SÃO
PAULO
TECH
SCHOOL

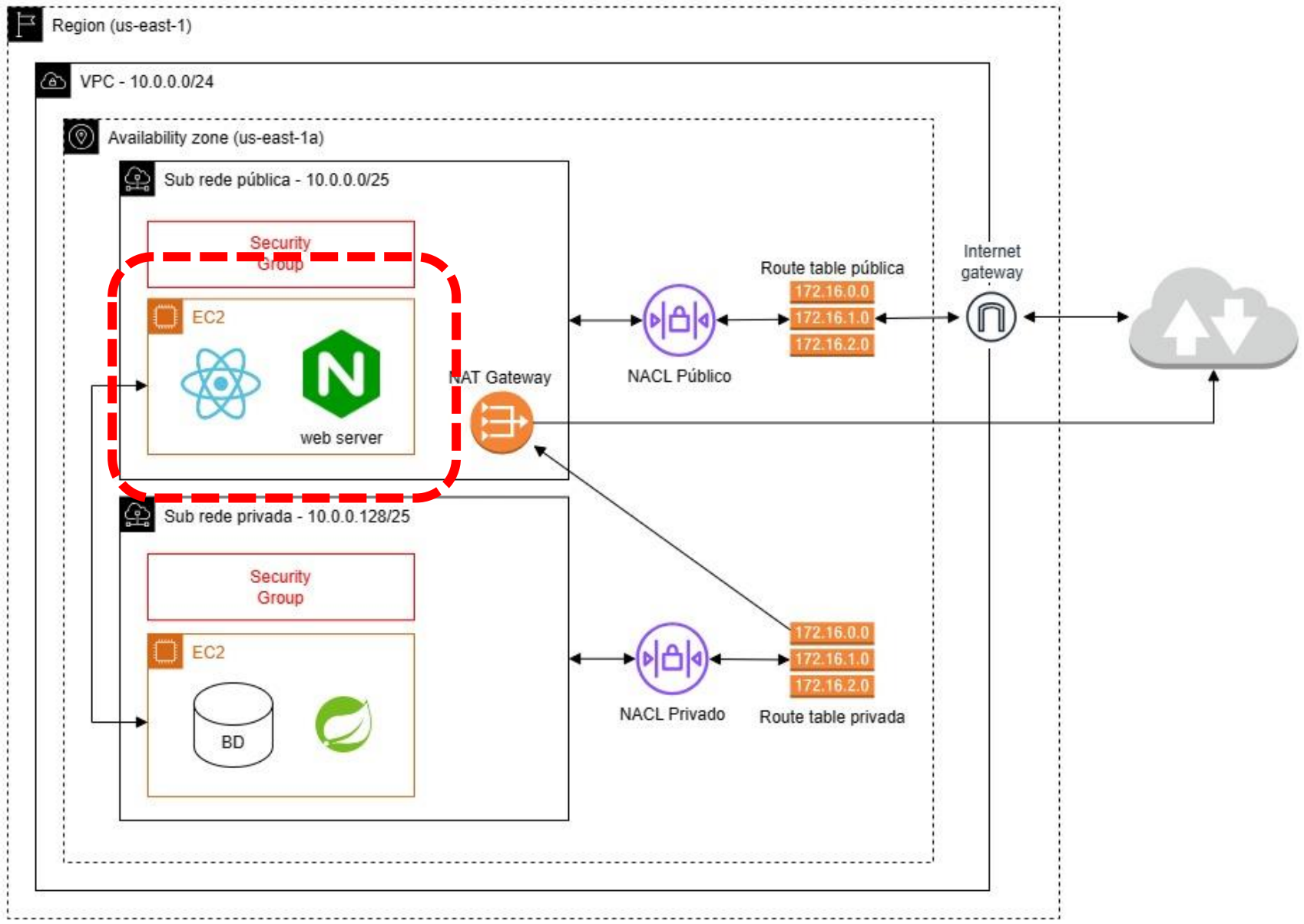
Computação e sistemas distribuídos

NGINX - Arquitetura e configuração pt.1

Diego Brito

diego.brito@sptech.school

NGINX – Arquitetura e configuração pt.1



Introdução à arquitetura NGINX

Modelo de Processamento Baseado em Eventos

O **NGINX** é conhecido por sua arquitetura escalável e de alto desempenho, principalmente devido ao seu modelo de processamento baseado em **eventos.** Este modelo permite que o NGINX maneje milhares de conexões simultâneas de forma eficiente, usando uma quantidade muito pequena de memória. Diferentemente dos servidores web tradicionais que criam um novo processo ou thread para cada conexão, o NGINX usa um modelo não bloqueante e orientado a eventos para servir solicitações. Isso significa que um número limitado de **worker processes** pode lidar com um grande número de conexões simultâneas, aguardando eventos e processando-os quando ocorrem.

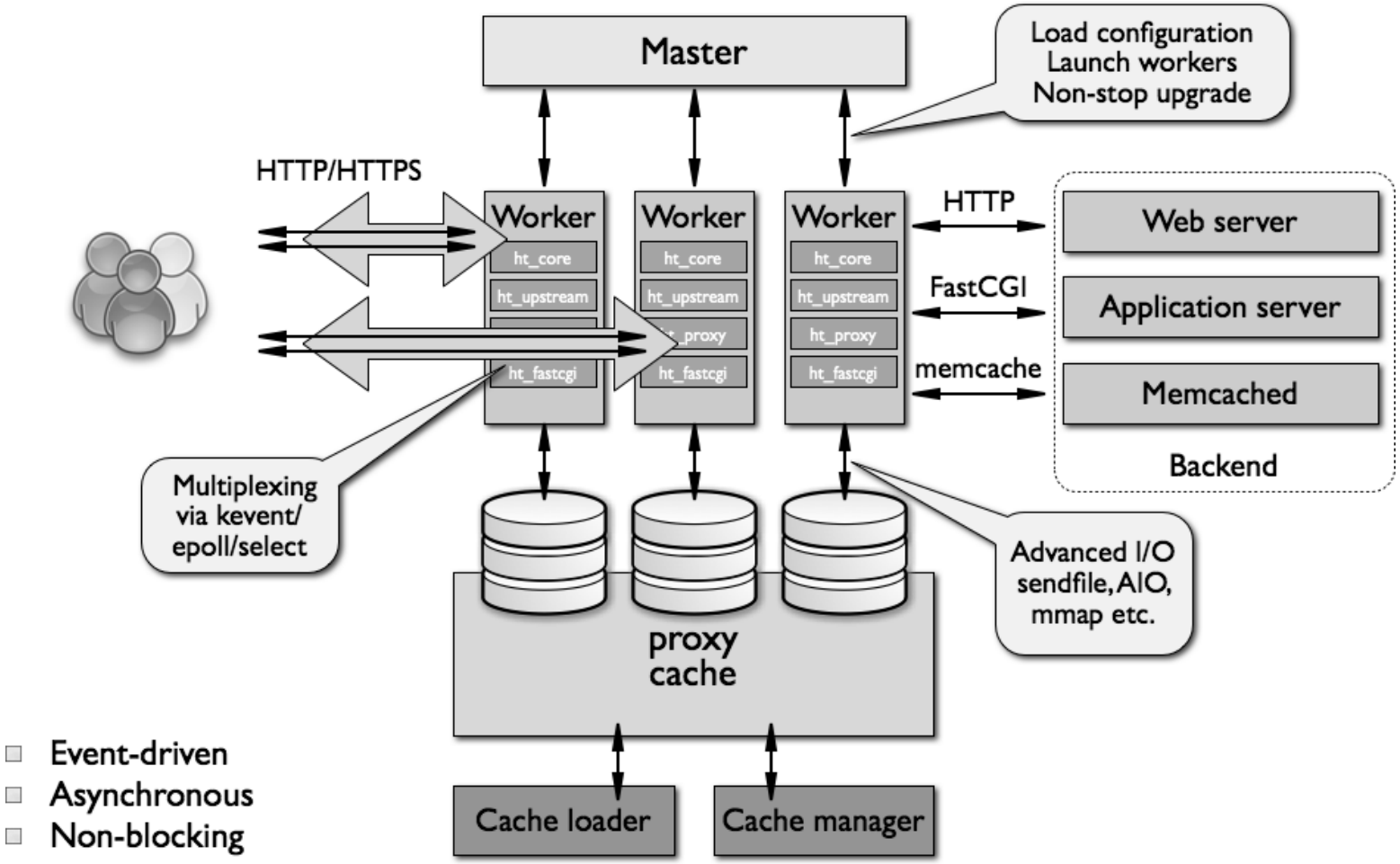
Benefícios da Arquitetura do NGINX

- **Alto Desempenho e Escalabilidade**: Graças ao seu modelo de processamento, o NGINX pode servir um número significativamente maior de solicitações por segundo em comparação com servidores que utilizam arquiteturas baseadas em processos ou threads. Isso o torna ideal para sites de alto tráfego e aplicações exigentes.
- **Uso Eficiente de Recursos**: O NGINX consome menos recursos de memória e CPU, graças à sua abordagem de manipulação de conexões. Isso se traduz em menor custo de hardware para suportar a mesma quantidade de tráfego, em comparação com outras arquiteturas.
- **Flexibilidade e Controle**: A arquitetura do NGINX oferece aos administradores de sistema um controle granular sobre a configuração e o comportamento do servidor, permitindo otimizações específicas para diferentes cargas de trabalho e ambientes.

Comparação com Arquiteturas Baseadas em Threads

- **Arquiteturas Tradicionais**: Muitos servidores web tradicionais, como o Apache na configuração padrão, usam um modelo baseado em processos ou threads, criando um novo para cada conexão. Embora isso possa ser eficaz para um número moderado de conexões, pode se tornar um gargalo quando o número de conexões simultâneas aumenta, levando a um aumento no uso de CPU e memória.
- **NGINX**: Em contraste, o NGINX evita esse problema ao utilizar um número fixo de **worker processes** para gerenciar todas as conexões. Cada worker pode lidar com milhares de conexões simultaneamente, graças ao uso de estruturas de dados eficientes e a um loop de eventos para processar solicitações. Isso não apenas melhora a escalabilidade e o desempenho, mas também garante uma utilização mais eficiente dos recursos do servidor.

Arquitetura Nginx



- Event-driven
- Asynchronous
- Non-blocking

Arquitetura Nginx

- **Master:** Inicializa o sistema, lê e aplica a configuração, e gerencia os processos workers. Pode recarregar sua configuração sem interrupções e realizar upgrades sem paradas.
- **Workers:** São os processos que realmente lidam com as requisições de rede. Cada worker é capaz de lidar com milhares de conexões simultâneas de forma eficiente devido ao modelo de eventos não-bloqueantes.
- **Multiplexing via kqueue/epoll/select:** É o mecanismo de I/O utilizado para monitorar vários file descriptors para ver se eles estão prontos para realizar I/O sem bloquear. epoll é usado em sistemas Linux, kqueue em BSDs e select pode ser usado em sistemas que não têm nem epoll nem kqueue.

Arquitetura Nginx

- **Proxy Cache:** O NGINX pode armazenar em cache conteúdo estático e dinâmico, reduzindo a carga sobre os servidores de aplicação e acelerando a entrega de conteúdo.
- **Cache Loader e Cache Manager:** Estes são processos que gerenciam o cache de proxy, carregando-o após um reinício e gerenciando a validade e a expiração dos dados em cache.
- **Web Server:** Serve conteúdo estático, por exemplo, imagens e HTML.
- **Application Server:** Aqui rodam aplicações de backend (por exemplo, PHP, Python, Ruby).
- **Memcached:** É um sistema de armazenamento em memória para objetos pequenos que também pode ser usado para cache.

Arquivos de configuraçõ̃es

Por onde começo?

- Você encontrará duas abordagens principais para configurar o NGINX em tutoriais e documentações. Alguns materiais sugerirão a edição ou adição de arquivos dentro do diretório **conf.d**, enquanto outros recomendarão o uso de arquivos dentro do diretório **sites-available.**
- O NGINX oferece flexibilidade em como você organiza e gerencia suas configurações. A escolha entre **conf.d** e **sites-available (juntamente com sites-enabled)** geralmente depende de preferências pessoais, convenções de equipe ou requisitos específicos do projeto.

Diferenças

- **conf.d:** Esta é uma pasta padrão lida pelo NGINX, que automaticamente inclui todos os arquivos de configuração dentro dela. Colocar arquivos aqui significa que eles serão automaticamente carregados sem necessidade de links simbólicos. É ideal para configurações globais ou quando você tem muitos pequenos arquivos de configuração que prefere manter separados por clareza ou organização.

Diferenças

- **sites-available/sites-enabled:** Esta é uma convenção comum no Debian e em sistemas baseados em Ubuntu, inspirada no Apache. Os arquivos de configuração para cada site ficam armazenados em sites-available. Para ativar um site, você cria um link simbólico dele em sites-enabled. Isso oferece mais controle sobre quais configurações de site estão ativas em qualquer momento, já que você pode facilmente ativar ou desativar sites criando ou removendo links simbólicos.

Diferença Principal:

- A principal diferença é o controle de ativação e desativação de sites ou serviços específicos**.** Usar **sites-available e sites-enabled** permite um **controle mais granular**, ativando ou desativando sites sem alterar os arquivos originais ou remover arquivos de configuração. Já o diretório **conf.d é mais direto,** pois tudo o que está dentro dele é automaticamente carregado, sem a necessidade de gerenciar links simbólicos.

- **Ambos os métodos são válidos** e a escolha entre um e outro depende das necessidades de gerenciamento da configuração e da preferência pessoal ou da equipe.

Formato do arquivo

```
# Redireciona todas as solicitações HTTP para HTTPS
server {
    listen 80;
    server_name seu-dominio.com www.seu-dominio.com;

    # Redirecionamento para HTTPS
    return 301 https://$host$request_uri;
}

# Configuração do servidor HTTPS
server {
    listen 443 ssl;
    server_name seu-dominio.com www.seu-dominio.com;

    # Localização dos certificados SSL/TLS obtidos via Let's Encrypt
    ssl_certificate /etc/letsencrypt/live/seu-dominio.com/fullchain.pem; # Caminho para o certificado completo
    ssl_certificate_key /etc/letsencrypt/live/seu-dominio.com/privkey.pem; # Caminho para a chave privada

    # Servindo conteúdo estático do diretório raiz
    location / {
        root /var/www/seu-dominio.com/html; # Caminho para o diretório com conteúdo estático
        index index.html index.htm;
    }

    # Configuração do Proxy Pass para /api
    location /api {
        proxy_pass http://10.0.0.1; # Endereço IP do servidor backend de classe A
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Decifrando termos

- **location:** Define como NGINX deve responder a requisições para recursos específicos dentro de um servidor. O conteúdo do bloco location define rotas e comportamentos para URIs específicas.
- **root**: Define o diretório raiz do servidor ou de uma localização específica, de onde o NGINX serve os arquivos.
- **index**: Especifica os arquivos que devem ser usados como índices quando um diretório é requisitado.

Decifrando termos

- **proxy_pass**: Define o protocolo e endereço de um servidor proxy para o qual o tráfego deve ser redirecionado.
- **proxy_set_header**: Modifica ou adiciona campos ao cabeçalho da requisição antes de enviar para o servidor backend.
- **ssl_certificate** e **ssl_certificate_key**: Especificam os caminhos para o certificado SSL/TLS e sua chave privada, respectivamente, usados para estabelecer conexões seguras.

Decifrando termos

- **proxy_cache:** Ativa o cache para respostas de um proxy reverso, sendo necessário definir uma zona de cache.
- **worker_processes:** Diretiva global que define o número de processos de trabalho do NGINX. Geralmente, é ajustado para o número de núcleos da CPU.
- **worker_connections:** Define o número máximo de conexões que cada processo de trabalho pode abrir. É uma diretiva dentro do bloco events.

Agradeço
a sua atenção!

SÃO
PAULO
TECH
SCHOOL