IBM Spectrum LSF
Version 10 Release 1

Configuring IBM Spectrum LSF Resource Connector for AWS

Contents

Chapter 1. About this document 1

Chapter 2. LSF resource connector overview 3

Chapter 3. Building a cloud image 5
Preparing Amazon Web Services components 5
Launching the Amazon Web Services EC2 instance 6
Installing an LSF slave host on the AWS EC2 instance 6

Chapter 4. Enabling LSF resource connector for Amazon Web Services (AWS) 9
The aws_enable.sh script 9
Executing AWS enablement script for LSF 10
Completing the enabling of Resource Connector for AWS 10
Configuring user scripts to register AWS hosts 11

Chapter 5. Configuring Bursting Behavior 13
Configuring a threshold 13
Providing specific policy configurations 14
Controlling reclaim behavior 14

Chapter 6. Configuring Amazon Web Services for LSF resource connector 17
Assign exclusive resources to a template 17
Configuring AWS access with federated accounts 18
Configure multiple resource providers 19
Pre-provisioning and post-provisioning 20
Define resource provisioning policies 22
Use AWS Spot instances 22
Configuring AWS Spot instances 24

Chapter 7. Submit jobs to resource connector 27
Submitting jobs to AWS 27
How LSF returns hosts to AWS 28

Chapter 8. Logging and troubleshooting 31

Chapter 9. Configuration reference 33
lsb.applications 33
RC_ACCOUNT 33
lsb.queues 33
RC_ACCOUNT 33
RC_DEMAND_POLICY 34
RC_HOSTS 34
lsf.conf 35
LSB_RC_DEFAULT_HOST_TYPE 35
LSB_RC_EXTERNAL_HOST_FLAG 35
LSB_RC_EXTERNAL_HOST_IDLE_TIME 36
LSB_RC_EXTERNAL_HOST_MAX_TTL 36
LSB_RC_QUERY_INTERVAL 37
LSB_RC_UPDATE_INTERVAL 37

LSF_RC_AUDIT_LOG....37
RC_MAX_AUDIT_LOG_KEEP_TIME....38
RC_MAX_AUDIT_LOG_SIZE....38
hostProviders.json....38
policy_config.json....40
awsprov_config.json....42
awsprov_templates.json....43

Chapter 1. About this document

The purpose of the Configuring IBM Spectrum LSF Resource Connector for AWS guide is to describe how to configure IBM Spectrum LSF Resource Connector to cloud-burst to Amazon Web Services (AWS) 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 including Amazon Web Services (AWS) which will be the focus of this document. Resource Connector currently only supports Linux. Some understanding of AWS is required in order to complete the tasks of the document and it is recommend going over the prerequisites as a starting point.

Terms used in this document

The following terms are used throughout this document 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.

Prerequisites

- **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 #unique_2**)

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 slave 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

Chapter 2. IBM Spectrum LSF resource connector overview

The resource connector for IBM Spectrum LSF feature enables LSF clusters to borrow resources from supported resource providers.

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 Resource Connector

The following are requirements for configuring IBM Spectrum LSF Resource Connector.

- You must have root access to the LSF master host.
- Both the LSF master 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 Master host

The following are Java requirements for the IBM Spectrum LSF master host before configuring IBM Spectrum LSF Resource Connector.

- Java Runtime Environment (JRE) version 8

Chapter 3. 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.

Before you begin

LSF should be installed on the local master 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 master host IP address to accept all traffic from the master 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 master host does not run on AWS. AWS slave hosts must have either a public IP address or the network must be configured in a way for the master LIM to connect to the LIMs on the AWS slave 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 slave host on the AWS EC2 instance

Follow these steps to install an LSF slave 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.
   - `lsf10.1.0.6_linux2.6-glibc2.3-x86_64.tar.Z`
   - `lsf10.1_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 master 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, Refer to

"Managing Users and User Groups" and "Between-Host User Account Mapping" in *Administering IBM Spectrum LSF*.

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 `slave.config` file and install LSF as a slave host on AWS EC2.

   ```
   ./lsfinstall -s -f slave.config
   ```

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

   ```
   $ cat /home/ec2-user/lsf/lsf10.1_lsfinstall/slave.config
   #  The LSF_ADMIN must be same admin as the master 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="master.myserver.com"

   # The [resource awshost] is required in the slave.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 master host of your cluster.

   LSF_SERVER_HOSTS (Required): List of master host candidates that the slave 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 slave hosts.

6. If required, update the /etc/hosts file on the EC2 host to add the master host name so the EC2 host can ping the master host.
7. Start the LSF daemons on the instance manually and make sure that the EC2 instance can join the master 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 master host candidates. Make sure that your environment settings for IP address resolution work between the LSF master host and borrowed hosts that join the cluster.

   Tip: Check whether the master 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 master and borrowed slave host IP ranges.

For more information on troubleshooting refer to Chapter 8, “Logging and troubleshooting the LSF resource connector,” on page 31.

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.

Chapter 4. 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 master host where the launch request of the AWS EC2 instance takes place.

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 master 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.

Executing the AWS enablement script for LSF

Before you begin

LSF should already be installed on an AWS AMI and already installed on the master on-premise 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" on page 38.

```
{
    "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 Chapter 7, “Submit jobs to LSF resource connector,” on page 27 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 master host

If a DNS server is not set up to resolve the LSF master 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 master host cannot recognize the private host name and IP address. For the master 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 master host directly. For example, add the following code to the `user_data.sh` script to give a machine host name that the master 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 Chapter 7, “Submit jobs to LSF resource connector,” on page 27 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 master host candidates. Make sure that your environment settings for IP address resolution work between the LSF master host and instances that join the cluster.

Check whether the master 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 master and borrowed slave host IP ranges.

If pre-provisioning steps are required to set up specifics on the master 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 ” on page 20 for more details.

Example

Chapter 5. 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)
- When to reclaim the AWS RC2 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 plugin 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” on page 40.

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
```

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” on page 40 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 Chapter 9, “LSF resource connector configuration reference,” on page 33 for the default values.

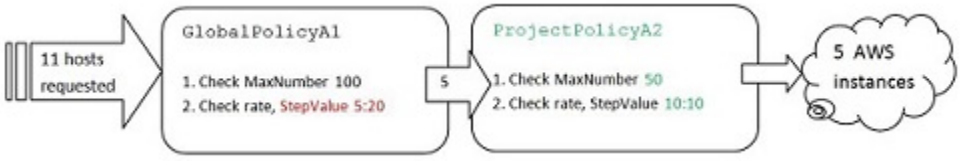
The following example uses the values of **LSB_RC_EXTERNAL_HOST_IDLE_TIME=60** and **LSB_RC_EXTERNAL_HOST_MAX_TTL=0**. This means that when the AWS EC2 host has no workload for 60 minutes, it will be terminated.

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.

Chapter 6. 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.

Assign 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 master 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+CBcqNTNDInLiM2ubPn8zj
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.

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" on page 38 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.

```
{
    "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 master host.

Related reference

"hostProviders.json" on page 38

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

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 master 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:

prePrvPath
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” on page 38

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

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” on page 40
The `policy_config.json` file configures custom policies for resource providers for LSF resource connector. The resource policy plug-in reads this file.

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” mycovfefe
```

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 master 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

Amazon Spot Instances

Amazon Spot Bid Advisor

“awsprov_templates.json” on page 43
The `awsprov_templates.json` file defines the mapping between LSF resource demand requests and AWS instances.

“awsprov_config.json” on page 42
The `awsprov_config.json` file contains administrative settings for the resource connector.

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`.

Example

The following template fragment, shows an example configuration for Spot instances:

```
{
     "Templates":
     [
       {
         "templateId": "SpotTemplate",
         "attributes":
         {
           "ncpus": ["Numeric", "4"],
           "awshost": ["Boolean", "1"]
         },
         "imageId": "ami-27ai",
         "vmType": "c4.xlarge, m4.large",
         "subnetId": "subnet-7c0dfb27,subnet-12286475,subnet-cc0248ba",
         "keyName": "LSF-Key",
         "MaxNumber": "10",
         "securityGroupIds": ["sg-7231"]
         "fleetRole": "arn:aws:iam::700071821657:role/EC2-Spot-Fleet-role",
         "spotPrice": "0.1",
         "allocationStrategy":"diversified",
       },
     ]
}
```

Chapter 7. Submit jobs to LSF resource connector

Submit jobs to borrow resources from a resource provider.

Check the resource connector status

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

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

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
   lsfmaster                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
lsfmaster                   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
lsfmaster       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" in *Administering IBM Spectrum LSF*.

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.

- 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.

Chapter 8. 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 master host to ensure the standard output format matches the one specified in section “Configuring AWS access with federated accounts” on page 18.
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.

Chapter 9. LSF resource connector configuration reference

Reference for configuring LSF resource connector. For detailed information about LSF configuration parameters, see the *IBM Spectrum LSF Configuration Reference*.

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.

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

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.

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 2 minutes. For example, if no pending jobs are in the cluster, for 2 minutes, **mbschd** stops evaluating them.

Example

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] ]
```

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.

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_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_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

LSF_RC_AUDIT_LOG

Syntax

LSF_RC_AUDIT_LOG=Y | y | N | n

Description

If set to Y or y, enables the LSF resource connector auditor to generate log files.

Default

N.

See also

RC_MAX_AUDIT_LOG_SIZE, **RC_MAX_AUDIT_LOG_KEEP_TIME**

RC_MAX_AUDIT_LOG_KEEP_TIME

Syntax
RC_MAX_AUDIT_LOG_KEEP_TIME=*integer*

Description

Specifies the amount of time that LSF keeps the resource connector audit log files, in months.

Default

Not defined.

See also

LSF_RC_AUDIT_LOG, **RC_MAX_AUDIT_LOG_SIZE**

RC_MAX_AUDIT_LOG_SIZE

Syntax
RC_MAX_AUDIT_LOG_SIZE=*integer*

Description

Specifies the maximum size of the LSF resource connector audit log files, in MB.

Default

Not defined.

See also

LSF_RC_AUDIT_LOG, **RC_MAX_AUDIT_LOG_KEEP_TIME**

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 master 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 **confPath**is relative to *LSF_TOP*/conf/resource_connector.

scriptPath
Path to the resource provider script directory. Full and relative paths are supported.

The default **confPath**is relative to *LSF_TOP*/*LSF_VERSION*/resource_connector.

scriptOptions

For AWS resource provider. By default, LSF assumes that the master 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
Optional. Resource connector runs the pre-provisioning script specified with absolute path after the instance is created and started successfully but before it is marked allocated to the LSF cluster.

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

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 10 minutes. If the pre- or post-provisioning program does not return after 10 minutes, it ends.

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, resource connnector 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. 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 `Main.py`:

```
"UserDefinedScriptPath" : "/usr/share/lsf/10.1/scripts/Main.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.

templateName
A list of template names. Supported values are `all` or any valid template name.

provider
A list of resource provider names. Supported values are `all` or any valid provider name.

For any attribute that is not defined, the default value is `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/launch for the consumer.

The value of **MaxNumber** must be less than the value of the **LSB_RC_MAX_INSTANCES_PER_TEMPLATE** parameter that is defined in the `lsf.conf` file or the **maxNumber** parameter in the template configuration file.

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.

Example

```
{
  “UserDefinedScriptPath” : “/usr/share/lsf/10.1/scripts/Main.py”,
  "Policies":
  [
    {
      "Name": "Policy1",
      "Consumer":
      {
        "rcAccount": ["all"],
        "templateName": ["all"],
        "provider": ["all"]
      },
      "MaxNumber": "100",
      "StepValue": "5:10"
    },
    {
      "Name": "Policy2",
      "Consumer":
       {
        "rcAccount": ["default", "project1"],
        "templateName": ["Template-VM-1"],
        "provider": ["aws"]
       },
      "MaxNumber": "50",
      "StepValue": "10:10"
    }
  ]
}
```

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 master host and the resource connector are deployed in an AWS instance, do not include the **AWS_CREDENTIAL_FILE**parameter. 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
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
Process 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`.

Example awsprov_config.json file

```
{
        "LogLevel": "INFO",
        "AWS_CREDENTIAL_FILE": "/home/aws/credentials",
        "AWS_REGION": "us-east-1",

}
```

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.

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 slave 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 master 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]`.

Supported attributes include the following:

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.

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 multiple subnets, separated by commas. For example:

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

Multiple subnets 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 plugin 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:

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.

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 I/O. This optimization improves performance for your EBS volumes by minimizing contention between Amazon EBS I/O and other traffic from your instance.

See the AWS documentation for more information about Amazon EBS-Optimized 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 Amazon 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 (LINK: 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.

Example 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"]
        },
        "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": "zone=us_east_2a"
        }
    ]
}
```

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
```

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"
        }
    ]
}
```

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"
        }

    ]
}
```

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"
        }
    ]
}
```