Prácticas TSR
Sesión 1

Juan Sánchez (jsanchez@dsic.upv.es)

Despacho 2D08

Acceso a las máquinas virtuales

- Las máquinas virtuales se gestionan en la dirección web: portal-ng.dsic.cloud, desde allí se arrancan/detienen

- La captura muestra la dirección IP de la máquina
- Las máquinas virtuales tienen de nombre genérico: TSR-XXXX-1718.dsic.cloud
- Cada alumno tiene asignada una máquina

Acceso a las máquinas virtuales

- En el laboratorio arrancar Windows o Linux, esto proporciona una conexión VPN.
- Desde fuera del laboratorio hacer una conexión de escritorio remoto a: windesktop.dsic.upv.es o bien a linuxdesktop.dsic.upv.es
- Arrancamos el navegador Web apuntado a la dirección: portal-ng.dsic.cloud
- Después de obviar las excepciones de seguridad en la conexión
- No identificamos con la clave del DSIC
- Veremos las máquinas que tenemos asignadas en el portal
- Si la máquina está apagada la debe arrancar.

Acceso a las máquinas virtuales

- Si está en windesktop.dsic.upv.es, pulse en descargar archivo RDP, abra el archivo y entrará en la pantalla de login de Centos 7 (root, GalYMatias)

- De manera alternativa puede pulsar en el icono de la pantalla de la figura inferior y entrará en la consola de Linux (use las mismas credenciales de antes)

Acceso a las máquinas virtuales

- Si se encuentra en linuxdesktop, arranque un terminal de Linux (usr: root)

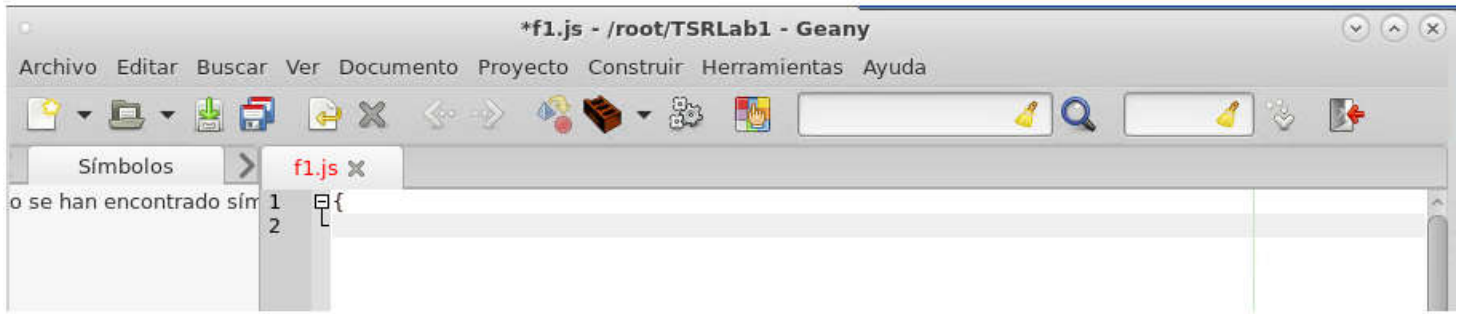
```
$ ssh tsr-XXX-prueba0.dsic.cloud
```

```
$ rdesktop tsr-XXX-prueba0.dsic.cloud -g80%
```

- En cualquiera de los casos debe cambiar el password por defecto con la instrucción passwd desde la línea de comandos

Máquinas virtuales

- Por problemas de instalación los símbolos { } [ ] no pueden ser tecleados dentro de la máquina virtual.
- Como solución podemos introducir el código Unicode de los mismos.
- Para el símbolo { debe usarse <Control> <Shift> U+007B se mantiene pulsado <Control> <Shift> U, se suelta y se pulsa +007B



Sesión 1

- Acceso a ficheros f1.js

```
var fs = require('fs');
fs.readFile('/etc/hosts', 'utf8', function (err,data)
{
if (err) {
return console.log(err);
}
console.log(data);
});
```

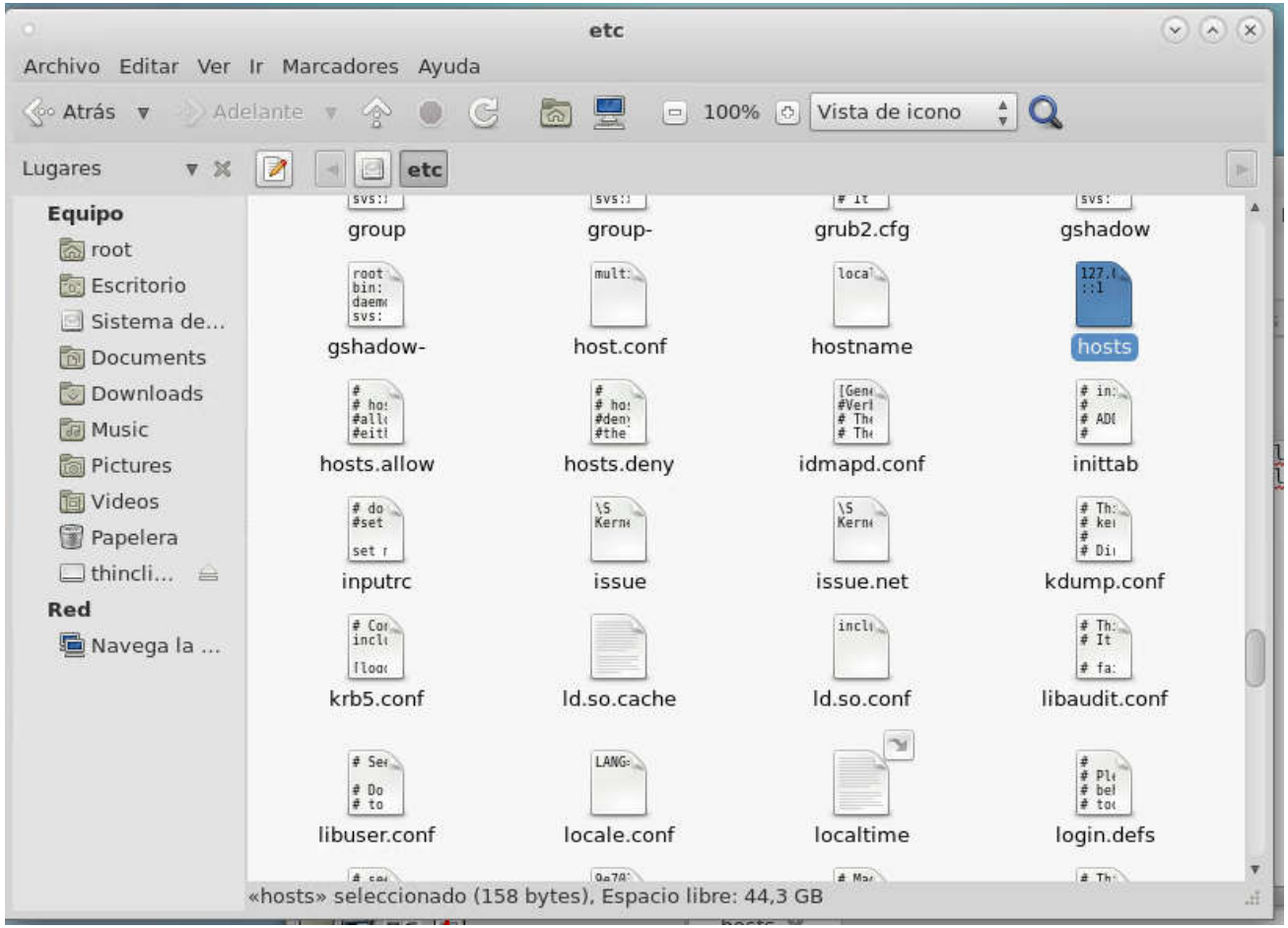
Incluye el sistema de archivos (file system)
Llamada asíncrona a la lectura del archivo hosts
readFile(path, codificación del archivo, callback)

- El archivo hosts contiene la correspondencia entre direcciones IP y nombres de hosts

```
[root@TSR-2007-1718 pruebas]# node f1
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

Sesión 1

- Dentro de Centos7 puede encontrar el archivo hosts



Sesión 1

- En Windows 10 utilice

```
var fs = require('fs');
fs.readFile('/Windows/system32/drivers/etc/hosts', 'utf8', function (err,data) {
if (err) {
return console.log(err);
}
console.log(data);
});
```

```
C:\NodeEjemplos>node f1S1.js
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost
```

Sesión 1

- Escribir en un fichero

```
var fs = require('fs');
fs.writeFile('/tmp/f', 'contenido del nuevo fichero', 'utf8',
function (err,data) {
if (err) {
return console.log(err);
}
console.log('se ha completado la escritura');
});
```

- Crea un fichero de nombre f en tmp

```
[root@TSR-2007-1718 pruebas]# node f2
se ha completado la escritura
[root@TSR-2007-1718 pruebas]#
```

Sesión 1

- Listar un directorio

```
var fs = require('fs');
function getFiles (dir, files_){
        files_ = files_ || [];
        var files = fs.readdirSync(dir);
        for (var i in files){
           var name = dir + '/' + files[i];
           if (fs.statSync(name).isDirectory()){
                                getFiles(name, files_);
           } else {
                    files_.push(name);
                    }
        }
return files_;
}
console.log(getFiles('.')); // search in current directory
```

- Salida generada en mi máquina.

```
[root@TSR-2007-1718 pruebas]# node f3
[ './f1.js', './f1.js~', './f2.js', './f3.js' ]
```

Programación asíncrona

```
function fib(n) {
        return (n<2)? 1: fib(n-2)+fib(n-1);
}
console.log("iniciando ejecucion...");                    1
setTimeout( //espera 10 mseg y ejecuta la funcion
     function() {
          console.log('M1: Quiero escribir ...');
                    }, 10);
var j = fib(40); // cuesta mas de 1seg                    2

function otherMsg(m,u) {
       console.log(m + ": El resultado es "+u);
}
otherMsg("M2",j); //M2 se escribe antes que M1 porque el hilo principal rara vez se suspende   3
setTimeout( // M3 se escribe tras M1
     function() {otherMsg('M3',j);}, 1);
```

Ejecución

```
iniciando ejecucion...
M2: El resultado es 165580141
M1: Quiero escribir ...
M3: El resultado es 165580141
```

Sesión 1

- Path
  - La función exists está depreciada, utilice en su lugar la versión síncrona de la misma existsSync

```
var path = require('path')
> path.exists('/etc', function(exists){
console.log("Does the file exist?", exists)})
> Does the file exist? true
```

  - Código síncrono

```
var fs = require('fs');
var existe =fs.existsSync('/etc')
console.log(">>> Existe "+ existe)
```

Emisores de eventos

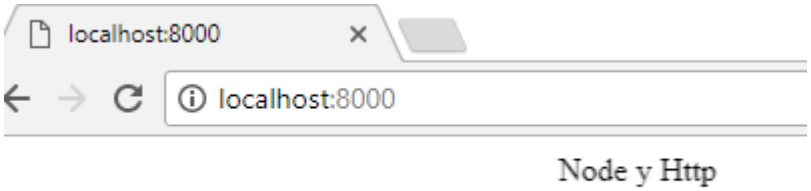
```
var ev = require('events');
var emitter = new ev.EventEmitter;
var e1 = "print", e2= "read"; // name of events
var n1 = 0, n2 = 0; // auxiliary vars
// register listener functions on the event emitter
emitter.on(e1,
function() {
console.log('event '+e1+': '+ (++n1) +'
times')})
emitter.on(e2,
function() {
console.log('event '+e2+': '+ (++n2) +'
times')})
emitter.on(e1, // more than one listener for the same
event is possible
function() {
console.log('something has been printed!')})
// generate the events periodically
setInterval(
function() {emitter.emit(e1);}, // generates e1
2000); // every 2 seconds
setInterval(
function() {emitter.emit(e2);}, // generates e2
8000); // every 8 seconds
```

Sesión 1

- HTTP

```
var http = require('http');
function dd(i) {return (i<10?"0":"")+i;}
var server = http.createServer(
    function (req,res) {
        res.writeHead(200,{'Content-Type':'text/html'});
        res.end('<marquee>Node y Http</marquee>');
        var d = new Date();
        console.log('alguien ha accedido a las '+
            d.getHours() +":"+
            dd(d.getMinutes()) +":"+
            dd(d.getSeconds()));
    }).listen(8000);
```

- En el navegador



- En la consola

```
alguien ha accedido a las 16:47:09
```

Sesión 1

- Servidor: Para el ejemplo debe arrancar dos consolas para usar la línea de comandos: node servidor y cliente

```
var net = require('net');
var server = net.createServer(
    function(c) { //connection listener
        console.log('server: client connected');
        c.on('end',
            function() {
                console.log('server: client disconnected');
            });

        c.on('data',
            function(data) {
                c.write('Hello\r\n'+ data.toString()); // send  resp
                c.end(); // close socket
            });
    });
server.listen(8000,
    function() { //listening listener
        console.log('server bound');
    });
```

Fin conexión

Recepción de datos

- Al ejecutar la consola con el servidor se muestra

```
server bound
```

Cliente

- En la parte cliente

```
var net = require('net');
var client = net.connect({port:8000},
    function() { //connect listener
        console.log('client connected');
        client.write('world!\r\n');
    });
```

Conexión

```
client.on('data',
    function(data) {
        console.log(data.toString());
        client.end(); //no more data written to the stream
    });
```

Recepción de datos

```
client.on('end',
    function() {
        console.log('client disconnected');
    });
```

Desconexión cliente

- Al ejecutar la segunda consola con el cliente

```
client connected
Hello
world!

client disconnected
```

- Mientras que en el servidor se muestra

```
server bound
server: client connected
server: client disconnected
```

JSON

- Servidor

```
var net = require('net');
var server = net.createServer(
    function(c) {
        c.on('data',
            function(data) {
                var person = JSON.parse(data);
                console.log(person.name); //mkyong
                console.log(person.address.street); //8th Street
                console.log(person.address.city); //New York
                console.log(person.phone[0].number); //111-1111
                console.log(person.phone[1].type); //fax
            })})
server.listen(8000,
    function() { //listening listener
        console.log('server escuchando');
    });
```

- Espera que le lleguen datos en el formato JSON desde el cliente

JSON

- Al ejecutar el código cliente

```
var net = require('net');
var msg = JSON.stringify(
    {
        "name": "mkyong",
        "age": 30,
        "address": {
            "street": "8th Street",
            "city": "New York"
        },
        "phone": [
            {
                "type": "home",
                "number": "111-1111"
            },
            {
                "type": "fax",
                "number": "222-2222"
            }
        ]
    });
var socket = net.connect({port:8000},
    function()
    {socket.write(msg);})
```

- La consola del servidor muestra

```
mkyong
8th Street
New York
111-1111
fax
```