1) Descargar Selenium Server y ChromeDriver

1.1) Selenium Server 2.39

Abre el index de releases y descarga selenium-server-standalone-2.39.0.jar. (selenium-release.storage.googleapis.com)

1.2) ChromeDriver (para nodos con Chrome)

- Consulta la versión de Chrome del nodo (Menú → Ayuda → Acerca de Google Chrome).
- Descarga la versión de ChromeDriver que coincida con la versión mayor de Chrome desde la página oficial. Coloca chromedriver.exe en una carpeta accesible (por ejemplo C:\drivers\chromedriver.exe) o agrégala al PATH. (Chrome for Developers)

Consejo práctico: mantener una carpeta C:\selenium\ con el selenium-server-standalone.jar, chromedriver.exe, geckodriver.exe (si usas Gecko/Firefox), y scripts .bat para arrancar hub/nodo.

2) Levantar el HUB

2.1) En la máquina que hará de **Hub**, abre una terminal (cmd o PowerShell en Windows). Sitúate en la carpeta donde está el .jar.

2.2) Comando clásico (Selenium 2.x):

java -jar selenium-server-standalone-2.39.0.jar -role hub

Qué hace cada parte:

- java -jar ...: ejecuta el jar de Selenium.
- -role hub: indica que esta instancia será el hub (gestiona la consola y las peticiones de RemoteWebDriver).

Si arrancó bien, abre en un navegador del Hub:

http://localhost:4444/grid/console

Ahí verás la consola con nodos registrados cuando se conecten. (Esto es el flujo clásico de Grid 2/3). (selenium-release.storage.googleapis.com)

2.3) Nota (Grid 4): en Grid 4 también puedes usar el modo *standalone* con un único jar y este comando:

java -jar selenium-server-<version>.jar standalone

En Grid 4 la manera de iniciar nodos y la sintaxis cambian un poco (y la detección de drivers tiende a ser más automática), por lo que si quieres modernizarlo más adelante conviene leer la documentación oficial. (Selenium)

3) Levantar un NODO Firefox

En la máquina que actuará como nodo (puede ser la misma que el hub para pruebas locales):

3.1) Abre terminal y ve a la carpeta con selenium-server-standalone-2.39.0.jar.

3.2) Ejecuta (comando clásico):

java -jar selenium-server-standalone-2.39.0.jar -role node -hub http://<HUB_IP>:4444/grid/register

Explicación:

- -role node: arranca como nodo (acepta tareas del hub).
- -hub http://<HUB_IP>:4444/grid/register: URL donde está el hub (si hub y nodo son la misma máquina usar localhost).

Después revisa la consola del hub http://<HUB_IP>:4444/grid/console y verás el nodo y los navegadores detectados (Firefox, etc.). (selenium-release.storage.googleapis.com)

4) Levantar un NODO Chrome

Problema común: el nodo arranca pero al intentar ejecutar tests en Chrome falla porque no encuentra chromedriver. Tienes dos opciones: (A) declarar el path con una propiedad Java al arrancar el nodo, o (B) poner el chromedriver en el PATH de Windows/Linux.

4.1) Opción A — pasar la propiedad al arrancar el nodo (recomendado, explícito):

Windows (cmd):

java -Dwebdriver.chrome.driver="C:\drivers\chromedriver.exe" -jar selenium-server-standalone-2.39.0.jar -role node -hub http://<HUB_IP>:4444/grid/register -browser "browserName=chrome,version=ANY,platform=WINDOWS,maxInstances=5" -maxSession 5

Linux (ejemplo):

java -Dwebdriver.chrome.driver="/home/selenium/drivers/chromedriver" -jar selenium-server-standalone-2.39.0.jar -role node -hub http://<HUB_IP>:4444/grid/register -browser "browserName=chrome,version=ANY,platform=LINUX,maxInstances=5" -maxSession 5

Explicación detallada:

- -Dwebdriver.chrome.driver="...": informa a la JVM la ubicación del ejecutable chromedriver. **Debe colocarse antes** de -jar o justo junto al comando java (varía por versión, pero en práctica funciona con -D... -jar ...).
- -browser "browserName=chrome,version=ANY,platform=WINDOWS,maxInstances=5": le dice al nodo qué navegador (y cuántas sesiones Máx) anuncia al hub.
- -maxSession 5: número máximo total de sesiones concurrentes que el nodo acepta (útil si tienes varias instancias).

Si la ruta del driver no está bien, el nodo puede registrarse pero no aceptará pruebas para Chrome o fallarán al crear la sesión (error típico: “path to driver executable must be set”). La comunidad recomienda colocar -Dwebdriver.chrome.driver en la línea de arranque del nodo. Ver ejemplos y soluciones en StackOverflow. (Stack Overflow)

4.2) Opción B — añadir chromedriver al PATH del sistema

- Copia chromedriver.exe a C:\Windows\System32 o añade C:\drivers\ a la variable PATH. Reinicia la terminal y lanza el nodo sin -Dwebdriver.chrome.driver. Funciona, pero es menos explícito.

4.3) Verifica en la consola del hub que el nodo anuncia chrome en las opciones de navegadores disponibles.

5) Código C#

A continuación tienes el ejemplo completo (ya preparado para ejecutar en Grid). Después lo desgranamos:

```
using System;

using NUnit.Framework;

using OpenQA.Selenium;

using OpenQA.Selenium.Remote;


namespace TestOfGrid

{

  [TestFixture]

  [Parallelizable(ParallelScope.All)]

  public class Driver

  {

    private IWebDriver driver;


    [SetUp]

    public void Setup()

    {

      // URL del Hub (ajusta si está en otra máquina)

      string hubUrl = "http://localhost:4444/wd/hub";


      // Obtén el parámetro "browser" desde NUnit (por defecto "firefox")
```

```
        string navegador = TestContext.Parameters.Get("browser", "firefox");

        DesiredCapabilities capabilities;

        if (navegador.Equals("chrome", StringComparison.OrdinalIgnoreCase))
        {
            capabilities = DesiredCapabilities.Chrome();
            capabilities.SetCapability(CapabilityType.BrowserName, "chrome");
        }
        else
        {
            capabilities = DesiredCapabilities.Firefox();
            capabilities.SetCapability(CapabilityType.BrowserName, "firefox");
        }

        // Indica la plataforma (opcional)
        capabilities.SetCapability(CapabilityType.Platform, new
Platform(PlatformType.Windows));

        // Crea el RemoteWebDriver apuntando al hub (el hub delegará a un nodo)
        driver = new RemoteWebDriver(new Uri(hubUrl), capabilities);
    }

    [TearDown]
    public void Teardown()
    {
        if (driver != null)
        {
            driver.Quit();
        }
    }
```

```
    [Test]
    public void GoogleSearch()
    {
      driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
      driver.Navigate().GoToUrl("http://www.google.com");
      Assert.IsTrue(driver.Title.ToLower().Contains("google"));
    }
  }
}
```

Explicación línea a línea

- TestContext.Parameters.Get("browser", "firefox"): permite pasar --params desde el runner para seleccionar el navegador en tiempo de ejecución. Así puedes lanzar la suite una vez para Chrome y otra para Firefox.
- DesiredCapabilities: define *qué* tipo de navegador/plataforma pedimos al Hub. En Selenium 2/3 se usa así; en Selenium 4 hay APIs alternativas y ciertas cosas cambian (nota abajo).
- new RemoteWebDriver(new Uri(hubUrl), capabilities): crea una sesión remota en el hub; el hub elegirá uno de los nodos que cumpla esas capacidades.
- ImplicitWait: útil para sincronización simple; en pruebas modernas se recomienda WebDriverWait (explicit waits) para evitar esperas globales innecesarias.

6) Configurar NUnit para ejecutar pruebas en paralelo

6.1) Assembly-level (archivo AssemblyInfo.cs) — indica el nivel de paralelismo:

```
using NUnit.Framework;


[assembly: Parallelizable(ParallelScope.Fixtures)]
[assembly: LevelOfParallelism(4)]  // Ajusta según núcleos / capacidad del Grid
```

- ParallelScope.Fixtures: permite ejecutar fixtures en paralelo. ParallelScope.All en la clase permite métodos en paralelos.
- LevelOfParallelism(4): número máximo de hilos que NUnit usará; pon un número coherente con el total de instancias que tu Grid puede manejar.

6.2) Ejecutar la suite con parámetros para elegir navegador (ejemplos):

- Ejecutar toda la suite en Firefox:

nunit3-console.exe MyTests.dll --params "browser=firefox"

- Ejecutar toda la suite en Chrome:

nunit3-console.exe MyTests.dll --params "browser=chrome"

Si quieres **hacer que ambas se ejecuten al mismo tiempo en el hub** desde un mismo runner, necesitas lanzar dos procesos nunit3-console con distintos --params, o configurar un runner que cree múltiples instancias. Con LevelOfParallelism y Parallelizable activados, NUnit lanzará tests concurrentemente dentro de su propio proceso (si tus tests piden navegadores diferentes y el Grid tiene nodos adecuados, el hub asignará cada sesión a los nodos correctos).

7) Ejemplo de ejecución paralela práctica

Supongamos que tienes:

- Nodo A con Firefox (maxInstances=5).
- Nodo B con Chrome (maxInstances=5).
- Hub en 192.168.0.10.

Puedes:

1. Abrir dos consolas de runner y ejecutar:

nunit3-console.exe MyTests.dll --params "browser=firefox"

nunit3-console.exe MyTests.dll --params "browser=chrome"

2. O usar un script que llame al runner dos veces en paralelo (PowerShell):

Start-Job { & "C:\path\to\nunit3-console.exe" "C:\build\MyTests.dll" --params "browser=firefox" }

Start-Job { & "C:\path\to\nunit3-console.exe" "C:\build\MyTests.dll" --params "browser=chrome" }

Cada proceso pedirá sesiones al hub; éste las atenderá y las redirigirá a nodos compatibles.