticos	9
2.1.5. Análisis pangenómico	9
2.2. Herramientas en informática	9
2.2.1. Plataforma de soporte del workflow	9
2.2.2. Virtualización	10
3. Sistema, diseño y desarrollo	11
3.1. Estructura general del sistema	11
3.2. Desarrollo y configuración de <i>Docker</i>	11
3.3. Desarrollo y configuración del workflow en <i>Galaxy</i>	12
3.3.1. <i>Prinseq</i>	13
3.3.2. <i>SPAdes</i>	13
3.3.3. <i>ABRicate</i>	13
3.3.4. <i>Prokka</i>	14
3.3.5. <i>Roary</i>	14

3.4. Desarrollo de la capa externa con <i>Python</i>	15
3.4.1. Capa de uso simplificado	15
3.4.2. Agrupación de resultados	15
4. Experimentos Realizados y Resultados	17
4.1. Bases de datos y protocolo	17
4.2. Sistemas de referencia	17
4.3. Escenarios de pruebas	17
4.4. Experimentos del sistema completo	17
5. Conclusiones y trabajo futuro	19
Glosario de acrónimos	21
Bibliografía	22
A. Manual de utilización	29
A.1. Requisitos	29
A.2. Utilización	30
A.2.1. Instalación de Docker	30
A.2.2. Descarga de la imagen <i>Galaxy</i>	30
A.2.3. Control del contenedor <i>Docker</i>	31
A.2.4. Funcionamiento del workflow en <i>Galaxy</i>	31
A.2.5. Utilidad de ejecución simplificada	33
A.2.6. Tratamiento de datos obtenidos	35
B. Manual del programador	37
B.1. Requisitos hardware recomendados	37
B.2. Requisitos software necesarios	37
B.2.1. <i>Docker</i>	37
B.2.2. <i>Python</i>	38
B.3. Estructura del proyecto	38
B.3.1. Directorio de ensamblado: « <i>build</i> »	38
B.3.2. Directorio de desarrollo: « <i>dev</i> »	40
B.3.3. Directorio de ejecución: « <i>exes</i> »	40

Índice de Figuras

2.1. Ejemplo de grafo De Bruijn de tamaño de k-mer 3	8
3.1. Estructura del sistema	12
A.1. Interfaz de subida de los conjuntos de datos	32
A.2. Interfaz de edición del workflow	33
A.3. Selección de las colecciones de entrada del workflow	34
A.4. Ejemplo de un comando ls	35
B.1. Contenido del proyecto	39

Índice de Tablas

1.1. Estimación en horas del plan de trabajo	3
2.1. Herramientas externas de <i>Prokka</i> [1]	8
A.1. Requisitos del sistema	29

1

Introducción

1.1. Motivación del proyecto

Campylobacter jejuni es una bacteria Gram negativa que, a pesar de tener unas condiciones complicadas de crecimiento [2], es la zoonosis bacteriana que produce un mayor número de intoxicaciones alimentarias en los países tanto desarrollados como en vías de desarrollo. Por ejemplo, en la EU en el año 2016 se declararon del orden de 250.000 casos comprobados [3]. El coste debido a la campilobacteriosis se estima en la EU en torno a 2,4 billones de euros anuales. La fuente de contaminación más habitual es el consumo de carne de pollo poco cocinada [4]. El grupo de investigación Tecnofood¹ lleva varios años investigando sobre las fuentes de contaminación de este microorganismo a lo largo de la cadena alimentaria [2, 4, 5]. En la actualidad se dispone de una colección de *Campylobacter* spp. de alrededor de 2000 cepas. Con el fin de obtener una información más precisa sobre la persistencia de este microorganismo a lo largo de la cadena alimentaria, se han aislado varios genotipos persistentes en el matadero. De estos se han secuenciado con un equipo MiSeq (Illumina) 45 de ellas.

El proyecto consiste en diseñar un workflow que permita, a partir de los datos obtenidos en formato *fastq* proporcionados por el equipo, conseguir realizar las fases de trimming y evaluación de la calidad de las secuencias obtenidas, obtención de contigs, assembling y anotación, para poder tener la información de la secuencia de genes del genoma completo [6, 7, 8] de las cepas de *Campylobacter jejuni* secuenciadas. En la actualidad, existen diferentes programas desarrollados por varios grupos de investigación internacional que se encargan de coordinar cada una de las tareas mencionadas, además de la relación de los datos de salida de unas herramientas con los de entrada de otras. Se trata de buscar la solución más eficaz y fácil de implementar y que dé los mejores resultados, por lo que habrá que comparar diferentes programas y estrategias. Adicionalmente, se requiere incorporar en este workflow, o en análisis paralelos [9], la posibilidad de detectar insertos de origen viral y/o plásmidos en el genoma y herramientas que permitan la comparación rápida de los genomas de las distintas cepas aisladas, algunas de ellas pertenecientes a cepas altamente clonales. Esta herramienta se ha demandado por parte de un grupo sin conocimientos informáticos, por lo que se requiere desarrollar un entorno de fácil uso por su parte.

¹<https://www.ubu.es/tecnologia-de-los-alimentos-tecnofood>

El proyecto plantea una colaboración entre los grupos ADMIRABLE² y TECNOFOOD de la Universidad de Burgos. Especializados en informática y ciencia y tecnología de los alimentos respectivamente. Dada esta combinación de disciplinas, el proyecto se encuentra en el marco de los trabajos considerados dentro del campo de la bioinformática.

1.2. Objetivos y enfoque

El objetivo principal del proyecto es el desarrollo del workflow que permita el análisis de las cepas de *Campylobacter jejuni*, para lo que se utilizarán *Galaxy* [10] y las herramientas disponibles en la «tool shed» (conjunto de herramientas que ofrece *Galaxy* para su instalación) para cada paso. *Galaxy* es una herramienta que permite análisis computacionales de datos biológicos. El segundo objetivo se centra en crear una herramienta de utilización simplificada del workflow, en la que los pasos para su ejecución se configuren automáticamente, y que se llevará a cabo utilizando *Python* y la API de *Galaxy* a través de Bioblend [11]. Además, para facilitar el despliegue y ejecución de la herramienta desarrollada, se va a crear un contenedor *Docker*. Por lo tanto, el siguiente objetivo parcial será desarrollar la imagen *Docker* que sirva de base. Finalmente, se desea tener alguna forma sencilla de tratar los datos de salida del workflow, por lo que dentro de la interfaz gráfica se incluirán herramientas con las que gestionar toda la información resultante en forma de gráficos, tablas tipo hoja de cálculo, formatos *pdf*, etc.

Objetivos del proyecto:

- Crear un sistema *Docker* sobre el que desplegar el proyecto
- Desarrollar el workflow necesario para el análisis de las cepas en *Galaxy*
- Desarrollar una capa de utilización simplificada del workflow
- Añadir un sistema de gestión sencilla de los datos de salida.

En la tabla 1.3.2 podemos ver un despliegue con la planificación temporal de cada uno de estos objetivos.

1.3. Metodología y plan de trabajo

1.3.1. Metodología

La metodología utilizada en el desarrollo del proyecto, dada su cercanía en su estructura a un proyecto de software tradicional, será de tipo ágil, basada en reuniones en cada sprint. La carga de trabajo, como aproximación a falta de conocer ciertos requisitos que puedan surgir durante el desarrollo, se dividirá en la estructura definida en la estimación en horas del plan de trabajo.

1.3.2. Plan de Trabajo

Sprint 1 (18/9/2018 - 3/10/2018)

El primer sprint ha estado centrado tanto en definir con más exactitud la dirección del proyecto como en un primer acercamiento a las principales herramientas con las que va a desarrollarse.

Tras unos primeros pasos con *Galaxy* [12] y *Docker* [13], se ha tomado como referencia el trabajo Bioinfworkflow de Sergio Chico [14] como base para la imagen *Docker* del proyecto. Dado que el proyecto de Github daba algunos problemas en la instalación, se ha desarrollado un script propio que produce los mismos resultados.

²<https://www.ubu.es/advanced-data-mining-research-and-bioinformatics-learning-admirable>

Tarea 1 - Desarrollo de la imagen <i>Docker</i>	
Tarea 1.1 - Adaptar la imagen previa orientada a <i>Galaxy</i>	50 horas
Tarea 1.2 - Permitir desplegar la imagen en un servidor	10 horas
Tarea 2 - Desarrollo del workflow en <i>Galaxy</i>	
Tarea 2.1 - Selección de las herramientas	30 horas
Tarea 2.2 - Configuración completa del workflow	100 horas
Tarea 3 - Desarrollo del modo de uso simplificado	
	60 horas
Tarea 4 - Desarrollo del sistema de tratamiento de datos de salida	
	50 horas

Cuadro 1.1: Estimación en horas del plan de trabajo

Una vez se ha tenido disponible la imagen de *Docker*, el sprint se ha centrado en algunos aspectos importantes para partes futuras del desarrollo. Entre ellos destaca la investigación acerca del formato de los workflows de *Galaxy* (.ga) ya que en un futuro será necesario generar este tipo de ficheros para introducirlos en *Galaxy*. También resulta relevante la investigación acerca de las posibilidades que ofrece la API de *Galaxy* [12] y su utilidad en Bioblend [15], que nos facilitan la opción de utilizar *Galaxy* sin necesidad de hacerlo a través de su interfaz.

Sprint 2 (4/10/2018 - 17/10/2018)

La primera semana de este sprint ha estado dirigida a lograr una imagen *Docker* de *Galaxy* que contenga un set de herramientas básicas para formar un primer workflow. Se han valorado varias opciones de instalación en las que se han utilizado tanto la imagen básica de *Galaxy* [16] como la imagen de Bioinformatics [14]. Finalmente, se ha optado por utilizar Bioinformatics ya que parte de las herramientas necesarias ya estaban incluidas. Para realizar esta tarea se ha creado un nuevo fichero *Dockerfile* así como el listado de herramientas necesarias para su instalación.

Sprint 3 (18/10/2018 - 31/10/2018)

El sprint ha estado centrado en la correcta ejecución del workflow con las herramientas iniciales desde *Galaxy*. Durante el proceso de configuración, han surgido varias complicaciones que han impedido terminar el workflow completo en este sprint. En un principio, han surgido problemas con el filtrado de calidad utilizando *Prinseq*. Este problema no ha llegado a ser resuelto en este sprint a falta de tratar el tema con el grupo de Tecnofood. A continuación, se encontraron ciertos problemas con el formato de salida de la herramienta *Prokka*. A pesar de que la salida está marcada como formato *gff3*, un parámetro interno lo etiquetaba como *gff*. Esto impedía que la salida de *Prokka* pudiese ser utilizada como entrada en las herramientas siguientes.

Dados estos errores, se decidió trabajar en paralelo con la API de *Galaxy* desde *Python*, para intentar ejecutar tanto las herramientas como el workflow de una manera menos restringida. Finalmente, se ha llevado a cabo el desarrollo necesario para subir los ficheros a un historial y ejecutar cada una de las herramientas del workflow desde *Python*.

Sprint 4 (1/11/2018 - 14/11/2018)

La prioridad en este punto se ha centrado en completar el workflow desde *Galaxy*. Han surgido varios problemas en esta tarea. La primera es un bug en Mac por el cual los archivos eliminados

dentro de *Docker* no se eliminan del todo y quedan fijados en un fichero residual. Quizá esto pueda deberse a que *OSX* no soporta de manera nativa la virtualización a nivel de sistema operativo, sino que se basa en *Hyperkit* para crear una capa de virtualización. Esto implica que cada cierto tiempo hay que eliminar la imagen completa de *Docker* para poder liberar espacio, dado el gran tamaño de los archivos con los que se trabaja. Debido a ello, la tarea de completar el workflow se ha visto retrasada. Además, la ejecución de la herramienta *Roary* a través de *Galaxy* ejecuta sin errores pero no devuelve ningún resultado, lo que ha impedido continuar con la parte final del workflow.

Además de la tarea ya comentada, en este sprint se ha trabajado en el acceso al contenedor *Docker* desde otro ordenador en una red local. Con el objetivo de desplegar el servicio en un servidor.

También se ha estudiado la posibilidad de desarrollar una interfaz gráfica, realizando unas pruebas en las que simplemente se muestra alguna información extraída de la API de *Galaxy* en etiquetas creadas con *PyQt*.

Sprint 5 (15/11/2018 - 28/11/2018)

Al igual que en los sprints anteriores, gran parte de la carga de trabajo se ha centrado en resolver problemas en la ejecución del workflow a través de *Galaxy*. Se han realizado numerosas ejecuciones para comprobar si las salidas de cada herramienta eran las correctas. Esto ha servido para concluir que, al parecer, un fallo en la herramienta *Roary* incluida en *Galaxy*, impide que los ficheros retornados tengan contenido. Esto se ha comprobado a través de la ejecución de *Roary* standalone con los mismos datos de entrada y los mismos parámetros, obteniendo de esta manera los ficheros correctos.

Para ahorrar en tiempos de ejecución se han utilizado los ficheros fasta ya generados previamente, no los generados con la herramienta *Spades* de nuestro propio workflow. En el próximo sprint se tratará de integrar la parte previa a los pasos que ya son correctos.

También se ha añadido un fichero *.gitignore* para evitar la existencia de ficheros irrelevantes en el repositorio.

Parte del trabajo de este sprint se ha destinado a la redacción de la propuesta de proyecto y de la introducción a la documentación del mismo.

Sprint 6 (29/11/2018 - 12/12/2018)

El trabajo de este sprint se ha centrado en fragmentar el proceso del workflow lo máximo posible para detectar dónde se están produciendo los errores. Inicialmente se ha acortado el workflow hasta el paso de ensamblaje con *Spades*, ya que los problemas surgían en este punto. Posteriormente se ha ejecutado el workflow individualmente en lugar de por colecciones. De esta manera, se ha detectado que el problema se está dando en la ejecución de *Spades* con dos secuencias concretas: 590 y 443. Tras investigar probando varias ejecuciones con diferentes parámetros, se ha llegado a la conclusión de que no era un fallo de configuración de la herramienta, sino de hardware. Al ceder a *Docker* una cantidad mayor de memoria RAM (8 Gb), el problema se ha solucionado. A continuación, se ha pasado a ejecutar de nuevo por colecciones hasta el paso de *Spades*, para comprobar si con este cambio ha sido suficiente para que la ejecución sea correcta con este formato.

También se ha realizado una modificación del fichero *.gitignore* para mantener los archivos *.pdf* generados por L^AT_EX.

Sprint 7 (13/12/2018 - 26/12/2018)

La tarea principal de este sprint ha sido la creación de una herramienta *Roary* que poder integrar en *Galaxy*. Se valoró la opción de crear una nueva imagen *Galaxy* instalando la herramienta desde la creación inicial de *Docker*, pero finalmente se ha optado por subir esta versión de *Roary* al *tool shed* de *Galaxy*.

A continuación, se han realizado varias comprobaciones del funcionamiento de la integración de esta herramienta, con resultados positivos. Sin embargo, al ejecutar el workflow completo, parecen surgir de nuevo errores en la parte de *Prokka*, provocados por la ejecución previa de *Spades*.

También se ha añadido el apartado de la Introducción de la documentación.

Sprint 8 (27/12/2018 - 9/1/2019)

El trabajo en este sprint se ha enfocado a completar definitivamente la ejecución del workflow solucionando los errores existentes. La máquina virtual con la que se realizaban estas pruebas disponía de 4 GB de RAM, insuficiente para la ejecución con este conjunto de datos. Esto causaba algunos errores poco descriptivos al ejecutarlo. El equipo de pruebas se ha aumentado a 32 GB de RAM, cediendo 16 GB a la máquina virtual, solucionando así estos errores.

A continuación, se ha desarrollado un script *Python* a partir del cual realizar todas las tareas necesarias para ejecutar el workflow, utilizando la API de *Galaxy*.

Se ha invertido el poco tiempo restante del sprint en el desarrollo de la documentación, definiendo la estructura de los apartados de «estado del arte» y «sistema, diseño y desarrollo».

2

Estado del arte

2.1. Técnicas en bioinformática

2.1.1. Control de calidad

Las nuevas técnicas de secuenciación masiva proporcionan una gran ventaja en cuanto a la velocidad de lectura de las secuencias [17]. Sin embargo, hay que tener especial precaución en el análisis de la calidad de estas, ya que una mayor velocidad puede llegar a implicar errores en las lecturas en algunos casos. Especialmente, en aquellos en los que se va a realizar un análisis en el que se pueden transmitir estos errores a través de cada una de las herramientas que componen el proceso, pudiendo llegar a provocar conclusiones erróneas al final del análisis.

Los artefactos, las lecturas de baja calidad, las inserciones y deleciones y la contaminación con adaptadores y primers son varias de las causas que pueden llevar a análisis problemáticos [18]. No obstante, en la actualidad, existen numerosos métodos enfocados al tratamiento de estos problemas, normalmente agrupados en herramientas entre las que destacan algunas como *FastQC* [19] o *Prinseq* [20].

2.1.2. Ensamblado

Los sistemas de secuenciación actuales pierden calidad cuando aumenta el tamaño de las lecturas, por lo tanto, se generan gran cantidad de pequeñas lecturas que deberemos unir. El objetivo del proceso de ensamblado es recoger todos estos fragmentos menores que conformarían una secuencia mayor para alinearlos y empalmarlos con el objetivo de reconstruirla [21]. Dada la tendencia al aumento de la cantidad de datos a procesar por los ensambladores a raíz de las nuevas técnicas de secuenciación, estos se han visto empujados a emplear técnicas cada vez más eficientes para evitar crear un cuello de botella en esta fase del análisis. Los diferentes aspectos de cada secuenciación como el tamaño, si se trata de un organismo procariota o eucariota, de un transcriptoma, metagenoma, etc, influyen en la eficacia de unos u otros algoritmos. En nuestro caso, al tratarse de bacterias, cuyo genoma es pequeño, se ha decidido utilizar una herramienta destinada a esta característica, como es *SPAdes* [22].

Hay que destacar que el problema del ensamblado es, computacionalmente, extremadamente costoso, por lo que la importancia de encontrar un algoritmo que resuelva el problema en un

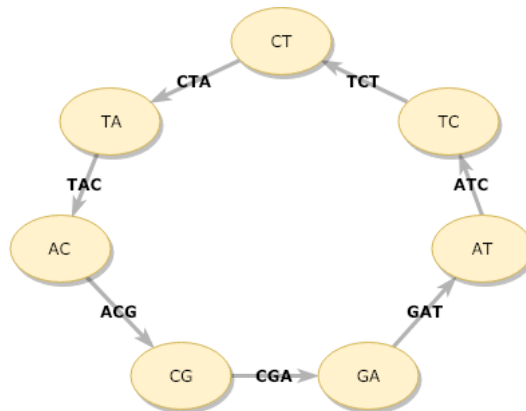


Figura 2.1: Ejemplo de grafo De Bruijn de tamaño de k-mer 3

tiempo aceptable es enorme. En este caso, la tecnología en la que se basa tanto esta herramienta como otras muy potentes [23] son los grafos de De Bruijn [24]. Se trata de un tipo de grafos utilizados para representar solapamientos entre secuencias. Es importante tener en cuenta que, en esta representación, los enlaces del grafo estarán formados por las subcadenas de tamaño k -mer de cada fragmento, mientras que los nodos serán las uniones que utilicen las dos subcadenas para lograr empalmar el final de una con el inicio de la otra 2.1. La base del algoritmo centrado en este tipo de grafos es encontrar su camino hamiltoniano (que recorra todos los nodos de un grafo una sola vez), siendo este camino la secuencia ensamblada completa.

2.1.3. Anotación

El proceso de anotación del genoma se basa en identificar y etiquetar todos los aspectos relevantes encontrados en una secuencia, incluyendo, principalmente, regiones codificantes; aunque también son interesantes otros elementos como ARN no codificante, péptidos señal, etc [25].

En algunos casos el proceso se basa en la ejecución de un pipeline automático de anotación seguido de un proceso de filtrado manual [26]. El objetivo en este proyecto es la automatización del proceso en un tiempo de ejecución asequible sin dejar de lado la precisión. En experimentos realizados, *Prokka* ha dado mejores resultados que alternativas como *RAST* o *xBase2* y ha demostrado ser eficiente incluso en ordenadores convencionales de usuario [1].

Prokka basa su anotación en una serie de herramientas externas, cada una de ellas especializada en la identificación de una característica de la secuencia concreta 2.1.3.

Herramientas	Características objetivo
Prodigal [27]	Secuencias codificantes (CDS)
RNAmmmer [28]	RNA Ribosómico (rRNA)
Aragorn [29]	RNA de transferencia
SignalP [30]	Péptidos señal
Infernal [31]	ARN no codificante

Cuadro 2.1: Herramientas externas de *Prokka* [1]

2.1.4. Análisis de resistencia a antibióticos

Desde el comienzo de la utilización de antibióticos por parte del ser humano, se ha sometido a las bacterias a un proceso de selección natural por el cual aquellas con la capacidad de sobrevivir a cierta concentración de antibiótico, proliferarán. La resistencia a antibióticos se está convirtiendo en uno de los problemas más amenazantes para el ser humano [32]. *Campylobacter jejuni*, la bacteria que nos ocupa en este caso, no es una excepción y ya se han detectado casos de resistencia en ella [33].

Comienzan a identificarse algunos de los genes responsables de esta resistencia y se están creando bases de datos con las que registrarlos [34]. Consecuentemente, han ido surgiendo herramientas basadas en estas bases de datos con las que contrastar si los fragmentos de cierta secuencia están incluidos en una de ellas. *ABRRicate* [35] es una de las muchas que existen e incluye, tanto bases de datos dedicadas a genes de resistencia a antibióticos como bases de datos de genes de virus.

2.1.5. Análisis pangenómico

El análisis pangenómico en bacterias es un proceso complejo, condicionado siempre a la aparición de nuevos genes. No obstante, su resolución puede suponer ventajas muy significativas [36]. Modelos matemáticos aplicados a este problema concluyen que genes únicos continuarán apareciendo independientemente de la cantidad de genomas que se secuencien [37]. Además, la construcción de este genoma supone un coste computacional muy relevante debido a que se trata de un problema NP-complejo [38].

Llegados a este punto, el objetivo en este caso es la construcción de un pangenoma a partir de todas las cepas de las que disponemos y realizar un análisis en el que se indique qué cepas contienen cada gen. En la actualidad se están desarrollando herramientas dedicadas a esta tarea. *Roary* [39] es una de ellas, basada en la utilización de los fragmentos de secuencias codificantes etiquetados por otras herramientas como *Prokka* para transformarlos en secuencias de proteínas y así simplificar el conjunto de datos. De esta manera, la comparación «todos contra todos» de *BLASTP* [40], se ve muy optimizada.

2.2. Herramientas en informática

Además de las herramientas puramente enfocadas al análisis genético, son necesarias otras técnicas más centradas en la informática para facilitar la utilización del workflow. En este caso, una plataforma con la que poder interactuar con él y un sistema en el que sostener toda su infraestructura.

2.2.1. Plataforma de soporte del workflow

Existe una gran variedad de plataformas sobre las que construir y ejecutar un workflow en la actualidad. Uno de los primeros en aparecer fue *Discovery Net* [41], una utilidad enfocada a coordinar trabajos de servicios ejecutados de manera remota. Sin embargo, es un sistema basado en código y no posee su propia interfaz gráfica, lo que dificultaría su utilización por parte de usuarios ajenos a la programación. Lo mismo ocurre con muchos de los sistemas más utilizados, como *Anduril* [42], *BioQueue* [43] o *Cuneiform* [44]. Sin embargo, hay varios frameworks que permiten un control gráfico del workflow, como *Apache Taverna* [45], *Triana* [46], *Kepler* [47], *Galaxy* [12] o *Yawl* [48]. Existen varias publicaciones que analizan las ventajas e inconvenientes

de cada uno de los sistemas [49, 50, 51, 52]. Entre ellos, por su desarrollo más avanzado y su capacidad para desplegar un sistema completo en un navegador, además de una interfaz fácil de utilizar y una API *Python* [15] disponible, destaca *Galaxy*.

2.2.2. Virtualización

Llamamos virtualización a un conjunto de tecnologías destinadas a simular componentes hardware en forma de software. Eso permite la creación de entornos en los que podemos disponer de software y hardware dentro de nuestro propio sistema. Por ejemplo, existe la posibilidad de establecer un sistema *Ubuntu* virtual dentro de un sistema Windows antifrón. En el caso de este ejemplo, estaríamos hablando de una virtualización total, en la que se simula el sistema operativo completo, lo que supone una carga muy pesada debido a que existirá una gran cantidad de software que no va a ser utilizado. En los últimos años se han ido mejorando las técnicas de virtualización parcial, en las que es posible contener solamente la parte mínima de sistema operativo y de software necesarios para la herramienta que queramos simular. Estas técnicas se basan en contenedores, en los que se encapsulan las imágenes con las dependencias necesarias para su funcionamiento. Entre las herramientas más utilizadas, destaca *Docker* [13], siendo la más reconocida por su popularidad entre los usuarios.

Dado que existe una imagen *Galaxy* estable para *Docker* a partir de la cual poder trabajar, así como una capa adicional desarrollada para el mismo grupo de investigación anteriormente, *Docker* es el sistema que más se adecuaba a las necesidades del proyecto.

3

Sistema, diseño y desarrollo

3.1. Estructura general del sistema

El sistema desarrollado se ha basado en *Docker* para evitar problemas de configuración y portabilidad, es decir, podrá ser desplegado en cualquier sistema que soporte esta tecnología. La estructura está basada en varias capas, todas ellas sobre el sistema anfitrión que elijamos, pudiendo adaptarse incluso a un servidor 3.1.

- La primera capa de esta estructura es el propio sistema anfitrión que, normalmente, será el equipo de utilice el usuario regularmente. Su única función es la de servir de base para la ejecución de la imagen *Docker*.
- Inmediatamente por encima del sistema anfitrión se encuentra la imagen *Docker*, encargada de facilitar la configuración y las dependencias necesarias para la ejecución de *Galaxy*.
- La imagen *Docker* incluye el despliegue de *Galaxy* sobre el puerto 8080. Por lo que podemos considerar como otra capa del sistema a la propia ejecución de *Galaxy*.
- El punto principal del trabajo se encuentra en el apartado de workflows dentro de *Galaxy*. En él se localiza la secuencia de herramientas que conforma el núcleo del proyecto.
- Exteriormente a *Galaxy* y *Docker*, se han desarrollado dos herramientas que interactúan con el workflow. La primera es un script *Python* que se encarga de toda la configuración y ejecución del workflow, para facilitar su utilización a usuarios no experimentados en informática. La segunda herramienta se encarga de tratar los datos obtenidos de la ejecución para generar un formato más fácil de comprender y manipular por el usuario.

3.2. Desarrollo y configuración de *Docker*

La imagen *Docker* utilizada está basada, con algunas modificaciones, en la desarrollada por Sergio Chico en su Trabajo de Fin de Máster [14]. La cual, a su vez, hereda de una imagen *Docker Galaxy* estable desarrollada por Björn A. Grüning [16].

Utilizando como punto de partida la imagen de Sergio Chico, se han realizado una serie de adaptaciones para adecuarla al uso de nuestro workflow. La primera modificación realizada fue la instalación de las nuevas herramientas necesarias que no estaban disponibles en ese momento. Al conjunto de herramientas iniciales se añadieron *Prinseq*, *Spades*, *Roary*, *ABRicate* y *Prokka*.

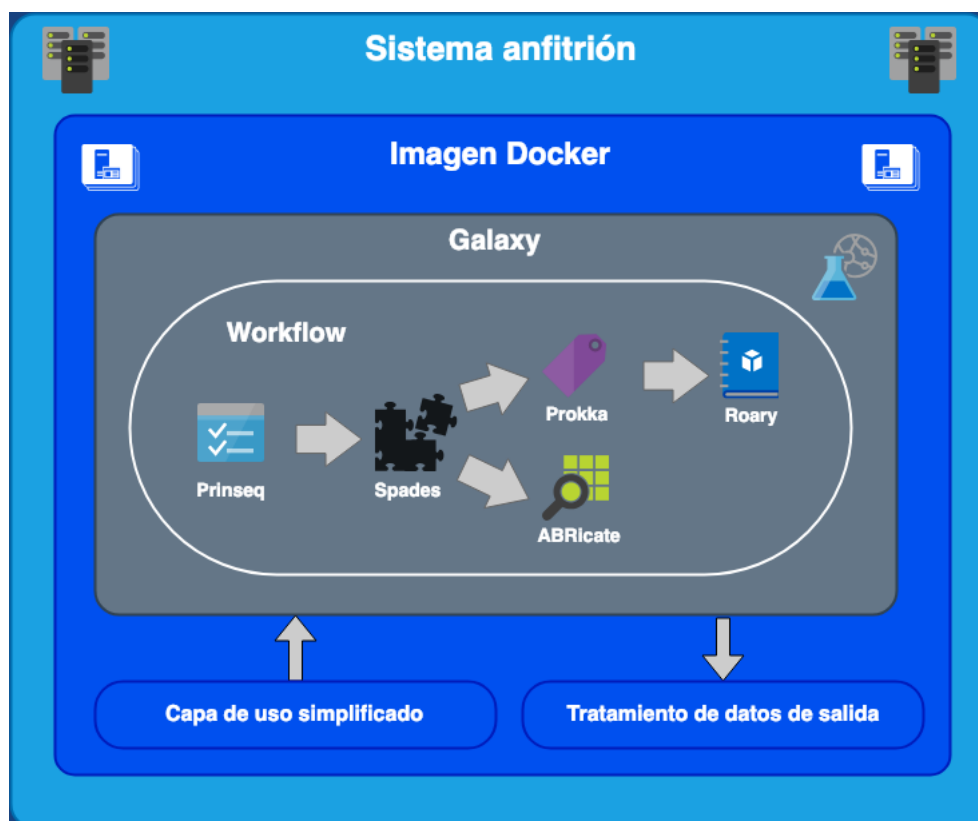


Figura 3.1: Estructura del sistema

Además, en las opciones de instalación de herramientas de *Galaxy* solo estaba disponible para la versión previa de *Roary* que producía, por algún tipo de bug, ficheros de salida vacíos. Por lo tanto, se creó una nueva herramienta a partir de la original con la nueva versión disponible para escritorio de *Roary*.

Una vez realizadas estas adaptaciones a la imagen, se ha dejado disponible en *Dockerhub* bajo el nombre *jbarberoaparcio/workflowmanager*¹.

Además de esto, se han desarrollado varios scripts simples para facilitar el uso de *Docker*. El primero de ellos se encarga del montaje completo de la imagen a partir del Dockerfile, en caso de que quisieran realizarse modificaciones en un futuro. El segundo se encarga del arranque de la imagen desplegándola a través de *Docker*. El tercero se desarrolló para resolver un bug en *OSX* por el cual los ficheros eliminados dentro de la imagen no se eliminaban correctamente. Esto provocaba que un fichero tipo *log* ocupando todo el espacio disponible en el disco. El script se encarga de eliminar esta información sin borrar las imágenes existentes.

3.3. Desarrollo y configuración del workflow en *Galaxy*

A lo largo del workflow, se realizarán varios análisis diferenciados de las secuencias. En algunos casos estos análisis servirán únicamente de entrada para el siguiente paso y en otros aportarán por sí mismos parte de la información deseada. Cabe destacar que se ha definido la entrada del workflow como un formato de dos colecciones, una para cada dirección de las lecturas de las secuencias.

¹<https://hub.docker.com/r/jbarberoaparcio/workflowmanager>

3.3.1. *Prinseq*

Con el objetivo de realizar un filtrado de calidad de las secuencias, se ha utilizado *Prinseq* [20]. Esta herramienta, desarrollada en *Perl*, ofrece la posibilidad de generar un informe en el cual se indican diferentes estadísticas sobre la calidad de cierta secuencia. Además, utiliza esa información para realizar el filtrado que le indiquemos, pudiendo ser en función de la longitud de las lecturas o de su calidad.

Como formato de entrada, introduciremos en *Prinseq* un fichero *fastq*, en nuestro caso proveniente de las secuenciaciones de diferentes cepas de *Campylobacter jejuni* utilizando un equipo Illumina MiSeq. Como salida, obtendremos un fichero *fastq* en el que se habrán eliminado todos los fragmentos que no hayan cumplido las condiciones que se hayan fijado en los parámetros de ejecución. En nuestro caso, se han eliminado las secuencias de longitud menor que 50 y de índice de calidad menor de 15. El resto de parámetros se han mantenido por defecto a excepción de la indicación de que se trata de una secuencia «paired-end», necesaria para la ejecución de la manera en la que deseamos llevarla a cabo.

Prinseq ha sido una de las herramientas que menos complicaciones han presentado a lo largo del proyecto, devolviendo errores comprensibles cuando las ejecuciones no eran correctas y ofreciendo toda la información necesaria para resolverlo. A pesar de ello, la versión de *Prinseq* disponible en *Galaxy* tiene la desventaja de no generar el informe con todas las estadísticas resultantes que sí genera su versión web.

3.3.2. *SPAdes*

SPAdes [22] ha sido la utilidad seleccionada para la fase de ensamblado. Se trata de un conjunto de herramientas que facilitan un modo de recoger todas las lecturas validadas en el proceso de filtrado y encadenarlas de la manera en la que corresponden en la secuencia. Fue desarrollado enfocándose a genomas pequeños, como bacterias y hongos. Por lo tanto, se adapta bien al caso de uso que estamos tratando.

Para su ejecución, se debe indicar a *SPAdes* la estructura que estamos siguiendo en el workflow, introduciendo ficheros separados en los dos sentidos de lectura de las secuencias en forma de colecciones. Estos ficheros provendrán de la salida de la ejecución de *Prinseq*, tras realizar el filtrado de calidad. El resto de parámetros de la ejecución, a excepción de la longitud de los k-mer, establecida a 77, se han mantenido por defecto. También se le ha indicado que las bibliotecas a utilizar sean «paired-end».

Dado el gran tamaño de las lecturas de los casos en los que el workflow ha sido probado, *SPAdes* ha resultado problemático por tener que almacenar en RAM todas las lecturas al mismo tiempo. Esto ha implicado que, para las muestras de prueba, se haya requerido de 16 GB de memoria RAM para poder ejecutarlo. Además, los mensajes de error de la versión *Galaxy* de *SPAdes* no dan demasiada información en muchos casos. En algunos otros, relacionados con la capacidad de memoria, no se producía ningún mensaje de error, sino que se daba por correcta la ejecución a pesar de que los ficheros retornados estaban vacíos. A pesar de estos problemas de desarrollo, *SPAdes* ha terminado cumpliendo su función correctamente tras identificar los errores del proceso.

3.3.3. *ABRicate*

Uno de los principales intereses al comenzar el proyecto era el análisis de resistencia a antibióticos de las secuencias. *ABRicate* [35] es una herramienta desarrollada en *Perl* que permite

la localización de genes vinculados a resistencia a antibióticos o relacionados con virus realizando un cribado masivo de los contigs introducidos. Para ello utiliza las bases de datos de *Resfinder* [53], *CARD* [54], *ARG-ANNOT* [55], *NCBI BARRGD*, *EcOH*, *PlasmidFinder* [56], *Ecoli_VF* y *VFDB* [57]. En nuestro caso, ese input serán los contigs obtenidos por *SPAdes* para cada una de las secuencias. Esta herramienta forma uno de los finales del workflow, es decir, su salida no sirve como entrada a ninguna otra herramienta sino que los datos que se obtienen de ella tienen su utilidad propia.

La configuración de *ABRicate* es muy sencilla y ofrece pocas posibilidades de variación, por lo que no ha sido necesario modificar ningún parámetro de la herramienta. La ejecución por defecto no ha presentado problemas.

Una de las limitaciones de *ABRicate* es que no admite lecturas tipo *fastq*, solamente contigs. En nuestro caso los contigs provienen de la salida de *SPAdes*, por lo que es una herramienta adecuada.

3.3.4. *Prokka*

Para la fase de anotación se ha decidido utilizar *Prokka* [1], una herramienta desarrollada en *Perl* destinada específicamente a genomas procariotas y que destaca por su rapidez. El proceso de anotación de *Prokka* se divide en dos fases. En la primera, se utiliza *Prodigal* para la identificación de regiones codificadoras de proteínas. En la segunda, se compara con varias bases de datos para reconocer, por similitud, cuál es la proteína de la que se trata.

La configuración de *Prokka* en *Galaxy* no ha resultado complicada a pesar de que los parámetros son numerosos. El único problema encontrado en la configuración ha sido a raíz del formato *GFF3*. A pesar de que se indique que los ficheros de salida siguen esa estructura, realmente no es así, sino que internamente son formato *GFF*. Para solucionar este inconveniente se ha añadido una fase nueva en el workflow, en la que una herramienta simple de concatenación une la salida *GFF* con la propia secuencia del genoma.

3.3.5. *Roary*

Uno de los objetivos principales del proyecto era la realización de un análisis pangenómico. Utilizando como entrada las anotaciones provenientes de *Prokka* de todas las muestras, *Roary* [39] se encarga de calcular el pangenoma. Ha resultado ser una herramienta realmente rápida, con la que en unos minutos se obtienen los resultados.

Roary está desarrollado para recibir ficheros *GFF3* desde *Prokka*, por lo que no debería haber problemas de compatibilidad (una vez solucionado el error de formato comentado en el punto anterior). El problema con la configuración de *Roary* dentro del workflow es que, si se instala desde el repositorio oficial en el *Tool Shed* de *Galaxy*, no se obtiene ningún tipo de contenido en los ficheros de salida. Estudiando este problema, se llegó a la conclusión de que se podría deber a que la versión disponible en este repositorio no es la versión final publicada para escritorio. Por lo tanto, se creó una herramienta *Galaxy* nueva utilizando la versión *standalone* de *Roary*. Una vez disponible esta herramienta en *Galaxy* y manteniendo la configuración establecida para la versión anterior, los ficheros fueron obtenidos correctamente.

Roary recibe todos los ficheros tratados por el workflow (a excepción de la salida independiente de *ABRicate*) y sirve como herramienta final. La clasificación de cada cepa según su coincidencia en cada gen con el pangenoma creado por *Roary* no son enviados a ninguna otra herramienta dentro de *Galaxy* y finalizan el workflow.

3.4. Desarrollo de la capa externa con *Python*

A pesar de que el workflow completo ha sido desarrollado dentro de *Galaxy*, se han añadido algunas funcionalidades extra utilizando *Python*.

3.4.1. Capa de uso simplificado

Dado que el equipo al que está destinada la herramienta no pertenece al campo de la informática, se solicitó una herramienta que fuese sencilla de utilizar. Por ello, se ha creado un script que realiza todos los pasos necesarios para ejecutar el workflow simplemente con ejecutarlo, sin ser necesario ningún tipo de configuración. En caso de que el usuario tuviese conocimientos técnicos, seguirá teniendo la posibilidad de utilizar la herramienta sin este script.

Para ello, se ha utilizado la API de *Galaxy*, parte de la librería *BioBlend*². Esta API nos permite controlar todos los aspectos de una instancia *Galaxy* desde código *Python*.

Desde el script se crea una sesión en la instancia de *Galaxy* a partir de la cual se realizan todas las tareas. Una vez conectado, el script carga el archivo que almacena el workflow que va a ejecutar. Después, para mantener los datos aislados e identificados, crea historiales nuevos para los ficheros de entrada y los de salida con un nombre único a partir de la fecha y hora de creación. A continuación carga los datos brutos de inicio, lo que, dependiendo de su tamaño, puede llevar unos minutos. Para que el workflow funcione correctamente, necesita recibir los datos en forma de colección, por lo que el script se encarga de crear una para cada conjunto de datos, es decir, lecturas en un sentido y en el inverso. El paso siguiente, la ejecución del workflow, es el fundamental y el más costoso en tiempo. Para controlar el estado de la tarea en cada momento, se va realizando un control cada pocos segundos que comprueba si las tareas siguen en proceso. Una vez finalizado, los datos de salida se mantienen en un historial propio, pero para poder utilizarlos fuera de *Galaxy*, el script se encarga de almacenarlos en el disco local del equipo en una carpeta con el nombre del historial.

3.4.2. Agrupación de resultados

Para facilitar el acceso a los resultados de todas las herramientas, se ha planteado la creación de un solo fichero que pueda abrirse con *Excel*, herramienta que resulta familiar al grupo que utilizará el workflow. Para ello, desde un script de *Python* se genera un solo fichero en el que cada hoja contiene los resultados de uno de los pasos del workflow.

²<https://bioblend.readthedocs.io/en/latest/>

4

Experimentos Realizados y Resultados

4.1. Bases de datos y protocolo

4.2. Sistemas de referencia

4.3. Escenarios de pruebas

4.4. Experimentos del sistema completo

5

Conclusiones y trabajo futuro

Glosario de acrónimos

- **DNA**: Deoxyribonucleic acid
- **RNA**: Ribonucleic acid
- **CDS**: Coding Sequence
- **GFF**: General Feature Format

Bibliografía

- [1] T. Seemann. Prokka: rapid prokaryotic genome annotation. *Bioinformatics*, 30(14):2068–2069, jul 2014.
- [2] Lourdes García-Sánchez, Beatriz Melero, Isabel Jaime, Marja-Liisa Hänninen, Mirko Rossi, and Jordi Rovira. Campylobacter jejuni survival in a poultry processing plant environment. *Food Microbiology*, 65:185–192, aug 2017.
- [3] Campylobacteriosis - Annual Epidemiological Report 2016 [2014 data].
- [4] Lourdes García-Sánchez, Beatriz Melero, Ana Ma Diez, Isabel Jaime, and Jordi Rovira. Characterization of Campylobacter species in Spanish retail from different fresh chicken products and their antimicrobial resistance. *Food Microbiology*, 76:457–465, dec 2018.
- [5] Beatriz Melero, Pekka Juntunen, Marja-Liisa Hänninen, Isabel Jaime, and Jordi Rovira. Tracing Campylobacter jejuni strains along the poultry meat production chain from farm to retail by pulsed-field gel electrophoresis, and the antimicrobial resistance of isolates. *Food Microbiology*, 32(1):124–128, oct 2012.
- [6] Clifford G. Clark, Chrystal Berry, Matthew Walker, Aaron Petkau, Dillon O. R. Barker, Cai Guan, Aleisha Reimer, and Eduardo N. Taboada. Genomic insights from whole genome sequencing of four clonal outbreak Campylobacter jejuni assessed within the global C. jejuni population. *BMC Genomics*, 17(1):990, dec 2016.
- [7] Ann-Katrin Llarena, Eduardo Taboada, and Mirko Rossi. Whole-Genome Sequencing in Epidemiology of Campylobacter jejuni Infections. *Journal of clinical microbiology*, 55(5):1269–1275, may 2017.
- [8] S. Zhao, G. H. Tyson, Y. Chen, C. Li, S. Mukherjee, S. Young, C. Lam, J. P. Folster, J. M. Whichard, and P. F. McDermott. Whole-Genome Sequencing Analysis Accurately Predicts Antimicrobial Resistance Phenotypes in Campylobacter spp. *Appl. Environ. Microbiol.*, 82(2):459–466, jan 2016.
- [9] C. P. A. Skarp, O. Akinrinade, A. J. E. Nilsson, P. Ellström, S. Myllykangas, and H. Rautealin. Comparative genomics and genome biology of invasive Campylobacter jejuni. *Scientific Reports*, 5(1):17300, dec 2015.
- [10] Enis Afgan, Dannon Baker, Bérénice Batut, Marius Van Den Beek, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Björn A Grüning, et al. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic acids research*, 46(W1):W537–W544, 2018.
- [11] Clare Sloggett, Nuwan Goonasekera, and Enis Afgan. BioBlend: automating pipeline analyses within Galaxy and CloudMan. *Bioinformatics (Oxford, England)*, 29(13):1685–6, jul 2013.
- [12] Galaxy. <https://usegalaxy.org/>.

- [13] Docker - Build, Ship, and Run Any App, Anywhere. <https://www.docker.com/>.
- [14] Sergio Chico. :whale: Docker with Galaxy for Bioinformatic Bacterial Sequencing Workflows: Serux/docker-galaxy-BioInfWorkflow. <https://github.com/Serux/docker-galaxy-BioInfWorkflow>, June 2018. original-date: 2018-05-17T01:10:05Z.
- [15] Galaxy API. <https://galaxyproject.org/develop/api/>.
- [16] Björn Grüning. :whale::bar_chart::books: Docker Images tracking the stable Galaxy releases.: bgruening/docker-galaxy-stable. <https://github.com/bgruening/docker-galaxy-stable>, October 2018. original-date: 2014-08-12T13:26:14Z.
- [17] Elaine R. Mardis. Next-generation dna sequencing methods. *Annual Review of Genomics and Human Genetics*, 9(1):387–402, 2008. PMID: 18576944.
- [18] Ravi K. Patel and Mukesh Jain. Ngs qc toolkit: A toolkit for quality control of next generation sequencing data. *PLOS ONE*, 7(2):1–7, 02 2012.
- [19] Babraham Bioinformatics - FastQC A Quality Control tool for High Throughput Sequence Data.
- [20] Robert Schmieder and Robert Edwards. Quality control and preprocessing of metagenomic datasets. *Bioinformatics (Oxford, England)*, 27(6):863–864, March 2011. PMID: 21278185.
- [21] Wikipedia contributors. Sequence assembly — Wikipedia, the free encyclopedia, 2019. [Online; accessed 5-February-2019].
- [22] Sergey Nurk, Anton Bankevich, Dmitry Antipov, Alexey Gurevich, Anton Korobeynikov, Alla Lapidus, Andrey Prjibelsky, Alexey Pyshkin, Alexander Sirotkin, Yakov Sirotkin, Ramunas Stepanauskas, Jeffrey McLean, Roger Lasken, Scott R. Clingenpeel, Tanja Woyke, Glenn Tesler, Max A. Alekseyev, and Pavel A. Pevzner. Assembling Genomes and Mini-metagenomes from Highly Chimeric Reads. pages 158–170. Springer, Berlin, Heidelberg, 2013.
- [23] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*, 18(5):821–9, may 2008.
- [24] Phillip E C Compeau, Pavel A Pevzner, and Glenn Tesler. How to apply de Bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987–91, nov 2011.
- [25] Emily J. Richardson and Mick Watson. The automatic annotation of bacterial genomes. *Briefings in Bioinformatics*, 14(1):1–12, 02 2012.
- [26] Paul Stothard and David S Wishart. Automated bacterial genome analysis and annotation. *Current Opinion in Microbiology*, 9(5):505–510, oct 2006.
- [27] Doug Hyatt, Gwo-Liang Chen, Philip F LoCascio, Miriam L Land, Frank W Larimer, and Loren J Hauser. Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinformatics*, 11(1):119, dec 2010.
- [28] Karin Lagesen, Peter Hallin, Einar Andreas Rødland, Hans-Henrik Stærfeldt, Torbjørn Rognes, and David W. Ussery. RNAmmer: consistent and rapid annotation of ribosomal RNA genes. *Nucleic Acids Research*, 35(9):3100–3108, may 2007.
- [29] D. Laslett and Bjorn Canback. ARAGORN, a program to detect tRNA genes and tmRNA genes in nucleotide sequences. *Nucleic Acids Research*, 32(1):11–16, jan 2004.

- [30] Thomas Nordahl Petersen, Søren Brunak, Gunnar von Heijne, and Henrik Nielsen. SignalP 4.0: discriminating signal peptides from transmembrane regions. *Nature Methods*, 8(10):785–786, oct 2011.
- [31] D. L. Kolbe and S. R. Eddy. Fast filtering for RNA homology search. *Bioinformatics*, 27(22):3102–3109, nov 2011.
- [32] Harold C. Neu. The crisis in antibiotic resistance. *Science*, 257(5073):1064–1073, 1992.
- [33] Kirk E. Smith, John M. Besser, Craig W. Hedberg, Fe T. Leano, Jeffrey B. Bender, Julie H. Wicklund, Brian P. Johnson, Kristine A. Moore, and Michael T. Osterholm. Quinolone-resistant campylobacter jejuni infections in minnesota, 1992–1998. *New England Journal of Medicine*, 340(20):1525–1532, 1999. PMID: 10332013.
- [34] Bo Liu and Mihai Pop. ARDB—Antibiotic Resistance Genes Database. *Nucleic Acids Research*, 37(suppl_1) : D443 – –D447, 102008.
- [35] Torsten Seemann. Mass screening of contigs for antimicrobial and virulence genes: tseemann/a-bricate, January 2019. original-date: 2014-07-17T00:59:35Z.
- [36] Duccio Medini, Claudio Donati, Hervé Tettelin, Vega Massignani, and Rino Rappuoli. The microbial pan-genome. *Current Opinion in Genetics Development*, 15(6):589 – 594, 2005. Genomes and evolution.
- [37] Hervé Tettelin, Vega Massignani, Michael J. Cieslewicz, Claudio Donati, Duccio Medini, Naomi L. Ward, Samuel V. Angiuoli, Jonathan Crabtree, Amanda L. Jones, A. Scott Durkin, Robert T. DeBoy, Tanja M. Davidsen, Marirosa Mora, Maria Scarselli, Immaculada Margarit y Ros, Jeremy D. Peterson, Christopher R. Hauser, Jaideep P. Sundaram, William C. Nelson, Ramana Madupu, Lauren M. Brinkac, Robert J. Dodson, Mary J. Rosovitz, Steven A. Sullivan, Sean C. Daugherty, Daniel H. Haft, Jeremy Selengut, Michelle L. Gwinn, Liwei Zhou, Nikhat Zafar, Hoda Khouri, Diana Radune, George Dimitrov, Kisha Watkins, Kevin J. B. O’Connor, Shannon Smith, Teresa R. Utterback, Owen White, Craig E. Rubens, Guido Grandi, Lawrence C. Madoff, Dennis L. Kasper, John L. Telford, Michael R. Wessels, Rino Rappuoli, and Claire M. Fraser. Genome analysis of multiple pathogenic isolates of streptococcus agalactiae: Implications for the microbial “pan-genome”. *Proceedings of the National Academy of Sciences*, 102(39):13950–13955, 2005.
- [38] Ngan Nguyen, Glenn Hickey, Daniel R Zerbino, Brian Raney, Dent Earl, Joel Armstrong, W James Kent, David Haussler, and Benedict Paten. Building a pan-genome reference for a population. *Journal of computational biology : a journal of computational molecular cell biology*, 22(5):387–401, may 2015.
- [39] Andrew J. Page, Carla A. Cummins, Martin Hunt, Vanessa K. Wong, Sandra Reuter, Matthew T.G. Holden, Maria Fookes, Daniel Falush, Jacqueline A. Keane, and Julian Parkhill. Roary: rapid large-scale prokaryote pan genome analysis. *Bioinformatics*, 31(22):3691–3693, nov 2015.
- [40] Tom Madden. Chapter 16. The BLAST Sequence Analysis Tool. Technical report.
- [41] V. Čurčín, M. Ghanem, Y. Guo, M. Köhler, A. Rowe, J. Syed, and P. Wendel. Discovery net: Towards a grid of knowledge discovery. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’02, pages 658–663, New York, NY, USA, 2002. ACM.
- [42] Kristian Ovaska, Marko Laakso, Saija Haapa-Paananen, Riku Louhimo, Ping Chen, Viljami Aittomäki, Erkkä Valo, Javier Núñez-Fontarnau, Ville Rantanen, Sirkku Karinen, Kari Nousiainen, Anna-Maria Lahesmaa-Korpinen, Minna Miettinen, Lilli Saarinen, Pekka Kohonen, Jianmin Wu,

- Jukka Westermarck, and Sampsa Hautaniemi. Large-scale data integration framework provides a comprehensive view on glioblastoma multiforme. *Genome Medicine*, 2(9):65, sep 2010.
- [43] Yuanyuan Song, Guangchao Sui, Li Yao, and Heming Wang. BioQueue: a novel pipeline framework to accelerate bioinformatics analysis. *Bioinformatics*, 33(20):3286–3288, 06 2017.
- [44] JÖRGEN BRANDT, WOLFGANG REISIG, and ULF LESER. Computation semantics of the functional scientific workflow language cuneiform. *Journal of Functional Programming*, 27:e22, 2017.
- [45] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi, and Carole Goble. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 05 2013.
- [46] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. *The Triana Workflow Environment: Architecture and Applications*, pages 320–339. Springer London, London, 2007.
- [47] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, August 2006.
- [48] Wil M. P. Aalst and Arthur Ter. Yawl: Yet another workflow language. *Information Systems*, 30:245–275, 06 2004.
- [49] V. Curcin and M. Ghanem. Scientific workflow systems - can one size fit all? In *2008 Cairo International Biomedical Engineering Conference*, pages 1–9, Dec 2008.
- [50] Mohamed Abouelhoda, Shady Alaa, and Moustafa Ghanem. Meta-workflows: Pattern-based interoperability between galaxy and taverna. In *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*, Wands '10, pages 2:1–2:8, New York, NY, USA, 2010. ACM.
- [51] Tommi Henrik Nyrönen, Jarno Laitinen, Olli Tourunen, Danny Sternkopf, Risto Laurikainen, Per Öster, Pekka T. Lehtovuori, Timo A. Miettinen, Tomi Simonen, Teemu Perheentupa, Imre Västriik, Olli Kallioniemi, Andrew Lyall, and Janet Thornton. Delivering ict infrastructure for biomedical research. In *Proceedings of the WICSA/ECSA 2012 Companion Volume*, WICSA/ECSA '12, pages 37–44, New York, NY, USA, 2012. ACM.
- [52] Jeremy Leipzig. A review of bioinformatic pipeline frameworks. *Briefings in Bioinformatics*, 18(3):530–536, 03 2016.
- [53] Frank M. Aarestrup, Henrik Hasman, Martin Vestergaard, Ea Zankari, Mette Voldby Larsen, Ole Lund, Salvatore Cosentino, and Simon Rasmussen. Identification of acquired antimicrobial resistance genes. *Journal of Antimicrobial Chemotherapy*, 67(11):2640–2644, 07 2012.
- [54] Amogelang R. Raphenya, Andrew C. Pawlowski, Arjun N. Sharma, Baofeng Jia, Biren M. Dave, Brian Alcock, Briony A. Lago, Daim Sardar, Erin L. Westman, Gerard D. Wright, Kara K. Tsang, Nicholas Waglechner, Peiyao Guo, Sachin Doshi, Sheldon Pereira, Timothy A. Johnson, Andrew G. McArthur, Fiona S.L. Brinkman, Mélanie Courtot, Raymond Lo, Jonathan G. Frye, Laura E. Williams, and Tariq Elsayegh. CARD 2017: expansion and model-centric curation of the comprehensive antibiotic resistance database. *Nucleic Acids Research*, 45(D1):D566–D573, 10 2016.

- [55] Sushim Kumar Gupta, Babu Roshan Padmanabhan, Seydina M. Diene, Rafael Lopez-Rojas, Marie Kempf, Luce Landraud, and Jean-Marc Rolain. Arg-annot, a new bioinformatic tool to discover antibiotic resistance genes in bacterial genomes. *Antimicrobial Agents and Chemotherapy*, 58(1):212–220, 2014.
- [56] Alessandra Carattoli, Ea Zankari, Aurora García-Fernández, Mette Voldby Larsen, Ole Lund, Laura Villa, Frank Møller Aarestrup, and Henrik Hasman. In silico detection and typing of plasmids using plasmidfinder and plasmid multilocus sequence typing. *Antimicrobial Agents and Chemotherapy*, 58(7):3895–3903, 2014.
- [57] Bo Liu, Dandan Zheng, Qi Jin, Lihong Chen, and Jian Yang. VFDB 2019: a comparative pathogenomic platform with an interactive web interface. *Nucleic Acids Research*, 47(D1):D687–D692, 11 2018.



Manual de utilización

En este apartado se detallarán todos los aspectos necesarios para la ejecución y el correcto funcionamiento del proyecto. Se guiará al usuario a través de todo el proceso, desde la instalación del software necesario hasta todas las utilidades que tienen que ver con el workflow, pasando por una breve guía de uso de *Galaxy* orientada a nuestro caso.

A.1. Requisitos

Requisitos hardware

15 GB de espacio libre en disco
16 GB de memoria RAM
procesador de 4 núcleos a 2.5 Ghz (recomendado)

Requisitos software

Sistema *Ubuntu* (recomendada versión cercana a la 18.04)
Docker 18.09.1 o superior

Cuadro A.1: Requisitos del sistema

En primer lugar, debido a la complejidad de las ejecuciones y al gran tamaño de la imagen *Galaxy* y de los datos de entrada de las ejecuciones, se requiere de unos componentes hardware relativamente exigentes A.1. Una de las herramientas utilizadas en el workflow, *SPAdes*, necesita cargar gran cantidad de datos en memoria RAM, por lo que se recomienda una capacidad de 16 GB. Se requiere, también, de 20 GB de espacio libre en disco para almacenar cómodamente la imagen *Galaxy* y los datos que se utilicen. El procesador es un aspecto menos restrictivo, pero las ejecuciones serán más lentas con peores procesadores. Como orientación, en un procesador de 8 núcleos a 3.6 Ghz, una ejecución con unos 10 GB de datos de entrada, tarda aproximadamente 3 horas.

En cuanto al software A.1, el primer requisito es un sistema operativo Linux. Preferiblemente una versión cercana a *Ubuntu 18.04.1*, la utilizada para el desarrollo.

También será necesario disponer de *Docker* en el equipo. Si no está instalado, en el apartado de «Instalación de *Docker*» a continuación, se detallará el proceso a realizar.

A.2. Utilización

A.2.1. Instalación de Docker

Partiendo de un sistema *Ubuntu*, lo primero que deberemos instalar será *Docker*. Para ello, abriremos la terminal de *linux* y seguiremos el tutorial de su web oficial¹. El primer paso será actualizar la lista de paquetes disponibles para la instalación. Para ello, escribiremos en la terminal:

```
$ sudo apt-get update
```

A continuación, instalaremos varios paquetes que serán necesarios más adelante.

```
$ sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
gnupg-agent \
software-properties-common
```

Ahora añadiremos la clave para el repositorio oficial de *Docker* al sistema.

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
sudo apt-key add -
```

Necesitaremos añadir el repositorio *Docker*.

```
$ sudo add-apt-repository \
‘‘deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable’’
```

Antes de finalizar, actualizaremos de nuevo la lista de paquetes disponibles.

```
$ sudo apt-get update
```

Finalmente, instalaremos *Docker*.

```
$ sudo apt install docker-ce
```

Dado que realizaremos frecuentemente ejecuciones con el comando «docker», añadiremos nuestro usuario al grupo de *Docker* para evitar introducir la clave en cada llamada.

```
$ sudo usermod -aG docker ${USER}
```

Para hacerlo efectivo, reiniciaremos nuestra sesión de usuario.

```
$ su - ${USER}
```

De esta manera, podremos dar por finalizada la instalación de *Docker*.

A.2.2. Descarga de la imagen *Galaxy*

En la carpeta «exes» del proyecto, encontraremos todo lo necesario para interactuar con *Docker* y *Galaxy*. En este caso, para la instalación de la imagen, utilizaremos el script «run.sh»

¹<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

que se encargará de buscar la imagen *Galaxy* en nuestro ordenador, descargarla, en caso de que no la encuentre, y desplegarla en un contenedor *Docker*. Para ello, desde la terminal nos situaremos en el directorio «exes» y escribiremos:

```
$ ./run.sh
```

Tras la ejecución de ese comando, en la terminal debería mostrarse un registro del funcionamiento de *Galaxy*, que podremos cerrar cuando deseemos. Si se quiere comprobar el estado de los contenedores *Docker*, se ejecutará

```
$ docker ps -a
```

Dado que *Galaxy* ya está funcionando, podremos acceder simplemente abriendo cualquier navegador web y escribiendo en la barra de navegación «<http://localhost:8080/>».

A.2.3. Control del contenedor *Docker*

En el mismo directorio «exes» en el que nos situábamos en el paso anterior encontramos varias utilidades para el control de *Docker*.

run.sh despliega la imagen de *Galaxy* en un contenedor *Docker*. Sobrescribirá la imagen *Galaxy* actual si existe alguna.

stop.sh detiene el contenedor *Docker*.

start.sh arranca el contenedor si está detenido. Para iniciar *Galaxy*, a excepción de la primera vez, deberá usarse este script.

remove.sh elimina el contenedor.

purge.sh elimina ficheros temporales de *Docker* que pueden causar problemas de espacio.

A.2.4. Funcionamiento del workflow en *Galaxy*

Aunque existe una utilidad para simplificar el proceso de uso de workflow, explicada en la sección A.2.5, a continuación analizaremos el uso de *Galaxy* de la manera estándar.

Desde un navegador web cualquiera, si accedemos a la dirección «<http://localhost:8080/>», veremos la página de inicio de *Galaxy*. En ella, tendremos la posibilidad de identificarnos mediante la pestaña «Login or Register». Existe un usuario preestablecido con correo «admin@galaxy.org» y contraseña «admin». Una vez identificados como administrador tendremos la posibilidad de acceder a los workflows almacenados desde la pestaña «Workflow». En ella, veremos un listado con todas las opciones posibles, la que nos interesa es «CJ_Workflow». Si desplegamos utilizando la flecha a la derecha del nombre, podremos acceder a las acciones con las que interactuaremos con este workflow. La primera de ellas es el modo edición, a través de «edit».

En la vista de edición A.1 encontraremos un diagrama formado por las herramientas y uniones que indican las interacciones entre los ficheros de cada una de ellas. Al comienzo del proceso veremos dos cajas con el título «Input dataset collection». Desde este paso se introducirán los datos de entrada al workflow en forma de una estructura denominada colección. Para ver cómo crear esa estructura saldremos un momento de la pestaña workflow, haciendo clic en la opción «Analyze data» de la parte superior. A continuación, en el listado de categorías de herramientas que podemos ver en la parte izquierda, abriremos «Get Data» seguido de «Upload File». Veremos como se despliega una ventana nueva en la que existen tres opciones de estructuras: «regular», «composite» y «collection». Deberemos seleccionar esta tercera opción y «Choose local files» para subir todos los ficheros del primer sentido de las secuencias. Es importante que los subamos en el mismo orden en el que lo vayamos a hacer con la colección de las secuencias en el sentido

Download from web or upload from disk

Regular Composite **Collection**

You added 6 file(s) to the queue. Add more files or click 'Start' to proceed.

Name	Size	Status
200_TCCTGAGC-AGAGTAGA_L004_R1_001.fastq	536.5 MB	<div></div>
443_CTCTCTAC-AAGGAGTA_L004_R1_001.fastq	944.7 MB	<div></div>
590_CGAGGCTG-GCGTAAGA_L004_R1_001.fastq	1.4 GB	<div></div>
629_GCTACGCT-GCGTAAGA_L004_R1_001.fastq	698.1 MB	<div></div>
680_GCTACGCT-TATCCTCT_L004_R1_001.fastq	265.8 MB	<div></div>
681_GCTACGCT-AGAGTAGA_L004_R1_001.fastq	202.9 MB	<div></div>

Collection Type: File Type: Genome (set all):

Figura A.1: Interfaz de subida de los conjuntos de datos

opuesto. Una vez seleccionados deberemos pulsar «Start» y, cuando las barras de progreso estén completas, «build». Esto nos llevará a otra ventana en la que seleccionaremos el nombre de la colección con las lecturas en este sentido. Para finalizar, pulsaremos «create list». Con este proceso tendremos una de las colecciones, y deberemos repetirlo para la colección con las lecturas en el sentido contrario. Cuando se hayan completado todos estos pasos, podremos volver a la vista de edición del workflow para seguir entendiendo su funcionamiento.

Una vez visto el proceso de introducción de datos de entrada, encontramos las herramientas del workflow, cuyo funcionamiento se explica con más profundidad en el Capítulo 3: «Sistema, Diseño y Desarrollo». Para modificar los parámetros de cualquiera de las herramientas, podremos seleccionarla y veremos una serie de opciones en la parte derecha de la pantalla, como vemos en la imagen A.2. Todos los parámetros modificados ya están guardados en el workflow, pero si se desea cambiar alguno de ellos, se podrá hacer desde esa interfaz. En caso de que se necesite añadir alguna herramienta extra, en el listado de la izquierda se encuentran ordenadas por categorías y con un clic serán añadidas, a falta de enlazar sus ficheros de entrada y de salida con las demás herramientas. Otro aspecto relevante a destacar es que, por defecto, en *Galaxy* los datos se establecen como ocultos a no ser que se marquen con el asterisco que podemos ver a la derecha de los nombres de cada conjunto de datos de salida. Si hacemos clic en ese símbolo, los ficheros ya no aparecerán como ocultos y serán visibles en nuestro historial.

Los historiales son el sistema con el que *Galaxy* ordena los conjuntos de datos del usuario. Desde la página de inicio «Analyze Data» podremos consultar el historial activo en este momento en la parte derecha de la pantalla, pero este no tiene por qué ser el único que existe. Para consultar todos los historiales, desde la propia columna del historial activo, en la parte superior derecha, veremos tres símbolos con las opciones «Refresh history», «History options» y «View all histories». Desde esta última accederemos a una nueva pantalla en la que podremos consultar todos los historiales. Es posible que queramos ver conjuntos de datos ocultos de un historial, para ello, podremos activar la opción «show hidden datasets» situada debajo del nombre del historial.

Una vez comprendido el funcionamiento del workflow y los historiales, podremos ejecutar el workflow. Para ello, accederemos de nuevo a la pestaña «Workflow» y desde el desplegable de

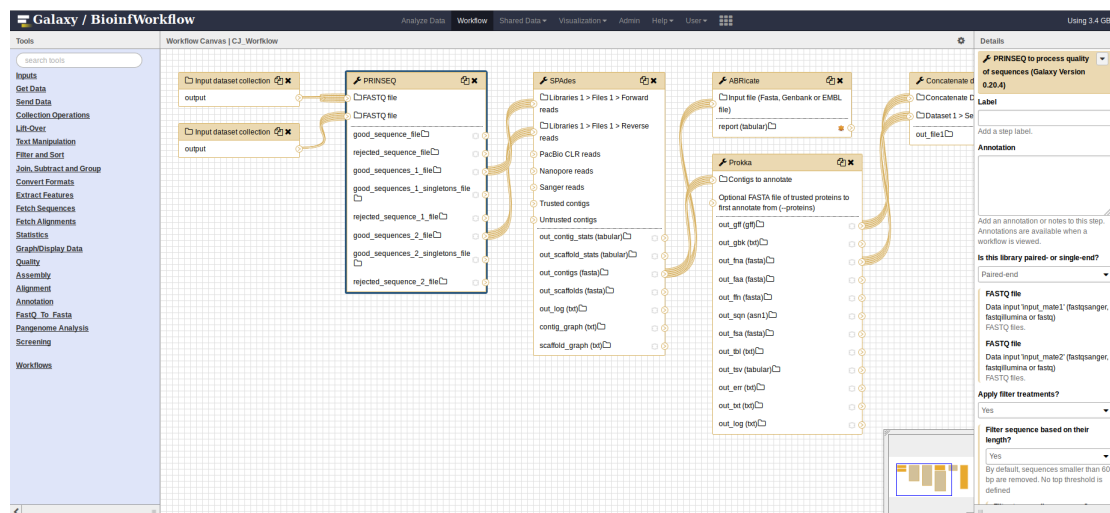


Figura A.2: Interfaz de edición del workflow

«CJ_Workflow» seleccionaremos la opción «run». Esto nos situará en una nueva pantalla en la que, de nuevo, podremos modificar todos los parámetros de las herramientas del workflow. Sin embargo, es algo opcional en este momento. El punto principal es la introducción de los datos de entrada, que se seleccionarán desde dos desplegables (uno para cada sentido de las lecturas) en los apartados «Input dataset collection». Si hemos creado las colecciones como se ha explicado previamente, estas aparecerán en el desplegable y solo tendremos que seleccionarlas. El paso siguiente, teniendo en cuenta que no queremos modificar ningún parámetro, será la ejecución del workflow usando el botón «Run Workflow» de la parte superior derecha.

A través del uso de los historiales, tendremos la opción de consultar el estado de las ejecuciones de cada una de las tareas del workflow. Teniendo en cuenta los pasos para consultar los historiales explicados anteriormente, podremos ver si una tarea se está ejecutando, en cola, pausada o si ha causado algún error. Hay que tener en cuenta que es una ejecución computacionalmente muy costosa, por lo que puede llevar varias horas completarla. Por ello, es importante seguir el progreso a través del historial con el objetivo de asegurarnos de que no ha habido ningún error durante el proceso. Cuando todas las tareas se muestren en color verde, el workflow habrá completado su ejecución y podremos descargar los conjuntos de datos resultantes que nos interesen haciendo clic en ellos y seleccionando el icono con la opción «download».

A.2.5. Utilidad de ejecución simplificada

Para facilitar el proceso explicado anteriormente, se ha desarrollado una utilidad que automatiza todos los pasos descritos. En el directorio «exes» del proyecto, encontramos un ejecutable llamado «GalaxyWorkflow». Este programa es capaz de cargar los datos de entrada, crear las colecciones, ejecutar el workflow y descargar los resultados en una carpeta local sin necesidad de que el usuario interactúe con *Galaxy*.

Es necesario conocer algunos comandos para navegar y realizar operaciones básicas con la terminal. En este caso solo veremos los comandos «ls», «cd» y «cp», pero es muy recomendable consultar la guía de *Ubuntu* que incluye otros comandos básicos que pueden ser útiles ². Supongamos que nos encontramos en el directorio de nuestro usuario, la ruta por defecto en la que se abrirá nuestro terminal. Si escribimos «ls» obtendremos un listado de los directorios y ficheros

²<https://help.ubuntu.com/community/UsingTheTerminal>

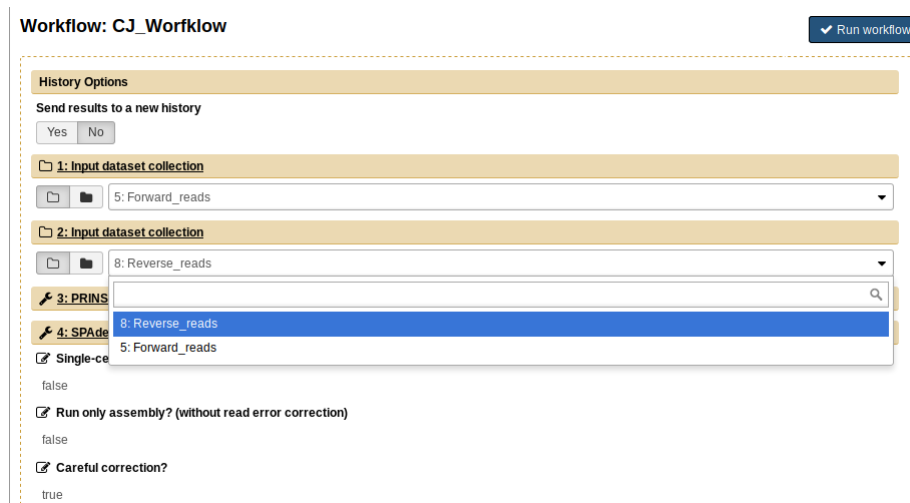


Figura A.3: Selección de las colecciones de entrada del workflow

que nos podemos encontrar A.4. Tanto para este comando como para cualquier otro, podremos obtener más información sobre su uso añadiendo «man» antes del comando, es decir: «man ls». Si ahora nuestro objetivo es movernos a uno de esos directorios que aparecen en el listado, ejecutaremos «cd» y el directorio al que queremos movernos. Por ejemplo, si fuese el escritorio: «cd Desktop». En caso de que necesitemos volver al directorio padre del que nos encontramos ahora mismo usaremos «cd ..».

En principio, se pueden copiar los ficheros directamente utilizando la interfaz gráfica de *Ubuntu* al directorio que deseemos, pero si se quiere usar la terminal, el comando necesario para copiar será «cp» y para mover «mv», que funcionan de la misma manera. Un ejemplo para copiar un fichero «cepa200.fastq» al escritorio sería

```
$ cp cepa200.fastq ./Desktop/cepa200.fastq
```

y para moverlo

```
$ mv cepa200.fastq ./Desktop/.
```

Si se quisiese hacer lo mismo para todos los ficheros de determinada extensión, por ejemplo, fastq:

```
$ cp *.fastq ./Desktop/
```

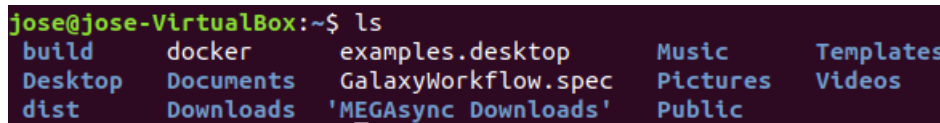
Por último, veamos como arrancar un ejecutable. Si suponemos que en el escritorio existe un fichero «run.sh», deberemos escribir la ruta hasta ese fichero precedida de un punto. Por ejemplo, si estuviésemos en el directorio comentado anteriormente, ejecutaríamos «run.sh» escribiendo

```
$ ./Desktop/run.sh
```

Si nos encontramos en el mismo directorio que el ejecutable, la ruta hasta ese fichero será simplemente un punto:

```
$ ./run.sh
```

Pasemos ahora a la utilización del propio workflow. En primer lugar, se requiere que la imagen *Galaxy* esté funcionando y podamos acceder a ella desde el navegador en la dirección «<http://localhost:8080/>». Es conveniente realizar esta comprobación antes de ejecutar la



```
jose@jose-VirtualBox:~$ ls
build      docker     examples.desktop  Music      Templates
Desktop    Documents  GalaxyWorkflow.spec Pictures    Videos
dist       Downloads  'MEGAsync Downloads' Public
```

Figura A.4: Ejemplo de un comando ls

aplicación. Si se desea, el usuario puede identificarse como administrador antes de comenzar, para poder consultar lo que está ocurriendo con *Galaxy* mientras la utilidad está funcionando. Es importante dejar claro que interaccionará con dos nuevos historiales, así que para comprobar el funcionamiento, si es que se desea, se deberá hacer desde la vista de todos los historiales.

El proceso para ejecutarlo es muy sencillo, solamente deberemos abrir la terminal de *Ubuntu* y situarnos en el directorio «exes» del proyecto. Una vez ahí, veremos que existen dos carpetas «Forward» y «Reverse». En ellas deberemos introducir los ficheros *.fastq* de las lecturas que deseamos analizar. Los ficheros de cada sentido de lectura deben dividirse en las dos carpetas. Es necesario que ambas tengan el mismo número de ficheros y muy recomendable que los nombres de los dos sentidos de una lectura comiencen por el mismo identificador único. Por ejemplo, dos nombres de una pareja podrían ser «lectura1_forward» y «lectura1_reverse». Cuando contemos con todos los ficheros situados en los directorios correspondientes, escribiremos el siguiente comando.

```
$ ./GalaxyWorkflow
```

Con ello, comenzará la ejecución y cada minuto recibiremos una actualización del estado de las tareas en marcha, pudiendo comprobar si ha habido algún error y ser capaz de detener la ejecución desde *Docker*. Si esto sucediera, la manera más fácil sería detener la imagen *Galaxy* con el comando:

```
$ ./stop.sh
```

O, incluso, resetear la imagen de cero para eliminar todos los ficheros que ya no nos interesan, con la combinación de comandos

```
$ ./stop.sh
$ ./remove.sh
$ ./run.sh
```

Normalmente, esto no será necesario y, tras varias horas de ejecución, obtendremos los resultados en un directorio «ResultsHistory[+timestamp]» en el que serán ordenados en carpetas dependiendo de la herramienta de la que provengan.

A.2.6. Tratamiento de datos obtenidos

Con la utilidad del apartado anterior, obtenemos un gran número de ficheros independientes. El objetivo de esta herramienta es agrupar los resultados de *ABRicate* y *Roary* en un solo documento *.xls* que podamos abrir como una hoja de cálculo. Para ello, deberemos situarnos de nuevo en el directorio «exes». Nuestros datos de entrada estarán situados en una carpeta con un nombre que comience por «Results» y que contenga varias subcarpetas, cada una con el nombre de la herramienta de origen de los ficheros que contiene. Por ejemplo, dos carpetas: «roary» y «abricate» con los ficheros resultantes de cada una en su interior. Una vez tengamos esta estructura (que por defecto se cumple si se utiliza la herramienta de ejecución automática del workflow) podremos abrir la terminal en el directorio «exes» e introducir el comando

\$./OutputToXls

Esto generará un fichero con el mismo nombre que la carpeta que hemos indicado anteriormente pero con extensión *.xls* en el que cada hoja del documento corresponderá a una de las herramientas.



Manual del programador

Este capítulo tiene como objetivo facilitar el acercamiento al proyecto por parte de un programador externo. Siguiendo los apartados desarrollados a continuación, tendrá la información necesaria para realizar modificaciones o ampliar el trabajo ya realizado.

B.1. Requisitos hardware recomendados

Debido a que se realizan ejecuciones muy costosas en local, los requisitos recomendados para este proyecto son bastante exigentes. En primer lugar, dado que las imágenes *Galaxy* son muy pesadas (del orden de los 10 GB) y los datos generados por el workflow también son considerables, se recomienda un espacio libre en disco de unos 20 GB para una utilización cómoda de la aplicación. En cuanto a la memoria RAM, se recomienda utilizar una capacidad mayor que el tamaño de los datos de entrada. Esto se debe a la forma de ejecución de algunas herramientas del workflow, que necesitan cargar todas las secuencias a la vez en memoria. En nuestro caso se han utilizado 16 GB de RAM para un conjunto de datos de entrada de 10 GB. El caso del procesador no es tan limitante como el anterior, sin embargo, definirá el tiempo total de ejecución. El caso de prueba mencionado anteriormente se ha ejecutado con un procesador Intel i7-9700K, con un tiempo de ejecución de unas 3 horas. Por lo tanto, cualquier procesador de un sistema de 64 bits será capaz de realizar la ejecución.

B.2. Requisitos software necesarios

El proyecto está enfocado a ser ejecutado en un sistema *Ubuntu*, en concreto, se ha utilizado la versión 18.04.1. Sin embargo, desde la perspectiva de un desarrollador, con unos cambios menores puede llegar a adaptarse a cualquier sistema sin demasiado esfuerzo.

B.2.1. *Docker*

Docker es un sistema de código abierto basado en contenedores que permiten el despliegue de aplicaciones dentro de sí mismos. Los contenedores ofrecen la posibilidad de empaquetar todo

el software necesario para ejecutar nuestra aplicación, ofreciendo un modo de virtualización mucho más aislado y ligero que el uso de máquinas virtuales. Además, se mantiene la seguridad de que nuestra aplicación será utilizable en cualquier sistema en el que se pueda realizar esta virtualización.

Durante el desarrollo del proyecto se ha utilizado la versión 18.09.1 de *Docker*, por lo que se recomienda la instalación de una versión igual o superior a esta. El proceso para instalar *Docker* en *Ubuntu* no es complejo y en la propia web encontramos una guía de instalación ¹.

B.2.2. Python

Python es uno de los lenguajes de programación más utilizados en el mundo. Es un lenguaje interpretado, de alto nivel, de propósito general, de código abierto y multiplataforma. Su facilidad de uso y comprensión han hecho que su crecimiento haya sido notable en los últimos años, especialmente en investigación. Algunas de las funcionalidades del desarrollo se han creado a través de *Python* 3.7.2. Se recomienda la utilización de una versión superior a 3.6. Además de *Python* estándar, se han utilizado varias librerías:

***Bioblend* 0.12.0** es un paquete que posibilita la interacción con la API de *Galaxy*, permitiendo actuar sobre la instancia desplegada en *Docker*.

***Pandas* 0.24** permite la utilización de nuevas estructuras y métodos para el análisis de datos. Se utiliza en este caso para tareas simples, como gestionar ficheros de tipo *tabular*, *csv* y *xls*.

***Numpy* 1.15.4** es uno de los paquetes más utilizados de *Python*, facilitando gran variedad de funciones matemáticas. En el caso de *Numpy* es importante utilizar esta versión en concreto, ya que en este momento la versión actual (1.16.0) presenta incompatibilidades con otra de las librerías instaladas, *PyInstaller*.

***PyInstaller* 3.4** es una utilidad que permite generar ejecutables a partir del código *Python* para así evitar las instalaciones, tanto del propio *Python*, como de sus dependencias.

Las instalaciones de todas estas librerías pueden realizarse sin problema desde *pip* con el comando

```
$ pip install <<libreria_deseada>>.
```

En caso de que se necesite instalar una versión concreta, como en *Numpy*, se utilizará

```
$ pip install numpy==1.15.4
```

B.3. Estructura del proyecto

B.3.1. Directorio de ensamblado: «*build*»

Todos los componentes necesarios para construir la imagen *Docker* de *Galaxy* se encuentran en el directorio «*build*». El fichero más relevante es «*Dockerfile*», que contiene todas las directivas que indican cómo construir esta imagen. En concreto, realiza algunas variaciones sobre una imagen ya existente. Para realizar este montaje, será necesario el fichero «*mount.sh*», que se encargará de utilizar las instrucciones de «*Dockerfile*» para construir y desplegar la imagen en un contenedor *Docker*. Los dos ficheros «*install_workflows_wrapper.sh*» y «*my_tool_list.yml*» son utilidades para la instalación de workflows y herramientas de *Galaxy* respectivamente. Las herramientas indicadas se instalarán desde el «*tool shed*» de *Galaxy*, mientras que los workflows se encontrarán en el directorio «*workflows*».

¹<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

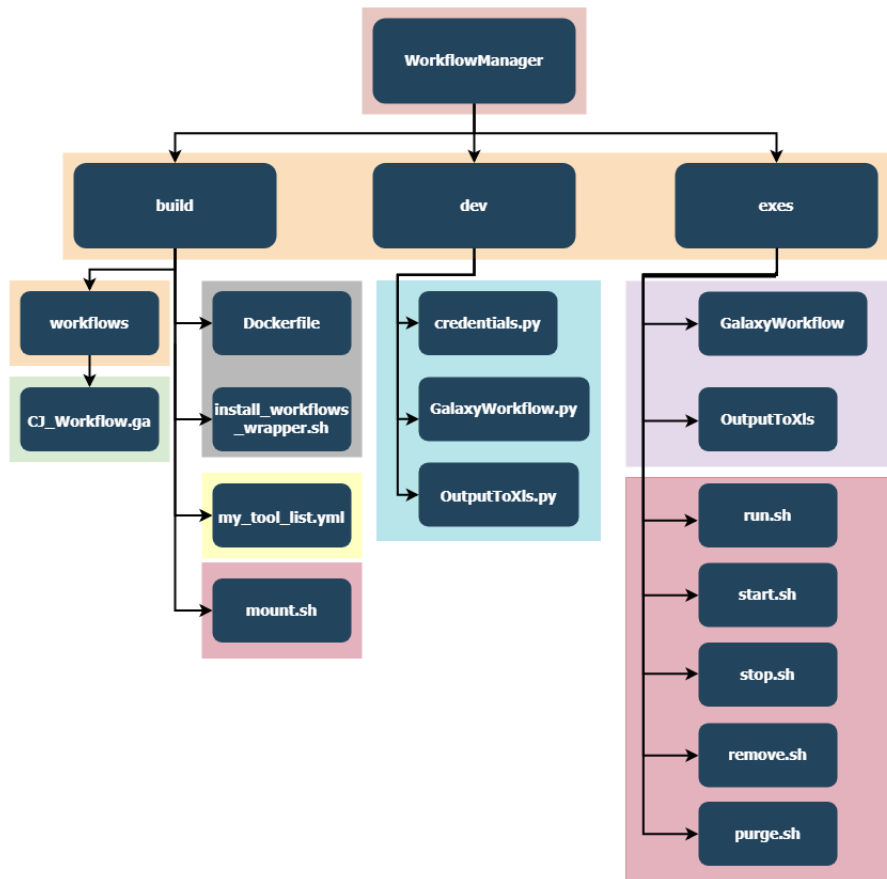


Figura B.1: Contenido del proyecto

B.3.2. Directorio de desarrollo: «dev»

Las utilidades desarrolladas en *Python* se almacenan en este directorio. Aquí encontramos dos ficheros principales, «GalaxyWorkflow.py» y «OutputToXls.py». El primero de ellos es el encargado de simplificar la utilización del workflow a los usuarios. Basándose en la API de *Galaxy* a través del uso de la librería *BioBlend*, se identifica en *Galaxy* utilizando las credenciales indicadas en el fichero «credentials.py». A continuación, genera nuevos historiales para almacenar los datos de entrada y los futuros datos de salida. Buscará dentro de los directorios «Forward» y «Reverse» los ficheros de las secuencias a analizar. Con ellos creará dos colecciones que servirán como entrada definitiva del workflow. Una vez ejecutado, el script se encargará de descargar los ficheros de salida de las tres últimas herramientas, que son los que en este caso se desean utilizar posteriormente.

Por su parte, el script «OutputToXls.py» se encargará, a través del uso de *Pandas*, de crear un solo fichero «.xls» a partir de las salidas generadas por las herramientas *Roary* y *ABRicate* para facilitar el tratamiento de los datos por parte del equipo que los va a utilizar, dado que están familiarizados con *Excel*.

B.3.3. Directorio de ejecución: «exes»

En este directorio encontraremos, en primer lugar, los ejecutables correspondientes a los ficheros *Python* del directorio «dev», con el objetivo de evitar la instalación de *Python* y sus dependencias al usuario. Además, se han desarrollado varios ficheros para simplificar el manejo de la imagen *Galaxy* de *Docker*, que simplemente contienen comandos para su instalación, arranque, parada y borrado. Se incluye un script «purge.sh» debido a que son frecuentes las acumulaciones de ficheros basura que acaban por ocupar demasiado espacio de almacenamiento. Este script es ejecutado desde los demás para evitar estos problemas.