



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Nginx Essentials

Excel in Nginx quickly by learning to use its most essential features in real-life applications

Valery Kholodkov

[PACKT] open source*
PUBLISHING

community experience distilled

Nginx Essentials

Excel no Nginx rapidamente aprendendo a usar seus recursos mais essenciais em aplicativos da vida real

Valery Kholodkov



BIRMINGHAM - MUMBAI

Nginx Essentials

Copyright © 2015 Packt Publishing

Todos os direitos reservados. Nenhuma parte deste livro pode ser reproduzida, armazenada em sistema de recuperação ou transmitida de qualquer forma ou por qualquer meio, sem a permissão prévia por escrito do editor, exceto no caso de breves citações incorporadas em artigos críticos ou resenhas.

Todos os esforços foram feitos na preparação deste livro para garantir a precisão das informações apresentadas. No entanto, as informações contidas neste livro são vendidas sem garantia, expressa ou implícita. Nem o autor nem a Packt Publishing e seus revendedores e distribuidores serão responsabilizados por quaisquer danos causados ou alegadamente causados direta ou indiretamente por este livro.

A Packt Publishing se esforçou para fornecer informações sobre marcas registradas de todas as empresas e produtos mencionados neste livro, usando letras maiúsculas de forma apropriada.

No entanto, a Packt Publishing não pode garantir a precisão dessas informações.

Publicado pela primeira vez: julho de 2015

Referência de produção: 1170715

Publicado por Packt Publishing Ltd.

Local de libré

Rua da libré 35

Birmingham B3 2PB, Reino Unido.

ISBN 978-1-78528-953-8

www.packtpub.com

Créditos

Autor

Valery Kholodkov

Coordenador de projeto

Vijay Kushlani

Revisores

Markus Jelsma

Revisor

Saber Edição

Jesse Estill Lawson

Indexador

Daniel Parraz

Priya Sane

Editor comissionado

Dipika Gaonkar

Coordenador de produção

Shantanu N. Zagade

Editor de Aquisição

Usha Iyer

Trabalho de capa

Shantanu N. Zagade

Editor de Desenvolvimento de Conteúdo

Nikhil Potdukhe

Editor Técnico

Manali Gonçalves

Editor de cópia

Roshni Banerjee

Sobre o autor

Valery Kholodkov é um profissional de TI experiente com uma década de experiência na criação, construção, dimensionamento e manutenção de serviços da Web de nível industrial, aplicativos da Web e back-ends de aplicativos móveis. Ao longo de sua carreira, ele trabalhou para marcas conhecidas, como Yandex, Booking.com e AVG. Atualmente, ele trabalha para sua própria empresa de consultoria. Valery tem um profundo conhecimento de tecnologia e é capaz de expressar sua essência, vantagens e riscos para um leigo, o que o torna um autor talentoso de livros de tecnologia.

Sobre os revisores

Markus Jelsma é CTO e co-proprietário da Openindex BV, uma empresa holandesa especializada em soluções de busca e rastreamento de código aberto. Como committer e membro do PMC do Apache Nutch, ele é especialista em tecnologia de mecanismos de pesquisa e soluções de rastreamento da web.

Jesse Estill Lawson é cientista da computação e pesquisador em ciências sociais que trabalha no ensino superior. Ele consultou dezenas de faculdades em todo o país para ajudá-los a projetar, desenvolver e implantar sistemas de informações de computador em tudo, desde Windows e Apache a Nginx e servidores de nó, e concentra sua pesquisa na coexistência de ciência de dados e sociologia. Além de sua formação tecnológica, Jesse possui mestrado em inglês e atualmente está trabalhando em seu doutorado em educação. Você pode saber mais sobre ele em seu site em <http://lawsonry.com>.

Daniel Parraz é administrador de sistemas Linux com 15 anos de experiência em sites de varejo eletrônico de alto volume, armazenamento de grande sistema e empresas de segurança. Ele está atualmente trabalhando com um provedor de serviços gerenciados, onde é responsável por todos os aspectos de sistemas do tipo Unix na organização. Daniel também foi editor técnico do *Learning Nagios 4*, *Packt Publishing*, e co-escreveu material de treinamento para o servidor de armazenamento IBM DS8000.

Gostaria de agradecer à minha família, amigos e mentores pelo apoio constante ao longo dos anos.

www.PacktPub.com

Arquivos de suporte, eBooks, ofertas de desconto e muito mais
Para arquivos de suporte e downloads relacionados ao seu livro, visite www.PacktPub.com.

Você sabia que a Packt oferece versões eBook de todos os livros publicados, com PDF e ePub files disponíveis? Você pode atualizar para a versão eBook em www.PacktPub.com e como cliente do livro impresso, você tem direito a um desconto na cópia do eBook. Entre em contato conosco em service@packtpub.com para mais detalhes.

Em www.PacktPub.com, você também pode ler uma coleção de artigos técnicos gratuitos, inscrever-se em uma variedade de boletins informativos gratuitos e receber descontos e ofertas exclusivas em livros e e-books Packt.



<https://www2.packtpub.com/books/subscription/packtlib>

Você precisa de soluções instantâneas para suas questões de TI? PacktLib é a biblioteca de livros digitais online da Packt. Aqui, você pode pesquisar, acessar e ler toda a biblioteca de livros da Packt.

Por que se inscrever?

- Totalmente pesquisável em todos os livros publicados pela Packt
- Copie e cole, imprima e marque o conteúdo
- Sob demanda e acessível por meio de um navegador da web

Acesso gratuito para titulares de contas Packt

Se você tiver uma conta com Packt em www.PacktPub.com, você pode usar isso para acessar o PacktLib hoje e ver 9 livros totalmente gratuitos. Basta usar suas credenciais de login para acesso imediato.

Índice

Prefácio

Capítulo 1: Começando com o Nginx	1
Instalando o Nginx	1
Instalando o Nginx no Ubuntu	2
Alternativas	2
Instalando o Nginx no Red Hat Enterprise Linux ou CentOS/Scientific Linux	3
Instalando o Nginx a partir de arquivos de origem	4
Baixando os arquivos de origem do Nginx	5
Construindo Nginx	6
Copiando a configuração do código-fonte de pacotes pré-compilados	7
A estrutura da instalação do Nginx	8
A pasta de configuração do Nginx	8
A pasta de host virtual padrão	8
A pasta de configuração de hosts virtuais	9
A pasta de registro	9
A pasta temporária	9
Configurando o Nginx	9
Tipos de valor	10
Variáveis	10
Inclusões	12
Seções	13
A seção http	13
A seção do servidor	13
A seção a montante	14
A seção de localização	14
Simples	14
Exato	15
Locais de expressão regular	15
A seção if	15
A seção limit_except	18

Índice

Outros tipos de seção	18
Regras de herança das configurações de configuração	18
A primeira configuração de amostra	21
Práticas recomendadas de configuração	22
Resumo	23
Capítulo 2: Gerenciando o Nginx	25
A arquitetura de processamento de conexão Nginx	26
Iniciando e parando o Nginx	28
Sinais de controle e seu uso	30
Desligamento rápido	31
Desligamento normal	31
Reconfiguração	32
Reabrindo o arquivo de log	34
Atualização binária do Nginx	35
Desligamento gracioso do trabalhador	37
Finalizando o procedimento de atualização	38
Lidando com casos difíceis	39
Scripts de inicialização específicos de distribuição	40
Alocando processos de trabalho	40
Configurando o Nginx para servir dados estáticos	42
Instalando certificados SSL	44
Criando uma solicitação de assinatura de certificado	44
Instalando um certificado SSL emitido	45
Redirecionamento permanente de um host virtual não seguro	46
Gerenciando arquivos temporários	47
Como comunicar problemas aos desenvolvedores	49
Criando um binário com informações de depuração	50
Resumo	50
Capítulo 3: Proxy e Cache	51
Nginx como proxy reverso	51
Configurando o Nginx como um proxy reverso	52
Configurando o back-end da maneira certa	53
Adicionando transparência	54
Manipulando redirecionamentos	55
Manipulação de cookies	57
Usando SSL	58
Manipulação de erros	59
Escolhendo um endereço IP de saída	60
Acelerando downloads	61

Índice

Cache	61
Configurando caches	61
Ativando o cache	63
Escolhendo uma chave de cache	64
Melhorando a eficiência e a disponibilidade do cache	66
Tratamento de exceções e casos limítrofes	68
Resumo	70
Capítulo 4: Reescrever Mecanismo e Controle de Acesso	71
O básico do mecanismo de reescrita	71
Mais sobre regras de reescrita	74
Padrões	76
Capturas e parâmetros posicionais	77
Outras funcionalidades do mecanismo de reescrita	77
Atribuindo variáveis	77
Avaliando predicados usando seções if	78
Respondendo com um código de status HTTP especificado	79
Controle de acesso	80
Restringindo o acesso por endereço IP	80
Usando a diretiva geo para restringir o acesso por endereço IP	82
Usando autenticação básica para restrição de acesso	85
Autenticação de usuários com uma subsolicitação	88
Combinando vários métodos de restrição de acesso	90
Resumo	91
Capítulo 5: Gerenciando o tráfego de entrada e saída	93
Gerenciando o tráfego de entrada	93
Limitando a taxa de solicitação	94
Limitando o número de conexões simultâneas	96
Limitando a taxa de transferência de uma conexão	97
Aplicando várias limitações	98
Gerenciando o tráfego de saída	99
Declarando servidores upstream	99
Usando servidores upstream	101
Escolhendo uma estratégia de distribuição de solicitações	103
Configurando servidores de backup	106
Determinando se um servidor está disponível	107
Ativando conexões persistentes	109
Limitando a taxa de transferência de uma conexão upstream	110
Resumo	110

Índice

Capítulo 6: Ajuste de desempenho	111
Otimizando a recuperação de arquivos estáticos	111
Ativando a compactação de resposta	114
Otimizando a alocação de buffer	116
Habilitando a reutilização de sessão SSL	120
Alocação de processos de trabalho em sistemas multicore	123
Resumo	124
Índice	125

Prefácio

2006 foi um ano emocionante. A decepção que cercou o crash das pontocom foi praticamente superada por um crescimento renovado e mais confiável da Web 2.0 e inspirou uma busca por tecnologias de uma nova era.

Naquela época, eu estava procurando um servidor web para alimentar meus projetos que faria muitas coisas de uma maneira diferente. Depois de obter alguma experiência em projetos on-line de grande escala, eu sabia que a popular pilha LAMP era abaixo do ideal e às vezes não resolia certos desafios, como uploads eficientes, limitação de taxa geodependente assim por diante.

Depois de tentar e rejeitar várias opções, conheci o Nginx e imediatamente senti que minha busca havia terminado. É pequeno, mas poderoso, com uma base de código limpa, boa extensibilidade, conjunto relevante de recursos e vários desafios de arquitetura resolvidos. Nginx definitivamente se destacou da multidão!

Imediatamente me inspirei e senti alguma afinidade com este projeto. Tentei participar da comunidade Nginx, aprendi, compartilhei meus conhecimentos e contribuí com o que pude.

Com o tempo, meu conhecimento do Nginx cresceu. Comecei a receber pedidos de consultoria e tenho conseguido atender casos bastante sofisticados. Depois de algum tempo, percebi que alguns dos meus conhecimentos poderiam valer a pena compartilhar com todos. Foi assim que comecei um blog em www.nginxguts.com.

Um blog acabou por ser um meio autor-dirigido. Um meio mais focado no leitor e mais completo estava em demanda, então reservei algum tempo para reunir meu conhecimento na forma mais sólida de um livro. Foi assim que o livro que você está segurando em suas mãos agora surgiu.

Prefácio

O que este livro cobre

O Capítulo 1, Introdução ao Nginx, fornece o conhecimento mais básico sobre o Nginx, incluindo como realizar a instalação básica e colocar o Nginx em funcionamento rapidamente. Uma explicação detalhada da estrutura do arquivo de configuração é fornecida para que você saiba exatamente onde os trechos de código do restante do livro se aplicam.

O Capítulo 2, Gerenciando o Nginx, explica como gerenciar uma(s) instância(s) operacional(is) do Nginx.

O Capítulo 3, Proxy and Caching, explica como transformar o Nginx em um poderoso proxy e cache da Web.

O Capítulo 4, Mecanismo de reescrita e controle de acesso, explica como usar o mecanismo de reescrita para manipular URLs e proteger seus recursos da web.

O Capítulo 5, Gerenciando o tráfego de entrada e saída, descreve como aplicar várias restrições ao tráfego de entrada e como usar e gerenciar o upstream.

O Capítulo 6, Ajuste de desempenho, explica como extrair o máximo do seu servidor Nginx.

O que você precisa para este livro

É necessário um bom conhecimento de sistemas operacionais do tipo Unix, presumivelmente Linux, juntamente com alguma experiência em webmaster.

Para quem é este livro

Este livro pretende enriquecer o conhecimento dos webmasters e engenheiros de confiabilidade de sites sobre sutilezas conhecidas por aqueles que têm uma compreensão profunda do núcleo do Nginx. Ao mesmo tempo, este livro é um guia *inicial* que permite que os iniciantes mudem facilmente para o Nginx sob orientação experiente.

Convenções

Neste livro, você encontrará vários estilos de texto que distinguem diferentes tipos de informação. Aqui estão alguns exemplos desses estilos e uma explicação de seu significado.

Palavras de código em texto, nomes de pastas, nomes de arquivos, extensões de arquivos, nomes de caminho, URLs fictícios e entrada do usuário são mostrados da seguinte forma: "Podemos incluir outros contextos por meio do uso da diretiva include ."

Prefácio

Um bloco de código é definido da seguinte forma:

```
tipos {
    text/html text/          html htm html;
    css text/xml            css;
    image/gif               xml;
    image/jpeg              gif;
    application/x-          jpeg jpg;
    javascript application/atom+xml   js;
    application/rss+xml     átomo;
                                rss;
}
```

Quando desejamos chamar sua atenção para uma parte específica de um bloco de código, as linhas ou itens relevantes são definidos em negrito:

```
tipos {
    text/html text/          html htm html;
    css text/xml            css;
    image/gif               xml;
    image/jpeg              gif;
    application/x-          jpeg jpg;
    javascript application/atom+xml   js;
    application/rss+xml     átomo;
                                rss;
}
```

Qualquer entrada ou saída de linha de comando é escrita da seguinte forma:

```
# cp /usr/local/nginx/nginx.conf.default
      /etc/nginx/nginx.conf
```

Novos termos e palavras importantes são mostrados em negrito. As palavras que você vê na tela, por exemplo, em menus ou caixas de diálogo, aparecem no texto assim: "Clicar no botão Avançar move você para a próxima tela".



Avisos ou notas importantes aparecem em uma caixa como esta.



Dicas e truques aparecem assim.

Prefácio

Elições de seções de arquivos de configuração são mostradas como [...] ou com um comentário [... esta parte do arquivo de configuração é com você...]

Comentários do leitor

O feedback dos nossos leitores é sempre bem-vindo. Deixe-nos saber o que você pensa sobre este livro - o que você gostou ou não gostou. O feedback dos leitores é importante para nós, pois nos ajuda a desenvolver títulos dos quais você realmente tirará o máximo proveito.

Para nos enviar um feedback geral, basta enviar um e-mail para feedback@packtpub.com e mencionar o título do livro no assunto da sua mensagem.

Se houver um tópico no qual você tenha experiência e estiver interessado em escrever ou contribuir para um livro, consulte nosso guia do autor em www.packtpub.com/authors.

Supporte ao cliente

Agora que você é o orgulhoso proprietário de um livro Packt, temos várias coisas para ajudá-lo a obter o máximo de sua compra.

Errata

Embora tenhamos tomado todos os cuidados para garantir a precisão de nosso conteúdo, erros acontecem. Se você encontrar um erro em um de nossos livros - talvez um erro no texto ou no código - ficaríamos gratos se você pudesse relatar isso para nós. Ao fazer isso, você pode salvar outros leitores da frustração e nos ajudar a melhorar as versões subsequentes deste livro. Se você encontrar alguma errata, por favor, denuncie-a visitando [com/submit-errata](https://www.packtpub.com/books/com/submit-errata), selecionando seu livro, clicando no Formulário de Envio de Errata link, e inserindo os detalhes de sua errata. Assim que sua errata for verificada, seu envio será aceito e a errata será carregada em nosso site ou adicionada a qualquer lista de errata existente na seção Errata desse título.

Para visualizar a errata enviada anteriormente, acesse <https://www.packtpub.com/books/content/support> e digite o nome do livro no campo de busca. As informações necessárias aparecerão na seção Errata.

Pirataria

A pirataria de material protegido por direitos autorais na Internet é um problema contínuo em todas as mídias. Na Packt, levamos muito a sério a proteção de nossos direitos autorais e licenças. Se você encontrar cópias ilegais de nossos trabalhos de qualquer forma na Internet, forneça o endereço do local ou o nome do site imediatamente para que possamos buscar uma solução.

Entre em contato conosco em copyright@packtpub.com com um link para o material suspeito de pirataria.

Agradecemos sua ajuda para proteger nossos autores e nossa capacidade de oferecer conteúdo valioso.

eBooks, ofertas de desconto e muito mais

Você sabia que a Packt oferece versões eBook de todos os livros publicados, com PDF e ePub files disponíveis? Você pode atualizar para a versão eBook em www.PacktPub.com e como cliente do livro impresso, você tem direito a um desconto na cópia do eBook. Entre em contato conosco em customercare@packtpub.com para mais detalhes.

Em www.PacktPub.com, você também pode ler uma coleção de artigos técnicos gratuitos, inscrever-se em uma variedade de boletins informativos gratuitos e receber descontos e ofertas exclusivas em livros e eBooks Packt.

Questões

Se você tiver algum problema com qualquer aspecto deste livro, entre em contato conosco pelo e-mail question@packtpub.com e faremos o possível para resolver o problema.

Machine Translated by Google

1

Introdução ao Nginx

O Nginx emergiu como um servidor web robusto e escalável de uso geral na última década. É uma escolha de muitos webmasters, fundadores de startups e engenheiros de confiabilidade do site por causa de sua arquitetura simples, mas escalável e expansível, configuração fácil e espaço de memória leve. O Nginx oferece muitos recursos úteis, como compactação imediata e armazenamento em cache pronto para uso.

O Nginx integra-se a tecnologias da web existentes, como servidor web Apache e PHP, e ajuda a resolver problemas do dia-a-dia de maneira fácil. A Nginx é apoiada por uma comunidade grande e ativa, bem como por uma empresa de consultoria financiada por capital de risco. Portanto, éativamente suportado.

Este livro irá ajudá-lo a começar com o Nginx e aprender as habilidades necessárias para transformá-lo em uma ferramenta poderosa, um cavalo de batalha que o ajudará a resolver seus desafios do dia-a-dia.

Instalando o Nginx

Antes de mergulhar nos recursos específicos do Nginx, você precisa aprender como instalar o Nginx em seu sistema.

É altamente recomendável que você use pacotes binários pré-construídos do Nginx se eles estiverem disponíveis em sua distribuição. Isso garante a melhor integração do Nginx com seu sistema e a reutilização das melhores práticas incorporadas ao pacote pelo mantenedor do pacote. Os pacotes binários pré-construídos do Nginx mantêm automaticamente as dependências para você e os mantenedores de pacotes geralmente são rápidos em incluir patches de segurança, para que você não receba reclamações dos oficiais de segurança. Além disso, o pacote geralmente fornece um script de inicialização específico da distribuição, que não vem de fábrica.

Consulte o diretório do seu pacote de distribuição para saber se você tem um pacote pré-compilado para o Nginx. Pacotes pré-construídos do Nginx também podem ser encontrados no link de download no site oficial do Nginx.org local.

Introdução ao Nginx

Neste capítulo, veremos rapidamente as distribuições mais comuns que contêm pacotes pré-construídos para o Nginx.

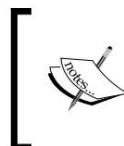
Instalando o Nginx no Ubuntu

A distribuição Ubuntu Linux contém um pacote pré-compilado para Nginx. Para instalá-lo, basta executar o seguinte comando:

```
$ sudo apt-get install nginx
```

O comando anterior instalará todos os arquivos necessários em seu sistema, incluindo o script logrotate e scripts de execução automática de serviço. A tabela a seguir descreve o layout de instalação do Nginx que será criado após a execução deste comando, bem como a finalidade dos arquivos e pastas selecionados:

Descrição	Caminho/Pasta
Arquivos de configuração do Nginx	/etc/nginx
Arquivo de configuração principal	/etc/nginx/nginx.conf
Arquivos de configuração de hosts virtuais (incluindo um padrão)	/etc/nginx/sites habilitado
Arquivos de configuração personalizados	/etc/nginx/conf.d
Arquivos de log (tanto de acesso quanto de log de erros)	/var/log/nginx
Arquivos temporários	/var/lib/nginx
Arquivos de host virtual padrão	/usr/share/nginx/html



Os arquivos de host virtual padrão serão colocados em /usr/share/nginx/html. Lembre-se de que este diretório é apenas para o host virtual padrão. Para implantar seu aplicativo da Web, use as pastas recomendadas pelo Filesystem Hierarchy Standard (FHS).



Agora você pode iniciar o serviço Nginx com o seguinte comando:

```
$ sudo service nginx start
```

Isso iniciará o Nginx em seu sistema.

Alternativas

O pacote Nginx pré-construído no Ubuntu tem várias alternativas. Cada um deles permite que você ajuste a instalação do Nginx para o seu sistema.

Instalando o Nginx no Red Hat Enterprise Linux ou CentOS/Scientific Linux

O Nginx não é fornecido de fábrica no Red Hat Enterprise Linux ou CentOS/Scientific Linux. Em vez disso, usaremos o repositório Extra Packages for Enterprise Linux (EPEL). EPEL é um repositório mantido pelos mantenedores do Red Hat Enterprise Linux, mas contém pacotes que não fazem parte da distribuição principal por vários motivos. Você pode ler mais sobre o EPEL em <https://fedoraproject.org/wiki/EPEL>.

Para habilitar o EPEL, você precisa baixar e instalar o pacote de configuração do repositório:

- Para RHEL ou CentOS/SL 7, use o seguinte link:
http://download.fedoraproject.org/pub/epel/7/x86_64/repoview/epel-release.html
- Para RHEL/CentOS/SL 6, use o seguinte link:
<http://download.fedoraproject.org/pub/epel/6/i386/repoview/epel-release.html>
- Se você tiver uma versão mais recente/antiga do RHEL, dê uma olhada em *Como posso usar esses pacotes extras?* seção no wiki original da EPEL no seguinte link:
<https://fedoraproject.org/wiki/EPEL>

Agora que você está pronto para instalar o Nginx, use o seguinte comando:

```
#yum install nginx
```

O comando anterior instalará todos os arquivos necessários em seu sistema, incluindo o script logrotate e scripts de execução automática de serviço. A tabela a seguir descreve o layout de instalação do Nginx que será criado após a execução deste comando e a finalidade dos arquivos e pastas selecionados:

Descrição	Caminho/Pasta
Arquivos de configuração	/etc/nginx
do Nginx Arquivo de configuração principal	/etc/nginx/nginx.conf
Arquivos de configuração personalizados	/etc/nginx/conf.d
Arquivos de log (tanto de acesso quanto de log de erros)	/var/log/nginx
Arquivos temporários	/var/lib/nginx
Arquivos de host virtual padrão	/usr/share/nginx/html

Introdução ao Nginx

 Os arquivos de host virtual padrão serão colocados em /usr/share/nginx/html. Lembre-se de que este diretório é apenas para o host virtual padrão. Para implantar seu aplicativo da Web, use as pastas recomendadas pelo FHS.

Por padrão, o serviço Nginx não iniciará automaticamente na inicialização do sistema, então vamos habilitá-lo. Consulte a tabela a seguir para os comandos correspondentes à sua versão do CentOS:

Função	Cent OS 6	Cent OS 7
Habilite a inicialização do Nginx na inicialização do sistema	<code>chkconfig nginx ativado</code>	<code>systemctl habilitar nginx</code>
Iniciar manualmente o serviço Nginx	<code>nginx start</code>	<code>systemctl iniciar nginx</code>
Parar manualmente o serviço Nginx	<code>nginx stop</code>	<code>systemctl parar nginx</code>

Instalando o Nginx a partir de arquivos de origem

Tradicionalmente, o Nginx é distribuído no código-fonte. Para instalar o Nginx a partir do código-fonte, você precisa baixar e compilar os arquivos-fonte em seu sistema.

 Não é recomendado que você instale o Nginx a partir do código-fonte. Faça isso apenas se você tiver um bom motivo, como os seguintes cenários:

- Você é um desenvolvedor de software e deseja depurar ou estender o Nginx
- Você se sente confiante o suficiente para manter seu próprio pacote
- Um pacote de sua distribuição não é bom o suficiente para você
- Você deseja ajustar seu binário Nginx

Em ambos os casos, se você planeja usar essa forma de instalação para uso real, esteja preparado para resolver desafios como manutenção de dependência, distribuição e aplicação de patches de segurança.

Nesta seção, estaremos nos referindo ao script de configuração. O script de configuração é um script de shell semelhante ao gerado pelo autoconf, que é necessário para configurar corretamente o código-fonte do Nginx antes que ele possa ser compilado. Este script de configuração não tem nada a ver com o arquivo de configuração Nginx que discutiremos mais tarde.

Baixando os arquivos de origem do Nginx

A fonte primária do Nginx para um público que fala inglês é o Nginx.org. Abra <http://nginx.org/en/download.html> em seu navegador e escolha a versão estável mais recente do Nginx. Baixe o arquivo escolhido em um diretório de sua escolha (/usr/local ou /usr/src são diretórios comuns a serem usados para compilar software):

```
$ wget -q http://nginx.org/download/nginx-1.7.9.tar.gz
```

Extraia os arquivos do arquivo baixado e mude para o diretório correspondente à versão escolhida do Nginx:

```
$ tar xf nginx-1.7.9.tar.gz  
$ cd nginx-1.7.9
```

Para configurar o código-fonte, precisamos executar o script ./configure incluído no arquivo:

```
$ ./configure  
verificando o SO  
+ Linux 3.13.0-36-genérico i686  
verificando o compilador C ... encontrado  
+ usando o compilador GNU C  
[...]
```

Este script produzirá muita saída e, se for bem-sucedido, gerará um Makefile arquivo para os arquivos de origem.

Observe que mostramos o prompt de usuário não privilegiado \$ em vez do root # nas linhas de comando anteriores. Você é encorajado a configurar e compilar o software como um usuário regular e instalar apenas como root. Isso evitaria muitos problemas relacionados à restrição de acesso ao trabalhar com o código-fonte.

Solução de problemas

A etapa de solução de problemas, embora muito simples, tem algumas armadilhas comuns. A instalação básica do Nginx requer a presença de pacotes de desenvolvedor Regex (PCRE) compatíveis com OpenSSL e Perl para compilar.

Se esses pacotes não estiverem instalados corretamente ou não estiverem instalados em locais onde o script de configuração do Nginx possa localizá-los, a etapa de configuração poderá falhar.

Em seguida, você deve escolher entre desabilitar os módulos internos do Nginx afetados (reescrever ou SSL, instalar os pacotes necessários corretamente ou apontar o script de configuração do Nginx para o local real desses pacotes, se estiverem instalados).

Introdução ao Nginx

Construindo Nginx

Você pode construir os arquivos de origem agora usando o seguinte comando:

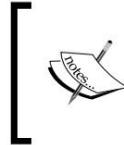
```
$ fazer
```

Você verá muita saída na compilação. Se a compilação for bem-sucedida, você poderá instalar o arquivo Nginx em seu sistema. Antes de fazer isso, certifique-se de escalarizar seus privilégios para o superusuário para que o script de instalação possa instalar os arquivos necessários nas áreas do sistema e atribuir os privilégios necessários. Uma vez bem sucedido, execute o comando make install :

```
#faça a instalação
```

O comando anterior instalará todos os arquivos necessários em seu sistema. A tabela a seguir lista todos os locais dos arquivos Nginx que serão criados após a execução deste comando e seus propósitos:

Descrição	Caminho/Pasta
Arquivos de configuração do Nginx	/usr/local/nginx/conf
Arquivo de configuração principal	/usr/local/nginx/conf/nginx.conf
Arquivos de log (tanto de acesso quanto de log de erros)	/usr/local/nginx/logs
Arquivos temporários	/usr/local/nginx
Arquivos de host virtual padrão	/usr/local/nginx/html



Ao contrário das instalações de pacotes pré-construídos, a instalação de arquivos de origem não aproveita as pastas Nginx para os arquivos de configuração personalizados ou arquivos de configuração de host virtual. O arquivo de configuração principal também é muito simples em sua natureza. Você tem que cuidar disso sozinho.

O Nginx deve estar pronto para uso agora. Para iniciar o Nginx, altere seu diretório de trabalho para o diretório /usr/local/nginx e execute o seguinte comando:

```
#sbin/nginx
```

Isso iniciará o Nginx em seu sistema com a configuração padrão.

Solução de problemas

Este estágio funciona ilegalmente na maioria das vezes. Um problema pode ocorrer nas seguintes situações:

Capítulo 1

- Você está usando uma configuração de sistema fora do padrão. Tente brincar com as opções do script de configuração para superar o problema.
- Você compilou em módulos de terceiros e eles estão desatualizados ou não mantido.

Desative os módulos de terceiros que interrompem sua compilação ou entre em contato com o desenvolvedor para obter assistência.

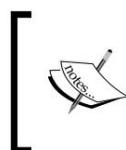
Copiando a configuração do código-fonte de pacotes pré-compilados

Ocasionalmente, você pode querer alterar o binário Nginx de um pacote pré-construído com suas próprias alterações. Para fazer isso, você precisa reproduzir a árvore de compilação que foi usada para compilar o binário Nginx para o pacote pré-compilado.

Mas como você saberia qual versão do Nginx e quais opções de script de configuração foram usadas no momento da compilação? Felizmente, o Nginx tem uma solução para isso. Basta executar o binário Nginx existente com a opção de linha de comando -V . O Nginx imprimirá as opções de tempo de configuração. Isso é mostrado a seguir:

```
$ /usr/sbin/nginx -V
Versão do nginx: nginx/1.4.6 (Ubuntu)
construído pelo gcc 4.8.2 (Ubuntu 4.8.2-19ubuntu1)
Suporte TLS SNI ativado
configurar argumentos: --with-cc-opt='-g -O2 -fstack-protector --param=ssp-buffer-size=4 -Wformat
-Werror=format-security -D_FORTIFY_SOURCE=2' --with-ld-opt ='-WI,-Bfunções-simbólicas -WI,-z,relro' ...
```

Usando a saída do comando anterior, reproduza todo o ambiente de compilação, incluindo a árvore de origem Nginx da versão correspondente e os módulos que foram incluídos na compilação.



Aqui, a saída do comando Nginx -V é cortada para simplificar.
Na realidade, você poderá ver e copiar toda a linha de comando que foi passada para o script de configuração no momento da compilação.

Você pode até querer reproduzir a versão do compilador usada para produzir um arquivo executável Nginx binário idêntico (discutiremos isso mais tarde quando discutirmos como solucionar problemas de travamento).

Introdução ao Nginx

Feito isso, execute o script `./configure` da sua árvore de origem do Nginx com as opções da saída da opção `-V` (com as alterações necessárias) e siga as etapas restantes do procedimento de compilação. Você obterá um executável Nginx alterado na pasta `objs/` da árvore de origem.

A estrutura da instalação do Nginx

Quando o Nginx está instalado, podemos estudar rapidamente a estrutura da instalação. Isso ajudará você a conhecer melhor sua instalação e gerenciá-la com mais confiança.

Para cada método de instalação, temos um conjunto de locais genéricos e caminhos padrão. Vamos ver o que esses locais padrão contêm.

A pasta de configuração do Nginx

Esta pasta contém o arquivo de configuração principal e um conjunto de arquivos de parâmetros. A tabela a seguir descreve a finalidade de cada um dos arquivos de parâmetro padrão:

Nome do arquivo	Descrição
<code>mime.types</code>	Ele contém o mapa de tipo MIME padrão para converter extensões de arquivo em tipos MIME.
<code>fastcgi_params</code>	Ele contém os parâmetros padrão FastCGI necessários para que o FastCGI funcione.
<code>scgi_params</code>	Ele contém os parâmetros padrão do SCGI necessários para o funcionamento do SCGI.
<code>uwsgi_params</code>	Ele contém os parâmetros UWCGI padrão necessários para que o UWCGI funcione.
<code>proxy_params</code>	Ele contém os parâmetros do módulo proxy padrão. Esse conjunto de parâmetros é necessário para determinados servidores da Web quando eles estão atrás do Nginx, para que eles possam descobrir que estão atrás de um proxy.
<code>naxsi.rules</code> (opcional)	Este é o principal conjunto de regras para o módulo de firewall do aplicativo da web NAXSI.
<code>koi-utf</code> , <code>koi-win</code> e <code>win-utf</code>	Estas são as tabelas de conversão do conjunto de caracteres cirílicos.

A pasta de host virtual padrão

A configuração padrão contém referências a este site como root. Não é recomendado que você use esse diretório para sites reais, pois não é uma boa prática que a hierarquia de pastas do Nginx contenha a hierarquia do site. Use este diretório para fins de teste ou para servir arquivos auxiliares.

A pasta de configuração de hosts virtuais

Este é o local dos arquivos de configuração do host virtual. A estrutura recomendada desta pasta é ter um arquivo por host virtual nesta pasta ou uma pasta por host virtual, contendo todos os arquivos relacionados a este host virtual. Desta forma, você sempre saberá quais arquivos foram usados e quais estão sendo usados agora, e o que cada um dos arquivos contém e quais arquivos podem ser removidos.

A pasta de registro

Este é o local dos arquivos de log do Nginx. O arquivo de log de acesso padrão e o arquivo de log de erro serão gravados neste local. Para instalação a partir de arquivos de origem, não é recomendável usar o local padrão /usr/local/nginx/logs para sites reais. Em vez disso, certifique-se de que todos os seus arquivos de log estejam armazenados no local do arquivo de log do sistema, como /var/log/nginx, para fornecer uma melhor visão geral e gerenciamento de seus arquivos de log.

A pasta temporária

O Nginx usa arquivos temporários para receber grandes corpos de solicitação e faz proxy de arquivos grandes do upstream. Os arquivos criados para esta finalidade podem ser encontrados nesta pasta.

Configurando o Nginx

Agora que você já sabe como instalar o Nginx e a estrutura de sua instalação, podemos estudar como configurar o Nginx. A simplicidade de configuração é uma das razões pelas quais o Nginx é popular entre os webmasters, porque isso economiza muito tempo.

Em poucas palavras, os arquivos de configuração do Nginx são simplesmente sequências de diretivas que podem receber até oito argumentos separados por espaço, por exemplo:

```
gzip_types texto/texto simples/aplicativo css/texto x-javascript/aplicativo xml/aplicativo xml/
xml+rss texto/javascript;
```

No arquivo de configuração, as diretivas são delimitadas por ponto e vírgula (;) umas das outras. Algumas das diretivas podem ter um bloco em vez de um ponto e vírgula. Um bloco é delimitado por chaves ({}). Um bloco pode conter dados de texto arbitrários, por exemplo:

```
tipos {
    texto/texto                                html htm html;
    html/texto                                 css;
    css/imagem                                xml;
    xml/gif                                    gif;
```

Introdução ao Nginx

```

    imagem/                jpeg jpg;
    aplicativo jpeg/aplicativo x-      js;
    javascript/atom+xml application/  átomo;
    rss+xml                  rss;
}

```

Um bloco também pode conter uma lista de outras diretivas. Nesse caso, o bloco é chamado de seção. Uma seção pode incluir outras seções, estabelecendo assim uma hierarquia de seções.

As diretivas mais importantes têm nomes curtos; isso reduz o esforço necessário para manter o arquivo de configuração.

Tipos de valor

Em geral, uma diretiva pode ter strings arbitrárias entre aspas ou não como argumentos.

Mas muitas diretivas têm argumentos que possuem tipos de valores comuns. Para ajudá-lo a entender rapidamente os tipos de valor, listei-os na tabela a seguir:

Tipo de valor	Formato	Exemplo de um valor
Bandeira	[ligado]	Ligado desligado
Número inteiro assinado	desligado]	1024
Tamanho	-?[0-9]+ [0-9]+([mM])	23M, 12348k
Desvio	[kK])? [0-9]+([mM][kK][gG])?	43G, 256M
Milissegundos	[0-9]+[yMwdhms]?	30s, 60m

Variáveis

Variáveis são objetos nomeados que podem receber um valor textual. As variáveis só podem aparecer dentro da seção http . Uma variável é referida pelo seu nome, precedido pelo símbolo do dólar (\$) . Como alternativa, uma referência de variável pode incluir um nome de variável entre colchetes para evitar a fusão com o texto ao redor.

As variáveis podem ser usadas em qualquer diretiva que as aceite, conforme mostrado aqui:

```
proxy_set_header Host $http_host;
```

Essa diretiva define o host do cabeçalho HTTP em uma solicitação encaminhada para o nome do host HTTP da solicitação original. Isso é equivalente ao seguinte:

```
proxy_set_header Host ${http_host};
```

Com a seguinte sintaxe, você pode especificar o nome do host:

```
proxy_set_header Host ${http_host}_squirrel;
```

Capítulo 1

O comando anterior anexará uma string `_squirrel` ao valor do nome do host original. Sem chaves, a string `_squirrel` teria sido interpretada como parte do nome da variável, e a referência teria apontado para uma variável "http_host_squirrel" em vez de `http_host`.

Existem também nomes de variáveis especiais:

- Variáveis de \$1 a \$9 referem-se aos argumentos de captura nas expressões regulares, conforme mostrado aqui:

```
localização ~ /(.+)\.php$ {
    [...]
    proxy_set_header X-Script-Name $1;
}
```

A configuração anterior definirá o cabeçalho HTTP `X-Script-Name` na solicitação encaminhada para o nome do script PHP no URI da solicitação.

As capturas são especificadas em uma expressão regular usando colchetes.

- Variáveis que começam com `$arg_` referem-se ao argumento de consulta correspondente na solicitação HTTP original, conforme mostrado aqui:

```
proxy_set_header X-Versão-Nome $arg_ver;
```

A configuração anterior definirá o cabeçalho HTTP `X-Version-Name` na solicitação encaminhada para o valor do argumento de consulta `ver` na solicitação original.

- Variáveis que começam com `$http_` referem-se ao cabeçalho HTTP correspondente linha na solicitação original.
- Variáveis que começam com `$sent_http_` referem-se ao HTTP correspondente linha de cabeçalho na solicitação HTTP de saída.
- As variáveis que começam com `$upstream_http_` referem-se à linha de cabeçalho HTTP correspondente na resposta recebida de um upstream.
- Variáveis que começam com `$cookie_` referem -se ao cookie correspondente no pedido inicial.
- Variáveis que começam com `$upstream_cookie_` referem-se ao correspondente cookie na resposta recebida de um upstream.

As variáveis devem ser declaradas pelos módulos Nginx antes que possam ser usadas na configuração. Os módulos Nginx integrados fornecem um conjunto de variáveis principais que permitem operar com os dados de solicitações e respostas HTTP. Consulte a documentação do Nginx para obter a lista completa de variáveis principais e suas funções.

Módulos de terceiros podem fornecer variáveis extras. Essas variáveis devem ser descritas na documentação do módulo de terceiros.

Introdução ao Nginx

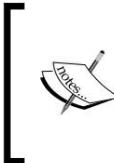
Inclusões

Qualquer seção de configuração do Nginx pode conter inclusões de outros arquivos via include diretiva. Esta diretiva recebe um único argumento contendo um caminho para um arquivo a ser incluído, conforme mostrado aqui:

```
/*
 * Uma inclusão relativa simples. O caminho do arquivo de destino
 * é relativo ao local do arquivo de configuração atual.
 */
incluir mime.types;

/*
 * Uma inclusão simples usando um caminho absoluto.
 */
incluir /etc/nginx/conf/site-defaults.conf;
```

Uma vez especificada, a diretiva include instrui o Nginx a processar o conteúdo do arquivo ou arquivos especificados pelo argumento desta diretiva como se eles fossem especificados no lugar da diretiva include .



Os caminhos relativos são resolvidos em relação ao caminho do arquivo de configuração em que a diretiva é especificada. É bom ter isso em mente quando a diretiva de inclusão é especificada em outro arquivo incluído, como quando um arquivo de configuração de host virtual contém uma diretiva de inclusão relativa .

A diretiva include também pode conter um caminho globbed com curingas, relativos ou absolutos. Nesse caso, o caminho globbed é expandido e todos os arquivos correspondentes ao padrão especificado são incluídos em nenhuma ordem específica. Dê uma olhada no código a seguir:

```
/*
 * Uma inclusão glob simples. Isso incluirá todos os arquivos
 * terminando em ".conf" localizado em /etc/nginx/sites-enabled
 */
incluir /etc/nginx/sites-enabled/*.conf;
```

A diretiva include com curingas é uma solução óbvia para incluir configurações de site, pois seu número pode variar muito. Usando a diretiva include , você pode estruturar adequadamente o arquivo de configuração ou reutilizar determinadas partes várias vezes.

Seções

Uma seção é uma diretiva que inclui outras diretivas em seu bloco. Os delimitadores de cada seção devem estar localizados no mesmo arquivo, enquanto o conteúdo de uma seção pode abranger vários arquivos por meio da diretiva include .

Não é possível descrever todas as diretivas de configuração possíveis neste capítulo. Consulte a documentação do Nginx para obter mais informações. No entanto, examinarei rapidamente os tipos de seção de configuração do Nginx para que você possa orientar na estrutura dos arquivos de configuração do Nginx.

A seção http

A seção http habilita e configura o serviço HTTP no Nginx. Ele tem o servidor e as declarações upstream. No que diz respeito às diretivas individuais, o http seção geralmente contém aqueles que especificam padrões para todo o serviço HTTP.

A seção http deve conter pelo menos uma seção de servidor para processar solicitações HTTP. Aqui está um layout típico da seção http :

```
http{  
    [...]  
    servidor {  
        [...]  
    }  
}
```

Aqui e em outros exemplos deste livro, usamos [...] para nos referirmos a partes irrelevantes omitidas da configuração.

A seção do servidor

A seção do servidor configura um host virtual HTTP ou HTTPS e especifica endereços de escuta para eles usando a diretiva listen . No final do estágio de configuração, todos os endereços de escuta são agrupados e todos os endereços de escuta são ativados na inicialização.

A seção do servidor contém as seções de localização , bem como as seções que podem ser incluídas na seção de localização (consulte a descrição de outros tipos de seções para obter detalhes). As diretivas especificadas na própria seção do servidor vão para o chamado local padrão. A esse respeito, a seção do servidor serve ao propósito da localização própria seção.

Introdução ao Nginx

Quando uma solicitação chega por meio de um dos endereços de escuta, ela é roteada para as seções do servidor que correspondem a um padrão de host virtual especificado pelo `server_name` diretiva. A solicitação é então roteada para o local que corresponde ao caminho do URI de solicitação ou processada pelo local padrão se não houver correspondência.

A seção a montante

A seção `upstream` configura um servidor lógico para o qual o Nginx pode passar solicitações para processamento adicional. Este servidor lógico pode ser configurado para ser apoiado por um ou mais servidores físicos externos ao Nginx com nomes de domínio concretos ou endereços IP.

O `upstream` pode ser referido pelo nome de qualquer lugar no arquivo de configuração onde possa ocorrer uma referência a um servidor físico. Desta forma, sua configuração pode ser feita independente da estrutura subjacente do `upstream`, enquanto a estrutura `upstream` pode ser alterada sem alterar sua configuração.

A seção de localização

A seção de localização é um dos cavalos de batalha no Nginx. A diretiva de localização usa parâmetros que especificam um padrão que corresponde ao caminho do URI de solicitação. Quando uma solicitação é roteada para um local, o Nginx ativa a configuração incluída por essa seção de local .

Existem três tipos de padrões de localização: padrões de localização de expressão simples, exatos e regulares.

Simples

Um local simples tem uma string como o primeiro argumento. Quando essa string corresponde à parte inicial do URI de solicitação, a solicitação é roteada para esse local. Aqui está um exemplo de um local simples:

```
local/imagens {  
    root /usr/local/html/imagens;  
}
```

Qualquer solicitação com um URI que comece com `/images`, como `/images/powerlogo.png`, `/images/calendar.png` ou `/images/social/github-icon.png` será roteada para este local. Um URI com um caminho igual a `/images` também será roteado para esse local.

Exato

Locais exatos são designados com um caractere igual (=) como o primeiro argumento e têm uma string como segundo argumento, assim como os locais simples. Essencialmente, locais exatos funcionam como locais simples, exceto que o caminho no URI de solicitação deve corresponder exatamente ao segundo argumento da diretiva de localização para ser roteado para esse local:

```
local = /imagens/vazio.gif {
    vaziogif;
}
```

A configuração anterior retornará um arquivo GIF vazio se e somente se o URI /images/empty.gif for solicitado.

Locais de expressão regular

Os locais de expressão regular são designados com um caractere til (~) ou ~* (para correspondências que não diferenciam maiúsculas de minúsculas) como o primeiro argumento e têm uma expressão regular como o segundo argumento. As localizações das expressões regulares são processadas após as localizações simples e exatas. O caminho no URI de solicitação deve corresponder à expressão regular no segundo argumento da diretiva de localização para ser roteado para esse local. Um exemplo típico é o seguinte:

```
localização ~ \.php$ {
    [...]
}
```

De acordo com a configuração anterior, solicitações com URIs que terminam com .php será encaminhado para este local.

As seções de localização podem ser aninhadas. Para isso, você só precisa especificar um local seção dentro de outra seção de localização .

A seção if

A seção if inclui uma configuração que se torna ativa quando uma condição especificada pela diretiva if é satisfeita. A seção if pode ser delimitada pelas seções servidor e local e só estará disponível se o módulo de reescrita estiver presente.

Introdução ao Nginx

Uma condição de uma diretiva if é especificada entre colchetes e pode assumir as seguintes formas:

- Uma variável simples, conforme mostrado aqui:

```
if ($arquivo_presente) {
    taxa_limite 256k;
}
```

Se a variável for avaliada como valor verdadeiro em tempo de execução, a configuração seção ativa.

- Uma expressão unária que consiste em um operador e uma string com variáveis, conforme mostrado aqui:

```
if ( -d "${path}" ) try_files {
    "${path}/default.png" "${path}/default.jpg";
}
```

Os seguintes operadores unários são suportados:

Descrição do Operador	Descrição do Operador	Descrição do Operador	Descrição do Operador
-f	Verdadeiro se o arquivo especificado existir	!-f	Verdadeiro se o arquivo especificado não existir
-d	Verdadeiro se o diretório especificado existir	!-d	Verdadeiro se o diretório especificado não existir
-e	Verdadeiro se o arquivo especificado existir e for um link simbólico	!-e	Verdadeiro se o arquivo especificado não existir ou não for um link simbólico
-x	Verdadeiro se o arquivo especificado existir e for executável	!-x	Verdadeiro se o arquivo especificado não existir ou não for executável

- Uma expressão binária que consiste em um nome de variável, um operador e uma string com variáveis. Os seguintes operadores binários são suportados:

Descrição do Operador	Descrição do Operador	Descrição do Operador	Descrição do Operador
=	Verdadeiro se uma variável corresponder a uma string	!=	True se uma variável não corresponder a uma string
~	Verdadeiro se uma expressão regular corresponder ao valor de uma variável	!~	Verdadeiro se uma expressão regular não corresponder ao valor de uma variável Verdadeiro se uma expressão
~*	True se uma expressão regular que não diferencia maiúsculas de minúsculas corresponde ao valor de uma variável	!~*	regular que não diferencia maiúsculas de minúsculas não corresponder ao valor de uma variável

Vamos discutir alguns exemplos da diretiva if .

Capítulo 1

Este adiciona um preix /msie/ à URL de qualquer solicitação que contenha MSIE no campo user-agent:

```
if ($http_user_agent ~ MSIE) {
    reescrever ^(.*)$ /msie/$1 break;
}
```

O próximo exemplo define a variável \$id com o valor do cookie chamado id, se estiver presente:

```
if ($http_cookie ~* "id=([^;]+)(?:;|$)") {
    defina $id $1;
}
```

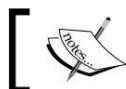
O próximo retorna o status HTTP 405 ("Método não permitido") para cada solicitação com o método POST:

```
if ($request_method = POST) {
    retornar 405;
}
```

Por fim, a configuração no exemplo a seguir limita a taxa a 10 KB sempre que a variável \$slow for avaliada como verdadeira:

```
if ($lento) {
    taxa_limite 10k;
}
```

A diretiva if parece um instrumento poderoso, mas deve ser usada com cautela. Isso porque a configuração dentro da seção if não é imperativa, ou seja, não altera o baixo processamento da requisição conforme a ordem das diretivas if .



Devido ao comportamento não intuitivo da diretiva if, seu uso é desencorajado.

As condições não são avaliadas na ordem em que são especificadas no arquivo de configuração. Eles são meramente aplicados simultaneamente e as configurações de configuração das seções para as quais as condições foram satisfeitas são mescladas e aplicadas de uma só vez.

Introdução ao Nginx

A seção limit_except

A seção **limit_except** ativa a configuração que inclui se o método de solicitação **não corresponder a nenhum da lista de métodos especificados por esta diretiva**. Especificar o método **GET** na lista de métodos assume automaticamente o **HEAD** método. Esta seção só pode aparecer dentro da seção de **localização**, conforme mostrado aqui:

```
limit_except GET {  
    retornar 405;  
}
```

A configuração anterior responderá com o status HTTP 405 ("Método não permitido") para cada solicitação que não for feita usando o método **GET ou HEAD**.

Outros tipos de seção

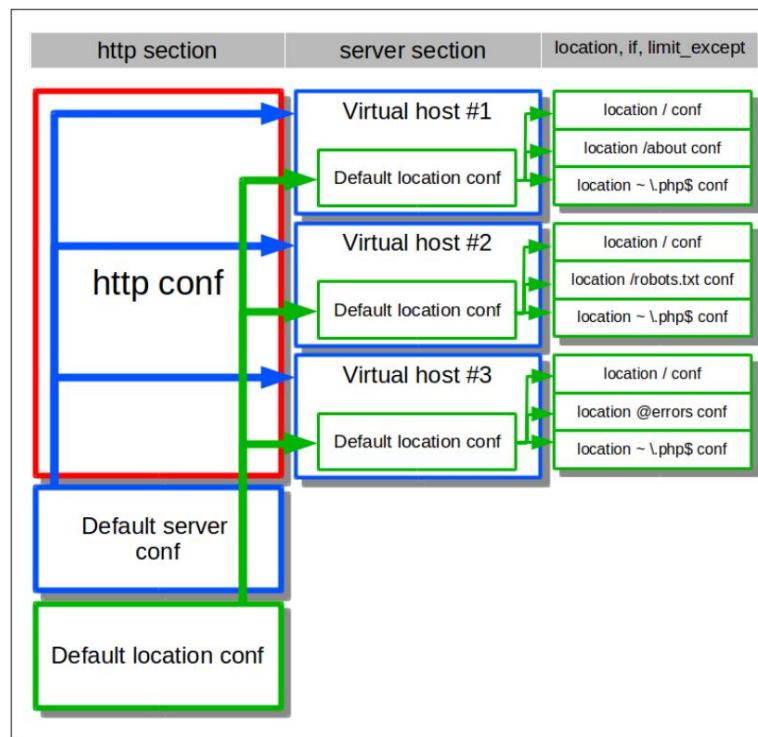
A configuração do Nginx pode conter outros tipos de seção, como **main** e **server** na seção principal, bem como tipos de seção fornecidos por módulos de terceiros. Neste livro, não daremos muita atenção a eles.

Consulte a documentação dos módulos correspondentes para obter informações sobre esses tipos de seções de configuração.

Regras de herança das configurações de configuração

Muitas configurações de configuração do Nginx podem ser herdadas de uma seção de nível externo para uma seção de nível interno. Isso economiza muito tempo ao configurar o Nginx.

A figura a seguir ilustra como as regras de herança funcionam:



Todas as configurações podem ser atribuídas a três categorias:

- Aqueles que fazem sentido apenas em todo o serviço HTTP (marcados em vermelho)
- Aqueles que fazem sentido na configuração do host virtual (marcados em azul)
- Aqueles que fazem sentido em todos os níveis de configuração (marcados em verde)

As configurações da primeira categoria não possuem regras de herança, pois não podem herdar valores de nenhum lugar. Eles podem ser especificados apenas na seção http e podem ser aplicados a todo o serviço HTTP. Essas são configurações definidas por diretivas, como variables_hash_max_size, variables_hash_bucket_size, server_hash_max_size e server_names_hash_bucket_size.

As configurações da segunda categoria podem herdar valores apenas da seção http . Eles podem ser especificados nas seções http e servidor , mas as configurações aplicadas a um determinado host virtual são determinadas por regras de herança. Essas são configurações definidas por diretivas, como client_header_timeout, client_header_buffer_size e large_client_header_buffers.

Introdução ao Nginx

Finalmente, as configurações da terceira categoria podem herdar valores de qualquer seção até http. Eles podem ser especificados em qualquer seção dentro da configuração do serviço HTTP, e as configurações aplicadas a um determinado contexto são determinadas por regras de herança.

As setas na figura ilustram os caminhos de propagação de valor. As cores das setas especificam o escopo da configuração. As regras de propagação ao longo de um caminho são as seguintes:

Quando você especifica um valor para um parâmetro em um determinado nível da configuração, ele substitui o valor do mesmo parâmetro nos níveis externos se estiver definido e se propaga automaticamente para os níveis internos da configuração. Vamos dar uma olhada no seguinte exemplo:

```
local / {
    # A seção externa
    root /var/www/example.com;
    gzip ativado;

    local ~ \.js$ {
        # Seção interna 1
        gzip off;

    }
    localização ~ \.css$ {
        # Seção interna 2
    }
    [...]
}
```

O valor da diretiva root será propagado para as seções internas, portanto, não há necessidade de especificá-lo novamente. O valor da diretiva gzip na seção externa será propagado para as seções internas, mas será substituído pelo valor do gzip diretiva dentro da primeira seção interna. O efeito geral disso será que o gzip a compressão será habilitada em todos os lugares da outra seção, exceto na primeira seção interna.

Quando um valor para algum parâmetro não é especificado em uma determinada seção de configuração, ele é herdado de uma seção que inclui a seção de configuração atual. Se a seção envolvente não tiver esse parâmetro definido, a pesquisa vai para o nível externo e assim sucessivamente. Se um valor para um determinado parâmetro não for especificado, um valor padrão interno será usado.

A primeira configuração de amostra

A essa altura do capítulo, você pode ter acumulado muito conhecimento sem ter uma ideia de como é uma configuração de trabalho completa. Estudaremos uma configuração curta, mas funcional, que lhe dará uma ideia de como pode ser um arquivo de configuração completo:

```
error_log logs/error.log;

eventos {
    use epoll;
    trabalhador_conexões 1024;
}

http{
    incluir          mime.types;
    tipo_padrão     aplicação/fluxo de octetos;

    servidor {
        ouço          80;
        nome_do_servidor exemplo.org www.exemplo.org;

        local / {
            proxy_pass http://localhost:8080;
            inclua proxy_params;
        }

        local ~ ^/(images|js|css) {
            html raiz;
            expira 30d;
        }
    }
}
```

Esta configuração primeiro instrui o Nginx a gravar o log de erros em logs/error.log. Em seguida, ele configura o Nginx para usar o método de processamento de eventos epoll (use epoll) e aloca memória para 1024 conexões por trabalhador (worker_connections 1024). Depois disso, ele habilita o serviço HTTP e configura certas configurações padrão para o serviço HTTP (incluindo mime.types, default_type application/octet-stream). Ele cria um host virtual e define seus nomes como example.org e www.example.org (nome_do_servidor exemplo.org www.exemplo.org). O host virtual é disponibilizado no endereço de escuta padrão 0.0.0.0 e na porta 80 (ouvir 80).

Introdução ao Nginx

Em seguida, configuraremos dois locais. O primeiro local passa todas as solicitações roteadas para ele em um servidor de aplicativos da Web em execução em `http://localhost:8080` (`proxy_pass http://localhost:8080`). O segundo local é um local de expressão regular. Ao especificá-lo, efetivamente excluímos um conjunto de caminhos do primeiro local. Usamos esse local para retornar dados estáticos, como imagens, arquivos JavaScript e arquivos CSS. Definimos o diretório base para nossos arquivos de mídia como `html` (`html` raiz). Para todos os arquivos de mídia, definimos a data de validade como 30 dias (expira em `30d`).

Para experimentar esta configuração, faça backup de seu arquivo de configuração padrão e substitua o conteúdo do arquivo de configuração padrão pela configuração anterior.

Em seguida, reinicie o Nginx para que as configurações tenham efeito. Feito isso, você pode navegar até a URL `http://localhost/` para verificar sua nova configuração.

Práticas recomendadas de configuração

Agora que você sabe mais sobre os elementos e a estrutura do arquivo de configuração do Nginx, pode estar curioso sobre quais práticas recomendadas existem nessa área. Aqui está uma lista de recomendações que ajudarão você a manter sua configuração de forma mais eficiente e torná-la mais robusta e gerenciável:

- Estruture bem sua configuração. Observe quais partes comuns do configuração são usados com mais frequência, mova-os para arquivos separados e reutilize-os usando a diretiva `include`. Além disso, tente fazer com que cada arquivo em sua hierarquia de arquivos de configuração tenha um tamanho razoável, idealmente não mais que duas telas. Isso ajudará você a ler seus arquivos mais rapidamente e navegar por eles com eficiência.



É importante saber exatamente como sua configuração funciona para gerenciá-la com sucesso. Se a configuração não funcionar da maneira esperada, você poderá ter problemas devido à aplicação de configurações incorretas, por exemplo, indisponibilidade de URIs arbitrários, interrupções inesperadas e brechas de segurança.

- Minimize o uso da diretiva `if`. A diretiva `if` tem um comportamento não intuitivo. Tente evitar usá-lo sempre que possível para garantir que as configurações de configuração sejam aplicadas às solicitações recebidas conforme o esperado.
- Use bons padrões. Experimente com regras de herança e tente criar padrões para suas configurações para que resultem no menor número de diretivas a serem configuradas. Isso inclui mover configurações comuns de localização para o nível do servidor e depois para o nível HTTP.

Resumo

Neste capítulo, você aprendeu como instalar o Nginx a partir de várias fontes disponíveis, a estrutura de instalação do Nginx e a finalidade de vários arquivos, os elementos e a estrutura do arquivo de configuração do Nginx e como criar um arquivo de configuração do Nginx de trabalho mínimo. Você também aprendeu sobre algumas práticas recomendadas para configuração do Nginx.

No próximo capítulo, você aprenderá como colocar o Nginx em operação e como gerenciá-lo em ação.

Machine Translated by Google

2

Gerenciando Nginx

Em um servidor web rodando em escala total, milhares de eventos estão ocorrendo a cada segundo. O microgerenciamento desses eventos obviamente não é possível, mas mesmo pequenas falhas podem causar séria deterioração na qualidade do serviço e afetar a experiência do usuário.

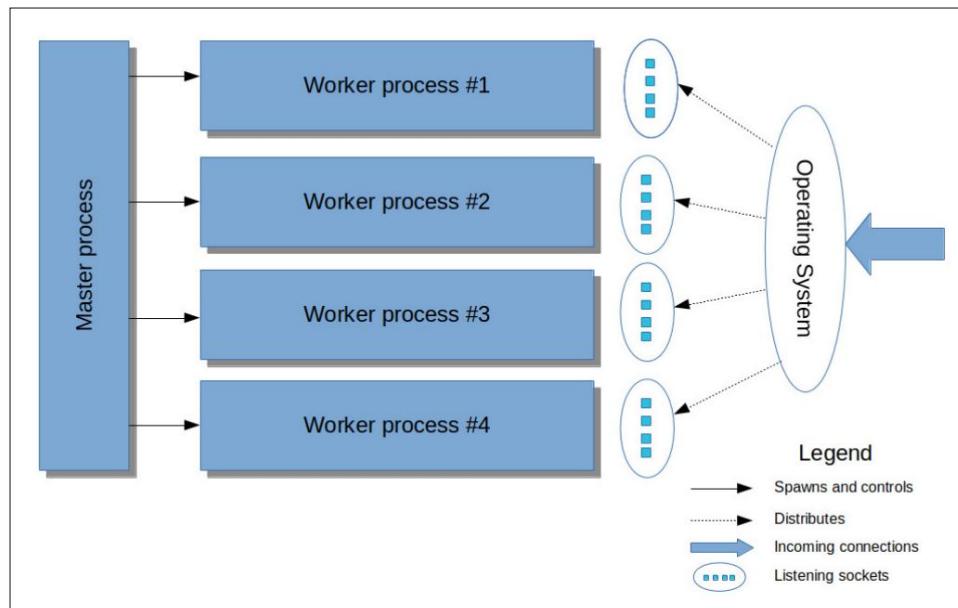
Para evitar que essas falhas aconteçam, um webmaster dedicado ou engenheiro de confiabilidade do site deve ser capaz de entender e gerenciar adequadamente os processos nos bastidores.

Neste capítulo, você aprenderá a gerenciar uma instância do Nginx em operação e discutiremos os seguintes tópicos:

- Iniciando e parando o Nginx
- Recarregar e reconfigurar processos
- Alocação de processos de trabalho
- Outras questões de gestão

A arquitetura de processamento de conexão Nginx

Antes de estudar os procedimentos de gerenciamento do Nginx, você precisa ter uma ideia de como o Nginx processa as conexões. No modo em escala real, uma única instância do Nginx consiste no processo mestre e nos processos de trabalho, conforme mostrado na figura a seguir:



O processo mestre gera processos de trabalho e os controla enviando e encaminhando sinais e ouvindo notificações de saída deles. Os processos de trabalho aguardam nos soquetes de escuta e aceitam as conexões de entrada. O sistema operacional distribui as conexões de entrada entre os processos de trabalho de forma round-robin.

O processo mestre é responsável por todas as tarefas de inicialização, desligamento e manutenção, como as seguintes:

- Lendo e relendo arquivos de configuração
- Abrindo e reabrindo arquivos de log
- Criando soquetes de escuta
- Iniciando e reiniciando processos de trabalho
- Encaminhamento de sinais para os processos de trabalho
- Iniciando um novo binário

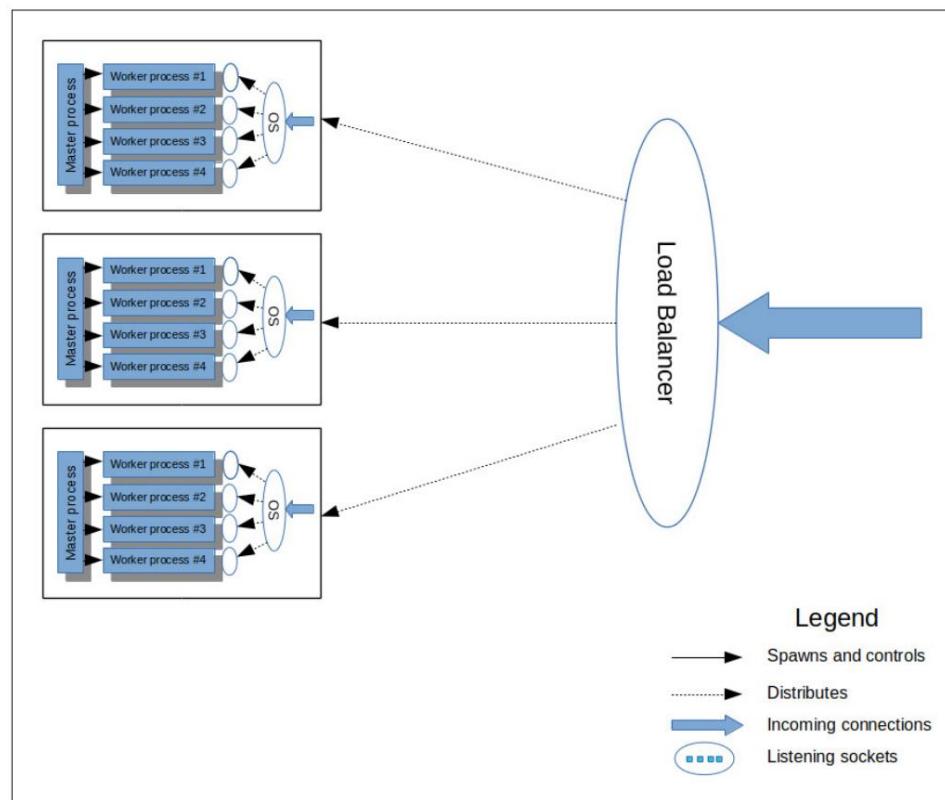
O processo mestre garante, assim, a operação contínua de uma instância Nginx diante de várias mudanças no ambiente e falhas ocasionais de processos de trabalho.

Os processos de trabalho são responsáveis por servir conexões e aceitar novas.

Os processos de trabalho também podem executar determinadas tarefas de manutenção.

Por exemplo, eles reabrem os arquivos de log por conta própria depois que o processo mestre garante que essa operação é segura. Cada processo de trabalho lida com várias conexões. Isso é conseguido executando um loop de eventos que extrai eventos que ocorreram em soquetes abertos do sistema operacional por meio de uma chamada de sistema especial e processando rapidamente todos os eventos puxados lendo e gravando em soquetes ativos. Os recursos necessários para manter uma conexão são alocados quando um processo de trabalho é iniciado. O número máximo de conexões que um processo de trabalho pode manipular simultaneamente é configurado pela diretiva `worker_connections` e o padrão é 512.

Em uma configuração em cluster, um dispositivo de roteamento especial, como um平衡ador de carga ou outra instância Nginx, é usado para balancear conexões de entrada entre um conjunto de instâncias Nginx idênticas, cada uma delas consistindo em um processo mestre e uma coleção de processos de trabalho. Isso é mostrado na figura a seguir:



[Gerenciando Nginx](#)

Nesta configuração, o balanceador de carga roteia conexões apenas para as instâncias que estão escutando conexões de entrada. O balanceador de carga garante que cada uma das instâncias ativas obtenha uma quantidade aproximadamente igual de tráfego e roteie o tráfego para fora de uma instância se ela apresentar algum problema de conectividade.

Devido à diferença na arquitetura, os procedimentos de gerenciamento para uma configuração em cluster são ligeiramente diferentes dos de uma instância autônoma. Discutiremos essas diferenças em breve.

Iniciando e parando o Nginx

No capítulo anterior, você aprendeu um pouco sobre como iniciar sua instância Nginx.

Em sistemas Ubuntu, Debian ou semelhantes a Redhat, você pode executar o seguinte comando:

```
# serviço nginx start
```

Na ausência de scripts de inicialização, você pode simplesmente executar o binário usando o seguinte comando:

```
#sbin/nginx
```

O Nginx lerá e analisará o arquivo de configuração, criará um arquivo PID (um arquivo contendo seu ID de processo), abrirá arquivos de log, criará soquetes de escuta e iniciará processos de trabalho. Depois que os processos de trabalho são iniciados, uma instância do Nginx é capaz de responder às conexões de entrada. É assim que uma instância do Nginx em execução se parece na lista de processos:

```
# ps -C nginx -f
UID          PID  PPID C STIME TTY          TIME CMD
raiz        2324      1 0 15:30 ?
usr/sbin/nginx
www-data  2325 2324 0 15:30 ?          0:00:00 nginx: processo mestre /
www-data  2326 2324 0 15:30 ?          0:00:00 nginx: processo de trabalho
www-data  2327 2324 0 15:30 ?          0:00:00 nginx: processo de trabalho
www-data  2328 2324 0 15:30 ?          0:00:00 nginx: processo de trabalho
```

Capítulo 2

Cada processo Nginx define seu título de processo de forma que ele reflita convenientemente a função do processo. Aqui, por exemplo, você vê o processo mestre da instância com ID de processo 2324 e quatro processos de trabalho com IDs de processo 2325, 2326, 2327 e 2328. Observe como a coluna de ID do processo pai (PPID) aponta para o processo mestre.

Faremos referência ao ID do processo mestre mais adiante nesta seção.

Se você não consegue encontrar sua instância na lista de processos ou vê uma mensagem de erro no console na inicialização, algo está impedindo o Nginx de iniciar. A tabela a seguir lista os possíveis problemas e suas soluções:

Mensagem	Questão	Resolução
[emerg] bind() to xxxx:x falhou (98: Endereço já em uso)	Ponto de extremidade de escuta conflitante	Certifique-se de que os endpoints especificados pelo listen diretiva não entre em conflito com outros serviços
[emerg] open() "<path to file>" falhou (2: Nenhum arquivo ou diretório)	Caminho inválido para um arquivo	Certifique-se de que todos os caminhos em sua configuração apontam para diretórios existentes
[emerg] open() "<path to file>" falhou (13: Permissão negada)	Privilégios insuficientes	Certifique-se de que todos os caminhos em sua configuração apontam para diretórios aos quais o Nginx tem acesso

Para parar o Nginx, você pode executar o seguinte comando se um script de inicialização estiver disponível:

```
# serviço nginx parar
```

Como alternativa, você pode enviar o sinal TERM ou INT para o processo mestre de sua instância para acionar um desligamento rápido ou o sinal QUIT para acionar um desligamento normal, conforme mostrado aqui:

```
# matar - SAIR 2324
```

O comando anterior acionará o procedimento de desligamento normal na instância e todos os processos serão encerrados. Aqui, nos referimos ao ID do processo do processo mestre da lista de processos anterior.

Gerenciando Nginx

Sinais de controle e seu uso

O Nginx, como qualquer outro serviço de segundo plano do Unix, é controlado por sinais.

Sinais são eventos assíncronos que interrompem a execução normal de um processo e ativam determinadas funções. A tabela a seguir lista todos os sinais que o Nginx suporta e as funções que eles acionam:

Sinal	Função
PRAZO, INT	Desligamento rápido
SAIR	Desligamento normal
HUP	Reconfiguração
USR1	Reabertura do arquivo de log
USR2	Atualização binária do Nginx
GUINCHO	Desligamento gracioso do trabalhador

Todos os sinais devem ser enviados ao processo mestre de uma instância. O processo mestre de uma instância pode ser localizado pesquisando-o na lista de processos:

```
# ps -C nginx -f
UID          PID  PPID C STIME TTY          TIME CMD
raiz        4754 3201 0 11:10 ?
usr/sbin/nginx
www-data    4755 4754 0 11:10 ?
www-data    4756 4754 0 11:10 ?
www-data    4757 4754 0 11:10 ?
www-data    4758 4754 0 11:10 ?
```

Nesta lista, o processo mestre tem uma identificação de processo 4754 e quatro processos de trabalho. O ID do processo do processo mestre também pode ser obtido examinando o conteúdo do arquivo PID:

```
# cat /var/run/nginx.pid
4754
```



Nota: O caminho de nginx.pid pode variar em sistemas diferentes. Você pode usar o comando /usr/sbin/nginx -V para descobrir o caminho exato.

Para enviar um sinal para uma instância, use o comando kill e especifique o ID do processo mestre como o último argumento:

```
#mata -HUP 4754
```

Capítulo 2

Alternativamente, você pode usar a substituição de comando para obter o ID do processo do processo mestre diretamente do arquivo PID:

```
# kill -HUP `cat /var/run/nginx.pid`
```

Você também pode usar o seguinte comando:

```
# kill - HUP $(cat /var/run/nginx.pid)
```

Os três comandos anteriores acionarão a reconfiguração da instância. Vamos agora discutir cada uma das funções que os sinais acionam no Nginx.

Desligamento rápido

Os sinais TERM e INT são enviados para o processo mestre de uma instância Nginx para acionar o procedimento de desligamento rápido. Todos os recursos como conexões, arquivos abertos e arquivos de log que cada processo de trabalho possui são imediatamente fechados.

Depois disso, cada processo de trabalho é encerrado e o processo mestre é notificado. Depois que todos os processos de trabalho são encerrados, o processo mestre é encerrado e o desligamento é concluído.

Um desligamento rápido obviamente causa uma interrupção visível do serviço. Portanto, ele deve ser usado em situações de emergência ou quando você tem certeza absoluta de que ninguém está usando sua instância.

Desligamento normal

Uma vez que o Nginx recebe o sinal QUIT , ele entra no modo de desligamento normal. O Nginx fecha os soquetes de escuta e não aceita novas conexões a partir de então. As conexões existentes ainda são atendidas até que não sejam mais necessárias. Portanto, o desligamento normal pode levar muito tempo para ser concluído, especialmente se algumas das conexões estiverem no meio de um longo download ou upload.

Depois de sinalizar o desligamento normal para o Nginx, você pode monitorar sua lista de processos para ver quais processos de trabalho do Nginx ainda estão em execução e acompanhar o andamento do procedimento de desligamento:

```
# ps -C nginx -f
UID          PID  PPID C STIME TTY          TIME CMD
raiz        5813 3201 0 12:07 ?  usr/sbin/          00:00:00 nginx: processo mestre /
nginx      www-data 5814 5813 11 12:07 ?
www-data    5814 5813 11 12:07 ?          00:00:01 nginx: o processo de trabalho é
desligando
```

Nesta lista, você pode ver uma instância depois que um desligamento normal foi acionado.

Um único processo de trabalho tem um rótulo está sendo encerrado e seu título de processo está marcando um processo que está sendo encerrado no momento.

Gerenciando Nginx

Depois que todas as conexões tratadas por um trabalhador são fechadas, o processo de trabalho é encerrado e o processo mestre é notificado. Depois que todos os processos de trabalho são encerrados, o processo mestre é encerrado e o desligamento é concluído.

Em uma configuração em cluster ou com平衡amento de carga, o desligamento normal é uma maneira típica de colocar uma instância fora de operação. O uso do desligamento normal garante que não haja interrupções visíveis de seu serviço devido à reconfiguração ou manutenção do servidor.

Em uma única instância, o desligamento normal só pode garantir que as conexões existentes não sejam fechadas abruptamente. Depois que o desligamento normal for acionado em uma única instância, o serviço ficará imediatamente indisponível para novos visitantes.

Para garantir a disponibilidade contínua em uma única instância, use procedimentos de manutenção como reconfiguração, reabertura do arquivo de log e atualização binária do Nginx.

Reconfiguração

O sinal HUP pode ser usado para sinalizar ao Nginx para reler os arquivos de configuração e reiniciar os processos de trabalho. Este procedimento não pode ser executado sem reiniciar os processos de trabalho, pois as estruturas de dados de configuração não podem ser alteradas enquanto um processo de trabalho está em execução.

Uma vez que o processo mestre recebe os sinais do HUP, ele tenta reler os arquivos de configuração. Se os arquivos de configuração puderem ser analisados e não contiverem erros, o processo mestre sinalizará a todos os processos de trabalho existentes para serem encerrados normalmente. Após a sinalização, ele inicia novos processos de trabalho com a nova configuração.

Assim como no desligamento normal, o procedimento de reconfiguração pode levar muito tempo para ser concluído. Depois de sinalizar a reconfiguração para o Nginx, você pode monitorar sua lista de processos para ver quais processos antigos de trabalho do Nginx ainda estão em execução e acompanhar o progresso de sua reconfiguração.

 Se outra reconfiguração for acionada durante um procedimento de reconfiguração em execução, o Nginx iniciará uma nova coleção de processos de trabalho - mesmo que os processos de trabalho das duas últimas rodadas não tenham terminado. Isso, em princípio, pode levar ao uso excessivo da tabela de processos, portanto, é recomendável aguardar até que o procedimento de reconfiguração atual seja concluído antes de iniciar um novo.

Capítulo 2

Aqui está um exemplo de um procedimento de reconfiguração:

```
# ps -C nginx -f
UID          PID  PPID C STIME TTY          TIME CMD
raiz        5887 3201 0 12:14 ?
usr/sbin/nginx
www-data    5888 5887 0 12:14 ?
www-data    5889 5887 0 12:14 ?
www-data    5890 5887 0 12:14 ?
www-data    5891 5887 0 12:14 ?
```

Esta listagem mostra uma instância operacional do Nginx. O processo mestre tem um ID de processo de 5887. Vamos enviar um sinal HUP para o processo mestre da instância:

```
# matar -HUP 5887
```

A instância será alterada da seguinte maneira:

```
# ps -C nginx -f
UID          PID  PPID C STIME TTY          TIME CMD
raiz        5887 3201 0 12:14 ? /usr/sbin/
nginx
www-data    5888 5887 5 12:14 ? desligando
www-data    5889 5887 0 12:14 ? desligando
www-data    5890 5887 0 12:14 ? desligando
www-data    5891 5887 0 12:14 ? desligando
www-data    5918 5887 0 12:16 ?
www-data    5919 5887 0 12:16 ?
www-data    5920 5887 0 12:16 ?
www-data    5921 5887 0 12:16 ?
```

Como você pode ver, os processos de trabalho antigos com IDs de processo 5888, 5889, 5890 e 5891 estão sendo encerrados no momento. O processo mestre releu os arquivos de configuração e gerou uma nova coleção de processos de trabalho com IDs de processo 5918, 5919, 5920 e 5921.

Gerenciando Nginx

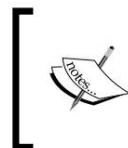
Depois de um tempo, os processos de trabalho antigos serão encerrados e a instância ficará como antes:

```
# ps -C nginx -f
UID          PID  PPID C STIME TTY          TIME CMD
raiz        5887 3201 0 12:14 ?  usr/sbin/
nginx
www-data  5918 5887 1 12:16 ?
www-data  5919 5887 3 12:16 ?
www-data  5920 5887 6 12:16 ?
www-data  5921 5887 3 12:16 ?
```

Os novos processos de trabalho adotaram a nova configuração agora.

Reabrindo o arquivo de log

Reabrir o arquivo de log é simples, mas extremamente importante para a operação contínua do seu servidor. Quando a reabertura do arquivo de log é acionada com o sinal **USR1**, o processo mestre de uma instância pega a lista de arquivos de log configurados e abre cada um deles. Se for bem-sucedido, ele fecha os arquivos de log antigos e sinaliza aos processos de trabalho para reabrir os arquivos de log. Os processos de trabalho agora podem repetir com segurança o mesmo procedimento e, depois disso, a saída do log é redirecionada para os novos arquivos. Depois disso, os processos de trabalho fecham todos os descritores de arquivo de log antigos que eles mantêm abertos no momento.



Os caminhos para arquivos de log não mudam durante este procedimento. O Nginx espera que os arquivos de log antigos sejam renomeados antes de acionar esta função. É por isso que ao abrir arquivos de log com os mesmos caminhos, o Nginx efetivamente cria ou abre novos arquivos.



As etapas do procedimento de reabertura do arquivo de log são as seguintes:

1. Os arquivos de log são renomeados ou movidos para novos locais por meio de uma ferramenta externa.
2. Você envia ao Nginx o sinal **USR1**. Nginx fecha os arquivos antigos e abre novos.
3. Os arquivos antigos agora estão fechados e podem ser arquivados.
4. Novos arquivos agora estão ativos e sendo usados.

Capítulo 2

Uma ferramenta típica para gerenciar arquivos de log do Nginx é o logrotate. A ferramenta logrotate é uma ferramenta bastante comum que pode ser encontrada em muitas distribuições Linux. Aqui está um exemplo de arquivo de configuração para logrotate que executa automaticamente o procedimento de rotação do arquivo de log:

```
/var/log/nginx/*.log {
    diário
    ausente
    girar 7
    comprimir
    compressão retardada
    notificação vazia
    criar 640 nginx adm
    scripts compartilhados
    pós-rotação
    [ -f /var/run/nginx.pid ] && kill -USR1 `cat
    /var/run/nginx.pid`
    texto final
}
```

O script anterior gira diariamente cada arquivo de log que pode encontrar no diretório /var/log/nginx pasta. Os arquivos de log são mantidos até que sete arquivos tenham se acumulado. A compressão de atraso As opções especificam que os arquivos de log não devem ser compactados imediatamente após a rotação para evitar uma situação em que o Nginx continue gravando em um arquivo sendo compactado.

Problemas no procedimento de rotação do arquivo de log podem levar a perdas de dados. Aqui está uma lista de verificação que o ajudará a configurar seu procedimento de rotação do arquivo de log corretamente:

- Certifique-se de que o sinal USR1 seja entregue somente após a movimentação dos arquivos de log.
Se isso não for feito, o Nginx gravará em arquivos girados em vez de novos.
- Certifique-se de que o Nginx tenha direitos suficientes para criar arquivos na pasta de log. Se o Nginx não conseguir abrir novos arquivos de log, o procedimento de rotação falhará.

Atualização binária do Nginx

O Nginx é capaz de atualizar seu próprio binário durante a operação. Isso é feito passando soquetes de escuta para um novo binário e listando-os em uma variável de ambiente especial.

Esta função pode ser usada para atualizar seu binário com segurança para uma nova versão ou experimentar novos recursos se você usar um binário personalizado com plugins.

Gerenciando Nginx

Com outros servidores da Web, essa operação exigiria parar completamente o servidor e iniciá-lo novamente com um novo binário. Seu serviço ficaria indisponível por um breve período. A função de atualização do binário Nginx é usada para evitar a interrupção do seu serviço e fornece uma opção de retorno se algo der errado com o novo binário.

Para atualizar seu binário, primeiro certifique-se de que ele tenha a mesma configuração de código-fonte que o binário antigo. Consulte a *configuração Copiando o código-fonte de pacotes pré-criados* seção no *Capítulo 1, Começando com Nginx*, para aprender como construir um binário com configuração de código-fonte de outro binário.

Quando o novo binário for compilado, renomeie o antigo e coloque o novo binário em seu lugar:

```
#mv/usr/sbin/nginx/usr/sbin/nginx.old  
# mv objs/nginx /usr/sbin/nginx
```

A sequência anterior assume que seu diretório de trabalho atual contém uma árvore de código-fonte Nginx.

Em seguida, envie o sinal USR2 para o processo mestre da instância em execução:

```
# matar -USR2 12995
```

O processo mestre renomeará seu arquivo PID adicionando um sufixo .oldbin e iniciará o novo binário que criará um novo processo mestre. O novo processo mestre lerá e analisará a configuração e gerará novos processos de trabalho. A instância agora se parece com isso:

UID	PID	PPID	C	STIME	TTY	TIME	CMD
raiz	12995			1 0	13:28 ?	00:00:00	nginx: processo mestre / usr/sbin/nginx
www-data	12996	12995	0	13:28 ?		00:00:00	nginx: processo de trabalho
www-data	12997	12995	0	13:28 ?		00:00:00	nginx: processo de trabalho
www-data	12998	12995	0	13:28 ?		00:00:00	nginx: processo de trabalho
www-data	12999	12995	0	13:28 ?		00:00:00	nginx: processo de trabalho
raiz	13119	12995	0	13:30 ?		00:00:00	nginx: processo mestre / usr/sbin/nginx
www-data	13120	13119	2	13:30 ?		00:00:00	nginx: processo de trabalho
www-data	13121	13119	0	13:30 ?		00:00:00	nginx: processo de trabalho
www-data	13122	13119	0	13:30 ?		00:00:00	nginx: processo de trabalho
www-data	13123	13119	0	13:30 ?		00:00:00	nginx: processo de trabalho

Capítulo 2

No código anterior, podemos ver dois processos mestres: um para o binário antigo (12995) e outro para o novo binário (13119). O novo processo mestre herda os soquetes de escuta do antigo processo mestre e os trabalhadores de ambas as instâncias aceitam conexões de entrada.

Desligamento gracioso do trabalhador

Para testar totalmente o novo binário, precisamos pedir ao antigo processo mestre para encerrar normalmente seus processos de trabalho. Depois que o novo binário for iniciado e os processos de trabalho do novo binário estiverem em execução, envie ao processo mestre da instância antiga o sinal WINCH usando o seguinte comando:

```
# matar -GUINHO 12995
```

Então, as conexões serão aceitas apenas pelos trabalhadores da nova instância. Os processos de trabalho da instância antiga serão encerrados normalmente:

UID	PID PPID C STIME TTY	TIME CMD
raiz	12995	1 0 13:28 ?
usr/sbin/nginx		00:00:00 nginx: processo mestre /
www-data	12996 12995 2 13:28 ?	desligando
		00:00:17 nginx: o processo de trabalho é
www-data	12998 12995 1 13:28 ?	
		00:00:13 nginx: o processo de trabalho é
desligando		
www-data	12999 12995 2 13:28 ?	
		00:00:18 nginx: o processo de trabalho é
desligando		
raiz	13119 12995 0 13:30 ?	
usr/sbin/nginx		00:00:00 nginx: processo mestre /
www-data	13120 13119 2 13:30 ?	
		00:00:18 nginx: processo de trabalho
www-data	13121 13119 2 13:30 ?	
		00:00:16 nginx: processo de trabalho
www-data	13122 13119 2 13:30 ?	
		00:00:12 nginx: processo de trabalho
www-data	13123 13119 2 13:30 ?	
		00:00:15 nginx: processo de trabalho

Finalmente, os processos de trabalho do binário antigo serão encerrados e apenas os processos de trabalho do novo binário permanecerão:

UID	PID PPID C STIME TTY	TIME CMD
raiz	12995	1 0 13:28 ?
usr/sbin/nginx		00:00:00 nginx: processo mestre /
raiz	13119 12995 0 13:30 ?	
usr/sbin/nginx		00:00:00 nginx: processo mestre /
www-data	13120 13119 3 13:30 ?	
		00:00:20 nginx: processo de trabalho
www-data	13121 13119 3 13:30 ?	
		00:00:20 nginx: processo de trabalho
www-data	13122 13119 2 13:30 ?	
		00:00:16 nginx: processo de trabalho
www-data	13123 13119 2 13:30 ?	
		00:00:17 nginx: processo de trabalho

Gerenciando Nginx

Agora, apenas os processos de trabalho do novo binário estão aceitando e processando conexões de entrada.

Finalizando o procedimento de atualização

Uma vez que apenas os trabalhadores do novo binário estão sendo executados, você tem duas opções.

Se o novo binário estiver funcionando bem, o antigo processo mestre pode ser encerrado enviando o sinal QUIT :

```
# kill -QUIT 12995
```

O processo mestre antigo removerá seu arquivo PID e a instância estará pronta para a próxima atualização. Mais tarde, se você encontrar algum problema com o novo binário, poderá fazer o downgrade para o binário antigo repetindo todo o procedimento de atualização do binário.

Se o novo binário não estiver funcionando corretamente, você pode reiniciar os processos de trabalho do antigo processo mestre enviando o sinal HUP :

```
# matar -HUP 12995
```

O antigo processo mestre reiniciará seus processos de trabalho sem reler os arquivos de configuração, e os trabalhadores de binários antigos e novos agora aceitarão conexões de entrada:

```
# ps -C nginx -f
```

UID	PID	PPID	C	S	TIME	TTY	CMD
raiz	12995		1	0	13:28 ?		00:00:00 nginx: processo mestre /usr/sbin/nginx
raiz	13119	12995	0	13:30 ?			00:00:00 nginx: processo mestre /usr/sbin/nginx
www-data	13120	13119	4	13:30 ?			00:01:25 nginx: processo de trabalho
www-data	13121	13119	4	13:30 ?			00:01:29 nginx: processo de trabalho
www-data	13122	13119	4	13:30 ?			00:01:21 nginx: processo de trabalho
www-data	13123	13119	4	13:30 ?			00:01:27 nginx: processo de trabalho
www-data	13397	12995	4	14:02 ?			00:00:00 nginx: processo de trabalho
www-data	13398	12995	0	14:02 ?			00:00:00 nginx: processo de trabalho
www-data	13399	12995	0	14:02 ?			00:00:00 nginx: processo de trabalho
www-data	13400	12995	0	14:02 ?			00:00:00 nginx: processo de trabalho

Capítulo 2

Os processos do novo binário podem ser encerrados normalmente enviando ao novo processo mestre o sinal QUIT :

```
# matar - SAIR 13119
```

Depois disso, você precisa retornar o binário antigo de volta ao seu local:

```
# mv /usr/sbin/nginx.old/usr/sbin/nginx
```

A instância agora está pronta para a próxima atualização.



Se um processo de trabalho estiver demorando muito para encerrar por algum motivo, você pode forçá-lo a encerrar enviando diretamente o sinal KILL.

Se o novo binário não estiver funcionando corretamente e você precisar de uma solução urgente, você pode encerrar urgentemente o novo processo mestre enviando o sinal TERM :

```
# matar -TERMO 13119
```

Os processos do novo binário serão encerrados imediatamente. O antigo processo mestre será notificado e iniciará novos processos de trabalho. O processo mestre antigo também moverá seu arquivo PID de volta para seu local original para que ele substitua o arquivo PID do novo binário. Depois disso, você precisa retornar o binário antigo de volta ao seu local original:

```
# mv /usr/sbin/nginx.old/usr/sbin/nginx
```

A instância agora está pronta para operação adicional ou a próxima atualização.

Lidando com casos difíceis

Em casos extremamente raros, você pode se deparar com uma situação difícil. Se um processo de trabalho não for encerrado quando solicitado em um tempo razoável, pode haver um problema com ele. Os sinais típicos de tais problemas são os seguintes:

- Um processo passa muito tempo no estado de execução (R) e não é encerrado
- Um processo passa muito tempo no estado de suspensão ininterrupta (D) e não desliga
- Um processo está dormindo (S) e não é encerrado

Gerenciando Nginx

Em cada um desses casos, você pode forçar o encerramento do processo de trabalho enviando primeiro o sinal TERM diretamente ao processo de trabalho. Se o processo de trabalho não reagir em 30 segundos, você pode forçar o encerramento do processo enviando o sinal KILL .

Scripts de inicialização específicos de distribuição

No Ubuntu, Debian e RHEL, o script de inicialização automatiza as sequências de controle anteriores. Ao usar o script de inicialização, você não precisa se lembrar da sequência exata dos comandos e dos nomes dos sinais. A tabela a seguir ilustra o uso do script de inicialização:

Comando	Equivalente a
serviço nginx iniciar	sbin/nginx
serviço nginx parar serviço	TERM, espere 30 segundos e depois KILL
nginx reiniciar serviço nginx	serviço nginx stop e serviço nginx start
configtest nginx -t <arquivo de configuração>	
serviço nginx recarregar	serviço nginx configtest e HUP
serviço nginx girar serviço	USR1
nginx upgrade	USR2 e QUIT para o antigo mestre
status do serviço nginx	mostrar o status da instância

O procedimento de atualização do binário é limitado a iniciar o novo binário e sinalizar o antigo processo mestre para desligar normalmente, então você não tem a opção de testar o novo binário neste caso.

Alocando processos de trabalho

Agora consideramos recomendações sobre a alocação de processos de trabalho. Primeiro, vamos discutir um pouco sobre o fundo. Nginx é um servidor web assíncrono, o que significa que as operações reais de entrada/saída são executadas de forma assíncrona com a execução de um processo de trabalho. Cada processo de trabalho executa um loop de eventos que busca todos os descritores de arquivo que precisam de processamento usando uma chamada de sistema especial e, em seguida, atende a cada um desses descritores de arquivo usando operações de E/S sem bloqueio. Portanto, cada processo de trabalho atende a várias conexões.

Capítulo 2

Nessa situação, o tempo entre um evento ocorre em um descritor de arquivo e esse descritor de arquivo pode ser atendido (ou seja, latência) depende de quanto tempo um ciclo completo de processamento de evento pode ser concluído. Portanto, para obter maior latência, faz sentido penalizar a competição por recursos de CPU entre processos de trabalho em favor de mais conexões por processo, pois isso reduziria o número de trocas de contexto entre processos de trabalho.

Portanto, nos sistemas vinculados à CPU, faz sentido alocar tantos processos de trabalho quantos os núcleos de CPU no sistema. Por exemplo, considere esta saída do comando top (esta saída pode ser obtida pressionando 1 no teclado após o início do top):

```
topo - 10:52:54 até 48 min, 2 usuários, carga média: 0,11, 0,18, 0,27
Tarefas: 273 no total, 2 em execução, 271 dormindo, 0 paradas, 0 zumbis
% Cpu0: 1,7 us, 0,3 sy, 0,0 ni, 97,7 id, 0,3 wa, 0,0 hi, 0,0 si,
0,0 ponto
%Cpu1: 0,7 us, 0,3 sy, 0,0 ni, 94,7 id, 4,0 wa, 0,0 hi, 0,3 si, 0,0 st

%Cpu2: 1,7 us, 1,0 sy, 0,0 ni, 97,3 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

%Cpu3: 3,0 us, 1,0 sy, 0,0 ni, 95,0 id, 1,0 wa, 0,0 hi, 0,0 si, 0,0 st

%Cpu4: 0,0 us, 0,0 sy, 0,0 ni, 100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

%Cpu5: 0,3 us, 0,3 sy, 0,0 ni, 99,3 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

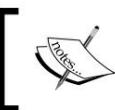
%Cpu6: 0,3 us, 0,0 sy, 0,0 ni, 99,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

%Cpu7: 0,0 us, 0,3 sy, 0,0 ni, 99,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
```

Este sistema tem oito núcleos de CPU independentes. O número máximo de processos de trabalho que não competirão por núcleos de CPU neste sistema é, portanto, oito. Para configurar o Nginx para iniciar um número específico de processos de trabalho, você pode usar a diretiva worker_processes no arquivo de configuração principal:

```
trabalhadores_processos 8;
```

O comando anterior instruirá o Nginx a iniciar oito processos de trabalho para atender às conexões de entrada.

Gerenciando Nginx

Se o número de processos de trabalho for definido como um número menor que o número de núcleos de CPU, o Nginx não poderá aproveitar todo o paralelismo disponível em seu sistema.

Para estender o número máximo de conexões que podem ser processadas por um processo de trabalho, use a diretiva `worker_connections` :

```
eventos {
    trabalhador_conexões 10000;
}
```

O comando anterior estenderá o número total de conexões que podem ser alocadas para 10.000. Isso inclui conexões de entrada (conexões de clientes) e de saída (conexões com servidores proxy e outros recursos externos).

Em sistemas vinculados a E/S de disco, na ausência do recurso AIO, latência adicional pode ser introduzida no ciclo de eventos devido ao bloqueio de operações de E/S de disco. Enquanto um processo de trabalho está aguardando a conclusão de uma operação de E/S de disco de bloqueio em um determinado descritor de arquivo, os outros descritores de arquivo não podem ser atendidos. No entanto, outros processos podem usar os recursos de CPU disponíveis. Portanto, adicionar processos de trabalho além do número de canais de E/S disponíveis pode não levar a uma melhoria no desempenho.

Em sistemas com demandas de recursos mistos, uma estratégia de alocação de processo de trabalho diferente das duas mencionadas anteriormente pode ser necessária para obter um melhor desempenho. Tente variar o número de trabalhadores para obter a configuração que funciona melhor. Isso pode variar de um trabalhador a centenas de trabalhadores.

Configurando o Nginx para servir dados estáticos

Agora que você é mais proficiente na instalação, configuração e gerenciamento do Nginx, podemos prosseguir com algumas questões práticas. Vamos ver como podemos configurar o Nginx para servir dados estáticos, como imagens, CSS ou arquivos JavaScript.

Primeiro, pegaremos a configuração de amostra do capítulo anterior e faremos com que ela suporte vários hosts virtuais usando a inclusão de curingas:

```
error_log logs/error.log;

trabalhadores_processos 8;

eventos {
    use epoll;
```

```

trabalhador_conexões 10000;
}

http{
    incluir          mime.types;
    tipo_padrão     aplicação/fluxo de octetos;

    inclua /etc/nginx/site-enabled/*.conf;
}

```

Configuramos o Nginx para aproveitar oito núcleos de processador e incluir todos os arquivos de configuração localizados em /etc/nginx/site-enabled.

Em seguida, vamos configurar um host virtual static.example.com para servir dados estáticos.

O conteúdo a seguir vai para o arquivo /etc/nginx/site-enabled/static.example.com.conf:

```

servidor {
    ouço          80;
    nome_do_servidor static.example.com;

    access_log /var/log/nginx/static.example.com-access.log main;

    enviar arquivo ativado;
    sendfile_max_chunk 1M;
    tcp_nopush on;
    gzip_static ativado;

    root /usr/local/www/static.example.com;
}

```

Este arquivo configura o host virtual static.example.com. O local raiz do host virtual é definido como /usr/local/www/static.example.com. Para permitir uma recuperação mais eficiente de arquivos estáticos, encorajamos o Nginx a usar a chamada de sistema sendfile() (sendfile ativado) e definir o máximo de parte sendfile para 1 MB. Também habilitamos a opção "TCP_NOPUSH" para melhorar a utilização do segmento TCP ao usar sendfile() (tcp_nopush ativado).

A diretiva gzip_static on instrui o Nginx a verificar cópias gzipadas de arquivos estáticos, como main.js.gz para main.js e styles.css.gz para styles.css. Se eles forem encontrados, o Nginx indicará a presença da codificação de conteúdo .gzip e usará o conteúdo dos arquivos compactados em vez do original.

Essa configuração é adequada para hosts virtuais que atendem a arquivos estáticos de tamanho pequeno a médio.

Instalando certificados SSL

Hoje, mais de 60% do tráfego HTTP na Internet é protegido por SSL.

Na presença de ataques sofisticados, como envenenamento de cache e sequestro de DNS, o SSL é obrigatório se o conteúdo da web tiver algum valor.

O Nginx tem suporte SSL de alta classe e facilita a configuração. Vamos percorrer o procedimento de instalação de um host virtual SSL.

Antes de começarmos, verifique se o pacote openssl está instalado em seu sistema:

```
# apt-get install openssl
```

Isso garantirá que você tenha as ferramentas necessárias para passar pelo procedimento de emissão do certificado SSL.

Criando uma solicitação de assinatura de certificado

Você precisa de um certificado SSL para configurar um host virtual SSL. Para obter um certificado real, você precisa entrar em contato com uma autoridade de certificação para emitir um certificado SSL. Uma autoridade de certificação geralmente cobrará uma taxa por isso.

Para emitir um certificado SSL, uma autoridade de certificação precisa de uma **Solicitação de Assinatura de Certificado (CSR)** de você. Um CSR é uma mensagem criada por você e enviada a uma autoridade de certificação contendo seus dados de identificação, como nome distinto, endereço e sua chave pública.

Para gerar um CSR, execute o seguinte comando:

```
openssl req -new -newkey rsa:2048 -nodes -keyout your_domain_name.key -out  
your_domain_name.csr
```

Isso iniciará o processo de geração de dois arquivos: uma chave privada para a descriptografia de seu certificado SSL (`your_domain_name.key`) e uma solicitação de assinatura de certificado (`your_domain_name.csr`) usada para solicitar um novo certificado SSL.

Este comando solicitará seus dados de identificação:

- **Nome do país (C):** Este é um código de país de duas letras, por exemplo, NL ou US.
- **Estado ou província (S):** Este é o nome completo do estado em que você ou sua empresa está, por exemplo, Noord-Holland.
- **Localidade ou cidade (L):** Esta é a cidade em que você ou sua empresa estão, por exemplo, Amsterdã.

Capítulo 2

- **Organização (O):** Se sua empresa ou departamento tem &, @, ou qualquer outro símbolo usando a tecla **Shift** em seu nome, você deve soletrar o símbolo ou omiti-lo para se inscrever. Por exemplo, XY & Z Corporation seria XYZ Corporation ou XY and Z Corporation.
- **Unidade Organizacional (OU):** Este campo é o nome do departamento ou unidade organizacional que faz a solicitação.
- **Nome comum (CN):** Este é o nome completo do host que você está protegendo.

O último campo é de particular importância aqui. Ele deve corresponder ao nome completo do host que você está protegendo. Por exemplo, se você registrou um domínio example.com e os usuários se conectarão a www.example.com, você deverá inserir www.example.com no campo de nome comum. Se você inserir example.com nesse campo, o certificado não será válido para www.example.com.



Não adote atributos opcionais como endereço de e-mail, senha de desafio ou o nome da empresa opcional ao gerar o CSR. Eles não agregam muito valor, mas apenas expõem mais dados pessoais.

Seu CSR está pronto agora. Depois de salvar sua chave privada em algum local seguro, você pode entrar em contato com uma autoridade de certificação e se inscrever para um certificado SSL. Apresente seu CSR assim que solicitado.

Instalando um certificado SSL emitido

Uma vez que seu certificado é emitido, você pode prosseguir com a configuração do seu servidor SSL. Salve seu certificado com um nome descritivo, como `your_domain_name.crt`. Mova-o para um diretório seguro ao qual apenas o Nginx e o superusuário tenham acesso. Usaremos `/etc/ssl` para simplificar como exemplo de tal diretório.

Agora, você pode começar a adicionar configuração para seu host virtual seguro:

```
servidor {
    ouça 443;
    nome_do_servidor seu.domínio.com;
    ssl ligado;
    ssl_certificate /etc/ssl/your_domain_name.crt;
    ssl_certificate_key /etc/ssl/your_domain_name.key;
    [... o resto da configuração...]
}
```

Gerenciando Nginx

O nome do domínio na diretiva `server_name` deve corresponder ao valor do campo de nome comum em sua solicitação de assinatura de certificado.

Após salvar a configuração, reinicie o Nginx usando o seguinte comando:

```
# serviço nginx reinicie
```

Navegue até `https://seu.domínio.com` para abrir uma conexão segura com seu servidor agora.

Redirecionamento permanente de um host virtual não seguro

A configuração anterior trata apenas de solicitações emitidas para o serviço HTTPS (porta 443) do seu servidor. Na maioria das vezes, você estará executando o serviço HTTP simples (porta 80) próximo ao seguro.

Por vários motivos, não é aconselhável ter configurações diferentes para os serviços HTTP e HTTPS simples para o mesmo nome de host. Se determinados recursos estiverem disponíveis por HTTP simples, mas não por SSL ou vice-versa, isso poderá levar a referências incorretas se uma URL apontando para um de seus recursos for tratada de maneira independente de esquema.

Da mesma forma, se determinados recursos forem disponibilizados por engano por HTTP simples e SSL, será um erro de segurança porque o recurso pode ser obtido e interagido de maneira não segura simplesmente alterando o esquema `https://` para `http://`.

Para evitar esses problemas e simplificar sua configuração, você pode configurar um redirecionamento permanente simples do host virtual não SSL para o host virtual SSL:

```
servidor {  
    ouço          80;  
    nome_do_servidor seu.domínio.com;  
  
    reescrever ^/(.*)$ https://seu.domínio.com/$1 permanente;  
}
```

Isso garante que todas as solicitações por HTTP simples para qualquer recurso em seu site serão ser redirecionado para o recurso idêntico no host virtual SSL.

Gerenciando arquivos temporários

Gerenciar arquivos temporários geralmente não é grande coisa, mas você deve estar ciente disso.

O Nginx usa arquivos temporários para armazenar dados transitórios, como os seguintes:

- Corpos de solicitação grandes recebidos de usuários
- Grandes corpos de resposta recebidos de servidores proxy ou via protocolos FastCGI, SCGI ou UWCGI.

Na seção *Instalando o Nginx* do Capítulo 1, *Introdução ao Nginx*, você viu o local padrão das pastas temporárias para esses arquivos. A tabela a seguir lista as diretivas de configuração que especificam pastas temporárias para vários Nginx Módulos principais:

Diretiva	Objetivo
client_body_temp_path	Especifica o caminho temporário para os dados do corpo da solicitação do cliente
proxy_temp_path	Especifica o caminho temporário para respostas de servidores proxy
fastcgi_temp_path	Especifica o caminho temporário para respostas de servidores FastCGI
scgi_temp_path	Especifica o caminho temporário para respostas de servidores SCGI
uwsgi_temp_path	Especifica o caminho temporário para respostas de servidores UWCGI

Os argumentos das diretivas anteriores são os seguintes

```
proxy_temp_path <caminho> [<nível1> [<nível2> [<nível3>]]]
```

No código anterior, <path> especifica o caminho para o diretório que contém arquivos temporários e os níveis especificam o número de caracteres em cada nível de diretórios com hash.

O que é um diretório com hash? No UNIX, um diretório no sistema de arquivos é essencialmente um arquivo que simplesmente contém uma lista de entradas desse diretório. Então, imagine que um de seus diretórios temporários contém 100.000 entradas. Cada pesquisa neste diretório verifica rotineiramente todas essas 100.000 entradas, o que não é muito eficiente. Para evitar isso, você pode dividir seu diretório temporário em vários subdiretórios, cada um deles contendo um conjunto limitado de arquivos temporários.

Ao especificar níveis, você instrui o Nginx a dividir seu diretório temporário em um conjunto de subdiretórios, cada um com um número específico de caracteres em seu nome, por exemplo, uma diretiva:

```
proxy_temp_path /var/lib/nginx/proxy 2;
```

Gerenciando Nginx

A linha de código anterior instrui o Nginx a armazenar um arquivo temporário chamado 3924510929 no caminho /var/lib/nginx/proxy/29/3924510929.

Da mesma forma, a diretiva proxy_temp_path /var/lib/nginx/proxy 1 2 instrui o Nginx a armazenar um arquivo temporário chamado 1673539942 no caminho /var/lib/nginx/proxy/2/94/1673539942.

Como você pode ver, os caracteres que constituem os nomes dos diretórios intermediários são extraídos da cauda do nome do arquivo temporário.

As estruturas de diretórios temporários hierárquicos e não hierárquicos precisam ser limpas de tempos em tempos. Isso pode ser feito percorrendo a árvore de diretórios e removendo todos os arquivos que residem nesses diretórios. Você pode usar um comando como o seguinte:

```
find /var/lib/nginx/proxy -type f -regex '.*[0-9]+'\$' | xargs -I '{}' rm "{}"
```

Você pode usar o comando do shell interativo. Este comando localizará todos os arquivos que terminam com dígitos localizados no diretório temporário e removerá cada um desses arquivos executando rm. Este comando solicitará a remoção se encontrar algo estranho.

Para o modo não interativo, você pode usar um comando mais perigoso:

```
find /var/lib/nginx/proxy -type f -regex '.*[0-9]+'\$' | xargs -I '{}' rm -f "{}"
```

Este comando não solicitará a remoção de arquivos.

Este comando é perigoso, pois remove cegamente um conjunto de arquivos amplamente especificado. Para evitar a perda de dados, siga os seguintes princípios ao gerenciar diretórios temporários:

- Nunca armazene nada além de arquivos temporários dentro de diretórios temporários
- Sempre use caminhos absolutos no primeiro argumento de um comando
- Se possível, verifique o que você está prestes a remover substituindo rm por echo para imprimir a lista de arquivos a serem fornecidos ao rm
- Certifique-se de que o Nginx armazene arquivos temporários em um usuário especialmente designado, como ninguém ou www-data, e nunca sob o superusuário
- Certifique-se de que o comando acima seja executado sob um usuário especialmente designado, como ninguém ou www-data, e nunca sob o superusuário

Como comunicar problemas aos desenvolvedores

Se você estiver executando versões não estáveis do Nginx para avaliação ou usando módulos próprios ou de terceiros para o Nginx, sua instância poderá ocasionalmente apresentar falhas. Se você decidir comunicar esses problemas aos desenvolvedores, aqui está um guia que o ajudará a fazê-lo da maneira mais eficiente.

Os desenvolvedores geralmente não têm acesso aos sistemas de produção, mas conhecer o ambiente em que sua instância Nginx está sendo executada é crucial para rastrear a causa do problema.

Portanto, você precisa fornecer informações detalhadas sobre o problema. Informações detalhadas sobre um travamento podem ser encontradas no arquivo principal que foi criado após o travamento.

Aviso!



O arquivo principal contém um dump de memória de um processo de trabalho no momento de uma falha e, portanto, pode conter informações confidenciais, como senhas, chaves ou dados privados. Portanto, nunca compartilhe arquivos principais com pessoas em quem você não confia.

Em vez disso, use o procedimento a seguir para obter informações detalhadas sobre uma falha:

1. Obtenha uma cópia do binário Nginx que você executa com informações de depuração (veja as instruções a seguir)
2. Se um arquivo principal estiver disponível, execute gdb no binário com as informações de depuração:

```
# gdb ./nginx-binary core
```

3. Se a execução for bem-sucedida, isso abrirá o prompt do gdb . Digite bt full nele:

```
(gdb) bt completo  
[... produz um despejo ... ]
```

O comando anterior produzirá um longo despejo da pilha no momento da falha e geralmente é suficiente para depurar uma ampla variedade de problemas. Faça um resumo da configuração que resultou em uma falha e envie para o desenvolvedor junto com o rastreamento de pilha completo.

Criando um binário com informações de depuração

Um rastreamento de pilha detalhado pode ser obtido apenas de um binário com informações de depuração. Você não precisa necessariamente executar um binário com informações de depuração. Só é necessário ter um binário idêntico ao que você executa, mas com informações extras de depuração em cima dele.

É possível produzir tal binário a partir do código-fonte do binário que você está executando, configurando a árvore de origem com uma opção extra `--with-debug`. Os passos são os seguintes:

1. Primeiro, obtenha os argumentos do script de configuração do binário de sua instância está correndo:

```
$ /usr/sbin/nginx -V
```

2. Adicione a opção `--with-debug` na frente da string do argumento e execute os scripts de configuração:

```
$ ./configure --with-debug --with-cc-opt='-g -O2 -fstack-protector --param=ssp-buffer-size=4
-Wformat -Werror=format-security -D_
FORTIFY_SOURCE=2' --with-ld-opt=' -Wl,-Bsymbolic-functions -Wl,
com, relro '...
```

Siga as etapas restantes do procedimento de compilação (consulte o capítulo anterior para obter detalhes). Ao terminar, um binário idêntico ao que você está executando, mas com informações de depuração, aparece no diretório `objs` de sua árvore de origem:

`$ arquivo objs/nginx`

`objs/nginx: executável LSB de 32 bits ELF, Intel 80386, versão 1 (SYSV), vinculado dinamicamente (usa bibliotecas compartilhadas), para GNU/Linux 2.6.32, BuildID[sha1]=7afba0f9be717c965a3cfaaefb6e2325bdcea676, não removido`

Agora, você pode usar esse binário para obter um rastreamento de pilha completo do arquivo principal produzido por seu binário gêmeo.

Consulte a seção anterior para aprender como produzir um rastreamento de pilha.

Resumo

Neste capítulo, você aprendeu muitas técnicas de gerenciamento do Nginx. Cobrimos quase todo o ciclo de operação do Nginx, exceto pelos detalhes dependentes do problema.

Nos próximos capítulos, você começará a aprender sobre recursos específicos do Nginx e como aplicá-los. Isso adicionará um pouco mais de força às suas habilidades principais do Nginx.

3

Proxy e cache

Projetado como um acelerador da web e um servidor front-end, o Nginx possui ferramentas poderosas para delegar tarefas complexas a servidores upstream enquanto se concentra no trabalho pesado. O proxy reverso é uma dessas ferramentas que transforma o Nginx em um componente essencial de qualquer serviço web de alto desempenho.

Ao abstrair as complexidades do HTTP e manipulá-las de maneira escalável e eficiente, o Nginx permite que os aplicativos da Web se concentrem na solução do problema para o qual foram projetados sem tropeçar em detalhes de baixo nível.

Neste capítulo, você aprenderá:

- Como configurar o Nginx como proxy reverso
- Como tornar o proxy transparente para o servidor upstream e o usuário final
- Como lidar com erros de upstream
- Como usar o cache Nginx

Você descobrirá como usar todos os recursos do proxy reverso Nginx e transformá-lo em uma ferramenta poderosa para acelerar e dimensionar seu serviço web.

Nginx como proxy reverso

O HTTP é um protocolo complexo que lida com dados de diferentes modalidades e possui inúmeras otimizações que, se implementadas corretamente, podem levar a um aumento significativo no desempenho do serviço web.

Ao mesmo tempo, os desenvolvedores de aplicativos da Web têm menos tempo para lidar com otimizações e problemas de baixo nível. A mera ideia de desacoplar um servidor de aplicativos da Web de um servidor de front-end muda o foco no gerenciamento do tráfego de entrada para o front-end, enquanto muda o foco na funcionalidade, lógica do aplicativo e recursos para o servidor de aplicativos da Web. É aqui que o Nginx entra em ação como um ponto de desacoplamento.

Proxy e cache

Um exemplo de ponto de desacoplamento é a terminação SSL: o Nginx recebe e processa conexões SSL de entrada, encaminha a solicitação por HTTP simples para um servidor de aplicativos e encapsula a resposta recebida de volta em SSL. O servidor de aplicativos não precisa mais se preocupar em armazenar certificados, sessões SSL, lidar com transmissões criptografadas e não criptografadas e assim por diante.

Outros exemplos de desacoplamento são os seguintes:

- Manipulação eficiente de arquivos estáticos e delegação da parte dinâmica ao upstream
- Limitação de taxa, solicitação e conexão
- Compressão de respostas do upstream
- Cache de respostas do upstream
- Acelerar uploads e downloads

Ao mudar essas funções para um frontend com Nginx, você está investindo essencialmente na confiabilidade do seu site.

Configurando o Nginx como um proxy reverso

O Nginx pode ser facilmente configurado para funcionar como um proxy reverso:

```
local/exemplo {
    proxy_pass http://upstream_server_name;
}
```

No código anterior, `upstream_server_name` é o nome do host do servidor upstream. Quando uma solicitação de localização for recebida, ela será passada para o servidor upstream com um nome de host especificado.

Se o servidor upstream não tiver um nome de host, um endereço IP pode ser usado:

```
local/exemplo {
    proxy_pass http://192.168.0.1;
}
```

Se o servidor upstream estiver escutando em uma porta não padrão, a porta pode ser adicionada à URL de destino:

```
local/exemplo {
    proxy_pass http://192.168.0.1:8080;
}
```

A URL de destino nos exemplos anteriores não tem um caminho. Isso faz com que o Nginx passe a solicitação como está, sem reescrever o caminho na solicitação original.

Capítulo 3

Se um caminho for especificado no URL de destino, ele substituirá uma parte do caminho da solicitação original que corresponde à parte correspondente do local. Por exemplo, considere a seguinte configuração:

```
local/baixar {
    proxy_pass http://192.168.0.1/media;
}
```

Se uma solicitação para /download/BigFile.zip for recebida, o caminho na URL de destino será /media e corresponderá à parte /download correspondente do URI de solicitação original. Esta parte será substituída por /media antes de passar para o servidor upstream, de modo que o caminho de solicitação passado será parecido com /media/BigFile.zip.

Se a diretiva proxy_pass for usada dentro de um local regex, a parte correspondente não poderá ser calculada. Nesse caso, um URI de destino sem caminho deve ser usado:

```
local ~* (script1|script2|script3)\.php$ {
    proxy_pass http://192.168.0.1;
}
```

O mesmo se aplica aos casos em que o caminho da solicitação foi alterado com a diretiva reescrita e é usado por uma diretiva proxy_pass .

As variáveis também podem fazer parte do URL de destino:

```
localização ~* ^/(index|content|sitemap)\.html$ {
    proxy_pass http://192.168.0.1/html/$1;
}
```

Na verdade, qualquer parte ou mesmo todo o URL de destino pode ser especificado por uma variável:

```
local/exemplo {
    proxy_pass $destino;
}
```

Isso dá flexibilidade suficiente para especificar a URL de destino para o servidor upstream. No Capítulo 5, Gerenciando o tráfego de entrada e saída, descobriremos como especificar vários servidores como upstream e distribuir conexões entre eles.

Configurando o back-end da maneira certa

A maneira correta de configurar um backend é evitar passar tudo para ele. O Nginx possui diretivas de configuração poderosas que ajudam a garantir que apenas solicitações específicas sejam delegadas ao backend.

Proxy e cache

Considere a seguinte configuração:

```
localização ~* \.php$ {
    proxy_pass http://backend;
    [...]
}
```

Isso passa cada solicitação com um URI que termina com .php para o interpretador PHP.

Isso não é apenas ineficiente devido ao uso intensivo de expressões regulares, mas também um sério problema de segurança na maioria das configurações do PHP, pois pode permitir a execução de código arbitrário por um invasor.

Nginx tem uma solução elegante para este problema na forma de try_files diretiva. A diretiva try_files recebe uma lista de arquivos e um local como último argumento. O Nginx tenta os arquivos especificados em ordem consecutiva e se nenhum deles existir, ele faz um redirecionamento interno para o local especificado. Considere o seguinte exemplo:

```
local / {
    try_files $uri $uri/ @proxy;
}

local @proxy {
    proxy_pass http://backend;
}
```

A configuração anterior primeiro procura um arquivo correspondente ao URI de solicitação, procura um diretório correspondente ao URI de solicitação na esperança de retornar um índice desse diretório e, finalmente, faz um redirecionamento interno para o local nomeado @proxy se nenhum desses existem arquivos ou diretórios.

Essa configuração garante que sempre que um URI de solicitação apontar para um objeto no sistema de arquivos, ele será tratado pelo próprio Nginx usando operações de arquivo eficientes, e somente se não houver correspondência no sistema de arquivos para o URI de solicitação fornecido, ele será delegado a o back-end.

Adicionando transparência

Uma vez encaminhado para um servidor upstream, um pedido perde certas propriedades do pedido original. Por exemplo, o host virtual em uma solicitação encaminhada é substituído pela combinação host/porta da URL de destino. A solicitação encaminhada é recebida de um endereço IP do proxy Nginx, e a funcionalidade do servidor upstream com base no endereço IP do cliente pode não funcionar corretamente.

A solicitação encaminhada precisa ser ajustada para que o servidor upstream possa obter as informações ausentes da solicitação original. Isso pode ser feito facilmente com a diretiva proxy_set_header :

```
proxy_set_header <header> <valor>;
```

A diretiva proxy_set_header recebe dois argumentos, sendo o primeiro o nome do cabeçalho que você deseja definir na solicitação com proxy e o segundo é o valor desse cabeçalho. Novamente, ambos os argumentos podem conter variáveis.

Veja como você pode passar o nome do host virtual da solicitação original:

```
local @proxy {
    proxy_pass http://192.168.0.1;
    proxy_set_header Host $host;
}
```

A variável \$host tem uma funcionalidade inteligente. Ele não passa simplesmente o nome do host virtual da solicitação original, mas usa o nome do servidor pelo qual a solicitação é processada se o cabeçalho do host da solicitação original estiver vazio ou ausente. Se você insistir em usar o nome do host virtual simples da solicitação original, poderá usar a variável \$http_host em vez de \$host.

Agora que você sabe como manipular a solicitação com proxy, podemos informar ao servidor upstream o endereço IP do cliente original. Isso pode ser feito configurando os cabeçalhos X-Real-IP e/ou X-Forwarded-For :

```
local @proxy {
    proxy_pass http://192.168.0.1;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
```

Isso tornará o servidor upstream cliente do endereço IP do cliente original por meio do X-Real-IP ou do cabeçalho X-Forwarded-For . A maioria dos servidores de aplicativos oferece suporte a esse cabeçalho e realiza as ações apropriadas para selecionar corretamente o endereço IP original em sua API.

Manipulando redirecionamentos

O próximo desafio é reescrever redirecionamentos. Quando o servidor upstream emite um redirecionamento temporário ou permanente (códigos de status HTTP 301 ou 302), o URI absoluto nos cabeçalhos de localização ou atualização precisa ser reescrito para que contenha um nome de host adequado (o nome de host do servidor da solicitação original veio).

Proxy e cache

Isso pode ser feito usando a diretiva proxy_redirect :

```
local @proxy {
    proxy_pass http://localhost:8080;
    proxy_redirect http://localhost:8080/app http://www.example.com;
}
```

Considere um aplicativo da Web que está sendo executado em http://localhost:8080/app, enquanto o servidor original tem o endereço http://www.example.com. Suponha que o aplicativo da Web emita um redirecionamento temporário (HTTP 302) para http://localhost:8080/applicativo/login. Com a configuração anterior, o Nginx reescreverá o URI no cabeçalho de localização para http://www.example.com/login.

Se o URI de redirecionamento não fosse reescrito, o cliente seria redirecionado para http://localhost:8080/app/login, que é válido apenas em um domínio local, portanto, o aplicativo da Web não poderá funcionar corretamente. Com a diretiva proxy_redirect , o URI de redirecionamento será reescrito corretamente pelo Nginx e o aplicativo da Web poderá executar o redirecionamento corretamente.

O nome do host no segundo argumento da diretiva proxy_redirect pode ser omitido:

```
local @proxy {
    proxy_pass http://localhost:8080;
    proxy_redirect http://localhost:8080/app/;
}
```

O código anterior pode ser reduzido ainda mais para a seguinte configuração usando variáveis:

```
local @proxy {
    proxy_pass http://localhost:8080;
    proxy_redirect http://$proxy_host/app /;
}
```

A mesma opção de transparência pode ser aplicada aos cookies. No exemplo anterior, considere que os cookies estão configurados para o domínio localhost:8080, pois o servidor de aplicativos responde em http://localhost:8080. Os cookies não serão retornados pelo navegador, pois o domínio do cookie não corresponde ao domínio da solicitação.

Manipulação de cookies

Para que os cookies funcionem corretamente, o nome de domínio nos cookies precisa ser reescrito pelo proxy Nginx. Para fazer isso, você pode usar a diretiva `proxy_cookie_domain` como mostrado aqui:

```
local @proxy {
    proxy_pass http://localhost:8080;
    proxy_cookie_domain localhost:8080 www.example.com;
}
```

No exemplo anterior, o Nginx substitui o domínio do cookie localhost:8080 na resposta upstream por www.example.com. Os cookies definidos pelo servidor upstream farão referência ao domínio www.example.com e o navegador retornará cookies em solicitações subsequentes.

Se o caminho do cookie também precisar ser reescrito devido ao servidor de aplicativos estar enraizado em um caminho diferente, você poderá usar a diretiva `proxy_cookie_path` conforme mostrado no código a seguir:

```
local @proxy {
    proxy_pass http://localhost:8080;
    proxy_cookie_path /my_webapp/ /;
}
```

Neste exemplo, sempre que o Nginx detecta um cookie com um prefixo especificado no primeiro argumento da diretiva `proxy_cookie_path` (/my_webapp/), ele substitui esse prefixo pelo valor no segundo argumento da diretiva `proxy_cookie_path` (/).

Juntando tudo para o domínio www.example.com e a aplicação web rodando em localhost:8080, obtemos a seguinte configuração:

```
local @proxy {
    proxy_pass http://localhost:8080;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_redirect http://$proxy_host/app /;
    proxy_cookie_domain $proxy_host www.example.com;
    proxy_cookie_path /my_webapp/ /;
}
```

A configuração anterior garante transparência para um servidor de aplicativos da Web para que ele nem precise saber em qual host virtual está sendo executado.

Proxy e cache

Usando SSL

Se o servidor upstream oferecer suporte a SSL, as conexões com o servidor upstream poderão ser protegidas simplesmente alterando o esquema de URL de destino para https:

```
local @proxy {
    proxy_pass https://192.168.0.1;
}
```

Se a autenticidade do servidor upstream precisar ser verificada, isso pode ser ativado usando a diretiva `proxy_ssl_verify`:

```
local @proxy {
    proxy_pass https://192.168.0.1;
    proxy_ssl_verify ativado;
}
```

O certificado do servidor upstream será verificado em relação aos certificados de autoridades de certificação bem conhecidas. Em sistemas operacionais do tipo Unix, eles geralmente são armazenados em /etc/ssl/certs.

Se um upstream usa um certificado confiável que não pode ser verificado por autoridades de certificação conhecidas ou um certificado autoassinado, ele pode ser especificado e declarado como confiável usando a diretiva `proxy_ssl_trusted_certificate`. Esta diretiva especifica o caminho para o certificado do servidor upstream ou uma cadeia de certificados necessária para autenticar o servidor upstream no formato PEM. Considere o seguinte exemplo:

```
local @proxy {
    proxy_pass https://192.168.0.1;
    proxy_ssl_verify ativado;
    proxy_ssl_trusted_certificate /etc/nginx/upstream.pem;
}
```

Se o Nginx precisar se autenticar no servidor upstream, o certificado do cliente e a chave podem ser especificados usando o `proxy_ssl_certificate` e `proxy_ssl_certificate_key`. A diretiva `proxy_ssl_certificate` especifica o caminho para o certificado do cliente no formato PEM, enquanto `proxy_ssl_certificate_key` especifica o caminho para a chave privada do certificado do cliente no formato PEM. Considere o seguinte exemplo:

```
local @proxy {
    proxy_pass https://192.168.0.1;
    proxy_ssl_certificate /etc/nginx/client.pem;
    proxy_ssl_certificate_key /etc/nginx/client.key;
}
```

O certificado especificado será apresentado durante a configuração da conexão segura com o servidor upstream, e sua autenticidade será verificada pela chave privada especificada.

Manipulação de erros

Se o Nginx tiver um problema ao contatar o servidor upstream ou o servidor upstream retornar um erro, há uma opção para executar determinadas ações.

Os erros de conectividade do servidor upstream podem ser tratados usando o `error_page` diretiva:

```
local ~* (script1|script2|script3).php$ {
    proxy_pass http://192.168.0.1;
    página_de_erro 500 502 503 504 /50x.html;
}
```

Isso fará com que o Nginx retorne o documento do arquivo `50x.html` assim que ocorrer um erro de conectividade upstream.

Isso não alterará o código de status HTTP na resposta. Para alterar o código de status HTTP para bem-sucedido, você pode usar a seguinte sintaxe:

```
local ~* (script1|script2|script3).php$ {
    proxy_pass http://192.168.0.1;
    página_de_erro 500 502 503 504 =200 /50x.html;
}
```

Uma ação mais sofisticada pode ser tomada em caso de falha de um servidor upstream usando uma diretiva `error_page` que aponta para um local nomeado:

```
local ~* (script1|script2|script3).php$ {
    proxy_pass http://upstreamA;
    página_de_erro 500 502 503 504 @retry;
}

local @retry {
    proxy_pass http://upstreamB;
    página_de_erro 500 502 503 504 =200 /50x.html;
}
```

Na configuração anterior, o Nginx primeiro tenta completar a solicitação encaminhando-a para o servidor `upstreamA`. Se isso resultar em um erro, o Nginx alterna para um local nomeado `@retry` em uma tentativa de tentar com o servidor `upstreamB`. Solicite um URI durante a comutação para que o servidor `upstreamB` receba uma solicitação idêntica. Se isso também não ajudar, o Nginx retornará um arquivo estático `50x.html` fingindo que nenhum erro ocorreu.

Proxy e cache

Se um upstream respondeu, mas retornou um erro, ele pode ser interceptado em vez de passado para o cliente usando a diretiva `proxy_intercept_errors` :

```
local ~* (script1|script2|script3).php$ {
    proxy_pass http://upstreamA;
    proxy_intercept_errors ativado;
    página_de_erro 500 502 503 504 403 404 @retry;
}

local @retry {
    proxy_pass http://upstreamB;
    página_de_erro 500 502 503 504 =200 /50x.html;
}
```

Na configuração anterior, o servidor `upstreamB` será chamado mesmo quando o servidor `upstreamA` responder, mas retornar um código de status HTTP errôneo, como 403 ou 404. Isso dá ao `upstreamB` a oportunidade de corrigir os erros leves de `upstreamA`, se necessário.

No entanto, este padrão de configuração não deve proliferar muito. No *Capítulo 5, Gerenciando o tráfego de entrada e saída*, descobriremos como lidar com essas situações de maneira mais elegante, sem estruturas de configuração sofisticadas.

Escolhendo um endereço IP de saída

Às vezes, quando seu servidor proxy possui várias interfaces de rede, torna-se necessário escolher qual endereço IP deve ser usado como endereço de saída para conexões upstream. Por padrão, o sistema escolherá o endereço da interface que se conecta à rede que contém o host usado como destino na rota padrão.

Para escolher um endereço IP específico para conexões de saída, você pode usar a diretiva `proxy_bind` :

```
local @proxy {
    proxy_pass https://192.168.0.1;
    proxy_bind 192.168.0.2;
}
```

Isso fará com que o Nginx vincule os soquetes de saída ao endereço IP 192.168.0.2 antes de fazer uma conexão. O servidor upstream verá as conexões provenientes de IP endereço 192.168.0.2.

Acelerando downloads

O Nginx é muito eficiente em operações pesadas, como lidar com grandes uploads e downloads.

Essas operações podem ser delegadas ao Nginx usando funcionalidades internas e módulos de terceiros.

Para acelerar o download, o servidor upstream deve ser capaz de emitir o cabeçalho X-Accel-Redirect que aponta para a localização de um recurso que precisa ser retornado, em vez da resposta obtida do upstream. Considere a seguinte configuração:

```
local ~* (script1|script2|script3)\.php$ {
    proxy_pass https://192.168.0.1;
}

localização /internal-media/ {
    interno;
    alias /var/www/media/;
}
```

Com a configuração anterior, uma vez que o Nginx detecta o cabeçalho X-Accel-Redirect na resposta upstream, ele executa um redirecionamento interno para o local especificado neste cabeçalho. Suponha que o servidor upstream instrua o Nginx a executar um redirecionamento interno para /internal-media/ BigFile.zip. Este caminho será comparado com o local /internal-media. Este local especifica a raiz do documento em /var/www/ meios de comunicação. Portanto, se existir um arquivo /var/www/media/BigFile.zip , ele será retornado ao cliente usando operações de arquivo eficientes.

Para muitos servidores de aplicativos da Web, esse recurso oferece uma enorme velocidade — tanto porque eles podem não lidar com grandes downloads com eficiência quanto porque o proxy reduz a eficiência de grandes downloads.

Cache

Uma vez que o Nginx é configurado como um proxy reverso, é lógico transformá-lo em um proxy de cache. Felizmente, isso pode ser alcançado muito facilmente com o Nginx.

Configurando caches

Antes de habilitar o cache para um determinado local, você precisa configurar um cache.

Um cache é um diretório do sistema de arquivos contendo arquivos com itens armazenados em cache e um segmento de memória compartilhada onde as informações sobre os itens armazenados em cache são armazenadas.

Proxy e cache

Um cache pode ser declarado usando a diretiva **proxy_cache_path** :

```
proxy_cache_path <path> keys_zone=<name>:<size> [outros parâmetros...];
```

O comando anterior declara um cache enraizado no caminho **<path>** com um segmento de memória compartilhada chamado **<name>** do tamanho **<size>**.

Esta diretiva deve ser especificada na seção **http** da configuração. Cada instância da diretiva declara um novo cache e deve especificar um nome exclusivo para um segmento de memória compartilhada. Considere o seguinte exemplo:

```
http{
    proxy_cache_path /var/www/cache keys_zone=my_cache:8m;
    [...]
}
```

A configuração anterior declara um cache enraizado em **/var/www/cache** com um segmento de memória compartilhada chamado **my_cache**, que tem 8 MB de tamanho. Cada item de cache ocupa cerca de 128 bytes na memória, portanto, a configuração anterior aloca espaço para cerca de 64.000 itens.

A tabela a seguir lista outros parâmetros de **proxy_cache_path** e seus significados:

Parâmetro	Descrição
níveis	Especifica os níveis de hierarquia do diretório de cache
inativo	Especifica o tempo após o qual um item de cache será removido do cache se não tiver sido usado, independentemente da atualização
max_size	Especifica o tamanho máximo (tamanho total) de todos os itens de cache
loader_files	Especifica o número de arquivos que um processo do carregador de cache carrega em cada iteração
loader_sleep	Especifica o intervalo de tempo que um processo do carregador de cache dorme entre cada iteração
loader_threshold	Especifica o limite de tempo para cada iteração de um processo do carregador de cache

Uma vez que o Nginx é iniciado, ele processa todos os caches configurados e aloca segmentos de memória compartilhada para cada um dos caches.

Depois disso, um processo especial chamado **cache loader** cuida do carregamento de itens em cache na memória. O carregador de cache carrega itens em iterações. Os parâmetros **loader_files**, **loader_sleep** e **loader_threshold** definem o comportamento do processo do cache loader.

Durante a execução, um processo especial chamado gerenciador de cache monitora o espaço total em disco ocupado por todos os itens de cache e remove os itens menos solicitados se o espaço total consumido for maior que o especificado no parâmetro `max_size`.

Ativando o cache

Para habilitar o cache para um local, você precisa especificar o cache usando a diretiva `proxy_cache`:

```
local @proxy {
    proxy_pass http://192.168.0.1:8080;
    proxy_cache meu_cache;
}
```

O argumento da diretiva `proxy_cache` é o nome de um segmento de memória compartilhada que aponta para um dos caches configurados usando o `proxy_cache_path` diretiva. O mesmo cache pode ser usado em vários locais. A resposta upstream será armazenada em cache se for possível determinar o intervalo de expiração para ela. A fonte primária do intervalo de expiração do Nginx é o próprio upstream. A tabela a seguir explica qual cabeçalho de resposta upstream influencia o armazenamento em cache e como:

Cabeçalho de resposta upstream	Como isso influencia o cache
X-Accel-Expira	Isso especifica o intervalo de expiração do item de cache em segundos. Se o valor começar em @, o número seguinte será o carimbo de data/hora do UNIX quando o item estiver prestes a expirar. Este cabeçalho tem a prioridade mais alta.
Expira	Isso especifica o carimbo de data/hora de expiração do item de cache.
Controle de Cache	Isso habilita ou desabilita o cache
Set-Cookie	Isso desativa o cache
Variar	O valor especial * desativa o cache.

Também é possível especificar explicitamente um intervalo de expiração para vários códigos de resposta usando a diretiva `proxy_cache_valid`:

```
local @proxy {
    proxy_pass http://192.168.0.1:8080;
    proxy_cache meu_cache;
    proxy_cache_valid 200 301 302 1h;
}
```

Proxy e cache

Isso define o intervalo de expiração para respostas com códigos 200, 301, 302 para 1h (1 hora). Observe que a lista de códigos de status padrão para a diretiva proxy_cache_valid é 200, 301 e 302, portanto, a configuração anterior pode ser simplificada da seguinte maneira:

```
local @proxy {
    proxy_pass http://192.168.0.1:8080;
    proxy_cache meu_cache;
    proxy_cache_valid 10m;
}
```

Para habilitar o cache para respostas negativas, como 404, você pode estender a lista de códigos de status na diretiva proxy_cache_valid :

```
local @proxy {
    proxy_pass http://192.168.0.1:8080;
    proxy_cache meu_cache;
    proxy_cache_valid 200 301 302 1h;
    proxy_cache_valid 404 1m;
}
```

A configuração anterior armazenará em cache 404 respostas por 1m (1 minuto). O intervalo de expiração para respostas negativas é deliberadamente definido para valores muito mais baixos do que os das respostas positivas. Essa abordagem otimista garante maior disponibilidade ao esperar que as respostas negativas melhorem, considerando-as como transitórias e assumindo um tempo de vida esperado mais curto.

Escolhendo uma chave de cache

Escolher a chave de cache correta é importante para a melhor operação do cache. A chave de cache deve ser selecionada de forma que maximize a eficiência esperada do cache, desde que cada item armazenado em cache tenha conteúdo válido para todas as solicitações subsequentes que avaliam a mesma chave. Isso requer alguma explicação.

Primeiro, vamos considerar a eficiência. Quando o Nginx se refere ao servidor upstream para revalidar um item de cache, obviamente estressa o servidor upstream. Com cada acerto de cache subsequente, o Nginx reduz o estresse no servidor upstream em comparação com a situação em que as solicitações eram encaminhadas para o upstream sem armazenamento em cache. Assim, a eficiência da cache pode ser representada como $Eficiência = (Número\ de\ acertos + Número\ de\ erros) / Número\ de\ erros$.

Assim, quando nada pode ser armazenado em cache, cada solicitação leva a um erro de cache e a eficiência é 1. Mas quando obtemos 99 acertos de cache subsequentes para cada erro de cache, a eficiência é avaliada como $(99 + 1) / 1 = 100$, que é 100 vezes maior!

Capítulo 3

Em segundo lugar, se um documento estiver armazenado em cache, mas não for válido para todas as solicitações avaliadas para a mesma chave, os clientes poderão ver conteúdo inválido para suas solicitações.

Por exemplo, o upstream analisa o cabeçalho `Accept-Language` e retorna a versão do documento no idioma mais adequado. Se a chave de cache não incluir o idioma, o primeiro usuário a solicitar o documento o obterá em seu idioma e acionará o cache nesse idioma. Todos os usuários que solicitarem este documento posteriormente verão a versão em cache do documento e, portanto, poderão vê-lo no idioma errado.

Se a chave de cache incluir o idioma do documento, o cache conterá vários itens separados para o mesmo documento em cada idioma solicitado e todos os usuários o verão no idioma apropriado.

A chave de cache padrão é `$scheme$proxy_host$request_uri`.

Isso pode não ser ideal devido aos seguintes motivos:

- O servidor de aplicativos da web em `$proxy_host` pode ser responsável por vários domínios
- As versões HTTP e HTTPS do site podem ser idênticas (`$scheme` variável é redundante, duplicando itens no cache)
- O conteúdo pode variar dependendo dos argumentos da consulta

Assim, considerando tudo o que foi descrito anteriormente e dado que o HTTP e HTTPS do site são idênticas e o conteúdo varia dependendo dos argumentos da consulta, podemos definir a chave de cache para um valor mais ideal `$host$request_uriis_argsargs`. Para alterar a chave de item de cache padrão, você pode usar a diretiva `proxy_cache_key`:

```
local @proxy {
    proxy_pass http://192.168.0.1:8080;
    proxy_cache meu_cache;
    proxy_cache_key "$host$uri$is_args$args";
}
```

Esta diretiva recebe um script como seu argumento que é avaliado em um valor de uma chave de cache em tempo de execução.

Proxy e cache

Melhorando a eficiência e a disponibilidade do cache

A eficiência e a disponibilidade do cache podem ser melhoradas. Você pode impedir que um item seja armazenado em cache até que ele receba um determinado número mínimo de solicitações. Isso pode ser feito usando a diretiva `proxy_cache_min_uses` :

```
local @proxy {
    proxy_pass http://192.168.0.1:8080;
    proxy_cache meu_cache;
    proxy_cache_min_uses 5;
}
```

No exemplo anterior, a resposta será armazenada em cache assim que o item receber pelo menos cinco solicitações. Isso evita que o cache seja preenchido por itens usados com pouca frequência, reduzindo assim o espaço em disco usado para armazenamento em cache.

Uma vez que o item tenha expirado, ele pode ser revalidado sem ser despejado. Para habilitar a revalidação, use a diretiva `proxy_cache_revalidate` :

```
local @proxy {
    proxy_pass http://192.168.0.1:8080;
    proxy_cache meu_cache;
    proxy_cache_revalidate ativado;
}
```

No exemplo anterior, quando um item de cache expira, o Nginx o revalida fazendo uma solicitação condicional ao servidor upstream. Essa solicitação incluirá os cabeçalhos `If-Modified-Since` e/ou `If-None-Match` como uma referência à versão em cache. Se o servidor upstream responder com uma resposta `304 Not Modified`, o item de cache permanecerá no cache e o carimbo de data/hora de expiração será redefinido.

Múltiplas solicitações simultâneas podem ser proibidas de encher o cache ao mesmo tempo. Dependendo do tempo de reação do upstream, isso pode acelerar o preenchimento do cache e, ao mesmo tempo, reduzir a carga no servidor upstream. Para habilitar esse comportamento, você pode usar a diretiva `proxy_cache_lock` :

```
local @proxy {
    proxy_pass http://backend;
    proxy_cache meu_cache;
    proxy_cache_lock ativado;
}
```

Capítulo 3

Uma vez que o comportamento esteja habilitado, apenas uma solicitação terá permissão para preencher um item de cache ao qual está relacionado. As outras solicitações relacionadas a este item de cache aguardarão até que o item de cache seja preenchido ou o tempo limite de bloqueio expire. O tempo limite de bloqueio pode ser especificado usando a diretiva `proxy_cache_lock directive`.

Se for necessária uma maior disponibilidade do cache, você pode configurar o Nginx para responder com dados obsoletos quando uma solicitação se referir a um item em cache. Isso é muito útil quando o Nginx atua como um servidor de borda em uma rede de distribuição. Os usuários e rastreadores de mecanismos de pesquisa verão seu site disponível, mesmo que o site principal apresente problemas de conectividade. Para habilitar a resposta com dados obsoletos, use o `proxy_cache_use_stale directive`:

```
local @proxy {
    proxy_pass http://backend;
    proxy_cache meu_cache;
    tempo limite de erro proxy_cache_use_stale http_500 http_502 http_503 http_504;
}
```

A configuração anterior permite responder com dados obsoletos em caso de erro de conectividade, erro de upstream (502, 503 ou 504) e tempo limite de conexão. A tabela a seguir lista todos os valores possíveis para argumentos da diretiva `proxy_cache_use_stale`:

Valor	Significado
erro	Ocorreu um erro de conexão ou ocorreu um erro durante o envio de uma solicitação ou o recebimento de uma resposta
tempo esgotado	Uma conexão expirou durante a configuração, enviando uma solicitação ou recebendo uma resposta
invalid_header	O servidor upstream retornou uma resposta vazia ou inválida
atualizando	Habilita respostas obsoletas enquanto o item de cache está sendo atualizado
http_500	O servidor upstream retornou uma resposta com o código de status HTTP 500 (Erro do Servidor Interno)
http_502	O servidor upstream retornou uma resposta com o código de status HTTP 502 (Gateway ruim)
http_503	O servidor upstream retornou uma resposta com o código de status HTTP 503 (Serviço não disponível)
http_504	O servidor upstream retornou uma resposta com o código de status HTTP 504 (Tempo limite do gateway)
http_403	O servidor upstream retornou uma resposta com o código de status HTTP 403 (Proibido)
http_404	O servidor upstream retornou uma resposta com o código de status HTTP 404 (Não encontrado)
fora	Desativa o uso de respostas obsoletas

Proxy e cache

Tratamento de exceções e casos limítrofes

Quando o cache não é desejável ou não é eficiente, ele pode ser ignorado ou desabilitado.

Isso pode acontecer nos seguintes casos:

- Um recurso é dinâmico e varia de acordo com fatores externos
- Um recurso é específico do usuário e varia de acordo com os cookies
- O armazenamento em cache não agrupa muito valor
- Um recurso não é estático, por exemplo, um fluxo de vídeo

Quando o bypass é forçado, o Nginx encaminha a solicitação para o back-end sem procurar um item no cache. O bypass pode ser configurado usando o `proxy_cache_bypass` diretiva de desvio :

```
local @proxy {
    proxy_pass http://backend;
    proxy_cache meu_cache;
    proxy_cache_bypass $do_not_cache $arg_nocache;
}
```

Esta diretiva pode receber um ou mais argumentos. Quando qualquer um deles é avaliado como verdadeiro (valor não vazio e não 0), o Nginx não procura um item no cache para uma determinada solicitação. Em vez disso, ele encaminha diretamente a solicitação para o servidor upstream. O item ainda pode ser armazenado no cache.

Para evitar que um item seja armazenado no cache, você pode usar o `proxy_no_cache` diretiva:

```
local @proxy {
    proxy_pass http://backend;
    proxy_cache meu_cache;
    proxy_no_cache $do_not_cache $arg_nocache;
}
```

Essa diretiva funciona exatamente como a diretiva `proxy_cache_bypass`, mas impede que itens sejam armazenados no cache. Quando apenas a diretiva `proxy_no_cache` é especificada, os itens ainda podem ser retornados do cache. A combinação de `proxy_cache_bypass` e `proxy_no_cache` desabilita completamente o cache.

Agora, vamos considerar um exemplo do mundo real quando o cache precisa ser desabilitado para todas as páginas específicas do usuário. Suponha que você tenha um site desenvolvido com WordPress e deseja habilitar o cache para todas as páginas, mas desabilitar o cache para todas as páginas personalizadas ou específicas do usuário. Para implementar isso, você pode usar uma configuração semelhante à seguinte:

```
localização ~* wp\-.*\.\php|wp\-\admin {  
    proxy_pass http://backend;  
  
    proxy_set_header Host $http_host;  
    proxy_set_header X-Real-IP $remote_addr;  
}  
  
local / {  
    if ($http_cookie ~* "comment_author_\wordpress_\wp-postpass_") {  
        definir $do_not_cache 1;  
    }  
  
    proxy_pass http://backend;  
  
    proxy_set_header Host $http_host;  
    proxy_set_header X-Real-IP $remote_addr;  
  
    proxy_cache meu_cache;  
    proxy_cache_bypass $do_not_cache;  
    proxy_no_cache $do_not_cache;  
}
```

Na configuração anterior, primeiro delegamos todas as solicitações pertencentes à área administrativa do WordPress ao servidor upstream. Em seguida, usamos a diretiva `if` para procurar cookies de login do WordPress e definir a variável `$do_not_cache` como 1, se eles estiverem presentes. Em seguida, habilitamos o cache para todos os outros locais, mas desabilitamos o cache sempre que a variável `$do_not_cache` for definida como 1 usando as diretivas `proxy_cache_bypass` e `proxy_no_cache`. Isso desativa o cache para todas as solicitações com cookies de login do WordPress.

A configuração anterior pode ser estendida para extrair atrasos sem cache de argumentos ou cabeçalhos HTTP, para ajustar ainda mais seu cache.

Proxy e cache

Resumo

Neste capítulo, você aprendeu como trabalhar com proxy e cache — alguns dos recursos mais importantes do Nginx. Esses recursos praticamente definem o Nginx como um acelerador da web e ser proficiente neles é essencial para tirar o máximo proveito do Nginx.

No próximo capítulo, veremos como reescrever o funcionamento do mecanismo no Nginx e os fundamentos do controle de acesso.

4

Reescrever Motor e Controle de acesso

A World Wide Web e HTTP como seu bloco de construção operam em URLs. Como as URLs são tão fundamentais, a capacidade de um servidor de manipular URLs é essencial.

O Nginx permite manipular URLs usando um mecanismo de reescrita integrado. O mecanismo de reescrita Nginx tem uma ampla funcionalidade e é muito fácil de configurar, o que o torna uma ferramenta muito poderosa. Vamos percorrer todo o mecanismo de reescrita neste capítulo.

Outro tópico que vamos explorar neste capítulo é o controle de acesso. Esta é, obviamente, uma função essencial de todo sistema de software que mantém o sistema seguro e confiável. Vamos percorrer os métodos de controle de acesso disponíveis no Nginx e explorar suas sutilezas, e você aprenderá como combiná-los.

O básico do mecanismo de reescrita

O mecanismo de reescrita permite manipular o URI de solicitação de solicitações de entrada.

O mecanismo de reescrita é configurado usando regras de reescrita. As regras de reescrita são usadas quando o URI de solicitação precisa passar por transformação antes do processamento adicional. As regras de reescrita instruem o Nginx a corresponder o URI de solicitação com uma expressão regular e substituir o URI de solicitação por um padrão especificado sempre que uma correspondência for pontuada.

As regras de reescrita podem ser especificadas dentro das seções `server`, `location` e `if` da configuração.

Reescrever mecanismo e controle de acesso

Vamos estudar alguns exemplos de regras de reescrita em ação. Considere um caso simples em que um recurso precisa ser substituído por outro:

```
local / {
    reescrever ^/css/default.css$ /css/styles.css break;
    root /var/www/example.com;
}
```

Com a configuração anterior, cada solicitação para /css/default.css terá seu URI reescrito em /css/styles.css e buscará esse recurso. A diretiva de reescrita especifica um padrão que deve corresponder ao URI de solicitação para cumprir a regra e uma string de substituição que diz como o URI de solicitação deve ficar após a transformação. O terceiro argumento, break, é um atraso que instrui o Nginx a parar de processar as regras de reescrita assim que uma correspondência para essa regra for pontuada.

A configuração anterior também pode ser estendida para trabalhar com vários recursos. Para isso, você precisa usar capturas (colchetes no primeiro argumento) e parâmetros posicionais (variáveis com números que se referem a capturas):

```
local / {
    reescrever ^/styles/(.+).css$ /css/$1.css break;
    root /var/www/example.com;
}
```

Com a configuração anterior, cada solicitação para qualquer arquivo CSS em /styles/ terá seu URI reescrito para o recurso correspondente em /css/.

Nos dois últimos exemplos, usamos o intervalo de interrupção para interromper o processamento das regras de reescrita assim que uma correspondência for encontrada (assumindo que mais regras podem ser adicionadas a essas configurações). Se quisermos combinar esses dois exemplos, precisamos eliminar o intervalo de interrupção e permitir a aplicação em cascata de regras de reescrita:

```
local / {
    reescrever ^/styles/(.+).css$ /css/$1.css;
    reescrever ^/css/default.css$ /css/styles.css;
    root /var/www/example.com;
}
```

Agora, cada solicitação para folhas de estilo em /styles/ será redirecionada para o recurso correspondente em /css/, e /css/default.css será reescrito em /css/styles.css. Uma solicitação para /styles/default.css passará por duas reescritas, pois corresponde sequencialmente a ambas as regras.

Capítulo 4

Observe que todas as transformações de URI são executadas pelo Nginx internamente. Isso significa que para um cliente externo, os URLs originais retornam recursos comuns, assim as configurações anteriores parecerão externamente com uma série de documentos com conteúdo idêntico (ou seja, /css/default.css será idêntico a /css/styles.css).

Este não é um efeito desejável no caso de páginas da web comuns, pois os mecanismos de pesquisa podem penalizar seu site por conteúdo duplicado.

Para evitar esse problema, é necessário substituir as cópias de um recurso por redirecionamentos permanentes para o recurso mestre, conforme mostrado na configuração a seguir:

```
local / {
    reescrever ^/styles/(.+).css$ /css/$1.css permanente;
    root /var/www/example.com;
}
```

Isso funciona bem para seções inteiras de um site:

```
local / {
    reescrever ^/download/(.+$) /media/$1 permanente;
    root /var/www/example.com;
}
```

Também funciona para um host virtual inteiro:

```
servidor {
    ouça 80;
    nome_do_servidor exemplo.com;
    reescrever ^/(.+$) http://www.example.com/$1 permanente;
}
```

A configuração anterior para qualquer URL solicitada realiza um redirecionamento permanente de um domínio de nível superior example.com para o subdomínio www , tornando-o o principal ponto de entrada do site.

A próxima aplicação poderosa das regras de reescrita é traduzir uma URL semântica em uma URL com uma consulta (seção de uma URL após o caractere ?). Essa funcionalidade tem sua aplicação principal em Search Engine Optimization (SEO) e usabilidade de sites, e é impulsionada pela necessidade de obter URLs semânticos para cada recurso e desduplicar o conteúdo.



Você pode encontrar mais informações sobre URLs semânticos em https://en.wikipedia.org/wiki/Semantic_URL.

Reescrever mecanismo e controle de acesso

Consider the following configuration:

```
servidor {
    [...]
    reescreva ^/products/$ /products.php por último;
    reescreva ^/products/(.+)$/products.php?name=$1 último;
    reescrever
    ^/products/(.+)(.+)$ /products.php?name=$1&page=$2 último;
    [...]
}
```

The configuration above transforms URLs that consist of several path sections starting with /products in a URL starting with /products.php and arguments. In this way, it is possible to hide implementation details of users and mechanisms of search and generate semantic URLs.

Observe que the delays of rewrite directives are now defined to last. This makes the Nginx search for a new location for a rewritten URL and process the request with a recently found location.

Now that you have studied some examples of rewrite rules in action, you can learn more about the basic details to master the rewrite rule. The sections that follow examine more thoroughly its syntax and functionality.

Mais sobre regras de reescrita

Now, let's discuss some interesting details of the rewrite rules. Here is the complete syntax of the rewrite directive:

```
reescrever <padrão> <substituição> [<flag>];
```

The first argument of this directive, <pattern>, is a regular expression that needs to match the URI of the request to activate the substitution. The <substituição> argument is a script evaluated when a correspondence is scored and the value produced by the evaluation replaces the URI of the request. Special variables \$1...\$9 can be used to refer to a pattern and its substitution referring to a capture position. The <flag> argument affects the behavior of the rewrite directive. The following table lists all the possible flags for the rewrite directive and their functions:

Bandeira	Função
pausa	Interrompe o processamento de regras de reescrita
último	Interrompe o processamento de regras de reescrita e procura um local para o novo URI de solicitação

Capítulo 4

Bandeira	Função
redirecionar	Retorna um redirecionamento temporário (status HTTP 302) para o novo URI de solicitação
permanente	Retorna um redirecionamento permanente (status HTTP 301) para o novo URI de solicitação

O mecanismo de reescrita faz várias passagens antes que um local para a solicitação seja encontrado e, em seguida, no local da solicitação e nos locais subsequentes para os quais a solicitação é redirecionada (como aqueles que são invocados pela diretiva `error_page`).

As regras de regravação especificadas diretamente na seção do servidor são processadas na primeira passagem, enquanto as regras de regravação no local, if e outras seções dentro do servidor seção são processados em passagens subsequentes. Considere o seguinte exemplo:

```
servidor {
    <regras de reescrita aqui são processadas na primeira passagem>

    local /a {
        <regras de regravação aqui são processadas em passagens subsequentes>;
    }
    local /b {
        <regras de regravação aqui são processadas em passagens subsequentes>;
    }
}
```

Depois que a primeira passagem é concluída, o Nginx procura um local que corresponda ao URI de solicitação reescrito se uma reescrita foi realizada, ou um local que corresponda ao URI de solicitação original (se nenhuma reescrita ocorreu). As passagens subsequentes alteram o URI de solicitação sem alterar o local.

As regras de reescrita em cada passagem são processadas em ordem de aparição. Depois que uma correspondência é pontuada, a substituição é aplicada e o processamento é retomado com regras de reescrita subsequentes - a menos que um atraso seja especificado para interromper o processamento.

Se o URI de solicitação resultante começar com `http://` ou `https://`, ele será tratado como absoluto e o Nginx retornará um redirecionamento temporário (302 "Encontrado") ou permanente (301 "Movido permanentemente") para o local resultante.

Reescrever mecanismo e controle de acesso

Padrões

Agora, vamos voltar ao argumento `<pattern>` e ver como um padrão de correspondência pode ser especificado. A tabela a seguir fornece uma breve visão geral da sintaxe de expressão regular usada em regras de reescrita:

Padrão	Exemplos	Descrição
<code><padrão A></code>	<code>Ab, (aa) (bb)</code>	Segue
<code><padrão B></code>		
<code><padrão A> <padrão B></code>	<code>a b, (aa) (bb)</code>	Alternativo
<code><padrão>?</code>	<code>(.gz)?</code>	Opção
<code><padrão>*</code>	<code>A*, (aa)*</code>	Repetição de <code><padrão></code> de 0 ao <i>infinito</i>
<code><padrão>+</code>	<code>a+, (aa)+</code>	Repetição de <code><padrão></code> de 1 ao <i>infinito</i>
<code><padrão>{n} a{5}, (aa){6}</code>		Repetição de <code><padrão></code> <i>n</i> vezes
<code><padrão>{n,} a{3,}, (aa){7,}</code>	Repetição de <code><padrão></code> de <i>n</i> ao <i>infinito</i>	
<code><padrão>{m} a{,6}, (aa){,3}</code>	Repetição de <code><padrão></code> de 0 a <i>m</i>	
<code><padrão>{n,m} a{5,6}, (aa){1,3}</code>		Repetição de <code><padrão></code> de <i>n</i> para <i>m</i>
<code>(<padrão>)</code>	<code>(aa)</code>	Agrupamento ou captura de parâmetros
<code>.</code>	<code>.+</code>	Qualquer caractere
<code>^</code>	<code>^/índice</code>	Começo de linha
<code>\$</code>	<code>\.php\$</code>	Fim da linha
<code>[<personagens>]</code>	<code>[A-Za-z]</code>	Qualquer caractere do conjunto especificado
<code>[^<personagens>]</code>	<code>[^0-9]</code>	Qualquer caractere fora do conjunto especificado

Os padrões são listados em ordem crescente de prioridade. Ou seja, o padrão `a|bb` será interpretado como `a(a|b)b`, enquanto o padrão `a{5}aa{6}` será interpretado como `(a{5})(a{6})` e assim por diante.

Para especificar caracteres que fazem parte da sintaxe de expressão regular, você pode usar o caractere de barra invertida `\`, por exemplo, `*` corresponderá a um asterisco `*`, `\.` corresponderá a um caractere de ponto, `\t` corresponderá ao próprio caractere de barra invertida e `\{` corresponderá a uma chave de abertura `{`.

Capítulo 4

Mais informações sobre a sintaxe de expressões regulares em regras de reescrita podem ser encontradas no site do PCRE www.pcre.org.

Capturas e parâmetros posicionais

As capturas são designadas com colchetes e marcam seções de URLs correspondentes que precisam ser extraídas. Os parâmetros posicionais referem-se a substrings das URLs correspondentes extraídas pela captura correspondente, ou seja, se o padrão for o seguinte:

```
^/usuários/(.+)/(.+)/$
```

Além disso, se o URL da solicitação for assim:

```
/users/id/23850/
```

Os parâmetros posicionais \$1 e \$2 serão avaliados para id e 23850, respectivamente.

Os parâmetros posicionais podem ser usados em qualquer ordem dentro da string de substituição e é assim que você os conecta com o padrão de correspondência.

Outras funcionalidades do mecanismo de reescrita

O mecanismo de reescrita também pode ser usado para executar outras tarefas:

- Atribuindo variáveis
- Avaliando predicados usando a diretiva if
- Respondendo com o código de status HTTP especificado

Uma combinação dessas operações e regras de reescrita pode ser executada a cada passagem do mecanismo de reescrita. Observe que , se as seções forem locais separados, ainda é possível que o local seja alterado na passagem de reescrita do local.

Atribuindo variáveis

As variáveis podem ser atribuídas usando a diretiva set :

```
set $fruit "maçã";
```

Os valores das variáveis podem ser scripts com texto e outras variáveis:

```
set $path "static/$arg_filename";
```

Uma vez definidas na fase de reescrita, as variáveis podem ser usadas em qualquer diretiva no restante do arquivo de configuração.

Reescrever mecanismo e controle de acesso

Avaliando predicados usando seções if

Você provavelmente deduziu do título que se as seções fazem parte do mecanismo de reescrita. Isso é verdade. As seções if podem ser usadas para aplicar condicionalmente as regras de reescrita selecionadas:

```
if ($request_method = POST)      {  
    reescrever ^/media/$ /upload por último;  
}
```

Na configuração anterior, qualquer tentativa de fazer uma solicitação POST para a URL / media/ resultará em regravação para a URL / upload, enquanto solicitações com outros métodos para a mesma URL não resultarão em regravações. Várias condições também podem ser combinadas. Vejamos o seguinte código:

```
definir $c1 "";  
definir $c2 "";  
  
if ( $request_method = POST ) set $c1      {  
    "sim";  
}  
  
if ( $scheme = "https" ) set $c2      {  
    "sim";  
}  
  
set $e "${c1}_${c2}";  
  
if ( $and = "yes_yes" ) reescrever  {  
    [...];  
}
```

A configuração anterior aplica a regravação somente quando ambas as condições são atendidas, ou seja, quando o método de solicitação é POST e o esquema de URL de solicitação é https.

Agora que você sabe como usar a seção if , vamos falar sobre seus efeitos colaterais. Lembre-se de que as condições nas diretivas if são avaliadas durante o processamento da diretiva de reescrita . O que significa é que quando a seção if contém diretivas que não fazem parte do mecanismo de reescrita, o comportamento da seção if se torna não intuitivo. Isso foi discutido no Capítulo 1, *Introdução ao Nginx*.

Considere a seguinte configuração:

```
if ( $request_method = POST ) set $c1      {  
    "sim";  
    proxy_pass http://localhost:8080;
```

```

    }

if ( $scheme = "https" ) set $c2      {
    "sim";
    gzip em
}

```

Cada seção if individual contém um conjunto atômico de configurações de configuração.

Suponha que o Nginx receba uma solicitação POST com o esquema de URL https de modo que

ambas as condições sejam avaliadas como verdadeiras. Todas as diretivas definidas serão

processadas corretamente pelo mecanismo de reescrita e serão atribuídas aos valores apropriados.

No entanto, o Nginx não pode mesclar outras configurações de configuração e não pode ter várias configurações ativas ao mesmo

Quando o processamento de reescrita é concluído, o Nginx simplesmente alterna a configuração para

a última seção if com suas condições avaliadas como verdadeiras. Por isso, na configuração anterior,

a compactação será ativada, mas a solicitação não será processada de acordo com a diretiva

proxy_pass . Isso não é algo que você pode esperar.

Para evitar esse comportamento não intuitivo, siga as seguintes práticas recomendadas:

- Minimize o uso da diretiva if
- Combine as avaliações if usando a diretiva set
- Execute ações apenas na última seção if .

Respondendo com um código de status HTTP especificado

Se uma resposta definida com um código de status HTTP especificado for necessária em um

determinado local, você poderá usar a diretiva return para habilitar esse comportamento e especificar

o código de status, um corpo de resposta ou um URL de redirecionamento. Vejamos o seguinte código:

```

local / {
    return 301 "https://www.example.com$uri";
}

```

A configuração anterior executará um redirecionamento permanente (301) para a parte segura do domínio www.example.com e o caminho de URI idêntico ao caminho de URI na solicitação original.

Assim, o segundo argumento da diretiva return será tratado como um URI de redirecionamento. Os outros códigos de status que tratam o segundo argumento do retorno diretiva como um URI de redirecionamento são 302, 303 e 307.

 Executar um redirecionamento com a diretiva return é muito mais rápido do que com a diretiva rewrite, porque não executa nenhuma expressão regular. Use a diretiva return em sua configuração ao invés da diretiva rewrite sempre que possível.

Reescrever mecanismo e controle de acesso

O código de status 302 é bastante comum, então a diretiva return possui uma sintaxe simplificada para redirecionamentos temporários:

```
local / {
    return "https://www.example.com$uri";
}
```

Como você pode ver, se a diretiva return tiver um único argumento, ela será tratada como URI de redirecionamento e fará com que o Nginx execute um redirecionamento temporário. Esse argumento deve começar em http:// ou https:// para acionar tal comportamento.

A diretiva return pode ser usada para retornar uma resposta com um corpo especificado. Para acionar tal comportamento, o código de status deve simplesmente ser diferente de 301, 302, 303 ou 307. O segundo argumento da diretiva return especifica o conteúdo do corpo da resposta:

```
local /desativado {
    default_type text/plain;
    retorno 200 "OK";
}
```

A configuração anterior retornará o status HTTP 200 (OK) com o corpo de resposta especificado. Para confirmar o processamento correto do conteúdo do corpo, definimos o tipo de conteúdo da resposta como text/plain usando a diretiva default_type .

Controle de acesso

As restrições de controle de acesso são essenciais para a operação do dia-a-dia. O Nginx inclui um grupo de módulos que permitem permitir ou negar acesso dependendo de várias condições. O Nginx nega o acesso a um recurso retornando um status 403 (HTTP proibido) ou 401 (não autorizado) se o acesso ao recurso exigir autenticação. Esse código de status 403 (Proibido) pode ser interceptado e personalizado usando a diretiva error_page .

Restringindo o acesso por endereço IP

O Nginx permite que você permita ou negue o acesso a um host virtual ou local por endereço IP. Para isso, você pode usar as diretivas allow e deny. Possuem o seguinte formato:

```
permitir <endereço IP> | <endereço IP>/<tamanho do prefixo> | tudo;
negar <endereço IP> | <endereço IP>/<tamanho do prefixo> | tudo;
```

Especificar um endereço IP permite ou nega acesso a um único endereço IP em um local, enquanto especifica um endereço IP com tamanho preix (por exemplo, 192.168.0.0/24 ou 200.1:980::/32) permite ou nega acesso a um intervalo de endereços IP.

Capítulo 4

As diretivas de permissão e negação são processadas em ordem de aparição em um local. O endereço IP remoto de um cliente solicitante é comparado com o argumento de cada diretiva. Uma vez que uma diretiva de permissão com um endereço correspondente é encontrada, o acesso é imediatamente permitido. Uma vez que uma diretiva de negação com um endereço correspondente é encontrada, o acesso é imediatamente negado. Uma vez que o Nginx atinge a diretiva permitir ou negar com o all argumento, o acesso é imediatamente permitido ou negado, independentemente do endereço IP do cliente.

Isso, obviamente, permite alguma variação. Aqui estão alguns exemplos simples:

```
servidor {
    negar 192.168.1.0/24;
    permitir todos;
    [...]
}
```

A configuração anterior faz com que o Nginx negue o acesso aos endereços IP 192.168.1.0 a 192.168.1.255, enquanto permite o acesso a todos os outros. Isso acontece porque a diretiva deny é processada primeiro e, se houver correspondência, é aplicada imediatamente. Todo o servidor será proibido para endereços IP especificados.

```
servidor {
    [...]
    local /administrador {
        permitir 10.132.3.0/24;
        negar tudo;
    }
}
```

A configuração anterior faz com que o Nginx permita o acesso ao local /admin apenas para endereços IP no intervalo 10.132.3.0 a 10.132.3.255. Supondo que este intervalo de endereços IP corresponda a algum grupo privilegiado de usuários, esta configuração faz todo o sentido, pois somente eles podem acessar a área administrativa desta aplicação web.

Agora, podemos melhorar isso e tornar a configuração mais complicada. Suponha que mais redes precisem acessar a interface administrativa deste aplicativo da web, enquanto o endereço IP 10.132.3.55 precisa ter o acesso negado por motivos técnicos ou administrativos. Então, podemos estender a configuração anterior da seguinte forma:

```
servidor {
    [...]
    local /administrador {
        permitir 10.129.1.0/24;
        permitir 10.144.25.0/24;
```

Reescrever mecanismo e controle de acesso

```

negar 10.132.3.55;
permitir 10.132.3.0/24;
negar tudo;
}
}

```

Como você pode ver, as diretivas permitir e negar são bastante intuitivas de usar. Use-os desde que a lista de endereços IP para correspondência não seja muito longa. O Nginx processa essas diretivas em ordem sequencial, de modo que o tempo necessário para verificar o endereço IP do cliente em relação à lista é em média proporcional ao tamanho da lista, não importando com qual diretiva o endereço corresponde.

Se você precisar comparar o endereço IP do cliente com uma lista maior de endereços, considere usar a diretiva geo .

Usando a diretiva geo para restringir o acesso por endereço de IP

Com a diretiva geo , você pode transformar um endereço IP em um valor literal ou numérico que pode ser usado posteriormente para acionar algumas ações durante o processamento de uma solicitação.

A diretiva geo tem o seguinte formato:

```
geo [$<variável de origem>] $<variável de destino> { <mapeamento de endereço> }
```

Se a variável de origem for omitida, a variável \$remote_addr será usada. O mapeamento de endereço é uma lista de pares chave/valor, separados por espaços em branco. Uma chave geralmente é um endereço IP ou um endereço IP com um tamanho de preix especificando uma sub-rede. Um valor é uma cadeia de caracteres arbitrária ou um número. Vejamos o seguinte código:

```

geo $admin_access {
    predefinição           negar;
    10.129.1.0/24          permitir;
    10.144.25.0/24         permitir;
    10.132.3.0/24          permitir;
}

```

O valor da variável de origem é usado como chave para pesquisar uma entrada no mapeamento de endereço. Uma vez encontrada, a variável de destino é atribuída ao valor pesquisado. Caso contrário, o valor padrão é usado.

Com a configuração anterior, será atribuído à variável \$admin_access o valor allow se o endereço IP do cliente remoto for originário da sub-rede 10.129.1.0/24, 10.144.25.0/24 ou 10.132.3.0/24, caso contrário.



A diretiva geo constrói uma estrutura de dados sucinta eficiente para pesquisar os valores por endereço IP na memória. Ele pode lidar com centenas de milhares de endereços IP e sub-redes. Para acelerar o tempo de inicialização, especifique os endereços IP para a diretiva geo em ordem crescente, por exemplo, 1.xxx a 10.xxx, 1.10.xx a 1.30.xx

A seção de mapeamento de endereço pode conter diretivas que afetam o comportamento de geo mapeamento de endereços. A tabela a seguir lista essas diretivas junto com suas funções:

Diretiva	Função
predefinição	Especifica um valor que é retornado quando nenhuma correspondência é encontrada no mapeamento de endereço IP.
procurador	Especifica o endereço de um servidor proxy. Se uma solicitação se originar de um endereço especificado por uma das diretivas de proxy, geo usará o último endereço do cabeçalho "X-Forwarded-For" e não da variável de origem.
proxy_recursive	Se uma solicitação se originar de um endereço especificado por uma das diretivas de proxy, geo processará endereços no cabeçalho "X-Forwarded-For" da direita para a esquerda em busca de um endereço fora da lista especificada pela diretiva de proxy. Em outras palavras, esta diretiva faz com que o geo faça um esforço maior na busca de um endereço IP real.
gamas	Habilita os intervalos de endereços IP na lista de mapeamento.
excluir	Remove a sub-rede especificada do mapeamento.

Vamos dar uma olhada em alguns exemplos.

Considere que o Nginx recebe tráfego HTTP de um平衡ador de carga de nível de aplicativo ou um proxy de entrada localizado no IP 10.200.0.1. Como todas as solicitações se originarão desse IP, precisamos examinar o cabeçalho "X-Forwarded-For" para obter o endereço IP real do cliente.

Precisamos então alterar a configuração anterior da seguinte forma:

```
geo $ exemplo {
    predefinição negar;
    procuração 10.200.0.1;
    10.129.1.0/24     permitir;
    10.144.25.0/24    permitir;
    10.132.3.0/24    permitir;
}
```

Reescrever mecanismo e controle de acesso

Se o servidor estiver atrás de uma cadeia de proxies, o endereço IP real pode ser obtido especificando a diretiva proxy_recursive e listando todos os proxies na cadeia:

```
geo $ exemplo {
    predefinição negar;
    procuração 10.200.0.1;
    procuração 10.200.1.1;
    procuração 10.200.2.1;
    proxy_recursive;
    10.129.1.0/24 permitir;
    10.144.25.0/24 permitir;
    10.132.3.0/24 permitir;
}
```

No exemplo anterior, os proxies têm endereços IP 10.200.0.1, 10.200.1.1 e 10.200.2.1. A ordem em que os endereços são listados não é importante, pois o Nginx simplesmente itera sobre os endereços especificados no cabeçalho "X-Forwarded-For" da direita para a esquerda e verifica sua presença no bloco geográfico . O primeiro endereço fora da lista de proxy se torna o endereço IP real do cliente.

Se os endereços IP precisarem ser especificados como intervalos ou além de sub-redes, você poderá habilitar isso especificando a diretiva ranges :

```
geo $ exemplo {
    predefinição negar;
    gamas;
    10.129.1.0-10.129.1.255 permitir;
    10.144.25.0-10.144.25.255 permititem;
    10.132.3.0/24 permitir;
}
```

Finalmente, com a ajuda da diretiva delete , podemos definir o mapeamento do endereço IP que nos permite implementar um procedimento de controle de acesso análogo para permitir e negar diretivas em maior escala:

```
geo $admin_access {
    predefinição negar;
    10.129.1.0/24 permititem;
    10.144.25.0/24 permititem;
    10.132.3.0/24 permititem;
    excluir 10.132.3.55;
}
```

Para colocar essa configuração em ação, precisamos usar a seção if para proibir a solicitação de que o endereço IP desses clientes não caia no intervalo de permissão da diretiva geo :

```

servidor {
    [...]
    geo $admin_access {
        predefinição             negar;
        10.129.1.0/24            permitir;
        10.144.25.0/24           permitir;
        10.132.3.0/24            permitir;
        excluir 10.132.3.55;
    }

    local /administrador {
        if($admin_access != permitir) {
            retornar 403;
        }
        [...]
    }
}

```

Como você pode ver, a diretiva geo é uma ferramenta poderosa e muito escalável, e a restrição de acesso é uma das muitas aplicações em que ela pode ser colocada.

Usando autenticação básica para restrição de acesso

Você pode configurar o Nginx para permitir acesso apenas aos usuários que podem fornecer a combinação correta de nome de usuário e senha. A verificação de nome de usuário/senha é habilitada usando a diretiva auth_basic :

```
auth_basic <nome do domínio> | fora;
```

Nome do território especifica o nome de um território (uma área autenticada). Esse argumento geralmente é definido como uma string que ajuda os usuários a identificar a área que estão tentando acessar (por exemplo , área administrativa, correio da Web e assim por diante). Essa string será passada para o navegador e exibida na caixa de diálogo de entrada de nome de usuário/senha. Além do nome do realm, você precisa especificar um arquivo contendo um banco de dados do usuário usando a diretiva auth_basic_user_file :

```
auth_basic_user_file <caminho para um arquivo>;
```

Reescrever mecanismo e controle de acesso

Este arquivo deve conter informações de autenticação com um nome de usuário e uma senha em cada linha:

```
nome de usuário1:senha_criptografada1
nome_do_usuário2:senha_criptografada2
nome de usuário3:senha_criptografada3
username4:encrypted_password4
nome de usuário5:senha_criptografada5
```

Este arquivo presumivelmente deve ser colocado fora da raiz do documento de qualquer site que você esteja hospedando. Os direitos de acesso devem ser configurados de forma que o Nginx possa apenas ler este arquivo, nunca escrever ou executar.

As senhas devem ser criptografadas usando um dos seguintes algoritmos:

Algoritmos	Comentários
CRIPTA	Algoritmo de criptografia de senha baseado em Unix DES
SSHA	Algoritmo de Hash Seguro Salgado 1
<i>Obsoleto: Não use</i>	
MD5	Algoritmo Message Digest 5
BEBER	Algoritmo de Hash Seguro Sem Sal 1

O arquivo de senhas pode ser gerenciado usando o utilitário htpasswd do servidor web Apache. aqui estão alguns exemplos:

Instrução	Comando
Crie um arquivo de senha e adicione o usuário john ao arquivo de senha	\$ htpasswd -b -d -c /etc/nginx/auth.d/auth.pwd john test
Adicionar usuário thomas para o arquivo de senha	\$ htpasswd -b -d /etc/nginx/auth.d/auth.pwd thomas test
Substituir a senha de John	\$ htpasswd -b -d /etc/nginx/auth.d/auth.pwd john teste
Remover usuário john do arquivo de senha	\$ htpasswd -D /etc/nginx/auth.d/auth.pwd john

Capítulo 4

A opção **-d** força a criptografia de senhas usando o algoritmo CRYPT, que é relativamente menos seguro que o SSHA (Salted SHA). Para criptografar senhas usando SSHA e obter maior segurança de suas senhas, você pode usar o utilitário slappasswd do pacote slapd :

```
$ sudo apt-get install slapd
$ slappasswd -s teste
{SSHA}ZVG7uXWXQVpITwohT0F8yMDGWs0AbYd3
```

Copie a saída de slappasswd no arquivo de senha. O arquivo de senha agora se parece com isso:

```
john:{SSHA}ZVG7uXWXQVpITwohT0F8yMDGWs0AbYd3
```

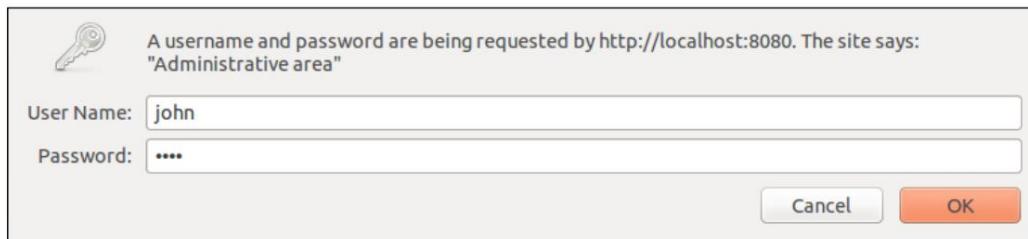
Isso pode ser ainda mais automatizado usando o comando echo :

```
echo "john:"$(slappasswd -s test) > /etc/nginx/auth.d/auth.pwd
```

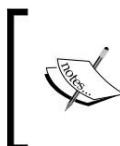
Quando o arquivo de senha estiver pronto, podemos configurar a autenticação de senha:

```
local /administrador {
    auth_basic "Área administrativa";
    auth_basic_user_file /etc/nginx/auth.d/auth.pwd;
    [...]
}
```

A autenticação de senha agora está habilitada; você pode navegar até o local /admin e ver o prompt de senha:



O acesso ao recurso protegido será concedido somente quando uma combinação válida de nome de usuário e senha for inserida no prompt de senha.



O Nginx lê e analisa o arquivo de senha toda vez que uma solicitação para recursos protegidos é feita. Isso é escalável apenas quando o número de entradas no arquivo de senha não excede algumas centenas.

Reescrever mecanismo e controle de acesso

Autenticação de usuários com uma subsolicitação

A autenticação do usuário pode ser delegada a outro servidor web usando o módulo de solicitação de autenticação. Este módulo deve primeiro ser habilitado no estágio de configuração do código-fonte usando a opção de linha de comando `--with-http_auth_request_module`:

```
$ ./configure --with-http_auth_request_module
$ fazer
$ fazer instalar
```

Agora o módulo auth_request está pronto para ser usado. A delegação pode ser configurada da seguinte forma:

```
local/exemplo {
    auth_request /auth;
    [...]
}

local = /auth{
    interno;
    proxy_pass http://backend;
    proxy_set_header Content-Length "";
    proxy_pass_request_body desativado;
}
```

Com a configuração anterior, o Nginx executará uma subsolicitação para location / auth. Esse local passará a subsolicitação para um aplicativo da Web externo (usando a diretiva proxy_pass). Como a solicitação original pode ter um corpo de solicitação que o aplicativo de autenticação não espera, nós a descartamos especificando proxy_pass_request_body off e anulando o cabeçalho "Content-Length" usando proxy_set_header.

Para responder a uma subsolicitação emitida pelo módulo de solicitação de autenticação, é necessário criar um aplicativo que analise os dados da solicitação original e responda com status HTTP 401 (Não autorizado) ou 403 (Proibido) para bloquear o acesso, e com sucesso Status HTTP 200 a 299 para permitir o acesso. Aqui está um exemplo de um aplicativo desse tipo em node.js:

```
var http          = exigir('http');
foi expresso     = exigir('expresso')
var cookieParser = require('cookie-parser')

var app = express()
```

```

app.use(cookieParser())

app.get('/auth', function(req, res) { if(req.cookies.uid) {

    res.sendStatus (200);

} senão {
    res.sendStatus (403);

}})

app.listen(3000)

```

Este aplicativo permite o acesso desde que um uid de nomes de cookies esteja presente e proíbe o acesso de outra forma.

Para executar este aplicativo, crie um diretório, crie um arquivo chamado auth.js nesse diretório e coloque o código-fonte anterior nesse arquivo. Depois disso, instale os módulos necessários express e cookie-parser usando npm:

\$ npm instalar analisador de cookies expresso

Depois disso, você pode executar o aplicativo:

\$ node auth.js

O aplicativo começará a escutar na porta 3000. A seguinte configuração Nginx pode ser usada para testar o aplicativo:

```

local /exemplo { auth_request /
    auth;
}

local = /auth { interno;
    proxy_pass http://
    localhost:3000; proxy_set_header Content-Length
    ""; proxy_pass_request_body desativado;

}

```

A subsolicitação será delegada à porta 3000 do host, onde o Nginx está sendo executado, e o aplicativo responderá a essa solicitação.

Se o aplicativo precisar examinar o URI de solicitação original, ele poderá ser passado usando a diretiva proxy_set_header :

```
proxy_set_header X-Auth-URI $request_uri;
```

Reescrever mecanismo e controle de acesso

O endereço IP original e outros parâmetros de solicitação originais podem ser passados para o aplicativo de autenticação da mesma maneira.

É assim que uma lógica de autenticação mais sofisticada pode ser implementada no Nginx. Se você fizer com que o aplicativo sempre responda com status HTTP 200, ele poderá ser usado para outros fins que não a autenticação, como log ou injeção de dados.

Combinando vários métodos de restrição de acesso

Vários métodos de restrição de acesso podem ser combinados. Para isso, eles devem estar configurados e habilitados. Por padrão, todos os métodos de restrição de acesso configurados devem ser satisfeitos para permitir a solicitação. Se algum dos métodos de restrição de acesso não for satisfeito, o Nginx rejeita a solicitação com o status 403 Forbidden HTTP.

Esse comportamento pode ser alterado usando a diretiva `satisfazer` :

`satisfazer todos | algum;`

Especificar `satisfazer qualquer um` em um local faz com que o Nginx aceite a solicitação se qualquer um dos métodos de restrição de acesso habilitados for satisfeito, enquanto especificar `satisfazer todos` (o padrão) faz com que o Nginx aceite a solicitação somente se todos os métodos de restrição de acesso habilitados forem satisfeitos. Para demonstrar como funciona, vamos estender o exemplo anterior:

```
servidor {
    [...]
    local /administrador {
        auth_basic "Área administrativa";
        auth_basic_user_file /etc/nginx/auth.d/admin.users;
        permitir 10.132.3.0/24;
        negar tudo;
        satisfazer qualquer;
    }
}
```

Essa configuração habilita e configura a autenticação de senha e a restrição de endereço IP. Com `satisfazer` definido como `qualquer`, um usuário precisa inserir uma combinação correta de nome de usuário/senha ou originar do intervalo de endereços IP 10.132.3.0 a 10.132.3.255. Isso torna os usuários desta rede de alguma forma mais confiáveis, pois eles não precisam digitar seu nome de usuário e senha para acessar a área administrativa.

Resumo

Neste capítulo, você aprendeu como usar o mecanismo de reescrita e as funções de controle de acesso. Essas são ferramentas essenciais de todo webmaster e engenheiro de confiabilidade do site. A excelência na configuração e utilização desses recursos o ajudará a resolver os problemas do dia-a-dia de forma mais eficiente.

No próximo capítulo, falaremos sobre como gerenciar o tráfego de entrada e saída. Você aprenderá como definir várias limitações no tráfego de entrada, como configurar o upstream e como aplicar várias opções ao tráfego de saída.

Machine Translated by Google

5

Gerenciamento de entrada e Tráfego de saída

A Internet é um meio aberto onde é fácil e barato usar os recursos de outra pessoa. O baixo custo de uso torna os sistemas vulneráveis a abusos intencionais e não intencionais e picos de uso de recursos. A Internet moderna está cheia de perigos, como bots, rastreadores abusivos, negação de serviço e ataques distribuídos de negação de serviço.

É aqui que o Nginx é útil, com uma variedade de recursos para gerenciamento de tráfego de entrada e saída que permite que você mantenha o controle da qualidade do seu serviço da web.

Neste capítulo, você aprenderá:

- Como aplicar várias limitações ao tráfego de entrada
- Como configurar upstreams
- Como usar várias opções para gerenciamento de conexão de saída

Gerenciando o tráfego de entrada

O Nginx tem várias opções para gerenciar o tráfego de entrada. Isso inclui o seguinte:

- Limitando a taxa de solicitação
- Limitando o número de conexões simultâneas
- Limitando a taxa de transferência de uma conexão

Esses recursos são muito úteis para gerenciar a qualidade do seu serviço web e para prevenir e mitigar abusos.

Gerenciando o tráfego de entrada e saída

Limitando a taxa de solicitação

O Nginx possui um módulo embutido para limitar a taxa de solicitação. Antes de habilitá-lo, você precisa configurar um segmento de memória compartilhada (também conhecido como *zona*) no http seção usando a diretiva `limit_req_zone`. Esta diretiva tem o seguinte formato:

```
limit_req_zone <chave> zona=<nome>:<tamanho> taxa=<taxa>;
```

O argumento `<key>` especifica uma única variável ou um script (desde a versão 1.7.6) ao qual o limite de taxa está vinculado. Em termos simples, especificando a `<key>` argumento, você está criando um número de pequenos pipes para cada valor do `<key>` argumento avaliado em tempo de execução, cada um deles com sua taxa de requisição limitada com `<rate>`. Cada solicitação de um local onde esta zona é utilizada será submetida à tubulação correspondente e se o limite de taxa for atingido, a solicitação será atrasada para que o limite de taxa dentro da tubulação seja satisfeito.

O argumento `<name>` define o nome da zona e o argumento `<size>` define o tamanho da zona. Considere o seguinte exemplo:

```
http{
    limit_req_zone $remote_addr zone=rate_limit1:12m taxa=30r/m;
    [...]
}
```

No código anterior, definimos uma zona chamada primária que tem 12 MB de tamanho e um limite de taxa de 30 solicitações por minuto (0,5 solicitação por segundo). Usamos a variável `$remote_addr` como chave. Essa variável avalia em um valor simbólico do endereço IP de onde veio a solicitação, que pode ocupar até 15 bytes por endereço IPv4 e ainda mais por endereço IPv6.

Para conservar o espaço ocupado pela chave, podemos usar a variável `$binary_remote_addr` que avalia em um valor binário do endereço IP remoto:

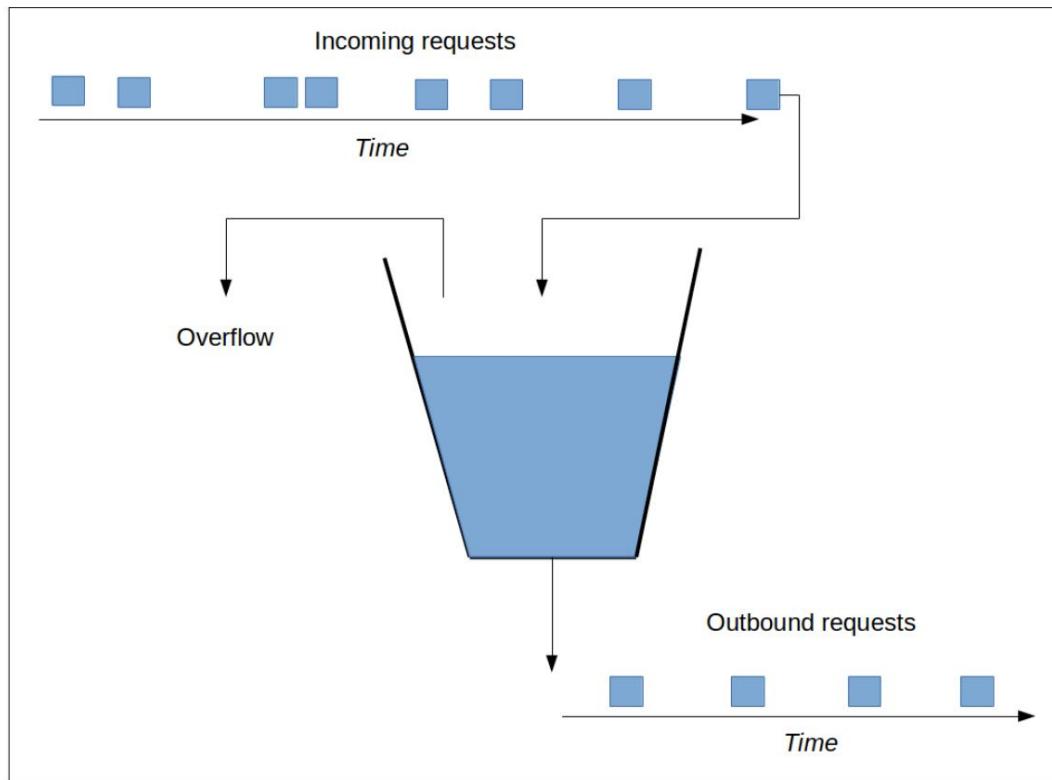
```
http{
    limit_req_zone $binary_remote_addr zone=rate_limit1:12m
    taxa=30r/m;
    [...]
}
```

Para habilitar a limitação da taxa de solicitação em um local, use a diretiva `limit_req`:

```
local / {
    limit_req zona=rate_limit1;
}
```

capítulo 5

Uma vez que uma solicitação é roteada para a localização /, um estado de limitação de taxa será recuperado do segmento de memória compartilhada especificado e o Nginx aplicará o *Leaky Bucket* algoritmo para gerenciar a taxa de solicitação, conforme mostrado na figura a seguir:



O algoritmo Leaky Bucket

De acordo com este algoritmo, as requisições de entrada podem chegar a uma taxa arbitrária, mas a taxa de requisições de saída nunca será maior que a especificada. As solicitações recebidas "inutilizam o bucket" e, se o "bucket" transbordar, solicitações excessivas obterão a resposta de status HTTP 503 (Serviço Temporariamente Indisponível).

Gerenciando o tráfego de entrada e saída

Limitando o número de conexões simultâneas

Embora muito prática, a limitação da taxa de solicitação não pode ajudar a mitigar abusos em caso de solicitações de longa duração, como uploads e downloads longos.

Nessa situação, é útil limitar o número de conexões simultâneas.

Em particular, é vantajoso limitar o número de conexões simultâneas de um único endereço IP.

A habilitação do limite de conexões simultâneas começa com a configuração de um segmento de memória compartilhada (uma zona) para armazenar informações de estado, assim como ao limitar a taxa de solicitação. Isso é feito na seção http usando o `limit_conn_zone` diretiva. Essa diretiva é semelhante à diretiva `limit_req_zone` e tem o seguinte formato:

```
limit_conn_zone <chave> zona=<nome>:<tamanho>;
```

No comando anterior, o argumento `<key>` especifica uma única variável ou um script (desde a versão 1.7.6) ao qual o estado de limitação da conexão está vinculado. O `<nome>` argumento define o nome da zona e o argumento `<size>` define o tamanho da zona. Considere o seguinte exemplo:

```
http{
    limit_conn_zone $remote_addr zone=conn_limit1:12m;
    [...]
}
```

Para conservar o espaço ocupado pela chave, podemos usar novamente a variável `$binary_remote_addr`. Ele avalia em um valor binário do endereço IP remoto:

```
http{
    limit_conn_zone $binary_remote_addr zone=conn_limit1:12m;
    [...]
}
```

Para habilitar a limitação de conexão simultânea em um local, use o `limit_conn` diretiva:

```
local/baixar {
    limit_conn conn_limit1 5;
}
```

O primeiro argumento da diretiva `limit_conn` especifica a zona usada para armazenar informações de estado de limitação de conexão, e o segundo argumento é o número máximo de conexões simultâneas.

Para cada conexão com uma solicitação ativa roteada para location/download, o argumento <key> é avaliado. Se o número de conexões simultâneas compartilhando o mesmo valor da chave ultrapassar 5, o servidor responderá com status HTTP 503 (Serviço Temporariamente Indisponível).

 Observe que o tamanho do segmento de memória compartilhada que o limit_conn_ A diretiva de zona aloca é fixa. Quando o segmento de memória compartilhada alocado fica cheio, o Nginx retorna o status HTTP 503 (Serviço Temporariamente Indisponível). Portanto, você precisa ajustar o tamanho do segmento de memória compartilhada para levar em conta o tráfego de entrada potencial do seu servidor.

Limitando a taxa de transferência de uma conexão

A taxa de transferência de uma conexão também pode ser limitada. O Nginx tem várias opções para esse fim. A diretiva limit_rate limita a taxa de transferência de uma conexão em um local ao valor especificado no primeiro argumento:

```
local/baixar {
    taxa_limite 100k;
}
```

A configuração anterior limitará a taxa de download de qualquer solicitação de localização/download a 100 KBps. O limite de taxa é definido por solicitação. Portanto, se um cliente abrir várias conexões, a taxa total de download será maior.

Definir o limite de taxa para 0 desativa o limite de taxa de transferência. Isso é útil quando um determinado local precisa ser excluído da restrição de limite de taxa:

```
servidor {
    [...]
    taxa_limite 1m;

    localização/rápido {
        taxa_limite 0;
    }
}
```

A configuração anterior limita a taxa de transferência de cada solicitação para um determinado host virtual a 1 MBps, exceto para local/rápido, onde a taxa é ilimitada.

Gerenciando o tráfego de entrada e saída

A taxa de transferência também pode ser limitada definindo o valor da variável `$limit_rate`. Esta opção pode ser usada elegantemente quando a limitação de taxa precisa ser habilitada em uma condição específica:

```
if ($lento) {
    defina $limit_rate 100k;
}
```

Há também a opção de adiar a restrição de taxa até que uma certa quantidade de dados seja transferida. Isso pode ser feito usando a diretiva `limit_rate_after`:

```
local/mídia {
    taxa_limite 100k;
    taxa_limite_após 1m;
}
```

A configuração anterior aplicará o limite de taxa somente após o primeiro megabyte de dados de solicitação ter sido enviado. Esse comportamento é útil, por exemplo, ao transmitir vídeo, pois a parte inicial do fluxo geralmente é pré-bufferizada pelo player de vídeo.

Retornar a parte inicial mais rapidamente melhora o tempo de inicialização do vídeo sem obstruir a largura de banda de E/S do disco do servidor.

Aplicando várias limitações

As limitações descritas na seção anterior podem ser combinadas para produzir estratégias de gerenciamento de tráfego mais sofisticadas. Por exemplo, você pode criar duas zonas para limitar o número de conexões simultâneas com diferentes variáveis e aplicar vários limites de uma só vez:

```
http{
    limit_conn_zone $binary_remote_addr zone=conn_limit1:12m;
    limit_conn_zone $server_name zona=conn_limit2:24m;
    [...]
    servidor {
        [...]
        local/baixar {
            limit_conn conn_limit1 5;
            limit_conn conn_limit2 200;
        }
    }
}
```

A configuração anterior limitará o número de conexões simultâneas por endereço IP a cinco; ao mesmo tempo, o número total de conexões simultâneas por host virtual não excederá 200.

Gerenciando o tráfego de saída

O Nginx também possui uma variedade de opções para gerenciamento de tráfego de saída:

- Distribuindo conexões de saída entre vários servidores
- Configurando servidores de backup
- Habilitando conexões persistentes com servidores back-end
- Limitar a taxa de transferência durante a leitura de servidores de back-end

Para habilitar a maioria dessas funções, a primeira coisa que você precisa é declarar seus servidores upstream explicitamente.

Declarando servidores upstream

Nginx permite que você declare servidores upstream explicitamente. Você pode então referir-se a eles várias vezes como uma única entidade de qualquer parte da configuração http . Se a localização do seu servidor ou servidores mudar, não há necessidade de percorrer toda a configuração e ajustá-la. Se novos servidores ingressarem em um grupo, ou servidores existentes deixarem um grupo, é necessário apenas ajustar a declaração e não o uso.

Um servidor upstream é declarado na seção upstream :

```
http{
    back-end upstream {
        servidor server1.example.com;
        servidor server2.example.com;
        servidor server3.example.com;
    }
    [...]
}
```

A seção upstream só pode ser especificada dentro da seção http . A configuração anterior declara um upstream lógico denominado back- end com três servidores físicos. Cada servidor é especificado usando a diretiva do servidor . A diretiva do servidor tem a seguinte sintaxe:

```
servidor <endereço> [<parâmetros>];
```

O parâmetro <address> especifica um endereço IP ou um nome de domínio de um servidor físico. Se um nome de domínio for especificado, ele será resolvido no momento da inicialização e o endereço IP resolvido será usado como o endereço de um servidor físico. Se o nome de domínio for resolvido em vários endereços IP, uma entrada separada será criada para cada um dos endereços IP resolvidos. Isso equivale a especificar uma diretiva de servidor para cada um desses endereços.

Gerenciando o tráfego de entrada e saída

O endereço pode conter especificação de porta opcional, por exemplo, server1.example.com: 8080. Se esta especificação for omitida, a porta 80 será usada. Vejamos um exemplo de declaração upstream:

```
upstream numérico e simbólico {  
    servidor servidor.exemplo.com:8080;  
    servidor 127.0.0.1;  
}
```

A configuração anterior declara um upstream denominado numeric-and-symbolic. O primeiro servidor na lista de servidores tem um nome simbólico e sua porta mudou para 8080. O segundo servidor possui o endereço numérico 127.0.0.1 que corresponde ao host local e a porta é 80.

Vejamos outro exemplo:

```
upstream somente numérico {  
    servidor 192.168.1.1;  
    servidor 192.168.1.2;  
    servidor 192.168.1.3;  
}
```

A configuração anterior declara um upstream denominado numeric-only, que consiste em três servidores com três endereços IP numéricos diferentes escutando na porta padrão.

Considere o seguinte exemplo:

```
mesmo host upstream {  
    servidor 127.0.0.1:8080;  
    servidor 127.0.0.1:8081;  
}
```

A configuração anterior declara um upstream chamado same-host, que consiste em dois servidores com o mesmo endereço (127.0.0.1) que escutam em portas diferentes.

Vejamos o seguinte exemplo:

```
servidor único upstream {  
    servidor 192.168.0.1;  
}
```

A configuração anterior declara um servidor único chamado upstream, que consiste em apenas um servidor.

A tabela a seguir lista os parâmetros opcionais da diretiva do servidor e sua descrição:

Sintaxe	Descrição
<code>weight=<number></code>	Especifica o peso numérico do servidor. É usado para distribuindo conexões entre os servidores. O valor padrão é 1.
<code>max_falha=<número></code>	Isso especifica o número máximo de tentativas de conexão após as quais o servidor é considerado indisponível. O valor padrão é 1.
<code>falhou_tempo limite=<número></code>	Isso especifica o tempo após o qual um servidor com falha será marcado como indisponível. O valor padrão é 10 segundos.
<code>cópia de segurança</code>	Isso rotula um servidor como um servidor de backup.
<code>baixa</code>	Isso rotula um servidor como indisponível.
<code>max_conexões = <número></code>	Isso limita o número de conexões simultâneas com o servidor.
<code>resolver</code>	Isso instrui o Nginx a atualizar automaticamente os endereços P de um servidor especificado usando um nome simbólico e aplicar esses endereços sem reiniciar o Nginx.

Usando servidores upstream

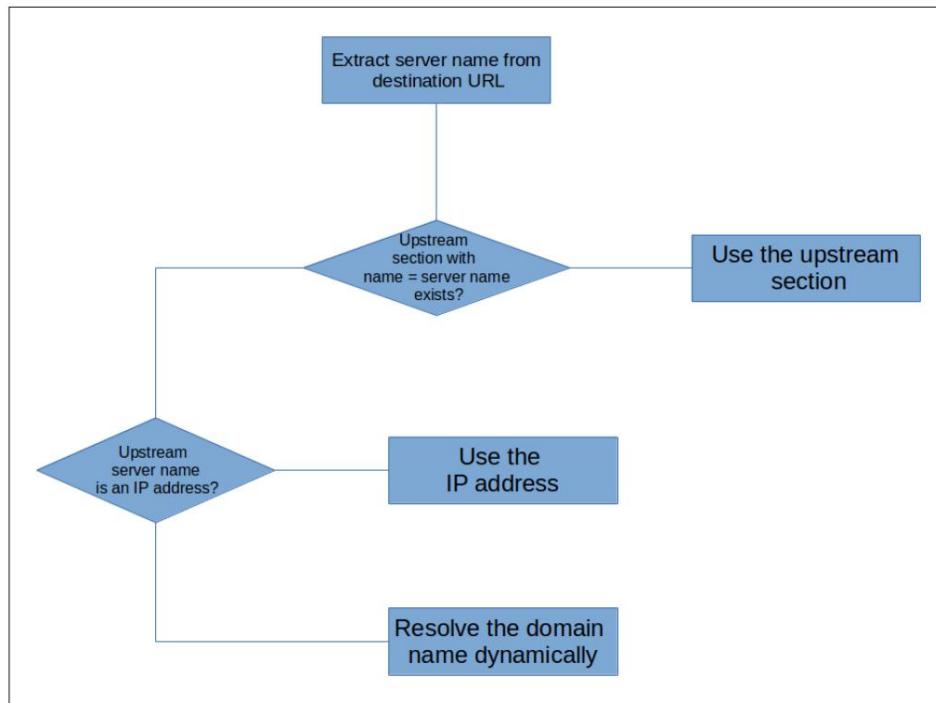
Uma vez que um servidor upstream é declarado, ele pode ser usado na diretiva `proxy_pass`:

```
http{
    upstream meu-cluster {
        servidor server1.example.com;
        servidor server2.example.com;
        servidor server3.example.com;
    }
    [...]
    servidor {
        [...]
        local @proxy {
            proxy_pass http://meu-cluster;
        }
    }
}
```

O upstream pode ser referido várias vezes a partir da configuração. Com a configuração anterior, uma vez solicitada a localização `@proxy`, o Nginx passará a solicitação para um dos servidores da lista de servidores do upstream.

Gerenciando o tráfego de entrada e saída

O algoritmo para resolver o endereço final de um servidor upstream é mostrado na figura a seguir:



Um algoritmo para resolver o endereço de um servidor upstream

Como uma URL de destino pode conter variáveis, ela é avaliada em tempo de execução e analisada como URL HTTP. O nome do servidor é extraído da URL de destino avaliada. O Nginx procura uma seção upstream que corresponda ao nome do servidor e, se existir, encaminha a solicitação para um dos servidores da lista de servidores upstream de acordo com uma estratégia de distribuição de solicitações.

Se existir uma seção upstream que corresponda ao nome do servidor, o Nginx verifica se o nome do servidor é um endereço IP. Nesse caso, o Nginx usa o endereço IP como o endereço final do servidor upstream. Se o nome do servidor for simbólico, o Nginx resolverá o nome do servidor no DNS em um endereço IP. Se for bem-sucedido, o endereço IP resolvido é usado como o endereço final do servidor upstream.

O endereço do servidor ou servidores DNS pode ser configurado usando a diretiva `resolver` :

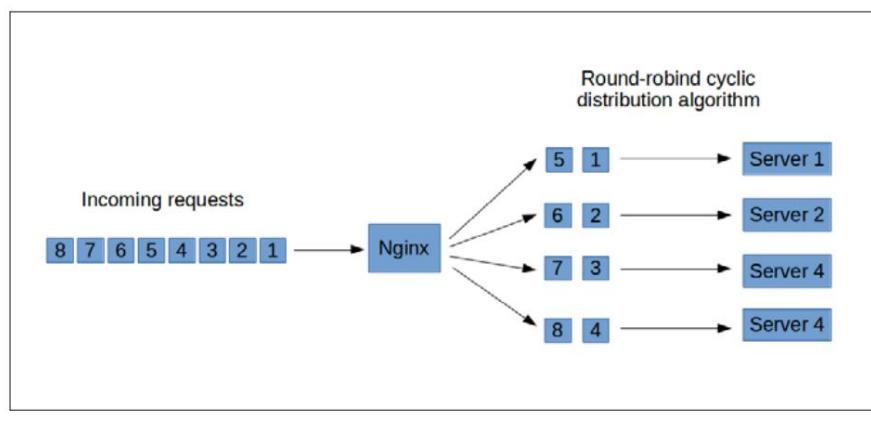
```
resolver 127.0.0.1;
```

A diretiva anterior usa uma lista de endereços IP dos servidores DNS como seus argumentos. Se um nome de servidor não puder ser resolvido com sucesso usando o resolvelor configurado, o Nginx retornará o status HTTP 502 (Gateway inválido).

Quando um upstream contém mais de um servidor na lista de servidores, o Nginx distribui as solicitações entre esses servidores na tentativa de dividir a carga entre os servidores disponíveis. Isso também é chamado de clustering, pois vários servidores agem como um só — eles são chamados de cluster.

Escolhendo uma estratégia de distribuição de solicitações

Por padrão, o Nginx usa o algoritmo Round-robin ao distribuir solicitações entre os servidores upstream disponíveis, conforme mostrado na figura a seguir:



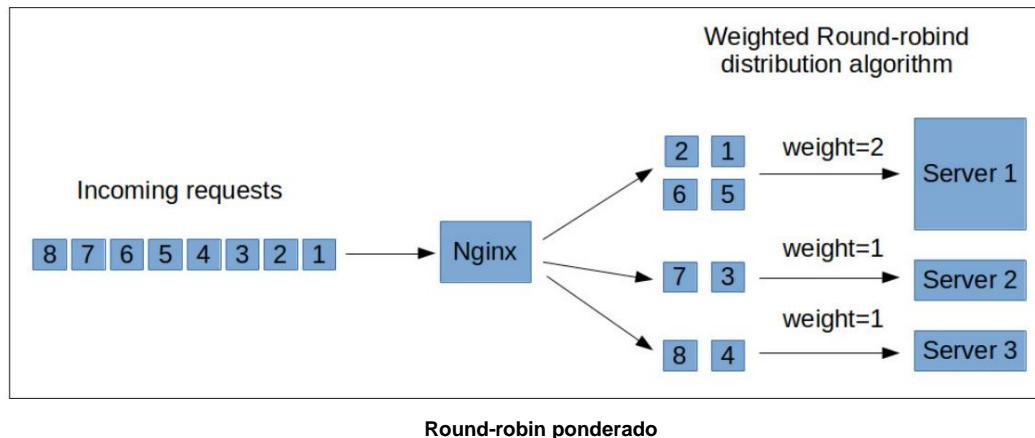
De acordo com esse algoritmo, as solicitações de entrada são atribuídas aos servidores da lista de servidores upstream em proporções iguais e ordem cíclica. Isso garante a distribuição igual das solicitações de entrada entre os servidores disponíveis, mas não garante a distribuição igual da carga entre os servidores.

Se os servidores na lista de servidores upstream tiverem capacidades variadas, o algoritmo de distribuição poderá ser alterado para levar em conta isso. É para isso que o peso do parâmetro é usado. Este parâmetro especifica o peso relativo de um servidor em comparação com outros servidores. Considere uma instalação em que um dos servidores seja duas vezes mais capaz que os outros dois. Podemos configurar o Nginx para esta instalação da seguinte forma:

```
upstream meu-cluster {
    servidor servidor1.example.com peso=2;
    servidor server2.example.com;
    servidor server3.example.com;
}
```

Gerenciando o tráfego de entrada e saída

O primeiro servidor é configurado para ter um peso duas vezes maior que os outros servidores e a estratégia de distribuição de solicitações muda de acordo. Isso é mostrado na figura a seguir:



Na figura anterior, podemos ver que duas em cada quatro solicitações recebidas irão para o servidor 1, uma irá para o servidor 2 e outra irá para o servidor 3.

A estratégia round-robin não garante que as solicitações do mesmo cliente sejam sempre encaminhadas para o mesmo servidor. Este último pode ser um desafio para aplicações web que esperam que o mesmo cliente seja atendido pelo mesmo servidor ou pelo menos precisem de alguma afinidade de usuários com servidores para um desempenho eficiente.

Com o Nginx, você pode resolver isso usando a estratégia de distribuição de solicitação de hash de IP. Com a estratégia de distribuição de hash IP, cada solicitação de um determinado endereço IP será encaminhada para o mesmo servidor de back-end. Isso é obtido fazendo hash do endereço IP do cliente e usando o valor numérico do hash para escolher o servidor da lista de servidores upstream. Para habilitar a estratégia de distribuição de solicitação de hash IP, use a diretiva ip_hash na seção upstream :

```
upstream meu-cluster {
    ip_hash;
    servidor server1.example.com;
    servidor server2.example.com;
    servidor server3.example.com;
}
```

A configuração anterior declara um upstream com três servidores subjacentes e habilita a estratégia de distribuição de solicitação de hash IP para cada um deles. Um pedido de um cliente remoto será encaminhado para um dos servidores desta lista e é sempre o mesmo para todos os pedidos do cliente.

capítulo 5

Se você adicionar ou remover um servidor da lista, a correspondência entre endereços IP e servidores será alterada e seu aplicativo da web terá que lidar com essa situação. Para tornar esse problema mais fácil de lidar, você pode marcar um servidor como indisponível usando o parâmetro `down`. As solicitações para este servidor serão encaminhadas para o próximo servidor disponível:

```
upstream meu-cluster {
    ip_hash;
    servidor server1.example.com;
    servidor server2.example.com inativo;
    servidor server3.example.com;
}
```

A configuração anterior declara o servidor `server2.example.com` indisponível e uma vez que uma solicitação é direcionada para este servidor, o próximo servidor disponível será escolhido (`server1.example.com` ou `server3.example.com`).

Se um endereço IP não for uma entrada conveniente para a função hash, você pode usar o hash diretiva em vez de `ip_hash` para escolher uma entrada que seja mais conveniente. O único argumento desta diretiva é um script, que é avaliado em tempo de execução e produz um valor usado como entrada para a função hash. Esse script pode conter, por exemplo, um cookie, um cabeçalho HTTP, uma combinação de um endereço IP e um agente do usuário, um endereço IP e um endereço IP com proxy e assim por diante. Dê uma olhada no exemplo a seguir:

```
upstream meu-cluster {
    hash "$cookie_uid";
    servidor server1.example.com;
    servidor server2.example.com;
    servidor server3.example.com;
}
```

A configuração anterior usa um cookie chamado `uid` como entrada para a função hash. Se o cookie armazenar um ID exclusivo de um usuário, as solicitações de cada usuário serão encaminhadas para um servidor fixo na lista de servidores `upstream`. Se um usuário ainda não tiver um cookie, a variável `$cookie_uid` será avaliada como uma string vazia e produzirá um valor de hash fixo. Portanto, todas as solicitações de usuários sem o cookie `uid` são encaminhadas para um servidor fixo da lista anterior.

No próximo exemplo, usaremos uma combinação de um endereço IP remoto e o campo do agente do usuário como entrada para a função hash:

```
upstream meu-cluster {
    hash "$remote_addr$http_user_agent";
    servidor server1.example.com;
    servidor server2.example.com;
    servidor server3.example.com;
}
```

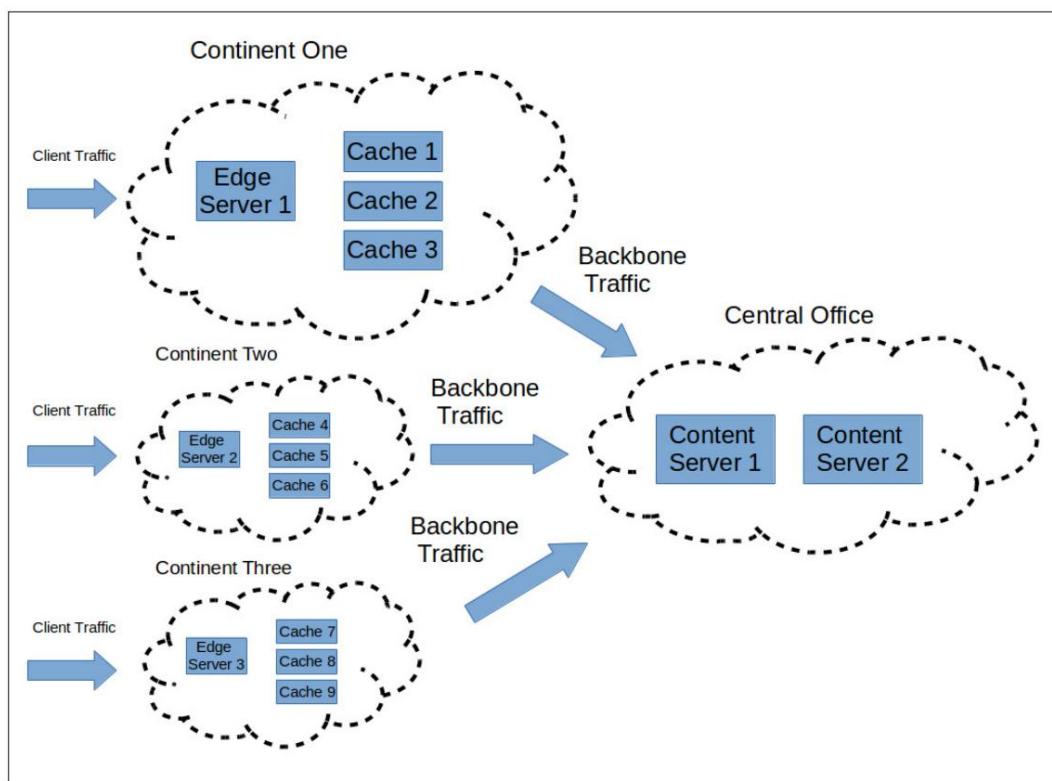
Gerenciando o tráfego de entrada e saída

A configuração anterior depende da diversidade de campos do agente de usuário e evita uma concentração de usuários de endereços IP proxy em um único servidor.

Configurando servidores de backup

Alguns servidores na lista de servidores podem ser marcados como *backup*. Ao fazer isso, você informa ao Nginx que esses servidores não devem ser usados normalmente e usados apenas quando todos os servidores que não são de backup não respondem.

Para ilustrar o uso de servidores de backup, imagine que você executa uma Rede de Distribuição de Conteúdo (CDN) onde vários servidores de borda distribuídos geograficamente lidam com o tráfego de usuários e um conjunto de servidores de conteúdo centralizado gera e distribui conteúdo para os servidores de borda. Isso é mostrado na figura a seguir.



Uma rede de distribuição de conteúdo

Os servidores de borda são co-localizados com um conjunto de caches altamente disponíveis que não alteram o conteúdo obtido dos servidores de conteúdo, mas simplesmente o armazenam. As caches têm de ser utilizadas enquanto alguma delas estiver disponível.

No entanto, quando nenhum dos caches está disponível por algum motivo, o servidor de borda pode entrar em contato com os servidores de conteúdo, embora isso não seja desejável. Tal comportamento (chamado degradação) pode remediar a situação até que a interrupção dos caches seja resolvida, mantendo o serviço disponível.

Em seguida, o upstream no servidor de borda pode ser configurado da seguinte forma:

```
upstream my-cache {
    servidor cache1.mycdn.com;
    servidor cache2.mycdn.com;
    servidor cache3.mycdn.com;

    backup do servidor content1.mycdn.com;
    backup do servidor content2.mycdn.com;
}
```

A configuração anterior declara os servidores cache1.mycdn.com, cache2.mycdn.com e cache3.mycdn.com como servidores primários a serem contatados. Eles serão usados enquanto algum deles estiver disponível.

Em seguida, listamos os servidores content1.mycdn.com e content2.mycdn.com como backup especificando o parâmetro backup . Esses servidores serão contatados somente se nenhum dos servidores primários estiver disponível. Esse recurso do Nginx fornece flexibilidade na maneira como a disponibilidade do seu sistema é gerenciada.

Determinando se um servidor está disponível

Como você define que um servidor está disponível? Para a maioria dos aplicativos, os erros de conectividade são sinais concretos de um servidor indisponível, mas e se um erro for gerado pelo software? Pode valer a pena tentar o próximo servidor se um servidor estiver disponível na camada de transporte (por TCP/IP), mas retornar erros HTTP como 500 (Erro de servidor interno) e 503 (Serviço indisponível) ou erros ainda mais suaves, como 403 (Proibido) ou 404 (Não encontrado). Se o servidor upstream for um proxy em si, pode ser necessário lidar com os erros HTTP 502 (Gateway incorreto) e 504 (Tempo limite do gateway).

O Nginx permite especificar condições de disponibilidade e nova tentativa usando as diretivas proxy_next_upstream, fastcgi_next_upstream, uwsgi_next_upstream, scgi_next_upstream e memcached_next_upstream. Cada uma dessas diretivas recebe uma lista de condições que serão tratadas como erros durante a comunicação com um servidor upstream e fazem o Nginx tentar novamente com outro servidor. Além disso, se o número de tentativas de interação malsucedidas com um servidor for maior que o valor do parâmetro max_fails para o servidor (o valor padrão é 1), o servidor será marcado como indisponível por um período especificado pela diretiva fail_timeout (o valor padrão é 10 segundos).

Gerenciando o tráfego de entrada e saída

A tabela a seguir lista todos os valores possíveis para os argumentos das diretrivas `proxy_next_upstream`, `fastcgi_next_upstream`, `uwsgi_next_upstream`, `scgi_next_upstream` e `memcached_next_upstream`:

Valor	Significado
<code>erro</code>	Ocorreu um erro de conexão ou ocorreu um erro durante o envio de uma solicitação ou o recebimento de uma resposta
<code>tempo esgotado</code>	Uma conexão expirou durante a configuração, enviando uma solicitação ou recebendo uma resposta
<code>invalid_header</code>	O servidor upstream retornou uma resposta vazia ou inválida
<code>http_500</code>	O servidor upstream retornou uma resposta com o código de status HTTP 500 (Erro do Servidor Interno)
<code>http_502</code>	O servidor upstream retornou uma resposta com o código de status HTTP 502 (Gateway ruim)
<code>http_503</code>	O servidor upstream retornou uma resposta com o código de status HTTP 503 (Serviço não disponível)
<code>http_504</code>	O servidor upstream retornou uma resposta com o código de status HTTP 504 (Tempo limite do gateway)
<code>http_403</code>	O servidor upstream retornou uma resposta com o código de status HTTP 403 (Proibido)
<code>http_404</code>	O servidor upstream retornou uma resposta com o código de status HTTP 404 (Não encontrado)
<code>fora</code>	Desabilita a passagem de solicitações para o próximo servidor

O valor padrão para as diretrivas anteriores é o tempo limite de erro. Isso faz com que o Nginx tente novamente uma solicitação com outro servidor somente se ocorrer um erro de conectividade ou um tempo limite.

Aqui está um exemplo de uma configuração que usa a diretiva `proxy_next_upstream` :

```
local @proxy {
    proxy_pass http://backend;
    tempo limite de erro proxy_next_upstream http_500 http_502 http_503 http_504;
}
```

A configuração anterior estende a opção padrão de nova avaliação e disponibilidade e permite tentar novamente com o próximo servidor em caso de erro de conectividade, erro de upstream (502, 503 ou 504) ou um tempo limite de conexão.

Ativando conexões persistentes

Por padrão, o Nginx não mantém abertas as conexões com servidores upstream. Manter as conexões abertas pode melhorar significativamente o desempenho do seu sistema. Isso ocorre porque as conexões persistentes eliminam a sobrecarga de configuração da conexão toda vez que uma solicitação é feita para um determinado servidor upstream.

Para habilitar conexões persistentes para um upstream, use a diretiva `keepalive` na seção `upstream` :

```
upstream meu-cluster {
    manter vivo 5;
    servidor server1.example.com;
    servidor server2.example.com;
    servidor server3.example.com;
}
```

O único argumento da diretiva `keepalive` especifica o número mínimo de conexões persistentes inativas no pool de conexões deste upstream. Se o número de conexões persistentes inativas crescer além desse número, o Nginx fechará quantas conexões forem necessárias para permanecer dentro desse número. Isso garante que um número específico de conexões quentes e prontas para uso estejam sempre disponíveis para uso. Ao mesmo tempo, essas conexões consomem os recursos dos servidores backend, portanto, esse número deve ser escolhido com cautela.

Para usar conexões persistentes com proxy HTTP, são necessários mais ajustes:

```
local @proxy {
    proxy_pass http://backend;
    proxy_http_versão 1.1;
    proxy_set_header Conexão "";
}
```

Na configuração anterior, alteramos a versão HTTP para 1.1 para que as conexões persistentes sejam esperadas por padrão. Também limpamos o cabeçalho `Connection` para que o cabeçalho `Connection` da solicitação original não influencie a solicitação com proxy.

Gerenciando o tráfego de entrada e saída

Limitando a taxa de transferência de uma conexão upstream

A taxa de transferência de uma conexão com um upstream pode ser limitada. Esse recurso pode ser usado para reduzir o estresse no servidor upstream. A diretiva `proxy_limit_rate` limita a taxa de transferência de uma conexão upstream em um local ao valor especificado no primeiro argumento:

```
local @proxy {  
    proxy_pass http://backend;  
    proxy_buffering ativo;  
    proxy_limit_rate 200k;  
}
```

A configuração anterior limitará a taxa de conexões com o backend especificado a 200 KBps. O limite de taxa é definido por solicitação. Se o Nginx abrir várias conexões com o servidor upstream, a taxa total será maior.



A limitação de taxa funciona apenas se o buffer de resposta do proxy estiver ativado usando a diretiva `proxy_buffering`.



Resumo

Neste capítulo, você aprendeu sobre várias ferramentas para gerenciamento de tráfego de entrada e saída. Essas ferramentas ajudarão você a garantir a confiabilidade de seu serviço da Web e a implementar esquemas de cache complexos.

No próximo capítulo, você aprenderá como extrair o máximo de desempenho de seu servidor web e otimizar o uso de recursos — ajuste de desempenho.

6

Ajuste de desempenho

O ajuste de desempenho é a melhoria do desempenho do sistema. Em nosso contexto, é o desempenho de um serviço web inteiro ou de um servidor web individual. A necessidade de tal atividade surge quando há um problema de desempenho real ou antecipado, como latência de resposta excessiva, taxa de upload ou download insuficiente, falta de escalabilidade do sistema ou uso excessivo de recursos do sistema de computador para uso aparentemente baixo do serviço.

Neste capítulo, veremos vários tópicos que lidam com problemas de desempenho usando recursos do Nginx. Cada seção explica quando e como uma solução é aplicável; isto é, que tipo de problemas de desempenho ele aborda.

Neste capítulo você aprenderá sobre:

- Como otimizar a recuperação de arquivos estáticos
- Como configurar a compactação de resposta
- Como otimizar a alocação de buffer de dados
- Como acelerar o SSL ativando o cache de sessão
- Como otimizar a alocação do processo de trabalho em sistemas multicore

Otimizando a recuperação de arquivos estáticos

O desempenho da recuperação de arquivos estáticos afeta diretamente o desempenho do site percebido pelos visitantes. Isso acontece porque as páginas da Web geralmente contêm várias referências a recursos dependentes. Esses recursos precisam ser recuperados rapidamente antes que a página inteira possa ser renderizada. Quanto mais rápido o servidor web pode começar a retornar um arquivo estático (menor latência) e quanto maior o paralelismo de recuperação, maior o desempenho percebido do site.

Ajuste de desempenho

Quando a latência é o fator determinante, é importante que os arquivos sejam retornados predominantemente da memória principal, pois ela tem uma latência muito menor em comparação com os discos rígidos.

Felizmente, o sistema operacional já cuida muito bem disso através do cache do ilesystem. Você só precisa estimular o uso do cache especificando alguns parâmetros consultivos e eliminando o desperdício:

```
local /css {
    enviar arquivo ativado;
    sendfile_max_chunk 1M;
    [...]
}
```

Por padrão, o Nginx lê o conteúdo de um arquivo no espaço do usuário antes de enviar ao cliente. Isso não é o ideal e pode ser evitado usando a chamada de sistema `sendfile()` se estiver disponível. A função `sendfile()` implementa uma estratégia de transferência zero-copy copiando dados de um descriptor de arquivo para outro, ignorando o espaço do usuário.

Habilitamos `sendfile()` especificando o parâmetro `sendfile on` no código. Limitamos a quantidade máxima de dados que `sendfile()` pode enviar em uma invocação a 1 MB usando a diretiva `sendfile_max_chunk`. Dessa forma, evitamos que uma única conexão rápida ocupe todo o processo de trabalho.



Os filtros do corpo de resposta, como o compressor `.gzip`, exigem dados de resposta no espaço do usuário. Eles não podem ser combinados com uma estratégia de cópia zero e, consequentemente, com `sendfile()`. Quando habilitados, eles cancelam o efeito de `sendfile()`.

A configuração anterior é otimizada para latência. Compare-o com o exemplo da seção *Configurando o Nginx para servir dados estáticos* no Capítulo 2, Gerenciando o Nginx. Você verá que a diretiva `tcp_nopush` desapareceu. O estado desligado desta opção tornará a utilização da rede um pouco menos eficiente, mas entregará dados—incluindo o cabeçalho HTTP—ao cliente o mais rápido possível.

Com o `tcp_nopush` ativado, o primeiro pacote da resposta será enviado assim que o bloco de dados for obtido por `sendfile()`.

Outro aspecto da recuperação de arquivos estáticos é o download de arquivos grandes. Nesse caso, o tempo de inicialização não é tão importante quanto a taxa de download ou, em outras palavras, a velocidade de download que um servidor pode atingir ao retornar um arquivo grande. O cache deixa de ser desejável para arquivos grandes. O Nginx os lê sequencialmente, portanto, as ocorrências de cache são muito menos prováveis para eles. Os segmentos armazenados em cache de um arquivo grande, portanto, simplesmente poluiriam o cache.

Capítulo 6

No Linux, o armazenamento em cache pode ser ignorado usando E/S direta. Com a E/S direta habilitada, o sistema operacional traduz os deslocamentos de leitura nos endereços de dispositivo de bloco subjacentes e enfileira as solicitações de leitura diretamente na fila de dispositivos de bloco subjacentes. A configuração a seguir mostra como habilitar a E/S direta:

```
local/mídia {
    enviar arquivo desligado;
    direção 4k;
    output_buffers 1 256k;
    [...]
}
```

A diretiva **directio** recebe um único argumento que especifica o tamanho mínimo que um arquivo deve ter para ser lido com E/S direta. Além de especificar a direção, estendemos o buffer de saída usando a diretiva **output_buffers** para aumentar a eficiência das chamadas do sistema.

Observe que a E/S direta bloqueia os processos de trabalho durante as leituras. Isso reduz o paralelismo e a taxa de transferência de recuperação de arquivos. Para evitar o bloqueio e aumentar o paralelismo, você pode habilitar a E/S assíncrona (AIO):

```
local/mídia {
    enviar arquivo desligado;
    vai ser;
    direção 4k;
    output_buffers 1 256k;
    [...]
}
```

No Linux, o AIO está disponível a partir da versão 2.6.22 do kernel e não é bloqueante apenas em combinação com E/S direta. AIO e Direct I/O podem ser combinados com **sendfile()**:

```
local/mídia {
    enviar arquivo ativado;
    vai ser;
    direção 4k;
    output_buffers 1 256k;
    [...]
}
```

Neste caso, arquivos menores que o tamanho especificado na direção serão enviados usando **sendfile()**, ou então com AIO mais Direct I/O.

Ajuste de desempenho

A partir do Nginx versão 1.7.11, você pode delegar operações de leitura de arquivo para um conjunto de threads. Isso faz todo o sentido se você não estiver limitado por recursos de memória ou CPU.

Como os encadeamentos não exigem E/S direta, habilitá-los em arquivos grandes levará a um cache agressivo:

```
local/mídia {
    enviar arquivo ativado;
    aio fios;
    [...]
}
```

Threads não são compilados por padrão (no momento em que escrevo este capítulo), então você deve habilitá-los usando a opção de configuração `with-threads`. Além disso, os encadeamentos podem funcionar apenas com os métodos de processamento de eventos `epoll`, `kqueue` e `eventport`.

Com threads, tanto o paralelismo quanto o armazenamento em cache podem ser alcançados sem bloquear o processo de trabalho, embora os threads e a comunicação entre threads exijam alguns recursos adicionais.

Ativando a compactação de resposta

O desempenho do seu site pode ser melhorado ativando a compactação de resposta usando GZIP. A compactação reduz o tamanho de um corpo de resposta, reduz a largura de banda necessária para transferir os dados de resposta e, por fim, garante que os recursos do seu site sejam entregues ao lado do cliente mais cedo.

A compactação pode ser habilitada usando a diretiva `gzip`:

```
local / {
    gzip ativado;
    [...]
}
```

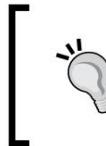
Essa diretiva é válida nas seções `http`, `server`, `location` e `if`. Especificar `off` como o primeiro argumento desta diretiva desativa a compactação no localização se foi ativado em seções externas.

Capítulo 6

Por padrão, somente documentos com *texto/HTML* do tipo MIME são compactados. Para habilitar a compactação para outros tipos de documentos, use a diretiva `gzip_types`:

```
local {
    gzip ativado;
    gzip_types
    text/html text/plain text/css application/x-javascript text/xml application/xml application/xml+rss
    text/javascript;
    [...]
}
```

A configuração anterior permite a compactação de tipos MIME nos quais documentos de hipertexto, folhas de estilo em cascata e arquivos JavaScript parecem estar. Esses são os tipos de documentos que mais se beneficiam da compactação, como arquivos de texto e arquivos de código-fonte - se forem grandes o suficiente — geralmente contêm muita entropia.



Arquivos, imagens e filmes não são adequados para compactação, pois geralmente já estão compactados. Arquivos executáveis são menos adequados para compactação, mas podem se beneficiar dela em alguns casos.

Faz sentido desabilitar a compactação para documentos pequenos, pois a eficiência da compactação pode não valer os esforços — ou pior ainda — pode ser negativa. No Nginx, você pode implementar a compactação usando a diretiva `gzip_min_length`. Esta diretiva especifica o tamanho mínimo que um documento deve ter para ser elegível para compactação:

```
local {
    gzip ativado;
    gzip_min_length 512;
    [...]
}
```

Com a configuração anterior, todos os documentos menores que 512 bytes não serão compactados. As informações de comprimento usadas para aplicar essa restrição são extraídas do cabeçalho de resposta `Content-Length`. Se nenhum cabeçalho estiver presente, a resposta será compactada independentemente de seu comprimento.



A compactação de resposta tem um custo: consome muita CPU. Você precisa considerar isso em seu planejamento de capacidade e design do sistema. Se a utilização da CPU se tornar um gargalo, tente reduzir o nível de compactação usando a diretiva `gzip_comp_level`.

Ajuste de desempenho

A tabela a seguir lista algumas outras diretivas que afetam o comportamento da compactação:

Diretiva	Função
<code>gzip_disable <regex></code>	Se o campo User-Agent de uma solicitação corresponder ao especificado expressão regular, a compactação para essa solicitação será desabilitada.
<code>gzip_comp_level <nível></code>	Isso especifica o nível de compactação GZIP a ser usado. O menor é 1 e o maior é 9. Esses valores correspondem às opções -1 ... -9 do comando gzip.

As diretivas anteriores podem ajudá-lo a ajustar a compactação de resposta em seu sistema.

A eficiência da compressão do corpo de resposta pode ser monitorada através do `$gzip_ratio` variável. Esta variável indica a razão de compressão alcançada igual à razão entre o tamanho do corpo de resposta original e o tamanho do corpo comprimido.

O valor dessa variável pode ser gravado no arquivo de log e posteriormente extraído e selecionado pelo seu sistema de monitoramento. Considere o seguinte exemplo:

```
http{
    gzip log_format
    '$remote_addr - $remote_user [$time_local] $status '
        '"$request" $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$host" $gzip_ratio';

    servidor {
        [...]
        access_log /var/log/nginx/access_log gzip;
        [...]
    }
}
```

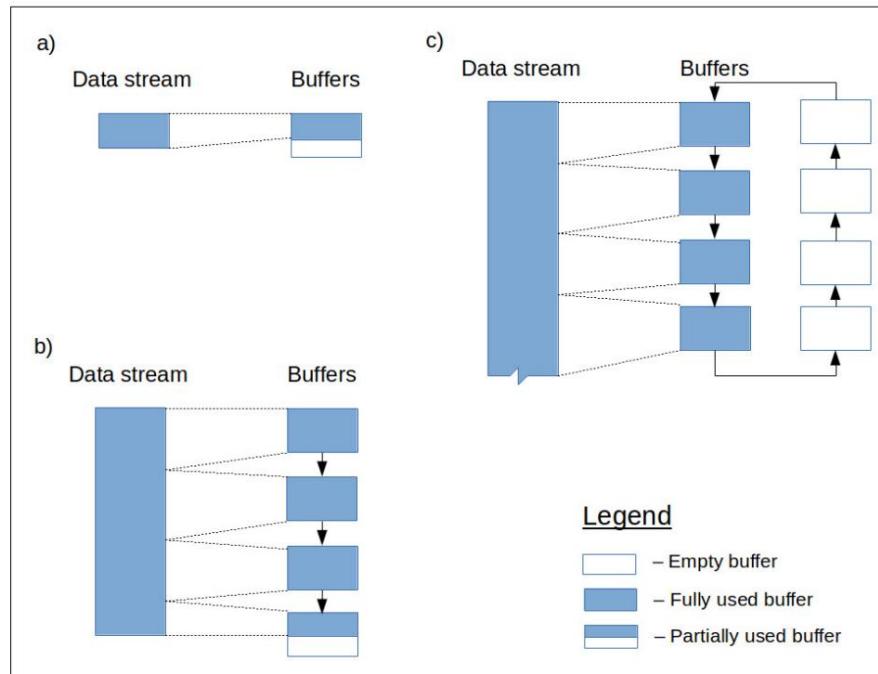
A configuração anterior cria um formato de arquivo de log chamado gzip e usa esse formato para registrar solicitações HTTP em um dos hosts virtuais. O último campo no arquivo de log indicará a taxa de compressão alcançada.

Otimizando a alocação de buffer

O Nginx usa buffers para armazenar dados de solicitação e resposta em vários estágios. A alocação ideal de buffer pode ajudar a poupar o consumo de memória e reduzir o uso da CPU. A tabela a seguir lista as diretivas que controlam a alocação de buffer e os estágios aos quais elas são aplicadas:

Diretiva	Função
<code>client_body_buffer_size <tamanho></code>	Isso especifica o tamanho do buffer usado para receber o corpo da solicitação do cliente.
<code>output_buffers <número> <tamanho></code>	Isso especifica o número e o tamanho dos buffers usados para enviar o corpo da resposta ao cliente caso nenhuma aceleração seja usada.
<code>gzip_buffers <número> <tamanho></code>	Isso especifica o número e o tamanho dos buffers usados para compactar o corpo da resposta.
<code>proxy_buffers <número> <tamanho></code>	Isso especifica o número e o tamanho dos buffers usados para receber o corpo da resposta de um servidor proxy. Esta diretiva só faz sentido se o buffer estiver habilitado.
<code>fastcgi_buffers <número> <tamanho></code>	Isso especifica o número e o tamanho dos buffers usados para receber o corpo da resposta de um servidor FastCGI.
<code>uwsgi_buffers <número> <tamanho></code>	Isso especifica o número e o tamanho dos buffers usados para receber o corpo da resposta de um servidor UWCGI.
<code>scgi_buffers <número> <tamanho></code>	Isso especifica o número e o tamanho dos buffers usados para receber o corpo da resposta de um servidor SCGI.

Como você pode ver, a maioria das diretivas recebe dois argumentos: um número e um tamanho. O argumento number especifica o número máximo de buffers que podem ser alocados por solicitação. O argumento size especifica o tamanho de cada buffer.



Ajuste de desempenho

A figura anterior ilustra como os buffers são alocados para um fluxo de dados. A parte a mostra o que acontece quando um fluxo de dados de entrada é menor que o tamanho do buffer especificado nas diretivas acima. O fluxo de dados ocupa todo o buffer, embora o espaço para todo o buffer seja alocado no heap. A parte b mostra um fluxo de dados maior que um único buffer, mas menor que a cadeia de buffers mais longa permitida. Como você pode ver, se os buffers forem usados da maneira mais eficiente, alguns deles serão totalmente usados e o último poderá ser usado apenas parcialmente. A parte c mostra um fluxo de dados muito mais longo do que a cadeia mais longa de buffers permitida. O Nginx tenta encher todos os buffers disponíveis com dados de entrada e os extingue assim que os dados são enviados.

Depois disso, os buffers vazios esperam até que mais dados de entrada fiquem disponíveis.

Novos buffers são alocados desde que não haja buffers livres disponíveis e os dados de entrada estejam disponíveis. Uma vez que o número máximo de buffers é alocado, o Nginx espera até que os buffers usados sejam esvaziados e os reutiliza. Isso garante que, independentemente da duração do fluxo de dados, ele não consumirá mais memória por solicitação (o número de buffers multiplicado pelo tamanho) do que o especificado pela diretiva correspondente.

Quanto menores os buffers, maior a sobrecarga de alocação. O Nginx precisa gastar mais ciclos de CPU para alocar e liberar buffers. Quanto maiores os buffers, maior a sobrecarga de consumo de memória. Se uma resposta ocupar apenas uma fração de um buffer, o restante do buffer não será usado, mesmo que todo o buffer precise ser alocado do heap.

A parte mínima da configuração à qual as diretivas de tamanho de buffer podem ser aplicadas é um local. Isso significa que, se combinações de respostas grandes e pequenas compartilharem o mesmo local, o padrão de uso de buffer combinado variará.

Arquivos estáticos são lidos em buffers controlados pela diretiva `output_buffers`, a menos que `sendfile` esteja ativado. Para arquivos estáticos, vários buffers de saída não fazem muito sentido, pois eles são preenchidos no modo de bloqueio de qualquer maneira (isso significa que um buffer não pode ser esvaziado enquanto o outro está sendo preenchido). No entanto, buffers maiores levam a uma menor taxa de chamadas do sistema. Considere o seguinte exemplo:

```
local/mídia {
    output_buffers 1 256k;
    [...]
}
```

Se o tamanho do buffer de saída for muito grande sem threads ou AIO, isso pode levar a longas leituras de bloqueio que afetarão a capacidade de resposta do processo de trabalho.

Quando um corpo de resposta é canalizado de um servidor proxy, FastCGI, UWCGI ou servidor SCGI, o Nginx pode ler dados em uma parte dos buffers e enviar simultaneamente a outra parte para o cliente. Isso faz mais sentido para respostas longas.

Capítulo 6

Suponha que você tenha ajustado sua pilha TCP antes de ler este capítulo. O tamanho total de uma cadeia de buffer é então conectado aos tamanhos de buffer de leitura e gravação do soquete do kernel. No Linux, o tamanho máximo de um buffer de leitura de soquete do kernel pode ser examinado usando o seguinte comando:

```
$ cat /proc/sys/net/core/rmem_max
```

Embora o tamanho máximo de um buffer de gravação de soquete do kernel possa ser examinado usando o seguinte comando:

```
$ cat /proc/sys/net/core/wmem_max
```

Essas configurações podem ser alteradas usando o comando `sysctl` ou via `/etc/sysctl.conf` na inicialização do sistema.

No meu caso, ambos estão definidos para 163840 (160 KB). Isso é baixo para um sistema real, mas vamos usá-lo como exemplo. Esse número é a quantidade máxima de dados que o Nginx pode ler ou gravar em um soquete em uma chamada de sistema sem que o soquete seja suspenso. Com leituras e gravações ocorrendo de forma assíncrona, precisamos de um espaço de buffer não inferior à soma de `rmem_max` e `wmem_max` para uma taxa de chamada de sistema ideal.

Suponha que o Nginx anterior faça proxy de arquivos longos com `rmem_max` e `wmem_max` definições. A configuração a seguir deve gerar a menor taxa de chamadas do sistema com a quantidade mínima de memória por solicitação no caso mais extremo:

```
local @proxy {  
    proxy_pass http://backend;  
    proxy_buffers 8 40k;  
}
```

As mesmas considerações se aplicam às diretivas `fastcgi_buffers`, `uwsgi_buffers` e `scgi_buffers`.

Para corpos de resposta curtos, o tamanho do buffer deve ser um pouco maior que o tamanho predominante de uma resposta. Nesse caso, todas as respostas serão armazenadas em um buffer - apenas uma alocação por solicitação será necessária.

Para a configuração anterior, suponha que a maioria das respostas tenha 128 KB, enquanto algumas abrangem dezenas de megabytes. A configuração de buffer ideal estará em algum lugar entre `proxy_buffers 2 160k` e `proxy_buffers 4 80k`.

Ajuste de desempenho

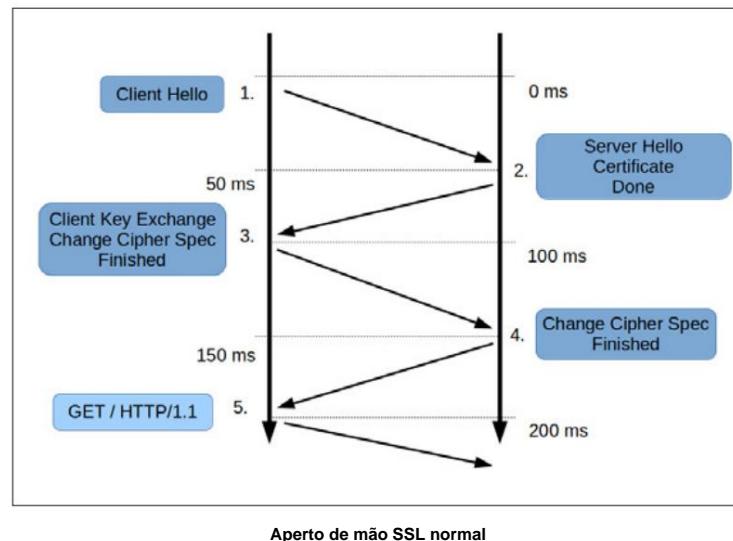
No caso de compactação do corpo de resposta, o tamanho da cadeia de buffer GZIP deve ser reduzido pela taxa de compactação média. Para a configuração anterior, suponha que a taxa de compactação média seja 3,4. A configuração a seguir deve gerar a menor taxa de chamadas do sistema com uma quantidade mínima de memória por solicitação na presença de compactação do corpo da resposta:

```
local @proxy {
    proxy_pass http://backend;
    proxy_buffers 8 40k;
    gzip ativo;
    gzip_buffers 4 25k;
}
```

Na configuração anterior, garantimos que, no caso mais extremo, se metade dos buffers de proxy estiver sendo usada para recepção, a outra metade estará pronta para compactação. Os buffers GZIP são configurados de forma a garantir que a saída do compressor para metade dos dados não compactados ocupe metade dos buffers de saída, enquanto a outra metade dos buffers com dados compactados são enviados ao cliente.

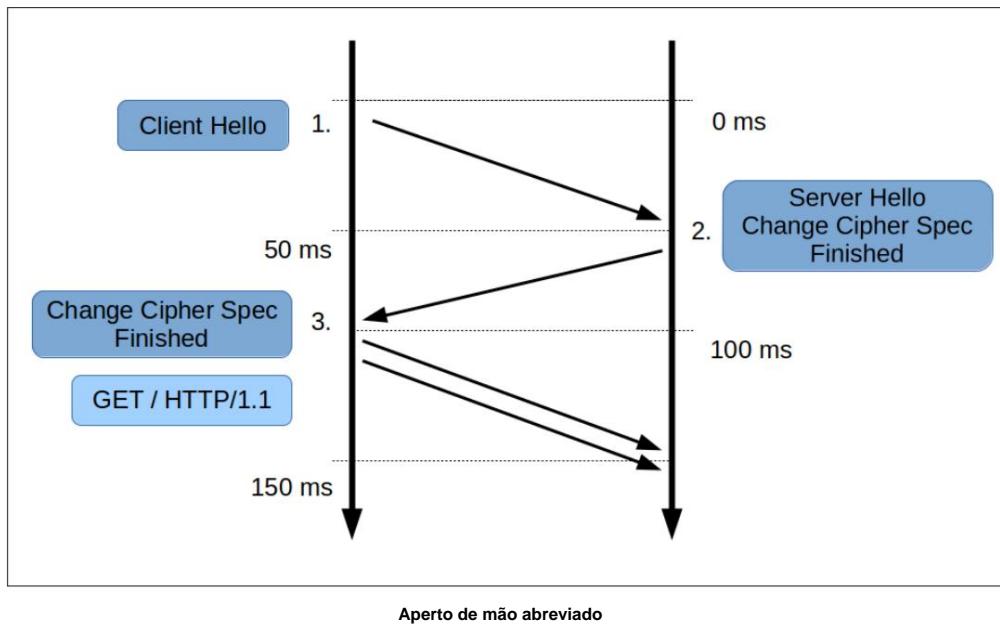
Habilitando a reutilização de sessão SSL

Uma sessão SSL é iniciada por um procedimento de handshake que envolve várias viagens de ida e volta (consulte a figura a seguir). O cliente e o servidor precisam trocar quatro mensagens com uma latência de cerca de 50 milissegundos cada. No total, temos pelo menos 200 milissegundos de sobrecarga ao estabelecer uma conexão segura. Além disso, tanto o cliente quanto o servidor precisam realizar operações criptográficas de chave pública para compartilhar um segredo comum. Essas operações são computacionalmente caras.



Capítulo 6

O cliente pode solicitar um handshake abreviado em vigor (veja a figura a seguir), economizando uma viagem completa de 100 milissegundos e evitando a parte mais cara do handshake SSL completo:



O handshake abreviado pode ser realizado através dos identificadores de sessão mecanismo definido pela RFC 5246, ou através do mecanismo de *tickets* de sessão detalhado na RFC 5077.

Para possibilitar handshakes abreviados com identificadores de sessão, o servidor precisa armazenar os parâmetros de sessão em um cache codificado por um identificador de sessão. No Nginx, esse cache pode ser configurado para ser compartilhado com todos os processos de trabalho. Quando um cliente solicita um handshake abreviado, ele fornece ao servidor um identificador de sessão para que ele possa recuperar os parâmetros de sessão do cache. Depois disso, o procedimento de handshake pode ser encurtado e a criptografia de chave pública pode ser ignorada.

Para habilitar o cache de sessão SSL, use a diretiva `ssl_session_cache` :

```
http{
    ssl_session_cache embutido:40000;
    [...]
}
```

Ajuste de desempenho

Essa configuração permite o cache de sessão SSL com cache de sessão OpenSSL integrado. O número no primeiro argumento (40000) especifica o tamanho do cache em sessões. O cache integrado não pode ser compartilhado entre processos de trabalho. Consequentemente, isso reduz a eficiência da reutilização da sessão SSL.

A configuração a seguir habilita o cache de sessão SSL com um cache compartilhado entre processos de trabalho:

```
http{
    ssl_session_cache compartilhado:ssl:1024k;
    [...]
}
```

Isso cria um cache de sessão SSL compartilhado denominado `ssl` e permite a reutilização da sessão SSL com esse cache. O tamanho do cache agora é especificado em bytes. Cada sessão ocupa cerca de 300 bytes nesse cache.

É possível realizar um handshake SSL abreviado sem o estado do servidor usando um mecanismo de tíquetes de sessão SSL. Isso é feito empacotando parâmetros de sessão em um objeto binário e criptografando-o com uma chave conhecida apenas pelo servidor. Esse objeto criptografado é chamado de tíquete de sessão.

Um ticket de sessão pode ser transferido com segurança para o cliente. Quando o cliente deseja retomar uma sessão, ele apresenta o ticket de sessão ao servidor. O servidor o descriptografa e extrai os parâmetros da sessão.

Os tickets de sessão são uma extensão do protocolo TLS e podem ser usados com o TLS 1.0 e posteriores (o SSL é um predecessor do TLS).

Para habilitar tickets de sessão, use a diretiva `ssl_session_tickets` :

```
http{
    ssl_session_tickets ativados;
    [...]
}
```

Naturalmente, ambos os mecanismos podem ser ativados ao mesmo tempo:

```
http{
    ssl_session_cache compartilhado:ssl:1024k;
    ssl_session_tickets ativados;
    [...]
}
```

Capítulo 6

Por motivos de segurança, o tempo de vida da sessão em cache é limitado para que os parâmetros da sessão não possam ser atacados enquanto a sessão estiver ativa. O Nginx define o tempo de vida máximo da sessão SSL padrão para 5 minutos. Se a segurança não for uma grande preocupação e os visitantes passarem um tempo considerável em seu site, você poderá estender a vida útil máxima da sessão, aumentando a eficiência do SSL em vigor.

O tempo de vida máximo da sessão SSL é controlado pelo `ssl_session_timeout` diretiva:

```
http{
    ssl_session_cache compartilhado:ssl:1024k;
    ssl_session_tickets ativados;
    ssl_session_timeout 1h;
    [...]
}
```

A configuração anterior habilita ambos os mecanismos de reutilização de sessão e define o tempo de vida máximo da sessão SSL para 1 hora.

Alocação de processos de trabalho em sistemas multicore

Se sua carga de trabalho Nginx estiver vinculada à CPU, como ao usar compactação de resposta em conteúdo com proxy, em sistemas com vários processadores ou vários núcleos de processador, talvez seja possível obter desempenho adicional associando cada processo de trabalho a seu próprio processador/núcleo.

Em um processador multinúcleo, cada núcleo tem sua própria instância do Translation Lookaside Buffer (TLB) que é usado pela unidade de gerenciamento de memória para acelerar a tradução de endereços virtuais. Em um sistema operacional multitarefa preemptivo, cada processo tem seu próprio contexto de memória virtual. Quando um sistema operacional atribui um processo ativo a um núcleo de processador e o contexto de memória virtual não corresponde ao contexto que enchia o TLB desse núcleo de processador, o sistema operacional precisa luxar o TLB, pois seu conteúdo não é mais válido.

O novo processo ativo então recebe uma penalidade de desempenho, porque ele precisa preencher o TLB com novas entradas enquanto lê ou grava locais de memória.

O Nginx tem a opção de "colar" um processo a um núcleo de processador. Em um sistema com uma única instância Nginx, os processos de trabalho serão agendados na maioria das vezes. Em tais circunstâncias, há uma probabilidade muito alta de que o contexto de memória virtual não precise ser comutado e o TLB não precise ser exuberante. A "aderência" de um processo torna-se então útil. A "aderência" é chamada de afinidade da CPU.

Ajuste de desempenho

Considere um sistema com quatro núcleos de processador. A afinidade da CPU pode ser configurada da seguinte forma:

```
trabalhadores_processos 4;  
worker_cpu_affinity 0001 0010 0100 1000;
```

Essa configuração atribui a cada trabalhador seu próprio núcleo de processador. A diretiva de configuração `worker_cpu_affinity` recebe muitos argumentos, pois muitos processos de trabalho devem ser iniciados. Cada argumento especifica uma máscara, onde um bit com valor 1 tem afinidade com o processador correspondente e um bit com valor 0 não tem afinidade com o processador correspondente.



A afinidade da CPU não garante um aumento no desempenho, mas certifique-se de experimentá-lo se o seu servidor Nginx estiver executando tarefas vinculadas à CPU.

Resumo

Neste capítulo, você aprendeu várias receitas que o ajudarão a enfrentar os desafios de desempenho e escalabilidade do seu sistema. É importante lembrar que essas receitas não são soluções para todos os possíveis problemas de desempenho e representam meras trocas entre diferentes formas de utilização dos recursos do seu sistema.

No entanto, eles são itens obrigatórios na caixa de ferramentas de um web master ou de um engenheiro de confiabilidade de site que deseja dominar o Nginx e seus recursos de desempenho e escalabilidade.

Índice

UMA

controle de acesso cerca
de 80 acessos, restringindo por endereço IP 80, 81 autenticação básica, usando para restrição de acesso 85-87 geo diretiva usada, para restringir acesso por Endereço IP 82-85
métodos de restrição de acesso múltiplo, combinando 90 usuários, autenticando com subsolicitação 88, 89 algoritmos, para criptografia de

senha
CRIPTA 86
MD5 86
SHA 86
SSHA 86

E/S Assíncrona (AIO) 113

B

criação
binária, com informações de depuração 50 expressão binária 16 operadores binários != 16 != 16 != ~ 16
= 16
~ 16
~* 16

break lag 72
alocação de buffer
otimizando 116-120

C

processo do carregador de cache 62 gerenciador de cache 63 cache de cerca de 61 casos de borda, manipulação de 68, 69 disponibilidade de cache, melhoria de 66, 67 eficiência de cache, melhoria de 66 chaves de cache, seleção de 64, 65 caches, configuração de 61-63 habilitação de 63, 64 exceções, manipulação de 68, 69 capturas de cabeçalho de resposta a montante 63 77

Certificate Signing Request (CSR) 44 comando de configuração em cluster 27, dados de identificação
Nome comum (CN) 45
Nome do país (C) 44
Localidade ou cidade (L) 44
Unidade Organizacional (UO) 45
Organização (O) 45
Estado ou província (S) 44
comandos, versão CentOS
definindo 4 diretivas de configuração
client_body_temp_path 47
fastcgi_temp_path 47
proxy_temp_path 47
scgi_temp_path 47
uwsgi_temp_path 47
Rede de Distribuição de Conteúdo (CDN) 106

sinais de controle	últimos
cerca de 30	74 permanentes 75
casos difíceis, manuseio 39	redirecionamento 75
modo de desligamento rápido	
31 modo de desligamento normal	
31 desligamento normal do	
trabalhador 37 arquivo de log, reabertura 34, 35	
Atualização binária Nginx 35, 36	
reconfiguração 32, 33 processo de	
desligamento 31 procedimento de	
atualização, finalizando 38, 39 usando 30	
D	G
informações de depuração	formato de
binário, criando com 50	diretiva geo
diretiva padrão 83 delete	82 usando 82
diretiva 83 problemas de	
desenvolvedores, comunicando-	
se com 49	
diretivas, alocação de buffer	
client_body_buffer_size <size> 117	
fastcgi_buffers <number> <size> 117	
gzip_buffers <number> <size> 117	
output_buffers <number> <size> 117	
proxy_buffers <number> <size> 117	
scgi_buffers <number> <size> 117	
uwsgi_buffers <number> <size> 117	
diretivas, compressão gzip_comp_level <level> 116	
gzip_disable <regex> 116 scripts de	
inicialização específicos de distribuição	
cerca de 40 usando 40	
E	H
Pacotes extras para Enterprise Linux (EPEL)	utilitário htpasswd
cerca de	usando 86
3 habilitando 3	
URL 3	Sinal HUP 32
F	
Filesystem Hierarchy Standard (FHS) 2 atrasos,	
quebra de diretiva de reescrita 74	
	eu
	padrões de localização
	exatos 15 expressão
	regular 15 simples 14 etapas
	do procedimento de
	reabertura do arquivo log, definindo
	34 lista de verificação do
	procedimento de rotação do arquivo
	log 35
	logrotar 35

M

processo mestre
cerca de 26
definindo 26
 alocação de processos
 de trabalho de sistemas multi-
 core, usados em 123, 124

N

Nginx
sobre 1, 35
construindo
6 melhores práticas de configuração
22 regras de herança de configurações
de configuração 18-20 configurando
9 arquitetura de processamento de
conexão 26-28 http seção 13 se seção 15-17
inclusões 12 instalando 1 instalando, de arquivos
de origem 4 instalando, no CentOS/ Scientific
Linux 3, 4 instalando, no Red Hat Enterprise
Linux 3, 4 instalando, no Ubuntu 2 questões 29
limit_except seção 18 local seção 14 outros
tipos de seção 18 amostra de configuração 21, 22
seções 13 servidor seção 13 configurando, para
servir dados estáticos 42, 43 soluções 29 iniciando
28, 29 parando 28, 29 solução de problemas 7 seção
upstream 14

Tipos

de valor de URL
1 10 variáveis 10, 11
Nginx, como proxy reverso
sobre 51, 52 configuração
de back-end, caminho certo 53, 54

cookies, tratamento de
57 downloads, aceleração de 61
erros, tratamento de 59, 60
endereços IP de saída, seleção de 60
redirecionamentos, tratamento de 55, 56
configuração de 52, 53
SSL, usando
transparência 58, 59, adicionando 54, 55

Configuração do Nginx

URL 22
Locais de
arquivos Nginx 6
Estrutura de instalação
do Nginx 8
Nginx, nas alternativas
do Ubuntu 2
Arquivos de origem
do Nginx baixando
5 solução de problemas 5
URL 5

O

tráfego de saída, gerenciando
cerca de 99 servidores de
backup, configurando 106, 107 conexões
persistentes, habilitando a estratégia de
distribuição de 109 solicitações,
selecionando a disponibilidade de
103-105 servidores, determinando 107, 108
taxas de transferência de conexão upstream,
limitando 110 servidores upstream,
declarando 99, 100 servidores upstream,
usando 101, 102

P

arquivos de parâmetro, pasta de configuração Nginx
fastcgi_params 8 koi-utf 8 koi-win 8 mime.types 8
naxsi.rules (opcional) 8 proxy_params 8
scgi_params 8 uwsgi_params 8 win-utf 8

ID do processo pai (PPID) 29
PCRE
 por volta de 5
 URL 77
ajuste de desempenho 111
variável simples 16
proxy_cache_path, parâmetros inativos
 62 níveis 62 loader_iles 62
 loader_sleep 62 loader_threshold 62
 max_size 62 valores da diretiva
proxy_cache_use_stale, para
 argumentos 67 diretiva proxy 83
 diretiva proxy_limit_rate 110 diretiva
proxy_recursive 83

R

diretiva ranges 83 syntax
de expressão regular usada, nas
 regras de reescrita 76
compressão de resposta
 habilitando 114-116
diretiva de reescrita 72
mecanismo de reescrita
 sobre 71-74 práticas
 recomendadas 79 captura
 77 funcionalidades 77
 padrões 76, 77 parâmetros
 posicionais 77 avaliação
 de predicados, se as seções
 usadas 78, 79 respondendo,
 com código de status HTTP
 especificado 79

 reescrever regras, definindo 74, 75
 usando 77 variáveis, atribuindo 77
 regras de reescrita sobre 71
 definindo 74, 75

RHEL/CentOS/SL 6
URL 3

S

Diretiva do servidor Search Engine Optimization (SEO) 73
 parâmetros 101
bilhetes de sessão
 cerca de 122
 habilitando 122
sinais, Nginx
 HUP 30
 SAIR 30
PRAZO, INT 30
USR1 30
USR2 30
Cópia de
configuração de código-fonte
 WINCH 30, de pacotes pré-construídos 7, 8
certificados SSL
 Solicitação de assinatura de certificado
 (CSR), criando 44 instalando 44
 certificados SSL emitidos, instalando 45
 redirecionando, de host virtual não seguro
 46
Reutilização de sessão
 SSL permitindo
120-123 instância autônoma
28 recuperação de arquivo estático
 otimizando a estrutura
111-114, pasta de host virtual padrão
 de instalação do Nginx 8 pasta de
 log 9
Pasta de configuração do Nginx 8
pasta temporária 9 pasta de
configuração de hosts virtuais 9

T

gerenciamento de
 arquivos temporários 47, 48
Buffer Lookaside de Tradução (TLB) 123

DENTRO

Ubuntu

**Nginx, instalando 2
expressões unárias
16 operadores unários
-d 16 !-d 16 -e 16 !-e
16**

**-f
16 !-f
16 -x
16 !-x 16**

Dentro

**o trabalhador
processa cerca
de 26 alocando
40-42 alocando, em sistemas multi-core 123, 124**

Machine Translated by Google



Obrigado por comprar Nginx Essentials

Sobre a Publicação Packt

Packt, pronunciado 'packed', publicou seu primeiro livro, *Mastering phpMyAdmin for Effective MySQL Management*, em abril de 2004, e posteriormente continuou a se especializar na publicação de livros altamente focados em tecnologias e soluções específicas.

Nossos livros e publicações compartilham as experiências de seus colegas profissionais de TI na adaptação e personalização dos sistemas, aplicativos e estruturas atuais. Nossos livros baseados em soluções fornecem a você o conhecimento e o poder para personalizar o software e as tecnologias que você está usando para realizar o trabalho. Os livros Packt são mais específicos e menos gerais do que os livros de TI que você viu no passado. Nosso modelo de negócios exclusivo nos permite trazer a você informações mais focadas, dando a você mais do que você precisa saber e menos do que você não precisa.

A Packt é uma editora moderna e exclusiva que se concentra na produção de livros de qualidade e de ponta para comunidades de desenvolvedores, administradores e novatos.

Para obter mais informações, visite nosso site em www.packtpub.com.

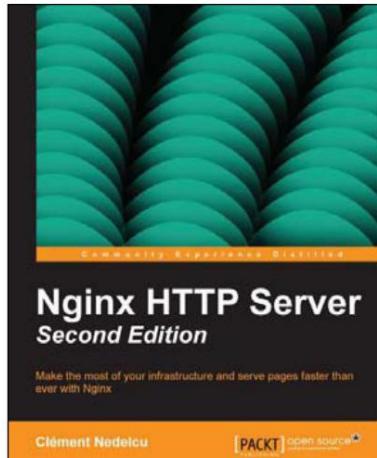
Sobre o Packt Open Source

Em 2010, a Packt lançou duas novas marcas, Packt Open Source e Packt Enterprise, para continuar seu foco na especialização. Este livro faz parte da marca Packt Open Source, que abriga livros publicados sobre software construído em torno de licenças de código aberto e oferece informações para qualquer pessoa, desde desenvolvedores avançados a web designers iniciantes. A marca Open Source também administra o Open Source Royalty Scheme da Packt, pelo qual Packt dá royalties a cada projeto de código aberto sobre cujo software um livro é vendido.

Escrevendo para Packt

Congratulamo-nos com todas as perguntas de pessoas interessadas em autoria. As propostas de livros devem ser enviadas para author@packtpub.com. Se a sua ideia de livro ainda está em estágio inicial e você gostaria de discuti-la antes de escrever uma proposta formal de livro, entre em contato conosco; um de nossos editores de comissionamento entrará em contato com você.

Não estamos apenas procurando autores publicados; Se você tem fortes habilidades técnicas, mas não tem experiência em redação, nossos editores experientes podem ajudá-lo a desenvolver uma carreira de escritor ou simplesmente obter alguma recompensa adicional por sua experiência.



Servidor HTTP Nginx

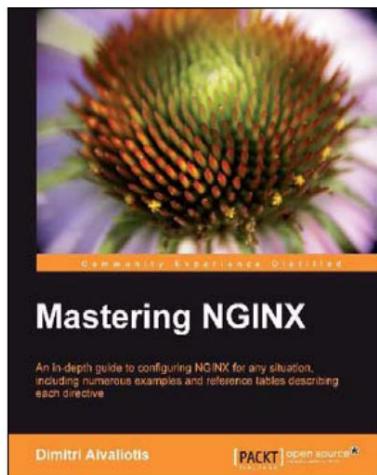
Segunda edição

ISBN: 978-1-78216-232-2

Brochura: 318 páginas

Aproveite ao máximo sua infraestrutura e veicule páginas mais rápido do que nunca com o Nginx

1. Diretiva de configuração completa e referência do módulo.
2. Descubra possíveis interações entre Nginx e Apache para obter o melhor dos dois mundos.
3. Aprenda a configurar seus servidores e hosts virtuais de forma eficiente.
4. Um guia passo a passo para mudar do Apache para o Nginx.



Dominando NGINX

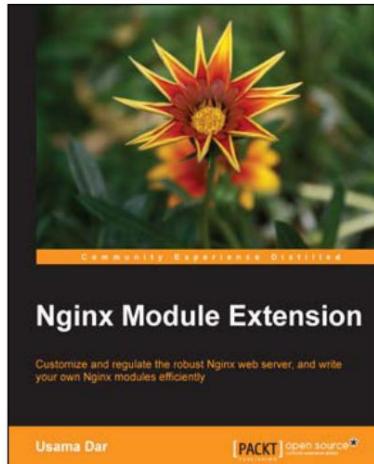
ISBN: 978-1-84951-744-7

Brochura: 322 páginas

Um guia detalhado para configurar o NGINX para qualquer situação, incluindo vários exemplos e tabelas de referência que descrevem cada diretiva

1. Um guia de configuração detalhado para ajudá-lo a entender como configurar melhor o NGINX para qualquer situação.
2. Inclui exemplos de código úteis para ajudá-lo a integrar o NGINX em sua arquitetura de aplicativo.
3. Cheio de trechos de configuração de exemplo, descrições de práticas recomendadas e tabelas de referência para cada diretiva.

Por favor, verifique www.PacktPub.com para obter informações sobre nossos títulos



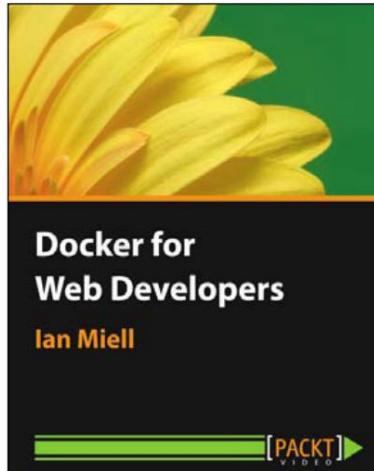
Extensão do módulo Nginx

ISBN: 978-1-78216-304-6

Brochura: 128 páginas

Personalize e regule o robusto servidor web Nginx e escreva seus próprios módulos Nginx com eficiência

1. Instale o Nginx de sua fonte em várias plataformas.
2. Familiarize-se com os módulos principais do Nginx e suas opções de configuração.
3. Explore o módulo opcional e de terceiros extensões junto com diretivas de configuração.



Docker para desenvolvedores web [Vídeo]

ISBN: 978-1-78439-067-9

Duração: 01:31 horas

Acelere suas habilidades de desenvolvimento web em projetos web reais em tempo recorde com o Docker

1. Turbine seu processo de desenvolvimento web enquanto garante que tudo funcione sem problemas.
2. Ganhe em 2048 usando a funcionalidade de confirmação e restauração do Docker.
3. Use o fluxo de trabalho do Docker Hub para automatizar o reconstrução de seus projetos web.
4. Cheio de exemplos realistas, esta é uma jornada passo a passo para se tornar um especialista em Docker!.

Por favor, verifique www.PacktPub.com para obter informações sobre nossos títulos

Machine Translated by Google