

# **Containerized HPCC Systems® Platform**

**Equipe de documentação de Boca Raton**



## Containerized HPCC Systems® Platform

Equipe de documentação de Boca Raton

Copyright © 2025 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.

2025 Version 9.12.16-1

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 Personalizadas .....	16
Técnicas de Customização .....	16
.....	21
Rastreamento de Custos de Contêineres .....	26
Protegendo Credenciais .....	31
Configuration Values .....	34
O Ambiente do Contêiner .....	34
Componentes HPCC Systems e o Arquivo <i>values.yaml</i> .....	35
O arquivo HPCC Systems <i>values.yaml</i> .....	41
Pods e Nós .....	52
Fundamentos de Helm e YAML .....	58
Registro em Contêiner .....	62
Contexto de Registro .....	62
Solução Gerenciada Elastic Stack .....	64
Solução de Análise de Logs do Azure .....	68
Controlling HPCC Systems Logging Output .....	72

# Visão geral do HPCC em contêineres

Desde a versão 8.0, a Plataforma HPCC Systems® começou a focar significativamente em implantações containerizadas. Isso é útil para implantações baseadas em nuvem (grandes ou pequenas) ou implantações de teste/desenvolvimento local.

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 nessas implantações containerizadas. A primeira seção é sobre o uso de contêineres Docker e gráficos Helm localmente. Docker e Helm fazem muito do trabalho por 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 utilizar qualquer tipo de serviços de Cloud, desde que suporte Docker, Kubernetes e Helm. Este livro, no entanto, vai focar nos Serviços de Nuvem da Microsoft Azure.

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.

O HPCC Systems segue 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.

# Bare-metal vs Containers

Se você está familiarizado com as implantações tradicionais da plataforma HPCC Systems em bare-metal, 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

Windows	Mac	Linux
<ul style="list-style-type: none"><li>• Docker Desktop &amp; WSL 2</li><li>• Helm</li></ul> OR <ul style="list-style-type: none"><li>• Docker Desktop &amp; Hyper-V</li><li>• Helm</li></ul> OR <ul style="list-style-type: none"><li>• Docker</li><li>• <u>Kubectl</u></li><li>• Helm</li><li>• <u>Minikube</u></li></ul>	<ul style="list-style-type: none"><li>• Docker Desktop</li><li>• Helm</li></ul>	<ul style="list-style-type: none"><li>• Docker</li><li>• Helm</li><li>• <u>Minikube</u></li></ul>

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:

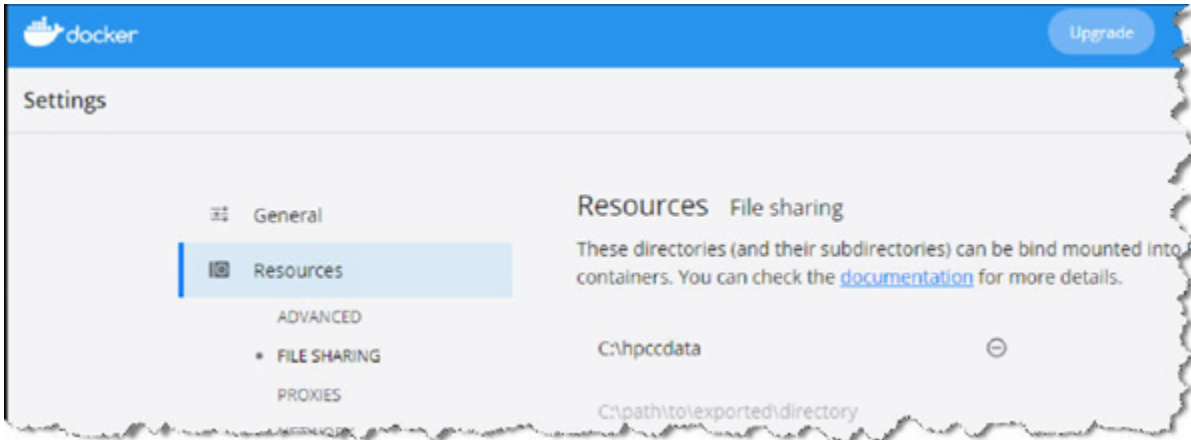
```
storage:
  planes:
    - name: dali
      pvc: dali-hpcc-localfile-pvc
      prefix: "/var/lib/HPCCSystems/dalistorage"
      category: dali
    - name: dll
      pvc: dll-hpcc-localfile-pvc
      prefix: "/var/lib/HPCCSystems/queries"
      category: dll
    - name: sasha
      pvc: sasha-hpcc-localfile-pvc
      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
```

- 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.



- 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.

- 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);
```

- Use o comando de desinstalação do helm para encerrar seu cluster e reinicie seu deploy.
- 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::hpcc-data::tutorial::originalperson'
```

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 Personalizadas

## Técnicas de Customização

Esta seção percorre a criação de um arquivo de configuração YAML personalizado e a implantação de uma plataforma HPCC Systems® usando a configuração padrão mais as personalizações. Uma vez que você entenda os conceitos neste capítulo, você pode se referir ao próximo capítulo para uma referência a todos os ajustes de valor de configuração.

Há várias maneiras de personalizar uma implantação da plataforma. Nós recomendamos o uso de métodos que permitem que você aproveite melhor a 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:

**Observação:** 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** e abra-o 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.

## Crie um Chart de Configuração Personalizado para AllowPipePrograms

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

Por exemplo:

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

Nesta seção, modificaremos o Thor para permitir alguns Programas PIPE. Na versão 9.2.0 e superior, os comandos usados no PIPE são restritos por padrão em implantações containerizadas a menos que sejam explicitamente permitidos no Helm chart.

1. Se você ainda não adicionou o repositório HPCC Systems a lista de repositórios Helm, faça agora.

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

Se você já adicionou, atualiza os últimos:

```
helm repo update
```

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

Você pode usar qualquer editor de texto.

3. Abra o arquivo padrão de valores que você salvou previamente (myvalues.yaml) em um editor de texto.
4. Copie a seção **thor**: toda e cole no novo arquivo thorWithPipe.yaml.
5. Adicione um bloco no fim:

```
allowedPipePrograms:
- sort
- grep
- echo
```

Este exemplo habilita três programas. Você pode um deles.

6. Salve o arquivo e feche o editor de texto.

O resultado deve do arquivo thorWithPipe.yaml deve ficar assim

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

```
thor:
- name: thor
  prefix: thor
  numWorkers: 2
  maxJobs: 4
  maxGraphs: 2
  allowedPipePrograms:
    - sort
    - grep
    - echo
```

### Faça o deploy utilizando o novo chart de configuração customizado.

1. Abra uma janela de terminal e navegue até o diretório onde o arquivo thorWithPipe.yaml foi salvo.
2. Faça o deploy da sua plataforma HPCC Systems, adicionando o novo arquivo de configuração por meio do comando:

```
# If you have previously stopped your cluster

helm install mycluster hpcc/hpcc -f thorWithPipe.yaml

# To upgrade without stopping

helm upgrade mycluster hpcc/hpcc -f thorWithPipe.yaml
```

3. Após confirmar o deploy em execução, execute um job que use uma ação PIPE e especifica um dos programas especificado.

**Nota:** Se o job for muito simples, será executado no hThor ao inves do Thor e este exemplo não habilita programas PIPE no hThor.

Você pode criar um novo arquivo yaml para permitir Programas PIPE no ECL Agent ou pode utilizar:

```
#OPTION('pickBestEngine',FALSE);
```

para forçar o job ser executado no Thor.

## Criar uma Configuração Personalizada do Chart para o Thor

Nesta seção, criaremos um arquivo YAML para especificar uma implementação de plataforma sem Thor.

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

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

Caso já tenha feito, atualize os últimos charts:

```
helm repo update
```

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

Você pode usar qualquer editor.

3. Edite, conforme o exemplo a seguir, o arquivo para desabilitar o Thor:

```
thor: []
```

4. Salve o arquivo e feche o editor de texto.

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

1. Abra o terminal e navegue até o diretório onde está salvo o arquivo nothor.yaml.
2. Faça o deploy do HPCC Systems, incluindo no seu comando a nova configuração:

```
# If you have previously stopped your cluster
helm install mycluster hpcc/hpcc -f nothor.yaml

# To upgrade without stopping
helm upgrade mycluster hpcc/hpcc -f nothor.yaml
```

3. Após confirmar que seu deploy está em execução, abra o ECL Watch.

Você não irá ver nenhum cluster Thor como destino.

## Crie um Chart de Configuração Personalizada para o Roxie

Nesta seção, criaremos um arquivo YAML para especificar uma implementação de plataforma sem Roxie. Embora o resultado seja semelhante ao que fizemos na seção anterior para sem Thor, a técnica é diferente.

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

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

Caso já tenha feito, atualize os últimos charts:

```
helm repo update
```

2. Crie um novo arquivo de texto e o nomeie para **noroxie.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 a seção **roxie**: e cole no novo arquivo noroxie.yaml.

6. Copie a seção **eclagent**: e cole no novo arquivo noroxie.yaml.

7. No bloco **roxie**, edite o valor para **disabled**: e mude para **true**.

Você pode remover todo o resto do bloco roxie: exceto o nome.

8. No bloco **eclagent**, exclua todo o bloco **name: roxie-workunit**.

Isso remove a instância de um Roxie atuando como um Agente ECL.

9. Salve e feche o editor de texto.

O arquivo noroxie.yaml deverá parecer como este:

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

```
roxie:
- name: roxie
  disabled: true

eclagent:
- name: hthor
  replicas: 1
  maxActive: 4
  prefix: hthor
  useChildProcesses: false
  type: hthor
```

**Faça o deploy usando o novo chart de configuração personalizada.**

1. Abra uma janela de terminal e navegue até o diretório onde está salvo o arquivo noroxie.yaml.
2. Faça o deploy do HPCC Systems, incluindo no seu comando a nova configuração:

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

3. Após confirmar que seu deploy está em execução, abra o ECL Watch.

Você não irá mais ver nenhum cluster Roxie disponível como destino.

## Crie um Chart de Configuração Personalizado para Múltiplos Thors Ouvindo uma Fila Comum.

Nesta seção, criaremos três Thors que escutam uma fila comum (além de sua própria fila). Isso fornece a capacidade de definir configurações distintas de cluster Thor, mas permite que eles formem um único alvo atrás de uma única fila. Esses clusters podem ser vinculados a determinados conjuntos de nós em diferentes zonas de disponibilidade, se desejado. Você pode usar este exemplo como um ponto de partida e ajustar o número de clusters Thor que deseja.

Isso é alcançado definindo filas de destino auxiliares adicionais para cada definição de Thor e usando um nome comum como uma fila auxiliar.

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

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

Se já fez isso, atualize os últimos charts:

```
helm repo update
```

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

Você pode usar qualquer editor de texto.

3. Abra o arquivo padrão de valores que você salvou previamente (myvalues.yaml) em um editor de texto.

4. Copie **thor:** e cole no novo arquivo threethorsonequeue.yaml file.

5. Copie todo o conteúdo do novo arquivo yaml file, exceto a primeira linha (thor:), e cole no final do arquivo **twice**.

Isso cria três seções - name:.

6. Edite o arquivo da seguinte maneira:

a. De um único valor de **name** para cada Thor:.

Neste exemplo, utilizaremos **thor1**, **thor2**, e **thor3**.

b. Adicione a entrada **auxQueues:** para cada block Thor block utilizando um nome comum

Neste exemplo estamos utilizando:

**auxQueues: [ thorQ ]**

c. Certifique-se que o **prefix:** é o mesmo que cada bloco do Thor.

7. Salve e feche o editor de texto.

O arquivo `threethorsonequeue.yaml` deverá ficar assim:

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

```
thor:
- name: thor1
  auxQueues: [ thorQ ]
  maxGraphs: 2
  maxJobs: 2
  numWorkers: 4
  numWorkersPerPod: 2
  prefix: thor
- name: thor2
  maxGraphs: 2
  maxJobs: 2
  numWorkers: 4
  numWorkersPerPod: 2
  prefix: thor
  auxQueues: [ thorQ ]
- name: thor3
  maxGraphs: 2
  maxJobs: 2
  numWorkers: 4
  numWorkersPerPod: 2
  prefix: thor
  auxQueues: [ thorQ ]
```

**Faça o deploy usando o novo chart de configuração personalizada.**

1. Abra uma janela de terminal e navegue até o diretório onde está salvo o arquivo `threethorsonequeue.yaml`.
2. Faça o deploy do HPCC Systems, incluindo no seu comando a nova configuração:

```
# If you have previously stopped your cluster

helm install mycluster hpcc/hpcc -f threethorsonequeue.yaml

# To upgrade without stopping

helm upgrade mycluster hpcc/hpcc -f threethorsonequeue.yaml
```

3. Após confirmar que seu deploy está em execução, abra o ECL Watch.

Você deve ver quatro clusters Thor disponíveis como Alvos - `thor1`, `thor2`, `thor3` e uma quarta fila que todos os três Thors ouvem - `thorQ`.

## Crie um Chart de Configuração Personalizado somente para Landing Zone

Nessa seção, nós iremos criar uma configuração personalizada que implementa uma "plataforma" contendo apenas uma Landing Zone. Isso pode ser útil se tudo o que você precisa é um servidor de landing zone com o `dafilesrv` rodando.

**Nota:** Isso só pode ser implementado em um namespace diferente de qualquer outra instância de plataforma.

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



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

Se já fez isso, atualize os últimos charts:

```
helm repo update
```

2. Criar um novo arquivo de texto e o nomeie para **lz.yaml** e abra-o em um editor de texto.

Você pode usar qualquer editor de texto.

3. Copie e cole este código no arquivo:

```
dafilesrv:
- name: direct-access
  application: directio
  service:
    servicePort: 7100
    visibility: local
    tls: false
  resources:
    cpu: "2"
    memory: "8G"
dali: []
dfuserver: []
eclagent: []
eclccserver: []
eclscheduler: []
esp: []
roxie: []
sasha: null
thor: []
```

4. Salve e feche o editor de texto.

**Faça o deploy usando o novo chart de configuração personalizada.**

1. Abra uma janela de terminal e navegue até o diretório onde está salvo o arquivo lz.yaml file.
2. Implante esta "plataforma" LZ somente com a nova configuração adicionada ao seu comando.

```
helm install mylz hpcc/hpcc -f lz.yaml
```

3. Confirme que está instalado usando este comando:

```
helm list
```

# Rastreamento de Custos de Contêineres

Com o advento da plataforma de sistemas HPCC em contêineres, introduzimos informações de rastreamento de custos. Isso é particularmente útil ao usar instâncias de plataforma HPCC Systems nativas da nuvem em uma configuração de nuvem em que algum planejamento e configuração podem ajudar a reduzir as despesas.

Novas colunas foram adicionadas às páginas de workunits e arquivos lógicos no ECL Watch. Essas colunas podem ser ordenadas por custo, assim como as outras colunas no ECL Watch, clicando no topo da de cada uma. Além disso, os custos das operações de arquivos executadas por workunits são fornecidos nas métricas da workunit.

À medida que o rastreamento de custos amadurece, os cálculos de custos também melhoram significativamente, proporcionando um rastreamento de custos mais preciso. Por exemplo, dados que foram acessados a partir do cache de página não incorrem em nenhum custo de acesso a arquivos. O rastreamento de custos do HPCC Systems agora detecta dados que foram retornados do cache de página e ajusta os cálculos de custos de forma apropriada, resultando em cálculos de custos mais precisos.

## Tipos de custos

Existem uma série de tipos de custos que são rastreados.

- Custos de Execução
- Custos de Compilação
- Custo de Acesso a Arquivo
- Custo do Arquivo em Repouso

**OBSERVAÇÃO:** Todos os valores de custo calculados e exibidos são aproximados. Existem muitas variáveis que podem resultar em imprecisões. Esses valores de custo devem ser usados apenas como um guia.

## Custos de Execução

Custo de execução é o valor referente a custo de execução da workunit, do graph e subgraphs no cluster Thor. Inclui o custo de todos os nós diretamente necessários para executar o trabalho e inclui o custo de:

- Nós do executor
- Nós do Compilador
- Nós do agente e do gerenciador

O valor do custo de execução de uma workunit é exibido no ECL Watch em sua página de resumo e é organizado em graph, subgraph e nível de atividade. Os valores de custo do graph e do subgraph estão disponíveis no visualizador de métricas e graph.

**Observação:** O custo de execução das workunits ROXIE não está implementado atualmente.

## Job Guilhotina

O risco de custos descontrolados é uma preocupação para cobrança baseada em uso potencialmente ilimitada. Assim, o recurso job guilhotina é fornecido para gerenciar esse cenário, limitando os custos por meio dos valores limite e do limite rígido. Quando o custo de um job atinge um valor definido, ele pode ser encerrado, controlando os custos que podem incorrer.

**Observação:** Esta funcionalidade atualmente é suportada somente para os jobs do Thor.

Refresh Copy WUID Save Delete Restore Reschedule Deschedule Set To Fail

**W20220218-154420**

WUID: W20220218-154420

Action: compile

State: failed

Owner:

Job Name: writesuper

Description:

☒ 2 Error(s) ☒ 0 Warning(s) ☒ 0 Info(s) ☒ 0 Other(s)

Severity	Source	Code	Message
Error	eclagent	10142	System error: 10142: Job cost exceeds limit

System error: 10142: Job cost exceeds limit

## Custos de Compilação

Os custos de compilação são os custos de compilar o código ECL. O custo de compilar o código ECL está incluído como uma coluna na lista de workunit. Na página de resumo da workunit, há um campo de custo de compilação.

## Custos de armazenamento

Este é o custo de hospedar os dados no plano de armazenamento ou o valor do Custo do Arquivo em Repouso e o custo de leitura/escrita no armazenamento é o valor do *Custo de Acesso ao Arquivo*.

**Observação:** Os custos não são registrados para arquivos temporários ou de derramamento, porque o armazenamento local está incluído no preço da VM usada para calcular os custos de execução.

Para arquivos lógicos, os custos calculados estão baseados em dois tipos de acesso ao arquivo.

- Custos de Acesso ao arquivo
- Custos de Arquivo em Repouso

## Custo de acesso aos arquivos

Os custos de leitura e gravação em arquivos são referidos como custos de acesso a arquivos. Vários planos de armazenamento cobram por operações de dados separadamente. O valor do custo de acesso

ao arquivo incluirá o custo de leitura e gravação. Neste momento, quaisquer outros custos relacionados com ações de arquivo (como excluir ou copiar) não serão registrados ou incluídos como parte dos custos.

Os custos incorridos por uma workunit para acessar arquivos lógicos também são registrados nas estatísticas e atributos da workunit. O custo de leitura/escrita é registrado no registro de atividade e acumulado no nível do grafo, do subgrafo e do escopo do fluxo de trabalho. O Rastreamento de Custos detecta dados que foram retornados do cache de página e ajusta os cálculos de custos de forma apropriada para produzir cálculos de custos mais precisos. Esses custos de acesso a arquivos para uma unidade de trabalho são registrados com a workunit e exibidos na página de resumo e nas métricas da workunit.

### Custo do Arquivo em Repouso

O campo Custo do Arquivo em Repouso é mostrado na página de resumo do Arquivo Lógico. É o custo de armazenar o arquivo sem acessar os dados. Apenas os custos de armazenamento associados à hospedagem do arquivo na nuvem. Este valor foi adicionado para diferenciar melhor entre os custos de armazenamento de arquivos.

## Custo de Configuração

Esta seção detalha a configuração dos parâmetros de configuração dos jobst. A configuração dos custos do jobs trabalho em uma instância HPCC Systems sistemas HPCC nativos da nuvem é feita usando o chart helm. Por padrão, o arquivo *values.yaml* fornecido contém uma seção para configurar custos.

Por exemplo:

1. Crie um novo arquivo de texto e o nomeie **mycosts.yaml** e abra em um editor de texto.

Você pode usar um editor de texto.

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

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

3. Abra o arquivo salvo (myvalues.yaml) em um editor de texto.
4. Copie a sessão **cost:** e cole em outro novo arquivo mycosts.yaml.
5. Altere quaisquer valores relacionados a custos desejados, conforme apropriado.
6. Salve o arquivo e feche o editor de texto.
7. Implante sua plataforma HPCC Systems, adicionando a nova configuração ao seu comando:

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

Os valores de configuração fornecem as informações de preços e informações de formatação de moeda. Os seguintes parâmetros de configuração de custo são suportados: The configuration values provide the pricing information and currency formatting information. The following cost configuration parameters are supported:

<i>currencyCode</i>	Usado para formatação de moeda de valores de custo.
<i>perCpu</i>	Custo por hora de uma única CPU.
<i>storageAtRest</i>	Custo de armazenamento por gigabyte por mês.
<i>storageReads</i>	Custo por 10.000 operações de leitura.
<i>storageWrites</i>	Custo por 10.000 operações de gravação.

## Configurando custos da Nuvem

O arquivo de configuração *values.yaml* padrão é configurado com os seguintes parâmetros de custo na seção global/cost:

```
cost:
  currencyCode: USD
  perCpu: 0.126
  storageAtRest: 0.0135
  storageReads: 0.0485
  storageWrites: 0.0038
```

O atributo **currencyCode** deve ser configurado com o código de país ISO 4217. (O padrão da plataforma HPCC Systems é USD se o código da moeda estiver faltando).

O **perCpu** da seção global/cost se aplica a todos os componentes que não foram configurados com seu próprio valor perCpu.

Um valor perCpu específico para um componente pode ser definido adicionando um atributo cost/perCPU na seção desse componente.

Para componentes Dali:

```
dali:
- name: mydali
  cost:
    perCpu: 0.24
```

## Configuração dos Custos do Thor

Os componentes Thor suportam parâmetros de custo adicionais que são usados para o recurso de "guilhotina" de trabalho:

<i>limit</i>	Define o limite de custo "flexível" que uma unidade de trabalho pode incorrer. O limite é "suave" no sentido de que pode ser substituído pela opção do ECL <b>maxCost</b> . Um nó será encerrado se exceder seu <b>maxCost</b> (se definido) ou o valor do atributo limite (se o <b>maxCost</b> não for definido).
<i>hardlimit</i>	Define o limite de custo máximo absoluto, um limite que não pode ser substituído pela configuração da opção ECL. O valor <b>maxCost</b> que exceder o hardlimit será ignorado.

O exemplo a seguir define os limites de custo dos trabalhos, adicionando os atributos à seção Thor do yaml de configuração.

```
thor:
- name: thor
  prefix: thor
  numWorkers: 2
  maxJobs: 4
  maxGraphs: 2
  cost:
    limit: 10.00 # maximum cost is $10, overridable with maxCost option
    hardlimit: 20.00 # maximum cost is $20, cannot be overridden
```

## Parâmetros dos Custos de Armazenamentos

Os parâmetros de custos de armazenamentos (**storageAtRest**, **storageReads** e **storageWrites**) podem ser adicionados na seção de custo do plano de armazenamento para definir parâmetros de custo específicos para o plano de armazenamento.

Por exemplo:

```
storage:
  planes:
    - name: dali
      storageClass: ""
      storageSize: 1Gi
      prefix: "/var/lib/HPCCSystems/dalistorage"
      pvc: mycluster-hpcc-dalistorage-pvc
      category: dali
      cost:
        storageAtRest: 0.01
        storageReads: 0.001
        storageWrites: 0.04
```

Os parâmetros de custo de armazenamento na seção global são usados apenas se nenhum parâmetro de custo for especificado no plano de armazenamento.

# Protegendo Credenciais

Utilizar o HPCC Systems em um ambiente containerizado tem algumas preocupações de segurança únicas, ao externalizar componentes normalmente internalizados, como as credenciais dos administradores LDAP.

A proteção das credenciais dos administradores LDAP é realizada usando os segredos do Kubernetes ou do Hashicorp Vault. Como pré-requisito, você deve estar familiarizado com a configuração de segredos do Kubernetes e/ou do Hashicorp Vault.

A conta dos administradores LDAP deve ter direitos de administrador para todos os Base DN's usados pela plataforma HPCC Systems. Em uma implantação na nuvem, essas credenciais podem ser expostas. Portanto, uma boa prática para essas credenciais de administrador é serem protegidas usando segredos do Kubernetes ou o HashiCorp Vault.

## Protegendo as credenciais no Kubernetes

Para criar um secret no Kubernetes para armazenar as credenciais da conta do usuário do administrador do LDAP, use uma interface de linha de comando para o Kubernetes e execute um comando semelhante ao seguinte exemplo:

```
kubectl create secret generic admincredssecretname --from-literal=username=hpcc_admin \
--from-literal=password=t0pS3cr3tP@ssw0rd
```

No exemplo acima, o nome do secret do Kubernetes é "admincredssecretname", e ele contém as chaves/valores "username" e "password" da conta do administrador do LDAP. Isso armazena o nome de usuário e a senha do administrador do LDAP como um segredo do Kubernetes. Quaisquer propriedades adicionais são ignoradas.

Você pode verificar o secret que acabou de criar executando o seguinte comando no Kubernetes.

```
kubectl get secret admincredssecretname
```

Para obter mais informações sobre o Kubernetes, consulte a documentação específica para a sua implementação..

## Utilizando secrets do Kubernetes

Para implementar os secrets do Kubernetes, substitua a seção "secrets:" em HPCC-Platform/helm/hpcc/values.yaml, ou implemente com o seu próprio gráfico personalizado. Para mais informações sobre como personalizar sua implementação containerizada de Sistemas HPCC, veja as seções acima sobre técnicas de personalização.

No seu chat, crie um nome de chave único usado para referenciar o segredo, e defina-o como o nome do secret que você criou na etapa anterior. No exemplo acima, era "admincredssecretname".

Você pode, opcionalmente, definir segredos adicionais conforme necessário pela configuração de segurança da sua plataforma. Cada um desses segredos seria criado conforme descrito acima e receberia nomes únicos. O exemplo abaixo indica como você pode adicionar quaisquer credenciais ou secrets adicionais aos seus chart Helm, se necessário.

A chave/valor "admincredsmountname" já existe por padrão no arquivo values.yaml fornecido pelo HPCC Systems. A chave é referenciada no arquivo ldap.yaml do componente. Você pode substituir estes e adicionar chaves/valores adicionais conforme necessário. O seguinte exemplo ilustra a adição de "additionalsecretname" e esse nome deve corresponder ao nome do segredo adicional criado usando as etapas acima.

```
secrets:
  authn:
    admincredsmountname: "admincredssecretname"    #externalize HPCC Admin creds
    additionalmountname: "additionalsecretname"    #alternate HPCC Admin creds
```

## Habilitar autenticação LDAP

No arquivo `ldap.yaml` fornecido em `HPCC-Platform/esp/applications/common/ldap/`, a `"ldapAdminSecretKey"` já está configurada para o nome da chave de montagem ilustrado no exemplo acima. Para habilitar a autenticação LDAP e modificar este valor, você ou seu administrador de sistemas podem substituir o componente Helm ESP/ECLWatch localizado no `chart values.yaml`, conforme ilustrado no exemplo seguinte:

```
esp:
- name: eclwatch
  application: eclwatch
  auth: ldap
  ldap:
    ldapAddress: "myldapserver"
    ldapAdminSecretKey: "additionaltmountname" # use alternate secrets creds
```

## Protegendo credenciais no HashiCorp Vault

Para criar e armazenar secrets no HashiCorp Vault, a partir da linha de comando, execute os seguintes comandos Vault. O nome secreto usado no exemplo abaixo é `"myvaultadmincreds"` e deve ser prefixado com `"secret/authn/"` conforme ilustrado. As chaves/valores `"username"` e `"password"` do administrador LDAP são necessários. Propriedades adicionais são ignoradas.

```
vault kv put secret/authn/myvaultadmincreds username=hpcc_admin password=t0pS3cr3tP@ssw0rd
```

Onde `"secret/authn/myvaultadmincreds"` é o nome do secret que contém o nome de usuário e a senha do administrador LDAP.

Para verificar e confirmar os valores do secret, execute o seguinte comando:

```
vault kv get secret/authn/myvaultadmincreds
```

Para obter mais informações sobre como criar segredos para o HashiCorp Vault, consulte a documentação apropriada da HashiCorp para sua implementação.

## Deploy do HashiCorp Vault

Deploy dos secrets do HashiCorp Vault quando você sobrescrever a seção `"secrets:"` em `HPCC-Platform/helm/hpcc/values.yaml`, ou em seu chart de configuração personalizado. Para mais informações sobre como personalizar sua implantação containerizada do HPCC Systems, veja as seções acima sobre técnicas de personalização.

O valor do nome do Vault é definido para este exemplo no chart gráfico de configuração `values-secrets.yaml`. Você pode encontrar um exemplo deste gráfico no repositório HPCC-Platform em `/helm/examples/secrets/values-secrets.yaml`.

```
vaults:
  authn:
    - name: my-authn-vault
      #The data node in the URL is there for use by the REST API
      #The path inside the vault starts after /data
      url: http://${env.VAULT_SERVICE_HOST}:${env.VAULT_SERVICE_PORT}/v1/secret/data/authn/${secret}
      kind: kv-v2
```



Você pode inserir isso em seu próprio gráfico de personalização onde você fornece à sua implantação o nome do cofre que contém as credenciais.

## Referenciando Vault Stored Authentication

Os nomes de chave "ldapAdminSecretKey" e "ldapAdminVaultId" são usados pelo gerente de segurança do HPCC Systems para resolver os segredos, e devem corresponder exatamente quando se utiliza o nome do Vault configurado nas etapas anteriores.

```
esp:
- name: eclwatch
  application: eclwatch
  auth: ldap
  ldap:
    ldapAddress: "myldapserver"
    ldapAdminSecretKey: "myvaultadmincreds"
    ldapAdminVaultId: "my-authn-vault"
```

# Configuration Values

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 contêiner é gerida por um único arquivo, um arquivo *values.yaml* e associado ao schema (*values-schema.json*) file.

## O *values.yaml* e como é utilizado

O arquivo de estoque *values.yaml*, fornecido no repositório HPCC Systems, são os valores de configuração fornecidos para o Helm chart "hpcc". O arquivo *values.yaml* é usado pelo Helm chart para controlar como HPCC systems é implantado na nuvem. Este arquivo *values.yaml* é um único arquivo 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 implantação em contêiner. Você deve usar o arquivo de valores fornecido como base para personalizações da modelagem do seus requisitos para sua implantação específica.

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 e o Arquivo values.yaml

Os gráficos de 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"
```

The stock values file will work and allow you to stand up a functional system, however you should define the component resources in a manner that corresponds best to your operational strategy.

## 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 arquivo YAML, assim como o ECL Agent.

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

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

If you were to add a mydfuserver as follows

```
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.

**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.

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

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.

## Containerized HPCC Systems® Platform Configuration Values

```
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:

```
storage:
  hostGroups:
    - name: groupABCDE # Explicit list of hosts
      hosts: [A, B, C, D, E]
    - name: groupCDE # Subset of the group last 3 hosts
      hostGroup: groupABCDE
      count: 3
      offset: 2
    - name: groupDEC # Same set of hosts, but different part->host mapping
      hostGroup: 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.
```

## Armazenamento Remoto

Você pode configurar sua implantação em nuvem do HPCC Systems para acessar arquivos lógicos de outros ambientes remotos. Você configura esse armazenamento remoto adicionando uma seção "remota" ao seu gráfico de helm.

A seção *storage.remote* é uma lista de ambientes remotos nomeados que definem a URL do serviço remoto e uma seção que mapeia os nomes dos planos remotos para os nomes dos planos locais. Os planos locais referenciados são planos especiais com a categoria 'remoto'. Eles são somente leitura e só estão disponíveis para os motores que podem lê-los.

Por exemplo:

```
storage:
  planes:
  ...
  - name: hpcc2-stddata
    pvc: hpcc2-stddata-pv
    prefix: "/var/lib/HPCCSystems/hpcc2/hpcc-stddata"
    category: remote
  - name: hpcc2-fastdata
    pvc: hpcc2-fastdata-pv
    prefix: "/var/lib/HPCCSystems/hpcc2/hpcc-fastdata"
    category: remote
  remote:
  - name: hpcc2
    service: http://20.102.22.31:8010
    planes:
    - remote: data
      local: hpcc2-stddata
    - remote: fastdata
      local: hpcc2-fastdata
```

Este exemplo define um alvo remoto chamado "hpcc2" cuja URL do serviço DFS é http://20.102.22.31:8010 e cujo plano local é "hpcc2data". O plano local deve ser definido de modo que compartilhe o mesmo armazenamento que o ambiente remoto. Espera-se que isso seja feito através de um PVC que foi pré-configurado para usar o mesmo armazenamento.

Para acessar o arquivo lógico em ECL use o seguinte formato:

```
ds := DATASET('~remote::hpcc2::somescope1::somefn1', rec);
```

## Armazenamento Preferencial

A opção *preferredReadPlanes* está disponível para cada tipo de cluster - hThor, Thor e Roxie.

Esta opção é significativa apenas para arquivos lógicos que residem em múltiplos planos de armazenamento. Quando especificado, a plataforma HPCC Systems buscará ler arquivos do(s) plano(s) preferido(s) se um arquivo residir neles. Estes planos preferenciais devem existir e ser definidos em *storage.planes*.

O próximo trecho é um exemplo de um cluster Thor configurado com a opção *preferredDataReadPlanes*.

```
thor:
- name: thor
  prefix: thor
  numWorkers: 2
  maxJobs: 4
  maxGraphs: 3
  preferredDataReadPlanes:
  - data-copy
```

- indexdata-copy

No exemplo acima, ao executar uma consulta que lê um arquivo que reside tanto em "data" quanto em "data-copy" (nessa ordem), normalmente leria a primeira cópia em "data". Com *preferredDataReadPlanes* especificado, se esse arquivo também residir em "data-copy", Thor lerá essa cópia.

Isso pode ser útil quando existem múltiplas cópias de arquivos em diferentes planos com características distintas que podem impactar o desempenho.

## 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.

## Egresso

A fim de permitir que os clusters sejam trancados de forma segura, mas ainda permitir o acesso aos serviços de que precisam, existe um mecanismo de saída. O egresso fornece um mecanismo similar ao ingresso, sendo capaz de definir quais endpoints e portas os componentes têm permissão para se conectar.

A maioria dos componentes do HPCC Systems possui suas próprias políticas de rede geradas automaticamente. As políticas de rede geradas tipicamente trabalham para limitar o ingresso e egresso à comunicação inter-componente, ou apenas ao ingresso de serviço externo esperado.

Por exemplo, em um sistema implantado por padrão com políticas de rede em vigor, uma consulta em execução (em hThor, Thor ou Roxie) não será capaz de se conectar com um serviço de terceiros, como um serviço LDAP ou pilha de log.

Na configuração padrão, qualquer pod com acesso à API do Kube também terá acesso a qualquer endpoint externo. Isso ocorre porque uma *NetworkPolicy* é gerada para acesso de egresso para componentes que precisam de acesso à API do Kube. No entanto, *global.egress.kubeApiCidr* e *global.egress.kubeApiPort* no *values.yaml* devem ser configurados em um sistema seguro para restringir esse acesso de egresso, de modo que ele apenas exponha acesso de egresso para o endpoint da API do Kube.

Adicionamos um mecanismo semelhante às definições de visibilidade, que permite seções de saída nomeadas, que então podem ser referenciadas por componente.

Por exemplo:

```
global:
  egress:
    engineEgress:
      - to:
        - ipBlock:
            cidr: 201.13.21.0/24
        - ipBlock:
            cidr: 142.250.187.0/24
      ports:
        - protocol: TCP
          port: 6789
        - protocol: TCP
          port: 7890
    ...
  thor:
    ...
    egress: engineEgress
```

Note que o nome 'engineEgress' é um nome arbitrário, qualquer nome pode ser escolhido, e qualquer número dessas seções de egresso nomeadas pode ser definido.

Para mais informações, por favor, veja a seção *egress*: no arquivo YAML padrão fornecido pelo HPCC Systems. O arquivo *values.yaml* pode ser encontrado no diretório *helm/hpcc/* no repositório github do HPCC Systems:

<https://github.com/hpcc-systems/HPCC-Platform>

## 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 o local onde é possível obter informações no sistema, caso não as queira na fonte. Tal como código embarcado, você pode definir isso nas seções de sign do código. Se você tiver informações que não deseja publicar, mas precisa executá-las, poderá usar secrets. Existe uma categoria chamada "eclUser", onde você colocaria o conteúdo que deseja ler diretamente do código ECL. Outras categorias de secrets, incluindo a categoria "ecl", são lidas internamente pelos componentes do sistema e não expostas diretamente ao código ECL.

## Vaults

Vaults é outra maneira de esconder informações. A seção de Vaults espelha a seção Secrets, mas aproveita *HashiCorpVault* para o armazenamento dos secrets. Existe uma categoria para vaults chamada "eclUser". A intenção da categoria de vault eclUser é ser legível diretamente do código ECL. Adicione apenas configurações de vault à categoria "eclUser" que você deseja que os usuários ECL possam acessar. Outras categorias de vault, incluindo a categoria "ecl", são lidas internamente pelos componentes do sistema e não expostas diretamente ao código ECL.

## Manipulação de Recursos de Origem Cruzada

O compartilhamento de recursos de origem cruzada (CORS) é um mecanismo para integrar aplicativos em diferentes domínios. CORS define como as aplicações web do cliente em um domínio podem interagir com recursos em outro domínio. Você pode configurar as configurações de suporte ao CORS na seção ESP do arquivo values.yaml, conforme ilustrado abaixo:

```
esp:
- name: eclwatch
  application: eclwatch
  auth: ldap
  replicas: 1
  # The following 'corsAllowed' section is used to configure CORS support
  #   origin - the origin to support CORS requests from
  #   headers - the headers to allow for the given origin via CORS
  #   methods - the HTTP methods to allow for the given origin via CORS
  #
  corsAllowed:
  # origin starting with https will only allow https CORS
  - origin: https://*.example2.com
    headers:
      - "X-Custom-Header"
    methods:
      - "GET"
  # origin starting with http will allow http or https CORS
```

```
- origin: http://www.example.com
  headers:
  - "*"
  methods:
  - "GET"
  - "POST"
```

## Visibilidades

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

## Réplicas e Recursos

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.

## Valores do Ambiente

Você pode definir variáveis de ambiente em um arquivo YAML. Os valores do ambiente são definidos na seção *global.env* do arquivo values.yaml fornecido pelo HPCC Systems. Esses valores são especificados como uma lista de pares de nome-valor, conforme ilustrado abaixo.

```
global:
  env:
  - name: SMTPserver
    value: mysmtpserver
```

A seção global.env do arquivo values.yaml fornecido adiciona variáveis de ambiente padrão para todos os componentes. Você também pode especificar variáveis de ambiente para os componentes individuais. Consulte o esquema para definir esse valor para componentes individuais.

Para adicionar valores de ambiente, você pode inseri-los no seu arquivo YAML de configuração de personalização quando fizer o deploy do seu HPCC Systems contêinerizado.

## Variáveis de Ambiente para Sistemas Containerizados

Existem várias configurações em `environment.conf` para sistemas bare-metal. Embora muitas configurações de `environment.conf` não sejam válidas para contêineres, algumas podem ser úteis. Em uma implantação na nuvem, essas configurações são herdadas de variáveis de ambiente. Essas variáveis de ambiente são configuráveis usando o `values.yaml` globalmente ou no nível do componente.

Algumas dessas variáveis estão disponíveis para implantações em contêineres e na nuvem e podem ser definidas usando o gráfico do Helm. Os seguintes valores de `environment.conf` para sistemas bare-metal têm valores equivalentes que podem ser definidos para instâncias containerizadas.

Environment.conf Value	Helm Environment Variable
<code>skipPythonCleanup</code>	<code>SKIP_PYTHON_CLEANUP</code>
<code>jvmlibpath</code>	<code>JAVA_LIBRARY_PATH</code>
<code>jvmoptions</code>	<code>JVM_OPTIONS</code>
<code>classpath</code>	<code>CLASSPATH</code>

O seguinte exemplo define a variável de ambiente para pular a limpeza do Python no componente Thor:

```
thor:
  env:
    - name: SKIP_PYTHON_CLEANUP
      value: true
```

## Index Build Plane

Defina o valor `indexBuildPlane` como uma opção de gráfico de helm para permitir que arquivos de índice sejam escritos por padrão em um plano de dados diferente. Ao contrário de arquivos planos, os arquivos de índices têm requisitos diferentes. Os arquivos de índice se beneficiam de armazenamento de acesso aleatório rápido. Normalmente, arquivos planos e arquivos de índices são gravados nos planos de dados padrão definidos. Usando esta opção, você pode definir que os arquivos de índice são construídos em um plano de dados separado de outros arquivos comuns. Este valor de gráfico pode ser fornecido em um componente ou nível global.

Por exemplo, adicionando o valor a um nível global em `global.storage`:

```
global:
  storage:
    indexBuildPlane: myindexplane
```

Optionally, you could add it at the component level, as follows:

```
thor:
- name: thor
  prefix: thor
  numWorkers: 2
  maxJobs: 4
  maxGraphs: 2
  indexBuildPlane: myindexplane
```

Opcionalmente, você poderia adicioná-lo no nível do componente, conforme segue:

## Pods e Nós

Uma das principais características do Kubernetes é sua capacidade de agendar pods nos nós do cluster. Um pod é a unidade mais pequena e simples no ambiente Kubernetes que você pode criar ou implantar. Um nó é uma máquina "trabalhadora", física ou virtual, no Kubernetes.

A tarefa de agendar pods para nós específicos no cluster é gerenciada pelo kube-scheduler. O comportamento padrão deste componente é filtrar os nós com base nas solicitações e limites de recursos de cada recipiente no pod criado. Os nós viáveis são então pontuados para encontrar o melhor candidato para a localização do pod. O agendador também leva em conta outros fatores, como afinidade e anti-afinidade de pods, marcas e tolerâncias, restrições de dispersão de topologia de pod, e os rótulos de seleção de nó. O agendador pode ser configurado para usar esses diferentes algoritmos e políticas para otimizar a colocação do pod de acordo com as necessidades do seu cluster.

Você pode implantar esses valores usando o arquivo values.yaml ou pode colocá-los em um arquivo e fazer com que o Kubernetes leia os valores do arquivo fornecido. Veja a seção acima *Técnicas de personalização* para mais informações sobre como personalizar sua implantação.

## Placements

O termo "Placements" é usado pelo HPCC Systems para descrever o que o Kubernetes se refere como agendador ou agendamento/atribuição. Para evitar confusão com a terminologia específica do agendador do HPCC Systems e ECL, usamos o termo "Placements" para nos referir ao agendamento do Kubernetes. Os Placements são um valor na configuração do HPCC Systems que reside em um nível acima de entidades como nodeSelector, Tolerância, Afinidade e Anti-Afinidade, e TopologySpreadConstraints.

O Placement é responsável por encontrar o melhor nó para um pod. Na maioria das vezes, a colocação é gerenciada automaticamente pelo Kubernetes. Você pode restringir um Pod para que ele possa ser executado apenas em um conjunto específico de Nós.

Placements, então, seriam usadas para garantir que pods ou jobs que requerem nós com características específicas sejam alocados nestes nós.

Por exemplo, um cluster Thor poderia ser direcionado para machine learning usando nós com GPU. Outro job pode precisar de nós com uma quantidade maior de memória ou outro para mais CPU.

Usando placements, 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>
```

## Escope do Placement

Use padrões de pods para aplicar o escopo para os placements.

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

Tipo: <component>	Cobre todos os pods/trabalhos sob este tipo de componente. Isso é comumente utilizado para os componentes do HPCC Systems. Por exemplo, o <i>type:thor</i> que se aplicará a qualquer componente do tipo Thor; thoragent, thormanager, thoragent e thorworker, etc.
-------------------	---

Destino: <name>	O campo "name" de cada componente, uso típico para componentes do HPCC Systems refere-se ao nome do cluster. Por exemplo, <i>Roxie</i> : <i>-name: roxie</i> que será o destino alvo "Roxie" (cluster). Você também pode definir vários alvos, cada um com um nome único, como "roxie", "roxie2", "roxie-web", etc.
Pod: <name>	Este é o nome dos metadados de "Implantação" a partir do nome do item de array de um tipo. Por exemplo, "eclwatch-", "mydali-", "thor-thoragent", o uso de uma expressão regular é preferido, pois o Kubernetes usará o nome dos metadados como prefixo e gerará dinamicamente o nome do pod, como, eclwatch-7f4dd4dd44cb-c0w3x.
Nome do Job:	O nome do job é geralmente também uma expressão regular, já que o nome do job é gerado dinamicamente. Por exemplo, um job de compilação compile-54eB67e567e, pode usar "compile-" ou "compile-.*" ou "^compile-.*"
All:	Aplica para todos os componentes do HPCC Systems. O padrão de implantação dos placements para pods é [all]

Independentemente da ordem em que os placements aparecem na configuração, eles serão processados na seguinte ordem: "all", "type", "target", e então "pod"/"job".

## Combinações mistas

NodeSelector, taints e tolerations, e outros valores pode ser colocado nos mesmos pods: [list] ambos por zona e por nós no Azure

```
placements:
- pods: ["eclwatch", "roxie-workunit", "^compile-.*$", "mydali"]
  placement:
    nodeSelector:
      name: npone
```

## Seleção de Nós

Em um ambiente de contêineres Kubernetes, existem várias maneiras de agendar seus nós. As abordagens recomendadas utilizam seletores de rótulos para facilitar a seleção. Geralmente, você pode não precisar definir tais restrições, pois o agendador normalmente faz um posicionamento razoavelmente aceitável automaticamente. No entanto, em algumas implantações, você pode querer ter mais controle sobre pods específicos.

O Kubernetes utiliza os seguintes métodos para escolher onde agendar os pods:

- campo nodeSelector correspondendo aos rótulos dos nós
- Afinidade e anti-afinidade
- Taints and Tolerâncias
- campo nodeName
- restrições de dispersão topológica de pods

## Rótulo dos Nós

Os nós do Kubernetes possuem rótulos. Kubernetes possui um conjunto padrão de rótulos utilizados para nós em um cluster. Você também pode anexar rótulos manualmente, o que é recomendado, já que o valor desses rótulos é específico do provedor de nuvem e não é garantido ser confiável.

Adicionar rótulos aos nós permite agendar pods para nós ou grupos de nós específicos. Você pode então usar essa funcionalidade para garantir que pods específicos sejam executados apenas em nós com determinadas propriedades.

## O nodeSelector

O nodeSelector é um campo na especificação do Pod que permite especificar um conjunto de rótulos de nó que devem estar presentes no nó de destino para que o Pod seja agendado lá. É a forma mais simples de restrição de seleção de nó. Ele seleciona nós com base nos rótulos, mas possui algumas limitações. Suporta apenas uma chave de rótulo e um valor de rótulo. Se você deseja corresponder a vários rótulos ou usar expressões mais complexas, precisa usar a Afinidade de Nó (node Affinity)

Adicione o campo nodeSelector à especificação do seu pod e especifique os rótulos de nó que você deseja que o nó de destino tenha. Você deve ter os rótulos de nó definidos no trabalho (job) e no pod. Em seguida, você precisa especificar cada grupo de nós que o rótulo de nó deve usar. O Kubernetes agendará o pod apenas nos nós que possuem os rótulos que você especificar.

O exemplo a seguir mostra o nodeSelector colocado na lista de pods. Este exemplo agenda todos os componentes do HPCC para usar o grupo de nós com o rótulo "group: hpcc".

```
placements:
  - pods: ["all"]
    placement:
      nodeSelector:
        group: "hpcc"
```

**Nota:** O rótulo group:hpcc corresponde ao rótulo do grupo de nós hpcc.

Este próximo exemplo mostra como configurar um grupo de nós para evitar o agendamento de um componente Dali neste grupo de nós rotulado com a chave spot: com o valor false. Como esse tipo de nó nem sempre está disponível e pode ser revogado, você não deseja usar o grupo de nós spot para componentes Dali. Este é um exemplo de como configurar um tipo específico (Dali) de componente do HPCC Systems para não usar um determinado grupo de nós.

```
placements:
  - pods: ["type:dali"]
    placement:
      nodeSelector:
        spot: "false"
```

Ao usar nodeSelector, múltiplos nodeSelectors podem ser aplicados. Se chaves duplicadas forem definidas, apenas a última prevalecerá.

## Taints e Tolerâncias

Taints e Tolerâncias são tipos de restrições de nó do Kubernetes, também referidas como Afinidade de Nó. Apenas uma afinidade pode ser aplicada a um pod. Se um pod corresponder a várias listas de posicionamento ('pods'), então apenas a última definição de afinidade será aplicada.

Taints e tolerâncias trabalham juntos para garantir que os pods não sejam agendados em nós inadequados. Tolerâncias são aplicadas aos pods e permitem (mas não exigem) que os pods sejam agendados em nós com taints correspondentes. Taints funcionam de forma oposta - elas permitem que um nó repila um conjunto de pods. Uma forma de implantar usando taints é configurar para repelir todos os nós exceto um específico. Então, esse pod pode ser agendado em outro nó que seja tolerante.

Por exemplo, os jobs do Thor devem estar todos no tipo apropriado de VM. Se um trabalho grande do Thor aparecer, então o nível de taints repele qualquer pod que tente ser agendado em um nó que não atenda aos requisitos.

Para mais informações e exemplos de nossos Taints, Tolerâncias, e Placements, por favor verifique nossa documentação de desenvolvimento:

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

## Exemplos de Taints e tolerâncias

Os exemplos a seguir ilustram como alguns taints e tolerâncias podem ser aplicados.

O Kubernetes pode agendar um pod para qualquer grupo de nós sem um taint. Nos exemplos seguintes, o Kubernetes só pode agendar os dois componentes para os grupos de nós com esses rótulos exatos, group e gpu.

```
placements:
- pods: ["all"]
  tolerations:
    key: "group"
    operator: "Equal"
    value: "hpcc"
    effect: "NoSchedule"
```

```
placements:
- pods: ["type:thor"]
  tolerations:
    key: "gpu"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"
```

Também é possível usar múltiplas tolerations. O exemplo a seguir possui duas tolerations, group e gpu.

```
#The settings will be applied to all thor pods/jobs and myecleccserver pod and job
- pods: ["thorworker-", "thor-thoragent", "thormanagent-", "thor-eclagent", "hthor-"]
  placement:
    nodeSelector:
      app: tf-gpu
    tolerations:
      - key: "group"
        operator: "Equal"
        value: "hpcc"
        effect: "NoSchedule"
      - key: "gpu"
        operator: "Equal"
        value: "true"
        effect: "NoSchedule"
```

Neste exemplo, o nodeSelector está impedindo o agendador do Kubernetes de implantar qualquer pod neste grupo de nós. Sem taints, o agendador poderia implantar qualquer pod no grupo de nós. Ao utilizar o nodeSelector, o taint forçará o pod a ser implantado apenas nos nós que correspondem a esse rótulo de nó. Existem, então, duas restrições neste exemplo: uma do grupo de nós e outra do pod.

## Restrições de Dispersão Topológica

Você pode usar restrições de dispersão topológica para controlar como os pods são distribuídos em seu cluster entre domínios de falha, como regiões, zonas, nós e outros domínios de topologia definidos pelo usuário. Isso pode ajudar a alcançar alta disponibilidade e utilização eficiente de recursos. Você pode definir restrições de dispersão topológica ao nível do cluster como padrão, ou configurá-las para cargas de trabalho individuais. As restrições de dispersão topológica **topologySpreadConstraints** requerem Kubernetes v1.19 ou superior.

Para mais informações:

<https://kubernetes.io/docs/concepts/workloads/pods/pod-topology-spread-constraints/> and

<https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints/>

Usando o exemplo de "topologySpreadConstraints", há dois grupos de nós que têm "hpcc=nodepool1" e "hpcc=nodepool2", respectivamente. Os pods do Roxie serão agendados de forma equitativa nos dois grupos de nós.

Após a implantação, você pode verificar emitindo o seguinte comando:

```
kubect1 get pod -o wide | grep roxie
```

o código dos placements:

```
- pods: ["type:roxie"]
  placement:
    topologySpreadConstraints:
      - maxSkew: 1
        topologyKey: hpcc
        whenUnsatisfiable: ScheduleAnyway
        labelSelector:
          matchLabels:
            roxie-cluster: "roxie"
```

## Afinidade e Anti-Afinidade

A afinidade e anti-afinidade ampliam os tipos de restrições que você pode definir. As regras de afinidade e anti-afinidade ainda são baseadas nos rótulos. Além dos rótulos, elas fornecem regras que orientam o agendador do Kubernetes sobre onde colocar pods com base em critérios específicos. A linguagem de afinidade/anti-afinidade é mais expressiva do que simples rótulos e oferece mais controle sobre a lógica de seleção.

Existem dois tipos principais de afinidade: Afinidade de Nó (Node Affinity) e Afinidade de Pod (Pod Affinity).

### Afinidade de Nó

A afinidade de nó (Node Affinity) é semelhante ao conceito de nodeSelector, que permite restringir em quais nós seu pod pode ser agendado com base nos rótulos dos nós. Ela é usada para restringir os nós que podem receber um pod, combinando os rótulos desses nós. A afinidade de nó e anti-afinidade só podem ser usadas para definir afinidades positivas que atraem pods para o nó.

Não há verificação de esquema para o conteúdo da afinidade. Apenas uma afinidade pode ser aplicada a um pod ou job. Se um pod/job corresponder a várias listas de posicionamento de pods, então apenas a última definição de afinidade será aplicada.

Para maiores informações, veja <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/>

Existem dois tipos de afinidade de nós:

*requiredDuringSchedulingIgnoredDuringExecution*: O agendador não pode agendar o pod a menos que esta regra seja cumprida. Essa função é semelhante ao nodeSelector, mas com uma sintaxe mais expressiva.

*preferredDuringSchedulingIgnoredDuringExecution*: O agendador tenta encontrar um nó que atenda à regra. Se um nó correspondente não estiver disponível, o agendador ainda assim agenda o pod.

Você pode especificar afinidades de nó usando o campo *spec.affinity.nodeAffinity* na especificação do seu pod.



## Afinidade de Pod

A afinidade de pod ou Inter-Pod Affinity é usada para restringir os nós que podem receber um pod combinando os rótulos dos pods existentes já em execução nesses nós. A afinidade de pod e anti-afinidade pode ser tanto uma afinidade atrativa quanto uma anti-afinidade repelente.

A afinidade inter-pod funciona de maneira muito semelhante à afinidade de nó, mas apresenta algumas diferenças importantes. Os modos "hard" e "soft" são indicados usando os mesmos campos *requiredDuringSchedulingIgnoredDuringExecution* e *preferredDuringSchedulingIgnoredDuringExecution*. No entanto, esses devem estar aninhados sob os campos *spec.affinity.podAffinity* ou *preferredDuringSchedulingIgnoredDuringExecution*, dependendo se você deseja aumentar ou reduzir a afinidade do pod.

## Exemplo de Afinidade

O código a seguir ilustra um exemplo de afinidade:

```
- pods: ["thorworker-.*"]
  placement:
    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
            - matchExpressions:
                - key: kubernetes.io/e2e-az-name
                  operator: In
                  values:
                    - e2e-az1
                    - e2e-az2
```

Na seção `schedulerName` a seguir, as configurações de "afinidade" também podem ser incluídas com esse exemplo.

**Nota:** O valor "afinidade" no campo "schedulerName" é suportado apenas nas versões beta do Kubernetes 1.20.0 e posteriores.

## schedulerName

O campo **schedulerName** especifica o nome do agendador responsável por agendar um pod ou uma tarefa. No Kubernetes, você pode configurar vários agendadores com nomes e perfis diferentes para serem executados simultaneamente no cluster.

Apenas um "schedulerName" pode ser aplicado a qualquer pod ou job.

Exemplo de schedulerName:

```
- pods: ["target:roxie"]
  placement:
    schedulerName: "my-scheduler"
#The settings will be applied to all thor pods/jobs and myec1ccserver pod and job
- pods: ["target:myec1ccserver", "type:thor"]
  placement:
    nodeSelector:
      app: "tf-gpu"
    tolerations:
      - key: "gpu"
        operator: "Equal"
        value: "true"
        effect: "NoSchedule"
```

# Fundamentos de Helm e YAML

Esta seção destina-se a fornecer informações básicas para ajudá-lo a iniciar sua implantação containerizada do HPCC Systems. Existem inúmeros recursos disponíveis para aprender sobre Kubernetes, Helm e arquivos YAML. Para mais informações sobre o uso dessas ferramentas, ou para implantações em nuvem ou container, consulte a documentação respectiva.

Na seção anterior, mencionamos os arquivos *values.yaml* e *values-schema.json*. Esta seção expande alguns desses conceitos e como eles podem ser aplicados ao usar a versão containerizada da plataforma HPCC Systems.

## A estrutura de arquivos *values.yaml*

O arquivo *values.yaml* é um arquivo YAML, que é um formato frequentemente usado para arquivos de configuração. A estrutura que compõe a maior parte de um arquivo YAML é o par chave-valor, às vezes referido como um dicionário. A construção de par chave-valor consiste em uma chave que aponta para algum valor ou valores. Esses valores são definidos pelo esquema.

Nos arquivos de configuração, a indentação usada para representar a estrutura do documento é bastante importante. Espaços à frente são significativos e o uso de tabulações não é permitido.

Os arquivos YAML são compostos principalmente por dois tipos de elementos: dicionários e listas.

### Dicionário

Dicionários são coleções de mapeamentos chave-valor. Todas as chaves são sensíveis a maiúsculas e minúsculas, e a indentação também é crucial. Essas chaves devem ser seguidas por dois pontos (:) e um espaço.

Por exemplo:

```
logging:
  detail: 80
```

Este é um exemplo de um dicionário para logging.

Os dicionários nos arquivos de valores passados, como no arquivo *myoverrides.yaml* no exemplo abaixo, serão mesclados nos dicionários correspondentes nos valores existentes, começando pelos valores padrão do gráfico do Helm do HPCC entregue.

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

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

### Listas

Listas são grupos de elementos que começam no mesmo nível de indentação, iniciando com um hífen e um espaço (-). Cada elemento da lista é indentado no mesmo nível e começa com um hífen e um espaço. Listas também podem ser aninhadas, e até mesmo ser listas de dicionários.

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

```
placements:
- pods: ["all"]
```

```
placement:
  tolerations:
    - key: "kubernetes.azure.com/scalesetpriority"
```

A entrada da lista aqui é denotada usando o hífen, que é um item de entrada na lista, que por sua vez é um dicionário com atributos aninhados. Em seguida, o próximo hífen (no mesmo nível de indentação) é a próxima entrada nessa lista. Uma lista pode ser uma lista de elementos de valor simples, ou os elementos podem ser listas ou dicionários.

## Sessões do HPCC Systems Values.yaml

A primeira seção do arquivo *values.yaml* descreve valores globais. Global se aplica, em geral, a tudo.

```
# Default values for hpcc.
global:
  # Settings in the global section apply to all HPCC components in all subcharts
```

No trecho do arquivo *values.yaml* do HPCC Systems fornecido (acima), *global*: é o dicionário de nível superior. Conforme observado nos comentários, as configurações na seção global se aplicam a todos os componentes do HPCC Systems. Note pelo nível de indentação que os outros valores estão aninhados nesse dicionário global.

Itens definidos na seção global são compartilhados entre todos os componentes.

Alguns exemplos de valores globais no arquivo *values.yaml* fornecido são as 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 gráfico.

## Utilização do HPCC Systems values.yaml

O arquivo *values.yaml* do HPCC Systems é usado pelo Helm chart para controlar como o HPCC Systems é implantado. O *values.yaml* do HPCC Systems fornecido é destinado a ser um guia de instalação do tipo quick start, que não é apropriado para uso prático não trivial. Você deve personalizar sua implantação para algo mais adequado às suas necessidades específicas.

Informações adicionais sobre implantações personalizadas são abordadas em seções anteriores, assim como na documentação do Kubernetes e Helm.

## Mesclagem e Sobreposição

Ter vários arquivos YAML, como um para logging, outro para armazenamento, e ainda outro para segredos, permite uma configuração granular. Esses arquivos de configuração podem todos estar sob controle de

versão, onde podem ser versionados, verificados, etc. Isso tem o benefício de definir/mudar apenas a área específica necessária, enquanto garante que áreas não modificadas permaneçam intocadas.

A regra a ser lembrada aqui, ao aplicar vários arquivos YAML, é que os últimos sempre sobrescreverão os valores nos primeiros. Eles sempre são mesclados em sequência. Os valores são sempre mesclados na ordem em que são fornecidos na linha de comando do Helm.

Outro ponto a considerar é que, quando há um dicionário global como root: e seu valor é redefinido em um segundo arquivo (como um dicionário), ele não será sobrescrito. Você simplesmente não pode sobrescrever um dicionário. Você pode redefinir um dicionário e defini-lo como nulo, o que efetivamente apagará o primeiro.

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

Another means to wipe out every value in a list is to pass in an empty set denoted by a `[ ]` such as this example:

```
bundles: [ ]
```

Isso apagaria quaisquer propriedades definidas para bundles.

## Aplicações gerais

Esses itens são geralmente aplicáveis aos nossos arquivos YAML do HPCC Systems Helm.

- Todos os nomes devem ser únicos.
- Todos os prefixos devem ser únicos.
- Serviços devem ser únicos.
- Arquivos YAML são mesclados em sequência.

Em relação aos componentes do HPCC Systems, os componentes são principalmente listas. Se você tiver uma lista de valores vazia denotada por `[ ]`, isso invalidaria essa lista em outros lugares.

## Utilização adicional

Os componentes do HPCC Systems são adicionados ou modificados passando valores de substituição. Os valores do Helm chart são substituídos, seja passando arquivos de substituição usando `-f` (para arquivo de substituição) ou via `--set`, onde você pode substituir um único valor. Esses valores passados são sempre mesclados na mesma ordem em que são fornecidos na linha de comando do Helm.

Por exemplo:

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

Substitui quaisquer valores no *values.yaml* fornecido, passando valores definidos em *myoverrides.yaml*.

Você também pode usar `--set` conforme o seguinte exemplo:

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

Para sobrepor somente um valor em específico.

É até possível combinar substituições de arquivo e de valor único, por exemplo:

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

No exemplo anterior, a flag `--set` substitui o valor para `storage.daliStorage.plane` (se) definido no `myoverrides.yaml`, que substituiria qualquer configuração do arquivo `values.yaml` e resultaria na definição de seu valor para `dali-plane`.

Se a flag `--set` for usada no comando `helm install` ou `helm upgrade`, esses valores são simplesmente convertidos para YAML no lado do cliente.

Você pode especificar as flags de substituição várias vezes. A prioridade será dada ao último (mais à direita) arquivo especificado.

## Configurações Global/Expert

A seção 'expert' em 'global' do `values.yaml` deve ser usada para definir configurações de baixo nível, de teste ou de desenvolvedor. Esta seção do helm chart é destinada a ser usada para opções personalizadas, de baixo nível ou de depuração, portanto, na maioria das implantações, deve permanecer vazia.

Este é um exemplo de como a seção global/avançada pode se parecer:

```
global:
  expert:
    numRenameRetries: 3
    maxConnections: 10
    keepalive:
      time: 200
      interval: 75
      probes: 9
```

NOTA: Alguns componentes (como o DfuServer e Thor) também possuem uma área de configurações 'avançadas' (veja o esquema de valores) que pode ser usada para configurações relevantes em uma base por instância de componente, em vez de defini-las globalmente.

As seguintes opções estão disponíveis:

- numRenameRetries** (unsigned) Se definido para um número positivo, a plataforma tentará novamente renomear um arquivo físico em caso de falha (após um curto atraso). Normalmente, isso não deveria ser necessário, mas em alguns sistemas de arquivos pode ajudar a mitigar problemas onde o arquivo acabou de ser fechado e não foi exposto corretamente na camada posix.
- maxConnections** (unsigned) Esta é uma configuração do Servidor DFU. Se definido, irá limitar o número máximo de conexões paralelas e streams de partição que estarão ativas ao mesmo tempo. Por padrão, um trabalho DFU executará tantas conexões/streams ativas quanto houver partições envolvidas no spray, limitado a um máximo absoluto de 800. A configuração `maxConnections` pode ser usada para reduzir essa concorrência. Isso pode ser útil em alguns cenários onde a concorrência está causando congestionamento na rede e desempenho degradado.
- keepalive** (time: sem sinal, interval: sem sinal, probes: sem sinal) Veja o exemplo de keepalive acima. Se definido, essas configurações irão substituir as configurações padrão de keepalive do soquete do sistema cada vez que a plataforma criar um soquete. Isso pode ser útil em alguns cenários se as conexões forem fechadas prematuramente por fatores externos (por exemplo, firewalls). Um exemplo disso é que instâncias do Azure irão fechar soquetes que permaneceram ociosos por mais de 4 minutos quando conectados fora de suas redes.

# Registro em Contêiner

## Contexto de Registro

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

Seguindo as metodologias containerizadas mais amplamente aceitas, as informações de log dos componentes do HPCC Systems são direcionadas para os fluxos de saída padrão, em vez de arquivos locais. Em implantações containerizadas, não existem logs de componentes escritos em arquivos como nas edições anteriores.

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

Por exemplo:

```
>kubectl 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 registros são efêmeros por natureza, e podem ser perdidos se o pod for despejado, o contêiner travar, o nó morrer, etc. Devido à natureza dos sistemas containerizados, é provável que os registros relacionados se originem de vários locais e precisem ser coletados e processados. 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 "faça você mesmo", ou recursos gerenciados disponíveis de 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, conforme a complexidade do cluster cresce ou a importância de reter o conteúdo dos registros cresce, uma arquitetura de log de nível de cluster deve ser empregada.

O registro de nível de cluster para o cluster do HPCC Systems em contêiner pode ser realizado incluindo um agente de log em cada nó. A tarefa de cada agente é expor os registros ou empurrá-los para um back-end de processamento de registro. Os agentes de registro geralmente não são fornecidos prontos, mas há vários disponíveis, como Elasticsearch e Stackdriver Logging. Vários provedores de nuvem oferecem soluções integradas que colhem 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 do log de streaming.

## Soluções de Processamento de logs

Existem várias soluções de processamento de logs disponíveis. Você poderia optar por integrar os dados de log do HPCC Systems com quaisquer soluções de log existentes, ou implementar outra especificamente

para os dados do HPCC Systems. A partir da versão 8.4 do HPCC Systems, fornecemos uma solução de processamento de logs leve, mas completa, para sua conveniência. As próximas seções irão analisar algumas das possíveis soluções.

# Solução Gerenciada Elastic Stack

O HPCC Systems fornece um chart Helm gerenciado, *elastic4hpcclogs*, que utiliza os chart Helm da Elastic Stack para Elastic Search, Filebeats e Kibana. Este chart descreve uma instância local mínima da Elastic Stack para processamento de log de componentes do HPCC Systems. Uma vez implantado com sucesso, os logs de componentes do HPCC produzidos no mesmo namespace devem ser automaticamente indexados no ponto de extremidade do Elastic Search. Os usuários podem consultar esses logs emitindo consultas da API RESTful do Elastic Search, ou via interface de usuário do Kibana (após a criação de um padrão de índice simples).

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

Um padrão de índice Kibana é criado automaticamente com base no layout de índice filebeat padrão.

## Instalando o chart elastic4hpcclogs

Instalar a solução simples fornecida é, como o nome indica, simples e uma maneira conveniente de reunir e filtrar dados de log. Ela é instalada através de nossos charts helm do repositório HPCC Systems. No diretório HPCC-platform/helm, o gráfico *elastic4hpcclogs* é entregue junto com os outros componentes da plataforma HPCC Systems. As próximas seções mostrarão como instalar e configurar a solução de log da Elastic Stack para o HPCC Systems.



**NOTA:** O chart *elastic4hpcclogs* não habilita nenhuma segurança. A responsabilidade de determinar a necessidade de segurança e habilitar a segurança em qualquer instância do Elastic Stack implantada ou componentes é de sua responsabilidade e de sua organização.

## Adicionando o Repositório HPCC Systems

O chart Elastic para HPCC Systems fornecido pode ser encontrado no repositório Helm do HPCC Systems. Para buscar e implantar os gráficos gerenciados pelo HPCC Systems, adicione o repositório Helm do HPCC Systems, se ainda não o fez:

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

Uma vez que este comando tenha sido concluído com sucesso, o chart *elastic4hpcclogs* estará acessível.

Confirme se o chart apropriado foi descarregado.

```
helm list
```

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



```
Windows PowerShell
PS C:\hd41> helm search repo hpcc

NAME                CHART VERSION  APP VERSION  DESCRIPTION
hpcc/elastic4hpcclogs 1.0.2          7.12.0       A Helm chart for launching a lightweight Elastic Search instance
hpcc/hpcc            8.4.18         8.4.18       A Helm chart for launching a HPCC cluster using Kubernetes
hpcc/hpcc-azurefile 0.1.0          1.16.0       A helm chart to provision HPCC PVC's using Azure File Storage
hpcc/hpcc-efs        0.1.0          1.16.0       A helm chart to provision HPCC PVC's using Amazon EFS
hpcc/hpcc-filestore 0.1.0          0.1.0        A helm chart to provision HPCC PVC's using Amazon File Storage
hpcc/hpcc-localfile  0.1.0          1.16.0       A helm chart to provision HPCC PVC's using local storage
hpcc/hpcc-localplanes 0.1.0          1.16.0       A helm chart to provision multiple HPCC PVC's using local storage
hpcc/hpcc-nfs        0.1.0          1.16.0       A Helm chart to provision HPCC PVC's using NFS
hpcc/prometheus4hpccmetrics 0.0.1        0.50.0       A Helm chart for deploying a Kubernetes Prometheus
```

## Instalando o chart elastic4hpcc

Instale 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", caso em que você emitiria o comando de instalação da seguinte forma:

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

Após a execução com sucesso, 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:** OBSERVE: 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 continuar a acumular custos mesmo após não usar mais a sua instância. Você deve garantir que nenhum componente (como PVCs) persista e continue a acumular custos.

NOTA: Dependendo da versão do Kubernetes, os usuários podem ser alertados sobre APIs obsoletas nos charts Elastic (ClusterRole e ClusterRoleBinding estão obsoletos na v1.17+). Implementações baseadas em Kubernetes < v1.22 não devem ser impactadas.

## Confirme se os Pods estão Prontos

Confirme que os pods Elastic estão prontos. Às vezes, após a instalação, os pods podem demorar alguns segundos para iniciar. Confirmar que os pods estão em um estado de prontidão é uma boa ideia antes de prosseguir. Para fazer isso, use o seguinte comando:

```
kubectl get pods
```

Este comando retorna as seguintes informações, exibindo os 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

```
Windows PowerShell
PS C:\hd41> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
dfuserver-7d5456576c-swdx             1/1     Running   0           18m
elclqueries-6df8f987c8-ggkx2          1/1     Running   0           18m
elcscheduler-7d8cf4b59d-lntxv         1/1     Running   0           18m
elcservices-59d4dd6c8d-4cpr5          1/1     Running   0           18m
elclwatch-595f856856-7bhc             1/1     Running   0           18m
elasticsearch-master-0                1/1     Running   0           17m
esdl-sandbox-76fddc544b-vr6pf         1/1     Running   0           18m
hthor-7db5d4cb5b-4hpkx                1/1     Running   0           18m
mydali-576d474c6-hzdqv                2/2     Running   0           18m
myelasticsearch-7f855f0c-p2f6m        1/1     Running   0           18m
myelk-filebeat-tjrs                   1/1     Running   0           17m
myelk-kibana-65cfcfb99c-wrtvf         1/1     Running   0           17m
roxie-agent-1-5c9dd45fd56-qpv28      1/1     Running   0           18m
roxie-agent-1-5c9dd45fd56-tsbth       1/1     Running   0           18m
roxie-agent-2-859bd57c8f-6vjxz       1/1     Running   0           18m
roxie-agent-2-859bd57c8f-95tfd       1/1     Running   0           18m
roxie-toposerver-7658ddc4-2lbg2       1/1     Running   0           18m
roxie-workunit-5cf7996b54-z4qc4       1/1     Running   0           18m
sasha-dfurecovery-archiver-8574c9d68c-shkjq 1/1     Running   0           18m
sasha-dfuwu-archiver-5f49b549bf-bxmn9 1/1     Running   0           18m
sasha-file-expiry-dc6f8df7-f2wwk      1/1     Running   0           18m
sasha-wu-archiver-7f48cc5c78-fkwf4    1/1     Running   0           18m
solr-agent-1-5c9dd45fd56-qpv28      1/1     Running   0           18m
```

Uma vez que todos os pods estejam indicando um estado 'ready' e 'Running', incluindo os três componentes para filebeats, Elastic Search e Kibana (destacados acima), você pode prosseguir.

## Confirmando os serviços Elastic

Para confirmar se os serviços Elastic estão em execução, utilize o seguinte comando:

```
$ kubectl get svc
```

Isto exibe a seguinte 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 delcarado como LoadBalancer para conveniência.

## Configurando os componentes do Elastic Stack

Você pode precisar ou querer personalizar os componentes da Elastic Stack. Os valores dos charts dos 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
```

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

## Utilizar os componentes de logs do HPCC Systems no Kibana

Uma vez habilitado e em execução, você pode explorar e consultar os logs de componentes do HPCC Systems a partir da interface de usuário do Kibana. Os padrões de índice do Kibana são necessários para explorar dados do Elastic Search a partir da interface de usuário do Kibana. Para mais informações sobre como usar a interface Elastic-Kibana, por favor, consulte a documentação correspondente:

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

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

e

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

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

## Configurando o logAccess para Elasticstack

O recurso *logAccess* permite que o HPCC Systems consulte e empacote logs relevantes para vários recursos, como o relatório ZAP, logs de assistente de WorkUnit, visualizador de logs ECLWatch, etc.

Uma vez que os logs são migrados ou direcionados para a instância de pilha elástica. A plataforma HPCC Systems precisa ser capaz de acessar esses logs. A forma como você direciona o HPCC Systems para fazer isso é fornecendo um arquivo de valores que inclui os mapeamentos de logs. Fornecemos um arquivo de valores padrão e fornecemos um exemplo de linha de comando que insere esse arquivo de valores em seu deployment. Esse arquivo de valores de configuração sugeridos para habilitar o acesso ao log pode ser encontrado no repositório GitHub da plataforma HPCC Systems.

<https://github.com/hpcc-systems/HPCC-Platform>

Em seguida, navegue até o arquivo *helm/examples/azure/log-analytics/loganalytics-hpcc-logaccess.yaml*.

Você pode usar o gráfico Elastic4HPCCLogs fornecido ou pode adicionar os valores lá ao seu arquivo yaml de valores de configuração personalizada.

Você pode então instalá-lo usando um comando, como o:

```
helm install mycluster hpcc/hpcc -f elastic4hpcclogs-hpcc-logaccess.yaml
```

# Solução de Análise de Logs do Azure

Os Serviços Kubernetes do Azure (AKS) e a Análise de Logs do Azure (ALA) são um recurso opcional projetado para ajudar a monitorar o desempenho e a saúde dos clusters baseados em Kubernetes. Uma vez habilitado e associado a um determinado AKS com um cluster do HPCC Systems, os logs de componentes do HPCC são automaticamente capturados pela Análise de Logs. Todos os dados STDERR/STDOUT são capturados e disponibilizados para fins de monitoramento e/ou consulta. Como geralmente ocorre com os recursos dos provedores de nuvem, o custo é uma consideração significativa e deve ser bem compreendido antes da implementação. O conteúdo do log é escrito na loja de logs associada à sua área de trabalho de Análise de Logs.

## Habilitando Azure Log Analytics

Habilite o Azure Log Analytics (ALA) no cluster AKS alvo usando uma das seguintes opções: Linha de comando direta, Linha de comando scriptada, ou pelo portal Azure.

Para obter informações mais detalhadas, consulte a documentação do Azure:

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

### Linha de comando

Para habilitar os insights do Azure Log Analytics a partir da linha de comando:

Você pode criar manualmente um workspace dedicado à análise de logs, ou pode pular esta etapa e utilizar o workspace padrão.

Para criar um workspace dedicado insira este comando:

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

Para habilitar o recurso de Análise de Logs em um cluster AKS de destino, faça referência ao id do recurso do workspace criado na etapa anterior:

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

### Linha de Comando Scriptada

Para conveniência, o HPCC Systems oferece um script para habilitar o ALA (com um workspace dedicado à análise de logs) no cluster AKS alvo.

O script `enable-loganalytics.sh` está localizado em:

<https://github.com/hpcc-systems/HPCC-Platform/tree/master/helm/examples/azure/log-analytics>

O script requer o preenchimento dos seguintes valores no arquivo de ambiente `env-loganalytics`.

Fornecer esses valores na ordem do arquivo de ambiente `env-loganalytics` para criar um novo workspace no Azure Log Analytics, associá-lo a um cluster AKS de destino, e habilitar o processamento de logs:

- **LOGANALYTICS\_WORKSPACE\_NAME** O nome desejado para o workspace do Azure Log Analytics a ser associada ao cluster AKS de destino. Um novo workspace é criado se esse valor não existir.

- **LOGANALYTICS\_RESOURCE\_GROUP** O grupo de recursos do Azure associado ao cluster AKS de destino.
- **AKS\_CLUSTER\_NAME** O nome do cluster AKS de destino para associar a análise de logs.
- **TAGS** As tags associadas com o novo workspace.

Por exemplo: "admin=MyName email=my.email@mycompany.com environment=myenv justification=testing"

- **AZURE\_SUBSCRIPTION** [Opcional] Garante que esta assinatura esteja configurada antes de criar o novo workspace

Uma vez que esses valores estejam preenchidos, o script `enable-loganalytics.sh` pode ser executado e ele irá criar o workspace de análise de logs e associá-la ao cluster AKS de destino.

## Portal Azure

Para habilitar Azure Log Analytics no portal Azure:

1. Select Target AKS cluster
2. Selecione Monitoring
3. Selecione Insights
4. Enable - escolha default workspace

## Configure o logAccess do HPCC para Azure

O recurso *logAccess* permite que o HPCC Systems consulte e empacote logs relevantes para várias funcionalidades, como o relatório ZAP, logs auxiliares da WorkUnit, visualizador de log do ECLWatch, etc.

## Obtenha o Service Principal

Para conceder acesso à API Log Analytics, o Azure requer uma aplicação registrada no Azure Active Directory (AAD). Obtenha uma aplicação registrada no AAD.

Para mais informações sobre o registro de um Azure Active Directory, veja a documentação oficial do Azure:

<https://docs.microsoft.com/en-us/power-apps/developer/data-platform/walkthrough-register-app-azure-active-directory>

Dependendo da estrutura de sua assinatura Azure, pode ser necessário solicitar isso de um administrador de assinatura

## Forneça Informações da Aplicação Registrada no AAD

O *logAccess* do HPCC Systems requer acesso ao inquilino AAD, cliente, token e ID workspace alvo por meio de um objeto secreto seguro. Espera-se que o segredo esteja na categoria 'esp', e nomeado '*azure-logaccess*'.

Os seguintes pares de chave-valor são suportados

- aad-tenant-id

- aad-client-id
- aad-client-secret
- ala-workspace-id

O script está disponível em 'create-azure-logaccess-secret.sh':

<https://github.com/hpcc-systems/HPCC-Platform/tree/master/helm/examples/azure/log-analytics>

O script pode ser usado para criar o segredo necessário.

Exemplo de comando para criação manual de segredo (supondo que ./secrets-templates contenha um arquivo nomeado exatamente como as chaves acima):

```
create-azure-logaccess-secret.sh .HPCC-Platform/helm/examples/azure/log-analytics/secrets-templates/
```

Caso contrário, crie o segredo manualmente.

Exemplo de comando para criação manual de segredo (supondo que ./secrets-templates contenha um arquivo nomeado exatamente como as chaves acima):

```
kubectrl create secret generic azure-logaccess \
  --from-file=HPCC-Platform/helm/examples/azure/log-analytics/secrets-templates/
```

## Configure o logAccess do HPCC

A implantação do HPCC Systems alvo deve ser configurada para se direcionar para a área de trabalho do Azure Log Analytics acima, fornecendo os valores de logAccess apropriados (como ./loganalytics-hpcc-logaccess.yaml). O secret azure-logaccess previamente criado deve ser declarado e associado à categoria esp, isso pode ser realizado através do valor yaml dos segredos (como ./loganalytics-logaccess-secrets.yaml).

Exemplo:

```
helm install myhpcc hpcc/hpcc \
  -f HPCC-Platform/helm/examples/azure/log-analytics/loganalytics-hpcc-logaccess.yaml
```

## Acessando os Logs do HPCC Systems

A interface AKS Log Analytics no Azure fornece visualizações de métricas de saúde de cluster/nó/contêiner centradas em Kubernetes e links diretos para logs de contêineres por meio de interfaces de "análise de log". Os logs podem ser consultados via a linguagem de consulta "Kusto" (KQL).

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

Exemplo de consulta KQL para buscar entradas de log de "Resumo de transações" 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`

Output de exemplo:

```
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 do log. A interface de Análise de Log facilita a criação de alertas baseados nessas consultas, que podem ser usados para acionar e-mails, SMS, execução de Logic App, e muitas outras ações.



# Controlling HPCC Systems Logging Output

Os logs do HPCC Systems fornecem uma riqueza de informações que podem ser usadas para benchmarking, auditoria, debugging, monitoramento, etc. O tipo de informação fornecida nos logs e seu formato é trivialmente controlado via configuração padrão do Helm. Lembre-se que, no modo de contêiner, cada linha de saída de log é passível de incorrer um custo, dependendo do provedor e plano que você possui, e a verbosidade deve ser cuidadosamente controlada usando as seguintes opções.

Por padrão, os logs do componente não são filtrados, e contêm as seguintes colunas:

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

Os logs podem ser filtrados por Público-Alvo, Categoria ou Nível de Detalhe. Além disso, as colunas de saída podem ser configuradas. As configurações de logging podem ser aplicadas no nível global ou no nível do componente.

## Filtragem do Público-Alvo

Os públicos-alvo disponíveis incluem operador (OPR), usuário (USR), programador (PRO), monitor (MON), 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, toda a saída do log do componente para incluir apenas mensagens de Programador e Usuário:

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

## Filtragem da Categoria de Destino

As categorias de destino disponíveis incluem disaster (DIS), error (ERR), information (INF), warning (WRN), progress (PRO), event (EVT), metrics (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 de log da instância mydali deve incluir todas as classes, exceto progress:

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

## Configuração do Nível de Detalhe do Log

A verbosidade da saída do log pode ser ajustada de "apenas mensagens críticas" (1) até "relatar todas as mensagens" (100). O nível de log padrão é bastante alto (80) e deve ser ajustado de acordo.

Estes são os níveis de log disponíveis:

- CriticalMsgThreshold = 1;
- FatalMsgThreshold = 1;
- ErrMsgThreshold = 10;
- WarnMsgThreshold = 20;



- AudMsgThreshold = 30;
- ProgressMsgThreshold = 50;
- InfoMsgThreshold = 60;
- DebugMsgThreshold = 80;
- ExtraneousMsgThreshold = 90;

Por exemplo, para exibir somente o progresso e o baixo nível (mais crítico)

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

## Configuração da Coluna de Dados de Log

As colunas de dados de log disponíveis incluem messageid (MID), audience (AUD), class (CLS), date (DAT), time (TIM), node (NOD), millitime (MLT), microtime (MCT), nanotime (NNT), processid (PID), threadid (TID), job (JOB), use (USE), session (SES), code (COD), component (COM), quotedmessage (QUO), prefix (PFX), all (ALL), e standard (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, toda a saída de log do componente deve incluir as colunas padrão, exceto a coluna do ID do job:

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

O ajuste dos valores de log por componente pode exigir a afirmação de vários valores específicos do componente, o que pode ser inconveniente de fazer via o parâmetro da linha de comando `--set`. Nestes casos, um arquivo de valores personalizados poderia ser usado para definir todos os campos requeridos.

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
```

## Configuração de Logging Assíncrono

Por padrão, as entradas de log serão criadas e registradas de forma assíncrona, para não bloquear o cliente que está registrando. As entradas de log serão mantidas em uma fila e dispensadas em uma thread em segundo plano. Essa fila tem um limite, quando atingido, o cliente ficará bloqueado aguardando disponibilidade. Alternativamente, o comportamento pode ser configurado para que, quando esse limite for atingido, as entradas de log sejam descartadas e perdidas para evitar qualquer bloqueio potencial.

NB: normalmente, espera-se que a pilha de registro acompanhe e que o limite de fila padrão seja suficiente para evitar qualquer bloqueio.

Os padrões podem ser configurados definindo a `<section>.logging.queueLen` e/ou `<section>.logging.queueDrop`.

Definir `<section>.logging.queueLen` como 0, desativará o registro assíncrono, ou seja, cada registro bloqueará até ser concluído.

Definir `<section>.logging.queueDrop` para um valor diferente de zero (N) fará com que N entradas de registro da fila sejam descartadas se a `queueLen` for atingida.