Testes Linx+Neemu+Chaordic

Por Edilson Azevedo

Teste 1 - Binários

Binário: da87fa
Função: Cria - a cada 10secs - processos filhos usando o clone
Método utilizado
./da87fa

ps -aux |grep da87fa (pegar PID)

strace -f -p <PID>

Output de exemplo:

```
[pid 17664] exit_group(0)               = ?
[pid 17608] <... clone resumed> child_stack=0,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fec8e3ae9d0) = 17664
[pid 17664] +++ exited with 0 +++
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fec8e3ae9d0) = 17665
strace: Process 17665 attached
[pid 17665] exit_group(0)               = ?
[pid 17608] clone(strace: Process 17666 attached
 <unfinished ...>
[pid 17665] +++ exited with 0 +++
[pid 17608] <... clone resumed> child_stack=0,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fec8e3ae9d0) = 17666
[pid 17608] clone( <unfinished ...>
[pid 17666] exit_group(0)               = ?
strace: Process 17667 attached
[pid 17666] +++ exited with 0 +++
[pid 17667] exit_group(0)               = ?
[pid 17608] <... clone resumed> child_stack=0,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fec8e3ae9d0) = 17667
[pid 17667] +++ exited with 0 +++
nanosleep({10, 0}, {3, 34937876})     = ? ERESTART_RESTARTBLOCK (Interrupted by signal)
--- SIGINT {si_signo=SIGINT, si_code=SI_KERNEL} ---
+++ killed by SIGINT +++
```

Binário: cc9621
Função: Recebe input e salva-o em /tmp/$UID
Método utilizado
strace ./cc9621

Output de exemplo:

```
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f8be2bfb000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f8be2bfa000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f8be2bf9000
arch_prctl(ARCH_SET_FS, 0x7f8be2bfa700) = 0
mprotect(0x7f8be29ed000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ)     = 0
mprotect(0x7f8be2c1c000, 4096, PROT_READ) = 0
munmap(0x7f8be2bfc000, 121567)              = 0
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
brk(NULL)                                   = 0x2442000
brk(0x2463000)                                  = 0x2463000
read(0, sd
"sd\n", 1024)                               = 3
open("/tmp/eazevedo", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
fstat(3, {st_mode=S_IFREG|0664, st_size=0, ...}) = 0
write(3, "sd", 2)                           = 2
close(3)                                    = 0
lseek(0, -1, SEEK_CUR)                          = -1 ESPIPE (Illegal seek)
exit_group(0)                               = ?
+++ exited with 0 +++
eazevedo@eazevedo-aximcloud:~/Chaordic$ cat /tmp/eazevedo
sd
eazevedo@eazevedo-aximcloud:~/Chaordic$
```

Binário: 4caa50
Função: Conta o numero de caracteres em input
Método utilizado
strace ./4caa50

Output de exemplo:
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f2f86f45000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f2f86f44000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f2f86f43000
arch_prctl(ARCH_SET_FS, 0x7f2f86f44700) = 0
mprotect(0x7f2f86d37000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ) = 0
mprotect(0x7f2f86f66000, 4096, PROT_READ) = 0
munmap(0x7f2f86f46000, 121567) = 0
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
brk(NULL) = 0x1a2a000
brk(0x1a4b000) = 0x1a4b000
read(0, sdsdss
"sdsdss\n", 1024) = 7
lseek(0, -1, SEEK_CUR) = -1 ESPIPE (Illegal seek)
exit_group(6) = ?
+++ exited with 6 +++ (Inseridos 6 caracteres)

Binário: 334b32
Função: Executa 400 interações e plota imagens randomicas em /tmp/$UID
Método utilizado
strace ./334b32

Output de exemplo:

```
nanosleep({0, 1000000}, NULL)       = 0
nanosleep({0, 1000000}, NULL)       = 0
nanosleep({0, 1000000}, NULL)       = 0
nanosleep({0, 1000000}, NULL)       = 0
(...)
nanosleep({0, 1000000}, NULL)       = 0
brk(NULL)                           = 0x1445000
brk(0x1466000)                              = 0x1466000
open("/tmp/eazevedo", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
fstat(3, {st_mode=S_IFREG|0664, st_size=0, ...}) = 0
write(3, "                  \n        "..., 420) = 420
close(3)                            = 0
exit_group(0)                       = ?
+++ exited with 0 +++
```

```
        **
        *
```

```
                *
        * *
                *
```

Teste 2 - Ambiente

Descrição:

O ambiente deve ser todo configurado através de gerenciador de configuração, o que deverá ser entregue é um repositório git contendo os arquivos de configuração que serão aplicados em uma máquina virtual "zerada". Caso necessário descrever como executar o processo de aplicação da configuração na máquina virtual. Ao final da tarefa e execução do processo, deveremos ter um ambiente funcional;

- É recomendado que o repositório git seja entregue com commits pariciais, mostrando a evolução de seu código e pensamento. Caso prefira nos informe um url de git público ou então compacte todos os arquivos em um .tar.gz mantendo a pasta .git em conjunto;
- No ambiente deverá estar rodando uma aplicação node.js de exemplo, conforme código abaixo. A versão do node.js deverá ser a última versão LTS disponível em: https://nodejs.org/en/download/. A aplicação node abaixo possui a dependência da biblioteca express. Garanta que seu processo de bootstrap instale essa dependência (última versão estável disponível em: http://expressjs.com/) e rode o processo node em background. De uma forma dinâmica garanta que seja criado uma instância node para cada processador existente na máquina (a máquina poderá ter de 1 a 32 processadores);

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
res.send('Hello World!');
});
app.listen(3000, function () {
console.log('Example app listening on port 3000!');
});
```

- Construa dentro de sua automação um processo de deploy e rollback seguro e rápido da aplicação node. O deploy e rollback deverá garantir a instalação das dependências node novas (caso sejam adicionadas ou alteradas a versão de algum dependência por exemplo), deverá salvar a versão antiga para possível rollback e reiniciar todos processos node sem afetar a disponibilidade global da aplicação na máquina;
- A aplicação Node deverá ser acessado através de um Servidor Web configurado como Proxy Reverso e que deverá intermediar as conexões HTTP e HTTPS com os clientes e processos node. Um algoritmo de balanceamento deve ser configurado para distribuir entre os N processos node a carga recebida;
- A fim de garantir a disponibilidade do serviço, deverá estar funcional uma monitoração do processo Node e Web para caso de falha, o mesmo deve reiniciar ou subir novamente os serviços em caso de anomalia;

- Desenvolva um pequeno script que rode um teste de carga e demostre qual o Throughput máximo que seu servidor consegue atingir;

- Desenvolva um script que parseie o log de acesso do servidor Web e deverá rodar diariamente e enviar por e-mail um simples relatório, com a frequência das requisições e o respectivo código de resposta (ex:5 /index.html 200);
- Por fim; rode o seu parser de log para os logs gerados pelo teste de carga, garantindo que seu script terá perfomance mesmo em casos de logs com milhares de acessos;

Premissas:

- Não foi mencionada a utilização do ambiente AWS - o que honestamente tornaria o trabalho mais fácil com o uso de suas ferramentas. Pensando nisso, utilizei AWS apenas para a criação automatizada da instancia.
- Utilizei puppet para iniciar a instance com uma image padrão e bootstrapping.
- Utilizei cluster para a distribuição dos nodes e haproxy para concentração de acesso e balanceamento round robin.
- Para monitoramento do script utilizei o básico: Shell script e cron.
- Para estatísticas ativei o stats no haproxy (acessível de http://<host>/haproxy?stats)
- Para load test, utilizei o ab (apache benchmark) iniciado do Puppet Server

Material:

Todo o material pode ser encontrado em https://github.com/eazeved0/chaordic