

# **Containerized HPCC Systems® Platform**

**Equipe de documentação de Boca Raton**



## Containerized HPCC Systems® Platform

Equipe de documentação de Boca Raton

Copyright © 2023 HPCC Systems®. All rights reserved

Sua opinião e comentários sobre este documento são muito bem-vindos e podem ser enviados por e-mail para <docfeedback@hpccsystems.com>

Inclua a frase **Feedback sobre documentação** na linha de assunto e indique o nome do documento, o número das páginas e número da versão atual no corpo da mensagem.

LexisNexis e o logotipo Knowledge Burst são marcas comerciais registradas da Reed Elsevier Properties Inc., usadas sob licença.

HPCC Systems® é uma marca registrada da LexisNexis Risk Data Management Inc.

Os demais produtos, logotipos e serviços podem ser marcas comerciais ou registradas de suas respectivas empresas.

Todos os nomes e dados de exemplo usados neste manual são fictícios. Qualquer semelhança com pessoas reais, vivas ou mortas, é mera coincidência.

2023 Version 9.0.2-1

|  |    |
|--|----|
| Visão geral do HPCC em contêineres .....                     | 4  |
| Bare-metal vs Containers .....                               | 5  |
| Deploy Local (Desenvolvimento e Teste) .....                 | 7  |
| Pré-requisitos .....   | 7  |
| Adicionar Repositório .....                                  | 7  |
| Iniciar um Sistema Padrão .....                              | 8  |
| Acesso padrão do sistema .....                               | 10 |
| Encerrar (Descomissionar) o Sistema .....                    | 11 |
| Persistent Storage para um Deploy local .....                | 12 |
| Importar: Planos de armazenamento e como usá-los .....       | 15 |
| Configurações Customizadas .....                             | 16 |
| Técnicas de Customização .....                               | 16 |
| Configuração dos Valores .....                               | 20 |
| O Ambiente do Contêiner .....                                | 20 |
| Componentes HPCC Systems no Arquivo <i>values.yaml</i> ..... | 21 |
| O arquivo HPCC Systems <i>values.yaml</i> .....              | 27 |
| Mais Helm e Yaml .....                                       | 33 |
| Logging em contêiner .....                                   | 39 |
| Histórico de Logging .....                                   | 39 |
| Soluções de Processamento de Log .....                       | 40 |
| Instalando o chart <i>elastic4hpclogs</i> .....              | 41 |
| Azure AKS Insights .....                                     | 44 |
| Controlando a saída de registro do HPCC Systems .....        | 46 |

# Visão geral do HPCC em contêineres

A partir da versão 8.0, a plataforma HPCC Systems® focará em deploys em contêineres. Isso é útil para implantações baseadas em nuvem (grandes ou pequenas) ou implantações de teste/desenvolvimento locais.

Os contêineres do Docker gerenciados pelo Kubernetes (K8s) são um novo ambiente operacional de destino, juntamente com o suporte contínuo para instalações tradicionais "bare metal" usando arquivos do instalador .deb ou .rpm. O suporte para instaladores tradicionais continua e esse tipo de implantação é viável para implantações bare metal ou configurações manuais na nuvem.

Esta não é uma mudança do tipo "*rehosting*", em que a plataforma executa sua estrutura legada inalterada e trata os contêineres apenas como uma forma de fornecer *máquinas virtuais* e para serem executadas, mas uma mudança significativa em como os componentes são configurados, como e quando eles iniciam e onde armazenam seus dados.

Este livro se concentra em implantações em contêineres. A primeira seção é sobre o uso de contêineres Docker e gráficos Helm localmente. Docker e Helm fazem muito do trabalho para você. A segunda parte usa as mesmas técnicas na nuvem.

Para pequenas implantações locais (para desenvolvimento e teste), sugerimos o uso de Docker Desktop e Helm. Isto é útil para aprendizagem, desenvolvimento e teste.

Para implantações em nuvem, você pode usar qualquer tipo de serviço em nuvem, se for compatível com Docker, Kubernetes e Helm. Este livro, no entanto, se concentrará no Microsoft Azure para serviços em nuvem. As versões futuras podem incluir especificações para outros provedores de nuvem.

Se você deseja gerenciar manualmente sua implantação local ou na nuvem, ainda pode usar os instaladores tradicionais e o Configuration Manager, mas isso remove muitos dos benefícios que o Docker, Kubernetes e Helm fornecem, como instrumentação, monitoramento, escalonamento e custo ao controle.

Os sistemas HPCC seguem as convenções padrão sobre como as implantações do Kubernetes são normalmente configuradas e gerenciadas, portanto, deve ser fácil para alguém familiarizado com o Kubernetes e o Helm instalar e gerenciar a plataforma HPCC Systems.

**Note:** A versão tradicional bare-metal da plataforma de sistemas HPCC está madura e tem sido amplamente usada em aplicativos comerciais por quase duas décadas e é totalmente destinada para uso em produção. A versão em contêiner é nova e ainda não está 100% pronta para produção. Além disso, alguns aspectos dessa versão podem ser alterados sem aviso prévio. Nós encorajamos você a usá-lo e fornecer feedback para que possamos tornar esta versão tão robusta quanto uma instalação bare-metal.

# Bare-metal vs Containers

Se você estiver familiarizado com a plataforma HPCC Systems, há algumas mudanças fundamentais a serem observadas.

## Processos e pods, não máquinas

Qualquer pessoa familiarizada com o sistema de configuração existente saberá que parte da configuração envolve a criação de instâncias de cada processo e a especificação de quais máquinas físicas devem ser executadas.

Em um mundo Kubernetes, isso é gerenciado dinamicamente pelo próprio sistema K8s (e pode ser alterado dinamicamente enquanto o sistema é executado).

Além disso, um sistema em contêiner é muito mais simples de gerenciar se você adotar o paradigma de um processo por contêiner, em que as decisões sobre quais contêineres precisam ser agrupados em um pod e quais pods podem ser executados em nós físicos de maneira automática.

## Helm charts

No mundo em contêineres, as informações que o operador precisa fornecer para configurar um ambiente HPCC Systems são bastante reduzidas. Não há necessidade de especificar qualquer informação sobre quais máquinas estão em uso e por qual processo. Conforme mencionado acima, também não há necessidade de alterar muitas opções que podem ser dependentes do ambiente operacional, uma vez que muito disso foi padronizado no momento em que as imagens do contêiner foram criadas.

Portanto, na maioria dos casos, a maior parte das configurações devem ser ignoradas para usar o padrão. Como tal, o novo paradigma de configuração requer que apenas o mínimo de informações seja especificado e quaisquer parâmetros não especificados façam usos dos padrões apropriados.

O **environment.xml** padrão que incluímos em nossos pacotes bare-metal para descrever o sistema de nó único padrão contém aproximadamente 1300 linhas e é complexo o suficiente para que recomendamos o uso de uma ferramenta especial para editá-lo.

O **values.yaml** do gráfico de helm padrão é relativamente pequeno e pode ser aberto em qualquer editor e/ou modificado por meio das substituições de linha de comando do helm. Também é auto-documentado com extensos comentários.

## Static vs On-Demand Services

A fim de realizar a economia de custo potencial de um ambiente de nuvem e, ao mesmo tempo, aproveitar a escalabilidade quando necessário, alguns serviços que estão sempre ativos na tradição de instalações bare-metal são lançados sob demanda em instalações em contêiner.

Por exemplo, um componente eclccserver inicia um stub que requer recursos mínimos, onde a única tarefa é observar as workunits enviadas para compilação e lançar um job K8s independente para realizar a compilação atual.

Da mesma forma, o componente eclagent também é um stub que ativa um job K8s quando uma workunit é enviada e o stub Thor inicia um cluster apenas quando necessário. Usando esse design, não apenas a capacidade do sistema aumenta automaticamente para usar quantos pods forem necessários para lidar com a carga enviada, como também diminui para usar recursos mínimos (como uma fração de um único nó) durante os tempos de inatividade quando aguardando que os trabalhos sejam enviados.

Os componentes ESP e Dali estão sempre ligados, desde que o cluster K8s seja iniciado. Não é viável iniciá-los e interrompê-los sob demanda sem latência excessiva. No entanto, o ESP pode ser ampliado e reduzido dinamicamente para executar quantas instâncias forem necessárias para lidar com a carga atual.

## Configurações de topologia – Clusters vs filas

Em implantações bare-metal, há uma seção chamada **Topologia** onde as várias filas às quais as workunits podem ser enviadas são configuradas. É responsabilidade da pessoa que edita o ambiente garantir que cada destino nomeado tenha as instâncias eclccserver, hThor (ou ROXIE) e Thor (se desejado) configuradas, para lidar com as workunit enviadas para a fila de destino.

Essa configuração foi bastante simplificada ao usar Helm charts para configurar um sistema em contêiner. Cada Thor nomeado ou componente eclagent cria uma fila correspondente (com o mesmo nome) e cada eclccserver escuta em todas as filas por padrão (mas você pode restringir a certas filas apenas se realmente quiser). Definir um componente do Thor automaticamente garante que os componentes do agente necessários sejam provisionados.

# Deploy Local (Desenvolvimento e Teste)

Embora haja muitas maneiras de instalar uma plataforma HPCC Systems de nó único local, esta seção se concentra no uso local do Docker Desktop.

## Pré-requisitos

Todas ferramentas de terceiros devem ser 64-bits.

## Adicionar Repositório

Para usar o helm charts do HPCC Systems, você deve adicioná-lo à lista de repositório do helm, conforme mostrado abaixo:

```
helm repo add hpcc https://hpcc-systems.github.io/helm-chart/
```

Resposta esperada:

```
"hpcc" has been added to your repositories
```

Para atualizar os últimos charts:

```
helm repo update
```

Você deve atualizar seu repositório local antes de qualquer desenvolvimento, assim garante que está com o último código disponível.

Resposta esperada:

```
Hang tight while we grab the latest from your chart repositories...  
...Successfully got an update from the "hpcc" chart repository  
Update Complete. Happy Helming!
```

# Iniciar um Sistema Padrão

O helm chart padrão inicia um sistema de teste simples com Dali, ESP, ECL CC Server, duas filas ECL Agent (modo ROXIE e hThor) e uma fila Thor.

## Para iniciar este sistema simples:

```
helm install mycluster hpcc/hpcc --version=8.6.14
```

**Nota:** O argumento `--version` é opcional, mas recomendado. Ele garante que você saiba a versão que você está instalando. Se omitido, a última versão não-desenvolvimento será instalada. Este exemplo usa 8.6.14, mas você deve usar a versão que deseja.

## Resposta esperada:

```
NAME: mycluster
LAST DEPLOYED: Tue Apr 5 14:45:08 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing the HPCC chart version 8.6.14 using image "hpccsystems/platform-core:8.6.14"
**** WARNING: The configuration contains ephemeral planes: [dali sasha dll data mydropzone debug] ****

This chart has defined the following HPCC components:
dali.mydali
dfuserver.dfuserver
eclagent.hthor
eclagent.roxie-workunit
eclccserver.myeclccserver
eclscheduler.eclscheduler
esp.eclwatch
esp.eclservices
esp.eclqueries
esp.esdl-sandbox
esp.sql2ecl
esp.dfs
roxie.roxie
thor.thor
dali.sasha.coalescer
sasha.dfurecovery-archiver
sasha.dfuwu-archiver
sasha.file-expiry
sasha.wu-archiver
```

Observe o aviso sobre planos efêmeros. Isso ocorre porque essa implantação criou armazenamento temporário e efêmero para uso. Quando o cluster for desinstalado, o armazenamento não existirá mais. Isso é útil para um teste rápido, mas para um trabalho mais complexo, você desejará um armazenamento mais persistente. Isso é abordado em uma seção posterior.

## Para verificar o status:

```
kubectl get pods
```

## Resposta esperada:

| NAME                        | READY | STATUS  | RESTARTS | AGE  |
|-----------------------------|-------|---------|----------|------|
| eclqueries-7fd94d77cb-m7lmb | 1/1   | Running | 0        | 2m6s |
| eclservices-b57f9b7cc-bhwtm | 1/1   | Running | 0        | 2m6s |



Containerized HPCC Systems® Platform  
Deploy Local (Desenvolvimento e Teste)

|   |     |         |   |      |
|---|-----|---------|---|------|
| eclwatch-599fb7845-2hq54                    | 1/1 | Running | 0 | 2m6s |
| esdl-sandbox-848b865d46-9bv9r               | 1/1 | Running | 0 | 2m6s |
| hthor-745f598795-ql9dl                      | 1/1 | Running | 0 | 2m6s |
| mydali-6b844bfcfb-jv7f6                     | 2/2 | Running | 0 | 2m6s |
| myeclccserver-75bcc4d4d-gflfs               | 1/1 | Running | 0 | 2m6s |
| roxie-agent-1-77f696466f-tl7bb              | 1/1 | Running | 0 | 2m6s |
| roxie-agent-1-77f696466f-xzrtf              | 1/1 | Running | 0 | 2m6s |
| roxie-agent-2-6dd45b7f9d-m22wl              | 1/1 | Running | 0 | 2m6s |
| roxie-agent-2-6dd45b7f9d-xmlmk              | 1/1 | Running | 0 | 2m6s |
| roxie-toposerver-695fb9c5c7-9lnp5           | 1/1 | Running | 0 | 2m6s |
| roxie-workunit-d7446699f-rvf2z              | 1/1 | Running | 0 | 2m6s |
| sasha-dfurecovery-archiver-78c47c4db7-k9mdz | 1/1 | Running | 0 | 2m6s |
| sasha-dfuwu-archiver-576b978cc7-b47v7       | 1/1 | Running | 0 | 2m6s |
| sasha-file-expiry-8496d87879-xct7f          | 1/1 | Running | 0 | 2m6s |
| sasha-wu-archiver-5f64594948-xjblh          | 1/1 | Running | 0 | 2m6s |
| sql2ecl-5c8c94d55-tj4td                     | 1/1 | Running | 0 | 2m6s |
| dfs-4a9f12621-jabc1                         | 1/1 | Running | 0 | 2m6s |
| thor-eclagent-6b8f564f9c-qnczz              | 1/1 | Running | 0 | 2m6s |
| thor-thoragent-56d788869f-7trxk             | 1/1 | Running | 0 | 2m6s |

**Observação:** Pode demorar um pouco antes de todos os componentes estarem em execução, especialmente na primeira vez, pois as imagens do contêiner precisam ser baixadas do Docker Hub.

## Acesso padrão do sistema

Seu sistema agora está pronto para uso. O primeiro passo usual é abrir o ECL Watch.

**Observação:** Algumas páginas no ECL Watch, como aquelas que exibem informações de topologia, ainda não estão totalmente funcionais no modo em contêiner.

Use este comando para obter uma lista de serviços em execução e endereços IP:

```
kubectl get svc
```

Resposta esperada:

| NAME                 | TYPE         | CLUSTER-IP     | EXTERNAL-IP      | PORT(S)               | AGE  |
|----------------------|--------------|----------------|------------------|-----------------------|------|
| eclqueries           | LoadBalancer | 10.108.171.35  | localhost        | 8002:31615/TCP        | 2m6s |
| eclservices          | ClusterIP    | 10.107.121.158 | <none>           | 8010/TCP              | 2m6s |
| <b>eclwatch</b>      | LoadBalancer | 10.100.81.69   | <b>localhost</b> | <b>8010:30173/TCP</b> | 2m6s |
| esdl-sandbox         | LoadBalancer | 10.100.194.33  | localhost        | 8899:30705/TCP        | 2m6s |
| kubernetes           | ClusterIP    | 10.96.0.1      | <none>           | 443/TCP               | 2m6s |
| mydali               | ClusterIP    | 10.102.80.158  | <none>           | 7070/TCP              | 2m6s |
| roxie                | LoadBalancer | 10.100.134.125 | localhost        | 9876:30480/TCP        | 2m6s |
| roxie-toposerver     | ClusterIP    | None           | <none>           | 9004/TCP              | 2m6s |
| sasha-dfuwu-archiver | ClusterIP    | 10.110.200.110 | <none>           | 8877/TCP              | 2m6s |
| sasha-wu-archiver    | ClusterIP    | 10.111.34.240  | <none>           | 8877/TCP              | 2m6s |
| sql2ecl              | LoadBalancer | 10.107.177.180 | localhost        | 8510:30054/TCP        | 2m6s |
| dfs                  | LoadBalancer | 10.100.52.9    | localhost        | 8520:30184/TCP        | 2m6s |

Localize o serviço ECL Watch e identifique o EXTERNAL-IP e PORTA(S) para eclwatch. Neste caso, é localhost:8010.

Abra um navegador e acesse o ECLWatch, pressione o botão ECL e selecione a aba Playground.

A partir daqui, você pode usar o ECL de exemplo ou inserir outras consultas de teste e escolher entre os clusters disponíveis para enviar suas workunit.

## Encerrar (Descomissionar) o Sistema

Para verificar quais helm charts estão instalados atualmente, execute este comando:

```
helm list
```

Isso exibe os gráficos instalados e seus nomes. Neste exemplo, mycluster.

Para interromper os pods do HPCC Systems, use o helm para desinstalar:

```
helm uninstall mycluster
```

Isso interrompe o cluster, exclui os pods e, com as configurações padrão e os volumes persistentes, também exclui o armazenamento usado.

# Persistent Storage para um Deploy local

Ao executar em um sistema de teste de nó único, como o Docker Desktop, a classe de armazenamento padrão normalmente significa que todas as declarações de volume persistente (PVCs) mapeiam para diretórios locais temporários na máquina host. Normalmente, eles são removidos quando o cluster é interrompido. Isso é bom para testes simples, mas para qualquer aplicativo real, você deseja armazenamento persistente.

Para manter os dados com uma implantação do Docker Desktop, a primeira etapa é garantir que os diretórios relevantes existam:

1. Crie diretório de dados utilizando uma janela de terminal:

Para Windows, use este comando:

```
mkdir c:\hpccdata
mkdir c:\hpccdata\dalistorage
mkdir c:\hpccdata\hpcc-data
mkdir c:\hpccdata\debug
mkdir c:\hpccdata\queries
mkdir c:\hpccdata\sasha
mkdir c:\hpccdata\dropzone
```

Para macOS, use este comando:

```
mkdir -p /Users/myUser/hpccdata/{dalistorage,hpcc-data,debug,queries,sasha,dropzone}
```

Para Linux, use este comando:

```
mkdir -p ~/hpccdata/{dalistorage,hpcc-data,debug,queries,sasha,dropzone}
```

**Nota:** Se todos esses diretórios não existirem, seus pods podem não iniciarem.

2. Instale o hpcc-localfile Helm chart.

Este chart cria volumes persistentes com base nos diretórios de host que você criou anteriormente.

```
# for a WSL2 deployment:
helm install hpcc-localfile hpcc/hpcc-localfile --set common.hostpath=/run/desktop/mnt/host/c/hpccdata

# for a Hyper-V deployment:
helm install hpcc-localfile hpcc/hpcc-localfile --set common.hostpath=/c/hpccdata

# for a macOS deployment:
helm install hpcc-localfile hpcc/hpcc-localfile --set common.hostpath=/Users/myUser/hpccdata

# for a Linux deployment:
helm install hpcc-localfile hpcc/hpcc-localfile --set common.hostpath=~/hpccdata
```

**--set common.hostpath=** opção que especifica o diretório base:

O caminho **/run/desktop/mnt/host/c/hpccdata** provê acesso ao arquivo de host para WSL2.

O caminho **/c/hpccdata** provê acesso ao arquivo de host para Hyper-V.

O caminho **/Users/myUser/hpccdata** provê acesso ao arquivo de host para Mac OSX.

O caminho **~/hpccdata** provê acesso ao arquivo de host para Linux.

**Nota:** O valor do `--set common-hostpath` é *case sensitive*.

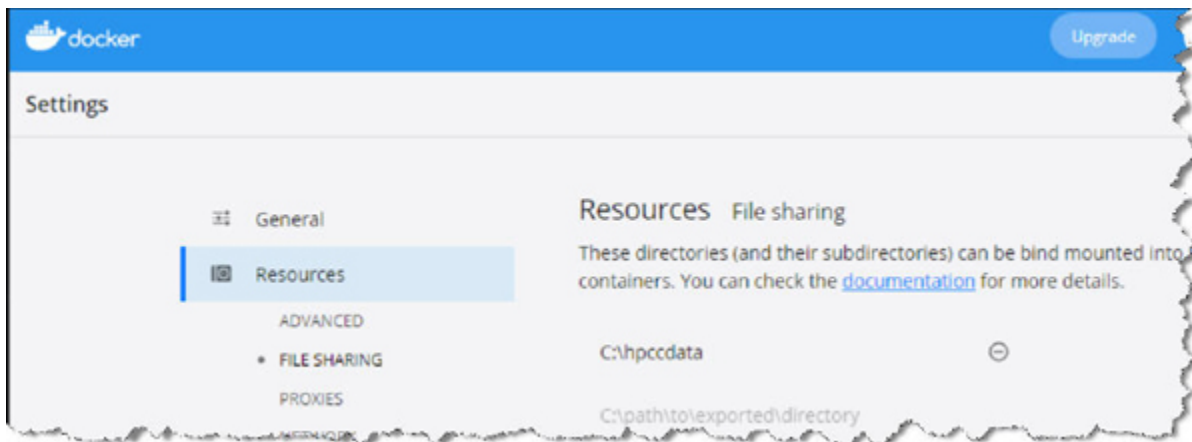
3. No comando de instalação do helm, selecione desde a palavra **storage**: até o final e salve-a em um arquivo de texto.

Neste exemplo, nós vamos chamar o arquivo `mystorage.yaml`. O arquivo deve se parecer com este aqui:

```
storage:
  planes:
    - name: dali
      pvc: dali-hpcc-localfile-pvc
      prefix: "/var/lib/HPCCSystems/dalistorage"
      category: dali
    - name: dll
      pvc: dll-hpcc-localfile-pvc
      prefix: "/var/lib/HPCCSystems/queries"
      category: dll
    - name: sasha
      pvc: sasha-hpcc-localfile-pvc
      prefix: "/var/lib/HPCCSystems/sasha"
      category: sasha
    - name: debug
      pvc: debug-hpcc-localfile-pvc
      prefix: "/var/lib/HPCCSystems/debug"
      category: debug
    - name: data
      pvc: data-hpcc-localfile-pvc
      prefix: "/var/lib/HPCCSystems/hpcc-data"
      category: data
    - name: mydropzone
      pvc: mydropzone-hpcc-localfile-pvc
      prefix: "/var/lib/HPCCSystems/dropzone"
      category: lz
sasha:
  wu-archiver:
    plane: sasha
  dfuwu-archiver:
    plane: sasha
```

4. Se você estiver utilizando Docker Desktop com Hyper-V, adicione o diretório compartilhado (neste exemplo, `C:\hpccdata`) nas configurações do Docker Desktop clicando no botão Add e digitando `c:\hpccdata`.

Isto **não** é necessário em ambientes MacOS ou WSL 2.



5. Por fim, instale o chart hpcc Helm e forneça um arquivo yaml que contenha as informações de armazenamento criadas na etapa anterior

```
helm install mycluster hpcc/hpcc --version=8.6.14 -f mystorage.yaml
```

**Nota:** O argumento `--version` é opcional, mas recomendado. Ele garante que você saiba qual versão está instalando. Se omitido, a versão sem desenvolvimento mais recente será instalada. Este exemplo usa 8.6.14, mas você deve usar a versão desejada.

6. Para testar, abra um navegador e acesse o ECLWatch, pressione o botão ECL e selecione a aba Playground, então crie alguns arquivos de dados e workunits enviando ao Thor algum código ECL como o seguinte:

```
LayoutPerson := RECORD
  UNSIGNED1 ID;
  STRING15  FirstName;
  STRING25  LastName;
END;
allPeople := DATASET([ {1,'Fred','Smith'},
                       {2,'Joe','Jones'},
                       {3,'Jane','Smith'}],LayoutPerson);
OUTPUT(allPeople, 'MyData::allPeople', THOR, OVERWRITE);
```

7. Use o comando de desinstalação do helm para encerrar seu cluster e reinicie seu deploy.
8. Abra o ECL Watch e observe que suas workunits e arquivos lógicos ainda estão lá.

# Importar: Planos de armazenamento e como usá-los

Os planos de armazenamento oferecem a flexibilidade de configurar onde os dados são armazenados em uma plataforma HPCC Systems implantada, mas não aborda diretamente a questão de como colocar os dados na plataforma em primeiro lugar.

As plataformas em contêineres oferecem suporte à importação de dados de duas maneiras:

- Upload do arquivo para uma Landing Zone e Importe (Spray)
- Copie um arquivo para o Plano de Armazenamento e acesse diretamente

A partir da versão 7.12.0, uma nova sintaxe ECL foi adicionada para acessar arquivos diretamente de um plano de armazenamento. Isso é semelhante à sintaxe **file::** usada para ler arquivos diretamente de uma máquina física, geralmente uma landing zone.

A nova sintaxe é:

```
~plane::<storage-plane-name>::<path>::<filename>
```

Onde a sintaxe do caminho e do nome do arquivo são as mesmas usadas com a sintaxe **file::**. Isso inclui exigir que letras maiúsculas sejam citadas com um símbolo ^. Para obter mais detalhes, consulte a seção Arquivos da Landing Zone do documento *Referência a Linguagem ECL*.

Se você tiver planos de armazenamento configurados como na seção anterior e copiar o arquivo **originalperson** para **C:\hpccdata\hpcc-data\tutorial**, poderá fazer referência ao arquivo usando esta sintaxe:

```
'~plane::data::tutorial::originalperson'
```

**Nota:** O arquivo **originalperson** está disponível no site do HPCC Systems Web ([https://cdn.hpccsystems.com/install/docs/3\\_8\\_0\\_8rc\\_CE/OriginalPerson](https://cdn.hpccsystems.com/install/docs/3_8_0_8rc_CE/OriginalPerson)).

# Configurações Customizadas

## Técnicas de Customização

Nesta seção, nós vamos abordar a criação de uma configuração customizada do arquivo YAML e do lançamento de uma plataforma HPCC Systems® utilizando os configurações padrão mas as customizações. Depois de entender os conceitos de deste capítulo, você pode consultar o próximo para uma referência a todos as configurações de valores de configuração.

Há várias maneiras de personalizar uma implantação da plataforma. Nós recomendamos o uso de métodos que permitem que você aproveite melhor o configuração como práticas de código (CaC). A configuração como código é a padrão de gerenciamento de arquivos de configuração em um sistema de controle de versão ou repositório.

A seguir uma lista de técnicas de customização:

- A primeira maneira de substituir uma configuração padrão é através da linha de comando usando o parâmetro **--set**.

Este é o mais fácil, mas o menos compatível com as diretrizes CaC. Também é mais difícil de rastrear as alterações.

- A segunda maneira é modificando os valores padrão e salvando por meio da seguinte linha de comando:

```
helm show values hpcc/hpcc > myvalues.yaml
```

Isso pode estar em conformidade com as diretrizes do CaC, se você colocar esse arquivo sob controle de versão, mas dificulta a utilização de uma nova configuração padrão quando uma estiver disponível.

- A terceira maneira, é a que normalmente usamos. Usar o padrão configuração, mais um arquivo YAML de personalização e o parâmetro **-f** (ou parâmetro **--values**) para o comando helm. Isso usa o padrão configuração e apenas substitui as configurações especificadas no personalização YAML. Além disso, você pode passar vários arquivos YAML no mesmo comando, se desejado.

Para este tutorial, usaremos o terceiro método para levantar um plataforma com todas as configurações padrão, mas adicionar algumas personalizações. No primeiro exemplo, em vez de um Roxie, teremos dois. No segundo exemplo, ele adicionará um segundo 10-way Thor.



## Criar um Custom Configuration Chart para Dois Roxies

1. Se você ainda não adicionou o repositório HPCC Systems a sua lista de repositórios do helm, adicione-o agora.

```
helm repo add hpcc https://hpcc-systems.github.io/helm-chart/
```

Se você já adicionou, atualize para os últimos charts:

```
helm repo update
```

2. Crie um novo arquivo de texto, o nomeie para **tworoxies.yaml** e abra-o em um editor de texto.

Você pode usar qualquer editor de texto.

3. Salve os valores padrão em um arquivo de texto:

```
helm show values hpcc/hpcc > myvalues.yaml
```

4. Abra o arquivo salvo (myvalues.yaml) em um editor de texto.
5. Copie toda seção **roxie**: e cole dentro no novo arquivo **tworoxies.yaml**.
6. Copie todo o conteúdo do novo arquivo **tworoxies.yaml**, exceto a primeira linha (roxie:), e cole no final do arquivo.
7. No primeiro bloco, edite o valor para **name**: e altere o valor para **roxie2**.
8. No segundo bloco, edite o valor para **prefix**: e altere para **roxie2**.
9. No segundo bloco, edite o valor para **name**: abaixo de **services**: e altere para **roxie2**.
10. Salve o arquivo e feche o editor de texto.

O arquivo **tworoxies.yaml** file deve se parecer com este aqui:

**Nota:** Os comentários foram removidos para simplificar o exemplo:

```
roxie:
- name: roxie
  disabled: false
  prefix: roxie
  services:
    - name: roxie
      servicePort: 9876
      listenQueue: 200
      numThreads: 30
      visibility: local
  replicas: 2
  numChannels: 2
  serverReplicas: 0
  localAgent: false
  traceLevel: 1
  topoServer:
    replicas: 1
- name: roxie2
  disabled: false
  prefix: roxie2
  services:
    - name: roxie2
```

```
servicePort: 9876
listenQueue: 200
numThreads: 30
visibility: local
replicas: 2
numChannels: 2
serverReplicas: 0
localAgent: false
traceLevel: 1
topoServer:
  replicas: 1
```

### Deploy utilizando um novo chart de configuração personalizado.

1. Abra uma janela de terminal e navegue para o diretório onde você salvou o arquivo `tworoxies.yaml`.
2. Faça o deploy do seu HPCC Systems Platform, adicionando a nova configuração ao comando:

```
helm install mycluster hpcc/hpcc -f tworoxies.yaml
```

3. Após você confirmar que seu deploy está sendo executado, abra o ECL Watch.

Você deverá ver dois clusters Roxie disponíveis como Targets -- roxie e roxie2.

## Crie um Novo Chart de Configuração para Dois Thors

Você pode especificar mais de uma configuração de customização repetindo o parâmetro `-f`.

Por exemplo:

```
helm install mycluster hpcc/hpcc -f tworoxies.yaml -f twothors.yaml
```

Nesta seção, nós vamos adicionar um segundo Thor 10-way.

1. Se você ainda não adicionou o repositório do HPCC Systems a lista de repositórios helm, adicione agora.

```
helm repo add hpcc https://hpcc-systems.github.io/helm-chart/
```

Se você já adicionou, atualize os últimos charts:

```
helm repo update
```

2. Crie um novo arquivo de texto e nomeie-o **twothors.yaml**, em seguida abra em um editor de texto.

Você pode usar qualquer editor de texto.

3. Em um editor de texto, abra o arquivo de valores padrão que você salvou anteriormente (`myvalues.yaml`).
4. Copie por inteiro a seção **thor**: e cole no novo arquivo `twothors.yaml`.
5. Copie todo conteúdo para o novo arquivo `twothors.yaml`, exceto a primeira linha (`thor:`), e cole no final do arquivo.
6. No segundo bloco, edite o valor para **name**: e altere-o para **thor10**.
7. No segundo bloco, edite o valor para **prefix**: e altere-o para **thor10**.
8. No segundo bloco, edite o valor para **numWorkers**: e altere-o para **10**.
9. Salve o arquivo e feche o editor de texto.

O resultado do arquivo do twothors.yaml deve se parecer assim

**Nota:** Os comentários foram removidos para simplificar o exemplo:

```
thor:
- name: thor
  prefix: thor
  numWorkers: 2
  maxJobs: 4
  maxGraphs: 2
- name: thor10
  prefix: thor10
  numWorkers: 10
  maxJobs: 4
  maxGraphs: 2
```

### Deploy utilizando o novo chart de configuração personalizado.

1. Abra uma janela de terminal e navegue para o diretório onde você salvou o arquivo twothors.yaml.
2. Faça o deploy do seu HPCC Systems Platform, adicionando a nova configuração ao comando:

```
# If you have previously stopped your cluster

helm install mycluster hpcc/hpcc -f tworoxies.yaml -f twothors.yaml

# To upgrade without stopping

helm upgrade mycluster hpcc/hpcc -f tworoxies.yaml -f twothors.yaml
```

3. Após você confirmar que seu deploy está sendo executado, abra o ECL Watch.

Você deverá ver dois clusters Thor disponíveis como Targets -- thor and thor10.

# Configuração dos Valores

Este capítulo descreve a configuração do HPCC Systems para uma implantação Kubernetes em contêineres. As seções a seguir detalham como as configurações são fornecidas aos charts do helm, como descobrir quais opções estão disponíveis e alguns detalhes da estrutura do arquivo de configuração. As seções subsequentes também fornecerão uma breve explicação de alguns dos conteúdos do arquivo padrão *values.yaml*, usado na configuração do HPCC Systems para uma implantação em contêiner.

## O Ambiente do Contêiner

Uma das ideias por trás de nossa mudança para a nuvem foi tentar simplificar a configuração do sistema e, ao mesmo tempo, fornecer uma solução flexível o suficiente para atender às demandas de nossa comunidade, aproveitando os recursos do contêiner sem sacrificar o desempenho.

Toda a configuração do HPCC Systems no espaço do contêiner é governada por um único arquivo, um arquivo *values.yaml* e seu arquivo de esquema associado.

## O *values.yaml* e como é utilizado

O arquivo *values.yaml* são os valores de configuração fornecidos para um chart do Helm. O arquivo *values.yaml* é usado pelo chart do Helm para controlar como o HPCC Systems é implantado na nuvem. Esse arquivo de valores é usado para configurar e obter uma instância do HPCC Systems em execução no Kubernetes. O arquivo *values.yaml* define tudo o que acontece para configurar e/ou definir seu sistema para uma implantação em contêiner. Você deve usar o arquivo de valores fornecido como base para modelar as personalizações específicas para sua implantação de acordo com seus requisitos.

O arquivo *values.yaml* do HPCC Systems pode ser encontrado no repositório github do HPCC Systems. Para usar o chart Helm do HPCC Systems, primeiro adicione o repositório de charts hpcc usando o Helm e, em seguida, acesse os valores do chart Helm dos charts nesse repositório.

Por exemplo, ao adicionar o repositório "hpcc", conforme recomendado antes de instalar o chart do Helm com o seguinte comando:

```
helm repo add hpcc https://hpcc-systems.github.io/helm-chart
```

Agora você pode visualizar os charts entregues do HPCC Systems e ver os valores lá emitindo:

```
helm show values hpcc/hpcc
```

Você pode capturar a saída deste comando, ver como os padrões são configurados e usá-lo como base para sua customização.

## O values-schema.json

O *values-schema.json* é um arquivo JSON que declara o que é válido e o que não está dentro da soma total dos valores mesclados que são passados para o Helm no momento da instalação. Ele define quais valores são permitidos e valida o arquivo de valores em relação a eles. Todos os itens principais são declarados no arquivo de esquema, enquanto o arquivo default *values.yaml* também contém comentários sobre os elementos mais importantes. Se você quiser saber quais opções estão disponíveis para qualquer componente específico, o esquema é um bom lugar para começar.

O arquivo de esquema normalmente contém (para uma propriedade) um nome e uma descrição. Muitas vezes, incluirá detalhes do tipo e os itens que pode conter se for uma lista ou dicionário. Por exemplo:

```
"roxie": {
  "description": "roxie process",
  "type": "array"
  "items": { "$ref": "#/definitions/roxie" }
},
```

Cada plano, no arquivo de esquema, tem uma lista de propriedades geralmente contendo um prefixo (caminho), um subcaminho (subcaminho) e propriedades adicionais. Por exemplo, para um plano de armazenamento, o arquivo de esquema possui uma lista de propriedades, incluindo o prefixo. Os "planos" neste caso são uma referência (\$ref) para outra seção do esquema. O arquivo de esquema deve ser completo e conter tudo o que é necessário, incluindo descrições que devem ser relativamente autoexplicativas.

```
"storage": {
  "type": "object",
  "properties": {
    "hostGroups": {
      "$ref": "#/definitions/hostGroups"
    },
    "planes": {
      "$ref": "#/definitions/storagePlanes"
    }
  },
  "additionalProperties": false
```

Observe o valor de *additionalProperties* normalmente no final de cada seção no esquema. Ele especifica se os valores permitem propriedades adicionais ou não. Se esse valor *additionalProperties* estiver presente e definido como false, nenhuma outra propriedade será permitida e a lista de propriedades estará completa.

Ao trabalhar com o HPCC Systems *values.yaml*, o arquivo de valores deve ser validado em relação a esse esquema. Se houver um valor que não seja permitido conforme definido no arquivo de esquema, ele não será iniciado e, em vez disso, gerará um ERRO.

## Componentes HPCC Systems no Arquivo values.yaml

Os chart do Helm do HPCC Systems são enviados com valores de estoque/padrão. Esses charts do Helm têm um conjunto de valores padrão idealmente para serem usados como guia na configuração de sua implantação. Geralmente, cada componente do HPCC Systems é uma lista. Essa lista define as propriedades para cada instância do componente.

Esta seção fornecerá detalhes adicionais e qualquer percepção digna de nota para os componentes do HPCC Systems definidos no arquivo *values.yaml*.

## Os Componentes do HPCC Systems

Uma das principais diferenças entre o bare metal e o contêiner/nuvem é que o armazenamento bare metal está diretamente vinculado aos nós do job trabalho Thor ou Thor e aos nós de trabalho Roxie, ou mesmo no caso do servidor ECLCC as DLLs. Nos contêineres, eles são completamente separados e qualquer coisa relacionada a arquivos é definida no arquivo *values.yaml*

Em contêineres, as instâncias de componentes são executadas dinamicamente. Por exemplo, se você configurou seu sistema para usar um Thor de 50 vias, então um Thor de 50 vias será gerado quando um trabalho for enfileirado para ele. Quando esse trabalho for concluído, a instância Thor desaparecerá. Este é o mesmo padrão para os outros componentes também.

Cada componente deve ter uma entrada de recursos, no arquivo *values.yaml* entregues os recursos estão presentes, mas comentados conforme indicado aqui.

```
#resources:  
#  cpu: "1"  
#  memory: "4G"
```

O arquivo de valores de estoque funcionará e permitirá que você mantenha um sistema funcional, porém você deve definir os recursos dos componentes da maneira que melhor corresponda à sua estratégia operacional.

## Os serviços do Sistema

A maioria dos componentes do HPCC Systems tem uma entrada de definição de serviço, semelhante à entrada de recursos. Todos os componentes que possuem definições de serviço seguem esse mesmo padrão.

Qualquer informação relacionada ao serviço precisa estar em um objeto de serviço, por exemplo:

```
service:  
  servicePort: 7200  
  visibility: local
```

Isso se aplica à maioria dos componentes do HPCC Systems, ESP, Dali, dafilesrv e Sasha. A especificação do Roxie é um pouco diferente, pois tem seu serviço definido em "roxieservice". Cada Roxie pode ter várias definições de "roxieservice". (ver esquema).

## Dali

Ao configurar o Dali, que também possui uma seção de recursos, ele também precisará de muita memória e uma boa quantidade de CPU. É muito importante defini-los com cuidado. Caso contrário, o Kubernetes pode atribuir todos os pods à mesma máquina virtual e os componentes que lutam pela memória os esmagarão. Portanto, mais memória atribuída melhor. Se você definir isso errado e um processo usar mais memória do que o configurado, o Kubernetes matará o pod.

## Componentes: dafilesvrs, dfuserver

Os componentes do HPCC Systems de dafilesvrs, eclccservers, dfuserver, são declarados como listas no yaml, assim como o ECL Agent.

Considere o dfuserver que está nos *values.yaml* entregues do HPCC Systems como:

```
dfuserver:  
- name: dfuserver  
  maxJobs: 1
```

Se você adicionar um mydfuserver da seguinte maneira:

```
dfuserver:
- name: dfuserver
  maxJobs: 1
- name: mydfuserver
  maxJobs: 1
```

Nesse cenário, você teria outro item aqui chamado mydfuserver, ele apareceria no ECLWatch e você poderia enviar itens para ele.

Se você quiser adicionar outro dfuserver, poderá adicioná-lo à lista da mesma forma. Você também pode instanciar outros componentes adicionando-os às suas respectivas listas.

## ECL Agent e ECLCC Server

Values of note for the ECL Agent and ECLCC Server.

**useChildProcess** -- Conforme definido no esquema, iniciada cada compilação da workunit como um processo secundário em vez de em seu próprio contêiner. Quando você envia um job ou consulta para compilar, ele é enfileirado e processado, com essa opção definida como true, ele gerará um processo secundário utilizando quase nenhuma sobrecarga adicional na inicialização. Ideal para enviar muitos jobs pequenos para compilar. No entanto, como cada job de compilação não é mais executado como um pod independente com suas próprias especificações de recursos, mas é executado como um processo secundário no próprio pod do servidor ECLCC, o pod do servidor ECLCC deve ser definido com recursos adequados para si mesmo (mínimo para ouvir para a fila etc.) e todos os jobs que ele possa ter que executar em paralelo.

Por exemplo, imagine que *maxJobs* está definido como 4 e 4 consultas grandes são enfileiradas rapidamente, o que significa que 4 processos secundário são iniciados, cada cpu consumindo e memória dentro do pod do servidor ECLCC. Com o componente configurado com *useChildProcesses* definido como true, cada trabalho será executado no mesmo pod (até o valor de *maxJobs* em paralelo). Portanto, com *useChildProcesses* habilitado, os recursos do componente devem ser definidos de forma que o pod tenha recursos suficientes para lidar com as demandas de recursos de todos esses trabalhos para poder ser executado em paralelo.

Com *useChildProcess* ativado, pode ser bastante caro na maioria dos modelos de preços de nuvem e bastante dispendioso se não houver nenhum job em execução. Em vez disso, você pode definir esse *useChildprocess* como false (o padrão) para iniciar um pod para compilar cada consulta apenas com a memória necessária para o trabalho que será descartado quando concluído. Agora, esse modelo também ouviu, talvez 20 segundos a um minuto para gerar o cluster Kubernetes para processar o trabalho. O que pode não ser ideal para um ambiente que está enviando vários trabalhos pequenos, mas sim jobs maiores que minimizariam o efeito da sobrecarga ao iniciar o cluster Kubernetes.

Definir *useChildProcess* como false permite melhor a possibilidade de dimensionamento dinâmico. Para jobs que levariam muito tempo para compilar, a sobrecarga extra (inicialização) é mínima, e esse seria o caso ideal para ter o *useChildProcess* como falso. Definir *useChildProcess* como false permite apenas 1 pod por compilação, embora haja um atributo para colocar um limite de tempo nessa compilação.

**ChildProcessTimeLimit** é o tempo limite (em segundos) para compilação de processos secundários antes de abortarem e usarem um contêiner separado, quando o *useChildProcesses* é false.

**maxActive** -- O número máximo de jobs que podem ser executadas em paralelo. Novamente, tome cuidado porque cada job precisará de memória suficiente para ser executado. Por exemplo, se *maxActive* estiver definido como 2000, você poderá enviar um trabalho muito grande e, nesse caso, gerar cerca de 2.000 trabalhos usando uma quantidade considerável de recursos, o que poderia gerar uma conta de compilação bastante cara, novamente dependendo do seu provedor de nuvem e seu plano de faturamento.

## Sasha

A configuração para Sasha é uma exceção, pois é uma estrutura do tipo dicionário e não uma lista. Você não pode ter mais de um arquivador ou dfuwu-archiver, pois isso é uma limitação de valor, você pode optar por ter o serviço ou não (defina o valor 'disabled' como true).

## Thor

As instâncias Thor são executadas dinamicamente, assim como os outros componentes em contêineres. A configuração do Thor também consiste em uma lista de instâncias do Thor. Cada instância gera dinamicamente uma coleção de pods (manager + N workers) quando os workers são enfileirados para ela. Quando ocioso, não há pods de worker (ou manager) em execução.

Se você quisesse um Thor de 50 vias, você definiria o número de workers, o valor **numWorkers** para 50 e você teria um Thor de 50 vias. Conforme indicado no exemplo a seguir:

```
thor:
- name: thor
  prefix: thor
  numWorkers: 50
```

Ao fazer isso, o ideal é renomear o recurso para algo que o descreva claramente, como *thor\_50* como no exemplo a seguir.

```
-name: thor_50
```

A atualização do valor *numWorkers* reiniciará o agente Thor ouvindo a fila, fazendo com que todos os novos jobs usem a nova configuração.<sup>3</sup>

**maxJobs** -- Controla o número de jobs, especificamente *maxJobs* define o número máximo de jobs.

**maxGraphs** -- Limita a quantidade máxima de charts. Geralmente faz sentido manter esse valor abaixo ou no mesmo número de *maxJobs*, pois nem todos os jobs enviam charts e quando fazem os jobs Thor não estão executando charts o tempo todo. Se houver mais de 2 charts enviados (Thor), o segundo será bloqueado até que a próxima instância Thor fique disponível.

A ideia aqui é que os jobs podem passar uma quantidade significativa de tempo fora dos charts, como aguardar um estado de fluxo de trabalho (fora do próprio mecanismo Thor), bloqueado em uma persistência ou atualizando super arquivos etc. ter um limite maior de trabalhos simultâneos (*maxJobs*) do que gráficos (instâncias *maxGraphs* / Thor). Como as instâncias Thor (charts) são relativamente caras (muitos pods/ maior uso de recursos), enquanto os pods de fluxo de trabalho (jobs) são comparativamente baratos.

Assim, os valores de charts entregues (exemplo) definem *maxJobs* como maior que *maxGraphs*. Os jobs enfileirados para um Thor nem sempre estão executando charts. Portanto, pode fazer sentido ter mais desses trabalhos, que não consomem um Thor grande e todos os seus recursos, mas restringem o número máximo de instâncias do Thor em execução.

Thor têm 3 componentes (o que corresponde as seções de recurso).

1. Workflow
2. Manager
3. Workers

O Manager e os Workers são lançados juntos e normalmente consomem bastante recursos (e nós). Enquanto o Workflow é barato e geralmente não requer tantos recursos. Você pode esperar em um mundo



Kubernetes, muitos deles coexistiriam no mesmo nó (e, portanto, seriam baratos). Portanto, faz sentido que *maxJobs* seja maior e *maxGraphs* seja menor

No Kubernetes, os jobs são executados de forma independente em seus próprios pods. Enquanto no bare metal, podemos ter jobs que podem afetar outros trabalhos porque estão sendo executados no mesmo espaço de processo.

## Memórias Thor e hThor

As seções de *memory* Thor e hThor permitem que a memória de recursos do componente seja refinada em diferentes áreas.

Por exemplo, o "workerMemory" para um Thor é definido como:

```
thor:
- name: thor
  prefix: thor
  numWorkers: 2
  maxJobs: 4
  maxGraphs: 2
  managerResources:
    cpu: "1"
    memory: "2G"
  workerResources:
    cpu: "4"
    memory: "4G"
  workerMemory:
    query: "3G"
    thirdParty: "500M"
  eclAgentResources:
    cpu: "1"
    memory: "2G"
```

A seção "*workerResources*" informará ao Kubernetes para recursos 4G por pod de worker. Por padrão, o Thor reservará 90% dessa memória para usar na memória de consulta HPCC (roxiemem). Os 10% restantes são deixados para todos os outros usos não baseados em linha (roxiemem), como heap geral, sobrecarga do sistema operacional etc. Não há permissão para qualquer biblioteca de terceiros, plug-ins ou uso de linguagem incorporada dentro desse padrão. Em outras palavras, se, por exemplo, o python incorporado alocar 4G, o processo logo falhará com um erro de falta de memória, quando começar a usar qualquer memória, pois esperava que 90% desse 4G estivesse disponível gratuitamente para uso próprio.

Esses padrões podem ser substituídos pelas seções de memória. Neste exemplo, *workerMemory.query* define que 3G da memória com recursos disponíveis deve ser atribuído à memória de consulta e 500M para usos de "thirdParty".

Isso limita o uso de roxiemem de memória do HPCC Systems para exatamente 3G, deixando 1G livre para outros propósitos. O "thirdParty" não é realmente alocado, mas é usado apenas como parte do total em execução, para garantir que a configuração não especifique um total nesta seção maior que a seção de recursos, por exemplo, se "thirdParty" foi definido como "2G" na seção acima, haveria uma reclamação de tempo de execução quando Thor executasse que a definição excedeu o limite de recursos.

Também é possível substituir a porcentagem recomendada padrão (90% por padrão), definindo *maxMem-Percentage*. Se "query" não estiver definida, ela será calculada como a memória máxima recomendada menos a memória definida (por exemplo, "thirdParty").

No Thor existem 3 áreas de recursos, *eclAgent*, *ThorManager* e *ThorWorker(s)*. Cada um tem uma área \*Resource que define suas necessidades de recursos do Kubernetes e uma seção \*Memory correspondente que pode ser usada para substituir os requisitos de alocação de memória padrão.

Essas configurações também podem ser substituídas por consulta, por meio de opções de workunits seguindo o padrão: <memory-section-name>.<property>. Por exemplo: #option('workerMemory.thirdParty', "1G");

**Nota:** Atualmente, há apenas "consulta" (uso de HPCC roxiemem) e "thirdParty" para todos/qualquer uso de terceiros. É possível que outras categorias sejam adicionadas no futuro, como "python" ou "java" - que definem especificamente os usos de memória para esses destinos.

# O arquivo HPCC Systems *values.yaml*

O arquivo HPCC systems *values.yaml* entregue é mais um exemplo que fornece uma configuração de tipo básico que deve ser personalizada para suas necessidades específicas. Uma das principais ideias por trás do arquivo de valores é poder personalizá-lo com relativa facilidade para seu cenário específico. O chart entregue é configurado para ser sensato o suficiente para ser entendido, ao mesmo tempo em que permite uma personalização relativamente fácil para configurar um sistema de acordo com seus requisitos específicos. Esta seção examinará mais de perto alguns aspectos do arquivo *values.yaml*.

O arquivo HPCC Systems Values entregue consiste principalmente nas seguintes áreas:

|             |              |              |
|-------------|--------------|--------------|
| global      | storage      | visibilities |
| data planes | certificates | security     |
| secrets     | components   |              |

As seções subsequentes examinarão alguns deles mais de perto e por que cada um deles está lá.

## Armazenamento

O armazenamento em contêiner é outro conceito-chave que difere do bare metal. Existem algumas diferenças entre contêiner e armazenamento em metal. A seção Storage é bastante bem definida entre o arquivo de esquema e o *values.yaml*. Uma boa abordagem para armazenamento é entender claramente suas necessidades de armazenamento e descrevê-las, e uma vez que você tenha essa estrutura básica em mente, o esquema pode ajudar a preencher os detalhes. O esquema deve ter uma descrição decente para cada atributo. Todo o armazenamento deve ser definido por meio de planos. Há um comentário relevante no arquivo *values.yaml* descrevendo melhor o armazenamento.

```
## storage:
##
## 1. If an engine component has the dataPlane property set,
##    then that plane will be the default data location for that component.
## 2. If there is a plane definition with a category of "data"
##    then the first matching plane will be the default data location
##
## If a data plane contains the storageClass property then an implicit pvc
##    will be created for that data plane.
##
## If plane.pvc is defined, a Persistent Volume Claim must exist with that name,
##    storageClass and storageSize are not used.
##
## If plane.storageClass is defined, storageClassName: <storageClass>
## If set to "-", storageClassName: "", which disables dynamic provisioning
## If set to "", choosing the default provisioner.
##    (gp2 on AWS, standard on GKE, AWS & OpenStack)
##
## plane.forcePermissions=true is required by some types of provisioned
## storage, where the mounted filing system has insufficient permissions to be
## read by the hpcc pods. Examples include using hostpath storage (e.g. on
## minikube and docker for desktop), or using NFS mounted storage.
```

Existem diferentes categorias de armazenamento, para uma implantação de HPCC Systems você deve ter no mínimo uma categoria dali, uma categoria dll e pelo menos 1 categoria de dados. Esses tipos geralmente são aplicáveis a todas as configurações, além de outras categorias opcionais de dados.

Todo o armazenamento deve estar em uma definição de plano de armazenamento. Isso é melhor descrito no comentário na definição de armazenamento no arquivo de valores.

```
planes:
#   name: <required>
#   prefix: <path>                                # Root directory for accessing the plane
#                                                    # (if pvc defined),
#                                                    # or url to access plane.
#   category: data|dali|lz|dll|spill|temp          # What category of data is stored on this plane?
#
# For dynamic pvc creation:
#   storageClass: ''
#   storageSize: 1Gi
#
# For persistent storage:
#   pvc: <name>                                    # The name of the persistent volume claim
#   forcePermissions: false
#   hosts: [ <host-list> ]                          # Inline list of hosts
#   hostGroup: <name>                               # Name of the host group for bare metal
#                                                    # must match the name of the storage plane..
#
# Other options:
#   subPath: <relative-path>                        # Optional sub directory within <prefix>
#                                                    # to use as the root directory
#   numDevices: 1                                   # number of devices that are part of the plane
#   secret: <secret-id>                             # what secret is required to access the files.
#                                                    # This could optionally become a list if required
#                                                    # (or add secrets:).
#
#   defaultSprayParts: 4                            # The number of partitions created when spraying
#                                                    # (default: 1)
#
#   cost:                                           # The storage cost
#   storageAtRest: 0.0135                          # Storage at rest cost: cost per GiB/month
```

Cada plano tem 3 campos obrigatórios: O nome, a categoria e o prefixo.

Quando o sistema estiver instalado, usando os valores fornecidos em estoque, ele criará um volume de armazenamento com capacidade de 1 GB através das seguintes propriedades.

Por exemplo:

```
- name: dali
  storageClass: ""
  storageSize: 1Gi
  prefix: "/var/lib/HPCCSystems/dalistorage"
  category: dali
```

Mais comumente o prefixo: define o caminho dentro do contêiner onde o armazenamento está montado. O prefixo pode ser uma URL para armazenamento de blobs. Todos os pods usarão o caminho (prefixo: ) para acessar o armazenamento.

Para o exemplo acima, quando você observar a lista de armazenamento, o *storageSize* criará um volume com 1 GB de capacidade. O prefixo será o caminho, a categoria é usada para limitar o acesso aos dados e minimizar o número de volumes acessíveis de cada componente.

As listas de armazenamento dinâmico no arquivo *values.yaml* são caracterizadas pelos valores *storageClass*: e *storageSize*:

**storageClass**: define qual storage deve ser usado para alocar o armazenamento. Uma classe de armazenamento em branco indica que deve usar a classe de armazenamento de provedores de nuvem padrão.

**storageSize**: Conforme indicado no exemplo, define a capacidade do volume.

## Categoria de Armazenamento

A categoria de armazenamento (Storage Category) é usada para indicar o tipo de dados que está sendo armazenado nesse local. Diferentes planos são usados para as diferentes categorias para isolar os diferentes tipos de dados uns dos outros, mas também porque eles geralmente exigem características de desempenho diferentes. Um plano nomeado pode armazenar apenas uma categoria de dados. As seções a seguir examinam as categorias de dados com suporte atualmente usadas em nossa implantação em contêiner.

```
category: data|dali|lz|dll|spill|temp # What category of data is stored on this plane?
```

O próprio sistema pode gravar em qualquer plano de dados. É assim que a categoria de dados pode ajudar a melhorar o desempenho. Por exemplo, se você tiver um índice, o Roxie desejará acesso rápido aos dados, em vez de outros componentes.

Alguns componentes podem usar apenas 1 categoria, alguns podem usar várias. O arquivo de valores pode conter mais de uma definição de plano de armazenamento para cada categoria. O primeiro plano de armazenamento na lista para cada categoria é usado como local padrão para armazenar essa categoria de dados. Essas categorias minimizam a exposição dos dados do avião a componentes que não precisam deles. Por exemplo, o componente ECLCC Server não precisa saber sobre as landing zones ou onde Dali armazena seus dados, portanto, ele monta apenas as categorias de avião necessárias.

## Armazenamento Temporário

O armazenamento temporário (Ephemeral storage) é alocado quando o cluster HPCC Systems é instalado e excluído quando o chart é desinstalado. Isso é útil para manter os custos de nuvem baixos, mas pode não ser apropriado para seus dados.

Em seu sistema, você deseja substituir o(s) valor(es) de estoque fornecido(s) pelo armazenamento apropriado para suas necessidades específicas. Os valores fornecidos criam volumes persistentes efêmeros ou temporários que são excluídos automaticamente quando o chart é desinstalado. Você provavelmente quer que o armazenamento seja persistente. Você deve personalizar o armazenamento para uma configuração mais adequada às suas necessidades.

## Armazenamento Persistente

O Kubernetes usa declarações de volume persistentes (pvcs) para fornecer acesso ao armazenamento de dados. O HPCC Systems oferece suporte ao armazenamento em nuvem por meio do provedor de nuvem que pode ser exposto por meio dessas declarações de volume persistentes.

As Declarações de Volume Persistentes podem ser criadas substituindo os valores de armazenamento no chart do Helm entregue. Os valores no arquivo `example/local/values-localfile.yaml` fornecidos substituem as entradas correspondentes no chart de comando original da pilha entregue do HPCC Systems. O chart `localfile` cria volumes de armazenamento persistentes. Você pode usar o `values-localfile.yaml` diretamente (como demonstrado em documentos/tutoriais separados) ou pode usá-lo como base para criar seu próprio chart de substituição.

Para definir um plano de armazenamento que utiliza um PVC, você deve decidir onde esses dados residirão. Você cria os diretórios de armazenamento, com os nomes apropriados e, em seguida, pode instalar o chart do Helm de arquivos locais para criar os volumes para usar a opção de armazenamento local, como no exemplo a seguir:

```
helm install mycluster hpcc/hpcc -f examples/local/values-localfile.yaml
```

**Nota:** As configurações para os PVCs devem ser `ReadWriteMany`, exceto para Dali que pode ser `ReadWriteOnce`.

Há vários recursos, blogs, tutoriais e até mesmo vídeos de desenvolvedores que fornecem detalhes passo a passo para a criação de volumes de armazenamento persistentes.

## Armazenamento Bare Metal

Há dois aspectos no uso do armazenamento bare metal no sistema Kubernetes. A primeira é a entrada *hostGroups* na seção de armazenamento que fornece listas nomeadas de hosts. As entradas *hostGroups* podem assumir uma das duas formas. Essa é a forma mais comum e associa diretamente uma lista de nomes de host a um nome:

```
storage:
  hostGroups:
    - name: <name> "The name of the host group"
      hosts: [ "a list of host names" ]
```

A segunda forma permite que um grupo de hosts seja derivado de outro:

```
storage:
  hostGroups:
    - name: "The name of the host group process"
      hostGroup: "Name of the hostgroup to create a subset of"
      count: <Number of hosts in the subset>
      offset: <the first host to include in the subset>
      delta: <Cycle offset to apply to the hosts>
```

Alguns exemplos típicos com clusters bare-metal são subconjuntos menores do host ou os mesmos hosts, mas armazenando partes diferentes em nós diferentes, por exemplo:

```
Group: groupCDE
  delta: 1
```

O segundo aspecto é adicionar uma propriedade à definição do plano de armazenamento para indicar quais hosts estão associados a ela. Existem duas opções:

- **hostGroup: <name>** O nome do grupo de hosts para bare metal. O nome do hostGroup deve corresponder ao nome do plano de armazenamento..
- **hosts: <list-of-namesname>** Uma lista embutida de hosts. Principalmente útil para landing zones.

Por Exemplo:

```
storage:
  planes:
    - name: demoOne
      category: data
      prefix: "/home/demo/temp"
      hostGroup: groupABCD # The name of the hostGroup
    - name: myDropZone
      category: lz
      prefix: "/home/demo/mydropzone"
      hosts: [ 'mylandingzone.com' ] # Inline reference to an external host.
```

## Itens de armazenamento para componentes de HPCC Systems

### Armazenamento geral de dados

Os arquivos de dados gerais gerados pelo HPCC são armazenados em dados. Para Thor, os custos de armazenamento de dados provavelmente podem ser significativos. A velocidade de acesso sequencial

é importante, mas o acesso aleatório é muito menos importante. Para ROXIE, a velocidade de acesso aleatório provavelmente será mais importante.

## LZ

LZ ou lz, utilizados para dados da landing zone. É aqui que colocamos os dados brutos que chegam ao sistema. Uma landing zone onde usuários externos podem ler e gravar arquivos. O HPCC Systems podem importar ou exportar arquivos para uma landing zone. Normalmente, o desempenho é um problema menor, pode ser armazenamento de bucket blob/s3, acessado diretamente ou por meio de uma montagem NFS.

## dali

A localização do repositório de metadados dali, que precisa dar suporte ao acesso aleatório rápido.

## dll

Onde as consultas ECL compiladas são armazenadas. O armazenamento precisa permitir que objetos compartilhados sejam carregados diretamente a partir dele de forma eficiente. Se você quiser dados Dali e dll no mesmo plano, é possível usar o mesmo prefixo para ambas as propriedades do subcaminho. Ambos usariam o mesmo prefixo, mas deveriam ter subcaminhos diferentes.

## sasha

Este é o local onde as workunitss são arquivadas, etc., são armazenadas e normalmente é menos crítico de velocidade, exigindo menores custos de armazenamento.

## spill

Uma categoria opcional na qual os arquivos spill são gravados. Os discos NVMe locais são potencialmente uma boa opção para isso.

## temp

Uma categoria opcional onde os arquivos temporários podem ser gravados.

## Valores de Segurança

Esta seção examinará as seções de *values.yaml* que tratam dos componentes de segurança do sistema.

## Certificados

A seção de certificados pode ser usada para permitir que o cert-manager gere certificados TLS para cada componente na implantação do HPCC Systems.

```
certificates:
  enabled: false
  issuers:
    local:
      name: hpcc-local-issuer
```

No arquivo yaml entregue, os certificados não estão habilitados, conforme ilustrado acima. Você deve primeiro instalar o cert-manager para usar esse recurso.

## Secrets

A seção Secrets contém um conjunto de categorias, cada uma contendo uma lista de secrets. A seção Secrets é onde obter informações no sistema se você não as quiser na fonte. Como código incorporado, você pode definir isso nas seções de sinal de código. Se você tiver informações que não deseja que sejam públicas, mas precisa executá-las, poderá usar segredos.

## Vaults

Vaults é outra maneira de fazer Secrets. A seção de vaults espelha a seção secrets, mas aproveita o HashiCorp Vault para o armazenamento de secrets. Há uma categoria adicional para vaults chamada "ecl-user". A intenção dos secrets do vaults do usuário ecl é ser legível diretamente do código ECL. Outras categorias vaults são lidas internamente pelos componentes do sistema e não expostas diretamente ao código ECL.

## Visibilidades

A seção de visibilidades pode ser usada para definir rótulos, anotações e tipos de serviço para qualquer serviço com a visibilidade especificada.

## Réplicas e Resources

Outros valores dignos de nota nos charts que têm relação com a instalação e configuração do HPCC Systems.

## Réplicas

replicas: define quantos nós de réplica surgem, quantos pods são executados para equilibrar uma carga. Para ilustrar, se você tiver um Roxie de 1 via e definir réplicas como 2, você terá 2 Roxies de 1 via.

## Recursos

A maioria dos componentes tem uma seção de recursos que define quantos recursos são atribuídos a esse componente. Nos arquivos de valores entregues em estoque, as seções recursos: existem apenas para fins ilustrativos e são comentadas. Qualquer implantação em nuvem que venha a desempenhar qualquer função não trivial, esses valores devem ser definidos adequadamente com recursos adequados para cada componente, da mesma forma que você alocaria recursos físicos adequados em um data center. Os recursos devem ser configurados de acordo com os requisitos específicos do sistema e o ambiente em que você os executaria. A definição inadequada de recursos pode resultar em falta de memória e/ou remoção do Kubernetes, pois o sistema pode usar quantidades ilimitadas de recursos, como memória e os nós ficarão sobrecarregados, momento em que o Kubernetes começará a despejar os pods. Portanto, se sua implantação estiver vendo despejos frequentes, convém ajustar sua alocação de recursos.

```
#resources:
#  cpu: "1"
#  memory: "4G"
```

Cada componente deve ter uma entrada de recursos, mas alguns componentes, como Thor, possuem vários recursos. Os componentes manager, worker e eclagent têm requisitos de recursos diferentes.

## Taints, Tolerations, e placements

Esta é uma consideração importante para sistemas em contêineres. Taints e Tolerations são tipos de restrições de nó do Kubernetes também referidas por **Node Affinity**. A afinidade do nó é uma maneira de



restringir os pods aos nós. Apenas uma "afinidade" pode ser aplicada a um pod. Se um pod corresponder a várias listas de "pods" de canais, somente a última definição de "afinidade" será aplicada.

Os taints e as tolerations trabalham juntos para garantir que os pods não sejam agendados em nós inadequados. As tolerâncias são aplicadas aos pods e permitem (mas não exigem) que os pods sejam agendados em nós com taints correspondentes. Taints são o oposto - elas permitem que um nó repele um conjunto de cápsulas.

Por exemplo, todos os workers Thor devem estar no tipo apropriado de VM. Se um grande job de Thor aparecer – então o nível de taints entra em jogo.

Para obter mais informações e exemplos de nossos Taints, Tolerations e Placements, consulte nossa documentação do desenvolvedor:

<https://github.com/hpcc-systems/HPCC-Platform/blob/master/helm/hpcc/docs/placements.md>

## Placement

O Placement é responsável por encontrar o melhor nó para um pod. Na maioria das vezes, o placement é tratado automaticamente pelo Kubernetes. Você pode restringir um pod para que ele possa ser executado apenas em um conjunto específico de nós. Usando canais, você pode configurar o agendador do Kubernetes para usar uma lista de "pods" para aplicar configurações aos pods. Por exemplo:

```
placements:
  - pods: [list]
    placement:
      <supported configurations>
```

Os pods: [list] podem conter uma variedade de itens.

1. Tipos de componentes do HPCC Systems, usando o *tipo* de prefixo: pode ser: dali, esp, eclagent, eclcc-server, roxie, thor. Por exemplo "tipo:esp"
2. Alvo; o nome de um item de array dos tipos acima usando o prefixo "target:" Por exemplo "target:roxie" ou "target:thor".
3. Pod, nome de metadados "Implantação" do nome do item de matriz de um tipo. Por exemplo, "eclwatch", "mydali", "thor-thoragent"
4. Expressão regular do nome do trabalho: Por exemplo, "compile-" ou "compile-". ou correspondência exata "^compile-.\$"
5. Todos: para solicitar todos os componentes do HPCC Systems. Os canais padrão para os pods que entregamos são [all]

**Placements** – no Kubernetes, o conceito de placement permite distribuir seus pods por tipos de nós com características particulares. Os placements seriam usados para garantir que os pods ou trabalhos que desejam nós com características específicas sejam colocados neles.

Por exemplo, um cluster Thor pode ser direcionado para machine learning usando nós com uma GPU. Outro trabalho pode querer nós com uma boa quantidade de memória ou outro para mais CPU. Você pode usar posicionamentos para garantir que os pods com requisitos específicos sejam colocados nos nós apropriados.

## Mais Helm e Yaml

Esta seção destina-se a fornecer algumas informações úteis para começar com uma implantação em contêiner. Existem vários recursos para usar arquivos Kubernetes, Helm e Yaml. Anteriormente, abordamos o

arquivo *values.yaml* e o arquivo *values-schema.json*. Esta seção expande alguns desses conceitos e como eles podem ser aplicados ao usar a versão em contêiner da plataforma HPCC Systems. Para obter mais informações sobre como usar arquivos Kubernetes, Helm ou YAML, ou para implantações de nuvem ou contêiner, consulte a respectiva documentação.

## Estrutura do arquivo *values.yaml*

O arquivo *values.yaml* é um arquivo yaml. Yaml é uma linguagem de serialização de dados frequentemente usada como formato para arquivos de configuração. A construção que compõe a maior parte de um arquivo yaml é o par chave-valor, às vezes chamado de hash ou dicionário. A construção do par chave-valor consiste em uma chave que aponta para algum(s) valor(es). Os valores podem ser strings, números, booleanos, inteiros, arrays ou dicionários e listas. Esses valores são definidos pelo esquema.

Em arquivos yaml, o recuo é usado para representar a estrutura e o aninhamento do documento. Espaços à esquerda são significativos e tabulações não são permitidas.

### Dicionário

Dicionários são coleções de mapeamentos de valores-chave. Todas as chaves diferenciam maiúsculas de minúsculas e, como mencionamos anteriormente, o recuo também é crucial. Essas chaves devem ser seguidas por dois pontos (:) e um espaço. Os dicionários também podem ser aninhados.

Dictionary is a key: value, followed by another key: value:, for example:

```
logging:
  detail: 80
```

Isso é uma exemplo de dicionário para registro.

Os dicionários nos arquivos de valores passados, como os do arquivo *myoverrides.yaml* no exemplo abaixo, serão mesclados nos dicionários correspondentes nos valores existentes, começando com os valores padrão do chart helm entregue.

```
helm install myhpcc hpcc/hpcc -f myoverrides.yaml
```

Observe que você pode passar quantos arquivos yaml desejar, eles serão mesclados na ordem em que aparecem na linha de comando.

Quaisquer valores pré-existent em um dicionário que não sejam substituídos continuarão presentes no resultado mesclado. No entanto, você pode excluir o conteúdo de um dicionário definindo-o como nulo.

### Listas

Listas são grupos de elementos começando no mesmo nível de recuo começando com um - (um traço e um espaço). Cada elemento da lista é recuado no mesmo nível e começa com um traço e um espaço. As listas também podem ser aninhadas e podem ser listas de dicionários, que por sua vez também podem ter propriedades de lista.

Um exemplo de uma lista de dicionários, com *placement.tolerations* como uma lista aninhada.:

```
placements:
- pods: ["all"]
  placement:
    tolerations:
    - key: "kubernetes.azure.com/scalesetpriority"
```

Uma chave é denotada usando um sinal de menos, que é um item de entrada na lista, que é um dicionário com atributos aninhados. Em seguida, o próximo sinal de menos (no mesmo nível de recuo) é a próxima entrada nessa lista.

## Global

A primeira seção do arquivo *values.yaml* descreve os valores globais. O *global.image.root* é uma string que indica qual versão extrair. Global se aplica geralmente a tudo.

```
# Default values for hpcc.

global:
  # Settings in the global section apply to all HPCC components in all subcharts

  image:
    ## It is recommended to name a specific version rather than latest, for any non-trivial
    ## For best results, the helm chart version and platform version should match - default if version
    ## not specified. Do not override without good reason as undefined behavior may result.
    ## version: x.y.z
    root: "hpccsystems"      # change this to pull from somewhere other than DockerHub hpccsystems
    pullPolicy: IfNotPresent

  # logging sets the default logging information for all components. Can be overridden locally
  logging:
    detail: 80
```

No trecho do arquivo *value.yaml* entregue do HPCC Systems (acima) *global:* é um dicionário de nível superior. Conforme observado nos comentários, as configurações na seção global se aplicam a todos os componentes do HPCC Systems. Observe a partir do recuo que os outros valores estão aninhados nesse dicionário global.

## Imagem

Em nosso arquivo *values.yaml* entregue, o valor imediatamente após *global:* é *image:* você deve usar uma versão nomeada específica em vez de usar o "latest", como também indicado nos comentários no arquivo de valores. A versão do chart do Helm e a versão da plataforma devem corresponder. Idealmente, você não deveria ter que definir o *image.version*. Por padrão, ele corresponderá à versão do chart do helm.

## O valor root

A entrada de nível de definição/dicionário global é *root*. Por exemplo

```
root: "hpccsystems" # change to pull your images somewhere other than DockerHub hpccsystems
```

No arquivo *values.yaml*, isso usa nosso repositório específico do HPCC Systems. É possível que você queira extrair de algum outro repositório, então é onde definir esse valor.

**root:** SomeValue

## Outros valores de chart

Itens definidos na seção global são compartilhados entre componentes.

Exemplos de valores global são seções de armazenamento e segurança.

```
storage:
  planes:
```

e também

```
security:
  eclSecurity:
    # Possible values:
    # allow - functionality is permitted
    # deny - functionality is not permitted
    # allowSigned - functionality permitted only if code signed
    embedded: "allow"
    pipe: "allow"
    extern: "allow"
    datafile: "allow"
```

Nos exemplos acima, storage: e security: são valores globais do chart.

## Usage

O arquivo *values.yaml* do HPCC Systems é usado pelo chart Helm para controlar como o HPCC Systems é implantado. O arquivo de valores contém dicionários e listas, e eles podem ser aninhados para criar estruturas mais complexas. O arquivo HPCC Systems *values.yaml* destina-se a ser um guia de instalação de demonstração de início rápido que não é apropriado para uso prático não trivial. Você deve personalizar sua implantação para uma que seja mais adequada às suas necessidades específicas. Para personalizar sua implantação, substitua os valores de estoque no arquivo *values.yaml*, como no exemplo a seguir:

```
helm install myhpcc hpcc/hpcc -f myoverrides.yaml
```

O exemplo acima usa o arquivo *myoverrides.yaml* por meio do parâmetro *-f*, que substitui quaisquer valores especificados no arquivo *values.yaml* do HPCC Systems. É importante observar que isso mescla as substituições de *myoverrides.yaml*. Qualquer coisa que esteja nos valores no próprio chart do helm que não seja substituído pelos valores passados permanecerá ativo. Quando houver 2 arquivos yaml como este exemplo (as pilhas *values.yaml* e *myoverrides.yaml*), se houver uma entrada correspondente (qualquer coisa que não seja um dicionário), o valor do segundo arquivo substituirá o primeiro. Os dicionários, no entanto, sempre serão mesclados.

Mais informações sobre implantações personalizadas são abordadas em outras seções, bem como na documentação do Kubernetes Helm. Consultar a documentação do Helm fornece detalhes completos para todos os aspectos do uso do gráfico do Helm, e não apenas para alguns casos selecionados descritos.

## Caso de Uso

Por exemplo, você deseja atualizar os detalhes do log. Você pode ter outro arquivo yaml para atualizar esse valor ou qualquer outro valor de lista usando um arquivo yaml de substituição.

Como veremos mais adiante, os componentes são definidos como listas, portanto, qualquer definição de um componente em um arquivo de valores do usuário substituirá todas as instâncias do componente no chart padrão. Você pode remover todos os componentes definidos em uma lista, substituindo a lista por uma lista nula, por exemplo,

```
thor: []
```

Isso removerá todos os componentes Thor.

Outras opções (por exemplo, configurar os custos para cpu ou acesso a arquivos) são implementadas como um dicionário, de modo que as opções podem ser definidas seletivamente em um arquivo de valores de usuários, e as outras opções serão mantidas.

## Mesclando e Sobrescrevendo

Tendo vários arquivos yaml, como um para registro, outro para armazenamento, outro para segredos e assim por diante, os arquivos podem estar no controle de versão. Eles podem ser versionados, verificados,

etc. e têm o benefício de apenas definir/alterar a área específica necessária, garantindo que todas as áreas não alteradas sejam deixadas intocadas. A regra aqui para manter em mente onde vários arquivos yaml são aplicados, os mais recentes sempre substituirão os valores dos anteriores. Eles são mesclados em sequência.

Outro ponto a considerar, onde existe um dicionário global como root: e seu valor é redefinido no 2º arquivo (como um dicionário) ele não seria sobrescrito. Você não pode simplesmente substituir um dicionário. Você pode redefinir um dicionário e defini-lo como nulo (como o exemplo Thor na seção anterior), o que efetivamente o eliminará.

**ATENÇÃO:** Se você tivesse uma definição global (como storage.planes) e a mesclasse onde ela fosse redefinida, eliminaria todas as definições da lista.

Outro meio de eliminar todos os valores em uma lista é passar um conjunto vazio denotado por um [ ], como este exemplo:

```
bundles: [ ]
```

Isso eliminaria quaisquer propriedades definidas para pacotes configuráveis.

## Geralmente aplicável

Esses itens são geralmente aplicáveis para nossos arquivos yaml do HPCC Systems Helm.

- Todos os nomes devem ser únicos.
- Todos os prefixos devem ser únicos.
- Os serviços devem ser únicos.
- yaml são mesclados em sequência.

Geralmente em relação aos componentes do HPCC Systems, os componentes são listas. Como dito anteriormente, se você tiver uma lista de valores vazia [ ], isso invalidaria essa lista em outro lugar.

## Uso adicional

Os componentes são adicionados ou modificados passando sobreposições. Os valores do chart são substituídos apenas, passando no arquivo de valores de substituição usando -f, (para arquivo de substituição) ou via --set, onde você pode substituir um único valor. Os valores passados são sempre mesclados na ordem em que são fornecidos na linha de comando do helm.

Por exemplo, você pode

```
helm install myhpcc hpcc/hpcc -f myoverrides.yaml
```

Para sobrepor quaisquer valores entregues no *values.yaml*. Ou você pode usar --set conforme o exemplo:

```
helm install myhpcc hpcc/hpcc --set storage.daliStorage.plane=dali-plane
```

Para substituir apenas o valor global.image.version. Novamente, a ordem em que os valores são mesclados é a mesma em que são emitidos na linha de comando. Agora considere:

```
helm install myhpcc hpcc/hpcc -f myoverrides.yaml --set storage.daliStorage.plane=dali-plane
```

No exemplo anterior, a flag --set no comando acima substitui o valor de storage.daliStorage.plane (if) definido em myoverrides.yaml, que substitui qualquer configuração de arquivo *values.yaml* e resulta em

defini-lo como dali-plane. Portanto, independentemente do valor no arquivo yaml para essa configuração específica, a ordem especificada na linha de comando o substitui na ordem fornecida na linha de comando.

## Opções de linha de comando

Se a flag `--set` for usada na instalação do helm ou na atualização do helm, esses valores serão simplesmente convertidos em YAML no lado do cliente.

Você pode especificar a flag `-f` várias vezes. A prioridade será dada ao último arquivo (mais à direita) especificado.

```
$ helm install myhpcc hpcc/hpcc -f myvalues.yaml -f override.yaml
```

Para o exemplo acima, se `myvalues.yaml` e `override.yaml` contivessem uma chave chamada 'Test', o valor definido em `override.yaml` teria precedência.

# Logging em contêiner

## Histórico de Logging

Os logs de componentes do Bare-metal HPCC Systems são gravados em arquivos persistentes no sistema de arquivos local. Em contraste, os logs HPCC em contêiner são efêmeros e sua localização nem sempre é bem definida. Os componentes do HPCC Systems fornecem logs informativos no nível do aplicativo para fins de depuração de problemas, ações de auditoria e monitoramento do progresso.

Seguindo as metodologias em contêiner mais amplamente aceitas, as informações de log de componentes do HPCC Systems são roteadas para os fluxos de saída padrão em vez de arquivos locais. Em implantações em contêiner, não há logs de componentes gravados em arquivos como nas edições anteriores.

Esses logs são gravados no fluxo de erro padrão (stderr). No nível do nó, o conteúdo do erro padrão e dos fluxos de saída são redirecionados para um local de destino por um mecanismo de contêiner. Em um ambiente Kubernetes, o mecanismo de contêiner do Docker redireciona os fluxos para um driver de log, que o Kubernetes configura para gravar em um arquivo no formato JSON. Os logs são expostos pelo Kubernetes por meio do comando "logs" apropriadamente chamado.

Por exemplo:

```
>kubectll logs myesp-6476c6659b-vqckq
>0000CF0F PRG INF 2020-05-12 17:10:34.910 1 10690 "HTTP First Line: GET / HTTP/1.1"
>0000CF10 PRG INF 2020-05-12 17:10:34.911 1 10690 "GET /, from 10.240.0.4"
>0000CF11 PRG INF 2020-05-12 17:10:34.911 1 10690 "TxSummary[activeReqs=22; rcv=5ms;total=6ms;]"
```

É importante entender que esses logs são de natureza efêmera e podem ser perdidos se o pod for despedido, o contêiner travar, o nó morrer etc. Além disso, devido à natureza das soluções em contêiner, os logs relacionados provavelmente se originam de vários locais e pode precisar ser coletado e processado. É altamente recomendável desenvolver uma estratégia de retenção e processamento com base em suas necessidades.

Muitas ferramentas estão disponíveis para ajudar a criar uma solução apropriada com base em uma abordagem do tipo "faça você mesmo" ou em recursos gerenciados disponíveis em provedores de nuvem.

Para os ambientes mais simples, pode ser aceitável confiar no processo padrão do Kubernetes que encaminha todo o conteúdo de stdout/stderr para o arquivo. No entanto, à medida que a complexidade do cluster aumenta ou a importância de reter o conteúdo dos logs aumenta, uma arquitetura de log em nível de cluster deve ser empregada.

O registro em nível de cluster para o cluster do HPCC Systems em contêiner pode ser realizado incluindo um agente de registro em cada nó. A tarefa de cada agente é expor os logs ou enviá-los por push para um back-end de processamento de log. Os agentes de registro geralmente não são fornecidos prontos para uso, mas há vários disponíveis, como o Elasticsearch e o Stackdriver Logging. Vários provedores de nuvem oferecem soluções integradas que coletam automaticamente todos os fluxos stdout/err e fornecem armazenamento dinâmico e ferramentas analíticas poderosas, além da capacidade de criar alertas personalizados com base em dados de log.

É sua responsabilidade determinar a solução apropriada para processar os dados de log de streaming.

## Soluções de Processamento de Log

Existem várias soluções de processamento de log disponíveis. Você pode optar por integrar os dados de registro do HPCC Systems com qualquer uma de suas soluções de registro existentes ou implementar outra especificamente para os dados do HPCC Systems. A partir do HPCC Systems versão 8.4, fornecemos uma solução de processamento de log leve e completa para sua conveniência. Como afirmado existem várias soluções possíveis, você deve escolher a opção que melhor atende às suas necessidades. As seções a seguir examinarão duas soluções possíveis.

### O chart Elastic4hpcclogs

O HPCC Systems fornece um chart Helm gerenciado, *elastic4hpcclogs*, que utiliza os charts Elastic Stack Helm para Elastic Search, Filebeats e Kibana. Este gráfico descreve uma instância local e mínima do Elastic Stack para processamento de log de componentes do HPCC Systems. Depois de implantados com êxito, os logs de componentes do HPCC produzidos no mesmo namespace devem ser indexados automaticamente no ponto de extremidade do Elastic Search. Os usuários podem consultar esses logs emitindo consultas de API RESTful do Elastic Search ou por meio da interface do usuário do Kibana (depois de criar um padrão de índice simples).

Pronto para uso, o Filebeat encaminha as entradas de log do componente HPCC para um índice com nome genérico: 'hpcc-logs' - <DATE\_STAMP> e grava os dados de log em campos prefixados 'hpcc.log.\*'. Ele também agrega k8s, Docker e metadados do sistema para ajudar o usuário a consultar as entradas de log de seu interesse.

Um padrão de índice do Kibana é criado automaticamente com base no layout de índice de batida de arquivo padrão.



# Instalando o chart elastic4hpcclogs

Instalar a solução simples fornecida é, como o nome indica, simples e uma maneira conveniente de coletar e filtrar dados de log. Ele é instalado por meio de nossos gráficos de leme do repositório HPCC Systems. No diretório HPCC-platform/helm, o gráfico elastic4hpcclogs é fornecido junto com os outros componentes da plataforma HPCC Systems. As próximas seções mostrarão como instalar e configurar a solução Elastic stack logging para HPCC Systems.

## Adicionar o Repositório HPCC Systems

O chart Elastic for HPCC Systems entregue pode ser encontrado no repositório HPCC Systems Helm. Para buscar e implantar os gráficos gerenciados do HPCC Systems, adicione o repositório do HPCC Systems Helm, caso ainda não tenha feito isso:

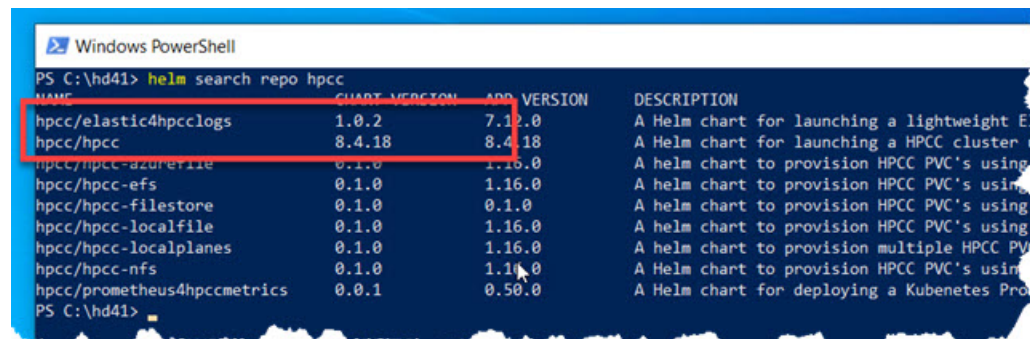
```
helm repo add hpcc https://hpcc-systems.github.io/helm-chart/
```

Depois que esse comando for concluído com êxito, o chart *elastic4hpcclogs* estará acessível.

Confirme se o chart apropriado foi puxado para baixo.

```
helm list
```

A emissão do comando `helm list` exibirá os gráficos e repositórios do HPCC Systems disponíveis. O gráfico *elastic4hpcclogs* está entre eles.



## Instalar o chart elastic4hpcc

Instalar o chart *elastic4hpcclogs* utilizando o seguinte comando:

```
helm install <Instance_Name> hpcc/elastic4hpcclogs
```

Forneça o nome que você deseja chamar sua instância do Elastic Search para o parâmetro `<Instance_Name>`. Por exemplo, você poderia chamar sua instância de "myelk" e, nesse caso, emitiria o comando de instalação da seguinte maneira:

```
helm install myelk hpcc/elastic4hpcclogs
```

Após a conclusão bem-sucedida, a seguinte mensagem é exibida:

```
Thank you for installing elastic4hpcclogs.
```

```
A lightweight Elastic Search instance for HPCC component log processing.
```

```
This deployment varies slightly from defaults set by Elastic, please review the effective values.
```

PLEASE NOTE: Elastic Search declares PVC(s) which might require explicit manual removal when no longer needed.



**IMPORTANTE:** O Elastic Search declara PVC(s) que podem exigir remoção manual explícita quando não forem mais necessários. Isso pode ser particularmente importante para alguns provedores de nuvem que podem acumular custos mesmo depois de não usar mais sua instância. Você deve garantir que nenhum componente (como PVCs) persista e continue acumulando custos.

**OBSERVAÇÃO:** dependendo da versão do Kubernetes, os usuários podem ser avisados sobre APIs obsoletas nos gráficos elásticos (ClusterRole e ClusterRoleBinding estão obsoletos na v1.17+). As implantações baseadas em Kubernetes < v1.22 não devem ser afetadas.

## Confirmar se seus Pods estão Prontos

Confirme se os pods estão prontos. Às vezes, após instalação, os pods podem levar alguns segundos para aparecerem. Confirme se os pods estão prontos antes de proceder. Para fazer isso, use o seguinte comando:

```
kubectl get pods
```

Este comando retorna a seguinte informação, exibindo o status dos pods.

|                               |     |         |   |
|-------------------------------|-----|---------|---|
| elasticsearch-master-0        | 1/1 | Running | 0 |
| myelk-filebeat-6wd2g          | 1/1 | Running | 0 |
| myelk-kibana-68688b4d4d-d489b | 1/1 | Running | 0 |

Quando todos os pods estiverem indicando um estado 'ready' e 'Running', incluindo os três componentes para filebeats, Elastic Search e Kibana (destacado acima), você poderá prosseguir.

## Confirmar os Serviços Elastic

Para garantir que os serviços Elastic estejam em execução, entre com o seguinte comando:

```
$ kubectl get svc
```

Isso exibe as seguintes informações de confirmação:

```
...
elasticsearch-master ClusterIP 10.109.50.54 <none> 9200/TCP,9300/TCP 68m
elasticsearch-master-headless ClusterIP None <none> 9200/TCP,9300/TCP 68m
myelk-kibana LoadBalancer 10.110.129.199 localhost 5601:31465/TCP 68m
...
```

Nota: O serviço myelk-kibana é declarado como LoadBalancer por conveniência.

## Configurando Componentes do Elastic Stack

Você pode precisar ou querer personalizar os componentes do Elastic Stack. Os valores dos charts do componentes Elastic podem ser substituídos como parte do comando de implantação do HPCC Systems.

Por exemplo:

```
helm install myelk hpcc/elastic4hpcclogs --set elasticsearch.replicas=2
```

Consulte o repositório GitHub do Elastic Stack para obter a lista completa de todas as opções do Filebeat, Elastic Search, LogStash e Kibana com descrições.

## Use of HPCC Systems Component Logs in Kibana

Uma vez ativado e em execução, você pode explorar e consultar os logs de componentes do HPCC Systems na interface do usuário do Kibana. O uso da interface do Kibana é bem suportado e documentado. Os padrões de índice do Kibana são necessários para explorar os dados do Elastic Search na interface do usuário do Kibana. A Elastic fornece explicações detalhadas das informações necessárias para entender e utilizar efetivamente a interface Elastic-Kibana. A documentação robusta do Kibana deve ser consultada para obter mais informações sobre como usar a interface do Kibana. Por favor, veja:

<https://www.elastic.co/>

e

<https://www.elastic.co/elastic-stack/>

Incluídos na documentação completa também estão vídeos de início rápido e outros recursos úteis.

# Azure AKS Insights

O Azure AKS Insights é um recurso opcional projetado para ajudar a monitorar o desempenho e a integridade de clusters baseados em Kubernetes. Uma vez habilitado e associado um determinado AKS a um cluster do HPCC Systems ativo, os logs do componente HPCC são capturados automaticamente pelo Insights. Todos os dados STDERR/STDOUT são capturados e disponibilizados para fins de monitoramento e/ou consulta. Como geralmente acontece com os recursos do provedor de nuvem, o custo é uma consideração importante e deve ser bem entendido antes da implementação. O conteúdo do log é gravado no armazenamento de logs associado ao seu espaço de trabalho do Log Analytics.

## Habilitar Azure Insights

A habilitação do Azure's Insights no cluster AKS de destino pode ser feita no portal do Azure ou via CLI. Para obter documentação detalhada do Azure: [Habilite insights de contêiner: Enabling Azure's Insights on the target AKS cluster can be done from the Azure portal or via CLI. For detailed Azure documentation: Enable Container insights:](https://docs.microsoft.com/en-us/azure/azure-monitor/containers/container-insights-onboard)

<https://docs.microsoft.com/en-us/azure/azure-monitor/containers/container-insights-onboard>

## Portal Azure

Para habilitar o insights do Azure no portal:

1. Selecione cluster AKS de Destino
2. Selecione Monitoring
3. Selecione Insights
4. Habilite - escolha ao workspace padrão

## Linha de Comando

Para habilitar os Azure insights na linha de comando:

Opcionalmente, crie o espaço de trabalho de análise de log [espaço de trabalho padrão, caso contrário]

Entre:

```
az monitor log-analytics workspace create -g myresourcegroup -n myworkspace --query-access Enabled
```

Habilitar no cluster AKS de destino (referência ao ID do recurso do workspace da etapa anterior)

```
az aks enable-addons -g myresourcegroup -n myaks -a monitoring --workspace-resource-id \
"/subscriptions/xyz/resourcegroups/myresourcegroup/providers/ \
microsoft.operationalinsights/workspaces/myworkspace"
```

A interface do AKS Insights no Azure fornece visualizações de métricas de integridade em nível de cluster/nó/contêiner centradas em Kubernetes e links diretos para logs de contêiner por meio de interfaces de "análise de log". Os logs podem ser consultados através da linguagem de consulta "Kusto" (KQL). Consulte a documentação do Azure para obter detalhes sobre como consultar os logs.

Consulte a documentação do Azure para obter detalhes sobre como consultar os logs.

Exemplo de consulta KQL para buscar entradas de registro "Transaction summary" de um contêiner ECLWatch:

```
let ContainerIdList = KubePodInventory
| where ContainerName =~ 'xyz/myesp'
| where ClusterId =~ '/subscriptions/xyz/resourceGroups/xyz/providers/Microsoft.
ContainerService/managedClusters/aks-clusterxyz'
| distinct ContainerID;
ContainerLog
| where LogEntry contains "TxSummary["
| where ContainerID in (ContainerIdList)
| project LogEntrySource, LogEntry, TimeGenerated, Computer, Image, Name, ContainerID
| order by TimeGenerated desc
| render table
```

#### Amostra de saída



```
11/5/2021, 9:02:00.000 PM    prometheus    esp_requests_active    0    {"app":"eclservices","namespace":"default","pod_name":"eclservices-778477d679-vgpj2"}
11/5/2021, 9:02:00.000 PM    prometheus    esp_requests_active    3    {"app":"eclservices","namespace":"default","pod_name":"eclservices-778477d679-vgpj2"}
```

Consultas mais complexas podem ser formuladas para buscar informações específicas fornecidas em qualquer uma das colunas de log, incluindo dados não formatados na mensagem de log. A interface do Insights facilita a criação de alertas com base nessas consultas, que podem ser usadas para acionar e-mails, SMS, execução de Logic App e muitas outras ações.

# Controlando a saída de registro do HPCC Systems

Os logs do HPCC Systems fornecem uma riqueza de informações que podem ser usadas para benchmarking, auditoria, depuração, monitoramento, etc. O tipo de informação fornecida nos logs e seu formato são controlados trivialmente através da configuração padrão do Helm. Tenha em mente que no modo de contêiner, cada linha de saída de log é passível de incorrer em um custo dependendo do provedor e do plano que você possui e a verbosidade deve ser cuidadosamente controlada usando as opções a seguir. Por padrão, os logs de componentes não são filtrados e contêm as seguintes colunas.

Por padrão, os logs de componentes não são filtrados e contêm as seguintes colunas:

```
MessageID TargetAudience LogEntryClass JobID DateStamp TimeStamp ProcessId ThreadID QuotedLogMessage
```

Os logs podem ser filtrados por TargetAudience, Category ou Detail Level. Além disso, as colunas de saída podem ser configuradas. As definições de configuração de registro podem ser aplicadas no nível global ou de componente.

## Target Audience Filtering

Os públicos-alvo disponíveis incluem operador (OPR), usuário (USR), programador (PRO), auditoria (ADT) ou todos. O filtro é controlado pelo valor <section>.logging.audiences. O valor da string é composto por códigos de 3 letras delimitados pelo operador de agregação (+) ou pelo operador de remoção (-).

Por exemplo, todas as saídas de log de componentes devem incluir apenas mensagens do programador e do usuário:

```
helm install myhpcc ./hpcc --set global.logging.audiences="PRO+USR"
```

## Filtragem de Categoria de Destino

As categorias de destino disponíveis incluem desastre (DIS), erro (ERR), informações (INF), aviso (WRN), progresso (PRO), métricas (MET). O filtro de categoria (ou classe) é controlado pelo valor <section>.logging.classes, composto por códigos de 3 letras delimitados pelo operador de agregação (+) ou pelo operador de remoção (-).

Por exemplo, a saída do log da instância mydali para incluir todas as classes, exceto o progresso:

```
helm install myhpcc ./hpcc --set dali[0].logging.classes="ALL-PRO" --set dali[0].name="mydali"
```

## Log Detail Level Configuration

A verbosidade da saída do log pode ser ajustada de "critical messages only" (1) até "report all messages" (100). O nível de log padrão é bastante alto (80) e deve ser ajustado de acordo.

Por exemplo, a verbosidade deve ser média para todos os componentes:

```
helm install myhpcc ./hpcc --set global.logging.detail="50"
```

## Configuração da Coluna de Dados de Registro

As colunas de dados de log disponíveis incluem messageid(MID), público(AUD), class(CLS), date(DAT), time(TIM), node(NOD), militime(MLT), microtime(MCT), nanotime(NNT), processid(PID), threadid(TID),

job(JOB), use(USE), session(SES), code(COD), component(COM), quotemessage(QUO), prefix(PFX), all(ALL) e padrão (STD). A configuração das colunas (ou campos) de dados de log é controlada pelo valor <section>.logging.fields, composto por códigos de 3 letras delimitados pelo operador de agregação (+) ou pelo operador de remoção (-). Por exemplo, todas as saídas de log de componentes devem incluir as colunas padrão, exceto a coluna de ID do job:

Por exemplo, todas as saídas de log de componentes devem incluir as colunas padrão, exceto a coluna de ID do job:

```
helm install myhpcc ./hpcc --set global.logging.fields="STD-JOB"
```

O ajuste de valores de registro por componente pode exigir a afirmação de vários valores específicos de componentes, o que pode ser inconveniente de fazer por meio do parâmetro de linha de comando --set. Nesses casos, um arquivo de valores personalizados pode ser usado para definir todos os campos obrigatórios.

Por exemplo, a instância do componente ESP 'eclwatch' deve gerar um log mínimo:

```
helm install myhpcc ./hpcc --set -f ./examples/logging/esp-eclwatch-low-logging-values.yaml
```