Red Hat Enterprise Linux 8

Construção, funcionamento e gerenciamento de contêineres

Construção, execução e gerenciamento de recipientes Linux no Red Hat Enterprise Linux 8

Last Updated: 2021-02-25

Red Hat Enterprise Linux 8 Construção, funcionamento e gerenciamento de contêineres

Construção, execução e gerenciamento de recipientes Linux no Red Hat Enterprise Linux 8

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

Nota Legal

Resumo

Este guia descreve como trabalhar com recipientes Linux em sistemas RHEL 8 usando ferramentas de linha de comando como podman, buildah, skopeo e runc.

Índice

PREFÁCIO

A Red Hat classifica os casos de uso de recipientes em dois grupos distintos: nó único e multi-nó, com vários nós às vezes chamados de sistemas distribuídos. O OpenShift foi construído para fornecer implantações públicas e escalonáveis de aplicações em contêineres. Além do OpenShift, entretanto, é útil ter um conjunto pequeno e ágil de ferramentas para trabalhar com contêineres.

O conjunto de ferramentas de recipiente a que nos referimos pode ser usado em um estojo de uso de um único nó. Entretanto, você também pode ligar essas ferramentas em sistemas de construção existentes, ambientes CI/CD, e até mesmo usá-las para lidar com casos de uso específicos da carga de trabalho, tais como grandes dados. Para visar o caso de uso de nó único, o Red Hat Enterprise Linux (RHEL) 8 oferece um conjunto de ferramentas para encontrar, executar, construir e compartilhar containers individuais.

Este guia descreve como trabalhar com recipientes Linux em sistemas RHEL 8 usando ferramentas de linha de comando como podman, buildah, skopeo e runc. Além destas ferramentas, a Red Hat fornece imagens base, para atuar como base para suas próprias imagens. Algumas destas imagens base visam casos de uso que vão desde aplicações comerciais (como Node.js, PHP, Java e Python) até infra-estrutura (como logging, coleta de dados e autenticação).

TORNANDO O CÓDIGO ABERTO MAIS INCLUSIVO

A Red Hat tem o compromisso de substituir a linguagem problemática em nosso código, documentação e propriedades da web. Estamos começando com estes quatro termos: master, slave, blacklist e whitelist. Por causa da enormidade deste esforço, estas mudanças serão implementadas gradualmente ao longo de vários lançamentos futuros. Para mais detalhes, veja a mensagem de nosso CTO Chris Wright.

FORNECENDO FEEDBACK SOBRE A DOCUMENTAÇÃO DA RED HAT

Agradecemos sua contribuição em nossa documentação. Por favor, diga-nos como podemos melhorá-la. Para fazer isso:

- Para comentários simples sobre passagens específicas:
  1. Certifique-se de que você está visualizando a documentação no formato *Multi-page HTML*. Além disso, certifique-se de ver o botão **Feedback** no canto superior direito do documento.
  2. Use o cursor do mouse para destacar a parte do texto que você deseja comentar.
  3. Clique no pop-up **Add Feedback** que aparece abaixo do texto destacado.
  4. Siga as instruções apresentadas.
- Para enviar comentários mais complexos, crie um bilhete Bugzilla:
  1. Ir para o site da Bugzilla.
  2. Como Componente, use **Documentation**.
  3. Preencha o campo **Description** com sua sugestão de melhoria. Inclua um link para a(s) parte(s) relevante(s) da documentação.
  4. Clique em **Submit Bug**.

CAPÍTULO 1. COMEÇANDO COM RECIPIENTES

Os recipientes Linux surgiram como uma tecnologia chave de embalagem e entrega de aplicações de código aberto, combinando o isolamento de aplicações leves com a flexibilidade dos métodos de implementação baseados em imagem.

O Red Hat Enterprise Linux implementa recipientes Linux usando tecnologias centrais como:

- Grupos de controle (cgroups) para gestão de recursos
- Namespaces para isolamento do processo
- SELinux para segurança
- Multitenancy seguro

para reduzir o potencial de explorações de segurança. Tudo isso para proporcionar um ambiente para a produção e funcionamento de contêineres de qualidade empresarial.

Red Hat OpenShift fornece ferramentas poderosas de linha de comando e Web UI para construir, gerenciar e executar containers em unidades referidas como **pods**. Entretanto, há momentos em que você pode querer construir e gerenciar containers individuais e imagens de containers fora do OpenShift. As ferramentas fornecidas para executar essas tarefas que rodam diretamente nos sistemas RHEL são descritas neste guia.

Ao contrário de outras implementações de ferramentas de contêineres, as ferramentas aqui descritas não se concentram em torno do motor monolítico de contêineres Docker e do comando **docker**. Em vez disso, fornecemos um conjunto de ferramentas de linha de comando que podem operar sem um motor de contêineres. Estas incluem:

- **podman** - Para gerenciar diretamente pods e imagens de containers (correr, parar, iniciar, ps, anexar, executar e assim por diante)
- **buildah** - Para construir, empurrar e assinar imagens de contêineres
- **skopeo** - Para copiar, inspecionar, apagar e assinar imagens
- **runc** - Para fornecer recursos de funcionamento e construção de containers para podman e buildah

Como estas ferramentas são compatíveis com a Open Container Initiative (OCI), elas podem ser usadas para gerenciar os mesmos recipientes Linux que são produzidos e gerenciados pela Docker e outros motores de recipientes compatíveis com a OCI. Entretanto, elas são especialmente adequadas para rodar diretamente no Red Hat Enterprise Linux, em casos de uso de um único nó.

Para uma plataforma de contêineres multi-nó, veja OpenShift. Em vez de confiar nas ferramentas de nó único, sem daemon descritas neste documento, o OpenShift requer um motor de contêineres baseado em daemon. Consulte Utilizando o motor de contêineres CRI-O para obter detalhes.

1.1. CONTAINERS SEM DOCKER

O Red Hat não removeu apenas o motor de contêiner Docker do OpenShift. Também removeu o motor de contêineres Docker, juntamente com o comando **docker**, do Red Hat Enterprise Linux 8 por completo. Para a RHEL 8, o Docker não está incluído e não é suportado pela Red Hat (embora ainda esteja disponível em outras fontes).

A remoção do Docker reflete uma mudança na maneira de pensar da Red Hat sobre como os contêineres são manuseados:

- Na empresa, o foco não está em executar recipientes individuais a partir da linha de comando. O principal local para a execução de containers é uma plataforma baseada em Kubernetes, como o OpenShift.
- Ao reposicionar o OpenShift como a plataforma para operar containers, motores de containers como Docker se tornam apenas mais um componente abstraído pelo OpenShift.
- Como o motor de contêineres no OpenShift não é para ser usado diretamente, ele pode ser implementado com um conjunto limitado de recursos que se concentra em fazer tudo o que o OpenShift precisa, sem ter que implementar muitos recursos autônomos.

Embora o Docker tenha saído do RHEL 8 e o motor de contêineres do OpenShift esteja desconectado do uso de um único nó, as pessoas ainda querem usar comandos para trabalhar com contêineres e imagens manualmente. Assim, a Red Hat se preparou para criar um conjunto de ferramentas para implementar a maior parte do que o comando **docker** faz.

Ferramentas como **podman**, **skopeo**, e **buildah** foram desenvolvidas para assumir esses recursos de comando **docker**. Cada ferramenta neste cenário pode ser mais leve e focada em um subconjunto de características. E sem a necessidade de um processo de daemon funcionando para implementar um motor de contêineres, estas ferramentas podem funcionar sem a sobrecarga de ter que trabalhar com um processo de daemon.

Se você ainda quiser usar o Docker no RHEL 8, você pode obter o Docker de diferentes projetos upstream, mas ele não tem suporte no RHEL 8. Como tantos recursos de linha de comando **docker** foram implementados exatamente em **podman**, você pode configurar um pseudônimo para que digitando **docker** faça com que o podman seja executado.

A instalação do pacote podman-docker configura tal pseudônimo. Assim, toda vez que você executar uma linha de comando **docker**, ela realmente roda **podman** para você.

1.2. ESCOLHENDO UMA ARQUITETURA RHEL PARA CONTÊINERES

A Red Hat fornece imagens de contêineres e software relacionado a contêineres para as seguintes arquiteturas de computador:

- AMD64 e Intel 64 (imagens de base e em camadas; sem suporte para arquiteturas de 32 bits)
- PowerPC 8 e 9 64 bits (imagem base e a maioria das imagens em camadas)
- IBM Z (imagem base e a maioria das imagens estratificadas)
- ARM 64-bit (somente imagem base)

Embora nem todas as imagens da Red Hat fossem suportadas em todas as arquiteturas no início, quase todas estão agora disponíveis em todas as arquiteturas listadas. Veja Universal Base Images (UBI): Imagens, repositórios e pacotes para uma lista de imagens suportadas.

1.3. OBTENÇÃO DE FERRAMENTAS DE CONTÊINERES

Para obter um ambiente onde você possa manipular recipientes individuais, você pode instalar um sistema Red Hat Enterprise Linux 8, depois adicionar um conjunto de ferramentas de recipientes para encontrar, rodar, construir e compartilhar recipientes. Aqui estão exemplos de ferramentas relacionadas a recipientes que você pode instalar com o RHEL 8:

- **podman** - Ferramenta do cliente para o gerenciamento de containers. Pode substituir a maioria das características do comando **docker** para trabalhar com recipientes e imagens individuais.
- **buildah** - Ferramenta do cliente para a construção de imagens de contêineres em conformidade com a OCI.
- **skopeo** - Ferramenta do cliente para cópia de imagens de contêineres de e para registros de contêineres. Inclui também recursos para assinatura e autenticação de imagens.
- **runc** - Cliente de tempo de execução de contêineres para funcionamento e trabalho com contêineres no formato Open Container Initiative (OCI).

Se você quiser criar imagens de contêineres usando o modelo de assinatura RHEL, você deve se registrar corretamente e dar direito ao computador anfitrião sobre o qual você as constrói. Quando você instala pacotes, como parte do processo de construção de um container, o processo de construção tem automaticamente acesso aos direitos disponíveis a partir do host RHEL. Assim, ele pode obter pacotes RPM a partir de qualquer repositório habilitado nesse host.

1. **Install RHEL**: Se você estiver pronto para começar, pode começar instalando um sistema Red Hat Enterprise Linux.
2. **Register RHEL**: Uma vez que a RHEL esteja instalada, registre o sistema. Você será solicitado a digitar seu nome de usuário e senha. Note que o nome de usuário e a senha são os mesmos que suas credenciais de login para o Portal do Cliente da Red Hat.

```
# subscription-manager register
Registering to: subscription.rhsm.redhat.com:443/subscription
Username: ********
Password: **********
```

3. **Subscribe RHEL**: Assine automaticamente ou determine a ID do pool de uma assinatura que inclua o Red Hat Enterprise Linux. Aqui está um exemplo de auto-inscrição de uma assinatura:

```
# assinatura-gerente anexar --auto
```

4. **Install packages**: Para começar a construir e trabalhar com containers individuais, instale o módulo container-tools, que puxa o conjunto completo de pacotes de software de containers:

```
# instalação do módulo yum -y container-tools
```

5. **Install podman-docker (optional)**: Se você estiver confortável com o comando **docker** ou usar scripts que ligam diretamente para **docker**, você pode instalar o pacote podman-docker. Esse pacote instala um link que substitui a interface de linha de comando **docker** com os comandos correspondentes **podman**. Ele também liga as páginas man juntas, portanto **man docker info** mostrará a página man **podman info**.

```
# yum install -y podman-docker
```

1.4. RECIPIENTES FUNCIONANDO COMO RAIZ OU SEM RAIZ

Executar as ferramentas de contêineres como **podman**, **skopeo**, ou **buildah** como um usuário com privilégio de super usuário (usuário root) é a melhor maneira de garantir que seus contêineres tenham pleno acesso a qualquer recurso disponível em seu sistema. Entretanto, com o recurso chamado

"Rootless Containers", geralmente disponível a partir do RHEL 8.1, você pode trabalhar com containers como um usuário regular.

Embora os motores de contêineres, como o Docker, permitam executar os comandos **docker** como um usuário regular (não-root), o daemon da doca que executa esses pedidos funciona como raiz. Assim, efetivamente, usuários regulares podem fazer solicitações através de seus contêineres que prejudicam o sistema, sem que haja clareza sobre quem fez essas solicitações. Ao configurar usuários de contêineres sem raiz, os administradores do sistema limitam as atividades de contêineres potencialmente prejudiciais dos usuários regulares, enquanto ainda permitem que esses usuários executem com segurança muitos recursos de contêineres sob suas próprias contas.

Esta seção descreve como configurar seu sistema para usar ferramentas de contêineres (Podman, Skopeo e Buildah) para trabalhar com contêineres como um usuário não-raiz (rootless). Ela também descreve algumas das limitações que você encontrará porque as contas de usuários regulares não têm acesso total a todas as características do sistema operacional que seus contêineres podem precisar para funcionar.

1.4.1. Preparado para recipientes sem raiz

Você precisa se tornar um usuário root para configurar seu sistema RHEL para permitir que contas de usuários não root utilizem ferramentas de contêineres:

1. **Install RHEL**: Instale RHEL 8.1 ou atualize para RHEL 8.1 a partir de RHEL 8.0. Versões anteriores da RHEL 7 não possuem os recursos necessários para este procedimento. Se você estiver atualizando a partir da RHEL 7.6 ou antes, continue para "Upgrade to rootless containers" depois que este procedimento for feito.

2. **Install podman and slirp4netns**: Se ainda não estiver instalado, instale os pacotes podman e slirp4netns:

   ```
   # yum instalar slirp4netns podman -y
   ```

3. **Increase user namespaces**: Para aumentar o número de espaços de nomes de usuários no núcleo, digite o seguinte:

   ```
   # echo "user.max_user_namespaces=28633" > /etc/sysctl.d/userns.conf
   # sysctl -p /etc/sysctl.d/userns.conf
   ```

4. **Create a new user account** Para criar uma nova conta de usuário e adicionar uma senha para essa conta (por exemplo, joe), digite o seguinte:

   ```
   # useradd -c "Joe Jones" joe
   # passwd joe
   ```

   O usuário é configurado automaticamente para poder utilizar o podman sem raiz. Caso você queira habilitar um usuário existente a utilizar o podman sem raiz, veja a seção Atualização para recipientes sem raiz.

5. Se você quiser executar containers com systemd, veja a seção Utilizando as imagens init do UBI .

6. **Try a podman command**: Entre diretamente como o usuário que você acabou de configurar (não use **su** ou **su -** para se tornar esse usuário porque isso não define as variáveis de ambiente corretas) e tente puxar e executar uma imagem:

   ```
   $ podman pull registry.access.redhat.com/ubi8/ubi
   ```

```
$ podman run registry.access.redhat.com/ubi8/ubi cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.1 (Ootpa)"
...
```

7. **Check rootless configuration**: Para verificar se sua configuração sem raiz está configurada corretamente, você pode executar comandos dentro do espaço de nomes de usuário modificado com o comando **podman unshare**. Como usuário rootless, o seguinte comando lhe permite ver como as uids são atribuídas ao espaço de nomes de usuário:

```
$ podman unshare cat /proc/self/uid_map
         0       1001        1
         1      65537    65536
```

1.4.2. Atualização para recipientes sem raiz

Se você atualizou a partir do RHEL 7, você deve configurar manualmente os valores de subuid e subgid para qualquer usuário existente que você queira ser capaz de usar o podman sem raiz.

Usando um nome de usuário e nome de grupo existentes (por exemplo, jill), defina a gama de IDs de usuário e grupo acessíveis que podem ser usados para seus recipientes. Aqui estão algumas advertências:

- Não inclua o UID e GID do usuário sem raiz nestas faixas
- Se você definir vários usuários de recipientes sem raiz, use faixas exclusivas para cada usuário
- Recomendamos 65536 UIDs e GIDs para máxima compatibilidade com as imagens de contêineres existentes, mas o número pode ser reduzido
- Nunca use UIDs ou GIDs abaixo de 1000 ou reutilize UIDs ou GIDs de contas de usuários existentes (que, por padrão, começam em 1000)
  Aqui está um exemplo:

```
# echo "jill:165537:65536" >> /etc/subuid
# echo "jill:165537:65536" >> /etc/subgid
```

  O usuário/grupo jill está agora alocado 65535 IDs de usuário e grupo, variando de 165537-231072. Esse usuário deve ser capaz de começar a executar comandos para trabalhar com containers agora.

1.4.3. Considerações especiais para os sem raiz

Aqui estão algumas coisas a serem consideradas ao rodar containers como um usuário não-root:

- Como usuário de contêineres não-rootados, as imagens dos contêineres são armazenadas sob seu diretório pessoal (**$HOME/.local/share/containers/storage/**), ao invés de **/var/lib/containers**.
- Os usuários que rodam containers sem raiz recebem permissão especial para rodar como uma gama de IDs de usuário e grupo no sistema hospedeiro. No entanto, eles não têm privilégios de root para o sistema operacional no host.
- Se você precisar configurar seu ambiente de recipientes sem raiz, edite os arquivos de configuração em seu diretório pessoal (**$HOME/.config/containers**). Os arquivos de

configuração incluem **storage.conf** (para configuração de armazenamento) e **libpod.conf** (para uma variedade de configurações de contêineres). Você também pode criar um arquivo **registries.conf** para identificar registros de contêineres disponíveis quando você usa **podman** para puxar, pesquisar ou executar imagens.

- Um contêiner que funciona como raiz em uma conta sem raiz pode se basear em características privilegiadas dentro de seu próprio espaço de nome. Mas isso não proporciona nenhum privilégio especial para acessar recursos protegidos no host (além de ter UIDs e GIDs extras). Aqui estão exemplos de ações de contêineres que você pode esperar que funcionem a partir de uma conta sem raiz que não funcionará:
  - Qualquer coisa que você queira acessar a partir de um diretório montado a partir do host deve ser acessível pelo UID que executa seu contêiner ou seu pedido de acesso a esse componente falhará.
  - Há algumas características do sistema que você não será capaz de mudar sem privilégio. Por exemplo, você não pode mudar o relógio do sistema simplesmente ajustando uma capacidade SYS_TIME dentro de um container e executando o serviço de tempo de rede (ntpd). Você teria que executar esse contêiner como raiz, contornando seu ambiente de contêiner sem raiz e usando o ambiente do usuário raiz, para que essa capacidade funcione, como por exemplo:

    ```
    $ sudo podman run -d --cap-add SYS_TIME ntpd
    ```

    Observe que este exemplo permite que o ntpd ajuste o tempo para todo o sistema, e não apenas dentro do container.

- Um contêiner sem raiz não tem capacidade de acesso a um porto inferior a 1024. Dentro do namespace do contêiner sem raiz ele pode, por exemplo, iniciar um serviço que expõe a porta 80 a partir de um serviço httpd do contêiner, mas não será acessível fora do namespace:

  ```
  $ podman run -d httpd
  ```

  Entretanto, um contêiner precisaria do privilégio de raiz, novamente usando o ambiente de contêineres do usuário raiz, para expor esse porto ao sistema hospedeiro:

  ```
  $ sudo podman run -d -p 80:80 httpd
  ```

- O administrador de uma estação de trabalho pode configurá-la para que os usuários possam expor serviços abaixo de 1024, mas eles devem entender as implicações de segurança. Um usuário regular poderia, por exemplo, executar um servidor web na porta oficial 80 e enganar usuários externos para que acreditassem que ele foi configurado pelo administrador. Isto é geralmente OK em uma estação de trabalho, mas pode não estar em um servidor de desenvolvimento acessível à rede, e definitivamente não deve ser feito em servidores de produção. Para permitir que os usuários se liguem às portas até a porta 80, execute o seguinte comando:

  ```
  # echo 80 > /proc/sys/net/ipv4/ip_unprivileged_port_start
  ```

- Atualmente, os recipientes sem raiz dependem da definição de subuídeas estáticas e faixas subgidas. Se você estiver usando LDAP ou Active Directory para fornecer autenticação de usuário, não há maneira automatizada de fornecer essas faixas de UID e GID aos usuários. Uma alternativa atual poderia ser definir intervalos estáticos em arquivos /etc/subuid e /etc/subgid para corresponder aos UIDs e GIDs conhecidos em uso.

- O armazenamento de contêineres deve estar em um sistema de arquivo local, porque os sistemas de arquivo remoto não funcionam bem com espaços de nomes de usuários não privilegiados.

- Uma lista contínua de deficiências de rodar **podman** e ferramentas relacionadas sem privilégios de raiz está contida em Shortcomings of Rootless Podman.

CAPÍTULO 2. TRABALHANDO COM IMAGENS DE CONTÊINERES

As imagens base do Red Hat Enterprise Linux (RHEL) podem ser usadas como base para as imagens do contêiner. Para a RHEL 8, todas as imagens base da Red Hat estão disponíveis como novas Imagens Base Universais (UBI), o que significa que você pode obtê-las e redistribuí-las livremente. Estas incluem versões do padrão RHEL, minimal, init e Red Hat Software Collections que agora estão disponíveis gratuitamente e podem ser redistribuídas. As imagens de base RHEL são:

- **Supported**: Apoiado pela Red Hat para uso com aplicações em contêineres. Eles contêm os mesmos pacotes de software seguros, testados e certificados encontrados no Red Hat Enterprise Linux.
- **Cataloged**: Listado no Catálogo de Recipientes Red Hat , com descrições, detalhes técnicos, e um índice de saúde para cada imagem.
- **Updated**: Oferecido com um calendário de atualizações bem definido, para que você saiba que está recebendo o software mais recente (veja Red Hat Container Image Updates).
- **Tracked**: Rastreado por erratas para ajudar a entender as mudanças que entram em cada atualização.
- **Reusable**: As imagens de base precisam ser baixadas e colocadas em cache em seu ambiente de produção uma vez. Cada imagem base pode ser reutilizada por todos os recipientes que a incluem como sua base.

Os UBIs para RHEL 8 fornecem a mesma qualidade de software RHEL para construir imagens de contêineres que seus predecessores não-UBI (**rhel6**, **rhel7**, **rhel-init**, e **rhel-minimal** imagens base), mas oferecem mais liberdade na forma como são usadas e distribuídas.

Para o RHEL 8, estão disponíveis imagens padrão, mínimas e de base init. A Red Hat também fornece um conjunto de imagens de tempo de execução de linguagem, baseadas em Application Streams, que você pode desenvolver quando estiver criando recipientes para aplicações que requerem tempos de execução específicos. As imagens em tempo de execução incluem python, php, ruby, nodejs, e outros.

Há também um conjunto de imagens RHEL 7 que você pode rodar nos sistemas RHEL 8. Para a RHEL 7, há imagens base tanto UBI (redistribuíveis) quanto não UBI (requerem acesso por assinatura e são não redistribuíveis). Essas imagens incluem três imagens base regulares (**rhel7**, **rhel-init**, e **rhel-minimal**) e três imagens UBI (**ubi7**, **ubi7-init**, e **ubi7-minimal**).

Embora a Red Hat não ofereça ferramentas para rodar containers nos sistemas RHEL 6, ela oferece imagens de containers RHEL 6 que você pode usar. Há uma imagem base padrão (**rhel6**) e init (**rhel6-init**) disponível para a RHEL 6, mas nenhuma imagem mínima RHEL 6. Da mesma forma, não há imagens RHEL 6 UBI.

Embora as imagens base herdadas da RHEL 7 continuarão a ser suportadas, recomenda-se que as imagens da UBI avancem. Por esse motivo, exemplos no restante deste capítulo são feitos com as imagens UBI da RHEL 8. Para uma lista das imagens UBI disponíveis da Red Hat, e informações associadas sobre repositórios UBI e código fonte, veja o artigo Imagens Base Universais (UBI): Imagens, repositórios e pacotes.

2.1. DIFERENÇAS ENTRE AS IMAGENS RHEL E AS IMAGENS UBI

As imagens UBI foram criadas para que você possa construir suas imagens de contêineres sobre uma base de software oficial da Red Hat que pode ser livremente compartilhada e implantada. De uma

perspectiva técnica, elas são quase idênticas às imagens herdadas do Red Hat Enterprise Linux, o que significa que elas têm grande segurança, desempenho e ciclos de vida, mas são lançadas sob um Contrato de Licença de Usuário Final diferente. Aqui estão alguns atributos das imagens UBI da Red Hat:

- **Built from a subset of RHEL content** As imagens da Red Hat Universal Base são construídas a partir de um subconjunto de conteúdo normal do Red Hat Enterprise Linux. Todo o conteúdo usado para construir imagens UBI selecionadas é lançado em um conjunto de repositórios yum disponíveis publicamente. Isto permite instalar pacotes extras, assim como atualizar qualquer pacote nas imagens base do UBI.

- **Redistributable**: A intenção das imagens UBI é permitir que clientes, parceiros, ISVs e outros da Red Hat padronizem as imagens UBI para que você possa construir suas imagens de contêiner sobre uma base de software oficial da Red Hat que possa ser livremente compartilhada e implantada. De uma perspectiva técnica, elas são quase idênticas às imagens herdadas do Red Hat Enterprise Linux, o que significa que elas têm grande segurança, desempenho e ciclos de vida, mas são lançadas sob um Contrato de Licença de Usuário Final diferente.

- **Base and runtime images** Além dos três tipos de imagens de base, também estão disponíveis versões UBI de várias imagens em tempo de execução. Estas imagens de tempo de execução fornecem uma base para aplicações que podem se beneficiar de tempos de execução padrão e suportados, como python, php, nodejs e ruby.

- **Enabled yum repositories**: Os seguintes repositórios yum estão habilitados dentro de cada imagem da RHEL 8 UBI:

  - O repositório **ubi-8-baseos** contém o subconjunto redistribuível de pacotes RHEL que você pode incluir em seu contêiner.

  - O repositório **ubi-8-appstream** contém pacotes de fluxos de aplicações que você pode adicionar a uma imagem UBI para ajudá-lo a padronizar os ambientes que você usa com aplicações que requerem tempos de execução específicos.

- **Licensing**: Você é livre para usar e redistribuir imagens UBI, desde que você adira ao Contrato Universal de Licenciamento de Imagens Base da Red Hat.

- **Adding UBI RPMs**: Você pode adicionar pacotes RPM a imagens UBI a partir de repositórios UBI pré-configurados. Se você estiver em um ambiente desconectado, você deve permitir a listagem da UBI Content Delivery Network(https://cdn-ubi.redhat.com) para usar esse recurso. Consulte a solução Connect to https://cdn-ubi.redhat.com para obter detalhes.

2.2. ENTENDENDO AS IMAGENS BÁSICAS PADRÃO DA RED HAT

As imagens base padrão RHEL 8 (**ubi8**) possuem um conjunto robusto de recursos de software que incluem o seguinte:

- **init system**: Todas as características do sistema de inicialização do sistema que você precisa para gerenciar os serviços do sistema estão disponíveis nas imagens base padrão. Estes sistemas init permitem instalar pacotes RPM que são pré-configurados para iniciar serviços automaticamente, como um servidor Web (**httpd**) ou um servidor FTP (**vsftpd**).

- **yum**: O software necessário para instalar pacotes de software está incluído através do conjunto padrão de comandos **yum** (**yum**, **yum-config-manager**, **yumdownloader**, e assim por diante). Para as imagens base da UBI, você tem acesso a repositórios de yum gratuitos para adicionar e atualizar software.

- **utilities**: A imagem base padrão inclui algumas utilidades úteis para trabalhar dentro do contêiner. As utilidades que estão nesta imagem base que não estão nas imagens mínimas incluem **tar**, **dmidecode**, **gzip**, **getfacl** (e outros comandos acl), **dmsetup** (e outros comandos de mapeador de dispositivos), e outros.

2.3. ENTENDENDO IMAGENS MÍNIMAS DA BASE RED HAT

As imagens **ubi8-minimal** são despojadas de imagens RHEL para serem usadas quando uma imagem de base de ossos nus é desejada. Se você estiver procurando a menor imagem base possível para usar como parte do ecossistema maior da Red Hat, você pode começar com estas imagens mínimas.

As imagens mínimas RHEL fornecem uma base para suas próprias imagens de contêineres que é menos da metade do tamanho da imagem padrão, ao mesmo tempo em que ainda é capaz de utilizar os repositórios de software RHEL e manter quaisquer requisitos de conformidade que seu software tenha.

Aqui estão algumas características das imagens de base mínimas:

- **Small size**: As imagens mínimas são cerca de 92M em disco e 32M comprimidas. Isto faz com que tenha menos da metade do tamanho das imagens padrão.

- **Software installation (microdnf)**: Em vez de incluir o recurso completo **yum** para trabalhar com repositórios de software e pacotes de software RPM, as imagens mínimas incluem o utilitário **microdnf**. o **microdnf** é uma versão reduzida de **dnf**. Inclui apenas o necessário para ativar e desativar os repositórios, bem como instalar, remover e atualizar os pacotes. Ele também tem uma opção limpa, para limpar o cache depois que os pacotes forem instalados.

- **Based on RHEL packaging**: Como imagens mínimas incorporam pacotes RPM regulares do software RHEL, com alguns recursos removidos, tais como arquivos de linguagem extra ou documentação, você pode continuar confiando nos repositórios RHEL para construir suas imagens. Isto permite que você ainda mantenha os requisitos de conformidade que você tem e que são baseados no software RHEL. Características de imagens mínimas as tornam perfeitas para experimentar aplicações que você deseja executar com o RHEL, enquanto carrega a menor quantidade possível de despesas gerais. O que você não obtém com imagens mínimas é um sistema de gerenciamento de inicialização e serviços (systemd ou System V init), um ambiente de tempo de execução Python e um monte de utilitários shell comuns.

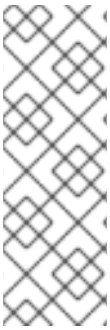
- **Modules for microdnf are not supported**: Os módulos utilizados com o comando **dnf** permitem instalar várias versões do mesmo software, quando disponíveis. O utilitário **microdnf** incluído com o mínimo de imagens não suporta módulos. Portanto, se forem necessários módulos, você deve usar uma base de imagens não-minimal, que inclui **yum**.

Se seu objetivo, entretanto, é apenas tentar executar alguns simples binários ou softwares pré-embalados que não tenham muitos requisitos do sistema operacional, as imagens mínimas podem se adequar às suas necessidades. Se sua aplicação tiver dependências de outros softwares da RHEL, você pode usar **microdnf** para instalar os pacotes necessários no momento da compilação.

A Red Hat pretende que você sempre utilize a última versão das imagens mínimas, o que está implícito ao solicitar **ubi8**/**ubi-minimal** ou **ubi8-minimal**. A Red Hat não espera suportar versões mais antigas de imagens mínimas que vão adiante.

2.4. ENTENDENDO AS IMAGENS BASE DO INIT RED HAT

As imagens do UBI **ubi8-init** contêm o sistema de inicialização do sistema, tornando-as úteis para a construção de imagens nas quais você deseja executar serviços do sistema, como um servidor web ou um servidor de arquivos. O conteúdo da imagem init é menor que o que você obtém com as imagens padrão, mas mais que o que está nas imagens mínimas.



NOTA

Como a imagem **ubi8-init** é construída em cima da imagem **ubi8**, seu conteúdo é na maioria das vezes o mesmo. Há, no entanto, algumas diferenças críticas. Em **ubi8-init**, o Cmd está configurado para /**sbin**/**init**, ao invés de **bash**, para iniciar o serviço Init do sistema por padrão. Ele inclui **ps** e comandos relacionados ao processo (pacote **procps-ng**), o que o ubi8 não faz. Além disso, **ubi8-init** define **SIGRTMIN 3** como o **StopSignal**, como systemd em **ubi8-init** ignora sinais normais para sair (**SIGTERM** e **SIGKILL**), mas terminará se receber **SIGRTMIN 3**.

Historicamente, as imagens do contêiner base do Red Hat Enterprise Linux foram projetadas para que os clientes da Red Hat executassem aplicações empresariais, mas não eram livres para redistribuir. Isto pode criar desafios para algumas organizações que precisam redistribuir suas aplicações. É aí que entram as Imagens Base Universais da Red Hat.

2.5. USANDO AS IMAGENS INIT DA UBI

Este procedimento mostra como construir um container usando um Dockerfile que instala e configura um servidor Web (**httpd**) para iniciar automaticamente pelo serviço systemd (/**sbin**/**init**) quando o container é executado em um sistema host.

Procedimento

1. Crie um Dockerfile com o seguinte conteúdo para um novo diretório:

```
FROM registry.access.redhat.com/ubi8/ubi-init
RUN yum -y install httpd; yum clean all; systemctl enable httpd;
RUN echo "Successful Web Server Test" > /var/www/html/index.html
RUN mkdir /etc/systemd/system/httpd.service.d/; echo -e '[Service]\nRestart=always' >
/etc/systemd/system/httpd.service.d/httpd.conf
EXPOSE 80
CMD [ "/sbin/init" ]
```

   O Dockerfile instala o pacote **httpd**, permite que o serviço **httpd** comece no momento da inicialização, cria um arquivo de teste (**index.html**), expõe o servidor Web ao host (porta 80) e inicia o serviço de inicialização do sistema (/**sbin**/**init**) quando o container começa.

2. Construa o recipiente:

```
# podman build --format=docker -t mysysd .
```

3. Opcionalmente, se você quiser executar containers com systemd e SELinux está habilitado em seu sistema, você deve definir a variável booleana **container_manage_cgroup**:

```
# Setsebool -P container_manage_cgroup 1
```

4. Execute o recipiente com o nome **mysysd_run**:

```
# podman run -d --name=mysd_run -p 80:80 mysysd
```

   A imagem **mysysd** funciona como o contêiner **mysysd_run** como um processo daemon, com o porto 80 do contêiner exposto ao porto 80 no sistema hospedeiro.

5. Verifique se o contêiner está funcionando:

```
# podman ps
a282b0c2ad3d  localhost/mysysd:latest  /sbin/init  15 seconds ago  Up 14 seconds ago
0.0.0.0:80->80/tcp  mysysd_run
```

6. Teste o servidor web:

```
# curl localhost/index.html
Successful Web Server Test
```

2.6. REDISTRIBUINDO IMAGENS UBI

Este procedimento descreve como redistribuir as imagens UBI. Depois de puxar uma imagem UBI, você é livre para empurrar uma imagem UBI para seu próprio registro ou para outro registro de terceiros e compartilhá-la com outros. Você pode atualizar ou adicionar a essa imagem a partir de repositórios UBI yum, como quiser.

Procedimento

1. Para obter a imagem **ubi** do registro.redhat.io, entre:

```
# podman pull registry.redhat.io/ubi8/ubi
```

2. Para adicionar um nome adicional à imagem **ubi**, entre:

```
# podman tag registry.redhat.io/ubi8/ubi registry.example.com:5000/ubi8/ubi
```

3. Para empurrar a imagem **ubi** de seu armazenamento local para um registro, entre:

```
# podman push registry.com.example.com:5000/ubi8/ubi
```

Embora haja poucas restrições sobre como usar essas imagens, há algumas restrições sobre como você pode se referir a elas. Por exemplo, você não pode chamar essas imagens de Red Hat certified ou Red Hat supported a menos que você as certifique através do Programa Red Hat Partner Connect , seja com Red Hat Container Certification ou Red Hat OpenShift Operator Certification.

2.7. BUSCA DE IMAGENS DE CONTÊINERES

O comando **podman search** permite a busca de imagens nos registros de contêineres selecionados.

NOTA

Você também pode procurar por imagens no Red Hat Container Registry . O Registro de Contêineres da Red Hat inclui a descrição da imagem, conteúdo, índice de saúde e outras informações.

Você pode encontrar a lista de registros no arquivo de configuração **registries.conf**:

```
[registries.search]
registries = ['registry.access.redhat.com', 'registry.redhat.io', 'docker.io']

[registries.insecure]
```

```
registries = []

[registries.block]
registries = []
```

- Por padrão, o comando **podman search** procura por imagens de contêineres dos registros listados na seção **[registries.search]** na ordem dada. Neste caso, o comando **podman search** procura a imagem solicitada em registry.access.redhat.com, registry.redhat.io e docker.io nesta ordem.

- A seção **[registries.insecure]** acrescenta os registros que não utilizam TLS (um registro inseguro).

- A seção **[registries.block]** permite o acesso ao registro a partir de seu sistema local.

Como um usuário root, você pode editar o arquivo **/etc/containers/registries.conf** para alterar as configurações padrão de busca em todo o sistema.

Como usuário regular (sem raiz) do **podman**, você pode criar seu próprio arquivo **registries.conf** em seu diretório pessoal (**$HOME/.config/containers/registries.conf**) para substituir as configurações de todo o sistema.

Certifique-se de que você segue as condições ao configurar os registros de contêineres:

- Cada registro deve ser cercado por aspas simples.

- Se houver vários registros definidos para o **registries = value**, você deve separar esses registros por vírgulas.

- Você pode identificar os registros por endereço IP ou hostname.

- Se o registro usa uma porta não padrão - que não seja a porta TCP 443 para seguro e 80 para inseguro, digite o número dessa porta com o nome do registro. Por exemplo: host.example.com:9999.

- O sistema procura por registros na ordem em que eles aparecem na lista **registries.search** do arquivo **registries.conf**.

Seguem alguns exemplos de comando **podman search**. O primeiro exemplo ilustra a busca mal sucedida de todas as imagens do cais.io. O forwardlash no final significa procurar em todo o registro por todas as imagens acessíveis a você:

```
# podman search quay.io/
ERRO[0000] error searching registry "quay.io": couldn't search registry "quay.io":
unable to retrieve auth token: invalid username/password
```

Para pesquisar o registro do cais.io, faça o login primeiro:

```
# podman login quay.io
Username: johndoe
Password: ***********
Login Succeeded!
# podman search quay.io/
INDEX     NAME                          DESCRIPTION   STARS   OFFICIAL   AUTOMATED
quay.io   quay.io/test/myquay                         0
quay.io   quay.io/test/redistest                      0
```

```
quay.io   quay.io/johndoe/websrv21                    0
quay.io   quay.io/johndoe/mydbtest                    0
quay.io   quay.io/johndoe/newbuild-10                 0
```

Procure em todos os registros disponíveis por imagens **postgresql** (resultando em mais de 40 imagens encontradas):

```
# podman search postgresql-10
INDEX       NAME                                DESCRIPTION                STARS OFFICIAL
AUTOMATED
redhat.io   registry.redhat.io/rhel8/postgresql-10          This container image ...       0
redhat.io   registry.redhat.io/rhscl/postgresql-10-rhel7    PostgreSQL is an advanced ...  0
quay.io     quay.io/mettle/postgresql-database-provisioning
docker.io   docker.io/centos/postgresql-10-centos7          PostgreSQL is an advanced ... 13
...
```

Para limitar sua busca por **postgresql** às imagens de registry.redhat.io, digite o seguinte comando. Observe que, inserindo o registro e o nome da imagem, qualquer repositório no registro pode ser correspondido:

```
# podman search registry.redhat.io/postgresql-10
INDEX       NAME                                DESCRIPTION          STARS   OFFICIAL
AUTOMATED
redhat.io   registry.redhat.io/rhel8/postgresql-10          This container image ...  0
redhat.io   registry.redhat.io/rhscl/postgresql-10-rhel7   PostgreSQL is an  ...     0
```

Para obter descrições mais longas para cada imagem de recipiente, adicione **--no-trunc** ao comando:

```
# podman search --no-trunc registry.redhat.io/rhel8/postgresql-10
INDEX       NAME   DESCRIPTION              STARS   OFFICIAL   AUTOMATED
redhat.io   registry.redhat.io/rhel8/postgresql-10
              This container image provides a containerized
              packaging of the PostgreSQL postgres daemon and
              client application. The postgres server daemon
              accepts connections from clients and provides
              access to content from PostgreSQL databases on
              behalf of the clients.   0
```

Para acessar os registros inseguros, adicione o nome totalmente qualificado do registro à seção **[registries.insecure]** do arquivo **/etc/containers/registries.conf**. Por exemplo:

```
[registries.search]
registries = ['myregistry.example.com']

[registries.insecure]
registries = ['myregistry.example.com']
```

Em seguida, procure por imagens em **myimage**:

```
# podman search myregistry.example.com/myimage
INDEX       NAME  DESCRIPTION                                STARS  OFFICIAL  AUTOMATED
example.com  myregistry.example.com/myimage
  The myimage container executes the ...   0
```

Agora você pode puxar a imagem **myimage**:

```
# podman pull myimage.example.com/myimage
```

2.8. DEFININDO A POLÍTICA DE VERIFICAÇÃO DE ASSINATURAS DE IMAGEM

A Red Hat entrega assinaturas para as imagens no Red Hat Container Registry. Ao rodar como raiz, **/etc/containers/policy.json**, e os arquivos YAML no diretório **/etc/containers/registries.d/** definem a política de verificação de assinaturas. A política de confiança em **/etc/containers/policy.json** descreve um escopo de registro (registro e/ou repositório) para a confiança.

Por padrão, a ferramenta de contêineres lê a política de **$HOME/.config/containers/policy.json**, caso exista, caso contrário, de **/etc/containers/policy.json**.

A confiança é definida usando três parâmetros:

1. O nome *registry* ou *registry/repository*
2. Uma ou mais chaves GPG públicas
3. Um servidor de assinatura

A Red Hat serve assinaturas destas URIs:

```
https://access.redhat.com/webassets/docker/content/sigstore
https://registry.redhat.io/containers/sigstore
```

Procedimento

1. Exibir o arquivo **/etc/containers/policy.json**:

```
# cat /etc/containers/policy.json
{
    "default": [
        {
            "type": "insecureAcceptAnything"
        }
    ],
    "transports":
    {
        "docker-daemon":
        {
            "": [{"type":"insecureAcceptAnything"}]
        }
    }
}
```

2. Para atualizar um escopo de confiança existente para os registros registry.access.redhat.com e registry.redhat.io, entre:

```
# podman image trust set -f /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
registry.access.redhat.com
# podman image trust set -f /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

```
registry.redhat.io
```

3. Para verificar a configuração da política de confiança, exibir o arquivo **/etc/containers/policy.json**:

```
        "docker": {
            "registry.access.redhat.com": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
            ],
            "registry.redhat.io": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
            ]
        },
```

Você pode ver que as seções **"registry.access.redhat.com"** e **"registry.redhat.io"** são adicionadas.

4. Criar o arquivo **/etc/containers/registries.d/registry.access.redhat.com.yaml** para identificar a loja de assinaturas de imagens de contêineres do registro registry.access.redhat.com:

```
docker:
    registry.access.redhat.com:
        sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

5. Crie o arquivo **etc/containers/registries.d/registry.redhat.io.yaml** com o seguinte conteúdo:

```
docker:
    registry.redhat.io:
        sigstore: https://registry.redhat.io/containers/sigstore
```

6. Para exibir a configuração de confiança, entre:

```
# podman image trust show                                    default                 accept
registry.access.redhat.com  signedBy              security@redhat.com, security@redhat.com
https://access.redhat.com/webassets/docker/content/sigstore
registry.redhat.io          signedBy              security@redhat.com, security@redhat.com
https://registry.redhat.io/containers/sigstore
                            insecureAcceptAnything
```

7. Para rejeitar a política de confiança padrão, digite:

```
# podman image trust set -t rejeita o padrão
```

8. Para verificar a configuração da política de confiança, exibir o site **/etc/containers/policy.json**:

```
# cat /etc/containers/policy.json
```

```
{
    "default": [
    {
        "type": "reject"
    }
...
}
```

Você pode ver que a seção **"default"** mudou de **"insecureAcceptAnything"** para **"reject"**.

9. Tente extrair a imagem mínima da Red Hat Universal Base Image 8 (**ubi8-minimal**) do registro registry.access.redhat.com:

```
# podman --log-level=debug pull registry.access.redhat.com/ubi8-minimal
....
DEBU[0000] Using registries.d directory /etc/containers/registries.d for sigstore configuration
DEBU[0000]  Using "docker" namespace registry.access.redhat.com
DEBU[0000]   Using https://access.redhat.com/webassets/docker/content/sigstore
...
```

Você vê que o endereço de armazenamento de assinaturas **access.redhat.com/webassets/docker/content/sigstore** corresponde ao endereço que você especificou no site **/etc/containers/registries.d/registry.access.redhat.com.yaml**.

10. Entrar no registro.redhat.io

```
# podman login registry.redhat.io
Username: username
Password: ***********
Login Succeeded!
```

11. Tente tirar a imagem **support-tools** do registro.redhat.io

```
# podman --log-level=debug pull registry.redhat.io/rhel8/support-tools
...
DEBU[0000] Using registries.d directory /etc/containers/registries.d for sigstore configuration
DEBU[0000]  Using "docker" namespace registry.redhat.io
DEBU[0000]   Using https://registry.redhat.io/containers/sigstore
...
```

Você pode ver que o endereço de armazenamento de assinaturas **registry.redhat.io/containers/sigstore** corresponde ao endereço que você especificou no site **/etc/containers/registries.d/registry.redhat.io.yaml**.

12. Para listar todas as imagens puxadas para seu sistema local, entre:

```
# podman images
REPOSITORY                              TAG     IMAGE ID      CREATED       SIZE
registry.redhat.io/rhel8/support-tools  latest  5ef2aab09451  13 days ago   254 MB
registry.access.redhat.com/ubi8-minimal  latest  86c870596572  13 days ago   146 MB
```

Recursos adicionais

- Para mais informações sobre o **podman image trust**, digite **man podman-image-trust**.

- Para mais informações sobre a verificação de imagens de contêineres, veja o artigo Verificação de assinatura de imagem para o Red Hat Container Registry.

2.9. TIRANDO IMAGENS DE REGISTROS

Para obter imagens de contêineres de um registro remoto (como o próprio registro de contêineres da Red Hat) e adicioná-las ao seu sistema local, use o comando **podman pull**:

```
# podman pull <registry>[:<port>]/[<nomespace>/]<nomee>:<tag>
```

O *<registry>* é um host que fornece um serviço de registro de contêineres no TCP *<port>*. Juntos, *<namespace>* e *<name>* identificam uma imagem particular controlada por *<namespace>* nesse registro. O *<tag>* é um nome adicional para a imagem armazenada localmente, a etiqueta padrão é **latest**. Sempre utilize nomes de imagens totalmente qualificados, incluindo: registro, espaço de nomes, nome e etiqueta da imagem. Ao usar nomes curtos, há sempre um risco inerente de falsificação. Adicione registros que são confiáveis, ou seja, registros que não permitem que usuários desconhecidos ou anônimos criem contas com nomes arbitrários.

Alguns registros também apóiam *<name>* em bruto; para esses, *<namespace>* é opcional. Quando é incluído, porém, o nível adicional de hierarquia que *<namespace>* fornece é útil para distinguir entre imagens com o mesmo *<name>*. Por exemplo, o nível de hierarquia adicional que fornece é útil para distinguir entre imagens com o mesmo :

Namespace	Exemplos (*<namespace>*/*<name>*)
organização	**redhat/kubernetes**, **google/kubernetes**
login (nome de usuário)	**alice/application**, **bob/application**
papel	**devel/database**, **test/database**, **prod/database**

Os registros que a Red Hat fornece são registry.redhat.io (requer autenticação), registry.access.redhat.com (não requer autenticação), e registry.connect.redhat.com (contém imagens do programa Red Hat Partner Connect). Para detalhes sobre a transição para registry.redhat.io, veja Red Hat Container Registry Authentication . Antes de poder puxar os contêineres do registry.redhat.io, você precisa autenticar. Por exemplo:

```
# podman login registry.redhat.io
Username: myusername
Password: ************
Login Succeeded!
```

Use a opção puxar para puxar uma imagem de um registro remoto. Para puxar a imagem base da RHEL **ubi** e **rsyslog** imagem de registro do registro da Red Hat, digite:

```
# podman pull registry.redhat.io/ubi8/ubi
# podman pull registry.redhat.io/rhel8/rsyslog
```

Uma imagem é identificada por um nome de registro (registry.redhat.io), um namespace name name (ubi8) e o nome da imagem (ubi). Você também poderia adicionar uma tag (que por padrão é :latest se não for inserida). O nome do repositório **ubi**, quando passado ao comando **podman pull** sem o nome de

um registro que o precede, é ambíguo e pode resultar na recuperação de uma imagem que se origina de um registro não confiável. Se houver várias versões da mesma imagem, a adição de uma tag, como **latest** para formar um nome como **ubi8/ubi:latest**, permite escolher a imagem de forma mais explícita.

Para ver as imagens resultantes do comando **podman pull** acima, juntamente com quaisquer outras imagens em seu sistema, digite **podman images**:

```
REPOSITORY                      TAG    IMAGE ID      CREATED     SIZE
registry.redhat.io/ubi8/ubi     latest eb205f07ce7d  2 weeks ago 214MB
registry.redhat.io/rhel8/rsyslog  latest 85cfba5cd49c  2 weeks ago 234MB
```

As imagens **ubi** e **rsyslog** estão agora disponíveis em seu sistema local para que você possa trabalhar com elas.

2.10. LISTAGEM DE IMAGENS

Você pode usar o comando **podman images** para ver quais imagens foram puxadas para seu sistema local e estão disponíveis para uso.

Procedimento

- Para listar imagens em armazenamento local, entre:

```
# podman images
REPOSITORY                          TAG      IMAGE ID     CREATED      VIRTUAL SIZE
registry.redhat.io/rhel8/support-tools   latest   b3d6ce4e0043 2 days ago   234MB
registry.redhat.io/ubi8/ubi-init         latest   779a05997856 2 days ago   225MB
registry.redhat.io/ubi8/ubi              latest   a80dad1c1953 3 days ago   210MB
```

2.11. INSPEÇÃO DE IMAGENS LOCAIS

Depois de puxar uma imagem para seu sistema local e antes de executá-la, é uma boa idéia investigar essa imagem. As razões para investigar uma imagem antes de executá-la incluem:

- Entendendo o que a imagem faz
- Verificar que software está dentro da imagem

Procedimento

O comando **podman inspect** exibe informações básicas sobre o que uma imagem faz. Você também tem a opção de montar a imagem em seu sistema host e usar ferramentas do host para investigar o que está na imagem. Aqui está um exemplo de como investigar o que a imagem de um container faz antes de executá-la.

1. **Inspect an image**: Execute **podman inspect** para ver qual comando é executado quando você executa a imagem do recipiente, assim como outras informações. Aqui estão exemplos de como examinar as imagens do recipiente ubi8/ubi e rhel8/rsyslog (com apenas trechos de informações mostradas aqui):

```
# podman pull registry.redhat.io/ubi8/ubi
# podman inspect registry.redhat.io/ubi8/ubi | less
...
    "Cmd": [
```

```
        "/bin/bash"
    ],
    "Labels": {
        "License": "GPLv3",
        "architecture": "x86_64",
        "authoritative-source-url": "registry.redhat.io",
        "build-date": "2018-10-24T16:46:08.916139",
        "com.redhat.build-host": "cpt-0009.osbs.prod.upshift.rdu2.redhat.com",
        "com.redhat.component": "rhel-server-container",
        "description": "The Red Hat Enterprise Linux Base image is designed to be a fully
supported...
...
```

```
# podman pull registry.redhat.io/rhel8/rsyslog
# podman inspect registry.redhat.io/rhel8/rsyslog
    "Cmd": [
      "/bin/rsyslog.sh"
    ],
    "Labels": {
      "License": "GPLv3",
      "architecture": "x86_64",
...
      "install": "podman run --rm --privileged -v /:/host -e HOST=/host -e IMAGE=IMAGE -e
NAME=NAME IMAGE /bin/install.sh",
...
      "run": "podman run -d --privileged --name NAME --net=host --pid=host -v
/etc/pki/rsyslog:/etc/pki/rsyslog -v /etc/rsyslog.conf:/etc/rsyslog.conf -v
/etc/sysconfig/rsyslog:/etc/sysconfig/rsyslog -v /etc/rsyslog.d:/etc/rsyslog.d -v /var/log:/var/log
-v /var/lib/rsyslog:/var/lib/rsyslog -v /run:/run -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -e IMAGE=IMAGE -e NAME=NAME --restart=always IMAGE
/bin/rsyslog.sh",
      "summary": "A containerized version of the rsyslog utility
...
```

O recipiente ubi8/ubi executará a concha do bash, se nenhum outro argumento for dado quando você o iniciar com **podman run**. Se um Ponto de Entrada fosse definido, seu valor seria usado em vez do valor Cmd (e o valor de Cmd seria usado como argumento para o comando Ponto de Entrada).

No segundo exemplo, a imagem do contêiner rhel8/rsyslog tem rótulos embutidos em **install** e **run**. Essas etiquetas dão uma indicação de como o recipiente deve ser instalado no sistema (instalar) e executado (rodar).

2. **Mount a container**: Usando o comando **podman**, monte um recipiente ativo para investigar melhor seu conteúdo. Este exemplo executa e lista um container **rsyslog** em execução, depois exibe o ponto de montagem a partir do qual você pode examinar o conteúdo de seu sistema de arquivos:

```
# podman run -d registry.redhat.io/rhel8/rsyslog
# podman ps
CONTAINER ID IMAGE          COMMAND       CREATED       STATUS     PORTS
NAMES
1cc92aea398d ...rsyslog:latest /bin/rsyslog.sh 37 minutes ago Up 1 day ago     myrsyslog
# podman mount 1cc92aea398d
/var/lib/containers/storage/overlay/65881e78.../merged
```

```
# ls /var/lib/containers/storage/overlay/65881e78*/merged
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr
var
```

Após o **podman mount**, o conteúdo do recipiente é acessível a partir do diretório listado no host. Use **ls** para explorar o conteúdo da imagem.

3. **Check the image's package list** Para verificar os pacotes instalados no container, diga ao comando **rpm** para examinar os pacotes instalados no ponto de montagem do container:

```
# rpm -qa --root=/var/lib/containers/storage/overlay/65881e78.../merged
redhat-release-server-7.6-4.el7.x86_64
filesystem-3.2-25.el7.x86_64
basesystem-10.0-7.el7.noarch
ncurses-base-5.9-14.20130511.el7_4.noarch
glibc-common-2.17-260.el7.x86_64
nspr-4.19.0-1.el7_5.x86_64
libstdc++-4.8.5-36.el7.x86_64
```

2.12. INSPEÇÃO DE IMAGENS REMOTAS

Para inspecionar a imagem de um container antes de puxá-lo para seu sistema, você pode usar o comando **skopeo inspect**. Com **skopeo inspect**, você pode exibir informações sobre uma imagem que reside em um registro remoto de contêineres.

Procedimento

O seguinte comando inspeciona a imagem **ubi8-init** do registro da Red Hat.

- Para inspecionar o **ubi8-init** a partir do registro.redhat.io, entre:

```
# skopeo inspect docker://registry.redhat.io/ubi8/ubi-init
{
    "Name": "registry.redhat.io/ubi8/ubi8-init",
    "Digest": "sha256:53dfe24...",
    "RepoTags": [
        "8.0.0-9",
        "8.0.0",
        "latest"
    ],
    "Created": "2019-05-13T20:50:11.437931Z",
    "DockerVersion": "1.13.1",
    "Labels": {
        "architecture": "x86_64",
        "authoritative-source-url": "registry.access.redhat.com",
        "build-date": "2019-05-13T20:49:44.207967",
        "com.redhat.build-host": "cpt-0013.osbs.prod.upshift.rdu2.redhat.com",
        "com.redhat.component": "ubi8-init-container",
        "description": "The Red Hat Enterprise Linux Init image is designed to be...
```

2.13. MARCAÇÃO DE IMAGENS

Você pode adicionar nomes às imagens para torná-las mais intuitivas para entender o que elas contêm. As imagens de marcação também podem ser usadas para identificar o registro de destino para o qual a

imagem é destinada. Usando o comando **podman tag**, você essencialmente adiciona um pseudônimo à imagem que pode consistir de várias partes. Essas partes podem incluir:

registryhost/username/NAME:tag

Você pode adicionar apenas *NAME*, se quiser. Por exemplo:

```
# podman tag 474ff279782b myrhel8
```

No exemplo anterior, a imagem **rhel8** tinha um ID de imagem de 474ff279782b. Usando **podman tag**, o nome **myrhel8** agora também está anexado ao ID da imagem. Assim, você poderia executar este recipiente pelo nome (rhel8 ou myrhel8) ou pelo ID da imagem. Note que sem adicionar uma :tag ao nome, ela foi atribuída :última como a tag. Você poderia ter definido a etiqueta como 8.0, como segue:

```
# podman tag 474ff279782b myrhel8:8.0
```

Ao início do nome, você pode opcionalmente adicionar um nome de usuário e/ou um nome de registro. O nome do usuário é na verdade o repositório no Docker.io que se refere à conta do usuário que possui o repositório. A marcação de uma imagem com um nome de registro foi mostrada na seção "Marcação de imagens", no início deste documento. Aqui está um exemplo de adição de um nome de usuário:

```
# podman tag 474ff279782b jsmith/myrhel8
# podman images | grep 474ff279782b
rhel8          latest  474ff279782b  7 days ago  139.6 MB
myrhel8        latest  474ff279782b  7 months ago  139.6 MB
myrhel8        7.1     474ff279782b  7 months ago  139.6 MB
jsmith/myrhel8  latest  474ff279782b  7 months ago  139.6 MB
```

Acima, você pode ver todos os nomes de imagens atribuídos ao ID da imagem única.

2.14. SALVANDO E CARREGANDO IMAGENS

Se você quiser salvar uma imagem de container que tenha armazenado localmente, você pode usar **podman save** para salvar a imagem em um arquivo ou diretório e restaurá-la posteriormente em outro ambiente de container. O arquivo que você salvar pode estar em qualquer um dos vários formatos diferentes de imagem de container: docker-archive, oci-archive, oci-dir (diretório com tipo oci manifext), ou docker-dir (diretório com tipo manifesto v2s2). Depois de salvar uma imagem, você pode armazená-la ou enviá-la para outra pessoa, depois **load** a imagem para reutilizá-la mais tarde. Aqui está um exemplo de como salvar uma imagem como uma tarball no formato padrão docker-archive:

```
# podman save -o myrsyslog.tar registry.redhat.io/rhel8/rsyslog:latest
# file myrsyslog.tar
myrsyslog.tar: POSIX tar archive
```

O arquivo **myrsyslog.tar** está agora armazenado em seu diretório atual. Mais tarde, quando você estiver pronto para reutilizar o tarball como uma imagem de container, você poderá importá-lo para outro ambiente podman como a seguir:

```
# podman load -i myrsyslog.tar
# podman images
REPOSITORY                    TAG    IMAGE ID      CREATED     SIZE
registry.redhat.io/rhel8/rsyslog latest 1f5313131bf0  7 weeks ago 235 MB
```

Em vez de usar **save** e **load** para armazenar e recarregar uma imagem, você pode fazer uma cópia de um container, usando **podman export** e **podman import**.

2.15. REMOÇÃO DE IMAGENS

Para ver uma lista de imagens que estão em seu sistema, execute o comando **podman images**. Para remover imagens que você não precisa mais, use o comando **podman rmi**, com o ID ou nome da imagem como opção. (Você deve parar qualquer recipiente rodando de uma imagem antes de poder remover a imagem) Aqui está um exemplo:

```
# podman rmi ubi8-init
7e85c34f126351ccb9d24e492488ba7e49820be08fe53bee02301226f2773293
```

Você pode remover várias imagens na mesma linha de comando:

```
# podman rmi registry.redhat.io/rhel8/rsyslog support-tools
46da8e23fa1461b658f9276191b4f473f366759a6c840805ed0c9ff694aa7c2f
85cfba5cd49c84786c773a9f66b8d6fca04582d5d7b921a308f04bb8ec071205
```

Se você quiser limpar todas as suas imagens, você poderia usar um comando como o seguinte para remover todas as imagens de seu registro local (certifique-se de que você fala sério antes de fazer isso!):

```
# podman rmi -a
1ca061b47bd70141d11dcb2272dee0f9ea3f76e9afd71cd121a000f3f5423731
ed904b8f2d5c1b5502dea190977e066b4f76776b98f6d5aa1e389256d5212993
83508706ef1b603e511b1b19afcb5faab565053559942db5d00415fb1ee21e96
```

Para remover imagens que têm múltiplos nomes (tags) associados a elas, é necessário adicionar a opção de força para removê-las. Por exemplo, a opção de força:

```
# podman rmi -a
A container associated with containers/storage, i.e. via Buildah, CRI-O, etc., may be associated with this image: 1de7d7b3f531

# podman rmi -f 1de7d7b3f531
1de7d7b3f531...
```

CAPÍTULO 3. TRABALHANDO COM RECIPIENTES E CÁPSULAS

Os contêineres representam um processo em execução ou parado gerado a partir dos arquivos localizados em uma imagem de contêiner descompactada. As ferramentas para operar os recipientes e trabalhar com eles são descritas nesta seção.

3.1. CONTÊINERES EM FUNCIONAMENTO

Quando você executa um comando **podman run**, você essencialmente gira e cria um novo recipiente a partir de uma imagem de recipiente. O comando que você passa para a linha de comando **podman run** vê o interior do contêiner como seu ambiente de funcionamento, portanto, por padrão, muito pouco pode ser visto do sistema hospedeiro. Por padrão, por exemplo, o aplicativo em execução vê:

- O sistema de arquivo fornecido pela imagem do contêiner.
- Uma nova tabela de processos do interior do contêiner (nenhum processo do hospedeiro pode ser visto).

Se você quiser disponibilizar um diretório do host para o container, mapear as portas de rede do container para o host, limitar a quantidade de memória que o container pode usar, ou expandir as partes da CPU disponíveis para o container, você pode fazer essas coisas a partir da linha de comando **podman run**. Aqui estão alguns exemplos de linhas de comando **podman run** que permitem diferentes características.

EXAMPLE #1 (Run a quick command): Este comando podman executa o comando **cat /etc/os-release** para ver o tipo de sistema operacional utilizado como base para o contêiner. Depois que o recipiente executa o comando, o recipiente sai e é excluído (**--rm**).

```
# podman run --rm registry.redhat.io/ubi8/ubi cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.2 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.2"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.2 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8.2:GA"
HOME_URL="https://www.redhat.com/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.2
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.2"
...
```

EXAMPLE #2 (View the Dockerfile in the container) Este é outro exemplo de execução de um comando rápido para inspecionar o conteúdo de um contêiner a partir do host. Todas as imagens em camadas que a Red Hat fornece incluem o Dockerfile a partir do qual elas são construídas em **/root/buildinfo**. Neste caso, você não precisa montar nenhum volume a partir do host.

```
podman run --rm  \
```

```
    registry.redhat.io/rhel8/rsyslog  \
    ls /root/buildinfo
Dockerfile-rhel8-rsyslog-8.2-25
```

Agora que você sabe como se chama o Dockerfile, você pode listar seu conteúdo:

```
# podman run --rm registry.redhat.io/rhel8/rsyslog \
    cat /root/buildinfo/Dockerfile-rhel8-rsyslog-8.2-25
FROM sha256:eb205f07ce7d0bb63bfe560...
LABEL maintainer="Red Hat, Inc."

RUN INSTALL_PKGS="\
rsyslog \
rsyslog-gnutls \
rsyslog-gssapi \
rsyslog-mysql \
rsyslog-pgsql \
rsyslog-relp \
" && dnf -y install $INSTALL_PKGS && rpm -V --nosize
    --nofiledigest --nomtime --nomode $INSTALL_PKGS && dnf clean all
LABEL com.redhat.component="rsyslog-container"
LABEL name="rhel8/rsyslog"
LABEL version="8.2"
...
```

EXAMPLE #3 (Run a shell inside the container) A utilização de um recipiente para lançar uma concha bash permite olhar dentro do recipiente e mudar o conteúdo. Isto define o nome do contêiner para **mybash**. O **-i** cria uma sessão interativa e o **-t** abre uma sessão terminal. Sem **-i**, a concha se abriria e depois sairia. Sem **-t**, a concha ficaria aberta, mas você não seria capaz de digitar nada na concha.

Uma vez executado o comando, você é apresentado com um prompt de shell e pode começar a executar comandos de dentro do recipiente:

```
# podman run --name=mybash -it registry.redhat.io/ubi8/ubi /bin/bash
[root@ed904b8f2d5c/]#  yum install procps-ng
[root@ed904b8f2d5c/]#  ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 00:46 pts/0    00:00:00 /bin/bash
root        35     1  0 00:51 pts/0    00:00:00 ps -ef
[root@49830c4f9cc4/]# exit
```

Embora o container não esteja mais funcionando uma vez que você saia, o container ainda existe com o novo pacote de software ainda instalado. Use **podman ps -a** para listar o contêiner:

```
# podman ps -a
CONTAINER ID IMAGE                COMMAND   CREATED        STATUS                PORTS NAMES
IS INFRA
1ca061b47bd7 .../ubi8/ubi:latest   /bin/bash 8 minutes ago  Exited 12 seconds ago       musing_brown
false
...
```

Você poderia recomeçar esse recipiente usando **podman start** com as opções **-ai**. Por exemplo:

```
# podman start -ai mybash
[root@ed904b8f2d5c/]#
```

EXAMPLE #4 (Bind mounting log files) Uma maneira de disponibilizar mensagens de registro do interior de um contêiner para o sistema hospedeiro é ligar o dispositivo hospedeiro /dev/log dentro do contêiner. Este exemplo ilustra como executar uma aplicação em um contêiner RHEL chamado **log_test** que gera mensagens de log (apenas o comando logger neste caso) e direciona essas mensagens para o dispositivo /dev/log que é montado no contêiner a partir do host. A opção **--rm** remove o contêiner após a sua execução.

```
# podman run --name="log_test" -v /dev/log:/dev/log --rm \
    registry.redhat.io/ubi8/ubi logger "Testing logging to the host"
# journalctl -b | grep Testing
Nov 12 20:00:10 ubi8 root[17210]: Testing logging to the host
```

EXAMPLE #5 (Run a service as a daemon with a static IP address) O exemplo a seguir apresenta um serviço **rsyslog** como um processo daemon, portanto, ele funciona continuamente em segundo plano. Ele também diz a **podman** para configurar a interface da rede de contêineres para um determinado endereço IP (por exemplo, 10.88.0.44). Depois disso, você pode executar o comando **podman inspect** para verificar se o endereço IP foi configurado corretamente:

```
# podman run -d --ip=10.88.0.44 registry.access.redhat.com/rhel7/rsyslog
efde5f0a8c723f70dd5cb5dc3d5039df3b962fae65575b08662e0d5b5f9fbe85
# podman inspect efde5f0a8c723 | grep 10.88.0.44
          "IPAddress": "10.88.0.44",
```

3.2. INVESTIGAÇÃO DE CONTÊINERES EM FUNCIONAMENTO E PARADOS

Depois de ter alguns contêineres em funcionamento, você pode listar tanto os contêineres que ainda estão em funcionamento como os que saíram ou pararam com o comando **podman ps**. Você também pode usar o **podman inspect** para ver informações específicas dentro desses contêineres.

3.2.1. Listagem de recipientes

Digamos que você tenha um ou mais recipientes funcionando em seu anfitrião. Para trabalhar com containers do sistema hospedeiro, você pode abrir uma concha e tentar alguns dos seguintes comandos.

podman ps: A opção **ps** mostra todos os containers que estão atualmente em funcionamento:

```
# podman run -d registry.redhat.io/rhel8/rsyslog
# podman ps
CONTAINER ID IMAGE              COMMAND       CREATED       STATUS          PORTS NAMES
74b1da000a11 rhel8/rsyslog /bin/rsyslog.sh 2 minutes ago Up About a minute      musing_brown
```

Se houver containers que não estão funcionando, mas não foram removidos (opção**--rm**), os containers estão presentes e podem ser reiniciados. O comando **podman ps -a** mostra todos os contêineres, funcionando ou parados.

```
# podman ps -a
CONTAINER ID IMAGE        COMMAND    CREATED    STATUS              PORTS NAMES     IS
INFRA
d65aecc325a4 ubi8/ubi      /bin/bash  3 secs ago Exited (0) 5 secs ago peaceful_hopper false
74b1da000a11 rhel8/rsyslog rsyslog.sh 2 mins ago Up About a minute     musing_brown    false
```

3.2.2. Inspeção de recipientes

Para inspecionar os metadados de um contêiner existente, use o comando **podman inspect**. Você pode mostrar todos os metadados ou apenas os metadados selecionados para o contêiner. Por exemplo, para mostrar todos os metadados de um contêiner selecionado, digite:

```
# podman inspect 74b1da000a11
    ...
 "ID": "74b1da000a114015886c557deec8bed9dfb80c888097aa83f30ca4074ff55fb2",
 "Created": "2018-11-13T10:30:31.884673073-05:00",
 "Path": "/bin/rsyslog.sh",
 "Args": [
     "/bin/rsyslog.sh"
 ],
 "State": {
     OciVersion": "1.0.1-dev",
     Status": "running",
     Running": true,
    ...
```

Você também pode usar a inspeção para retirar determinados pedaços de informação de um contêiner. As informações são armazenadas em uma hierarquia. Portanto, para ver o endereço IP do contêiner (IPAddress em NetworkSettings), use a opção **--format** e a identidade do contêiner. Por exemplo, o endereço IP do container:

```
# podman inspect --format='{{.NetworkSettings.IPAddress}}' 74b1da000a11
10.88.0.31
```

Exemplos de outras informações que você pode querer inspecionar incluem .Path (para ver o comando rodar com o contêiner), .Args (argumentos para o comando), .Config.ExposedPorts (portos TCP ou UDP expostos do contêiner), .State.Pid (para ver o id do processo do contêiner) e .HostConfig.PortBindings (mapeamento do porto do contêiner para o host). Aqui está um exemplo de **.State.Pid** e **.State.StartedAt**:

```
# podman inspect --format='{{.State.Pid}}' 74b1da000a11
19593
# ps -ef | grep 19593
root     19593 19583  0 10:30 ?        00:00:00 /usr/sbin/rsyslogd -n
# podman inspect --format='{{.State.StartedAt}}' 74b1da000a11
2018-11-13 10:30:35.358175255 -0500 EST
```

No primeiro exemplo, você pode ver o ID do processo do executável contentorizado no sistema hospedeiro (PID 19593). O comando **ps -ef** confirma que é o daemon **rsyslogd** em execução. O segundo exemplo mostra a data e a hora em que o contêiner foi executado.

3.2.3. Investigando dentro de um contêiner

Para investigar dentro de um contêiner em funcionamento, você pode usar o comando **podman exec**. Com **podman exec**, você pode executar um comando (como **/bin/bash**) para entrar em um processo de contêiner em funcionamento para investigar esse contêiner.

A razão para usar **podman exec**, em vez de apenas lançar o recipiente em uma concha bash, é que você pode investigar o recipiente enquanto ele está rodando sua aplicação pretendida. Anexando-se ao contêiner enquanto ele está executando sua tarefa pretendida, você obtém uma visão melhor do que o contêiner realmente faz, sem necessariamente interromper a atividade do contêiner.

Aqui está um exemplo usando o site **podman exec** para procurar em um **rsyslog** em funcionamento, depois procure dentro desse recipiente.

1. **Launch a container**: Lançar um recipiente como a imagem do recipiente **rsyslog** descrita anteriormente. Digite **podman ps** para ter certeza de que ele está funcionando:

```
# podman ps
CONTAINER ID  IMAGE          COMMAND          CREATED      STATUS       PORTS
NAMES
74b1da000a11  rsyslog:latest "/usr/rsyslog.sh  6 minutes ago Up 6 minutes          rsyslog
```

2. Digite o recipiente com **podman exec**: Use o ID ou nome do recipiente para abrir uma concha bash para acessar o recipiente em funcionamento. Em seguida, você pode investigar os atributos do contêiner da seguinte forma:

```
# podman exec -it 74b1da000a11 /bin/bash
[root@74b1da000a11 /]# cat /etc/redhat-release
Red Hat Enterprise Linux release 8.0
[root@74b1da000a11 /]# yum install procps-ng
[root@74b1da000a11 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 15:30 ?        00:00:00 /usr/sbin/rsyslogd -n
root         8     0  6 16:01 pts/0    00:00:00 /bin/bash
root        21     8  0 16:01 pts/0    00:00:00 ps -ef
[root@74b1da000a11 /]# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          39G  2.5G   37G   7% /
tmpfs            64M     0   64M   0% /dev
tmpfs           1.5G  8.7M  1.5G   1% /etc/hosts
shm              63M     0   63M   0% /dev/shm
tmpfs           1.5G     0  1.5G   0% /sys/fs/cgroup
tmpfs           1.5G     0  1.5G   0% /proc/acpi
tmpfs           1.5G     0  1.5G   0% /proc/scsi
tmpfs           1.5G     0  1.5G   0% /sys/firmware
[root@74b1da000a11 /]# uname -r
4.18.0-80.1.2.el8_0.x86_64
[root@74b1da000a11 /]# rpm -qa | more
redhat-release-8.0-0.44.el8.x86_64
filesystem-3.8-2.el8.x86_64
basesystem-11-5.el8.noarch
ncurses-base-6.1-7.20180224.el8.noarch
...
bash-4.2# free -m
              total        used        free      shared  buff/cache   available
Mem:           1941         560         139          10        1241        1189
Swap:          1023          15        1008
[root@74b1da000a11 /]# exit
```

Os comandos apenas funcionam a partir do bash shell (que corre dentro do contêiner) e mostram várias coisas.

- O container foi construído a partir de uma imagem RHEL release 8.0.
- A tabela de processos (**ps -ef**) mostra que o comando **/usr/sbin/rsyslogd** é o ID de processo 1.

- Os processos em execução na tabela de processos do host não podem ser vistos de dentro do contêiner. Embora o processo **rsyslogd** possa ser visto na tabela de processos do host (era o ID do processo 19593 no host).
- Não há kernel separado rodando no recipiente (**uname -r** mostra o kernel do sistema hospedeiro).
- O comando **rpm -qa** permite que você veja os pacotes RPM que estão incluídos dentro do contêiner. Em outras palavras, há um banco de dados RPM dentro do contêiner.
- A visualização da memória (**free -m**) mostra a memória disponível no host (embora o que o container pode realmente usar possa ser limitado usando cgroups).

3.3. INÍCIO E PARADA DE CONTAINERS

Se você operou um container, mas não o removeu (**--rm**), esse container é armazenado em seu sistema local e está pronto para funcionar novamente. Para iniciar um contêiner que não foi removido anteriormente, use a opção **start**. Para parar um contêiner em funcionamento, use a opção **stop**.

3.3.1. Containers de partida

Um contêiner que não precisa funcionar interativamente às vezes pode ser reiniciado após ser parado apenas com a opção **start** e o ID ou nome do contêiner. Por exemplo:

```
# podman start myrhel_httpd
myrhel_httpd
```

Para iniciar um recipiente para que você possa trabalhar com ele a partir da concha local, use as opções **-a** (anexar) e **-i** (interativo). Uma vez que a concha bash for iniciada, execute os comandos desejados dentro do contêiner e digite saída para matar a concha e parar o contêiner.

```
# podman start -a -i agitated_hopper
[root@d65aecc325a4 /]#  exit
```

3.3.2. Parada de contêineres

Para parar um contêiner em funcionamento que não esteja ligado a uma sessão terminal, use a opção de parada e o ID ou número do contêiner. Por exemplo, a opção de parada:

```
# podman stop 74b1da000a11
74b1da000a114015886c557deec8bed9dfb80c888097aa83f30ca4074ff55fb2
```

A opção **stop** envia um sinal SIGTERM para encerrar um contêiner em funcionamento. Se o contêiner não parar após um período de carência (10 segundos por padrão), **podman** envia um sinal SIGKILL. Você também pode usar o comando **podman kill** para matar um recipiente (SIGKILL) ou enviar um sinal diferente para um recipiente. Aqui está um exemplo de envio de um sinal SIGHUP para um contêiner (se suportado pelo aplicativo, um SIGHUP faz com que o aplicativo releia seus arquivos de configuração):

```
# podman kill --signal="SIGHUP" 74b1da000a11
74b1da000a114015886c557deec8bed9dfb80c888097aa83f30ca4074ff55fb2
```

3.4. COMPARTILHAMENTO DE ARQUIVOS ENTRE DOIS CONTÊINERES

Você pode usar volumes para persistir dados em recipientes, mesmo quando um recipiente é excluído. Os volumes podem ser usados para compartilhar dados entre vários recipientes. O volume é uma pasta que é armazenada na máquina host. O volume pode ser compartilhado entre o contêiner e o host.

As principais vantagens são:

- Os volumes podem ser compartilhados entre os recipientes.
- Os volumes são mais fáceis de fazer backup ou migrar.
- Os volumes não aumentam o tamanho dos recipientes.

Procedimento

1. Para criar um volume, entre:

```
Volume do podman cria volume de host
```

2. Para exibir informações sobre o volume, entre:

```
$ podman volume inspect hostvolume
[
    {
        "name": "hostvolume",
        "labels": {},
        "mountpoint":
"/home/username/.local/share/containers/storage/volumes/hostvolume/_data",
        "driver": "local",
        "options": {},
        "scope": "local"
    }
]
```

Note que ele cria um volume no diretório de volumes. Você pode salvar o caminho do ponto de montagem para a variável para facilitar a manipulação: **$ mntPoint=$(podman volume inspect hostvolume --format {{.Mountpoint}})**.

Note que se você executar **sudo podman volume create hostvolume**, então o ponto de montagem muda para **/var/lib/containers/storage/volumes/hostvolume/_data**.

3. Criar um arquivo de texto dentro do diretório usando o caminho é armazenado na variável **mntPoint**:

```
$ echo "Olá do anfitrião" >> $mntPoint/host.txt
```

4. Liste todos os arquivos no diretório definido pela variável **mntPoint**:

```
$ ls $mntPoint/
host.txt
```

5. Executar o container com o nome **myubi1** e mapear o diretório definido pelo nome do volume **hostvolume** no host para o diretório **/containervolume1** no container:

```
$ podman run -it --nome myubi1 -v hostvolume:/containervolume1
registry.access.redhat.com/ubi8/ubi /bin/bash
```

Note que se você usar o caminho do volume definido pela variável **mntPoint** (**-v $mntPoint:/containervolume1**), os dados podem ser perdidos ao executar o comando **podman volume prune**, que remove os volumes não utilizados. Use sempre o comando **-v *hostvolume_name:/containervolume_name***.

6. Liste os arquivos no volume compartilhado no contêiner:

```
# ls /containervolume1
host.txt
```

Você pode ver o arquivo **host.txt** que você criou no host.

7. Criar um arquivo de texto dentro do diretório **/containervolume1**:

```
# echo "Olá do recipiente 1 >> /containervolume1/container1.txt
```

8. Desprenda-se do recipiente com **CTRL p** e **CTRL q**.

9. Liste os arquivos no volume compartilhado no host, você deve ver dois arquivos:

```
$ ls $mntPoint
container1.rxt  host.txt
```

Neste ponto, você está compartilhando arquivos entre o contêiner e o anfitrião. Para compartilhar arquivos entre dois contêineres, execute outro contêiner chamado **myubi2**. Os passos 10 - 13 são análogos aos passos 5 - 8.

10. Executar o container com o nome **myubi2** e mapear o diretório definido pelo nome do volume **hostvolume** no host para o diretório **/containervolume2** no container:

```
$ podman run -it --name myubi2 -v hostvolume:/containervolume2
registry.access.redhat.com/ubi8/ubi  /bin/bash
```

11. Liste os arquivos no volume compartilhado no contêiner:

```
# ls /containervolume2
container1.txt host.txt
```

Você pode ver o arquivo **host.txt** que você criou no host e **container1.txt** que você criou dentro do container **myubi1**.

12. Criar um arquivo de texto dentro do diretório **/containervolume2**:

```
# echo "Olá do recipiente 2 >> /containervolume2/container2.txt
```

13. Desprenda-se do recipiente com **CTRL p** e **CTRL q**.

14. Liste os arquivos no volume compartilhado no host, você deve ver três arquivos:

```
$ ls $mntPoint
container1.rxt  container2.txt host.txt
```

15. Para parar e remover ambos os recipientes, entre:

```
$ podman stop myubi1
$ podman stop myubi2
$ podman rm myubi1
$ podman rm myubi2
```

16. Para remover o volume do hospedeiro, entre:

```
podman volume rm hostvolume
```

17. Para verificar se você apagou o volume, entre:

```
Volume ls de podman
```

Recursos adicionais

- Para mais informações sobre o comando **podman volume**, digite **man podman-volume**.

3.5. REMOÇÃO DE RECIPIENTES

Para ver uma lista de containers que ainda estão pendurados em seu sistema, execute o comando **podman ps -a**. Para remover os containers que você não precisa mais, use o comando **podman rm**, com o ID ou nome do container como opção. Você deve parar qualquer contêiner que ainda esteja rodando antes de removê-los. Aqui está um exemplo:

```
# podman rm goofy_wozniak
```

Você pode remover vários recipientes na mesma linha de comando:

```
# podman rm esperto_yonath furioso_shockley bêbado_newton
```

Se você quiser limpar todos os seus recipientes, você poderia usar um comando como o seguinte para remover todos os recipientes (não as imagens) de seu sistema local (certifique-se de que você fala sério antes de fazer isso!):

```
# podman rm -a
56c496350bd534da7620fe2fa660526a6fc7f1c57b0298291cd2210311fe723b
83ad58c17b20f9e8271171f3023ae094dbfab6ce5708344a68feb121916961ca
a93b696a1f5629300382a8ce860c4ba42f664db98101e82c2dbcc2074b428faf
bee71e61b53bd8b036b2e8cb8f570ef8308403502760a27ee23a4b675d92b93d
```

3.6. CRIAÇÃO DE CÁPSULAS

Os contêineres são a menor unidade que você pode gerenciar com Podman, Skopeo e ferramentas de contêineres Buildah. Um Podman podman é um grupo de um ou mais contêineres. O conceito de Podman foi introduzido pela Kubernetes. As cápsulas Podman são similares à definição da Kubernetes. Pods são as menores unidades de computação que você pode criar, implementar e gerenciar em ambientes OpenShift ou Kubernetes. Cada Podman podman inclui um contêiner de infra-contêiner. Este container contém os espaços de nomes associados ao pod e permite que o Podman conecte outros containers ao pod. Ele permite iniciar e parar os containers dentro do pod e o pod continuará funcionando. A infra-container padrão é baseada na imagem Kubernetes **k8s.gcr.io/pause**.

Este procedimento mostra como criar uma cápsula com um recipiente.

Procedimento

1. Criar uma cápsula vazia:

```
$ podman pod create --name mypod
223df6b390b4ea87a090a4b5207f7b9b003187a6960bd37631ae9bc12c433aff
The pod is in the initial state Created.
```

   A cápsula está no estado inicial Criada.

2. Liste todas as cápsulas:

```
$ podman pod ps
POD ID        NAME    STATUS    CREATED               # OF CONTAINERS   INFRA ID
223df6b390b4  mypod   Created   Less than a second ago  1               3afdcd93de3e
```

   Observe que a cápsula tem um recipiente dentro dela.

3. Liste todas as cápsulas e recipientes associados a elas:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                COMMAND  CREATED               STATUS   PORTS
NAMES              POD
3afdcd93de3e  k8s.gcr.io/pause:3.1          Less than a second ago  Created
223df6b390b4-infra  223df6b390b4
```

   Você pode ver que o ID da cápsula do comando **podman ps** corresponde ao ID da cápsula no comando **podman pod ps**. O contêiner padrão da infra-contentor é baseado na imagem **k8s.gcr.io/pause**.

4. Para operar um contêiner chamado **myubi** na cápsula existente, chamado **mypod**, digite:

```
$ podman run -dt --name myubi --pod mypod registry.access.redhat.com/ubi8/ubi  /bin/bash
5df5c48fea87860cf75822ceab8370548b04c78be9fc156570949013863ccf71
```

5. Liste todas as cápsulas:

```
$ podman pod ps
POD ID        NAME    STATUS    CREATED               # OF CONTAINERS   INFRA ID
223df6b390b4  mypod   Running   Less than a second ago  2               3afdcd93de3e
```

   Você pode ver que a cápsula tem dois recipientes dentro.

6. Liste todas as cápsulas e recipientes associados a elas:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                 COMMAND    CREATED
STATUS                  PORTS  NAMES              POD
5df5c48fea87  registry.access.redhat.com/ubi8/ubi:latest  /bin/bash  Less than a second ago
Up Less than a second ago         myubi              223df6b390b4
3afdcd93de3e  k8s.gcr.io/pause:3.1                          Less than a second ago  Up Less
than a second ago         223df6b390b4-infra  223df6b390b4
```

Recursos adicionais

- Para mais informações sobre o comando **podman pod create**, digite **man podman-pod-create**.
- Para mais informações sobre as cápsulas, veja o artigo Podman: Gerenciamento de cápsulas e recipientes em um contêiner local administrado por Brent Baude.

3.7. EXIBIÇÃO DE INFORMAÇÕES SOBRE A CÁPSULA

Esta seção fornece informações sobre como exibir as informações da cápsula.

Pré-requisitos

- A cápsula foi criada. Para obter detalhes, consulte a seção Criação de cápsulas.

Procedimento

- Exibir processos ativos em execução em uma cápsula:
  - Para exibir os processos de funcionamento dos recipientes em uma cápsula, entre:

```
$ podman pod top mypod
USER   PID   PPID   %CPU    ELAPSED         TTY     TIME   COMMAND
0      1     0      0.000   24.077433518s   ?       0s     /pause
root   1     0      0.000   24.078146025s   pts/0   0s     /bin/bash
```

  - Para exibir um fluxo vivo de estatísticas de uso de recursos para recipientes em uma ou mais cápsulas, entre:

```
$ podman pod stats -a --no-stream
ID            NAME             CPU %   MEM USAGE / LIMIT   MEM %   NET IO    BLOCK IO
PIDS
a9f807ffaacd  frosty_hodgkin   --      3.092MB / 16.7GB    0.02%   -- / --   -- / --    2
3b33001239ee  sleepy_stallman  --      -- / --             --      -- / --   -- / --    --
```

  - Para exibir informações descrevendo a cápsula, entre:

```
$ podman pod inspect mypod
{
   "Id": "db99446fa9c6d10b973d1ce55a42a6850357e0cd447d9bac5627bb2516b5b19a",
   "Name": "mypod",
   "Created": "2020-09-08T10:35:07.536541534+02:00",
   "CreateCommand": [
      "podman",
      "pod",
      "create",
      "--name",
      "mypod"
   ],
   "State": "Running",
   "Hostname": "mypod",
   "CreateCgroup": false,
   "CgroupParent": "/libpod_parent",
   "CgroupPath":
"/libpod_parent/db99446fa9c6d10b973d1ce55a42a6850357e0cd447d9bac5627bb2516b5
b19a",
```

```
    "CreateInfra": false,
    "InfraContainerID":
"891c54f70783dcad596d888040700d93f3ead01921894bc19c10b0a03c738ff7",
    "SharedNamespaces": [
        "uts",
        "ipc",
        "net"
    ],
    "NumContainers": 2,
    "Containers": [
        {
            "Id":
"891c54f70783dcad596d888040700d93f3ead01921894bc19c10b0a03c738ff7",
            "Name": "db99446fa9c6-infra",
            "State": "running"
        },
        {
            "Id":
"effc5bbcfe505b522e3bf8fbb5705a39f94a455a66fd81e542bcc27d39727d2d",
            "Name": "myubi",
            "State": "running"
        }
    ]
}
```

Você pode ver informações sobre recipientes na cápsula.

Recursos adicionais

- Para mais informações sobre o comando **podman pod top**, digite **man podman-pod-top**.
- Para mais informações sobre o comando **podman pod stats**, digite **man podman-pod-stats**.
- Para mais informações sobre o comando **podman pod inspect**, digite **man podman-pod-inspect**.

3.8. PARANDO AS CÁPSULAS

Você pode parar uma ou mais cápsulas usando o comando **podman pod stop**.

Pré-requisitos

- A cápsula foi criada. Para obter detalhes, consulte a seção Criação de cápsulas.

Procedimento

1. Para parar a cápsula **mypod**, digite:

```
$ podman podman stop mypod
```

2. Liste todas as cápsulas e recipientes associados a elas:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                         COMMAND    CREATED         STATUS
```

```
PORTS   NAMES              POD ID        PODNAME
5df5c48fea87  registry.redhat.io/ubi8/ubi:latest  /bin/bash  About a minute ago  Exited (0) 7
seconds ago          myubi              223df6b390b4  mypod

3afdcd93de3e  k8s.gcr.io/pause:3.2                      About a minute ago  Exited (0) 7
seconds ago          8a4e6527ac9d-infra  223df6b390b4  mypod
```

Você pode ver que a cápsula **mypod** e o recipiente **myubi** estão em status de "Exilados".

Recursos adicionais

- Para mais informações sobre o comando **podman pod stop**, digite **man podman-pod-stop**.

3.9. REMOÇÃO DE CÁPSULAS

Você pode remover uma ou mais cápsulas e recipientes parados usando o comando **podman pod rm**.

Pré-requisitos

- A cápsula foi criada. Para obter detalhes, consulte a seção Criação de cápsulas.
- A cápsula foi parada. Para detalhes, consulte a seção Parando as cápsulas.

Procedimento

1. Para remover a cápsula **mypod**, digite:

```
$ podman pod rm mypod
223df6b390b4ea87a090a4b5207f7b9b003187a6960bd37631ae9bc12c433aff
```

   Note que a remoção da cápsula remove automaticamente todos os recipientes dentro dela.

2. Para verificar se todos os recipientes e cápsulas foram removidos, digite:

```
$ podman ps
$ podman pod ps
```

Recursos adicionais

- Para mais informações sobre o comando **podman pod rm**, digite **man podman-pod-rm**.

CAPÍTULO 4. ADICIONANDO SOFTWARE A UM RECIPIENTE UBI EM FUNCIONAMENTO

As imagens UBI são construídas a partir do conteúdo da Red Hat. Estas imagens UBI também fornecem um subconjunto de pacotes Red Hat Enterprise Linux que estão livremente disponíveis para instalação para uso com UBI. Para adicionar ou atualizar software, as imagens UBI são pré-configuradas para apontar para os repositórios yum disponíveis gratuitamente que possuem RPMs oficiais da Red Hat.

Para adicionar pacotes do UBI repos a contêineres UBI em funcionamento:

- Em **ubi** imagens, o comando **yum** é instalado para permitir que você desenhe pacotes.
- Em **ubi-minimal** imagens, o comando **microdnf** (com um conjunto de recursos menor) está incluído em vez de **yum**.

Tenha em mente que instalar e trabalhar com pacotes de software diretamente em containers em execução é apenas para adicionar pacotes temporariamente ou aprender sobre os repositórios. Consulte a seção "Construir uma imagem baseada em UBI" para formas mais permanentes de construir imagens baseadas em UBI.

Aqui estão algumas questões a serem consideradas ao trabalhar com imagens do UBI:

- Centenas de pacotes de RPM usados nos fluxos de aplicações existentes são armazenados nos repositórios de yum, empacotados com as novas imagens UBI. Esteja à vontade para instalar esses RPMs em suas imagens UBI para emular o tempo de execução (python, php, nodejs, etc.) que lhe interessa.
- Como alguns arquivos de idiomas e documentação foram retirados da imagem mínima da UBI (**ubi8/ubi-minimal**), rodando **rpm -Va** dentro daquele contêiner mostrará o conteúdo de muitos pacotes como estando faltando ou modificado. Se tiver uma lista completa de arquivos dentro daquele contêiner é importante para você, considere o uso de uma ferramenta como **Tripwire** para registrar os arquivos no contêiner e verificá-lo mais tarde.
- Após a criação de uma imagem em camadas, use **podman history** para verificar em qual imagem UBI ela foi construída. Por exemplo, depois de completar o exemplo do servidor web mostrado anteriormente, digite **podman history johndoe/webserver** para ver que a imagem em que foi construída inclui o ID da imagem UBI que você adicionou na linha FROM do Dockerfile.

Quando você adiciona software a um contêiner UBI, os procedimentos diferem para atualizar as imagens UBI em um host RHEL subscrito ou em um sistema não-RHEL (ou não-RHEL) não subscrito. Essas duas formas de trabalhar com imagens UBI são ilustradas abaixo.

4.1. ADICIONANDO SOFTWARE A UM CONTÊINER UBI EM UM HOST SUBSCRITO

Se você estiver rodando um contêiner UBI em um host RHEL registrado e subscrito, o repositório principal do servidor RHEL é habilitado dentro do contêiner padrão UBI, juntamente com todos os repositórios UBI. Portanto, o conjunto completo de pacotes da Red Hat está disponível. A partir do contêiner mínimo UBI, todos os repositórios UBI são habilitados por padrão, mas nenhum repositório é habilitado a partir do host por padrão.

4.2. ADIÇÃO DE SOFTWARE DENTRO DO CONTÊINER PADRÃO UBI

Para garantir que os recipientes que você constrói possam ser redistribuídos, desabilite os repositórios não UBI yum na imagem padrão UBI quando você adiciona software. Se você desabilitar todos os repositórios yum exceto os repositórios UBI, somente pacotes dos repositórios disponíveis gratuitamente são usados quando você adiciona software.

Com uma concha aberta dentro de um container de imagem base padrão UBI (**ubi8**/**ubi**) de um host RHEL subscrito, execute o seguinte comando para adicionar um pacote a esse container (por exemplo, o pacote **bzip2**):

```
# yum install --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-8-baseos bzip2
```

Para adicionar software dentro de um container UBI padrão que está no repositório do servidor RHEL, mas não nos repositórios UBI, não desative nenhum repositório e instale apenas o pacote:

```
# yum instalar zsh
```

Para instalar um pacote que está em um repositório hospedeiro diferente de dentro do contêiner padrão UBI, você tem que habilitar explicitamente o repositório que você precisa. Por exemplo:

```
# yum install --enablerepo=rhel-7-server-optional-rpms zsh-html
```

ATENÇÃO

A instalação de pacotes Red Hat que não estejam dentro do repositório UBI da Red Hat pode limitar o quanto você pode distribuir o contêiner fora dos hosts subscritos.

4.3. ADIÇÃO DE SOFTWARE DENTRO DO CONTÊINER UBI MÍNIMO

Os repositórios UBI yum são habilitados dentro da imagem mínima UBI por padrão.

Para instalar o mesmo pacote demonstrado anteriormente (**bzip2**) a partir de um desses repositórios UBI yum em um host RHEL subscrito a partir do container mínimo UBI, digite:

```
# microdnf instalar bzip2
```

Para instalar pacotes dentro de um contêiner UBI mínimo de repositórios disponíveis em um host subscrito que não fazem parte de um repositório UBI yum, você teria que habilitar explicitamente esses repositórios. Por exemplo:

```
# microdnf install --enablerepo=rhel-7-server-rpms zsh
# microdnf install --enablerepo=rhel-7-server-rpms \
      --enablerepo=rhel-7-server-optional-rpms zsh-html
```

ATENÇÃO

O uso de repositórios RHEL não-UBI para instalar pacotes em suas imagens UBI poderia restringir sua capacidade de compartilhar essas imagens para rodar fora dos sistemas RHEL subscritos.

4.4. ADICIONAR SOFTWARE A UM CONTÊINER UBI EM UM HOST NÃO INSCRITO

Para adicionar pacotes de software a um contêiner em execução que esteja em um host RHEL não inscrito ou em algum outro sistema Linux, você não precisa desabilitar nenhum repositório yum. Por exemplo, o sistema RHEL não tem que desabilitar nenhum repositório yum:

```
# yum instalar bzip2
```

Para instalar esse pacote em um host RHEL não subscrito do UBI, digite o mínimo container:

```
# microdnf instalar bzip2
```

Como observado anteriormente, estes dois meios de adicionar software a um contêiner UBI em funcionamento não se destinam a criar imagens permanentes de contêineres baseados em UBI. Para isso, você deve construir novas camadas sobre as imagens UBI, conforme descrito na seção seguinte.

4.5. CONSTRUINDO UMA IMAGEM BASEADA NA UBI

Você pode construir imagens de contêineres baseados no UBI da mesma forma que constrói outras imagens, com uma exceção. Você deve desativar todos os repositórios de yum não baseados no UBI quando você realmente construir as imagens, se você quiser ter certeza de que sua imagem contém apenas o software Red Hat que você pode redistribuir.

Aqui está um exemplo de criação de um container de servidor Web baseado no UBI a partir de um Dockerfile com o utilitário **buildah**:

NOTA

Para imagens de **ubi8/ubi-minimal**, use **microdnf** ao invés de **yum** abaixo:

```
RUN microdnf update -y && rm -rf /var/cache/yum
RUN microdnf install httpd -y && microdnf clean all
```

1. **Create a Dockerfile**: Adicione um **Dockerfile** com o seguinte conteúdo a um novo diretório:

```
FROM registry.access.redhat.com/ubi8/ubi
USER root
LABEL maintainer="John Doe"
# Update image
RUN yum update --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-8-baseos
-y && rm -rf /var/cache/yum
```

```
RUN yum install --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-8-baseos
httpd -y && rm -rf /var/cache/yum
# Add default Web page and expose port
RUN echo "The Web Server is Running" > /var/www/html/index.html
EXPOSE 80
# Start the service
CMD ["-D", "FOREGROUND"]
ENTRYPOINT ["/usr/sbin/httpd"]
```

2. **Build the new image**: Enquanto estiver nesse diretório, use **buildah** para criar uma nova imagem em camadas da UBI:

```
# buildah bud -t johndoe/webserver .
STEP 1: FROM registry.access.redhat.com/ubi8/ubi:latest
STEP 2: USER root
STEP 3: LABEL maintainer="John Doe"
STEP 4: RUN yum update --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-
8-baseos -y
. . .
No packages marked for update
STEP 5: RUN yum install --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-
8-baseos httpd -y
Loaded plugins: ovl, product-id, search-disabled-repos
Resolving Dependencies
--> Running transaction check
================================================================
 Package                Arch             Version                      Repository                 Size
================================================================
Installing:
 httpd              x86_64 2.4.37-10
                                        latest-rhubi-8.0-appstream 1.4 M
Installing dependencies:
 apr                x86_64 1.6.3-9.el8        latest-rhubi-8.0-appstream 125 k
 apr-util           x86_64 1.6.1-6.el8        latest-rhubi-8.0-appstream 105 k
 httpd-filesystem   noarch 2.4.37-10
                                        latest-rhubi-8.0-appstream  34 k
 httpd-tools        x86_64 2.4.37-10.
...

Transaction Summary
...
Complete!
STEP 6: RUN echo "The Web Server is Running" > /var/www/html/index.html
STEP 7: EXPOSE 80
STEP 8: CMD ["-D", "FOREGROUND"]
STEP 9: ENTRYPOINT ["/usr/sbin/httpd"]
STEP 10: COMMIT
...
Writing manifest to image destination
Storing signatures
--> 36a604cc0dd3657b46f8762d7ef69873f65e16343b54c63096e636c80f0d68c7
```

3. **Test**: Teste a imagem do servidor web em camadas da UBI:

```
# podman run -d -p 80:80 johndoe/webserver
```

```
bbe98c71d18720d966e4567949888dc4fb86eec7d304e785d5177168a5965f64
# curl http://localhost/index.html
The Web Server is Running
```

4.6. USANDO IMAGENS EM TEMPO DE EXECUÇÃO DO APPLICATION STREAM

O Red Hat Enterprise Linux 8 Application Stream oferece outro conjunto de imagens de contêineres que você pode usar como base para a construção de seus contêineres. Estas imagens são construídas sobre imagens base padrão RHEL, com a maioria já atualizada como imagens UBI. Cada uma destas imagens inclui software adicional que você pode querer usar para ambientes específicos de tempo de execução.

Se você espera construir várias imagens que requerem, por exemplo, um software de tempo de execução php, você pode usar uma plataforma mais consistente para essas imagens, começando com uma imagem PHP Application Stream.

Aqui estão alguns exemplos de imagens de contêineres Application Stream construídos em imagens base UBI, que estão disponíveis no Registro da Red Hat (registry.access.redhat.com ou registry.redhat.io):

- **ubi8/php-72**: Plataforma PHP 7.2 para construção e execução de aplicações
- **ubi8/nodejs-10**: Plataforma Node.js 10 para construção e execução de aplicações. Usado por Node.js 10 Construções Fonte-To-Image
- **ubi8/ruby25**: Plataforma Ruby 2.5 para construção e execução de aplicações
- **ubi8/python-27**: Plataforma Python 2.7 para construção e execução de aplicações
- **ubi8/python-36**: Plataforma Python 3.6 para construção e execução de aplicações
- **ubi8/s2i-core**: Imagem base com bibliotecas e ferramentas essenciais utilizadas como base para imagens de construtores como perl, python, ruby, etc
- **ubi8/s2i-base**: Imagem base para a construção Fonte a Imagem

Como estas imagens UBI contêm o mesmo software básico que suas contrapartes de imagens legadas, você pode aprender sobre estas imagens no guia Using Using Red Hat Software Collections Container Images. Certifique-se de usar os nomes das imagens UBI para puxar essas imagens.

As imagens do contêiner RHEL 8 Application Stream são atualizadas cada vez que as imagens base RHEL 8 são atualizadas. Para a RHEL 7, estas mesmas imagens (referidas como imagens da Red Hat Software Collections) são atualizadas em um cronograma separado das atualizações de imagens base da RHEL (assim como as imagens relacionadas para Dotnet e DevTools). Pesquise no Catálogo de Recipientes da Red Hat para obter detalhes sobre qualquer uma destas imagens. Para mais informações sobre a programação de atualizações, consulte Atualizações de Imagens do Container da Red Hat.

4.7. OBTENDO O CÓDIGO FONTE DA IMAGEM DO CONTÊINER UBI

O código fonte está disponível para todas as imagens baseadas no Red Hat UBI na forma de recipientes para download. Antes de continuar, esteja atento aos recipientes-fonte da Red Hat:

- As imagens dos recipientes de origem não podem ser executadas, apesar de serem embaladas como recipientes. Para instalar imagens de recipientes-fonte da Red Hat em seu sistema, use o comando **skopeo command**, ao invés de usar o comando **podman pull**.
  - Use o comando **skopeo copy** para copiar uma imagem do recipiente de origem para um diretório em seu sistema local.
  - Use o comando **skopeo inspect** para inspecionar a imagem do recipiente de origem.
- Para obter mais detalhes sobre o comando **skopeo**, consulte a Seção 1.5. Usando o skopeo para trabalhar com registros de contêineres.
- As imagens dos recipientes de origem são nomeadas com base nos recipientes binários que eles representam. Por exemplo, para um recipiente RHEL UBI 8 padrão em particular **registry.access.redhat.com/ubi8:8.1-397** anexar **-source** para obter a imagem do recipiente de origem (**registry.access.redhat.com/ubi8:8.1-397-source**).
- Uma vez copiada uma imagem do recipiente de origem para um diretório local, você pode usar uma combinação dos comandos **tar**, **gzip**, e **rpm** para trabalhar com esse conteúdo.
- Pode levar várias horas após o lançamento de uma imagem de contêiner para que o contêiner de origem associado esteja disponível.

Procedimento

1. Use o comando **skopeo copy** para copiar a imagem do recipiente de origem para um diretório local:

```
$ skopeo copy \
docker://registry.access.redhat.com/ubi8:8.1-397-source \
dir:$HOME/TEST
...
Copying blob 477bc8106765 done
Copying blob c438818481d3 done
Copying blob 26fe858c966c done
Copying blob ba4b5f020b99 done
Copying blob f7d970ccd456 done
Copying blob ade06f94b556 done
Copying blob cc56c782b513 done
Copying blob dcf9396fdada done
Copying blob feb6d2ae2524 done
Copying config dd4cd669a4 done
Writing manifest to image destination
Storing signatures
```

2. Use o comando **skopeo inspect** para inspecionar a imagem do recipiente de origem:

```
$ skopeo inspect dir:$HOME/TEST
{
    "Digest":
"sha256:7ab721ef3305271bbb629a6db065c59bbeb87bc53e7cbf88e2953a1217ba7322",
    "RepoTags": [],
    "Created": "2020-02-11T12:14:18.612461174Z",
    "DockerVersion": "",
    "Labels": null,
    "Architecture": "amd64",
```

```
    "Os": "linux",
    "Layers": [
        "sha256:1ae73d938ab9f11718d0f6a4148eb07d38ac1c0a70b1d03e751de8bf3c2c87fa",
        "sha256:9fe966885cb8712c47efe5ecc2eaa0797a0d5ffb8b119c4bd4b400cc9e255421",
        "sha256:61b2527a4b836a4efbb82dfd449c0556c0f769570a6c02e112f88f8bbcd90166",
        ...
        "sha256:cc56c782b513e2bdd2cc2af77b69e13df4ab624ddb856c4d086206b46b9b9e5f",
        "sha256:dcf9396fdada4e6c1ce667b306b7f08a83c9e6b39d0955c481b8ea5b2a465b32",

"sha256:feb6d2ae252402ea6a6fca8a158a7d32c7e4572db0e6e5a5eab15d4e0777951e"
    ],
    "Env": null
}
```

3. Para desarmar todo o conteúdo, digite:

```
$ cd $HOME/TEST
$ for f in $(ls); do tar xvf $f; done
```

4. Para verificar os resultados, digite:

```
$ find blobs/ rpm_dir/
blobs/
blobs/sha256
blobs/sha256/10914f1fff060ce31388f5ab963871870535aaaa551629f5ad182384d60fdf82
rpm_dir/
rpm_dir/gzip-1.9-4.el8.src.rpm
```

5. Comece a examinar e utilizar o conteúdo.

4.8. RECURSOS ADICIONAIS

- Os parceiros e clientes da Red Hat podem solicitar novos recursos, incluindo solicitações de pacotes, preenchendo um ticket de suporte através de métodos padrão. Os clientes que não são da Red Hat não recebem suporte, mas podem solicitar através da Red Hat Bugzilla padrão para o produto RHEL apropriado. Para mais informações, consulte a fila Red Hat Bugzilla

- Os parceiros e clientes da Red Hat podem arquivar bilhetes de suporte através de métodos padrão ao executar o UBI em uma plataforma compatível com a Red Hat (OpenShift/RHEL). A equipe de suporte da Red Hat guiará os parceiros e clientes. Para mais informações, consulte Abrir um caso de suporte

CAPÍTULO 5. FUNCIONAMENTO DE SKOPEO E BUILDAH EM UM CONTÊINER

Com Skopeo, você pode inspecionar imagens em um registro remoto sem ter que baixar a imagem inteira com todas as suas camadas. Você também pode usar o Skopeo para copiar imagens, assinar imagens, sincronizar imagens, e converter imagens através de diferentes formatos e compressões de camadas.

O Buildah facilita a construção de imagens de contêineres OCI. Com o Buildah, você pode criar um container funcional, seja do zero ou usando uma imagem como ponto de partida. Você pode criar uma imagem a partir de um contêiner funcional ou através das instruções em um Dockerfile. Você pode montar e desmontar o sistema de arquivos raiz de um contêiner funcional.

Razões para executar Buildah e Skopeo em um contêiner:

- **Skopeo**: Você pode executar um sistema CI/CD dentro da Kubernetes ou usar o OpenShift para construir suas imagens de contêineres, e possivelmente distribuir essas imagens em diferentes registros de contêineres. Para integrar o Skopeo em um fluxo de trabalho Kubernetes, você precisa executá-lo em um contêiner.

- **Buildah**: Você quer construir imagens OCI/container dentro de um sistema Kubernetes ou OpenShift CI/CD que estão constantemente construindo imagens. Anteriormente, as pessoas usavam uma tomada Docker para se conectar ao motor do contêiner e executar um comando **docker build**. Isto era o equivalente a dar acesso root ao sistema sem exigir uma senha que não é segura. Por este motivo, a Red Hat recomenda o uso do Buildah em um contêiner.

- **Both**: Você está rodando um sistema operacional antigo no host, mas quer rodar a última versão do Skopeo, Buildah, ou ambos. A solução é rodar o Buildah em um container. Por exemplo, isto é útil para rodar a última versão do Skopeo, Buildah, ou ambas fornecidas no RHEL 8 em um host de containers RHEL 7 que não tem acesso às versões mais novas nativamente.

- **Both**: Uma restrição comum em ambientes HPC é que não é permitido aos usuários não-rootores instalar pacotes no host. Quando você executa Skopeo, Buildah, ou ambos em um container, você pode executar estas tarefas específicas como um usuário não-root.

5.1. FUNCIONAMENTO DO SKOPEO EM UM CONTÊINER

Este procedimento demonstra como inspecionar a imagem de um contêiner remoto usando o Skopeo. Rodar o Skopeo em um contêiner significa que o sistema de arquivos raiz do contêiner é isolado do sistema de arquivos raiz do hospedeiro. Para compartilhar ou copiar arquivos entre o host e o container, você tem que montar arquivos e diretórios.

Procedimento

1. Entrar no registro.redhat.io

```
$ podman login registry.redhat.io
Username: myuser@mycompany.com
Password: ***********
Login Succeeded!
```

2. Obtenha a imagem do recipiente **registry.redhat.io/rhel8/skopeo**:

```
$ podman pull registry.redhat.io/rhel8/skopeo
```

3. Inspecione uma imagem de contêiner remoto **registry.access.redhat.com/ubi8/ubi** usando Skopeo:

```
$ podman run --rm registry.redhat.io/rhel8/skopeo skopeo inspect
docker://registry.access.redhat.com/ubi8/ubi
{
    "Name": "registry.access.redhat.com/ubi8/ubi",
    ...
    "Labels": {
        "architecture": "x86_64",
        ...
        "name": "ubi8",
        ...
        "summary": "Provides the latest release of Red Hat Universal Base Image 8.",
        "url":
"https://access.redhat.com/containers/#/registry.access.redhat.com/ubi8/images/8.2-347",
        ...
    },
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
    ...
    ],
    "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "container=oci"
    ]
}
```

A opção **--rm** remove a imagem **registry.redhat.io/rhel8/skopeo** após a saída do contêiner.

Recursos adicionais

- Para mais informações sobre como executar o Skopeo em um contêiner, veja o artigo de Valentin Rothberg " Como executar o Skopeo em um contêiner ".

5.2. EXECUTANDO O SKOPEO EM UM CONTÊINER USANDO CREDENCIAIS

O trabalho com registros de contêineres requer uma autenticação para acessar e alterar dados. O Skopeo suporta várias maneiras de especificar as credenciais.

Com esta abordagem você pode especificar as credenciais na linha de comando usando a opção **--cred USERNAME[:PASSWORD]**.

Procedimento

- Inspecione uma imagem de contêiner remoto usando Skopeo contra um registro bloqueado:

```
$ podman run --rm registry.redhat.io/rhel8/skopeo inspect --creds $USER:$PASSWORD
docker://$IMAGE
```

Recursos adicionais

- Para mais informações sobre como executar o Skopeo em um contêiner, veja o artigo de Valentin Rothberg " Como executar o Skopeo em um contêiner ".

5.3. FUNCIONAMENTO DO SKOPEO EM UM CONTÊINER UTILIZANDO AUTHFILES

Você pode usar um arquivo de autenticação (authfile) para especificar as credenciais. O comando **skopeo login** faz o login no registro específico e armazena o token de autenticação no authfile. A vantagem do uso de authfiles é evitar a necessidade de inserir repetidamente as credenciais.

Ao rodar no mesmo host, todas as ferramentas de contêineres como Skopeo, Buildah, e Podman compartilham o mesmo authfile. Ao rodar o Skopeo em um contêiner, você tem que compartilhar o authfile no host, montando o authfile no contêiner, ou você tem que reautenticar dentro do contêiner.

Procedimento

- Inspecione uma imagem de contêiner remoto usando Skopeo contra um registro bloqueado:

```
$ podman run --rm -v $AUTHFILE:/auth.json registry.redhat.io/rhel8/skopeo inspect
docker://$IMAGE
```

  A opção **-v $AUTHFILE:/auth.json** volume - monta um authfile em /auth.json dentro do contêiner. Skopeo pode agora acessar os tokens de autenticação no authfile no host e obter acesso seguro ao registro.

Outros comandos Skopeo funcionam de forma semelhante, por exemplo:

- Use o comando **skopeo-copy** para especificar as credenciais na linha de comando para a imagem de origem e destino usando as opções **--source-creds** e **--dest-creds**. Ele também lê o arquivo /**auth.json** authfile.

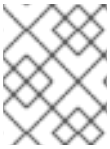
- Se você quiser especificar arquivos automáticos separados para a imagem de origem e destino, use as opções **--source-authfile** e **--dest-authfile** e monte esses arquivos automáticos do hospedeiro no contêiner.

Recursos adicionais

- Para mais informações sobre como executar o Skopeo em um contêiner, veja o artigo de Valentin Rothberg " Como executar o Skopeo em um contêiner ".

5.4. CÓPIA DE IMAGENS DE CONTÊINERES DE OU PARA O ANFITRIÃO

Skopeo, Buildah e Podman compartilham o mesmo armazenamento local de imagens de contêineres. Se você quiser copiar os contêineres para ou do armazenamento de contêineres anfitrião, você precisa montá-los dentro do contêiner Skopeo.



NOTA

O caminho para o armazenamento do recipiente hospedeiro difere entre usuários root (**/var/lib/containers/storage**) e não-root (**$HOME/.local/share/containers/storage**).

Procedimento

1. Copie a imagem **registry.access.redhat.com/ubi8/ubi** para o armazenamento local do seu container:

```
$ podman run --privileged --rm -v
$HOME/.local/share/containers/storage:/var/lib/containers/storage
registry.redhat.io/rhel8/skopeo skopeo copy docker://registry.access.redhat.com/ubi8/ubi
containerss-storage:registry.access.redhat.com/ubi8/ubi
```

   - A opção **--privileged** desabilita todos os mecanismos de segurança. A Red Hat recomenda o uso desta opção somente em ambientes confiáveis.
   - Para evitar desativar os mecanismos de segurança, exportar as imagens para um tarball ou qualquer outro transporte de imagens baseado no caminho e montá-las no contêiner Skopeo:
     - **$ podman save --format oci-archive -o oci.tar $IMAGE**
     - **$ podman run --rm -v oci.tar:/oci.tar registry.redhat.io/rhel8/skopeo copy oci-archive:/oci.tar $DESTINATION**

2. Para listar imagens em armazenamento local:

```
$ podman images
REPOSITORY                              TAG     IMAGE ID      CREATED       SIZE
registry.access.redhat.com/ubi8/ubi     latest  ecbc6f53bba0  8 weeks ago   211 MB
```

Recursos adicionais

- Para mais informações sobre como executar o Skopeo em um contêiner, veja o artigo de Valentin Rothberg " Como executar o Skopeo em um contêiner ".

5.5. FUNCIONANDO BUILDAH EM UM CONTAINER

O procedimento demonstra como executar o Buildah em um container e criar um container funcional com base em uma imagem.

Procedimento

1. Entrar no registro.redhat.io

```
$ podman login registry.redhat.io
Username: myuser@mycompany.com
Password: ***********
Login Succeeded!
```

2. Puxe e execute a imagem **registry.redhat.io/rhel8/buildah**:

```
# podman run --rm --device /dev/fuse -it registry.redhat.io/rhel8/buildah  /bin/bash
```

   - A opção **--rm** remove a imagem **registry.redhat.io/rhel8/buildah** após a saída do contêiner.
   - A opção **--device** adiciona um dispositivo hospedeiro ao contêiner.

3. Criar um novo container usando uma imagem **registry.access.redhat.com/ubi8**:

```
# buildah --storage-opt=overlay.mount_program=/usr/bin/fuse-overlayfs from
registry.access.redhat.com/ubi8
...
ubi8-working-container
```

- A opção **--storage-opt** define o driver de armazenamento. Esta opção anula todas as opções configuradas em **/etc/containers/storage.conf** e **STORAGE_OPTS** variável de ambiente.

- O **/usr/bin/fuse-overlayfs** é uma implementação do FUSE (Filesystem in Userspace) e permite que usuários não root criem seus sistemas de arquivos sem modificar o código do kernel.

4. Execute o comando **ls** / dentro do contêiner **ubi8-working-container**:

```
# buildah --storage-opt=overlay.mount_program=/usr/bin/fuse-overlayfs run --
isolation=chroot ubi8-working-container ls /
bin boot dev etc home lib lib64 lost+found media mnt opt proc root run sbin srv
```

5. Para listar todas as imagens em um depósito local, entre:

```
# buildah images
REPOSITORY                       TAG      IMAGE ID       CREATED       SIZE
registry.access.redhat.com/ubi8  latest   ecbc6f53bba0   5 weeks ago   211 MB
```

6. Para listar os recipientes de trabalho e suas imagens de base, entre:

```
# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER NAME
0aaba7192762     *     ecbc6f53bba0 registry.access.redhat.com/ub... ubi8-working-container
```

7. Para empurrar a imagem **registry.access.redhat.com/ubi8** para um registro local localizado em **registry.example.com**:

```
# buildah push ecbc6f53bba0 registry.example.com:5000/ubi8/ubi
```

Recursos adicionais

- Para mais informações sobre como executar o Buildah em um contêiner, veja as Melhores práticas para executar o Buildah em um artigo de Daniel Walsh sobre contêineres.

CAPÍTULO 6. EXECUÇÃO DE IMAGENS ESPECIAIS DE CONTÊINERES

Use este capítulo para aprender sobre alguns tipos especiais de imagens de recipientes. Estas incluem:

- **Toolbox**: Ao invés de sobrecarregar um sistema host instalando ferramentas necessárias para depurar problemas ou monitorar recursos, você pode executar o comando **toolbox**. A Toolbox inicia uma imagem de container **support-tools** que contém ferramentas que você pode usar para executar relatórios ou diagnosticar problemas no host.
- **Runlabels**: Algumas imagens de contêineres têm rótulos incorporados que lhe permitem executar esses contêineres com opções e argumentos pré-definidos. O comando **podman container runlabel <label>**, permite executar o comando definido naquele **<label>** para a imagem do contêiner. Os rótulos suportados são **install**, **run** e **uninstall**.

6.1. SOLUÇÃO DE PROBLEMAS DE CONTENTORES COM CAIXA DE FERRAMENTAS

Em vez de instalar ferramentas de solução de problemas diretamente em seu sistema RHEL 8, o utilitário **toolbox** oferece uma maneira de adicionar temporariamente essas ferramentas, e depois descartá-las facilmente quando você estiver pronto. O utilitário **toolbox** funciona por:

- Trazendo a imagem **registry.redhat.io/rhel8/support-tools** para seu sistema local.
- Iniciando um recipiente a partir da imagem, depois rodando uma concha dentro do recipiente a partir do qual você pode acessar o sistema hospedeiro.

O recipiente **support-tools** permite que você o faça:

- Executar comandos que não podem ser instalados no sistema hospedeiro, tais como **sosreport**, **strace**, ou **tcpdump**, de uma forma que lhes permita agir no sistema hospedeiro.
- Instale mais software dentro do contêiner para usar no sistema host.
- Descarte o recipiente quando estiver pronto.

O seguinte ilustra uma típica sessão **toolbox**.

Procedimento

1. Certifique-se de que os pacotes **toolbox** e **podman** estejam instalados:

   ```
   # lista de módulos de yum - ferramentas para contêineres
   ```

   Para instalar o conjunto completo de ferramentas de contêineres, digite:

   ```
   # Módulo yum instala ferramentas para container -y
   ```

2. Execute o comando da caixa de ferramentas para puxar e executar a imagem **support-tools** (inserindo suas credenciais do Portal do Cliente da Red Hat quando solicitado):

   ```
   # toolbox
   Trying to pull registry.redhat.io/rhel8/support-tools...
   ...
   ```

```
Would you like to authenticate to registry: 'registry.redhat.io' and try again? [y/N] y
Username: johndoe
Password: **************
Login Succeeded!
Trying to pull registry.redhat.io/rhel8/support-tools...Getting image source signatures
...
Storing signatures
30e261462851238d38f4ef2afdaf55f1f8187775c5ca373b43e0f55722faaf97
Spawning a container 'toolbox-root' with image 'registry.redhat.io/rhel8/support-tools'
Detected RUN label in the container image. Using that as the default...
command: podman run -it --name toolbox-root --privileged --ipc=host --net=host --pid=host -e
HOST=/host -e NAME=toolbox-root -e IMAGE=registry.redhat.io/rhel8/support-tools:latest -v
/run:/run -v /var/log:/var/log -v /etc/machine-id:/etc/machine-id -v /etc/localtime:/etc/localtime -
v /:/host registry.redhat.io/rhel8/support-tools:latest
```

3. Abra uma concha de bash para executar comandos dentro do contêiner:

```
# bash-4.4#
```

4. De dentro do container, o sistema de arquivo raiz no host está disponível no diretório **/host**. Os outros diretórios mostrados estão todos dentro do contêiner.

```
# ls /
bin  dev  home  lib lost+found  mnt  proc  run   srv  tmp  var
boot  etc  host  lib64 media     opt  root  sbin  sys  usr
```

5. Tente executar um comando dentro de seu contêiner. O comando **sosreport** permite que você gere informações sobre seu sistema para enviar ao suporte da Red Hat:

```
bash-4.4# sosreport

sosreport (version 3.6)
This command will collect diagnostic and configuration information from
this Red Hat Enterprise Linux system and installed applications.

An archive containing the collected information will be generated in
/host/var/tmp/sos.u82evisb and may be provided to a Red Hat support
representative.
...
Press ENTER to continue, or CTRL-C to quit.   <Press ENTER>
...
Your sosreport has been generated and saved in:
  /host/var/tmp/sosreport-rhel81beta-12345678-2019-10-29-pmgjncg.tar.xz
The checksum is: c4e1fd3ee45f78a17afb4e45a05842ed
Please send this file to your support representative.
```

Note que o comando **sosreport** salva o relatório para o anfitrião (**/host/var/tmp/sosreport-<ID>**).

6. Instale um pacote de software dentro do recipiente, para adicionar ferramentas que ainda não estejam no recipiente. Por exemplo, para obter um núcleo de um processo em execução no host, instale os pacotes **procps** e **gcore**, use **ps** para obter o ID do processo de um daemon em execução, depois use **gcore** para obter um núcleo de um dump:

```
bash-4.4# yum install procps gdb -y
```

```
bash-4.4# ps -ef | grep chronyd
994        809     1  0 Oct28 ?        00:00:00 /usr/sbin/chronyd
bash-4.4# gcore -o /host/tmp/chronyd.core 809
Missing separate debuginfo for target:/usr/sbin/chronyd
Try: dnf --enablerepo='*debug*' install /usr/lib/debug/.build-
id/96/0789a8a3bf28932b093e94b816be379f16a56a.debug
...
Saved corefile /host/tmp/chronyd.core.809
[Inferior 1 (process 809) detached]
```

7. Para deixar o recipiente e voltar para o anfitrião, digite **exit**. O arquivo é salvo em **/host/tmp/chronyd.core.809** e está disponível em **/tmp/chronyd.core.809** no host.

8. Para remover a caixa de ferramentas-root container, digite:

```
# podman rm toolbox-root
```

Você pode alterar o registro, a imagem ou o nome do recipiente utilizado pela caixa de ferramentas, acrescentando o seguinte:

- REGISTRY: Alterar o registro a partir do qual a imagem da caixa de ferramentas é retirada. Por exemplo **REGISTRY=registry.example.com**
- IMAGE: Mude a imagem que é utilizada. Por exemplo, **IMAGE=mysupport-tools**
- TOOLBOX_NAME: Alterar o nome atribuído ao recipiente em funcionamento. Por exemplo, **TOOLBOX_NAME=mytoolbox**

Na próxima vez que você executar **toolbox**, serão utilizados os novos valores do arquivo **.toolboxrc**.

6.1.1. Privilégios de abertura para o anfitrião

Quando você executa outros comandos de dentro do contêiner **support-tools** (ou de qualquer contêiner privilegiado), eles podem se comportar de forma diferente do que quando executados em um contêiner não-privilegiado. Embora **sosreport** possa dizer quando ele está rodando em um contêiner, outros comandos precisam ser ditos para agir no sistema hospedeiro (o diretório **/host**). Aqui estão exemplos de características que podem ou não estar abertas para o host a partir de um contêiner:

- Privileges: Um container privilegiado (**--privileged**) executa aplicações como usuário root no host por padrão. O contêiner tem esta capacidade porque roda com um contexto de segurança **unconfined_t** SELinux. Assim você pode, por exemplo, excluir arquivos e diretórios montados do host que são de propriedade do usuário root.
- Process tables: Ao contrário de um contêiner normal que só vê os processos rodando dentro do contêiner, rodar um comando **ps -e** dentro de um contêiner privilegiado (com o conjunto **--pid=host**) permite que você veja cada processo rodando no host. Você pode passar um ID de processo do host para comandos que rodam no contêiner privilegiado (por exemplo, **kill <PID>**). Com alguns comandos, entretanto, podem ocorrer problemas de permissões quando tentam acessar processos a partir do contêiner.
- Network interfaces: Por padrão, um recipiente tem apenas uma interface de rede externa e uma interface de rede de loopback. Com interfaces de rede abertas para o host (**--net=host**), você pode acessar essas interfaces de rede diretamente de dentro do contêiner.

- **Inter-process communications**: A instalação IPC no host é acessível de dentro do contêiner privilegiado. Você pode executar comandos como **ipcs** para ver informações sobre filas de mensagens ativas, segmentos de memória compartilhada e conjuntos de semáforos no host.

6.2. FUNCIONAMENTO DE CONTÊINERES COM RUNLABELS

Algumas imagens da Red Hat incluem etiquetas que fornecem linhas de comando pré-definidas para trabalhar com essas imagens. Usando o comando **podman container runlabel <label>**, você pode dizer a **podman** para executar o comando definido naquele **<label>** para a imagem. As runlabels existentes incluem:

- **install**: Configura o sistema hospedeiro antes de executar a imagem. Normalmente, isto resulta na criação de arquivos e diretórios no host que o container pode acessar quando for executado mais tarde.

- **run**: Identifica as opções de linha de comando podman a serem usadas ao executar o contêiner. Tipicamente, as opções abrem privilégios no host e montam o conteúdo do host que o container precisa para permanecer permanentemente no host.

- **uninstall**: Limpa o sistema hospedeiro depois que você terminar de operar o contêiner.

As imagens da Red Hat que têm um ou mais runlabels incluem as imagens **rsyslog** e **support-tools**. O procedimento a seguir ilustra como usar essas imagens.

6.2.1. Rodando rsyslog com runlabels

A imagem do contêiner **rhel8/rsyslog** é feita para executar uma versão em contêiner do daemon **rsyslogd**. Dentro da imagem **rsyslog** estão **install**, **run** e **uninstall** runlabels. Os seguintes passos de procedimento são realizados através da instalação, execução e desinstalação da imagem **rsyslog**:

Procedimento

1. Puxe a imagem **rsyslog**:

```
# podman pull registry.redhat.io/rhel8/rsyslog
```

2. Exibir (mas ainda não rodar) o runlabel **install** para **rsyslog**:

```
# podman container runlabel install --display rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/install.sh
```

   Isto mostra que o comando abrirá privilégios para o host, montará o sistema de arquivos raiz do host em **/host** no container, e executará um script **install.sh**.

3. Execute o runlabel **install** para **rsyslog**:

```
# podman container runlabel install rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/install.sh
Creating directory at /host//etc/pki/rsyslog
Creating directory at /host//etc/rsyslog.d
```

```
Installing file at /host//etc/rsyslog.conf
Installing file at /host//etc/sysconfig/rsyslog
Installing file at /host//etc/logrotate.d/syslog
```

Isto cria arquivos no sistema hospedeiro que a imagem **rsyslog** utilizará mais tarde.

4. Mostrar o runlabel **run** para **rsyslog**:

```
# podman container runlabel run --display rhel8/rsyslog
command: podman run -d --privileged --name rsyslog --net=host --pid=host -v
/etc/pki/rsyslog:/etc/pki/rsyslog -v /etc/rsyslog.conf:/etc/rsyslog.conf -v
/etc/sysconfig/rsyslog:/etc/sysconfig/rsyslog -v /etc/rsyslog.d:/etc/rsyslog.d -v /var/log:/var/log
-v /var/lib/rsyslog:/var/lib/rsyslog -v /run:/run -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -e IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e
NAME=rsyslog --restart=always registry.redhat.io/rhel8/rsyslog:latest /bin/rsyslog.sh
```

Isto mostra que o comando abre privilégios para o host e monta arquivos e diretórios específicos do host dentro do container, quando lança o container **rsyslog** para rodar o daemon **rsyslogd**.

5. Execute o runlabel **run** para **rsyslog**:

```
# podman container runlabel run rhel8/rsyslog
command: podman run -d --privileged --name rsyslog --net=host --pid=host -v
/etc/pki/rsyslog:/etc/pki/rsyslog -v /etc/rsyslog.conf:/etc/rsyslog.conf -v
/etc/sysconfig/rsyslog:/etc/sysconfig/rsyslog -v /etc/rsyslog.d:/etc/rsyslog.d -v /var/log:/var/log
-v /var/lib/rsyslog:/var/lib/rsyslog -v /run:/run -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -e IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e
NAME=rsyslog --restart=always registry.redhat.io/rhel8/rsyslog:latest /bin/rsyslog.sh
28a0d719ff179adcea81eb63cc90fcd09f1755d5edb121399068a4ea59bd0f53
```

O recipiente **rsyslog** abre privilégios, monta o que precisa do anfitrião e executa o daemon **rsyslogd** em segundo plano (**-d**). O daemon **rsyslogd** começa a reunir mensagens de registro e direciona as mensagens para arquivos no diretório **/var/log**.

6. Mostrar o runlabel **uninstall** para **rsyslog**:

```
# podman container runlabel uninstall --display rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/uninstall.sh
```

7. Execute o runlabel **uninstall** para **rsyslog**:

```
# podman container runlabel uninstall rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/uninstall.sh
```

Neste caso, o script **uninstall.sh** apenas remove o arquivo **/etc/logrotate.d/syslog**. Note que ele não limpa os arquivos de configuração.

6.2.2. Executando ferramentas de apoio com runlabels

A imagem do recipiente **rhel8/support-tools** é feita para executar ferramentas como **sosreport** e **sos-**

collector para ajudá-lo a analisar seu sistema hospedeiro. Para simplificar a execução da imagem **support-tools**, ela inclui um runlabel **run**. O procedimento a seguir descreve como executar a imagem **support-tools**:

Procedimento

1. Puxe a imagem **support-tools**:

```
# podman pull registry.redhat.io/rhel8/support-tools
```

2. Exibir (mas ainda não rodar) o runlabel **run** para **support-tools**:

```
# podman container runlabel run --display rhel8/support-tools
command: podman run -it --name support-tools --privileged --ipc=host --net=host --pid=host -
e HOST=/host -e NAME=support-tools -e IMAGE=registry.redhat.io/rhel8/support-tools:latest
-v /run:/run -v /var/log:/var/log -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -v /:/host registry.redhat.io/rhel8/support-tools:latest
```

Isto mostra que o comando monta diretórios e abre privilégios e espaços de nomes (ipc, net, e pid) para o sistema hospedeiro. Ele atribui o sistema de arquivos raiz do host ao diretório /**host** no container.

3. Executar o runlabel **run** para ferramentas de apoio:

```
# podman container runlabel run rhel8/support-tools
command: podman run -it --name support-tools --privileged --ipc=host --net=host --pid=host -
e HOST=/host -e NAME=support-tools -e IMAGE=registry.redhat.io/rhel8/support-tools:latest
-v /run:/run -v /var/log:/var/log -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -v /:/host registry.redhat.io/rhel8/support-tools:latest
bash-4.4#
```

Isto abre uma casca de bash dentro do contêiner **support-tools**. Agora você pode executar relatórios ou ferramentas de depuração contra o sistema hospedeiro (/**host**).

4. Para deixar o recipiente e voltar para o anfitrião, digite **exit**.

```
# saída
```

CAPÍTULO 7. PORTANDO CONTAINERS PARA OPENSHIFT USANDO PODMAN

Este capítulo descreve como gerar descrições portáteis de recipientes e cápsulas usando o formato YAML (YAML Ain't Markup Language). O YAML é um formato de texto usado para descrever os dados de configuração.

Os arquivos da YAML são:

- Legível.
- Fácil de gerar.
- Portátil entre ambientes (por exemplo, entre RHEL e OpenShift).
- Portátil entre linguagens de programação.
- Conveniente de usar (não há necessidade de adicionar todos os parâmetros à linha de comando).

Razões para usar os arquivos YAML:

1. Você pode executar novamente um conjunto orquestrado local de recipientes e cápsulas com o mínimo de entradas necessárias, o que pode ser útil para o desenvolvimento iterativo.
2. Você pode operar os mesmos recipientes e cápsulas em outra máquina. Por exemplo, para executar uma aplicação em um ambiente OpenShift e para garantir que a aplicação esteja funcionando corretamente. Você pode usar o comando **podman generate kube** para gerar um arquivo Kubernetes YAML. Depois, você pode usar o comando **podman play** para testar a criação de pods e containers em seu sistema local antes de transferir os arquivos YAML gerados para o ambiente Kubernetes ou OpenShift. Usando o comando **podman play**, você também pode recriar os pods e containers originalmente criados em ambientes OpenShift ou Kubernetes.

7.1. GERAÇÃO DE UM ARQUIVO KUBERNETES YAML USANDO PODMAN

Este procedimento descreve como criar uma cápsula com um recipiente e gerar o arquivo Kubernetes YAML usando o comando **podman generate kube**.

Pré-requisitos

- A cápsula foi criada. Para obter detalhes, consulte Criando cápsulas.

Procedimento

1. Liste todas as cápsulas e recipientes associados a elas:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                  COMMAND    CREATED
STATUS                  PORTS  NAMES              POD
5df5c48fea87  registry.access.redhat.com/ubi8/ubi:latest  /bin/bash  Less than a second ago
```

```
Up Less than a second ago         myubi               223df6b390b4
3afdcd93de3e  k8s.gcr.io/pause:3.1                          Less than a second ago  Up Less
than a second ago         223df6b390b4-infra  223df6b390b4
```

2. Use o nome ou ID da cápsula para gerar o arquivo Kubernetes YAML:

```
$ podman gera kube mypod > mypod.yaml
```

Observe que o comando **podman generate** não reflete nenhum volume lógico do Logical Volume Manager (LVM) ou volumes físicos que possam estar presos ao contêiner.

3. Exibir o arquivo **mypod.yaml**:

```
$ cat mypod.yaml
# Generation of Kubernetes YAML is still under development!
#
# Save the output of this file and use kubectl create -f to import
# it into Kubernetes.
#
# Created with podman-1.6.4
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2020-06-09T10:31:56Z"
  labels:
app: mypod
  name: mypod
spec:
  containers:
  - command:
        - /bin/bash
        env:
        - name: PATH
            value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
        - name: TERM
            value: xterm
        - name: HOSTNAME
        - name: container
            value: oci
        image: registry.access.redhat.com/ubi8/ubi:latest
        name: myubi
        resources: {}
        securityContext:
            allowPrivilegeEscalation: true
            capabilities: {}
            privileged: false
            readOnlyRootFilesystem: false
        tty: true
        workingDir: /
status: {}
```

4. Para parar a cápsula **mypod**:

```
$ podman podman stop mypod
```

5. Liste todas as cápsulas e recipientes associados a elas:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                           COMMAND    CREATED           STATUS
PORTS   NAMES              POD ID        PODNAME
12fc1ada3d4f  registry.redhat.io/ubi8/ubi:latest  /bin/bash  About a minute ago  Exited (0) 7
seconds ago          myubi              8a4e6527ac9d  mypod

8bb1daaf81fe  k8s.gcr.io/pause:3.2                      About a minute ago  Exited (0) 7 seconds
ago          8a4e6527ac9d-infra  8a4e6527ac9d  mypod
```

   Aqui, a cápsula **mypod** e o recipiente **myubi** estão em status de "Exilados".

6. Para remover a cápsula **mypod**:

```
$ podman pod rm mypod
8a4e6527ac9d2276e8a6b9c2670608866dbcb5da3efbd06f70ec2ecc88e247eb
```

   Note que a remoção da cápsula remove automaticamente todos os recipientes dentro dela.

7. Para verificar se todos os recipientes e cápsulas foram removidos:

```
$ podman ps
$ podman pod ps
```

Recursos adicionais

- A página do homem **podman-generate-kube**.
- Podman: Artigo de Brent Baude sobre a gestão de cápsulas e recipientes em um contêiner local em tempo real.

7.2. GERAÇÃO DE UM ARQUIVO KUBERNETES YAML EM AMBIENTE OPENSHIFT

No ambiente OpenShift, use o comando **oc create** para gerar os arquivos YAML que descrevem sua aplicação.

Procedimento

- Gerar o arquivo YAML para sua aplicação **myapp**:

```
$ oc create myapp --image=me/myapp:v1 -o yaml --dry-run > myapp.yaml
```

  O comando **oc create** cria e executa a imagem **myapp**. O objeto é impresso usando a opção **--dry-run** e redirecionado para o arquivo de saída **myapp.yaml**.

NOTA

No ambiente Kubernetes, você pode usar o comando **kubectl create** com as mesmas bandeiras.

7.3. INÍCIO DE RECIPIENTES E CÁPSULAS COM PODMAN

Com os arquivos YAML gerados, você pode iniciar automaticamente recipientes e cápsulas em qualquer ambiente. Observe que os arquivos YAML não devem ser gerados pelo Podman. O comando **podman play kube** permite que você recrie pods e containers com base no arquivo de entrada YAML.

Procedimento

1. Criar a cápsula e o recipiente a partir do arquivo **mypod.yaml**:

```
$ podman play kube mypod.yaml
Pod:
b8c5b99ba846ccff76c3ef257e5761c2d8a5ca4d7ffa3880531aec79c0dacb22
Container:
848179395ebd33dd91d14ffbde7ae273158d9695a081468f487af4e356888ece
```

2. Liste todas as cápsulas:

```
$ podman pod ps
POD ID        NAME    STATUS    CREATED         # OF CONTAINERS   INFRA ID
b8c5b99ba846  mypod   Running   19 seconds ago  2                 aa4220eaf4bb
```

3. Liste todas as cápsulas e recipientes associados a elas:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                  COMMAND    CREATED            STATUS
PORTS  NAMES              POD
848179395ebd  registry.access.redhat.com/ubi8/ubi:latest  /bin/bash  About a minute ago  Up
About a minute ago        myubi               b8c5b99ba846
aa4220eaf4bb  k8s.gcr.io/pause:3.1                        About a minute ago  Up About a
minute ago        b8c5b99ba846-infra  b8c5b99ba846
```

Os IDs das cápsulas do comando **podman ps** correspondem ao ID da cápsula do comando **podman pod ps**.

Recursos adicionais

- A página do homem **podman-play-kube**.
- Podman pode agora facilitar a transição para Kubernetes e o artigoCRI-O de Brent Baude.

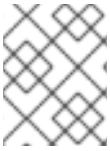
7.4. INÍCIO DE RECIPIENTES E CÁPSULAS EM AMBIENTE OPENSHIFT

Você pode usar o comando **oc create** para criar pods e containers no ambiente OpenShift.

Procedimento

- Criar um pod a partir do arquivo YAML no ambiente OpenShift:

```
$ oc create -f mypod.yaml
```



NOTA

No ambiente Kubernetes, você pode usar o comando **kubectl create** com as mesmas bandeiras.

CAPÍTULO 8. PORTANDO CONTÊINERES PARA O SISTEMA USANDO PODMAN

Podman (Pod Manager) é um motor de contêiner cheio de recursos que é uma simples ferramenta sem daemon. Podman fornece uma linha de comando comparável Docker-CLI que facilita a transição de outros motores de contêineres e permite o gerenciamento de cápsulas, contêineres e imagens. Não foi originalmente projetado para criar um sistema Linux completo ou gerenciar serviços para coisas como ordem de partida, verificação de dependência e recuperação de serviços falhados. Esse é o trabalho de um sistema de inicialização completo como o systemd. A Red Hat tornou-se líder na integração de containers com o systemd, de modo que containers em formato OCI e Docker construídos pela Podman possam ser gerenciados da mesma forma que outros serviços e recursos são gerenciados em um sistema Linux. Você pode usar o serviço de inicialização do systemd para trabalhar com pods e containers. Você pode usar o comando **podman generate systemd** para gerar um arquivo de unidade systemd para containers e pods.

Com os arquivos da unidade systemd, você pode:

- Configure um container ou cápsula para começar como um serviço de sistema.
- Definir a ordem na qual o serviço de contêineres funciona e verificar as dependências (por exemplo, certificando-se de que outro serviço esteja funcionando, que um arquivo esteja disponível ou que um recurso esteja montado).
- Controle o estado do sistema através do comando **systemctl**.

Este capítulo fornece informações sobre como gerar descrições portáteis de recipientes e cápsulas usando arquivos de unidade do sistema.

8.1. HABILITAÇÃO DE SERVIÇOS DE SISTEMA

Ao habilitar o serviço, você tem diferentes opções.

Procedimento

- Habilite o serviço:
  - Para ativar um serviço no início do sistema, não importa se o usuário está logado ou não, entre:

    ```
    # Systemctl enable <service>
    ```

    É necessário copiar os arquivos da unidade systemd para o diretório **/etc/systemd/system**.
  - Para iniciar um serviço no login do usuário e interrompê-lo no logout do usuário, entre:

    ```
    $ systemctl --utilizador habilitado <service>
    ```

    É necessário copiar os arquivos da unidade systemd para o diretório **$HOME/.config/systemd/user**.
  - Para permitir que os usuários iniciem um serviço no início do sistema e persistam ao longo da logout, entre:

    ```
    # loginctl enable-linger <username>
    ```

Recursos adicionais

- Para mais informações sobre os comandos **systemctl** e **loginctl**, digite **man systemctl** ou **man loginctl**, respectivamente.
- Para saber mais sobre a configuração de serviços com o systemd, consulte o capítulo Configurando configurações básicas do sistema chamado Gerenciando serviços com o systemd.

8.2. GERAÇÃO DE UM ARQUIVO DE UNIDADE DO SISTEMA USANDO PODMAN

Podman permite que o sistemad controle e gerencie os processos de contêineres. Você pode gerar um arquivo de unidade systemd para os contêineres e pods existentes usando o comando **podman generate systemd**. Recomenda-se usar **podman generate systemd** porque os arquivos de unidades gerados mudam freqüentemente (via atualizações para Podman) e o **podman generate systemd** garante que você obtenha a última versão dos arquivos de unidades.

Procedimento

1. Criar um recipiente (por exemplo, **myubi**):

```
$ podman create -d --name myubi registry.access.redhat.com/ubi8:latest top
0280afe98bb75a5c5e713b28de4b7c5cb49f156f1cce4a208f13fee2f75cb453
```

2. Use o nome ou ID do container para gerar o arquivo da unidade do sistema e direcione-o para o arquivo **~/.config/systemd/user/container-myubi.service**:

```
$ podman generate systemd --name myubi > ~/.config/systemd/user/container-myubi.service
```

Etapas de verificação

- Para exibir o conteúdo do arquivo de unidade do sistema gerado, entre:

```
$ cat ~/.config/systemd/user/container-myubi.service
# container-myubi.service
# autogenerated by Podman 2.0.0
# Tue Aug 11 10:51:04 CEST 2020

[Unit]
Description=Podman container-myubi.service
Documentation=man:podman-generate-systemd(1)
Wants=network.target
After=network-online.target

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
ExecStart=/usr/bin/podman start myubi
ExecStop=/usr/bin/podman stop -t 10 myubi
ExecStopPost=/usr/bin/podman stop -t 10 myubi
PIDFile=/run/user/1000/containers/overlay-
containers/0280afe98bb75a5c5e713b28de4b7c5cb49f156f1cce4a208f13fee2f75cb453/userdat
a/conmon.pid
```

```
KillMode=none
Type=forking

[Install]
WantedBy=multi-user.target default.target
```

- A linha **Restart=on-failure** define a política de reinício e instrui o sistema a reiniciar quando o serviço não puder ser iniciado ou interrompido de forma limpa, ou quando o processo sair fora de zero.
- A linha **ExecStart** descreve como iniciamos o contêiner.
- A linha **ExecStop** descreve como paramos e removemos o container.

Recursos adicionais

- Artigo de Valentin Rothberg sobre aexecução de contêineres com Podman e serviços de sistema compartilhável.

8.3. AUTO-GERAÇÃO DE UM ARQUIVO DE UNIDADE DO SISTEMA USANDO PODMAN

Por padrão, Podman gera um arquivo de unidade para containers ou cápsulas existentes. Você pode gerar mais arquivos de unidade portáteis do sistema usando o **podman generate systemd --new**. A bandeira **--new** instrui Podman a gerar arquivos unitários que criam, iniciam e removem containers.

Procedimento

1. Puxe a imagem que você deseja usar em seu sistema. Por exemplo, para puxar a imagem **busybox**:

```
# podman pull busybox:latest
```

2. Liste todas as imagens disponíveis em seu sistema:

```
# podman images
REPOSITORY                TAG     IMAGE ID      CREATED      SIZE
docker.io/library/busybox  latest  c7c37e472d31  3 weeks ago  1.45 MB
```

3. Crie o recipiente **busybox**:

```
# podman create --name busybox busybox:latest
1e12cf95e305435c0001fa7d4a14cf1d52f737c1118328937028c0bd2fdec5ca
```

4. Para verificar se o container foi criado, liste todos os containers:

```
# podman ps -a
CONTAINER ID  IMAGE                           COMMAND  CREATED       STATUS   PORTS
NAMES
1e12cf95e305  docker.io/library/busybox:latest  sh       7 seconds ago  Created          busybox
```

5. Gerar um arquivo de unidade do sistema para o contêiner **busybox**:

```
# podman generate systemd --new --files --name busybox
/root/container-busybox.service
```

6. Exibir o conteúdo do arquivo da unidade gerada **container-busybox.service** systemd:

```
# vim container-busybox.services

# container-busybox.service
# autogenerated by Podman 2.0.0-rc7
# Mon Jul 27 11:06:32 CEST 2020

[Unit]
Description=Podman container-busybox.service
Documentation=man:podman-generate-systemd(1)
Wants=network.target
After=network-online.target

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
ExecStartPre=/usr/bin/rm -f %t/container-busybox.pid %t/container-busybox.ctr-id
ExecStart=/usr/bin/podman run --conmon-pidfile %t/container-busybox.pid --cidfile
%t/container-busybox.ctr-id --cgroups=no-conmon -d --replace --name busybox
busybox:latest
ExecStop=/usr/bin/podman stop --ignore --cidfile %t/container-busybox.ctr-id -t 10
ExecStopPost=/usr/bin/podman rm --ignore -f --cidfile %t/container-busybox.ctr-id
PIDFile=%t/container-busybox.pid
KillMode=none
Type=forking

[Install]
WantedBy=multi-user.target default.target
```

Observe que os arquivos de unidade gerados usando a opção **--new** não esperam que os recipientes e as cápsulas existam. Portanto, eles executam o comando **podman run** ao iniciar o serviço (veja a linha **ExecStart**) em vez do comando **podman start**. Por exemplo, veja a seção 7.2. Gerando um arquivo de unidade systemd usando Podman .

- O comando **podman run** usa as seguintes opções de linha de comando:
  - A opção **--conmon-pidfile** aponta para um caminho para armazenar a identificação do processo para o processo **conmon** em execução no host. O processo **conmon** termina com o mesmo status de saída do contêiner, o que permite que o sistema informe o status correto do serviço e reinicie o contêiner, se necessário.
  - A opção **--cidfile** aponta para o caminho que armazena a identificação do contêiner.
  - O **%t** é o caminho para a raiz do diretório de tempo de execução, por exemplo **/run/user/$UserID**.
  - O **%n** é o nome completo do serviço.

7. Copie os arquivos da unidade para **/usr/lib/systemd/system** para instalá-los como usuário root:

```
# cp -Z container-busybox.service  /usr/lib/systemd/system
Created symlink /etc/systemd/system/multi-user.target.wants/container-busybox.service
```

```
/usr/lib/systemd/system/container-busybox.service.
Created symlink /etc/systemd/system/default.target.wants/container-busybox.service →
/usr/lib/systemd/system/container-busybox.service.
```

Recursos adicionais

- Melhoria da integração do sistema com o artigo Podman 2.0 de Valentin Rothberg e Dan Walsh.
- Para saber mais sobre a configuração de serviços com o systemd, consulte o capítulo Configurando configurações básicas do sistema chamado Gerenciando serviços com o systemd.

8.4. CONTAINERS DE PARTIDA AUTOMÁTICA USANDO SYSTEMD

Você pode controlar o estado do sistema e do gerente de serviços do sistema usando o comando **systemctl**. Esta seção mostra o procedimento geral sobre como habilitar, iniciar, parar o serviço como um usuário não-root. Para instalar o serviço como um usuário root, omitir a opção **--user**.

Procedimento

1. Configuração do gerente do sistema de recarga:

```
# systemctl -- daemon-reload do usuário
```

2. Habilite o serviço **container.service** e inicie-o no momento da inicialização:

```
# systemctl -- usuário habilita container.service
```

3. Para iniciar o serviço imediatamente:

```
# systemctl -- usuário iniciar container.service
```

4. Verifique o status do serviço:

```
$ systemctl --user status container.service
● container.service - Podman container.service
   Loaded: loaded (/home/user/.config/systemd/user/container.service; enabled; vendor
preset: enabled)
   Active: active (running) since Wed 2020-09-16 11:56:57 CEST; 8s ago
     Docs: man:podman-generate-systemd(1)
  Process: 80602 ExecStart=/usr/bin/podman run --conmon-pidfile
//run/user/1000/container.service-pid --cidfile //run/user/1000/container.service-cid -d ubi8-
minimal:>
  Process: 80601 ExecStartPre=/usr/bin/rm -f //run/user/1000/container.service-pid
//run/user/1000/container.service-cid (code=exited, status=0/SUCCESS)
 Main PID: 80617 (conmon)
   CGroup: /user.slice/user-1000.slice/user@1000.service/container.service
           ├─ 2870 /usr/bin/podman
           ├─80612 /usr/bin/slirp4netns --disable-host-loopback --mtu 65520 --enable-sandbox -
-enable-seccomp -c -e 3 -r 4 --netns-type=path /run/user/1000/netns/cni->
           ├─80614 /usr/bin/fuse-overlayfs -o
lowerdir=/home/user/.local/share/containers/storage/overlay/l/YJSPGXM2OCDZPLMLXJOW3N
RF6Q:/home/user/.local/share/contain>
```

```
          ├─80617 /usr/bin/conmon --api-version 1 -c
cbc75d6031508dfd3d78a74a03e4ace1732b51223e72a2ce4aa3bfe10a78e4fa -u
cbc75d6031508dfd3d78a74a03e4ace1732b51223e72>
          └─cbc75d6031508dfd3d78a74a03e4ace1732b51223e72a2ce4aa3bfe10a78e4fa
            └─80626 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1d
```

Você pode verificar se o serviço está habilitado usando o comando **systemctl is-enabled container.service**.

Etapas de verificação

- Liste os contêineres que estão funcionando ou já saíram:

```
# podman ps
CONTAINER ID  IMAGE                                  COMMAND  CREATED         STATUS
PORTS  NAMES
f20988d59920  registry.access.redhat.com/ubi8-minimal:latest  top      12 seconds ago  Up 11
seconds ago          funny_zhukovsky
```

NOTA

Para parar **container.service**, entre:

```
# systemctl -- contêiner.serviço de parada do usuário
```

Recursos adicionais

- Para mais informações sobre o comando **systemctl**, digite **man systemctl**.
- Para mais informações, veja o artigo de Valentin Rothberg " Correndo containers com Podman e serviços de sistema compartilhável ".
- Para saber mais sobre a configuração de serviços com o systemd, consulte o capítulo Configurando configurações básicas do sistema chamado Gerenciando serviços com o systemd.

8.5. VAGENS DE INICIALIZAÇÃO AUTOMÁTICA USANDO SYSTEMD

Você pode iniciar múltiplos containers como serviços de sistema. Note que o comando **systemctl** deve ser usado somente no pod e você não deve iniciar ou parar containers individualmente via **systemctl**, pois eles são gerenciados pelo serviço de pod junto com o infra-container interno.

Procedimento

1. Criar uma cápsula vazia, por exemplo, chamada **systemd-pod**:

```
$ podman pod create --name systemd-pod
11d4646ba41b1fffa51c108cbdf97cfab3213f7bd9b3e1ca52fe81b90fed5577
```

2. Liste todas as cápsulas:

```
$ podman pod ps
POD ID        NAME        STATUS   CREATED         # OF CONTAINERS  INFRA ID
```

```
11d4646ba41b  systemd-pod  Created  40 seconds ago  1              8a428b257111
11d4646ba41b1fffa51c108cbdf97cfab3213f7bd9b3e1ca52fe81b90fed5577
```

3. Criar dois recipientes na cápsula vazia. Por exemplo, para criar **container0** e **container1** em **systemd-pod**:

```
$ podman create --pod systemd-pod --name container0 registry.access.redhat.com/ubi8 top
$ podman create --pod systemd-pod --name container1 registry.access.redhat.com/ubi8 top
```

4. Liste todas as cápsulas e recipientes associados a elas:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                COMMAND  CREATED        STATUS
PORTS   NAMES              POD ID        PODNAME
24666f47d9b2  registry.access.redhat.com/ubi8:latest  top      3 minutes ago  Created
container0          3130f724e229  systemd-pod
56eb1bf0cdfe  k8s.gcr.io/pause:3.2                          4 minutes ago  Created
3130f724e229-infra  3130f724e229  systemd-pod
62118d170e43  registry.access.redhat.com/ubi8:latest  top      3 seconds ago  Created
container1          3130f724e229  systemd-pod
```

5. Gerar o arquivo da unidade do sistema para o novo módulo:

```
$ podman generate systemd --files --name systemd-pod
/home/user1/pod-systemd-pod.service
/home/user1/container-container0.service
/home/user1/container-container1.service
```

Observe que são gerados três arquivos de unidade do sistema, um para o módulo **systemd-pod** e dois para os recipientes **container0** e **container1**.

6. Exibir o arquivo da unidade **pod-systemd-pod.service**:

```
$ cat pod-systemd-pod.service
# pod-systemd-pod.service
# autogenerated by Podman 2.0.3
# Tue Jul 28 14:00:46 EDT 2020


[Unit]
Description=Podman pod-systemd-pod.service
Documentation=man:podman-generate-systemd(1)
Wants=network.target
After=network-online.target
Requires=container-container0.service container-container1.service
Before=container-container0.service container-container1.service

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
ExecStart=/usr/bin/podman start c852fbaba568-infra
ExecStop=/usr/bin/podman stop -t 10 c852fbaba568-infra
ExecStopPost=/usr/bin/podman stop -t 10 c852fbaba568-infra
PIDFile=/run/user/1000/containers/overlay-
containers/a7ff86382608add27a03ac2166d5d0164199f01eadf80b68b06a406c195105fc/userd
```

```
ata/conmon.pid
KillMode=none
Type=forking

[Install]
WantedBy=multi-user.target default.target
```

- A linha **Requires** na seção **[Unit]** define as dependências dos arquivos das unidades **container-container0.service** e **container-container1.service**. Ambos os arquivos unitários serão ativados.

- As linhas **ExecStart** e **ExecStop** na seção **[Service]** iniciam e param o infracontainer, respectivamente.

7. Exibir o arquivo da unidade **container-container0.service**:

```
$ cat container-container0.service
# container-container0.service
# autogenerated by Podman 2.0.3
# Tue Jul 28 14:00:46 EDT 2020

[Unit]
Description=Podman container-container0.service
Documentation=man:podman-generate-systemd(1)
Wants=network.target
After=network-online.target
BindsTo=pod-systemd-pod.service
After=pod-systemd-pod.service

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
ExecStart=/usr/bin/podman start container0
ExecStop=/usr/bin/podman stop -t 10 container0
ExecStopPost=/usr/bin/podman stop -t 10 container0
PIDFile=/run/user/1000/containers/overlay-
containers/12e85378f2854b8283f791974494a02aa6c92630d76d1050237839b61508a008/user
data/conmon.pid
KillMode=none
Type=forking

[Install]
WantedBy=multi-user.target default.target
```

- A linha **BindsTo** na seção **[Unit]** define a dependência do arquivo da unidade **pod-systemd-pod.service**

- As linhas **ExecStart** e **ExecStop** na seção **[Service]** iniciam e param o **container0** respectivamente.

8. Exibir o arquivo da unidade **container-container1.service**:

```
Container-container1.serviço de $ cat
```

9. Copiar todos os arquivos gerados para **$HOME/.config/systemd/user** para instalação como usuário não-rootal:

```
$ cp pod-systemd-pod.service container-container0.service container-container1.service
$HOME/.config/systemd/user
```

10. Habilite o serviço e comece com o login do usuário:

```
$ systemctl enable --user pod-systemd-pod.service
Created symlink /home/user1/.config/systemd/user/multi-user.target.wants/pod-systemd-
pod.service → /home/user1/.config/systemd/user/pod-systemd-pod.service.
Created symlink /home/user1/.config/systemd/user/default.target.wants/pod-systemd-
pod.service → /home/user1/.config/systemd/user/pod-systemd-pod.service.
```

Note que o serviço pára no logout do usuário.

Etapas de verificação

- Verifique se o serviço está habilitado:

```
$ systemctl is-enabled pod-systemd-pod.service
enabled
```

Recursos adicionais

- Para mais informações sobre o comando **podman create**, digite **man podman-create**.
- Para mais informações sobre o comando **podman generate systemd**, digite **man podman-generate-systemd**.
- Para mais informações sobre o comando **systemctl**, digite **man systemctl**.
- Para mais informações, veja o artigo de Valentin Rothberg, " Running containers with Podman and shareable systemd services ".
- Para saber mais sobre a configuração de serviços com o systemd, consulte o capítulo Configurando configurações básicas do sistema chamado Gerenciando serviços com o systemd.

CAPÍTULO 9. CONSTRUINDO IMAGENS DE CONTÊINERES COM BUILDAH

O comando **buildah** permite criar imagens de contêineres a partir de um contêiner funcional, um Dockerfile, ou do zero. As imagens resultantes são compatíveis com a OCI, portanto funcionarão em qualquer tempo de funcionamento do contêiner que atenda à Especificação de tempo de funcionamento da OCI (como Docker e CRI-O).

Esta seção descreve como usar o comando **buildah** para criar e trabalhar com recipientes e imagens de recipientes.

9.1. ENTENDENDO BUILDAH

Usar Buildah é diferente de construir imagens com o comando **docker** das seguintes maneiras:

- **No Daemon!**: Desvia o daemon Docker! Portanto, não é necessário tempo de funcionamento do contêiner (Docker, CRI-O, ou outro) para usar o Buildah.
- **Base image or scratch** Permite não somente construir uma imagem baseada em outro recipiente, mas também permite que você comece com uma imagem vazia (scratch).
- **Build tools external**: Não inclui ferramentas de construção dentro da própria imagem. Como resultado, Buildah:
  - Reduz o tamanho das imagens que você constrói
  - Torna a imagem mais segura por não ter o software utilizado para construir o recipiente (como gcc, make, e yum) dentro da imagem resultante.
  - Cria imagens que requerem menos recursos para transportar as imagens (porque são menores).

A Buildah é capaz de operar sem Docker ou outros tempos de funcionamento de containers, armazenando dados separadamente e incluindo recursos que permitem não apenas construir imagens, mas executar essas imagens também como containers. Por padrão, a Buildah armazena imagens em uma área identificada como **containers-storage** (/var/lib/contêineres).

NOTA

O local de armazenamento de contêineres que o comando **buildah** usa por padrão é o mesmo local que o motor de contêineres CRI-O usa para armazenar cópias locais de imagens. Assim, as imagens retiradas de um registro por CRI-O ou Buildah, ou comprometidas pelo comando **buildah**, serão armazenadas na mesma estrutura de diretório. Atualmente, entretanto, CRI-O e Buildah não podem compartilhar contêineres, embora possam compartilhar imagens.

Há mais de uma dúzia de opções para usar com o comando **buildah**. Algumas das principais atividades que você pode fazer com o comando **buildah** incluem:

- **Build a container from a Dockerfile** Utilize um Dockerfile para construir uma nova imagem do container (**buildah bud**).
- **Build a container from another image or scratch** Construir um novo container, começando com uma imagem de base existente (**buildah from <imagename>**) ou do zero (**buildah from scratch**)

- **Inspecting a container or image**: Ver metadados associados ao recipiente ou à imagem (**buildah inspect**)
- **Mount a container**: Montar o sistema de arquivos raiz de um contêiner para adicionar ou alterar o conteúdo (**buildah mount**).
- **Create a new container layer**: Use o conteúdo atualizado do sistema de arquivos raiz de um contêiner como uma camada do sistema de arquivos para comprometer o conteúdo a uma nova imagem (**buildah commit**).
- **Unmount a container**: Desmontar um recipiente montado (**buildah umount**).
- **Delete a container or an image**: Remover um recipiente (**buildah rm**) ou uma imagem do recipiente (**buildah rmi**).

Para mais detalhes sobre Buildah, veja a página de GitHub Buildah . O site GitHub Buildah inclui páginas de homem e software que podem ser mais recentes do que as disponíveis com a versão RHEL. Aqui estão alguns outros artigos sobre Buildah que podem lhe interessar:

- Buildah Tutorial 1: Construindo imagens de contêineres OCI
- Buildah Tutorial 2: Usando Buildah com registros de contêineres
- Blocos de construção - Como se adaptar

9.1.1. Instalando o Buildah

O pacote buildah está disponível com o módulo container-tools no RHEL 8 (**yum module install container-tools**). Você pode instalar o pacote buildah separadamente, digitando:

```
# yum -y install buildah
```

Com o pacote buildah instalado, você pode consultar as páginas de manual incluídas com o pacote buildah para obter detalhes sobre como usá-lo. Para ver as páginas de manual disponíveis e outra documentação, então abra uma página de manual, digite:

```
# rpm -qd buildah
# man buildah
buildah(1)         General Commands Manual         buildah(1)

NAME
 Buildah - A command line tool that facilitates building OCI container images.
...
```

As seções seguintes descrevem como usar **buildah** para obter recipientes, construir um recipiente a partir de um Dockerfile, construir um do zero e gerenciar recipientes de várias maneiras.

9.2. OBTENDO IMAGENS COM BUILDAH

Para obter uma imagem do recipiente para usar com **buildah**, use o comando **buildah from**. Observe que se você estiver usando o RHEL 8.0, poderá encontrar problemas com a autenticação no repositório, veja o bug. Veja aqui como obter uma imagem RHEL 8 do Red Hat Registry como um contêiner funcional para usar com o comando **buildah**:

```
# buildah from registry.redhat.io/ubi8/ubi
```

```
Getting image source signatures
Copying blob…
Writing manifest to image destination
Storing signatures
ubi-working-container
# buildah images
IMAGE ID      IMAGE NAME                       CREATED AT         SIZE
3da40a1670b5  registry.redhat.io/ubi8/ubi:latest  May 8, 2019 21:55  214 MB
# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME         CONTAINER NAME
c6c9279ecc0f     *     3da40a1670b5 ...ubi8/ubi:latest ubi-working-container
```

Observe que o resultado do comando **buildah from** é uma imagem (registry.redhat.io/ubi8/ubi:latest) e um recipiente de trabalho que está pronto para funcionar a partir dessa imagem (ubi-working-container). Aqui está um exemplo de como executar um comando a partir daquele contêiner:

```
# buildah run ubi-working-container cat /etc/redhat-release
Red Hat Enterprise Linux release 8.0
```

A imagem e o recipiente estão agora prontos para uso com Buildah.

9.3. CONSTRUINDO UMA IMAGEM DE UM DOCKERFILE COM BUILDAH

Com o comando **buildah**, você pode criar uma nova imagem a partir de um Dockerfile. Os passos seguintes mostram como construir uma imagem que inclui um script simples que é executado quando a imagem é executada.

Este exemplo simples começa com dois arquivos no diretório atual: Dockerfile (que contém as instruções para construir a imagem do recipiente) e myecho (um script que ecoa algumas palavras para a tela):

```
# ls
Dockerfile  myecho
# cat Dockerfile
FROM registry.redhat.io/ubi8/ubi
ADD myecho /usr/local/bin
ENTRYPOINT "/usr/local/bin/myecho"
# cat myecho
echo "This container works!"
# chmod 755 myecho
# ./myecho
This container works!
```

Com o Dockerfile no diretório atual, construa o novo container da seguinte forma:

```
# buildah bud -t myecho .
STEP 1: FROM registry.redhat.io/ubi8/ubi
STEP 2: ADD myecho /usr/local/bin
STEP 3: ENTRYPOINT "/usr/local/bin/myecho"
```

O comando **buildah bud** cria uma nova imagem chamada myecho. Para ver essa nova imagem, digite:

```
# buildah images
IMAGE NAME        IMAGE TAG  IMAGE ID      CREATED AT          SIZE
localhost/myecho  latest     a3882af49784  Jun 21, 2019 12:21  216 MB
```

Em seguida, você pode executar a imagem, para ter certeza de que ela está funcionando.

9.3.1. Executando a imagem que você construiu

Para verificar se a imagem que você construiu anteriormente funciona, você pode executar a imagem usando **podman run**:

```
# podman run localhost/myecho
This container works!
```

9.3.2. Inspeção de um contêiner com Buildah

Com **buildah inspect**, você pode mostrar informações sobre um recipiente ou imagem. Por exemplo, para construir e inspecionar a imagem **myecho**, digite:

```
# buildah from localhost/myecho
# buildah inspect localhost/myecho | less
{
 "Type": "buildah 0.0.1",
 "FromImage": "docker.io/library/myecho:latest",
 "FromImage-ID": "e2b190ac8...",
 "Config": "{\"created\":\"2018-11-13...

 "Entrypoint": [
     "/usr/local/bin/myecho"
  ],
  "WorkingDir": "/",
  "Labels": {
     "architecture": "x86_64",
     "authoritative-source-url": "registry.access.redhat.com",
     "build-date": "2018-09-19T20:46:28.459833",
```

Para inspecionar um recipiente a partir dessa mesma imagem, digite o seguinte:

```
# buildah inspect myecho-working-container | less
{
    "Type": "buildah 0.0.1",
    "FromImage": "docker.io/library/myecho:latest",
    "FromImage-ID": "e2b190a...",
    "Config": "{\"created\":\"2018-11-13T19:5...
...
    "Container": "myecho-working-container",
    "ContainerID": "c0cd2e494d...",
    "MountPoint": "",
    "ProcessLabel": "system_u:system_r:svirt_lxc_net_t:s0:c89,c921",
    "MountLabel": "",
```

Observe que a saída do recipiente acrescentou informações, tais como o nome do recipiente, identificação do recipiente, etiqueta de processo e etiqueta de montagem ao que estava na imagem.

9.4. MODIFICANDO UM RECIPIENTE PARA CRIAR UMA NOVA IMAGEM COM BUILDAH

Há várias maneiras de modificar um contêiner existente com o comando **buildah** e comprometer essas mudanças para uma nova imagem do contêiner:

- Montar um recipiente e copiar arquivos para ele
- Use **buildah copy** e **buildah config** para modificar um recipiente

Uma vez modificado o recipiente, use **buildah commit** para comprometer as mudanças em uma nova imagem.

9.4.1. Usando buildah mount para modificar um recipiente

Depois de obter uma imagem com **buildah from**, você pode usar essa imagem como base para uma nova imagem. O texto a seguir mostra como criar uma nova imagem montando um recipiente de trabalho, adicionando arquivos a esse recipiente e, em seguida, submetendo as alterações a uma nova imagem.

Digite o seguinte para visualizar o recipiente de trabalho que você usou anteriormente:

```
# buildah containers
CONTAINER ID BUILDER IMAGE ID     IMAGE NAME  CONTAINER NAME

dc8f21af4a47   *     1456eedf8101 registry.redhat.io/ubi8/ubi:latest
            ubi-working-container
6d1ffccb557d   *     ab230ac5aba3 docker.io/library/myecho:latest
            myecho-working-container
```

Monte a imagem do recipiente e defina o ponto de montagem para uma variável ($mymount) para facilitar o manuseio:

```
# mymount=$(buildah mount myecho-working-container)
# echo $mymount
/var/lib/containers/storage/devicemapper/mnt/176c273fe28c23e5319805a2c48559305a57a706cc7ae7b
ec7da4cd79edd3c02/rootfs
```

Adicione conteúdo ao roteiro criado anteriormente no recipiente montado:

```
# echo 'echo \\i1}>> $mymount/usr/local/bin/myecho
```

Para comprometer o conteúdo que você adicionou para criar uma nova imagem (chamada myecho), digite o seguinte

```
# buildah commitdah myecho-working-container containers-containers-storage:myecho2
```

Para verificar se a nova imagem inclui suas mudanças, crie um recipiente de trabalho e execute-o:

```
# buildah images
IMAGE ID     IMAGE NAME     CREATED AT          SIZE
a7e06d3cd0e2 docker.io/library/myecho2:latest
                       Oct 12, 2017 15:15  3.144 KB
```

```
# buildah from docker.io/library/myecho2:latest
myecho2-working-container
# podman run docker.io/library/myecho2
This container works!
We even modified it.
```

Você pode ver que o novo comando **echo** adicionado ao script exibe o texto adicional.

Quando terminar, você pode desmontar o recipiente:

```
# constrói umount myecho-working-container
```

9.4.2. Usando buildah copy e buildah config para modificar um recipiente

Com **buildah copy**, você pode copiar arquivos para um recipiente sem montá-lo primeiro. Aqui está um exemplo, usando o **myecho-working-container** criado (e não montado) na seção anterior, para copiar um novo script para o container e alterar a configuração do container para executar esse script por padrão.

Crie um roteiro chamado **newecho** e torne-o executável:

```
# cat newecho
echo "I changed this container"
# chmod 755 newecho
```

Criar um novo recipiente de trabalho:

```
# buildah from myecho:latest
myecho-working-container-2
```

Copiar **newecho** para /usr/local/bin dentro do contêiner:

```
# buildah copy myecho-working-container-2 newecho /usr/local/bin
```

Alterar a configuração para usar o script **newecho** como o novo ponto de entrada:

```
# buildah config --entrypoint" /bin/sh -c /usr/local/bin/newecho" myecho-working-container-2
```

Execute o novo container, o que deve resultar na execução do comando **newecho**:

```
# buildah run myecho-working-container-2
I changed this container
```

Se o recipiente se comportasse como você esperava, você poderia então comprometê-lo com uma nova imagem (mynewecho):

```
# buildah commitdah myecho-working-container-2 containerss-storage:mynewecho
```

9.5. CRIANDO IMAGENS A PARTIR DO ZERO COM BUILDAH

Em vez de começar com uma imagem de base, você pode criar um novo recipiente que não contenha nenhum conteúdo e apenas uma pequena quantidade de metadados do recipiente. Isto é referido como

um contêiner **scratch**. Aqui estão algumas questões a serem consideradas ao optar por criar uma imagem a partir de um contêiner a partir do zero com o comando **buildah**:

- Ao construir um contêiner de risco você pode copiar um executável sem dependências na imagem de risco e fazer algumas configurações para conseguir um contêiner mínimo para funcionar.
- Para usar ferramentas como **yum** ou pacotes **rpm** para preencher o contêiner de raspar, é necessário pelo menos inicializar um banco de dados RPM no contêiner e adicionar um pacote de liberação. O exemplo abaixo mostra como fazer isso.
- Se você acabar adicionando muitos pacotes de RPM, considere o uso das imagens de base **ubi** ou **ubi-minimal** em vez de uma imagem de arranhão. Essas imagens de base tiveram documentação, pacotes de idiomas e outros componentes cortados, o que pode resultar em uma imagem menor.

Este exemplo acrescenta um serviço Web (httpd) a um recipiente e o configura para funcionar. Para começar, crie um contêiner de arranhões:

```
# buildah from scratch
working-container
```

Isto cria apenas um recipiente vazio (sem imagem) que você pode montar da seguinte forma:

```
# scratchmnt=$(buildah mount working-container)
# echo $scratchmnt
/var/lib/containers/storage/devicemapper/mnt/cc92011e9a2b077d03a97c0809f1f3e7fef0f29bdc6ab5e86
b85430ec77b2bf6/rootfs
```

Inicializar um banco de dados RPM dentro da imagem raspada e adicionar o pacote redhat-release (que inclui outros arquivos necessários para que os RPMs funcionem):

```
# yum install -y --releasever=8 --installroot=$scratchmnt redhat-release
```

Instale o serviço httpd no diretório de raspadinhas:

```
# yum install -y --setopt=reposdir=/etc/yum.repos.d \
    --installroot=$scratchmnt \
    --setopt=cachedir=/var/cache/dnf httpd
```

Adicione algum texto a um arquivo index.html no recipiente, para que você possa testá-lo mais tarde:

```
# echo "Seu container httpd do zero funcionou" > $scratchmnt/var/wwww/html/index.html
```

Em vez de executar o httpd como um serviço init, defina algumas opções em **buildah config** para executar o daemon httpd diretamente do container:

```
# buildah config --cmd "/usr/sbin/httpd -DFOREGROUND" working-container
# buildah config --port 80/tcp working-container
# buildah commit working-container localhost/myhttpd:latest
```

Por enquanto, você pode usar o ID da imagem para executar a nova imagem como um recipiente com o comando **podman**:

```
# podman images
REPOSITORY          TAG              IMAGE ID          CREATED           SIZE
localhost/myhttpd   latest           47c0795d7b0e      9 minutes ago     665.6 MB
# podman run -p 8080:80 -d --name httpd-server 47c0795d7b0e
# curl localhost:8080
Your httpd container from scratch worked.
```

9.6. REMOÇÃO DE IMAGENS OU RECIPIENTES COM BUILDAH

Quando você terminar com determinados recipientes ou imagens, você pode removê-los com **buildah rm** ou **buildah rmi**, respectivamente. Aqui estão alguns exemplos.

Para remover o recipiente criado na seção anterior, você poderia digitar o seguinte para ver o recipiente montado, desmontá-lo e removê-lo:

```
# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER NAME
05387e29ab93     *     c37e14066ac7 docker.io/library/myecho:latest  myecho-working-container
# buildah mount
05387e29ab93 /var/lib/containers/storage/devicemapper/mnt/9274181773a.../rootfs
# buildah umount 05387e29ab93
# buildah rm 05387e29ab93
05387e29ab93151cf52e9c85c573f3e8ab64af1592b1ff9315db8a10a77d7c22
```

Para remover a imagem que você criou anteriormente, você poderia digitar o seguinte:

```
# buildah rmi docker.io/library/myecho:latest
untagged: docker.io/library/myecho:latest
ab230ac5aba3b5a0a7c3d2c5e0793280c1a1b4d2457a75a01b70a4b7a9ed415a
```

9.7. USANDO REGISTROS DE CONTÊINERES COM BUILDAH

Com o Buildah, você pode empurrar e puxar imagens de contêineres entre seu sistema local e os registros de contêineres públicos ou privados. Os exemplos a seguir mostram como fazê-lo:

- Empurrar os contêineres para e puxá-los de um registro privado com buildah.
- Empurre e puxe o container entre seu sistema local e o Docker Registry.
- Use credenciais para associar seus contêineres a uma conta de registro quando você os empurra.

Use o comando skopeo, em conjunto com o comando **buildah**, para consultar os registros para obter informações sobre imagens de contêineres.

9.7.1. Empurrar containers para um registro privado

Empurrar contêineres para um registro de contêineres privado com o comando **buildah** funciona muito parecido com empurrar contêineres com o comando **docker**. Você precisa fazer isso:

- Configurar um registro privado (OpenShift fornece um registro de contêineres ou você pode configurar um registro de contêineres Red Hat Quay).

- Crie ou adquira a imagem do recipiente que você deseja empurrar.
- Use **buildah push** para empurrar a imagem para o registro.

Para empurrar uma imagem do armazenamento local de contêineres Buildah, verifique o nome da imagem e depois empurre-a usando o comando **buildah push**. Lembre-se de identificar tanto o nome da imagem local quanto um novo nome que inclua o local. Por exemplo, um registro rodando no sistema local que está escutando na porta TCP 5000 seria identificado como localhost:5000.

```
# buildah images
IMAGE ID     IMAGE NAME                       CREATED AT          SIZE
cb702d492ee9 docker.io/library/myecho2:latest Nov 12, 2018 16:50     3.143 KB

# buildah push --tls-verify=false myecho2:latest localhost:5000/myecho2:latest
Getting image source signatures
Copying blob sha256:e4efd0...
...
Writing manifest to image destination
Storing signatures
```

Use o comando **curl** para listar as imagens no registro e **skopeo** para inspecionar os metadados sobre a imagem:

```
# curl http://localhost:5000/v2/_catalog
{"repositories":["myatomic","myecho2"]}
# curl http://localhost:5000/v2/myecho2/tags/list
{"name":"myecho2","tags":["latest"]}
# skopeo inspect --tls-verify=false docker://localhost:5000/myecho2:latest | less
{
    "Name": "localhost:5000/myecho2",
    "Digest": "sha256:8999ff6050...",
    "RepoTags": [
        "latest"
    ],
    "Created": "2017-11-21T16:50:25.830343Z",
    "DockerVersion": "",
    "Labels": {
        "architecture": "x86_64",
        "authoritative-source-url": "registry.redhat.io",
```

Neste ponto, qualquer ferramenta que possa extrair imagens de contêineres de um registro de contêineres pode obter uma cópia de sua imagem empurrada. Por exemplo, em um sistema RHEL 7, você poderia iniciar o daemon de doca e tentar puxar a imagem para que ela possa ser usada pelo comando **docker**, como segue:

```
# systemctl start docker
# docker pull localhost:5000/myecho2
# docker run localhost:5000/myecho2
This container works!
```

9.7.2. Empurrando recipientes para o Docker Hub

Você pode usar suas credenciais do Docker Hub para empurrar e puxar imagens do Docker Hub com o comando **buildah**. Para este exemplo, substitua o nome de usuário e a senha (testaccountXX:My00P@sswd) por suas próprias credenciais do Docker Hub:

```
# buildah push --creds testaccountXX:My00P@sswd \
    docker.io/library/myecho2:latest docker://testaccountXX/myecho2:latest
```

Assim como no registro privado, você pode então obter e executar o container do Docker Hub com o comando **podman**, **buildah** ou **docker**:

```
# podman run docker.io/textaccountXX/myecho2:latest
This container works!
# buildah from docker.io/textaccountXX/myecho2:latest
myecho2-working-container-2
# podman run myecho2-working-container-2
This container works!
```

CAPÍTULO 10. MONITORAMENTO DE CONTÊINERES

Este capítulo se concentra nos comandos úteis do Podman que permitem gerenciar um ambiente Podman, incluindo a determinação da saúde do recipiente, a exibição de informações do sistema e da cápsula, e o monitoramento de eventos Podman.

10.1. REALIZAÇÃO DE UM EXAME DE SAÚDE EM UM RECIPIENTE

O healthcheck permite determinar a saúde ou a prontidão do processo que corre dentro do recipiente. Um "healthcheck" consiste em cinco componentes básicos:

- Comando
- Retries
- Intervalo
- Período de início
- Desconto de tempo

A descrição dos componentes do healthcheck é a seguinte.

Comando

Podman executa o comando dentro do recipiente alvo e aguarda o código de saída.

Os outros quatro componentes estão relacionados com a programação do exame de saúde e são opcionais.

Retries

Define o número de exames de saúde falhados consecutivos que precisam ocorrer antes que o recipiente seja marcado como "insalubre". Um exame de saúde bem sucedido restabelece o contador de tentativas.

Intervalo

Descreve o tempo entre a execução do comando do healthcheck. Observe que pequenos intervalos fazem com que seu sistema passe muito tempo executando os cheques de saúde. Os grandes intervalos causam lutas com os intervalos de captura de tempo.

Período de início

Descreve o tempo entre quando o recipiente começa e quando você quer ignorar as falhas do exame de saúde.

Desconto de tempo

Descreve o período de tempo que o exame de saúde deve ser concluído antes de ser considerado mal sucedido.

Os cheques de saúde são feitos dentro do contêiner. O exame de saúde só faz sentido se você souber o que é um estado de saúde do serviço e puder diferenciar entre um exame de saúde bem sucedido e mal sucedido.

Procedimento

1. Definir um exame de saúde:

```
$ sudo podman run -dt --name hc1 --health-cmd='curl http://localhost || exit 1' --health-
interval=0 quay.io/libpod/alpine_nginx:latest
D25ee6faaf6e5e12c09e734b1ac675385fe4d4e8b52504dd01a60e1b726e3edb
```

- A opção **--health-cmd** define um comando de verificação de saúde para o recipiente.
- A opção **-health-interval=0** com valor 0 indica que você deseja executar o exame de saúde manualmente.

2. Executar o exame de saúde manualmente:

```
$ sudo podman healthcheck run hc1
Healthy
```

3. Opcionalmente, você pode verificar o status de saída do último comando:

```
$ echo $?
0
```

O valor "0" significa sucesso.

Recursos adicionais

- Para mais informações sobre o comando **podman run**, digite **man podman-run**.
- Para mais informações, veja o artigo Monitorando a vitalidade e disponibilidade de recipientes com Podman de Brent Baude.

10.2. EXIBIÇÃO DE INFORMAÇÕES DO SISTEMA PODMAN

O comando **podman system** permite gerenciar os sistemas Podman. Esta seção fornece informações sobre como exibir as informações do sistema Podman.

Procedimento

- Exibir informações do sistema Podman:
  - Para mostrar o uso do disco Podman, entre:

```
$ podman system df
TYPE           TOTAL   ACTIVE   SIZE    RECLAIMABLE
Images         3       1        255MB   255MB (100%)
Containers     1       0        0B      0B (0%)
Local Volumes  0       0        0B      0B (0%)
```

  - Para mostrar informações detalhadas sobre o uso do espaço, entre:

```
$ podman system df -v
Images space usage:

REPOSITORY                    TAG      IMAGE ID       CREATED        SIZE     SHARED
SIZE   UNQUE SIZE   CONTAINERS
docker.io/library/alpine      latest   e7d92cdc71fe   3 months ago   5.86MB   0B
5.86MB       0
```

```
registry.access.redhat.com/ubi8/ubi   latest   8121a9f5303b   6 weeks ago    240MB   0B
240MB       1
quay.io/libpod/alpine_nginx           latest   3ef70f7291f4   18 months ago   9.21MB   0B
9.21MB      0

Containers space usage:

CONTAINER ID   IMAGE   COMMAND    LOCAL VOLUMES   SIZE   CREATED
STATUS   NAMES
ff0167c6c271   8121    /bin/bash  0               0B     10 seconds ago   exited   playTest

Local Volumes space usage:

VOLUME NAME   LINKS   SIZE
```

- Para exibir informações sobre o anfitrião, estatísticas atuais de armazenamento e construção do Podman, entre:

```
$ podman system info
host:
  arch: amd64
  buildahVersion: 1.15.0
  cgroupVersion: v1
  conmon:
    package: conmon-2.0.18-1.module+el8.3.0+7084+c16098dd.x86_64
    path: /usr/bin/conmon
    version: 'conmon version 2.0.18, commit:
7fd3f71a218f8d3a7202e464252aeb1e942d17eb'
  cpus: 1
  distribution:
    distribution: '"rhel"'
    version: "8.3"
  eventLogger: file
  hostname: localhost.localdomain
  idMappings:
    gidmap:
    - container_id: 0
      host_id: 1000
      size: 1
    - container_id: 1
      host_id: 100000
      size: 65536
    uidmap:
    - container_id: 0
      host_id: 1000
      size: 1
    - container_id: 1
      host_id: 100000
      size: 65536
  kernel: 4.18.0-227.el8.x86_64
  linkmode: dynamic
  memFree: 69713920
  memTotal: 1376636928
  ociRuntime:
    name: runc
    package: runc-1.0.0-66.rc10.module+el8.3.0+7084+c16098dd.x86_64
```

```
    path: /usr/bin/runc
    version: 'runc version spec: 1.0.1-dev'
  os: linux
  remoteSocket:
    path: /run/user/1000/podman/podman.sock
  rootless: true
  slirp4netns:
    executable: /usr/bin/slirp4netns
    package: slirp4netns-1.1.1-1.module+el8.3.0+7084+c16098dd.x86_64
    version: |-
      slirp4netns version 1.1.1
      commit: bbf27c5acd4356edb97fa639b4e15e0cd56a39d5
      libslirp: 4.3.0
      SLIRP_CONFIG_VERSION_MAX: 3
  swapFree: 1833693184
  swapTotal: 2147479552
  uptime: 145h 19m 14.55s (Approximately 6.04 days)
registries:
  search:
  - registry.access.redhat.com
  - registry.redhat.io
  - docker.io
store:
  configFile: /home/user/.config/containers/storage.conf
  containerStore:
    number: 1
    paused: 0
    running: 0
    stopped: 1
  graphDriverName: overlay
  graphOptions:
    overlay.mount_program:
      Executable: /usr/bin/fuse-overlayfs
      Package: fuse-overlayfs-1.1.1-1.module+el8.3.0+7121+472bc0cf.x86_64
      Version: |-
        fuse-overlayfs: version 1.1.0
        FUSE library version 3.2.1
        using FUSE kernel interface version 7.26
  graphRoot: /home/user/.local/share/containers/storage
  graphStatus:
    Backing Filesystem: xfs
    Native Overlay Diff: "false"
    Supports d_type: "true"
    Using metacopy: "false"
  imageStore:
    number: 15
  runRoot: /run/user/1000/containers
  volumePath: /home/user/.local/share/containers/storage/volumes
version:
  APIVersion: 1
  Built: 0
  BuiltTime: Thu Jan  1 01:00:00 1970
  GitCommit: ""
  GoVersion: go1.14.2
  OsArch: linux/amd64
  Version: 2.0.0
```

- Para remover todos os recipientes, imagens e dados de volume não utilizados, digite:

```
$ podman system prune
WARNING! This will remove:
        - all stopped containers
        - all stopped pods
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
```

  - O comando **podman system prune** remove todos os contêineres não utilizados (tanto os não utilizados como os não referenciados), as cápsulas e, opcionalmente, os volumes do armazenamento local.
  - Use a opção **--all** para excluir todas as imagens não utilizadas. Imagens não utilizadas são imagens suspensas e qualquer imagem que não tenha nenhum recipiente baseado nelas.
  - Use a opção **--volume** para podar volumes. Por padrão, os volumes não são removidos para evitar que dados importantes sejam apagados se não houver atualmente nenhum recipiente usando o volume.

Recursos adicionais

- Para mais informações sobre o comando **podman system df**, digite **man podman-system-df**.
- Para mais informações sobre o comando **podman system info**, digite **man podman-system-info**.
- Para mais informações sobre o comando **podman system prune**, digite **man podman-system-prune**.

10.3. TIPOS DE EVENTOS PODMAN

Você pode monitorar os eventos que ocorrem em Podman. Existem vários tipos de eventos e cada tipo de evento reporta diferentes status.

O tipo de evento *container* informa os seguintes status:

- anexar
- ponto de verificação
- limpeza
- comprometer
- criar
- exec
- exportação
- importação
- init

- matar
- monte
- pausa
- poda
- remover
- reiniciar
- restaurar
- início
- parar
- sincronia
- desmontar
- sempausa

O tipo de evento *pod* informa os seguintes status:

- criar
- matar
- pausa
- remover
- início
- parar
- sempausa

O tipo de evento *image* informa os seguintes status:

- poda
- empurrar
- puxar
- salvar
- remover
- tag
- untag

O tipo *system* informa os seguintes status:

- atualizar
- renumerar

O tipo *volume* informa os seguintes status:

- criar
- poda
- remover

Recursos adicionais

- Para mais informações sobre o comando **podman events**, digite **man podman-events**.

10.4. MONITORAMENTO DE EVENTOS PODMAN

Você pode monitorar e imprimir eventos que ocorrem em Podman. Cada evento incluirá um carimbo da hora, um tipo, um status, um nome (se aplicável) e uma imagem (se aplicável).

Procedimento

- Mostrar eventos Podman:
  - Para mostrar todos os eventos Podman, entre:

```
$ podman events
2020-05-14 10:33:42.312377447 -0600 CST container create 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:46.958768077 -0600 CST container init 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:46.973661968 -0600 CST container start 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:50.833761479 -0600 CST container stop 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:51.047104966 -0600 CST container cleanup 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
```

    Para sair do registro, pressione CTRL c.

  - Para mostrar apenas Podman criar eventos, entre:

```
$ podman events --filter event=create
2020-05-14 10:36:01.375685062 -0600 CST container create 20dc581f6fbf
(image=registry.access.redhat.com/ubi8/ubi:latest)
2019-03-02 10:36:08.561188337 -0600 CST container create 58e7e002344c
(image=registry.access.redhat.com/ubi8/ubi-minimal:latest)
2019-03-02 10:36:29.978806894 -0600 CST container create d81e30f1310f
(image=registry.access.redhat.com/ubi8/ubi-init:latest)
```

Recursos adicionais

- Para mais informações sobre o comando **podman events**, digite **man podman-events**.

CAPÍTULO 11. USANDO AS FERRAMENTAS DO CONTAINER CLI

11.1. PODMAN

O comando **podman** (que significa Pod Manager) permite que você execute containers como entidades autônomas, sem exigir que Kubernetes, o Docker runtime, ou qualquer outro tempo de execução de containers esteja envolvido. É uma ferramenta que pode atuar como um substituto para o comando **docker**, implementando a mesma sintaxe de linha de comando, enquanto adiciona ainda mais recursos de gerenciamento de contêineres. As características do **podman** incluem:

- **Based on the Docker interface**: Como **podman** é o espelho da sintaxe do comando **docker**, a transição para **podman** deve ser fácil para quem está familiarizado com **docker**.
- **Managing containers and images**: Tanto as imagens de contêineres compatíveis com Docker como com OCI podem ser usadas com **podman** para:
  - Correr, parar e reiniciar containers
  - Criar e gerenciar imagens de containers (empurrar, comprometer, configurar, construir, e assim por diante)
- **Managing pods**: Além de operar containers individuais, **podman** pode operar um conjunto de containers agrupados em uma cápsula. Um pod é a menor unidade de contêineres que a Kubernetes administra.
- **Working with no runtime**: Nenhum ambiente de tempo de execução é usado por **podman** para trabalhar com contêineres.

Aqui estão algumas características de implementação do Podman que você deve conhecer:

- Podman, Buildah e o motor de contêineres CRI-O usam todos o mesmo diretório de back-end, **/var/lib/containers**, em vez de usar o local de armazenamento Docker (**/var/lib/docker**), por padrão.
- Embora Podman, Buildah e CRI-O compartilhem o mesmo diretório de armazenamento, eles não podem interagir com os containers um do outro. Essas ferramentas podem, no entanto, compartilhar imagens. Eventualmente, essas características serão capazes de compartilhar os containers.
- O comando **podman**, como o comando **docker**, pode construir imagens de contêineres a partir de um Dockerfile.
- O comando **podman** pode ser uma ferramenta útil de solução de problemas quando o serviço **CRI-O** não estiver disponível.
- As opções para o comando **docker** que não são suportadas por **podman** incluem rede, nó, plugin (**podman** não suporta plugins), renomear (usar rm e criar para renomear containers com **podman**), segredo, serviço, pilha, e enxame (**podman** não suporta enxame de Docker Swarm). As opções de container e imagem são usadas para executar subcomandos que são usados diretamente em **podman**.
- Para interagir programmaticamente com o podman, você pode usar o Podman v2.0 RESTful API, ele funciona tanto em um ambiente sem raízes quanto em um ambiente sem raízes. Para mais informações, consulte o capítulo Utilização da API de ferramentas de contêiner .

11.1.1. Usando comandos podman

Se você está acostumado a usar o comando **docker** para trabalhar com recipientes, você encontrará a maioria das características e opções que correspondem às do **podman**. A Tabela 1 mostra uma lista de comandos que você pode usar com **podman** (digite **podman -h** para ver esta lista):

Tabela 11.1. Comandos apoiados por podman

comando podman	Descrição	comando podman	Descrição
attach	Fixar a um recipiente em funcionamento	**commit**	Criar uma nova imagem a partir de um container modificado
build	Construir uma imagem usando as instruções do Dockerfile	**create**	Criar, mas não começar, um recipiente
diff	Inspecionar mudanças nos sistemas de arquivos dos contêineres	**exec**	Executar um processo em um contêiner em funcionamento
export	Conteúdo do sistema de arquivos do contêiner de exportação como um arquivo de alcatrão	**help, h**	Mostra uma lista de comandos ou ajuda para um comando
history	Mostrar o histórico de uma imagem específica	**images**	Listar imagens em armazenamento local
import	Importar um tarball para criar uma imagem do sistema de arquivos	**info**	Exibir informações do sistema
inspect	Exibir a configuração de um recipiente ou imagem	**kill**	Enviar um sinal específico para um ou mais recipientes em funcionamento
load	Carregar uma imagem de um arquivo	**login**	Login em um registro de contêineres
logout	Logout de um registro de contêineres	**logs**	Buscar os logs de um contêiner
mount	Montar o sistema de arquivos raiz de um contêiner em funcionamento	**pause**	Pausa todos os processos em um ou mais recipientes

ps	Listar recipientes	**port**	Lista de mapeamentos de portas ou um mapeamento específico para o contêiner
pull	Puxar uma imagem de um registro	**push**	Empurrar uma imagem para um destino específico
restart	Reiniciar um ou mais recipientes	**rm**	Remover um ou mais recipientes do hospedeiro. Adicione **-f** se estiver funcionando.
rmi	remove uma ou mais imagens do armazenamento local	**run**	executar um comando em um novo container
save	Salvar imagem em um arquivo	**search**	registro de busca de imagem
start	Iniciar um ou mais recipientes	**stats**	Exibir porcentagem de CPU, memória, E/S da rede, E/S de bloco e PIDs para um ou mais recipientes
stop	Parar um ou mais recipientes	**tag**	Acrescentar um nome adicional a uma imagem local
top	Mostrar os processos em andamento de um recipiente	**umount, unmount**	Desmontar o sistema de arquivos raiz de um contêiner em funcionamento
unpause	Despausar os processos em um ou mais recipientes	**version**	Exibir informações sobre a versão podman
wait	Bloqueio em um ou mais recipientes		

11.1.2. Criação de políticas SELinux para contêineres

Para gerar políticas SELinux para contêineres, utilize a ferramenta UDICA. Para mais informações, consulte Introdução ao gerador de políticas SELinux da udica .

11.1.3. Usando podman com MPI

Você pode usar Podman com MPI (Message Passing Interface) aberto para executar containers em um ambiente de computação de alto desempenho (HPC).

O exemplo é baseado no programa ring.c retirado do Open MPI. Neste exemplo, um valor é passado por todos os processos de uma forma semelhante a um anel. Cada vez que a mensagem passa de nível 0, o valor é decrescido. Quando cada processo recebe a mensagem 0, ele a passa para o processo seguinte e depois desiste. Ao passar o 0 primeiro, cada processo recebe a mensagem 0 e pode desistir normalmente.

Procedimento

1. Instalar MPI aberto:

```
$ sudo yum instalar openmpi
```

2. Para ativar os módulos do ambiente, digite:

```
$ . /etc/profile.d/modules.sh
```

3. Carregue o módulo **mpi/openmpi-x86_64**:

```
$ módulo de carga mpi/openmpi-x86_64
```

   Opcionalmente, para carregar automaticamente o módulo **mpi/openmpi-x86_64**, adicione esta linha ao arquivo **.bashrc**:

```
$ echo "module load mpi/openmpi-x86_64\" >> .bashrc
```

4. Para combinar **mpirun** e **podman**, criar um recipiente com a seguinte definição:

```
$ cat Containerfile
FROM registry.access.redhat.com/ubi8/ubi

RUN yum -y install openmpi-devel wget && \
    yum clean all

RUN wget https://raw.githubusercontent.com/open-mpi/ompi/master/test/simple/ring.c && \
    /usr/lib64/openmpi/bin/mpicc ring.c -o /home/ring && \
    rm -f ring.c
```

5. Construa o recipiente:

```
$ podman build --tag=mpi-ring .
```

6. Comece o recipiente. Em um sistema com 4 CPUs este comando inicia 4 contêineres:

```
$ mpirun \
  --mca orte_tmpdir_base /tmp/podman-mpirun \
  podman run --env-host \
    -v /tmp/podman-mpirun:/tmp/podman-mpirun \
    --userns=keep-id \
    --net=host --pid=host --ipc=host \
    mpi-ring /home/ring
```

```
Rank 2 has cleared MPI_Init
Rank 2 has completed ring
Rank 2 has completed MPI_Barrier
Rank 3 has cleared MPI_Init
Rank 3 has completed ring
Rank 3 has completed MPI_Barrier
Rank 1 has cleared MPI_Init
Rank 1 has completed ring
Rank 1 has completed MPI_Barrier
Rank 0 has cleared MPI_Init
Rank 0 has completed ring
Rank 0 has completed MPI_Barrier
```

Como resultado, **mpirun** inicia 4 recipientes Podman e cada recipiente está rodando uma instância do binário **ring**. Todos os 4 processos estão se comunicando por MPI entre si.

As seguintes opções de **mpirun** são usadas para iniciar o recipiente:

- a linha**--mca orte_tmpdir_base /tmp/podman-mpirun** diz à Open MPI para criar todos os seus arquivos temporários em **/tmp/podman-mpirun** e não em **/tmp**. Se utilizar mais de um nó, este diretório será nomeado de forma diferente em outros nós. Isto requer a montagem do diretório **/tmp** completo dentro do container, o que é mais complicado.

O comando **mpirun** especifica o comando a ser iniciado, o comando **podman**. As seguintes opções **podman** são usadas para iniciar o contêiner:

- o comando**run** executa um contêiner.
- a opção**--env-host** copia todas as variáveis de ambiente do host para o container.
- **-v /tmp/podman-mpirun:/tmp/podman-mpirun** diz à Podman para montar o diretório onde a Open MPI cria seus diretórios e arquivos temporários para estarem disponíveis no container.
- a linha**--userns=keep-id** garante o mapeamento da identificação do usuário dentro e fora do contêiner.
- a linha**--net=host --pid=host --ipc=host** define a mesma rede, PID e IPC namespaces.
- **mpi-ring** é o nome do recipiente.
- **/home/ring** é o programa MPI no contêiner.

Para mais informações, veja o artigo Podman em ambientes HPC por Adrian Reber.

11.1.4. Criação e restauração de pontos de verificação de contêineres

Checkpoint/Restore In Userspace (CRIU) é um software que permite definir um ponto de verificação em um container em funcionamento ou em um aplicativo individual e armazenar seu estado em disco. Você pode usar os dados salvos para restaurar o contêiner após um reinício no mesmo ponto no tempo em que ele foi verificado.

11.1.4.1. Criação e restauração local de um ponto de verificação de contêineres

Este exemplo é baseado em um servidor web baseado em Python que retorna um único número inteiro que é incrementado após cada solicitação.

Procedimento

1. Criar um servidor baseado em Python:

```
# cat counter.py
#!/usr/bin/python3

import http.server

counter = 0

class handler(http.server.BaseHTTPRequestHandler):
   def do_GET(s):
      global counter
        s.send_response(200)
        s.send_header('Content-type', 'text/html')
        s.end_headers()
        s.wfile.write(b'%d\n' % counter)
        counter += 1


server = http.server.HTTPServer(('', 8088), handler)
server.serve_forever()
```

2. Crie um recipiente com a seguinte definição:

```
# cat Containerfile
FROM registry.access.redhat.com/ubi8/ubi

COPY counter.py /home/counter.py

RUN useradd -ms /bin/bash counter

RUN yum -y install python3 && chmod 755 /home/counter.py

USER counter
ENTRYPOINT /home/counter.py
```

   O recipiente é baseado na Imagem Base Universal (UBI 8) e utiliza um servidor baseado em Python.

3. Construa o recipiente:

```
# podman build . --contador de etiquetas
```

   Os arquivos **counter.py** e **Containerfile** são a entrada para o processo de construção do contêiner (**podman build**). A imagem construída é armazenada localmente e etiquetada com a etiqueta **counter**.

4. Comece o recipiente como raiz:

```
# podman run --name criu-test --detach counter
```

5. Para listar todos os contêineres em funcionamento, entre:

```
# podman ps
CONTAINER ID  IMAGE  COMMAND  CREATED   STATUS  PORTS NAMES
e4f82fd84d48  localhost/counter:latest  5 seconds ago  Up 4 seconds ago  criu-test
```

6. Mostrar o endereço IP do recipiente:

```
# podman inspect criu-test --format "{{.NetworkSettings.IPAddress}}"
10.88.0.247
```

7. Enviar pedidos para o container:

```
# curl 10.88.0.247:8080
0
# curl 10.88.0.247:8080
1
```

8. Criar um ponto de controle para o contêiner:

```
# podman ponto de verificação de contêineres criu-teste
```

9. Reinicie o sistema.

10. Restaurar o recipiente:

```
# podman container restore -- manter criu-test
```

11. Enviar pedidos para o container:

```
# curl 10.88.0.247:8080
2
# curl 10.88.0.247:8080
3
# curl 10.88.0.247:8080
4
```

O resultado agora não começa novamente em **0**, mas continua com o valor anterior.

Desta forma, você pode facilmente salvar o estado completo do recipiente através de uma reinicialização.

Para mais informações, consulte o artigo Adicionando ponto de verificação/restauração ao Podman por Adrian Reber.

11.1.4.2. Redução do tempo de partida usando o container restore

Você pode usar a migração de contêineres para reduzir o tempo de inicialização dos contêineres que requerem um certo tempo para serem inicializados. Usando um ponto de verificação, você pode restaurar o container várias vezes no mesmo host ou em hosts diferentes. Este exemplo é baseado no container da seção Criando e restaurando um ponto de verificação de container localmente .

Procedimento

1. Criar um ponto de controle do container e exportar a imagem do ponto de controle para um arquivo **tar.gz**:

```
# podman ponto de verificação criu-teste de contêineres - exportação /tmp/chkpt.tar.gz
```

2. Restaurar o recipiente a partir do arquivo **tar.gz**:

```
# podman container restore --import /tmp/chkpt.tar.gz --name counter1
# podman container restore --import /tmp/chkpt.tar.gz --name counter2
# podman container restore --import /tmp/chkpt.tar.gz --name counter3
```

   A opção **--name** (**-n**) especifica um novo nome para os contêineres restaurados a partir do ponto de controle exportado.

3. Mostrar ID e nome de cada recipiente:

```
# podman ps -a --format "{{.ID}} {{.Names}}"
a8b2e50d463c counter3
faabc5c27362 counter2
2ce648af11e5 counter1
```

4. Mostrar o endereço IP de cada contêiner:

```
#  podman inspect counter1 --format "{{.NetworkSettings.IPAddress}}"
10.88.0.248

#  podman inspect counter2 --format "{{.NetworkSettings.IPAddress}}"
10.88.0.249

#  podman inspect counter3 --format "{{.NetworkSettings.IPAddress}}"
10.88.0.250
```

5. Enviar pedidos para cada contêiner:

```
#  curl 10.88.0.248:8080
4
#  curl 10.88.0.249:8080
4
#  curl 10.88.0.250:8080
4
```

   Note que o resultado é **4** em todos os casos, porque você está trabalhando com diferentes recipientes restaurados a partir do mesmo ponto de verificação.

Usando esta abordagem, você pode iniciar rapidamente réplicas do contêiner inicialmente marcado.

Para mais informações, consulte o artigo Migração de contêineres com Podman sobre a RHEL , de Adrian Reber.

11.1.4.3. Migração de contêineres entre sistemas

Este procedimento mostra a migração de containers em funcionamento de um sistema para outro, sem perder o estado das aplicações em funcionamento no container. Este exemplo é baseado no container da seção Criando e restaurando um ponto de verificação de container localmente etiquetado com

counter.

Pré-requisitos

As seguintes etapas não são necessárias se o contêiner for empurrado para um registro, pois Podman fará automaticamente o download do contêiner de um registro se ele não estiver disponível localmente. Este exemplo não utiliza um registro, você tem que exportar o contêiner previamente construído e etiquetado (veja Criação e restauração de um ponto de verificação de contêineres localmente) localmente e importar o contêiner no sistema de destino desta migração.

- Exportar contêineres previamente construídos:

```
# podman save --output counter.tar counter
```

- Cópia da imagem do recipiente exportado para o sistema de destino (***other_host***):

```
# scp counter.tar other_host:
```

- Importação de contêineres exportados no sistema de destino:

```
# ssh other_host podman load --input counter.tar
```

Agora o sistema de destino desta migração de contêineres tem a mesma imagem de contêiner armazenada em seu armazenamento local de contêineres.

Procedimento

1. Comece o recipiente como raiz:

```
# podman run --name criu-test --detach counter
```

2. Mostrar o endereço IP do recipiente:

```
# podman inspect criu-test --format "{{.NetworkSettings.IPAddress}}"
10.88.0.247
```

3. Enviar pedidos para o container:

```
# curl 10.88.0.247:8080
0
# curl 10.88.0.247:8080
1
```

4. Criar um ponto de controle do container e exportar a imagem do ponto de controle para um arquivo **tar.gz**:

```
# podman ponto de verificação de contêineres criu-teste - exportação /tmp/chkpt.tar.gz
```

5. Copiar o arquivo do ponto de verificação para o anfitrião de destino:

```
# scp /tmp/chkpt.tar.gz other_host:/tmp/
```

6. Restaurar o ponto de controle no host de destino (***other_host***):

```
# podman container restore --import /tmp/chkpt.tar.gz
```

7. Enviar um pedido ao contêiner no host de destino (***other_host***):

```
# curl 10.88.0.247:8080
2
```

Como resultado, o contentor estatal foi migrado de um sistema para outro sem perder seu estado.

Para mais informações, consulte o artigo Migração de contêineres com Podman sobre a RHEL , de Adrian Reber.

11.2. RUNC

A RunC é uma implementação leve e portátil da Iniciativa de Contêineres Abertos (OCI), especificação de tempo de funcionamento de contêineres. O runC reúne muitas das características de baixo nível que tornam possível a utilização de contêineres em funcionamento. Ele compartilha muito código de baixo nível com o Docker, mas não depende de nenhum dos componentes da plataforma Docker.

runc suporta namespaces Linux, migração ao vivo, e tem perfis de desempenho portáteis. Ele também fornece suporte total para recursos de segurança Linux como SELinux, grupos de controle (cgroups), seccomp, e outros. Você pode construir e executar imagens com o runc, ou pode executar imagens compatíveis com o OCI com o runc.

11.2.1. Contêineres com runc

Com runc, os recipientes são configurados utilizando fardos. Um pacote para um contêiner é um diretório que inclui um arquivo de especificação chamado **config.json** e um sistema de arquivos raiz. O sistema de arquivos raiz contém o conteúdo do contêiner.

Para criar um pacote, execute:

```
Especificações de $ runc
```

Este comando cria um arquivo **config.json** que contém apenas uma estrutura de ossos nus que você precisará editar. Mais importante ainda, você precisará alterar o parâmetro **args** para identificar o executável a ser executado. Por padrão, **args** está configurado para **sh**.

```
"args": [
  "sh"
],
```

Como exemplo, você pode baixar a imagem base do Red Hat Enterprise Linux (**ubi8**/**ubi**) usando o podman e depois exportá-la, criar um novo pacote para ela com o runc e editar o arquivo **config.json** para apontar para essa imagem. Você pode então criar a imagem do contêiner e executar uma instância dessa imagem com o runc. Use os seguintes comandos:

```
# podman pull registry.redhat.io/ubi8/ubi
# podman export $(podman create registry.redhat.io/ubi8/ubi) > rhel.tar
# mkdir -p rhel-runc/rootfs
# tar -C rhel-runc/rootfs -xf rhel.tar
# runc spec -b rhel-runc
# vi rhel-runc/config.json   Change any setting you like
```

```
# runc create -b rhel-runc/ rhel-container
# runc start rhel-container
sh-4.2#
```

Neste exemplo, o nome da instância do recipiente é **rhel-container**. Executando esse recipiente, por padrão, inicia-se uma concha, para que você possa começar a olhar ao redor e executar comandos a partir do interior daquele recipiente. Digite **exit** quando estiver pronto.

O nome de uma instância de recipiente deve ser único no anfitrião. Para iniciar uma nova instância de um contêiner:

```
# runc start <container_name>
```

Você pode fornecer o diretório do pacote usando a opção **-b**. Por padrão, o valor para o pacote é o diretório atual.

Você precisará de privilégios de raiz para iniciar recipientes com runc. Para ver todos os comandos disponíveis para o runc e sua utilização, execute **runc --help**.

11.3. SKOPEO

Com o comando **skopeo**, você pode trabalhar com imagens de contêineres de registros sem usar o daemon de doca ou o comando **docker**. Os registros podem incluir o Docker Registry, seus próprios registros locais, Red Hat Quay ou OpenShift. As atividades que você pode fazer com o skopeo incluem:

- **inspect**: A saída de um comando **skopeo inspect** é semelhante ao que você vê de um comando **docker inspect**: informações de baixo nível sobre a imagem do contêiner. Essa saída pode ser em formato json (padrão) ou em formato raw (usando a opção **--raw**).
- **copy**: Com **skopeo copy** você pode copiar uma imagem de um container de um registro para outro registro ou para um diretório local.
- **layers**: O comando **skopeo layers** permite baixar as camadas associadas às imagens para que elas sejam armazenadas como tarballs e arquivos de manifesto associados em um diretório local.

Como o comando **buildah** e outras ferramentas que dependem da biblioteca de contêineres/imagens, o comando **skopeo** pode trabalhar com imagens de áreas de armazenamento de contêineres diferentes daquelas associadas ao Docker. Os transportes disponíveis para outros tipos de armazenamento de contêineres incluem: containers-storage (para imagens armazenadas por **buildah** e CRI-O), ostree (para contêineres atômicos e de sistema), oci (para conteúdo armazenado em um diretório compatível com OCI), e outros. Veja a página de manual skopeo para detalhes.

Para experimentar o skopeo, você poderia criar um registro local, depois executar os comandos que se seguem para inspecionar, copiar e baixar as camadas de imagem. Se você quiser seguir junto com os exemplos, comece fazendo o seguinte:

- Instale um registro local (como o Red Hat Quay. O software de registro de contêineres disponível no pacote doca-distribuição para a RHEL 7, não está disponível para a RHEL 8.
- Puxe a imagem mais recente da RHEL para seu sistema local (**podman pull ubi8/ubi**).
- Retagne a imagem RHEL e empurre-a para seu registro local como a seguir:

```
# podman tag ubi8/ubi localhost/myubi8
# podman push localhost/myubi8
```

O restante desta seção descreve como inspecionar, copiar e obter camadas da imagem RHEL.

NOTA

A ferramenta **skopeo**, por padrão, requer uma conexão TLS. Ela falha ao tentar usar uma conexão não criptografada. Para anular o padrão e usar um registro http, prefira **http:** à string **<registry>/<image>**.

11.3.1. Inspeção de imagens de contêineres com skopeo

Quando você inspeciona uma imagem do contêiner a partir de um registro, você precisa identificar o formato do contêiner (como docker.io ou localhost), a localização do registro (como docker.io ou localhost), e o repositório/imagem (como ubi8/ubi).

O seguinte exemplo inspeciona a imagem do container mariadb do Docker Registry:

```
# skopeo inspect docker://docker.io/library/mariadb
{
    "Name": "docker.io/library/mariadb",
    "Tag": "latest",
    "Digest": "sha256:d3f56b143b62690b400ef42e876e628eb5e488d2d0d2a35d6438a4aa841d89c4",
    "RepoTags": [
        "10.0.15",
        "10.0.16",
        "10.0.17",
        "10.0.19",
...
    "Created": "2018-06-10T01:53:48.812217692Z",
    "DockerVersion": "1.10.3",
    "Labels": {},
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
...
```

Assumindo que você empurrou uma imagem de container etiquetada **localhost/myubi8** para um registro de container rodando em seu sistema local, o seguinte comando inspeciona essa imagem:

```
# skopeo inspect docker://localhost/myubi8
{
    "Name": "localhost/myubi8",
    "Tag": "latest",
    "Digest": "sha256:4e09c308a9ddf56c0ff6e321d135136eb04152456f73786a16166ce7cba7c904",
    "RepoTags": [
        "latest"
    ],
    "Created": "2018-06-16T17:27:13Z",
    "DockerVersion": "1.7.0",
    "Labels": {
        "Architecture": "x86_64",
        "Authoritative_Registry": "registry.access.redhat.com",
        "BZComponent": "rhel-server-docker",
        "Build_Host": "rcm-img01.build.eng.bos.redhat.com",
        "Name": "myubi8",
```

```
        "Release": "75",
        "Vendor": "Red Hat, Inc.",
        "Version": "8.0"
    },
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
        "sha256:16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf"
    ]
}
```

11.3.2. Copiando imagens de contêineres com skopeo

Este comando copia a imagem do recipiente myubi8 de um registro local para um diretório no sistema local:

```
# skopeo copy docker://localhost/myubi8 dir:/root/test/
INFO[0000] Downloading
myubi8/blobs/sha256:16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf
# ls /root/test
16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf.tar manifest.json
```

O resultado do comando **skopeo copy** é um arquivo tarball (16d*.tar) e um arquivo manifest.json representando a imagem que está sendo copiada para o diretório que você identificou. Se houvesse múltiplas camadas, haveria múltiplas tarballs. O comando **skopeo copy** também pode copiar imagens para outro registro. Se você precisar fornecer uma assinatura para escrever no registro de destino, você pode fazer isso adicionando uma opção **--sign-by=** à linha de comando, seguida do key-id necessário.

11.3.3. Obtendo camadas de imagem com skopeo

O comando **skopeo layers** é similar ao **skopeo copy**, com a diferença de que a opção **copy** pode copiar uma imagem para outro registro ou para um diretório local, enquanto a opção **layers** apenas deixa cair as camadas (tarballs e manifesto.json file) no diretório atual. Por exemplo

```
# skopeo layers docker://localhost/myubi8
INFO[0000] Downloading
myubi8/blobs/sha256:16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf
# find .
./layers-myubi8-latest-698503105
./layers-myubi-latest-698503105/manifest.json
./layers-myubi8-latest-
698503105/16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf.tar
```

Como você pode ver neste exemplo, um novo diretório é criado (**layers-myubi8-latest-698503105**) e, neste caso, um tarball de uma camada e um arquivo **manifest.json** são copiados para esse diretório.

CAPÍTULO 12. USANDO O API DE FERRAMENTAS DE RECIPIENTE

A nova API Podman 2.0 baseada em REST substitui a antiga API remota para Podman que usava a biblioteca varlink. A nova API funciona tanto em um ambiente sem raízes quanto em um ambiente sem raízes.

O Podman v2.0 RESTful API consiste no Libpod API que fornece suporte para Podman, e Docker API compatível com Docker.

Com esta nova API REST, você pode chamar Podman de plataformas como cURL, Postman, cliente REST avançado do Google, e muitas outras.

12.1. HABILITANDO O PODMAN API USANDO O SYSTEMD NO MODO RAIZ

Este procedimento mostra como fazer o seguinte:

1. Use o systemd para ativar o soquete Podman API.
2. Use um cliente Podman para executar comandos básicos.

Pré-requisitos

- O pacote **podman-remote** está instalado.

```
# yum instalar podman-remote
```

Procedimento

1. Configurar o arquivo da unidade do sistema para o soquete Podman:

```
# cat /usr/lib/systemd/system/podman.socket

[Unit]
Description=Podman API Socket
Documentation=man:podman-api(1)

[Socket]
ListenStream=%t/podman/podman.sock
SocketMode=0660

[Install]
WantedBy=sockets.target
```

2. Recarregar a configuração do gerenciador do sistema:

```
# systemctl daemon-reload
```

3. Iniciar o serviço imediatamente:

```
# systemctl enable --now podman.socket
```

4. Para habilitar o link para **var/lib/docker.sock** usando o pacote **docker-podman**:

```
# yum instalar podman-docker
```

Etapas de verificação

- Exibir informações do sistema Podman:

```
# podman-remote info
```

- Verifique o link:

```
# ls -al /var/run/docker.sock
lrwxrwxrwx. 1 root root 23 Nov  4 10:19 /var/run/docker.sock -> /run/podman/podman.sock
```

Recursos adicionais

- Para mais informações sobre o Podman 2.0 API, veja a documentação do Podman v2.0 RESTful API.
- Para mais exemplos sobre como usar o Podman 2.0 API, veja o artigo A First Look At Podman 2.0 API, de Scott McCarty.
- Para mais exemplos de como usar o Podman 2.0 API, veja o Sneak peek: O novo artigo REST API da Podman, de Tom Sweeney.

12.2. HABILITANDO O PODMAN API USANDO O SYSTEMD EM MODO SEM RAIZ

Este procedimento mostra como usar o sistemad para ativar o soquete Podman API e o serviço Podman API.

Pré-requisitos

- O pacote **podman-remote** está instalado.

```
# yum instalar podman-remote
```

Procedimento

1. Criar um soquete sem raiz para Podman:

```
$ vim .config/systemd/user/podman.socket

[Unit]
Description=Podman API Socket
Documentation=man:podman-api(1)

[Socket]
ListenStream=/home/username/podman.sock
SocketMode=0660
```

```
[Install]
WantedBy=sockets.target
```

2. Criar um serviço sem raízes para Podman:

```
$ vim .config/systemd/user/podman.service
[Unit]
Description=Podman API Service
Requires=podman.socket
After=podman.socket
Documentation=man:podman-api(1)
StartLimitIntervalSec=0

[Service]
Type=oneshot
Environment=REGISTRIES_CONFIG_PATH=/etc/containers/registries.conf
ExecStart=/usr/bin/podman system service unix:///home/username/podman.sock
TimeoutStopSec=30
KillMode=process

[Install]
WantedBy=multi-user.target
Also=podman.socket
```

   - A linha **After** na seção **[Unit]** define a dependência do arquivo da unidade **podman.socket**. A unidade **podman.socket** foi iniciada antes da configurada **podman.service**.

3. Recarregar a configuração do gerenciador do sistema:

```
$ systemctl -- daemon-reload do usuário
```

4. Habilitar e iniciar o serviço imediatamente:

```
$ systemctl -- ativação do usuário --now podman.socket
```

5. Para permitir que os programas que utilizam o Docker interajam com o soquete Podman sem raiz:

```
$ exportação DOCKER_HOST=unix:///var/run/<username>/podman.sock
```

Etapas de verificação

- Exibir informações do sistema Podman:

```
$ podman-remote info
```

Recursos adicionais

- Para mais informações sobre o Podman 2.0 API, veja a documentação do Podman v2.0 RESTful API.

- Para mais exemplos sobre como usar o Podman 2.0 API, veja o artigo A First Look At Podman 2.0 API, de Scott McCarty.
- Para mais exemplos de como usar o Podman 2.0 API, veja o Sneak peek: O novo artigo REST API da Podman, de Tom Sweeney.
- Para exemplos de utilização do Podman 2.0 API com Python e Bash, veja o artigo Exploring Podman RESTful API usando Python e Bash de Jhon Honce.

12.3. EXECUTANDO O PODMAN API MANUALMENTE

Este procedimento descreve como executar o Podman API. Isto é útil para depuração de chamadas API, especialmente quando se usa a camada de compatibilidade Docker.

Pré-requisitos

- O pacote **podman-remote** está instalado.

```
# yum instalar podman-remote
```

Procedimento

1. Execute o serviço para o REST API:

```
# podman system service -t 0 --log-level=debug
```

   - O valor de 0 significa que não há timeout. O ponto final padrão para um serviço de raiz é **unix:/run/podman/podman.sock**.
   - A opção **--log-level <level>** define o nível de extração. Os níveis padrão de extração são **debug**, **info**, **warn**, **error**, **fatal**, e **panic**.

2. Em outro terminal, exibir informações do sistema Podman. O comando **podman-remote**, ao contrário do comando normal **podman**, comunica-se através do soquete Podman:

```
# podman-remote info
```

3. Para solucionar problemas na API do Podman e exibir solicitações e respostas, use o comando **curl**. Para obter as informações sobre a instalação do Podman no servidor Linux em formato JSON:

```
# curl -s --unix-socket /run/podman/podman.sock http://d/v1.0.0/libpod/info | jq
   {
 "host": {
   "arch": "amd64",
   "buildahVersion": "1.15.0",
   "cgroupVersion": "v1",
   "conmon": {
     "package": "conmon-2.0.18-1.module+el8.3.0+7084+c16098dd.x86_64",
     "path": "/usr/bin/conmon",
     "version": "conmon version 2.0.18, commit:
7fd3f71a218f8d3a7202e464252aeb1e942d17eb"
   },
   …
```

```
  "version": {
    "APIVersion": 1,
    "Version": "2.0.0",
    "GoVersion": "go1.14.2",
    "GitCommit": "",
    "BuiltTime": "Thu Jan  1 01:00:00 1970",
    "Built": 0,
    "OsArch": "linux/amd64"
  }
}
```

Um utilitário **jq** é um processador JSON de linha de comando.

4. Puxe a imagem do recipiente **registry.access.redhat.com/ubi8/ubi**:

```
# curl -XPOST --unix-socket /run/podman/podman.sock -v 'http://d/v1.0.0/images/create?
fromImage=registry.access.redhat.com%2Fubi8%2Fubi'
*   Trying /run/podman/podman.sock...
* Connected to d (/run/podman/podman.sock) port 80 (#0)
> POST /v1.0.0/images/create?fromImage=registry.access.redhat.com%2Fubi8%2Fubi
HTTP/1.1
> Host: d
> User-Agent: curl/7.61.1
> Accept: /
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Date: Tue, 20 Oct 2020 13:58:37 GMT
< Content-Length: 231
<
{"status":"pulling image () from registry.access.redhat.com/ubi8/ubi:latest,
registry.redhat.io/ubi8/ubi:latest","error":"","progress":"","progressDetail":
{},"id":"ecbc6f53bba0d1923ca9e92b3f747da8353a070fccbae93625bd8b47dbee772e"}
* Connection #0 to host d left intact
```

5. Exibir a imagem puxada:

```
# curl --unix-socket /run/podman/podman.sock -v 'http://d/v1.0.0/libpod/images/json' | jq
*   Trying /run/podman/podman.sock...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0* Connected to d
(/run/podman/podman.sock) port 80 (0) > GET /v1.0.0/libpod/images/json HTTP/1.1 > Host: d
> User-Agent: curl/7.61.1 > Accept: / > < HTTP/1.1 200 OK < Content-Type: application/json
< Date: Tue, 20 Oct 2020 13:59:55 GMT < Transfer-Encoding: chunked < { [12498 bytes
data] 100 12485 0 12485 0 0 2032k 0 --:--:-- --:--:-- --:--:-- 2438k * Connection #0 to host d
left intact [ { "Id":
"ecbc6f53bba0d1923ca9e92b3f747da8353a070fccbae93625bd8b47dbee772e",
"RepoTags": [ "registry.access.redhat.com/ubi8/ubi:latest", "registry.redhat.io/ubi8/ubi:latest"
], "Created": "2020-09-01T19:44:12.470032Z", "Size": 210838671, "Labels": { "architecture":
"x86_64", "build-date": "2020-09-01T19:43:46.041620", "com.redhat.build-host": "cpt-
1008.osbs.prod.upshift.rdu2.redhat.com", ... "maintainer": "Red Hat, Inc.", "name": "ubi8", ...
"summary": "Provides the latest release of Red Hat Universal Base Image 8.", "url":
"https://access.redhat.com/containers//registry.access.redhat.com/ubi8/images/8.2-347",
      ...
    },
```

```
    "Names": [
      "registry.access.redhat.com/ubi8/ubi:latest",
      "registry.redhat.io/ubi8/ubi:latest"
    ],
    ...
    ]
  }
]
```

Recursos adicionais

- Para mais informações sobre o Podman 2.0 API, veja a documentação do Podman v2.0 RESTful API.
- Para mais exemplos de como usar o Podman 2.0 API, veja o Sneak peek: O novo artigo REST API da Podman, de Tom Sweeney.
- Para exemplos de utilização do Podman 2.0 API com Python e Bash, veja o artigo Exploring Podman RESTful API usando Python e Bash de Jhon Honce.
- Para mais informações sobre o comando **podman system service**, consulte a página de manual **podman-system-service**.

CAPÍTULO 13. RECURSOS ADICIONAIS

- Buildah - uma ferramenta para a construção de imagens de contêineres OCI
- Podman - uma ferramenta para o funcionamento e gerenciamento de containers
- Skopeo - uma ferramenta para copiar e inspecionar imagens de contêineres