Practica 4

Inti María Tidball 17612/3
Juan Pablo Sanchez Magariños 13238/3

1. Programar un agente para que periódicamente recorra una secuencia de computadoras y reporte al lugar de origen:
   a. El tiempo total del recorrido para recolectar la información.
   b. La carga de procesamiento de cada una de ellas.
   c. La cantidad de memoria total disponible.
   d. Los nombres de las computadoras.

   Comente la relación entre este posible estado del sistema distribuido y el estado que se obtendría implementando el algoritmo de instantánea.

Se crea una clase que se llama **`MultipleContainers`** que crea 10 containers de manera programática, se encarga de configurar una red de diez contenedores, simulando una serie de computadoras interconectadas en una red. El Main-Container actúa como el núcleo de esta red, mientras que los otros nueve contenedores representan nodos individuales en la red, que son los que se va a recorrer juntando información.
De la misma manera, el AgenteMovil tiene una lista de nombres de nodos con el fin de poder recorrerlos. Está programado para desplazarse secuencialmente entre los contenedores, su función es recolectar datos críticos de cada nodo. Estos datos incluyen la carga de procesamiento, la memoria disponible y los nombres de los nodos. La movilidad del agente se logra mediante el método doMove(), que le permite visitar todos los contenedores de manera eficiente y ordenada, gracias a una lógica modular basada en el índice del contenedor y el tamaño total de la lista de contenedores.
Se considera simplemente imprimir la información en cada contenedor, por consejo de una profesional en esta area, se decidió guardar la información usando una clase interna seralizable al Agente llamada **`ContainerInfo`**, Durante su recorrido, el AgenteMovil almacena la información recolectada en instancias de la clase interna **`ContainerInfo`**. Esta clase, diseñada para ser serializable, facilita la gestión eficiente de los datos recogidos en cada nodo. La función getInfoAsString() de esta clase permite la presentación estandarizada de la información recogida, que luego es reportada al Main-Container al finalizar el recorrido del agente.

Por último, se recorre la última lista imprimiendo toda la información solicitada y se imprime el tiempo de recorrido. El tiempo total que el agente toma para completar su recorrido se calcula utilizando marcas de tiempo generadas por System.currentTimeMillis() al comienzo y al final del ciclo de recorrido.

Nota: Para compilar y ejecutar todos los ejercicios de esta práctica es necesario obtener y colocar el `.jar` de JADE en el directorio `lib/`. No se incluye en la entrega como pedía la consigna. Para cada ejercicio, se tienen 3 scripts, para compilar, correr la gui y ejecutar los agentes, los cuales se explican en cada punto.

En este caso el código se encuentra en el directorio **`1/`**

Instrucciones para compilar (**`./1.compilacion.sh`**):

```
javac -classpath ../lib/jade.jar -d ./classes ./myexamples/AgenteMovil.java
./myexamples/MultipleContainers.java
```

Instrucciones para ejecutarlo:

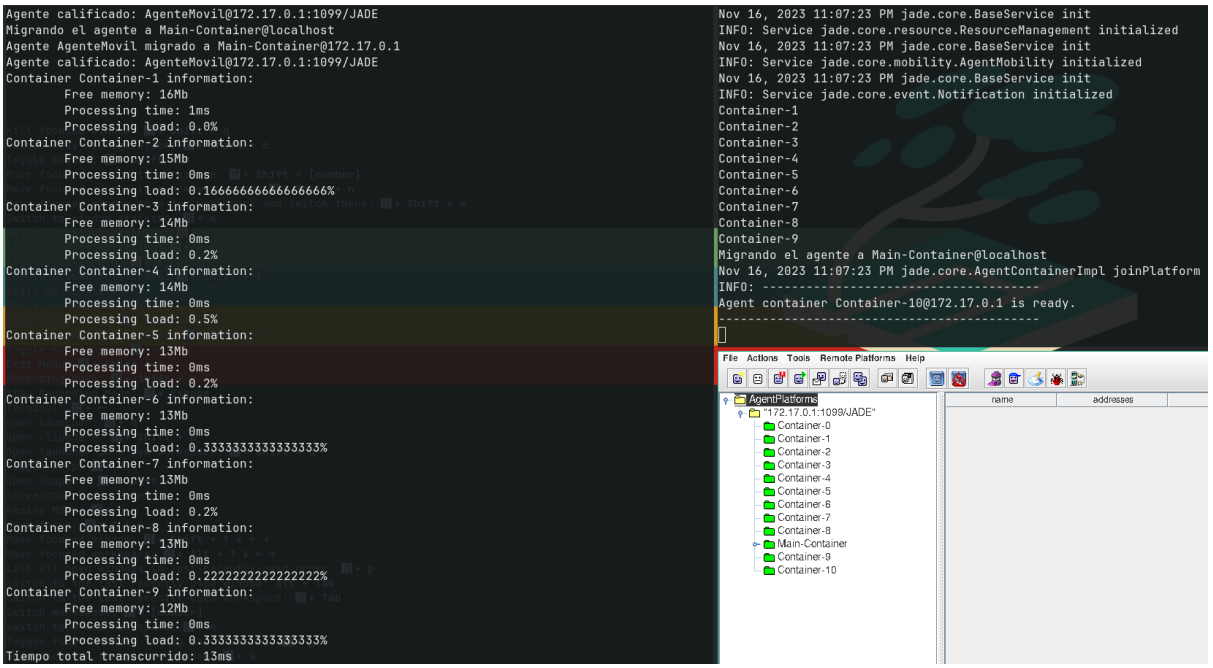
Terminal 1 (**`./2.gui.sh`**)

```
java -cp ../lib/jade.jar:classes myexamples.MultipleContainers
```

Terminal 2 (**`./3.agente.sh`**)

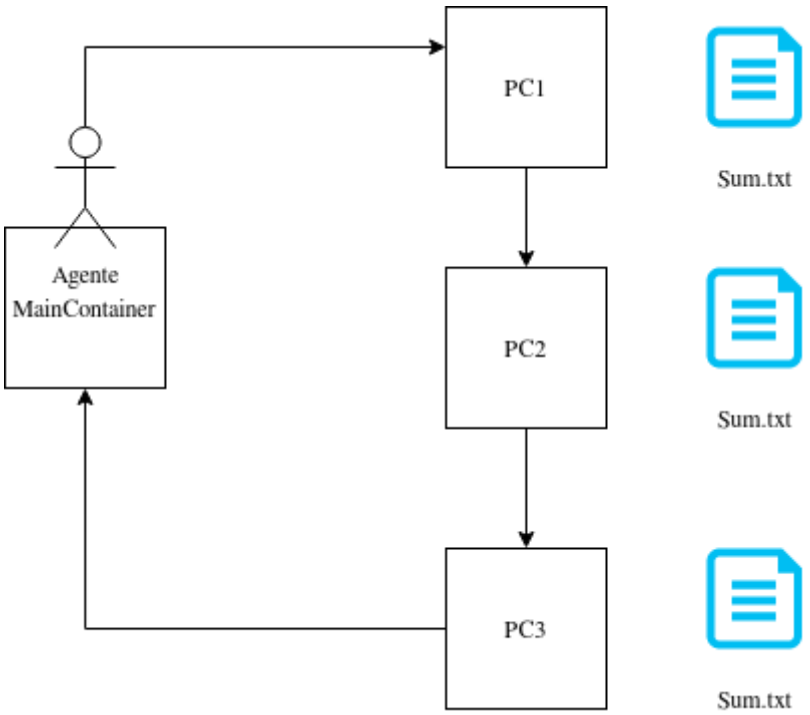
```
java -cp ../lib/jade.jar:classes jade.Boot -container -host localhost -port
1099 -agents "AgenteMovil:myexamples.AgenteMovil()"
```

Si se utilizara el algoritmo de instantánea, cada uno de los containers tendría un proceso y estos deberían sincronizarse entre ellos para ejecutarse al mismo tiempo.
Es por ello que la principal diferencia se daría en el tiempo de ejecución, el algoritmo de instantánea tendrá un tiempo menor al original.



2. Programe un agente para que calcule la suma de todos los números almacenados en un archivo de una computadora que se le pasa como parámetro. Comente cómo se haría lo mismo con una aplicación cliente/servidor. Comente qué pasaría si hubiera otros sitios con archivos que deben ser procesados de manera similar.

Se toma la decisión de resolver el problema con múltiples contenedores para poder también entender lo que se plantea en la pregunta final. Similarmente al ejercicio anterior, se crea una clase que se llama **`MultipleContainers`** que crea 3 containers de manera programática, se encarga de configurar una red de 3 contenedores, simulando una serie de computadoras interconectadas en una red.
Se crea adicionalmente una clase de tipo agente, **`AgenteMovil`**, que recibe por parámetro el *path* del archivo, y luego de migrar a la primera computadora, lee el archivo, realiza la suma de los números almacenados en dicho archivo, se guarda el resultado, migra a la próxima computadora, vuelve a leer el archivo, realiza la suma de los números almacenados en ese archivo, guarda la suma, y así en cada computadora hasta retornar al *Main-Container*, adonde suma todas las sumas almacenadas y muestra el resultado.



Instrucciones para compilar (**`./1.compilacion.sh`**):

```
javac -classpath ../lib/jade.jar -d classes myexamples/AgenteMovil.java
myexamples/MultipleContainers.java -Xdiags:verbose
```

Instrucciones para ejecutarlo

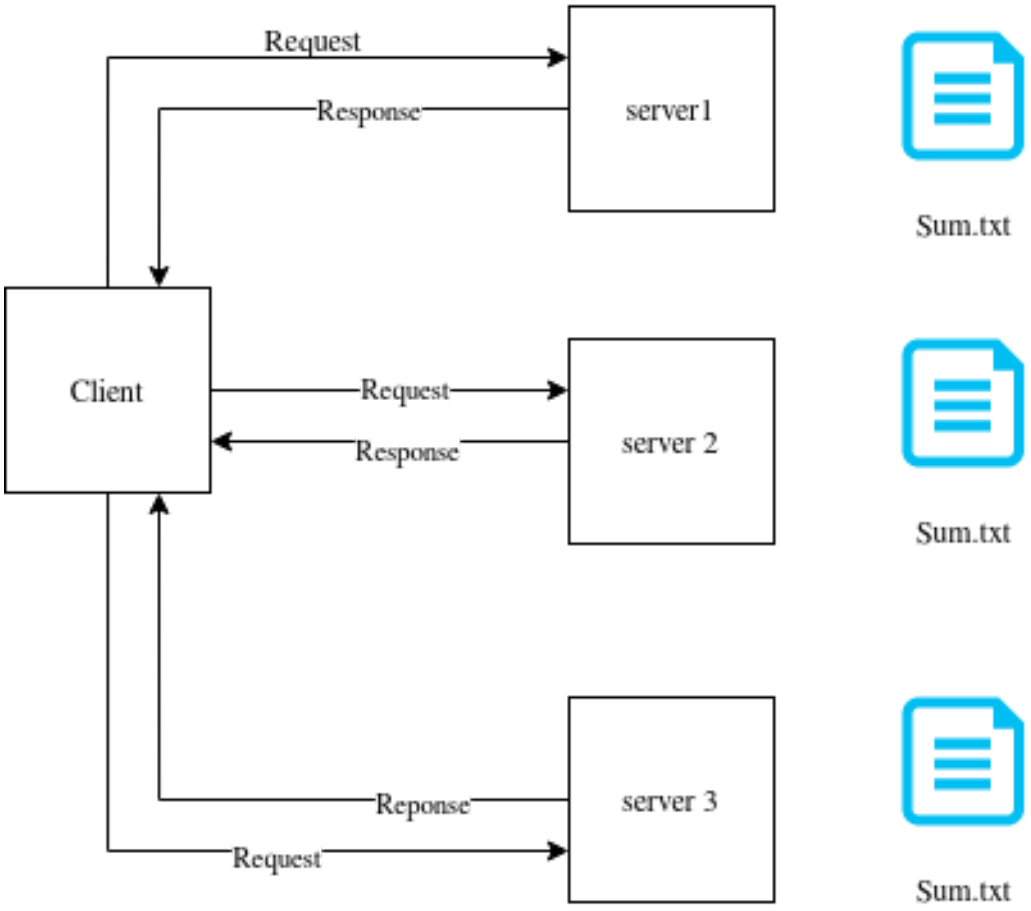
Terminal 1 (**`./2.gui.sh`**)

```
java -cp ../lib/jade.jar:classes myexamples.MultipleContainers
```

Terminal 2 (**`./3.agente.sh`**)

```
java -cp ../lib/jade.jar:classes jade.Boot -container -host localhost
-port 1099 -agents "AgenteMovil:myexamples.AgenteMovil($PWD/sum.txt)"
```

Si se quisiera implementar este funcionamiento en una app de arquitectura cliente/servidor, el servidor debería tener acceso al filesystem en el cual se almacena el archivo, y se debería exponer un servicio que lee los contenidos del archivo. El servidor se encargará de procesar y retornar la suma solicitada, o a lo sumo retornar los números en el archivo y delegar al cliente el procesamiento. En el caso de haber que leer varios archivos distribuidos, debería haber varios servidores con servicios similares expuestos al cual el cliente pueda acceder. En este caso, serìa necesario que el cliente se encargue de hacer la suma final.



3. Defina e implemente con agentes un sistema de archivos distribuido similar al de las prácticas anteriores.

a. Debería tener como mínimo la misma funcionalidad, es decir las operaciones (definiciones copiadas aquí de la práctica anterior):

- leer: dado un nombre de archivo, una posición y una cantidad de bytes a leer, retorna 1) la cantidad de bytes del archivo pedida a partir de la posición dada o en caso de haber menos bytes, se retornan los bytes que haya y 2) la cantidad de bytes que efectivamente se retornan leídos
- Escribir: dado un nombre de archivo, una cantidad de bytes determinada, y un buffer a partir del cual están los datos, se escriben los datos en el archivo dado. Si el archivo existe, los datos se agregan al final, si el archivo no existe, se crea y se le escriben los datos. En todos los casos se retorna la cantidad de bytes escritos.

b. Implemente un agente que copie un archivo de otro sitio del sistema distribuido en el sistema de archivos local y genere una copia del mismo archivo en el sitio donde está originalmente. Compare esta solución con la de los sistemas cliente/servidor de las prácticas anteriores.

En este proyecto, se planteó un agente llamado FPTAgent, con una clase auxiliar que procesa los comandos. La clase del FTPAgent ejecuta operaciones de lectura, escritura y lectura-escritura sobre archivos en un sistema de archivos entre contenedores. Este agente se comporta tanto como cliente como servidor.

Funcionalidad:

<u>FTPAgent:</u>

- **Función read (Leer)**
  La función read está diseñada para leer un archivo especificado en un contenedor remoto. Esta operación se realiza en segmentos, lo que significa que el archivo se lee por partes hasta completar su totalidad. El agente lee una porción del archivo remoto, específicamente 200,000 bytes cada vez, y los almacena en una variable. Si el archivo es más grande, se repite este proceso hasta completar la lectura.

  Cuando el agente se encuentra en un contenedor diferente al de origen, utiliza la función FTPUtils.readFile para leer una porción del archivo remoto. La cantidad de bytes leídos se controla mediante transferredBytes y totalBytes. Después de leer una parte, el agente migra de regreso al contenedor de origen para continuar con la operación.

Se hace escritura local; una vez que el agente está de vuelta en el contenedor de origen, los datos leídos se escriben localmente usando FTPUtils.writeFile. Este proceso se repite hasta que se hayan leído todos los bytes del archivo remoto.

- **Función write (Escribir):**
  Similar a la lectura, pero en sentido inverso. Esta operación también se realiza en segmentos. Aquí, el agente escribe los datos en el servidor, leyendo del archivo local. La función write permite escribir un archivo local en un contenedor remoto.
  Antes de migrar al contenedor remoto, el agente lee el archivo local utilizando FTPUtils.readFile y almacena los datos en un buffer. Luego, migra al contenedor remoto.
  En el contenedor remoto, el agente utiliza FTPUtils.writeFile para escribir los datos del buffer en el archivo especificado en la ruta remota. Este proceso continúa, leyendo del archivo local y escribiendo en el remoto, hasta que se completa la transferencia del archivo.

- **Función readwrite (Leer y Escribir):**
  La función readwrite es una operación híbrida que integra las capacidades de lectura y escritura en una única secuencia de acciones. Su propósito principal es realizar una copia de un archivo desde un contenedor remoto al contenedor de origen, y viceversa, asegurando que tanto el archivo local como el remoto sean idénticos al final del proceso.

  Este procedimiento inicia con la lectura del archivo local en el contenedor de origen. Los datos leídos se almacenan temporalmente en un buffer, originBuffer, para luego ser transferidos al contenedor remoto. Una vez que el agente llega al destino, procede a escribir estos datos en el archivo remoto especificado, utilizando la ruta remotePath.

  Tras completar la escritura en el contenedor remoto, el agente lleva a cabo la lectura del mismo archivo que acaba de escribir, almacenando los nuevos datos leídos en remoteBuffer. Esta acción garantiza que la información más actualizada se mantenga sincronizada entre ambos contenedores.

  Finalmente, el agente retorna al contenedor de origen y procede a escribir los datos del remoteBuffer en un archivo local. Esta última etapa del proceso implica crear una copia exacta del archivo remoto en el sistema local. El agente repite este ciclo de leer en un contenedor y escribir en el otro hasta que se transfieran todos los bytes del archivo, asegurando así que las versiones local y remota del archivo sean copias idénticas entre sí.

<u>FTPUtils:</u>

- **Función readFile (leer archivo):**
  Esta función lee una parte específica de un archivo, comenzando desde un desplazamiento (offset) hasta el tamaño total definido. Se utiliza para leer archivos tanto en operaciones de lectura como de escritura.

- **Función writeFile (escribir archivo):**
  Esta función escribe datos en un archivo especificado. Si el archivo no existe, lo crea; si ya existe, agrega los datos al final del archivo. Se usa en todas las operaciones de transferencia de archivos

Para la ejecución, se reutilizaron los scripts de ejemplo:

1.compilacion.sh

```
javac -classpath ../lib/jade.jar -d classes FTPAgent.java FTPUtils.java
```

2.gui.sh

```
java -cp ../lib/jade.jar:classes jade.Boot -gui
```

Y al **3.agente.sh** se le agrega una logica para recibir paramentros:

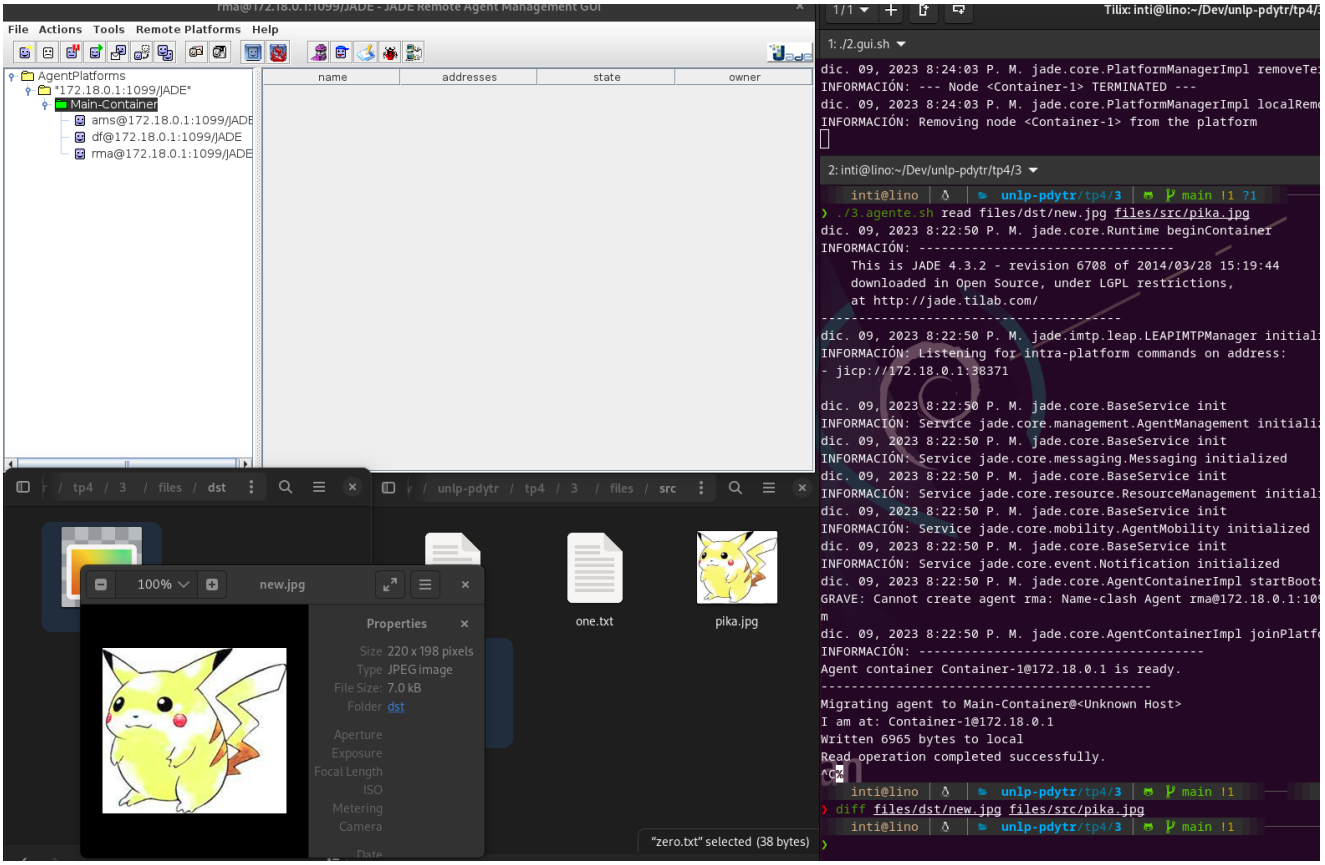
```
#!/bin/bash

if [ "$#" -ne 3 ]; then
    echo "Usage: $0 [read|write|readwrite] localPath remotePath"
    exit 1
fi

op=$1
local_path=$2
remote_path=$3


java -cp ../lib/jade.jar:classes jade.Boot -gui -container  -host
localhost -agents "mol:FTPAgent($op,$local_path,$remote_path)"
```

Ejemplos con archivos binarios (una imagen):

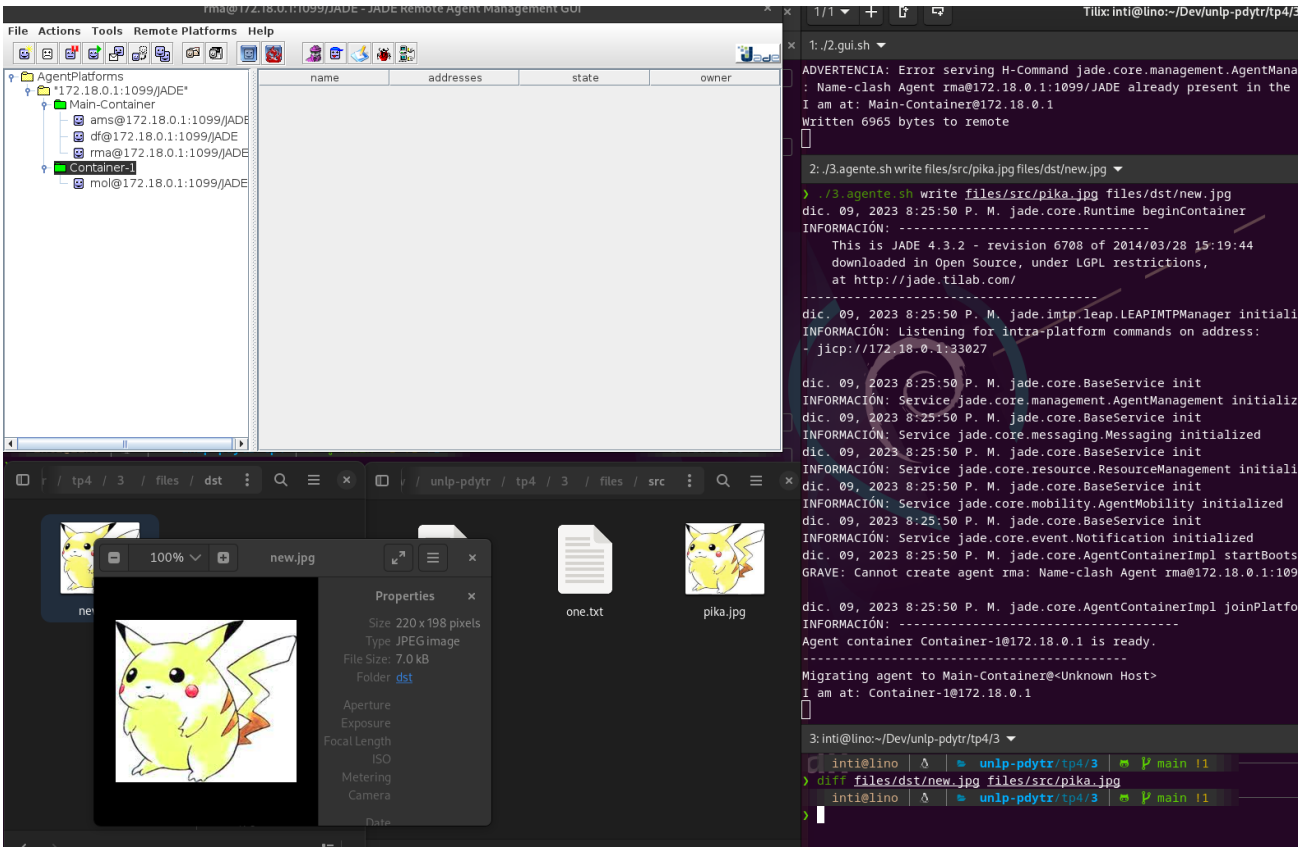
READ:

```
❯ ./3.agente.sh read files/dst/new.jpg files/src/pika.jpg
```

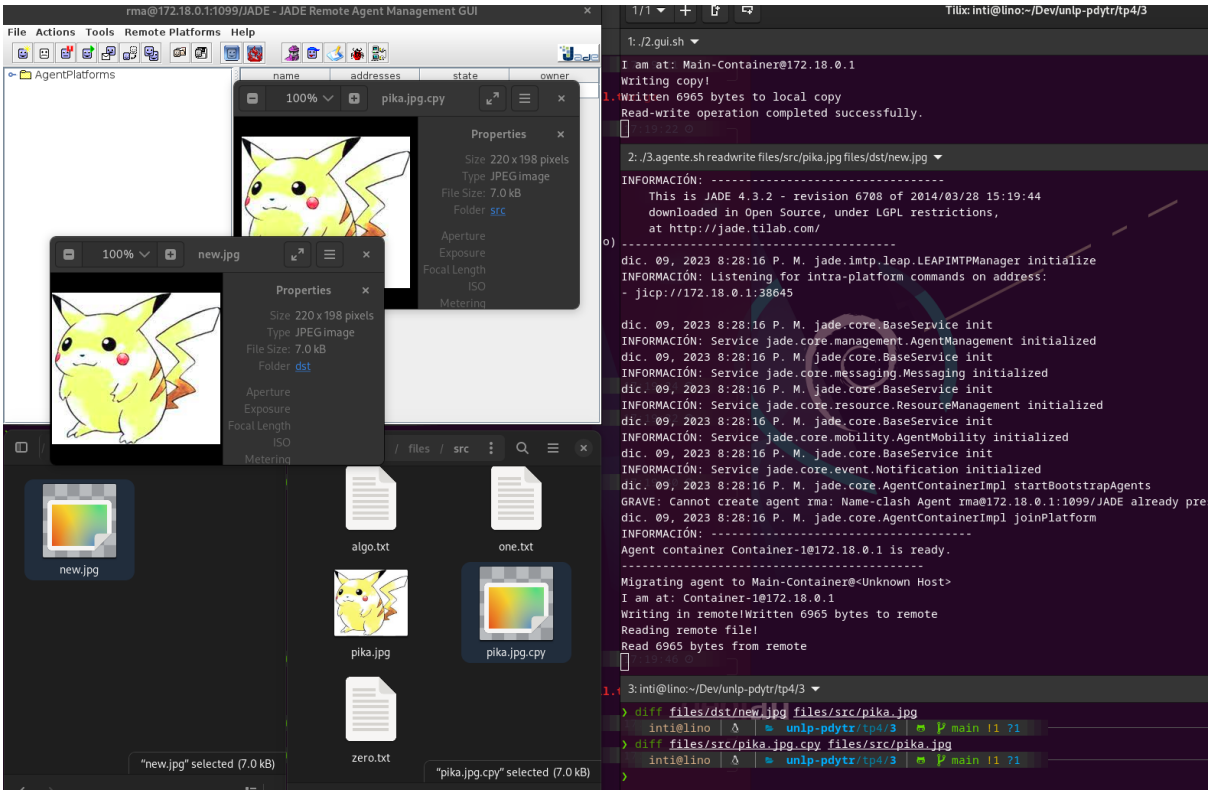
READ:


File Actions Tools Remote Platforms Help
AgentPlatforms
"172.18.0.1:1099/JADE"
Main-Container
ams@172.18.0.1:1099/JADE
df@172.18.0.1:1099/JADE
rma@172.18.0.1:1099/JADE
name
addresses
state
owner
new.jpg
Properties
Size 220 x 198 pixels
Type JPEG image
File Size: 7.0 kB
Folder dst
one.txt
pika.jpg
"zero.txt" selected (38 bytes)
dic. 09, 2023 8:24:03 P. M. jade.core.PlatformManagerImpl removeTe
INFORMACIÓN: --- Node <Container-1> TERMINATED ---
dic. 09, 2023 8:24:03 P. M. jade.core.PlatformManagerImpl localRem
INFORMACIÓN: Removing node <Container-1> from the platform
./3.agente.sh read files/dst/new.jpg files/src/pika.jpg
dic. 09, 2023 8:22:50 P. M. jade.core.Runtime beginContainer
This is JADE 4.3.2 - revision 6708 of 2014/03/28 15:19:44
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
dic. 09, 2023 8:22:50 P. M. jade.imtp.leap.LEAPIMTPManager initial
INFORMACIÓN: Listening for intra-platform commands on address:
- jicp://172.18.0.1:38371
dic. 09, 2023 8:22:50 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.management.AgentManagement initiali
INFORMACIÓN: Service jade.core.messaging.Messaging initialized
INFORMACIÓN: Service jade.core.resource.ResourceManagement initial
INFORMACIÓN: Service jade.core.mobility.AgentMobility initialized
INFORMACIÓN: Service jade.core.event.Notification initialized
dic. 09, 2023 8:22:50 P. M. jade.core.AgentContainerImpl startBoot
GRAVE: Cannot create agent rma: Name-clash Agent rma@172.18.0.1:10
dic. 09, 2023 8:22:50 P. M. jade.core.AgentContainerImpl joinPlatf
Agent container Container-1@172.18.0.1 is ready.
Migrating agent to Main-Container@<Unknown Host>
I am at: Container-1@172.18.0.1
Written 6965 bytes to local
Read operation completed successfully.
diff files/dst/new.jpg files/src/pika.jpg

WRITE


File Actions Tools Remote Platforms Help
AgentPlatforms
"172.18.0.1:1099/JADE"
Main-Container
ams@172.18.0.1:1099/JADE
df@172.18.0.1:1099/JADE
rma@172.18.0.1:1099/JADE
Container-1
mol@172.18.0.1:1099/JADE
name
addresses
state
owner
new.jpg
Properties
Size 220 x 198 pixels
Type JPEG image
File Size: 7.0 kB
Folder dst
one.txt
pika.jpg
ADVERTENCIA: Error serving H-Command jade.core.management.AgentMana
: Name-clash Agent rma@172.18.0.1:1099/JADE already present in the
I am at: Main-Container@172.18.0.1
Written 6965 bytes to remote
./3.agente.sh write files/src/pika.jpg files/dst/new.jpg
dic. 09, 2023 8:25:50 P. M. jade.core.Runtime beginContainer
This is JADE 4.3.2 - revision 6708 of 2014/03/28 15:19:44
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
dic. 09, 2023 8:25:50 P. M. jade.imtp.leap.LEAPIMTPManager initiali
INFORMACIÓN: Listening for intra-platform commands on address:
- jicp://172.18.0.1:33027
dic. 09, 2023 8:25:50 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.management.AgentManagement initializ
INFORMACIÓN: Service jade.core.messaging.Messaging initialized
INFORMACIÓN: Service jade.core.resource.ResourceManagement initiali
INFORMACIÓN: Service jade.core.mobility.AgentMobility initialized
INFORMACIÓN: Service jade.core.event.Notification initialized
dic. 09, 2023 8:25:50 P. M. jade.core.AgentContainerImpl startBoots
GRAVE: Cannot create agent rma: Name-clash Agent rma@172.18.0.1:109
dic. 09, 2023 8:25:50 P. M. jade.core.AgentContainerImpl joinPlatfo
Agent container Container-1@172.18.0.1 is ready.
Migrating agent to Main-Container@<Unknown Host>
I am at: Container-1@172.18.0.1
diff files/dst/new.jpg files/src/pika.jpg

READWRITE:



Este enfoque, aunque unifica el cliente y el servidor en un solo agente, resulta en una mayor complejidad y potencialmente en una menor escalabilidad y mantenimiento. Además, el uso intensivo de migraciones puede incrementar el overhead en comparación con las implementaciones cliente/servidor tradicionales.