IBM Spectrum LSF 10.1

Resource Ceonnector

© Copyright IBM Corp. 2024.
US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Tables of Contents

IBM Spectrum LSF resource connector overview

The resource connector for IBM® Spectrum LSF feature (previously referred to as *host factory*) enables LSF clusters to borrow resources from supported resource providers.

LSF resource connector plug-ins support the following resource providers:

- IBM Cloud Virtual Servers for VPC Gen 2 (IBM Cloud Gen 2), configured with the ibmcloudgen2_config.json and ibmcloudgen2_templates.json files. The IBM Cloud provider requires IBM Spectrum LSF Fix Pack 11.
- Amazon Web Services (AWS), which is configured with the awsprov_config.json and awsprov_templates.json files. The AWS provider requires IBM Spectrum LSF Fix Pack 2.
- Microsoft Azure, which is configured with the azureprov_config.json and azureprov_templates.json files. The Microsoft Azure provider requires IBM Spectrum LSF Fix Pack 3.
- Google Compute Cloud, configured with the googleprov_config.json and googleprov_templates.json files. The Google Compute provider requires IBM Spectrum LSF Fix Pack 4.
- OpenStack, which is configured with the osprov_config.json and osprov_templates.json files. The OpenStack provider requires IBM Spectrum LSF Fix Pack 1.

LSF clusters can borrow hosts from a resource provider to satisfy pending workload. The borrowed resources join the LSF cluster as hosts. When the resources become idle, LSF resource connector returns them to the resource provider.

The resource connector generates requests for extra hosts from the resource provider and dispatches jobs to dynamic hosts that join the LSF cluster. When the resource provider reclaims the hosts, the resource connector requeues the jobs that are running on the LSF hosts, shuts down LSF daemons, and releases the hosts to the provider.

Requirements for configuring the resource connector

The following are requirements for configuring the IBM Spectrum LSF resource connector.

- You must have root access to the LSF management host.
- Both the LSF management host and the compute nodes (provider instances) must be reachable from each other.
- You must be able to restart the LSF cluster.
- You must be familiar with the provider's concepts and be able to perform administrative tasks.
- The virtual network to be used by the provider's virtual instances must be configured so that they can communicate with the LSF hosts on-premise.
- You must configure the provider instances to map users to the LSF cluster submission users. For example, add the submission users to the provider instance or synchronize the users on the launched provider instances.
- Decide how you want LSF to authenticate to the provider to access their services.

Java requirements for LSF management host

The following are Java requirements for the IBM Spectrum LSF management host before configuring the IBM Spectrum LSF resource connector.

- Java Runtime Environment (JRE) version 8

How LSF borrows hosts from a resource provider

The following workflow summarizes how your job uses resources that are borrowed from a resource provider:

1. A user submits a job to LSF as normal. The job generates demand, but the cluster doesn't have enough resources to service it, so it must borrow hosts from an external provider.
2. The **mbatchd** daemon checks if hosts are already allocated that match the demand, and LSF calculates how many of each template type it requires to run the job. LSF sends this demand to the resource connector **ebrokerd** daemon.
   The administrator configures templates representing LSF hosts. Each resource provider has its own template file that defines the mapping between LSF resource demand requests and hosts that the provider allocates to LSF. Each template in the file represents a set of hosts that share some attributes, such as the number of CPUs, the amount of available memory, the installed software stack, and operating system image.
3. Based on the demand from the submitted job, LSF resource connector makes an allocation request to the resource provider.
   For example, if the resource provider is IBM Cloud Gen 2, resource connector makes an allocation request to IBM Cloud Gen 2 as the `LSF_Consumer`.
4. For IBM Cloud Gen 2 resources, if enough resources are available in the `rg_shared` resource group, the allocation request succeeds.
5. The **ebrokerd** daemon monitors the status of the request in the resource provider until it detects that the request succeeds, starts LSF daemons on the allocated hosts, and notifies LSF that the hosts join the cluster and are ready to use.
6. When the host joins the cluster, the job is dispatched to the host.
7. When there is no more demand for borrowed resources, LSF lets resource connector know and the **ebrokerd** daemon returns the resources to the provider.
8. Some resource providers (for example, IBM Cloud Gen 2) can also be configured to reclaim borrowed resources from LSF when they require the resources to satisfy their workload demand.

Example of borrowing hosts from IBM Cloud Gen 2

In the following example, the resource provider is IBM Cloud Gen 2:

1. ```
   bhosts -a
   HOST_NAME          STATUS      JL/U    MAX  NJOBS  RUN  SSUSP  USUSP  RSV
   lsfmanagement      ok          -       1    1      0    0      0      0
   ```
2. IBM Cloud Gen 2 has a host `ibmcloud01` with `ncpus=1`.
3. Resource connector is configured to connect to the IBM Cloud Gen 2 cluster.
4. A template is created that provides a numeric attribute `ncpus` with range [1:1].
5. The **bsub** command submits a job that requires a single slot.
6. Eventually host `ibmcloud01` joins the cluster, and the new job runs.

```
bhosts -a
HOST_NAME          STATUS    JL/U    MAX  NJOBS   RUN  SSUSP  USUSP  RSV
lsfmangemnt        ok        -       1    1       1    0      0      0
ibmcloud01         ok        -       1    1       1    0      0      0
```

View the status of provisioned hosts
```

Use the **bhosts -rc** or the **bhosts -rconly** command to see information about resources that are provisioned by LSF resource connector.

The -rc and -rconly options make use of the third-party **mosquitto** message queue application. LSF resource connector publishes additional provider host information to displayed by these **bhosts** options. The **mosquitto** binary file is included as part of the LSF distribution.

To use the -rc and -rconly options, LSF resource connector must be enabled with the LSB_RC_EXTERNAL_HOST_FLAG parameter in the lsf.conf file.

If you use the MQTT message broker that is distributed withLSF, you must configure the LSF_MQ_BROKER_HOSTS and MQTT_BROKER_HOST parameters in the lsf.conf file. The LSF_MQ_BROKER_HOSTS and MQTT_BROKER_HOST parameters must specify the same host name. The LSF_MQ_BROKER_HOSTS parameter enables LIM to start the **mosquitto** daemon.

If you use an existing MQTT message broker, you must configure the MQTT_BROKER_HOST parameter. You can optionally specify an MQTT broker port with the MQTT_BROKER_PORT parameter.

Use the **ps** command to check that the MQTT message broker daemon (**mosquitto**) is installed and running: `ps -ef | grep mosquitto`.

Configure the EBROKERD_HOST_CLEAN_DELAY to specify a delay, in minutes, after which the **ebrokerd** daemon removes information about relinquished or reclaimed hosts. This parameter allows the **bhosts** command to get LSF resource connector provider host information for some time after they are deprovisioned.

Three more columns are shown in the **bhosts** command host list:

`RC_STATUS`
: LSF resource connector status.

  `Preprovision_Started`
  : Resource connector started the preprovisioning script for the new host.

  `Preprovision_Failed`
  : The preprovisioning script returned an error.

  `Allocated`
  : The host is ready to join the LSF cluster.

  `Reclaim_Received`
  : A host reclaim request was received from the provider (for example, for an AWS spot instance).

  `RelinquishReq_Sent`
  : LSF started to relinquish the host.

  `Relinquished`
  : LSF finished relinquishing the host.

  `Deallocated_Sent`
  : LSF sent a return request to the provider.

  `Postprovision_Started`
  : LSF started the postprovisioning script after the host was returned.

  `Done`
  : The host life cyle is complete.

`PROV_STATUS`
: Provider status. This status depends the provider. For example, AWS has pending, running, shutting down, terminated, and others. Check documentation for the provider to understand the status that is displayed.

`UPDATED_AT`
: Time stamp of the latest status change.

For hosts provisioned by resource connector, these columns show appropriate status values and a time stamp. A dash (–) is displayed in these columns for other hosts in the cluster.

For example,

```
bhosts -rc
HOST_NAME               STATUS  JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
RC_STATUS       PROV_STATUS    UPDATED_AT
icgen2host-10-240-0-37 ok       -      1      0      0      0      0      0
Allocated       running        2017-04-07T12:28:46CDT
lsf1.aws.               closed   -      1      0      0      0      0      0    -
-               -
```

The -l option shows more detailed information about provisioned hosts:

```
bhosts -rc -l
HOST  icgen2host-10-240-0-37
STATUS  CPUF  JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV RC_STATUS
PROV_STATUS    UPDATED_AT             DISPATCH_WINDOW
ok      60.00 -       1    0        0    0      0        0   Allocated
running        2017-04-07T12:28:46CDT      -

 CURRENT LOAD USED FOR SCHEDULING:
                r15s   r1m  r15m    ut    pg    io   ls    it   tmp   swp   mem
slots
 Total           1.0   0.0   0.0    1%   0.0    33    0     3 5504M    0M  385M
1
 Reserved        0.0   0.0   0.0    0%   0.0     0    0     0    0M    0M    0M
-
```

The -rconly option shows the status of all hosts that are provisioned by LSF resource connector, no matter if they are in the cluster or not.

The following information is shown:

PUB_DNS_NAME and PUB_IP_ADDRESS
: Public DNS name and IP address of the host.

PRIV_DNS_NAME and PRIV_IP_ADDRESS
: Private DNS name and IP address of the host.

RC_STATUS
: LSF resource connector status.

PROV_STATUS
: Resource provider status.

TAG
: The `RC_ACCOUNT` value that is defined in the lsb.queues or lsb.applications files.

UPDATED_AT
: Time stamp of the latest status change.

For example,

```
bhosts -rconly
PROVIDER : aws
  TEMPLATE : aws-vm-1
    PUB_DNS_NAME           PUB_IP_ADDRESS  PRIV_DNS_NAME      PRIV_IP_ADDRESS
RC_STATUS              PROV_STATUS           TAG            UPDATED_AT
    icgen2host-10-240-0-37 52.43.171.109   ip-192-168-0-85.us
192.168.0.85    Done                    terminated            default        2017-
05-31T14:30:47CDT
    icgen2host-10-240-0-47 35.160.157.112  ip-192-168-0-69.us
```

```
192.168.0.69    Allocated               running                 default         2017-
05-31T14:32:00CDT
```

Limitations

Do not create advanced reservations on AWS instances because the reservations might be terminated after idle time. If advanced reservations are created on instances, they remain active if the instances are destroyed. However, jobs are not able to run on the instance since the LSF daemons are shut down on terminated instances and the jobs become unavailable.

Hosts can be returned to their resource provider at any time by idle time or time-to-live policy, EGO reclaim, or AWS reclaim. The hosts might be closed or unavailable when the advanced reservation starts.

Hosts can be returned to their resource provider at any time by idle time or time-to-live policy, or AWS reclaim. The hosts might be closed or unavailable when the advanced reservation starts.

It also is possible for resource connector to over-demand for its workload if a borrowed host joins the cluster but it is not immediately usable by scheduler.

If borrowed hosts cannot resolve host names, then commands like **lsrcp** do not work when used to copy the files from one instance to another.

The HOSTS parameter in the lsb.queues file and the job-level -m option do not apply to borrowed hosts managed through the resource connector.

Administrators must use the RC_HOSTS parameter in the queue to specify the external resources that resource connector can borrow resources from. A queue can borrow hosts only from the resource that the RC_HOSTS parameter defines. For example, if the queue defines only the AWS resources (`RC_HOSTS=awshost`), it cannot borrow EGO or OpenStack resources.

The RC_ACCOUNT parameter that is defined in an application profile in the lsb.applications file is not displayed in the **bapp -l** command. The **bqueues -l** command shows the value of the RC_ACCOUNT and RC_HOSTS parameters that are defined in queues. The **bhosts -rconly** option displays the RC_ACCOUNT value under the `TAG` column.

If you configure the LSF_MQ_BROKER_HOSTS parameter to enable the **bhosts -rc** and **bhosts -rconly** command options to display resource provider host information, the -rc and -rconly options do not support host groups or CUs.

Configuring resource providers for LSF resource connector

Modify resource connector configuration files after installation to enable resource providers.

- **Initial configuration**
  Set the basic configuration files for the LSF resource connector.
- **Configure multiple resource providers**
  You can configure multiple resource providers in one LSF cluster.
- **Configuring different templates to create instances**
  If you configure multiple templates in a provider's template file, you can run different types of jobs on different template instances.

- **Assigning exclusive resources to a template**
  When you assign exclusive resources to a template, LSF recognizes the exclusive resource definition for demand calculation. You must set up the exclusive resource when launching the instance.
- **Configuring Amazon Web Services for LSF resource connector**
  Follow these steps to configure Amazon Web Services (AWS) to create instances for LSF resource connector to make allocation requests on behalf of LSF.

Initial configuration

Set the basic configuration files for the LSF resource connector.

About this task

If you are using LSF 10.1, after installing the LSF cluster, use the configuration files that are included with the LSF resource connector to set up a basic configuration.

Procedure

1. Create an initial host provider configuration file from the example hostProviders.json file.
   Copy the *LSF_TOP*/10.1.0/resource_connector/example_hostProviders.json file to *LSF_TOP*/conf/conf/resource_connector/hostProviders.json and customize this file for your cluster.

   For more details on the hostProviders.json file, refer to hostProviders.json.

2. Create initial configuration files for the required resource providers from the example resource connector configuration files.
   Copy all the configuration files for your resource providers from *LSF_TOP*/10.1.0/resource_connector/*provider_name*/conf file to *LSF_TOP*/conf/resource_connector/*provider_name*/conf and customize these files for your cluster.

   For more details on the configuration files for your resource providers, refer to Configuring different templates to create instances and LSF resource connector configuration reference.

   Important: Use the default scriptPath parameter value, which refers to the LSF_TOP/10.1.0/resource_connector/*provider_name* file path. If you put the script files in the LSF_TOP/conf/resource_connector directory or use a custom value for scriptPath, LSF resource connector patch files do not automatically update the script and library files. This is because the LSF **patchinstall** command only updates files under the LSF_TOP/10.1.0 directory.
3. Create a user_data.sh script file that runs when each new instance launches.
   Save the user_data.sh script file in the LSF_TOP/10.1.0/resource_connector/*provider_name*/scripts directory. You can create a script file based on the example script file in the following file path: LSF_TOP/10.1.0/resource_connector/*provider_name*/scripts/example_user_data.sh

4. Change the ownership of all new files and directories in the LSF_TOP/conf directory to the cluster administrator.

Configure multiple resource providers

You can configure multiple resource providers in one LSF cluster.

Each provider has its own set of configurable parameters that are defined in the hostProviders.json file (see hostProviders.json for more information). A log file for each provider (<*provider_name*>.log.<*host_name*>) is located in the LOGDIR and a persistence file is located in LSF_SHAREDIR/<*cluster_name*>/resource_connector/aws-db.json.

All the providers must have the same LSF administrator account.

Each host provider must have a unique name. If two different host providers use the same name, LSF logs a warning and ignores one of the entries.

The LSF administrator must have access to the directories specified by the `confPath` and `scriptPath` attributes in the hostProviders.json file. For example:

```
{
    "providers":[
        {
            "name": "aws1",
            "type": "awsProv",
            "confPath": "resource_connector/aws1",
            "scriptPath": "resource_connector/aws1",
            "scriptOptions": "-Dhttps.proxyHost=10.115.206.146 -
Dhttps.proxyPort=8888",
            "billingPeriod":  "60",
            "preProvPath":  "/usr/share/lsf/scripts/pre_provision_aws1.sh",
            "postProvPath": "/usr/share/lsf/scripts/post_provision_aws1.sh",
            "provTimeOut" : 10
        },
        {
            "name": "aws2",
            "type": "awsProv",
            "confPath": "resource_connector/aws2",
            "scriptPath": "resource_connector/aws2",
            "billingPeriod":  "30",
            "preProvPath":  "/usr/share/lsf/scripts/pre_provision_aws2.sh",
            "postProvPath": "/usr/share/lsf/scripts/post_provision_aws2.sh",
            "provTimeOut" : 20
        }
    ]
}
```

Both full and relative paths are supported for configuration and script files. The default assumes a path relative *LSF_TOP*/conf/resource_connector for configuration files and *LSF_TOP*/*LSF_VERSION*/resource_connector/ for scripts.

Changes to the hostProviders.json configuration file requires a reconfiguration of LSF by running the command **badmin mbdrestart** on the LSF management host.

Related reference

- hostProviders.json

Configuring different templates to create instances

If you configure multiple templates in a provider's template file, you can run different types of jobs on different template instances.

About this task

LSF resource connector can map resources to attributes in the cloud provider template. You can define one String or Numeric resource, or a series of Boolean resources, and configure different values in attributes for different templates. For example, you can define a static String resource such as vm_type, and map different values for different templates..

Procedure

1. Define the resource in the lsf.shared file.
   To define a String resource named vm_type:

```
Begin Resource
RESOURCENAME  TYPE    INTERVAL INCREASING  DESCRIPTION          # Keywords
...
  vm_type   String   ()       ()       (vm types for different templates)
...
End Resource
```

2. In the cloud provider template file, configure the different values of the resource attribute for the different templates, then configure the corresponding key-value pair in userData.
   Tip:
   - For each applicable template, add the resource name to attributes and the corresponding key-value pair to userData.
   - The sum of the maxNumber value for all templates should not be greater than 25120; otherwise, any hosts above this 25120 limit will be ignored.

   For example, for AWS, edit the *LSF_TOP*/conf/resource_connector/aws/conf/awsprov_templates.json file. To associate `vm_type=micro` with the cloud-VM-1 template and to associate `vm_type=small` with the cloud-VM-2 template, add the vm_type resource to attributes and the corresponding value for the vm_type key in userData:

```
{
    "templates": [
       {
            "templateId": "cloud-VM-1",
            "maxNumber": 5,
            "attributes": {
                "type": ["String", "X86_64"],
                "ncores": ["Numeric", "1"],
                "ncpus": ["Numeric", "1"],
                "nthreads": ["Numeric", "2"],
                "mem": ["Numeric", "455001"],
                "vm_type": ["String", "micro"],
                "awshost": ["Boolean", "1"],
            },
            ...
            "userData": "vm_type=micro"
        },
        {
            "templateId": "cloud-VM-2",
            "maxNumber": 5,
            "attributes": {
                "type": ["String", "X86_64"],
                "ncores": ["Numeric", "1"],
```

```
                "ncpus": ["Numeric", "1"],
                "nthreads": ["Numeric", "1"],
                "mem": ["Numeric", "512"],
                "vm_type": ["String", "small"],
                "awshost": ["Boolean", "1"]
            },
            ...
            "userData": "vm_type=small"
        }
    ]
}
```

3. Edit the user_data.sh script file for the cloud provider to modify the value of the LSF_LOCAL_RESOURCES parameter in the lsf.conf file for new instances.
   The user_data.sh script runs when each new instance launches.

   For example, for AWS, modify the *LSF_TOP*/10.1/resource_connector/aws/scripts/user_data.sh file. Add the following lines to enable the user_data.sh script to modify the value of the LSF_LOCAL_RESOURCES parameter in the lsf.conf file on the newly-created instance.

```
if [ -n "$vm_type" ]; then
  sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap
$vm_type*vm_type]\"/" $LSF_CONF_FILE
fi
```

4. To select different templates in LSF, use the resource key-value pair in the selection string when specifying the resource requirements for your job.
   - The following command triggers LSF resource connector to create an instance from the cloud-VM-1 template, and to run the job on this instance:

     ```
     bsub -R "select[vm_type==micro]" myjob
     ```

   - The following command triggers LSF resource connector to create an instance from the cloud-VM-2 template, and to run the job on this instance:

     ```
     bsub -R "select[vm_type==small]" myjob
     ```

What to do next

- The AWS and Google cloud providers use a local user_data.sh script on the LSF management host in *LSF_TOP*/10.1/resource_connector/*provider_name*/scripts/user_data.sh. Some other cloud providers, such as Azure, CycleCloud, and IBM Cloud use the user data script's URL in the template configuration. Modify the URL to the user_data.sh file to let it take effect.
- Instead of the String resource, you can also use the Numeric resource or a series of Boolean resources for different templates. For Boolean resources, you must modify the user_data.sh file to add `[resource <bool_resource_name>]` into the LSF_LOCAL_RESOURCES parameter.
  For example:

```
if [ -n ${bool_resource_A} ]; then
 sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resource bool_resource_A]\"/"
$LSF_CONF_FILE
fi

if [ -n ${bool_resource_B} ]; then
 sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resource bool_resource_B]\"/"
$LSF_CONF_FILE
fi
```

- You can also use the LSF-defined String resource templateID to select a different template. To select a different template, configure a different value for the templateID attribute in different templates, but you do not have to configure its key-value pair in the userData attribute. Each cloud provider creates an internal environment template_id for different templates. Copy the following lines (which are from the example_user_data.sh file) to your user_data.sh file:

```
if [ -n "${template_id}" ]; then
 sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap
${template_id}*templateID]\"/" $LSF_CONF_FILE
fi
```

- You cannot use the LSF-defined resource rc_account to select different templates.

Assigning exclusive resources to a template

When you assign exclusive resources to a template, LSF recognizes the exclusive resource definition for demand calculation. You must set up the exclusive resource when launching the instance.

An exclusive resource is a special resource that is assignable to a host. A host with an exclusive resource does not receive jobs unless that job explicitly requests the resource.

For example, you might want to run test jobs only on the cheapest instance type configured for your resource provider. You want to be able to select a template with that vmType only when you want to run on it. Unless specifically requested, this template is not chosen by the scheduler.

LSF resource connector supports an exclusive Boolean resource (for example `instance_store`) that is defined in the attribute section of a template. Resource connector recognizes the exclusive resource definition when it creates hosts based on that template. The logical not (`!`) operator is used to create hosts do not use the exclusive resource (`!instance_store`). For example,

```
{
"templates": [
{
"templateId": "Template-VM-1",
"attributes": {
"type": ["String", "X86_64"],
"ncores": ["Numeric", "1"],
"ncpus": ["Numeric", "1"],
"mem": ["Numeric", "1024"],
"awshost1": ["Boolean", "1"],
"!instance_store": ["Boolean", "1"]
},
...
"userData": "zone=us_west_2a;instance_store=!instance_store"
}
]
}
```

Add the exclusive resource to the user_data.sh file to set up the exclusive resource in the LSF_LOCAL_RESOURCES parameter when the instance is launched, and refer to it in the userData attribute in the template. For example,

```
cat user_data.sh
#
# Support rc_account resource to enable RC_ACCOUNT policy
# Add additional local resources if needed
#
```

```
if [ -n "${rc_account}" ]; then
sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap
${rc_account}*rc_account]\"/" $LSF_CONF_FILE
echo "update LSF_LOCAL_RESOURCES lsf.conf successfully, add [resourcemap
${rc_account}*rc_account]" >> $logfile
fi

if [ -n "${instance_store}" ]; then
sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resource ${instance_store}]\"/"
$LSF_CONF_FILE
echo "Updated LSF_LOCAL_RESOURCES in $LSF_CONF_FILE successfully, add [resource
${instance_store}]" >> $logfile
else
echo "instance_store does not exist in the environment variable" >> $logfile
fi
```

Configuring Amazon Web Services for LSF resource connector

Follow these steps to configure Amazon Web Services (AWS) to create instances for LSF resource connector to make allocation requests on behalf of LSF.

- **Preparing to configure LSF resource connector for AWS**
  The purpose of the Configuring IBM Spectrum LSF resource connector guide is to describe how to configure IBM® Spectrum LSF resource connector to cloud-burst to a cloud provider and have LSF automatically borrow hosts in the cloud to grow the cluster when demand is high. Resource connector will release and terminate the borrowed hosts when the demand is low and hosts are idle.
- **Building a cloud image**
  To create an Amazon Machine Image (AMI) for an LSF cloud compute host, the cluster administrator must first manually launch an instance and install LSF on an EC2 instance. An AMI is then created from this instance. Subsequent cloud instances can be dynamically launched by LSF using this AMI.
- **Enabling LSF resource connector for Amazon Web Services (AWS)**
  Enabling LSF resource connector for AWS is done through first executing aws_enable.sh script then following with a few manual steps. The script needs to be executed on the LSF management host where the launch request of the AWS EC2 instance takes place.
- **Configuring bursting behavior**
  LSF has two places to configure the overall decision on how to configure bursting behavior because only the scheduler (the **mbsched** daemon) has information about jobs and this information is not needed by the resource connector policy module. A decision is first made to determine if demand is generated or not at the scheduler. This is controlled at the queue level using the queue threshold.
- **Assigning exclusive resources to a template**
  When you assign exclusive resources to a template, LSF recognizes the exclusive resource definition for demand calculation. You must set up the exclusive resource when launching the instance.
- **Configuring AWS access with federated accounts**
  Resource connector supports *federated accounts* for LSF resource connector as an alternative to requiring permanent AWS IAM account credentials. Federated users are external identities that are granted temporary credentials with secure access to resources in AWS without requiring creation of IAM users. Users are authenticated outside of AWS (for example, through Windows Active Directory). **All AWS resource connector features are supported when you use federated accounts instead of IAM credentials.**

- **Configure launch templates for AWS**
  A launch template is an Amazon Elastic Compute Cloud (EC2) feature that reduces the number of steps that are required to create an AWS instance by capturing all launch parameters within one resource.
- **Attach EFA network interfaces to AWS templates**
  The Elastic Fabric Adapter (EFA) is a network interface for Amazon Elastic Compute Cloud (EC2) instances that allows you to run HPC applications with improved levels of communication between several different nodes.
- **Use AWS spot instances**
  Use *spot instances* to bid on spare Amazon EC2 computing capacity. Since spot instances are often available at a discount compared to the pricing of On-Demand instances, you can significantly reduce the cost of running your applications, grow your application's compute capacity and throughput for the same budget, and enable new types of cloud computing applications.
- **Using Amazon EC2 Fleet**
  As of Fix Pack 14, the LSF resource connector for Amazon Web Services (AWS) uses an Amazon EC2 Fleet API to create multiple (that is, a fleet of) instances. EC2 Fleet is an AWS feature that extends the existing spot fleet, which gives you a unique ability to create fleets of EC2 instances composed of a combination of EC2 on-demand, reserved, and spot instances, by using a single API. Follow these steps to configure AWS using Amazon EC2 Fleet to create instances for LSF resource connector to make allocation requests on behalf of LSF.
- **Submitting jobs to launch instances from Amazon Web Services**
  Use the **bsub** command to submit jobs that require instances that are launched from AWS as the resource provider. Use the **bhosts** to monitor borrowed hosts. Use the **bhosts** command to monitor host status.

Preparing to configure LSF resource connector for AWS

The purpose of the Configuring IBM Spectrum LSF resource connector guide is to describe how to configure IBM Spectrum LSF resource connector to cloud-burst to a cloud provider and have LSF automatically borrow hosts in the cloud to grow the cluster when demand is high. Resource connector will release and terminate the borrowed hosts when the demand is low and hosts are idle.

IBM Spectrum LSF resource connector supports multiple cloud providers resource connector currently only supports Linux. Some understanding of those cloud providers is required in order to complete the tasks of the document and it is recommend going over these prerequisites as a starting point.

Amazon Web Services terms

The following terms are used throughout this guide and are specific to AWS features and functionality.

- **AWS EC2 Instance**: Amazon Web Services virtual server in Amazon's elastic compute cloud.
- **AMI**: Amazon Machine Image for launching EC2 instances.
- **ARN**: Amazon Resource Name is a format for uniquely naming AWS resources (for example, `arn:aws:iam::<account number>:instance-profile/LSFRole`
- **AWS IAM**: Amazon Web Services identity and access management for AWS resources.
- **Federated Account**: Federation enables users access to AWS cloud resources through single sign-on to access AWS accounts using credentials from your corporate directory.
- **VPC**: Amazon virtual private cloud allows networking configuration that the EC2 instances will reside in.

Prerequisite reading

- **AWS Management Console - Getting Started**: The console is one of the main access points to configure AWS related settings including creation of the access key, users, roles, AMI and launching the first instance.
  https://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/getting-started.html
- **Amazon Machine Images (AMI)**: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html
- **Getting Started with Amazon EC2 Linux Instances**: The AMI that LSF will use requires the user to build it based on a launched instance.
  https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html
- **AWS IAM Users**: https://docs.aws.amazon.com/IAM/latest/UserGuide/id.html
  OR

  AWS Identity and Federation:
  https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers.html

  and

  https://aws.amazon.com/identity/federation/
- **AWS VPC**: The launched instances from LSF will need to reside in a VPC which allows access to the on-premise hosts in the same LSF cluster.
  http://docs.aws.amazon.com/AmazonVPC/latest/GettingStartedGuide/ExerciseOverview.html

Gathering your requirements

There are some details not covered in this document since they are environment dependent and are varied for each organization. You are expected to obtain this information through your organization either ahead of time or when the step requires the information. For example, this may include:

- How to connect from an on-premise host machine to a launched AWS EC2 instance.
- What type of user authentication is used for AWS access; IAM Credentials or Account Federation
  Note: Both IAM user credentials and federated accounts support all AWS RC features. The user for either option must be assigned the correct permissions. (**See Enabling LSF resource connector for Amazon Web Services (AWS)**)

Required configuration concepts

Below is an overview of the required steps required to cloud-burst. This document is written with the assumption that you are familiar with these concepts.

- Launch AWS EC2 instance from a public or private Linux AMI
- Install LSF as a server host on the AWS EC2 instance
- Build a cloud image by saving a new AMI based on the AWS EC2 instance
- Setting up required policies for the IAM Roles used from IAM users or federated accounts
- Create an AWS Key Pair
- Configure DNS settings to allow the on-premise cluster to communicate with AWS EC2 instances either through using public IP's or DNS network configurations

Building a cloud image

To create an Amazon Machine Image (AMI) for an LSF cloud compute host, the cluster administrator must first manually launch an instance and install LSF on an EC2 instance. An AMI is then created from this instance. Subsequent cloud instances can be dynamically launched by LSF using this AMI.

- **Preparing AWS components**
  Follow these steps to prepare Amazon Web Services (AWS) for LSF Resource Connector.
- **Launching the Amazon Web Services EC2 instance**
  AWS EC2 describes the instance type. Follow these steps to launch the Amazon Web Services (AWS) EC2 instance.
- **Installing an LSF server host on the AWS EC2 instance**
  Follow these steps to install an LSF server host on the Amazon Web Services (AWS) for EC2 instance.

Preparing AWS components

Follow these steps to prepare Amazon Web Services (AWS) for LSF Resource Connector.

Before you begin

LSF should be installed on the local management host already which means the LSF ports are already determined.

You must already have access to an AWS account or ask your AWS Admin to gain access.

Procedure

1. Login to the AWS console as the administrator or a user with privileges to create and modify VPC, subnets, and AMIs.
2. Create a security group for LSF.
   You can use the default security group that is provided by AWS or create your own with customized rules to open all LSF listening ports to the security groups that are used to launch instances. If the default is used, it must also be configured to allow network traffic through the LSF ports. The ports must match the ports in the existing LSF cluster.

   LSF has the following default port number values:
   - **LSF_LIM_PORT=7869** (TCP and UDP)
   - **LSF_RES_PORT=6878** (TCP)
   - **LSB_SBD_PORT=6882** (TCP)

   You can also add management host IP address to accept all traffic from the management host.

   Tip: Remember the security Group ID (for example, sg-1234) as this is required in the awsprov_templates.json file.
   Note: If you allow only the traffic from the default ports, some nioscommands, such as bsub –I for interactive jobs, might not work, since the ports are configured for them dynamically and are different from the default ports. Open additional ports in the firewall or network to allow NIOS to communicate. Use LSF_NIOS_PORT_RANGE to configure the port range which is opened for LSF to use.
3. Create a Virtual Private Cloud (VPC) and subnets.

A *Virtual Private Cloud* (VPC) is a virtual network that is dedicated to an AWS account.

AWS help: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html

A subnet is a range of IP addresses in your VPC. You can launch AWS resources into a subnet that you select. Use a public subnet for resources that must be connected to the internet, and a private subnet for resources that aren't connected to the internet.

If the Auto-Assign Public IP option is checked, public IP or public DNS is available for created instances in this subnet.

In most of the cases, the management host does not run on AWS. AWS server hosts must have either a public IP address or the network must be configured in a way for the management host LIM to connect to the LIMs on the AWS server host. By default, Amazon assigns a private DNS to the instances that are created.

Tip: Remember the subnet ID that will be needed in the awsprov_templates.json file that is used to launch the instance by LSF.

Launching the Amazon Web Services EC2 instance

AWS EC2 describes the instance type. Follow these steps to launch the Amazon Web Services (AWS) EC2 instance.

Procedure

1. Log in to the AWS Console with either the federated account or IAM user created above.
2. Select the security group that you created for LSF and pick the machine attributes, VPC, and subnet information that you created when preparing the AWS components.
3. Continue through the launch screens from AWS to launch the AWS EC2 instance
   Note: AWS help: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html#ec2-launch-instance_linux

Installing an LSF server host on the AWS EC2 instance

Follow these steps to install an LSF server host on the Amazon Web Services (AWS) for EC2 instance.

Before you begin

You must already have launched an AWS EC2 instance.

About this task

AWS uses public-key cryptography to secure the login information for an instance. A Linux instance has no password; you use a key pair to log in to your instance securely. You specify the name of the key pair when you launch your instance, then provide the private key when you log in using SSH.

Note: AWS help: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html
After the instance is created, use **ssh** to log in to the instance with the key file generated above and perform the following tasks to install LSF on the EC2 instance.

Procedure

1. Copy the LSF package to EC2 host.
   - lsf10.1.0.12_linux2.6-glibc2.3-x86_64.tar.Z
   - lsf10.1.0.12_lsfinstall_linux_x86_64.tar.Z
   - license.dat or lsf.entitlement
2. Set up firewall communications on the instance.
   Modify the instance's firewall to open all LSF listening ports. The ports must match those from the existing LSF cluster. The ports are required to be opened so the LSF daemons can communicate from the AWS instance to the on-premise management host. The following are the default port number values:

   - **LSF_LIM_PORT=7869** (TCP and UDP)
   - **LSF_RES_PORT=6878** (TCP)
   - **LSB_SBD_PORT=6882** (TCP)
3. Prepare users for LSF. (Optional)
   For jobs submitted by a user to run on the instance, the instance must have this user prepared or LSF user mapping configured. For more information about user groups and user account mapping, see Managing Users and User Groups and Between-Host User Account Mapping.

4. Install the required software (ed.x86_64).
   If no software is installed, you can use command yum install to install it:

   ```
   yum install ed
   ```

5. Configure the server.config file and install LSF as a server host on AWS EC2.
   ```
   ./lsfinstall -s -f server.config
   ```

   The following example shows typical installation parameters to set in the server.config file.

   ```
   $ cat /home/ec2-user/lsf/lsf10.1_lsfinstall/server.config
   #  The LSF_ADMIN must be same admin as the primary cluster and you must make
   sure the user
   exists in the instance
   LSF_TOP="/home/ec2-user/lsf"
   LSF_ADMINS="ec2-user"
   LSF_TARDIR="/home/ec2-user/lsf/"

   # The below is not required if an entitlement (LSF_ENTITLEMENT_FILE) is used.
   LSF_LICENSE="/home/ec2-user/lsf/license.dat"
   LSF_SERVER_HOSTS="management.myserver.com"

   # The [resource awshost] is required in the server.config
   LSF_LOCAL_RESOURCES="[resource awshost] [resource define_ncpus_threads]"
   LSF_LIM_PORT="7869"
   ```

   LSF_LOCAL_RESOURCES (Required): LSF uses the Boolean resource name **awshost** to identify AWS instances.

   LSF_LIM_PORT (Required): The port number must be the same as the one defined on the LSF management host of your cluster.

LSF_SERVER_HOSTS (Required): List of management host candidates that the server host will communicate with for the cluster.

Note: By default, LSF maps ncpus to cores (*number_processors* x *number_cores*), but AWS treats each virtual CPU as a hyperthreaded core. To map the exact number of AWS instance virtual CPUs to LSF ncpus, add the resource name define_ncpus_threads to the list of local resources. The define_ncpus_threads resource maps the number of LSF ncpus to threads instead of cores for AWS server hosts.

6. If required, update the /etc/hosts file on the EC2 host to add the management host name so the EC2 host can ping the management host.
7. Start the LSF daemons on the instance manually and make sure that the EC2 instance can join the management host cluster as a dynamic host.
   If the EC2 instance is started properly, the **lshosts** and **lsload** commands show this dynamic host information, and the **bhosts** command shows a status of `closed_RC`.

   LSF looks up host names and addresses for all communication between hosts borrowed from a resource provider and management host candidates. Make sure that your environment settings for IP address resolution work between the LSF management host and borrowed hosts that join the cluster.

   Tip: Check whether the management host candidates can ping the borrowed EC2 instances by using the host name from the **hostname** command and vice versa. Also, make sure that the borrowed EC2 instances can ping themselves with the reported host name from the **hostname** command.
   If the hosts can ping themselves and each other only by using IP address but not by using the host name, DNS settings need to be configured.

   If the borrowed host cannot join the cluster, check the VPN, firewall, or security group settings. You must open firewall rules for all LSF ports between the LSF management and borrowed server host IP ranges.

   For more information on troubleshooting refer to Logging and troubleshooting the LSF resource connector.

8. Shut down the daemons.
9. Create an Amazon machine image (AMI) from the EC2 instance.
   Tip: Remember the name of the image (`imageId`) as it is required in the awsprov_templates.json file for LSF to launch instances based on this AMI with LSF installed.
   Note: The hostsetup script **must not** be executed on the AWS EC2 instance. LSF uses the user_data.sh script to startup the LSF daemons on the launched instances.

Enabling LSF resource connector for Amazon Web Services (AWS)

Enabling LSF resource connector for AWS is done through first executing aws_enable.sh script then following with a few manual steps. The script needs to be executed on the LSF management host where the launch request of the AWS EC2 instance takes place.

- **aws_enable.sh script**
- **Choose account authentication method**
  Choose whether to use an IAM user or federated account to access AWS.
- **Executing the AWS enablement script for LSF**
- **Completing the enabling of Resource Connector for AWS**

aws_enable.sh script

The aws_enable.sh script updates the following resource connector configuration files:

- <LSF_TOP>/conf/resource_connector/aws/conf/awsprov_config.json
- <LSF_TOP>/conf/resource_connector/aws/conf/credentials
- <LSF_TOP>/<LSF_VERSION>/resource_connector/aws/scripts/user_data.sh
- <LSF_TOP>/conf/resource_connector/policy_config.json
- <LSF_TOP>/conf/resource_connector/hostProviders.json

These files are backed up to <*file_name*>.aws_backup before they are updated by the aws_enable.sh script. All updated files are rolled back if the script fails.

These configurations are mandatory to enable LSF resource connector for AWS.

Detailed steps for using the aws_enable.sh script are described in the aws_enable.config file. After installation of LSF on the management host, the aws_enable.sh and aws_enable.config files are located in <*LSF_TOP*>/lsf_version/install.

The aws_enable.sh script sets the following LSF configuration:

- The resource connector demand calculation scheduler module is configured. The schmod_demand plugin is uncommented in the lsb.modules file.
- The awshost Boolean resource is added to the lsf.shared file to identify hosts that are borrowed from AWS.
- The LSB_RC_EXTERNAL_HOST_FLAG=awshost parameter is configured in the lsf.conf file to enable the resource connector for AWS.
- An AWS queue awsexample is created with the RC_HOSTS=awshost parameter configured in the lsb.queues file. Set any other queue parameters you need in this queue.
- The LSF_REG_FLOAT_HOSTS=Y parameter is set in the lsf.conf file. The LSF_DYNAMIC_HOST_WAIT_TIME=2 parameter is also configured in the lsf.conf file if the parameter is not already set. If the LSF_DYNAMIC_HOST_WAIT_TIME parameter is already set, the script keeps the configured value.
- For increased security, the LSF_HOST_ADDR_RANGE parameter is configured in the lsf.cluster.<*cluster_name* > file to the value that you set in the aws_enable.config file. It is required in the aws_enable_script as it identifies the range of IP addresses that are allowed to be LSF hosts that can be dynamically added to or removed from the cluster.

The aws_enable.sh script requires the following information to be specified in the aws_enable.config file to enable RC to burst to AWS:

- AWS_LSF_TOP is a required parameter of the top directory that LSF was installed to in the AWS AMI created for cloud-bursting.
  Purpose: This value will be used to update the user_data.sh script which allows the newly launched instances to find LSF when the instance starts up.

- AWS_REGION is to be set as the Amazon Web Services region of the user's account.
  Purpose: LSF will gather information from this region only and launch instances into this region. Only one region can be supported per provider.

- If AWS_IAM_CREDENTIAL_ID and AWS_IAM_CREDENTIAL_KEY parameters are set in the aws_enable.config file, then these values will be set as the AWS parameters aws_access_key_id and aws_secret_access_key in the credentials file.

Purpose: LSF will use this credential to access AWS through the AWS API's

- If AWS_FED_CREDENTIAL_SCRIPT is set in the aws_enable.config file, then LSF will not use a fixed credential to access AWS. Instead it will use this user provided script to generate a temporary credential to use.
  Purpose: LSF will use this script to generate and renew temporary credentials to access AWS.

Choose account authentication method

Choose whether to use an IAM user or federated account to access AWS.

Before you begin

For either of the authentication methods that you choose, the user's role that is used for LSF configuration must have at least the following AWS permissions granted to that user for the minimal cloud bursting to AWS:

- ec2:DescribeInstances
- ec2:DescribeImages
- ec2:DescribeKeyPairs
- ec2:DescribeSecurityGroups
- ec2:DescribeAvailabilityZones
- ec2:RunInstances
- ec2:TerminateInstances
- ec2:StopInstances
- ec2:StartInstances"

Note: Some advanced configurations require additional policies. The `iam:PassRole` is needed if the instance profile feature is used.

About this task

Select one of the following account authentication methods to access AWS.

Procedure

- Create an IAM access key and credential files.
  To create an access key and credential files, log in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/. IAM allows secure access to AWS resources for users and also allows shared access to an AWS account. If you create an access key for each user using the web GUI, you must download the credentials. A credentials.csv file is generated.

  Tip: The access key ID and secret access key in the credentials.csv file is needed in the aws_enable.sh script or can be added directly to the LSF credentials file.
- Use federated accounts for AWS.
  A wrapper script is required for this authentication method.

  Federated users are external identities that are granted temporary credentials with secure access to resources in AWS without requiring creation of IAM users. Users are authenticated outside of AWS (for example, through Windows Active Directory). LSF resource connector integrates with federated accounts through a user defined script that requires specific format for the output.

The roles for the user must have the required policies and permissions attached in AWS.

For more information, refer to the following Amazon documentation:

- https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers.html
  Follow the instructions in the section, Configuring AWS access with federated accounts to create the script. The location of the script will need to be set as the value of `AWS_FED_CREDENTIAL_SCRIPT`. The AWS resource connector can be enabled first with IAM and switched later to federated accounts after through the awsprov_config.json file.
- https://aws.amazon.com/identity/federation/.

Note: The aws_enable.sh script must be executed on the local LSF management host. The local management host is the machine that initiates the AWS EC2 instances.

- Use the management host instance profile credentials.
  When the LSF management host and the resource connector are deployed in an AWS EC2 instance with an appropriate instance profile, the resource connector uses the instance profile's credentials to access the AWS API.

  This authentication method requires that the awsprov_config.json configuration file does not contain the AWS_CREDENTIAL_FILE and AWS_CREDENTIAL_SCRIPT parameters.

  For more information about using the management host instance profile credentials, see the Amazon documentation: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2.html.

Executing the AWS enablement script for LSF

Before you begin

LSF should already be installed on an AWS AMI and already installed on the on-premise management host.

About this task

To use the aws_enable.sh script, follow these steps.

Procedure

1. Source the LSF profile (for example, <*LSF_TOP*>/conf/profile.lsf).
2. Edit the <*LSF_TOP*>/<*LSF_VERSION*>/install/aws_enable.config file to configure the installation parameters.
3. Run `./aws_enable.sh -f aws_enable.config`.
   If live configuration in LSF is enabled, the lsb.queues and lsf.cluster.*cluster_name* under the LSF_LIVE_CONFDIR directory is updated by the aws_enable.sh script.

   Note: To disable the LSF resource connector for AWS after you run the aws_enable.sh script, comment out or remove the line in the lsf.conf where the AWS host resource is defined (in the LSB_RC_EXTERNAL_HOST_FLAG=awshost parameter).

Completing the enabling of Resource Connector for AWS

Before you begin

Before performing this task, the aws_enable.sh script must already have been executed successfully.

About this task

To complete the enabling of Resource Connector for AWS perform the following steps:

Procedure

1. Create the hostProviders.json file with `lsfadmin` as the owner.
   The hostProviders.json file specifies which resource providers that the LSF resource connector can use. Create this file in the default directory, <*LSF_TOP*>/conf/resource_connector

   For more information, refer to hostProviders.json.

```
{
    "providers":[
        {
            "name": "aws",
            "type": "awsProv",
            "confPath": "resource_connector/aws",
            "scriptPath": "resource_connector/aws"
        }
    ]
}
```

2. Create AWS instance templates.
   Create at least one template in the awsprov_templates.json file in the default directory, <*LSF_TOP*>/conf/resource_connector/aws/conf/awsprov_templates.json.
   Make sure that your template accurately defines at least the following attributes:
   - `ncpus`
   - `awshost`

   The following template is a minimal example for hosts with 4 CPUs:

```
{
     "Templates":
     [
       {
         "templateId": "TemplateA",
         "attributes":
         {
           "ncpus": ["Numeric", "4"],
           "awshost": ["Boolean", "1"]
         },
         "imageId": "ami-27ai",
         "vmType": "t2.micro",
         "subnetId": "subnet-b573",
         "keyName": "LSF-Key",
         "MaxNumber": "10",
         "securityGroupIds": ["sg-7231"]
       }
```

```
        ]
}
```

What to do next

To validate the configuration of LSF resource connector for AWS, refer to Submitting jobs to launch instances from Amazon Web Services to submit a test job to ensure LSF can launch an AWS EC2 instance with LSF daemons running. The host may not be able to join successfully to the LSF cluster due to DNS or network settings. If this is the case, the instance will be created and live for a few minutes and terminated. LSF will then retry and launch subsequent AWS EC2 instances.

Kill the job to stop LSF from retrying and continue to Configuring user scripts to register Amazon Web Services hosts with the LSF master host to configure the user_data.sh script to try fixing any specific DNS or environment issues.

- **Configuring user scripts to register Amazon Web Services hosts with the LSF management host**
  If a DNS server is not set up to resolve the LSF management host and AWS instances, use the user_data.sh script under the <*LSF_TOP*>/<*LSF_VERSION*>/resource_connector/aws/scripts directory to register the dynamic hosts and resolve the DNS entries on both sides.

Configuring user scripts to register Amazon Web Services hosts with the LSF management host

If a DNS server is not set up to resolve the LSF management host and AWS instances, use the user_data.sh script under the <*LSF_TOP*>/<*LSF_VERSION*>/resource_connector/aws/scripts directory to register the dynamic hosts and resolve the DNS entries on both sides.

About this task

When AWS creates an instance, the EC2 host has two IP addresses and two host names (internal and public). By default, AWS uses the internal IP address and host name that is reported by the hostname command to communicate, but an external (public) LSF management host cannot recognize the private host name and IP address. For the management host to recognize the instance and use public DNS to resolve the EC2 public host name and public IP address of the instance, you must update the internal host name of the instance to its public host name before it joins the LSF cluster.

Set up the user_data.sh script to use a public DNS to resolve an EC2 public host name and IP address to communicate with the LSF management host directly. For example, add the following code to the user_data.sh script to give a machine host name that the management host can recognize:

```
# Get my public IP
publicIP=$(curl whatismyip.akamai.com)
# Make up AWS EC2 host name with public DNS
publicName=ec2-${publicIP//./-}.us-west-2.compute.amazonaws.com
# Update host name
echo ${publicName} > /etc/hostname
hostname ${publicName}
```

Tip: To validate the configuration of LSF resource connector for AWS, refer to Submit jobs to resource connector to submit a test and make sure the launched AWS instances can join the LSF cluster.

LSF looks up host names and addresses for all communication between AWS instances and management host candidates. Make sure that your environment settings for IP address resolution work between the LSF management host and instances that join the cluster.

Check whether the management host candidate can ping the instance by using both its public IP address and the host name reported by the **hostname** command and vice versa. Make sure that the instances can ping themselves and each other with both public IP address and reported host name. If the instances can ping themselves and each other only by using IP address but not by using the host name, DNS settings need to be configured.

If the instance cannot join the cluster, check the VPN, firewall, or security group settings. You must open firewall rules for all LSF ports between the LSF management and borrowed server host IP ranges.

If pre-provisioning steps are required to set up specifics on the management host when an EC2 instance is launched, users can use the pre-provision feature of LSF resource connector. Post-provisioning can be used to clean up the steps done in the pre-provision. Refer to Pre-provisioning and post-provisioning for more details.

Example

Configuring bursting behavior

LSF has two places to configure the overall decision on how to configure bursting behavior because only the scheduler (the **mbsched** daemon) has information about jobs and this information is not needed by the resource connector policy module. A decision is first made to determine if demand is generated or not at the scheduler. This is controlled at the queue level using the queue threshold.

By default, any pending workload on a cloud enabled queue triggers demand to the resource connector policy module to determine how many AWS EC2 instances to launch. A more restrictive policy can be defined to fine tune the bursting behavior with the following:

- Configuring a threshold on when to launch instances in AWS (RC_DEMAND_POLICY, configured in the queue)
- Throttling the rate of launching instances in AWS (**StepValue**) and Maximum instances to request for AWS (**MaxNumber**)
- Considering when to reclaim the AWS EC2 instances (LSB_RC_EXTERNAL_HOST_IDLE_TIME and LSB_RC_EXTERNAL_HOST_MAX_TTL)

Additional fine tuning of the policies can be done through a customized policy script which is used after the default plug-in runs to determine the final demand to be used. This is done through the policy_config.json file using the UserDefinedScriptPath parameter, which is described in policy_config.json.

- **Configuring a threshold**
  Configuring the thresholds that determine whether bursting should be considered is done at the queue level in the LSF lsb.queues file. By default there are no queues configured with thresholds.
- **Providing specific policy configurations**
  Administrators can define several policies in the policy_config.json configuration file to be in effect for the cluster. LSF evaluates the policies one after another going through the list, so that all policies are valid before any demand or host requests are made. When the scope matches for several policies, each of the policies in effect has an AND relationship.
- **Controlling reclaim behavior**

Configuring a threshold

Configuring the thresholds that determine whether bursting should be considered is done at the queue level in the LSF lsb.queues file. By default there are no queues configured with thresholds.

About this task

The RC_DEMAND_POLICY parameter has the following syntax:

```
RC_DEMAND_POLICY = THRESHOLD[ [ num_pend_jobs[,duration]] ... ]
```

The demand policy defined by the RC_DEMAND_POLICY parameter can contain multiple conditions, in an OR relationship. A condition is defined as [*num_pend_jobs*[,*duration*]]. The queue has more than the specified number of eligible pending jobs that are expected to run at least the specified duration in minutes. The *num_pend_jobs* option is required, and the duration is optional. The default threshold is THRESHOLD[[1,0]].

In the following example for the admin queue, LSF calculates demand if one of the following conditions are met:

- The queue has 5 or more pending jobs in past 10 minutes
- There has been one or more pending jobs in past 60 minutes
- There are 100 or more pending jobs.

As long as pending jobs at the queue meet at least one threshold condition, LSF expresses the demand to resource connector to trigger borrowing.

Example bursting threshold configuration for lsb.queues

lsb.queues

```
Begin Queue
QUEUE_NAME          = admin
DESCRIPTION         = Sysadmin jobs, not preempted
PRIORITY            = 50
USERS               = lsfadmins
EXCLUSIVE           = Y
RERUNNABLE          = Y
RC_ACCOUNT          = ProjectB
RC_HOSTS            = awshost
RC_DEMAND_POLICY    = THRESHOLD[[5,10] [1,60] [100,0]]
End Queue
```

- lsb.queues

Providing specific policy configurations

Administrators can define several policies in the policy_config.json configuration file to be in effect for the cluster. LSF evaluates the policies one after another going through the list, so that all policies are valid before any demand or host requests are made. When the scope matches for several policies, each of the policies in effect has an AND relationship.

About this task

The scope of the policies is controlled in the "Consumer" component of the policy. The scope is determined by the values in the `rcAccount`, `templateName` and `provider` names. Refer to policy_config.json for more details.

In the following example, GlobalPolicyA1 applies to the whole cluster. Across all combined rcAccounts, templates and providers since the "all" keyword was used for all parameters of the scope. ProjectPolicyA2 applies only to ProjectB, jobs submitted to “admin” queue, across all combined templates and providers.

Example policies configuration in policy_config.json

```
{
  "Policies":
  [
    {
      "Name": "GlobalPolicyA1",
      "Consumer":
      {
        "rcAccount": ["all"],
        "templateName": ["all"],
        "provider": ["all"]
      },
      "MaxNumber": “100”,
      "StepValue": "5:20"
     },
{
            “Name” : “ProjectPolicyA2”,
            “Consumer”:
                {
                  “rcAccount”: [“ProjectB”],
                  “templateName”: [“all”],
                  “provider”: [“all”]
                },
                “MaxNumber”: “50”,
                “StepValue”: “10:10”
      }
  ]
}
```

Controlling reclaim behavior

About this task

Two parameters control when LSF will return and terminate the AWS EC2 instances launched by LSF; LSB_RC_EXTERNAL_HOST_IDLE_TIME and LSB_RC_EXTERNAL_HOST_MAX_TTL. Both define values in minutes. Refer to LSB_RC_EXTERNAL_HOST_IDLE_TIME and LSB_RC_EXTERNAL_HOST_MAX_TTL for the default values.

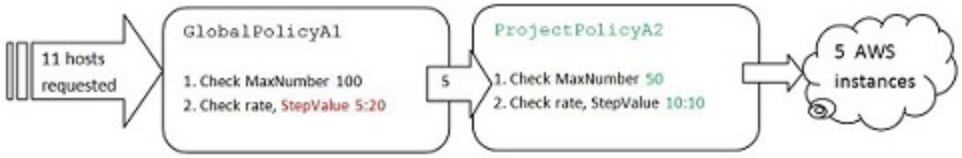
The following example uses the values of LSB_RC_EXTERNAL_HOST_IDLE_TIME=60 and LSB_RC_EXTERNAL_HOST_MAX_TTL=0. This means that when the AWS EC2 host has no workload for 60 minutes, it will be terminated.

These parameters are applied and effective for the whole LSF cluster.

Example

Combining the examples in this chapter, the following scenario describes and explain the behavior of LSF as jobs are submitted to the LSF cluster.

1. Initially at t0, (00:00), there are no AWS instances requested since none of the THRESHOLD conditions are met for the priority queue. 20 Jobs are submitted to the priority queue.
2. At t6, (00:06), there are 11 jobs submitted to the admin queue which belongs to ProjectB. Since the 11 jobs just arrived, it does not meet any of the THRESHOLD conditions for admin queue [[5,10] [1,60] [100,0]].
3. At t10, (00:10), The same 20 jobs from (1) and 11 jobs from (2) are still pending. The 11 jobs in the admin queue do not meet the THRESHOLD so no bursting is done for the admin queue. The priority queue is not cloud enabled, so no bursting for the priority queue's 20 jobs.
4. At t16, (00:16), the same 11 jobs from (2) are still pending, which has a pending time of 10 minutes. The scheduler will allow bursting as the THRESHOLD of [5, 10] is met for the admin queue. How many AWS EC2 instances will be launched is determined next.
5. The resource connector policy receives the request for 11 hosts and will evaluate the applicable policies. Both policies apply and need to be evaluated.



The order does not matter as both policies need to be met. Using GlobalPolicyA1 first, the 11 hosts request first checks MaxNumber which is 100 and it is not exceeded. Next checking the StepValue, it only allows 5 hosts to be launched every 20 minutes, so the 11 hosts request is reduced to 5.

Checking the next policy in scope, ProjectPolicyA2, the MaxNumber is not exceeded as 10 < 50. Next checking the StepValue it allows 10 hosts every 10 minutes, so since 5 < 10 the final number of launched AWS instances is 5. There will be 5 AWS EC2 instances launched at time t16.

Assigning exclusive resources to a template

When you assign exclusive resources to a template, LSF recognizes the exclusive resource definition for demand calculation. You must set up the exclusive resource when launching the instance.

An exclusive resource is a special resource that is assignable to a host. A host with an exclusive resource does not receive jobs unless that job explicitly requests the resource.

For example, you might want to run test jobs only on the cheapest instance type configured for your resource provider. You want to be able to select a template with that vmType only when you want to run on it. Unless specifically requested, this template is not chosen by the scheduler.

LSF resource connector supports an exclusive Boolean resource (for example `instance_store`) that is defined in the attribute section of a template. Resource connector recognizes the exclusive resource definition when it creates hosts based on that template. The logical not (`!`) operator is used to create hosts do not use the exclusive resource (`!instance_store`). For example,

```
{
"templates": [
{
```

```
"templateId": "Template-VM-1",
"attributes": {
"type": ["String", "X86_64"],
"ncores": ["Numeric", "1"],
"ncpus": ["Numeric", "1"],
"mem": ["Numeric", "1024"],
"awshost1": ["Boolean", "1"],
"!instance_store": ["Boolean", "1"]
},
...
"userData": "zone=us_west_2a;instance_store=!instance_store"
}
]
}
```

Add the exclusive resource to the user_data.sh file to set up the exclusive resource in the LSF_LOCAL_RESOURCES parameter when the instance is launched, and refer to it in the userData attribute in the template. For example,

```
cat user_data.sh
#
# Support rc_account resource to enable RC_ACCOUNT policy
# Add additional local resources if needed
#
if [ -n "${rc_account}" ]; then
sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap
${rc_account}*rc_account]\"/" $LSF_CONF_FILE
echo "update LSF_LOCAL_RESOURCES lsf.conf successfully, add [resourcemap
${rc_account}*rc_account]" >> $logfile
fi

if [ -n "${instance_store}" ]; then
sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resource ${instance_store}]\"/"
$LSF_CONF_FILE
echo "Updated LSF_LOCAL_RESOURCES in $LSF_CONF_FILE successfully, add [resource
${instance_store}]" >> $logfile
else
echo "instance_store does not exist in the environment variable" >> $logfile
fi
```

Configuring AWS access with federated accounts

Resource connector supports *federated accounts* for LSF resource connector as an alternative to requiring permanent AWS IAM account credentials. Federated users are external identities that are granted temporary credentials with secure access to resources in AWS without requiring creation of IAM users. Users are authenticated outside of AWS (for example, through Windows Active Directory). **All AWS resource connector features are supported when you use federated accounts instead of IAM credentials.**

Before you begin

The LSF administrator must create a script or executable that can be executed by the primary LSF administrator to generate temporary credentials to be used with AWS. The temporary credentials also must have the assigned role that has the policies associated for AWS permissions required by LSF.

The script must be accessible from LSF management candidate hosts. The script must also be set for the default profile. The script must output to stdout in the following format.

```
[default]
aws_access_key_id=<value>
aws_secret_access_key=<value>
aws_session_token=<value>
```

For example:

```
[default]
aws_access_key_id=aGJ3P1gFRQCEsPNFppTSen+fQTLqS1sLcHldPQmG
aws_secret_access_key=ASIAJ6UWHCWUWRECOKIQ
aws_session_token=
FQoDYXdzENr//////////wEaDHoDxdyu+3TeTAWQDSLTAYncjAgKT/6A1VKtbj6XJ/
l8fbMIzAg3yTIrfHNawTKBmIlAhT07HGN5zZd2YqhjHhKSNIHJUCDsW+pZ8WW+CBcqNTNDInLiM2ubPn8z
j
ItMeknniPMBfwZn+qfQCcl/QjaPgKGXzUBpfVhe202GuGr8bZno4Dzgy7yOmITTugiuUTBh9YKK27OBPZH
ieD6JzvAV0aV2mbFQaznWYhKq2s1MSy7JC4bmaFPNCN81igkfY7AVbYtwTxnFP6peVS2Dergd5Hllef9nU
+V
9WW7nk0yZLyYoxO+lxgU=
```

The script is executed automatically by LSF when a credential is required to access AWS services. The script cannot be interactive since it will be run automatically by LSF.
Note: If an Active Directory user name and password is required for the script, it must be done automatically by storing it in environment variables or a secure file. The environment variable or file must be readable by the LSF primary administrator because that user executes the script.

About this task

Identity federation in AWS allows external identities (federated accounts) to access AWS services and resources while being authenticated and authorized outside of AWS IAM. This allows integration of AWS with existing authentication services (for example, Active Directory) in enterprise environments instead of requiring administrators to create IAM user credentials for each existing user.

Important: To use existing enterprise users stored in Active Directory (federated accounts) you must use temporary credentials.
Federated accounts in AWS require temporary credentials to be generated to allow connection to AWS. These credentials are temporary and expire after a specific duration. LSF resource connector generates the temporary credentials through execution of an administrator-defined script or executable that is able to generate the temporary credentials for LSF to use. LSF uses the temporary credentials to establish a connection to AWS to launch, monitor and terminate EC2 instances.

Procedure

Use the AWS_CREDENTIAL_SCRIPT parameter in the awsprov_config.json file to specify a path to the script that generates temporary credentials for federated accounts.
For example,

```
AWS_CREDENTIAL_SCRIPT=/shared/dir/generateCredentials.py
```

LSF executes the script as the primary LSF administrator to generate a temporary credentials before it creates the EC2 instance.

- awsprov_config.json
- awsprov_templates.json

Configure launch templates for AWS

A launch template is an Amazon Elastic Compute Cloud (EC2) feature that reduces the number of steps that are required to create an AWS instance by capturing all launch parameters within one resource.

About this task

A launch template contains the configuration information to launch an instance so that you do not have to specify them each time you launch an instance. For example, a launch template can contain the AMI ID, instance type, and network settings that you typically use to launch instances.

You can configure the LSF resource connector template for AWS to specify the launch template to use. You can also create multiple launch template versions, each of which can have different launch parameters, and specify which launch template version to use.

Procedure

Edit the awsprov_templates.json file and specify the launch template parameters.

a. Set the launchTemplateId parameter value to the ID of the launch template.
   This can be a string between 1 and 255 characters in length.

b. Optional: Set the launchTemplateVersion parameter value to the version of the launch template to select when launching instances.
   Specify a version number or use one of the following keywords:

   $Latest
   : Amazon EC2 Auto Scaling selects the latest version of the launch template when launching instances.

   $Default
   : Amazon EC2 Auto Scaling selects the default version of the launch template when launching instances. This is the default value of the launchTemplateVersion attribute.

Results

You can launch on-demand or spot instances with launch templates by specifying the launch template. If the attached template is a `spot` type, LSF creates a spot instance without creating a fleet request. Create spot fleet requests by specifying the spot price and fleet role along with the launch template.

Users cannot override the subnet ID and security group from the launch template if the network interface is defined in the launch template, even if the network interface is defined as an empty value. Users can override launch parameters such as the instance type, AMI, keypair, and security group if the network interface is not defined in the specified template by specifying those values in the awsprov_templates.json file

Example

```
{
   "templateId": "aws-vm-1",
   "maxNumber": 6,
   "priority" : 80,
```

```
   "attributes": {
      "type": ["String", "X86_64"],
      "ncores": ["Numeric", "1"],
      "ncpus": ["Numeric", "1"],
      "mem": ["Numeric", "512"],
      "awshost": ["Boolean", "1"],
      "zone": ["String", "us_west_2b"],
      "pricing": ["String", "spot"]
   },
   "launchTemplateId" : "lt-007f0f860d19c8848",
   "launchTemplateVersion" : "$Latest",
   "fleetRole": "arn:aws:iam::700071821657:role/aws-ec2-spot-fleet-tagging-role",
   "allocationStrategy":"lowestPrice",
    "spotPrice": "1.16"
}
```

- awsprov_templates.json
- Attach EFA network interfaces to AWS templates

Attach EFA network interfaces to AWS templates

The Elastic Fabric Adapter (EFA) is a network interface for Amazon Elastic Compute Cloud (EC2) instances that allows you to run HPC applications with improved levels of communication between several different nodes.

Before you begin

You can only specify an EFA network interface for supported AMI or instance types. For more details on supported AMI or instance types for EFA interfaces, refer to the Amazon Web Services website (https://aws.amazon.com/).

Procedure

1. Install and configure the EFA network interface.
   a. Perform the relevant steps in *Getting Started with EFA and MPI* from the Amazon Web Services website (https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/efa-start.html).
   To install and configure the EFA network interface for the LSF resource connector templates for AWS, perform the following steps in *Getting Started with EFA and MPI*:

      i. Prepare an EFA-enabled security group.
      Ensure that this security group allows all inbound and outbound traffic to and from the security group itself.

      ii. Launch a temporary instance.
      iii. Install the EFA software.
      Install the EFA-enabled kernel, EFA drivers, Libfabric, and Open MPI stack, which are required to support EFA on the launched instance.

      iv. Install Intel MPI.
      v. Install the HPC application.
      vi. Create an EFA-enabled AMI.
      vii. Launch EFA-enabled instances into a cluster placement group.

2. Edit the awsprov_templates.json file and specify the EFA network interface for your specific LSF resource connector template for AWS.
    a. Set the imageId parameter value to the same ID when you created an EFA-enabled AMI.
    b. Set the securityGroupIds parameter value to the same security group ID when you created an EFA-enabled security group.
    Ensure that your EFA-enabled instances are members of a security group that allows all inbound and outbound traffice to itself.

    c. Add a new parameter, interfaceType, and set the parameter value to `efa`.
    If this parameter is not specified, the default value is `interface`, which indicates that the template uses a regular (non-EFA) network interface.

    d. Set the maxNumber parameter value to the number of EFA-enabled instances that you want to launch.
    e. Set the vmType parameter value to one of the supported instance types.
    For more details on supported instance types for EFA interfaces, refer to *Elastic Fabric Adapter* from the Amazon Web Services website (https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/efa.html).

    f. Set the subnetId and placementGroupName parameter values for your cluster.
    Ensure that you launch EFA-enabled instances into a cluster placement group.

    A new network interface is created per instance. These automatically-created network interfaces are deleted when the instance terminates.

Example

```
{
   "templateId": "aws-vm-1",
   "maxNumber": 2,
   "attributes": {
      "type": ["String", "X86_64"],
      "ncores": ["Numeric", "1"],
      "ncpus": ["Numeric", "1"],
      "mem": ["Numeric", "512"],
      "awshost": ["Boolean", "1"],
      "zone": ["String", "us_west_2a"],
      "pricing": ["String", "spot"]
   },
   "imageId": "ami-07e3bcfe7a2a2fbb8",
   "vmType": "c5n.18xlarge",
   "keyName": "ib19b07",
   "interfaceType" : "efa",
   "placementGroupName": "pg_1",
   "tenancy": "default",
   "securityGroupIds": ["sg-08f1a36be62fe02a4"],
   "subnetId": "subnet-0fe69d290ae026155",
   "fleetRole": "arn:aws:iam::700071821657:role/EC2-Spot-Fleet-role",
   "allocationStrategy":"lowestPrice",
   "spotPrice": "1.16",
   "userData": "zone=us_west_2a;pricing=spot"
}
```

- awsprov_templates.json
- Configure launch templates for AWS

Use AWS spot instances

Use *spot instances* to bid on spare Amazon EC2 computing capacity. Since spot instances are often available at a discount compared to the pricing of On-Demand instances, you can significantly reduce the cost of running your applications, grow your application's compute capacity and throughput for the same budget, and enable new types of cloud computing applications.

With spot instances you can reduce your operating costs by up to 50-90%, compared to on-demand instances. Since spot instances typically cost 50-90% less, you can increase your compute capacity by 2-10 times within the same budget.

Spot instances are supported on any Linux x86 system that is supported by LSF.

Spot Instances have some restrictions, including instance types and fleet limitations. For more information, see http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-limits.html

Requesting Spot instances

Submit the job that requires spot instance pricing with the `pricing==spot` resource requirement in the **bsub** command:

```
bsub -R "awshost && pricing==spot" myjob
```

The `pricing` resource must be configured in the lsf.shared file:

```
Begin Resource
RESOURCENAME    TYPE    INTERVAL   INCREASING   DESCRIPTION
...
pricing         String    ()         ()          (Pricing option: spot/ondemand)
...
End Resource
```

Spot instances are reclaimed when the spot price goes higher than the current bid price.

You can also configure an AWS template to use spot instances.

```
awsprov_templates.json:
{
             "templateId": "aws-spotvm-demo",
             "maxNumber": 2,
             "attributes": {
                 …
                  "awshost": ["Boolean", "1"],
                 "pricing": ["String", "spot"],
               },
             ...
             ...
             ...
             "userData": "pricing=spot"
    },
```

Edit the user_data.sh script to use the spot instance `pricing` resource:

```
#!/bin/bash
echo START >> /var/log/user-data.log 2>&1
# run hostsetup
```

```
...
if [ -n "${pricing}" ]; then
sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap ${pricing}*pricing]\"/"
$LSF_CONF_FILE
echo "update LSF_LOCAL_RESOURCES lsf.conf successfully, add [resourcemap
${pricing}*pricing]" >> $logfile
fi
...
```

The user_data.sh script is located in the <*LSF_TOP*>/<*LSF_VERSION*>/resource_connector/aws/scripts directory.

Security requirements for spot instances

- You must create a spot fleet role add the corresponding Amazon Resource Name (ARN) to the awsprov_templates.json template configuration file. For steps to create a spot fleet role, see http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet-requests.html#spot-fleet-prerequisites
- The AWS user linked to the access key that is stored in the credentials file must have the Spot fleet permissions to bid on, launch, and terminate the configured Spot fleets. For steps to add permissions to a user, see http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet-requests.html#spot-fleet-prerequisites

Logging and troubleshooting

To increase traceability, use the `TRACE` log level in the LogLevel parameter in the awsprov_config.json file. This log level prints the entry of the method with the value of the parameters and the exit of the method with the return value (if exists).

The following troubleshooting messages are created when the log level is configured as `DEBUG`. For troubleshooting purposes, every state change on a Spot instance request is logged with a predefined format:

```
Spot Fleet Request ID - Spot Instance Request Id- Spot Instance Machine ID: State
update message
```

Limitations and known issues

- The Spot Instance Termination Notice is not accurate if the system clock is not synchronized between the management host and the compute host. System clock synchronization is required for reclaim to work.
  The following AWS topic explains this issue: :
  http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/set-time.html

  .
- If a request remains pending for 60 minutes, resource connector assumes that the request is lost. The request is ignored and LSF recalculates the demand. In AWS Spot instances, the request remains pending and is not closed.
- LSF checks periodically for any hosts that are planned to be reclaimed and requeues the jobs within the 2 minute termination notice. However, it's possible that AWS might not honor the 2 minute termination notice, and machines are terminated without a termination notice. For more information, see: :
  http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-interruptions.html#spot-instance-termination-notices

- **Configuring AWS Spot instances**
  Configure LSF to make Spot instance requests.

- Amazon Spot Instances
- Amazon Spot Bid Advisor
- awsprov_templates.json
- awsprov_config.json

Configuring AWS Spot instances

Configure LSF to make Spot instance requests.

Procedure

1. Configure the `pricing` resource in the lsf.shared file:

```
Begin Resource
RESOURCENAME    TYPE    INTERVAL   INCREASING   DESCRIPTION
...
pricing         String    ()         ()          (Pricing option: spot/ondemand)
...
End Resource
```

2. Configure a Spot instance template in the awsprov_templates.json file.
   The following parameters enable Spot instance requests:

   vmType
   : Specify a machine type of the AWS instance you want to create. The `vmType` must support Spot instances.
     Use commas to separate multiple machine types. For example:

     ```
     "vmType": "c4.large, m4.large"
     ```

     For supported machine types, see http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-limits.html#spot-limits-unsupported

   fleetRole
   : For Spot Instance templates. Specify the role that grants the permission to bid on, launch, and terminate spot fleet instances on behalf of the user.
     For the steps to create a Fleet role, see:
     http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet-requests.html#spot-fleet-prerequisites

   spotPrice
   : For Spot instance templates. Specify the bid price for the instance. Set a suitable value (usually the On-Demand price) to make sure that the Spot price is equal to or above the market Spot price. The Spot instance is launched when the Spot price of the instance is below the bid specified in the `spotPrice` attribute.
     For more information about Spot pricing, see https://aws.amazon.com/ec2/spot/bid-advisor/

   allocationStrategy

For Spot instance templates. The allocation strategy for your Spot fleet determines how it fulfills your Spot fleet request from the possible Spot instance pools that are represented by its launch specifications. You can specify the following allocation strategies in your Spot fleet request:

lowestPrice
: The Spot instances come from the pool with the lowest price. This is the default strategy.

diversified
: The Spot instances are distributed across all pools.

3. Optional: In the awsprov_config.json file, configure the AWS_SPOT_TERMINATE_ON_RECLAIM parameter.
   The AWS_SPOT_TERMINATE_ON_RECLAIM parameter processes requests for terminating Amazon EC2 Spot instances that are planned to be reclaimed by AWS.
   If set to `true`, the AWS plugin sends an instance termination request to AWS when notified by LSF that the reclaimed Spot instance has been closed and the affected jobs are requeued.

   Valid values are `true` and `false`. The default value is `false`.

4. Optional: Starting in IBM Spectrum LSF Version 10.1 Fix Pack 13, if you set the allocationStrategy to lowestPrice, LSF automatically redirects to the next available template. The next available template is based on template priority, which can be the next cheapest Spot or On-Demand instance when the marketed spot price is higher than the bid price.. If another allocation strategy is set, such as diversified, LSF does not automatically switch to the cheapest template.
   You can view the current market price, by running the following command:

   ```
   badmin rc view -c templates -p aws
   ```

   An example of the output, when the template spot price is equal or greater than the market price, which enables the template:

   ```
   aws
       Templates
           templateId: aws2
               maxNumber: 2
               spotPrice: 0.004000     >>> This is the spot price set that is
   higher than the current market price.
               fleetRole: arn:aws:iam::700071821657:role/EC2-Spot-Fleet-role
               allocationStrategy: lowestPrice
               imageId: ami-0dbddce684d30c81d
               subnetId: subnet-0dfee843e19bfeb52
               vmType: t3a.micro
               keyName: ib19b07
               userData: pricing=spot;zone=us_west_2b
               marketSpotPrice: 0.003200      >>> This is the real current
   market price, which is lower than the spot price.
               securityGroupIds: [ "sg-08f1a36be62fe02a4" ]
               ebsOptimized: FALSE
               priority: 5
               attributes
                   mem: ["Numeric", "700"]
                   ncpus: ["Numeric", "2"]
                   zone: ["String", "us_west_2b"]
                   awshost: ["Boolean", "1"]
                   ncores: ["Numeric", "1"]
                   type: ["String", "X86_64"]
                   pricing: ["String", "spot"]
   ```

   An example output, when the template spot price is less than market price, which disables the template:

```
aws
templateId: aws-vm-3
            spotPrice: 0.003800
            marketSpotPrice: 0.031500
        Template disabled as the spot bid price is lesser than the market
spot price
```

Starting in IBM Spectrum LSF Version 10.1 Fix Pack 13, LSF automatically disables the template as the spot bid price is lesser than the market spot price and redirects to the next available template. The next available template is based on template priority, which might be the next cheapest Spot or On-Demand instance template. This template is temporarily disabled until the market price drops or the price is manually increased in the template. To manually increase the spot price, specify a greater bid price for the spotPrice parameter in the awsprov_templates.json file.

Using Amazon EC2 Fleet

As of Fix Pack 14, the LSF resource connector for Amazon Web Services (AWS) uses an Amazon EC2 Fleet API to create multiple (that is, a fleet of) instances. EC2 Fleet is an AWS feature that extends the existing spot fleet, which gives you a unique ability to create fleets of EC2 instances composed of a combination of EC2 on-demand, reserved, and spot instances, by using a single API. Follow these steps to configure AWS using Amazon EC2 Fleet to create instances for LSF resource connector to make allocation requests on behalf of LSF.

Before you begin

- For more information about Amazon EC2 Fleet, see the AWS documentation.
- Before using EC2 Fleet, ensure that you complete the AWS prerequisites for EC2 Fleet, and specifically note:
  - You require an AWS launch template. You can create this using the AWS EC2 console and set attributes accordingly.
  - The EC2 Fleet `maintain` type is not supported; only set to `instant` or `request` types. Additionally, before you can use an EC2 Fleet `request` type, you must create the service-linked role for EC2 Fleet, called `AWSServiceRoleForEC2Fleet`. This role grants the EC2 Fleet permission to request, launch, terminate, and tag instances on your behalf.

About this task

To use the EC2 Fleet API to create instances, the LSF configurations required after your AWS prerequisites include configuring the LSF awsprov_templates.json template and then to create a customized EC2 Fleet .json file. Finally, you can verify your LSF configuration by running the **badmin rc view** command.

Procedure

1. Configure your awsprov_template.json file, and ensure that it includes the ec2FleetConfig parameter, and optionally, the onDemandTargetCapacityRatio parameter, as follows:

   ec2FleetConfig
   : An absolute or relative path to the EC2 Fleet configuration file (for example, to a ec2-fleet-config.json file). For relative path, the path must be relative to LSF_TOP/conf/resource_connector/aws/conf directory.

Tip: If you have not yet created this file, follow the next step, and then return to this parameter to provide the path to the file.

onDemandTargetCapacityRatio
: Optional. Defines how on-demand and spot instances are distributed among the **`TotalTargetCapacity`** in each EC2 Fleet request.
Specify a value that is a positive float number between 0.0 and 1.0. The value represents the ratio between OnDemandTargetCapacity to TotalTargetCapacity. To request pure on-demand or pure spot instances, you can set this ratio to 1 or 0. If not defined, it follows the DefaultTargetCapacityType in the ec2FleetConfig file.

maxNumber
: The existing maxNumber parameter restricts the number of instances that can be provisioned in the template. Note that as of Fix Pack 14, to support AWS EC2 Fleet templates, the MaxNumber is a multiplier of the ncpu value, not a direct number of instances. For EC2 Fleet, maxNumber multiplied by ncpus is the maximum slots that EC2 Fleet template can get and can be provisioned in this template. For example, if the maxNumber is **`5`** and **`ncpus`** is **`2`**, then the maximum slots for the fleet request will be **`10`**.

Here is an example awsprov_templates.json template with EC2 Fleet parameters:

```
{
    "templates": [
        {
            "templateId": "fleet-lsf-template-1",
            "maxNumber": 5,
            "attributes": {
                "type": ["String", "X86_64"],
                "ncores": ["Numeric", "1"],
                "ncpus": ["Numeric", "2"],
                "mem": ["Numeric", "512"],
                "awshost": ["Boolean", "1"]
            },
            "priority": "121",
            "ec2FleetConfig": "ec2-fleet-config.json",
            "onDemandTargetCapacityRatio":"0.5",
            "instanceTags": "Name=fleet-lsf-template-1"
        }
    ]
}
```

Save the file. The default location for the file is <*LSF_TOP*>/conf/resource_connector/aws/conf; for example, <*LSF_TOP*>/conf/resource_connector/aws/conf/awsprov_templates.json .

2. Configure a customized EC2 Fleet configuration file (for example, one called ec2-fleet-config.json). AWS uses this file to override the sections from AWS launch template.
EC2 Fleet has various attributes. When you define the EC2 Fleet configuration file, use JSON format with the attributes that you want. For more information, see the AWS documentation: EC2 Fleet example configurations.
Note: The EC2 Fleet **`maintain`** type is not supported. It must be explicitly set to **`instant`** or **`request`**.
The following example EC2 Fleet configuration file shows an **`instant`** type.
When creating the EC2 Fleet configuration file, note the following parameters:

   LaunchTemplateId
   : Required. Identifies the AWS launch template.

   Overrides
   : Required. Contains all the customized attributes and values pairs. The WeightedCapacity must be defined in this Overrides section for each VM type in the EC2 Fleet configuration file. The value of each VM type's **`WeightedCapacity`** is the slots number in LSF, which depends on the

EGO_DEFINE_NCPUS=cores or **EGO_DEFINE_NCPUS=threads** setting in the lsf.conf configuration file. When EGO_DEFINE_NCPUS is not defined in the lsf.conf file, or if **EGO_DEFINE_NCPUS=cores**, then this value must match the instance number of cores. If **EGO_DEFINE_NCPUS=threads** is defined in the lsf.conf file, then this value must match the instance number of virtual CPUs. In the following example EC2 Fleet configuration file, if **EGO_DEFINE_NCPUS=cores**in lsf.conf, then the **WeightedCapacity** value is **2**, if **EGO_DEFINE_NCPUS=threads** in the file, then the **WeightedCapacity** value is **4**.

Priority
: Required. The AWS priority for the instance. A lower number will be a higher priority.

TargetCapacityUnitType and InstanceRequirements
: Note that these parameters are not supported.

TargetCapacitySpecification
: Required. The target capacity must be specified, for example:

```
"TargetCapacitySpecification":{
                "TotalTargetCapacity": $LSF_TOTAL_TARGET_CAPACITY,
                "OnDemandTargetCapacity": $LSF_ONDEMAND_TARGET_CAPACITY,
                "SpotTargetCapacity": $LSF_SPOT_TARGET_CAPACITY,
                "DefaultTargetCapacityType": "spot   |  on-demand"
        }
```

Here is an example ec2-fleet-config.json file:

```
{
    "LaunchTemplateConfigs": [
        {
            "LaunchTemplateSpecification": {
                "LaunchTemplateId": "lt-07e47351a93fc8b4f",
                "Version": "1"
            },
            "Overrides":[
                {
                        "InstanceType":"c3.large",
                        "SubnetId":"subnet-0fe69d290ae026155",
                        "WeightedCapacity": 2,
                        "Priority": 30
                },
                {
                        "InstanceType":"c3.xlarge",
                        "SubnetId":"subnet-0dfee843e19bfeb52",
                        "WeightedCapacity": 4,
                        "Priority": 20
                },
                {
                        "InstanceType":"c3.2xlarge",
                        "SubnetId":"subnet-0d206516fa58d74b8",
                        "WeightedCapacity": 8,
                        "Priority": 10
                }
            ]
        }
    ],
    "TargetCapacitySpecification": {
        "TotalTargetCapacity": $LSF_TOTAL_TARGET_CAPACITY,
        "OnDemandTargetCapacity": $LSF_ONDEMAND_TARGET_CAPACITY,
        "SpotTargetCapacity": $LSF_SPOT_TARGET_CAPACITY,
        "DefaultTargetCapacityType": "spot"
    },
    "OnDemandOptions": {
                "AllocationStrategy": "prioritized"
            },
```

```
    "SpotOptions": {
        "AllocationStrategy": "capacity-optimized-prioritized",
        "InstanceInterruptionBehavior": "terminate"
    },
    "Type": "instant"
}
```

Save the file. The default location for the file is <*LSF_TOP*>/conf/resource_connector/aws/conf; for example, <*LSF_TOP*>/conf/resource_connector/aws/conf/ec2-fleet-config.json

3. Verify your LSF EC2 Fleet configuration by running the **badmin rc -c templates** command.
   For example, running **badmin rc view -c templates** based on the example outputs:

```
# badmin rc view -c templates
aws
    Templates
        templateId: fleet-lsf-template-1
            maxNumber: 5
            instanceTags: Name=fleet-lsf-template-1
            ebsOptimized: FALSE
            priority: 121
            ec2FleetConfig: ec2-fleet-config.json
            onDemandTargetCapacityRatio: 0.500000
            attributes
                mem: ["Numeric", "512"]
                ncpus: ["Numeric", "2"]
                awshost: ["Boolean", "1"]
                ncores: ["Numeric", "1"]
                type: ["String", "X86_64"]
```

Running **badmin rc view** based on the example when there are jobs. The output shows **(slot-based-capacity)** to indicate that the template is an EC2 Fleet request template and therefore, the total target demand is slots based capacity. The example shows a request of ten slots (not ten VMs):

```
# badmin rc view
aws
    Instances
        Target update time: Thu May 26 17:05:59 2023
        Query request time: Thu May 26 17:05:52 2023
            templateId: fleet-lsf-template-1 (slot-based-capacity)
                rcAccount: default
                    Target: 10
                    Processing requests: 0
                    Fulfilled requests
                        Not available: 0
                        Available: 10
```

Submitting jobs to launch instances from Amazon Web Services

Use the **bsub** command to submit jobs that require instances that are launched from AWS as the resource provider. Use the **bhosts** to monitor borrowed hosts. Use the **bhosts** command to monitor host status.

About this task

In this task, `ip-10-11-13-19` is a sample instance from AWS.

An AWS allocation occurs as follows:

- Resource connector calls a command that makes a machine instance launch request to AWS.
- After the launch command returns successfully, resource connector notifies LSF that it can use the host after it joins the cluster.
- When the instance starts, LSF daemons start. When the host joins the cluster, the job is dispatched to the host.

Procedure

1. Use the **bsub** command to submit jobs that require instances that are launched from AWS as the resource provider.
   The following **bsub** command with no options submits a job that triggers a launch demand when no available resources are in the LSF cluster:

   ```
   bsub myjob
   ```

   You also can use the `awshost` resource in a `select[]` resource requirement string. Because the `awshost` resource is defined in a template as a Boolean attribute, it triggers a launch demand:

   ```
   bsub -R "select[awshost]" myjob
   ```

2. Use the **bhosts** command to monitor instances.
   The status of the instances becomes `ok` when they join the LSF cluster as dynamic hosts.
   Verify that the job is running on `ip-10-11-13-19`:

   ```
   bhosts -a
   HOST_NAME                  STATUS       JL/U    MAX  NJOBS    RUN  SSUSP
   USUSP    RSV
   lsfmanagement              ok              -      1      0      0      0
   0       0
   ip-10-11-13-19             ok              -      1      1      1      0
   0       0
   ```

3. Use the **bhosts** command to monitor the status of the instances.
   Run **bhosts** with -a option, which shows all hosts, including terminated instances.

   If an instance from AWS has no running jobs on it in the number of minutes specified by the LSF_EXTERNAL_HOST_IDLE_TIME parameter, it is relinquished and its host status changes to `closed_RC`.

   You cannot use the **badmin hopen** command to open a borrowed host in `closed_RC` status.

   ```
   bhosts -a
   HOST_NAME                  STATUS       JL/U    MAX  NJOBS    RUN  SSUSP
   USUSP    RSV
   lsfmanagement              ok              -      1      0      0      0
   0       0
   ip-10-11-13-19             closed_RC       -      1      0      0      0
   0       0
   ```

   If an instance is in the cluster more than the number of minutes specified by the LSB_RC_EXTERNAL_HOST_MAX_TTL parameter, it is closed (`closed` status) and any running jobs on the instance are allowed to run to completion.

   ```
   bhosts -a
   HOST_NAME      STATUS      JL/U   MAX   NJOBS   RUN   SSUSP   USUSP     RSV
   lsfmanagement   ok          -      1      0      0       0       0       0
   ip-10-11-13-19 closed_RC   -      1      1      1       0       0       0
   ```

4. Optional: Use external job submission and execution controls.
   Use the job submission and execution controls feature to use external, site-specific executable files to validate, modify, and reject jobs, transfer data, and modify the job execution environment. To control job submissions, such as permission checks before instances are launched from AWS, you can set up an external submission (**esub**) script.

   For more information, see External Job Submission and Execution Controls.

Results

AWS can reclaim EC2 Spot instances. Amazon EC2 reclaims a spot instance when the Spot price is greater than the bid price placed in the request.

- Every 30 seconds (the LSB_RC_QUERY_INTERVAL, configured in the lsf.conf file), a request is sent to AWS to check if any machine is eligible to be reclaimed by AWS. AWS provides a 2 minute termination notice.
- If an instance is marked to be terminated or was already terminated, resource connector sends a relinquish machine request to LSF.
  - If the AWS_SPOT_TERMINATE_ON_RECLAIM=true parameter is set in the awsprov_config.json file, the AWS plug in sends an instance termination request to AWS when notified by LSF that the reclaimed Spot instance has been closed and the affected jobs are requeued.
  - If the AWS_SPOT_TERMINATE_ON_RECLAIM=false or the parameter is set in the awsprov_config.json file, the AWS plug in does not send an instance termination request and allows the instance to be terminated by AWS.

  The default is AWS_SPOT_TERMINATE_ON_RECLAIM=false.
  Specify AWS_SPOT_TERMINATE_ON_RECLAIM=false has if you want AWS will reclaim the host. If the host is reclaimed in the middle of an instance hour, you will not be charged for this hour. For more information, see: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html#spot-pricing. The disadvantage is that the host is no longer managed by LSF. It is AWS's responsibility to terminate the host. LSF relies on AWS to perform the termination. If the termination is not done, extra charges can occur.

- LSF closes the reclaimed hosts and sends a requeue signal to the jobs on the host at a configurable time before the requested return time. After the hosts are drained of jobs, LSF notifies resource connector that the hosts are closed.

- **How LSF returns hosts to AWS**
  Amazon Web Services never reclaims an on-demand instance actively . Borrowed AWS instances are returned passively.

How LSF returns hosts to AWS

Amazon Web Services never reclaims an on-demand instance actively . Borrowed AWS instances are returned passively.

- It is idle for the time (configured by the LSB_RC_EXTERNAL_HOST_IDLE_TIME parameter in the lsf.conf file).
- A time-to-live period expires (configured by the LSB_RC_EXTERNAL_HOST_MAX_TTL parameter in lsf.conf file).
- If host factory notifies LSF that a host is ready to use, but the host does not join the cluster for 10 minutes. This value is hardcoded.

If the billingPeriod parameter is configured in the hostProviders.json file, the LSB_RC_EXTERNAL_HOST_IDLE_TIME value defined in the lsf.conf file is ignored. Resource connector uses the time the instance was launched to return the machine to the provider.

For example, if a host was launched at 10:00 AM, and billingPeriod is 60 minutes, and the host became idle at 10:55, it is returned before the next billing cycle that starts at 11 AM.

If set to 0 or not defined, resource connnector uses the value defined in the parameter LSB_RC_EXTERNAL_HOST_IDLE_TIME in the lsf.conf file.

If the host was previously in the cluster, LSF closes it (`closed_RC` status) and waits for any running jobs on the host to complete. After the host is drained of jobs, LSF notifies the resource connector. The resource connector deallocates the host by terminating the instance in AWS.

You cannot use the **badmin hopen** command to open a borrowed host in `closed_RC` status.

Updating LSF configuration for resource connector

Configure LSF to enable the resource connector.

About this task

Restart the LSF daemons on the management host for the changes to take effect.

Procedure

1. Log in to the LSF management host as root.
2. Configure DNS.
   LSF looks up host names and addresses for all communication between hosts borrowed from a resource provider and management host candidates. Make sure that your environment settings for IP address resolution work between the LSF management host and borrowed hosts that join the cluster.

   Check whether the management host candidates can ping the borrowed hosts by using the host name from the **hostname** command and vice versa. Also make sure that the borrowed hosts can ping themselves and each other with the reported host name. If the hosts can ping themselves and each other only by using IP address but not by using the host name, DNS settings need to be configured.

   If the borrowed host cannot join the cluster, check the VPN, firewall, or security group settings. You must open firewall rules for all LSF ports between the LSF management and borrowed server host IP ranges.

   If the LSF management host and the borrowed host reside in different host domains, specify the portion of both domain names in the LSF_STRIP_DOMAIN parameter in the $LSB_SHAREDIR/lsf.conf file that is accessible by the LSF management host, with both domain names separated by a colon (`:`).

3. Define the RC_DEMAND_POLICY parameter in the lsb.queues file to specify threshold conditions for triggering demand to borrow resources through resource connector for all the jobs in a queue.
   The RC_DEMAND_POLICY parameter has the following syntax:

   `RC_DEMAND_POLICY = THRESHOLD[ [` *num_pend_jobs*[,*duration*]] ...]

The demand policy defined by the RC_DEMAND_POLICY parameter can contain multiple conditions, in an OR relationship. A condition is defined as `[ num_pend_jobs[,duration]]`. The queue has more than the specified number of eligible pending jobs that are expected to run at least the specified duration in minutes. The *num_pend_jobs* option is required, and the duration is optional.

The default threshold is `THRESHOLD[[ 1,0]]`.

In the following example, LSF calculates demand if the queue has 5 or more pending jobs in past 10 minutes, or 1 or more pending jobs in past 60 minutes, or 100 or more pending jobs.

```
RC_DEMAND_POLICY = THRESHOLD[ [ 5, 10] [1, 60] [100] ]
```

As long as pending jobs at the queue meet at least one threshold condition, LSF expresses the demand to resource connector to trigger borrowing.

4. Define the MAX_SBD_CONNS parameter in the lsb.params file to set the maximum number of open file connections between the **mbatchd** and **sbatchd** daemons. As a best practice, set the value as `MAX_SBD_CONNS = 2 * maximum_number_of_hosts + 300`, where the *maximum_number_of_hosts* includes the number of active VMs created in the resource connector for future jobs.
5. Optional: Define optimizations to set rules after the calculation of demand to try to get better results and applied to the provisioning results. The Optimizations category is specified in the policy_config.json file. An example of optimization you can configure are allocation rules. The allocRules parameter is a list of allocation rule entries that specify how many hosts from a certain template are worth considering over another template. For more information about the allocRules parameter, see policy_config.json topic.
6. Optional: Define the account name to tag the borrowed hosts so they cannot be used by other groups, users, or jobs.
    - To define the account name at the queue or application level, define the RC_ACCOUNT parameter in the lsb.queues or lsb.applications file.
    - To set the project name as the default account name, enable `DEFAULT_RC_ACCOUNT_PER_PROJECT=Y` in the lsb.params file. This account name overrides the value of the RC_ACCOUNT parameter at the application and queue levels (lsb.applications and lsb.queues files).
    - To allow users to assign a specific account name at the job level, enable `ENABLE_RC_ACCOUNT_REQUEST_BY_USER=Y` in the lsb.params file. This allows users to use the `bsub -rcacct "rc_account_name"` command option to assign an account name. This account name overrides the account name that is set at the application and queue level, as well as the setting of the project name.
      This also allows you to use an **esub** script to set the LSB_SUB6_RC_ACCOUNT parameter to change the job level account name. The value of LSB_SUB6_RC_ACCOUNT overrides all other values of the account name, including the **bsub -rcacct** command option.

When a job is submitted to the queue or application, the host that the job borrows is tagged with the value of the RC_ACCOUNT parameter. Other applications or queues that have a different value for the RC_ACCOUNT parameter cannot use the borrowed host.

When the borrowed host joins the cluster, you can use the **lshosts -s** or **lshosts -l** command to view its RC_ACCOUNT value.

For AWS, edit the user_data.sh file to use the **lshosts** command to see the RC_ACCOUNT value.

The user_data.sh script is located in the <*LSF_TOP*>/<*LSF_VERSION*>/resource_connector/aws/scripts directory.

a. Make sure that the `LSF_TOP` variable points to the LSF_TOP directory for your cluster.

b. Make sure that the following lines are not commented out.

```
%EXPORT_USER_DATA%

logfile=/tmp/userscript.log
env > $logfile
if [ -n "${rc_account}" ]; then
sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap
${rc_account}*rc_account]\"/"
    $LSF_CONF_FILE
echo "update LSF_LOCAL_RESOURCES lsf.conf successfully,
    add [resourcemap ${rc_account}*rc_account]" >> $logfile
fi
```

What to do next

Restart the LSF daemons on the management host for the changes to take effect.

```
bctrld restart lim
bctrld restart res
badmin mbdrestart
```

- **Pre-provisioning and post-provisioning**
  Set up pre-provisioning in LSF resource connector to run commands before the resource instance joins the cluster. Configure post-provisioning scripts to run clean up commands after the instance is terminated, but before the host is removed from the cluster.
- **Configure resource provisioning policies**
  LSF resource connector provides built in policies for limiting the number of instances to be launched and the maximum number of instances to be created. The default plugin framework is a single python script that communicates via stdin and stdout in JSON data structures. LSF resource connector provides an interface for administrators to write their own resource policy plugin.
- **Use the LSF patch installer to update resource connector**
  The LSF patch installer (**patchinstall** command) doesn't support installing files under the *LSF_TOP*/conf directory. The **patchinstall** command operates only on files under the *LSF_TOP*/*<version>* directory.

- lsb.modules
- lsb.queues
- lsf.conf
- lsf.shared
- MAX_SBD_CONNS

Pre-provisioning and post-provisioning

Set up pre-provisioning in LSF resource connector to run commands before the resource instance joins the cluster. Configure post-provisioning scripts to run clean up commands after the instance is terminated, but before the host is removed from the cluster.

The pre- and post-provisioning script knows the instance ID, the instance name, and the instance IP address.

The pre-provisioning script runs on the management host right after the resource instance is created, but before the instance is marked as allocated to the LSF cluster, and before a job can start to run on it. Use the pre-provisioning script to run instance setup scripts (for example, network or user access configuration), run data transfer commands before the job starts on the instance, or add hosts to a specific group.

The post-provisioning script runs after the instance is terminated, but before it is relinquished to the provider. Use the post-provisioning script to run clean up tasks; for example, clean up job files or remove the host from the host cache file when the instance terminates.

You can specify a delayOnReturn attribute in your scripts that specifies the number of minutes that the resource connector waits before it returns the host in case the pre- or post-provisioning script fails. The default value is 20. For the scripts that are run successfully, resource connector does not apply the delay, even if delayOnReturn is set.

Enable pre- and post-provisioning

To enable pre- and post-provisioning scripts, set the following parameters in the hostProviders.json file:

preProvPath
: Resource connector runs the pre-provisioning script that is specified with absolute path after the instance is created and started successfully but before it is marked allocated to the LSF cluster.

postProvPath
: Resource connector runs the post-provisioning script that is specified with absolute path after the instance is terminated successfully but before it is removed from the LSF cluster.

provTimeOut
: This parameter is used to avoid the pre- or post-provisioning program from running for unlimited time. Specify a value in minutes.
If the program doesn't complete in the specified time, it is ended and reported as failed. You can disable pre- or post-provisioning by setting the provTimeOut value to 0.

The default value is 10 minutes. If the pre- or post-provisioning program doesn't return after 10 minutes, it ends.

Pre-provisioning and post-provisioning script output

Pre and post provision scripts are expected to return results to inform LSF is the hosts have been provisioned correctly or not. The output needs to look like the following example:

```
[{
"machines": [
          {
            "name": "instance name",
            "result": "succeed",
            "message": "User added successfully"
           },
          {
            "name": "ip-192-168-0-154.us-west-2.compute.internal",
            "result": "failed",
            "message": "Could not resolve host name",
            "delayOnReturn" : 30
          }
    ]
}]
```

Example input.json file

The input.json file is the data in json format which is sent by LSF to the user's provision scripts as input. The user defined provisioning scripts can use the input data of the AWS instances launched by LSF to do the necessary provisioning steps. The input.json file contains the following information:

```
[ {
"machines": [
          {
            "name": "ip-192-168-0-153.us-west-2.compute.internal",
            " publicIpAddress ": "55.66.xx.xx",
            " privateIpAddress ": "55.66.xx.xx",
            " machineId ": "i-19034xxxxx",
            "rcAccount": "project1" ,
            "providerName": "aws",
            "providerType": "awsProv",
            "templateName": "Template-VM-2"
          }
    ]
}]
```

Example

This example uses the jq package to parse the input.json from LSF for each of the automatically launched AWS instances. After parsing the instance details the user can customize code to configure any DNS settings or cleanup steps for each instance before passing a success or error result back to LSF in the specified format.

LSF reads the output to determine if the newly launched host was successfully provisioned.

```
#!/bin/sh
inputFile=$1
outputFile=$2
echo $inputFile $outputFile

count=$(cat $inputFile | jq '.machines[]' |grep 'name' | wc -l)

a=0
result="succeed"

while [ $a -lt $count ]
do
  hostName=$(cat $inputFile | jq '.machines['${a}'].name')
  privateIp=$(cat $inputFile | jq '.machines['${a}'].publicIpAddress')
  publicIp=$(cat $inputFile | jq '.machines['${a}'].privateIpAddress')
  instanceId=$(cat $inputFile | jq '.machines['${a}'].machineId')
  rcAccount=$(cat $inputFile | jq '.machines['${a}'].rcAccount')
  providerName=$(cat $inputFile | jq '.machines['${a}'].providerName')
  providerType=$(cat $inputFile | jq '.machines['${a}'].providerType')
  templateName=$(cat $inputFile | jq '.machines['${a}'].templateName')

  #add your custom code here for each machine in the request
  #write the output of each machine to the output json file

  sed -i '/]/i {\"name\": '${hostName}', \"result\": \"'${result}'\", \"message\":
\"'${message}'\" }' $outputFile
  a=`expr $a + 1`
done
```

Related reference

- hostProviders.json

Configure resource provisioning policies

LSF resource connector provides built in policies for limiting the number of instances to be launched and the maximum number of instances to be created. The default plugin framework is a single python script that communicates via stdin and stdout in JSON data structures. LSF resource connector provides an interface for administrators to write their own resource policy plugin.

Example policies

LSF resource connector provides an example policy, which you can configure in the example_policy_config.json file:

- Step index and step time determine how many instances to launch at a time. The StepValue parameter specifies step index and step time. For example, the `"StepValue": "10:10"` means that the resource connector launches no more than 10 instances every 10 minutes.
- Maximum number of instances (per account, per template, per provider) at any given time is specified by the MaxNumber parameter.

To enable this feature, rename the example_policy_config.json file under the LSF_TOP/conf/resource_connector directory to policy_config.json and set the cluster administrator as the file owner.

Resource policy plug in interface

You should not change the default resource policy plugin files (Main.py and Log.py in the *LSF_TOP*/*LSF_VERSION*/resource_connector/policy directory). Instead, you can create you own script or binary executable file and specify the path of that script in the UserDefinedScriptPath parameter in the policy_config.json. The default policy script provided by LSF runs your script and uses the demand calculated by your script to create hosts for the cluster.

Your script can have the following input for each provider, template and RC_ACCOUNT:

- The demand target requested by the cluster.
- The current allocation of hosts.
- The outstanding requests that have been made but not yet filled.
- The number of reclaimed hosts.

Related reference

- policy_config.json

Use the LSF patch installer to update resource connector

The LSF patch installer (**patchinstall** command) doesn't support installing files under the *LSF_TOP*/conf directory. The **patchinstall** command operates only on files under the *LSF_TOP*/<*version*> directory.

Update resource connector

When the patch installer installs new configuration files or other items under *LSF_TOP*/10.1.0/resource_connector/<*provider_name*>/conf, you must move them to the appropriate directory under *LSF_TOP*/conf/resource_connector/<*provider_name*>/conf. You must also change the ownership of any new files and directories to the cluster administrator. Everything under the *LSF_TOP*/conf/resource_connector directory must be owned by the cluster administrator.

For example, when the patch installer installs the following new configuration files for Amazon Web Services (AWS) under *LSF_TOP*/10.1.0/resource_connector/aws/conf/:

- awsprov_templates.json
- awsprov_config.json
- credentials

You must move these files to the *LSF_TOP*/conf/resource_connector/aws/conf directory and change the ownership of the aws directory and configuration files to the cluster administrator.

Roll back a resource connector patch

Follow these steps to roll back a patch:

1. Log on to the LSF management host as root
2. Set your environment:
   - For **csh** or **tcsh**: `% source LSF_TOP/conf/cshrc.lsf`
   - For **sh**, **ksh**, or **bash**: `$ . LSF_TOP/conf/profile.lsf`
3. Close all hosts and queues on your cluster:

   ```
   badmin hclose all
   badmin qinact all
   ```

4. Roll back the patch:

   ```
   ./patchinstall -r <patch>
   ```

5. Shut down the cluster:

   ```
   bctrld stop sbd all
   bctrld stop res all
   bctrld stop lim all
   ```

6. Restart the cluster:

   ```
   bctrld start lim all
   bctrld start res all
   bctrld start sbd all
   ```

7. Open hosts and queues again:

   ```
   badmin hopen all
   badmin qact all
   ```

Remember: Remove the most recent patch and return the cluster to the previous patch level. To roll back multiple versions, you must roll back one patch level at a time, in the reverse order of installation.
To roll back the same version of the patch applied on multiple platforms, you must roll back the same patch for multiple packages you applied, in the reverse order of installation, so that the resource connector common files are also rolled back to previous version.

For more information about patch rollback, see the -r option of the **patchinstall** command..

View information on the LSF resource connector

View information on the LSF resource connector by checking the status of the **ebrokerd** daemon, running the **badmin rc** subcommand, or by viewing the log files.

- **Checking the LSF resource connector status**
  To submit jobs that borrow resources from a resource provider, the LSF resource connector status must be enabled and the **ebrokerd** daemon process must be running.
- **Use the badmin command to view LSF resource connector information**
  Use the **badmin rc** subcommand to view information on the LSF resource connector.
- **Viewing LSF resource connector job events**
  The `JOB_FINISH2` LSF event contains details about LSF resource connector jobs. The LSF lsb.stream file can then capture and stream actions about the `JOB_FINISH2` event. Starting in Fix Pack 14, to provide more details in the `JOB_FINISH2` event logs, LSF includes the RC_ACCOUNT and VM_TYPE fields with the `JOB_FINISH2` event.
- **Logging and troubleshooting the LSF resource connector**
  Log files for the resource connector are located in the log directory that is defined by the LSF_LOGDIR parameter in the lsf.conf file.

Checking the LSF resource connector status

To submit jobs that borrow resources from a resource provider, the LSF resource connector status must be enabled and the **ebrokerd** daemon process must be running.

Procedure

On the LSF management host, verify that the **ebrokerd** daemon process is running after the resource connector is enabled.

```
# ps -ef | grep ebrokerd
```

Use the badmin command to view LSF resource connector information

Use the **badmin rc** subcommand to view information on the LSF resource connector.

Procedure

1. Use the **badmin rc view** command to view LSF resource connector information from the specified host providers.
   rc view [-c "instances | policies | templates ..."] [-p " provider ..."]

Use the -p option to specify the host provider from which to view information. Use a space to separate multiple host providers.

Use the -c option to specify whether you want to view information on instances, policies, or templates. Use a space to separate multiple types of information. By default, this command shows information on instances only.

2. Use the **badmin rc error** command to view LSF resource connector error messages from the specified host providers.
   To get the error messages, the third-party **mosquitto** message queue application must be running on the host.

   rc error [-t daysd | hoursh | minutesm] [-p " provider ..."]

   Use the -p option to specify the host provider from which to view information. Use a space to separate multiple host providers.

   Use the -t option to specify the earliest time from which to review the error messages.
   Note: When specifying days, badmin retrieves messages from this time at midnight. For example, when running `badmin rc error -t 1d`, **badmin** retrieves messages from today at midnight, and when running `badmin rc error -t 2d`, **badmin** retrieves messages from yesterday at midnight.

Viewing LSF resource connector job events

The **JOB_FINISH2** LSF event contains details about LSF resource connector jobs. The LSF lsb.stream file can then capture and stream actions about the **JOB_FINISH2** event. Starting in Fix Pack 14, to provide more details in the **JOB_FINISH2** event logs, LSF includes the RC_ACCOUNT and VM_TYPE fields with the **JOB_FINISH2** event.

Procedure

1. Enable JOB_FINISH2 LSF events:
   a. Log on to the host as the primary LSF administrator.
   b. Edit lsb.params configuration file to these include these configurations:
      - `ENABLE_EVENT_STREAM=Y`
      - `ALLOW_EVENT_TYPE=JOB_FINISH2`
   c. Save your changes to the lsb.params file and reconfigure the cluster for your changes to take effect:

      ```
      badmin reconfig
      ```

2. Starting in Fix Pack 14, to provide more details in the **JOB_FINISH2** event logs, the RC_ACCOUNT field is included with the **JOB_FINISH2** event. This field will reflect the RC_ACCOUNT value assigned to the job, and follow existing override policies if specified for the job, project, application, or queue level. (If not specified at any of these levels, then the RC_ACCOUNT value shows the RC_ACCOUNT field, with a value of **default**).
   a. The RC_ACCOUNT values specified for each of these levels are located in different configuration files; if required, set the value within the appropriate parameter and configuration file:
      - For RC_ACCOUNT at the job level, see the ENABLE_RC_ACCOUNT_REQUEST_BY_USER parameter in the lsb.params file.

- For RC_ACCOUNT at the project level, see the DEFAULT_RC_ACCOUNT_PER_PROJECT parameter in the lsb.params file.
- For RC_ACCOUNT at the application level, see the RC_ACCOUNT parameter in the lsb.applications file.
- For RC_ACCOUNT at the queue level, see the RC_ACCOUNT parameter in the lsb.queues file.

b. If you made changes to any of the configuration files, save your changes to the files and reconfigure the cluster for your changes to take effect:

```
badmin reconfig
```

Tip: If you do not set these resource connector account parameters, or if the LSF resource connector is not enabled, the RC_ACCOUNT field cannot be retrieved, but the **JOB_FINISH2** event still shows the RC_ACCOUNT field as a value of **default**.

3. Starting in Fix Pack 14, to provide the **JOB_FINISH2** event logs more detail, include the VM_TYPE field with the **JOB_FINISH2** event. To show the VM_TYPE field:
   a. Edit the lsf.shared file:
      i. Define the **vm_type** resource in the lsf.shared file. For example, to define a String resource called vm_type, specify:

```
Begin Resource
RESOURCENAME  TYPE    INTERVAL INCREASING  DESCRIPTION
# Keywords
...
vm_type       String  ()       ()          (The type of VM)
...
End Resource
```

      ii. Reconfigure LIM and restart the mbatchd daemon for your lsf.shared file change to take effect:

```
lsadmin reconfig
badmin mbdrestart
```

   b. Edit the user_data.sh script file for each cloud provider:
      i. LIM on the LSF management host retrieves the VM_TYPE field. A resource connector host them sends this information to the management host by way of the *LSF_TOP*/10.1/resource_connector/*provider*/scripts/user_data.sh script **vm_type** setting in the LSF_LOCAL_RESOURCES parameter within the lsf.conf file.
      Enable the user_data.sh script of each cloud provider to modify the value of the LSF_LOCAL_RESOURCES parameter in the lsf.conf file, by adding the following lines to the file:

```
if [ -n "${vm_type}" ]; then
  sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap
${vm_type}*vm_type]\"/" $LSF_CONF_FILE
fi
```

      Note that the method of getting this VM_TYPE field varies depending on the cloud provider. For example:

      Example IBM® Cloud user_data.sh script

```
vm_type=$(dmidecode |grep Manufacturer|grep IBM| cut -d ':' -f
4)
if [ -n "$vm_type" ]; then
    sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap
$vm_type*vm_type]\"/" $LSF_CONF_FILE
    echo "Update LSF_LOCAL_RESOURCES in $LSF_CONF_FILE
```

```
successfully, add [resourcemap ${vm_type}*vm_type]" >>
$logfile
else
    echo "Can not get instance VM type" >> $logfile
fi
```

Example Amazon Web Services (AWS) user_data.sh script

```
vm_type=$(curl http://169.254.169.254/latest/meta-
data/instance-type)
#Note that this is cloud provider specific
if [ -n "$vm_type" ]; then
sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap
$vm_type*vm_type]\"/" $LSF_CONF_FILE
else
echo "vm_type doesn't exist in envrionment variable" >>
$logfile
fi
```

Example Google Cloud Platform user_data.sh script

```
vm_type=$(dmidecode |grep Manufacturer|grep IBM| cut -d ':' -f
4)
if [ -n "$vm_type" ]; then
    sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap
$vm_type*vm_type]\"/" $LSF_CONF_FILE
    echo "Update LSF_LOCAL_RESOURCES in $LSF_CONF_FILE
successfully, add [resourcemap ${vm_type}*vm_type]" >>
$logfile
else
    echo "Can not get instance VM type" >> $logfile
fi
```

Example Microsoft Azure CycleCloud user_data.sh script

```
vm_type=$(curl -H Metadata:true
"http://169.254.169.254/metadata/instance/compute/vmSize?api-
version=2018-10-01&format=text")
#Note that this is cloud provider specific
if [ -n "$vm_type" ]; then
sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap
$vm_type*vm_type]\"/" $LSF_CONF_FILE
else
echo "vm_type doesn't exist in envrionment variable" >>
$logfile
fi
```

Example Microsoft Azure user_data.sh script

```
vm_type=$(curl -H Metadata:true
"http://169.254.169.254/metadata/instance/compute/vmSize?api-
version=2018-10-01&format=text")
#Note that this is cloud provider specific
if [ -n "$vm_type" ]; then
sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resourcemap
$vm_type*vm_type]\"/" $LSF_CONF_FILE
else
echo "vm_type doesn't exist in envrionment variable" >>
$logfile
fi
```

Results

The following **JOB_FINISH2** event log shows the addition of the rc_account and vm_type fields:

```
"JOB_FINISH2" "10.11" 1666989002 2321 45 "userId" "0" "userName" "root"
"numProcessors" "1"
"options" "33816578" "jStatus" "64" "submitTime" "1666988304" "termTime" "0"
"startTime" "1666988401"
"endTime" "1666989002" "queue" "normal" "resReq" "profile==cx2_4x8" "fromHost"
"rhel7x-mgmt-1" "cwd"
"/share/10.1/lsf_rc" "jobFile" "1666988304.2321" "numExHosts" "1" "execHosts"
"icgen2host-10-240-0-10"
"slotUsages" "1" "cpuTime" "0.441864" "command" "sleep 600" "ru_utime" "0.229670"
"ru_stime" "0.212194"
"ru_maxrss" "3072" "ru_nswap" "0" "projectName" "default" "exitStatus" "0"
"maxNumProcessors" "1"
"exitInfo" "0" "chargedSAAP" "/root" "numhRusages" "0" "runtime" "601" "maxMem"
"3072" "avgMem" "2048"
"effectiveResReq" "select[(profile == cx2_4x8 ) && (type == any)] order[r15s:pg] "
"subcwd"
"/share/10.1/lsf_rc" "serial_job_energy" "0.000000" "numAllocSlots" "1"
"allocSlots"
"icgen2host-10-240-0-10" "ineligiblePendingTime" "-1" "options2" "1040"
"hostFactor" "12.500000"
"cpuPeak" "0.000000" "cpuEfficiency" "0.000000" "memEfficiency" "0.000000"
"rc_account" "default" "vm_type" "cx2-4x8"
```

Related reference

- lsb.params
- lsb.applications
- lsb.queues
- lsf.shared
- lsf.conf

Logging and troubleshooting the LSF resource connector

Log files for the resource connector are located in the log directory that is defined by the LSF_LOGDIR parameter in the lsf.conf file.

Log files for LSF

To change the log level or log classes for LSF, update the following parameters in the lsf.conf file:

- LSF_LOG_MASK
- LSB_DEBUG_MBD
- LSB_DEBUG_EBROKERD

For example, the following parameters set the log level to LOG_INFO, and the debugging log class for the **mbatchd** and **ebrokerd** daemons to LC2_RC:

```
LSF_LOG_MASK=LOG_INFO
LSB_DEBUG_MBD="LC2_RC"
LSB_DEBUG_EBROKERD="LC2_RC"
```

Log files for the resource connector

To change the log level for the resource connecto r for AWS, update the LogLevel parameter in the <*LSF_TOP*>/conf/resource_connector/aws/conf/awsprov_config.json resource provider configuration file.

For example, set the log level to `INFO`:

```
{
 "LogLevel": "INFO",
}
```

Persistent files for the resource connector

The resource connector saves some state information for synchronization with LSF after failover or restart. This information is saved in persistent files, which are in the <*LSB_SHAREDIR*>/<*cluster_name*>/resource_connector/ directory.

Validating the federated account script

When using AWS_CREDENTIALS_SCRIPT for federated accounts, LSF executes the script specified by the user set as the value of this parameter. The temporary credential for LSF to use to launch and manage the AWS EC2 instance is stored internally under <*LSF_TOP*>/work/<clusterName>/resource_connector/aws_federatedUser_credentials.

1. Ensure resource connector logging is enabled and check for errors in the logs.
2. Execute the script as the primary LSF administrator on the LSF management host to ensure the standard output format matches the one specified in section Configuring AWS access with federated accounts.
3. If there are errors when executing the script, those need to be fixed to make sure it can create a temporary credential and output to stdout in the correct format to be used by LSF.
4. After correcting the script, if there is a stale aws_federateduser_credentials file, remove it so that a new file can be generated immediately. Otherwise, LSF will generate one automatically every 30 minutes.

LSF resource connector configuration reference

Reference for configuring LSF resource connector.

- **lsb.applications**
  Configure the operation of LSF resource connector in the lsb.applications file.
- **lsb.queues**
  Configure the operation of LSF resource connector in the lsb.queues file.
- **lsf.conf**
  Enable and configure the operation of LSF resource connector in the lsf.conf file.
- **hostProviders.json**
  The hostProviders.json file configures which resource providers LSF resource connector can use.
- **policy_config.json**
  The policy_config.json file configures custom policies for resource providers for LSF resource connector. The resource policy plug-in reads this file.
- **awsprov_config.json**
  The awsprov_config.json file contains administrative settings for the resource connector.

- **awsprov_templates.json**
  The awsprov_templates.json file defines the mapping between LSF resource demand requests and AWS instances.

lsb.applications

Configure the operation of LSF resource connector in the lsb.applications file.

- **RC_ACCOUNT**
  Assigns an account name (tag) to hosts borrowed through LSF resource connector, so that they cannot be used by other user groups, users, or jobs.
- **RC_RECLAIM_ACTION**
  Controls how the LSF resource connector takes action on jobs that are running on a host when that host is reclaimed.

RC_ACCOUNT

Assigns an account name (tag) to hosts borrowed through LSF resource connector, so that they cannot be used by other user groups, users, or jobs.

Syntax

`RC_ACCOUNT=`*account_name*

Description

When a job is submitted to an application profile with the RC_ACCOUNT parameter specified, hosts borrowed to run the job are tagged with the value of the RC_ACCOUNT parameter. The borrowed host cannot be used by other applications that have a different value for the RC_ACCOUNT parameter (or that don't have the RC_ACCOUNT parameter defined at all).

After the borrowed host joins the cluster, use the **lshosts -s** command to view the value of the RC_ACCOUNT parameter for the host.

Example

```
RC_ACCOUNT=project1
```

Default

No account defined for the application profile

RC_RECLAIM_ACTION

Controls how the LSF resource connector takes action on jobs that are running on a host when that host is reclaimed.

Syntax

```
RC_RECLAIM_ACTION = REQUEUE | TERMINATE
```

Description

Specify one of the following actions:

- REQUEUE: Requeue the jobs that are running on a host that is reclaimed.
- TERMINATE: Terminate the jobs that are running on a host that is reclaimed.

Default

TERMINATE for interactive jobs.

REQUEUE for all other jobs.

lsb.queues

Configure the operation of LSF resource connector in the lsb.queues file.

- **RC_ACCOUNT**
  Assigns an account name (tag) to hosts borrowed through LSF resource connector, so that they cannot be used by other user groups, users, or jobs.
- **RC_DEMAND_POLICY**
  Defines threshold conditions for the determination of whether demand is triggered to borrow resources through resource connector for all the jobs in a queue. As long as pending jobs at the queue meet at least one threshold condition, LSF expresses the demand to resource connector to trigger borrowing.
- **RC_HOSTS**
  Enables LSF resource connector to borrow specific host types from a resource provider.

RC_ACCOUNT

Assigns an account name (tag) to hosts borrowed through LSF resource connector, so that they cannot be used by other user groups, users, or jobs.

Syntax

RC_ACCOUNT=*account_name*

Description

When a job is submitted to a queue with the RC_ACCOUNT parameter specified, hosts borrowed to run the job are tagged with the value of the RC_ACCOUNT parameter. The borrowed host cannot be used by other queues that have a different value for the RC_ACCOUNT parameter (or that don't have the RC_ACCOUNT parameter defined).

After the borrowed host joins the cluster, use the **lshosts -s** command to view the value of the RC_ACCOUNT parameter for the host.

Example

```
RC_ACCOUNT=project1
```

Default

The string "default" - Meaning, no account is defined for the queue.

RC_DEMAND_POLICY

Defines threshold conditions for the determination of whether demand is triggered to borrow resources through resource connector for all the jobs in a queue. As long as pending jobs at the queue meet at least one threshold condition, LSF expresses the demand to resource connector to trigger borrowing.

Syntax

RC_DEMAND_POLICY = THRESHOLD[[*num_pend_jobs*[,*duration*]] ...]

Description

The demand policy defined by the RC_DEMAND_POLICY parameter can contain multiple conditions, in an **OR** relationship. A condition is defined as [*num_pend_jobs*[,*duration*]]. The queue has more than the specified number of eligible pending jobs that are expected to run at least the specified duration in minutes. The *num_pend_jobs* option is required, and the duration is optional. The default duration is 0 minutes.

LSF considers eligible pending jobs for the policy. An ineligible pending job (for example, a job dependency is not satisfied yet) keeps pending even thought hosts are available. The policy counts a job for eligibility no matter how many tasks or slots the job requires. Each job element is counted as a job. Pending demand for a resizable job is not counted, though LSF can allocate borrowed resources to the resizable job.

LSF evaluates the policies at each demand calculation cycle, and accumulates duration if the *num_pend_jobs* option is satisfied. The **mbschd** daemon resets the duration of the condition when it restarts or if the condition has not been evaluated in the past two minutes. For example, if no pending jobs are in the cluster, for two minutes, **mbschd** stops evaluating them.

Example

In the following example, LSF calculates demand if the queue has five or more pending jobs in past ten minutes, or one or more pending jobs in past 60 minutes, or 100 or more pending jobs.

```
RC_DEMAND_POLICY = THRESHOLD[ [ 5, 10] [1, 60] [100] ]
```

Default

Not defined for the queue

RC_HOSTS

Enables LSF resource connector to borrow specific host types from a resource provider.

Syntax

`RC_HOSTS=`*`string`*

`RC_HOSTS = none` | `all` | *host_type* [*host_type* ...]

Description

The *host_type* flag is a Boolean resource that is a member of the list of host resources that are defined in the LSB_RC_EXTERNAL_HOST_FLAG parameter in the lsf.conf file.

If the RC_HOSTS parameter is not defined in the queue, its default value is none. Borrowing is disabled for any queue that explicitly defines RC_HOSTS=none, even if the LSB_RC_EXTERNAL_HOST_FLAG parameter is defined in the lsf.conf file.

If the RC_HOSTS parameter is not defined in any queue, borrowing cannot happen for any job.

Note: The HOSTS parameter in the lsb.queues file and the **bsub -m** option do not apply to hosts that are managed through the resource connector. To specify the resource connector host types that can be used by a queue, you must specify the RC_HOSTS parameter in that queue.

Example

```
RC_HOSTS=awshost
```

Default

none - host borrowing from resource providers is disabled, and no borrowed hosts can be used by the queue.

lsf.conf

Enable and configure the operation of LSF resource connector in the lsf.conf file.

- **EBROKERD_HOST_CLEAN_DELAY**
  For LSF resource connector. Specifies the delay, in minutes, after which the **ebrokerd** daemon removes information about relinquished or reclaimed hosts. This parameter allows the **bhosts -rc** and **bhosts -**

rconly command options to get LSF resource connector provider host information for some time after they are no longer provisioned.

- **LSB_RC_DEFAULT_HOST_TYPE**
  LSF resource connector default host type.
- **LSB_RC_EXTERNAL_HOST_FLAG**
  Setting the LSB_RC_EXTERNAL_HOST_FLAG parameter enables the LSF resource connector feature.
- **LSB_RC_EXTERNAL_HOST_IDLE_TIME**
  For LSF resource connector. If no jobs are running on a resource provider instance for the specified number of minutes, LSF relinquishes the instances.
- **LSB_RC_EXTERNAL_HOST_MAX_TTL**
  For LSF resource connector. Maximum time-to-live for a resource provider instance. If an instance is in the cluster for this number of minutes, LSF closes it and its status goes to `closed_RC`).
- **LSB_RC_MQTT_ERROR_LIMIT**
  For LSF resource connector. The maximum number of API error messages that are stored in Mosquitto per host provider.
- **LSF_MQ_BROKER_HOSTS**
  For LSF resource connector. Enables support for the **bhosts -rc** and **bhosts -rconly** command options to get LSF resource connector provider host information
- **LSB_RC_QUERY_INTERVAL**
  For LSF resource connector. The interval in seconds that the resource connector checks host status and asynchronous requests from a resource provider.
- **LSB_RC_REQUEUE_BUFFER**
  For LSF resource connector. The number of seconds before the expiration of the reclaim grace period before which LSF starts to send re-queue signals to running jobs on reclaimed hosts.
- **LSB_RC_TEMPLATE_REQUEST_DELAY**
  For LSF resource connector. The amount of time that LSF waits before repeating a request for a template, in minutes, if the **ebrokerd** daemon encountered certain provider errors in a previous request.
- **LSB_RC_UPDATE_INTERVAL**
  For LSF resource connector. Configures how often LSF calculates demand for pending jobs and publishes this demand to the **ebrokerd** daemon.
- **MQTT_BROKER_HOST**
  For LSF resource connector. If you do not use the MQTT message broker daemon (**mosquitto**) that is provided with LSF, specifies the host name that **mosquitto** runs on. The MQTT message broker receives provider host information from **ebrokerd** and publishes that information for the **bhosts -rc** and **bhosts -rconly** command options to display.
- **MQTT_BROKER_PORT**
  For LSF resource connector. If you do not use the MQTT message broker daemon (**mosquitto**) that is provided with LSF, specifies the TCP port for the MQTT message broker daemon (**mosquitto**). The MQTT message broker receives provider host information from **ebrokerd** and publishes that information for the **bhosts -rc** and **bhosts -rconly** command options to display.

EBROKERD_HOST_CLEAN_DELAY

For LSF resource connector. Specifies the delay, in minutes, after which the **ebrokerd** daemon removes information about relinquished or reclaimed hosts. This parameter allows the **bhosts -rc** and **bhosts -rconly** command options to get LSF resource connector provider host information for some time after they are no longer provisioned.

Syntax

EBROKERD_HOST_CLEAN_DELAY=*minutes*

Description

After configuring this parameter, run **badmin mbdrestart** on the management host to restart **ebrokerd**.

After configuring this parameter, run **bctrld restart lim** on the management host and, if any of the specified host names is not the management host name, **lsadmin limrestart** *host_name*.

Example

```
EBROKERD_HOST_CLEAN_DELAY=30
```

Default

60 minutes

See also

- LSF_MQ_BROKER_HOSTS
- MQTT_BROKER_HOST
- MQTT_BROKER_PORT
- **bhosts -rc** and **bhosts -rconly**

LSB_RC_DEFAULT_HOST_TYPE

LSF resource connector default host type.

Syntax

LSB_RC_DEFAULT_HOST_TYPE=*string*

Description

Specifies the default host type to use for a template if the `type` attribute is not defined on a template in the template configuration files.

Example

```
LSB_RC_DEFAULT_HOST_TYPE=X86_64
```

Default

X86_64

LSB_RC_EXTERNAL_HOST_FLAG

Setting the LSB_RC_EXTERNAL_HOST_FLAG parameter enables the LSF resource connector feature.

Syntax

LSB_RC_EXTERNAL_HOST_FLAG="*string* ..."

Description

Specify a list of Boolean resource names that identify host providers that are available for borrowing. Any hosts or instances that provide a resource from the list are initially closed by LSF at startup. Hosts and instances are only opened when the resource connector informs LSF that the host was successfully allocated or the instance is launched.

Run the **badmin mbdrestart** command for this parameter to take effect.

Example

```
LSB_RC_EXTERNAL_HOST_FLAG="awshost googlehost azurehost"
```

Default

Not defined

LSB_RC_EXTERNAL_HOST_IDLE_TIME

For LSF resource connector. If no jobs are running on a resource provider instance for the specified number of minutes, LSF relinquishes the instances.

Syntax

LSB_RC_EXTERNAL_HOST_IDLE_TIME=*minutes*

Description

If the LSB_RC_EXTERNAL_HOST_IDLE_TIME parameter is set to 0, the policy is disabled and the resource provider instance is never shut down for lack of jobs.

Example

```
LSB_RC_EXTERNAL_HOST_IDLE_TIME=30
```

Default

60 minutes

LSB_RC_EXTERNAL_HOST_MAX_TTL

For LSF resource connector. Maximum time-to-live for a resource provider instance. If an instance is in the cluster for this number of minutes, LSF closes it and its status goes to `closed_RC`).

Syntax

`LSB_RC_EXTERNAL_HOST_MAX_TTL=`*`minutes`*

Description

If the resource provider is still active after the specified number of minutes, LSF changes the host status to `closed_RC`, which prevents the host from accepting additional workload. If the LSB_RC_EXTERNAL_HOST_MAX_TTL parameter is set to 0, the policy is disabled. The cloud resource is returned and terminated if the workload that associated with the host is done.

You cannot use the **badmin hopen** command to open a borrowed host in `closed_RC` status.

Example

```
LSB_RC_EXTERNAL_HOST_MAX_TTL=30
```

Default

0 minutes (disabled)

LSB_RC_MQTT_ERROR_LIMIT

For LSF resource connector. The maximum number of API error messages that are stored in Mosquitto per host provider.

Syntax

`LSB_RC_MQTT_ERROR_LIMIT=`*`integer`*

Description

This parameter specifies the maximum number of messages that the **badmin rc error** command displays for each host provider.

Run **badmin mbdrestart** for any change to take effect.

Example

```
LSB_RC_MQTT_ERROR_LIMIT=20
```

Default

10

LSF_MQ_BROKER_HOSTS

For LSF resource connector. Enables support for the **bhosts -rc** and **bhosts -rconly** command options to get LSF resource connector provider host information

Syntax

LSF_MQ_BROKER_HOSTS=*host_name*...

Description

If you use the MQTT message broker daemon (**mosquitto**) that is provided with LSF resource connector, specifies the host name where LIM starts that **mosquitto** daemon. The MQTT message broker receives provider host information from **ebrokerd** and publishes that information for the **bhosts -rc** and **bhosts -rconly** command options to display. If the LSF_MQ_BROKER_HOSTS parameter is not specified, the MQTT broker is not started, and the **bhosts -rc** and **bhosts -rconly** command options fail.

Specify hosts that run the MQTT message broker daemon (**mosquitto**). When LIM starts, it checks the LSF_MQ_BROKER_HOSTS parameter, and if a host is on the list, LIM starts the **mosquitto** daemon on this host. For failover to work properly for displaying resource connector hosts, define all the management candidate hosts in this list.

LIM manages the message broker lifecycle, terminating and restarting it when necessary.

Failure of the **mosquitto** daemon is recorded in the LIM log.

The *host_name* is one of the existing LSF cluster hosts. After configuring this parameter, run the **bctrld restart lim** command on the management host. If the specified host name is not the management host, run the **bctrld restart lim** *host_name* command.

To verify that the **mosquitto** daemon is up and running, use the `ps -ef | grep mosquitto` command.

Example

```
LSF_MQ_BROKER_HOSTS=hosta
```

Default

Not defined.

See also

- EBROKERD_HOST_CLEAN_DELAY
- MQTT_BROKER_HOST
- MQTT_BROKER_PORT
- **bhosts -rc** and **bhosts -rconly**

LSB_RC_QUERY_INTERVAL

For LSF resource connector. The interval in seconds that the resource connector checks host status and asynchronous requests from a resource provider.

Syntax

`LSB_RC_QUERY_INTERVAL=`*`seconds`*

Description

Run **badmin mbdrestart** for any change to take effect.

Example

```
LSB_RC_QUERY_INTERVAL=60
```

Default

30 seconds

LSB_RC_REQUEUE_BUFFER

For LSF resource connector. The number of seconds before the expiration of the reclaim grace period before which LSF starts to send re-queue signals to running jobs on reclaimed hosts.

Syntax

`LSB_RC_REQUEUE_BUFFER=`*`seconds`*

Description

The minimum value is 1, which means that LSF sends the re-queue signal 1 second before the hosts are reclaimed and the daemons are shut down. Use a low value for jobs on reclaimed hosts to run longer.

Note: A low value increases the risk that the re-queue operation does not complete before the host is reclaimed and the daemons are shut down. If a host is reclaimed before the re-queue is complete, the jobs on

the host go to `UNKNWN` status until the host is returned to LSF.

Example

```
LSB_RC_REQUEUE_BUFFER=20
```

Default

30 seconds

LSB_RC_TEMPLATE_REQUEST_DELAY

For LSF resource connector. The amount of time that LSF waits before repeating a request for a template, in minutes, if the **ebrokerd** daemon encountered certain provider errors in a previous request.

Syntax

`LSB_RC_TEMPLATE_REQUEST_DELAY=`*`integer`*

Description

This parameter takes effect if the **ebrokerd** daemon encounters one of the following provider errors when requesting a template from the host provider API:

- **`InsufficientAddressCapacity`**: Not enough available addresses to satisfy the minimum request.
- **`InsufficientCapacity`**: Not enough capacity to satisfy the import instance request.
- **`InsufficientInstanceCapacity`**: Not enough instance capacity available to satisfy the request.
- **`InsufficientHostCapacity`**: Not enough capacity to satisfy the dedicated host request.
- **`InsufficientReservedInstanceCapacity`**: Not enough available reserved instances to satisfy the minimum request.

As of Fix Pack 14, this parameter takes effect if the VPC Gen 2 (IBM Cloud Gen 2) API return instance status fails with the following capacity status reason codes:

- **`cannot_start_capacity`**
- **`cannot_start_compute`**
- **`cannot_start_ip_address`**
- **`cannot_start_network`**
- **`cannot_start_placement_group`**
- **`cannot_start_storage`**

Run **badmin mbdrestart** for any change to take effect.

Example

```
LSB_RC_TEMPLATE_REQUEST_DELAY=20
```

Default

10

LSB_RC_UPDATE_INTERVAL

For LSF resource connector. Configures how often LSF calculates demand for pending jobs and publishes this demand to the **ebrokerd** daemon.

Syntax

LSB_RC_UPDATE_INTERVAL=*seconds*

Description

This parameter updates the demand calculation according to the specified interval instead of calculating demand every scheduler cycle to avoid performance impact.

Example

```
LSB_RC_UPDATE_INTERVAL=20
```

Default

30 seconds

MQTT_BROKER_HOST

For LSF resource connector. If you do not use the MQTT message broker daemon (**mosquitto**) that is provided with LSF, specifies the host name that **mosquitto** runs on. The MQTT message broker receives provider host information from **ebrokerd** and publishes that information for the **bhosts -rc** and **bhosts -rconly** command options to display.

Syntax

MQTT_BROKER_HOST=*host_name*

Description

If you use an existing MQTT message broker, use the MQTT_BROKER_HOST parameter to specify the host that runs the **mosquitto** daemon. You can also optionally specify a port for the MQTT broker with the MQTT_BROKER_PORT parameter.

After configuring this parameter, run **badmin mbdrestart** on the management host to restart **ebrokerd**.

Example

```
MQTT_BROKER_HOST=hosta
```

Default

Not defined.

See also

- EBROKERD_HOST_CLEAN_DELAY
- LSF_MQ_BROKER_HOSTS
- MQTT_BROKER_PORT
- **bhosts -rc** and **bhosts -rconly**

MQTT_BROKER_PORT

For LSF resource connector. If you do not use the MQTT message broker daemon (**mosquitto**) that is provided with LSF, specifies the TCP port for the MQTT message broker daemon (**mosquitto**). The MQTT message broker receives provider host information from **ebrokerd** and publishes that information for the **bhosts -rc** and **bhosts -rconly** command options to display.

Syntax

MQTT_BROKER_PORT=*port_number*

Description

After configuring this parameter, run **badmin mbdrestart** on the management host to restart **ebrokerd**.

If you use the **mosquitto** daemon from the LSF distribution, and you define a port that is not the default 1883, you also must define the port in $LSF_ENVDIR/mosquitto.conf. LIM then starts the **mosquitto** daemon with the non-default port and **ebrokerd** connects the MQTT broker via the port defined in the lsf.conf file.

If you are using an external **mosquitto** daemon, the port number you define in MQTT_BROKER_PORT must be the same port in the external mosquitto.conf file.

Example

```
MQTT_BROKER_PORT=4477
```

Default

1883.

See also

- EBROKERD_HOST_CLEAN_DELAY
- LSF_MQ_BROKER_HOSTS

- MQTT_BROKER_HOST
- **bhosts -rc** and **bhosts -rconly**

hostProviders.json

The hostProviders.json file configures which resource providers LSF resource connector can use.

The default location for the hostProviders.json file is <*LSF_TOP*>/conf/resource_connector/hostProviders.json.

The hostProviders.json file contains a JSON list of named providers. For example, for AWS, the type is `awsProv`.

You can specify an absolute path for configuration and script files. The default assumes a path relative <*LSF_TOP*>/conf/resource_connector for configuration files and <*LSF_TOP*>/<*LSF_VERSION*>/resource_connector/ for scripts.

Changes to the hostProviders.json configuration file requires a reconfiguration of LSF by running the command **badmin mbdrestart** on the LSF management host.

You can also configure multiple instances of the the same provider, with different properties for different purposes.

For hosts to operate in the same cluster, all host providers must have the same LSF administrator. The LSF administrator must have access to the directories specified by confPath and scriptPath.

You cannot define more than one pre- or post-provisioning script.

The pre- or post-provisioning script is local to each provider defined in the providers list. If you want all providers to run the same script, you need to specify the same path inside each provider.

Each host provider must have a unique name. If two different host providers use the same name, LSF logs a warning and ignores one of the entries.

Parameters

providers
: A list of resource providers.

name
: The name of the resource provider.

type
: The type of the resource provider.

confPath
: Path to the resource provider configuration directory. Full and relative paths are supported.
The default confPathis relative to *LSF_TOP*/conf/resource_connector.

scriptPath
: Path to the resource provider script directory. Full and relative paths are supported.
The default scriptPath is relative to *LSF_TOP*/*LSF_VERSION*/resource_connector.

  Important: Use the default scriptPath parameter value, which refers to the LSF_TOP/*LSF_VERSION*/resource_connector/*provider_name* file path. If you put the script files in the LSF_TOP/conf/resource_connector directory or use a custom value for scriptPath, LSF resource

connector patch files do not automatically update the script and library files. This is because the LSF **patchinstall** command only updates files under the LSF_TOP/*LSF_VERSION* directory.

scriptOptions

For AWS resource provider. By default, LSF assumes that the management host has direct access to AWS. If your site's security policy requires the connection to AWS to be made through a proxy server, the scriptOptions attribute enables LSF to connect to AWS instances through the specified proxy host name or IP address, and proxy server port.

LSF sets environment variable SCRIPT_OPTIONS when launching the scripts.

preProvPath and postProvPath

Optional. The resource connector runs the pre-provisioning script (specified with absolute path, such as /usr/share/lsf/scripts/pre_provision.sh) after the instance is created and started successfully, but before it is marked allocated to the LSF cluster.
The resource connector runs the post-provisioning script (specified with absolute path, such as /usr/share/lsf/scripts/post_provision.sh) after the instance is terminated successfully, but before it is removed from the LSF cluster.

Here is an example input JSON file (note that the name value is the name assigned by the provider, not the hostname):

```
{
"machines": [
          {
            "name": "name_provided_by_provider",
            "publicIpAddress": "55.66.xx.xx",
            "privateIpAddress": "10.0.xx.xx",
            "machineId ": "i-19034xxxxx",
            "rcAccount": "project1" ,
            "providerName": "aws",
            "providerType": "awsProv",
            "templateName": "Template-VM-2"
          },
          {
            "name": "name_provided_by_provider",
            "machineId": "idxxxxxxxxxxxxx",
            "publicIpAddress": "55.66.xx.xx",
            "privateIpAddress": "10.0.xx.xx",
            "machineId ": "i-19034xxxxx",
            "rcAccount" : "project2",
            "providerName": "aws",
            "providerType": "awsProv",
            "templateName": "Template-VM-2"

          }
    ]
}
```

Next, the output JSON file is passed. The scripts run for each request and a request can have multiple machines in it.

Here is an example expected return for the scripts. Note that delayOnReturn value specifies the number of minutes that the resource connector will wait before returning the host if a fails. The default wait time is 20 minutes. The delayOnReturn parameter is not used if the scripts execute successfully:

```
{
"machines": [
          {
            "name": "name_provided_by_provider1",
            "machineId ": "i-19034xxxxx",
```

```
            "result": "succeed",
            "message": ""
          },
          {
            "name": "name_provided_by_provider",
            "machineId ": "i-19034xxxxx",
            "result": "failed",
            "message": "could not resolve host name”,
            "delayOnReturn": 20
          }
    ]
}
```

The script exit code is an integer:

- A `0` exit code indicates a successful completion with no errors and produces an output JSON file with content.
- A `-1` exit code indicates a failed script or one that runs with fatal errors and produces an empty output JSON file.

provTimeOut

Optional. This parameter is used to avoid the pre- or post-provisioning program from running for unlimited time. Specify a value in minutes.
If the program does not complete in the specified time, it is ended and reported as failed. Setting the provTimeOut value to `0` disables script timeout.

The default value is ten minutes. If the pre-provisioning or post-provisioning program does not return after ten minutes, it ends.

provHostTimeOut

Optional. The timeout value for each host provider. The default value is ten minutes.

If a resource connector host does not join the LSF cluster within this timeout value, then the host is relinquished.

billingPeriod

Ignore the LSB_RC_EXTERNAL_HOST_IDLE_TIME value defined in the lsf.conf file and use the time the instance was launched to return the machine to the provider.
For example, if a host was launched at 10:00 AM, and billingPeriod is 60 minutes, and the host became idle at 10:55, it is returned before the next billing cycle that starts at 11 AM.

If set to 0 or not defined, the resource connector uses the value defined in the parameter LSB_RC_EXTERNAL_HOST_IDLE_TIME in the lsf.conf file.

The default billing period is 0.

Example

```
{
    "providers":[
        {
            "name": "aws1",
            "type": "awsProv",
            "confPath": "resource_connector/aws",
            "scriptPath": "resource_connector/aws",
            "scriptOptions": "-Dhttps.proxyHost=10.115.206.146 -
Dhttps.proxyPort=8888",
            "billingPeriod":  "60",
```

```
            "preProvPath":  "/usr/share/lsf/scripts/pre_provision.sh",
            "postProvPath": "/usr/share/lsf/scripts/post_provision.sh",
            "provTimeOut" : 10
        },
        {
            "name": "aws2",
            "type": "awsProv",
            "confPath": "resource_connector/aws",
            "scriptPath": "resource_connector/aws",
            "billingPeriod":  "30",
            "preProvPath":  "/usr/share/lsf/scripts/pre_provision.sh",
            "postProvPath": "/usr/share/lsf/scripts/post_provision.sh",
            "provTimeOut" : 20
        }
    ]
}
```

policy_config.json

The policy_config.json file configures custom policies for resource providers for LSF resource connector. The resource policy plug-in reads this file.

The default location for the file is

***<LSF_TOP>*/conf/resource_connector/policy_config.json**

The policy_config.json file contains a JSON list of named policies and optimizations. Policies are rules set during the calculation of demand. Optimizations are rules set after the calculation of demand to try to get better results. Each policy contains a name, a consumer, a maximum number of instances that can be launched for the consumer, and maximum number of instances that can be launched in a specified period.

Parameters

UserDefinedScriptPath
: Optional. Specify the full path to your own resource provider policy script. Your custom policy script runs after the default plug-in runs with the same input JSON file, and the demand that is calculated by your script is used. Demand that is calculated by the default plug-in is ignored. If the UserDefinedScriptPath is defined and it fails to run, the demand is 0, which means no demand.
The following example defines the path to the script userscript.py:

  ```
  "UserDefinedScriptPath" : "/usr/share/lsf/10.1/scripts/userscript.py"
  ```

Policies
: Optional. A list of policies that apply on the demand calculation. If the policies are not defined, the demand that is calculated by resource connector is used.

  Name
  : Required. The name of the policy. You can define multiple policies in the list. Each policy must have a unique name.

  Consumer
  : Optional. The following consumer attributes are supported:

    rcAccount

A list of accounts that can borrow hosts through LSF resource connector. Supported values are `all` or any valid account name that is defined in the RC_ACCOUNT tag in the lsb.queues file. If this attribute is not defined, the default value is `all`.

templateName
: A list of template names. Supported values are `all` or any valid template name. If this attribute is not defined, the default value is `all`.

provider
: A list of resource provider names. Supported values are `all` or any valid provider name. If this attribute is not defined, the default value is `all`.wh

perRcAccount
: Used with the MaxNumber parameter in the Policies parameter of this policy_config.json file to define the maximum number of instances per resource connector account. Specify a list of accounts that can borrow hosts through LSF resource connector. The value cannot be set to `all`.
  If the perRcAccount value is not defined, the default value will be `default`.

perTemplateName
: Used with the MaxNumber parameter in the Policies parameter of this policy_config.json file to define the maximum number of instances per resource connector template. Specify a list of template names. Supported values are any valid template names. The value cannot be set to `all`.

perProvider
: Used with the MaxNumber parameter in the Policies parameter of this policy_config.json file to define the maximum number of instances per resource provider. Specify a list of resource provider names. Supported values are any valid provider names. The value cannot be set to `all`.

If a consumer is not defined, the following attributes apply to all providers, templates, accounts defined in the cluster:

MaxNumber
: Optional. The maximum number of instances a user can create or launch for the consumer. When specifying the MaxNumber parameter in the Policies parameter, you can also specify values for the perRcAccount, perTemplateName, and perProvider consumer attributes. Additionally, for the perRcAccount attribute, the value cannot be set to `all`; if the perRcAccount is not defined, the default value will be `default`.

StepValue
: Optional. The StepValue parameter has two values, which are separated by a colon (`:`). The *step index* is the maximum number of instances that can be launched at a time for the defined consumer. The *step time* controls how fast the cluster grows. The step time specifies how long the plug-in waits before it launches another set of instances that are specified by the step value. If the consumer is not defined, the parameter applies cluster wide.
  For example, if step value is defined as 5 and step time is defined as 10 (`"StepValue": "5:10"`) and a request comes in for 20 instances, 5 instances are launched in the first 10 minutes, 5 more in next 10 minutes until the demand is met or the maximum number instances that are specified by the MaxNumber parameter are launched.

  The default for step index to launch all the instances at the same time.

  Default Value for step time is 10 minutes. The default value that is applied only if a step value is defined but a step time is not defined.

Optimizations

Optional. Rules set after the calculation of demand to try to get better results. Optimizations to apply to the provisioning results.

allocRules
: Optional. A list of allocation rule entries that specify how many hosts from a certain template are worth considering over another template. For every allocation rule entry, each of the following allocation rule attributes are mandatory:

  fromTemplate
  : The following fromTemplate attributes are supported:

    provider
    : Specifies a resource provider name. Supported value is any valid provider name.

    templateName
    : Specifies a template name. The value must be a valid templateName under the provider.

    factor
    : Number of hosts that are being replaced.

  toTemplate
  : The following toTemplate attributes are supported:

    provider
    : Specifies a resource provider name. Supported value is any valid provider name.

    templateName
    : Specifies a template name. The value must be a valid templateName under the provider.

    factor
    : Number of hosts to be replaced.

  For any attribute that is not defined or any errors in a configuration, the allocation rule entry is ignored and the next entry is evaluated.

  Tip: Configuring this allocRules parameter, compliments configuring the RC_DEMAND_POLICY parameter in the lsf.queues file. The RC_DEMAND_POLICY parameter enables LSF to gather more pending jobs before provisioning. As a result, the jobs can be optimized with more information, but result in delayed run time. For more information about the RC_DEMAND_POLICY parameter, see RC_DEMAND_POLICY topic.
  Consider the following optimizations configuration example:

```
"Optimizations" : {
    "allocRules" : [
        {
            "fromTemplate": {
                "provider" : "aws",
                "templateName" : "aws_template1",
                "factor" : 4
            },
            "toTemplate" : {
                "provider" : "aws",
                "templateName": "aws_template3",
                "factor" : 1
            }
        },
        {
            "fromTemplate": {
                "provider" : "aws",
                "templateName" : "aws_template1",
```

```
                "factor" : 2
            },
            "toTemplate" : {
                "provider" : "aws",
                "templateName": "aws_template2",
                "factor" : 1
            }
        }
    ]
  }
```

The following command displays the policies and optimizations configured: **badmin rc view -c policies**
Here is the display output:

```
Optimizations
        4 hosts (aws:aws_template1) replaced by 1 hosts
(aws:aws_template3)
        2 hosts (aws:aws_template1) replaced by 1 hosts
(aws:aws_template2)
```

For this example, to ensure the rules works for most cases, a proper template priority needs to be set. In this example configuration, **aws_template1** is a replacement template so it needs to have a highest template priority than other templates. The **aws_template3** is the first rule to consider for the replacement, so the template priority needs to be lower than **aws_template1**, but higher than **aws_template2**. Therefore, the template priority can have the following settings:

Template name	Priority
aws_template1	10
aws_template3	9
aws_template2	8

Example

```
{
  "UserDefinedScriptPath" : "/usr/share/lsf/10.1/scripts/userscript.py",
  "Policies":
  [
    {
      "Name": "Policy1",
      "Consumer":
      {
        "rcAccount": ["all"],
        "templateName": ["all"],
        "provider": ["all"]
      },
      "MaxNumber": "100",
      "StepValue": "5:10"
    },
    {
      "Name": "Policy2",
      "Consumer":
       {
        "rcAccount": ["default", "project1"],
        "templateName": ["aws_template1"],
        "provider": ["aws"]
       },
      "MaxNumber": "50",
```

```
      "StepValue": "5:20"
    },
    {
      "Name": "Policy3",
      "Consumer":
      {
        "perRcAccount": ["project1","project2"],
        "perTemplateName": ["ibm_template1","ibm_template2"],
        "perProvider": ["ibmcloudhpc"]
      },
      "MaxNumber": "100",
      "StepValue": "5:10"
    }
  ],
"Optimizations" : {
    "allocRules" : [
        {
            "fromTemplate": {
                "provider" : "aws",
                "templateName" : "aws_template1",
                "factor" : 4
            },
            "toTemplate" : {
                "provider" : "aws",
                "templateName": "aws_template3",
                "factor" : 1
            }
        },
        {
            "fromTemplate": {
                "provider" : "aws",
                "templateName" : "aws_template1",
                "factor" : 2
            },
            "toTemplate" : {
                "provider" : "aws",
                "templateName": "aws_template2",
                "factor" : 1
            }
        }
    ]
  }
}
```

To view the policies and optimizations that are configured for your resource connector policy (policy_config.json) file, run the command:

```
badmin rc view -c policies
```

An example of the output:

```
Policies
    Name: Policy1
        Consumer
            rcAccount: ["all"]
            templateName: ["all"]
            provider: ["all"]
        MaxNumber: 100
        StepValue: 5:10
    Name: Policy2
        Consumer
            rcAccount: ["default", "project1"]
            templateName: ["aws_template1"]
```

```
            provider: ["aws"]
        MaxNumber: 50
        StepValue: 5:20
    Name: Policy3
        Consumer
            perRcAccount: ["project1", "project2"]
            perTemplateName: ["ibm_template1", "ibm_template2"]
            provider: ["ibmcloudhpc"]
        MaxNumber: 50
        StepValue: 5:20
Optimizations
        4 hosts (aws:aws_template1) replaced by 1 hosts (aws:aws_template3)
        2 hosts (aws:aws_template1) replaced by 1 hosts (aws:aws_template2)
```

In addition, consider the RC_DEMAND_POLICY parameter in the lsf.queues file contained the following example configuration:

```
RC_DEMAND_POLICY = THRESHOLD[[2,10] [4,5]]
```

This configuration sets optimization to first apply four hosts with **aws_template1** VMs with one host with **aws_template3** VM. Then, considers applying two hosts with **aws_template1** VMs with one hosts **aws_template2** VM. The RC_DEMAND_POLICY defined the buffer time for four jobs is shorter than two jobs, so that when the demand trigger by four or more jobs, then the first optimization rule is applied; otherwise, the second optimization rule is applied.

Related reference

- RC_DEMAND_POLICY

awsprov_config.json

The awsprov_config.json file contains administrative settings for the resource connector.

For example, you can configure the awsprov_config.json file to start remote AWS services, such as creating virtual instances.

The default location for the file is

<*LSF_TOP*>/conf/resource_connector/aws/conf/awsprov_config.json

Parameters

The file is a JSON format tuple that contains the following parameter fields:

LogLevel
: The log level for the host provider. Use the following log levels:

  - TRACE
  - INFO (the default level)
  - DEBUG
  - WARN
  - ERROR
  - FATAL

AWS_CREDENTIAL_FILE
: Defines the path to the AWS authentication file for using AWS IAM credentials. The file contains the user access key and user access secret key. The resource connector uses the credentials in this file to authenticate LSF users with the AWS identity service. These users must have administrative privileges on AWS to perform required AWS functions (start and stop instances, and so on).
If the LSF management host and the resource connector are deployed in an AWS instance, do not include the AWS_CREDENTIAL_FILEparameter. The AWS API credentials are retrieved from the AWS environment automatically.

AWS_CREDENTIAL_SCRIPT
: Defines the path to the AWS credential script for using federated accounts with AWS
You cannot define both the parameters AWS_CREDENTIAL_FILE and AWS_CREDENTIAL_SCRIPT.

  Define AWS_CREDENTIAL_SCRIPT only when federated accounts are used with AWS. When AWS_CREDENTIAL_SCRIPT is defined, the AWS_CREDENTIAL_FILE is ignored.

  For example:

  ```
  AWS_CREDENTIAL_SCRIPT=/shared/dir/generateCredentials.py
  ```

  When neither AWS_CREDENTIAL_FILE nor AWS_CREDENTIAL_SCRIPT parameters are defined, resource connector attempts to retrieve API credentials from an instance profile defined in , under the assumption that it is running in an AWS EC2 instance.

AWS_KEY_FILE
: Optional. The path to the AWS key pair file. The key pair name is used to log in to instances with **ssh**. If the AWS_KEY_FILE parameter is not specified, no key file is defined.

AWS_REGION
: AWS region name that is attached to instances and user account.
Specifies the region in which the user and the key file are created and where the instances are provisioned. The key file is specific to the region.

AWS_SPOT_TERMINATE_ON_RECLAIM
: Optional. Process requests for terminating Amazon EC2 Spot instances that are planned to be reclaimed by AWS.
If set to `true`, the AWS plug-in sends an instance termination request to AWS when notified by LSF that the reclaimed Spot instance has been closed and the affected jobs are re-queued.

  Valid values are `true` and `false`. The default value is `false`.

awsprov_templates.json

The awsprov_templates.json file defines the mapping between LSF resource demand requests and AWS instances.

The template represents a set of hosts that share some attributes, such as the number of CPUs, the amount of available memory, the installed software stack, operating system.

LSF requests resources from the resource connector by specifying the number of instances of a particular template that it requires to satisfy its demand. The resource connector uses the definitions in this file to map this demand into a set of allocation requests in AWS.

The default location for the file is <*LSF_TOP*>/conf/resource_connector/aws/conf/awsprov_templates.json.

Description

LSF requests resources from the resource connector by specifying the number of instances of a particular template that it requires to satisfy its demand. The resource connector uses the definitions in this file to map this demand into a set of allocation requests in AWS.

Important: When you define templates, you must make sure that the attribute definitions that are presented to LSF exactly match the attributes that are provided by AWS. If, for example, the attribute definition specifies hosts with `ncpus=4`, but the actual hosts that are returned by AWS report `ncpus=2`, the demand calculation in LSF is not accurate.

Parameters

The file contains a JSON-defined list called `templates`. Each template in the list is an object that contains the following parameters:

templateId
: The unique template name. The templateId cannot contain underscores (_).

maxNumber
: That maximum number of instances to provide. Set the MaxNumber to an appropriate value according to the instance quota of the LSF project.
As of Fix Pack 14, to support AWS EC2 Fleet templates, the MaxNumber is a multiplier of the ncpu value, not a direct number of instances. For EC2 Fleet, maxNumber multiplied by ncpus is the maximum slots that EC2 Fleet template can get and can be provisioned in this template. For example, if the maxNumber is **`5`** and **`ncpus`** is **`2`**, then the maximum slots for the fleet request will be **`10`**.

attributes
: A list of attributes that represent the hosts in the template from the LSF point of view. LSF attempts to place its pending workload on hosts that match these attributes to calculate how many instances of each template to request.
You can define any arbitrary string resource in the lsf.shared file and use that as an attribute in the awsprov_templates.json file. You can then use that attribute in a **bsub** select string (for example, `bsub -R "select[zone == us_east_2a"]`). If `zone == us_east_2a` is selected at job submission, hosts are created from the template that defines the `zone` attribute to `us_east_2a`.

  To submit a job with a specific template name or a template string attribute, you must define that string resource in the lsf.shared and in the user_data.sh script for that resource to be added to the lsf.conf file on the server host that is created from the template.

  The user_data.sh script is located in the *<LSF_TOP>*/*<LSF_VERSION>*/resource_connector/aws/scripts directory.

  Each attribute string in the list has the following format:

  `"attribute_name": ["attribute_type", "attribute_value"]`

  attribute_name
  : An LSF resource name, for example, `type` or `ncores`.
  The attribute name must either be a built-in resource (such `r15s` or `type`), or defined in the `Resource` section in the lsf.shared file on the LSF management host.

  attribute_type
  : Can be either `Boolean`, `String`, or `Numeric` and must correspond to the corresponding resource definition in the lsf.shared file.

attribute_value
: The value of the resource that is provided by hosts. For `Boolean` resources, use 1 to define the presence of the resource and 0 to define its absence. For `Numeric` resources, specify a range that uses `[min:max]`.

Depending on your cloud provider, various attributes are supported in the template.

The following attributes have default values if they are not defined:

`type`
: The default value is given by the setting of the LSB_RC_DEFAULT_HOST_TYPE in the lsf.conf file. The default value of LSB_RC_DEFAULT_HOST_TYPE is `X86_64`.

`ncpus`
: Default value is 1.

Take note of these attributes:

gpuextend
: Optional. A string that represents the GPU topology on the template host.
This attribute value is in the following format:

  `"key1=value1;key2=value2;..."`

  The following keys are supported in this attribute:

  ngpus
  : Total number of GPUs. This must be defined either as a key in gpuextend or defined as a separate attribute. If it is defined in both places, the key value in gpuextend takes precedence.

  nnumas
  : Total number of NUMA nodes. The default value is 1.

  gbrand
  : The GPU brand. This value is case sensitive, and supports NVIDIA GPUs. For a list of GPU brands and models, run the **nvidia-sml -L** command.
  For example, for `Tesla K80`, the GPU brand is **`Tesla`**.

  gmodel
  : The GPU model. This value is case sensitive, and supports NVIDIA GPUs. For a list of GPU brands and models, run the **nvidia-sml -L** command.
  For example, for `Tesla K80`, the GPU model is **`K80`**.

  gmem
  : The total GPU memory, in MB.

  nvlink
  : Specifies whether the GPU supports NVLink. Valid values (case insensitive) are `y`, `n`, `yes`, `no`.

imageId
: The ID of the Amazon Machine Image (AMI) that has LSF preinstalled on it. This AMI is used to launch virtual instances.

subnetId
: The subnet name (virtual private cloud) used to launch virtual instances. Use the subnet through which the instance can communicate with the LSF cluster.
For AWS spot instances only, you can specify more than one subnet, separated by commas. For example:

```
"subnetId": "subnet-bc219af5, subnet-ac819ch2"
```

More than one subnet are not supported for on-demand AWS instances.

vmType
: The machine type of the AWS instance you want to create. The `vmType` that is configured in each template must correctly represent the template attributes presented to LSF from AWS.
For AWS Spot instances only, you can specify multiple machine types, separated by commas. For example:

```
"vmType": "c4.large, m4.large"
```

Multiple machine types are not supported for on-demand AWS instances.

launchTemplateId
: Optional. The ID of the launch template. Specify a string between 1 and 255 characters in length.

launchTemplateVersion
: Optional. The version number of the launch template to select when launching instances. Specify the version number of the launch template or one of the following keywords:

  $Latest
  : Amazon EC2 Auto Scaling selects the latest version of the launch template when launching instances.

  $Default
  : Amazon EC2 Auto Scaling selects the default version of the launch template when launching instances. This is the default value of the launchTemplateVersion attribute.

fleetRole
: For Spot Instance templates. Specifies the role that grants the permission to bid on, launch, and terminate spot fleet instances on behalf of the user.

spotPrice
: For Spot Instance templates. Specifies the bid price for the instance. The Spot instance is launched when the Spot price of the instance is below the bid specified in the `spotPrice` attribute.
The spotPrice attribute is used in determining if the request is a spot request or an on-demand request. If you set the spotPrice attribute with a positive number, the AWS plug-in considers this request as a spot request. If the attributes fleetRole or allocationStrategy are defined, but the spotPrice is not defined, the request is considered an on-demand request.

  If an on-demand request is initiated using a template with multiple vmType or subnetId values, the request fails.

allocationStrategy
: Optional. For spot instance templates. The allocation strategy for your spot fleet determines how it fulfills your Spot fleet request from the possible spot instance pools that are represented by its launch specifications. You can specify the following allocation strategies in your spot fleet request:

  CapacityOptimized
  : The Spot instances come from the pools with optimal capacity for the number of instances that are launching. This is the default strategy.

  LowestPrice
  : The Spot instances come from the pool with the lowest price.

  Diversified
  : The Spot instances are distributed across all pools.

keyName
: Optional. The name of the key-pair file that is used by **ssh** to log in the launched instance. If no value is specified then the instance is launched with no key.
If the proper permission is not available, then the value is ignored and the AWS log will be informed.

interfaceType
: Optional. The type of network interface to attach to the instance.
Specify `efa` to attach an Elastic Fabric Adapter (EFA) interface to an instance. You can only specify an EFA network interface for supported AMI or instance types. For more details on supported AMI or instance types for EFA interfaces, refer to the Amazon Web Services website (https://aws.amazon.com/).

Note: If you defined `efa` in the AWS launch template, you cannot remove or unset the `efa` interface value by using this AWS launch template
The default value is `interface`, which specifies that a non-EFA network interface is attached to the template.

securityGroupIds
: Optional. A list of strings for AWS security groups that are applied to instances. If you don't specify securityGroupIds, AWS uses the default group.

instanceProfile
: Specifies an AWS IAM instance profile to assign to the requested instance. Jobs running in that instance can use the instance profile credentials to access other AWS resources.
The instance profile can be specified by one of the following methods:

- Short name; for example, `MyProfile`.
  Valid characters for the instance profile name are uppercase and lowercase alphanumeric characters and any of the following ASCII characters: equal sign (=), comma (,), period (.), at sign (@), minus sign (–).
- AWS Amazon Resource Name (ARN); for example, `arn:aws:iam::<account number>:instance-profile/LSFRole`.
  The colon character (:) cannot appear in the short name or path. The string `arn:` at the beginning of the profile reference determines whether the reference is an ARN or a short name.
  Note: In this context “IAM Role” is essentially equivalent to “Instance Profile”.

instanceTags
: Optional. A string that represents a list of keys and their values. These key-value pairs are used to tag the instance, by using Amazon instance tagging feature. If an instance is launched that uses `TemplateA`, it is tagged with value of the instanceTags attribute defined in `TemplateA`.
If instanceTags is not specified, LSF still tags the newly launched instances with the following key-value pair:

  `InstanceID = <ID of the instance created>`

  The instanceTags attribute also tags EBS volumes with the same tag as the instance. EBS volumes are persistent block storage volumes used with an EC2 instance. EBS volumes are expensive, so you can use the instance ID that tags the volumes for the accounting purposes.
  Note: The tags cannot start with the string `aws:`. This prefix is reserved for internal AWS tags. AWS gives an error if an instance or EBS volume is tagged with a keyword starting with `aws:`. Resource connector removes and ignores user-defined tags that start with `aws:`.

ebsOptimized
: An Amazon EBS-optimized instance provides additional, dedicated capacity for Amazon EBS input and output. This optimization improves performance for your EBS volumes by minimizing contention between Amazon EBS input and output and other traffic from your instance.

  See the AWS documentation for more information about Amazon EBS-optimized I\instances.

  Use the ebsOptimized attribute in your AWS template to create instances with Amazon EBS optimization enabled.

Valid values are Boolean `true` and `false`. The default is `false`. You must specify the proper vmType that supports EBS optimization.

The EBS optimization service is expensive and only available on high-end instance types. If the instance type does not support the attribute, an error messages is issued. Resource connector suspends the provider for 10 minutes. You can change the vmType in the template and restart **ebrokerd**.

priority
: By default, LSF sorts candidate template hosts by template name. However, an administrator might want to sort them by priority, so LSF favors one template to the other. The priority attribute has been added. LSF will use higher priority templates first (for example, less expensive templates should be assigned higher priorities).
The default value of priority is `0`, which means the lowest priority. If template hosts have the same priority, LSF sorts them by template name.

placementGroupName
: Optional. The name of the placement group that the instances are launched to. The group must exist on your AWS account. Successfully launching the instances into a placement group has the following requirements:

- A placement group can't span multiple availability zones.
- The name that you specify for a placement group must be unique within your AWS account.
- The instance type that is defined in the template must be supported by the placement group created.
- Terminate all the instances in the placement group before the placement group is deleted.

To use the tenancy attribute, if you do not have a placement group in your AWS account, you must at least insert a placement group with a blank name inside quotation marks. If you have a placement group, specify the placement group name inside the quotation marks. For example, `"placementGroupName": "",` or `"placementGroupName": "hostgroupA",`.

tenancy
: Requires placementGroupName in the template. The values for tenancy can be **`default`**, **`dedicated`**, and **`host`**. However, LSF currently only supports **`default`** and **`dedicated`**.
See the AWS documentation for more information about Amazon Dedicated Instances (https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/dedicated-instance.html)

userData
: Optional. A string that represents a list of keys and their values. The string has the following format:

`<key1>=<value1>;<key2>=<value2>; ...`

key
: The key name of the userData, such as "`packages`, `volume`, `zone`, or `templateName`.

value
: A comma-separated list of userData values, for example, `package1, package2`.

Each key is converted to uppercase by the resource connector and exported as an environment variable with the specified value inside the instance (and is accessible by the user script). After userData is defined, it is divided into keys and values and exported to the instance's environment variables.

For example, if the userData parameter is defined as `packages=M,N;logfile=X`, the following environment variables are exported inside the instance at start time:

```
PACKAGES=M,N
LOGFILE=X
```

These variables can be read by the user_data.sh script in the instance as the keys `PACKAGES` and `ZONE`.

ec2FleetConfig
: Required for AWS EC2 Fleet instances (offered as of Fix Pack 14). An absolute or relative path to the EC2 Fleet configuration file (for example, to a ec2-fleet-config.json file). For relative path, the path must be relative to LSF_TOP/conf/resource_connector/aws/conf directory.

onDemandTargetCapacityRatio
: Optional for AWS EC2 Fleet instances (offered as of Fix Pack 14). Defines how on-demand and spot instances are distributed among the **`TotalTargetCapacity`** in each EC2 Fleet request.
Specify a value that is a positive float number between 0.0 and 1.0. The value represents the ratio between OnDemandTargetCapacity to TotalTargetCapacity. To request pure on-demand or pure spot instances, you can set this ratio to 1 or 0. If not defined, it follows the DefaultTargetCapacityType in the ec2FleetConfig file.

Example overall awsprov_templates.json file

```
{
    "Templates":
    [
        {
        "templateId": "TemplateA",
        "attributes":
        {
           "type": ["String", "X86_64"],
           "ncpus": ["Numeric", "4"],
           "mem": ["Numeric", "480"],
           "maxmem": ["Numeric", "512"],
           "awshost": ["Boolean", "1"],
           "zone": ["String", "us_east_2a"]
           "pricing": ["String", "ondemand"],
           "computeUnit": ["String", "encl_3"]
        },
        "imageId": "ami-27b1",
        "subnetId": "subnet-b5738",
        "vmType": "t2.micro",
        "maxNumber": "1",
        "keyName": "LSF_Key",
        "securityGroupIds": ["sg-72314"],
        "placementGroupName": "lsfgrp1",
        "instanceTags": "group=LSF;project=Amazon",
        "userData": "pricing=ondemand;zone=us_west_2b"
        }
    ]
}
```

The example defines a template that is named `TemplateA`. LSF attempts to place any pending workload on hypothetical hosts of type **`X86_64`** with `ncpus=4` and `mem>480` MB. If LSF successfully places some of its pending workload on *N* number of hosts, it requests *N* instances of `TemplateA` to the resource connector.

If demand is generated for this template, the connector logic attempts to allocate *N* hosts with the configured image and `vmType` (instance type) in AWS. If it succeeds to obtain any instances, even if there are fewer than requested, the resource connector informs LSF that it can use the instances.

In this example, the template also defines the `awshost` resource. You can make sure that your jobs generate demand for AWS resources by using `'select[awshost]'` in your LSF job submission resource requirement strings.

The `zone` attribute is an example string resource that is defined in the lsf.shared file. If the `zone` attribute is specified, an instance is created in the specified zone.

The user script scripts/user_data.sh is included in the instance and run during instance startup.

The following script is an example of scripts/user_data.sh in a CentOS 6 image. It reads environment variables and updates the lsf.conf file on the instance to define the new `zone` attribute for that machine.

```
#!/bin/bash
LSF_TOP=/usr/share/lsf
LSF_CONF_FILE=$LSF_TOP/conf/lsf.conf

# run user script to enable selecting template based on zone
%EXPORT_USER_DATA%

logfile=/tmp/userscript.log
env > $logfile
if [ -n "${zone}" ]; then
sed -i "s/\(LSF_LOCAL_RESOURCES=.*\)\"/\1 [resource ${zone}]
       [resourcemap ${zone}*zone]\"/" $LSF_CONF_FILE
echo "update LSF_LOCAL_RESOURCES lsf.conf successfully,
       add [resource ${zone}] [resourcemap ${zone}*zone]" >> $logfile
else
echo "zone doesn't exist in environnment variable" >> $logfile
fi
```

Example awsprov_templates.json file for on-demand instances

The following template creates on-demand instances:

```
{
    "templates": [
         {
            "templateId": "templateA",
            "attributes": {
                "type": ["String", "X86_64"],
                "ncores": ["Numeric", "1"],
                "ncpus": ["Numeric", "1"],
                "nram": ["Numeric", "512"],
                "awshost1": ["Boolean", "1"],
                "zone": ["String", "us_west_2a"],
                "pricing": ["String", "ondemand"]
            },
            "imageId": "ami-8914cbe9",
            "subnetId": "subnet-cc0248ba",
            "vmType": "t2.nano",
            "keyName": "martin",
            "securityGroupIds": ["sg-b35182ca"],
            "instanceTags": "Name=aws1-vm-1-from-cluster-aws1",
            "userData": "zone=us_west_2a;pricing=ondemand"
        }
    ]
}
```

Example awsprov_templates.json file for spot instances

The following template creates spot instances:

```
{
    "templates": [
         {
```

```
            "templateId": "templateB",
            "attributes": {
                "type": ["String", "X86_64"],
                "ncores": ["Numeric", "1"],
                "ncpus": ["Numeric", "1"],
                "nram": ["Numeric", "512"],
                "awshost1": ["Boolean", "1"],
                "zone": ["String", "us_west_2b"],
                "pricing": ["String", "spot"]
            },
            "imageId": "ami-8914cbe9",
            "subnetId": "subnet-7c0dfb27,subnet-12286475,subnet-cc0248ba",
             "keyName": "martin",
            "vmType": "c4.xlarge, m4.large",
            "fleetRole": "arn:aws:iam::700071821657:role/EC2-Spot-Fleet-role",
            "securityGroupIds": ["sg-b35182ca"],
            "spotPrice": "0.1",
            "allocationStrategy":"diversified",
            "instanceTags": "Name=aws1-vm-3-spot-aws1",
            "userData": "zone=us_west_2b;pricing=spot"
        }

    ]
}
```

Example awsprov_templates.json file for EBS-optimized instances

The following template creates EBS-optimized instances. Note that the vmType is `m4.large`, which supports EBS optimization. EBS optimization is enabled by default in the `m4.large` instance type.

```
{
    "templates": [
        {
            "templateId": "Template-VM-1",
            "maxNumber": 4,
            "attributes": {
                "type": ["String", "X86_64"],
                "ncores": ["Numeric", "1"],
                "ncpus": ["Numeric", "1"],
                "mem": ["Numeric", "1024"],
                "awshost1": ["Boolean", "1"]
            },
            "imageId": "ami-40a8cb20",
           "vmType": "m4.large",
            "subnetId": "subnet-cc0248ba",
            "keyName": "martin",
            "securityGroupIds": ["sg-b35182ca"],
            "instanceTags" : "group=project1",
            "ebsOptimized" : true,
            "userData": "zone=us_west_2a"
        }
    ]
}
```

Example awsprov_templates.json file for Amazon EC2 Fleet instances

As of Fix Pack 14, the LSF resource connector for Amazon Web Services (AWS) uses an Amazon EC2 Fleet API to create multiple (that is, a fleet of) instances. EC2 Fleet is an AWS feature that extends the existing spot fleet, which gives you a unique ability to create fleets of EC2 instances composed of a combination of EC2 on-demand, reserved, and spot instances, by using a single API.
The following template creates Amazon EC2 Fleet instances

```
{
    "templates": [
        {
            "templateId": "fleet-lsf-template-1",
            "maxNumber": 5,
            "attributes": {
                "type": ["String", "X86_64"],
                "ncores": ["Numeric", "1"],
                "ncpus": ["Numeric", "2"],
                "mem": ["Numeric", "512"],
                "awshost": ["Boolean", "1"]
            },
            "priority": "121",
            "ec2FleetConfig": "ec2-fleet-config.json",
            "onDemandTargetCapacityRatio":"0.5",
            "instanceTags": "Name=fleet-lsf-template-1"
        }
    ]
}
```