

Guia de instalação e configuração do Keycloak

Visão geral do guia

O objetivo deste guia é percorrer as etapas que precisam ser concluídas antes de inicializar o servidor Keycloak pela primeira vez. Se você só quiser testar o Keycloak, ele praticamente sai da caixa com seu próprio banco de dados incorporado e somente local. Para implantações reais que serão executadas em produção, você precisará decidir como você deseja gerenciar a configuração do servidor em tempo de execução (modo autônomo ou de domínio), configurar um banco de dados compartilhado para armazenamento Keycloak, configurar criptografia e HTTPS e finalmente configurar o Keycloak para executar em um cluster. Este guia percorre cada aspecto de quaisquer decisões de pré-inicialização e configuração que você deve fazer antes de implantar o servidor.

Uma coisa a notar particularmente é que o Keycloak é derivado do WildFly Application Server. Muitos aspectos da configuração do Keycloak giram em torno de elementos de configuração do WildFly. Muitas vezes este guia irá direcioná-lo para a documentação fora do manual se você quiser mergulhar em mais detalhes.

Documentação externa adicional recomendada

O Keycloak é construído em cima do servidor de aplicativos WildFly e seus sub-projetos como Infinispan (para cache) e Hibernate (para persistência). Este guia cobre apenas o básico para configuração em nível de infraestrutura. É altamente recomendável que você peruse a documentação para WildFly e seus subprojetos. Aqui está o link para a documentação:

- [Documentação WildFly 23](#)

instalação

Instalar o Keycloak é tão simples quanto baixá-lo e descompactá-lo. Este capítulo revisa os requisitos do sistema, bem como a estrutura do diretório da distribuição.

Requisitos do sistema

Estes são os requisitos para executar o servidor de autenticação Keycloak:

- Pode ser executado em qualquer sistema operacional que execute Java
- Java 8 JDK
- zip ou gzip e piche
- Pelo menos 512M de RAM
- Pelo menos 1G de espaço em disco
- Um banco de dados externo compartilhado como PostgreSQL, MySQL, Oracle, etc. Keycloak requer um banco de dados compartilhado externo se você quiser executar em um cluster. Consulte a seção de [configuração do banco de dados](#) deste guia para obter mais informações.
- Suporte multicast de rede em sua máquina se você quiser executar em um cluster. O keycloak pode ser agrupado sem multicast, mas isso requer um monte de alterações de configuração. Consulte a seção de [agrupamento](#) deste guia para obter mais informações.
- No Linux, recomenda-se usar `/dev/urandom` como uma fonte de dados aleatórios para evitar que o Keycloak seja pendurado devido à falta de entropia disponível, a menos que `/dev/urandom` seja exigido pela sua política de segurança. Para conseguir isso no Oracle JDK 8 e OpenJDK 8, defina a propriedade do sistema `java.security.egd` na inicialização para `arquivar:/dev/urandom`.

Instalação de arquivos de distribuição

O Keycloak Server tem duas distribuições para download:

- 'keycloak-15.0.0. [zip|tar.gz]'
- 'keycloak-sobreposição-15.0.0. [zip|tar.gz]'

O 'keycloak-15.0.0. [zip|tar.gz]' arquivo é a distribuição apenas do servidor. Ele não contém nada além dos scripts e binários para executar o Servidor Keycloak. Para desempacotar este arquivo basta executar os utilitários `descompactar` ou `gunzip` e alcatrão do seu sistema operacional.

O 'keycloak-overlay-15.0.0. [zip|tar.gz]' arquivo é um complemento WildFly que permite instalar o Keycloak Server em cima de uma distribuição WildFly existente. Não suportamos usuários que desejam executar seus aplicativos e Keycloak na mesma instância do servidor. Para instalar o Keycloak Service Pack, basta descompactá-lo no

diretório raiz de sua distribuição WildFly, abrir o diretório de bin em um shell e executar `./jboss-cli.[sh|bat] --file=keycloak-install.cli`.

Para desempacotar esses arquivos executar os utilitários `descompactar` ou `gunzip` e alcatrão.

Estrutura do Diretório de Distribuição

Este capítulo o acompanha na estrutura do diretório da distribuição do servidor.

Vamos examinar o propósito de alguns dos diretórios:

bin/

Isso contém vários scripts para inicializar o servidor ou executar alguma outra ação de gerenciamento no servidor.

domínio/

Isso contém arquivos de configuração e diretório de trabalho ao executar Keycloak no [modo de domínio](#).

módulos/

Estas são todas as bibliotecas Java usadas pelo servidor.

autônomo/

Isso contém arquivos de configuração e diretório de trabalho ao executar o Keycloak no [modo autônomo](#).

autônomo/implantações/

Se você está escrevendo extensões para Keycloak, você pode colocar suas extensões aqui. Consulte o [Guia do Desenvolvedor do Servidor](#) para obter mais informações sobre isso.

temas/

Este diretório contém todos os html, folhas de estilo, arquivos JavaScript e imagens usadas para exibir qualquer tela de interface do usuário exibida pelo servidor. Aqui você pode modificar um tema existente ou criar o seu próprio. Consulte o [Guia do Desenvolvedor do Servidor](#) para obter mais informações sobre isso.

Escolhendo um modo operacional

Antes de implantar o Keycloak em um ambiente de produção, você precisa decidir que tipo de modo de operação você vai usar. Você vai executar Keycloak dentro de um aglomerado? Deseja uma maneira centralizada de gerenciar as configurações do servidor? Sua escolha de modo operacional afeta a forma como você configura bancos de dados, configura cache e até mesmo como você inicializa o servidor.

O Keycloak é construído em cima do WildFly Application Server. Este guia só passará por cima de um modo específico. Se você quiser informações específicas sobre isso, um lugar melhor para

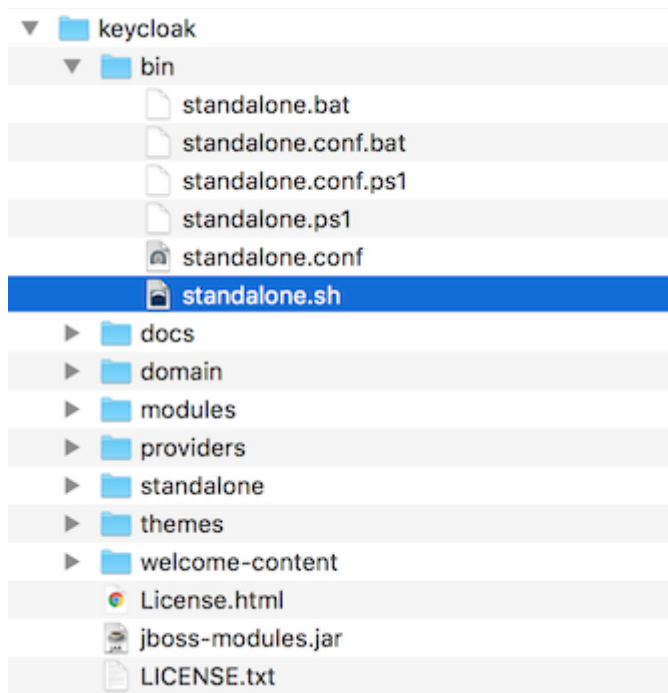
Modo autônomo

O modo de operação autônomo só é útil quando você deseja executar uma e apenas uma instância de servidor Keycloak. Não é utilizável para implantações em cluster e todos os caches não são distribuídos e somente locais. Não é recomendável que você use o modo autônomo na produção, pois você terá um único ponto de falha. Se o servidor do modo autônomo cair, os usuários não poderão fazer login. Este modo é realmente apenas útil para test drive e jogar com os recursos de Keycloak

Script de inicialização autônomo

Ao executar o servidor no modo autônomo, há um script específico que você precisa executar para inicializar o servidor, dependendo do seu sistema operacional. Esses scripts vivem no *bin/diretório* da distribuição do servidor.

Scripts de inicialização autônomos



Para inicializar o servidor:

Linux/Unix

```
$ .../bin/autônomo.sh
```

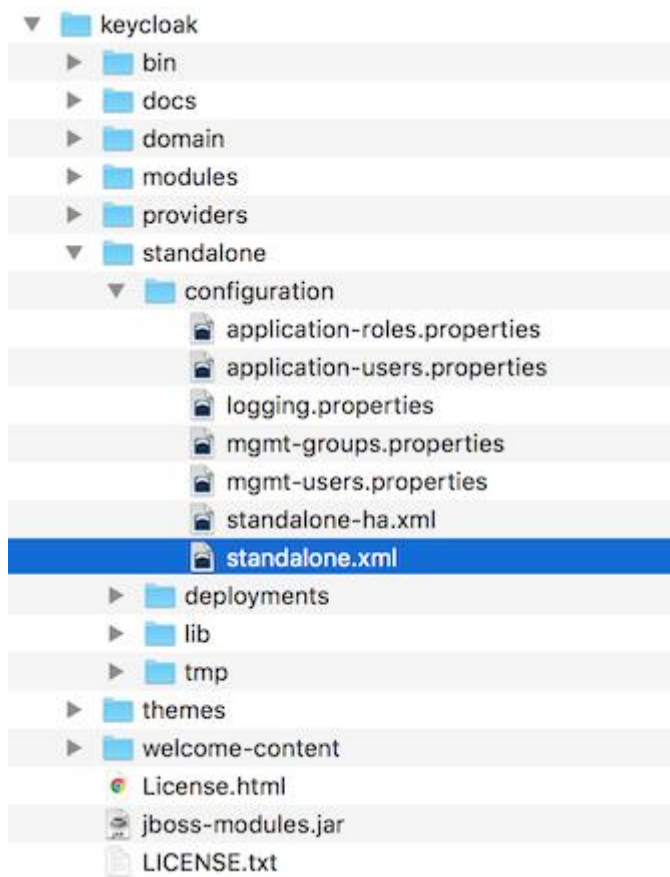
Windows

```
> .../bin/autônomo.bat
```

Configuração autônoma

A maior parte deste guia orienta como configurar aspectos do nível de infraestrutura do Keycloak. Esses aspectos são configurados em um arquivo de configuração específico para o servidor de aplicativos do que o Keycloak é um derivado. No modo de operação autônomo, este arquivo vive em `.../autônomo/configuração/autônomo.xml`. Este arquivo também é usado para configurar coisas de nível não-infra-estrutura que são específicas para componentes Keycloak.

Arquivo Config autônomo



Quaisquer alterações que você fizer neste arquivo enquanto o servidor estiver em execução serão substituídas pelo servidor. Em vez disso, use o scripting da linha de comando ou o console [WildFly 23](#) para obter mais informações.

Modo agrupado autônomo

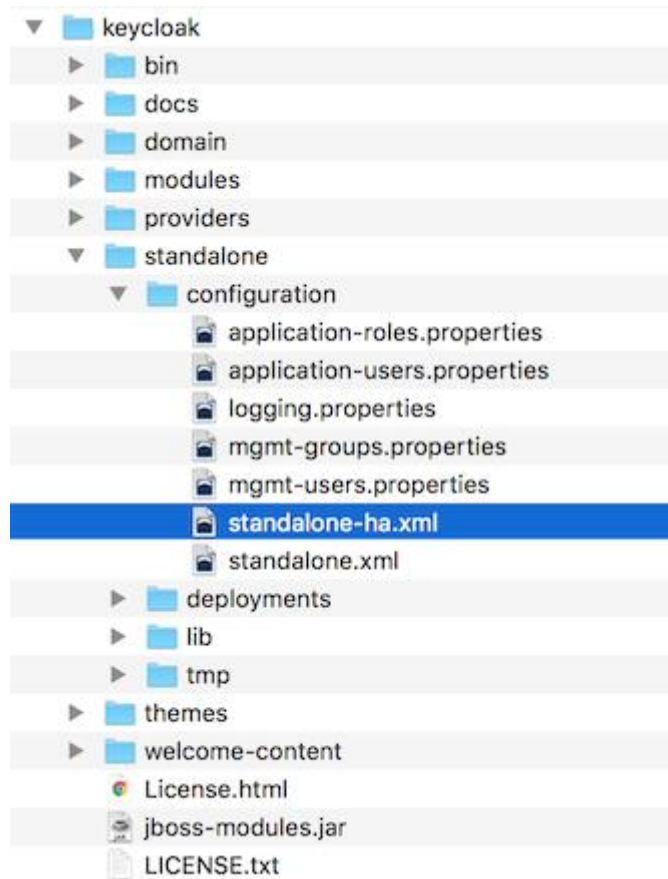
O modo de operação em cluster autônomo é para quando você deseja executar o Keycloak dentro de um cluster. Este modo requer que você tenha uma cópia da distribuição Keycloak em cada máquina que deseja executar uma instância de servidor. Este modo pode ser muito fácil de implantar inicialmente, mas pode se tornar bastante complicado. Para fazer uma alteração de configuração, você terá que modificar cada distribuição em cada máquina. Para um grande cluster, isso pode se tornar demorado e propenso a erros.

Configuração em cluster autônoma

A distribuição tem um arquivo de configuração de servidor de aplicativo em sua maioria pré-configurado para execução dentro de um cluster. Ele tem todas as configurações específicas de infraestrutura para rede, bancos de dados, caches e descoberta. Este arquivo reside em `.../autônomo/configuração/autônomo-ha.xml`. Faltam algumas coisas nesta configuração. Você não pode executar o Keycloak em um cluster sem configurar

uma conexão de banco de dados compartilhada. Você também precisa implantar algum tipo de balanceador de carga na frente do cluster. As seções [de clustering](#) e [banco de dados](#) deste guia o orientam através dessas coisas.

Autônomo HA Config

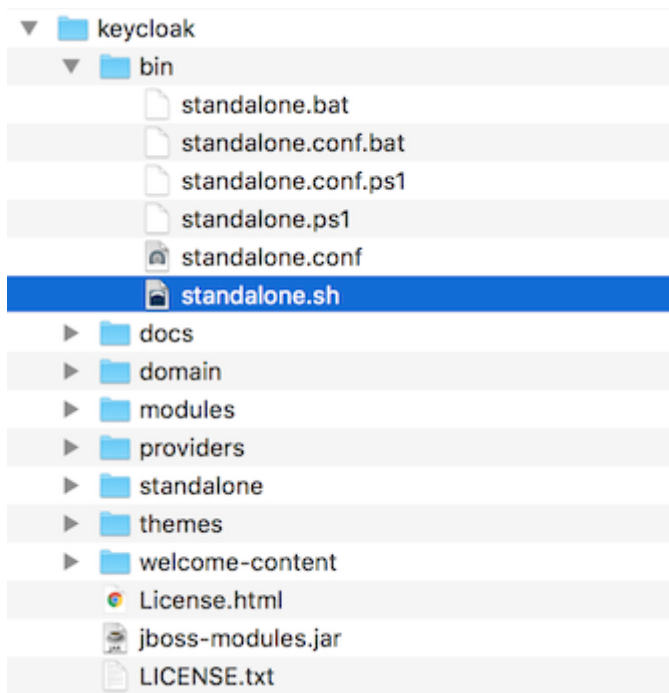


Quaisquer alterações que você fizer neste arquivo enquanto o servidor estiver em execução serão substituídas pelo servidor. Em vez disso, use o scripting da linha de comando ou o console para configurar o servidor. Consulte a [Documentação WildFly 23](#) para obter mais informações.

Script de inicialização agrupado autônomo

Você usa os mesmos scripts de inicialização para iniciar o Keycloak como você faz no modo autônomo. A diferença é que você passa em uma bandeira adicional para apontar para o arquivo ha config.

Scripts de inicialização agrupados autônomos



Para inicializar o servidor:

Linux/Unix

```
$ .../bin/autônomo.sh --servidor-config=standalone-ha.xml
```

Windows

```
> ...\bin\autônomo.bat --servidor-config=standalone-ha.xml
```

Modo clustered de domínio

O modo domínio é uma maneira de gerenciar e publicar centralmente a configuração para seus servidores.

Executar um cluster no modo padrão pode rapidamente se tornar agravante à medida que o cluster cresce em tamanho. Toda vez que você precisa fazer uma mudança de configuração, você tem que executá-lo em cada nó no cluster. O modo domínio resolve esse problema fornecendo um local central para armazenar e publicar configurações. Pode ser bastante complexo de configurar, mas vale a pena no final. Esse recurso é incorporado ao WildFly Application Server do qual o Keycloak deriva.

O guia passará pelo básico do modo de domínio. Passos detalhados sobre como configurar devem ser obtidos a partir da [Documentação WildFly 23](#).

Aqui estão alguns dos conceitos básicos de execução no modo de domínio.

controlador de domínio

O controlador de domínio é um processo responsável por armazenar, gerenciar e publicar a configuração geral de cada nó no cluster. Este processo é o ponto central a partir do qual os nós em um cluster obtêm sua configuração.

controlador de host

O controlador host é responsável pelo gerenciamento de instâncias do servidor em uma máquina específica. Você configura-o para executar uma ou mais instâncias de servidor. O controlador de domínio também pode interagir com os controladores host em cada máquina para gerenciar o cluster. Para reduzir o número de processos em execução, um controlador de domínio também atua como um controlador host na máquina em que ele é executado.

perfil de domínio

Um perfil de domínio é um conjunto de configuração nomeado que pode ser usado por um servidor para inicializar. Um controlador de domínio pode definir vários perfis de domínio que são consumidos por diferentes servidores.

grupo de servidores

Um grupo de servidores é uma coleção de servidores. Eles são gerenciados e configurados como um só. Você pode atribuir um perfil de domínio a um grupo de servidor e todos os serviços desse grupo usarão esse perfil de domínio como sua configuração.

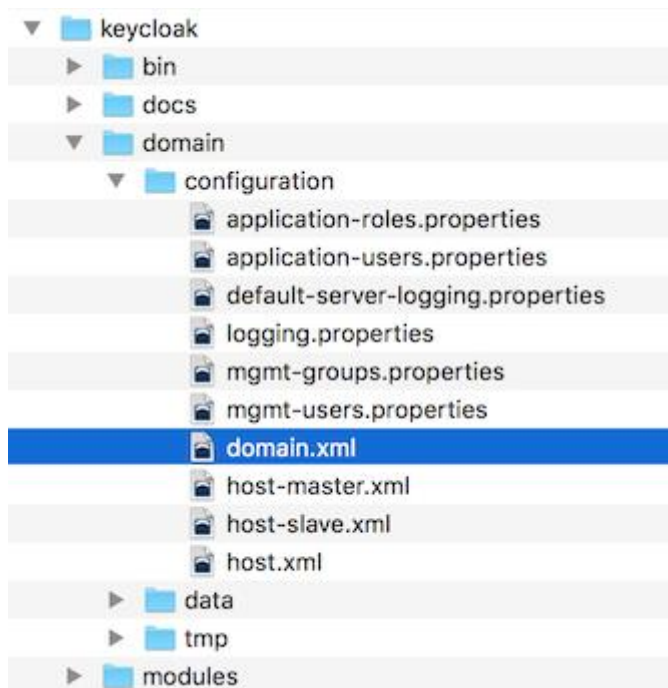
No modo de domínio, um controlador de domínio é iniciado em um nó mestre. A configuração do cluster reside no controlador de domínio. Em seguida, um controlador host é iniciado em cada máquina no cluster. Cada configuração de implantação do controlador de host especifica quantas instâncias de servidor Keycloak serão iniciadas naquela máquina. Quando o controlador host é inicializado, ele inicia o máximo de instâncias do servidor Keycloak como foi configurado para fazer. Essas instâncias do servidor puxam sua configuração do controlador de domínio.

Em alguns ambientes, como o Microsoft Azure, o modo de domínio não é aplicável. Consulte

Configuração de domínio

Vários outros capítulos neste guia o guiam configurando vários aspectos como bancos de dados, conexões de rede HTTP, caches e outras coisas relacionadas à infraestrutura. Enquanto o modo autônomo usa o arquivo *autônomo.xml* para configurar essas coisas, o modo de domínio usa o arquivo de configuração *.../domínio/configuração/domínio.xml*. É aí que o perfil de domínio e o grupo de servidor do servidor Keycloak são definidos.

domínio.xml



Quaisquer alterações que você fizer neste arquivo enquanto o controlador de domínio estiver rodando podem até ser substituídas pelo servidor. Em vez disso, use o scripting da linha de comando [Documentação WildFly 23](#) para obter mais informações.

Vamos olhar para alguns aspectos deste *arquivo .xml domínio*. Os blocos XML de perfil de perfil autônomo e com cluster de servidor auth estão onde você vai tomar a maior parte de suas decisões de configuração. Você estará configurando coisas aqui, como conexões de rede, caches e conexões de banco de dados.

perfil auth-server

```
<>
<profile nome=> "auth-server-standalone"
...
</perfil>
<profile nome=> "auth-server-clustered"
...
</perfil>
```

O perfil autônomo do servidor auth é uma configuração não agrupada. O perfil agrupado por auth-server é a configuração em cluster.

Se você rolar mais para baixo, você verá vários grupos de ligação de soquete definidos.

soquete-vinculativos-grupos

```
<socket-binding-grupos>
<socket-vinculato nome do grupo="standard-sockets" default-interface="público">
...
</grupo de ligação de soquete>
```

```

<socket-vinculato nome do grupo="ha-sockets" interface padrão="público">
    ...
</grupo de ligação de soquete>
    <!-- tomadas de balanceador de carga devem ser removidas em sistemas de
produção e substituídas por um melhor software ou hardware baseado em >
<socket-vinculato nome do grupo="load-balancer-sockets" interface padrão="público">
    ...
</grupo de ligação de soquete>
</grupos de ligação de soquete>

```

Essa confirmação define os mapeamentos de porta padrão para vários conectores abertos com cada instância de servidor Keycloak. Qualquer valor que contenha \${...} é um valor que pode ser substituído na linha de comando com o interruptor -D, ou seja.

```
$ domain.sh -Djboss.http.port=80
```

A definição do grupo de servidor para Keycloak reside no bloco XML dos grupos de servidores. Ele especifica o perfil de domínio usado (padrão) e também alguns argumentos de inicialização padrão para o Java VM quando o controlador host inicializa uma instância. Ele também liga um grupo de vinculação de soquetes ao grupo de servidores.

grupo de servidores

```

<serviente>
    <!-- grupo de balanceador de carga deve ser removido em sistemas de produção e
substituído por um melhor software ou hardware baseado em >
<doce do grupo de servidor= perfil "grupo de balanceador de carga" = "balanceador de
carga">
    <jvm nome = "padrão">
tamanho <heap = "64m" tamanho máximo="512m"/>
    </jvm>
    <socket-binding-group ref="load-balancer-sockets"/>
</grupo de servidores>
<do nome do grupo do servidor= perfil "auth-server-group" => "auth-server-clustered"
    <jvm nome = "padrão">
tamanho <heap = "64m" tamanho máximo="512m"/>
    </jvm>
    <socket-binding-group ref="ha-sockets"/>
</grupo de servidores>
</grupos de servidores>

```

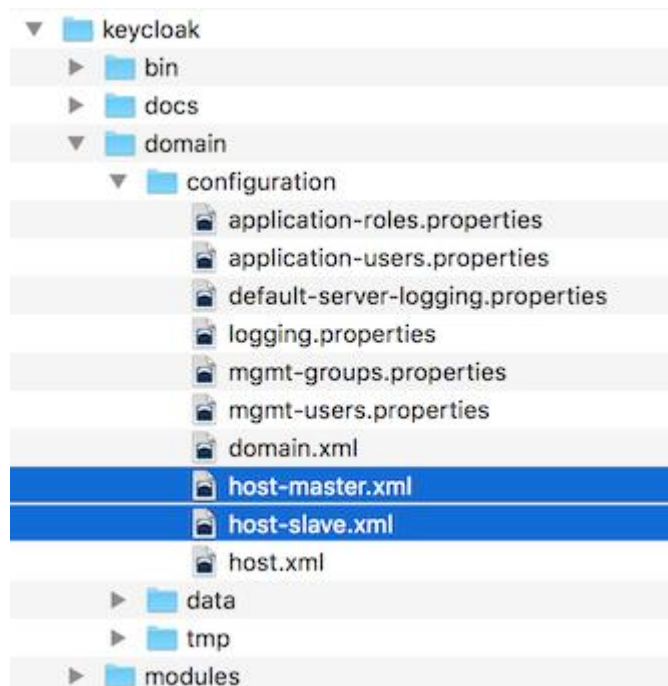
Configuração do controlador de host

O Keycloak vem com dois arquivos de configuração do controlador de host que residem no .../domínio/configuração/diretório: *host-master.xml* e *host-slave.xml*. *host-master.xml* está configurado para inicializar um controlador de domínio, um balanceador de carga e uma instância de servidor Keycloak. *host-slave.xml* está

configurado para falar com o controlador de domínio e inicializar uma instância de servidor Keycloak.

O balanceador de carga não é um serviço obrigatório. Ele existe para que você possa facilmente desenvolver. Embora utilizável na produção, você tem a opção de substituí-lo por um baseado em hardware ou software diferente que deseja usar.

Controlador de host Config



Para desativar a instância do servidor balanceador de carga, edite o *host-master.xml* e comente ou remova a entrada "balanceador de carga".

```
<servers>
  <!-- remover ou comentar a próxima linha -->
  <servidor nome = grupo "balanceador de carga" = "loadbalancer-group"/>
  ...
</servidores>
```

Outra coisa interessante a ser notou sobre este arquivo é a declaração da instância do servidor de autenticação. Tem uma configuração de deslocamento de porta. Qualquer porta de rede definida no *domínio.xml* grupo de vinculação de tomadas ou o grupo de servidor terá o valor de compensação da porta adicionado a ele. Para esta configuração de domínio de exemplo, fazemos isso para que as portas abertas pelo servidor balanceador de carga não entrem em conflito com a instância do servidor de autenticação que é iniciada.

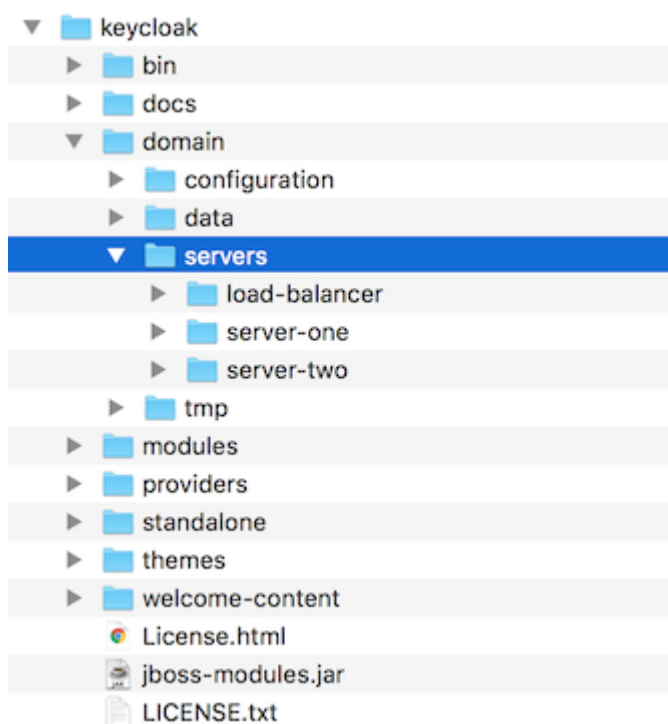
```
<servers>
  ...
  <servidor nome= grupo "servidor-um" = "auth-server-group" auto-start="true">
```

```
<socket-vincula-ses port-offset="150"/>
</servidor>
</servidores>
```

Diretórios de trabalho de instância do servidor

Cada instância do servidor Keycloak definida em seus arquivos host cria um diretório de trabalho em `.../domínio/servidores/{NOME DO SERVIDOR}`. Configuração adicional pode ser colocada lá, e qualquer arquivo temporário, log ou de dados que a instância do servidor precisa ou cria ir lá também. A estrutura desses diretórios por servidor acaba parecendo com qualquer outro servidor inicializado pelo WildFly.

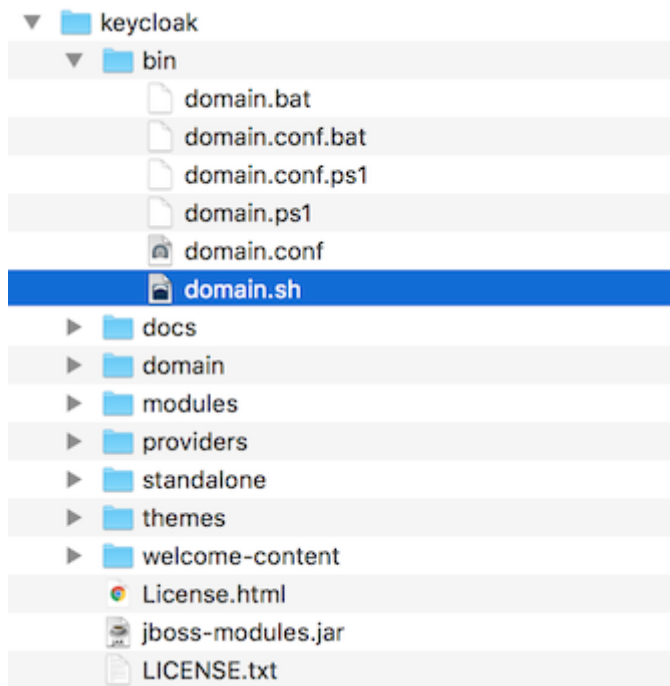
Diretórios de Trabalho



Script de inicialização de domínio

Ao executar o servidor no modo de domínio, há um script específico que você precisa executar para inicializar o servidor dependendo do seu sistema operacional. Esses scripts vivem no `bin/diretório` da distribuição do servidor.

Script de inicialização de domínio



Para inicializar o servidor:

Linux/Unix

```
$ .../bin/domínio.sh --host-config=host-master.xml
```

Windows

```
> ...\\bin\\domínio.bat --host-config=host-master.xml
```

Ao executar o script de inicialização, você precisará passar no arquivo de configuração de controle do host que você usará através do switch `--host-config`.

Exemplo de domínio agrupado

Você pode testar o clustering do drive usando a configuração de domínio fora da *caixa.xml*. Este domínio de exemplo é feito para ser executado em uma máquina e inicializado:

- um controlador de domínio
- um balanceador de carga HTTP
- 2 Instâncias do servidor Keycloak

Para simular a execução de um cluster em duas máquinas, você precisará executar o script `domain.sh` duas vezes para iniciar dois controladores host separados. O primeiro será o controlador de host mestre que iniciará um controlador de domínio, um balanceador de carga HTTP e uma instância de servidor de autenticação Keycloak. O segundo será um controlador de host escravo que só inicia uma instância de servidor de autenticação.

Configuração conexão de escravos ao controlador de domínio

Antes de iniciar as coisas, porém, você tem que configurar o controlador de host escravo para que ele possa falar com segurança com o controlador de domínio. Se você não fizer isso, então o host escravo não será capaz de obter a configuração centralizada a partir do controlador de domínio. Para configurar uma conexão segura, você tem que criar um usuário administrativo do servidor e um segredo que será compartilhado entre o mestre e o escravo. Você faz isso executando o `.../bin/adicionar-user.sh` script.

Quando você executa o script selecione **Usuário de gerenciamento** e responda **sim** quando ele perguntar se o novo usuário será usado para um processo AS para se conectar a outro. Isso gerará um segredo que você precisará cortar e colar no arquivo `.../domínio/configuração/host-slave.xml`.

Adicionar administrador do servidor de aplicativos

```
$ add-user.sh
```

Que tipo de usuário deseja adicionar?

- a) Usuário de gerenciamento (mgmt-users.properties)
- b) Usuário do aplicativo (usuários de aplicativos.propriedades)

a): um

Digite os detalhes do novo usuário para adicionar.

Usando o 'ManagementRealm' como descoberto a partir dos arquivos de propriedade existentes.

Nome de usuário : administrador

As recomendações de senha estão listadas abaixo. Para modificar essas restrições, edite o arquivo de configuração `add-user.properties`.

- A senha não deve ser um dos seguintes valores restritos {raiz, administrador, administrador}
- A senha deve conter pelo menos 8 caracteres, 1 caracteres alfabéticos, 1 dígito(s), 1 símbolo não alfanumérico(s)
- A senha deve ser diferente do nome de usuário

senha:

Re-digitar senha:

A quais grupos você deseja que esse usuário pertença? (Por favor, insira uma lista separada de vírgula ou deixe em branco para nenhum) []:

Prestes a adicionar 'administrador' do usuário para o 'ManagementRealm'

Isso está correto sim/não? Sim

Adicionado 'administrador' do usuário ao arquivo `'/.../autônomo/configuração/mgmt-users.properties'`

Adicionado 'administrador' do usuário ao arquivo `'/.../domínio/configuração/mgmt-users.'`

Adicionado 'administrador' do usuário com grupos para arquivar `'/.../autônomo/configuração/mgmt-groups.propriedades'`

Adicionado 'administrador' do usuário com grupos para arquivar `'/.../domínio/configuração/mgmt-groups.propriedades'`

Este novo usuário será usado para um processo AS para se conectar a outro processo AS?

por exemplo, para um controlador de host escravo conectado ao mestre ou para uma conexão remoting para chamadas EJB do servidor para servidor.

Sim/não? Sim

Para representar o usuário, adicione o seguinte à definição de identidades do servidor
<secret value="bWdtdDEyMyE=" />

O add-user.sh não adiciona usuário ao servidor Keycloak, mas à plataforma de aplicativos e credenciais usadas e geradas no script acima são apenas para fins de exemplo. Por favor, u

Em seguida, corte e cole o valor secreto no arquivo *.../domínio/configuração/host-slave.xml* da seguinte forma:

```
< gestão>
  <ssegurança>
    nome <security-realm ="ManagementRealm">
      <identes>
        <secret valor ="bWdtdDEyMyE="/>
      </identidades do servidor>
```

Você também precisará adicionar o *nome* de usuário do usuário criado no arquivo *.../domínio/configuração/host-slave.xml*:

```
<remto de segurança -reino=nome de usuário"ManagementRealm" ="administrador">
```

Execute os scripts de inicialização

Já que estamos simulando um cluster de dois nós em uma máquina de desenvolvimento, você executará o script de inicialização duas vezes:

Boot up mestre

```
$ domain.sh --host-config=host-master.xml
```

Bota até escravo

```
$ domain.sh --host-config=host-slave.xml
```

Para experimentá-lo, abra seu navegador e vá para <http://localhost:8080/auth>.

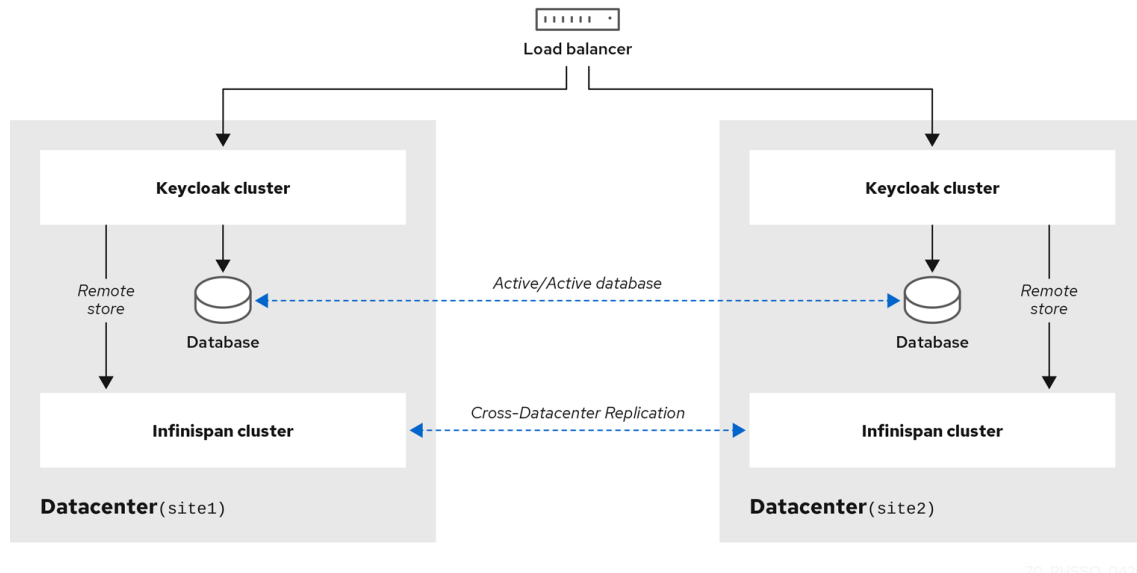
Modo de replicação do datacenter cruzado

O modo de replicação do datacenter cruzado é **visualização de** tecnologia e não é totalme

O modo de replicação do datacenter cruzado permite que você execute o Keycloak em um cluster em vários data centers, geralmente usando sites de data center que estão em diferentes regiões geográficas. Ao usar este modo, cada data center terá seu próprio cluster de servidores Keycloak.

Esta documentação se referirá ao diagrama de arquitetura de exemplo a seguir para ilustrar e descrever um simples caso de uso de replicação de datacenter cruzado.

Diagrama de arquitetura de exemplo



Pré-requisitos

Como este é um tópico avançado, recomendamos que você leia pela primeira vez o seguinte, que fornece conhecimentos de fundo valiosos:

- [Agrupamento com keycloak](#) Ao configurar a replicação do datacenter cruzado, você usará clusters Keycloak mais independentes, então você deve entender como um cluster funciona e os conceitos e requisitos básicos, como balanceamento de carga, bancos de dados compartilhados e multicasting.
- [A replicação entre sites infinispán](#) replica dados em clusters em locais geográficos separados.

Detalhes técnicos

Esta seção fornece uma introdução aos conceitos e detalhes de como a replicação do keycloak cross-datacenter é realizada.

dados

Keycloak é uma aplicação imponente. Ele usa o seguinte como fontes de dados:

- Um banco de dados é usado para persistir dados permanentes, como informações do usuário.

- Um cache Infinispan é usado para armazenar dados persistentes do banco de dados e também para salvar alguns metadados de curta duração e frequentemente mudando, como para sessões de usuário. O Infinispan geralmente é muito mais rápido do que um banco de dados, no entanto, os dados salvos usando o Infinispan não são permanentes e não se espera que persistam entre as reinicializações de cluster.

Em nossa arquitetura de exemplo, existem dois data centers chamados `site1` e `site2`. Para a replicação do centro de dados cruzados, devemos garantir que ambas as fontes de dados funcionem de forma confiável e que os servidores Keycloak do `site1` sejam eventualmente capazes de ler os dados salvos pelos servidores Keycloak no `site2`.

Com base no ambiente, você tem a opção de decidir se prefere:

- Confiabilidade - que é normalmente usada no modo Ativo/Ativo. Os dados escritos no `site1` devem ser visíveis imediatamente no `site2`.
- Desempenho - que normalmente é usado no modo Ativo/Passivo. Os dados escritos no `site1` não precisam ser visíveis imediatamente no `site2`. Em alguns casos, os dados podem não ser visíveis no `site2`.

Para obter mais detalhes, consulte [Modos](#).

Processamento de solicitação

O navegador de um usuário final envia uma solicitação HTTP para o [balanceador de carga frontal](#). Este balanceador de carga geralmente é HTTPD ou WildFly com `mod_cluster`, NGINX, HA Proxy, ou talvez algum outro tipo de software ou balanceador de carga de hardware.

O balanceador de carga então encaminha as solicitações HTTP que recebe para as instâncias keycloak subjacentes, que podem ser espalhadas entre vários data centers. Os balanceadores de carga normalmente oferecem suporte para [sessões pegajosas](#), o que significa que o balanceador de carga é capaz de sempre encaminhar todas as solicitações HTTP do mesmo usuário para a mesma instância Keycloak no mesmo data center.

Solicitações HTTP que são enviadas de aplicativos de clientes para o balanceador de carga são chamadas de solicitações de backchannel. Estes não são vistos pelo navegador de um usuário final e, portanto, não podem fazer parte de uma sessão pegajosa entre o usuário e o balanceador de carga. Para solicitações de backchannel, o loadbalancer pode encaminhar a solicitação HTTP para qualquer instância Keycloak em qualquer data center. Isso é desafiador, pois alguns fluxos OpenID Connect e alguns SAML requerem várias solicitações HTTP tanto do usuário quanto do aplicativo. Como não podemos depender de sessões pegajosas para forçar todas as solicitações relacionadas a serem enviadas para a mesma instância keycloak no mesmo data center, devemos, em vez disso, replicar alguns dados através de data centers, para que os dados sejam vistos por solicitações HTTP subsequentes durante um fluxo específico.

Modos

De acordo com seus requisitos, existem dois modos de operação básicos para a replicação do datacenter cruzado:

- Ativo/Passivo - Aqui os usuários e aplicativos do cliente enviam as solicitações apenas para os nodes keycloak em apenas um data center. O segundo data center é usado apenas como backup para salvar os dados. Em caso de falha no data center principal, os dados geralmente podem ser restaurados a partir do segundo data center.
- Ativo/Ativo - Aqui os usuários e aplicativos do cliente enviam as solicitações para os nós keycloak em ambos os data centers. Isso significa que os dados precisam ser visíveis imediatamente em ambos os sites e disponíveis para serem consumidos imediatamente dos servidores Keycloak em ambos os sites. Isso é especialmente verdade se o servidor Keycloak escrever alguns dados no site1, e é necessário que os dados estejam disponíveis imediatamente para leitura pelos servidores Keycloak no site2 imediatamente após o término da gravação no site1.

O modo ativo/passivo é melhor para o desempenho. Para obter mais informações sobre como configurar caches para ambos os modos, consulte: [backups SYNC ou ASYNC](#).

base de dados

A Keycloak usa um sistema de gerenciamento de banco de dados relacional (RDBMS) para persistir alguns metadados sobre reinos, clientes, usuários e assim por diante. Consulte [este capítulo](#) do guia de instalação do servidor para obter mais detalhes. Em uma configuração de replicação de datacenter cruzado, assumimos que ambos os data centers conversam com o mesmo banco de dados ou que cada data center tem seu próprio nó de banco de dados e ambos os nós de banco de dados são sincronizados replicados nos data centers. Em ambos os casos, é necessário que quando um servidor Keycloak no site1 persista alguns dados e cometa a transação, esses dados são imediatamente visíveis pelas transações subsequentes de DB no site2.

Os detalhes da configuração do DB estão fora de escopo para o Keycloak, no entanto, muitos fornecedores de RDBMS como MariaDB e Oracle oferecem bancos de dados replicados e replicação síncrona. Testamos keycloak com esses fornecedores:

- Banco de dados Oracle 19c RAC
- Aglomerado Galera 3.12 para servidor MariaDB versão 10.1.19-MariaDB

Caches infinispn

Esta seção começa com uma descrição de alto nível dos caches Infinispn. Mais detalhes da configuração do cache seguem.

Sessões de autenticação

Em Keycloak temos o conceito de sessões de autenticação. Existe um cache Infinispan separado chamado `authenticationSessions` usados para salvar dados durante a autenticação de determinado usuário. As solicitações deste cache geralmente envolvem apenas um navegador e o servidor Keycloak, não o aplicativo. Aqui podemos confiar em sessões pegajosas e o conteúdo de cache de `authenticationSessions` não precisa ser replicado em data centers, mesmo que você esteja no modo Ativo/Ativo.

Tokens de ação

Também temos o conceito de tokens de [ação](#), que são usados normalmente para cenários quando o usuário precisa confirmar uma ação assincronicamente por e-mail. Por exemplo, durante o fluxo de senha de esquecimento, o cache `actionTokens` Infinispan é usado para rastrear metadados sobre tokens de ação relacionados, como qual token de ação já foi usado, por isso não pode ser reutilizado pela segunda vez. Isso geralmente precisa ser replicado em data centers.

Cache e invalidação de dados persistentes

O Keycloak usa o Infinispan para armazenar dados persistentes para evitar muitas solicitações desnecessárias ao banco de dados. O cache melhora o desempenho, porém adiciona um desafio adicional. Quando algum servidor Keycloak atualiza qualquer dado, todos os outros servidores Keycloak em todos os data centers precisam estar cientes disso, para que eles invalidem dados específicos de seus caches. O Keycloak usa caches Infinispan locais chamados `reinos`, `usuários` e `autorização` para armazenar dados persistentes.

Usamos um cache separado, `trabalho`, que é replicado em todos os data centers. O cache de trabalho em si não armazena dados reais. É usado apenas para enviar mensagens de invalidação entre nós de cluster e data centers. Em outras palavras, quando os dados são atualizados, como o usuário `john`, o Keycloak envia a mensagem de invalidação para todos os outros nós de cluster no mesmo data center e também para todos os outros data centers. Após receber o aviso de invalidação, cada nó invalida os dados apropriados de seu cache local.

Sessões de usuários

Existem caches Infinispan chamados `sessions`, `clientSessions`, `offlineSessionse` `offlineClientSessions`, todos os quais geralmente precisam ser replicados em data centers. Esses caches são usados para salvar dados sobre sessões de usuário, que são válidos pelo tempo de sessão do navegador de um usuário. Os caches devem lidar com as solicitações HTTP do usuário final e do aplicativo. Como descrito acima, as sessões pegajosas não podem ser usadas de forma confiável neste caso, mas ainda queremos garantir que as solicitações HTTP subsequentes possam ver os dados mais recentes. Por essa razão, os dados geralmente são replicados em data centers.

Proteção contra a força bruta

Finalmente, o cache `loginFailures` é usado para rastrear dados sobre logins com falha, como quantas vezes o usuário digitou uma senha ruim. Os detalhes são descritos aqui. Cabe ao administrador se esse cache deve ser replicado em data centers. Para ter uma contagem precisa de falhas de login, a replicação é necessária. Por outro lado, não replicar esses dados pode salvar algum desempenho. Portanto, se o desempenho for mais importante do que contagem precisa de falhas de login, a replicação pode ser evitada.

Para obter mais detalhes sobre como os caches podem ser configurados, consulte [Sintonizando a configuração de cache Infinispan](#).

Detalhes de comunicação

O Keycloak usa vários clusters separados de caches Infinispan. Cada nó Keycloak está no cluster com os outros nós keycloak no mesmo data center, mas não com os nós keycloak em data centers diferentes. Um nó Keycloak não se comunica diretamente com os nós keycloak de diferentes data centers. Os nós Keycloak usam servidores Infinispan externos para comunicação em data centers. Isso é feito usando o [protocolo Infinispan Hot Rod](#).

Os caches Infinispan do lado Keycloak usam a configuração `remoteStore` para descarregar dados para um cluster Infinispan remoto. Os clusters infinispan em data centers separados replicam esses dados para garantir que eles são backup.

O servidor Infinispan receptor notifica os servidores Keycloak em seu cluster através de Clientes Ouvintes, que são uma característica do protocolo Hot Rod. Os nós keycloak no site2 atualizam seus caches Infinispan e a sessão de usuário em particular também é visível em os nodes keycloak no site2.

Consulte o [Diagrama de Arquitetura de Exemplo](#) para obter mais detalhes.

Configuração cross DC com Infinispan 11.0.9

Use os seguintes procedimentos para o Infinispan 11.0.9 para executar uma configuração básica da replicação do Cross-Datacenter.

Este exemplo para Infinispan 11.0.9 envolve dois data centers, `site1` e `site2`. Cada data center consiste em 1 servidor Infinispan e 2 servidores Keycloak. Vamos acabar com 2 servidores Infinispan e 4 servidores Keycloak no total.

- O Site1 é composto por servidores Infinispan, `server1` e 2 servidores Keycloak, `node11` e `node12`.
- O Site2 é composto por servidores Infinispan, `server2` e 2 servidores Keycloak, `node21` e `node22`.

- Os servidores Infinispan Server1 e server2 estão conectados entre si através do protocolo RELAY2 e caches Infinispan baseados em backup de forma semelhante à descrita na [documentação Infinispan](#).
- Os servidores Keycloak node11 e node12 formam um cluster entre si, mas não se comunicam diretamente com nenhum servidor no site2. Eles se comunicam com o servidor Infinispan1 usando o protocolo Hot Rod (cache remoto). Consulte [os detalhes da Comunicação](#) para obter mais informações.
- Os mesmos detalhes se aplicam ao nó21 e ao nó22. Eles se agrupam entre si e se comunicam apenas com o servidor server2 usando o protocolo Hot Rod.

Nossa configuração de exemplo pressupõe que os quatro servidores Keycloak conversem com o mesmo banco de dados. Na produção, recomendamos que você use bancos de dados sincronizados separados em data centers descritos no [Banco de Dados](#).

Configuração de servidores Infinispan

Para a replicação do Cross-Datacenter, você começa criando clusters Infinispan remotos que podem fazer backup de dados do Keycloak.

Pré-requisitos

- Baixe e instale o Infinispan Server 11.0.9.

O Servidor Infinispan 11.0.9 requer Java 11.

procedimento

1. Crie um usuário para autenticar conexões de clientes da Infinispan, por exemplo:

```
$ bin/cli.sh usuário criar myuser -p "qwer1234!"
```

Você especifica essas credenciais na configuração do cliente Hot Rod quando v

2. Crie uma loja de chaves SSL e um truststore para proteger conexões entre Infinispan e Keycloak, por exemplo:
 - a. Crie um armazenamento de chaves para fornecer uma identidade SSL ao seu cluster Infinispan


```
keytool -genkey -alias server -keyalg RSA -keystore server.jks -keysize 2048
```
 - b. Exporte um certificado SSL da loja de chaves.


```
keytool -exportcert -keystore server.jks -alias server -file server.crt
```
 - c. Importe o certificado SSL em uma loja de confiança que a Keycloak pode usar para verificar a identidade SSL da Infinispan.

```
keytool -importcert -keystore truststore.jks -alias server -file  
server.crt
```

- d. Remover server.crt.

```
rm server.crt
```

Configuração de clusters Infinispan

Configure clusters Infinispan para replicar dados Keycloak em data centers.

Pré-requisitos

- Instale e configure o Infinispan Server.

procedimento

1. Abra `infinispan.xml` para edição.

Por padrão, o Infinispan Server usa `servidor/conf/infinispan.xml` para configuração estática, como mecanismos de transporte e segurança de clusters.

2. Crie uma pilha que use o TCPING como protocolo de detecção de cluster.

```
3. <secoto="cluster global" estende-se ="tcp">  
4.   <!-- remover o protocolo MPING da pilha e adicionar TCPING -->  
5.   <TCPING initial_hosts ="server1[7800],server2[7800]"  
6.   stack.combine="REPLACE" stack.position="MPING"/>  
   </pilha>
```

Lista os nomes do host para `server1` e `server2`.

7. Configure o transporte de cluster Infinispan para executar a replicação do Centro de Dados Cruzado.
 - a. Adicione o protocolo RELAY2 a uma pilha JGroups.

```
b. <jgroups>  
c.   <seco="xsite" estende="udp">  
d.   <relay. SiteRELAY2="site1"  
e.   max_site_masters ="1000"/>  
f.   <remote-sites default-stack="global-cluster">  
g.     <remote-site nome="site1"/>  
h.     <remote-site nome ="site2"/>  
i.   </remotas>  
j.   </pilha>  
   </jgroups>
```

Cria uma pilha chamada `xsite` que estende o transporte padrão de cluster UDP.

Adiciona o protocolo RELAY2 e nomeia o cluster que você está configurando como site1. O nome do site deve ser exclusivo de cada cluster Infinispan.

Define 1000 como o número de nós de relé para o cluster. Você deve definir um valor igual ou maior do que o número máximo de nós em seu cluster Infinispan.

Nomeia todos os clusters Infinispan que armazenam caches com dados Infinispan e usa a pilha TCP padrão para transporte entre clusters.

k. Configure o transporte de cluster Infinispan para usar a pilha.

```
l. <cache-container nome = estatísticas"padrão" = "verdadeiro">  
m. <support cluster="{infinispan.cluster.name:cluster}"  
n.     pilha="xsite"/>  
    </contêiner de cache>
```

Usa a pilha de xsite para o cluster.

8. Configure o armazenamento de chaves como uma identidade SSL no domínio de segurança do servidor.

```
9. <identites>  
10.     <ssl>  
11.         <desajato = "server.jks"  
12.             relativo a="infinispan.server.config.path"  
13.             keystore-senha="senha"  
14.             alias="servidor" />  
15.     </ssl>  
    </identidades do servidor>
```

Especifica o caminho do armazenamento de chaves que contém a identidade SSL.

Especifica a senha para acessar o armazenamento de chaves.

Nomeia o pseudônimo do certificado na loja de chaves.

16. Configure o mecanismo de autenticação para o ponto final do Hot Rod.

```
17.     <endPontos de vinculação de soquete="padrão">  
18.         nome do conector<hotrod="hotrod">  
19.             <autenticação>  
20.                 <sasl mecanismos="SCRAM-SHA-512"  
21.                     nome do servidor="infinispan" />  
22.             </autenticação>  
23.         </hotrod-connector>  
24.         <se-conector nome = "descanso"/>  
    </pontos finais>
```

Configura o mecanismo de autenticação SASL para o ponto final do Hot Rod. SCRAM-SHA-512 é o mecanismo SASL padrão para Hot Rod. No

entanto, você pode usar o que for apropriado para o seu ambiente, como DIGEST-MD5 ou GSSAPI.

Define o nome que os servidores infinispán apresentam aos clientes. Você especifica este nome na configuração do cliente Hot Rod quando você configura o Keycloak.

25. Crie um modelo de cache.

Adicione o modelo de cache à `infinispan.xml` em cada nó no cluster Infinispan.

```
26. <cache-container ... >
27. <replicado nome de configuração de cache="sessions-cfg"
28.         modo="SYNC">
29. <sem tempo de aquisição="0" />
30. <recuos>
31. <deseto de retorno = estratégia"site2" = "SYNC" />
32. </backups>
33. </replicada configuração de cache>
34. </contêiner de cache>
```

Cria um modelo de cache chamado `sessions-cfg`.

Define um cache que replica sincronizadamente dados em todo o cluster.

Desabilita o tempo limite para aquisição de bloqueio.

Nomeia o site de backup do cluster Infinispan que você está configurando.

35. Inicie o servidor Infinispan1.

```
./server.sh -c infinispan.xml -b PUBLIC_IP_ADDRESS -k PUBLIC_IP_ADDRESS
-Djgroups.mcast_addr=228.6.7.10
```

36. Inicie o servidor Infinispan2.

```
./server.sh -c infinispan.xml -b PUBLIC_IP_ADDRESS -k PUBLIC_IP_ADDRESS
-Djgroups.mcast_addr=228.6.7.11
```

37. Verifique os registros do servidor Infinispan para verificar se os clusters formam visualizações entre sites.

```
38. INFO [org.infinispan.XSITE] (jgroups-5,{server.hostname})
    ISPN0000439: Recebeu nova visualização do site x: [site1]
    INFO [org.infinispan.XSITE] (jgroups-7,{server.hostname}) ISPN0000439:
    Recebeu nova visualização do site x: [site1, site2]
```

Criando caches Infinispan

Crie os caches Infinispan que o Keycloak requer.

Recomendamos que você crie caches em clusters Infinispan em tempo de execução em vez de adicionar caches à `infinispan.xml`. Essa estratégia garante que seus caches sejam sincronizados automaticamente em todo o cluster e armazenados permanentemente.

O procedimento a seguir usa a Interface da Linha de Comando Infinispan (CLI) para criar todos os caches necessários em um único comando de lote.

Pré-requisitos

- Configure seus clusters Infinispan.

procedimento

1. Crie um arquivo em lote que contenha caches, por exemplo:

```
2. > de gato /tmp/caches.batch<<EOF
3. ecoar "criando caches..."
4. criar trabalho de cache --template=sessions-cfg
5. criar sessões de cache --template=sessions-cfg
6. criar cache clienteSessions --template=sessions-cfg
7. criar cache offlineSessions --template=sessions-cfg
8. criar cache offlineClientSessions --template=sessions-cfg
9. criar ação de cacheTokens --template=sessions-cfg
10.      criar login de cacheFailures --template=sessions-cfg
11.      ecoar "verificando caches"
12.      ls caches
      Eof
```

13. Crie os caches com o CLI.

```
$ bin/cli.sh -c https://server1:11222 --trustall -f /tmp/caches.batch
```

Em vez do argumento `--trustall` você pode especificar o truststore com o argumento `-s`.

14. Crie os caches no outro site.

Configuração de lojas de cache remoto no Keycloak

Depois de configurar clusters Infinispan remotos, você configura o subsistema Infinispan no Keycloak para externalizar os dados para esses clusters através de lojas remotas.

Pré-requisitos

- Configure clusters Infinispan remotos para configuração entre sites.
- Crie um truststore que contenha o certificado SSL com a identidade Do Servidor Infinispan.

procedimento

1. Adicione a loja de confiança à implantação do Keycloak.

2. Crie uma ligação de soquete que aponta para o cluster Infinispan.

```
3. < nome de ligação de soquete de saída = > de "cache remoto"
4. host de destino < remote = "${remote.cache.host:server_hostname}"
5. porta = "${remote.cache.port:11222}" />
</out-socket-binding>
```

Nomeia a ligação do soquete como `cache remoto`.

Especifica um ou mais nomes de host para o cluster Infinispan.

Define a porta de `11222` onde o ponto final hot rod ouve.

6. Adicione o módulo `org.keycloak.keycloak-model-infinispan` ao recipiente de `cache keycloak` no subsistema Infinispan.

```
7. <subssistema xmlns="urna:jboss:domínio:infinispan:12.0">
8. <cache-container nome = "keycloak"
    módulos = "org.keycloak.keycloak-model-infinispan" />
```

9. Atualize o cache de trabalho no subsistema Infinispan para que ele tenha a seguinte configuração:

```
10. < nome de cache replicado = "trabalho">
11. < remote-store cache = "trabalho"
12.     servidores remotos = "cache remoto"
13.     passivação = "falso"
14.     buscar estado = "falso"
15.     purga = "falso"
16.     pré-carga = "falso"
17.     compartilhado = "verdadeiro">
18. < nome da propriedade = "rawValues"> verdadeiro </propriedade>
19. < nome da propriedade
    = "marshaller"> org.keycloak.cluster.infinispan.KeycloakHotRodMarshallerFactory </property>
20. < o nome da
    propriedade = "infinispan.client.hotrod.auth_username"> myuser </property>
21. < o nome da
    propriedade = "infinispan.client.hotrod.auth_password"> qwer1234!
    </propriedade>
22. < nome da
    < propriedade = "infinispan.client.hotrod.auth_realm"> padrão </propriedade>
23. < nome da
    propriedade = "infinispan.client.hotrod.auth_server_name"> infinispan </propriedade>
24. < nome da
    propriedade = "infinispan.client.hotrod.sasl_mechanism"> SCRAM-SHA-512 </propriedade>
25. < nome da
    propriedade = "infinispan.client.hotrod.trust_store_file_name"> /path/to/truststore.jks </property>
```

```

26.      < nomeda
propriedade="infinispan.client.hotrod.trust_store_type">JKS</propriedade
>
27.      < nomeda >infinispan.client.hotrod.trust_store_password<
=senha>/propriedade<>
28.      </loja remota>
</cache replicado>

```

Nomeia o cache na configuração Infinispan.

Nomeia o cache correspondente no cluster infinispan remoto.

Especifica a ligação do soquete de cache remoto.

A configuração de cache anterior inclui configurações recomendadas para caches Infinispan. As propriedades de configuração do cliente Hot Rod especificam as credenciais de usuário Infinispan e os detalhes da loja de chaves SSL e do Truststore.

Consulte a [documentação infinispan](#) para descrições de cada propriedade.

29. Adicione caches distribuídos ao subsistema Infinispan para cada um dos seguintes caches:

- Sessões
- sessões de clientes
- offlineSessions
- offlineClientSessions
- açãoTokens
- loginFailures

Por exemplo, adicione uma sessões de cache nomeadas com a seguinte configuração:

```

<distribuido nome de cache="sessões"
    proprietários="1">
    cache <remote-store ="sessões"
        servidores remotos="cache remoto"
        passivação="falso"
        buscar estado="falso"
        purga="falso"
        pré-carga="falso"
        compartilhado="verdadeiro">
< nomeda propriedade="rawValues">verdadeiro</propriedade>
< nome da propriedade
="marshaller">org.keycloak.cluster.infinispan.KeycloakHotRodM
arshallerFactory</property>

```

```

<o nomeda
propriedade="infinispan.client.hotrod.auth_username">myuser<
/property>
<o nomeda
propriedade="infinispan.client.hotrod.auth_password">qwer123
4! </propriedade>
< nomeda
<property="infinispan.client.hotrod.auth_realm">padrão</propriedade>
< nomeda
propriedade="infinispan.client.hotrod.auth_server_name">infinis
pan/propriedade>
< nomeda
propriedade="infinispan.client.hotrod.sasl_mechanism">SCRAM-
SHA-512</propriedade>
< nomeda
propriedade="infinispan.client.hotrod.trust_store_file_name">/p
ath/to/truststore.jks</property>
< nomeda
propriedade="infinispan.client.hotrod.trust_store_type">JKS</pr
opriedade>
< nomeda >infinispan.client.hotrod.trust_store_password<
=senha>/propriedade<>
</loja remota>
</cache distribuído>

```

Nomeia o cache na configuração Infinispan.

Configura uma réplica de cada entrada de cache através do cluster Infinispan.

Nomeia o cache correspondente no cluster infinispan remoto.

Especifica a ligação do soquete de cache remoto.

30. Copie o `NODE11` para outros 3 diretórios referidos posteriormente como `NODE12`, `NODE21` e `NODE22`.

31. Iniciar o `NODE11` :

```

32 .      cd NODE11/bin
33 .      ./standalone.sh -c autônomo-ha.xml -Djboss.node.name=node11 -
      Djboss.site.name=site1 \
34 .      -Djboss.default.multicast.address=234.56.78.1 -
      Dremote.cache.host=server1 \
      -Djava.net.preferIPv4Stack=true -b PUBLIC_IP_ADDRESS

```

Se você notar as seguintes mensagens de aviso em logs, você pode ignorá-las com segurança:

```
WARN [org.infinispan.CONFIG] (thread de serviço MSC 1-5) ISPN000292:
Atributo não reconhecido 'infinispan.client.hotrod.auth_password'. Por
favor, verifique sua configuração. Ignorando!
WARN [org.infinispan.CONFIG] (thread de serviço MSC 1-5) ISPN000292:
Atributo não reconhecido 'infinispan.client.hotrod.auth_username'. Por
favor, verifique sua configuração. Ignorando!
```

35. Iniciar o NODE12 :

```
36.      cd NODE12/bin
37.      ./standalone.sh -c autônomo-ha.xml -Djboss.node.name=node12 -
Djboss.site.name=site1 \
38.      -Djboss.default.multicast.address=234.56.78.1 -
Dremote.cache.host=server1 \
      -Djava.net.preferIPv4Stack=true -b PUBLIC_IP_ADDRESS
```

Os nós do cluster devem ser conectados. Algo assim deve estar no registro tanto do NODE11 quanto DO NODE12:

```
Recebeu nova visualização de cluster para keycloak do canal: [node11|1]
(2) [node11, node12]
```

O nome do canal no registro pode ser diferente.

39. Iniciar o NODE21:

```
40.      cd NODE21/bin
41.      ./standalone.sh -c standalone-ha.xml -Djboss.node.name=node21 -
Djboss.site.name=site2 \
42.      -Djboss.default.multicast.address=234.56.78.2 -
Dremote.cache.host=server2 \
      -Djava.net.preferIPv4Stack=true -b PUBLIC_IP_ADDRESS
```

Ele não deve ser conectado ao cluster com NODE11 e NODE12, mas a um separado:

```
Recebeu nova visualização de cluster para keycloak do canal: [node21|0]
(1) [node21]
```

43. Iniciar o NODE22 :

```
44.      cd NODE22/bin
45.      ./standalone.sh -c standalone-ha.xml -Djboss.node.name=node22 -
Djboss.site.name=site2 \
46.      -Djboss.default.multicast.address=234.56.78.2 -
Dremote.cache.host=server2 \
      -Djava.net.preferIPv4Stack=true -b PUBLIC_IP_ADDRESS
```

Deve estar em cluster com NODE21 :

```
Recebeu nova visualização de cluster para keycloak do canal: [node21|1]
(2) [node21, node22]
```

O nome do canal no registro pode ser diferente.

47. teste:

- Vá para <http://node11:8080/auth/> e crie o usuário administrativo inicial.
- Vá para <http://node11:8080/auth/admin> e faça login como administrador para console administrativo.
- Abra um segundo navegador e vá para qualquer um dos <http://node12:8080/auth/admin> ou <http://node21:8080/auth/admin> ou <http://node22:8080/auth/admin>. Após o login, você deve ser capaz de ver as mesmas sessões na guia Sessões de determinado usuário, cliente ou reino em todos os 4 servidores.
- Depois de fazer uma alteração no Keycloak Admin Console, como modificar um usuário ou um relam, essa alteração deve ser imediatamente visível em qualquer um dos quatro nós. Os caches devem ser devidamente invalidados em todos os lugares.
- Verifique o server.logs, se necessário. Após o login ou logout, a mensagem como esta deve estar em todos os nós NODEXY/autônomo/log/servidor.log:

```
2017-08-25 17:35:17.737 DEBUG
[org.keycloak.models.sessions.infinispan.remotestore.RemoteCacheSessionListener] (Client-Listener-sessions-30012a77422542f5) Recebeu evento da loja remota.
Evento 'CLIENT_CACHE_ENTRY_REMOVED', chave '193489e7-e2bc-4069-afe8-f1dfa73084ea', pule 'falso'
```

Configuração cross DC com Infinispan 9.4.19

Este exemplo para Infinispan 9.4.19 envolve dois data centers, `site1` e `site2`. Cada data center consiste em 1 servidor Infinispan e 2 servidores Keycloak. Vamos acabar com 2 servidores Infinispan e 4 servidores Keycloak no total.

- O `Site1` é composto por servidores Infinispan, `server1` e 2 servidores Keycloak, `node11` e `node12`.
- O `Site2` é composto por servidores Infinispan, `server2` e 2 servidores Keycloak, `node21` e `node22`.
- Os servidores Infinispan `Server1` e `server2` estão conectados entre si através do protocolo RELAY2 e caches Infinispan baseados em `backup` de forma semelhante à descrita na [documentação Infinispan](#).
- Os servidores Keycloak `node11` e `node12` formam um cluster entre si, mas não se comunicam diretamente com nenhum servidor no `site2`. Eles se

comunicam com o servidor Infinispan1 usando o protocolo Hot Rod (cache remoto). Consulte [os detalhes](#) da Comunicação para obter os detalhes.

- Os mesmos detalhes se aplicam ao nó21 e ao nó22. Eles se agrupam entre si e se comunicam apenas com o servidor server2 usando o protocolo Hot Rod.

Nossa configuração de exemplo pressupõe que todos os 4 servidores Keycloak conversem com o mesmo banco de dados. Na produção, recomenda-se usar bancos de dados sincronizados separados em data centers, conforme descrito no [Banco de Dados](#).

Configuração do servidor Infinispan

Siga estas etapas para configurar o servidor Infinispan:

1. Baixe o servidor Infinispan 9.4.19 e descompacte para um diretório escolhido. Este local será referido em etapas posteriores como `SERVER1_HOME`.
2. Alterar essas coisas no `SERVER1_HOME/servidor/conf/infinispan-xsite.xml` na configuração do subsistema JGroups:

- a. Adicione o canal `xsite`, que usará a pilha `tcp`, sob o elemento `canais`:

```
b. <canal padrão ="cluster">
c.   nome <canal ="cluster"/>
d. <canal nome = pilha"xsite" ="tcp"/>
   </canais>
```

- e. Adicione um elemento de relé ao final da pilha `udp`. Vamos configurá-lo de uma forma que nosso site é `site1` e o outro site, onde faremos backup, é o `site2`:

```
f. <dack nome="udp">
g.   ...
h.   <relay site ="site1">
i.   <remote-site nome = canal"site2" ="xsite"/>
j.   < nomeda propriedade="relay_multicasts">falso</propriedade>
k.   </revezamento>
     </pilha>
```

- l. Configure a pilha `tcp` para usar o protocolo `TCPPING` em vez de `MPING`. Remova o elemento `MPING` e substitua-o pelo `TCPPING`. O elemento `initial_hosts` aponta para o servidor `hosts1` e servidor2:

```
m. <seco=> "tcp"
n. <setode transporte="TCP" ligação de soquete="jgroups-tcp"/>
o.   <protocol tipo ="TCPPING">
p.   <o nomede
     initial_hosts>servidor1[7600],server2[7600]</propriedade>
```



```
q. <o nome da propriedade="ergonomia">falso</propriedade>
r. </protocolo>
s. <protocol="MERGE3"/>
t. ...
   </pilha>
```

Esta é apenas uma configuração de exemplo para que as coisas sejam obrigadas a usar a pilha tcp para o JGroups RELAY2, mas você pode usar a pilha udp padrão, se a rede entre seus data centers for capaz de suportar UDP. O Infinispan e o Keycloak são mutuamente indetectáveis. Da mesma forma, o protocolo de descoberta. E na produção, você provavelmente usará um domínio, os nomes do site também são configuráveis. Os detalhes desta configuração estão na documentação do Keycloak. Consulte a documentação do Infinispan e a documentação do Keycloak.

3. Adicione isso em `SERVER1_HOME/autônomo/configuração/agrupado.xml` em cache-container chamado `cluster`:

```

4. <cache-container nome="clustered" default-cache= estatísticas"padrão"
   ="verdadeiro">
5.     ...
6. <replicado nome de configuração de cache= modo"sessions-cfg" ="SYNC"
   start= loteamento"EAGER" ="falso">
7. <sem tempo de aquisição="0" />
8.     <recuos>
9. <desetode retorno="site2" falha-política= estratégia"FAIL" ="SYNC"
   ativado="verdadeiro">
10.     <take-offline min-wait="60000" pós-falhas="3" />
11.         </backup>
12.         </backups>
13.     </replicada configuração de cache>
14.
15.     <replicado nome de cache= configuração"work" ="sessions-cfg"/>
16.     <replicado nome de cache= configuração"sessions" ="sessions-cfg"/>
17.     <replicado nome de cache= configuração"clientSessions" ="sessions-
   cfg"/>
18.     <replicado nome de cache= configuração "offsessions offline"
   ="sessions-cfg"/>
19.     <replicado nome de cache= configuração "offlineClientSessions"
   ="sessions-cfg"/>
20.     <replicado nome de cache= configuração"actionTokens" ="sessions-
   cfg"/>
21.     <replicado nome de cache= configuração"loginFailures" =
   configuração"sessions-cfg"/>
22. </contêiner de cache>

```

Detalhes sobre as opções de configuração dentro da configuração de cache rep [do cache Infinispan](#), que inclui informações sobre o ajuste de algumas dessas op

Ao contrário da versão anterior, o servidor Infinispan replicado-cache-configura transação. Consulte [Solução de problemas](#) para obter mais detalhes.

23. Algumas versões do servidor Infinispan requerem autorização antes de acessar caches protegidos pela rede.

Você não deve ver nenhum problema se você usar o servidor Infinispan 9.4.19 ignorada. Problemas relacionados à autorização podem existir apenas para alg

24. Keycloak requer atualizações para `__script_cache` cache contendo scripts. Se você tiver erros acessando este cache, você precisará configurar a autorização em `cluster.xml` configuração conforme descrito abaixo:

- a. Na <seção > gestão, adicione um reino de segurança:

```
b. <gestão>
c.   <ssegurança>
d.     ...
e.     nome <security-realm="AllowScriptManager">
f.       <autenticação>
g.         <susuários>
h.         <o nome de usuáriodo usuário=> "__script_manager"
i.         <password>não tãosecreto-senha</senha>
j.         </usuário>
k.       </1>
l.     </autenticação>
m.   </segurança>
      </segurança>
```

- n. No subsistema do núcleo do servidor, adicione <segurança>como abaixo:

```
o. <subssistema xmlns="urna:infinispan:server:core:8.4">
p. <cache-container nome="clustered" default-cache=
  estatísticas"padrão"="verdadeiro">
q.   <segurança>
r.     <autorização>
s.       <-digitalista/>
t. <le nome=permissões "__script_manager"="ALL"/>
u.   </autorização>
v. </segurança>
      ...
```

- w. No subsistema de ponto final, adicione a configuração de autenticação ao conector Hot Rod:

```
x. <subssistema xmlns="urna:infinispan:servidor:endpoint:8.1">
y. <otrod-conector cache-container="clustered" soquete-
  vinculação="hotrod">
z.   ...
```

```

aa.      <authentication security-
        realm="AllowScriptManager">
bb.      <sasl mecanismos= qop "DIGEST-MD5" =nome do
        servidor"auth" ="keycloak-jdg-server">
cc.      <política>
dd.      <no-anônimo valor ="falso" />
ee.      </política>
ff.      </sasl>
        </autenticação>

```

25. Copie o servidor para o segundo local, que será referido posteriormente como SERVER2_HOME.

26. No SERVER2_HOME/autônomo/configuração/clusterado.xml trocar o site1 com o site2 e vice-versa, tanto na configuração do relé no subsistema JGroups quanto na configuração de backups no subsistema de cache. Por exemplo:

a. O elemento de relé deve ser assim:

```

b. <relay site ="site2">
c. <remote-site nome = canal "site1" ="xsite"/>
d. < nomeda propriedade="relay_multicasts">falso</propriedade>
   </revezamento>

```

e. O elemento backups como este:

```

f. <recuos>
g. <sitede retorno="site1" ....
   ...

```

O *PUBLIC_IP_ADDRESS* abaixo refere-se ao endereço IP ou nome do vincular. Observe que cada servidor Infinispan e o servidor Keycloak configuração de exemplo com todos os servidores em execução no n Djboss.bind.address.management=*PUBLIC_IP_ADDRESS*, pois cada ser diferente. Mas essa opção geralmente deve ser omitida em ambient remoto ao seu servidor. Para obter mais informações, consulte a *Do*

27. Iniciar servidor de servidor1:

```

28.      cd SERVER1_HOME/bin
29.      ./standalone.sh -c agrupado.xml -Djava.net.preferIPv4Stack=true \
30.      -Djboss.default.multicast.address=234.56.78.99 \
        -Djboss.node.name=server1 -b PUBLIC_IP_ADDRESS

```

31. Iniciar servidor de servidor2. Há um endereço multicast diferente, de modo que os servidores server1 e server2 não estão diretamente agrupados entre si; pelo contrário, eles estão apenas conectados através do protocolo RELAY2, e a pilha TCP JGroups é usada para comunicação entre eles. O comando de inicialção é assim:

```

32.      cd SERVER2_HOME/bin

```

```
33. ./standalone.sh -c agrupado.xml -Djava.net.preferIPv4Stack=true \  
34. -Djboss.default.multicast.address=234.56.78.100 \  
-Djboss.node.name=server2 -b PUBLIC_IP_ADDRESS
```

35. Para verificar se o canal funciona neste momento, você pode precisar usar o JConsole e conectar-se ao servidor SERVER1 em execução ou ao servidor SERVER2. Quando você usa os jgroups MBean:type=protocol,cluster="cluster",protocol=RELAY2 e operation printRoutes, você deve ver a saída assim:

```
36. site1 --> _server1:site1  
site2 --> _server2:site2
```

Quando você usa os jgroups

MBean:type=protocol,cluster="cluster",protocol=GMS, você deve ver que o membro do atributo contém apenas um único membro:

a. No SERVER1 deve ser assim:

```
(1) servidor1
```

b. E no SERVER2 assim:

```
(1) servidor2
```

Na produção, você pode ter mais servidores Infinispan em cada data center. Infinispan no mesmo data center estejam usando o mesmo endereço jboss.default.multicast.address durante a inicialização). Em seguida, membros do cluster atual.

Configuração de servidores Keycloak

1. Descompacte a distribuição do servidor Keycloak para um local escolhido. Ele será referido mais tarde como NODE11.
2. Configure um banco de dados compartilhado para a fonte de dados KeycloakDS. Recomenda-se o uso do MySQL ou Dom.s. Consulte [o Banco de Dados](#) para obter mais detalhes.

Na produção, você provavelmente precisará ter um servidor de banco de dados separado em cada data center e ambos os servidores de banco de dados devem ser síncronamente replicados um para o outro. Na configuração de exemplo, apenas usamos um único banco de dados e conectamos todos os 4 servidores Keycloak a ele.

3. Editar NODE11/autônomo/configuração/autônomo-ha.xml:

a. Adicione o site de atributos ao protocolo JGroups UDP:

```
b. <dack nome="udp">  
  <seque de transporte="UDP" soquete -vinculação= site"jgroups-  
  udp"="$jboss.site.name"/>
```

c. Adicione a loja remota em cache de trabalho:

```
d. <nome de cache replicado="trabalho">
e. <remote-store cache="work" remote-servers="remote-cache"
  passivation="false" fetch-state="falso" purga="falso" pré-
  carga="falso" compartilhado="verdadeiro">
f. <nome da propriedade="rawValues">verdadeiro</propriedade>
g. <nome da propriedade
  ="marshaller">org.keycloak.cluster.infinispan.KeycloakHotRodM
  arshallerFactory</property>
h. <nome da propriedade="protocolVersion">2.9</propriedade>
i. </loja remota>
   </cache replicado>
```

j. Adicione a loja remota como esta em cache de sessões:

```
k. <distribuido nome de cache="sessões" proprietários ="1">
l. <remote-store cache="sessions" servidores remotos=
  passivation"cache remoto" ="falso" buscar-estado="falso"
  expurgo="falso" pré-carga="falso"
  compartilhado="verdadeiro">
m. <nome da propriedade="rawValues">verdadeiro</propriedade>
n. <nome da propriedade
  ="marshaller">org.keycloak.cluster.infinispan.KeycloakHotRodM
  arshallerFactory</property>
o. <nome da propriedade="protocolVersion">2.9</propriedade>
p. </loja remota>
   </cache distribuído>
```

q. Faça o mesmo para offlineSessions, clientSessions, offlineClientSessions, loginFailure e actionTokens caches (a única diferença do cache de sessões é que o valor da propriedade de cache é diferente):

```
r. <distribuido nome de cache="offlineSessions" proprietários ="1">
s. <remote-store cache="off-offlineSessions" servidores remotos=
  passivation"cache remoto" ="falso" buscar-estado="falso"
  purga="falso" pré-carga="falso" compartilhado="verdadeiro">
t. <nome da propriedade="rawValues">verdadeiro</propriedade>
u. <nome da propriedade
  ="marshaller">org.keycloak.cluster.infinispan.KeycloakHotRodM
  arshallerFactory</property>
v. <nome da propriedade="protocolVersion">2.9</propriedade>
w. </loja remota>
x. </cache distribuído>
y.
z. <distribuido nome de cache="clientSessions" ="1">
aa. <remote-store cache="clientSessions" servidores
  remotos=passivation "cache remoto" ="falso" buscar-
```

```

estado="falso"  purga="falso"  pré-carga="falso"
compartilhado="verdadeiro">
bb.    < nomeda
propriedade="rawValues">verdadeiro</propriedade>
cc.    < nome dapropriedade
="marshaller">org.keycloak.cluster.infinispan.KeycloakHotRodM
arshallerFactory</property>
dd.    < nomeda
propriedade="protocolVersion">2.9</propriedade>
ee.    </loja remota>
ff.    </cache distribuído>
gg.
hh.    <distribuido nome de cache="offlineClientSessions"
proprietários ="1">
ii.    <remote-store cache="offlineClientSessions" servidores
remotos=passivation "cache remoto" ="falso" busca-
estado="falso" expurgo="falso" pré-carga ="falso"
compartilhado="verdadeiro">
jj.    < nomeda
propriedade="rawValues">verdadeiro</propriedade>
kk.    < nome dapropriedade
="marshaller">org.keycloak.cluster.infinispan.KeycloakHotRodM
arshallerFactory</property>
ll.    < nomeda
propriedade="protocolVersion">2.9</propriedade>
mm.    </loja remota>
nn.    </cache distribuído>
oo.
pp.    <distribuido nome de cache=proprietários de
"loginFailures" ="1">
qq.    <remote-store cache ="loginFailures" servidores remotos=
passivation"cache remoto" ="falso" buscar-estado="falso"
purga="falso" pré-carga ="falso" compartilhado="verdadeiro">
rr.    < nomeda
propriedade="rawValues">verdadeiro</propriedade>
ss.    < nome dapropriedade
="marshaller">org.keycloak.cluster.infinispan.KeycloakHotRodM
arshallerFactory</property>
tt.    < nomeda
propriedade="protocolVersion">2.9</propriedade>
uu.    </loja remota>
vv.    </cache distribuído>
ww.
xx.    <distribuido nome de cache=proprietários de
"actionTokens" ="2">
yy.    <object-memory size ="-1"/>
zz.    <expiração max-ociosa= intervalo"-1" ="300000"/>

```

```

aaa.    <remote-store cache ="actionTokens" servidores
        remotos= passivation"remote-cache" ="false" fetch-
        state="falso" purga="falsa" pré-carga="verdadeiro"
        compartilhado="verdadeiro">
bbb.    < nomeda
        propriedade="rawValues">verdadeiro</propriedade>
ccc.    < nomeda propriedade
        ="marshaller">org.keycloak.cluster.infinispan.KeycloakHotRodM
        arshallerFactory</property>
ddd.    < nomeda
        propriedade="protocolVersion">2.9</propriedade>
eee.    </loja remota>
        </cache distribuído>

```

fff. Adicione a ligação do soquete de saída para a loja remota na configuração do elemento de grupo de ligação de soquete:

```

ggg.    < nome de ligação de soquete de saída = > de "cache remoto"
hhh.    <remote-destino host =porta
        "${remote.cache.host:localhost}" =porta
        "${remote.cache.port:11222}"/>
        </out-socket-binding>

```

iii. A configuração de autenticações distribuídas de cacheSessões e outros caches permanecem inalteradas.

jjj. Recomenda-se adicionar a propriedade `remotaStoreSecurityEnabled` com o valor de `falso` (ou eventualmente `verdadeiro` se você habilitou a segurança dos servidores Infinispan conforme descrito acima) às `conexõesInfinispan` SPI no subsistema `keycloak-server`:

```

kkk.    nome <spi ="conexõesInfinispan">
lll.    ...
mmm.    <provider ... >
nnn.    <propriedades>
ooo.    ...
ppp.    < nomeda propriedade=
        valor"remoteStoreSecurityEnabled" =valor "falso"/>
qqq.    </propriedades>
        ...

```

rrr. Ative opcionalmente o registro DEBUG sob o subsistema de registro:

```

sss.    <logger categoria ="org.keycloak.cluster.infinispan">
ttt.    nome <nível ="DEBUG"/>
uuu.    </madeireiro>
vvv.    <logger categoria ="org.keycloak.connections.infinispan">
www.    nome <nível ="DEBUG"/>
xxx.    </madeireiro>

```

```

yyy.    <logger categoria
        ="org.keycloak.models.cache.infinispan">
zzz.    nome <nível ="DEBUG"/>
aaaa.   </madeireiro>
bbbb.   <logger categoria
        ="org.keycloak.models.sessions.infinispan">
cccc.   nome <nível ="DEBUG"/>
        </madeireiro>

```

4. Copie o NODE11 para outros 3 diretórios referidos posteriormente como NODE12, NODE21 e NODE22.

5. Iniciar o NODE11 :

```

6. cd NODE11/bin
7. ./standalone.sh -c autônomo-ha.xml -Djboss.node.name=node11 -
   Djboss.site.name=site1 \
8. -Djboss.default.multicast.address=234.56.78.1 -
   Dremote.cache.host=server1 \
   -Djava.net.preferIPv4Stack=true -b PUBLIC_IP_ADDRESS

```

9. Iniciar o NODE12 :

```

10. cd NODE12/bin
11. ./standalone.sh -c autônomo-ha.xml -Djboss.node.name=node12 -
   Djboss.site.name=site1 \
12. -Djboss.default.multicast.address=234.56.78.1 -
   Dremote.cache.host=server1 \
   -Djava.net.preferIPv4Stack=true -b PUBLIC_IP_ADDRESS

```

Os nós do cluster devem ser conectados. Algo assim deve estar no registro tanto do NODE11 quanto DO NODE12:

```

Recebeu nova visualização de cluster para keycloak do canal: [node11|1]
(2) [node11, node12]

```

O nome do canal no registro pode ser diferente.

13. Iniciar o NODE21:

```

14. cd NODE21/bin
15. ./standalone.sh -c standalone-ha.xml -Djboss.node.name=node21 -
   Djboss.site.name=site2 \
16. -Djboss.default.multicast.address=234.56.78.2 -
   Dremote.cache.host=server2 \
   -Djava.net.preferIPv4Stack=true -b PUBLIC_IP_ADDRESS

```

Ele não deve ser conectado ao cluster com NODE11 e NODE12, mas para separar um:

```

Recebeu nova visualização de cluster para keycloak do canal: [node21|0]
(1) [node21]

```


17. Iniciar o NODE22 :

```
18. cd NODE22/bin
19. ./standalone.sh -c standalone-ha.xml -Djboss.node.name=node22 -
    Djboss.site.name=site2 \
20. -Djboss.default.multicast.address=234.56.78.2 -
    Dremote.cache.host=server2 \
    -Djava.net.preferIPv4Stack=true -b PUBLIC_IP_ADDRESS
```

Deve estar em cluster com NODE21 :

Recebeu nova visualização de cluster para keycloak do canal: [node21|1]
(2) [node21, node22]

O nome do canal no registro pode ser diferente.

21. teste:

- Vá para <http://node11:8080/auth/> e crie o usuário administrativo inicial.
- Vá para <http://node11:8080/auth/admin> e faça login como administrador para console administrativo.
- Abra um segundo navegador e vá para qualquer um dos <http://node12:8080/auth/admin> ou <http://node21:8080/auth/admin> ou <http://node22:8080/auth/admin>. Após o login, você deve ser capaz de ver as mesmas sessões na guia Sessões de determinado usuário, cliente ou reino em todos os 4 servidores.
- Depois de fazer qualquer alteração no console administrativo Keycloak (por exemplo, atualizar algum usuário ou algum reino), a atualização deve ser imediatamente visível em qualquer um dos 4 nós, pois os caches devem ser devidamente invalidados em todos os lugares.
- Verifique o server.logs, se necessário. Após o login ou logout, a mensagem como esta deve estar em todos os nós NODEXY/autônomo/log/servidor.log:

```
f . 2017-08-25 17:35:17.737 DEBUG
[org.keycloak.models.sessions.infinispan.remotestore.RemoteCa
cheSessionListener] (Client-Listener-sessions-
30012a77422542f5) Recebeu evento da loja remota.
Evento 'CLIENT_CACHE_ENTRY_REMOVED', chave '193489e7-
e2bc-4069-afe8-f1dfa73084ea', pule 'falso'
```

Administração da implantação do Cross DC

Esta seção contém algumas dicas e opções relacionadas à replicação do datacenter cruzado.

- Quando você executa o servidor Keycloak dentro de um data center, é necessário que o banco de dados referenciado na fonte de dados KeycloakDS já esteja sendo executado e disponível nesse data center. Também é necessário que o servidor Infinispan referenciado pela [ligação de tomada de saída](#), que é referenciada a partir do elemento de armazenamento remoto de cache Infinispan, já esteja em execução. Caso contrário, o servidor Keycloak não será inicial.
- Cada data center pode ter mais nós de banco de dados se você quiser suportar failover do banco de dados e melhor confiabilidade. Consulte a documentação do seu banco de dados e do driver JDBC para obter os detalhes de como configurar isso no lado do banco de dados e como a fonte de dados KeycloakDS no lado Keycloak precisa ser configurada.
- Cada datacenter pode ter mais servidores Infinispan em execução no cluster. Isso é útil se você quiser alguma falha e melhor tolerância à falha. O protocolo Hot Rod usado para comunicação entre servidores Infinispan e servidores Keycloak tem um recurso que os servidores Infinispan enviarão automaticamente uma nova topologia para os servidores Keycloak sobre a mudança no cluster Infinispan, para que a loja remota no lado Keycloak saiba a quais servidores Infinispan ele pode conectar. Leia a documentação Infinispan e WildFly para obter mais detalhes.
- É altamente recomendável que um servidor Infinispan mestre esteja sendo executado em todos os sites antes que os servidores Keycloak em **qualquer** site sejam iniciados. Como em nosso exemplo, começamos tanto o `server1` quanto o `server2` primeiro, antes de todos os servidores Keycloak. Se você ainda precisar executar o servidor Keycloak e o site de backup estiver offline, é recomendável trocar manualmente o site de backup offline nos servidores Infinispan em seu site, conforme descrito em [Trazar sites offline e on-line](#). Se você não alternar manualmente o site indisponível offline, a primeira inicial pode falhar ou pode ser algumas exceções durante a inicialização até que o site de backup seja retirado offline automaticamente devido à contagem configurada de operações com falha.

Trazendo sites offline e online

Por exemplo, assuma este cenário:

1. O `site2` está totalmente offline do [ponto de vista do site1](#). Isso significa que todos os servidores Infinispan no `site2` estão desligados **ou** a rede entre o `site1` e o `site2` está quebrada.
2. Você executa servidores Keycloak e servidor Infinispan1 no `site1`
3. Alguém faz login em um servidor Keycloak no `site1`.
4. O servidor Keycloak do `site1` tentará gravar a sessão para o cache remoto no servidor `server1`, que deve fazer backup de dados para o `server2` no `site2`. Consulte [os detalhes da Comunicação](#) para obter mais informações.

5. O servidor `server2` está offline ou inalcançável do `servidor1`. Assim, o backup do `servidor1` para o `servidor2` falhará.
6. A exceção é lançada no log `server1` e a falha será propagada do servidor `server1` para servidores Keycloak também porque a política de falha de backup padrão está configurada. Consulte [a política de falha do Backup](#) para obter detalhes sobre as políticas de backup.
7. O erro acontecerá também no lado keycloak e o usuário pode não ser capaz de terminar seu login.

De acordo com o seu ambiente, pode ser mais ou menos provável que a rede entre os sites esteja indisponível ou temporariamente quebrada (cérebro dividido). Caso isso aconteça, é bom que os servidores Infinispan no `site1` estejam cientes do fato de que os servidores Infinispan no `site2` não estão disponíveis, então eles vão parar de tentar alcançar os servidores no `site do servidor2` e as falhas de backup não acontecerão. Isso se chama `Take site offline`.

Tire o site offline

Existem 2 maneiras de deixar o site offline.

Manualmente por administração - O administrador pode usar o `jconsole` ou outra ferramenta e executar algumas operações JMX para tirar manualmente o site em particular offline. Isso é útil especialmente se a paralisação for planejada. Com `jconsole` ou CLI, você pode se conectar ao `servidor1` e tirar o `site2` offline. Mais detalhes sobre isso estão disponíveis na documentação [infinispan](#).

Essas etapas geralmente precisam ser feitas para todos os caches Keycloak mencionados em

Automaticamente - Após alguma quantidade de backups com falha, o `site2` geralmente será retirado offline automaticamente. Isso é feito devido à configuração do elemento `take-offline` dentro da configuração de cache, conforme configurado na [configuração do servidor Infinispan](#).

```
<take-offline min-wait="60000" pós-falhas="3" />
```

Este exemplo mostra que o site será retirado offline automaticamente para o cache único específico se houver pelo menos 3 backups com falha subsequentes e não houver backup bem-sucedido dentro de 60 segundos.

Tirar automaticamente um site offline é útil especialmente se a rede quebrada entre os sites não for planejada. A desvantagem é que haverá alguns backups falhos até que a paralisação da rede seja detectada, o que também pode significar falhas no lado do aplicativo. Por exemplo, haverá logins com falha para alguns usuários ou grandes intervalos de login. Especialmente se a falha de política com o valor `FAIL` for usada.

O rastreamento de se um site está offline é rastreado separadamente para cada cache.

Faça o site on-line

Uma vez que sua rede esteja de volta e o site1 e o site2 possam conversar entre si, você pode precisar colocar o site online. Isso precisa ser feito manualmente através de JMX ou CLI de forma semelhante à tomada de um site offline. Novamente, você pode precisar verificar todos os caches e trazê-los on-line.

Uma vez que os sites são colocados on-line, geralmente é bom:

- Faça a [transferência do Estado](#).
- Limpar manualmente [caches](#).

Transferência estatal

A transferência do estado é uma etapa manual necessária. O servidor Infinispan não faz isso automaticamente, por exemplo, durante o cérebro dividido, é apenas o administrador que pode decidir qual site tem preferência e, portanto, se a transferência do estado precisa ser feita bidirecionalmente entre ambos os sites ou apenas unidirecionalmente, como apenas no site1 para o site2, mas não do site2 para o site1.

Uma transferência bidirecional do Estado garantirá que as entidades criadas **após** o cérebro dividido no local1 sejam transferidas para o site2. Este não é um problema, pois eles ainda não existem no site2. Da mesma forma, as entidades criadas **após** o cérebro dividido no local2 serão transferidas para o site1. Possivelmente partes problemáticas são aquelas entidades que existem **antes do cérebro dividido em ambos os locais e que** foram atualizadas durante o cérebro dividido em ambos os locais. Quando isso acontecer, um dos sites **ganhará** e substituirá as atualizações feitas durante o split-brain pelo segundo site.

Infelizmente, não há nenhuma solução universal para isso. Cérebros divididos e paralisações de rede são apenas estado, o que geralmente é impossível de ser tratado 100% corretamente com dados 100% consistentes entre os sites. No caso de Keycloak, normalmente não é uma questão crítica. Na pior das hipóteses, os usuários precisarão fazer login novamente em seus clientes, ou ter a contagem indevida de loginFailures rastreados para proteção de força bruta. Consulte a documentação infinispan/JGroups para obter mais dicas sobre como lidar com cérebro dividido.

A transferência estatal também pode ser feita no lado do servidor Infinispan através do JMX. O nome da operação é pushState. Há poucas outras operações para monitorar o status, cancelar o estado de pressão, e assim por diante. Mais informações sobre transferência estatal estão disponíveis nos [documentos da Infinispan](#).

Limpar caches

Depois de dividir o cérebro é seguro limpar manualmente caches no console administrativo Keycloak. Isso porque pode haver alguns dados alterados no banco de dados no site1 e, por causa do evento, que o cache deve ser invalidado não foi transferido durante o cérebro dividido para o site2. Portanto, os nódulos Keycloak no site2 ainda podem ter alguns dados obsoletos em seus caches.

Para limpar os caches, consulte [Limpar caches do servidor](#).

Quando a rede está de volta, é suficiente para limpar o cache apenas em um nó Keycloak em qualquer site aleatório. O evento de invalidação de cache será enviado para todos os outros nós keycloak em todos os sites. No entanto, ele precisa ser feito para todos os caches (reinos, usuários, chaves). Consulte [Limpar caches do servidor](#) para obter mais informações.

Afinando a configuração de cache Infinispan

Esta seção contém dicas e opções para configurar seu cache JDG.

Política de falha de backup

Por padrão, a configuração de falha de backup na configuração de cache Infinispan no arquivo `infinispan clustered.xml` está configurada como `FAIL`. Você pode alterá-lo para `WARN` ou `IGNORE`, como preferir.

A diferença entre `FAIL` e `WARN` é que quando `fail` é usado e o servidor Infinispan tenta fazer backup de dados até o outro site e o backup falha, então a falha será propagada de volta para o chamador (o servidor Keycloak). O backup pode falhar porque o segundo site é temporariamente inalcançável ou há uma transação simultânea que está tentando atualizar a mesma entidade. Neste caso, o servidor Keycloak tentará novamente a operação algumas vezes. No entanto, se a repetição falhar, então o usuário poderá ver o erro após um intervalo mais longo.

Ao usar o `WARN`, os backups com falha não são propagados do servidor Infinispan para o servidor Keycloak. O usuário não verá o erro e o backup com falha será ignorado. Haverá um tempo limite menor, normalmente 10 segundos, pois esse é o tempo limite padrão para backup. Ele pode ser alterado pelo tempo limite de atributo do elemento `backup`. Não haverá novos julgamentos. Haverá apenas uma mensagem DE AVISO no registro do servidor Infinispan.

O problema potencial é que, em alguns casos, pode haver apenas uma pequena paralisação de rede entre os sites, onde a repetição (uso da política `FAIL`) pode ajudar, portanto, com o `WARN` (sem tentar novamente), haverá algumas inconsistências de dados entre os sites. Isso também pode acontecer se houver uma tentativa de atualizar a mesma entidade simultaneamente em ambos os sites.

Quão ruins são essas inconsistências? Normalmente só significa que um usuário precisará re-autenticar.

Ao usar a diretiva `WARN`, pode acontecer que o cache de uso único, que é fornecido pelo `cache actionTokens` e que lida com essa chave específica, é realmente de uso único, mas pode "com sucesso" gravar a mesma tecla duas vezes. Mas, por exemplo, a especificação OAuth2 [menciona](#) que o código deve ser de uso único. Com a política `WARN`, isso pode não ser estritamente garantido e o mesmo código pode ser escrito duas vezes se houver uma tentativa de escrevê-lo simultaneamente em ambos os sites.

Se houver uma maior paralisação da rede ou cérebro dividido, em seguida, com `FAIL` e `WARN`, o outro site será retirado offline após algum tempo e falhas como descrito em [Trazar sites off-line e on-line](#). Com o tempo limite padrão de 1 minuto, geralmente é de 1 a 3 minutos até que todos os caches envolvidos sejam retirados off-line. Depois disso, todas as operações funcionarão bem do ponto de vista do usuário final. Você só precisa restaurar manualmente o site quando ele estiver de volta on-line como mencionado em [Trazar sites offline e on-line](#).

Em resumo, se você espera paralisações frequentes e mais longas entre sites e é aceitável que você tenha algumas inconsistências de dados e um cache de uso único não 100% preciso, mas você nunca quer que os usuários finais vejam os erros e longos intervalos, então mude para `WARN`.

A diferença entre `WARN` e `IGNORE` é que, com os avisos `IGNOR`, não estão escritos no registro `Infinispan`. Veja mais detalhes na documentação da `Infinispan`.

Bloquear tempo limite de aquisição

A configuração padrão é usar a transação no modo `NON_DURABLE_XA` com a aquisição de tempo limite 0. Isso significa que a transação falhará rapidamente se houver outra transação em andamento para a mesma chave.

A razão para mudar isso para 0 em vez de 10 segundos padrão foi para evitar possíveis problemas de impasse. Com o `Keycloak`, pode acontecer que a mesma entidade (tipicamente entidade de sessão ou `loginFailure`) seja atualizada simultaneamente de ambos os sites. Isso pode causar impasse em algumas circunstâncias, o que fará com que a transação seja bloqueada por 10 segundos. Veja [este relatório JIRA](#) para obter detalhes.

Com o tempo limite 0, a transação falhará imediatamente e, em seguida, será novamente julgado a partir de `Keycloak` se a falha de `backup` com o valor `FAIL` for configurada. Enquanto a segunda transação simultânea for concluída, a repetição geralmente será bem sucedida e a entidade terá aplicado atualizações de ambas as transações simultâneas.

Vemos consistência muito boa e resultados para transação simultânea com esta configuração, e é recomendável mantê-la.

O único problema (não funcional) é a exceção no registro do servidor `Infinispan`, que acontece sempre que o bloqueio não está disponível imediatamente.

Backups SYNC ou ASYNC

Uma parte importante do elemento de backup é o atributo de estratégia. Você deve decidir se precisa ser SYNC ou ASYNC. Temos 7 caches que podem estar cientes da replicação do datacenter cruzado, e estes podem ser configurados em 3 modos diferentes em relação ao cross-dc:

1. Backup sync
2. Backup ASYNC
3. Sem reforços

Se o backup SYNC for usado, o backup será síncrona e a operação será considerada concluída no lado do chamador (servidor Keycloak) assim que o backup for processado no segundo site. Isso tem desempenho pior do que o ASYNC, mas por outro lado, você tem certeza de que as leituras subsequentes da entidade específica, como a sessão do usuário, no site2 verão as atualizações do site1. Além disso, é necessário se você quiser consistência de dados. Assim como no ASYNC, o chamador não é notificado se o backup no outro site falhar.

Para alguns caches, é até possível não fazer backup e pular completamente a escrita de dados para o servidor Infinispan. Para configurar isso, não use o elemento de armazenamento remoto para o cache específico no lado Keycloak (KEYCLOAK_HOME/autônomo/configuração/autônomo-ha.xml) e, em seguida, o elemento de cache replicado específico também não é necessário no lado do servidor Infinispan.

Por padrão, todos os 7 caches são configurados com backup SYNC, que é a opção mais segura. Aqui estão algumas coisas a considerar:

- Se você estiver usando o modo ativo/passivo (todos os servidores Keycloak estão no único site1 e o servidor Infinispan no site2 será usado puramente como backup. Consulte **Modos** para obter mais detalhes), então geralmente é bom usar a estratégia ASYNC para todos os caches para salvar o desempenho.
- O cache de trabalho é usado principalmente para enviar algumas mensagens, como eventos de invalidação de cache, para o outro site. Também é usado para garantir que alguns eventos especiais, como sincronizações do UserStorage, aconteçam apenas em um único site. Recomenda-se manter este conjunto para SYNC.
- O cache actionTokens é usado como cache de uso único para rastrear que alguns tokens/tickets foram usados apenas uma vez. Por exemplo, tokens de ação ou códigos OAuth2. É possível definir isso para ASYNC para um desempenho ligeiramente melhorado, mas então não é garantido que o bilhete em particular é realmente de uso único. Por exemplo, se houver solicitação simultânea para o mesmo bilhete em ambos os sites, então é possível que ambas as solicitações sejam bem sucedidas com a estratégia

ASYNCR. Portanto, o que você definir aqui dependerá se você prefere melhor segurança (estratégiaSYNCR) ou melhor desempenho (estratégiaASYNCR).

- O cache loginFailures pode ser usado em qualquer um dos 3 modos. Se não houver backup, significa que a contagem de falhas de login para um usuário será contada separadamente para cada site (Consulte [caches Infinispan](#) para obter detalhes). Isso tem algumas implicações de segurança, porém tem algumas vantagens de desempenho. Também mitiga o possível risco de negação de serviço (DoS). Por exemplo, se um invasor simular 1000 solicitações simultâneas usando o nome de usuário e senha do usuário em ambos os sites, isso significará muitas mensagens sendo passadas entre os sites, o que pode resultar em congestionamento da rede. A estratégia ASYNCR pode ser ainda pior, pois os pedidos de invasor não serão bloqueados esperando o backup para o outro site, resultando em tráfego de rede potencialmente ainda mais congestionado. A contagem de falhas de login também não será precisa com a estratégia ASYNCR.

Para os ambientes com rede mais lenta entre data centers e probabilidade de DoS, recomenda-se não fazer backup do cache loginFailures em tudo.

- Recomenda-se manter as sessões e caches de sessões de clientes em SYNCR. Alterá-los para ASYNCR só é possível se você tiver certeza de que as solicitações do usuário e as solicitações de backchannel (solicitações de aplicativos de clientes para Keycloak conforme descrito no [processamento de solicitação](#)) serão sempre processadas no mesmo site. Isso é verdade, por exemplo, se:
 - Você usa o modo ativo/passivo como descrito [Modos](#).
 - Todos os aplicativos de seus clientes estão usando o [Adaptador JavaScriptKeycloak](#). O adaptador JavaScript envia as solicitações do backchannel dentro do navegador e, portanto, eles participam da sessão pegajosa do navegador e terminarão no mesmo nó de cluster (portanto no mesmo site) que as outras solicitações do navegador deste usuário.
 - Seu balanceador de carga é capaz de atender às solicitações com base no endereço IP do cliente (localização) e os aplicativos do cliente são implantados em ambos os sites.

Por exemplo, você tem 2 sites LON e NYC. Desde que seus aplicativos sejam implantados em sites lon e NYC também, você pode garantir que todas as solicitações de usuários dos usuários de Londres serão redirecionadas para os aplicativos no site lon e também para os servidores Keycloak no site LON. As solicitações de backchannel das implantações de clientes do site LON terminarão em servidores Keycloak no site LON também. Por outro lado, para os usuários americanos, todas as solicitações do Keycloak, solicitações de aplicativos e solicitações de backchannel serão processadas no site de NYC.

- Para sessões offlineS e off-lineClientSessions é semelhante, com a diferença de que você nem precisa fazer backup deles se você nunca planeja usar tokens off-line para qualquer um dos seus aplicativos de cliente.

Geralmente, se você está em dúvida e o desempenho não é um bloqueador para você, é mais seguro manter os caches na estratégia SYNC.

Em relação ao switch para backup SYNC/ASync, certifique-se de editar o atributo de estratégia de backup, por exemplo, assim:

```
<desetode retorno="site2" falha-política= estratégia"FAIL" ="ASync"
habilitado="verdadeiro">
```

Observe o atributo de modo do elemento de configuração de cache.

Solucionando problemas

As seguintes dicas são destinadas a ajudá-lo caso você precise solucionar problemas:

- Recomenda-se passar pelo Cross DC com o [Infinispan 9.4.19](#) e ter este funcionando primeiro, para que você tenha alguma compreensão de como as coisas funcionam. Também é sábio ler todo este documento para ter alguma compreensão das coisas.
- Verifique o status do cluster jconsole (GMS) e o status JGroups (RELAY) da Infinispan conforme descrito na [configuração do servidor Infinispan](#). Se as coisas não parecerem como esperado, então o problema provavelmente está na configuração de servidores Infinispan.
- Para os servidores Keycloak, você deve ver uma mensagem como esta durante a inicialização do servidor:

```
18:09:30.156 INFO
[org.keycloak.connections.infinispan.DefaultInfinispanConnectionProviderFactory] (ServerService Thread Pool -- 54)
Nome do nó: node11, Nome do site: site1
```

Verifique se o nome do site e o nome do nó pareçam como esperado durante a inicialização do servidor Keycloak.

- Verifique se os servidores Keycloak estão em cluster como esperado, incluindo que apenas os servidores Keycloak do mesmo data center estão em cluster entre si. Isso também pode ser verificado no JConsole através da exibição GMS. Consulte [a solução de problemas do cluster](#) para obter mais detalhes.
- Se houver exceções durante a inicialização do servidor Keycloak assim:

```
17:33:58.605 ERRO
[org.infinispan.client.hotrod.impl.operations.RetryOnFailureOperation]
```

(ServerService Thread Pool -- 59) ISPN004007: Exceção encontrada. Rejulgando 10 de 10: org.infinispan.client.hotrod.exceptions.TransportException:: Não foi possível buscar transporte

- ...
- Causado por: org.infinispan.client.hotrod.exceptions.TransportException:: Não foi possível conectar-se ao servidor: 127.0.0.1:12232 em org.infinispan.client.hotrod.impl.transport.tcp.TcpTransport.<init>(TcpTransport.java:82)

geralmente significa que o servidor Keycloak não é capaz de alcançar o servidor Infinispan em seu próprio datacenter. Certifique-se de que o firewall está definido como esperado e o servidor Infinispan é possível conectar.

- Se houver exceções durante a inicialização do servidor Keycloak assim:
- 16:44:18.321 WARN [org.infinispan.client.hotrod.impl.protocol.Codec21] (ServerService Thread Pool -- 57) ISPN004005: Erro recebido do servidor: javax.transaction.RollbackException: ARJUNA016053: Não foi possível cometer transação.
...

em seguida, verifique o registro do servidor Infinispan correspondente do seu site e verifique se não conseguiu fazer backup no outro site. Se o site de backup não estiver disponível, então é recomendável trocá-lo offline, para que o servidor Infinispan não tente fazer backup no site offline, fazendo com que as operações passem com sucesso no lado do servidor Keycloak também. Consulte [a implantação da Administração do Cross DC](#) para obter mais informações.

- Confira as estatísticas da Infinispan, que estão disponíveis através do JMX. Por exemplo, tente fazer login e, em seguida, ver se a nova sessão foi escrita com sucesso para ambos os servidores Infinispan e está disponível no cache de sessões lá. Isso pode ser feito indiretamente verificando a contagem de elementos no cache de sessões para o MBean `jboss.datagrid-infinispan:type=Cache,name="sessions(repl_sync)",manager="clustered",component=Statistics` e número de atributo `SOEntries`. Após o login, deve haver mais uma entrada para `entradas numédias` em ambos os servidores Infinispan em ambos os sites.
- Habilite o registro DEBUG conforme descrito [Configurando servidores Keycloak](#). Por exemplo, se você fizer login e achar que a nova sessão não está disponível no segundo site, é bom verificar os registros do servidor Keycloak e verificar se os ouvintes foram acionados conforme descrito nos [servidores Keycloak de configuração](#). Se você não sabe e quer perguntar na lista de discussão do usuário keycloak, é útil enviar os arquivos de log dos servidores Keycloak em ambos os data centers no e-mail. Adicione os trechos de registro aos e-mails ou coloque os registros em algum lugar e faça referência a eles no e-mail.

- Se você atualizou a entidade, como o usuário, no servidor Keycloak no site1 e não ver essa entidade atualizada no servidor Keycloak no site2, então o problema pode estar na replicação do próprio banco de dados síncrona ou que os caches Keycloak não são devidamente invalidados. Você pode tentar desativar temporariamente os caches Keycloak descritos [aqui](#) para descobrir se o problema estiver no nível de replicação do banco de dados. Além disso, pode ajudar a se conectar manualmente ao banco de dados e verificar se os dados são atualizados como esperado. Isso é específico para cada banco de dados, então você precisará consultar a documentação do seu banco de dados.
- Às vezes, você pode ver as exceções relacionadas a bloqueios como este no registro do servidor Infinispan:

- (HotRodServerHandler-6-35) ISPN000136: Comando de execução de erros Substituircomand,
- teclas de escrita [[B0x033E243034396234.. [39]:
org.infinispan.util.concurrent.TimeoutException: ISPN000299: Incapaz de adquirir bloqueio após
0 milissegundos para chave [B0x033E243034396234.. [39] e solicitador
GlobalTx:server1:4353. Bloqueio é mantido por GlobalTx:server1:4352

Essas exceções não são necessariamente um problema. Eles podem acontecer a qualquer momento quando uma edição simultânea da mesma entidade é acionada em ambos os DCs. Isso é comum em uma implantação. Normalmente, o servidor Keycloak é notificado sobre a operação falhada e irá rejulhá-la, então, do ponto de vista do usuário, geralmente não há nenhum problema.

- Se houver exceções durante a inicialização do servidor Keycloak, assim:
- 16:44:18.321 WARN [org.infinispan.client.hotrod.impl.protocol.Codec21] (ServerService Thread Pool -- 55) ISPN004005: Erro recebido do servidor: java.lang.SecurityException: ISPN000287: Acesso não autorizado: assunto 'Assunto com principal(s): []' carece de 'READ permission'
...

Essas entradas de registro são o resultado de keycloak detectar automaticamente se a autenticação é necessária no Infinispan e significa que a autenticação é necessária. Neste ponto, você notará que ou o servidor começa com sucesso e você pode ignorá-los com segurança ou que o servidor não é iniciado. Se o servidor não for inicializado, certifique-se de que o Infinispan foi configurado corretamente para autenticação conforme descrito na [configuração do servidor Infinispan](#). Para evitar que essa entrada de log seja incluída, você pode forçar a autenticação definindo a propriedade remotaStoreSecurityEnabled como verdadeira na configuração spi=connectionsInfinispan/provider=default:

```
<subistema xmlns="urna:jboss:domínio:keycloak-server:1.1">
...
nome <spi ="conexõesInfinispan">
```

```

...
<provo nome="padrão" ativado="verdadeiro">
  <propriedades>
    ...
    <o nome de propriedade= valor "remoteStoreSecurityEnabled" =valor
    "verdadeiro"/>
  </propriedades>
</provedor>
</spi>

```

- Se você tentar autenticar com o Keycloak para o seu aplicativo, mas a autenticação falhar com um número infinito de redirecionamentos no seu navegador e você ver os erros como este no registro do servidor Keycloak:

```

2017-11-27 14:50:31.587 WARN [org.keycloak.events] (tarefa padrão-17)
type=LOGIN_ERROR, realmId=master, clientId=null, userId=null,
ipAddress=aa.bb.cc.dd, error=expired_code, restart_after_timeout=true

```

provavelmente significa que o balanceador de carga precisa ser configurado para suportar sessões pegajosas. Certifique-se de que o nome da rota fornecido usado durante a inicialização do servidor Keycloak (Property `jboss.node.name`) contém o nome correto usado pelo servidor balanceador de carga para identificar o servidor atual.

- Se o cache de trabalho Infinispan crescer indefinidamente, você pode estar experimentando este problema [Infinispan](#), que é causado por itens de cache que não estão devidamente expirados. Nesse caso, atualize a declaração de cache com uma tag vazia `<expiration />` como esta:

- ```
<replicado nome de cache= configuração "work" = "sessions-cfg">
```
- ```
  <expiração />
```
- ```
</cache replicado>
```

- Se você ver avisos no registro do servidor Infinispan como:

- 18:06:19.687 WARN [org.infinispan.server.hotrod.Decoder2x] (HotRod-ServerWorker-7-12) ISPN006011: Operação 'PUT\_IF\_ABSENT' forçada a
- o valor anterior de retorno deve ser usado em caches transacionais, caso contrário, problemas de inconsistência de dados podem surgir em situações de falha
- 18:06:19.700 WARN [org.infinispan.server.hotrod.Decoder2x] (HotRod-ServerWorker-7-10) ISPN006010: Operação condicional 'REPLACE\_IF\_UNMODIFIED' deve
- ser usado com caches transacionais, caso contrário, problemas de inconsistência de dados podem surgir em situações de falha

você pode simplesmente ignorá-los. Para evitar o aviso, os caches no lado do servidor Infinispan podem ser alterados para caches transacionais, mas isso não é recomendado, pois pode causar alguns outros problemas causados pelo <https://issues.redhat.com/browse/ISPN-9323> de bug. Então, por enquanto, os avisos só precisam ser ignorados.

- Se você ver erros no registro do servidor Infinispan como:
- ```

12:08:32.921 ERRO [org.infinispan.server.hotrod.cacheDecodeContext]
(HotRod-ServerWorker-7-11) ISPN005003: Exceção relatada:
org.infinispan.server.hotrod.InvalidMagicIdceptionException: Erro leitura
de byte mágico ou id de mensagem: 7
• em org.infinispan.server.hotrod.HotRodDecoder.readHeader
(HotRodDecoder.java:184)
• em org.infinispan.server.hotrod.HotRodDecoder.decodeHeader
(HotRodDecoder.java:133)
• em org.infinispan.server.hotrod.HotRodDecoder.decode
(HotRodDecoder.java:92)
• em
io.netty.handler.codec.ByteToMessageDecoder.callDecode(ByteToMessage
Decoder.java:411)
    em
io.netty.handler.codec.ByteToMessageDecoder.channelRead(ByteToMessa
geDecoder.java:248)

```

e você vê alguns erros semelhantes no registro Keycloak, pode indicar que existem versões incompatíveis do protocolo Hot Rod sendo usado. Isso provavelmente acontece quando você tenta usar o Keycloak com uma versão antiga do servidor Infinispan. Ele ajudará se você adicionar a propriedade `protocolVersion` como uma propriedade adicional ao elemento de armazenamento remoto no arquivo de configuração Keycloak. Por exemplo:

```
< nome da propriedade="protocolVersion">2.6</propriedade>
```

Gerenciar configuração de subsistema

A configuração de baixo nível do Keycloak é feita editando o arquivo `autônomo.xml`, `autônomo-ha.xml` ou `domínio.xml` em sua distribuição. A localização deste arquivo depende do seu modo de [operação](#).

Embora existam configurações infinitas que você pode configurar aqui, esta seção se concentrará na configuração do subsistema *keycloak-server*. Não importa qual arquivo de configuração você esteja usando, a configuração do subsistema *keycloak-server* é a mesma.

O subsistema *keycloak-server* é normalmente declarado no final do arquivo como este:

```

<subssistema xmlns="urn:jboss:domínio:keycloak-server:1.1">
<se>auth</web-context>
...
</subssistema>

```

Observe que qualquer coisa alterada neste subsistema não terá efeito até que o servidor seja reiniciado.

Configurar provedores de SPI

As especificidades de cada configuração são discutidas em outros lugares em contexto com essa configuração. No entanto, é útil entender o formato usado para declarar configurações em provedores de SPI.

Keycloak é um sistema altamente modular que permite grande flexibilidade. Existem mais de 50 interfaces de provedor de serviços (SPIs), e você está autorizado a trocar implementações de cada SPI. A implementação de um SPI é conhecida como *provedor*.

Todos os elementos em uma declaração SPI são opcionais, mas uma declaração de SPI completa se parece com isso:

```
<spi nome ="myspi">
  <se-provedor de></provedorde inadimplência>
<provo nome="myprovider" ativado="verdadeiro">
  <propriedades>
<o nome da propriedade= valor "foo" ="bar"/>
  </propriedades>
</provedor>
<provider nome ="mysecondprovider" ativado="verdadeiro">
  <propriedades>
<o nome da propriedade=valor "foo" ="foo"/>
  </propriedades>
</provedor>
</spi>
```

Aqui temos dois provedores definidos para o SPI `myspi`. O `provedor padrão` está listado como `myprovider`. No entanto, cabe ao SPI decidir como tratará essa configuração. Algumas SPIs permitem mais de um provedor e outras não. Assim, o `provedor padrão` pode ajudar o SPI a escolher.

Observe também que cada provedor define seu próprio conjunto de propriedades de configuração. O fato de ambos os provedores acima terem uma propriedade chamada `foo` é apenas uma coincidência.

O tipo de valor de cada imóvel é interpretado pelo provedor. No entanto, há uma exceção. Considere o provedor `jpa` para o SPI de `eventosStore`:

```
nome <spi ="eventsStore">
<provo nome="jpa" ativado="verdadeiro">
  <propriedades>
< nome da propriedade= valor "excluir eventos" ="[" EVENT1" ,
```

```
" EVENT2" ]"/>
  </propriedades>
</provedor>
</spi>
```

Vemos que o valor começa e termina com suportes quadrados. Isso significa que o valor será repassado ao provedor como uma lista. Neste exemplo, o sistema passará ao provedor uma lista com dois valores de *elementos* *EVENT1* e *EVENT2*. Para adicionar mais valores à lista, basta separar cada elemento da lista com uma vírgula. Infelizmente, você precisa escapar das cotações em torno de cada elemento da lista com `"`.

Siga os passos no [Guia do Desenvolvedor de Servidores](#) para obter mais detalhes sobre provedores personalizados e a configuração dos provedores.

Inicie o WildFly CLI

Além de editar a configuração manualmente, você também tem a opção de alterar a configuração emitindo comandos através da ferramenta *jboss-cli*. O CLI permite configurar servidores local ou remotamente. E é especialmente útil quando combinado com scripting.

Para iniciar o WildFly CLI, você precisa executar `jboss-cli`.

Linux/Unix

```
$ .../bin/jboss-cli.sh
```

Windows

```
> ...\\bin\\jboss-cli.bat
```

Isso vai levá-lo a um prompt como este:

rápido

```
[desconectado /]
```

Se desejar executar comandos em um servidor em execução, executará primeiro o comando `connect`.

ligar

```
[desconexão /] conectar
```

ligar

```
[standalone@localhost:9990 /]
```

Você pode estar pensando consigo mesmo: "Eu não digitei nenhum nome de usuário ou senha!". Se você executar `jboss-cli` na mesma máquina que seu servidor autônomo em execução ou controlador de domínio e sua conta tiver permissões de arquivo

apropriadas, você não precisa configurar ou inserir em um nome de usuário e senha administrativos. Consulte a [Documentação WildFly 23](#) para obter mais detalhes sobre como tornar as coisas mais seguras se você estiver desconfortável com essa configuração.

Modo embarcado CLI

Se acontecer de você estar na mesma máquina que seu servidor autônomo e quiser emitir comandos enquanto o servidor não estiver ativo, você pode incorporar o servidor no CLI e fazer alterações em um modo especial que desautore as solicitações recebidas. Para fazer isso, execute primeiro o comando `incorporar servidor` com o arquivo config que deseja alterar.

incorporar servidor

```
[desconectado /] servidor incorporado --servidor-config=autônomo.xml  
[standalone@embedded /]
```

Modo CLI GUI

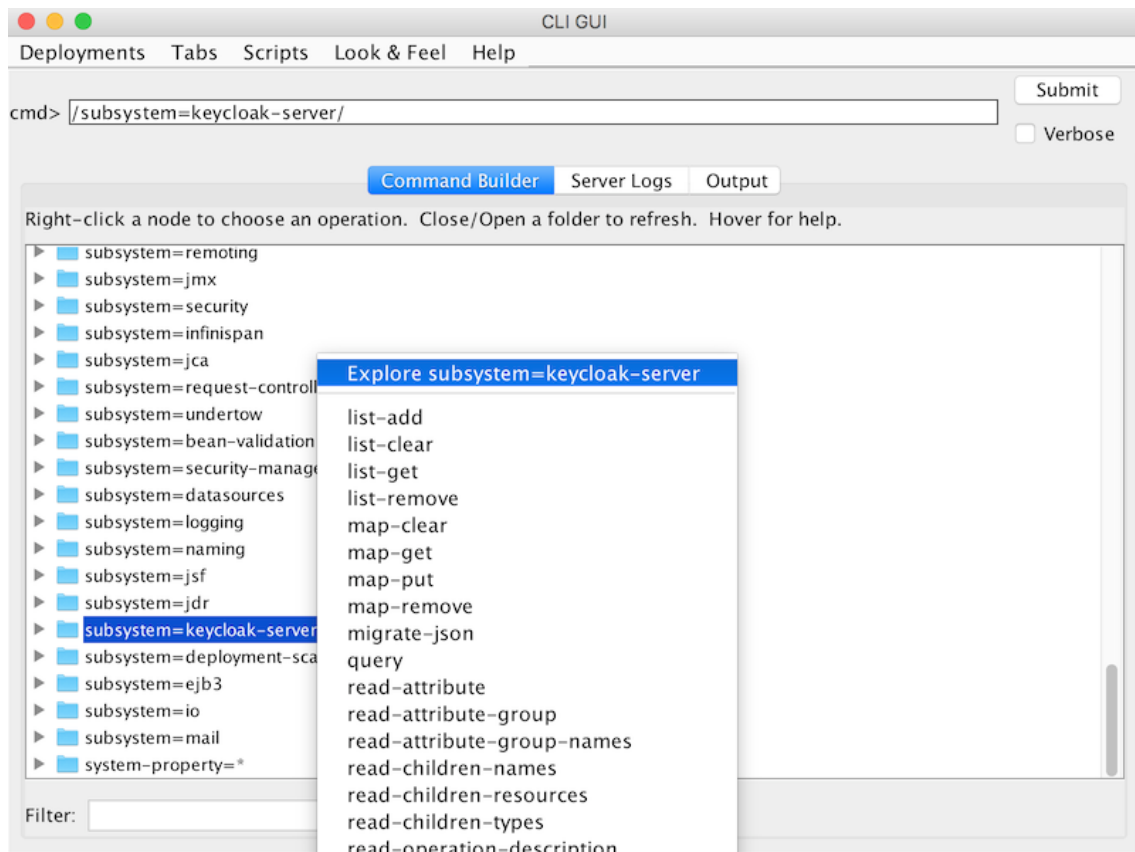
O CLI também pode ser executado no modo GUI. O modo GUI lança um aplicativo Swing que permite visualizar e editar graficamente todo o modelo de gerenciamento de um servidor *em execução*. O modo GUI é especialmente útil quando você precisa de ajuda para formatar seus comandos CLI e aprender sobre as opções disponíveis. A GUI também pode recuperar registros de servidores de um servidor local ou remoto.

Iniciar no modo GUI

```
$ ../bin/jboss-cli.sh --gui
```

Nota: para se conectar a um servidor remoto, você passa a opção `-conectar` também. Use a opção `--ajuda` para obter mais detalhes.

Depois de iniciar o modo GUI, você provavelmente vai querer rolar para baixo para encontrar o nó, `subsystem=keycloak-server`. Se você clicar com o botão direito do mouse no nó e clicar em **Explorar o subsistema=keycloak-server**, você receberá uma nova guia que mostra apenas o subsistema keycloak-server.



Roteiro CLI

O CLI tem extensas capacidades de scripting. Um script é apenas um arquivo de texto com comandos CLI nele. Considere um script simples que desaque o cache de tema e modelo.

turn-off-caching.cli

```
/subsystem=keycloak-server/theme=defaults/:write-attribute(name=cacheThemes,value=false)
/subsystem=keycloak-server/theme=defaults/:write-attribute(name=cacheTemplates,value=false)
```

Para executar o script, posso seguir o menu **Scripts** na GUI CLI, ou executar o script a partir da linha de comando da seguinte forma:

```
$ ../bin/jboss-cli.sh --file=turn-off-caching.cli
```

Receitas CLI

Aqui estão algumas tarefas de configuração e como executá-las com comandos CLI. Observe que em todos, exceto no primeiro exemplo, usamos o caminho curinga ****** para significar que você deve substituir ou o caminho para o subsistema keycloak-server.

Para autônomos, isso significa apenas:

```
** = /subsistema=keycloak-server
```

Para o modo de domínio, isso significaria algo como:

```
** = /profile=auth-server-clustered/subsystem=keycloak-server
```

Alterar o contexto web do servidor

```
/subsistema=keycloak-server/:write-attribute(name=web-context,value=myContext)
```

Defina o tema padrão global

```
*/theme=defaults/:write-attribute(name=default,value=myTheme)
```

Adicione um novo SPI e um provedor

```
*/spi=mySPI/:add  
*/spi=mySPI/provider=myProvider/:add(enabled=true)
```

Desativar um provedor

```
*/spi=mySPI/provider=myProvider/:write-attribute(name=enabled,value=false)
```

Alterar o provedor padrão para um SPI

```
*/spi=mySPI/:write-attribute (name=default-provider,value=myProvider)
```

Configure o SPI dblock

```
*/spi=dblock/:add(default-provider=jpa)  
*/spi=dblock/provider=jpa/:add(properties={lockWaitTimeout => "900"},enabled=true)
```

Adicionar ou alterar um único valor de propriedade para um provedor

```
*/spi=dblock/provider=jpa/:map-put(name=properties,key=lockWaitTimeout,value=3)
```

Remova uma única propriedade de um provedor

```
*/spi=dblock/provider=jpa/:map-remove(nome=propriedades,key=lockRecheckTime)
```

Definir valores em uma propriedade provedora de lista de tipo

```
*/spi=eventsStore/provider=jpa/:map-put(name=properties,key=exclude-  
events,value=[EVENT1,EVENT2])
```

Perfis

Existem recursos no Keycloak que não estão habilitados por padrão, estes incluem recursos que não são totalmente suportados. Além disso, existem alguns recursos que são habilitados por padrão, mas que podem ser desativados.

Os recursos que podem ser ativados e desativados são:

nome	descrição
conta2	Novo console de gerenciamento de contas
account_api	API REST de gerenciamento de contas
admin_fine_grained_authz	Permissões de administração de grãos finos
ciba	Autenticação de backchannel iniciada pelo cliente OpenID Connect (CIBA)
client_policies	Adicionar políticas de configuração do cliente
paridade	OAuth 2.0 Solicitações de Autorização Empurrada (PAR)
declarative_user_profile	Configure perfis de usuários usando um estilo declarativo
estivador	Protocolo de Registro Docker
representação	Capacidade de administradores se passarem por usuários
openshift_integration	Extensão para habilitar a garantia do OpenShift
Scripts	Escreva autenticadores personalizados usando JavaScript
token_exchange	Serviço de troca de tokens
upload_scripts	Enviar scripts
web_authn	Autenticação da Web W3C (WebAuthn)

Para habilitar todos os recursos de visualização, inicie o servidor com:

```
bin/autônomo.sh | bat -Dkeycloak.profile=preview
```

Você pode defini-lo permanentemente criando o arquivo `autônomo/configuração/profile.properties` (ou `domínio/servidores/servidor-um/configuração/profile.propriedades` para `servidor-um` no modo de domínio). Adicione o seguinte ao arquivo:

```
perfil=visualização
```

Para habilitar um recurso específico, inicie o servidor com:

```
bin/autônomo.sh | bat -Dkeycloak.profile.feature.<feature name>=enabled
```

Por exemplo, para ativar o uso do Docker `-Dkeycloak.profile.feature.docker=enabled`.

Você pode definir isso permanentemente no arquivo `profile.properties` adicionando:

```
feature.docker=ativado
```

Para desativar um recurso específico, inicie o servidor com:

```
bin/autônomo.sh | bat -Dkeycloak.profile.feature.<feature name>=disabled
```

Por exemplo, desativar o uso de personificação -
`Dkeycloak.profile.feature.personificação=desativado`.

Você pode definir isso permanentemente no arquivo `profile.properties` adicionando:

```
feature.personificação=desativado
```

Configuração do banco de dados relacional

Keycloak vem com seu próprio banco de dados relacional baseado em Java incorporado chamado H2. Este é o banco de dados padrão que o Keycloak usará para persistir dados e realmente só existe para que você possa executar o servidor de autenticação fora da caixa. Recomendamos que você o substitua por um banco de dados externo mais pronto para produção. O banco de dados H2 não é muito viável em situações de alta concorrência e também não deve ser usado em um cluster. O objetivo deste capítulo é mostrar como conectar o Keycloak a um banco de dados mais maduro.

Keycloak usa duas tecnologias em camadas para persistir seus dados relacionais. A tecnologia em camadas inferiores é jdbc. JDBC é uma API Java que é usada para se conectar a um RDBMS. Existem diferentes drivers JDBC por tipo de banco de dados que são fornecidos pelo seu fornecedor de banco de dados. Este capítulo discute como configurar o Keycloak para usar um desses drivers específicos do fornecedor.

A tecnologia em camadas superiores para persistência é a Hibernate JPA. Este é um objeto para a API de mapeamento relacional que mapeia Objetos Java para dados relacionais. A maioria das implantações do Keycloak nunca terá que tocar nos aspectos de configuração do Hibernate, mas discutiremos como isso é feito se você encontrar essa rara circunstância.

A configuração de datasource é coberta muito mais detalhadamente no [capítulo de configuração WildFly 23](#).

Lista de verificação de configuração do RDBMS

Estas são as etapas que você precisará executar para obter um RDBMS configurado para Keycloak.

1. Localize e baixe um driver JDBC para o seu banco de dados
2. Embale o JAR do driver em um módulo e instale este módulo no servidor
3. Declare o driver JDBC no perfil de configuração do servidor
4. Modifique a configuração de datasource para usar o driver JDBC do seu banco de dados
5. Modifique a configuração de datasource para definir os parâmetros de conexão ao seu banco de dados

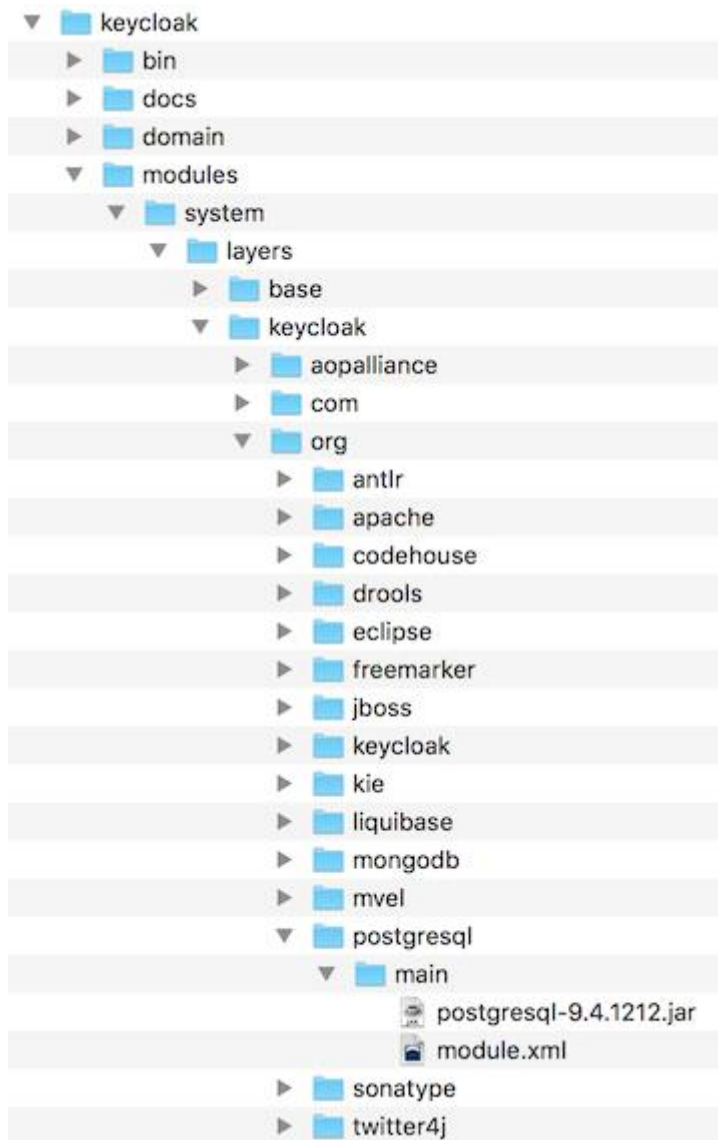
Este capítulo usará PostgreSQL para todos os seus exemplos. Outros bancos de dados seguem os mesmos passos para instalação.

Empacotar o Driver JDBC

Encontre e baixe o JAR do driver JDBC para o seu RDBMS. Antes de usar este driver, você deve empacotá-lo em um módulo e instalá-lo no servidor. Os módulos definem JARs que são carregados no keycloak classpath e as dependências que esses JARs têm em outros módulos. Eles são muito simples de configurar.

Dentro dos `.../módulos/diretório` de sua distribuição Keycloak, você precisa criar uma estrutura de diretório para manter a definição do módulo. A convenção é usar o nome do pacote Java do driver JDBC para o nome da estrutura do diretório. Para postgresSQL, crie a *orgia de diretório/postgresql/main*. Copie o JAR do driver do banco de dados neste diretório e crie um arquivo `.xml` *módulo* vazio dentro dele também.

Diretório de módulos



Depois de fazer isso, abra o *arquivo .xml módulo* e crie o seguinte XML:

Módulo XML

```
<?xml versão="1.0" ?>
<module xmlns="urn:jboss:module:1.3" nome="org.postgresql">

  <recursos>
    <resource-root path="postgresql-9.4.1212.jar"/>
  </recursos>

  <dependências>
    <module nome="javax.api"/>
    <module nome="javax.transaction.api"/>
  </dependências>
</módulo>
```

O nome do módulo deve corresponder à estrutura do diretório do seu módulo. Então, *mapas org/postgresql* para `org.postgresql`. O atributo de caminho raiz de recurso deve especificar o nome de arquivo JAR do driver. O resto são apenas as dependências normais que qualquer JAR driver JDBC teria.

Declarar e Carregar motorista JDBC

A próxima coisa que você precisa fazer é declarar seu driver JDBC recém-embalado em seu perfil de implantação para que ele carregue e fique disponível quando o servidor inicializado. Onde você executa esta ação depende do seu [modo de operação](#). Se estiver implantando no modo padrão, edite `.../autônomo/configuração/autônomo.xml`. Se estiver implantando no modo de cluster padrão, edite `.../autônomo/configuração/autônomo-ha.xml`. Se estiver implantando no modo de domínio, edite `.../domínio/configuração/domínio.xml`. No modo de domínio, você precisará ter certeza de editar o perfil que está usando: `autônomo do servidor auth` ou `agrupado por servidor auth`

Dentro do perfil, procure o bloco XML dos drivers dentro do subsistema de datasources. Você deve ver um driver pré-definido declarado para o driver H2 JDBC. É aqui que você declarará o driver JDBC para o seu banco de dados externo.

Motoristas JDBC

```
<subssistema xmlns="urna:jboss:domínio:datasources:6.0">
  <dados>
    ...
    <motoristas>
<driver name = módulo" h2" = "com.h2database.h2">
<xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
      </motorista>
    </motoristas>
  </dados>
</subssistema>
```

Dentro do bloco XML dos drivers, você precisará declarar um driver JDBC adicional. Ele precisa ter um nome que você pode escolher para ser o que quiser. Você especifica o atributo do módulo que aponta para o pacote de módulo que você criou anteriormente para o JAR do driver. Finalmente você tem que especificar a classe Java do motorista. Aqui está um exemplo de instalação do driver PostgreSQL que vive no exemplo do módulo definido anteriormente neste capítulo.

Declare seus drivers JDBC

```
<subssistema xmlns="urna:jboss:domínio:datasources:6.0">
  <dados>
    ...
    <motoristas>
<driver name = módulo" postgresql" = "org.postgresql">
```

```

<xa-datasource-class>org.postgresql.xa.PGXADatasource</xa-datasource-class>
  </motorista>
<driver name = módulo"h2" = "com.h2database.h2">
<xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
  </motorista>
</motoristas>
</dados>
</subsistema>

```

Modifique a fonte de dados Keycloak

Depois de declarar seu driver JDBC, você tem que modificar a configuração de datasource existente que o Keycloak usa para conectá-lo ao seu novo banco de dados externo. Você fará isso dentro do mesmo arquivo de configuração e do bloco XML em que registrou seu driver JDBC. Aqui está um exemplo que configura a conexão com seu novo banco de dados:

Declare seus drivers JDBC

```

<subsistema xmlns="urna:jboss:domínio:datasources:6.0">
  <dados>
    ...
    <datafonte jndi-name="java:jboss/datasources/KeycloakDS" nome do
pool="KeycloakDS" ativado="true" use-java-context="true">
    <connection-url>jdbc:postgresql://localhost/keycloak</connection-url>
    <></motorista>
      <pool>
        < tamanho dapiscina>20</tamanhomáximo da piscina>
      </piscina>
      < segurança>
        < nome dousuário>William</nomede usuário>
    <password></senha>
      </segurança>
    </datasource>
    ...
  </dados>
</subsistema>

```

Pesquise a definição de datasource para KeycloakDS. Primeiro você precisará modificar o url de conexão. A documentação para a implementação jdbc do seu fornecedor deve especificar o formato para esse valor de URL de conexão.

Em seguida, defina o driver que você usará. Este é o nome lógico do driver JDBC que você declarou na seção anterior deste capítulo.

É caro abrir uma nova conexão a um banco de dados toda vez que você quiser realizar uma transação. Para compensar, a implementação da fonte de dados mantém um pool de conexões abertas. O tamanho máximo do pool especifica o número máximo de conexões que irá acumular. Você pode querer alterar o valor disso dependendo da carga do seu sistema.

Finalmente, pelo menos com o PostgreSQL, você precisa definir o nome de usuário e a senha do banco de dados necessários para se conectar ao banco de dados. Você pode estar preocupado que isso esteja em texto claro no exemplo. Existem métodos para ofuscar isso, mas isso está além do escopo deste guia.

Para obter mais informações sobre recursos de datasource, consulte [o capítulo de configuração da Documentação WildFly 23](#).

Configuração do banco de dados

A configuração deste componente é encontrada no arquivo `autônomo.xml`, `autônomo-ha.xml` ou `domínio.xml` em sua distribuição. A localização deste arquivo depende do seu modo de [operação](#).

Banco de dados Config

```
<subsistema xmlns="urn:jboss:domínio:keycloak-server:1.1">
  ...
  nome <spi="conexõesJpa">
    <provo nome="padrão" ativado="verdadeiro">
      <propriedades>
        <nomeda propriedade= valor"dataSource"="java:jboss/datasources/KeycloakDS"/>
        <onomeda propriedade= valor"initializeEmpty"="falso"/>
        <onomeda propriedade= valor"migrationStrategy"="manual"/>
        <nomeda propriedade= valor"migrationExport"="\${jboss.home.dir}/keycloak-
        database-update.sql"/>
      </propriedades>
    </provedor>
  </spi>
  ...
</subsistema>
```

As opções de configuração possíveis são:

dataSource

Nome JNDI do dataSource

jta

propriedade booleana para especificar se a fonte de dados é capaz de JTA

driverDialect

Valor do dialeto do banco de dados. Na maioria dos casos, você não precisa especificar esta propriedade, pois o dialeto será detectado automaticamente pela Hibernate.

initializeEmpty

Inicialize o banco de dados se estiver vazio. Se definido como falso, o banco de dados deve ser inicializado manualmente. Se você quiser inicializar manualmente o conjunto de banco de dados `migrationStrategy` para `manual`, que criará um arquivo com comandos SQL para inicializar o banco de dados. Padrão para verdade.

migraçãoStrategy

Estratégia para usar para migrar banco de dados. Os valores válidos são `atualizar`, `manualmente` e `validar`. A atualização migrará automaticamente o esquema do banco de dados. O manual exportará as alterações necessárias para um arquivo com comandos SQL que você pode executar manualmente no banco de dados. A validação simplesmente verificará se o banco de dados está atualizado.

migraçãoExporte

Caminho para onde escrever arquivo manual de inicialização/migração do banco de dados.

showSql

Especifique se o Hibernate deve mostrar todos os comandos SQL no console (falso por padrão). Isso é muito verboso!

formatoSql

Especifique se o Hibernate deve formatar comandos SQL (verdadeiros por padrão)

globalStatsInterval

Registrará estatísticas globais da Hibernate sobre consultas DB executadas e outras coisas. As estatísticas são sempre relatadas ao registro do servidor no intervalo especificado (em segundos) e são limpas após cada relatório.

esquema

Especifique o esquema de banco de dados para usar

Esses switches de configuração e muito mais estão descritos no [Guia de Desenvolvimento](#)

Considerações unicode para bancos de dados

O esquema de banco de dados em Keycloak só contabiliza strings Unicode nos seguintes campos especiais:

- Reinos: nome de exibição, nome de exibição HTML
- Provedores da Federação: nome de exibição
- Usuários: nome de usuário, nome dado, sobrenome, nomes de atributos e valores
- Grupos: nome, nomes de atributos e valores
- Papéis: nome
- Descrições de objetos

Caso contrário, os caracteres são limitados àqueles contidos na codificação do banco de dados, que muitas vezes é de 8 bits. No entanto, para alguns sistemas de banco de dados, é possível habilitar a codificação UTF-8 de caracteres Unicode e usar o conjunto completo de caracteres Unicode em todos os campos de texto. Muitas vezes, isso é contrabalançado por um comprimento máximo mais curto das cordas do que no caso de codificações de 8 bits.

Alguns dos bancos de dados exigem configurações especiais para o banco de dados e/ou driver JDBC para ser capaz de lidar com caracteres Unicode. Por favor, encontre as configurações do seu banco de dados abaixo. Observe que se um banco de dados estiver listado aqui, ele ainda pode funcionar corretamente desde que manuseie a codificação UTF-8 corretamente, tanto no nível do banco de dados quanto no driver JDBC.

Tecnicamente, o principal critério para o suporte unicode para todos os campos é se o banco de dados permite a configuração do conjunto de caracteres Unicode para campos `VARCHAR` e `CHAR`. Se sim, há uma grande chance de que unicode será plausível, geralmente às custas do comprimento do campo. Se ele só suporta unicode nos campos `NVARCHAR` e `NCHAR`, o suporte unicode para todos os campos de texto é improvável, pois o esquema Keycloak usa extensivamente os campos `VARCHAR` e `CHAR`.

Banco de dados Oracle

Os caracteres Unicode são tratados corretamente desde que o banco de dados tenha sido criado com suporte unicode nos campos `VARCHAR` e `CHAR` (por exemplo, usando o conjunto de caracteres `AL32UTF8` como conjunto de caracteres de banco de dados). Não são necessárias configurações especiais para o driver JDBC.

Se o conjunto de caracteres do banco de dados não for Unicode, então para usar caracteres Unicode nos campos especiais, o driver JDBC precisa ser configurado com a propriedade de conexão `oracle.jdbc.defaultNChar` definida como `true`. Pode ser sábio, embora não estritamente necessário, também definir a propriedade de conexão `oracle.jdbc.convertNcharLiterals` como verdadeira. Essas propriedades podem ser definidas como propriedades do sistema ou como propriedades de conexão. Observe que a configuração `oracle.jdbc.defaultNChar` pode ter impacto negativo no desempenho. Para obter detalhes, consulte a documentação de configuração do driver Oracle JDBC.

Banco de dados do Microsoft SQL Server

Os caracteres Unicode são tratados adequadamente apenas para os campos especiais. Não são necessárias configurações especiais do driver ou banco de dados JDBC.

Banco de dados MySQL

Os caracteres Unicode são tratados corretamente desde que o banco de dados tenha sido criado com suporte unicode nos campos `VARCHAR` e `CHAR` no comando `CREATE DATABASE` (por exemplo, usando o conjunto de caracteres `utf8` como o caractere padrão do banco de dados definido no MySQL 5.5. Observe que o conjunto de caracteres `utf8mb4` não funciona devido a diferentes requisitos de armazenamento para `utf8` conjunto de caracteres ^[1]). Observe que, neste caso, a restrição de comprimento a campos não especiais não se aplica porque as colunas são criadas para acomodar a quantidade dada de caracteres, não bytes. Se o conjunto de caracteres padrão do banco de dados não permitir o armazenamento do Unicode, apenas os campos especiais permitirão armazenar valores Unicode.

Ao lado das configurações do driver JDBC, é necessário adicionar um caractere de propriedade de conexão `Encoding=UTF-8` às configurações de conexão JDBC.

Banco de dados pós-gresql

O Unicode é suportado quando o conjunto de caracteres do banco de dados é `UTF8`. Nesse caso, os caracteres Unicode podem ser usados em qualquer campo, não há redução do comprimento do campo para campos não especiais. Não são necessárias configurações especiais do driver JDBC.

O conjunto de caracteres de um banco de dados PostgreSQL é determinado no momento em que é criado. Você pode determinar o conjunto de caracteres padrão para um cluster PostgreSQL com o comando SQL

```
mostrar server_encoding;
```

Se o conjunto de caracteres padrão não for UTF 8, então você pode criar o banco de dados com UTF8 como seu conjunto de caracteres assim:

criar keycloak **de banco de dados** com codificação 'UTF8';

Nome do host

Keycloak usa o nome do hospedeiro público para uma série de coisas. Por exemplo, nos campos de emissores de tokens e URLs enviados em e-mails de redefinição de senha.

O Hostname SPI fornece uma maneira de configurar o nome do host para uma solicitação. O provedor padrão permite definir uma URL fixa para solicitações de frontend, ao mesmo tempo em que permite que as solicitações de backend sejam baseadas na URI de solicitação. Também é possível desenvolver seu próprio provedor caso o provedor incorporado não forneça a funcionalidade necessária.

Provedor padrão

O provedor de nome de host padrão usa o `frontendUrl` configurado como URL base para solicitações frontend (solicitações de agentes de usuário) e usa a URL de solicitação como base para solicitações de backend (solicitações diretas de clientes).

A solicitação frontend não precisa ter o mesmo caminho de contexto que o servidor Keycloak. Isso significa que você pode expor o Keycloak, por [exemplo](#), <https://auth.example.org> ou <https://example.org/keycloak> enquanto internamente sua URL pode ser <https://10.0.0.10:8080/auth>.

Isso torna possível que os agentes de usuário (navegadores) enviem solicitações ao Keycloak através do nome de domínio público, enquanto os clientes internos podem usar um nome de domínio interno ou endereço IP.

Isso se reflete no ponto final do OpenID Connect Discovery, por exemplo, onde o `authorization_endpoint` usa a URL frontend, enquanto `token_endpoint` usa a URL backend. Como nota aqui, um cliente público, por exemplo, entraria em contato com a Keycloak através do ponto final público, o que resultaria na base de `authorization_endpoint` e `token_endpoint` sendo o mesmo.

Para definir o `frontendUrl` para Keycloak, você pode passar adicionar - `Dkeycloak.frontendUrl=https://auth.example.org` à inicialização ou você pode configurá-lo em `autônomo.xml`. Veja o exemplo abaixo:

```
nome <spi ="hostname">
<se>efault</provedor deinadimplência>
<provo nome ="padrão" ativado="verdadeiro">
  <propriedades>
    <o nomeda propriedade= valor "frontendUrl" ="https://auth.example.com"/>
    < nomeda propriedade= valor "forceBackendUrlToFrontendUrl" =valor "falso"/>
  </propriedades>
```

```
</provedor>  
</spi>
```

Para atualizar o `frontendUrl` com `jboss-cli` use o seguinte comando:

```
/subsystem=keycloak-server/spi=hostname/provider=default:write-  
attribute(name=properties.frontendUrl,value="https://auth.example.com")
```

Se você quiser que todas as solicitações passem pelo nome de domínio público, você pode forçar as solicitações de backend a usar a URL frontend, bem como definindo `forceBackendUrlToFrontendUrl` como `true`.

Também é possível substituir a URL de frontend padrão para reinos individuais. Isso pode ser feito no console administrativo.

Se você não quiser expor os pontos finais do administrador e o console no domínio público use o `adminUrl` de propriedade para definir uma URL fixa para o console administrativo, que é diferente do `frontendUrl`. Também é necessário bloquear o acesso ao `/auth/administrador` externamente, para obter detalhes sobre como fazer isso se refere ao Guia de Administração do [Servidor](#).

Provedor personalizado

Para desenvolver um provedor personalizado de hostname, você precisa implementar `org.keycloak.urls.HostnameProviderFactory` e `org.keycloak.urls.HostnameProvider`.

Siga as instruções na seção Interfaces do Provedor de Serviços no [Server Developer Guide](#) para obter mais informações sobre como desenvolver um provedor personalizado.

Configuração da rede

Keycloak pode ficar sem a caixa com algumas limitações de rede. Por um lado, todos os pontos finais da rede se ligam ao `localhost` para que o servidor auth seja realmente apenas utilizável em uma máquina local. Para conexões baseadas em HTTP, ele não usa portas padrão como 80 e 443. HTTPS/SSL não está configurado fora da caixa e sem ele, o Keycloak tem muitas vulnerabilidades de segurança. Finalmente, o Keycloak pode muitas vezes precisar fazer conexões seguras de SSL e HTTPS para servidores externos e, portanto, precisar de uma loja de confiança configurada para que os pontos finais possam ser validados corretamente. Este capítulo discute todas essas coisas.

Endereços de vinculação

Por padrão, o Keycloak se liga ao endereço de loopback localhost 127.0.0.1. Esse não é um padrão muito útil se você quiser o servidor de autenticação disponível em sua rede. Geralmente, o que recomendamos é que você implante um proxy reverso ou um balanceador de carga em uma rede pública e via tráfego para instâncias individuais de servidor Keycloak em uma rede privada. Em ambos os casos, porém, você ainda precisa configurar suas interfaces de rede para se vincular a algo diferente do localhost.

Definir o endereço de vinculação é bastante fácil e pode ser feito na linha de comando com os scripts de inicialização *standalone.sh* ou *domain.sh* discutidos no capítulo [Escolher um Modo de Operação](#).

```
$ standalone.sh -b 192.168.0.5
```

O switch `-b` define o endereço de ligação IP para quaisquer interfaces públicas.

Alternativamente, se você não quiser definir o endereço de vinculação na linha de comando, você pode editar a configuração de perfil de sua implantação. Abra o arquivo de configuração de perfil (*autônomo.xml* ou *domínio.xml* dependendo do seu modo [de operação](#)) e procure as interfaces do bloco XML.

```
<interfaces>
  <interface nome="gerenciamento">
<indutode endereço="{jboss.bind.address.management:127.0.0.1}"/>
  </interface>
<interface nome => "público"
<em valorde endereço="{jboss.bind.address:127.0.0.1}"/>
  </interface>
</interfaces>
```

A interface pública corresponde a subsistemas que criam soquetes disponíveis publicamente. Um exemplo de um desses subsistemas é a camada web que serve os pontos finais de autenticação do Keycloak. A interface de gerenciamento corresponde a tomadas abertas pela camada de gerenciamento do WildFly. Especificamente os soquetes que permitem usar a interface de linha de comando *jboss-cli.sh* e o console web WildFly.

Ao olhar para a interface pública, você vê que ele tem uma string especial `{jboss.bind.address:127.0.0.1}`. Esta sequência denota um valor 127.0.0.1 que pode ser substituído na linha de comando definindo uma propriedade do sistema Java, ou seja:

```
$ domain.sh -Djboss.bind.address=192.168.0.5
```

O `-b` é apenas uma notação abreviada para este comando. Assim, você pode alterar o valor do endereço de vinculação diretamente no perfil config, ou alterá-lo na linha de comando quando você inicializar.

Há muito mais opções disponíveis ao configurar definições de interface. Para obter mais informações, consulte a *Documentação WildFly 23*.

Amarras da porta do soquete

As portas abertas para cada tomada têm um padrão pré-definido que pode ser substituído na linha de comando ou dentro da configuração. Para ilustrar essa configuração, vamos fingir que você está executando no [modo autônomo](#) e abrir o `.../autônomo/configuração/autônomo.xml`. Procure por grupo de ligação de soquete.

```
<socket-vinculação nome="grupo" grupo="standard-sockets" default-interface="público"
port-offset="{jboss.socket.binding.port-offset:0}">
<socket-vinculação nome="interface"management-http" = porta"gerenciamento"
="{jboss.management.http.port:9990}">
<socket-vinculação nome="interface"management-https" = porta"gerenciamento"
="{jboss.management.https.port:9993}">
<socket-vinculação nome="porta"ajp"="{jboss.ajp.port:8009}">
<socket-vinculação nome="porta"http"="{jboss.http.port:8080}">
<socket-vinculação nome="porta"https"="{jboss.https.port:8443}">
<socket-vinculação nome="porta"txn-recovery-environment"="4712"/>
<socket-vinculação nome="porta"txn-status-manager"="4713"/>
<nome de ligação de soquete de saída="mail-smtp"
<reter de destino de<remote = porta"localhost"="25"/>
  </out-socket-binding>
</grupo de ligação de soquete>
```

as ligações de soquete definem as conexões do soquete que serão abertas pelo servidor. Essas vinculações especificam a interface (endereço de ligação) que usam, bem como qual número de porta abrirão. Os que você mais se interessará são:

http

Define a porta usada para conexões KEYcloak HTTP

https

Define a porta usada para conexões HTTPS Keycloak

ajp

Esta ligação de soquete define a porta usada para o protocolo AJP. Este protocolo é usado pelo servidor Apache HTTPD em conjunto mod-cluster quando você está usando Apache HTTPD como um balanceador de carga.

gestão-http

Define a conexão HTTP usada pela WildFly CLI e pelo console web.

Ao executar no [modo de domínio](#), definir as configurações do soquete é um pouco mais complicado como o domínio *exemplo.xml* arquivo tem vários grupos de vinculação de soquete definidos. Se você rolar para baixo para as definições do grupo de servidor, você pode ver qual grupo de vinculação de soquete é usado para cada grupo de servidor.

ligações de tomada de domínio

```
<serviente>
<do grupo de servidor= perfil "grupo de balanceador de carga" = "balanceador de
carga">
    ...
    <socket-binding-group ref="load-balancer-sockets"/>
</grupo de servidores>
<do nome do grupo do servidor= perfil "auth-server-group" => "auth-server-clustered"
    ...
    <socket-binding-group ref="ha-sockets"/>
</grupo de servidores>
</grupos de servidores>
```

Há muito mais opções disponíveis ao configurar definições de grupo de ligação de soquete. grupo de vinculação de [soquetes](#) na *Documentação WildFly 23*.

Configuração https/SSL

O Keycloak não está configurado por padrão para lidar com SSL/HTTPS. É altamente recomendado usar próprio servidor Keycloak ou em um proxy reverso na frente do servidor Keycloak.

Esse comportamento padrão é definido pelo modo SSL/HTTPS de cada reino keycloak. Isso é discutido com mais detalhes no Guia de Administração do [Servidor](#), mas vamos dar algum contexto e uma breve visão geral desses modos.

solicitações externas

O keycloak pode ficar sem SSL desde que você se atenha a endereços IP privados como localhost, 127.0.0.1, 10.x.x.x, 192.168.x.x, e 172.16.x.x. Se você não tiver O SSL/HTTPS configurado no servidor ou tentar acessar o Keycloak em HTTP a partir de um endereço IP não privado, você terá um erro.

nenhum

Keycloak não requer SSL. Isso só deve ser usado no desenvolvimento quando você está brincando com as coisas.

todos os pedidos

Keycloak requer SSL para todos os endereços IP.

O modo SSL para cada reino pode ser configurado no console administrador Keycloak.

Habilitação de SSL/HTTPS para o Servidor Keycloak

Se você não estiver usando um proxy reverso ou um balanceador de carga para lidar com o tráfego HTTPS para você, você precisará habilitar HTTPS para o servidor Keycloak. Isso envolve

1. Obtendo ou gerando um armazenamento de chaves que contenha a chave privada e o certificado para tráfego SSL/HTTP
2. Configurando o servidor Keycloak para usar este servidor e certificado de chave.

Criando o Certificado e o Java Keystore

Para permitir conexões HTTPS, você precisa obter um certificado assinado por terceiros ou auto-assinados e importá-lo em uma loja de chaves Java antes de ativar https no contêiner web para o que você está implantando o Servidor Keycloak.

Certificado auto-assinado

Em desenvolvimento, você provavelmente não terá um certificado assinado por terceiros disponível para testar uma implantação do Keycloak, então você precisará gerar um auto-assinado usando o utilitário `keytool` que vem com o Java JDK.

```
$ keytool -genkey -alias localhost -keyalg RSA -keystore keystore keycloak.jks -validade 10950
```

Digite a senha do keystore: segredo

Reinsira nova senha: segredo

Qual é o seu primeiro e sobrenome?

[Desconhecido]: localhost

Qual é o nome da sua unidade organizacional?

[Desconhecido]: Keycloak

Qual é o nome da sua organização?

[Desconhecido]: Chapéu Vermelho

Qual é o nome da sua cidade ou localidade?

[Desconhecido]: Westford

Qual é o nome do seu Estado ou província?

[Desconhecido]: MA

Qual é o código de país de duas letras para esta unidade?

[Desconhecido]: EUA

CN=localhost, OU=Keycloak, O=Test, L=Westford, ST=MA, C=US correto?

[não]: sim

Você deve responder Qual é o seu primeiro e sobrenome? pergunta com o nome DNS da máquina em que você está instalando o servidor. Para fins de teste, o localhost deve ser usado. Após a execução deste comando, o arquivo keycloak.jks será gerado no mesmo diretório que você executou o comando keytool.

Se você quer um certificado assinado por terceiros, mas não tem um, você pode obter um gratuitamente em cacert.org. Você vai ter que fazer um pouco de configuração primeiro antes de fazer isso embora.

A primeira coisa a fazer é gerar uma Solicitação de Certificado:

```
$ keytool -certreq -alias yourdomain -keystore keystore keycloak.jks > keycloak.careq
```

Ondeseudomínio é um nome DNS para o qual este certificado é gerado. O keytool gera a solicitação:

```
-----A NOVA SOLICITAÇÃO DE CERTIFICADO-----
MIIC2jCCACiCAQAwZTELMAkGA1UEBhMCVMMMMMMMBGNBBTAK1BMIDWYDVQHE
whXZXN0Zm9y
ZDEQMA4GA1UEChMHUHVkIEhhdDEQMA4GA1UECMHUHVkIEhhdDESMBAGA1UEAx
MJlbG9jYWxob3N0
MIIBIjANBgkqhkiG9w0BAQEFAAACEAQUESOAQ8AMIIBCgKCAQEAR7kck2TaavLEOGbcp
9c0rncY4HhdzmY
Ax2nZfq1eZEaIPqI5aTxwQZzzLDK9qbeAd8Ji79HzSqNRDxNYaZu7mAYhFKHgixsolE3o5Yfz
bw1
29RvyeUve+WZxv5oo9wolVVpdSINIMEL2LaFhtX/c1dqiYVpfnvFshZQalg2nL8juzZcBjj4
as
H98glS7khql/dkZKsw9NLvyxgJvp7PaXurX29fNf3ihG+oFrL22oFyV54BWWxXCKU/GPn61
EGZGw
Ft2qSIGLdctpMD1aJR2bcnhEjZKDksjQQ5YMXaAGkcYkG6QkgrocDE2YXDbi7GlDf9MegVJ
35
2DQMpwIDAQABoDAwLgYJKoZIhvcNAQkOMSEwHzAdBgNVHQ4EFgQUQwIzJBA+fjiDdi
VzaO9vrE/i
n2swDQYJKoZIhvcNAQELBQADggEBAC5FRMkhal3q86tHPBYWBuTtmcSjs4qUm6V6f63fr
hveWHf
PzRrI1xH272XUleBk0gtzWo0nNZnf0mMCtUBbHhhDcG82xolikfqibZijoQZCiGiedVjJFtni
DQ
9bMDUOXEMQ7gHZg5q6mJfNG9MbMpQaUVEEFvfGEQQxbiFK7hRWU8S23/d80e8nEx
gQxdJWJ6vd0X
MzzFK6j4Dj55bJVuM7GFmfdNC52pNOD5vYe47Aqh8oajHX9XTycVtPXl45rrWAH3ftbrS8
SrZ2S
vqIFQeuLL3BaHwpl3t7t7j2IMWcK1p80laAxEASib/fAwrrHhLHBBXRcq6uALUOZI4Alt8=
-----A NOVA SOLICITAÇÃO DE CERTIFICADO-----
```

Envie esta solicitação ca para o seu CA. A CA emitirá um certificado assinado e enviará para você. Antes de importar seu novo cert, você deve obter e importar o certificado raiz do CA. Você pode baixar o cert de CA (ou seja.

```
$ keytool -import -keystore keystore keycloak.jks -file root.crt -alias root
```

O último passo é importar seu novo certificado gerado por CA para sua loja de chaves:

```
$ keytool -import -alias yourdomain -keystore keystore keycloak.jks -archive seu  
certificado.cer
```

Configure keycloak para usar o keystore

Agora que você tem uma loja de chaves Java com os certificados apropriados, você precisa configurar sua instalação Keycloak para usá-la. Primeiro, você deve editar o arquivo *autônomo.xml*, *autônomo-ha.xml* ou *host.xml* para usar a loja de chaves e ativar HTTPS. Em seguida, você pode mover o arquivo keystore para a *configuração/diretório* de sua implantação ou o arquivo em um local que você escolher e fornecer um caminho absoluto para ele. Se estiver usando caminhos absolutos, remova o parâmetro opcional relativo ao parâmetro da configuração (Veja [o modo de operação](#)).

Adicione o novo elemento de reino de segurança usando o CLI:

```
$ /core-service=management/security-realm=UndertowRealm:add()  
  
$ /core-service=management/security-realm=UndertowRealm/server-  
identity=ssl:add(keystore-path=keycloak.jks, keystore-relative-  
to=jboss.server.config.dir, keystore-password=secret)
```

Se usar o modo de domínio, os comandos devem ser executados em cada host usando o `/host=<host_name>/` prefixo (a fim de criar o domínio de segurança em todos eles), como este, que você repetiria para cada host:

```
$ /host=<host_name>/core-service=management/security-  
realm=UndertowRealm/server-identity=ssl:add(keystore-path=keycloak.jks, keystore-  
relative-to=jboss.server.config.dir, keystore-password=secret)
```

No arquivo de configuração autônomo ou host, o elemento de domínio de segurança deve ser assim:

```
nome <security-realm ="UndertowRealm">  
  <identites>  
    <ssl>  
<desseja="keycloak.jks" relativo a="jboss.server.config.dir" keystore-  
password="secret" />  
    </ssl>  
  </identidades do servidor>  
</segurança>
```

Em seguida, no arquivo de configuração autônoma ou em cada domínio, procure por quaisquer instâncias de domínio de segurança. Modifique o `https-ouvinte` para usar o reino criado:

```
$ /subsistema=undertow/server=default-server/https-listener=https:write-attribute(name=security-realm, value=UndertowRealm)
```

Se usar o modo de domínio, prefixe o comando com o perfil que está sendo usado com: `/profile=<profile_name>/`.

O elemento resultante, nome do servidor="servidor padrão", que é um elemento infantil do `subsistema` `xmlns="urn:jboss:domain:undertow:12.0"`, deve conter a seguinte estrofe:

```
<subsistema xmlns="urn:jboss:domínio:undertow:12.0">
nome <cache <buffer="padrão"/>
nome <servidor => "servidor padrão"
<https-ouvinte nome="https" soquete-vinculação="https" security-
realm="UndertowRealm"/>
...
</subsistema>
```

Solicitações HTTP de saída

O servidor Keycloak muitas vezes precisa fazer solicitações HTTP não-navegador para os aplicativos e serviços que ele protege. O servidor auth gerencia essas conexões de saída mantendo um pool de conexão de cliente HTTP. Há algumas coisas que você precisará configurar em `autônomo.xml`, `autônomo-ha.xml` ou `domínio.xml`. A localização deste arquivo depende do seu modo de [operação](#).

Exemplo de config cliente HTTP

```
nome <spi ="conexõesHttpClient">
<provo nome ="padrão" ativado="verdadeiro">
  <propriedades>
< nome da propriedade= valor "tamanho de piscina de conexão" ="256"/>
  </propriedades>
</provedor>
</spi>
```

As opções de configuração possíveis são:

estabelecer-conexão-tempoout-millis

Tempo limite para estabelecer uma conexão de soquete.

socket-timeout-millis

Se uma solicitação de saída não receber dados por esse período de tempo, faça o tempo limite da conexão.

tamanho da piscina de conexão

Quantas conexões podem estar no pool (128 por padrão).

max-pooled-por-rota

Quantas conexões podem ser agrupadas por host (64 por padrão).

conexão-ttl-millis

Tempo máximo de conexão para viver em milissegundos. Não definido por padrão.

max-connection-ocioso-tempo-milis

Tempo máximo a conexão pode ficar ociosa no pool de conexão (900 segundos por padrão). Iniciará o segmento de limpeza de fundo do cliente Apache HTTP. Defina para -1 para desativar esta verificação e o segmento de fundo.

desabilitar cookies

verdade por padrão. Quando definido como verdadeiro, isso desativará qualquer cache de biscoito.

cliente-keystore

Este é o caminho do arquivo para um arquivo java keystore. Esta keystore contém certificado de cliente para SSL bidiretivo.

cliente-keystore-senha

Senha para a loja de chaves do cliente. Isso é *NECESSÁRIO* se a loja de chaves do cliente estiver definida.

cliente-chave-senha

Senha para a chave do cliente. Isso é *NECESSÁRIO* se a loja de chaves do cliente estiver definida.

mapeamentos por proxy

Denota configurações de proxy para solicitações HTTP de saída. Consulte a seção sobre [mapeamentos proxy para pedidos HTTP de saída](#) para obter mais detalhes.

desabilitar-trust-manager

Se uma solicitação de saída exigir HTTPS e esta opção de config for definida como verdadeira, você não precisa especificar uma loja de confiança. Esta configuração só deve ser usada durante o desenvolvimento e **nunca** em produção, pois desativará a verificação dos certificados SSL. Isso é *OPCIONAL*. O valor padrão é falso.

Mapeamentos de proxy para solicitações HTTP de saída

As solicitações HTTP enviadas pelo Keycloak podem usar opcionalmente um servidor proxy com base em uma lista delimitada de vírgula de mapeamentos por proxy. Um mapeamento por proxy denota a combinação de um padrão de hostname baseado em regex e um proxy-uri na forma de `hostnamePattern;proxyUri`, por exemplo:

```
.*\. (google|googleapis)\.com;http://www-proxy.acme.com:8080
```

Para determinar o proxy de uma solicitação HTTP de saída, o nome do host de destino é compatível com os padrões de nome do host configurados. O primeiro padrão de correspondência determina o proxy-uri a ser usado. Se nenhum dos padrões configurados corresponder ao nome do host dado, então nenhum proxy será usado.

Se o servidor proxy exigir autenticação, inclua as credenciais do usuário proxy neste formato de nome de usuário:password@. Por exemplo:

```
.*\. (google|googleapis)\.com;http://user01:pas2w0rd@www-proxy.acme.com:8080
```

O valor especial `NO_PROXY` para o proxy-uri pode ser usado para indicar que nenhum proxy deve ser usado para hosts que correspondam ao padrão hostname associado. É possível especificar um padrão de captura no final dos mapeamentos de proxy para definir um proxy padrão para todas as solicitações de saída.

O exemplo a seguir demonstra a configuração de mapeamento de proxy.

```
# Todas as solicitações para APIs do Google devem usar http://www-  
proxy.acme.com:8080 como proxy
```

```
.*\. (google|googleapis)\.com;http://www-proxy.acme.com:8080
```

```
# Todas as solicitações para sistemas internos devem não usar proxy
```

```
.*\.acme\.com;NO_PROXY
```

```
# Todas as outras solicitações devem usar http://fallback:8080 como proxy
```

```
.*;http://fallback:8080
```

Isso pode ser configurado através do seguinte comando `jboss-cli`. Observe que você precisa escapar adequadamente do padrão regex, como mostrado abaixo.

```
configuração de eco: configure rotas proxy para HttpClient SPI
```

```
No caso de não haver conexõesA definição deCcscient ainda
```

```
/subsystem=keycloak-
```

```
server/spi=connectionsHttpClient/provider=default:add(enabled=true)
```

```
# Configure os mapeamentos por proxy
```

```
/subsystem=keycloak-server/spi=connectionsHttpClient/provider=default:write-  
attribute(name=properties.proxy-
```

```
mappings,value=[".*\\. ( google |googleapis)\\.com;http://www-proxy.acme.com:8080",".*\\.acme\\.com;NO_PROXY",".*;http://fallback:8080"])
```

O comando `jboss-cli` resulta na seguinte configuração do subsistema. Note que é preciso codificar " caracteres com " .

```
nome <spi ="conexõesHttpClient">
<provo nome ="padrão" ativado="verdadeiro">
  <propriedades>
    <property
      nome="mapeamentos por proxy"
      valor="[" .*\\. (google|googleapis)\\.com;http://www-proxy.acme.com:8080 "," .*\\.acme\\.com;NO_PROXY" ," .*;http://fallback:8080" ]"/>
    </propriedades>
  </provedor>
</spi>
```

Saída HTTPS Request Truststore

Quando o Keycloak invoca pontos finais HTTPS remotos, ele tem que validar o certificado do servidor remoto para garantir que ele esteja se conectando a um servidor confiável. Isso é necessário para evitar ataques homens no meio. Os certificados desses servidores remotos ou do CA que assinaram esses certificados devem ser colocados em uma loja de confiança. Esta loja de confiança é gerenciada pelo servidor Keycloak.

O truststore é usado ao conectar-se com segurança a corretores de identidade, provedores de identidade LDAP, ao enviar e-mails e para comunicação backchannel com aplicativos clientes.

Por padrão, um provedor de truststore não está configurado, e quaisquer conexões https r truststore, conforme descrito no [Guia de Referência JSSE da Java](#). Se não houver confiança saída falharão.

Você pode usar o `keytool` para criar um novo arquivo truststore ou adicionar certificados de host confiáveis a um já existente:

```
$ keytool -import -alias HOSTDOMAIN -keystore truststore.jks -file host-certificate.cer
```

O truststore é configurado dentro do arquivo `autônomo.xml`, `autônomo-ha.xml` ou `domínio.xml` em sua distribuição. A localização deste arquivo depende do seu modo de [operação](#). Você pode adicionar sua configuração truststore usando o seguinte modelo:

```
nome <spi ="truststore">
<provo nome="arquivo" ativado="verdadeiro">
  <propriedades>
```



```
< nome da propriedade= valor de "arquivo" ="caminho para o seu arquivo .jks contendo
certificados públicos"/>
< nome da propriedade = valor de "senha" ="senha"/>
< nome da propriedade = valor "hostname-verification-policy" ="CURCARD"/>
<o nome da propriedade= valor "ativado" ="verdadeiro"/>
    </propriedades>
  </provedor>
</spi>
```

As opções de configuração possíveis para esta configuração são:

arquivo

O caminho para um arquivo java keystore. As solicitações HTTPS precisam de uma maneira de verificar o host do servidor com o que estão conversando. Isso é o que o trustore faz. O armazenamento de chaves contém um ou mais certificados de host confiáveis ou autoridades de certificado. Este arquivo truststore deve conter apenas certificados públicos de seus hosts protegidos. Isso é *NECESSÁRIO* se `habilitado` for verdade.

senha

Senha para a loja de confiança. Isso é *NECESSÁRIO* se `habilitado` for verdade.

hostname-verificação-política

`CURCARD` por padrão. Para solicitações HTTPS, isso verifica o nome de host do certificado do servidor. `ANY` significa que o nome do host não é verificado. `CURCARD` Permite curingas em nomes de subdomínios, ou seja, `*.foo.com`. `STRICT` CN deve corresponder exatamente ao nome do host.

Habilitado

Se for falso (valor padrão), a configuração do truststore será ignorada e a verificação do certificado voltará à configuração JSSE conforme descrito. Se definido como verdadeiro, você deve configurar `arquivo` e `senha` para o truststore.

Clustering

Esta seção abrange configurar Keycloak para executar em um cluster. Há uma série de coisas que você tem que fazer ao configurar um cluster, especificamente:

- [Escolha um modo de operação](#)
- [Configure um banco de dados externo compartilhado](#)

- Configure um balanceador de carga
- Fornecendo uma rede privada que suporta multicast IP

A escolha de um modo de operação e a configuração de um banco de dados compartilhado foram discutidas anteriormente neste guia. Neste capítulo discutiremos a criação de um balanceador de carga e o fornecimento de uma rede privada. Também discutiremos algumas questões que você precisa estar ciente ao inicializar um host no cluster.

É possível agrupar Keycloak sem IP Multicast, mas este tópico está além do escopo deste guia. Consulte o capítulo [JGroups](#) da *Documentação WildFly 23*.

Arquitetura de rede recomendada

A arquitetura de rede recomendada para a implantação do Keycloak é configurar um balanceador de carga HTTP/HTTPS em um endereço IP público que encaminha solicitações para servidores Keycloak sentados em uma rede privada. Isso isola todas as conexões de cluster e fornece um bom meio de proteger os servidores.

Por padrão, não há nada que impeça os nós não autorizados de se juntar ao cluster e transferir dados. Os nós de cluster devem estar em uma rede privada, com um firewall protegendo-os de ataques externos.

Exemplo de agrupamento

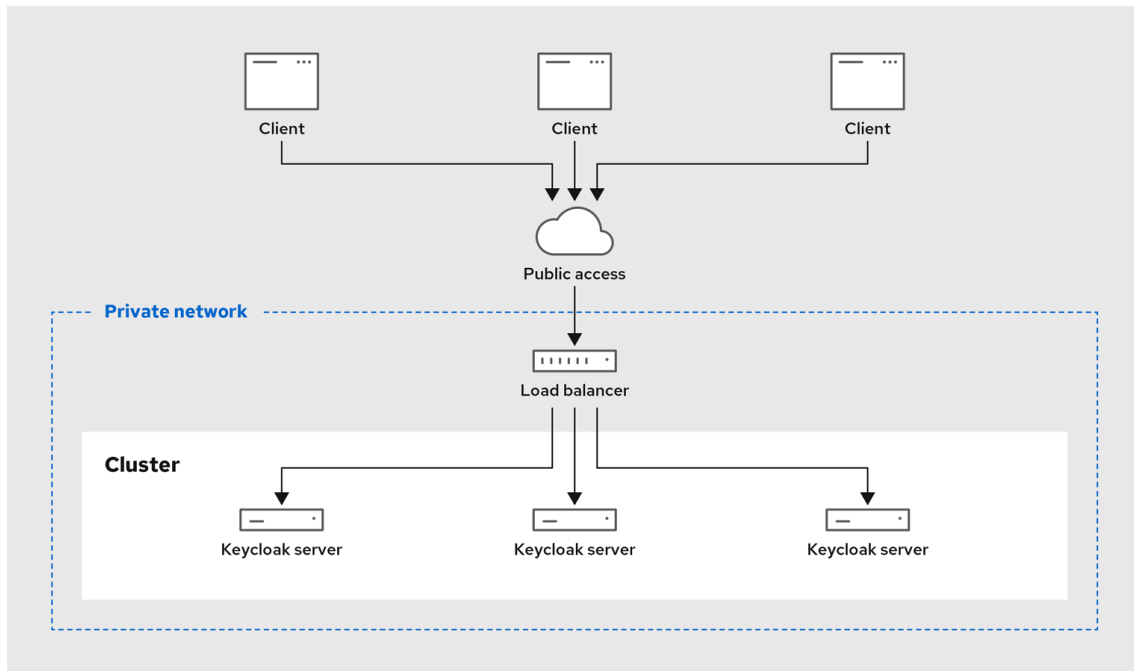
O Keycloak vem com uma demonstração de clustering fora da caixa que aproveita o modo de domínio. Revise o capítulo [Exemplo de Domínio Agrupado](#) para obter mais detalhes.

Configuração de um balanceador de carga ou proxy

Esta seção discute uma série de coisas que você precisa configurar antes de colocar um proxy reverso ou um balanceador de carga na frente da implantação do Keycloak agrupado. Ele também abrange a configuração do balanceador de carga embutido que era [o Exemplo de Domínio Agrupado](#).

O diagrama a seguir ilustra o uso de um balanceador de carga. Neste exemplo, o balanceador de carga serve como um proxy reverso entre três clientes e um cluster de três servidores Keycloak.

Diagrama do balanceador de carga de exemplo



70_RHSSO_0320

Identificação de endereços IP do cliente

Alguns recursos no Keycloak dependem do fato de que o endereço remoto do cliente HTTP conectado ao servidor de autenticação é o endereço IP real da máquina cliente. Exemplos incluem:

- Registros de eventos - uma tentativa de login falhada seria registrada com o endereço IP de origem errado
- SSL necessário - se o SSL necessário for definido como externo (o padrão) deve exigir SSL para todas as solicitações externas
- Fluxos de autenticação - um fluxo de autenticação personalizado que usa o endereço IP para, por exemplo, mostrar OTP apenas para solicitações externas
- Registro dinâmico do cliente

Isso pode ser problemático quando você tem um proxy reverso ou um balanceador de carga na frente do servidor de autenticação Keycloak. A configuração usual é que você tem um proxy frontend sentado em uma rede pública que carrega saldos e encaminha solicitações para apoiar as instâncias do servidor Keycloak localizadas em uma rede privada. Há alguma configuração extra que você tem que fazer neste cenário para que o endereço IP do cliente real seja encaminhado e processado pelas instâncias do servidor Keycloak. especificamente:

- Configure o proxy reverso ou o carrensebalador para definir corretamente os cabeçalhos X-Forwarded-For e X-Forwarded-Proto HTTP.
- Configure seu proxy reverso ou um carreçoieiro para preservar o cabeçalho HTTP 'Host' original.
- Configure o servidor de autenticação para ler o endereço IP do cliente a partir do cabeçalho X-Forwarded-For.

Configurar seu proxy para gerar os cabeçalhos X-Forwarded-For e X-Forwarded-Proto HTTP e preservar o cabeçalho HTTP original está além do escopo deste guia. Tome precauções extras para garantir que o cabeçalho X-Forwarded-For seja definido pelo seu proxy. Se o seu proxy não estiver configurado corretamente, então os clientes *desonestos* podem definir esse cabeçalho eles mesmos e enganar o Keycloak para pensar que o cliente está se conectando a partir de um endereço IP diferente do que realmente é. Isso se torna muito importante se você estiver fazendo qualquer listagem em preto ou branco de endereços IP.

Além do proxy em si, há algumas coisas que você precisa configurar no lado Keycloak das coisas. Se o seu proxy estiver encaminhando solicitações através do protocolo HTTP, então você precisa configurar o Keycloak para retirar o endereço IP do cliente do cabeçalho X-Forwarded-For em vez do pacote de rede. Para isso, abra o arquivo de configuração de perfil(*autônomo.xml*, *autônomo-ha.xml* ou *domínio.xml* dependendo do seu modo de [operação](#)) e procure o bloco XML de urna:jboss:domínio:undertow:12.0 XML.

X-Forwarded-For HTTP Config

```
<subssistema xmlns="urna:jboss:domínio:undertow:12.0">
  nome decache <buffer="padrão"/>
  nome <servidor => "servidor padrão"
  <ajp-ouvinte nome="ajp" soquete-vinculação="ajp"/>
  <http-ouvinte nome="padrão" soquete-vinculação="http" redirect-socket="https"
    encaminhamento de endereço por proxy="verdadeiro"/>
  ...
</servidor>
  ...
</subssistema>
```

Adicione o atributo de encaminhamento de endereço proxy ao elemento http-listener. Defina o valor como `verdadeiro`.

Se o seu proxy estiver usando o protocolo AJP em vez de HTTP para encaminhar solicitações (ou seja, Apache HTTPD + mod-cluster), então você terá que configurar as coisas um pouco diferente. Em vez de modificar o http-ouvinte, você precisa adicionar um filtro para retirar essas informações dos pacotes AJP.

X-Forwarded-For AJP Config

```
<subssistema xmlns="urna:jboss:domínio:undertow:12.0">
  nome decache <buffer="padrão"/>
```

```

nome <servidor => "servidor padrão"
<ajp-ouvinte nome="ajp" soquete-vinculação="ajp"/>
<http-ouvinte nome="padrão" soquete-vinculação="http" redirect-socket="https"/>
<o nome dehost= alias"default-host"="localhost">
    ...
    <filter-ref nome="proxy-peer"/>
</host>
</servidor>
...
<filtros>
    ...
    <filter nome="proxy-peer"
        nome de classe="io.undertow.server.handlers.ProxyPeerAddressHandler"
módulo="io.undertow.core" />
    </filtros>
</subsistema>

```

Habilite HTTPS/SSL com um proxy reverso

Supondo que seu proxy reverso não use a porta 8443 para SSL, você também precisa configurar para qual o tráfego https da porta é redirecionado.

```

<subsistema xmlns="urna:jboss:domínio:undertow:12.0">
    ...
<http-ouvinte nome="padrão" soquete-vinculação="http"
proxy-address-forwarding="true" redirect-socket="proxy-https"/>
    ...
</subsistema>

```

Adicione o atributo de redirecionamento ao elemento `http-listener`. O valor deve ser `proxy-https` que aponta para uma vinculação de soquete que você também precisa definir.

Em seguida, adicione um novo elemento de ligação de soquete ao elemento de grupo de ligação de soquete:

```

<socket-vinculato nomedo grupo="standard-sockets" default-interface="público"
port-offset="{jboss.socket.binding.port-offset:0}">
    ...
<socket-vinculação nome=porta "proxy-https"="443"/>
    ...
</grupo de ligação de soquete>

```

Verificar configuração

Você pode verificar a configuração do proxy reverso ou do balanceador de carga abrindo o caminho `/auth/realms/master/.bem conhecido/openid-configuração` através do proxy

reverso. Por exemplo, se o endereço proxy reverso estiver `https://acme.com/`, então abra a URL `https://acme.com/auth/realms/master/.well-known/openid-configuration`. Isso mostrará um documento JSON listando uma série de pontos finais para Keycloak. Certifique-se de que os pontos finais comecem com o endereço (esquema, domínio e porta) do seu proxy reverso ou balanceador de carga. Ao fazer isso, certifique-se de que o Keycloak está usando o ponto final correto.

Você também deve verificar se o Keycloak vê o endereço IP de origem correto para solicitações. Para verificar isso, você pode tentar fazer login no console administrativo com um nome de usuário e/ou senha inválidos. Isso deve mostrar um aviso no registro do servidor algo assim:

```
08:14:21.287 WARN XNIO-1 tarefa-45 [org.keycloak.events] type=LOGIN_ERROR,
realmId=master, clientId=security-admin-console, userId=8f20d7ba-4974-4811-a695-
242c8fbd1bf8, ipAddress=X.X.X.X, error=invalid_user_credentials,
auth_method=openid-connect, auth_type=code,
redirect_uri=http://localhost:8080/auth/admin/master/console/?redirect_fragment=
%2Frealms%2Fmaster%2Fevents-settings, code_id=a3d48b67-a439-4546-b992-
e93311d6493e, username=admin
```

Verifique se o valor do `ipAddress` é o endereço IP da máquina com a quem você tentou fazer login e não o endereço IP do proxy reverso ou do balanceador de carga.

Usando o balanceador de carga embutido

Esta seção abrange a configuração do balanceador de carga embutido que é discutido no [Exemplo de Domínio Agrupado](#).

O [exemplo de domínio agrupado](#) só foi projetado para ser executado em uma máquina. Para trazer um escravo em outro hospedeiro, você vai precisar

1. Edite o arquivo `.xml domínio` para apontar para o seu novo escravo hospedeiro
2. Copie a distribuição do servidor. Você não precisa dos arquivos `.xml domínio.xml`, `host.xml` ou `host-master`. Nem você precisa do *diretório autônomo*.
3. Edite o arquivo `host-slave.xml` para alterar os endereços de ligação usados ou substituí-los na linha de comando

Registre um novo host com balanceador de carga

Vamos primeiro olhar para o registro do novo escravo host com a configuração do balanceador de carga no `domínio.xml`. Abra este arquivo e vá para a configuração de subtow no perfil de balanceador de carga. Adicione uma nova definição de host chamada `remote-host3` dentro do bloco XML de proxy reverso.

domínio.xml config de proxy reverso

```
<subssistema xmlns="urna:jboss:domínio:undertow:12.0">
...
<handlers>
  <reverso-proxy nome="lb-handler">
<omerc="host1" outbound-socket-binding=esquema "remote-host1" = caminho"ajp"
="/" instance-id="myroute1"/>
<omerc="host2" outbound-socket-binding=esquema "remote-host2" =
caminho"ajp" ="/" instance-id="myroute2"/>
<omerc="remote-host3" outbound-socket-binding =esquema "remote-host3" =
caminho"ajp" ="/" instance-id="myroute3"/>
  </proxy reverso>
</manipuladores>
...
</subssistema>
```

A vinculação do soquete de saída é um nome lógico apontando para uma vinculação de soquete configurada posteriormente no arquivo *domínio.xml*. O atributo de id de instância também deve ser exclusivo do novo host, pois esse valor é usado por um cookie para permitir sessões pegajosas ao equilibrar a carga.

Em seguida, vá para o grupo de ligação de soquete de balanceador de carga e adicione a ligação de tomada de saída para host3 remoto. Esta nova ligação precisa apontar para o host e porta do novo host.

.xml de ligação de tomada de saída

```
<socket-vinculato nomedo grupo="load-balancer-sockets" interface padrão="público">
...
< nome de ligação de soquetedsaída => "remote-host1"
<reterde destino de<remote = porta"localhost" ="8159"/>
  </out-socket-binding>
< nome de ligação de soquete de saída="remote-host2">
<reterde destino de<remote = porta"localhost" ="8259"/>
  </out-socket-binding>
< nome de ligação de soquetedsaída => "remote-host3"
<remote-destino host=porta "192.168,0,5" ="8259"/>
  </out-socket-binding>
</grupo de ligação de soquete>
```

Endereços de ligação mestre

A próxima coisa que você terá que fazer é mudar o público e os endereços de vinculação de gestão para o anfitrião mestre. Editar o arquivo *domínio.xml*, conforme discutido no capítulo Bind [Addresses](#) ou especificar esses endereços de vinculação na linha de comando da seguinte forma:

```
$ domain.sh --host-config=host-master.xml -Djboss.bind.address=192.168.0.2 -  
Djboss.bind.address.management=192.168.0.2
```

Endereços de ligação de escravos de host

Em seguida, você terá que alterar os endereços de vinculação público, gerenciais controlador de domínio(jboss.domain.master-address). Edite o arquivo *host-slave.xml* ou especifique-os na linha de comando da seguinte forma:

```
$ domain.sh --host-config=host-slave.xml  
-Djboss.bind.address=192.168.0.5  
-Djboss.bind.address.management=192.168.0.5  
-Djboss.domain.master.address=192.168.0.2
```

Os valores de `jboss.bind.address` e `jboss.bind.address.management` pertencem ao endereço IP do escravo hospedeiro. O valor de `jboss.domain.master.address` precisa ser o endereço IP do controlador de domínio, que é o endereço de gerenciamento do host mestre.

Configuração de outros balanceadores de carga

Consulte a seção [de balanceamento de carga](#) na *Documentação WildFly 23* para obter informações sobre como usar outros balanceadores de carga baseados em software.

Sessões pegajosas

A implantação típica de cluster consiste no balanceador de carga (proxy reverso) e 2 ou mais servidores Keycloak na rede privada. Para fins de desempenho, pode ser útil se o balanceador de carga encaminhar todas as solicitações relacionadas à sessão específica do navegador para o mesmo nó keycloak backend.

A razão é que o Keycloak está usando cache distribuído Infinispan sob as capas para salvar dados relacionados à sessão de autenticação atual e à sessão do usuário. Os caches distribuídos infinispan são configurados com um proprietário por padrão. Isso significa que a sessão em particular é salva apenas em um nó de cluster e os outros nós precisam procurar a sessão remotamente se eles quiserem acessá-la.

Por exemplo, se a sessão de autenticação com ID 123 for salva no cache Infinispan no nó1e, em seguida, o node2 precisar procurar esta sessão, ele precisa enviar a solicitação para o nó1 na rede para retornar a entidade de sessão específica.

É benéfico se uma entidade de sessão específica estiver sempre disponível localmente, o que pode ser feito com a ajuda de sessões pegajosas. O fluxo de trabalho no ambiente de cluster com o balanceador de carga frontend público e dois nós keycloak backend pode ser assim:

- Usuário envia solicitação inicial para ver a tela de login keycloak
- Esta solicitação é atendida pelo balanceador de carga frontend, que o encaminha para algum nó aleatório (por exemplo, node1). Dito estritamente, o nó não precisa ser aleatório, mas pode ser escolhido de acordo com alguns outros critérios (endereço IP do cliente etc). Tudo depende da implementação e configuração do balanceador de carga subjacente (proxy reverso).
- O Keycloak cria uma sessão de autenticação com ID aleatório (por exemplo, 123) e salva-o no cache Infinispan.
- O cache distribuído infinispn atribui o proprietário principal da sessão com base no hash do ID da sessão. Consulte [a documentação da Infinispan](#) para obter mais detalhes sobre isso. Vamos supor que a Infinispan designou o nó2 para ser o proprietário desta sessão.
- Keycloak cria o cookie AUTH_SESSION_ID com o formato como <session-id>.<propador-node-> . No nosso exemplo, será 123.node2 .
- A resposta é devolvida ao usuário com a tela de login Keycloak e o cookie AUTH_SESSION_ID no navegador

A partir deste ponto, é benéfico se o balanceador de carga encaminhar todas as próximas solicitações para o nó2, pois este é o nó, que é proprietário da sessão de autenticação com ID 123 e, portanto, a Infinispan pode procurar esta sessão localmente. Após o término da autenticação, a sessão de autenticação é convertida em sessão de usuário, que também será salva no nó2 porque tem o mesmo ID 123 .

A sessão pegajosa não é obrigatória para a configuração do cluster, porém é boa para o desempenho pelas razões mencionadas acima. Você precisa configurar seu loadbalancer para pegar o AUTH_SESSION_ID cookie. Como exatamente fazer isso depende do seu segurança.

Recomenda-se no lado Keycloak usar a propriedade do sistema jboss.node.name durante a inicialização, com o valor correspondente ao nome da sua rota. Por exemplo, -Djboss.node.name=node1 usará o nó1 para identificar a rota. Esta rota será usada por caches Infinispan e será anexada ao cookie AUTH_SESSION_ID quando o nó for o proprietário da chave específica. Aqui está um exemplo do comando start-up usando esta propriedade do sistema:

```
cd $RHSSO_NODE1
./standalone.sh -c autônomo-ha.xml -Djboss.socket.binding.port-offset=100 -
Djboss.node.name=node1
```

Normalmente, no ambiente de produção, o nome da rota deve usar o mesmo nome do seu host backend, mas não é necessário. Você pode usar um nome de rota diferente. Por exemplo, se você quiser ocultar o nome host do servidor Keycloak dentro de sua rede privada.

Desativar a adição da rota

Alguns balanceadores de carga podem ser configurados para adicionar as informações de rota por si mesmos, em vez de depender do nó Keycloak back-end. No entanto, como descrito acima, a adição da rota pelo Keycloak é recomendada. Isso porque quando feito dessa forma o desempenho melhora, já que keycloak está ciente da entidade que é dona de sessão específica e pode encaminhar para esse nó, que não é necessariamente o nó local.

Você tem permissão para desativar a adição de informações de rota ao cookie AUTH_SESSION_ID pelo Keycloak, se preferir, adicionando o seguinte em seu arquivo RHSSO_HOME/autônomo/configuração/autônomo.xml na configuração do subsistema Keycloak:

```
<subsistema xmlns="urna:jboss:domínio:keycloak-server:1.1">
...
nome <spi => "stickySessionEncoder"
<provider nome="infinispan" ativado="verdadeiro">
  <propriedades>
<o nome de propriedade= valor "shouldAttachRoute" =valor "falso"/>
  </propriedades>
</provedor>
</spi>
</subsistema>
```

Configuração de rede multicast

O suporte ao clustering fora da caixa precisa de IP Multicast. Multicast é um protocolo de transmissão de rede. Este protocolo é usado na hora da inicialização para descobrir e se juntar ao cluster. Ele também é usado para transmitir mensagens para a replicação e invalidação de caches distribuídos usados pelo Keycloak.

O subsistema de clustering do Keycloak é executado na pilha JGroups. Fora da caixa, os endereços de vinculação para clustering estão vinculados a uma interface de rede privada com 127.0.0.1 como endereço IP padrão. Você tem que editar você é o .xml ou.xml *domínio autônomos* discutidos no capítulo [Bind Address](#).

rede privada config

```
<interfaces>
...
<interface nome => "privado"
<não - valor do endereço= "${jboss.bind.address.private:127.0.0.1}"/>
  </interface>
</interfaces>
```

```

<socket-vinculação nome do grupo="standard-sockets" default-interface="público"
port-offset="${jboss.socket.binding.port-offset:0}">
...
<socket-vinculação = interface "jgroups-mping" = porta "privada" = "0" multicast-
endereço="${jboss.default.multicast.address:230.0.0.4}" multicast-port="45700"/>
<sem nome de vinculação de bolso= interface "jgroups-tcp" = porta "privada" = "7600"/>
<sem nome de vinculação de bolso= interface "jgroups-tcp-fd" = porta "privada"
="57600"/>
<socket-vinculação = interface "jgroups-udp" = porta "privada" = "55200" multicast-
endereço="${jboss.default.multicast.address:230.0.0.4}" multicast-port="45688"/>
<socket-vinculação = interface "jgroups-udp-fd" = porta "privada" = "54200"/>
<socket-vinculação nome=porta "modcluster" = "0" multicast-endereço="224.0.1.105"
multicast-port="23364"/>
...
</grupo de ligação de soquete>

```

As coisas que você deseja configurar são o `jboss.bind.address.private` e `jboss.default.multicast.address`, bem como as portas dos serviços na pilha de clusters.

É possível agrupar Keycloak sem IP Multicast, mas este tópico está além do escopo deste guia. Consulte [JGroups](#) na *Documentação WildFly 23*.

Protegendo a comunicação de cluster

Quando os nós de cluster são isolados em uma rede privada, é necessário ter acesso à rede privada para poder se juntar a um cluster ou visualizar a comunicação no cluster. Além disso, você também pode habilitar autenticação e criptografia para comunicação em cluster. Enquanto sua rede privada estiver segura, não é necessário habilitar autenticação e criptografia. Keycloak não envia informações muito confidenciais sobre o cluster em ambos os casos.

Se você quiser habilitar autenticação e criptografia para comunicação de cluster, consulte o 'Guia de Alta Disponibilidade' na [documentação WildFly](#).

Startup de cluster serializada

Os nós de cluster keycloak são permitidos para inicializar simultaneamente. Quando a instância do servidor Keycloak é inicializado, ele pode fazer alguma migração de banco de dados, importação ou inicializações pela primeira vez. Um bloqueio DB é usado para evitar que as ações iniciais entrem em conflito entre si quando os nós de cluster inicializam simultaneamente.

Por padrão, o tempo limite máximo para esta trava é de 900 segundos. Se um nó estiver esperando nesta trava por mais do que o tempo limite, ele não será inicializado. Normalmente, você não precisará aumentar/diminuir o valor padrão, mas apenas no caso de ser possível configurá-lo em arquivo `autônomo.xml`, `autônomo-ha.xml` ou `domínio.xml` em sua distribuição. A localização deste arquivo depende do seu [modo de operação](#).

```
<spi nome="dblock">
<provo nome="jpa" ativado="verdadeiro">
  <propriedades>
< nome da propriedade= valor "lockWaitTimeout" = "900"/>
  </propriedades>
</provedor>
</spi>
```

Inicializando o Cluster

Inicializar keycloak em um cluster depende do seu [modo de operação](#)

Modo autônomo

```
$ bin/autônomo.sh --server-config=standalone-ha.xml
```

Modo de domínio

```
$ bin/domínio.sh --host-config=host-master.xml
```

```
$ bin/domínio.sh --host-config=host-slave.xml
```

Você pode precisar usar parâmetros adicionais ou propriedades do sistema. Por exemplo, o parâmetro `-b` para o host de ligação ou a propriedade do sistema `jboss.node.name` para especificar o nome da rota, conforme descrito na seção [Sticky Sessions](#).

Solucionando problemas

- Observe que quando você executar um cluster, você deve ver uma mensagem semelhante a esta no registro de ambos os nós de cluster:
- INFO [org.infinispan.remoting.transport.jgroups.JGroupsTransport] (Incoming-10,shared=udp) ISPN000094: Recebeu nova visualização de cluster: [node1/keycloak|1] (2) [node1/keycloak, node2/keycloak]

Se você vir apenas um nó mencionado, é possível que seus anfitriões não estejam unidos.

Normalmente é melhor ter seus nós de cluster em rede privada sem firewall para comunicação entre eles. O firewall pode ser ativado apenas no ponto de

acesso público à sua rede. Se por algum motivo você ainda precisar ter firewall ativado em nós de cluster, você precisará abrir algumas portas. Os valores padrão são a porta UDP 55200 e a porta multicast 45688 com endereço multicast 230.0.0.4. Observe que você pode precisar de mais portas abertas se quiser ativar recursos adicionais, como diagnósticos para a pilha JGroups. Keycloak delega a maior parte do trabalho de agrupamento para Infinispan/JGroups. Para obter mais informações, consulte [JGroups](#) na *Documentação WildFly 23*.

- Se você estiver interessado em suporte ao failover (alta disponibilidade), despejos, expiração e ajuste de cache, consulte [Configuração de cache do servidor](#).

Configuração de cache do servidor

Keycloak tem dois tipos de caches. Um tipo de cache fica na frente do banco de dados para diminuir a carga no DB e diminuir os tempos de resposta globais mantendo os dados na memória. Os metadados de reino, cliente, função e usuário são mantidos neste tipo de cache. Este cache é um cache local. Os caches locais não usam a replicação mesmo se você estiver no cluster com mais servidores Keycloak. Em vez disso, eles só mantêm cópias localmente e se a entrada for atualizada uma mensagem de invalidação é enviada para o resto do cluster e a entrada é despejada. Há um **trabalhador de cache replicado separado**, que tarefa é enviar as mensagens de invalidação para todo o cluster sobre quais entradas devem ser despejadas de caches locais. Isso reduz muito o tráfego de rede, torna as coisas eficientes e evita transmitir metadados sensíveis sobre o fio.

O segundo tipo de cache lida com o gerenciamento de sessões de usuário, tokens off-line e o controle de falhas de login para que o servidor possa detectar phishing de senha e outros ataques. Os dados mantidos nesses caches são temporários, apenas na memória, mas possivelmente são replicados em todo o cluster.

Este capítulo discute algumas opções de configuração para esses caches para implantações agrupadas e não agrupadas.

Configuração mais avançada desses caches pode ser encontrada na seção [Infinispan](#) da Do

Despejo e Expiração

Existem vários caches diferentes configurados para Keycloak. Existe um cache de reino que contém informações sobre aplicativos protegidos, dados gerais de segurança e opções de configuração. Há também um cache de usuário que contém metadados do usuário. Ambos os caches padrão para um máximo de 10000 entradas e usam uma estratégia de despejo menos usada recentemente. Cada um deles também está

vinculado a um cache de revisões de objetos que controla o despejo em uma configuração agrupada. Este cache é criado implicitamente e tem o dobro do tamanho configurado. O mesmo se aplica ao cache de autorização, que detém os dados de autorização. O cache de chaves contém dados sobre chaves externas e não precisa ter cache de revisões dedicadas. Em vez disso, tem expiração explicitamente declarada nele, de modo que as chaves são periodicamente expiradas e forçadas a ser baixadas periodicamente de clientes externos ou provedores de identidade.

A política de despejo e as entradas máximas desses caches podem ser configuradas no *autônomo.xml*, *autônomo-ha.xml* ou *domínio.xml* dependendo do seu modo de operação. No arquivo de configuração, há a parte com subsistema infinispán, que se parece com isso:

```
<subssistema xmlns="urna:jboss:domínio:infinispan:12.0">
  <cache-container nome="keycloak">
    <nome de cache local="reinos">
      <objeto-memória tamanho="10000"/>
    </cache local>
    <nome de cache local="usuários">
      <objeto-memória tamanho="10000"/>
    </cache local>
    ...
    <nome de cache local="chaves">
      <objeto-tamanho de memória="1000"/>
      <expiração max-ociosa="3600000"/>
    </cache local>
    ...
  </contêiner de cache>
```

Para limitar ou expandir o número de entradas permitidas basta adicionar ou editar o elemento objeto ou o elemento de expiração da configuração de cache específica.

Além disso, há também sessões separadas de caches, sessões de clientes, sessões offline, sessões offlineClientSess, loginFailures e actionTokens. Esses caches são distribuídos em ambiente de cluster e não são vinculados em tamanho por padrão. Se eles forem limitados, então seria possível que algumas sessões sejam perdidas. As sessões vencidas são liberadas internamente pelo próprio Keycloak para evitar o crescimento do tamanho desses caches sem limite. Se você ver problemas de memória devido a um grande número de sessões, você pode tentar:

- Aumente o tamanho do cluster (mais nós em cluster significa que as sessões são espalhadas mais igualmente entre nós)
- Aumente a memória do processo do servidor Keycloak
- Diminua o número de proprietários para garantir que os caches sejam salvos em um único lugar. Consulte [Replication e Failover](#) para obter mais detalhes

- Desabilitar a vida útil L1 para caches distribuídos. Consulte a documentação da Infinispan para obter mais detalhes
- Diminua os tempos de sessão, o que poderia ser feito individualmente para cada reino no console administrativo Keycloak. Mas isso pode afetar a usabilidade dos usuários finais. Consulte [timeouts](#) para obter mais detalhes.

Há um cache adicional replicado, `trabalho`, que é usado principalmente para enviar mensagens entre nós de cluster; ele também é desvinculado por padrão. No entanto, este cache não deve causar quaisquer problemas de memória, pois as entradas neste cache são de curta duração.

Replicação e Failover

Existem caches como `sessões`, `autenticações`, `Sessões`, `sessõesoffline`, `loginFailures` e alguns outros (Ver `Despejo` e [Expiração](#) para obter mais detalhes), que são configurados como caches distribuídos ao usar uma configuração em cluster. As entradas não são replicadas em cada nó, mas em vez disso, um ou mais nós são escolhidos como proprietários desses dados. Se um nó não for o proprietário de uma entrada específica de cache, ele consulta o cluster para obtê-lo. O que isso significa para failover é que se todos os nós que possuem um pedaço de dados caírem, esses dados são perdidos para sempre. Por padrão, o Keycloak especifica apenas um proprietário para dados. Então, se esse nó cair, os dados são perdidos. Isso geralmente significa que os usuários serão conectados e terão que fazer login novamente.

Você pode alterar o número de nós que replicam um pedaço de dados alterando o atributo dos proprietários na declaração de cache distribuído.

Proprietários

```
<subsystem xmlns="urn:jboss:domain:infinispan:12.0">
  <cache-container nome="keycloak">
    <distribuido nome="sessões" proprietários="2"/>
  ...
```

Aqui nós o mudamos para que pelo menos dois nós repliquem uma sessão específica de login do usuário.

O número de proprietários recomendados depende muito da sua implantação. Se você não estiver conectado quando um nó cai, então um proprietário é bom o suficiente e você evitará a re...

É geralmente sábio configurar seu ambiente para usar o loadbalancer com sessões pegajosas no servidor Keycloak, onde a solicitação específica é atendida, será geralmente o proprietário. Portanto, será capaz de procurar os dados localmente. Consulte [sessões sticky](#) para obter...

Cache incapacitante

Para desativar o reino ou o cache do usuário, você deve editar o arquivo `autônomo.xml`, `autônomo-ha.xml` ou `domínio.xml` em sua distribuição. A localização deste arquivo depende do seu modo de [operação](#). Aqui está como o config se parece inicialmente.

```
<spi nome="userCache">
<provo nome="padrão" ativado="verdadeiro"/>
</spi>

nome <spi="realmCache">
<provo nome="padrão" ativado="verdadeiro"/>
</spi>
```

Para desativar o cache, defina o atributo `ativado` como falso para o cache que deseja desativar. Você deve reiniciar seu servidor para que essa alteração entre em vigor.

Limpando caches no Runtime

Para limpar o reino ou o cache do usuário, vá para a página Realm Settings do console administrador Keycloak→Cache Config. Nesta página você pode limpar o cache do reino, o cache do usuário ou o cache das chaves públicas externas.

O cache será limpo para todos os reinos!

Operador keycloak

O Operador Keycloak é **Visualização de Tecnologia** e não é totalmente suportado.

O Operador Keycloak automatiza a administração Keycloak em Kubernetes ou Openshift. Você usa este Operador para criar recursos personalizados (CRs), que automatizam tarefas administrativas. Por exemplo, em vez de criar um cliente ou um usuário no console administrador Keycloak, você pode criar recursos personalizados para executar essas tarefas. Um recurso personalizado é um arquivo YAML que define os parâmetros para a tarefa administrativa.

Você pode criar recursos personalizados para executar as seguintes tarefas:

- [Instale o Keycloak](#)
- [Criar reinos](#)

- [Criar clientes](#)
- [Criar usuários](#)
- [Conecte-se a um banco de dados externo](#)
- [Agende backups no banco de dados](#)
- [Instalar extensões e temas](#)

Depois de criar recursos personalizados para reinos, clientes e usuários, você pode gerenciá-los no console administrativo do Keycloak ou como recursos personalizados usando o comando `kubectl`. No entanto, você não pode usar o Operator para executar uma sincronização de uma maneira para recursos personalizados que não são gerenciados pelo Operator. Se você modificar um recurso personalizado do reino, as alterações aparecerão no console administrativo, mas se você modificar o reino usando o console administrativo, essas alterações não terão efeito sobre o recurso personalizado.

Comece a usar o Operator [instalando o Operator Keycloak em um cluster](#).

Instalando o Operator Keycloak em um cluster

Para instalar o Operator Keycloak, você pode usar:

- [O Gerente do Ciclo de Vida do Operator \(OLM\)](#)
- [Instalação da linha de comando](#)

Instalação usando o Gerenciador de Ciclo de Vida do Operator

Você pode instalar o Operator em um cluster [OpenShift](#) ou [Kubernetes](#).

Instalação em um cluster OpenShift

Pré-requisitos

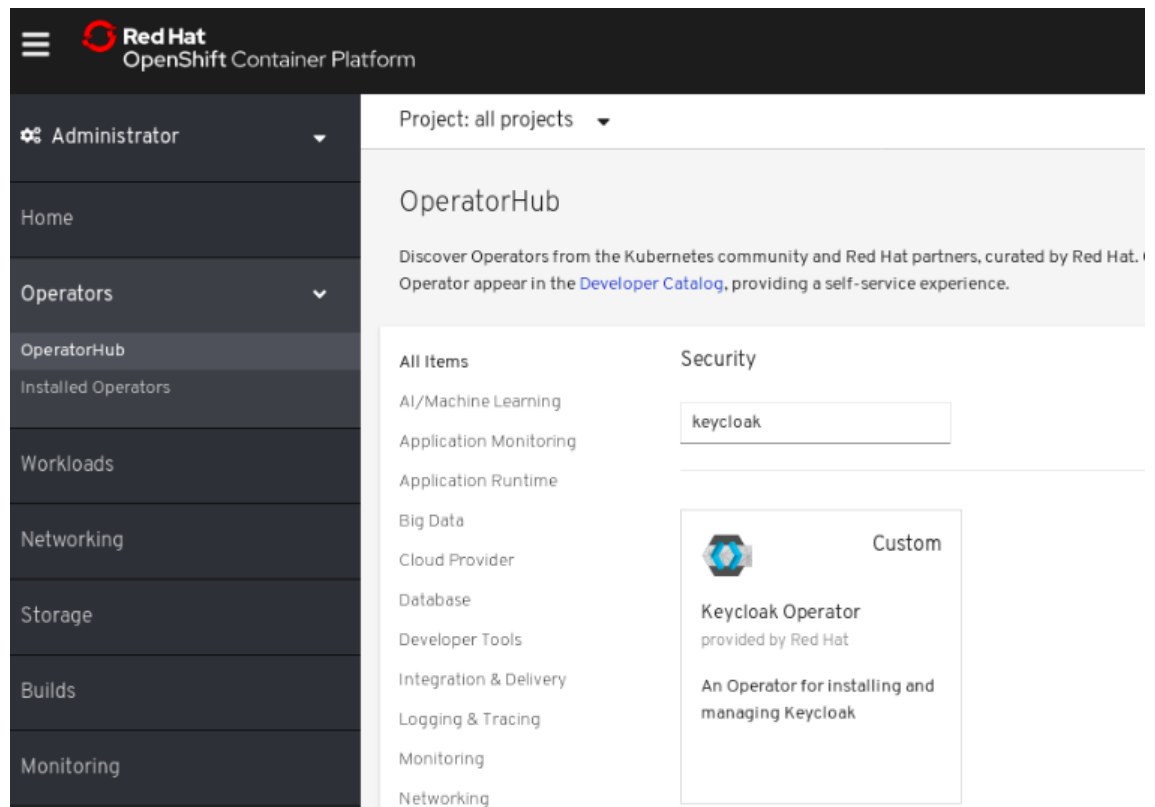
- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.

procedimento

Realize este procedimento em um cluster OpenShift 4.4.

1. Abra o console web OpenShift Container Platform.
2. Na coluna à esquerda, clique em **Operadores, OperatorHub**.
3. Procure por Keycloak Operator.

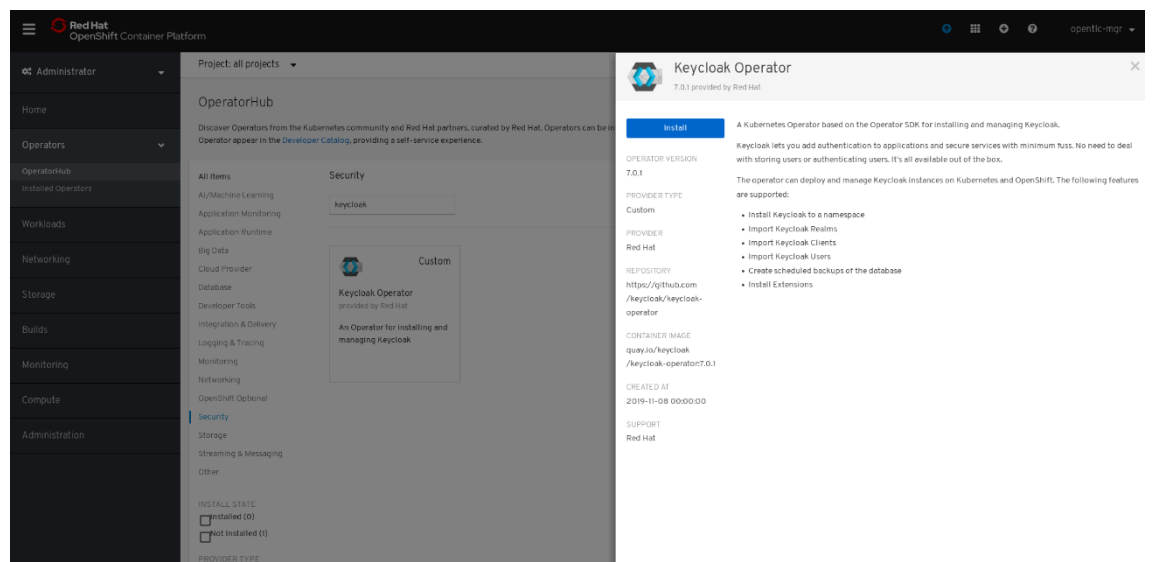
Guia OperatorHub no OpenShift



4. Clique no ícone Do operador de keycloak.

Uma página instalar é aberta.

Operadora Instalar página no OpenShift



5. Clique em Instalar.
6. Selecione um namespace e clique em Assinar.

Seleção namespace no OpenShift

Installation Mode *

- ☐ All namespaces on the cluster (default)
This mode is not supported by this Operator
- ☒ A specific namespace on the cluster
Operator will be available in a single namespace only.

Installed Namespace *

PR keycloak ▼

Update Channel *

- ☒ alpha

Approval Strategy *

- ☒ Automatic
- ☐ Manual

Subscribe

Cancel

O Operador começa a instalar.

Recursos adicionais

- Quando a instalação do Operador for concluída, você está pronto para criar seu primeiro recurso personalizado. Consulte [a instalação do Keycloak usando um recurso personalizado](#). No entanto, se você quiser começar a acompanhar todas as atividades do Operador antes de criar recursos personalizados, consulte o [Operador de Monitoramento de Aplicativos](#).
- Para obter mais informações sobre operadores OpenShift, consulte o [guia OpenShift Operators](#).

Instalação em um cluster Kubernetes

Pré-requisitos

- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.

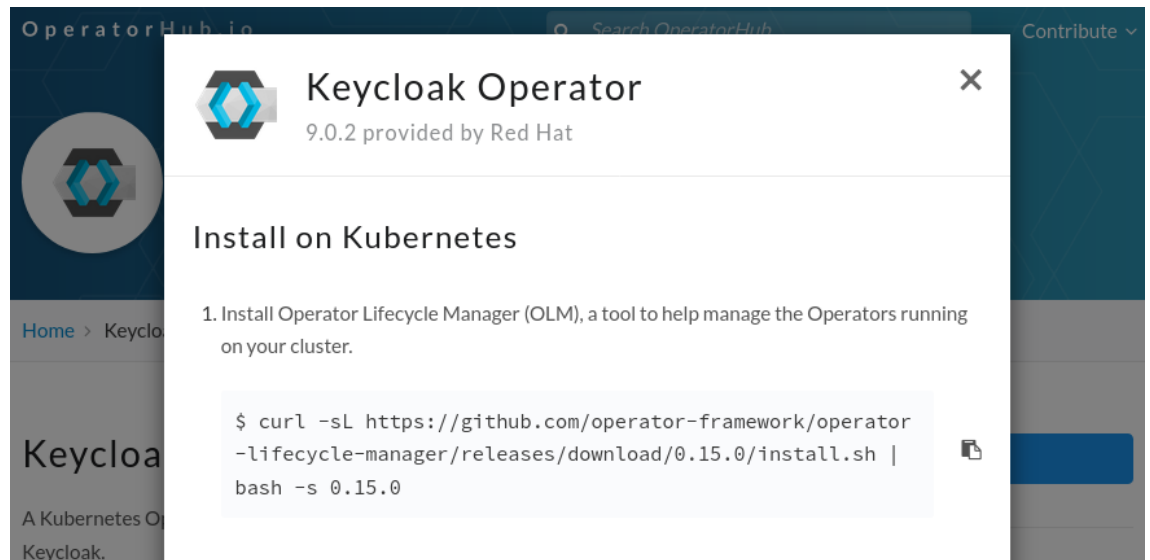
procedimento

Para um cluster Kubernetes, execute essas etapas.

1. Vá para [Keycloak Operator em OperatorHub.io](#).

2. Clique em **Instalar**.
3. Siga as instruções na tela.

Operadora Instalar página em Kubernetes



Recursos adicionais

- Quando a instalação do Operador for concluída, você está pronto para criar seu primeiro recurso personalizado. Consulte [a instalação do Keycloak usando um recurso personalizado](#). No entanto, se você quiser começar a acompanhar todas as atividades do Operador antes de criar recursos personalizados, consulte o [Operador de Monitoramento de Aplicativos](#).
- Para obter mais informações sobre uma instalação do Kubernetes, consulte [Como instalar um operador a partir de OperatorHub.io](#).

Instalação da linha de comando

Você pode instalar o Operador Keycloak a partir da linha de comando.

Pré-requisitos

- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.

procedimento

1. Obtenha o software para instalar a partir deste local: [Github repo](#).
2. Instale todas as definições de recursos personalizadas necessárias:

```
$ kubectl aplicar -f implantação/crds/
```

3. Crie um novo namespace (ou reaproveitar um já existente) como o `myprojectnamespace`:

```
$ kubectl aplicar namespace myproject
```

4. Implantar uma função, vinculação de função e conta de serviço para o Operador:

```
5. $ kubectl aplicar -f implantar/role.yaml -n myproject
6. $ kubectl aplicar -f implantação/role_binding.yaml -n myproject
   $ kubectl aplicar -f implantação/service_account.yaml -n myproject
```

7. Implantar o Operador:

```
$ kubectl aplicar -f implantação/operador.yaml -n myproject
```

8. Confirme se o Operador está funcionando:

```
9. $ kubectl obter implantação keycloak-operator
10.      NOME PRONTO PARA A DATA DISPONÍVEL
    keycloak-operator 1/1 1 41s
```

Recursos adicionais

- Quando a instalação do Operador for concluída, você está pronto para criar seu primeiro recurso personalizado. Consulte [a instalação do Keycloak usando um recurso personalizado](#). No entanto, se você quiser começar a acompanhar todas as atividades do Operador antes de criar recursos personalizados, consulte o [Operador de Monitoramento de Aplicativos](#).
- Para obter mais informações sobre uma instalação do Kubernetes, consulte [Como instalar um operador a partir de OperatorHub.io](#).
- Para obter mais informações sobre operadores OpenShift, consulte o [guia OpenShift Operators](#).

O Operador de Monitoramento de Aplicativos

Antes de usar o Operador para instalar o Keycloak ou criar componentes, recomendamos que você instale o Operador de Monitoramento de Aplicativos, que rastreia a atividade do Operador. Para visualizar métricas para o Operador, você pode usar os alertas grafana dashboard e prometheus do Operador de Monitoramento de Aplicativos. Por exemplo, você pode visualizar métricas como o número de loops de reconciliação de tempo de execução do controlador, o tempo de loop de reconciliação e erros.

A integração do Operador Keycloak com o Operador de Monitoramento de Aplicativos não requer nenhuma ação. Você só precisa instalar o Operador de Monitoramento de Aplicativos no cluster.

Instalação do Operador de Monitoramento de Aplicativos

Pré-requisitos

- O Operador Keycloak está instalado.

- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.

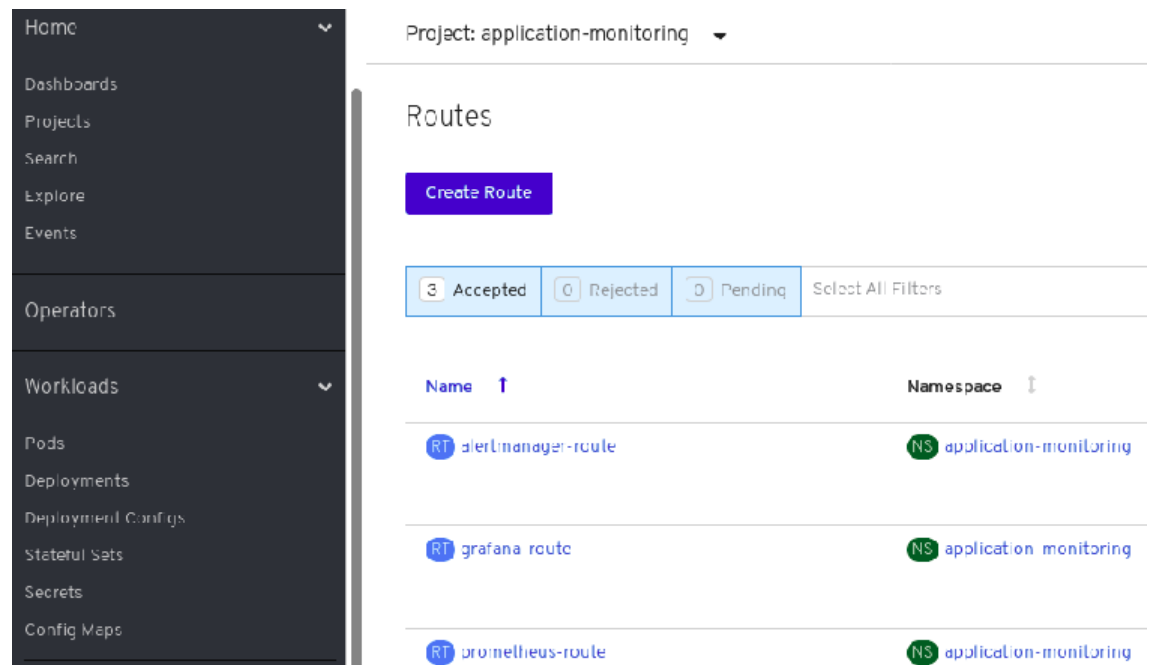
procedimento

1. Instale o Operador de Monitoramento de Aplicativos utilizando a [documentação](#).
2. Anote o namespace usado para a instalação do Keycloak Operator. por exemplo:

```
kubectl namespace de rótulo <namespace>monitoring-key=middleware
```

3. Faça login no console web OpenShift.
4. Confirme que o monitoramento está funcionando procurando a rota Prometheus e Grafana no namespace de monitoramento de aplicativos.

Rotas no console web OpenShift

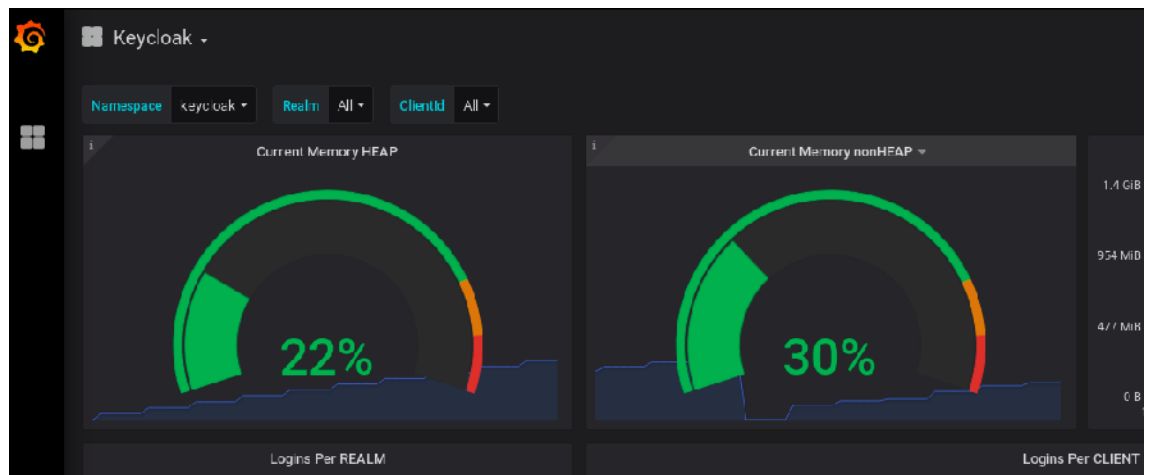


Visualização de métricas do operador

Grafana e Prometheus fornecem informações gráficas sobre as atividades do Operador.

- O Operador instala um painel grafana pré-definido, como mostrado aqui:

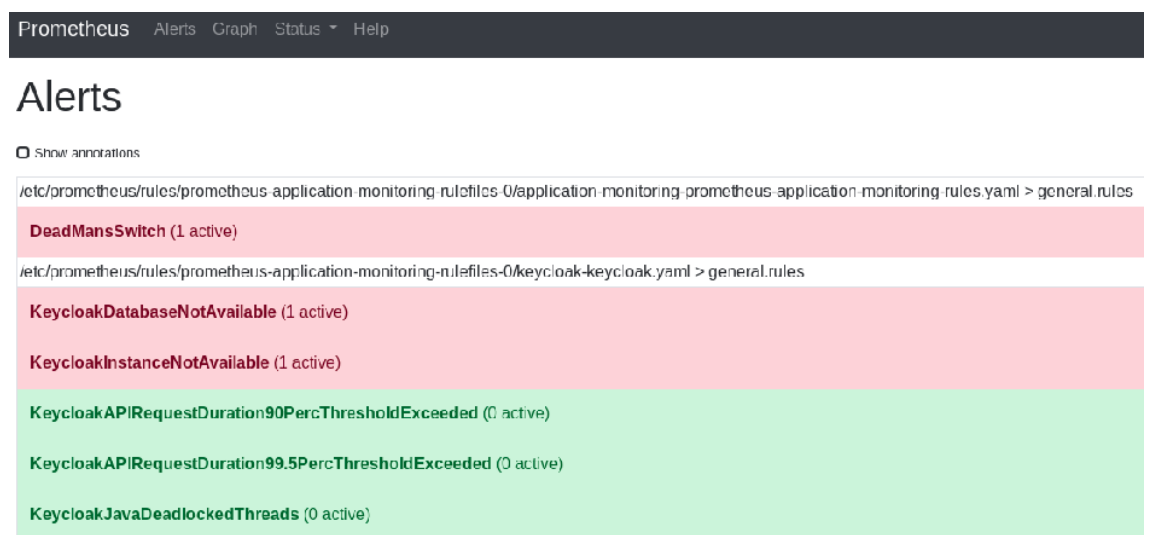
Painel grafana



Se você fizer personalizações, recomendamos que você clone o Painel Grafana durante uma atualização.

- O Operador instala um conjunto de alertas de prometeu pré-definidos, conforme mostrado aqui:

Alertas de Prometeu



Recursos adicionais

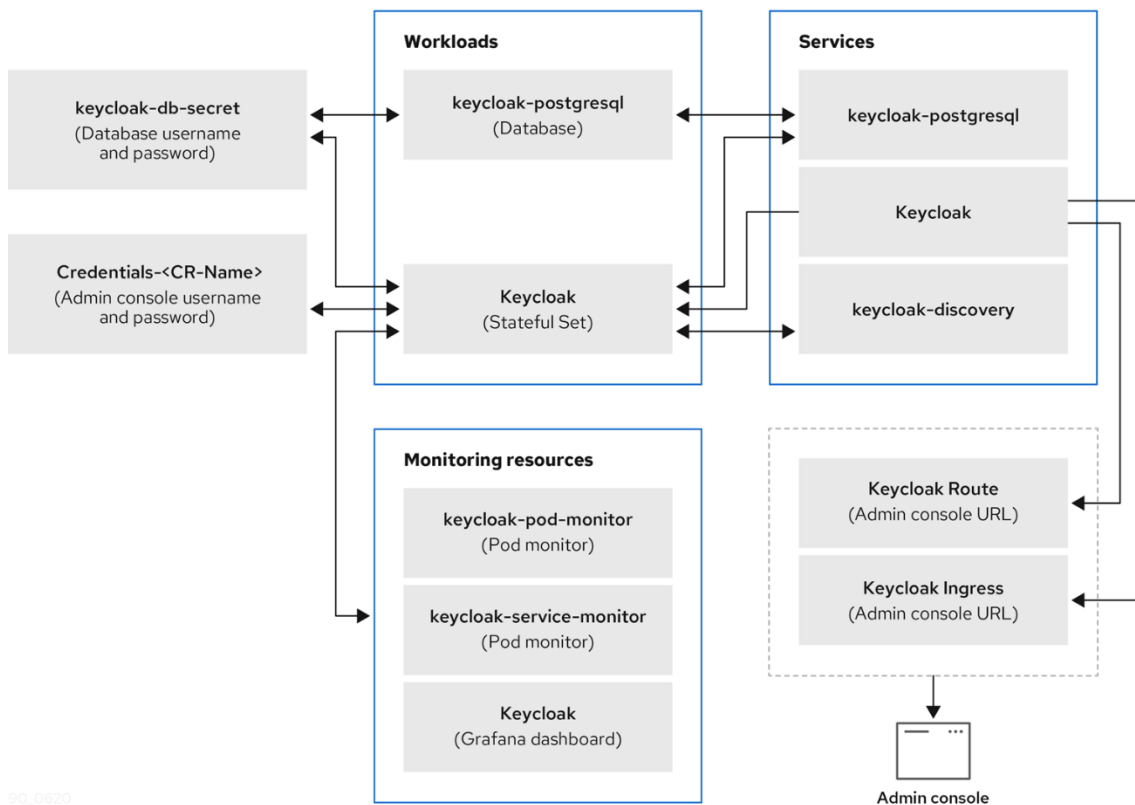
Para obter mais informações, consulte [Accessing Prometheus, Alertmanager e Grafana](#).

Instalação keycloak usando um recurso personalizado

Você pode usar o Operador para automatizar a instalação do Keycloak criando um recurso personalizado Keycloak. Quando você usa um recurso personalizado para instalar o Keycloak, você cria os componentes e serviços descritos aqui e ilustrados no gráfico que se segue.

- `keycloak-db-secret` - Armazena propriedades como o nome de usuário do banco de dados, senha e endereço externo (se você se conectar a um banco de dados externo)
- `credenciais-<CR-Name>` - Nome de usuário e senha do administrador para fazer login no console administrador Keycloak (o `<CR-Name>`is com base no nome do recurso personalizado Keycloak)
- `keycloak` - Especificação de implantação keycloak que é implementada como um StatefulSet com alto suporte de disponibilidade
- `keycloak-postgresql` - Inicia uma instalação de banco de dados PostgreSQL
- `Keycloak-discovery Service` - Realiza `JDBC_PING` descoberta
- `keycloak Service` - Conecta-se ao Keycloak através de HTTPS (HTTP não é suportado)
- `keycloak-postgresql Service` - Conecta uma instância de banco de dados interna e externa, se usada
- `keycloak Route` - A URL para acessar o console administrador Keycloak do OpenShift
- `keycloak Ingress` - A URL para acessar o console administrador Keycloak da Kubernetes

Como os componentes e serviços do Operador interagem



O recurso personalizado Keycloak

O recurso personalizado Keycloak é um arquivo YAML que define os parâmetros para instalação. Este arquivo contém três propriedades.

- `instâncias` - controla o número de instâncias em execução no modo de alta disponibilidade.
- `externalAccess` - se o `ativado` for `True`, o Operador cria uma rota para OpenShift ou uma Ingress para Kubernetes para o cluster Keycloak.
- `externalDatabase` - aplica-se somente se você quiser conectar um banco de dados hospedado externamente. Esse tópico é abordado na [seção de banco de dados externo](#) deste guia.

Exemplo arquivo YAML para um recurso personalizado Keycloak

```
apiVersion: keycloak.org/v1alpha1
tipo: Keycloak
metadados:
nome: exemplo-keycloak
rótulos:
app: exemplo-keycloak
especificação:
instâncias: 1
externalAccess:
habilitado: True
```

Você pode atualizar o arquivo YAML e as alterações aparecem no console administrativo K. O console administrativo não atualiza o recurso personalizado.

Criando um recurso personalizado keycloak no OpenShift

No OpenShift, você usa o recurso personalizado para criar uma rota, que é a URL do console administrativo, e encontrar o segredo, que contém o nome de usuário e senha para o console administrativo.

Pré-requisitos

- Você tem um arquivo YAML para este recurso personalizado.
- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.
- Se você quiser começar a acompanhar todas as atividades do Operador agora, instale o aplicativo de monitoramento antes de criar esse recurso personalizado. Consulte [o Operador de Monitoramento de Aplicativos](#).

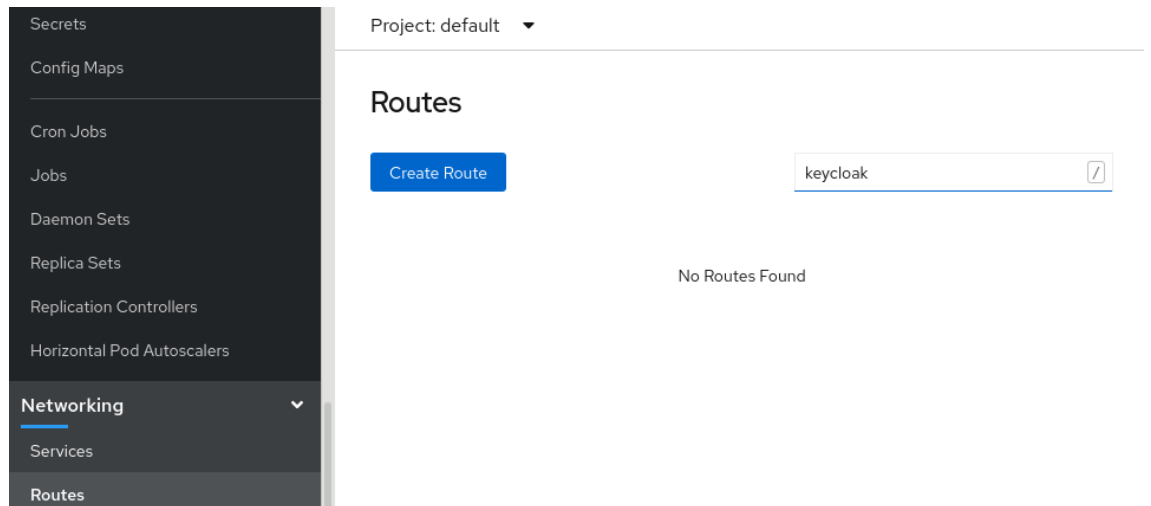
procedimento

1. Crie uma rota usando seu arquivo YAML: `kubectl apply -f <filename>.yaml -n <namespace>`. Por exemplo:
2. `$ kubectl apply -f keycloak.yaml -n keycloak`
`keycloak.keycloak.org/example-keycloak criado`

Uma rota é criada no OpenShift.

3. Faça login no console web OpenShift.
4. Selecione Networking, Rotas e pesquise por Keycloak.

Tela de rotas no console web OpenShift



5. Na tela com a rota Keycloak, clique na URL em Localização.

A tela de login do console administrador Keycloak é exibida.

Tela de login do console administrador

Username or email

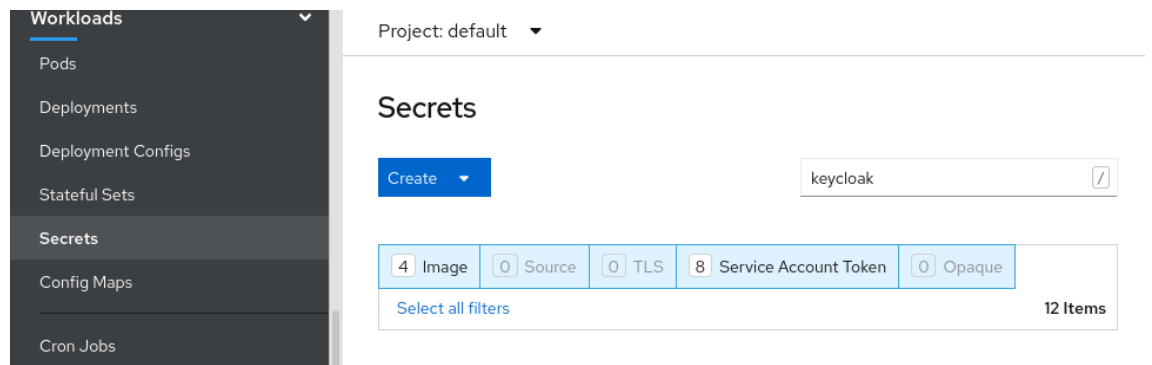
Password

☐ Remember me

Log In

6. Localize o nome de usuário e a senha para o console administrativo no console web OpenShift; em Workloads, clique em Segredos e procure keycloak.

Tela de segredos no console web OpenShift



7. Digite o nome de usuário e a senha na tela de login do console administrativo.

Tela de login do console administrador

Username or email

admin

Password

••••••••

☐ Remember me

Log In

Agora você está conectado a uma instância do Keycloak que foi instalada por um recurso personalizado keycloak. Você está pronto para criar recursos personalizados para reinos, clientes e usuários.

Reino mestre keycloak

8. Verifique o status do recurso personalizado:

```
$ kubectl describe keycloak<CR-nome>
```

Criando um recurso personalizado Keycloak em Kubernetes

No Kubernetes, você usa o recurso personalizado para criar uma entrada, que é o endereço IP do console administrativo, e encontrar o segredo, que contém o nome de usuário e senha para esse console.

Pré-requisitos

- Você tem um arquivo YAML para este recurso personalizado.
- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.

procedimento

1. Crie a entrada usando seu arquivo YAML. `kubectl aplicar -f <filename>.yaml -n <namespace>`. Por exemplo:
2. `$ kubectl aplicar -f keycloak.yaml -n keycloak`
`keycloak.keycloak.org/example-keycloak criado`
3. Encontre a entrada: `kubectl obter entrada -n <cr-nome>`. Por exemplo:
4. `$ kubectl obter entrada -n exemplo-keycloak`
5. NOME HOSPEDA ENDEREÇOS DA IDADE DOS PORTOS
`keycloak keycloak.redhat.com 192.0.2.0 80 3m`
6. Copie e cole o ENDEREÇO (a entrada) em um navegador da Web.

A tela de login do console administrador Keycloak é exibida.

Tela de login do console administrador

Username or email

Password

☐ Remember me

Log In

7. Localize o nome de usuário e a senha.

```
$ kubectl obter credenciais secretas-<CR-Name> -o go-template='{{range $k,$v:= .data}}{{printf "%s: " $k}}{{se não $v}}{{{$$v}}{{{$v | base64decode}}}/end}}{{"\n"}}/end}}'
```

8. Digite o nome de usuário e a senha na tela de login do console administrativo.

Tela de login do console administrador

Username or email

admin

Password

••••••••

☐ Remember me

Log In

Agora você está conectado a uma instância do Keycloak que foi instalada por um recurso personalizado keycloak. Você está pronto para criar recursos personalizados para reinos, clientes e usuários.

Reino mestre do console administrativo

Resultados

Após o Operador processa o recurso personalizado, visualize o status com este comando:

```
$ kubectl descrever keycloak<CR-nome>
```

Status do recurso personalizado keycloak

```
Nome: exemplo-keycloak
Namespace: keycloak
Rótulos: app=exemplo-keycloak
Anotações :<none>
Versão API: keycloak.org/v1alpha1
Tipo: Keycloak
Especificação:
  Acesso externo:
Habilitado: verdadeiro
Instâncias: 1
Status:
Segredo de Credencial: credencial-exemplo-keycloak
URL interno: https://<A URLExternal para a instância implantada>
Mensagem:
Fase: reconciliação
Pronto: verdadeiro
Recursos secundários:
  Implantação:
    keycloak-postgresql
Reivindicação de volume persistente:
  keycloak-postgresql-reivindicação
Regra dePrometeu:
  keycloak
```

Rota:
keycloak
Segredo:
credencial-exemplo-keycloak
keycloak-db-secret
Serviço:
keycloak-postgresql
keycloak
keycloak-discovery
Monitor de Serviços :
keycloak
Conjunto Stateful:
keycloak
Versão:
Eventos:

Recursos adicionais

- Uma vez concluída a instalação do Keycloak, você está pronto para [criar um recurso personalizado](#).
- Se você tiver um banco de dados externo, você pode modificar o recurso personalizado Keycloak para apoiá-lo. Consulte [Conectando-se a um banco de dados externo](#).

Criando um recurso personalizado de reino

Você pode usar o Operador para criar reinos em Keycloak conforme definido por um recurso personalizado. Você define as propriedades do recurso personalizado do reino em um arquivo YAML.

Você pode atualizar o arquivo YAML e as alterações aparecem no console administrativo K. O console administrativo não atualiza o recurso personalizado.

Exemplo arquivo YAML para um recurso personalizado Dom

```
apiVersion: keycloak.org/v1alpha1
tipo: KeycloakRealm
metadados:
nome: teste
rótulos:
app: exemplo-keycloak
especificação:
  reino:
id: "básico"
reino: "básico"
habilitado: True
displayName: "Reino Básico"
```

```
instanceSelector:
matchLabels:
app: exemplo-keycloak
```

Pré-requisitos

- Você tem um arquivo YAML para este recurso personalizado.
- No arquivo YAML, o `aplicativo` em `caso Selector` corresponde à etiqueta de um recurso personalizado Keycloak. A correspondência desses valores garante que você crie o reino na instância certa do Keycloak.
- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.

procedimento

1. Use este comando no arquivo YAML que você criou: `kubectl aplicar -f <rem-name>.yaml`. Por exemplo:
2. `$ kubectl aplicar -f initial_realm.yaml`
`keycloak.keycloak.org/test` criado
3. Faça login no console de administração para a instância relacionada do Keycloak.
4. Clique em Selecionar Reino e localize o reino que você criou.

O novo reino se abre.

Reino mestre do console administrativo

The screenshot shows the Keycloak Admin Console interface. On the left is a dark sidebar with a menu. The 'Test' realm is selected at the top. Under the 'Configure' section, 'Realm Settings' is highlighted. Other options include Clients, Client Scopes, Roles, Identity, Providers, User Federation, and Authentication. The main panel on the right is titled 'Test' and has tabs for General, Login, Keys, Email, Themes, Cache, Tokens, and Client Registration. The 'General' tab is active, showing configuration fields for the realm 'test'. The fields include: Name (test), Display name, HTML Display name, Frontend URL, Enabled (toggle set to ON), and User-Managed Access (toggle set to OFF).

Resultados

Após o Operador processa o recurso personalizado, visualize o status com este comando:

```
$ kubectl descrever keycloak<CR-nome>
```

Realm status de recurso personalizado

Nome: `exemplo-keycloakrealm`

Namespace: keycloak
Rótulos: app=exemplo-keycloak
Anotações :<none>
Versão API: keycloak.org/v1alpha1
Tipo: KeycloakRealm
Metadados:
Criação Timestamp: 2019-12-03T09:46:02Z
Finalizadores:
 reino.limpeza
Geração: 1
Versão de recursos : 804596
Self Link: /apis/keycloak.org/v1alpha1/namespaces/keycloak/keycloakrealm/example-keycloakrealm
UID: b7b2f883-15b1-11ea-91e6-02cb885627a6
Especificação:
 Seletor de instâncias :
 Rótulos de correspondência:
App: exemplo-keycloak
Reino:
Nome da exibição : Reino básico
Habilitado: verdadeiro
Id: básico
Reino: básico
Status:
 URL de login:
Mensagem:
Fase: reconciliação
Pronto: verdadeiro
Eventos :<none>

Recursos adicionais

- Quando a criação do reino for concluída, você está pronto para [criar um recurso personalizado do cliente](#).

Criando um recurso personalizado do cliente

Você pode usar o Operador para criar clientes no Keycloak conforme definido por um recurso personalizado. Você define as propriedades do reino em um arquivo YAML.

Você pode atualizar o arquivo YAML e as alterações aparecem no console administrativo. O console administrativo não atualiza o recurso personalizado.

Exemplo arquivo YAML para um recurso personalizado do cliente

apiVersion: keycloak.org/v1alpha1

```

tipo: KeycloakClient
metadados:
nome: exemplo-cliente
rótulos:
app: app=exemplo-keycloak
especificação:
  realistaSeletor:
matchLabels:
app: <setting labels para keycloakRealm recurso personalizado>
cliente:
  # gerado automaticamente se não for fornecido
  #id: 123
clienteld: segredo do cliente
segredo: segredo do cliente
# ...
# outras propriedades do Cliente Keycloak

```

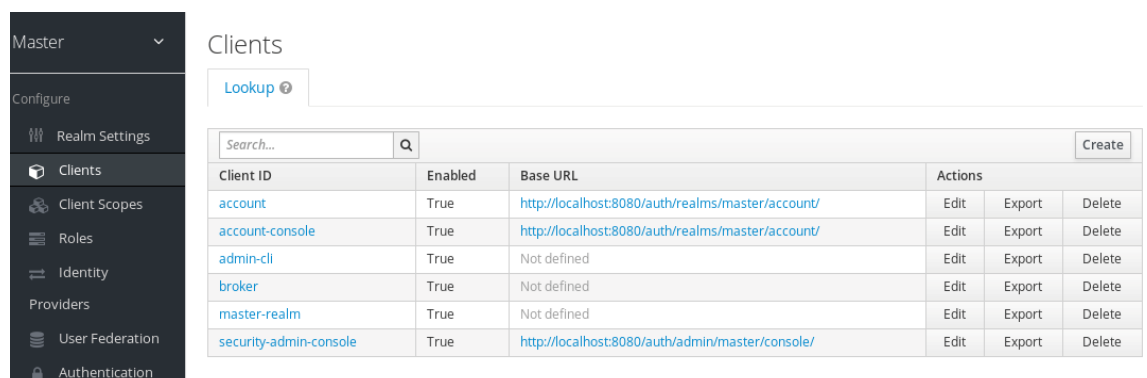
Pré-requisitos

- Você tem um arquivo YAML para este recurso personalizado.
- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.

procedimento

1. Use este comando no arquivo YAML que você criou: `kubectl aplicar -f <client-name>.yaml`. Por exemplo:
2. `$ kubectl aplicar -f initial_client.yaml`
keycloak.keycloak.org/example-client criado
3. Faça login no console de administração Keycloak para a instância relacionada do Keycloak.
4. Clique em Clientes.

O novo cliente aparece na lista de clientes.



Client ID	Enabled	Base URL	Actions
account	True	http://localhost:8080/auth/realms/master/account/	Edit Export Delete
account-console	True	http://localhost:8080/auth/realms/master/account/	Edit Export Delete
admin-cli	True	Not defined	Edit Export Delete
broker	True	Not defined	Edit Export Delete
master-realm	True	Not defined	Edit Export Delete
security-admin-console	True	http://localhost:8080/auth/admin/master/console/	Edit Export Delete

Resultados

Depois que um cliente é criado, o Operador cria um Segredo contendo o ID do Cliente e o segredo do cliente usando o seguinte padrão de nomeação: keycloak-cliente-secret-<nome de recurso>. Por exemplo:

Segredo do Cliente

```
apiVersion: v1
dados:
  CLIENT_ID: <base64 codificado ID do cliente>
  CLIENT_SECRET : <base64 codificado Client Secret>
tipo: Segredo
```

Após o Operador processa o recurso personalizado, visualize o status com este comando:

```
$ kubectl describe keycloak<CR-nome>
```

Status do recurso personalizado do cliente

```
Nome: segredo do cliente
Namespace: keycloak
Rótulos: app=exemplo-keycloak
Versão API: keycloak.org/v1alpha1
Tipo: KeycloakClient
Especificação:
Cliente:
Tipo de autenticador de clientes : segredo do cliente
ID do cliente: segredo do cliente
Id: keycloak-client-secret
  Seletor de Reinos :
    Rótulos de correspondência:
Aplicativo: keycloak
Status:
Mensagem:
Fase: reconciliação
Pronto: verdadeiro
  Recursos secundários:
Segredo:
  keycloak-cliente-segredo-cliente-segredo
Eventos :<none>
```

Recursos adicionais

- Quando a criação do cliente for concluída, você está pronto para [criar um recurso personalizado do usuário](#).

Criando um recurso personalizado do usuário

Você pode usar o Operador para criar usuários no Keycloak conforme definido por um recurso personalizado. Você define as propriedades do recurso personalizado do usuário em um arquivo YAML.

Você pode atualizar propriedades, exceto a senha, no arquivo YAML e as alterações aparecem no entanto as alterações no console administrativo não atualizam o recurso personalizado.

Exemplo arquivo YAML para um recurso personalizado do usuário

```
apiVersion: keycloak.org/v1alpha1
tipo: KeycloakUser
metadados:
nome: exemplo-usuário
especificação:
  usuário:
nome de usuário: "realm_user"
primeiro Nome: "John"
sobrenome: "Doe"
e-mail: "user@example.com"
habilitado: True
emailVerified: Falso
realmRoles:
  - "offline_access"
clienteRoles:
  conta:
  - "gerenciar conta"
  gestão de reinos :
  - "usuários de gerenciamento"
realistaSeletor:
matchLabels:
app: exemplo-keycloak
```

Pré-requisitos

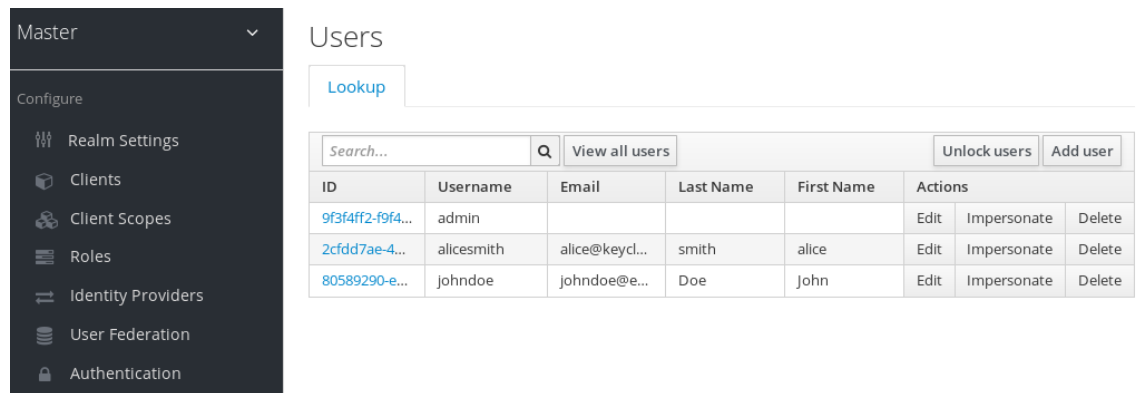
- Você tem um arquivo YAML para este recurso personalizado.
- O selecionador de reinos corresponde aos rótulos de um recurso personalizado existente.
- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.

procedimento

1. Use este comando no arquivo YAML que você criou: `kubectl aplicar -f <user_cr>.yaml`. Por exemplo:
2. `$ kubectl aplicar -f initial_user.yaml`
`keycloak.keycloak.org/example-user criado`
3. Faça login no console de administração para a instância relacionada do Keycloak.

4. Clique em Usuários.
5. Pesquise o usuário que você definiu no arquivo YAML.

O usuário está na lista.



ID	Username	Email	Last Name	First Name	Actions
9f3f4ff2-f9f4...	admin				Edit Impersonate Delete
2cfd7ae-4...	alicesmith	alice@keycl...	smith	alice	Edit Impersonate Delete
80589290-e...	johndoe	johndoe@e...	Doe	John	Edit Impersonate Delete

Resultados

Depois que um usuário é criado, o Operador cria um Segredo contendo o nome de usuário e a senha usando o seguinte padrão de nomeação: nome <realm de credencial><same>-<namespace>. Aqui está um exemplo:

KeycloakUser Secret

tipo: Segredo

apiVersion: v1

dados:

senha: <base64 senha codificada>

nome de usuário: <base64 nome de usuário codificado>

tipo: Opaco

Uma vez que o Operador processe o recurso personalizado, visualize o status com este comando:

```
$ kubectl describe keycloak<CR-nome>
```

Status do recurso personalizado do usuário

Nome: exemplo-reino-usuário

Namespace: keycloak

Rótulos: app=exemplo-keycloak

Versão API: keycloak.org/v1alpha1

Tipo: KeycloakUser

Especificação:

Seletor deReinos :

Rótulos de correspondência:

App: exemplo-keycloak

Usuário:

E-mail: realm_user@redhat.com

Credenciais:

Tipo: senha

Valor: <desuto de usuário>
E-mail verificado: falso
Habilitado: verdadeiro
Primeiro Nome: John
Sobrenome: Doe
Nome de usuário: realm_user
Status:
Mensagem:
Fase: reconciliado
Eventos :<none>

Recursos adicionais

- Se você tiver um banco de dados externo, você pode modificar o recurso personalizado Keycloak para apoiá-lo. Consulte [Conectando-se a um banco de dados externo](#).
- Para fazer backup do seu banco de dados usando recursos [personalizados](#), consulte backups de banco de dados de agendamento.

Conectando-se a um banco de dados externo

Você pode usar o Operador para se conectar a um banco de dados externo do PostgreSQL modificando o recurso personalizado Keycloak e criando um arquivo YAML secreto keycloak-db. Observe que os valores são codificados na Base64.

Exemplo arquivo YAML para keycloak-db-secret

```
apiVersion: v1
tipo: Segredo
metadados:
nome: keycloak-db-secret
namespace: keycloak
stringData:
POSTGRES_DATABASE : <Database Name>
POSTGRES_EXTERNAL_ADDRESS : <C ou URL do banco de dados externo (solucionável por K8s)>
POSTGRES_EXTERNAL_PORT : < Porta do Banco de Dados Externa >
  # Recomendou fortemente o uso <'Keycloak CR-Name'-postgresql>
POSTGRES_HOST : <De nome do serviço de base>
POSTGRES_PASSWORD : <Database Senha>
  # Necessário para a funcionalidade de backup AWS
POSTGRES_SUPERUSER: verdade
POSTGRES_USERNAME: <Database Username>
tipo: Opaco
```

As seguintes propriedades definem o nome do host ou endereço IP e a porta do banco de dados.

- `POSTGRES_EXTERNAL_ADDRESS` - um endereço IP ou um hostname do banco de dados externo. Este endereço precisa ser solucionável em um cluster Kubernetes.
- `POSTGRES_EXTERNAL_PORT` - (Opcional) Uma porta de banco de dados.

As outras propriedades funcionam da mesma forma para um banco de dados hospedado ou externo. Defina-os da seguinte forma:

- `POSTGRES_DATABASE` - Nome do banco de dados a ser utilizado.
- `POSTGRES_HOST` - O nome do Serviço usado para se comunicar com um banco de dados. Tipicamente `keycloak-postgresql`.
- `POSTGRES_USERNAME` - Nome de usuário do banco de dados
- `POSTGRES_PASSWORD` - Senha do banco de dados
- `POSTGRES_SUPERUSER` - Indica, se os backups devem ser executados como super usuário. Normalmente `verdade`.

O recurso personalizado Keycloak requer atualizações para habilitar o suporte externo ao banco de dados.

Exemplo de arquivo YAML para recurso personalizado Keycloak que suporta um banco de dados externo

```
apiVersion: keycloak.org/v1alpha1
tipo: Keycloak
metadados:
  rótulos:
    app: exemplo-keycloak
    nome: exemplo-keycloak
    namespace: keycloak
especificação:
  externalDatabase:
    habilitado: verdadeiro
  instâncias: 1
```

Pré-requisitos

- Você tem um arquivo YAML para `keycloak-db-secret`.
- Você modificou o recurso personalizado Keycloak para definir o `ExternalDatabase` como `true`.
- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.

procedimento

1. Localize o segredo para o seu banco de dados PostgreSQL: `kubectl obter segredo <secret_for_db>-o yaml`. Por exemplo:
2. `$ kubectl obter keycloak-db-secret -o yaml`

3. apiVersion: v1

4. dados

5. POSTGRES_DATABASE: cm9vdA==

6. POSTGRES_EXTERNAL_ADDRESS: MTcyLjE3LjAuMw==

POSTGRES_EXTERNAL_PORT: NTQzMg==

O POSTGRES_EXTERNAL_ADDRESS está no formato Base64.

7. Decodificar o valor para o segredo: `echo "<encoded_secret>" | base64 -decodificar`. Por exemplo:

```
8. $ echo "MTcyLjE3LjAuMw==" | base64 -decodificar
192.0.2.3
```

9. Confirme se o valor decodificado corresponde ao endereço IP do seu banco de dados:

```
10. $ kubectl obter pods -o de largura
11. NOME PRONTO STATUS REINICIA AGE IP
12. keycloak-0 1/1 Running 0 13m 192.0.2.0
    keycloak-postgresql-c8vv27m 1/1 Running 0 24m 192.0.2.3
```

13. Confirme se keycloak-postgresql aparece em uma lista de serviços em execução:

```
14. $ kubectl obter svc
15. IDADE DO TIPO DE NOME CLUSTER-IP EXTERNAL-IP PORT(S) AGE
16. keycloak ClusterIP 203.0.113.0 <none>8443/TCP 27m
17. keycloak-discovery ClusterIP None <none>8080/TCP 27m
    keycloak-postgresql ClusterIP 203.0.113.1 <none>5432/TCP 27m
```

O serviço keycloak-postgresql envia solicitações para um conjunto de endereços IP no backend. Esses endereços IP são chamados de pontos finais.

18. Veja os pontos finais usados pelo serviço keycloak-postgresql para confirmar que eles usam os endereços IP para o seu banco de dados:

```
19. $ kubectl obter pontos finais keycloak-postgresql
20. IDADE DOS PONTOS FINAIS DO NOME
    keycloak-postgresql 192.0.2.3.5432 27m
```

21. Confirme se keycloak está funcionando com o banco de dados externo. Este exemplo mostra que tudo está em execução:

```
22. $ kubectl obter pods
23. NOME PRONTO STATUS REINICIA AGE IP
24. keycloak-0 1/1 Running 0 26m 192.0.2.0
    keycloak-postgresql-c8vv27m 1/1 Running 0 36m 192.0.2.3
```

Recursos adicionais

- Para fazer backup do seu banco de dados usando recursos [personalizados](#), consulte [Agendamento de backups de banco de dados](#).

- Para obter mais informações sobre a codificação base64, consulte o [manual Segredos de Kubernetes](#).

Agendando backups de banco de dados

Você pode usar o Operador para agendar backups automáticos do banco de dados conforme definido por recursos personalizados. O recurso personalizado aciona um trabalho de backup (ou um CronJob no caso de Backups Periódicos) e reporta seu status.

Existem duas opções para agendar backups:

- [Backup para o armazenamento AWS S3](#)
- [Fazendo backup do armazenamento local](#)

Se você tiver armazenamento AWS S3, você pode executar um backup único ou backups periódicos. Se você não tiver armazenamento AWS S3, você pode fazer backup do armazenamento local.

Backup para o armazenamento AWS S3

Você pode fazer backup do seu banco de dados para o armazenamento AWS S3 uma vez ou periodicamente. Para fazer backup de seus dados periodicamente, insira um CronJob válido no cronograma.

Para armazenamento AWS S3, você cria um arquivo YAML para o recurso personalizado de backup e um arquivo YAML para o segredo AWS. O recurso personalizado de backup requer um arquivo YAML com a seguinte estrutura:

```
apiVersion: keycloak.org/v1alpha1
tipo: KeycloakBackup
metadados:
nome: <CR Nome>
especificação:
  aws:
    # Opcional - usado apenas para backups periódicos.
    # Segue a sintaxe crond usual (por exemplo, use "0 1 * * * *" para realizar o backup
    # todos os dias à 1:00.)
  horário: <Cron Job Schedule>
    Obrigatório - o nome do segredo contendo as credenciais para acessar o
    armazenamento S3
  credenciaisSecretName: <Um Segredo contendo credenciais S3>
```

O segredo AWS requer um arquivo YAML com a seguinte estrutura:

Segredo AWS S3

```
apiVersion: v1
tipo: Segredo
metadados:
nome: <Secret Name>
tipo: Opaco
stringData:
AWS_S3_BUCKET_NAME : <S3 Bucket Name>
AWS_ACCESS_KEY_ID : <AWS Access Key ID>
AWS_SECRET_ACCESS_KEY : chave secreta <AWS>
```

Pré-requisitos

- Seu arquivo YAML de recurso personalizado de backup inclui um Nome de credenciaisSecretName que faz referência a um segredo contendo credenciais AWS S3.
- Seu recurso personalizado KeycloakBackup tem sub-propriedades aws.
- Você tem um arquivo YAML para o AWS S3 Secret que inclui um nome <Secret> que corresponde ao identificado no recurso personalizado de backup.
- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.

procedimento

1. Crie o segredo com credenciais: `kubectl aplicar -f <secret_for_aws>.yaml`. Por exemplo:

```
2. $ kubectl aplicar -f secret.yaml
keycloak.keycloak.org/aws_s3_secret criado
```

3. Crie um trabalho de backup: `kubectl aplicar -f <backup_crname>.yaml`. Por exemplo:

```
4. $ kubectl aplicar -f aws_one-time-backup.yaml
keycloak.keycloak.org/aws_s3_backup criado
```

5. Veja uma lista de trabalhos de backup:

```
6. $ kubectl conseguir empregos
7. IDADE DE CONCLUSÃO DE NOME IDADE DE DURAÇÃO
aws_s3_backup 0/1 6s 6s
```

8. Veja a lista de trabalhos de backup executados.

```
9. $ kubectl obter pods
10. NOME PRONTO STATUS REINICIA A IDADE
11. aws_s3_backup-5b4rfdd 0/1 Concluído 0 24s
12. keycloak-0 1/1 Running 0 52m
keycloak-postgresql-c824c6-vv27m 1/1 Running 0 71m
```

13. Exibir o registro do seu trabalho de backup completo:

```
14. $kubectl toras aws_s3_backup-5b4rf
```

```
15.      ==> Despejo de dados de componentes concluído
16.      .
17.      .
18.      .
19.      .
[fonte,bash,subs=+atributos]
```

O status do trabalho de backup também aparece no console AWS.

Fazendo backup do armazenamento local

Você pode usar o Operator para criar um trabalho de backup que executa um backup único em um volume persistente local.

Exemplo arquivo YAML para um recurso personalizado de backup

```
apiVersion: keycloak.org/v1alpha1
tipo: KeycloakBackup
metadados:
nome: test-backup
```

Pré-requisitos

- Você tem um arquivo YAML para este recurso personalizado. Certifique-se de omitir as sub-propriedades `aws` deste arquivo.
- Você tem um `PersistentVolume` com um `ClaimRef` para reservá-lo apenas para um `PersistentVolumeClaim` criado pelo Operator Keycloak.

procedimento

1. Crie um trabalho de backup: `kubectl aplicar -f <backup_crname>`. Por exemplo:

```
2. $ kubectl aplicar -f uma vez-backup.yaml
keycloak.keycloak.org/test-backup
```

O Operador cria uma Reclamação `persistentevolume` com o seguinte esquema de nomeação: `Keycloak-backup-<CR-nome>`.

3. Veja uma lista de volumes:

```
4. $ kubectl obter pvc
5. VOLUME DE STATUS DO NOME
6. keycloak-backup-teste-backup Pvc-e242-ew022d5-093q-3134n-41-adff
keycloak-postresql-claim Bound pvc-e242-vs29202-9bcd7-093q-31-zadj
```

7. Veja uma lista de trabalhos de backup:

```
8. $ kubectl conseguir empregos
9. IDADE DE CONCLUSÃO DE NOME IDADE DE DURAÇÃO
test-backup 0/1 6s 6s
```

10. Veja a lista de trabalhos de backup executados:

```
11. $ kubectl obter pods
12. NOME PRONTO STATUS REINICIA A IDADE
13. teste-backup-5b4rf 0/1 Concluído 0 24s
14. keycloak-0 1/1 Running 0 52m
    keycloak-postgresql-c824c6-vv27m 1/1 Running 0 71m
```

15. Exibir o registro do seu trabalho de backup completo:

```
16. $ kubectl logs test-backup-5b4rf
17. ==> Despejo de dados de componentes concluído
18. .
19. .
20. .
    .
```

Recursos adicionais

- Para obter mais detalhes sobre volumes persistentes, consulte [Entendendo o armazenamento persistente](#).

Instalação de extensões e temas

Você pode usar o operador para instalar extensões e temas que você precisa para sua empresa ou organização. A extensão ou tema pode ser qualquer coisa que Keycloak possa consumir. Por exemplo, você pode adicionar uma extensão de métricas. Você adiciona a extensão ou o tema ao recurso personalizado Keycloak.

Exemplo arquivo YAML para um recurso personalizado Keycloak

```
apiVersion: keycloak.org/v1alpha1
tipo: Keycloak
metadados:
nome: exemplo-keycloak
rótulos:
aplicativo: keycloak
especificação:
instâncias: 1
extensões:
<url_for_extension_or_theme>
externalAccess:
habilitado: True
```

Você pode empacotar e implantar temas da mesma forma que qualquer outra extensão. Consulte a entrada manual [De Implantação de Temas](#) para obter mais informações.

Pré-requisitos

- Você tem um arquivo YAML para o recurso personalizado Keycloak.

- Você tem permissão de administrador de cluster ou um nível equivalente de permissões concedidas por um administrador.

procedimento

1. Edite o arquivo YAML para o recurso personalizado Keycloak: `edit kubectl edit<CR-name>`
2. Adicione uma linha chamada `extensões`: após a linha de `instâncias`.
3. Adicione uma URL a um arquivo JAR para sua extensão ou tema personalizado.
4. Salve o arquivo.

O Operador baixa a extensão ou o tema e instala-o.

Opções de comando para gerenciar recursos personalizados

Depois de criar uma solicitação personalizada, você pode editá-la ou excluí-la usando o comando `kubectl`.

- Para editar uma solicitação personalizada, use este comando: `edit kubectl<CR-nome>`
- Para excluir uma solicitação personalizada, use este comando: `kubectl delete<CR-name>`

Por exemplo, para editar uma solicitação personalizada de reino chamada `test-realm`, use este comando:

```
$ kubectl editar teste-reino
```

Uma janela se abre onde você pode fazer mudanças.

Você pode atualizar o arquivo YAML e as alterações aparecem no console administrativo K. O console administrativo não atualiza o recurso personalizado.

Estratégia de upgrade

Você pode configurar como o operador executa upgrades do Keycloak. Você pode escolher entre as seguintes estratégias de upgrade.

- **recriar**: Esta é a estratégia padrão. O operador remove todas as réplicas do Keycloak, cria opcionalmente um backup e cria as réplicas com base em uma imagem keycloak mais nova. Esta estratégia é adequada para grandes atualizações, pois uma única versão Keycloak está acessando o banco de dados subjacente. A desvantagem é que o Keycloak precisa ser desligado durante a atualização.
- **rolando**: O operador remove uma réplica de cada vez e a cria novamente com base em uma imagem keycloak mais nova. Isso garante uma atualização de tempo de inatividade zero, mas é mais adequado para atualizações de versão menor que não requerem migração de banco de dados, uma vez que o banco de dados é acessado por várias versões keycloak simultaneamente. Os backups automáticos não são suportados com essa estratégia.

Exemplo arquivo YAML para um recurso personalizado Keycloak

```
apiVersion: keycloak.org/v1alpha1
tipo: Keycloak
metadados:
  nome: exemplo-keycloak
rótulos:
  aplicativo: keycloak
especificação:
  instâncias: 2
  migração:
    estratégia: recriar
    backups:
      habilitado: True
  externalAccess:
    habilitado: True
```

Recursos adicionais

- Para obter mais informações sobre atualizações em andamento, consulte o [manual Update StatefulSets](#).

1. Rastreado como <https://issues.redhat.com/browse/KEYCLOAK-3873>