IBM Spectrum Computing Quickstart Guide

Version 0.3

December 2019

Table of Contents

Introduction

This document describes how to install the IBM Spectrum Computing Technical Preview (TP) on Kubernetes. It provides an overview of the installation so you can see the components that are deployed, as well as a discussion of how those components work, and how to test and use them.

IBM Spectrum Computing TP delivers three key capabilities:

1. Effectively manages highly variable demands in workloads within a finite supply of resources
2. Provides improved service levels for different consumers and workloads in a shared multitenant environment
3. Optimizes the usage of expensive resources such as general-purpose graphics processing units (GPGPUs) to help ensure that they are allocated the most important work

Overview

IBM Spectrum Computing TP builds on IBM Spectrum Computing's rich heritage in workload management and orchestration in demanding High Performance Computing (HPC) and enterprise environments. With this strong foundation, IBM Spectrum Computing TP brings a wide range of workload management capabilities that include:

- Multilevel priority queues and preemption
- Fairshare among projects and namespaces
- Resource reservation
- Dynamic load-balancing
- Topology-aware scheduling
- Capability to schedule GPU jobs with consideration for CPU or GPU topology
- Parallel Jobs
- Elastic jobs, a form of parallel job where the number of worker pods can change.
- Time-windows
- Time-based configuration
- Advanced reservation
- Workflows

Improved workload prioritization and management

IBM Spectrum Computing TP adds robust workload orchestration and prioritization capabilities to Kubernetes clusters. Kubernetes provides an application platform for developing and managing on-premises, containerized applications.

While the Kubernetes scheduler employs a basic “first come, first served" method for processing workloads, IBM Spectrum Computing TP enables organizations to effectively prioritize and manage workloads based on business priorities and objectives.

Key capabilities of IBM Spectrum Computing Technical Preview

Workload Orchestration

Kubernetes provides effective orchestration of workloads if there is capacity. In the public cloud, the environment can usually be enlarged to help ensure that there is always capacity in response to workload demands. However, in an on-premises deployment of Kubernetes, resources are ultimately finite. For workloads that dynamically create Kubernetes pods (such as Jenkins, Jupyter Hub, Apache Spark, Tensorflow, ETL, and so on), the default "first come, first served" orchestration policy is not enough to help ensure that important business workloads process first or get resources before less important workloads. IBM Spectrum Computing Pod Scheduler prioritizes access to the resources for key business processes while lower priority workloads are queued until resources can be made available.

Service Level Management

In a multitenant environment where there is competition for resources, workloads (users, user groups, projects, and namespaces) can be assigned to different service levels that help ensure the right workload gets access to the right resource at the right time. This function prioritizes workloads and allocates a minimum number of resources for each service class. In addition to service levels, workloads can also be subject to prioritization and multilevel fairshare policies, which maintain correct prioritization of workloads within the same Service Level Agreement (SLA).

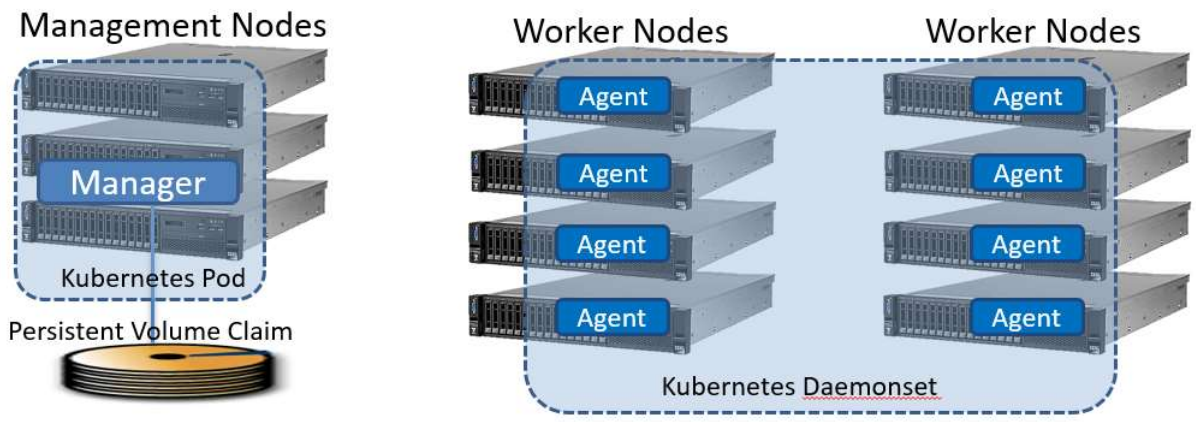
Resource Optimization

Environments are rarely homogeneous. There might be some servers with additional memory or some might have GPGPUs or additional capabilities. Running workloads on these servers that do not require those capabilities can block or delay workloads that do require additional functions. IBM Spectrum Computing TP provides multiple polices such as multilevel fairshare and service level management, enabling the optimization of resources based on business policy rather than by users competing for resources.

Architecture

Installed Components

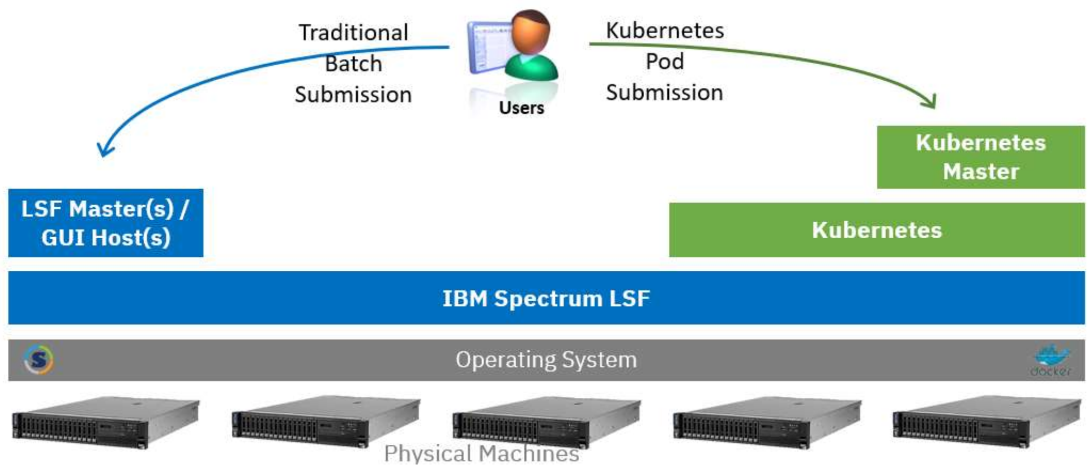
This installation takes an existing Kubernetes cluster and deploys the components needed to provide enhanced scheduling. When deployed you will have a cluster like the following example:



Agents will be deployed on all the worker nodes. These are deployed as a daemonset and gather information for the Manager to use for job execution and data collection. One of the management nodes may run the Manager pod. The NodeSelector on the Manager pod will control where it is located. The Manager pod runs the scheduling and resource management processes needed for the jobs. The Manager pod uses the persistent volume claim to store job data.

Alternate Installations

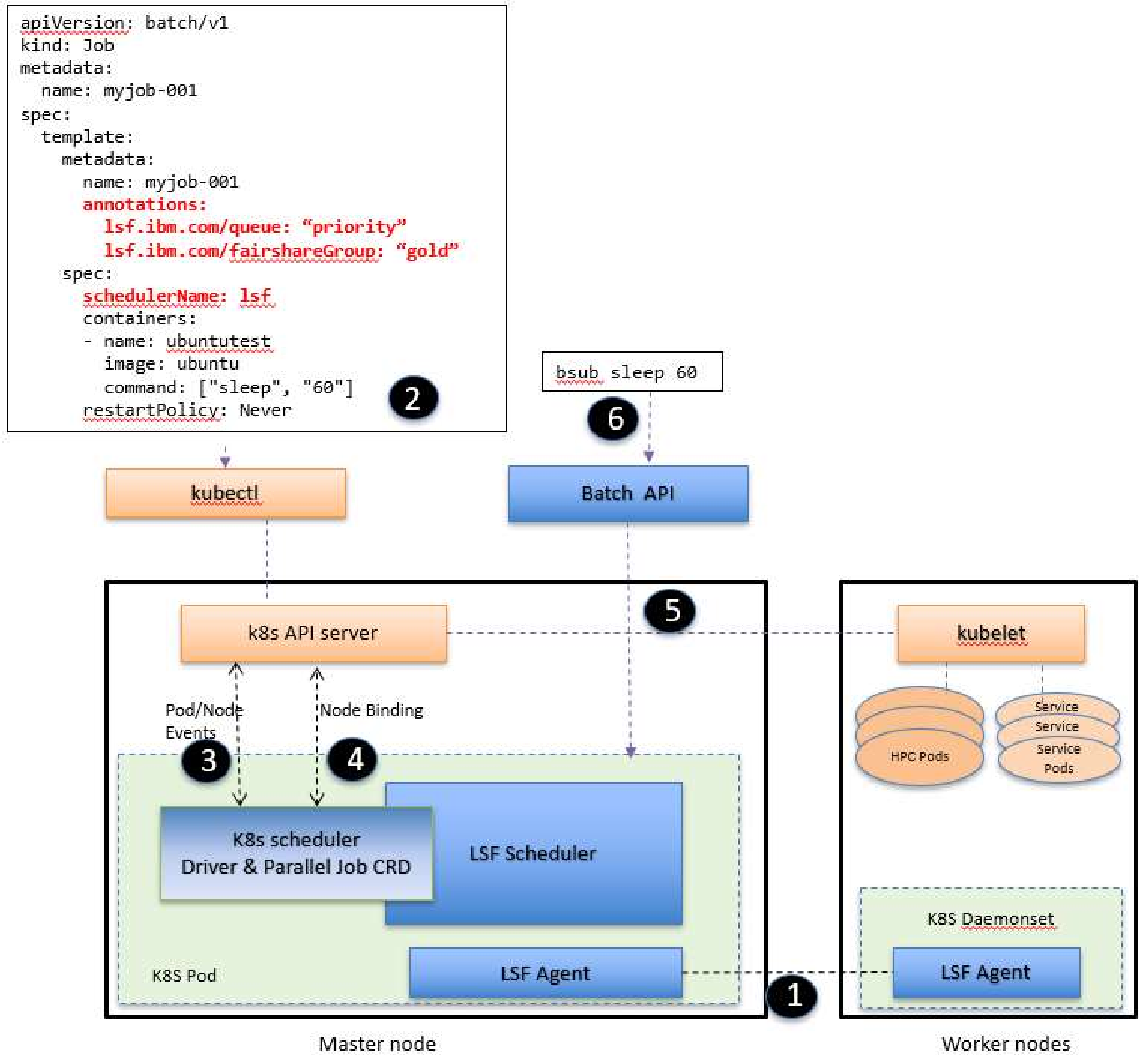
Existing IBM Spectrum LSF users that want to have the service capabilities of Kubernetes in an LSF Cluster can use LSF Suite 10.2.0.9 to get these features. The illustration below shows how that can be installed.



In this configuration users can submit both traditional LSF Jobs and Kubernetes Pods in a shared pool of resources. Consult the LSF Suite installation documentation for more information on how to install this configuration.

Enhanced Kubernetes Scheduling

The core Spectrum LSF scheduling technology has been integrated into Kubernetes to combine the expressive power of the Kubernetes API with the rich resource sharing and load-balancing technology that is at the heart of LSF. Here is how it works:



```
apiVersion: batch/v1
kind: Job
metadata:
  name: myjob-001
spec:
  template:
    metadata:
      name: myjob-001
      annotations:
        lsf.ibm.com/queue: "priority"
        lsf.ibm.com/fairshareGroup: "gold"
    spec:
      schedulerName: lsf
      containers:
      - name: ubuntutest
        image: ubuntu
        command: ["sleep", "60"]
      restartPolicy: Never
```

```
bsub sleep 60
```

1. The LSF Scheduler components are packaged into containers and deployed into a Kubernetes environment.
2. Users submits workload into K8S API via kubectl. To get the LSF Scheduler to be aware of the pod the "schedulerName" field must be set, otherwise the pod will be scheduled by the default scheduler. Scheduler directives can be specified using annotations in the pod. These scheduler directives can be provided by namespace annotations, mutating web hooks, or by the user. The workload may also be a parallel or elastic job.
3. In order to be aware of the status of pods and nodes, the LSF Scheduler uses a driver that listens to Kubernetes API server and translates pod requests into jobs in the LSF Scheduler.
4. Once the LSF Scheduler makes a policy decision on where to schedule the pod, the driver will bind the pod to specific node.
5. The Kubelet will execute and manages pod lifecycle on target nodes in the normal fashion.
6. The LSF Scheduler also supports jobs submitted from the native “bsub” CLI which are mapped to K8S pods and executed by Kubelet as well. In this way it is consistent.

Prerequisites

The following are needed to deploy IBM Spectrum Computing TP:

- A Kubernetes cluster with:
  - At least two machines with one dedicated “worker” node.
- A Persistent Volume (PV)
- Optional: Helm command line interfaces

IBM Spectrum Computing TP has been tested on OpenShift 3.11 and Kubernetes 1.16, 1.17, but should work on other Kubernetes versions or derivatives. A helm chart or a yaml file can be used to deploy the installation.

GPU Support requires use of the Nvidia Container Runtime. Documentation is provided here:
https://github.com/nvidia/nvidia-container-runtime

OpenShift 3.11 users may use the instructions here to enable GPU support without having to use privileged containers:
https://github.com/zvonkok/origin-ci-gpu/blob/release-3.11/doc/How%20to%20use%20GPUs%20with%20DevicePlugin%20in%20OpenShift%203.11%20.pdf

Installation and Verification Steps

Perform the following steps to evaluate IBM Spectrum Computing:

1. Extract the package and install the prerequisites
2. Create the namespace and service account
3. Deploy the IBM Spectrum Computing TP

4. Verify the IBM Spectrum Computing TP deployment
5. Deploy the sample jobs
6. Going further

The following sections will explain how to complete the steps.

Extract the Package and Install the Prerequisites

Extract the **`ibm-spectrum-computing-10.1.0.9.tgz`** file on the Kubernetes manager node, or a machine that can run "kubectl", or "oc" on the Kubernetes cluster. Inside the ibm-spectrum-computing-10.1.0.9 directory you will see:

- images - This contains the x86_64, and POWER images
- helm - This contains a Helm chart for deploying it
- openshift - This contain the files for deploying as a yaml file
- license - The licenses for this preview

A Persistent Volume (PV) is needed to store pod state information. It needs to have specific labels to ensure that it is the PV that is used during deployment. The sample yaml below creates a PV using NFS:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mypv
  labels:
    lsfvol: lsfvol
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: "Recycle"
  nfs:
    server: {Your NFS Servers IP}
    path: "{Your NFS Servers export path}"
```

Note the labels. These must be set. Other storage types may be used.

NOTE: When using an NFS persistent volume make sure to set the GID of the persistent volume.

If the NFS server is exporting "/export/stuff", check the group for /export/stuff for example,

```
# ls -la /export/stuff
total 8
drwxr-xr-x 2 mblack mygroup 4096 Mar 25 14:55 .
```

Here the group is “mygroup”.   Make sure the filesystem group read, write and execute bit is set by running:

```
# chmod 775 /export/stuff
```

Set the GID of the exported directory to 495 e.g.

```
# chgrp 495 /export/stuff
```

Create the Namespace and Service Account

IBM Spectrum Computing TP is isolated in its own namespace. This namespace is only for IBM Spectrum Computing TP components and should not be used for running any other pods. A script is provided to setup the environment. It is **`openshift/environment-setup.sh`**

The script will attempt to do the following:

1. Create a namespace to run the pods in
2. On OpenShift modify the admission controller to permit running on the master
3. May create the Security Context Constraints (SCC)
4. Create the Cluster role
5. Create the Clusterrole bindings to the Cluster role, and kube-scheduler
6. Grant the service account the cluster roles
7. May grant the service account the SCC
8. On OpenShift grant the service account access to the registry
9. Push the images to the registry using the service account
10. Generate a template to use to deploy

Before running the script edit it and set the registry location e.g.

```
# The registry which all machines in the cluster can access
# This should be a private registry.
# Do not push the images to a public registry
# Set the REGISTRY value to "none" if you are not using one
REGISTRY="docker-registry.default.svc:5000"
```

If you do not have an image registry set it to “none” e.g

```
REGISTRY="none"
```

Without an image registry it will be necessary to load the images on all worker nodes. This is done by logging into each worker node and running:

```
# cd openshift
```

```
# ./environment-setup.sh loadimage
```

After editing the script run it as the Kubernetes cluster administrator:

```
# cd openshift
# ./environment-setup.sh create
```

The script is re-runable, so if there is a failure it is safe to re-run it.

Deploy IBM Spectrum Computing TP

IBM Spectrum Computing TP can be deployed either using a Helm chart, or the supplied large yaml file. Using Helm allows for easier modification of some of the deployment parameters, however it requires Helm to be installed and configured. Choose the method you wish to use:

1. Helm chart installation
2. Yaml file deployment

Helm Chart Based Installation

A Helm chart for deploying is in the helm directory. To use this deployment method, you need to setup helm with the policies needed to create the deployment. From the Helm cli you can set many of the deployment parameters. The table below details the parameters and there default values that can be modified.

Parameter	**Description**	**Default**
`manager.image`	The image for the management pod	`lsf-master`
`manager.tag`	The image tag to use for the management pod	`10.1.0.9`
`manager.imagePullPolicy`	The image pull policy for the management image	`IfNotPresent`
`manager.cpu`	The amount of CPU to allocate to the manager	`1000m`
`manager.memory`	The amount of RAM to allocate to the manager	`4Gi`
`manager.excludeHostLabel`	Nodes with this label will be excluded	`excludelsf`
`manager.includeHostLabel`	Nodes with this label will be included	``
`compute.image`	The image for the worker pod	`lsf-comp`
`compute.tag`	The image tag to use for the worker pod	`10.1.0.9`
`compute.imagePullPolicy`	The image pull policy for the worker image	`Always`
`compute.cpu`	The amount of CPU to allocate to the agent	`500m`
`compute.memory`	The amount of RAM to allocate to the agent	`1Gi`
`compute.excludeHostLabel`	Nodes with this label will be excluded	`excludelsf`
`compute.includeHostLabel`	Nodes with this label will be included	``
`placement.tolerateName`	Nodes with this taint name can be used	``
`placement.tolerateVal`	Nodes with this taint value can be used	``
`placement.effect`	The effect used when the taint was created	`NoExecute`
`pvc.useDynamicProvisioning`	When true try to create storage automatically	`false`
`pvc.storageClassName`	Storage class to use for dynamic storage	`""`

Parameter	Description	Default
`pvc.existingClaimName`	When defined use this Persistent Volume Claim	`""`
`pvc.selector.label`	The selector label for the PV	`lsfvol`
`pvc.selector.value`	The selector label value for the PV	`lsfvol`
`pvc.size`	The minimum size of volume to use	`5Gi`
`pvc.fsGroup`	The GID to use for the fsGroup for the PVC	`495`
`pvc.supplementalGroups`	The GID to use in addition for the PVC	`495`
`monitoring.enableDashboard`	Flag to trigger data forwarding to Grafana	`false`
`serviceaccount`	The service account for scheduler access to API	`ibm-lsf-tp-sa`

Specify each parameter using the `--set key=value[,key=value]` argument to `helm install`.

To deploy using the default values use the following procedure after authenticating with Kubernetes.

```
# cd helm

# helm install --tls --namespace ibm-lsf-tp --name lsf .
```

NOTE: The "--tls" may not be necessary in your environment.

When the **`openshift/environment-setup.sh`** script was run, it attempted to upload the images to a registry. That registry should be used when running the helm cli e.g.

```
# cd helm

# helm install --tls --namespace ibm-lsf-tp --name lsf --set
manager.image={location of the lsf-master image},compute.image={Location of the lsf-
comp image} --debug .
```

If the **`REGISTRY`** was set to "none" in the **`openshift/environment-setup.sh`** then there should be no need to specify the image name. The default value will be correct.

If all goes well the pods should be ready for verification.

The deployment can also be deleted by running:

```
# cd helm

# helm delete --purge --tls lsf
```

Yaml File Based Installation

The Yaml file based installation provides an easier way to deploy, however it is more prescriptive in how the cluster is deployed. There are no options in how it is deployed, however users familiar with Kubernetes may still customize the deployment.

When the **openshift/environment-setup.sh** script was run it produced a yaml file called **ibm-lsf-tp-cluster-deploy.yam**l. This file can be used to deploy the cluster. This file may be modified before deployment. To deploy it use the following procedure:

For OpenShift:

```
# oc project ibm-lsf-tp

# oc create -f ibm-lsf-tp-cluster-deploy.yaml
```

For Kubernetes:

```
# kubectl create -n ibm-lsf-tp -f ibm-lsf-tp-cluster-deploy.yaml
```

Verify the IBM Spectrum Computing TP deployment

Use the procedures below to verify that the pods are running correctly. The OpenShift "oc" CLI can be used anywhere you see "kubectl" as it provides a superset of the "kubectl" CLI.

If you used Helm you can see the Helm deployment using:

```
$ helm list --tls |grep spectrum-computing
lsf          1                Mon Apr 25 15:07:56 2019        DEPLOYED
ibm-spectrum-computing-prod-1.0.0  2.1.0   ibm-lsf-tp
```

This shows that the chart is deployed and that it is called: "lsf".

Now check the daemonset by running the following command:

```
$ kubectl get daemonsets --namespace ibm-lsf-tp
NAME              DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE SELECTOR  AGE
lsf-..-agent      4        4        3      4           3          <none>         23m
lsf-..-gpu-agent  0        0        0      0           0          <none>         23m
```

There are two daemonsets, one for worker nodes with GPUs, and one for those without. Here we see there are no GPUs present. We also see desired number of pods is 4, but only 3 are running. List the pods to see which one is not working:

```
$ kubectl get pods -n ibm-lsf-tp
NAME                          READY   STATUS              RESTARTS   AGE
lsf-…-agent-7mbjj             1/1     Running             0          24m
lsf-…-agent-h4ht2             1/1     Running             0          24m
lsf-…-agent-njqll             0/1     ContainerCreating   0          24m
lsf-…-agent-xh8z2             1/1     Running             0          24m
lsf-…-master-56b55d8-84gcj 1/1     Running             0          24m
```

If you see that the master pod is Pending, this is typically related to the Persistent Volume. If you see it is in ImagePullBackOff state, that typically means that it cannot get the image from the registry. More information is available using the following.

Check the logs of the container that is not in Running state with the following command:

```
$ kubectl logs -n ibm-lsf-tp lsf-ibm-spectrum-computing-prod-agent-njqll
```

Or get more information about the pod with the following command:

```
$ kubectl describe pod -n ibm-lsf-tp lsf-ibm-spectrum-computing-prod-
agent-njqll
```

Connect to the `master` pod for the following tests with the command:

```
$ kubectl exec -ti -n ibm-lsf-tp lsf-ibm-spectrum-computing-prod-master-
56b55d8-84gcj  bash

LSF POD [root@master /]#
```

NOTE: The prompt changed to indicate you are operating in a pod.

Try some LSF commands to see the cluster state for example:

```
LSF POD [root@master /]# lsid
IBM Spectrum LSF Standard Edition 10.1.0.0, Dec 04 2019
Copyright International Business Machines Corp. 1992, 2016.
US Government Users Restricted Rights - Use, duplication or disclosure
restricted by GSA ADP Schedule Contract with IBM Corp.

My cluster name is myCluster
My master name is lsfmaster
```

This tells you that the manager processes are running. Next check the workers state by running the following command:

```
LSF POD [root@master /]# lshosts -w
HOST_NAME         type    model   cpuf  ncpus maxmem maxswp server RESOURCES
lsfmaster         X86_64  Intel_EM  60.0    20 127.8G   3.9G    Yes (mg)
worker-10-1-1-10  X86_64  PC6000  116.1    12  31.2G   3.9G    Yes (kubernetes)
worker-10-1-1-11  X86_64  PC6000  116.1    10  31.2G   3.9G    Yes (kubernetes)
worker-10-1-1-12  X86_64  PC6000  116.1   1 0  31.2G   3.9G    Yes (kubernetes)
```

and

```
LSF POD [root@master /]# bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
worker-10-1-1-10       ok          -   1000      0      0      0      0      0
worker-10-1-1-11       ok          -   1000      0      0      0      0      0
worker-10-1-1-12       ok          -   1000      0      0      0      0      0
lsfmaster          closed          -      0      0      0      0      0      0
```

NOTE: The host names that the scheduler uses are not the pod names, they are a representation of the IP address of the host that is running the pod, and not the pod IP.

The machines STATUS should be ok. If the worker machine state is ok, they are ready to accept jobs. If the cluster just started or you reconfigured the cluster, it may take a minute for all machines to be ok.

Using the Enhanced Scheduler Capabilities

Once installed the IBM Spectrum Computing TP enables new options for pods run through Kubernetes. These options are needed for running High Performance Computing (HPC), High Performance Analytics (HPA) applications, and large AI jobs. They extend the pod specification with annotations that enable the features. To access the enhanced features the pod specification must have the schedulerName set to "lsf" for example:

```
spec.template.spec.schedulerName:  lsf
```

The new features for kubernetes jobs that this provides includes the following:

- Job Priority
- Application Profiles
- Fair sharing of resources
- Parallel and Elastic Jobs
- GPU Management
- Pod Workflows

The following table shows the new annotations along with the IBM Spectrum LSF equivalent.

Pod Spec Field	Description	LSF Job Submission Option
`*.metadata.name`	A name to assign to the job	Job Name (-J)
`++.lsf.ibm.com/project`	A project name to assign to job	Project Name (-P)
`++.lsf.ibm.com/application`	An application profile to use	Application Profile (-app)
`++.lsf.ibm.com/gpu`	The GPU requirements for the job	GPU requirement (-gpu)
`++.lsf.ibm.com/queue`	The name of the job queue to run the job in	Queue (-q)
`++.lsf.ibm.com/jobGroup`	A job group to assign to job	Job Group (-g)
`++.lsf.ibm.com/fairshareGroup`	The fairshare group to assign the job to	Fairshare Group (-G)
`++.lsf.ibm.com/user`	The user to run the application as, and for accounting	Job submission user
`++.lsf.ibm.com/reservation`	The resources to reserve prior to running the job	Advanced Reservation (-U)
`*.spec.containers[].resources.requests.memory`	The amount of memory to reserve for the job	Memory Reservation (-R "rusage[mem=...]")
`*.spec.schedulerName`	Set to "lsf"	N/A

NOTE:

*** - The pod specification files should be prefaced with *spec.template*.**

++ - The pod specification files should be prefaced with *spec.template.metadata.annotations*.

For information on the annotations and their meanings refer to the following:

https://www.ibm.com/support/knowledgecenter/SSWRJV_10.1.0/lsf_command_ref/bsub.man_top.1.html

Parallel or elastic jobs are a new "kind". Just like pods, jobs, deployments and daemonsets are "kinds", parallel jobs are a new "kind" of pod specification that allow the definition of parallel jobs. More details are below.

These capabilities are accessed by modifying the pod specifications for jobs. Below is a minimal example:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: myjob-001
spec:
  template:
    metadata:
      name: myjob-001
    spec:
      schedulerName: lsf        # This directs scheduling to the LSF Scheduler
      containers:
      - name: ubuntutest
        image: ubuntu
        command: ["sleep", "60"]
        resources:
          requests:
            memory: 5Gi
      restartPolicy: Never
```

This example enables Kubernetes to use **lsf** as the job scheduler. The LSF job scheduler can then apply its policies to choose when and where the job will run.

Additional parameters can be added to the pod yaml file to control the job. The example below shows how to use the additional annotations:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: myjob-001
spec:
  template:
    metadata:
      name: myjob-001
      # The following annotations provide additional scheduling
      # information to better place the pods on the worker nodes
      # NOTE:  Some annotations require additional LSF configuration
      annotations:
        lsf.ibm.com/project: "big-project-1000"
        lsf.ibm.com/queue: "normal"
        lsf.ibm.com/jobGroup: "/my-group"
        lsf.ibm.com/fairshareGroup: "gold"
    spec:
      # This directs scheduling to the LSF Scheduler
      schedulerName: lsf
      containers:
      - name: ubuntutest
        image: ubuntu
        command: ["sleep", "60"]
      restartPolicy: Never
```

In the previous example, the annotations provide the LSF scheduler more information about the job and how it should be run.

Parallel Jobs in Kubernetes

Kubernetes does not natively support parallel jobs. The IBM Spectrum Computing TP provides a new parallel job kind. Using this kind users came create parallel jobs and elastic jobs that will run within Kubernetes. To check that the capability exists run:

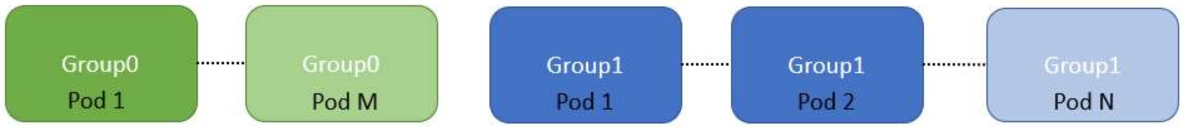
```
$ kubectl get crd
```

```
NAME                                        CREATED AT
<removed for clarity>
paralleljobs.ibm.com                        2019-11-19T20:02:44Z
```

NOTE: OpenShift users can use "oc" instead of "kubectl"

The "paralleljobs.ibm.com" Custom Resource Definition (CRD) extends the Kubernetes API to support the parallel job type. See Appendix A for the parallel job yaml file. Save the file as pjob-1.yaml

This sample parallel job has the following structure:



It can contain several groups of tasks. Each group is performing a specific function. For AI workloads one group might be used for running a parameter server. Within a group you can have one or more pods. These pods are running the same image in parallel.

In the sample provided there are two groups. The first group will run 1 pod and the second will run 2 pods. You can adjust the size by changing the number of replicas in each group.

Run the parallel job by running:

```
# kubectl create -f pjob-1.yaml
```

```
paralleljob.ibm.com/pjob-1 created
```

We can get more information about the parallel job by running:

```
# kubectl describe pj pjob-1
```

```
Name:          pjob-1
Namespace:     default
Labels:        test=experiment10
Annotations:   deletePodOnFlexDown=y
               lsf.ibm.com/queue=priority
API Version:   ibm.com/v1alpha1
Kind:          ParallelJob
  :                :
```

We can also see the pods in this job by running:

```
$ kubectl get pods -n default
```

```
NAME                        READY     STATUS    RESTARTS   AGE
pjob-1-group0-4pb4h         1/1       Running   0          13s
pjob-1-group1-9gkvx         1/1       Running   0          13s
pjob-1-group1-hl2kp         1/1       Running   0          13s
```

NOTE: The sample uses an Ubuntu image. If the image is not available a container creation error will be shown. Alter the sample to use an image you have access to.

To remove the parallel job run:

```
# kubectl delete -f pjob-1.yaml
```

```
paralleljob.ibm.com "pjob-1" deleted
```

Successful running pods demonstrates that the LSF Kubernetes connector is functioning properly and the Custom Resource Definition for parallel and elastic jobs is working.

A Note About Kubernetes

Users that submit a job through Kubernetes typically are trusted to run services and workloads as other users. For example, the pod specifications allow the pod to run as other users:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: myjob-uid1003-0002
spec:
  template:
    metadata:
      name: myjob-uid1003-0002
    spec:
      schedulerName: lsf
      containers:
      - name: ubuntutest
        image: ubuntu
        command: ["id"]
      restartPolicy: Never
      securityContext:
        runAsUser: 1003
        fsGroup: 100
        runAsGroup: 1001
```

In the previous example, the pod would run as UID 1003 and produce the following output:

```
uid=1003(billy) gid=0(root) groups=0(root),1001(users)
```

Note the GID and groups. Care should be taken to limit who can create pods. To eliminate this issue LSF applications can be used to allow the administrator to predefine pod specification file content and prevent users from switching UID. Alternatively, mutating web hooks may be used to modify pods as they are submitted.

Use Cases Enabled by the IBM Spectrum Computing Technical Preview

The table below outlines some of the test cases that can be evaluated with IBM Spectrum Computing. The test cases are not exhaustive but can help in evaluation. For more detailed or specific used cases contact IBM.

Num	Use Case Name	Use Case Description	Expected Results
1	User Pods & Applications	Not all user and pods are the same. Some may have much higher business impact. Some may be urgent. Some may be from a group contributing more to the operating expenses. Some may need special resources. How would a user: • Submit an urgent pod • Submit business important pod • Submit a pod needing GPUs • Submit multi-component applications • Submit a pod to only run on at night • Sequence pod execution	• Users can annotate their pods to get different SLAs • Users can submit their work to different queues with different policies • Users can assign their pods to different fairshare groups, to manage how the resource are shared • Users can annotate their pods to request GPUs • Users can create complex multi-component applications • Users can create pods that only run at a specific time
2	Pod Priority & Pre-emption	Company wants to run pods with different priorities. Wants high priority pods to be given the resources needed to finish quickly. Example: • A time sensitive pod needs to run before all other jobs • When resources are low, pre-empt lower priority pods	• Pods submitted to a higher priority queue get scheduled first. • When there is resource contention policies can be triggered
3	Run Windows	Company has different pods to run throughout the day. Wants to control the time of day at which some pods will run. Example: • Pods that can be run over night should not use resources during the day.	• Pods in the “night” queue do not start until the run window is open.
4	Resource Sharing	Company wants different groups to share the infrastructure, but not all groups are equal. Some groups should get higher resource budgets. Example:	• When there is contention for resources the “gold” group gets the highest resource share, while the others get less.

		• The core business group should have the highest resource budget, while others should have less. • Cap resource usage for those who have consumed their limits • No policies on how the resources are shared, just first in first run	• When there is contention for resources and a group has consumed there limit, allow other groups to consume their proportion of the resources. • With no policies the scheduler will default to First In First Run
5	Service / HPC Scheduling (Queue based)	Company wants to run a combination of services and computationally intensive pods on the same cluster but wants to reduce the "noisy neighbour" problem caused by computationally intensive jobs. Example: • Do not place service pods on machines that are "busy". Before placing a pod evaluate the worker nodes ability to run it. Busy can be defined via the builtin metrics, or extended with custom metrics • Create groups of workers to service the different types of pods	• Busy, but not over committed, worker nodes automatically are excluded from running some types of pods • A Service queue and a HPC queue can be defined with different host groups
6	Application Profiles	Administrators may wish to restrict how some pods are defined and used. They may also wish to have complex resource requirements, such as topology constraints for performance sensitive pods, or alternative resource requirements, or ranking of the suitability of worker nodes to run pods.	• The administrator can define application profiles that define complex resource requirements of pods. The users can use those profiles to launch there own pods.
7	Resource Reservation for Pods	Some resources can be too expensive to waste, or some applications can be too important to fail. The resources needed to run these applications must be reserved so that the application has the best chance of running successfully	• Jobs needing expensive are not delayed by may other smaller jobs.
8	Pod Dependencies	Many business processes require multiple steps to execute in a sequence. The results from one step	• A user can create a pod that will remain in a pending state until the pod it depends on finishes.

		in the process are the inputs to the next step in the process. Example: 1. Extract data from a database and stage to file 2. Process the file data and save to another file 3. Import the data from file to database	
9	GPU ROI	GPUs are very expensive resources to mismanage. Company wants GPU jobs to run as fast as possible, and with less chance of failure. Example: • Jobs typically require a specific GPU mode. Need to switch modes so job does not fail. • Multi-GPU jobs run faster on the same NUMA (best bandwidth)	• GPU mode changes based on the job needs, so they don't fail. • Jobs will be assigned the best set of GPUs to use, so the job runs faster.
10	Running other Application Middleware	Silos are a common response to dealing with the resource demands of complex middleware. Company wants to run IBM Spectrum Symphony Monti Carlo simulations in an OpenShift environment	• IBM Spectrum Symphony can run with other services and pods on OpenShift
11	Diagnostics	Users do not always request resources for pods that are reasonable or can be satisfied. Operators should not be burdened helping users figure out why their pod is not running.	• Users can use the Kubernetes CLI, or GUI to see the reasons why a job pod is not running
12	Queue based Run Limits	Misbehaving pods can waste resource and interfere with other important pods. Company wants to kill any pods that exhibit unusual resource consumption. Example: • An application should not run more than X minutes	• Pods running more than X minutes in a queue are normally killed

See Appendix B for additional information on how to run the test cases.

Going Further

Additional documentation and examples are available from:

https://github.com/IBMSpectrumComputing/lsf-kubernetes

Questions can also be emailed to: LSF-Inquiry@ca.ibm.com

Or posted on our slack channel. To join go here:

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W1559b1be149d_43b0_881e_9783f38faaff/page/Connect

Scheduler Administration

This section provides information on how to modify the schedulers configuration. The scheduler configuration files are stored in the persistent volume and in configMaps. The configuration can be changed by editing the files in the persistent volume claim or the configMaps. The persistent volume claim files can be edited either from within the pod or directly from the persistent volume.

The persistent volume claim has the following structure:

```
/{PV mount point} / lsf / conf
                        / work
```

Within the running containers the conf and work directories are symbolically linked to the following file paths:

```
/opt/ibm/lsfsuite/lsf/conf
/opt/ibm/lsfsuite/lsf/work
```

The conf directory has the following important files:

```
conf/cshrc.lsf
conf/profile.lsf
conf/hosts
conf/lsf.conf   ← ConfigMap
conf/lsf.cluster.myCluster
conf/lsf.shared
conf/lsf.task
conf/lsbatch/myCluster/configdir/lsb.users            ← ConfigMap
conf/lsbatch/myCluster/configdir/lsb.nqsmaps
conf/lsbatch/myCluster/configdir/lsb.reasons
conf/lsbatch/myCluster/configdir/lsb.hosts            ← ConfigMap
conf/lsbatch/myCluster/configdir/lsb.serviceclasses
conf/lsbatch/myCluster/configdir/lsb.resources        ← ConfigMap
conf/lsbatch/myCluster/configdir/lsb.modules
conf/lsbatch/myCluster/configdir/lsb.threshold
conf/lsbatch/myCluster/configdir/lsb.applications     ← ConfigMap
conf/lsbatch/myCluster/configdir/lsb.globalpolicies
conf/lsbatch/myCluster/configdir/lsb.params           ← ConfigMap
conf/lsbatch/myCluster/configdir/lsb.paralleljobs     ← ConfigMap
conf/lsbatch/myCluster/configdir/lsb.queues           ← ConfigMap
```

Details of the configuration files is in IBM Spectrum Computing documentation:

https://www.ibm.com/support/knowledgecenter/SSWRJV_10.1.0/lsf_welcome/lsf_kc_cluster_ops.html

Many configuration files are exposed as configMaps. They are:

1. **lsf.conf** - This is the main configuration file. It is exposed to allow changing the logging level
2. **lsb.applications** – This allows the administrator to define pre-set pod specifications that are not modifiable by the users.
3. **lsb.hosts** – This allows the administrator to create host groups and modify host scheduling metrics
4. **lsb.paralleljobs** – This contains configuration parameters for the parallel and elastic jobs
5. **lsb.params** – This contains parameters that control the operation of the scheduler
6. **lsb.queues** - This contains the queue definitions.
7. **lsb.resources** - This contains the users and user groups for configuring fairshare.
8. **lsb.users** - This contains the users and user groups for configuring fairshare.

Changes to the configMaps will be automatically applied to the cluster. Errors in the configMaps will cause the scheduler to revert to a default configuration, or not startup. To check for errors, it is necessary to connect to the manager pod and issue some commands as seen below.

1. Locate the master pod by looking for "ibm-spectrum-computing-prod-master" in the list of pods e.g.

   ```
   $ kubectl get pods -n ibm-lsf-tp |grep ibm-spectrum-computing-prod-master

   lsf-ibm-spectrum-computing-prod-master-56b55d6dc8-84gcj   1/1     Running
   0           3d19h
   ```

2. Connect to the management pod e.g.

   ```
   $  kubectl  exec  -ti  lsf-ibm-spectrum-computing-prod-master-56b55d6dc8-
   84gcj bash
   ```

3. Run the following command to check the configuration files e.g.

   ```
      LSF POD [root@lsfmaster /]# badmin chkconfig

      Checking configuration files ...

      No errors found.
   ```

4. An error might look like:

   ```
   LSF POD [root@lsfmaster /]# badmin chkconfig

   Checking configuration files ...
   ```

```
There are warning errors.

Do you want to see detailed messages? [y/n] y
Apr   22   13:14:49   2019   22437   4   10.1   orderQueueGroups():   File
/opt/ibm/lsfsuite/lsf/conf/lsbatch/myCluster/configdir/lsb.queues:
Priority value <20> of queue <night> falls in the range of priorities
defined for the queues that use the same cross-queue fairshare/absolute
priority scheduling policy. The priority value of queue <night> has been
set to 1
---------------------------------------------------------
No fatal errors found.
Warning: Some configuration parameters may be incorrect.
         They are either ignored or replaced by default values.

Do you want to restart MBD? [y/n]
```

NOTE: Changes to the configMaps can take a minute to be reflected in the LSF configuration file.

5. If errors are seen correct them in the configMap or configuration file. Changes to the configMaps will automatically trigger reconfiguration. Changes to other configuration files require the reconfiguration be triggered manually. This is done by running:

   ```
   LSF POD [root@lsfmaster /]# lsadmin reconfig
   LSF POD [root@lsfmaster /]# badmin mbdrestart
   ```

Log Files

The log files can provide useful information for troubleshooting problems. The log files to look at are in the Manager pod. To access the logs, use the following procedure:

1. Get the name of the Manager pod:

   ```
   $ kubectl get pods -n ibm-lsf-tp
   ```
   Look for the pod names containing "`ibm-spectrum-computing-prod-master`",

2. Dump the logs with:

   ```
   $ kubectl logs -n ibm-lsf-tp {Pod name from above}
   ```

3. Alternatively to see the logs in the pod connect to the manager pod:

   ```
   $ kubectl exec -ti {Pod name from above} bash
   ```

4. Disable forwarding the daemon logs to the pod STDOUT by running:

   ```
   LSF POD [root@master /]# touch /tmp/debug
   ```

5. Go to the log directory and look at logs:

   ```
   LSF POD [root@master /]# cd /opt/ibm/lsfsuite/lsf/log

   LSF POD [root@master /]# more kubebridge.lsfmaster.log
   ```

An example of a configuration error might look like the following:

```
LSF POD [root@master log]# more kubebridge.lsfmaster.log
Log file created at: 2019/03/27 14:38:40
Running on machine: lsfmaster
Binary: Built with gc go1.11.2 for linux/amd64
Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg
E0327 14:38:40.125268     376 jobhandler.go:308] Unable to submit the job
to lsf. cmd<bsub -q normal -g my-group -P big-project-1000 -G bestshare
-J  default/myjob-001-6hljx  -ext  kube[default/myjob-001-6hljx]  sleep
1000000> error<Bad or empty job group name. Job not submitted.>
```

To correct this problem, and ones like it, modify the job pod specification and make sure LSF is configured with the right fair share groups, job groups, users etc.

Backups

Configuration and state information is stored in the persistent volume claim. Backups of that data should be performed periodically. The state information can become stale very fast as users work is submitted and finished. Some job state data will be lost for jobs submitted between the last backup and current time.

NOTE: A reliable filesystem is critical to minimize job state loss.

Dynamic provisioning of the persistent volume is discouraged because of the difficulty in locating the correct resource to backup. Pre-creating a persistent volume claim, or labeling a persistent volume, for the deployment to use provides the easiest way to locates the storage to backup.

Restoring from a backup will require restarting the manager processes. Use the procedure below to reconfigure the entire cluster after restoring files.

1. Locate the master pod by looking for ***ibm-spectrum-computing-prod-master*** in the list of pods e.g.

   ```
   $ kubectl get pods |grep ibm-spectrum-computing-prod-master

   lsf-ibm-spectrum-computing-prod-master-56b55d6dc8-84gcj   1/1     Running
   0          3d19h
   ```

2. Connect to the management pod e.g.

```
$ kubectl exec -ti lsf-ibm-spectrum-computing-prod-master-56b55d6dc8-
84gcj bash
```

3. Run the command to re-read the configuration files

```
LSF POD [root@lsfmaster /]# lsadmin reconfig
LSF POD [root@lsfmaster /]# badmin mbdrestart
```

4. Wait for a minute and try some commands to see if the cluster is functioning okay e.g.

```
LSF POD [root@lsfmaster /]# lsid
IBM Spectrum LSF Standard Edition 10.1.0.0, Dec 04 2019
Copyright International Business Machines Corp. 1992, 2016.
US Government Users Restricted Rights - Use, duplication or disclosure
restricted by GSA ADP Schedule Contract with IBM Corp.

My cluster name is myCluster
My master name is lsfmaster
```

Command should report the software versions and manager hostname.

```
# bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
worker-10-1-1-10       ok          -   1000      0      0      0      0      0
worker-10-1-1-11       ok          -   1000      0      0      0      0      0
worker-10-1-1-12       ok          -   1000      0      0      0      0      0
lsfmaster          closed          -      0      0      0      0      0      0
```

Host status of the workers should be **ok**. Next check the queue status:

```
LSF POD [root@lsfmaster /]# bqueues
QUEUE_NAME    PRIO STATUS       MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
priority       43  Open:Active   -    -    -    -     0     0     0     0
normal         30  Open:Active   -    -    -    -     0     0     0     0
idle           20  Open:Active   -    -    -    -     0     0     0     0
night           1  Open:Inact     -    -    -    -     0     0     0     0
```

Queues should be **open**.

Appendix A

This is a sample parallel job, with two types of tasks, with one of those tasks being a parallel task.

```
apiVersion: ibm.com/v1alpha1
kind: ParallelJob
metadata:
  name: pjob-1
  namespace: default
  annotations:
    lsf.ibm.com/queue: "priority"
    # For Elastic jobs
    # deletePodOnFlexDown: "y"
  labels:
    test: experiment10
spec:
  name: pjob-1 #same with metadata/name
  description: This is a parallel job with 2 tasks.
  headerTask: group0
  priority: 100
  resizable: false
  schedulerName: lsf
  taskGroups:
  - metadata:
      name: group0
    spec:
      replica: 1
      template:
        spec:
          nodeSelector:
            beta.kubernetes.io/arch: amd64
            beta.kubernetes.io/os: linux
          containers:
          - args:
            command: ["sleep", "90"]
            image: ubuntu
            name: task1
            resources:
              limits:
                cpu: 1
                memory: 400Mi
              requests:
                cpu: 1
                memory: 400Mi
          restartPolicy: Never
  - metadata:
      name: group1
      # annotations:
      #   lsf.ibm.com/gpu: "num=1:j_exclusive=yes"
    spec:
      replica: 2
      template:
        spec:
          nodeSelector:
            beta.kubernetes.io/arch: amd64
            beta.kubernetes.io/os: linux
          containers:
          - args:
            command: ["sleep", "90"]
            image: ubuntu
            name: task1
            # For GPU Jobs add the following
            # env:
            # - name: NVIDIA_DRIVER_CAPABILITIES
            #   value: "compute,utility"
  # securityContext:
  #   allowPrivilegeEscalation: false
  #   capabilities:
  #     drop: ["ALL"]
  #   seLinuxOptions:
  #     type: nvidia_container_t
  resources:
    limits:
      cpu: 1
      memory: 200Mi
    requests:
      cpu: 1
      memory: 200Mi
restartPolicy: Never
```

Appendix B

Test Case Execution

This section provides detailed steps to evaluate the test cases above. This can be used to replicate the test and confirm the results. It is also instructional in demonstrating how IBM Spectrum Computing TP is configured and run.

Test Case: 1 – User Pods and Applications

Purpose:

This test will examine how users access the various policies provided by the scheduler. This test case is primarily instructional.

Overview:

The job queues provide a convenient way of applying different policies to the jobs that are run. This test will look at the out of box queues and the policies that are applied to them. We will test some of those policies in later test cases.

Pre-conditions:

- The IBM Spectrum Computing TP is deployed.
- The `kubectl` CLI is installed

Test Steps: Use the following steps to replicate the test.

Step	Action	Detailed Instructions

1	Display the queue configuration file	The queue configuration file is stored as a configMap.
		1. Get a list of the configMaps by running:
		`$ kubectl get cm -n ibm-lsf-tp`
		2. The queue configMap name is of the form:
		`{Something}-ibm-spectrum-computing-prod-queues`
		3. Display the contents of the config map by running:
		`$ kubectl describe cm -n ibm-lsf-tp {Something}-ibm-spectrum-computing-prod-queues`

2	Discover the names and priorities of the out of box queues	Look through the output and find the QUEUE_NAME, and PRIORITY lines. You will find: • `normal` priority 30 • `idle` priority 20 • `priority` priority 43 • `night` priority 15

3	Create a new queue	1. Edit the contents of the config map by running:
		`$ kubectl edit cm -n ibm-lsf-tp {Something}-ibm-spectrum-computing-prod-queues`
		2. After the “priority” queue add the following:
		`Begin Queue`

		```
		    QUEUE_NAME   = testqueue
		    PRIORITY     = 10
		    DESCRIPTION  = Very low priority test queue
		    End Queue
		```
		 NOTE: The spacing is significant.

4	Verify the new queue configuration by connecting to the master pod and inspecting the configuration	1. Locate the master pod by looking for "ibm-spectrum-computing-prod-master" in the list of pods e.g.
		`$ kubectl get pods -n ibm-lsf-tp |grep ibm-spectrum-computing-prod-master`
		
		`lsf-ibm-spectrum-computing-prod-master-56b55d6dc8-84gcj   1/1     Running   0   3d19h`
		2. Connect to the master pod with:
		`$ kubectl exec -ti -n ibm-lsf-tp lsf-ibm-spectrum-computing-prod-master-56b55d6dc8-84gcj bash`
		3. Change to the configuration directory:
		`LSF POD [root@lsfmaster /]# cd /opt/ibm/lsfsuite/lsf/conf/lsbatch/myCluster/configdir/`
		4. Check that the configMap file. It's called lsb.queues It should contain your edits. If not wait a minute to the configMap to update.
		5. Check that there are no errors in the file by running:
		`LSF POD [root@lsfmaster /]# badmin chkconfig`
		6. There should be no errors. If any are found, they need to be corrected.

Expected Results:

- The four out of box queues have been identified, along with there priorities

Additional Resources:

The Queue configuration has many more options not covered in the out of box defaults. Additional information can be found here:
https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_config_ref/lsb.queues.5.html

Test Case: 2 Pob Priority & Pre-emption

Purpose:

Demonstrate how IBM Spectrum Computing Accelerator handles workload that have different priorities.

Overview:

Not every job is important in a business sense. A payroll job is more important than a developer's test job. Higher priority jobs should be run before lower priority jobs. This test case will fill the cluster with low priority workloads, then once all the resources are busy, high priority workload will be introduced. In one case the lower priority jobs will not be preempted. In a second case the lower priority jobs will be preempted.

Pre-conditions:

- The IBM Spectrum Computing Accelerator cloud pack is deployed.
- The `kubectl` CLI is installed

Test Steps: Use the following steps to replicate the test

Step	Action	Detailed Instructions

1	Monitor the job queues	1. Locate the master pod by looking for "ibm-spectrum-computing-prod-master" in the list of pods e.g.
		`$ kubectl get pods -n ibm-lsf-tp |grep ibm-spectrum-computing-prod-master`
		
		`lsf-ibm-spectrum-computing-prod-master-56b55d6dc8-84gcj   1/1     Running   0          3d19h`
		2. Connect to the master pod with:
		`$ kubectl exec -ti -n ibm-lsf-tp lsf-ibm-spectrum-computing-prod-master-56b55d6dc8-84gcj bash`
		3. Watch the queue state by running:
		`$ watch -d bqueues`
		This will show the queues and state. NJOBS are the number of jobs in the queue. PEND are the pending jobs waiting for execution, and RUN are the running jobs.

2	Start the job submission script	Run the test script to start a large number of jobs in the normal queue. Run: `$ priority-test-non-preempt.sh` It will ask for the number of jobs to start. Enter a number approximately the number of CPU cores in the cluster.
3	Check what jobs are running	
4	Repeat the test	

Expected Results:

- When there is contention for resources the high priority jobs get scheduled before the lower priority jobs.
- When there is contention for resources the higher priority jobs can pre-empt the lower priority jobs.

Test Case: 3 - Run Windows

Purpose:

Show that run windows can be used to manage when jobs can have resources to run

Overview:

The cluster resources need to be managed and assigned to the jobs at the right time, to ensure critical business processes have the resources needed for timely completion. This test will look at how time windows can be applied to queues so that resources can be provided to jobs at certain times of the day, and certain days of the week.

NOTE: The pods typically use UTC time.

Pre-conditions:

- The IBM Spectrum Computing TP is deployed.
- The `kubectl` CLI is installed

Test Steps: Use the following steps to replicate the test

Step	Action	Detailed Instructions

1	Check the time in the scheduler pod	1. Locate the master pod by looking for "ibm-spectrum-computing-prod-master" in the list of pods e.g.
		`$ kubectl get pods -n ibm-lsf-tp |grep ibm-spectrum-computing-prod-master`
		
		`lsf-ibm-spectrum-computing-prod-master-56b55d6dc8-84gcj   1/1     Running   0          3d19h`
		2. Get the master pod time with:
		`$ kubectl exec -ti -n ibm-lsf-tp lsf-ibm-spectrum-computing-prod-master-56b55d6dc8-84gcj date`
		3. Get the local time and compute the difference. You will need to keep this difference in mind when modifying the queue.

2	Check the RUN_WINDOW of the night queue	The night queue has a default RUN_WINDOW that allows jobs to run on the weekend and week nights. Jobs can run starting Friday at 19:00 thru to Monday at 8:30, and weekdays from 20:00 thru to 8:30. It is defined as: `RUN_WINDOW = 5:19:00-1:8:30 20:00-8:30` Time windows can be specified as up to 3 fields -- [day:]hour[:minute]. Note that day=[0-6]: 0 is Sunday, 1 is Monday and 6 is Saturday. If only

		one field exists, it is assumed to be hour; if two fields exist, it is assumed to be hour[:minute]. Multiple windows can be specified. The default is any time.

3	Optional: If the RUN_WINDOW of the night queue is already open, based on the master pods UTC time, change the RUN_WINDOW so it is closed	1. Edit the contents of the config map by running:
		`$ kubectl edit cm -n ibm-lsf-tp {Something}-ibm-spectrum-computing-prod-queues`
		2. After the “night” queue and set the RUN_WINDOW so the queue should be closed. For example if there were a 3 hour time difference:
		`Begin Queue`
		`QUEUE_NAME    = night`
		`PRIORITY      = 15`
		`RUN_WINDOW    = 5:22:00-1:11:30 23:00-11:30`
		`End Queue`
		NOTE: The spacing is significant.

4	Submit a job to the night queue and see that it is pending	

Expected Results:

- Pods submitted to the night queue remain in a pending state until the run window opens. Once opened the pods start to run.

Getting Additional Test Cases

Sample scripts for evaluating other test cases are available on Github from here:

https://github.com/IBMSpectrumComputing/lsf-kubernetes/tree/master/examples

They currently include the following examples:

- **Pod_Dependencies** - This test provides an example of the workflow feature that is provided by this integration. The workflow feature allows you to define workflows to perform complex multi-pod operations.

- **Pod_Priority_and_Preemption** - This directory provides some tests that will help you explore the pod priority capabilities, which allow you to run pods with higher priorities before those with lower priorities. They also show how high priority pods can kill lower priority pods to free resources.

- **Resource_Sharing** - Resource contention happens, what will you do about it? This test looks at one of the sharing policies that this integration provides. It demonstrates how to align business priorities with the resources that are available so that the more critical pods have the resources they deserve.

- **Run_Limits** - Sometime a broken application may hold resources, when they should be freed. This test provides an example of run limits, which can be used to free resources from misbehaving pods.

- **Run_Windows** - Not every pod should run immediately. Sometimes it is better to queue pods till the night, or weekend so that resources are available for more business significant work. This test shows how to construct, and test run windows.

Copyright and trademark information