MUHAMMAD HAMMAD SANI (BSCS-2019-56)
MUHAMMAD FARHAN (BSCS-2020-40)

CS-420 Parallel and Distributed Computing

ASSIGNMENT#1

Following are the tools that we have explored in order to take a basic level sense of how parallel computing and distributed computing works.
Our work is related to Julia programming language. Some highlighting features of Julia Programming Language are

1. It is fast
2. It is dynamically typed (C++ is static typed, meaning data type is determined at compile time)

And the highlighting feature of Julia with respect to this assignment is that it has

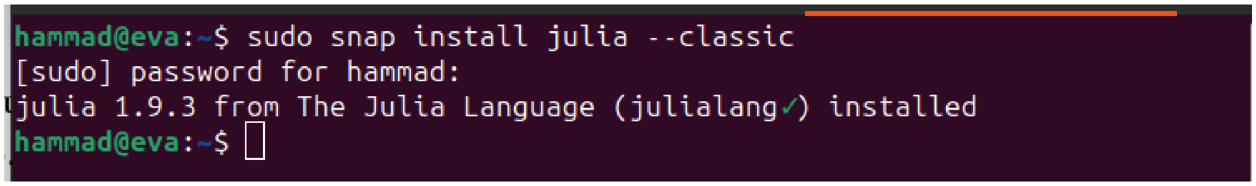
3. **Parallel and Heterogeneous Computing Nature.**
   It is designed for parallelism, and provides built-in primitives for parallel computing at every level.

We have tried on explore it on two levels

a. **Julia and its parallel nature (running multiple threads/processes in julia)**
b. **Julia on Amazon EC2 Instance**

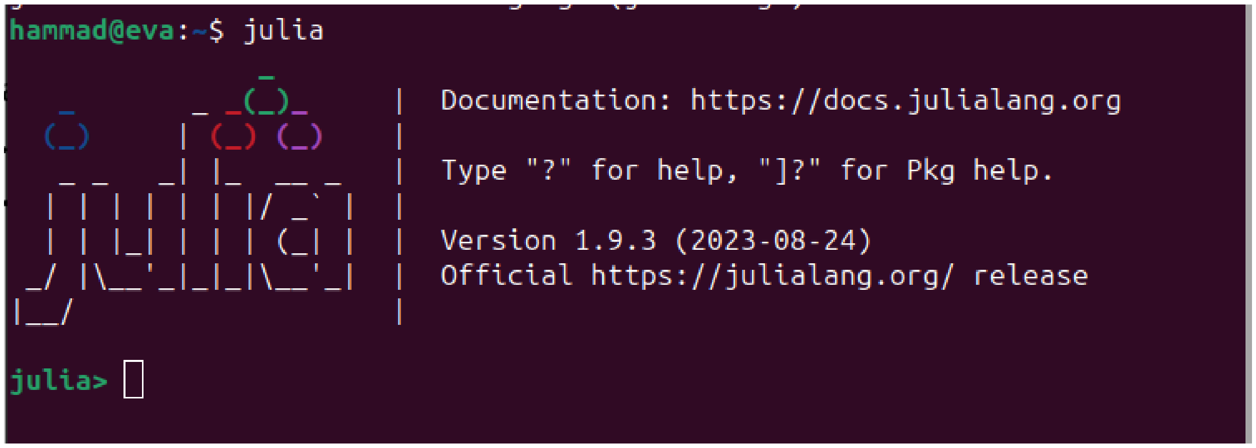
Setting Up Julia in Linux (Ubuntu Distribution)

I used "**sudo snap install julia --classic"** (Snap Package Manager is Used)



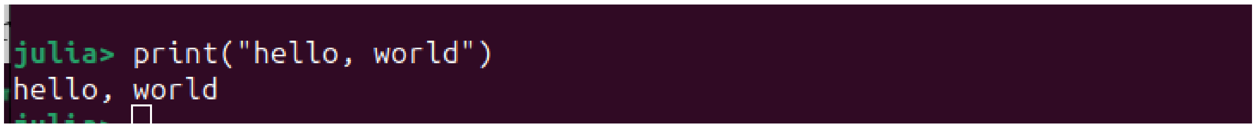
```
hammad@eva:~$ sudo snap install julia --classic
[sudo] password for hammad:
julia 1.9.3 from The Julia Language (julialang✓) installed
hammad@eva:~$
```

"**julia**" command simply opens an interactive terminal so we can write julia code and do the experimentation



```
hammad@eva:~$ julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.9.3 (2023-08-24)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia>
```

Testing julia interactive terminal by

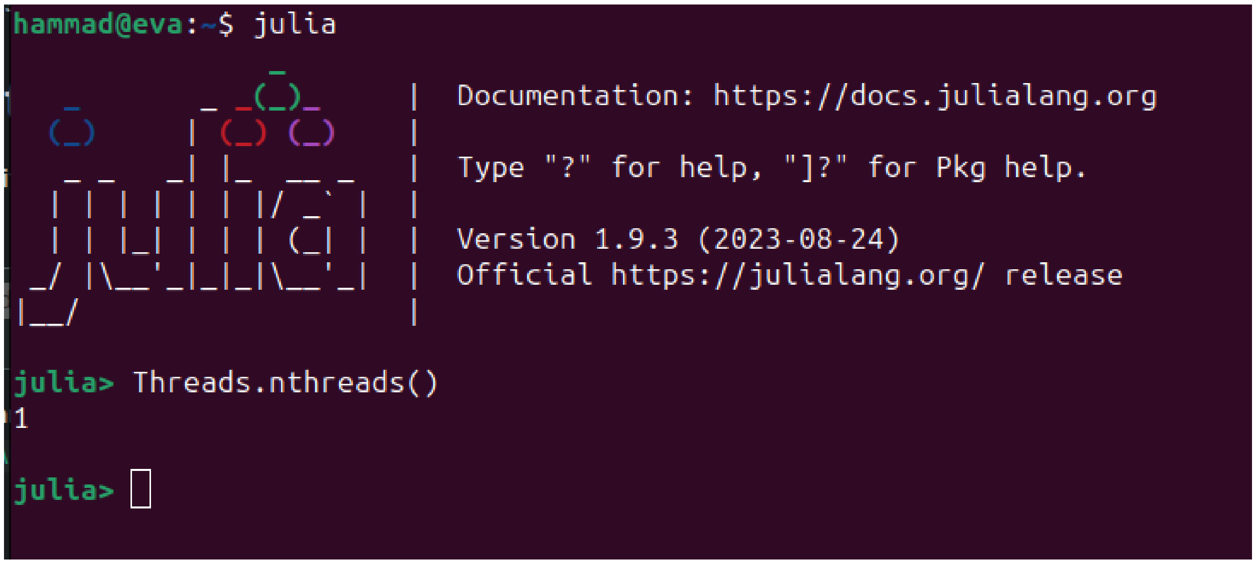


```
julia> print("hello, world")
hello, world
```

Now, exploring Julia Parallel Nature (Julia multi-core) as Tool 1

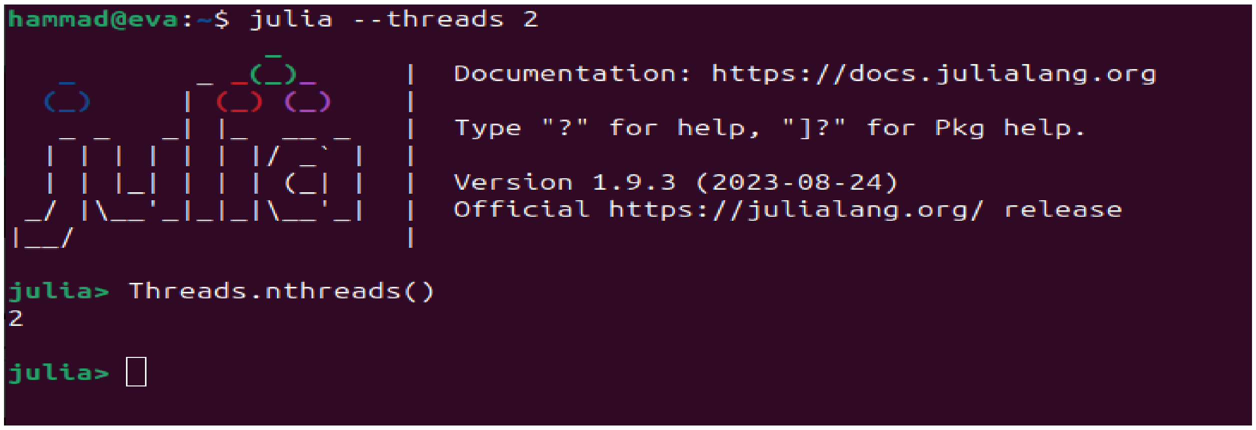
At thread Level

By default julia only creates one thread for execution (Threads.nthreads())



```
hammad@eva:~$ julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.9.3 (2023-08-24)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> Threads.nthreads()
1

julia> 
```

But we can tell Julia how many threads it should create by using a flag
- - threads x (x can be 2, 3, 4)

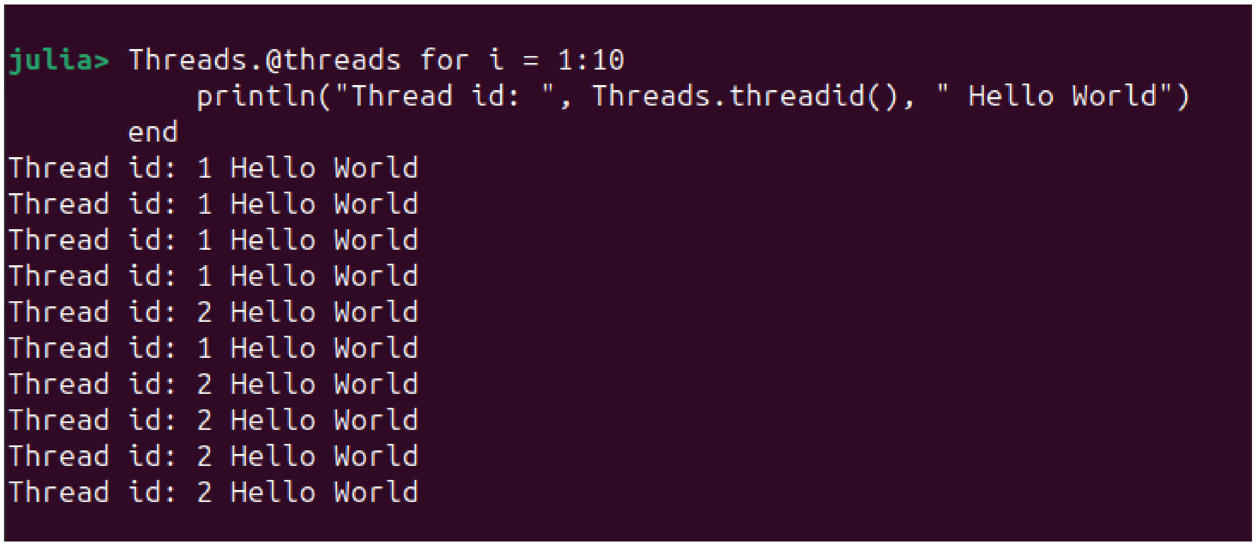


```
hammad@eva:~$ julia --threads 2
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.9.3 (2023-08-24)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> Threads.nthreads()
2

julia> 
```

Once two threads are created, we can simply run the following code

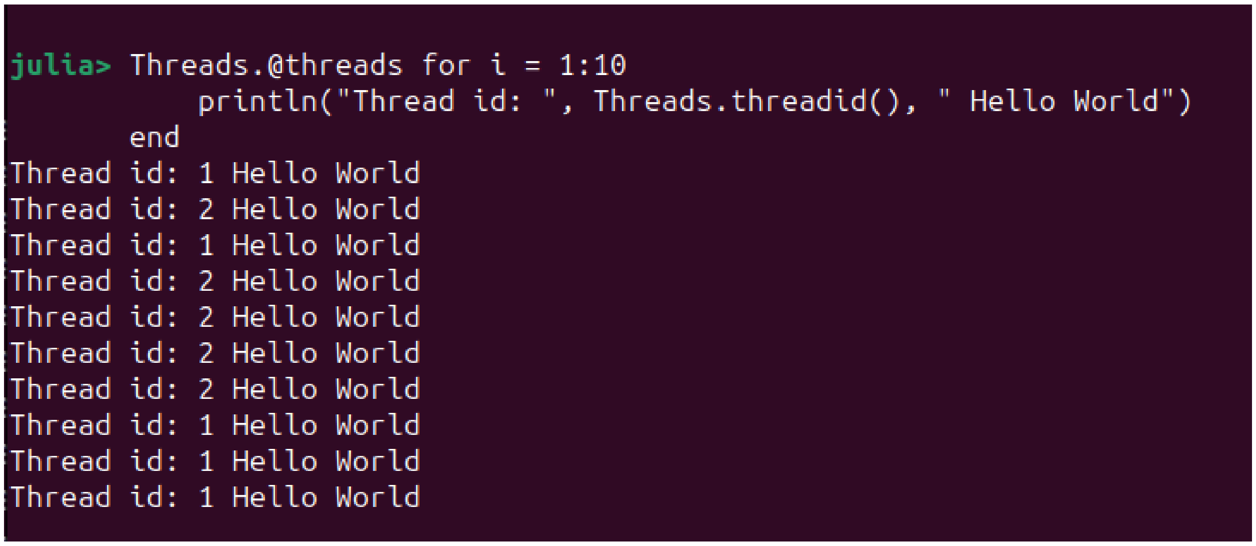
```
Threads.@threads for i = 1:10
    println("Thread id: ", Threads.threadid(), " Hello World")
end
```

The code is basically printing “Hello, World” in a for loop (executing 10 times) and also telling on which thread the line is being executed
I run this code multiple times and each time different output was generated in terms of multiple threads
One Time



```
julia> Threads.@threads for i = 1:10
           println("Thread id: ", Threads.threadid(), " Hello World")
       end
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 2 Hello World
Thread id: 1 Hello World
Thread id: 2 Hello World
Thread id: 2 Hello World
Thread id: 2 Hello World
Thread id: 2 Hello World
```

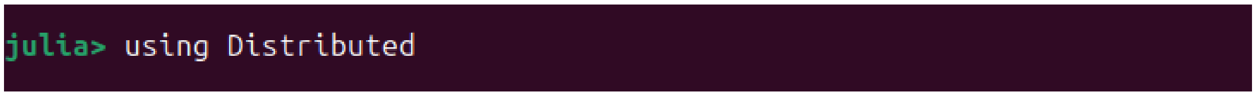
Second Time



```
julia> Threads.@threads for i = 1:10
           println("Thread id: ", Threads.threadid(), " Hello World")
       end
Thread id: 1 Hello World
Thread id: 2 Hello World
Thread id: 1 Hello World
Thread id: 2 Hello World
Thread id: 2 Hello World
Thread id: 2 Hello World
Thread id: 2 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
```

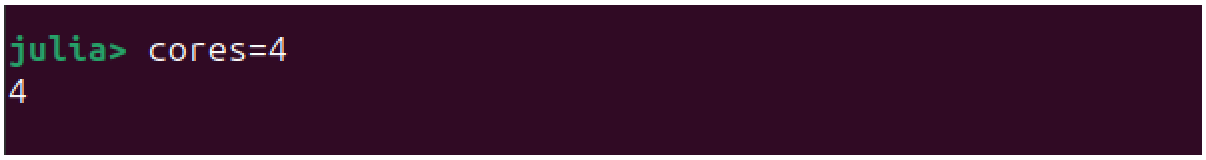
At Core Level

We can change code a bit to do it at process level (Using a module Distributed).
Using the Distributed module in julia



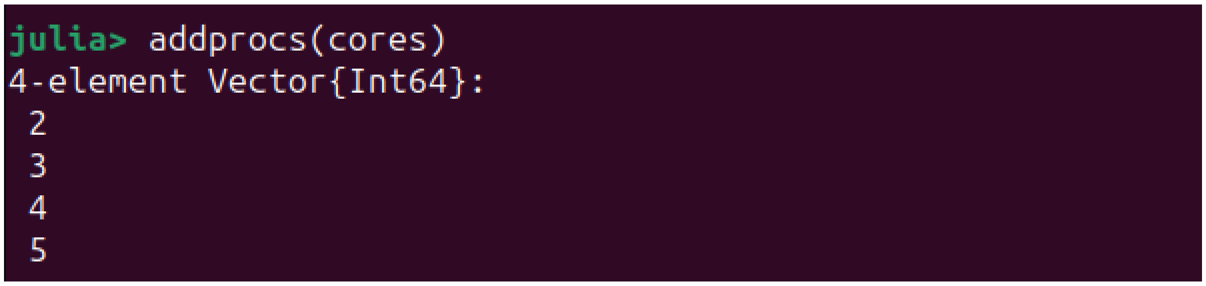
```
julia> using Distributed
```

Defining number of cores to be used



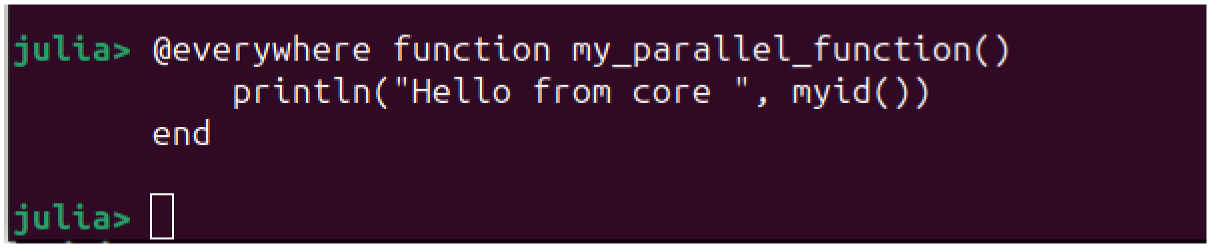
```
julia> cores=4
4
```

and making them available for use



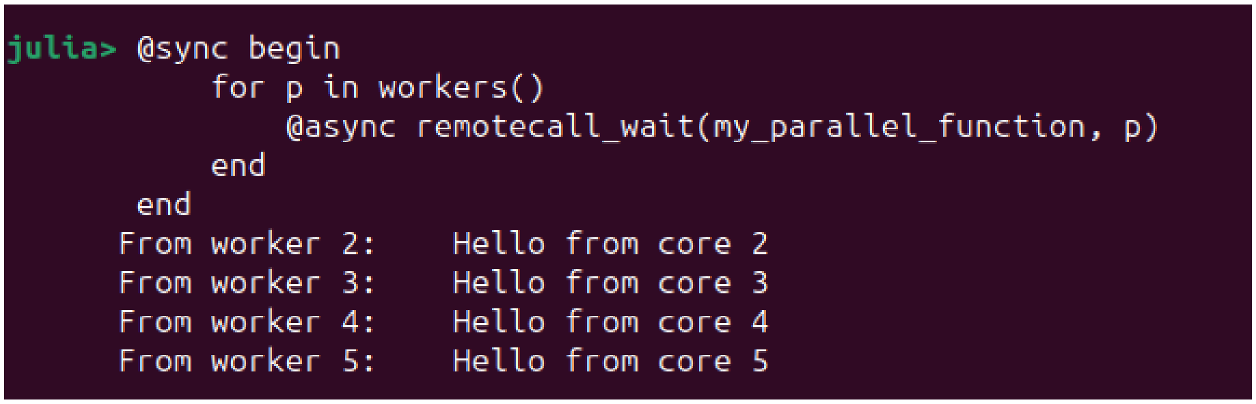
```
julia> addprocs(cores)
4-element Vector{Int64}:
 2
 3
 4
 5
```

Then defining a function using @everywhere. This basically make the piece of code available to every core



```
julia> @everywhere function my_parallel_function()
           println("Hello from core ", myid())
       end

julia> 
```

When program was running, the following was the output

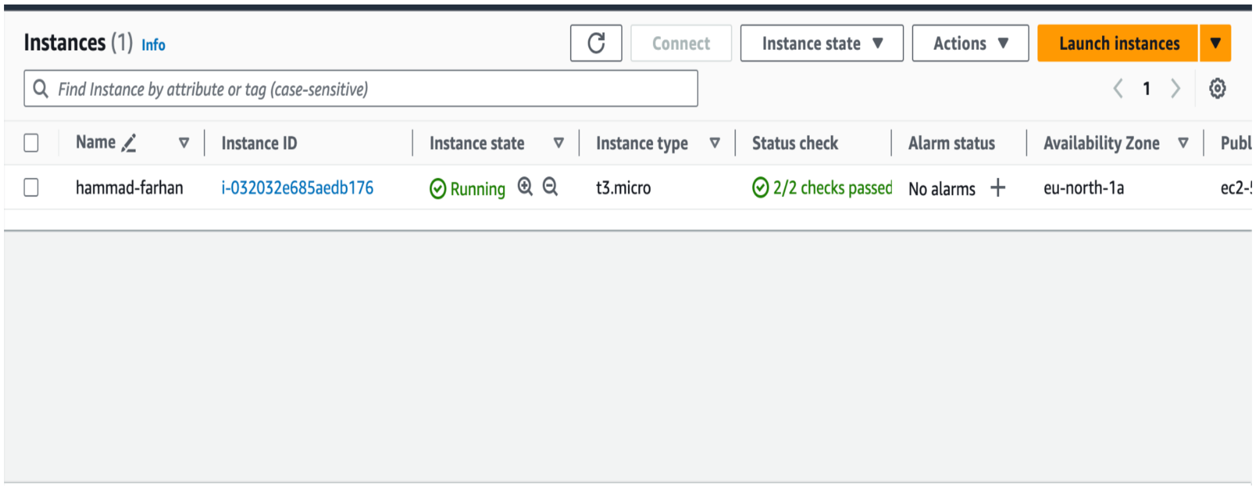


```
julia> @sync begin
           for p in workers()
               @async remotecall_wait(my_parallel_function, p)
           end
       end
      From worker 2:    Hello from core 2
      From worker 3:    Hello from core 3
      From worker 4:    Hello from core 4
      From worker 5:    Hello from core 5
```

Now, exploring Julia on EC2 as Tool 2

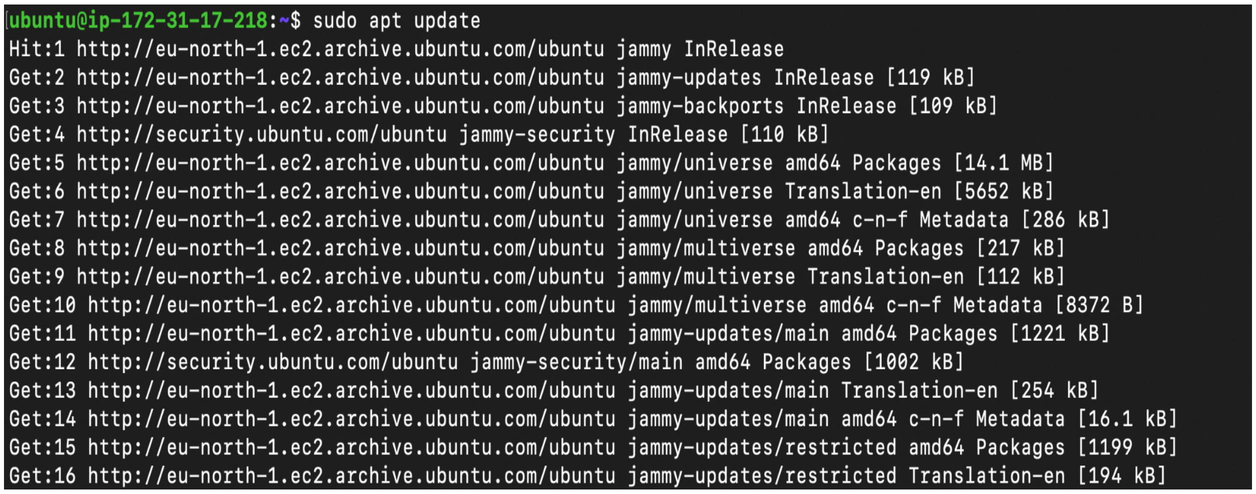
EC2 instance is like a computer on cloud as EC2 is available on AWS which is a cloud.

1. Making an EC2 instance on AWS



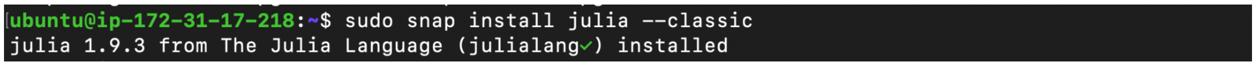
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Publ
hammad-farhan	i-032032e685aedb176	Running	t3.micro	2/2 checks passed	No alarms	eu-north-1a	ec2-

2. Updating the repositories



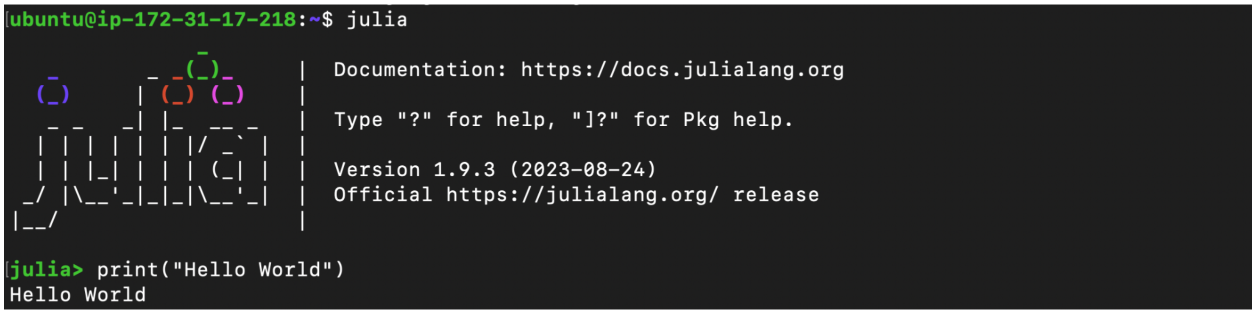
```
ubuntu@ip-172-31-17-218:~$ sudo apt update
Hit:1 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:5 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:6 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy/universe Translation-en [5652 kB]
Get:7 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 c-n-f Metadata [286 kB]
Get:8 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [217 kB]
Get:9 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse Translation-en [112 kB]
Get:10 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 c-n-f Metadata [8372 B]
Get:11 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1221 kB]
Get:12 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1002 kB]
Get:13 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [254 kB]
Get:14 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [16.1 kB]
Get:15 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [1199 kB]
Get:16 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [194 kB]
```

3. Installing Julia



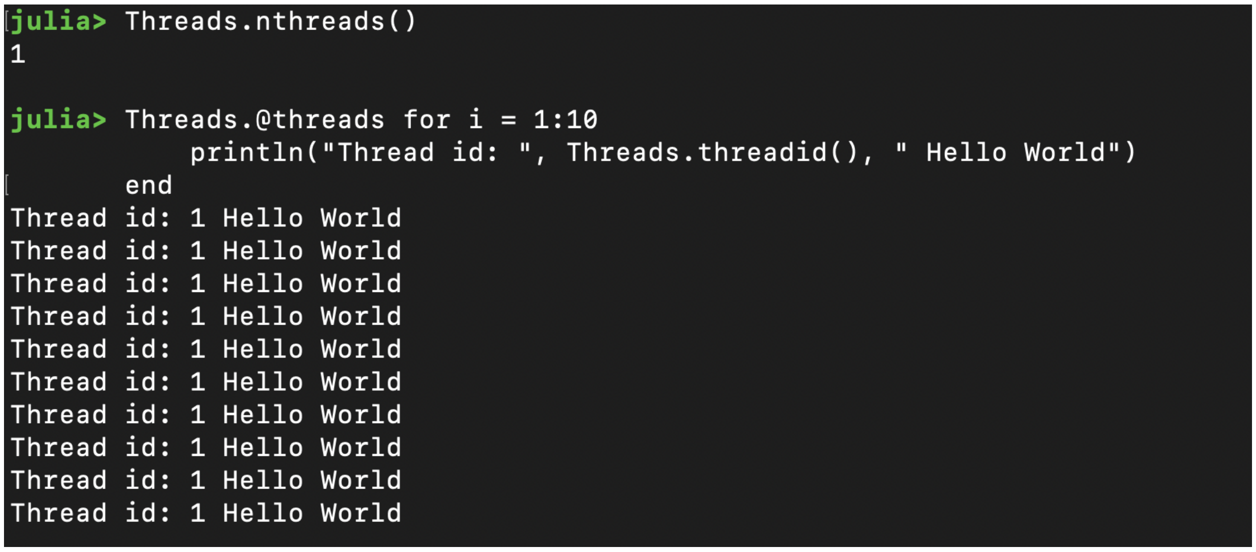
```
[ubuntu@ip-172-31-17-218:~$ sudo snap install julia --classic
julia 1.9.3 from The Julia Language (julialang✓) installed
```

4. Opening terminal for Julia



```
[ubuntu@ip-172-31-17-218:~$ julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.9.3 (2023-08-24)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

[julia> print("Hello World")
Hello World
```

5. Checking number of threads



```
[julia> Threads.nthreads()
1

julia> Threads.@threads for i = 1:10
           println("Thread id: ", Threads.threadid(), " Hello World")
[      end
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
Thread id: 1 Hello World
```

As we have 1 thread, all the code is run on thread. Multi-threading is done above.