TSR - PRÀCTICA 1 CURS 2017/18

INTRODUCCIÓ AL LABORATORI I NODE.JS

PROXY INVERS TCP/IP

Aquesta pràctica consta de tres sessions i té els següents objectius:

1. Introduir els procediments i les eines necessàries per al treball en el laboratori de TSR
2. Introduir tècniques bàsiques per a la programació en NodeJS
3. Desenvolupar l'esquelet d'aplicacions client/servidor flexibles basades en el concepte d'intermedario (Proxy i Proxy Invers)

Enllaços a informació externa:

- API de NodeJS
- REPL
- Gestió d'esdeveniments en Node, mòduls http i net

CONTINGUT

1 Sessió 1

1.1 Eines

Les pràctiques es desenvolupen en JavaScript sobre l'entorn NodeJS

Els equips d'escriptori dels laboratoris del DSIC s'implementen mitjançant sessions en màquines virtuals compartides. Aquesta característica **no és compatible** amb l'ús **de ports** o altres recursos exclusius, particularment en NodeJS, raó per la qual aquests escriptoris no s'usaran per a executar les aplicacions desenvolupades en NodeJS. Usarem **màquines virtuals no compartides** tal com s'explica en l'apèndix d'aquest document.

Les característiques d'aquestes màquines virtuals no compartides són:

- Disposem de control complet (root), i accedim mitjançant una VPN. Podem instal·lar qualsevol requisit que necessitem.
- LINUX (CentOS7 de 64 bits), 2GB RAM, 50GB emmagatzematge.
- Entorn NodeJS (versió 6.11), verificador eslint.
- Biblioteques auxiliars bàsiques (fs, http, net, ...)
- Gestor de paquets npm (permet instal·lar qualsevol altra biblioteca de NodeJS)
    - Necessitaràs executar "npm init" si ho instal·les pel teu compte
- Editors de textos (geany i Visual Code Studio en el menú Programació). Podem utilitzar qualsevol d'ells o instal·lar-nos-en un altre de la nostra elecció.

Qualsevol alumne pot instal·lar un entorn similar en els seus propis equips en Windows, LINUX o MacOS, sense les limitacions descrites quant a ports. Consultar http://nodejs.org

1.2 Procediments

Clàusula de bon ús del laboratori

Els recursos que es posen a la disposició de l'alumne es justifiquen pel seu caràcter educatiu. La seua correcta utilització és responsabilitat de l'alumne, que es compromet a:

- Vetlar per la confidencialitat de les seues claus d'accés.
- No interferir en l'activitat dels altres companys.
- No utilitzar aquests recursos per a finalitats diferents a les descrites en els butlletins de pràctiques

En les instal·lacions existeixen mecanismes que monitoritzen l'ús dels recursos i aporten informacions que poden ser consultades amb posterioritat.

En general escrivim amb qualsevol editor el codi JavaScript en un fitxer amb l'extensió js (x.js), i executem aquest codi amb l'ordre **node x.js [argsOpcionals]**

El punt següent (proves inicials) introdueix xicotets fragments de codi que corresponen a aspectes concrets que necessitarem dominar per al desenvolupament de les aplicacions de les sessions següents. Recomanacions:

1. Crear un directori específic **TSRprac1**, i crear dins del mateix els subdirectoris **proves**, **càrrega**, **proxy**, i **proxyInvers**

2. **Teclejar** el codi en lloc de "copiar i pegar" des del document PDF
    a. suposa poc esforç (els programes són curts)
    b. facilita la familiarització amb el codi
    c. permet enfrontar-nos ràpidament a errors sintàctics típics
    d. evita l'aparició d'errors difícils de depurar deguts a la introducció de caràcters erronis o no visibles en copiar des del PDF

Durant el desenvolupament de les pràctiques poden aparèixer diferents **tipus d'errors**.

<table>
<tr><td colspan="3">Error Sintàctic</td></tr>
<tr><td>Definició</td><td>Detecció/solució</td><td>Exemple</td></tr>
<tr><td>Sintaxi il·legal (identificador no vàlid, estructures mal niades, etc)</td><td>En interpretar el codi s'indica tipus d'error i la seua ubicació en el codi.
Corregir sobre codi font</td><td>> "creïlla"();
TypeError: string is not a function
at repl:1:9
at REPLServer.defaultEval (repl.js:132:27)
at bound (domain.js:254:14)
at REPLServer.runBound [as eval] (domain.js:267:12)
at REPLServer.<anonymous> (repl.js:279:12)
....</td></tr>
<tr><td colspan="3">Error Lògic</td></tr>
<tr><td>Definició</td><td>Detecció/solució</td><td>Exemple</td></tr>
<tr><td>Error de programació (intentar accedir a una propietat inexistent o 'undefined', passar com a argument un tipus de dades incorrecte, etc.)

JavaScript no és fortament orientat a tipus: alguns errors que altres llenguatges capturen en compilar, ací només es mostren en executar</td><td>Resultats incorrectes en executar. Modificar el codi. Convé que en el codi es verifiquen sempre les restriccions a aplicar sobre els arguments de les funcions</td><td>> function suma(array) {return array.reduce(function(x,i){return x+i})}
undefined
> suma([1,2,3])
6
> suma(1)
TypeError: undefined is not a function
at suma (repl:1:36)
at repl:1:1
at REPLServer.defaultEval (repl.js:132:27)
at bound (domain.js:254:14)
at REPLServer.runBound [as eval] (domain.js:267:12)
....</td></tr>
<tr><td colspan="3">Error Operacional</td></tr>
<tr><td colspan="2">Definició</td><td>Detecció/solució</td></tr>
<tr><td colspan="2">Situacions excepcionals que poden sorgir durant el funcionament normal del programa.

Poden deure's a:
• l'entorn (ex. ens quedem sense memòria, massa fitxers oberts)
• la configuració del sistema (ex. no hi ha una ruta cap a un host remot)
• ús de la xarxa (ex. problemes amb l'ús de sockets)
• problemes d'accés a un servei remot (ex. no puc connectar amb el servidor)
• inconsistència en dades introduïdes per l'usuari
• etc.</td><td>construccions try, catch, throw, similars a les de Java

Estratègia:
- Si està clar com resoldre l'error, gestiona-ho (ex. error en obrir un fitxer per a escriptura -> crear-lo com a fitxer nou)
- Si la gestió de l'error no és responsabilitat d'aquest fragment de programa, propagar l'error a l'invocador.
- Per a errors transitoris (ex. problemes amb la xarxa), reintentar l'operació.

Si no podem gestionar ni propagar l'error
- si impedeix continuar amb el programa, avorta
- en un altre cas, anota l'error en un log</td></tr>
</table>

Per a minimitzar errors, es recomana:

- utilitzar el mode estricte: **`node --use_strict fitxer.js`**
- utilitzar **`eslint`** per a diagnosticar la sintaxi i *estil* del nostre programa: **`eslint fitxer.js`**
- documentar correctament cada funció d'interfície
  - significat i tipus de cada argument, així com qualsevol restricció addicional
  - quin tipus d'errors operacionals poden aparèixer, i com es van a gestionar
  - el valor de tornada
- Utilitzar el paquet **assert** en el codi (amb les operacions bàsiques **equal**(expr1,expr2, missatgeError) i **ok**(exprLògica, missatgeError)

Exemple que il·lustra les recomanacions anteriors

```
/*
 * Make a TCP connection to the given IPv4 address.  Arguments:
 *    ip4addr        a string representing a valid IPv4 address
 *    tcpPort        a positive integer representing a valid TCP port
 *    timeout        a positive integer denoting the number of milliseconds
 *                   to wait for a response from the remote server before
 *                   considering the connection to have failed.
 *    callback       invoked when the connection succeeds or fails.  Upon
 *                   success, callback is invoked as callback(null, socket),
 *                   where `socket` is a Node net.Socket object.  Upon failure,
 *                   callback is invoked as callback(err) instead.
 *
 * This function may fail for several reasons:
 *    SystemError    For "connection refused" and "host unreachable" and other
 *                   errors returned by the connect(2) system call.  For these
 *                   errors, err.errno will be set to the actual errno symbolic
 *                   name.
 *    TimeoutError   Emitted if "timeout" milliseconds elapse without
 *                   successfully completing the connection.
 *
 * All errors will have the conventional "remoteIp" and "remotePort" properties.
 * After any error, any socket that was created will be closed.
 */
function connect(ip4addr, tcpPort, timeout, callback) {
  assert.equal(typeof (ip4addr), 'string', "argument 'ip4addr' must be a string");
  assert.ok(net.isIPv4(ip4addr), "argument 'ip4addr' must be a valid IPv4 address");
  assert.equal(typeof (tcpPort), 'number', "argument 'tcpPort' must be a number");
  assert.ok(!isNaN(tcpPort) && tcpPort > 0 && tcpPort < 65536,
      "argument 'tcpPort' must be a positive integer between 1 and 65535");
  assert.equal(typeof (timeout), 'number', "argument 'timeout' must be a number");
  assert.ok(!isNaN(timeout) && timeout>0, "argument 'timeout' must be a positive int");
  assert.equal(typeof (callback), 'function');
  /* do work */
}
```

1.3 Proves inicials

1.3.1 Accés a fitxers

Tots els mètodes corresponents a operacions sobre fitxers apareixen en el mòdul 'fs'. Les operacions són asincròniques, però per a cada funció asincrònica **xx** sol existir la variant sincrònica **xxSync**

- (**f1.js**) Llegir el contingut d'un fitxer

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

- (**f2.js**) Escriure contingut en un fitxer

```
var fs = require('fs');
fs.writeFile('/tmp/f', 'contingut del nou fitxer', 'utf8',
  function (err,data) {
  if (err) {
    return console.log(err);
  }
  console.log("s'ha completat l'escriptura");
});
```

- (**f3.js**) Accés a directoris

  Ex. obté els fitxers continguts en el directori DIR o algun dels seus subdirectoris

```
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
```

MÒDUL PATH

Conté funcions que simplifiquen la manipulació de rutes (noms de fitxers o directoris, que poden contenir el separador **/** i els noms especials **.** i **..)**.

Per a mostrar una alternativa en el desenvolupament i prova de codi, emprem en aquests exemples una modalitat interactiva denominada ***Read-Eval-Print-Loop*** (REPL).

- Per a fer això executem, sense arguments, l'ordre **node**. Això provocarà que s'òbriga un terminal on podem avaluar interactivament qualsevol expressió JavaScript, la qual cosa és molt útil per a l'experimentació

Els exemples són:

- **normalize**. A partir d'una tira representant el path, interpreta els separadors i els noms especials, i retorna una nova tira que correspon a aqueixa mateixa ruta normalitzada

```
> var path = require('path');
> path.normalize('/a/.///b/d/../c/')
'/a/b/c'
```

- **join**. A partir d'una llista variable d'arguments, els uneix i normalitza el path resultant, retornant la tira que correspon al path normalitzat

```
> var path = require('path');
> var url = '/index.html';
> path.join(process.cwd(), 'static', url);
'/home/nico/static/index.html'
```

- **basename**, **extname** i **dirname**. Permeten extraure els diferents components d'un path

```
> var path = require('path')
> var a = '/a/b/c.html'
> path.basename(a)
'c.html'
> path.extname(a)
'.html'
> path.dirname(a)
'/a/b'
```

- **exists**. Permet comprovar l'existència o no d'un path concret

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

1.3.2 Programació asincrònica: esdeveniments

single.js

```
function fib(n) {
  return (n<2)? 1: fib(n-2)+fib(n-1);
}

console.log("iniciant execució...");

setTimeout( // waits 10 ms. then runs function
   function() {
      console.log('M1: Vull escriure ...');
   }, 10);

var j = fib(40); // takes more than 1seg

function otherMsg(m,o) {
  console.log(m + ": El resultat és "+o);
}

otherMsg("M2",j); //M2 appears before M1 as
the main thread is rarely suspended

setTimeout(// M3 appears after M1
   function() {otherMsg('M3',j);},
   1);
```

eventsimple.js

```
var ev = require('events');
var emitter = new ev .EventEmitter;
var i1 = "print", i2= "read";   // name of events
var n1 = 0, n2 = 0;          // auxiliary vars
// register listener functions on the event emitter
emitter.on(i1,
   function() {
      console.log('event '+i1+':  '+(++n1)+'
times')})
emitter.on(i2,
   function() {
      console.log('event '+i2+':  '+(++n2)+'
times')})
emitter.on(i1,  // 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(i1);}, // generates i1
   2000);                          // every 2 seconds
setInterval(
   function() {emitter.emit(i2);}, // generates i2
   8000);                          // every 8 seconds
```

1.3.3 Interacció client/servidor

MÒDUL HTTP

Per a desenvolupament de servidors Web (servidors HTTP)

Ex.- servidor web que saluda al client que contacta amb ell

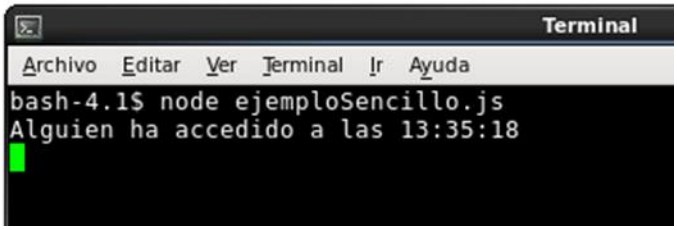
Codi	comentari
`var http = require('http');`	Importa mòdul http
`function dd(i) {return (i<10?"0":"")+i;}`	dd(8) -> "08" dd(16) -> "16"

`var server = http.createServer(`
`  function (req,res) {`
`  res.writeHead(200,{'Content-Type':'text/html'});`
`  res.end('<marquee>Node i Http</marquee>');`
`  var d = new Date ();`
`  console.log('algú ha accedit a les '+`
`  d.getHours() +":"+`
`  dd(d.getMinutes()) +":"+`
`  dd(d.getSeconds()));`
`}).listen(8000);`
	crea el servidor i
	li associa aquesta
	funció
	que retorna una
	resposta fixa i
	a més escriu l'hora en
	la consola
	
	El servidor escolta en
	el port 8000

Executa el servidor i utilitza un navegador web com a client

- Accedeix a la URL `http://localhost:8000`
- Comprova en el navegador la resposta del servidor, i en la consola el missatge escrit pel servidor

Node y HTTP



MÒDUL NET

Client (netClient.js)

```
var net = require('net');

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

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

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

Servidor (netServer.js)

```
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) {  // send resp
            c.write('Hello\r\n'+ data.toString());
            c.end(); // close socket
         });
   });

server.listen(8000,
   function() { //listening listener
      console.log('server bound');
   });
```

1.3.4 JSON (JavaScript Object Notation)

És un format per a representar dades que s'ha convertit en estàndard de facto per a la web.

- JSON.stringify(obj) construeix la representació JSON d'un objecte JavaScript
- JSON.parse(string) construeix un objecte JavaScript a partir de la representació JSON

Client (Jsonc.js)

```
...
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);})
```

Servidor (Jsons.js)

```
...

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 bound');
   });
```

1.3.5 Accés a arguments en línia d'ordres

El shell arreplega tots els arguments en línia d'ordres i li'ls passa a l'aplicació JS empaquetats en un vector denominat process.argv (abreviatura de 'argument values').

Process.argv és un vector, per la qual cosa podem calcular la seua longitud i accedir a cada argument per posició

- **`process.argv.length`**: nombre d'arguments passats per la línia d'ordres
- **`process.argv[i]`** : permet obtenir l'argument i-èssim. Si hem usat l'ordre "**`node programa arg1 …`**" llavors **`process.argv[0]`** conté la cadena '**`node`**', **`process.argv[1]`** conté la cadena '**`programa`**', **`process.argv[2]`** la cadena '**`arg1`**', etc.

Prova: Defineix un fitxer **args.js** amb el següent codi

```
console.log(process.argv);
```

i invoca'l amb diferents arguments. Per exemple:

```
node args.js un dos tres quatre
```

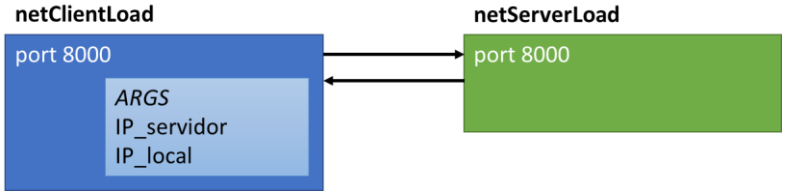
Els dos primers elements del vector corresponen a 'node' i el path del programa executat, respectivament. Amb freqüència s'aplica **process.argv.slice(2)** per a descartar 'node' i el path del programa, de manera que en process.argv només queden els arguments reals per a la nostra aplicació.

2 SESSIÓ 2

2.1 Consulta la càrrega de l'equip

En un entorn client/servidor a gran escala solen replicar-se els servidors (escalabilitat horitzontal), i repartir el treball entre ells segons el seu nivell actual de càrrega.

Anem a crear l'embrió per a un sistema d'aqueix tipus, amb un únic client i un únic servidor (possiblement en màquines diferents) que es comuniquen a través del port 8000



- El servidor es denomina **netserverload.js**, i no rep arguments en línia d'ordres.
  - La següent funció **getLoad** calcula la seua càrrega actual (pots copiar el codi directament). Aquesta funció llig les dades del fitxer /proc/loadavg, filtra els valors que li interessa (els suma una centèsima per a evitar la confusió entre el valor 0 i un error), i els processa calculant una mitjana ponderada (pes 10 a la càrrega de l'últim minut, pes 2 als últims 5 minuts, pes 1 als últims 15)

```
function getLoad(){
  data=fs.readFileSync("/proc/loadavg"); //requereix fs
  var tokens = data.toString().split(' ');
  var min1  = parseFloat(tokens[0])+0.01;
  var min5  = parseFloat(tokens[1])+0.01;
  var min15 = parseFloat(tokens[2])+0.01;
  return min1*10+min5*2+min15;
};
```

- El client es denomina **netClientLoad.js**: rep com a arguments en línia d'ordres l'adreça IP del servidor i la seua IP local
- Protocol: Quan el client envia una petició al servidor, inclou la seua pròpia IP, el servidor calcula la seua càrrega i retorna una resposta al client en la qual inclou la pròpia IP del servidor i el nivell de càrrega calculat amb la funció **getLoad**.

Per a facilitar les proves i depuració, s'ha deixat un servidor actiu en la IP **tsr1.dsic.upv.es**

Important:

- Has de començar amb el codi de netClient.js i netServer.js descrits en el punt anterior
- Cal assegurar-se que tot procés finalitza (ex. amb `process.exit()`)
- En els servidors virtuals es pot esbrinar la IP des de la interfície web del portal, o usant l'ordre **ip addr**
- Completa tots dos programes, col·loca'ls en equips diferents col·laborant amb algun company, i fes que es comuniquen mitjançant el port 8000: `netServerLoad` ha de calcular la càrrega com a resposta a cada petició rebuda des del client, i `netClientLoad` ha de mostrar la resposta en pantalla.

2.2 Intermediari entre 2 equips (proxy transparent)

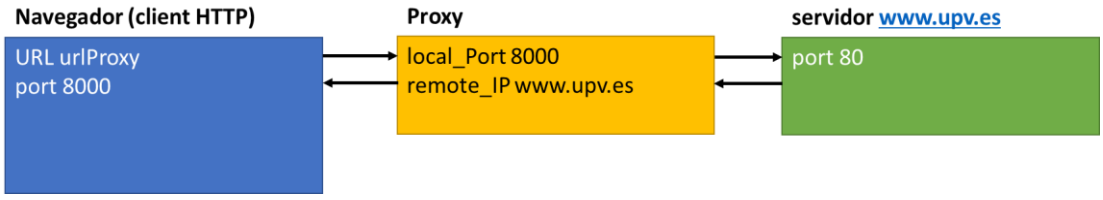
Un intermediari o proxy és un servidor que en ser invocat pel client redirigeix la petició a un tercer, i posteriorment encamina la resposta final de nou al client.

- Des del punt de vista del client, es tracta d'un servidor normal (oculta al servidor que realment completa el servei)
- Pot residir en una màquina diferent a la del client i la del servidor
- L'intermediari pot modificar ports i adreces, però no altera el cos de la petició ni de la resposta

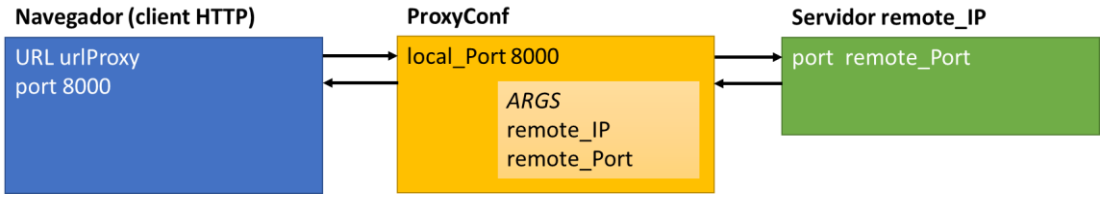
És la funcionalitat que ofereix un redirector, com un proxy HTTP, una passarel·la ssh o un servei similar. Més informació en http://en.wikipedia.org/wiki/proxy_server

Plantegem tres passos incrementals:

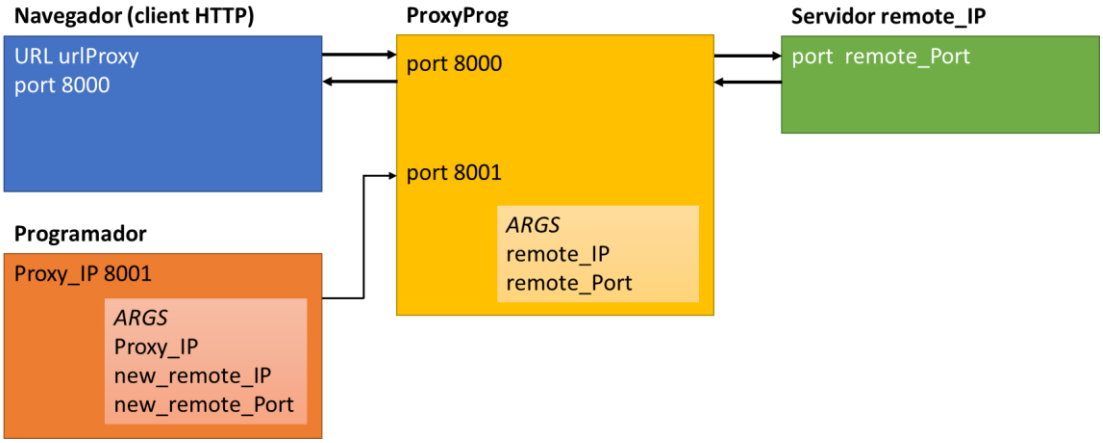
1. Proxy bàsic (**Proxy**). Quan el client HTTP (ex. navegador) contacta amb el proxy en el port 8000, el proxy redirigeix la petició al servidor web de la nostra universitat (158.42.4.23 port 80), i posteriorment retorna la resposta del servidor a l'invocador



2. Proxy configurable (**ProxyConf**): en lloc d'IP i port remots fixos en el codi, els rep com a arguments en línia d'ordres



3. Proxy programable (**ProxyProg**). Els valors d'IP i port del servidor es prenen inicialment des de línia d'ordres, però posteriorment es modifiquen en resposta a missatges del programador en el port 8001



Important:

- Et proporcionem el codi de Proxy: analitza-ho fins a comprendre el seu funcionament
  - Comprova que el seu funcionament és correcte usant un navegador web que apunte a http://adreça_del_proxy:8000/

Codi (Proxy.js)	comentaris

`var net = require('net');`
 	
`var LOCAL_PORT = 8000;`
`var LOCAL_IP = '127.0.0.1';`
`var REMOTE_PORT = 80;`
`var REMOTE_IP = '158.42.4.23'; // www.upv.es`
 	
`var server = net.createServer(function (socket) {`
`  var serviceSocket = new net .Socket();`
`  serviceSocket.connect(parseInt(REMOTE_PORT), REMOTE_IP, function () {`
`  socket.on('data', function (msg) {`
`  serviceSocket.write(msg);`
`  });`
`  serviceSocket.on('data', function (data) {`
`  socket.write(data);`
`  });`
`  });`
`}).listen(LOCAL_PORT, LOCAL_IP);`
`console.log("TCP server accepting connection on port: " + LOCAL_PORT);`
	Usa un socket per a dialogar amb el client (socket) i un altre per a dialogar amb el servidor (serviceSocket)
	
	1.- obri una connexió amb el servidor
	2.- llig un missatge (`msg`) del client
	3.- escriu una còpia del missatge
	4.- espera la resposta del servidor i retorna una còpia al client

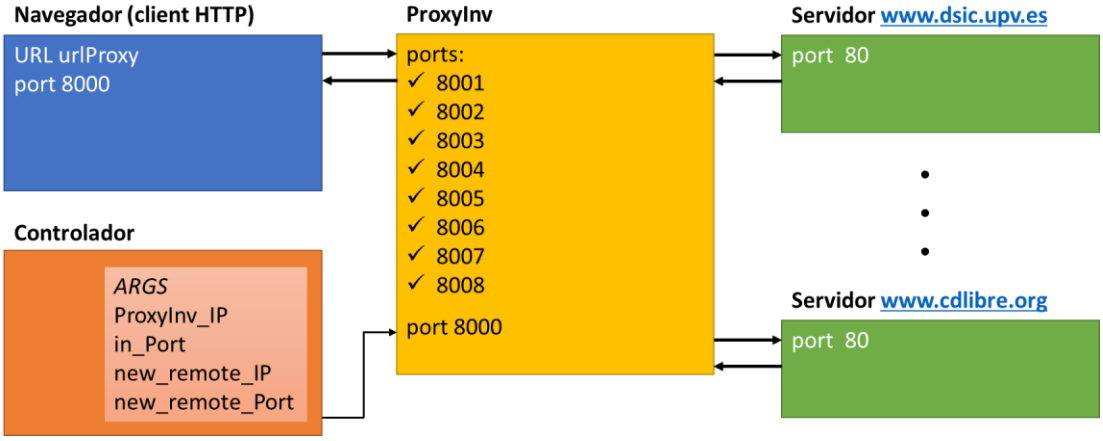
- En l'escenari 2, hem de realitzar les següents proves:
  - Accés al servidor de la UPV (seguim utilitzant com a port local d'atenció de peticions el port 8000)
  - ProxyConf com a intermediari entre netClientLoad i netServerLoad
    - Si ProxyConf s'executa en la mateixa màquina que netServerLoad tenim una col·lisió en l'ús de ports -> modifica el codi de netServerLoad perquè use un altre port
    - Si en executar el programa apareix l'error "EADDRINUSE", indica que estem referenciant un port que ja està en ús per part d'un altre programa
- En l'escenari 3 és necessari codificar el client que actua com a programador
  - Aquest programa (**`programador.js`**) rep en línia d'ordres l'adreça IP del proxy, i els nous valors d'IP i port corresponents al servidor
  - El programa codifica aqueixos valors i els remet com a missatge al port 8001 del proxy, després de la qual cosa acaba
  - El **`programador.js`** hauria d'enviar missatges amb un contingut com el següent:

```
var msg = JSON.stringify ({'remote_ip':"158.42.184.5", 'remote_port':80})
```

3 SESSIÓ 3. PROXY INVERS

Un proxy invers actua com a intermediari entre diversos clients remots i els nostres servidors.

- Permet ocultar la topologia dels servidors
  - Els clients contacten sempre amb el proxy (no necessiten conèixer la quantitat de servidors, les seues IPs, etc)
- Realitzar autenticació, control d'accés, xifrat, etc.
  - Inspeccionar, transformar i encaminar el tràfic HTTP (auditoria, logs, etc.)
- Redirigir el tràfic cap al servidor que corresponga
  - Per a equilibrar la càrrega dels servidors i tolerar la fallada d'un servidor



El nostre proxy invers **prescindeix** de totes les característiques orientades a eficiència i fiabilitat per a facilitar la seua comprensió. **ProxyInv** no té arguments en línia d'ordres, atén a peticions dels clients que arriben a un dels ports 8001 a 8008, i disposa inicialment d'IP i port associats a cada port d'entrada (valors de prova). Utilitza adreces numèriques!

Port del proxy	FQDN del servei remot	IP	Port del servei remot
8001	www.dsic.upv.es	158.42.184.5	80 (http)
8002	www.upv.es	158.42.4.23	80 (http)
8003	www.libreoffice.org	89.238.68.168	80 (http)
8004	memex.dsic.upv.es	158.42.179.56	8080 (http)
8005	www.cdlibre.org	147.156.222.65	80 (http)

ProxyInv es programa a través del port 8000. El client encarregat d'això es denomina **controlador**, i genera missatges del tipus

```
var msg =JSON.stringify ({'op':"set", 'inPort':8002, 'remote':{'ip':"158.42.4.23", 'port':80}})
```

`Controlador.js` ha de tenir els següents **arguments d'entrada**:

1. Adreça IP del proxy.
2. Port d'entrada del proxy (port a reprogramar, en el rang 8001..8008)
3. Nova adreça IP remota.
4. Nou port remot.

4 ANNEX. ÚS DE SERVIDORS VIRTUALS

Amb el propòsit d'oferir un major realisme als escenaris d'engegada i desplegament d'aplicacions en xarxa, utilitzem un sistema de virtualització amb un servidor per alumne. L'objectiu d'aquest annex és facilitar la seua comprensió i ús.

Les imatges a les quals es té accés són instal·lacions similars a les disponibles en els equips de laboratori del DSIC. Les diferències més significatives són:

- L'usuari té capacitat d'administració, permetent modificacions sobre el sistema de major importància que un usuari convencional.
- Les màquines virtuals posseeixen una adreça IP que les distingeix i per les quals són referenciables, sempre que el referenciador es trobe dins de la VPN que les conté.
- Les màquines virtuals posseeixen estat, de manera que les modificacions aplicades s'acumulen.
- Es troben tancades dins d'una infraestructura limitada, amb un pont VPN per al seu accés remot.
- El pont es troba obert automàticament en ***els escriptoris virtuals*** dels laboratoris del DSIC.
- Si es desitja accedir des d'altres llocs, dins de la UPV (o des de fora, accedint prèviament a la VPN de la UPV), el recomanable és obrir un escriptori remot a…
    - Windows: `windesktop.dsic.upv.es`
    - LINUX: `linuxdesktop.dsic.upv.es`

  I d'aquesta manera reproduïm les mateixes condicions que en els laboratoris del DSIC.

Superat aquest problema, en el que segueix ens centrarem en la connexió i forma habitual de treball des del laboratori i/o ***escriptori virtual***, on ja són visibles les adreces **`*.dsic.cloud`**.

Convé distingir entre les accions sobre les màquines virtuals i les accions dins de les màquines virtuals. Sembla obvi que no és possible engegar una màquina virtual sense comptar amb un servei extern, i aquesta serà una de les missions del portal…

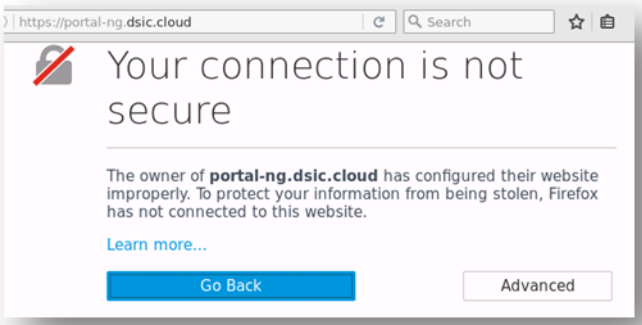
`https://portal-ng.dsic.cloud`.

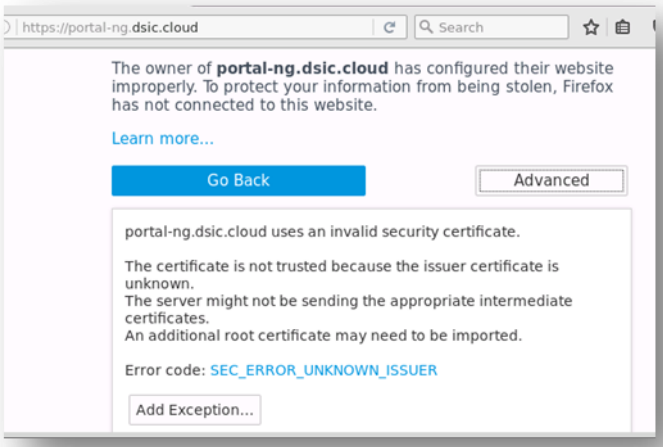
- Una vegada identificat un usuari, mostra la relació de màquines virtuals al seu nom, amb possibles operacions sobre elles.

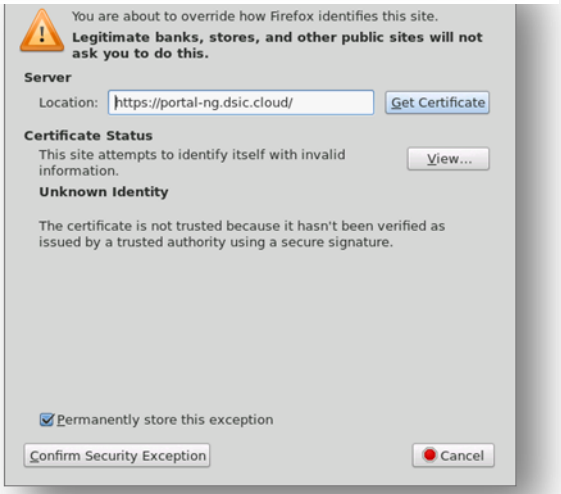
Parlant amb precisió, els usuaris poden agrupar-se en col·lectius als quals se'ls mostra un mateix grup de màquines virtuals. És un cas puntual que el col·lectiu estiga format per un únic individu, però és el nostre cas.

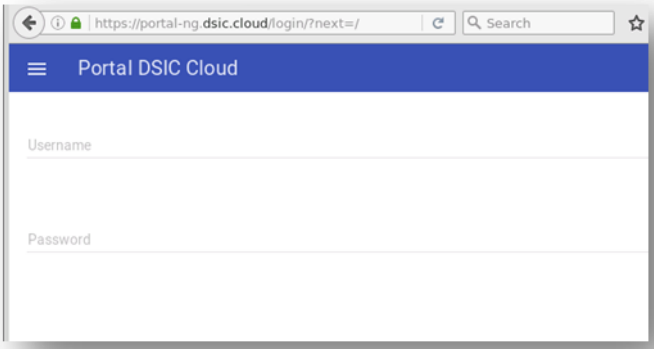
La seqüència per a l'accés ve il·lustrada a continuació:

1. En primer lloc, i per a la primera ocasió, l'URL abans esmentat provoca un avís o queixa del navegador. Admetrem l'ús d'un certificat no oficial tal com s'il·lustra en les següents captures
2. Després d'aquest pas de configuració, el servidor mostrarà un diàleg d'identificació en el qual espera el nostre identificador i contrasenya dels laboratoris del DSIC.

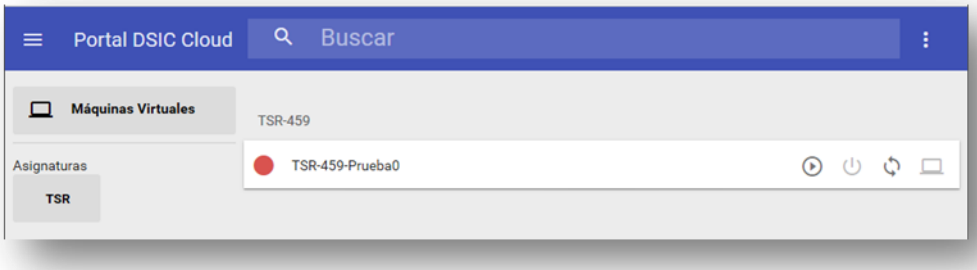








3. Després de franquejar la prova anterior, el portal ens mostrarà la relació de màquines virtuals que disposem. El color roig revela que la màquina virtual es troba detinguda. En la mateixa fila, a la dreta, apareixen 4 símbols per a poder iniciar ( ) la màquina, detenir-la ( ), refrescar ( ) la informació o iniciar una sessió ( ) interactiva per VNC. Els elements atenuats no són utilitzables en aqueix moment.

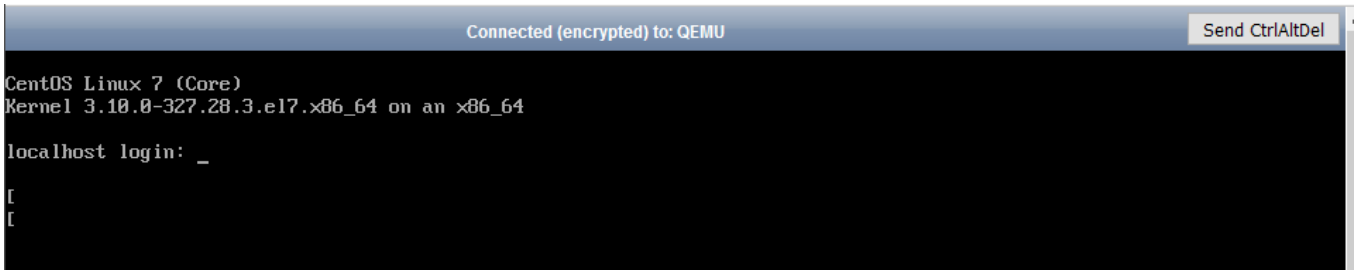


4. Prement sobre el nom de la virtual ens mostra alguns detalls, incloent la seua IP.

5. Després d'iniciar la màquina, l'aspecte que ens mostra és:
   a) El nom de la nostra màquina virtual ha de ser completat amb **`.dsic.cloud`**.

      En aquest exemple de captura, del nom *TSR-459-Prova0* es pot deduir que el FQDN de la nostra virtual és *`tsr-459-Prova0`*`.dsic.cloud`.

   b) És inevitable acudir a aquesta interfície per a iniciar una màquina virtual detinguda. És desaconsellable usar-ho per a parar-la si es troba en funcionament, perquè provoca un tancament "menys controlat". Es prefereix iniciar una sessió de terminal i parar la virtual amb un tancament més ordenat, com `shutdown -h now`
   c) L'accés per la consola de VNC està intervingut pel navegador, provocant un diàleg d'avís del navegador molt similar al primer que s'ha il·lustrat



A continuació, s'indica breument com realitzar les activitats més habituals: copiar fitxers i iniciar una sessió remota. En tots dos casos se suposa que hi ha establida una VPN que ens permet accedir als equips virtuals.

4.1 Copiar fitxers

a) Mode text (línia d'ordres) des de LINUX

- Si vols copiar arxius des d'una sessió Windows, hauràs d'obrir una consola d'ordres Git-Bash per a disposar de les ordres LINUX més freqüents.

`scp` *`arxius`* `root@`*`mivirtual`*`.dsic.cloud:`

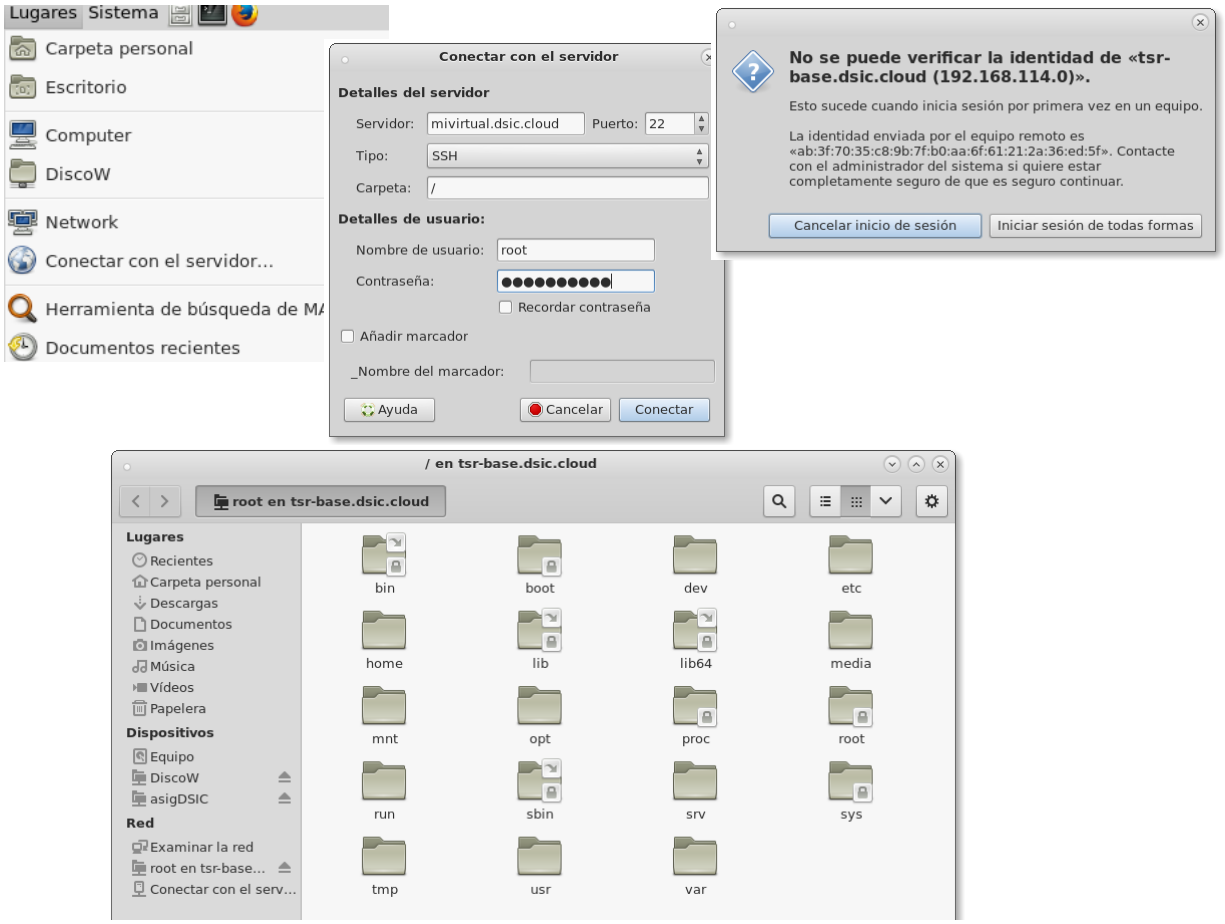
Consulta el manual de scp para més detalls

b) Mode gràfic

En el vostre equip d'escriptori LINUX, menú `Sistema`, trobareu una funció general `Connectar amb el servidor`…

- L'ajustem per a *`mavirtual`*`.dsic.cloud`, usuari root, port 22 i protocol ssh.

Ens apareixerà un avís sobre la identitat de l'equip que descartarem. Després d'executar aquesta operació podrem **treballar amb el sistema de fitxers** remot mitjançant una finestra de l'explorador d'arxius.



4.2 Iniciar sessió remota

a) Mode text (línia d'ordres)
   - Si vols connectar des d'una sessió Windows, hauràs d'obrir una consola d'ordres Git-Bash per a disposar de les ordres LINUX més freqüents.

     `ssh root@mivirtual.dsic.cloud`

     Consulta el manual de ssh per a més detalls.

b) Mode gràfic: connectar mitjançant un client d'escriptori remot
   - Malgrat tractar-se de la manera més pesada, també pot ser el més intuïtiu perquè ens col·loca en una finestra un escriptori LINUX

4.3 Detalls finals

La contrasenya de **root** per a la màquina virtual serà anunciada en el laboratori, i HA DE SER MODIFICADA EN EL PRIMER ACCÉS.

És una irresponsabilitat mantenir una contrasenya coneguda per tots. També és una *invitació* per a accessos no desitjats. Aquesta negligència TÉ conseqüències en la seguretat de la vostra instal·lació, responsabilitat vostra.

La configuració inicial de les màquines virtuals provoca que el **tallafocs** impedisca l'accés des de l'exterior a la majoria de ports de l'equip. Per a habilitar l'accés a un rang de ports (p. ex. des del 8000 fins al 8100), la primera vegada haurà'n d'executar-se les següents instruccions:

```
firewall-cmd --permanent --add-port=8000-8100/tcp
firewall-cmd --reload
```

Atès que els recursos del laboratori són limitats, és habitual que dos estudiants compartisquen el mateix equip en una sessió, la qual cosa sol conduir al fet que empren una única màquina virtual remota. En aquests casos cobra **importància l'organització de l'accés** perquè aquests alumnes no interferisquen entre si.

Serà d'ajuda …

1. Comprovar si hi ha altres usuaris connectats quan entrem o fem alguna cosa
2. Disposar d'un servei de missatgeria instantani, o bé l'ús del telèfon mòbil
3. Pactar amb antelació les responsabilitats de cadascun perquè les interferències es minimitzen