Imagem do Keycloak Docker

Imagem do Keycloak Server Docker.

Uso

Para inicializar no modo autônomo

```
docker executar jboss/keycloak
```

Expor no localhost

Para poder abrir Keycloak no localhost map port 8080 localmente

```
docker run -p 8080:8080 jboss/keycloak
```

Criando conta de administração

Por padrão, não há nenhum usuário administrativo criado para que você não possa fazer login no console administrativo. Para criar uma conta de administração, você precisa usar variáveis ambientais para passar em um nome de usuário inicial e senha. Isso é feito correndo:

```
docker executar -e KEYCLOAK_USER=<USERNAME> -e KEYCLOAK_PASSWORD=<PASSWORD>
jboss/keycloak
```

Você também pode criar uma conta em um contêiner já em execução:

```
<CONTAINER>/opt/jboss/keycloak/bin/add-user-keycloak.sh -u <USERNAME>-p<PASSWORD>
```

Em seguida, reiniciando o recipiente:

```
reiniciar docker<CONTAINER>
```

Fornecendo o nome de usuário e senha através de arquivos

Ao anexar `_FILE` às duas variáveis ambientais utilizadas acima (`KEYCLOAK_USER_FILE` e `KEYCLOAK_PASSWORD_FILE`), as informações podem serfornecidas através de arquivos em vez de valores variáveis de ambiente simples. A configuração e o suporte secreto no Docker Swarm combinam perfeitamente com este caso de uso.

Importando um reino

Para criar uma conta de administração e importar um reino previamente exportado:

```
docker executar -e KEYCLOAK_USER=<USERNAME> -e KEYCLOAK_PASSWORD=<PASSWORD> \
   -e KEYCLOAK_IMPORT=/tmp/example-realm.json -v /tmp/example-realm.json:/tmp/example-
realm.jboss/keycloak
```

Exportando um reino

Se você quiser exportar um reino que você criou/atualizou, em uma instância do Keycloak funcionando dentro de um contêiner docker. Você precisará garantir que o contêiner que executa o Keycloak tenha um volume mapeado. Por exemplo, você pode iniciar o Keycloak via docker com:

```
docker run -d -p 8180:8080 -e KEYCLOAK_USER=admin -e \
KEYCLOAK_PASSWORD=admin -v $(pwd):/tmp -nome kc \
jboss/keycloak
```

Você pode então obter a exportação a partir desta instância executando (aviso que usamos `-Djboss.socket.binding.port-offset=100` para que a exportação seja executada em uma porta diferente da própria Keycloak):

```
docker exec -it kc /opt/jboss/keycloak/bin/standalone.sh \
-Djboss.socket.binding.port-offset=100 -Dkeycloak.migration.action=export \
-Dkeycloak.migration.provider=singleFile \
-Dkeycloak.migration.realmName=my_realm \
-Dkeycloak.migration.usersExportStrategy=REALM_FILE \
-Dkeycloak.migration.file=/tmp/my_realm.json
```

base de dados

Esta imagem suporta usando h2, MySQL, PostgreSQL, MariaDB, Oracle ou Microsoft SQL Server como banco de dados.

Você pode especificar o fornecedor DB diretamente com a variável `DB_VENDOR` ambiente. Os valores suportados são:

- `h2` para o banco de dados H2 incorporado,
- `postgras` para o banco de dados Postgres,
- `mysql` para o banco de dados MySql.
- `mariadb` para o banco de dados MariaDB.
- `oracle` para o banco de dados Oracle.
- `mssql` para o banco de dados Microsoft SQL Server.

Se `DB_VENDOR` valor não for especificado, a imagem tentará detectar o fornecedor DB com base na seguinte lógica:

- É o nome de host padrão para o conjunto DB usando `hosts getent` (`postgres`, `mysql`, `mariadb`, `oracle`, `mssql`). Isso funciona se você estiver usando uma rede definida pelo usuário e os nomes padrão conforme especificado abaixo.
- Existe um conjunto de variável de ambiente `_ADDR` específico DB(`POSTGRES_ADDR`, `MYSQL_ADDR`, `MARIADB_ADDR`, `ORACLE_ADDR`). **Preterido**

Se o DB não puder ser detectado, ele será padrão para o banco de dados H2 incorporado.

Variáveis ambientais

Nomes de variáveis genéricas podem ser usados para configurar qualquer tipo de Banco de Dados, os padrões podem variar dependendo do Banco de Dados.

- DB_ADDR: Especifique o nome do host do banco de dados (opcional). Apenas para pós-lagres, você pode fornecer uma lista de nomes de host separados por círgula para host alternativo failover. O nome do host pode ser apenas o host ou par de host e porta, como exemplo host1,host2 ou host1:5421,host2:5436 ou host1,host2:5000. E o keycloak anexará DB_PORT (se especificar) aos hosts sem porta, caso contrário, anexará a porta padrão 5432, novamente ao endereço sem apenas porta.
- DB_PORT :Especificar a porta do banco de dados (opcional, padrão é a porta padrão do fornecedor DB)
- DB_DATABASE :Especificar o nome do banco de dados para usar (opcional, padrão é keycloak).
- DB_SCHEMA :Especificar o nome do esquema a ser usado para DB que suporta esquemas (opcional, padrão é público no Postgres).
- DB_USER: Especifique o usuário para usar para autenticar no banco de dados (opcional, padrão é '').
- DB_USER_FILE: Especifique o usuário para autenticar no banco de dados via entrada de arquivo (alternativa ao DB_USER).
- DB_PASSWORD :Especifique a senha do usuário para usar para autenticar no banco de dados (opcional, padrão é '').
- DB_PASSWORD_FILE :Especifique a senha do usuário para usar para autenticar no banco de dados via entrada de arquivo (alternativa ao DB_PASSWORD).

Exemplo MySQL

Crie uma rede definida pelo usuário

rede docker criar keycloak-network

Inicie uma instância MySQL

Primeiro inicie uma instância MySQL usando a imagem do docker MySQL:

docker run --nome mysql -d -net keycloak-network -e MYSQL_DATABASE=keycloak -e MYSQL_USER=keycloak -e MYSQL_PASSWORD=password -e MYSQL_ROOT_PASSWORD=root_password mysql

Inicie uma instância keycloak

Inicie uma instância keycloak e conecte-se à instância MySQL:

docker run --nome keycloak -net keycloak-network jboss/keycloak
Se você usou um nome diferente para a instância MySQL para mysql, você precisa especificar a variável de ambiente DB_ADDR.

Exemplo pós-gresql

Crie uma rede definida pelo usuário

rede docker criar keycloak-network

Inicie uma instância pós-ano

Primeiro inicie uma instância postgreSql usando a imagem do docker PostgreSQL:

docker run -d --name postgres --net keycloak-network -e POSTGRES_DB=keycloak -e POSTGRES_USER=keycloak -e POSTGRES_PASSWORD=password postgres

Inicie uma instância keycloak

Inicie uma instância keycloak e conecte-se à instância postgreSQL:

docker run --name keycloak -net keycloak-network jboss/keycloak -e DB_USER=keycloak -e DB_PASSWORD=password
Se você usou um nome diferente para a instância PostgreSQL para pós-gres, você precisa especificar a variável de ambiente DB_ADDR.

Exemplo mariadb

Crie uma rede definida pelo usuário

rede docker criar keycloak-network

Inicie uma instância MariaDB

Primeiro inicie uma instância do MariaDB usando a imagem do docker MariaDB:

docker run -d --name mariadb -net keycloak-network -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=keycloak -e MYSQL_USER=keycloak -e MYSQL_PASSWORD=senha mariadb

Inicie uma instância keycloak

Inicie uma instância keycloak e conecte-se à instância DoMB:

docker run --nome keycloak -net keycloak-network jboss/keycloak
Se você usou um nome diferente para a instância DoMB para mariadb, você precisa especificar a variável DB_ADDR ambiente.

Exemplo Oracle

O uso do Keycloak com um banco de dados Oracle requer que um driver JDBC seja fornecido à imagem do Docker.

Baixe o driver Oracle JDBC

1. Baixe o driver JDBC necessário para sua versão do Oracle.

2. **Importante:** renomeie o arquivo para ojdbc.jar

Crie uma rede definida pelo usuário

rede docker criar keycloak-network

Inicie uma instância Oracle

Se você já tiver um banco de dados Oracle executando esta etapa pode ser ignorada, caso contrário, aqui começaremos um novo contêiner Docker usando a imagem carloscastillo/rgt-oracle-xe-11g no Docker Hub:

docker run -d --name oracle -net keycloak-network -p 1521:1521 carloscastillo/rgt-oracle-xe-11g

Inicie uma instância keycloak

Inicie uma instância keycloak e conecte-se à instância Oracle:

docker run -d --name keycloak -net keycloak-network -p 8080:8080 -v /path/to/jdbc/driver:/opt/jboss/keycloak/modules/system/layers/base/com/oracle/jdbc/main/driver jboss/keycloak

Uma das peças-chave aqui é que estamos montando um volume a partir da localização do driver JDBC, para garantir que o caminho esteja correto. O volume montado deve conter o arquivo chamado ojdbc.jar.

Alternativamente, o arquivo JDBC pode ser copiado no contêiner usando o comando docker cp:

docker cp ojdbc.jar jboss/keycloak:/opt/jboss/keycloak/modules/system/layers/base/com/oracle/jdbc/main/driver/ojdbc.jar

Se você usou um nome para a instância Oracle além do oracle, você precisará especificar a variável DB_ADDR ambiente.

Configurações padrão do ambiente:

- DB_ADDR : oráculo
- DB_PORT : 1521
- DB_DATABASE : CAR
- DB_USER : SISTEMA
- DB_PASSWORD : oráculo

Exemplo do servidor SQL da Microsoft

Crie uma rede definida pelo usuário

rede docker criar keycloak-network

Inicie uma instância do Microsoft SQL Server

Primeiro inicie uma instância do Microsoft SQL Server usando a imagem do docker do Microsoft SQL Server:

```
docker run -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=Password!23' -d --name mssql -net keycloak-network mcr.microsoft.com/mssql/server
```

Ao contrário de algumas das outras bases de dados suportadas, como PostgreSQL, MySQL ou MariaDB, o SQL Server não suporta a criação do banco de dados inicial através de uma variável de ambiente. Consequentemente, o banco de dados deve ser criado para Keycloak de outra forma. Em princípio, isso pode ser feito criando uma imagem que se executa até que possa criar o banco de dados.

```
docker run -d --name mssql-scripts -net keycloak-network mcr.microsoft.com/mssql-tools /bin/bash -c 'até /opt/mssql-tools/bin/sqlcmd -S mssql -U sa -P "Password!23" -Q "criar banco de dados Keycloak"; dormir 5; feito '
```

Esta imagem tentará repetidamente criar o banco de dados e terminar assim que o banco de dados estiver no lugar.

Inicie uma instância keycloak

Inicie uma instância keycloak e conecte-se à instância do Microsoft SQL Server:

```
docker run --name keycloak -net keycloak-network -p 8080:8080 -e DB_VENDOR=mssql -e DB_USER=sa -e DB_PASSWORD=Password!23 -e DB_ADDR=mssql -e DB_DATABASE=Keycloak -e KEYCLOAK_USER=admin -e KEYCLOAK_PASSWORD=ad jmin
```

Se você usou um nome diferente para a instância do Microsoft SQL Server para `mssql`, você precisa especificar a variável de ambiente `DB_ADDR`.
Consulte `docker-compor-exemplos/keycloak-mssql.yml` para obter um exemplo completo.

Especifique parâmetros JDBC

Ao conectar a instância keycloak ao banco de dados, você pode especificar os parâmetros JDBC. Detalhes sobre os parâmetros JDBC podem ser encontrados aqui:

- Pós-Ano
- MySQL
- MariaDB
- oráculo
- Microsoft SQL Server

exemplo

```
docker run --name keycloak -e DB_VENDOR=postgres -e JDBC_PARAMS='connectTimeout=30' jboss/keycloak
```

Adicionando um tema personalizado

Para adicionar um tema personalizado, amplie a imagem keycloak adicione o tema ao diretório /opt/jboss/keycloak/themes.
Para definir o tema de boas-vindas, use o seguinte valor do ambiente :

- KEYCLOAK_WELCOME_THEME: Especifique o tema a ser usado para a página de boas-vindas (não deve ser vazio e deve corresponder a um nome temático existente)

Para definir seu tema personalizado como o tema global padrão, use o seguinte valor do ambiente:

- KEYCLOAK_DEFAULT_THEME :Especifique o tema a ser usado como tema global padrão (deve corresponder a um nome temático existente, se vazio usará keycloak)

Adicionando um provedor personalizado

Para adicionar um provedor personalizado, amplie a imagem Keycloak e adicione o provedor ao /opt/jboss/keycloak/autônomo/implantações/diretório.

Executando scripts personalizados na inicialização

Aviso: Os scripts personalizados não têm garantias. O layout do diretório dentro da imagem pode mudar a qualquer momento.

Para executar scripts personalizados na inicialização de contêineres coloque um arquivo no diretório /opt/jboss/startup-scripts.
Dois tipos de scripts são suportados:

- Scripts WildFly .cli . Na maioria dos casos, os scripts devem operar no modo offline (usando instrução de servidor incorporado). Também vale a pena mencionar que, por padrão, o keycloak usa configuração .xml autônoma (a menos que outra configuração do servidor seja especificada).
- Qualquer script executável(chmod +x)

Os scripts são exibidos em ordem alfabética.

Adicionando script personalizado usando Dockerfile

Um script personalizado pode ser adicionado criando seu próprio Dockerfile:
DE keycloak
COPIAR scripts personalizados/ /opt/jboss/startup-scripts/

Adicionando script personalizado usando volumes

Um único script personalizado pode ser adicionado como um volume: docker run -v /some/dir/my-script.cli:/opt/jboss/startup-scripts/my-script.cli Ou você pode volumer todo o diretório para fornecer um diretório de scripts.
Observe que ao combinar a abordagem de estender a imagem e volumede todo o diretório, o volume irá substituir todos os scripts enviados na imagem.

Inicie uma instância keycloak com opções personalizadas de linha de comando

Opções adicionais de inicialização de servidor (extensão de JAVA_OPTS) podem ser configuradas usando a variável de ambiente JAVA_OPTS_APPEND. Um caso de uso para isso é para habilitar recursos extras de perfil.

exemplo

Habilitar *upload_script* perfil:

```
docker executar -e JAVA_OPTS_APPEND="-Dkeycloak.profile.feature.upload_scripts=ativado" jboss/keycloak
```

Clustering

A substituição dos protocolos de detecção padrão (PING para a pilha UDP e MPING para o TCP) pode ser alcançada definindo algumas variáveis adicionais de ambiente:

- JGROUPS_DISCOVERY_PROTOCOL - nome do protocolo de descoberta, por exemplo, dns. DNS_PING
- JGROUPS_DISCOVERY_PROPERTIES - um parâmetro opcional com as propriedades do protocolo de detecção no seguinte formato: PROP1=FOO,PROP2=BAR
- JGROUPS_DISCOVERY_PROPERTIES_DIRECT - um parâmetro opcional com as propriedades do protocolo de detecção no formato jboss CLI: {PROP1=>FOO,PROP2=>BAR}
- JGROUPS_TRANSPORT_STACK - um nome opcional da pilha de transporte para usar udp ou tcp são valores possíveis. Padrão: tcp

Aviso: É um erro definir tanto JGROUPS_DISCOVERY_PROPERTIES quanto JGROUPS_DISCOVERY_PROPERTIES_DIRECT. Não mais do que um deles pode ser definido.

O script bootstrap detectará as variáveis e ajustará a configuração .xml autônoma com base nelas.

Exemplo ping

O protocolo de detecção ping é usado por padrão na pilha udp (que é usada por padrão em standalone-ha.xml). Uma vez que a imagem Keycloak é executada no modo agrupado por padrão, tudo o que você precisa fazer é executá-la:

```
docker executar jboss/keycloak
```

Se você duas instâncias localmente, você vai notar que eles formam um cluster.

Exemplo OpenShift com dns. DNS_PING

O clustering para OpenShift pode ser alcançado usando dns. DNS_PING e um serviço de governo ou kubernetes. KUBE_PING. Este último requer permissões de visualização que não são concedidas por padrão, por isso sugerimos o uso de dns. DNS_PING.

Usando o modelo

O exemplo completo foi colocado no diretório de exemplos de turno aberto. Basta executar um dos dois modelos. Aqui está um exemplo:

```
oc new-app -p NAMESPACE='oc project -q' -f keycloak-https.json
```

O que aconteceu debaixo do capô?

Ambos os modelos OpenShift usam dns. DNS_PING debaixo do capô. Aqui está um comando equivalente baseado em docker que o OpenShift está invocando:

```
docker executar \
-e JGROUPS_DISCOVERY_PROTOCOL=dns. DNS_PING -e \
JGROUPS_DISCOVERY_PROPERTIES=dns_query=keycloak.myproject.svc.cluster.local \
jboss/keycloak
```

Neste exemplo, os dns. DNS_PING que consultas A registros do Servidor DNS com a seguinte consulta keycloak.myproject.svc.cluster.local.

Adicionando protocolos de detecção personalizados

O mecanismo padrão para adicionar protocolos de detecção deve cobrir a maioria dos casos. No entanto, às vezes você precisa adicionar mais protocolos ao mesmo tempo, ou ajustar outros protocolos. Nesses casos, você precisará do seu próprio arquivo cli colocado em /opt/jboss/tools/cli/jgroups/discovery. O JGROUPS_DISCOVERY_PROTOCOL precisa combinar seu arquivo cli, por exemplo:

```
JGROUPS_DISCOVERY_PROTOCOL=custom_protocol
/opt/jboss/tools/cli/jgroups/discovery/custom_protocol.cli
```

Isso pode ser facilmente alcançado estendendo a imagem keycloak e adicionando apenas um arquivo.

Claro, nós encorajamos você a contribuir com seus scripts personalizados de volta para a imagem da comunidade!

Replicação e Fail-Over

Por padrão, o Keycloak NÃO replica caches como sessões, sessões de autenticação, sessões offline, 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 cairem, 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. Para mais informações sobre este assunto, consulte a Documentação Keycloak

Especifique os proprietários de cache destribuídos

- CACHE_OWNERS_COUNT :Especifique o número de proprietários de cache distribuído (padrão é 1)

AutenticaçõesSenções não serão replicadas por definição CACHE_OWNERS_COUNT>1, pois isso geralmente não é necessário/pretendido (https://www.keycloak.org/docs/latest/server_installation/#cache) Para permitir a replicação de sessões de autenticação, bem como usar:

- CACHE_OWNERS_AUTH_SESSIONS_COUNT :Especifique o número de réplicas para sessões de autenticação

abóbada

Configuração Kubernetes / OpenShift arquivos plaintext vault

Keycloak suporta a implementação do cofre para segredos kubernetes. Para usar o cofre de texto simples de arquivos no contêiner Docker, monte arquivos secretos para $JBOSS_HOME/secrets directory. Isso pode ser usado para consumir segredos do cluster Kubernetes / OpenShift.

Misc

Especificar URL base frontend

Para definir uma URL base fixa para solicitações frontend use o seguinte valor de ambiente (isso é altamente recomendado na produção):

- KEYCLOAK_FRONTEND_URL: Especifique url base para Keycloak (opcional, padrão é recuperado da solicitação)

Especificar o nível de registro

Existem duas variáveis de ambiente disponíveis para controlar o nível de registro do Keycloak:

- KEYCLOAK_LOGLEVEL :Especificar o nível de registro para Keycloak (opcional, padrão é INFO)
- ROOT_LOGLEVEL :Especificar o nível de registro para o contêiner subjacente (opcional, padrão é INFO)

Os níveis de registro suportados são ALL, DEBUG, ERROR, FATAL, INFO, OFF, TRACE e WARN.
O nível de registro também pode ser alterado no tempo de execução, por exemplo (assumindo o acesso executivo do docker):

./keycloak/bin/jboss-cli.sh --connect --command='/subsystem=logging/console-handler=CONSOLE:change-log-level (level=DEBUG)'
./keycloak/bin/jboss-cli.sh --connect --command='/subsystem=logging/root-logger=ROOT:change-root-log-level (level=DEBUG)'
./keycloak/bin/jboss-cli.sh --connect --command='/subsystem=logging/logger=org.keycloak:write-attribute(name=level,value=DEBUG)'

Habilitando o encaminhamento do endereço proxy

Ao executar o Keycloak atrás de um proxy, você precisará ativar o encaminhamento de endereço proxy.

docker run -e PROXY_ADDRESS_FORWARDING=verdadeiro jboss/keycloak

Configuração de TLS(SSL)

A imagem keycloak permite especificar uma chave privada e um certificado para servir HTTPS sobre a porta 8443. Nesse caso, você precisa fornecer dois arquivos:

- tls.crt - um certificado
- TLS.key - uma chave privada

Esses arquivos precisam ser montados em /etc/x509/https diretório. A imagem irá convertê-los automaticamente em uma loja de chaves Java e reconfigurar o Wildfly para usá-lo. NOTA: Ao usar montagens de volume em recipientes, os arquivos serão montados no recipiente como propriedade da raiz, já que a permissão padrão no arquivo de teclas provavelmente será de 700, resultará em uma loja de chaves vazia. Você terá que tornar o mundo-chave legível ou estender a imagem para adicionar as chaves com o proprietário apropriado.
Também é possível fornecer um pacote de CA adicional e configurar o Mutual TLS desta forma. Nesse caso, você precisa montar um volume adicional (ou vários volumes) na imagem. Esses volumes devem conter todos os arquivos crt necessários. O passo final é configurar a variável X509_CA_BUNDLE ambiente para conter uma lista dos locais dos vários arquivos de pacote de certificado ca especificados antes, separados por espaço ( ). Em caso de ambiente OpenShift, isso pode ser

/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt
/var/run/secrets/kubernetes.io/serviceaccount/ca.crt.
NOTA: Consulte o diretório de exemplos de turno aberto para uma configuração fora da caixa para OpenShift.

Habilite algumas métricas

A imagem keycloak pode coletar algumas estatísticas para vários subsistemas que estarão disponíveis no console de gerenciamento e no ponto final /metrics. Você pode habilitá-lo com as variáveis de ambiente KEYCLOAK_STATISTICS que fazem uma lista de estatísticas para habilitar:

- db para o subsistema de fontes de dados
- http para o subsistema undertow
- jgroups para o subsistema jgroups

por exemplo, KEYCLOAK_STATISTICS=db,http permitirá estatísticas para as fontes de dados e subsistema de subsequivação.
O valor especial tudo permite todas as estatísticas.
Uma vez ativado, você deve ver os valores das métricas mudando no ponto final /métricas para o ponto final do gerenciamento.

depuração

Para anexar um depurador Java, defina essas variáveis de ambiente:

- DEBUG=true: Agora o DEBUG_PORT vai ouvir
- DEBUG_PORT='*:8787' :A porta que o Keycloak ouve para conexões de um depurador. Por padrão, o JDK 9+ só ouve no localhost, então você vai querer a sintaxe *:8787 para fazê-lo ouvir conexões de todos os hosts. Lembre-se de citar o caractere * embora, especialmente sob zsh.

Além de definir DEBUG=true e DEBUG_PORT='*:8787', você também vai querer publicar a porta de depuração, como em:
docker run -e DEBUG=true -e DEBUG_PORT='*:8787' -p 8080:8080 -p '8787:8787' jboss/keycloak

Outros detalhes

Esta imagem estende a imagem base registry.access.redhat.com/ubi8-minimal e adiciona Keycloak e suas dependências em cima dela.

Construindo imagem com Keycloak de diferentes fontes

Construindo keycloak do repositório do GitHub

É possível construir o Keycloak a partir de um repositório do GitHub em vez de baixar o lançamento oficial. Para fazer isso, defina o argumento de GITHUB_REPO de compilar o nome do repositório DomHub e, opcionalmente, definir o GITHUB_BRANCH criar argumento para a filial construir. Por exemplo:

construção do docker --build-arg GIT_REPO=keycloak/keycloak --build-arg GIT_BRANCH=master .

Isso clonará o repositório e construirá Keycloak a partir da fonte. Se você não incluir GIT_BRANCH usará o ramo principal.

Baixe Keycloak de um local alternativo

É possível baixar a distribuição Keycloak de um local alternativo. Isso pode, por exemplo, ser útil se você quiser construir uma imagem Docker do Keycloak construída localmente. por exemplo:

construção do docker --build-arg KEYCLOAK_DIST=http://172.17.0.1:8000/keycloak-4.1.0.Final-SNAPSHOT.tar.gz .

Para keycloak construído localmente, você precisa primeiro construir a distribuição e servi-la com um navegador web. Por exemplo, use SimpleHTTPServer:

```
cd $KEYCLOAK_CHECKOUT/distribution/server-dist/target
python -m SimpleHTTPServer 8000
```

Docker começar e parar

Esta imagem suporta comandos de start e docker stop, mas não detectará alterações na configuração. Se você quiser reconfigurar o Keycloak, você deve criar um novo contêiner.