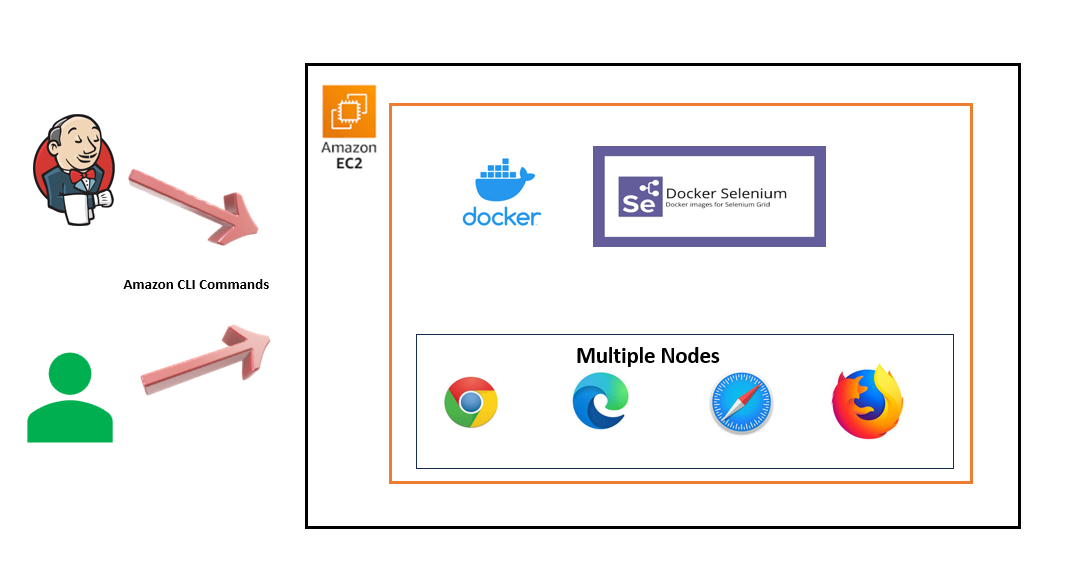
**Run your existing Selenium (Java / C#) scripts effortlessly on AWS with Just Two Lines of Code!**

Hello everyone! In this exciting use case, we'll explore running existing Selenium scripts on AWS EC2 using Docker and Selenium Grid for seamless automation. Let's dive in!

**Flow Diagram:**



Assuming you already have Selenium scripts (in any language binding) and an active AWS account, let's dive into the process of executing them.

**How to execute:**

**Step 1:** Create a file named “**user\_data.sh”** with the following content and save it in the home folder of your Amazon CLI**:**

#!/bin/bash

# Update and install Docker

sudo yum update

sudo yum install -y docker

sudo usermod -a -G docker ec2-user

sudo service docker start

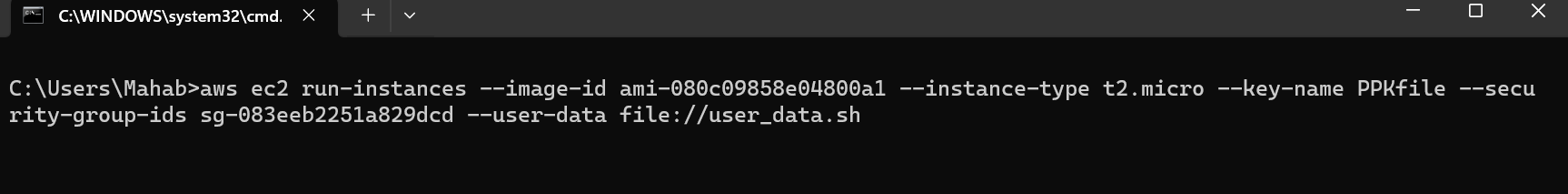
sudo chkconfig docker on

docker run -d -p 4444:4444 -p 7900:7900 selenium/standalone-chrome:latest

**Step 2:** Execute the two-liner script in Amazon CLI.

aws ec2 run-instances --image-id <<**Amazon Image ID**>> --instance-type << **Instance type**>> --key-name <<**Your key name**>> --security-group-ids <<**Security group id**>> --user-data <file://user_data.sh>

E.g.,

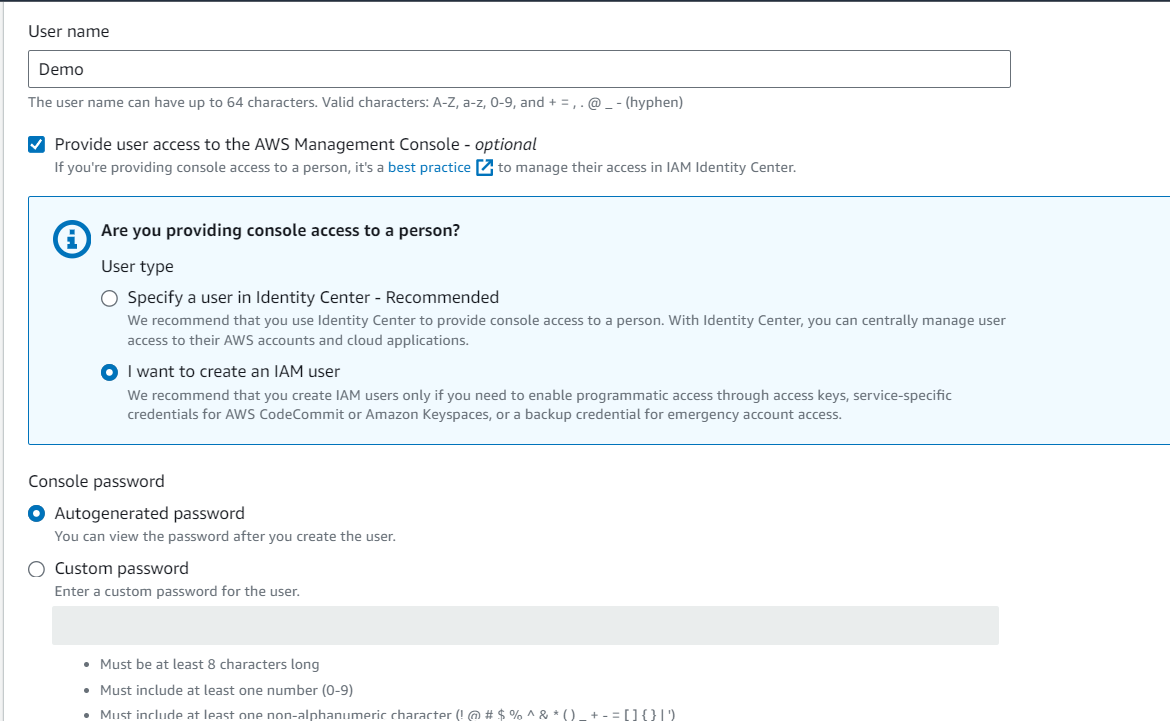


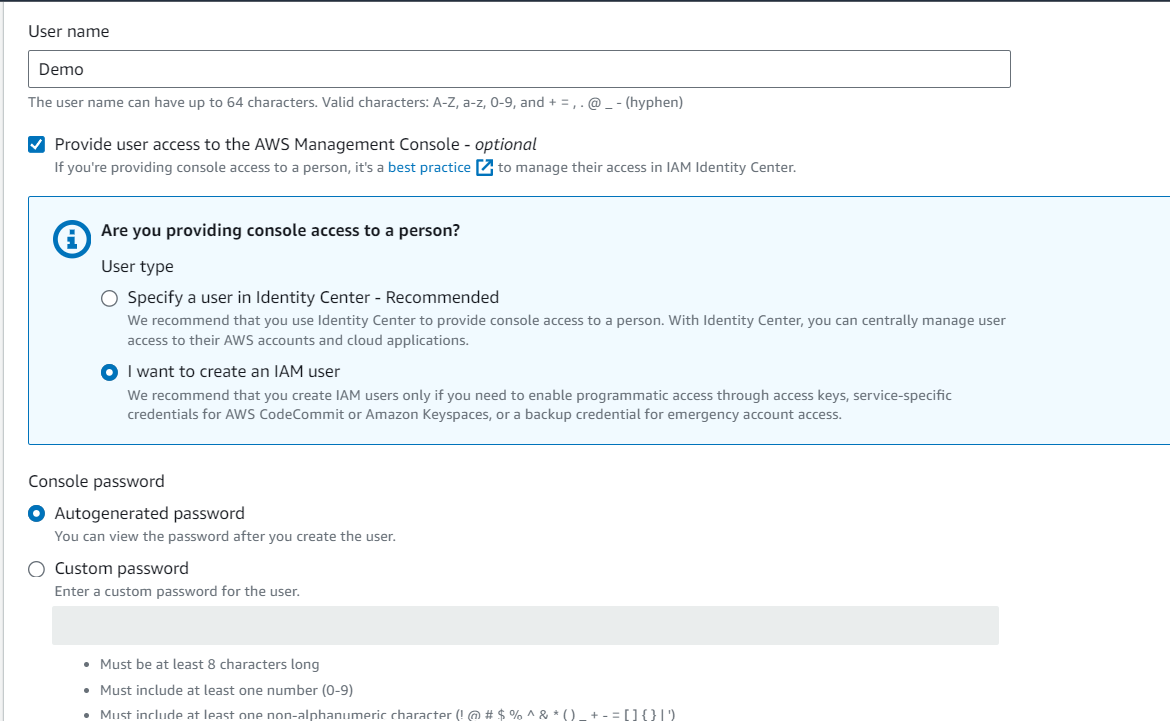
Just update the public IPV4 address in the Remote Driver URL. Your script will execute in AWS.

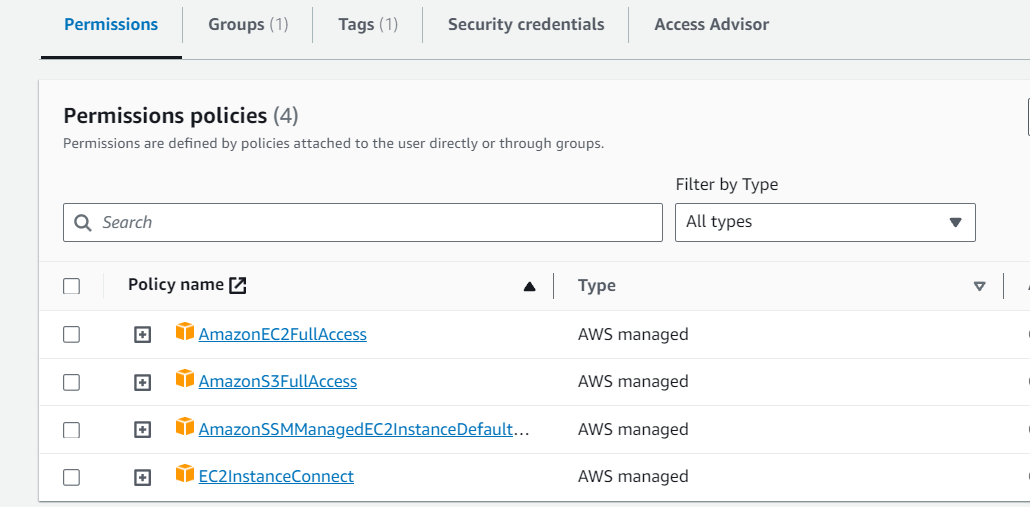
**New to AWS and eager to dive in? Keep scrolling!**

I'll provide step-by-step guidance and references to execute your Selenium scripts in AWS. While some familiarity with AWS is helpful, you can be confident even if your scripts are in different languages like Selenium, C#, Python, Ruby, or JavaScript.

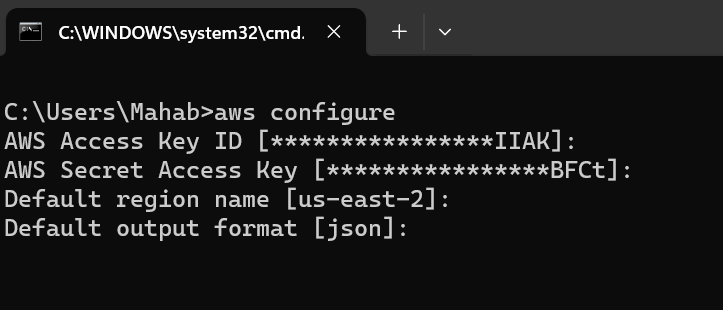
**Step 1:** Create an account in [AWS Management Console (amazon.com)](https://aws.amazon.com/console/)

**Step 2:** When logging into the Web UI Console, we use our ID and password for authentication. However, for CLI-based programs to access AWS, they require an access key and secret access key. To facilitate this, we create a new IAM user and grant it EC2FullAccess for both AWS CLI and Console (for our demo purposes).

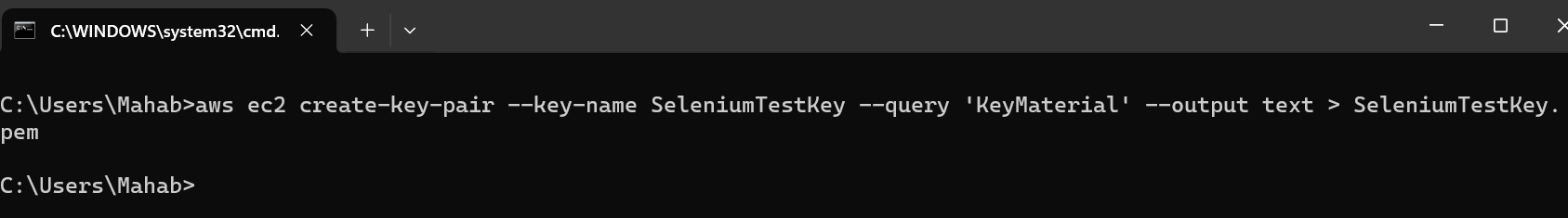




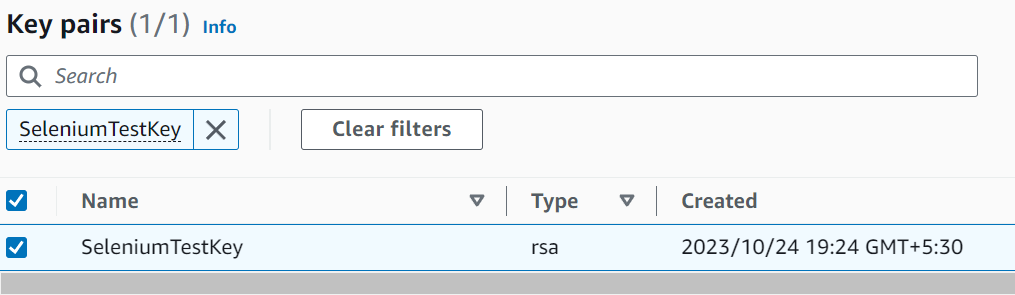
**Step 3:** Download and Install CLI (<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>) and configure AWS account.



**Step 4:** Generate a secure key for connecting to the instance. This key will be accessible in the folder after a successful command execution.



You can also view it in the EC2 instance console.



**Step 5:** Establish a Security Group to control inbound traffic for the EC2 instance.

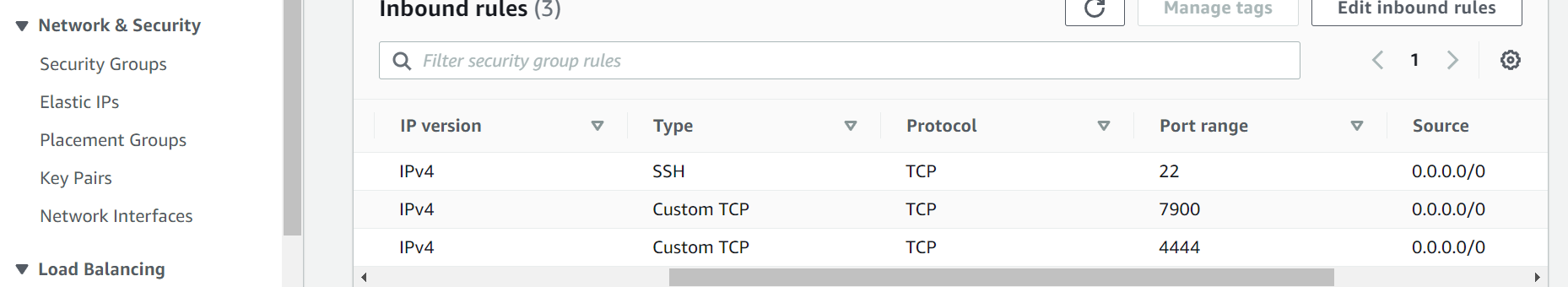
aws ec2 create-security-group --group-name SeleniumSecurityGroup --description "SecurityGroup for Selenium Grid"

aws ec2 authorize-security-group-ingress --group-name SeleniumSecurityGroup --protocol tcp --port 22 --cidr 0.0.0.0/0

aws ec2 authorize-security-group-ingress --group-name SeleniumSecurityGroup --protocol tcp --port 4444 --cidr 0.0.0.0/0

aws ec2 authorize-security-group-ingress --group-name SeleniumSecurityGroup --protocol tcp --port 7900 --cidr 0.0.0.0/0

aws ec2 describe-security-groups --group-names SeleniumSecurityGroup

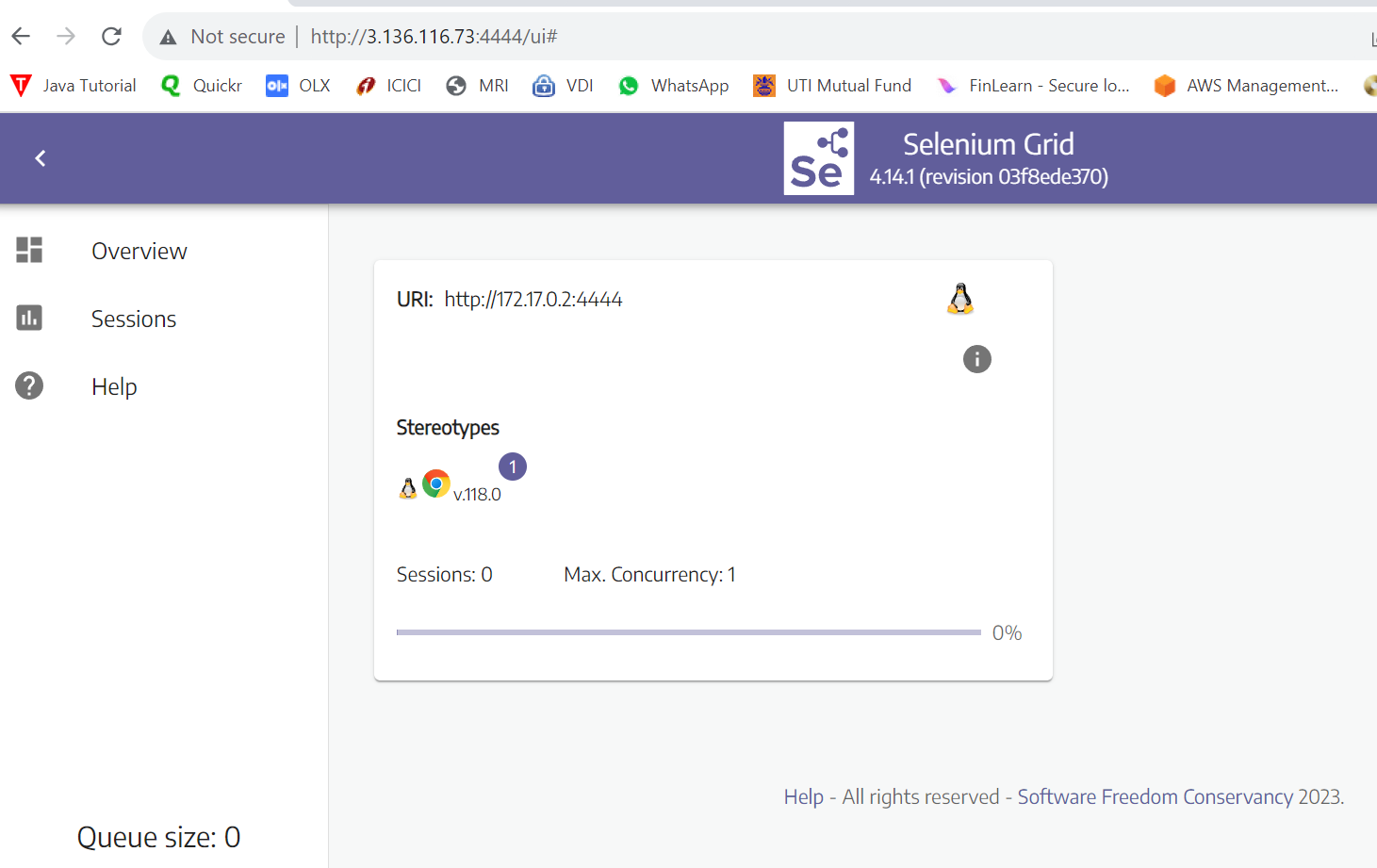


**Step 6:** Follow Step 1 and Step 2 (Described in top page)

The preceding six steps guide you through launching an EC2 Instance and configuring Selenium Grid with Docker.

Next, copy the public IPv4 address and replace the URL below with your own IP address. This step ensures that your Selenium Hub is up and running.

[http://3.136.116.73:4444/ui](http://3.21.37.44:4444/ui)



Update your Selenium script by modifying the driver initiation logic to use a Remote WebDriver and execute the test cases on AWS.

Here's the updated code in your Selenium C# project:

public IWebDriver CreateDriver()

{

string browser = Environment.GetEnvironmentVariable("BROWSER") ?? "CLOUD";

switch (browser.ToUpperInvariant())

{

case "CHROME":

return new ChromeDriver();

case "EDGE":

return new EdgeDriver();

case "CLOUD":

ChromeOptions chromeOptions = new ChromeOptions();

chromeOptions.AddArguments("ignore-certificate-error");

chromeOptions.AddArguments("disable-dev-shm-usage");

chromeOptions.AddArguments("--start-maximized");

chromeOptions.AddArguments("--headless");

chromeOptions.AddArguments("-disable\_gpu");

chromeOptions.AddArguments("-disable-extensions");

chromeOptions.PlatformName = "LINUX";

return new RemoteWebDriver(new Uri("http:// 3.136.116.73:4444/wd/hub"), chromeOptions);

default:

throw new ArgumentException($"Browser not yet implemented: {browser}");

}

}

Here's the updated code in your Selenium Java project:

public WebDriver createDriver(String browserName) {

switch (browserName.toUpperCase()) {

case "CHROME":

return new ChromeDriver();

case "EDGE":

return new EdgeDriver();

case "CLOUD":

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setBrowserName("chrome");

return new RemoteWebDriver(new URL("http://3.136.116.73:4444/wd/hub"), capabilities);

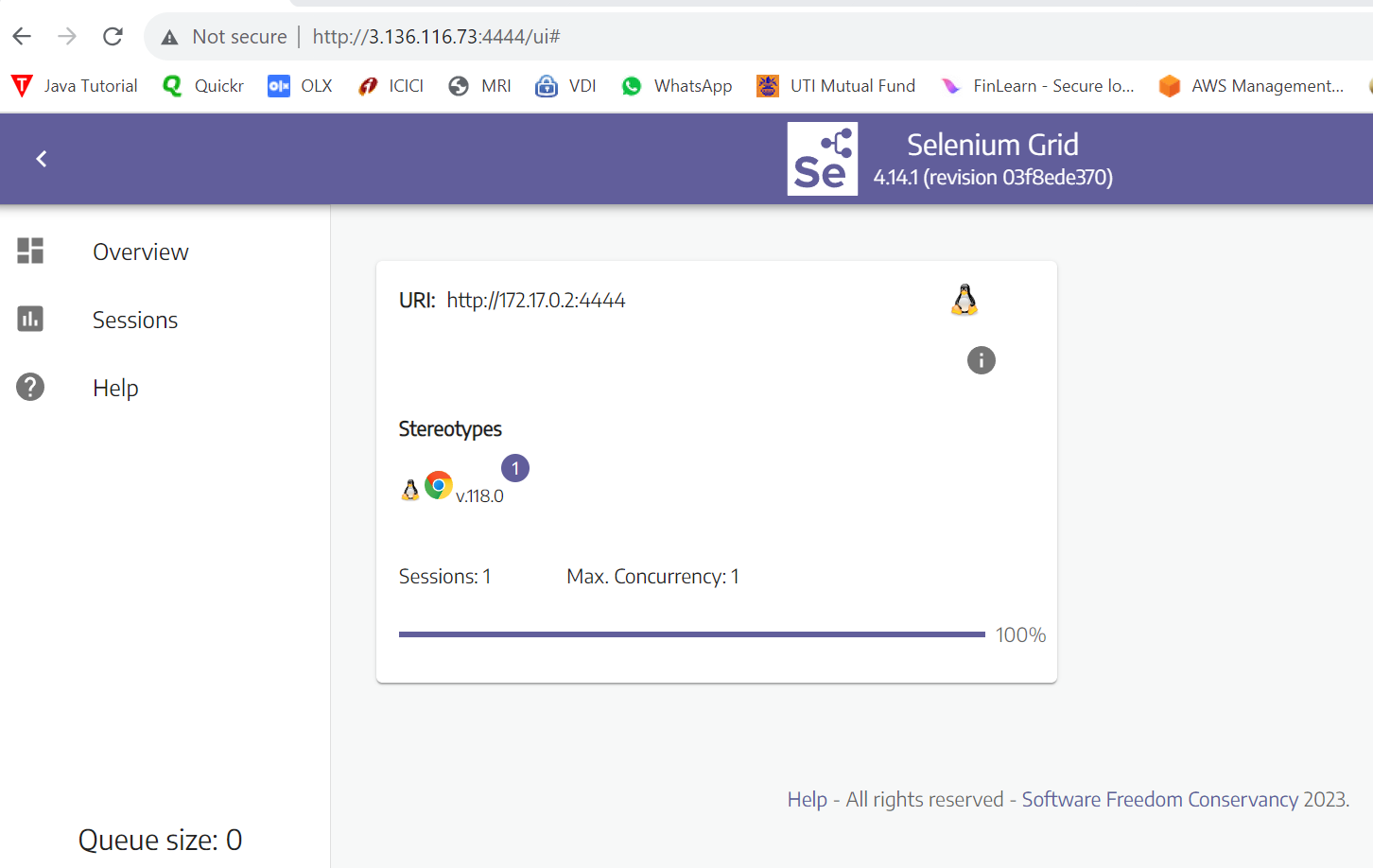
default:

throw new IllegalArgumentException(browserName + " not yet implemented");

}

}

When the execution starts the active session will be incremented from 0 to 1. Attached the screenshot for reference.



Remember to terminate your EC2 instances after use to prevent incurring additional costs.