# **INSTRUCTIVO AWS**

## Elementos básicos para usar la máquina virtual

### Acceder a la máquina virtual

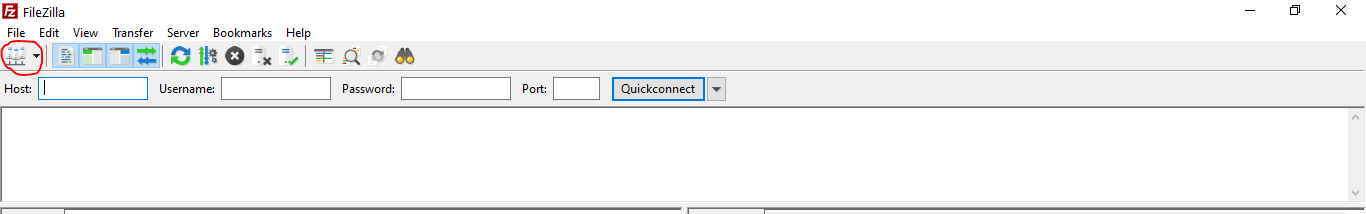
1. Ingresar a <https://us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#Instances>:
2. Log in (preguntar mail y contraseña)
3. Seleccionar la maquina *Calibration R- Nlopt4*
4. Clic en Estado de la Instancia-> Iniciar instancia
5. Una vez la instancia ha iniciado, clic en Conectar
6. En la nueva página, clic en Continuar de nuevo

Nunca olvidar al terminar de utilizar la instancia detenerla, se pueden generar cargos (dependiendo del tipo de instancia y tiempo de uso). Detenerla es equivalente a “apagar el computador”, no se pierde lo que se ha guardado en la máquina. **Para detenerla:**

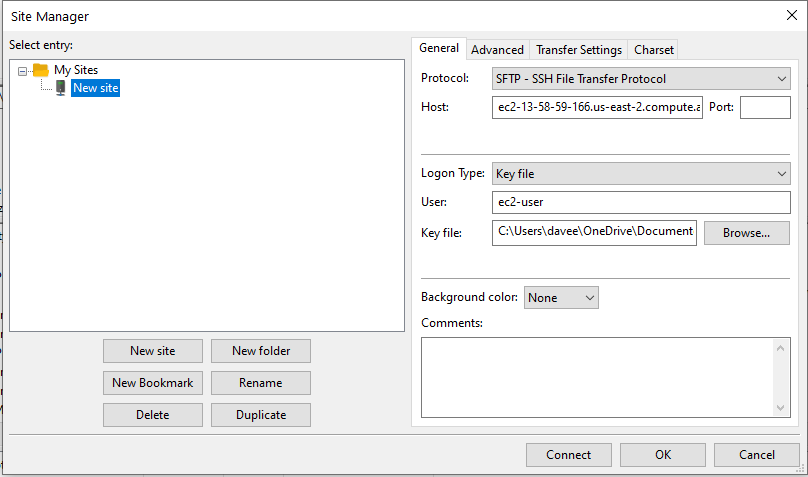
1. Volver al menú de Instancias (<https://us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#Instances>:)
2. Seleccionar la maquina *Calibration R- Nlopt4*
3. Clic en Estado de la Instancia-> Iniciar instancia

### Transferir archivos del computador a la máquina virtual o viceversa

1. Descargar FileZilla
2. Establecer la conexión con la máquina virtual
   1. Clic en “Open the Site Manager”, justo debajo de File

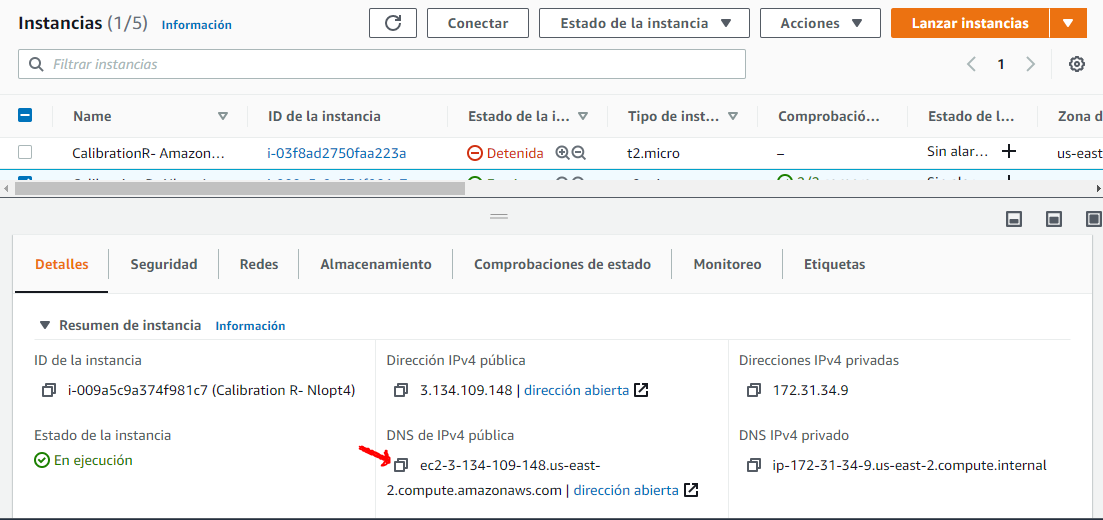


* 1. Una vez aparezca la ventana emergente, clic en New Site
  2. El nombre del nuevo sitio aparecerá sombreado, cambiarlo si se desea
  3. Diligenciar los datos solicitados de la pestaña General, de la siguiente manera:



**Protocol**: SFTP – SSH File Transfer Protocol

**Host**: Ir al menú de Instancias en AWS, y seleccionar la máquina virtual en cuestión. Luego, en el panel de información de la instancia de la parte inferior, seleccionar Detalles y copiar el “DNS de IPv4 Publica”



**Logon type**: Key File

**User**: ec2-user

**Key file**: Clic en Browse y buscar en la ubicación en la que se tenga guardada. La clave de la instancia *Calibration R- Nlopt4* está la carpeta de GitHub (Planner/Check in Aws/Keys) con el nombre *CalibrationR.pem*. En caso de olvidar el nombre de la clave, en el menú de Instancias de AWS este se muestra en la penúltima columna (mover hacia la derecha)

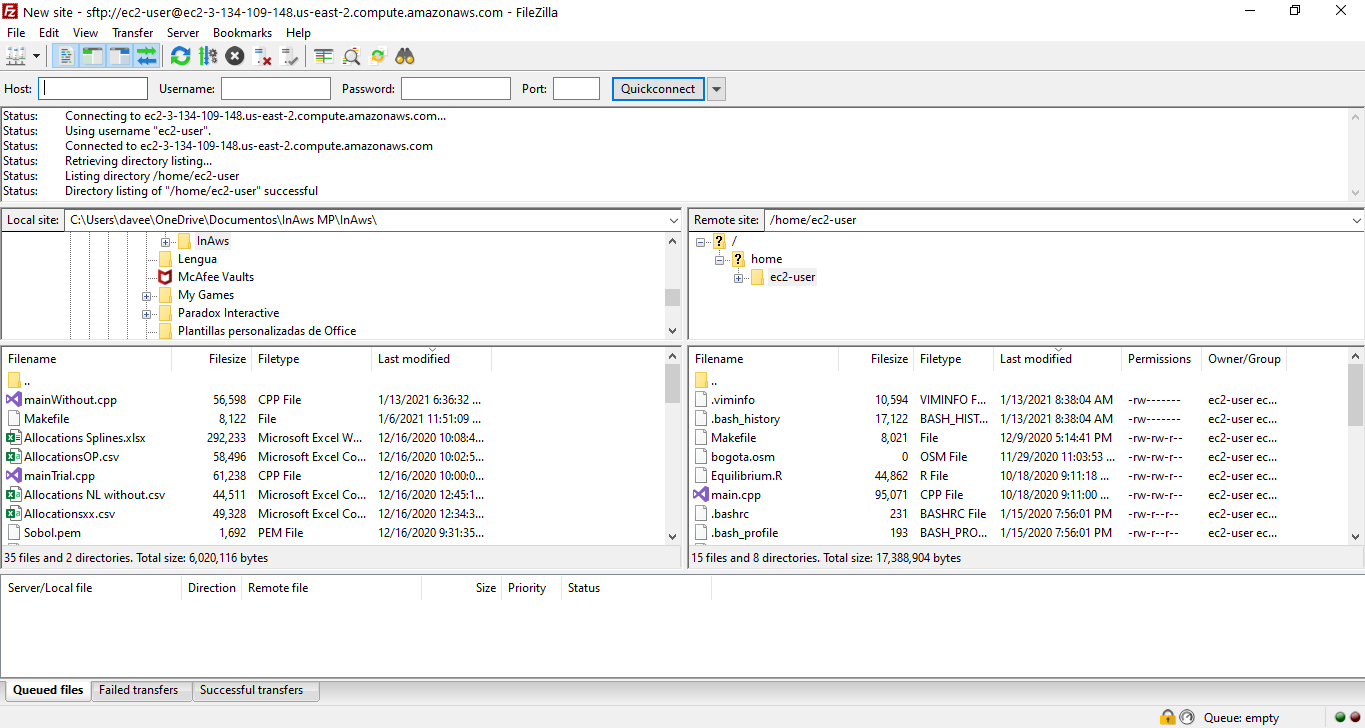
* 1. Clic en Connect

Nota: Aunque la mayoría de los datos quedan guardados, se debe actualizar el Host cada vez que se vuelva a iniciar la instancia

1. Actualizar las carpetas (del computador y de la maquina virtual). Se debe realizar cada vez que hay cambios en los archivos a transferir. Para ello, dar clic en el botón que se encuentra debajo de Server



1. FileZilla muestra dos paneles. En el lado derecho se encuentran las carpetas del computador local, en el izquierdo las de la maquina virtual. Para transferir un archivo de un lado al otro, ubicar en cada panel la carpeta de origen y de destino. Luego, hacer doble clic en el archivo a transferir



### Correr nuevos códigos

El makefile almacena diferentes comandos que queremos correr, entre ellos los que permiten correr los códigos. Para correr un nuevo código agregar al makefile:

1. Definir un nombre para el output en la sección 3 del código. Por ejemplo: OUTPUTPARALLELW=paralleloutputw
2. Definir el archivo fuente en la sección 4. Por ejemplo: SOURCEPARALLELW=mainWithout.cpp
3. Definir los comandos a correr. Los comandos que ya tenemos siguen dos partes.

Parte 1: Corre el código del archivo fuentes. Se puede seguir la siguiente estructura, cambiando el nombre del archivo fuente y del output

parallelw:

$(CC) $(SOURCEPARALLELW) $(INC\_PARAMS) $(LNLOPT) $(CFLAGS) $(DLIB) -o $(OUTPUTPARALLELW)

Parte 2: Corre la parte uno y el comando nohup (para que el proceso se mantenga en ejecución aun al salir de la terminal). Cambiar el nombre del output y del elemento a través del cual se visualizara después el output.

parallelrunw: parallelw

nohup ./$(OUTPUTPARALLELW) > nohupPARALLRW.out&

### Otros comandos útiles

cd: Definir el directorio en el cual se quiere trabajar

ls: Ver el contenido del directorio actual

vim: Visualizar un archivo (texto)

:q : Salir del visualizador de texto

## Códigos

### Problema con informalidad: mainTrial.cpp

#### ReadMe

Este código evalúa si las asignaciones óptimas derivadas del problema del Planner son implementables. Para ello, resuelve el problema descentralizado de los agentes, tomando las funciones de impuestos óptimas derivadas del problema del Planner. El código tiene las siguientes secciones:

1. Set space, general functions, and files to be included.
2. Import the planner´s output to check implementability: Toma el csv que contiene las asignaciones optimas y las funciones impuestos optimas (incluyendo la base gravable y sus derivadas) del problema del Planner resuelto en Julia. Dichos elementos quedan guardados en la matriz PLANNERINFO.
3. Create splines for taxes functions: Toma la base gravable y el impuesto correspondiente en el problema del Planner (guardados en la matriz PLANNERINFO), y crea funciones -Splines- que permiten calcular el impuesto óptimo para cualquier base gravable.
   * Para utilizar posteriormente estas funciones se requieren objetos tipo spline1dinterpolant, que contienen la información de la función. Estos objetos se crean en las secciones 2.1.1, 2.2.1 y 2.3.1. Los splines pueden ser: lineales, cúbicos o Hermite (incorporan información de la primera derivada).
   * En las secciones 2.1.2, 2.2.2 y 2.3.2 se establecen las funciones de impuestos que serán directamente utilizadas.
4. Create basic functions for the agents´ problems:
   * Contiene las funciones necesarias para el problema del trabajador y el emprendedor. Algunas de ellas son las funciones de impuestos reales (las que se usan en la calibración de los parámetros, con el sistema tributario de Perú) , pero no se usan luego.
   * Contiene las funciones del problema del trabajador (ValueWorker), y del emprendedor (FinProfits). El problema del emprendedor incluye diferentes métodos de optimización: locales (LN\_NELDERMEAD) y globales (DIRECT, COBYLA, ISRES, CRS, ESCH, StoGO y MLSL). Los métodos 1, 5, 6, 7 y 8 no permiten incluir restricciones no lineales, por lo cual no deben ser utilizados en este problema. Su “variación” bajo el NLOPT\_AUGLAG, si permite tomar las restricciones no lineales).
   * Al interior de las funciones a maximizar (o Nlopt Functions) se crean unas estructuras que permiten pasar parámetros y otros objetos necesarios para la función objetivo y las restricciones.
5. Decision vector function: Contiene la función que compara los resultados bajo el problema del trabajador y del emprendedor y define la elección ocupacional del agente.
6. Equilibrium function: Calcula las asignaciones optimas de cada agente (bajo el problema del trabajador y el emprendedor), dado un set de parámetros, asignaciones iniciales y la información del problema del Planner (funciones de impuetos y grid de thetas). Finalmente, exporta estas asignaciones a un archivo CSV.
   * En la parte 1 de la función se indica cuál es el salario de equilibrio (el mismo del problema del Planner)
   * En la parte dos se establece el conjunto de thetas sobre el cual se va a solucionar el problema. Se debe definir el argumento simdata=0 (en las funciones EquilibriumOP y OptParamsTM), para que el algoritmo tome los thetas de la matriz PLANNERINFO, y se pueda evaluar la implementabilidad. Cuando este argumento se define como 1, los thetas se simulan con una lognormal bivariada.
7. OptParams: Extrae los parámetros del archivo OptParams15.csv y con ellos corre la función EquilibriumOP.
8. main: Prueba algunas de las funciones intermedias y finalmente corre la función OptParamsTM, para calcular las asignaciones optimas, exportarlas a un csv y compararlas con el output del problema del Planner.

#### ¿Cómo correr el código en AWS?

1. Guardar en la carpeta InAws los inputs (Taxes\_info.csv y OptParamas15.csv) usando Filezilla
2. cd InAws
3. make parallelrunop
4. ls nohupPARALLR.out (para chequear que este corriendo)
5. Descargar el csv mediante Filezilla

### Problema sin informalidad: mainWithout.cpp

#### ReadMe

Este código evalúa si las asignaciones óptimas derivadas del problema del Planner (sin informalidad) son implementables. Para ello, resuelve el problema descentralizado de los agentes, tomando las funciones de impuestos óptimas derivadas del problema del Planner. El código tiene las siguientes secciones:

1. Set space, general functions, and files to be included.
2. Import the planner´s output to check implementability: Toma el csv que contiene las asignaciones optimas y las funciones impuestos optimas (incluyendo la base gravable y sus derivadas) del problema del Planner resuelto en Julia. Dichos elementos quedan guardados en la matriz PLANNERINFO.
3. Create splines for taxes functions: Toma la base gravable y el impuesto correspondiente en el problema del Planner (guardados en la matriz PLANNERINFO), y crea funciones -Splines- que permiten calcular el impuesto óptimo para cualquier base gravable.
   * Para utilizar posteriormente estas funciones se requieren objetos tipo spline1dinterpolant, que contienen la información de la función. Estos objetos se crean en las secciones 2.1.1, 2.2.1 y 2.3.1. Los splines pueden ser: lineales, cúbicos o Hermite (incorporan información de la primera derivada).
   * En las secciones 2.1.2, 2.2.2 y 2.3.2 se establecen las funciones de impuestos que serán directamente utilizadas.
4. Create basic functions for the agents´ problems:
   * Contiene las funciones necesarias para el problema del trabajador y el emprendedor. Algunas de ellas son las funciones de impuestos reales (las que se usan en la calibración de los parámetros, con el sistema tributario de Perú), pero no se usan luego.
   * Contiene las funciones del problema del trabajador (ValueWorker), y del emprendedor (FinProfits). El problema del emprendedor incluye diferentes métodos de optimización: locales (LN\_NELDERMEAD) y globales (DIRECT, COBYLA, ISRES, CRS, ESCH, StoGO y MLSL). Los métodos 1, 5, 6, 7 y 8 no permiten incluir restricciones no lineales, por lo cual no deben ser utilizados en este problema. Su “variación” bajo el NLOPT\_AUGLAG, si permite tomar las restricciones no lineales).
   * Al interior de las funciones a maximizar (o Nlopt Functions) se crean unas estructuras que permiten pasar parámetros y otros objetos necesarios para la función objetivo y las restricciones.
5. Decision vector function: Contiene la función que compara los resultados bajo el problema del trabajador y del emprendedor y define la elección ocupacional del agente.
6. Equilibrium function: Calcula las asignaciones optimas de cada agente (bajo el problema del trabajador y el emprendedor), dado un set de parámetros, asignaciones iniciales y la información del problema del Planner (funciones de impuetos y grid de thetas). Finalmente, exporta estas asignaciones a un archivo CSV.
   * En la parte 1 de la función se indica cuál es el salario de equilibrio (el mismo del problema del Planner)
   * En la parte dos se establece el conjunto de thetas sobre el cual se va a solucionar el problema. Se debe definir el argumento simdata=0 (en las funciones EquilibriumOP y OptParamsTM), para que el algoritmo tome los thetas de la matriz PLANNERINFO, y se pueda evaluar la implementabilidad. Cuando este argumento se define como 1, los thetas se simulan con una lognormal bivariada.
7. OptParams: Extrae los parámetros del archivo OptParams15.csv y con ellos corre la función EquilibriumOP.
8. main: Prueba algunas de las funciones intermedias y finalmente corre la función OptParamsTM, para calcular las asiganciones optimas, exportarlas a un csv y compararlas con el output del problema del Planner.

#### ¿Cómo correr el código en AWS?

1. Guardar en la carpeta InAws los inputs (Taxes\_info.csv y OptParamas15.csv) usando Filezilla
2. cd InAws
3. make parallelrunw
4. ls nohupPARALLRW.out (para chequear que este corriendo)
5. Descargar el csv mediante Filezilla

### Función de prueba de algoritmos de optimizacion: mainFunction

ReadMe

A través de este código se pueden explorar los algoritmos de optimización de Nlopt. Contiene un problema de optimización senciilo (una función cuadrática a la cual se le pueden agregar restricciones.

#### ¿Cómo correr el código en AWS?

1. Guardar en la carpeta InAws los inputs (Taxes\_info.csv y OptParamas15.csv) usando Filezilla
2. cd InAws
3. make tryfunctionrun
4. ls nohupNEW.out (para chequear que este corriendo)
5. Descargar el csv mediante Filezilla

### Código original: mainTesla

ReadMe

Código original de Rodrigo. Disponible también en la carpeta de Theoretical Moments.

#### ¿Cómo correr el código en AWS?

1. Guardar en la carpeta InAws los inputs (Taxes.csv y OptParamas15.csv) usando Filezilla
2. cd InAws
3. make parallelrun
4. ls nohupPARALLR.out (para chequear que este corriendo)
5. Descargar el csv mediante Filezilla