Linx Impulse

RELATÓRIO TÉCNICO DO
Processo seletivo para Cloud Engineering

Nome: **Alessandro Elias**

E-mail: ale.elias2011@gmail.com

fone: +55 41 99888 8962

Curitba

02 de fevereiro de 2020

Sumário

Todos os arquivos mencionados neste documento podem ser encontrados em um dos repositórios públicos citados abaixo. Eles estão organizados da seguinte maneira:
Repositório com todo conteúdo do desafio:
https://github.com/alessandro11/linx-challenge.git

Para clonar o repositório e seus submódulos, execute:

$$git\ clone\ --recurse-submodules\ https://github.com/alessandro11/linx-challenge.git$$

```
ansible-desafio-2 @ d7df815  Submodule - repositório com os arquivos do Ansible
desafio-1 @ 210e48b          Submodule - com binários e disassembly,
desafio-2 @ 15e4ef4          Submodule - com aplicação NodeJS
report                Dir - este relatório, fonte em latex
results               Dir - resultados e logs das analises
```

Ou cada repositório separado:
https://github.com/alessandro11/ansible-desafio-2.git
https://github.com/alessandro11/desafio-1.git
https://github.com/alessandro11/desafio-2.git

1 DESAFIO - Parte 1

Estamos enviando pelos links abaixo alguns binários executáveis (ELF 64-bit LSB) que realizam tarefas bem simples, que podem ou não ser úteis. O exercício é que você descubra o que esses binários fazem, utilizando as ferramentas que julgar mais adequadas. Como resposta, esperamos que você nos diga o que você acha que eles fazem e quais foram as ferramentas usadas para isso, bem como uma linha geral do seu raciocínio para chegar às conclusões.

Nesta seção, para encontrar as soluções dos problemas propostos, foi utilizado primordialmente as seguintes ferramentas:

`objdump (GNU Binutils) 2.33.1`

Esta ferramenta foi utilizada para obter a desmontagem (disassembly) dos binários, bem como fazer uma analise das seções e símbolos dos binários. Desta forma, possibilitando uma leitura do código montado (assembly).

Os seguintes comandos foram utilizados para analise das seções e ou cabeçalhos:
Para analises de cabeçalhos e seções

$$objdump\ -x\ <nome\_do\_arquivo>$$

$$objdump\ -T\ <nome\_do\_arquivo>$$

$$objdump\ -f\ <nome\_do\_arquivo>$$

Para disassembly:

$$objdump\ -S\ <nome\_do\_arquivo>$$

$$objdump\ -D\ <nome\_do\_arquivo>$$

$$objdump\ -d\ <nome\_do\_arquivo>$$

`gdb (GDB) 8.3.1`

Esta ferramenta possibilitou a confirmação de qualquer conjectura obtida pela ferramenta anterior (*objdump*), executando o assembly passo a passo.

Sua execução com os seguintes parâmetros:

$$gdb\ -tui\ <programa>$$

1.1 Salvando entrada do usuário em /tmp/$USER

Binário: cc9621

Disassembly: https://github.com/alessandro11/desafio-1/blob/master/cc9621.as

Os binários, seus respectivos disassembly e um *leia-me* podem ser encontrados no repositório: https://github.com/alessandro11/desafio-1

Este programa abre um arquivo no diretório "/tmp/", sendo o nome do arquivo extraído da variável de ambiente ($USER) do programa em execução. E então o programa espera ser digitado um texto (máximo 24 carácteres) para salvar o arquivo "/tmp/$USER".

```
000000000040072d <main>:
  40072d:       push   %rbp
  40072e:       mov    %rsp,%rbp
  400731:       push   %rbx
  400732:       sub    $0x1f8,%rsp
  400739:       mov    %fs:0x28,%rax
  400742:       mov    %rax,-0x18(%rbp)
  400746:       xor    %eax,%eax
  /*
   * The instruction below assign the string "/tmp/".
   * Note the optimization of the compiler result in a
   * direct assignment, no memoey access.
   * (Little endian) 0x2f=/ 0x74=t 0x6d=m 0x70=p 0x2f=/
   * The value "/tmp/" is stored at a variable on the stack.
   */
  400748:       movabs $0x2f706d742f,%rax
  400752:       mov    %rax,-0x1f0(%rbp)
  400759:       lea    -0x1e8(%rbp),%rsi
  400760:       mov    $0x0,%eax
  400765:       mov    $0x18,%edx
  40076a:       mov    %rsi,%rdi
  40076d:       mov    %rdx,%rcx
  /*
   * The following instruction fill the buffer with zeros pointing to
   * register %rsi, by %rcx times (0x18 = 24 chars). In C usually the
   * call would be memset(buff, 24, '\0');
   */
  400770:       rep stos %rax,%es:(%rdi)
  /* The address 0x4008b4 point to string "USER". */
  400773:       mov    $0x4008b4,%edi
  /*
   * The call getenv("USER"); it will return a char* at %rax with the user
   * logged.
   */
```

```
400778:       callq  4005b0 <getenv@plt>
40077d:       mov    %rax,-0x200(%rbp)
400784:       mov    -0x200(%rbp),%rdx
40078b:       lea    -0x1f0(%rbp),%rax
400792:       mov    %rdx,%rsi
400795:       mov    %rax,%rdi
/*
 * strcat(%rdi, %rsi) will concatenate the strings:
 * %rdi = "/tmp/" with, %rsi = "m3cool"; results the string
 * "/tmp/m3cool".
 * WARNING:
 *          m3cool is my user at my computer, so this program
 * when run from other computer/user may change.
 * the result string is stored at variable on the stack at offset
 * -0x1f0 from the base pointer register.
 */
400798:       callq  400630 <strcat@plt>
40079d:       lea    -0x120(%rbp),%rax
4007a4:       mov    %rax,%rsi
/* 0x4008b9 stores the string "%s" */
4007a7:       mov    $0x4008b9,%edi
4007ac:       mov    $0x0,%eax
/* scanf waits for a string to be typed, as the parameter at
 * %edi = "%s" indicates
 */
4007b1:       callq  400620 <__isoc99_scanf@plt>
4007b6:       lea    -0x1f0(%rbp),%rax
/* 0x4008bc stores string "w" for fopen */
4007bd:       mov    $0x4008bc,%esi
4007c2:       mov    %rax,%rdi
/*
 * tries to open a file for write only.
 * -0x1f8(%rbp) = fopen("/tmp/m3cool", "w");
 */
4007c5:       callq  400610 <fopen@plt>
4007ca:       mov    %rax,-0x1f8(%rbp)
4007d1:       cmpq   $0x0,-0x1f8(%rbp)
4007d9:       je     40080a <main+0xdd>
4007db:       mov    -0x1f8(%rbp),%rdx
4007e2:       lea    -0x120(%rbp),%rax
4007e9:       mov    %rdx,%rsi
4007ec:       mov    %rax,%rdi
/*
 * In case fopen succeeded, saves the content typed
 * in the file opened at "/tmp/m3cool".
 */
4007ef:       callq  4005e0 <fputs@plt>
```

```
4007f4:       mov    -0x1f8(%rbp),%rax
4007fb:       mov    %rax,%rdi
/* closes the stream file opened at "/tmp/m3cool"
4007fe:       callq  4005c0 <fclose@plt>
/*
 * The remaining instructions is to return and
 * shutdown the process gracefully.
 */
400803:       mov    $0x0,%eax
400808:       jmp    40080f <main+0xe2>
40080a:       mov    $0x1,%eax
40080f:       mov    -0x18(%rbp),%rbx
400813:       xor    %fs:0x28,%rbx
40081c:       je     400823 <main+0xf6>
40081e:       callq  4005d0 <__stack_chk_fail@plt>
400823:       add    $0x1f8,%rsp
40082a:       pop    %rbx
40082b:       pop    %rbp
40082c:       retq
40082d:       nopl   (%rax)
```

Assembly 1: Escreve a entrada do usuário em "/tmp/$USER".

No Assembly 1 é mostrado o *main* do programa (as outras seções foram truncadas) assim focamos apenas no núcleo de que o programa faz (não irei abordar cada instrução, apenas as relevantes). Também é apresentado como comentários em inglês mais detalhes das instruções relevantes.

A primeira instrução relevante é 4007{48,52}, após a linha 15. No qual armazena a string "/tmp/" em uma variável na pilha. A instrução 400770 preenche com zeros um buffer de tamanho 24, indicado pela constante armazenado em *RCX*, no qual controla quantas vezes será repetido a instrução *stos*. Este buffer será usado pelo *scanf*. Na instrução 400778 é obtido o usuário que está executando o programa, em meu caso o usuário é "m3cool". A 400798 concatena a string "/tmp/" com "m3cool". Na 4007b1 é feito a chamada ao *scanf* para obter a entrada do usuário, na string. A 4007c5 abre o arquivo no caminho "/tmp/m3cool". Na 4007ef *fputs* escreve a string obtida pelo *scanf* no arquivo aberto. Na 4007fe fecha o arquivo e o programa é encerrado.

1.2 Conway's Game of Life

Binário: d3ea79
Disassembly: https://github.com/alessandro11/desafio-1/blob/master/d3ea79.as

Este programa gera a saída do jogo Conway's Game of Life, escrevendo-o no arquivo "/tmp/$USER". O link [1] para a Wiki do jogo também é escrito no arquivo. A saída gerada na execução deste programa

[1] http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

está no repositório em /tmp/m3cool.

O disassembly parcial do binário pode ser observado nas rotinas abaixo:

```
00000000004011c7 <main>:
...
  4011eb:       callq  40094d <clear>
...
  4011f5:       callq  4009e6 <welcome>
...
  401208:       callq  400fc2 <play>
```

Assembly 2: Remove e recria o arquivo "/tmp/$USER" com o Conway's Game of Life

No Assembly 2 observamos as respectivas chamadas: *clear*, *welcome* e *play*. Iremos abordar cada uma das rotinas.

clear:

```
000000000040094d <clear>:
...
  400967:       movabs $0x2f706d742f,%rax
...
  400992:       mov    $0x4012a8,%edi
  400997:       callq  400750 <getenv@plt>
...
  4009bc:       callq  4007b0 <strncat@plt>
...
  4009cb:       callq  400760 <unlink@plt>
...
```

Assembly 3: Remove arquivo "/tmp/$USER"

A instrução 400967 atribui ao registrador *RAX* a string "/tmp/". Na instrução 400992 é atribuído o ponteiro para a string (constante) "USER", no qual é passado como parâmetro para *getenv*. Este retorna o valor da variável de ambiente $USER. A instrução 4009bc concatena a string que estava em *RAX* e valor da variável de ambiente. No meu caso, resultando na string "/tmp/m3cool". A 4009cb remove o arquivo "/tmp/m3cool".

welcome:

```
00000000004009e6 <welcome>:
...
  400a00:       movabs $0x2f706d742f,%rax
...
  400a2b:       mov    $0x4012a8,%edi
  400a30:       callq  400750 <getenv@plt>
...
```

```
  400a55:       callq  4007b0 <strncat@plt>
...
  400a69:       callq  400810 <fopen@plt>
...
  400aa7:       mov    $0x4012c3,%edi
  400aac:       callq  400830 <fwrite@plt>
...
  400ab8:       mov    $0x4012e8,%esi
  400ac5:       callq  4007f0 <fprintf@plt>
...
  400ad4:       callq  400780 <fclose@plt>
...
```

Assembly 4: Escreve cabeçalho no arquivo "/tmp/$USER"

O Assembly 4, na instrução 400a00 à 400a55 concatena strings como em 3. Instrução 400a69 abre o arquivo ("/tmp/m3cool"). Na instrução 400aa7 carrega ponteiro para string (constante) "Welcome to the Game of Life.\n" a subsequente escreve no arquivo esta string. A instrução 400ab8 carrega a string (constante) "http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life" e subsequente escreve no arquivo e então fecha-o.

play, ***print***, ***evolve*** são rotinas responsáveis por gerar o jogo.

1.3 Fork()

Binário: da87fa

Disassembly: https://github.com/alessandro11/desafio-1/blob/master/d3ea79.as

Este programa fica em um laço infinito, ou até receber um SIGHUP, executando *fork()* dez vezes. Na décima, é executado um *sleep* de dez segundos, e torna a executar outros dez forks. Como é mapeado SIGCHLD para SIGHUP, o processo filho inicia e encerra.

Analisando a desmontagem do binário no Assembly 5:

```
000000000040060d <main>:
  40060d:       push   %rbp
  40060e:       mov    %rsp,%rbp
  400611:       push   %rbx
  400612:       sub    $0x38,%rsp

  400616:       mov    $0x1,%esi
  40061b:       mov    $0x11,%edi
  400620:       callq  4004d0 <signal@plt>
  400625:       mov    $0x0,%ebx
  40062a:       jmp    400650 <main+0x43>
```

```
40062c:     callq  400510 <fork@plt>
400631:     movslq %ebx,%rdx
400634:     mov    %eax,-0x40(%rbp,%rdx,4)
400638:     movslq %ebx,%rax
40063b:     mov    -0x40(%rbp,%rax,4),%eax
40063f:     test   %eax,%eax
400641:     jne    40064d <main+0x40>
400643:     mov    $0x0,%edi
400648:     callq  4004f0 <exit@plt>
40064d:     add    $0x1,%ebx
400650:     cmp    $0x9,%ebx
400653:     jle    40062c <main+0x1f>
400655:     mov    $0xa,%edi
40065a:     mov    $0x0,%eax
40065f:     callq  400500 <sleep@plt>
400664:     jmp    400625 <main+0x18>
400666:     nopw   %cs:0x0(%rax,%rax,1)
```

Assembly 5: Fork do processo

As instruções 4006{16, 1b, 20} mapeiam o sinal de SIGCHLD para SIGHUP, o que acarreta na finalização do processo, assim que o processo filho inicia. Da instrução 40062a à 400653 ocorre o laço dos forks. Nas instruções 40065{0, 3} controla quantos forks serão feitos enquanto o registrador *EBX* não atinge dez, um if. Quando o registrador *EBX* atinge dez, então é executado um *sleep* de dez segundos e volta para o inicio do laço.

O ponto interessante a se observar neste código é a condição de saída do laçom a fim de finalizar o programa. Conforme as instruções 40063f executa uma comparação *test*, no qual somente será igual se o registrador *EAX* for zero, porém a instrução lê o pid do processo filho que foi salvo na pilha, atribui este valor ao registrador *EAX* e executa o *test*, logo não é atribuído pid zero para processos, por definição do SO. Então a instrução 400641 sempre fará o salto para a verificação se já executou os dez forks antes do *sleep*, e nunca executará 400648, finalizar processo.

1.4 Multi Thread escutando na porta 8011

Binário: ddb1c9
Disassembly: https://github.com/alessandro11/desafio-1/blob/master/ddb1c9.as

Este programa abre três threads e atende requisições http, cujo programa escuta em localhost na porta 8011. Ao fazer uma requisição http para `http://localhost:8011` apenas o caractere "!" é retornado. A depuração deste código é bem complicado, por ter sido compilado com a linguagem GO é gerado um backtrace de mais de 20 chamadas, e em paralelo para todas as threads, o resulta em vários fluxos de execuções. Portando a abordagem de leitura do dissasembly, como nos anteriores, e depuração via *gdb* não foram suficientes.

Para triangular o que o programa faz, utilizei mais algumas ferramentas: *lsof* para identificar quais descritores de arquivos abertos ele possui. O programa e seus parâmetros *netstat -natup* (*ss -latp*), no qual retornam processos e suas respectivas portas abertas. O *netcat localhost 8011*/*telnet localhost 8011* para tentar enviar GET, e outros comandos aleatórios. Para identificar possível comunicação nesta porta utilizei *tcpdump -vv -n -i lo port 8011* para monitorar o tráfego de rede.

Como conclusão do funcionamento deste programa, reforço o parágrafo um. Ele atende requisições http retornado "!".

2 DESAFIO - Parte 2

"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 parciais, 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);

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 performance mesmo em casos de logs com milhares de acessos;

Nesta seção irei abordar em linhas gerais a ideia da minha implementação, portanto não irei me deter em explicar o código, de qualquer forma se houver alguma dúvida estou a disposição para esclarecer qualquer ponto."

2.1 Introdução

Pós a leitura do enunciado do desafio, comecei a selecionar quais tecnologias iria utilizar, com exceção da NodeJS já pré estabelecida. Dentro dos meus critérios para a seleção das tecnologias estão: performance, atividade e correções de possíveis bugs, bem como novas funcionalidades, ou seja, desenvolvimento ativo da tecnologia. Como serviço http selecionei Nginx, sua característica de abrir processos leve (threads) ao invés de processos já o torna atraente. Como referência posso citar [2] com uma opinião, com tudo há diversos sites fazendo avaliações dos prós e contras de Apache e Nginx, os mais utilizados. Os outros quesitos também foram atendidos.

Ferramentas de automação, há diversas no mercado, como *Chef*, *Puppet*, *Ansible*, *Saltstack* etc. Selecionei a ***Ansible*** devido a Linx Impulse mencionar um diferencial na posição em aberto para Cloud Engineering.

Quanto aos scripts utilizei Bash e Python, no qual é nativo e amplamente utilizado por muitas distribuições Linux, muito comum em infraestrutura de nuvem.

Para simular meu ambiente, gerei três máquinas virtuais utilizando a Virtual Machine Monitor (VMM) Qemu e libvirt. Criei uma segunda rede e configurei Network Address Translation (NAT) pelo meu desktop para chegar a Internet. Gerei uma imagem de Ubuntu 18.04, e apliquei as mudanças em cima de alguns "snapshoots copy on write".

Serviços de inicialização da aplicação NodeJS foi utilizado o sistema nativo, *Systemd*, apesar de existirem outros como *PM2*, recomendado para node, *Supervisor* mais antigo, são boas ferramentas, com tudo se estes processos (daemon) falham, a aplicação ficará "órfã" de monitoramento. Por outro lado o *systemd* pode sofrer a mesma coisa, porém ele é o processo 1, em caso de falha todo o sistema operacional para, logo o confiamos, portanto é o mais robusto e ainda nativo. Como gerente de tarefas para os scripts *crontab*.

2.2 Ambiente utilizado

Em minha mini infra estrutura utilizei distribuições ArchLinux (pessoal), e Ubuntu como servidor, segue versão:

```
Distributor ID: Ubuntu
Description: Ubuntu 18.04.4 LTS
Release: 18.04
Codename: bionic

Linux webserver 4.15.0-76-generic #86-Ubuntu SMP
Fri Jan 17 17:24:28 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
```

[2] https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations

```
VM's com 4 VCPU's e 4 GB de RAM cada.

ArchLinux
Ansible
ansible-playbook 2.9.4
python version = 3.8.1
```

2.3 Execução do gerenciador de configuração; *Ansible*

Primeiramente é necessário preencher as variáveis do ambiente do Ansible: *create_user*, *copy_local_key*, *domain*, *release_version*. Para envio de e-mail: *email_crontab*, *smtp_server*, *smtp_port*, *email_from*, *email_to*, *email_login*, *email_pass*.

Esta primeira implementação visa deixar flexível o lançamento, nos quesitos: caminho de instalação da aplicação, domínio em que será ofertado a aplicação, versão do primeiro lançamento e configurações do usuário para enviar e-mail. Porém esta é uma versão beta no qual identifiquei um problema, desescalar privilégio dos nodes workers, no qual está estático. **Não mude o nome do usuário**, pois causará erro ao tentar desescalar. Ao corrigir este bug, seria possível escolher o nome do usuário (caminho) onde será instalado a aplicação. Outra melhora seria adicionar prefixo do caminho, mais nome do usuário.

Outro problema é não tratamento de erro (com um try, catch) no script python que, em caso de falha no módulo smtp, a aplicação dispara uma exceção de código, onde o crontab irá disparar e-mail com o erro.

Um problema operacional são os dados sensíveis do usuário de e-mail, geralmente em ambientes corporativos (nuvem) há um serviço de smtp sem a necessidade de autenticação, para hosts específicos, o que dispensaria este trecho. De qualquer forma o usuário que irá fazer o deploy pode preencher os dados e manter em sigilo em sua máquina de trabalho, com tudo, ficará exposto em texto puro no servidor. Acredito que este quesito, (apesar de super relevante) para este desafio neste momento dispense uma implementação com um mecanismo com hash ou até mesmo obter credenciais de um serviço centralizado, como LDAP.

Este *playbook* é dividido em duas partes, primeiro irá executar os plays como *root*, e depois como o usuário *webserver*. Em algum ponto é necessário escalar privilégios: como reiniciar/recarregar o serviço da aplicação, e assim o play o faz. Devido o primeiro contato com *Ansible* não apliquei as melhores práticas, porém são funcionais. Posso citar *become*, para escalar privilégios, pois ao tentar, falha. Isto ocorreu devido ao pensamento em sempre deixar o mínimo de privilégios para usuários de aplicação no servidor, o ideal é zero privilégios, mas sabemos que no mundo real nem sempre é possível. Para permitir o mínimo de privilégios, configurei o *sudoers* para dois comandos bem específicos, e devem ser executadas exatamente como está na regra, (*/bin/systemctl <start/reload> webserver.service*), sem por e nem tirar, por definição do *sudo*. O play com *become* falha como se não estive no *sudoers*, provavelmente o Ansible está executando **sem** o caminho absoluto, então não é aceito pelo *sudo*. Nestes casos imple-

mentei via script inline.

Executando o playbook:

$$ansible - playbook \quad playbook.yml \quad -l \quad < server >$$

Ao final da execução deste playbook você terá um ambiente funcional rodando a aplicação NodeJS de exemplo, e algumas rotas adicionais para gerar uma carga artificial na aplicação.

Os passos de cada play é:

1. Instalar os pacotes requeridos. A lista destes estão definida na variável do ambiente *Ansible*.
2. Criação de um usuário sem privilégios. Com exceção: iniciar e recarregar a aplicação node [3]. Configurar este usuário para executar ssh no localhost, para o primeiro deploy, ao final será removido a chave ssh sem senha gerada pelo play.
3. Cadastrar a chave pública (*ssh*) da origem do comando para o servidor, possibilitando mais segurança no ambiente de deploy.
4. Resiliência do serviço, através do *systemd*. No qual uma vez que processo(s) da aplicação morre, ele automaticamente o reinicia, e mais script de monitoramento para tal.
5. Nginx com Proxy Reverso; balanceando a carga com *round-robin* entre os possíveis 32 cores. Com um node worker para core, mais um Master.
6. Script que todos os dias à meia noite faz o parse do access log e gera uma tabela agregada por url (recurso), código e suas respectivas frequências de acesso.
7. Instalação do ambiente node virtual Node Version Manager (NVM).
8. Instalação do repositório da aplicação node.
9. Fácil e seguro mecanismo de *deploy* e *rollback* através do shipit.

2.3.1 Disponibilidade da aplicação

O mecanismo nativo do systemd 4 permite a monitoração dos processos, e em caso de falha e ou não resposta, os processos com falha recebem o sinal para encerrar e iniciar novamente, e ainda em caso de falha do envio do sinal ele permanece tentando com um intervalo de 5 sec (configurável). Um pequeno engano no comentário do código (commit: 134c9d) fez a carga do servidor subir, pois o systemd incessantemente, inciava o processo da aplicação node e em seguida ele "morria", pois a plicação gerava uma exceção de código não trada, neste caso comentários com '#', ao invés de padrão C (//, /**).

[3] Isto se faz necessário, devido a implementação através de clusterização

E ainda bash script minuto a minuto verificando se o processo está rodando, e se uma rota está retornando o código OK (200), se um dos casos não for satisfeito o serviço é reiniciado.

O link para o script pode ser encontrado em:
https://github.com/alessandro11/desafio-2/blob/master/scripts/health_check.sh

2.3.2 Balanceamento de carga

Além do mecanismo de balanceamento do item 5, foi implementado o serviço (daemon) utilizando cluster de workers (processos nodes), em outras palavras o daemon server.js irá executar um processo a mais do que o número de cores do servidor, este é o Master, quem delega para os demais workers, gera o arquivo de de seu pid, logs etc. O Master também possui mecanismo de balanceamento de carga, com tudo foi implementado balanceamento de carga com o Nginx. O motivo da preferência por Nginx é uso de memória, este é mais eficiente consequentemente utilizando menos memória. As conexões https estão configuradas, porém é preciso gerar o certificado e apontar para o caminho do certificado gerado.

A *daemon* da aplicação foi construída de forma modular, ou seja, *server.js*, apenas inclui *app.js* no qual poderia ter sua própria estrutura hierarquia de middlewares, rotas etc, como é mostrado no trecho de código abaixo:

Trecho de código retirado do fonte server.js:

```
...
100: } else {  // New worker running; load app
        var app = require('./app');

        app.listen(port, function () {
            console.log('Example app listening on port ' + port + '!');

            // Check if we are running as root
            if (process.getgid() === 0) {
                process.setgid('webserver');
                process.setuid('webserver');
            }
        });
112: }
...
```

A configuração do upstream pode ser encontrado no seguinte link:
https://github.com/alessandro11/ansible-desafio-2/blob/master/setup_desafio-2/webserver.com

2.3.3 Throughput máximo

A métrica utilizada em meu teste foi conexões por segundo simultaneamente. O script abaixo gera conexões com o comando *curl ... -m 2 -w %{http_code} %{time_starttransfer}*, no qual retorna código de respostas e tempo das requisições em no máximo 2 segundos.

Executando o teste (sem / no final do domínio):

$$./workload.sh < num\ req > < domain >$$

Caso não seja passado nenhum parâmetro, será executado 100 requisições para o domínio *webserver.com* (meu domínio local, em minha infra; note que este domínio tem registro),

Um em bash executa requisições em paralelo ao servidor http. Como não há carga no servidor pelo programa de exemplo, no qual apenas retorna 'Hello World!' código OK (200) (não gera carga). Então para gerá-la artificialmente, gerei outras três rotas que aplica carga simbólica no servidor. Cuja carga, é gerar contadores aleatórios, esperar por alguns milissegundos. Podemos observar o servidor como idle entre um sleep e outro como se estivesse esperando resposta de outro componente do sistema, deste modo pude contrapor a primeira carga. Esta, resultou em possíveis centenas de requisições simultâneas o que é irreal. Com a carga artificial cheguei em média atingir 24 conexões simultâneas, com apenas dois workers (duas VCPUS),

O script e resultados das análises, logs, estão no repositório em:
https://github.com/alessandro11/desafio-2/tree/master/scripts.

2.3.4 Parser do log e relatório por e-mail

Um script em python foi desenvolvido para está tarefa, este pode ser encontrado no link ao final desta seção.

Execução do script:
(Caso queira receber por e-mail é necessário ter preenchido as variáveis do Ansible, conforme seção 2.3.)

$$./log\_parser.py$$

A saída será uma tabela (no stdout) com as colunas URL, COD. e FREQ, este é o número de requisições nas últimas 24h, agregadas por código e url.

Explorando o algoritmo eficiente de dicionário do python, é possível agregar os dados de logs com milhares de linhas, em uns dos testes fora executado em um arquivo com mais 50k linhas. Os resultados e logs podem ser obtidos no repositório: resultados.

A entrada abaixo foi gerada no *crontab* do root (é necessário acesso ao */var/log/nginx/access_webserver.log*) para ser executado todos os dias à meia-noite.

```
0 0    * * *    /home/webserver/current/scripts/log\_parser.py
```

O fonte do script pode ser encontrado no link:
https://github.com/alessandro11/desafio-2/blob/master/scripts/log_parser.py

2.3.5 Deploy e Rollback

Este mecanismo foi implementado através de dois pacotes nodes obtidos com *npm*: shipit-cli@5.1.0 e shipit-deploy@5.1.0. **Para fazer o depoy, é necessário executar o git push das mudanças no repositório que se deseja lançar**.

Primeiramente você deve editar o arquivo shipitfile.js localizado na raiz do repositório, alterando as seguintes variáveis:

Home do usuário onde foi clonado o repositório no servidor. **Caso não tenha alterado o playbook não é necessário modificar.**

Aponte para um caminho que não seja a raiz do repositório, Este cuidado deve ser tomado se for setado a variável para limpar o diretório do deploy. Ou seja, em minha configuração, será clonado para */home/webserver/desafio-2*, e o deploy ocorrerá em */home/webserver/releases*.

```
...
   default: {
       deployTo: /home/webserver/
       ...
  }
...
```

Aponte para o branch ou tag que deseja fazer deploy,

```
...
banch: 'master',
ou
tag: 'rc-0.0.1',
...
```

Usuário e domínio/IP do servidor que será feio o deploy.

```
...
    production: {
        servers: 'webserver@webserver.com',
    },
...
```

É possível fazer uma sobre escrita das configurações padrões, como *branch* por exemplo, caso a configuração *production* possua um branch chamado production:

```
...
    production: {
```

```
        servers: 'webserver@webserver.com',
        branch: 'production',
    },
...

...
    development: {
        servers: 'webserver@webserverdev.com',
        branch: 'dev',
    },
...
```

Aponte para o caminho da $HOME onde a aplicação está. Este é o trigger que é gerado após o deploy com sucesso, então possíveis novos pacotes serão instalados, bem como o serviço irá recarregar os workers com as novas mudanças.

```
    await shipit.remote('cd current; \
         /home/webserver/.nvm/nvm-exec npm install; \
         sudo /bin/systemctl reload webserver.service')
```

Os serviços foram gerados para procurar a home do usuário em /home, portanto não altere o prefixo.

2.3.6 Deploy

Execute o comando abaixo a partir do diretório do *shipitfile.js* (**a *branch* ou *tag*** deve estar no repositório remoto):

npx shipit production deploy

A chave ssh será usado para acessar o servidor e executar o deploy. Será extraído o último commit da branch ou tag apontado pelo shipitfile e clonado para o diretório *<deployTo:>/releases* no servidor remoto. Você encontrará os diretórios com o último commit do deploy, o nome dos diretórios são um timestamp. Um link simbólico chamado *<deployTo:>/current* aponta para o diretório do deploy atual em *<deployTo:>/releases/20200208160632>* e.g.

2.3.7 Rollback

Execute o comando abaixo a partir do diretório do *shipitfile.js*:

npx shipit production rollback

Tudo que ocorrerá é apontar o link simbólico para o deploy anterior, e recarregar os nodes workers, recarregando a aplicação. Está pré configurado para armazenar os últimos dez deploys, isto pode ser configurado no shipitfile.