



Instituto Politécnico Nacional

Escuela Superior de Cómputo

Sistemas Distribuidos

Pineda Guerrero Carlos

Tarea 4

4CV13

Humberto Alejandro Ortega Alcocer (2016630495)

5 de Abril del 2023

Índice

Índice	1
Objetivo	2
Desarrollo	4
Creación de Máquina Virtual	4
Verificando que tengamos acceso a la máquina virtual.	10
Configurando el servidor para la tarea.	11
Instalando Java 11	11
Abriendo el puerto de ejecución.	13
Copiando los archivos necesarios.	15
Creación de Código (ChatGPT)	16
Ejecución del Programa	24
Conclusión	26

Objetivo

Utilizando ChatGPT desarrollar un solo programa en Java en modo consola que implemente un chat utilizando comunicación multicast mediante datagramas.

Se deberá ejecutar el programa en una máquina virtual con Windows Server 2012 en Azure.

Se deberá pasar como parámetro al programa el nombre del usuario que va escribir en el chat.

El programa deberá hacer lo siguiente:

- El programa creará un thread el cual recibirá los mensajes del resto de los nodos. Cada mensaje recibido será desplegado en la pantalla. El thread desplegará el mensaje que envía el mismo nodo.
- En el método main(), dentro de un ciclo infinito:
 - Se desplegará el siguiente prompt: "Escribe tu mensaje: " (sin las comillas), entonces se leerá del teclado el mensaje.
 - Se deberá enviar el mensaje a los nodos que pertenecen al grupo identificado por la IP 239.0.0.0 a través del puerto 50000. El mensaje a enviar en el datagrama deberá tener la siguiente forma: *nombre_usuario--->mensaje_ingresado*

Dónde *nombre_usuario* es el nombre del usuario que pasó como parámetro al programa, "*-->*" es un separador y *mensaje_ingresado* es el mensaje que el usuario ingresó por el teclado.

Debido a que el protocolo UDP no garantiza el orden en que se reciben los mensajes, no se deberá enviar la longitud del mensaje y luego el mensaje; se deberá utilizar una longitud de mensaje fija. Al desplegar el mensaje recibido se deberán eliminar los espacios adicionales a la derecha (si los hay).

Para demostrar el programa se deberá utilizar los siguientes usuarios: Hugo, Paco y Luis.

Se deberá ejecutar la siguiente conversación (la cual aparece en negritas) en tres ventanas de comandos (cmd) en la máquina virtual con Windows Server 2012. En la primera ventana escribirá Hugo, en la segunda ventana escribirá Paco y en la tercera ventana escribirá Luis:

Hugo escribirá:
hola a todos

Paco escribirá:
hola Hugo

Luis escribirá:

hola Paco
hola Hugo

Paco escribirá:

¿alguien sabe dónde será el concierto?

Hugo escribirá:

será en la plaza central

Paco escribirá:

¿a qué hora?

Luis escribirá:

a las 8 PM

Paco escribirá:

gracias, adiós

Luis debe escribir:

adiós Paco

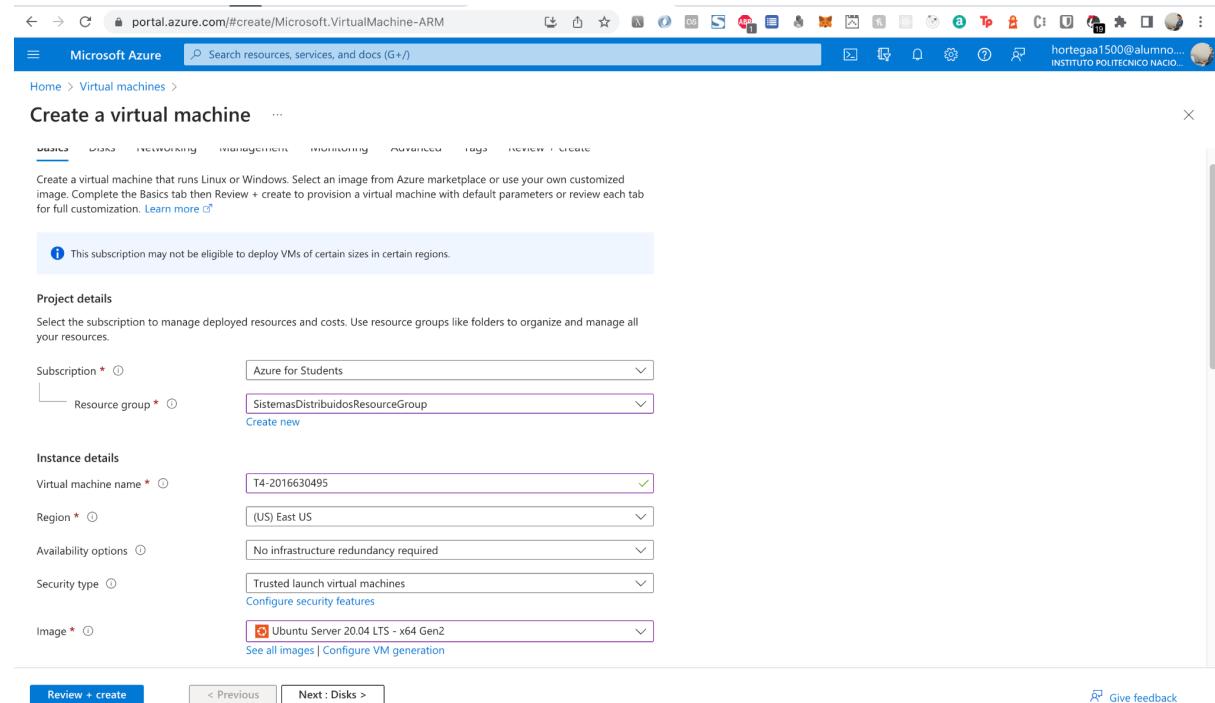
Los signos de interrogación y las letras acentuadas deberán desplegarse correctamente en la ventana de comandos de Windows.

Desarrollo

Para desarrollar esta tarea, se descompondrá en tres etapas: creación de máquina virtual creación de código (usando ChatGPT) y pruebas de ejecución. Esto con el fin de profundizar en cada sección de forma adecuada.

Creación de Máquina Virtual

Lo primero será seguir el asistente para crear una máquina virtual nueva:



The screenshot shows the Microsoft Azure portal interface for creating a new virtual machine. The URL in the address bar is `portal.azure.com/#create/Microsoft.VirtualMachine-ARM`. The page title is "Create a virtual machine". The top navigation bar includes "Home > Virtual machines >" and "Create a virtual machine". Below the title, there are tabs for "Basics", "Disks", "Networking", "Management", "Monitoring", "Advanced", "Tags", "Review + create", and "Create". A note at the top states: "Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization. [Learn more](#)". A warning message in a blue box says: "This subscription may not be eligible to deploy VMs of certain sizes in certain regions." The "Project details" section shows "Subscription" set to "Azure for Students" and "Resource group" set to "SistemasDistribuidosResourceGroup". The "Instance details" section includes fields for "Virtual machine name" (T4-2016630495), "Region" (US East US), "Availability options" (No infrastructure redundancy required), "Security type" (Trusted launch virtual machines), and "Image" (Ubuntu Server 20.04 LTS - x64 Gen2). At the bottom, there are buttons for "Review + create" and "Next : Disks >".

Lo siguiente será colocar la configuración de tamaño (1 vCPU y 1GB en RAM), así como las opciones de seguridad: la clave creada para la Tarea 1 (reutilizamos el mismo combo) y permitimos el tráfico TCP al puerto 22 para las sesiones de SSH (SCP, SFTP, etc).

VM architecture: Arm64 x64

Run with Azure Spot discount:

Size *: Standard_B1s - 1 vcpu, 1 GiB memory (\$7.59/month)
[See all sizes](#)

Administrator account:

Authentication type: SSH public key Password

Azure now automatically generates an SSH key pair for you and allows you to store it for future use. It is a fast, simple, and secure way to connect to your virtual machine.

Username *: humbertowoody

SSH public key source: Use existing key stored in Azure

Stored Keys: humbertowoody-azure

Inbound port rules: Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports *: None Allow selected ports

[Review + create](#) [< Previous](#) [Next : Disks >](#) [Give feedback](#)

Authentication type: SSH public key Password

Azure now automatically generates an SSH key pair for you and allows you to store it for future use. It is a fast, simple, and secure way to connect to your virtual machine.

Username *: humbertowoody

SSH public key source: Use existing key stored in Azure

Stored Keys: humbertowoody-azure

Inbound port rules: Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports *: None Allow selected ports

Select inbound ports *: SSH (22)

This will allow all IP addresses to access your virtual machine. This is only recommended for testing. Use the Advanced controls in the Networking tab to create rules to limit inbound traffic to known IP addresses.

[Review + create](#) [< Previous](#) [Next : Disks >](#) [Give feedback](#)

Lo siguiente será colocar la configuración de disco: HDD con 30GiB y sin redundancia.

Encryption at host is not registered for the selected subscription. [Learn more about enabling this feature](#)

OS disk

OS disk size * Default size (30 GiB)

OS disk type * Standard HDD (locally-redundant storage) Premium SSD (high IOPS)

The selected VM size supports premium disks. We recommend Premium SSD for high IOPS workloads. Virtual machines with Premium SSD disks qualify for the 99.9% connectivity SLA.

Delete with VM

Key management Platform-managed key

Enable Ultra Disk compatibility Ultra disk is not supported with selected security type.

Data disks for T4-2016630495

You can add and configure additional data disks for your virtual machine or attach existing disks. This VM also comes with a temporary disk.

LUN	Name	Size (GiB)	Disk type	Host caching	Delete with VM

Create and attach a new disk [Attach an existing disk](#)

Advanced

Review + create < Previous Next : Networking > Give feedback

go.microsoft.com/fwlink/?LinkId=2012733

Finalmente, el resumen de nuestra máquina virtual.

Validation passed

Basics Disks Networking Management Monitoring Advanced Tags Review + create

Cost given below is an estimate and not the final price. Please use [Pricing calculator](#) for all your pricing needs.

Price

1 X Standard B1s by Microsoft Subscription credits apply
0.0104 USD/hr [Pricing for other VM sizes](#)

TERMS

By clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) listed above; (b) authorize Microsoft to bill my current payment method for the fees associated with the offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact, usage and transactional information with the provider(s) of the offering(s) for support, billing and other transactional activities. Microsoft does not provide rights for third-party offerings. See the [Azure Marketplace Terms](#) for additional details.

Name

Preferred e-mail address

Preferred phone number

Create < Previous Next > Download a template for automation Give feedback

Basics

Subscription	Azure for Students
Resource group	SistemasDistribuidosResourceGroup
Virtual machine name	T4-2016630495
Region	East US
Availability options	No infrastructure redundancy required
Security type	Trusted launch virtual machines
Enable secure boot	Yes
Enable vTPM	Yes
Integrity monitoring	Yes
Image	Ubuntu Server 20.04 LTS - Gen2
VM architecture	x64
Size	Standard B1s (1 vcpu, 1 GiB memory)
Authentication type	SSH public key
Username	humbertowoody
Key pair name	humbertowoody-azure
Public inbound ports	SSH
Azure Spot	No

Disks

Create < Previous Next > Download a template for automation Give feedback

Networking

Virtual network	(new) T4-2016630495-vnet
Subnet	(new) default (10.0.0.0/24)
Public IP	(new) T4-2016630495-ip
Accelerated networking	Off
Place this virtual machine behind an existing load balancing solution?	No
Delete public IP and NIC when VM is deleted	Disabled

Management

Microsoft Defender for Cloud	None
System assigned managed identity	Off
Login with Azure AD	Off
Auto-shutdown	Off

Validation passed

Management

Microsoft Defender for Cloud	None
System assigned managed identity	Off
Login with Azure AD	Off
Auto-shutdown	Off
Enable hotpatch	Off
Patch orchestration options	Image Default

Monitoring

Alerts	Off
Boot diagnostics	On
Enable OS guest diagnostics	Off

Advanced

Extensions	None
VM applications	None
Cloud init	No
User data	No
Disk controller type	SCSI
Proximity placement group	None
Capacity reservation group	None

Create < Previous Next > Download a template for automation Give feedback

Creamos la máquina virtual.

Your deployment is complete

Deployment name: CreateVm-canonical.0001-com-ubuntu-server-f... Start time: 4/5/2023, 3:25:31 PM
Subscription: Azure for Students Correlation ID: 84878cc4-3fb3-4b66-bd4a-6bc8d296a3a8

Deployment details

Resource	Type	Status	Operation details
T4-2016630495/GuestAttestation	Microsoft.Compute/virtualMachine	OK	Operation details
T4-2016630495	Microsoft.Compute/virtualMachine	OK	Operation details
t4-2016630495231	Microsoft.Network/networkInterfaces	Created	Operation details
T4-2016630495-vnet	Microsoft.Network/virtualNetwork	OK	Operation details
T4-2016630495-ip	Microsoft.Network/publicIPAddresses	OK	Operation details
T4-2016630495-nsg	Microsoft.Network/networkSecurityGroups	OK	Operation details

Next steps

- Setup auto-shutdown Recommended
- Monitor VM health, performance and network dependencies Recommended
- Run a script inside the virtual machine Recommended

Cost Management
Get notified to stay within your budget and prevent unexpected charges on your bill.
[Set up cost alerts >](#)

Microsoft Defender for Cloud
Secure your apps and infrastructure
[Go to Microsoft Defender for Cloud >](#)

Free Microsoft tutorials
[Start learning today >](#)

Work with an expert
Azure experts are service provider partners who can help manage your assets on Azure and be your first line of support.
[Find an Azure expert >](#)

Go to resource Create another VM Give feedback Tell us about your experience with deployment

Verificamos configuración final de la máquina virtual (para ver la IP pública y detalles finales).

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes links for Home, CreateVm-canonical.0001-com-ubuntu-server-focal-2-20230405151808 | Overview, and other account and settings options. The main content area displays the details of a virtual machine named "T4-2016630495".

Essentials

Resource group (move)	: SistemasDistribuidosResourceGroup	Operating system	: Linux (ubuntu 20.04)
Status	: Running	Size	: Standard B1s (1 vcpu, 1 GiB memory)
Location	: East US	Public IP address	: 20.120.100.96
Subscription (move)	: Azure for Students	Virtual network/subnet	: T4-2016630495-vnet/default
Subscription ID	: a5d9d675-3ae1-4e2b-95b5-02150d67b8d2	DNS name	: Not configured
Tags (edit)	: Click here to add tags	JSON View	

Properties

Virtual machine	Networking
Computer name	T4-2016630495
Health state	-
Operating system	Linux (ubuntu 20.04)
Publisher	canonical
Offer	0001-com-ubuntu-server-focal
Plan	20_04-lts-gen2
VM generation	V2
VM architecture	x64
Agent status	Ready
Agent version	2.9.0.4
Host group	None
Host	-
Networking	
Public IP address	20.120.100.96 (Network interface t4-2016630495231)
Public IP address (IPv6)	-
Private IP address	10.0.0.4
Private IP address (IPv6)	-
Virtual network/subnet	T4-2016630495-vnet/default
DNS name	Configure

Size

Size	vCPUs	RAM
Standard B1s	1	1 GiB

Verificando que tengamos acceso a la máquina virtual.

Para verificar que tenemos acceso a la máquina virtual, nos conectaremos utilizando las llaves previamente configuradas en la Tarea #1.

```
> ls
humbertowoody-azure.pem
> ssh -i ./humbertowoody-azure.pem humbertowoody@20.120.100.96
The authenticity of host '20.120.100.96 (20.120.100.96)' can't be established.
ED25519 key fingerprint is SHA256:glcgDeBTov0U27650U8IkfusUkCeI3lotUAmmy9CVg.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '20.120.100.96' (ED25519) to the list of known hosts.
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-1035-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

System information as of Wed Apr  5 21:29:41 UTC 2023

System load:  0.07          Processes:           107
Usage of /:   5.2% of 28.89GB  Users logged in:    0
Memory usage: 32%           IPv4 address for eth0: 10.0.0.4
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

humbertowoody@T4-2016630495:~$ echo "wuuuuu!"
wuuuuu!
humbertowoody@T4-2016630495:~$
```

Configurando el servidor para la tarea.

Para poder comenzar a ejecutar nuestra tarea, necesitaremos realizar tres acciones en el servidor (máquina virtual Ubuntu 20.04) que nos permitan ejecutar nuestro programa así como comunicarnos con el mismo remotamente.

Lo primero será instalar Java dado que nuestra tarea se ha realizado en dicho lenguaje de programación y Ubuntu (20.04) no incluye una versión del JDK ni JRE en el sistema operativo por defecto.

En este punto, la máquina virtual únicamente cuenta con el puerto 22 abierto por el cortafuegos (firewall) de Azure, por lo tanto necesitaremos abrir el puerto dónde se ejecutará nuestra aplicación a manera de permitir la comunicación cliente-servidor.

Finalmente, deberemos copiar el código fuente del Servidor y nuestro archivo de claves para que la comunicación cliente-servidor se efectúe de manera segura.

Instalando Java 11

Se ha determinado utilizar Java en su versión 11 porque es lo que tengo instalado en mi computadora local y no quisiera que hubiera incompatibilidades entre las versiones, esto a pesar de que las funcionalidades que utilizamos (sockets seguros) no han cambiado desde Java 8.

Para instalar Java 11 en Ubuntu, usaremos el siguiente comando:

```
$ sudo apt install openjdk-11-jdk
```

A continuación mostramos como se ve este comando en la máquina virtual:

```
humberto@T4-2016630495:~$ sudo apt install openjdk-11-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
at-spi2-core ca-certificates-java fontconfig-config fonts-dejavu-core
fonts-dejavu-extra java-common libatk-bridge2.0-0 libatk-wrapper-java
libatk-wrapper-java-jni libatk1.0-0 libatk1.0-data libatspi2.0-0
libavahi-client3 libavahi-common-data libavahi-common3 libcups2 libdrm-amdgpu1
libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libfontconfig1 libfontenc1 libgif7
libgl1 libgl1-mesa-dri libglapi-mesa libglvnd0 libglx-mesa0 libglx0
libgraphite2-3 libharfbuzz0b libice-dev libice6 libjpeg-turbo8 libjpeg8
liblcms2-2 liblvm12 libpciaccess0 libpcslite1 libpthread-stubs0-dev
libsensors-config libsensors5 libsm-dev libsm6 libvulkan1 libwayland-client0
libx11-dev libx11-xcb1 libxau-dev libxaw7 libxcb-dri2-0 libxcb-dri3-0
libxcb-glx0 libxcb-present0 libxcb-randr0 libxcb-shape0 libxcb-shm0
libxcb-sync1 libxcb-xfixes0 libxcb1-dev libcomposite1 libxdmcp-dev libxfixes3
libxft2 libxi6 libxinerama1 libxkbfile1 libxmu6 libxpm4 libxrandr2 libxrender1
libxshmfence1 libxt-dev libxt6 libxtst6 libxv1 libxxf86dga1 libxxf86vm1
mesa-vulkan-drivers openjdk-11-jdk-headless openjdk-11-jre
openjdk-11-jre-headless x11-common x11-utils x11proto-core-dev x11proto-dev
xorg-sgml-doctools xtrans-dev
Suggested packages:
default-jre cups-common liblcms2-utils pcscd lm-sensors libsm-doc
libx11-doc libxcb-doc libxt-doc openjdk-11-demo openjdk-11-source visualvm
libnss-mdns fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei
| fonts-wqy-zenhei fonts-indic mesa-utils
The following NEW packages will be installed:
at-spi2-core ca-certificates-java fontconfig-config fonts-dejavu-core
fonts-dejavu-extra java-common libatk-bridge2.0-0 libatk-wrapper-java
libatk-wrapper-java-jni libatk1.0-0 libatk1.0-data libatspi2.0-0
libavahi-client3 libavahi-common-data libavahi-common3 libcups2 libdrm-amdgpu1
libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libfontconfig1 libfontenc1 libgif7
libgl1 libgl1-mesa-dri libglapi-mesa libglvnd0 libglx-mesa0 libglx0
libgraphite2-3 libharfbuzz0b libice-dev libice6 libjpeg-turbo8 libjpeg8
liblcms2-2 liblvm12 libpciaccess0 libpcslite1 libpthread-stubs0-dev
libsensors-config libsensors5 libsm-dev libsm6 libvulkan1 libwayland-client0
libx11-dev libx11-xcb1 libxau-dev libxaw7 libxcb-dri2-0 libxcb-dri3-0
libxcb-glx0 libxcb-present0 libxcb-randr0 libxcb-shape0 libxcb-shm0
libxcb-sync1 libxcb-xfixes0 libxcb1-dev libcomposite1 libxdmcp-dev libxfixes3
libxft2 libxi6 libxinerama1 libxkbfile1 libxmu6 libxpm4 libxrandr2 libxrender1
libxshmfence1 libxt-dev libxt6 libxtst6 libxv1 libxxf86dga1 libxxf86vm1
mesa-vulkan-drivers openjdk-11-jdk openjdk-11-jdk-headless openjdk-11-jre
openjdk-11-jre-headless x11-common x11-utils x11proto-core-dev x11proto-dev
xorg-sgml-doctools xtrans-dev
0 upgraded, 89 newly installed, 0 to remove and 0 not upgraded.
Need to get 307 MB of archives.
After this operation, 958 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://azure.archive.ubuntu.com/ubuntu focal/main amd64 libatspi2.0-0 amd64 2
.36.0-2 [64.2 kB]
Get:2 http://azure.archive.ubuntu.com/ubuntu focal/main amd64 x11-common all 1:7.7+
19ubuntu14 [22.3 kB]
Get:3 http://azure.archive.ubuntu.com/ubuntu focal/main amd64 libxtst6 amd64 2:1.2.
3-1 [12.8 kB]
Get:4 http://azure.archive.ubuntu.com/ubuntu focal/main amd64 at-spi2-core amd64 2
```

Una vez instalado, verificaremos que contamos con la versión correcta disponible en nuestro sistema operativo:

```
humbertowoody@T4-2016630495:~$ java --version
openjdk 11.0.18 2023-01-17
OpenJDK Runtime Environment (build 11.0.18+10-post-Ubuntu-0ubuntu120.04.1)
OpenJDK 64-Bit Server VM (build 11.0.18+10-post-Ubuntu-0ubuntu120.04.1, mixed mode,
sharing)
humbertowoody@T4-2016630495:~$ javac --version
javac 11.0.18
humbertowoody@T4-2016630495:~$
```

Abriendo el puerto de ejecución.

Para que nuestra aplicación pueda ejecutarse correctamente, deberemos también exponer el puerto donde se disponibilizará el servicio a internet, esto se logra mediante la configuración de red en Azure.

Lo primero será ir a la máquina virtual y buscar el apartado de *Networking*.

The screenshot shows the Azure portal's Networking page for a virtual machine named "T4-2016630495". The left sidebar has "Networking" selected under "Settings". The main area shows an "Inbound port rules" table with the following entries:

Priority	Name	Port	Protocol	Source	Destination	Action
300	SSH	22	TCP	Any	Any	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

Haremos click en “add inbound port rule” y ajustaremos la configuración para permitir el tráfico TCP en el puerto 8443 de cualquier destino con una prioridad de 310 (debajo de SSH).

The screenshot shows the Azure portal interface for managing a virtual machine named T4-2016630495. On the left, the 'Networking' settings are selected. In the main pane, under 'Inbound port rules', a new rule is being created. The 'Add inbound security rule' dialog box is open, showing the configuration for a new rule:

Setting	Value
Source	Any
Source port ranges	*
Destination	Any
Destination port ranges	8443
Protocol	TCP
Action	Allow
Priority	310
Name	AllowAnyCustom8443Inbound

Finalmente veremos cómo se creó correctamente la regla de puerto entrante.

The screenshot shows the same Azure portal interface after the rule has been added. The 'Inbound port rules' table now includes the new rule:

Priority	Name	Port	Protocol	Source	Destination	Action
300	SSH	22	TCP	Any	Any	Allow
310	AllowAnyCustom8443Inbound	8443	TCP	Any	Any	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

Copiando los archivos necesarios.

El último paso de configuración será copiar los archivos necesarios para la ejecución, estos archivos son:

- **Servidor.java**: es el archivo que implementa la clase Servidor dónde se encuentra toda la funcionalidad que ofrece el servicio en el puerto **8443**.
- **certificado_servidor.pem**: es la llave privada del servidor que se utilizará para encriptar y desencriptar el tráfico entrante y saliente.
- **keystore_servidor.jks**: es el archivo dónde Java puede almacenar el acceso correspondiente a distintas llaves públicas/privadas y se utiliza para usar **certificado_servidor.pem**.

Para copiar los archivos, usaremos el comando [scp\(1\)](#), con el cual podemos copiar archivos desde y hacia nuestro servidor remoto usando SSH como protocolo de conectividad. El comando final será:

```
> scp -i humbertowoody-azure.pem ../keystores_java/certificado_servidor.pem humbertowoody@20.120.100.96:/home/humbertowoody
certificado_servidor.pem                                         100% 1259      15.8KB/s  00:00
> scp -i humbertowoody-azure.pem ../keystores_java/keystore_servidor.jks humbertowoody@20.120.100.96:/home/humbertowoody
keystore_servidor.jks                                         100% 2772      31.6KB/s  00:00
> scp -i humbertowoody-azure.pem ../tareas/tarea-4/Servidor.java humbertowoody@20.120.100.96:/home/humbertowoody
Servidor.java                                                 100% 4302      53.7KB/s  00:00
> ssh -i ./humbertowoody-azure.pem humbertowoody@20.120.100.96
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-1035-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

System information as of Wed Apr  5 21:50:16 UTC 2023

System load:  0.0          Processes:           103
Usage of /:   7.3% of 28.89GB  Users logged in:    0
Memory usage: 33%          IPv4 address for eth0: 10.0.0.4
Swap usage:   0%

* Introducing Expanded Security Maintenance for Applications.
  Receive updates to over 25,000 software packages with your
  Ubuntu Pro subscription. Free for personal use.

  https://ubuntu.com/azure/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '22.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Apr  5 21:29:43 2023 from 189.132.94.124
humbertowoody@T4-2016630495:~$ ls
Servidor.java  certificado_servidor.pem  keystore_servidor.jks
humbertowoody@T4-2016630495:~$ █
```

ESCOM ER ♥ AC 97% Wed 05/04 03:50 PM CST 29°C Mexico City, Mexico City

Creación de Código (ChatGPT)

Para generar el código de esta tarea, se utilizó ChatGPT con la metodología que se empleó en la tarea anterior, primeramente se plantea la tarea en su totalidad y se verifica que el modelo comprenda los objetivos de la práctica, posteriormente se solicita uno a uno los programas a manera de maximizar la cantidad de tokens disponibles para cada respuesta y así permitir que el modelo pueda concluir en una iteración la generación del código necesario, finalmente se realizan preguntas generales para corregir detalles de implementación o posibles errores de ejecución.

A continuación, la conversación sostenida con ChatGPT:

Model: GPT-4

Hola! quisiera el código en Java (11) que resuelve el siguiente planteamiento, solamente quisiera el código del servidor por hora, muchas gracias!

1. Servidor multi-thread con sockets seguros, el cual recibirá del cliente dos tipos de peticiones:

1.1 La petición GET seguida del nombre de un archivo (sin directorios). El servidor deberá leer el archivo del disco local, si el archivo se pudo leer el servidor deberá enviar al cliente OK, la longitud del archivo y el contenido del archivo, de otra manera deberá enviar al cliente ERROR.

1.2 La petición PUT seguida del nombre de un archivo (sin directorios), la longitud del archivo y el contenido del archivo. El servidor deberá escribir el archivo en el disco local, si el archivo se pudo escribir en el disco, el servidor enviará al cliente OK de otra manera enviará ERROR.

2. Cliente PUT, Cliente con sockets seguros, el cual deberá recibir como parámetros la IP del servidor, el puerto que escucha el servidor y el nombre del archivo a enviar.

Entonces el cliente leerá el archivo del disco local, si puede leer el archivo se conectará al servidor y enviará una petición PUT, el nombre del archivo a enviar, la longitud del archivo y el contenido del archivo. Si el cliente no puede leer el archivo del disco local desplegará un mensaje de error indicando esta situación.

El cliente deberá esperar la respuesta del servidor, si el servidor responde OK el cliente deberá desplegar un mensaje indicando que el archivo fue recibido por el servidor con éxito, de otra manera desplegará un mensaje de error indicando que el servidor no pudo escribir el archivo en el disco local.

3. Cliente GET, Cliente con sockets seguros, el cual deberá recibir como parámetros la IP del servidor, el puerto que escucha el servidor y el nombre del archivo a recibir.

Entonces el cliente se conectará al servidor y enviará la petición GET y el nombre del archivo.

El cliente deberá esperar la respuesta del servidor, si el servidor responde OK el cliente deberá recibir la longitud del archivo y el contenido del archivo. Entonces el cliente deberá escribir el archivo en el disco local, si puede escribir el archivo deberá desplegar un mensaje indicando que el archivo se recibió con éxito, de otra manera deberá desplegar un mensaje de error.

Aquí tienes el código del servidor multi-thread con sockets seguros que cumple con los requisitos que has descrito:

```
java
import java.io.*;
import java.net.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import javax.net.ssl.*;
```

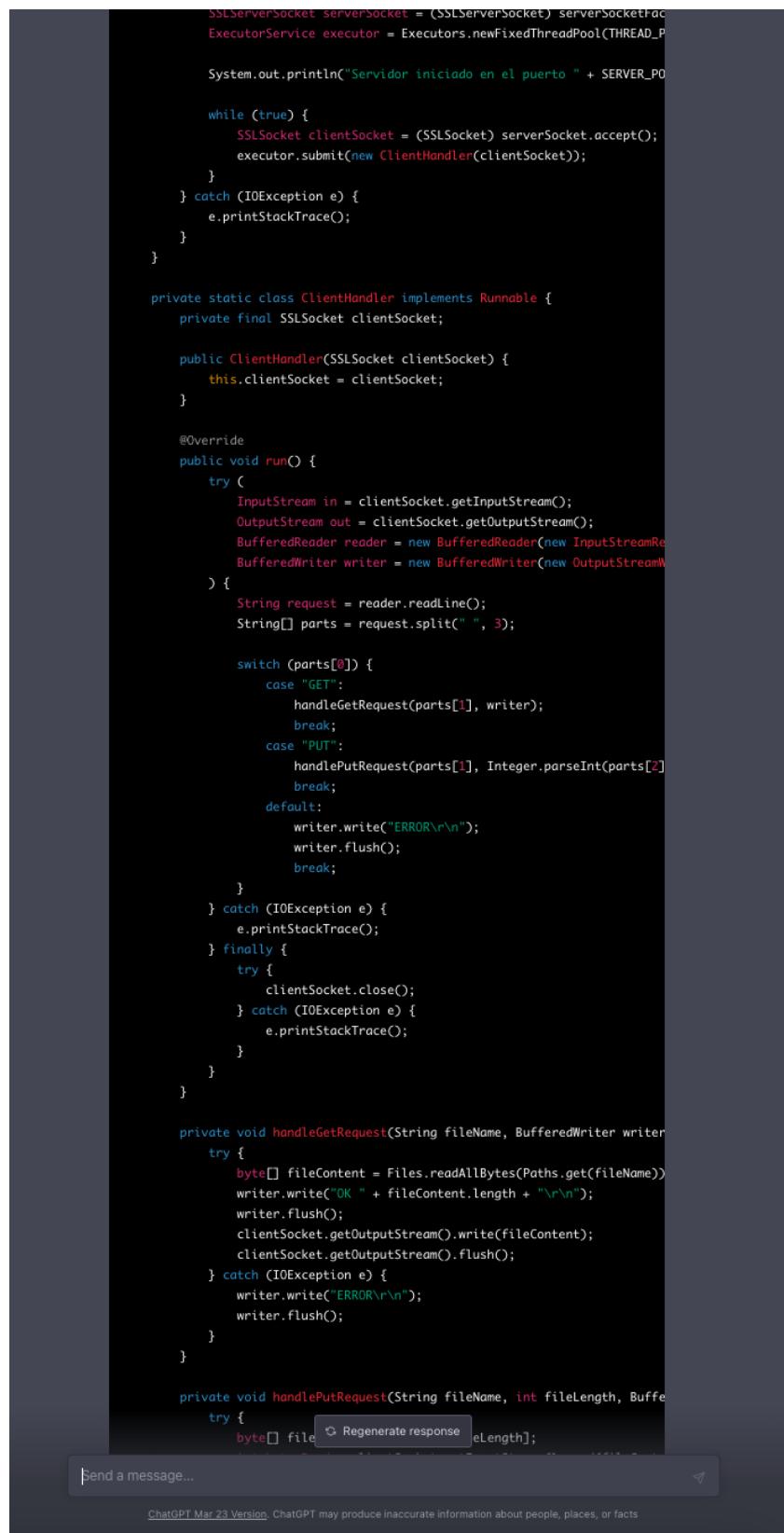
```
public class SecureServer {
    private static final int SERVER_PORT = 8443;
    private static final int THREAD_POOL_SIZE = 10;

    public static void main(String[] args) {
        try {
            SSLServerSocketFactory serverSocketFactory = (SSLServerSocketFactory) SSLSocket.class.getConstructor(
                ExecutorService executor = Executors.newFixedThreadPool(THREAD_POOL_SIZE);

            System.out.println("Servidor iniciado en el puerto " + SERVER_PORT);
            while (true) {
                Regenerate response
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Send a message...

ChatGPT Mar 23 Version. ChatGPT may produce inaccurate information about people, places, or facts



```
SSLServerSocket serverSocket = (SSLServerSocket) serverSocketFactory.createServerSocket();
ExecutorService executor = Executors.newFixedThreadPool(THREAD_POOL_SIZE);

System.out.println("Servidor iniciado en el puerto " + SERVER_PORT);

while (true) {
    SSLSocket clientSocket = (SSLSocket) serverSocket.accept();
    executor.submit(new ClientHandler(clientSocket));
}
} catch (IOException e) {
    e.printStackTrace();
}

private static class ClientHandler implements Runnable {
    private final SSLSocket clientSocket;

    public ClientHandler(SSLSocket clientSocket) {
        this.clientSocket = clientSocket;
    }

    @Override
    public void run() {
        try {
            InputStream in = clientSocket.getInputStream();
            OutputStream out = clientSocket.getOutputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(in));
            BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(out));
        } {
            String request = reader.readLine();
            String[] parts = request.split(" ", 3);

            switch (parts[0]) {
                case "GET":
                    handleGetRequest(parts[1], writer);
                    break;
                case "PUT":
                    handlePutRequest(parts[1], Integer.parseInt(parts[2]));
                    break;
                default:
                    writer.write("ERROR\r\n");
                    writer.flush();
                    break;
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                clientSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

private void handleGetRequest(String fileName, BufferedWriter writer)
    try {
        byte[] fileContent = Files.readAllBytes(Paths.get(fileName));
        writer.write("OK " + fileContent.length + "\r\n");
        writer.flush();
        clientSocket.getOutputStream().write(fileContent);
        clientSocket.getOutputStream().flush();
    } catch (IOException e) {
        writer.write("ERROR\r\n");
        writer.flush();
    }
}

private void handlePutRequest(String fileName, int fileLength, Bufferd
    try {
        byte[] file
    } Regenerate response
    eLength];
}
```

Send a message...

ChatGPT Mar 23 Version. ChatGPT may produce inaccurate information about people, places, or facts

```
}

private void handlePutRequest(String fileName, int fileLength, Buffer
    try {
        byte[] fileContent = new byte[fileLength];
        int bytesRead = clientSocket.getInputStream().read(fileContent);
        if (bytesRead != fileLength) {
            throw new IOException("No se pudo leer la longitud esperada");
        }
        Files.write(Paths.get(fileName), fileContent);
        writer.write("OK\r\n");
        writer.flush();
    } catch (IOException e) {
        writer.write("ERROR\r\n");
        writer.flush();
    }
}
}
```

Este código crea un servidor multi-thread con sockets seguros que escucha en el puerto 8443. Las peticiones GET y PUT son manejadas como se describió en el planteamiento. Por favor, asegúrate de tener configurado correctamente tu almacén de claves (keystore) y almacén de confianza (truststore) en tu entorno de ejecución Java.

HU excelente! oye, creo que no tengo configurado mi almacén de llaves ni almacén de confianza, me ayudas con los comandos necesarios? gracias!

Claro, te puedo guiar en la generación de un almacén de claves y un almacén de confianza usando la herramienta `keytool` incluida en el JDK de Java. Primero, vamos a generar un almacén de claves con un certificado auto-firmado:

1. Abre la línea de comandos o terminal en tu computadora.
2. Ejecuta el siguiente comando para generar un almacén de claves con un certificado auto-firmado:

```
bash Copy code
keytool -genkeypair -alias myserver -keyalg RSA -keysize 2048 -validity 365
```

Reemplaza `mykeystorepass` y `mykeypass` con contraseñas seguras para tu almacén de claves y clave privada, respectivamente. Ajusta los valores en el campo `-dname` para que coincidan con la información de tu organización y servidor.

Una vez que hayas ejecutado este comando, se generará un archivo llamado `keystore.jks` en el directorio actual.

Ahora, vamos a exportar el certificado de este almacén de claves para poder importarlo en nuestro almacén de confianza:

```
bash Copy code
keytool -exportcert -alias myserver -keystore keystore.jks -storepass mykeys
```

Esto generará un archivo llamado `myserver.cer` en el directorio actual.

A continuación, vamos a crear un almacén de confianza e importar el certificado que acabamos de exportar:

```
bash Regenerate response Copy code
Send a message...
```

ChatGPT Mar 23 Version. ChatGPT may produce inaccurate information about people, places, or facts

Esto generará un archivo llamado `myserver.cer` en el directorio actual.

A continuación, vamos a crear un almacén de confianza e importar el certificado que acabamos de exportar:

```
bash Copy code
keytool -importcert -alias myserver -file myserver.cer -keystore truststore.
```

Reemplaza `mytruststorepass` con una contraseña segura para tu almacén de confianza.

Ahora que tienes tus archivos `keystore.jks` y `truststore.jks`, debes configurar tu entorno de ejecución Java para usarlos. Puedes hacerlo proporcionando los siguientes argumentos al ejecutar tu aplicación Java:

```
bash Copy code
java -Djavax.net.ssl.keyStore=keystore.jks -Djavax.net.ssl.keyStorePassword=
```

Reemplaza `mykeystorepass`, `mytruststorepass` y `your-application.jar` con las contraseñas que estableciste y el archivo JAR de tu aplicación, respectivamente.

De esta manera, tu servidor Java estará configurado para usar el almacén de claves y el almacén de confianza que acabamos de crear.

 excelente! ahora me podrías dar el código del cliente GET?

 Aquí tienes el código del cliente GET con sockets seguros que cumple con los requisitos descritos en tu planteamiento:

```
java Copy code
import javax.net.ssl.*;
import java.io.*;
import java.net.Socket;
import java.nio.file.Files;
import java.nio.file.Paths;

public class SecureFileClientGet {

    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Uso: java SecureFileClientGet <IP del servidor> <puerto> <nombre de archivo>");
            return;
        }

        String serverIp = args[0];
        int serverPort = Integer.parseInt(args[1]);
        String fileName = args[2];

        try {
            SSLSocketFactory socketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
            SSLSocket socket = (SSLSocket) socketFactory.createSocket(serverIp, serverPort);

            try {
                InputStream in = socket.getInputStream();
                OutputStream out = socket.getOutputStream();
                BufferedReader reader = new BufferedReader(new InputStreamReader(in));
                BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(out));

                writer.write("GET /" + fileName + " HTTP/1.1\r\n");
                writer.write("Host: " + serverIp + ":" + serverPort + "\r\n");
                writer.write("User-Agent: Java/1.8.0_20\r\n");
                writer.write("Accept: */*\r\n");
                writer.write("\r\n");
                writer.flush();
            } catch (IOException e) {
                e.printStackTrace();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Send a message... 

ChatGPT Mar 23 Version. ChatGPT may produce inaccurate information about people, places, or facts

```
String fileName = args[2];

try {
    SSLSocketFactory socketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
    SSLSocket socket = (SSLocket) socketFactory.createSocket(serverAddress, port);

    try {
        InputStream in = socket.getInputStream();
        OutputStream out = socket.getOutputStream();
        BufferedReader reader = new BufferedReader(new InputStreamReader(in));
        BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(out));
    } {
        // Enviar la petición GET
        writer.write("GET " + fileName + "\r\n");
        writer.flush();

        // Leer la respuesta del servidor
        String response = reader.readLine();
        String[] responseParts = response.split(" ", 2);

        if ("OK".equals(responseParts[0])) {
            // Leer el archivo del servidor
            int fileLength = Integer.parseInt(responseParts[1]);
            byte[] fileContent = new byte[fileLength];
            int bytesRead = in.read(fileContent);

            if (bytesRead != fileLength) {
                throw new IOException("No se pudo leer la longitud esperada");
            }

            // Escribir el archivo en el disco local
            Files.write(Paths.get(fileName), fileContent);
            System.out.println("El archivo se recibió con éxito");
        } else {
            System.err.println("El servidor no pudo enviar el archivo");
        }
    } finally {
        socket.close();
    }
} catch (IOException e) {
    e.printStackTrace();
}
}
```

Este código crea un cliente GET con sockets seguros que se conecta al servidor y envía una petición GET para recibir un archivo. Si el servidor responde OK, el cliente recibe el archivo y lo guarda en el disco local. Asegúrate de configurar correctamente tu almacén de confianza (truststore) en tu entorno de ejecución Java antes de ejecutar este cliente.

Para ejecutar este cliente, puedes usar el siguiente comando, reemplazando las contraseñas y parámetros según sea necesario:

```
bash                                     Copy code
java -Djavax.net.ssl.trustStore=truststore.jks -Djavax.net.ssl.trustStorePass=mypassword
```

Reemplaza `mytruststorepass` con la contraseña que estableciste para tu almacén de confianza.

HU excelente! ahora quisiera porfavor el cliente PUT, muchas gracias!



Aquí tienes el código del cliente PUT que cumple con los requisitos descritos en tu planteamiento:

Send a message...



ChatGPT Mar 23 Version. ChatGPT may produce inaccurate information about people, places, or facts

The screenshot shows a ChatGPT interface. At the top, a message from 'HU' says: 'excellent! ahora quisiera porfavor el cliente PUT, muchas gracias!' Below it, a response from 'ChatGPT' provides the Java code for a secure file client:

```
java
import javax.net.ssl.*;
import java.io.*;
import java.net.Socket;
import java.nio.file.Files;
import java.nio.file.Paths;

public class SecureFileClientPut {

    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Usage: java SecureFileClientPut <IP del servidor> <puerto> <nombre archivo>");
            return;
        }

        String serverIp = args[0];
        int serverPort = Integer.parseInt(args[1]);
        String fileName = args[2];

        try {
            // Leer el archivo del disco local
            byte[] fileContent = Files.readAllBytes(Paths.get(fileName));

            SSLSocketFactory socketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
            SSLSocket socket = (SSLSocket) socketFactory.createSocket(serverIp, serverPort);

            try {
                InputStream in = socket.getInputStream();
                OutputStream out = socket.getOutputStream();
                BufferedReader reader = new BufferedReader(new InputStreamReader(in));
                BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(out));

                writer.write("PUT " + fileName + " " + fileContent.length + "\r\n");
                writer.flush();

                out.write(fileContent);
                out.flush();

                // Leer la respuesta del servidor
                String response = reader.readLine();

                if ("OK".equals(response)) {
                    System.out.println("El archivo fue recibido por el servidor");
                } else {
                    System.err.println("El servidor no pudo escribir el archivo");
                }
            } finally {
                socket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Below the code, a note states: 'Este código crea un cliente PUT con sockets seguros que lee un archivo del disco local, se conecta al servidor y envía una petición PUT con el nombre del archivo, la longitud del archivo y el contenido del archivo'. A checkbox labeled 'Regenerate response' is present. At the bottom, there's a message input field and a footer note: 'ChatGPT Mar 23 Version. ChatGPT may produce inaccurate information about people, places, or facts'.

Este código crea un cliente PUT con sockets seguros que lee un archivo del disco local, se conecta al servidor y envía una petición PUT con el nombre del archivo, la longitud del archivo y el contenido del archivo. Asegúrate de configurar correctamente tu almacén de confianza (truststore) en tu entorno de ejecución Java antes de ejecutar este cliente.

Para ejecutar este cliente, puedes usar el siguiente comando, reemplazando las contraseñas y parámetros según sea necesario:

```
bash Copy code
java -Djavax.net.ssl.trustStore=truststore.jks -Djavax.net.ssl.trustStorePas
```

Reemplaza `mytruststorepass` con la contraseña que estableciste para tu almacén de confianza.

HU muchísimas gracias!

me podrías ayudar con un Makefile que tenga todas las tareas de compilación, ejecución y limpieza para el proyecto en general? quisiera que me permita ejecutarlos individualmente y dar parámetros en la consola, algo así

make runServer 8443

si te fijas especifiqué el puerto

algo así, gracias!



Claro, puedo mostrarte cómo crear un Makefile para tu proyecto Java que incluya tareas para compilar, ejecutar y limpiar los archivos generados. Aquí tienes un ejemplo de Makefile que puedes adaptar a tu proyecto:

```
makefile Copy code
JC = javac
JFLAGS = -g
SRC = $(wildcard *.java)
CLASSES = $(SRC:.java=.class)

default: compile

compile: ${CLASSES}

%.class: %.java
    $(JC) $(JFLAGS) $<

runServer: compile
    @echo "Ejecutando el servidor en el puerto $(PORT)..."
    @java -Djavax.net.ssl.keyStore=keystore.jks -Djavax.net.ssl.keyStorePass

runClientGet: compile
    @echo "Ejecutando el cliente GET..."
    @java -Djavax.net.ssl.trustStore=truststore.jks -Djavax.net.ssl.trustSto

runClientPut: compile
    @echo "Ejecutando el cliente PUT..."
    @java -Djavax.net.ssl.trustStore=truststore.jks -Djavax.net.ssl.trustSto

clean:
    $(RM) *.class
```

Este Makefile define tareas para compilar todos los archivos `.java`, ejecutar el servidor y los clientes, y limpiar los archivos generados. Para ejecutar cada tarea, puedes usar el comando `make` seguido del nombre de la tarea necesaria.

Send a message...

ChatGPT Mar 23 Version. ChatGPT may produce inaccurate information about people, places, or facts

```
compiling: %<%.class>
%.class: %.java
    $(JAVAC) $(JFLAGS) $<

runServer: compile
    @echo "Ejecutando el servidor en el puerto $(PORT)..."
    @java -Djavax.net.ssl.keyStore=keystore.jks -Djavax.net.ssl.keyStorePass=123456 -Djavax.net.ssl.trustStore=truststore.jks -Djavax.net.ssl.trustStorePass=123456 -Djava.security.egd=file:/dev/./urandom SecureFileServer $(PORT)

runClientGet: compile
    @echo "Ejecutando el cliente GET..."
    @java -Djavax.net.ssl.trustStore=truststore.jks -Djavax.net.ssl.trustStorePass=123456 -Djava.security.egd=file:/dev/./urandom SecureFileClientGet $(IP) $(PORT) $(FILE)

runClientPut: compile
    @echo "Ejecutando el cliente PUT..."
    @java -Djavax.net.ssl.trustStore=truststore.jks -Djavax.net.ssl.trustStorePass=123456 -Djava.security.egd=file:/dev/./urandom SecureFileClientPut $(IP) $(PORT) $(FILE)

clean:
    $(RM) *.class
```

Este Makefile define tareas para compilar todos los archivos `*.java`, ejecutar el servidor y los clientes, y limpiar los archivos generados. Para ejecutar cada tarea, puedes usar el comando `make` seguido del nombre de la tarea y los parámetros necesarios.

Para ejecutar el servidor en un puerto específico, usa el siguiente comando:

```
bash Copy code
make runServer PORT=8443
```

Para ejecutar el cliente GET, usa el siguiente comando:

```
bash Copy code
make runClientGet IP=192.168.1.100 PORT=8443 FILE=example.txt
```

Reemplaza `192.168.1.100`, `8443` y `example.txt` con la IP del servidor, el puerto y el nombre del archivo según sea necesario.

Para ejecutar el cliente PUT, usa el siguiente comando:

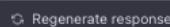
```
bash Copy code
make runClientPut IP=192.168.1.100 PORT=8443 FILE=example.txt
```

Reemplaza `192.168.1.100`, `8443` y `example.txt` con la IP del servidor, el puerto y el nombre del archivo según sea necesario.

Para limpiar los archivos generados, simplemente ejecuta:

```
bash Copy code
make clean
```

Asegúrate de reemplazar los nombres de las clases `SecureFileServer`, `SecureFileClientGet` y `SecureFileClientPut` en el Makefile si has utilizado nombres diferentes para tus archivos de código fuente. Además, no olvides ajustar las contraseñas del almacén de claves y el almacén de confianza según las contraseñas que hayas establecido.

 Regenerate response

Send a message...



ChatGPT Mar.23 Version. ChatGPT may produce inaccurate information about people, places, or facts

Ejecución del Programa

Antes de ejecutar nuestro programa, debemos considerar la funcionalidad que queremos probar:

- Que el **ClienteGET** pueda obtener un archivo desde el **Servidor**.
- Que el **ClientePUT** pueda subir un archivo a el **Servidor**.

Para probar estas funcionalidades, usaremos el siguiente flujo:

1. Mostraremos los contenidos de la carpeta del **Servidor**.
2. Ejecutaremos el **Servidor** en el puerto **8443**.
3. Mostraremos los contenidos de la carpeta dónde se encuentra el **ClientePUT**.
4. Usaremos el **ClientePUT** para subir un archivo llamado: **hola.txt** al **Servidor**.
5. Finalizaremos la ejecución del **Servidor**.
6. Mostraremos los contenidos de la carpeta del **Servidor** y el contenido del archivo **hola.txt** (si se recibió correctamente).
7. Iniciaremos de vuelta el **Servidor** en el puerto **8443**.
8. Mostraremos los contenidos de la carpeta dónde se encuentra el **ClienteGET**.
9. Usaremos el **ClienteGET** para obtener el archivo: **hola.txt**.
10. Finalizaremos la ejecución del **Servidor**.
11. Mostraremos los contenidos de la carpeta del **ClienteGET** y, de encontrarse, mostraremos el contenido del archivo: **hola.txt**.

Para la ejecución se usará la siguiente distribución de ventanas:

- Costado izquierdo: se ejecutará el **Servidor** desde **T4-2016630495** mediante **SSH**.
- Costado derecho.
 - Parte superior: será la ventana correspondiente al **ClienteGET**.
 - Parte inferior: será la ventana correspondiente al **ClientePUT**.

Nota: para obtener esta configuración de ventanas utilice tmux + un tema gráfico que se llama Dracula.

La ejecución final se observa de la siguiente manera:

```
humberto@T4-2016630495:~$ javac *.java && ls
'Servidor$ClientHandler.class'  Servidor.class  Servidor.java  certificado_servidor.pem  keystore_servidor.jks
humberto@T4-2016630495:~$ java Servidor
Servidor iniciado en el puerto 8443
Cliente conectado desde 189.132.94.124
Archivo hola.txt recibido
Cliente desconectado
^humberto@T4-2016630495:~$ ls
'Servidor$ClientHandler.class'  Servidor.java      hola.txt
  Servidor.class                certificado_servidor.pem  keystore_servidor.jks
humberto@T4-2016630495:~$ cat hola.txt
¡hola profesor! ¡qué calor tan bárbaro! ¿no cree?
humberto@T4-2016630495:~$ java Servidor
Servidor iniciado en el puerto 8443
Cliente conectado desde 189.132.94.124
Archivo hola.txt enviado
Cliente desconectado
```



```
> javac *.java && ls
ClienteGET.class  ClienteGET.java
> java ClienteGET 20.120.100.96 8443 hola.txt
El archivo se recibió con éxito
> ls && cat hola.txt
ClienteGET.class  ClienteGET.java  hola.txt
```

	File: hola.txt
1	¡hola profesor! ¡qué calor tan bárbaro! ¿no cree?

~/Proyectos/IPN/sistemas-distribuidos-escom/tareas/tarea-4/ClienteGET tarea-4* 16:15:30

>


```
> javac *.java && ls
ClientePUT.class  ClientePUT.java
> echo "¡hola profesor! ¡qué calor tan bárbaro! ¿no cree?" >hola.txt && ls && cat hola.txt
ClientePUT.class  ClientePUT.java  hola.txt
```

	File: hola.txt
1	¡hola profesor! ¡qué calor tan bárbaro! ¿no cree?

~/Proyectos/IPN/sistemas-distribuidos-escom/tareas/tarea-4/ClientePUT tarea-4* 16:14:20

>

Conclusión

Para realizar esta Tarea fue necesario aprender un poco sobre cómo funcionan los keystores en Java y cómo se puede configurar el programa para leer los archivos correspondientes de forma correcta.

Fue interesante realizar la apertura de los puertos necesarios para la ejecución de la Tarea y ver cómo es posible subir y bajar archivos usando la tarea de forma rápida y eficaz.

– Humberto Alejandro Ortega Alcocer.