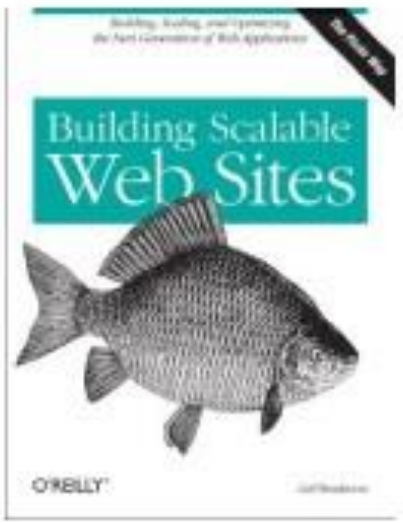
Corporate Training

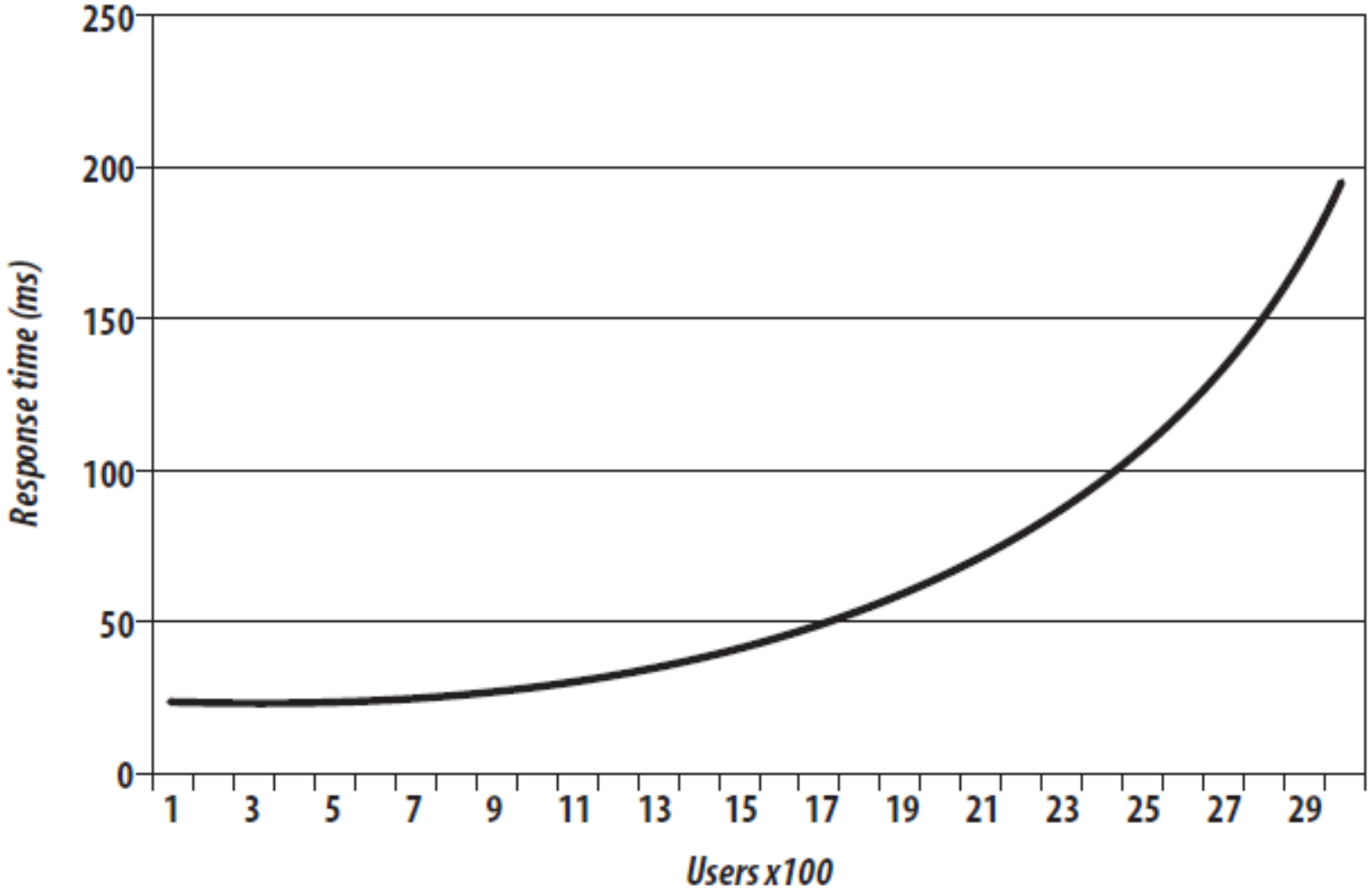
Alta disponibilidad en aplicaciones Web

Bibliografía

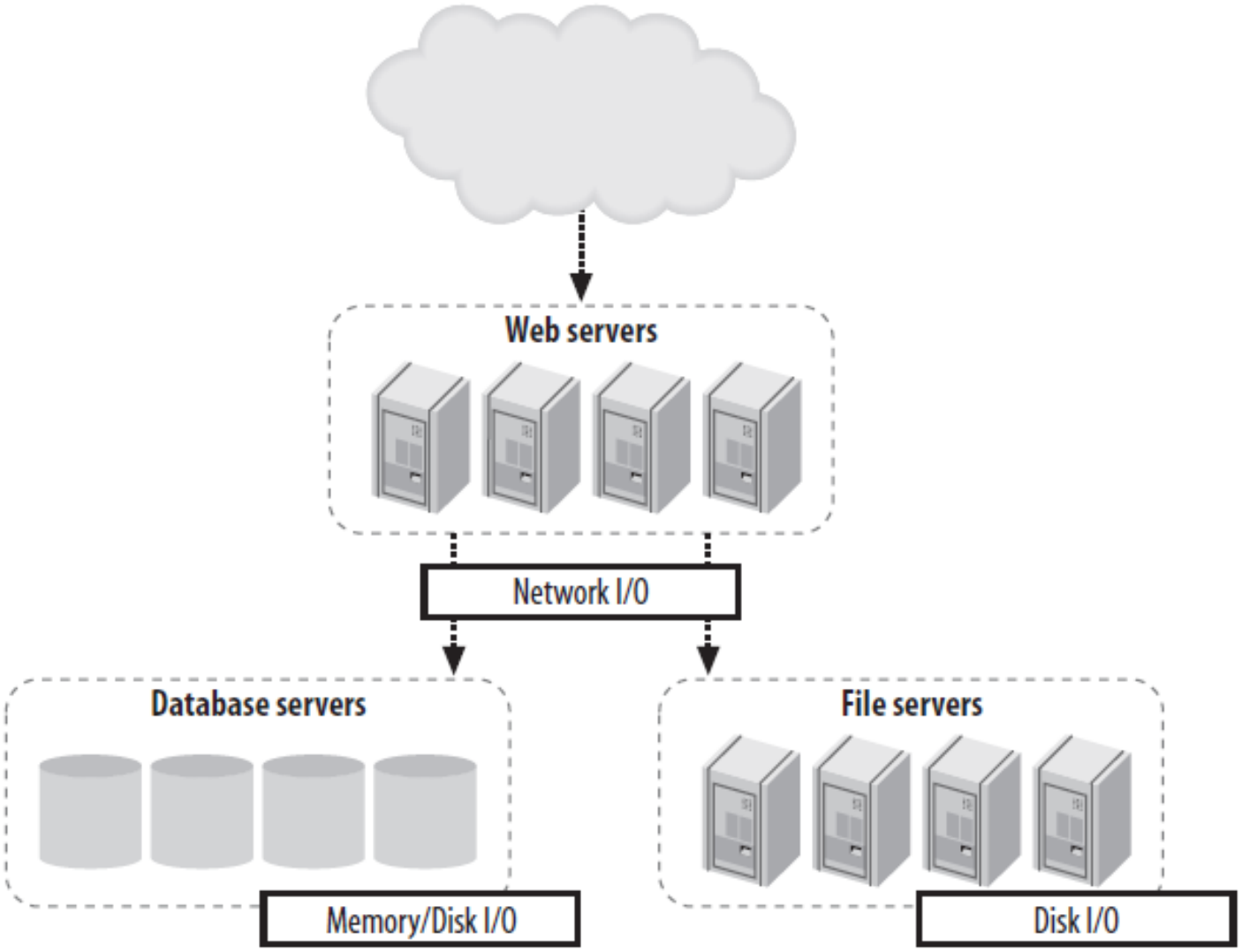
- Building Scalable Web Sites



Cuello de botella (bottleneck)



Cuello de botella - Mapa



CPU Analisis

- CPU

```
top - 22:12:38 up 28 days,  2:02,  1 user,  load average: 7.32, 7.15, 7.26
Tasks: 358 total,   3 running, 353 sleeping,   0 stopped,   2 zombie
Cpu(s): 35.3% us, 12.9% sy,  0.2% ni, 21.8% id, 23.5% wa,  0.6% hi,  5.8% si
Mem:  16359928k total, 16346352k used,    13576k free,    97296k buffers
Swap:  8387240k total,    80352k used,  8306888k free,  1176420k cached
```

- Soluciones
  - Code Profiling
  - Opcode Caching - cache the result of the PHP code compilation to bytecode

I/O Análisis

- I/O

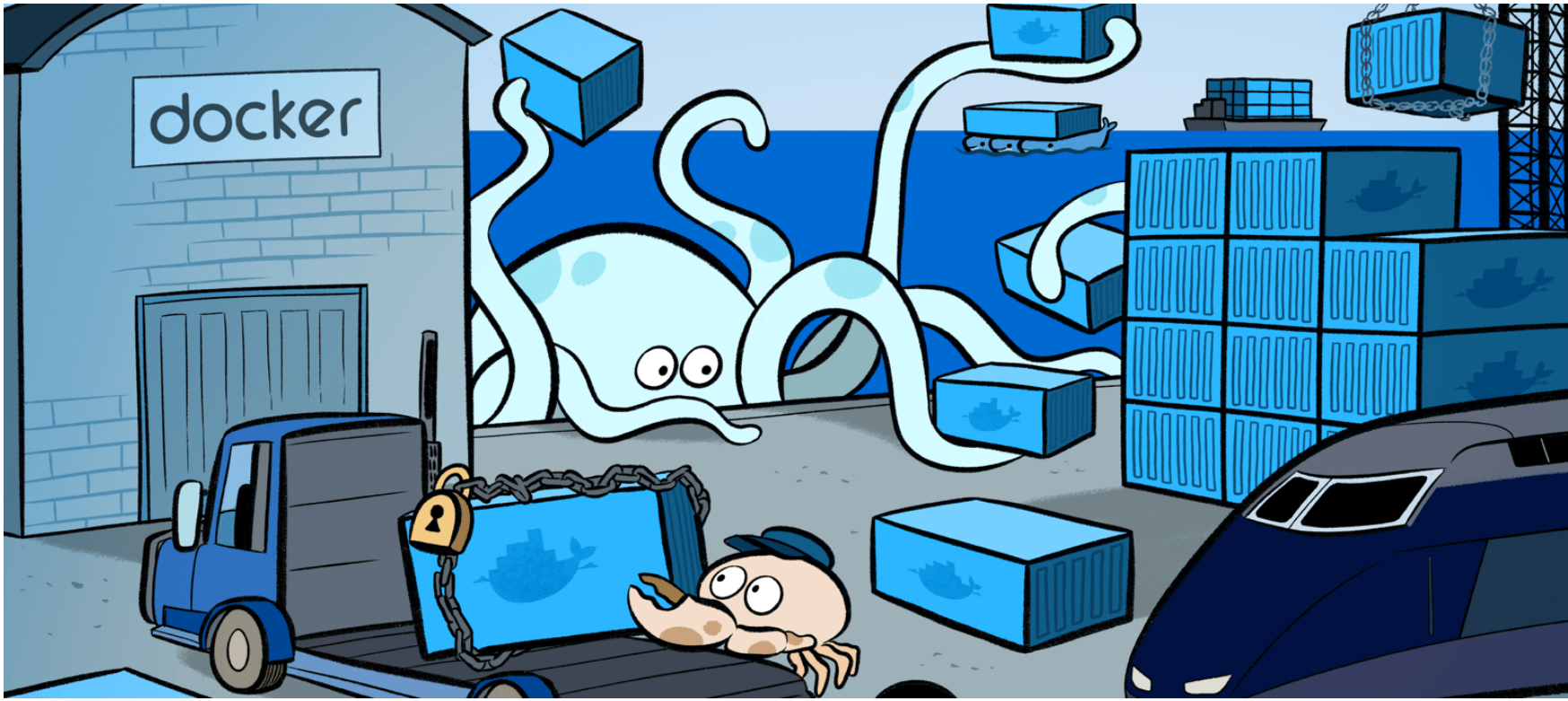
```
[calh@db1 ~] $ iostat -c
Linux 2.6.9-5.ELsmp (db1.flickr)  11/05/2005

avg-cpu:  %user   %nice    %sys %iowait   %idle
          35.27    0.17   19.24   43.49    1.83
```

- Soluciones
  - Cache

Network I/O Análisis

- No debería ser un problema salvo…
  - Problemas de configuración
    - Trabajando a 10Mb en lugar de 1000Mb
    - Los switch negocian la menor capacidad de transferencia de los dos endpoints
- Soluciones
  - Diseñar la red teniendo en cuenta los datos a transferir y capacidades de interfaces de red.

Docker



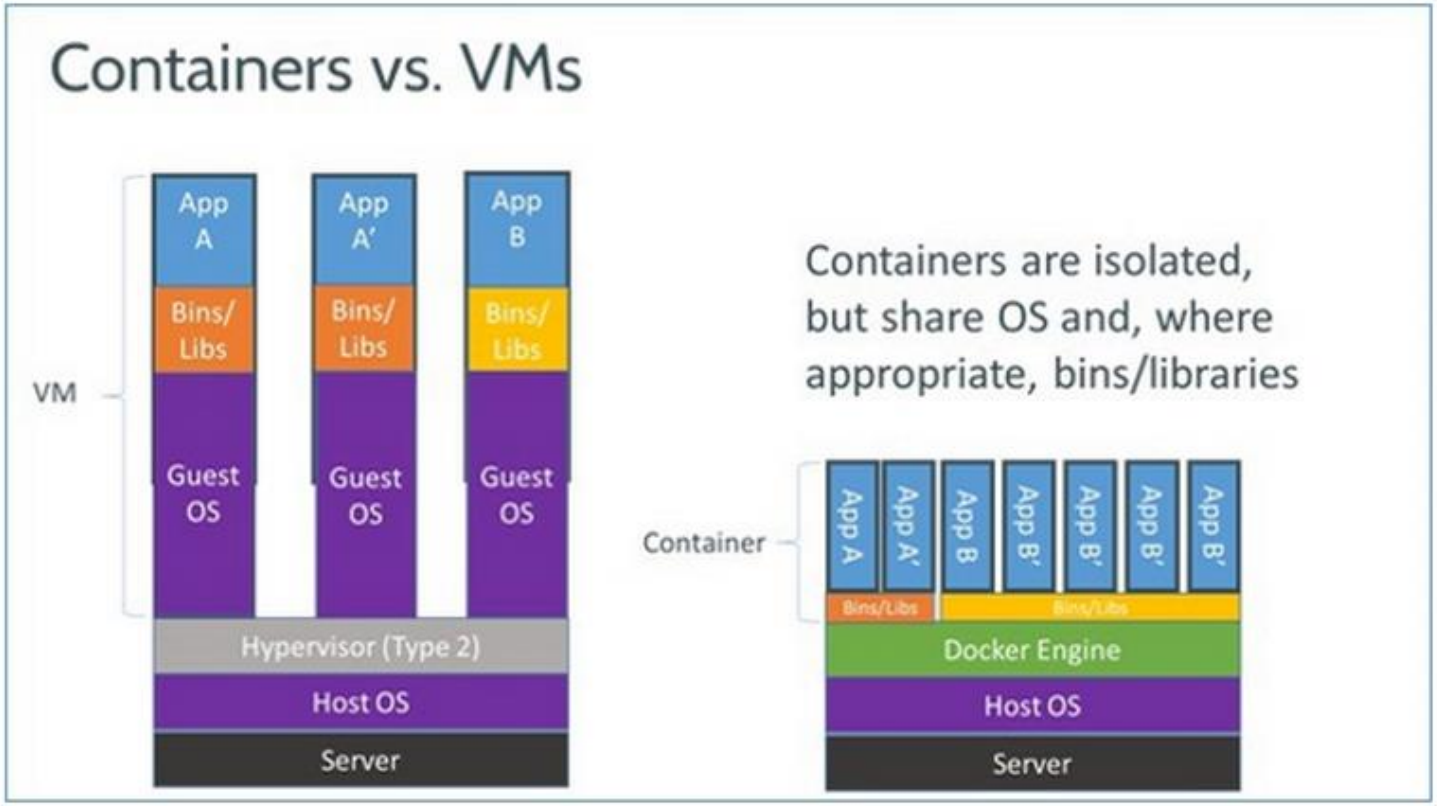
Docker

- Docker es una plataforma de gestión de contenedores

- Ok, pero qué es un contenedor?
  - Es una tecnología de empaquetar software de tal forma de que corra aislada dentro en OS que es compartido por otros contenedores.

Docker

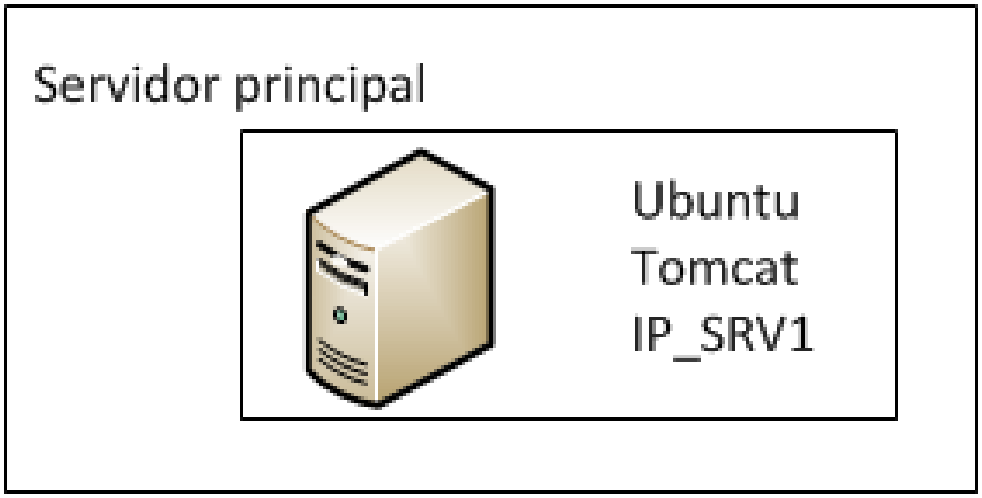
- Pero que diferencia hay con una Maquina Virtual?
  - Las maquinas virtuales son sistemas operativos completos empaquetados que comparten el hardware con el Host (el OS que lo ejecuta).
  - Los contenedores solo utilizan librerías del Host.



Docker - beneficios

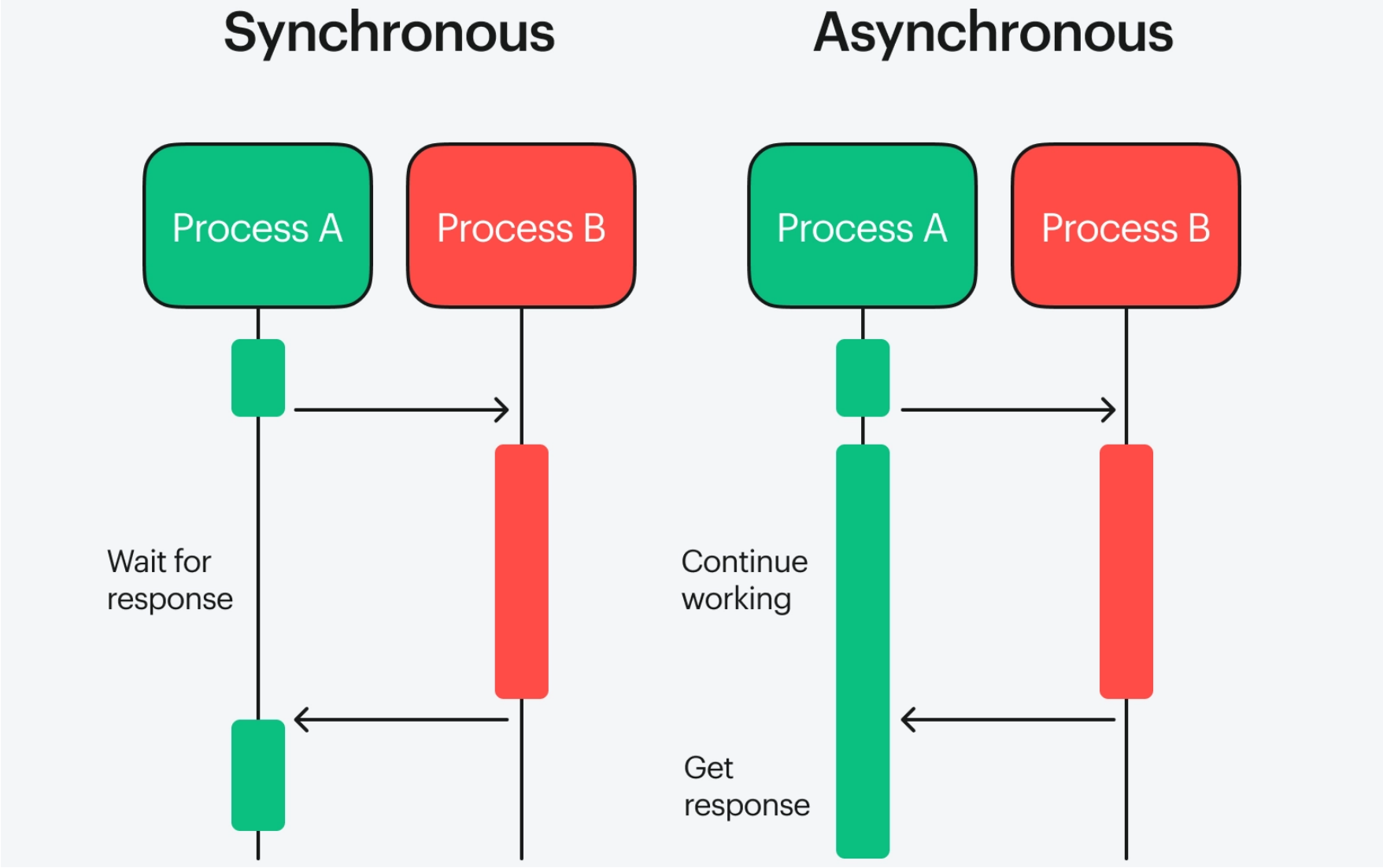
- Para desarrolladores
  - Simplifica la tarea de armar ambientes de desarrollo.
  - Permite armar ambientes para diferentes stacks basados en casi cualquier lenguajes (Java8 Oracle+MariaDB u OpenJDK 7 + PostGIS) , o incluso stacks conflictivos (Python 2.5 y Python 3) .
  - Reduce el tiempo de armado de ambientes un 65% (según el equipo de Docker)
  - Permite definir infraestructuras complejas compuesta de varios contenedores, con diferentes configuraciones de red, etc.

Veamos como funciona un solo servidor



- Problemas:
  - Capacidad limitada de CPU
  - Capacidad limitada de ancho de banda de red
  - Capacidad limitada de Memoria

Sync vs Async



Sync vs Async

Capacidad limitada de threads

- Usaremos el cálculo de fibonacchi para forzar el consumo de CPU:
  - http://localhost/fib?number=20
  - Parámetro "**number**" indica el parámetro de la función fibonacchi.

- Simularemos la dependencia con un recurso (por ej. Una base de datos) durmiendo el thread:
  - http://localhost/sleep?min=1
  - Parámetro "**min**" indica los minutos que dormirá el thread.
  - Parámetro "**seg**" indica los segundos que dormirá el thread.

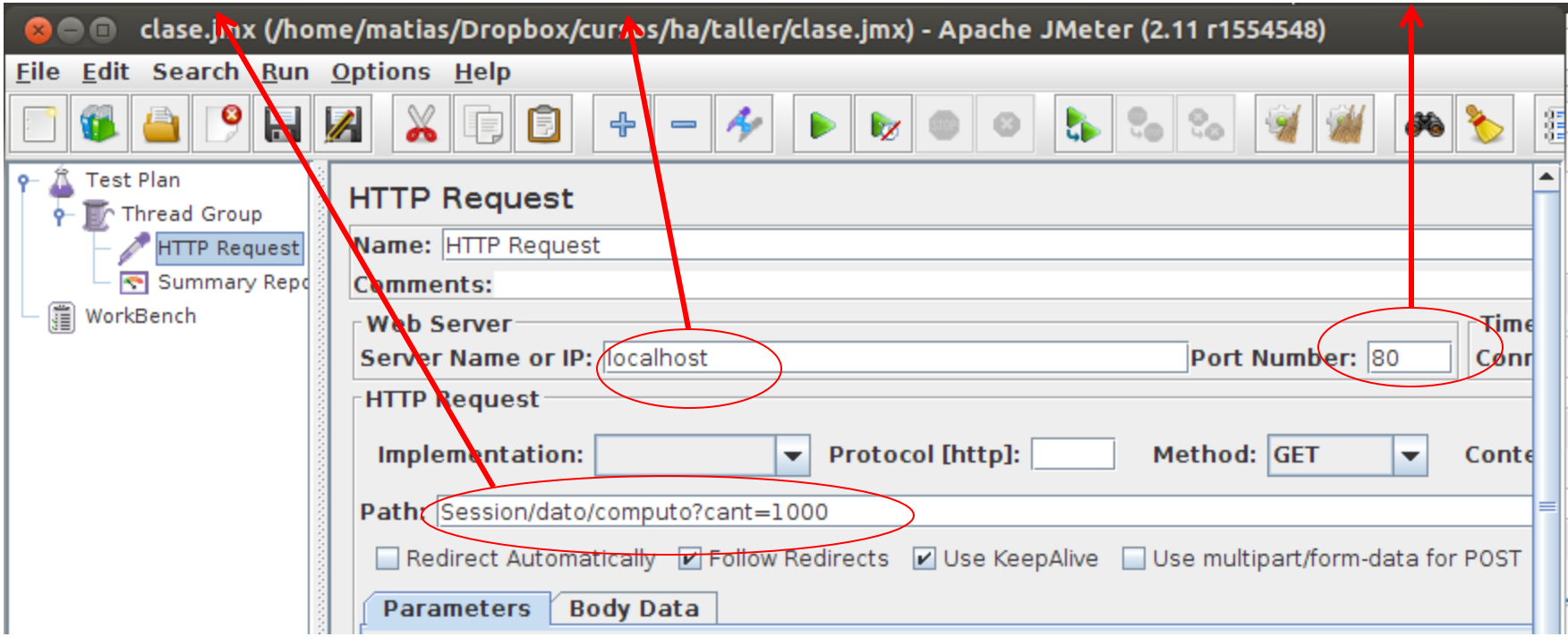
Capacidad limitada de CPU

- Usaremos el calculo de números primos:
  - http://localhost/fib?number=15
  - Parámetro cant indica los números primos a calcular mas alto es mas CPU requerirá mas tiempo.

Contexto y url de recursos

Dirección del servidor

Puerto del servidor



Armar el nodo del servidor

- Docker
  - Opción 1: Instalar Docker

    https://docs.docker.com/engine/installation/

  - Opción 2: Utilizar una Virtual Machine para correr docker:

    http://www.osboxes.org/ubuntu/

  - JMeter
    - Para probar las capacidades del servidor vamos a utilizar la herramienta open source Jmeter.
    - http://jmeter.apache.org/

Iniciar contenedor Web App - alternativa

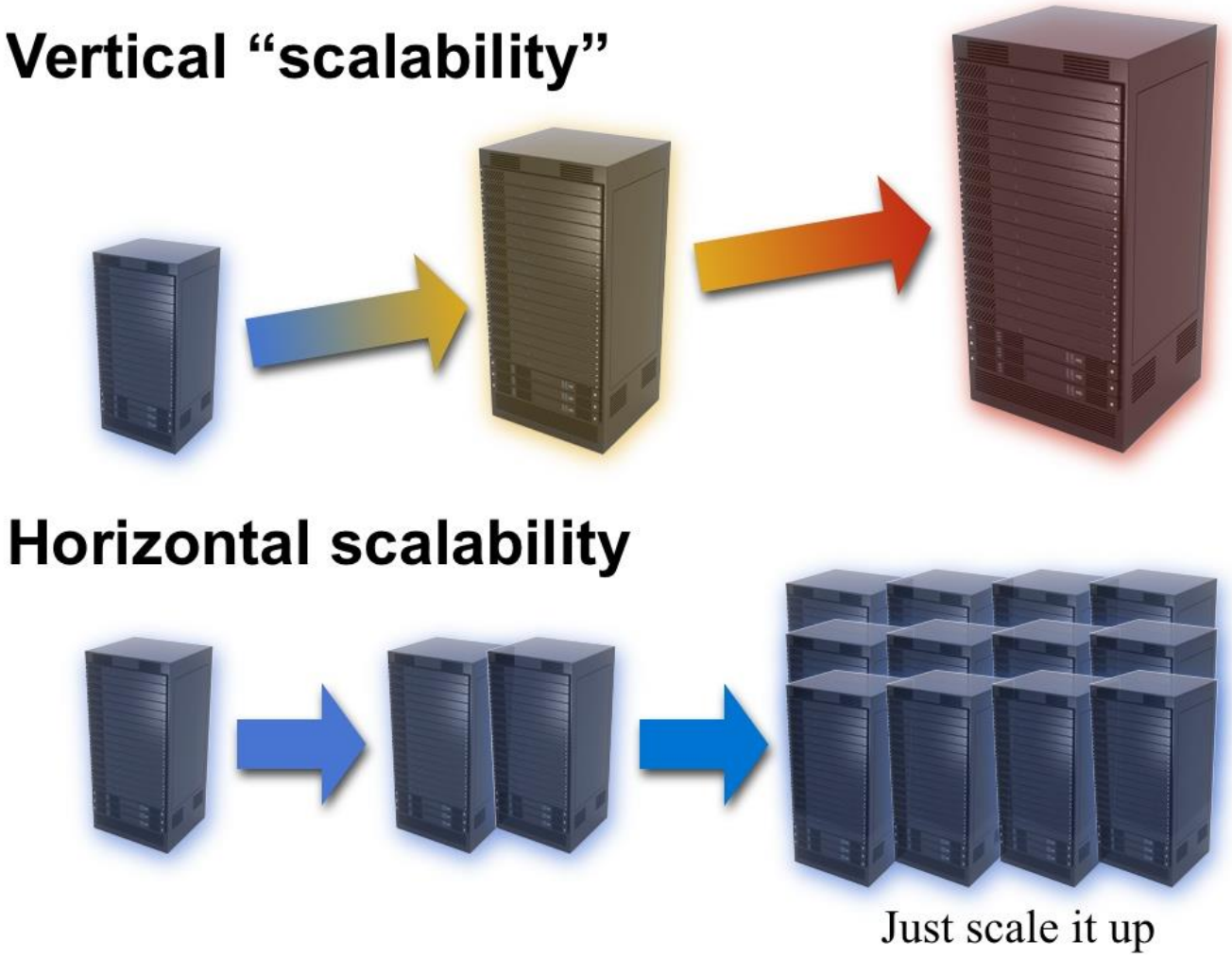
- Archivo tipo de Docker para construir el contenedor

```
# Plataforma base
FROM python:3.12-slim
# Crea y cambia al directorio de la aplicación llamado /app
WORKDIR /app
# Instala Poetry
RUN pip install poetry
# Copia los archivos de dependencias al contenedor
COPY pyproject.toml poetry.lock* requirements.txt ./
# Configura Poetry para no crear entornos virtuales
RUN poetry config virtualenvs.create false
# Instala las dependencias del proyecto
RUN poetry install --no-root
# Copia el código fuente de la aplicación al contenedor
COPY src/ ./src/
# Inicia la aplicación usando Uvicorn
CMD ["poetry", "run", "uvicorn", "src.app:app", "--host", "0.0.0.0", "--port", "5555", "--
workers", "1"]
```

- Acceder a la app

```
docker build -t servidor:latest .
docker run -it --rm --name webapp-node –p 80:5555 servidor
```

Escalabilidad



Fuente: http://www.internetria.com/blog/2013/05/08/nosql/

Comandos útiles

- Lista de contenedores activos

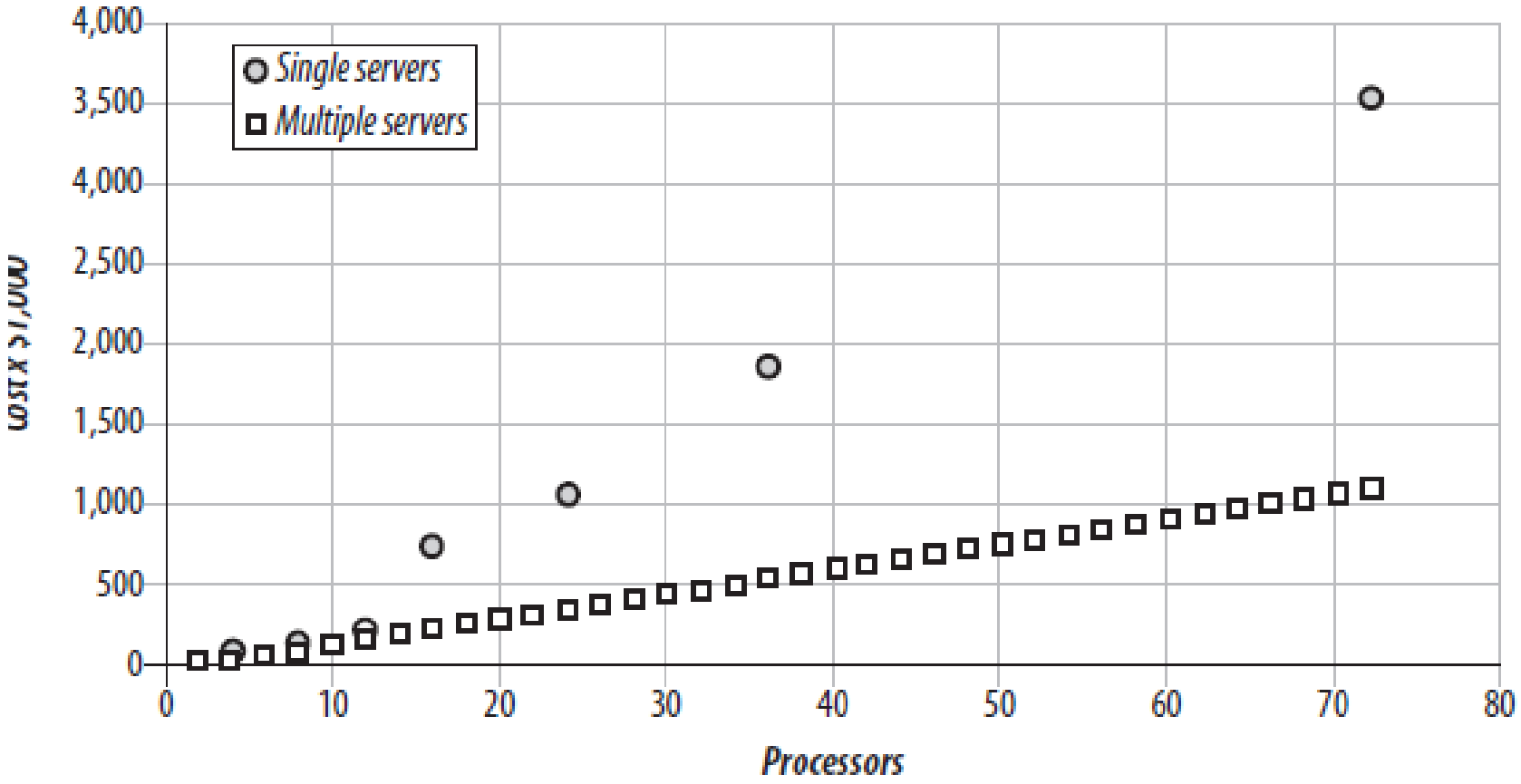
  Docker ps

- Apagar un contenedor. Por ejemplo *webapp-node*

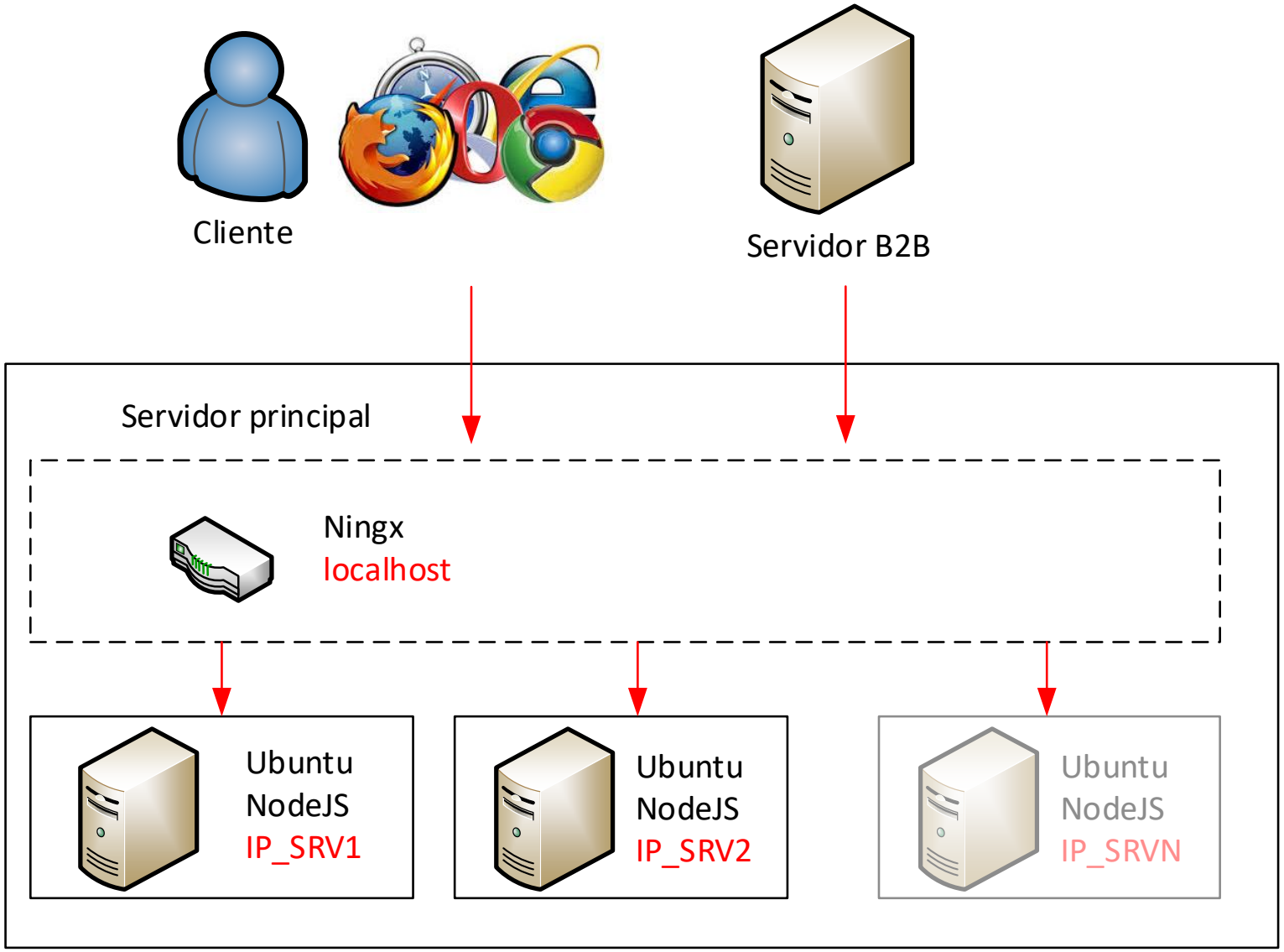
  docker stop webapp-node

- Opcionalmente deban utilizar "*sudo docker* …" en algunos ambientes

Escalabilidad - costo



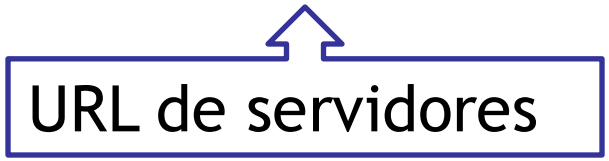
Y si sumamos
mas servidores?


Cliente
Servidor B2B
Servidor principal
Ningx
localhost
Ubuntu
NodeJS
IP_SRV1
Ubuntu
NodeJS
IP_SRV2
Ubuntu
NodeJS
IP_SRVN

Iniciar cluster

- Primero vamos a crear una carpeta al mismo nivel del contenedor de la aplicación donde pondremos la configuración del balanceador.
- Creemos el archivo de configuración de NginX (nginx/nginx.conf)

```
worker_processes 2;
events { worker_connections 1024; }
http {
          upstream node-app {
                    server node1:3000 ;
                    server node2:3000 ;
          }
server {
          listen 80;
          location / {
                    proxy_pass http://node-app;
          }
          }
}
```



Iniciar cluster

- El archivo de configuración de Docker-compose (Docker-compose.yml)

```
version: "2“
services:
        balancer:
                image: "nginx"
                volumes:
                - ./nginx/:/etc/nginx:ro
                links:
                - node1:node1
                - node2:node2

                ports:
                - "80:80"
                - "443:443"
```

Agregamos los nodos al cluster

- Nodo 1 y 2 en el archivo Docker-compose.yml

```
node1:
 image: ”servidor:latest"
 expose:
 - "3000"
 ports:
 - "3001:3000"
 cpu_quota: 25000
```

- Nodo 2

```
node2:
 image: ”servidor:latest"
 expose:
 - "3000"
 ports:
 - "3002:3000"
 cpu_quota: 25000
```

Prueba del cluster

- Iniciar el cluster con el siguiente comando:

  ```
  docker-compose up
  ```

- Para conocer qué servidor del cluster esta respondiendo, se debe utilizar la siguiente URL:
  - http://localhost/whoami

- Probar el rendimiento del cluster para el cálculo de fibonacchi.

Es privada la información que enviamos al servidor?

HTTP - Ejemplo

```
Escape character is '^]'.
GET /index.html HTTP/1.1
Host: www.example.com:80
User-Agent:yo

HTTP/1.1 200 OK
Date: Wed, 16 Sep 2009 12:50:03 GMT
Server: Apache/2.2.3 (Red Hat)
Last-Modified: Tue, 15 Nov 2005 13:24:10 GMT
ETag: "b80f4-1b6-80bfd280"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8

<HTML>
<HEAD>
  <TITLE>Example Web Page</TITLE>
</HEAD>
<body>
<p>You have reached this web page by typing &quot;example.com&quot;,
&quot;example.net&quot;,
  or &quot;example.org&quot; into your web browser.</p>
<p>These domain names are reserved for use in documentation and are not availabl
e
  for registration. See <a href="http://www.rfc-editor.org/rfc/rfc2606.txt">RFC
  2606</a>, Section 3.</p>
</BODY>
</HTML>
```

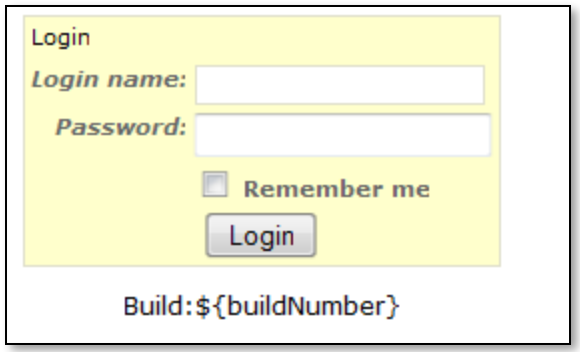
Request

Response

Veamos un ejemplo en el navegador…

HTTP - Problema

- Sniffing



- Captura

```
POST /cms-prototype/Login_login.action HTTP/1.1
Host: localhost:8080
..
Keep-Alive: 300
Connection: keep-alive
Referer: http://localhost:8080/cms-prototype/Login_view.action
Cookie: JSESSIONID=FD1B8BF5E9EAB0D6F0BD0D3076DFFFA4
Content-Type: application/x-www-form-urlencoded
Content-Length: 116

password=b6405986bcbacef04b814a26ecbb33e4afb&username=asas&__checkbox_rememberMe=
true
```

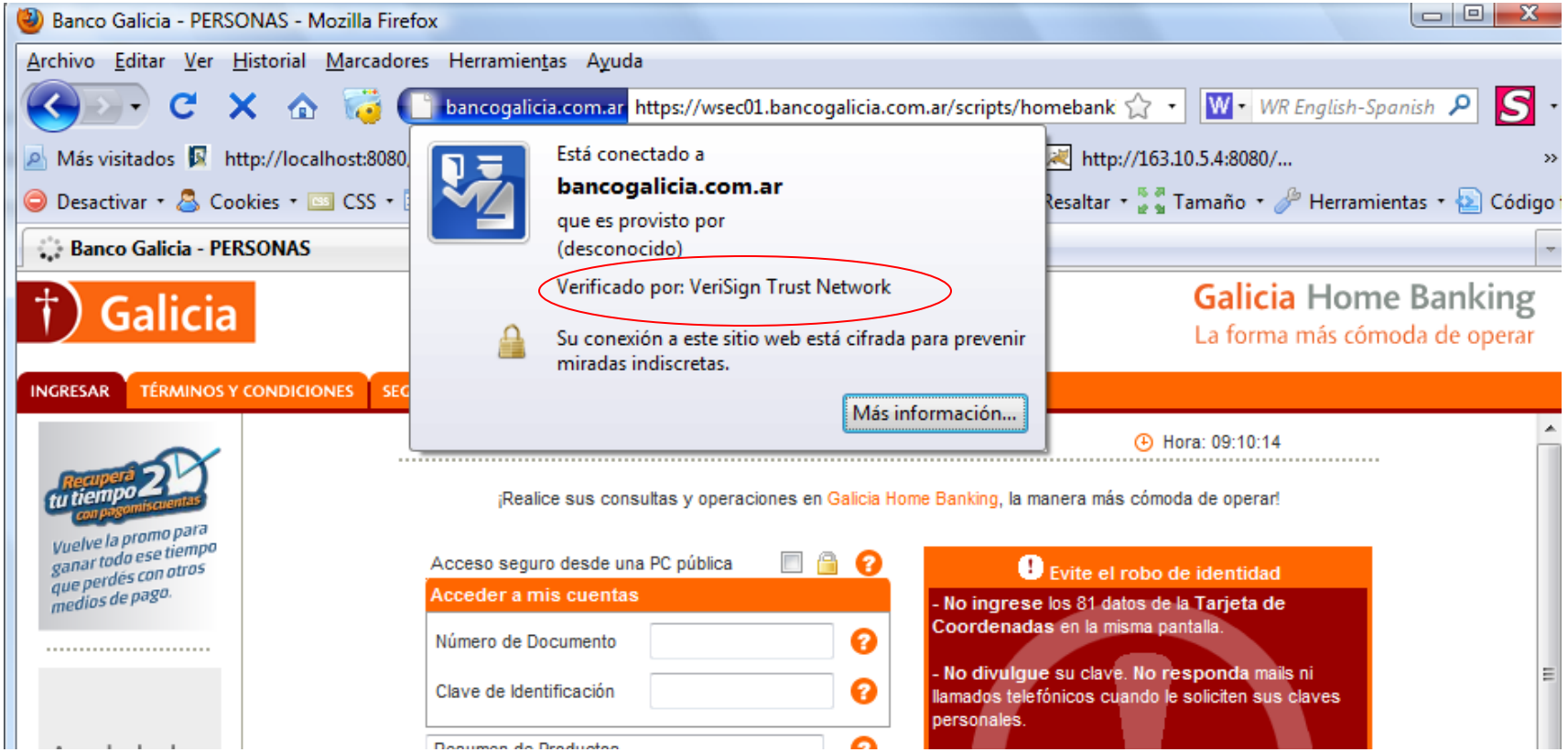
HTTPS - HTTP over SSL

- Hypertext Transfer Protocol Secure (HTTPS) = HyperText Transfer Protocol + SSL/TLS para
  - Proveer encriptación y asegurar la identificación del server.
  - Utilizados usualmente para comunicar información sensible como por ejemplo pagos electrónicos.
  - Evita "sniffing" de la red y ataques del tipo man-in-the-middle.
  - No debe ser confundido con Secure HTTP

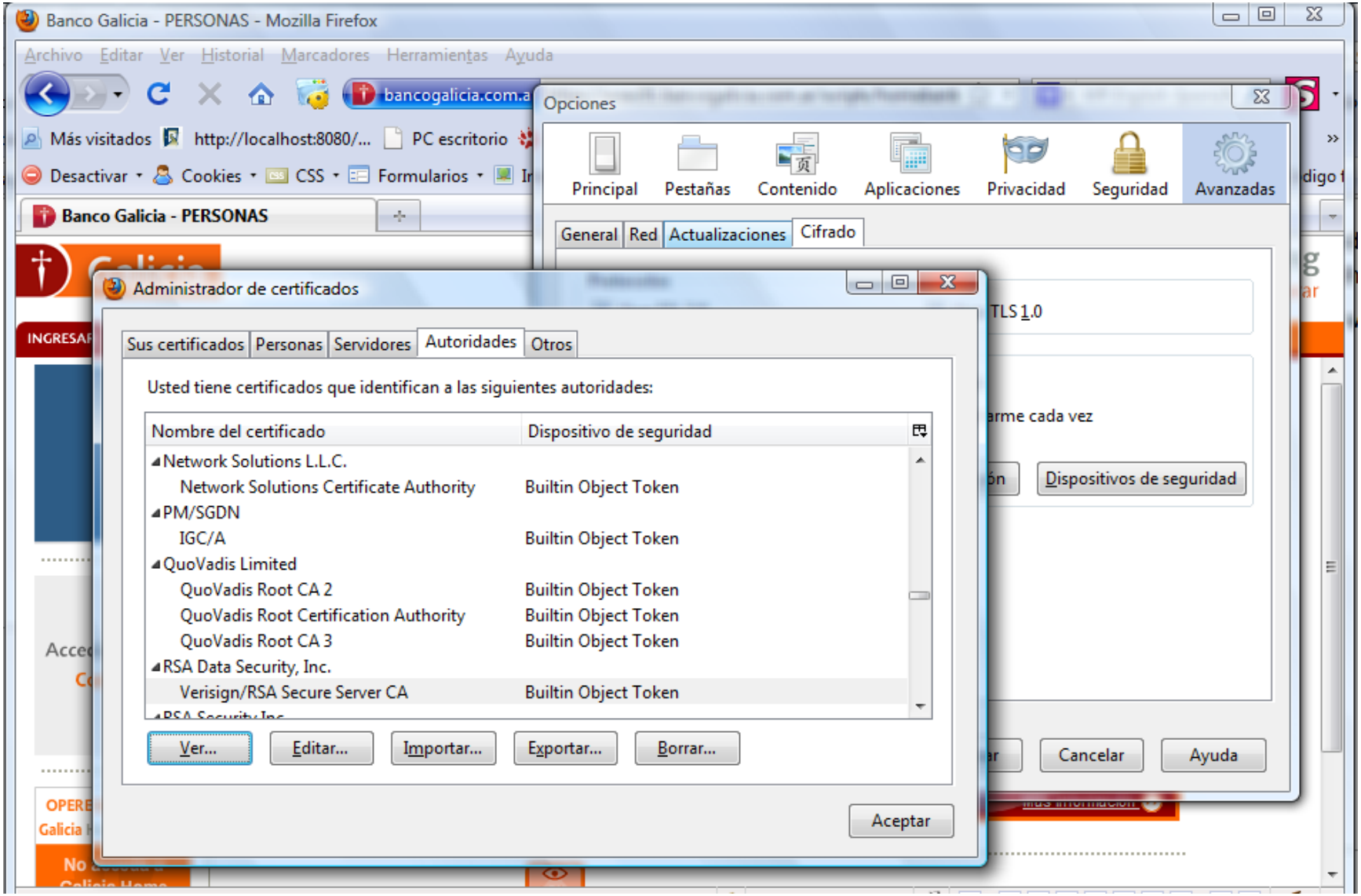
"Yo confío en la autoridad certificadora (Verisign,etc) para indicarme quién sería confiable."

- El usuario confía en la CA para que responda por sitios Web legítimos sin **errores de nombres**
- Un certificado de sitio es válido cuando fue firmado por una CA.

HTTPS - Ejemplo

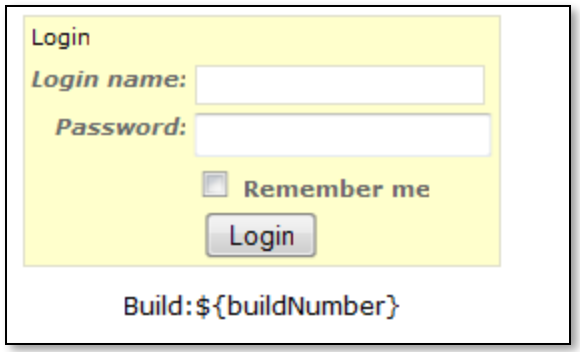


HTTPS - HTTP over SSL



HTTPS - Captura

- Sniffing



- Captura

```
CONNECT path.safehavenmn.org:443 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; es-AR; rv:1.9.1.2)
Gecko/20090729 Firefox/3.5.2 (.NET CLR 3.5.30729)
Proxy-Connection: keep-alive
Host: path.safehavenmn.org

Major Version: 3
Minor Version: 1
Random: 4A B2 40 82 22 A3 C1 …
SessionID: 4A B2 36 E4 A1 3A ..
Ciphers:
          [C00A] TLS1_CK_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
          [C014] TLS1_CK_ECDHE_RSA_WITH_AES_256_CBC_SHA
          [0088] Unrecognized cipher
          [0087] Unrecognized cipher
          [FEFF] SSL_RSA_FIPS_WITH_3DES_EDE_SHA
          [000A] SSL_RSA_WITH_3DES_EDE_SHA
```

Configuremos un NGINX - HTTPS

- Gestionamos certificados en una CA válida
  - Ambientes básicos: Startssl.com o letsencrypt.org para pruebas
  - Dominios complejos: generamos las claves 2048 bytes, y certificados con algún CA reconocido

Configuremos un NGINX - HTTPS

- Gestionamos certificados en una CA válida
  - Ambientes básicos: Startssl.com o letsencrypt.org para pruebas
  - Dominios complejos: generamos las claves 2048 bytes, y certificados con algún CA reconocido

Configuremos un NGINX - HTTPS

- Recursos necesarios
  - Una Web app (como la que venimos utilizando)
  - Instalar mkcerts
    - https://github.com/FiloSottile/mkcert#installation
  - Modificamos la tabla de hosts (/etc/hosts) para que dev.acme.com.ar sea un alias de localhost.
  - En Windows:
    - C:\Windows\System32\drivers\etc\hosts

    127.0.0.1 dev.acme.com.ar

Configuremos un NGINX - HTTPS

- Instalamos la app mkcert
  - Genera certificados confiables localmente para pruebas. https://github.com/FiloSottile/mkcert#installation

- Generamos un cert para acme.com.ar
  - Cluster

    http://bit.do/iaw-webapp-balancer

  - Modificamos la tabla de hosts (/etc/hosts) para que urbieta.com.ar sea un alias de localhost.

    127.0.0.1 server.acme.com.ar

  - En Windows:
    - C:\Windows\System32\drivers\etc\hosts
  - En OSX

    sudo vi /etc/hosts

Configuremos un NGINX - HTTPS

- Creemos una clave en la carpeta ssl

```
mkcert dev.acme.com.ar
```

- El resultado sería

```
nginx
└── nginx.conf
ssl
├── dev.acme.com.ar-key.pem
└── dev.acme.com.ar.pem
```

Configuremos un NGINX - HTTPS

- Incluimos la carpeta con certificados en la configuración de Docker-component

```
balancer:
        image: "nginx"
        volumes:
        - ./nginx/:/etc/nginx:ro
        - ./ssl/:/etc/ssl:ro
        links:
        - node1:node1
        - node2:node2
        ports:
        - "80:80"
        - "443:443"
```

Configuremos un NGINX - HTTPS

- Configuramos el conector HTTP para NginX

```
…
server {
        listen 443 ssl;
        server_name dev.acme.com.ar;
        fastcgi_param HTTPS on;
        ssl_certificate /etc/ssl/dev.acme.com.ar.pem;
        ssl_certificate_key /etc/ssl/dev.acme.com.ar-key.pem;
        ssl_protocols SSLv3 TLSv1 TLSv1.1 TLSv1.2;
        listen 80;
        location / {
                proxy_pass http://node-app;
        }
}….
```

Configuremos un Apache - HTTP

- Lo probamos!
  - https://localhost